

**HANDBOOK  
OF  
AUTOMATION  
COMPUTATION  
AND  
CONTROL**

**VOLUME 2**

**HANDBOOK OF AUTOMATION,  
COMPUTATION, AND CONTROL**

**Volume 2**

---

**COMPUTERS AND DATA PROCESSING**

**NEW YORK • JOHN WILEY & SONS, INC.**

**London • Chapman & Hall, Limited**

# **HANDBOOK OF AUTOMATION, COMPUTATION, AND CONTROL**

**Volume 2**

---

**COMPUTERS AND DATA PROCESSING**

Prepared by a Staff of Specialists

Edited by

**EUGENE M. GRABBE**

**SIMON RAMO**

**DEAN E. WOOLDRIDGE**

Thompson Ramo Wooldridge Inc.  
Los Angeles, California

Copyright © 1959, by John Wiley & Sons, Inc.

All Rights Reserved. This book or any part thereof must not be reproduced in any form without the written permission of the publisher.

Library of Congress Catalog Card Number: 58-10800  
Printed in the United States of America

## CONTRIBUTORS

- L. D. AMDAHL**, *Thompson Ramo Wooldridge Inc., Los Angeles, California (Chapter 17)*
- C. E. AMMANN**, *American Airlines, New York City, New York (Chapter 9)*
- I. L. AUERBACH**, *Auerbach Electronics Corporation, Narberth, Pennsylvania (Chapters 15 and 16)*
- G. A. BEKEY**, *Space Technology Laboratories, Inc., Los Angeles, California (Chapter 23)*
- E. E. BOLLES**, *Thompson Ramo Wooldridge Inc., Los Angeles, California (Co-Editor, Part D)*
- R. BOSAK**, *System Development Corporation, Santa Monica, California (Chapter 9)*
- J. K. BRIGDEN**, *Space Technology Laboratories, Inc., Los Angeles, California (Chapter 20)*
- D. R. BROWN**, *The MITRE Corporation (formerly with Massachusetts Institute of Technology), Lexington, Massachusetts (Chapter 19)*
- J. W. BUSBY**, *Alvac Corporation, New York City New York (Chapter 5)*
- J. O. CAMPEAU**, *Litton Industries, Beverly Hills, California (Chapter 31)*
- R. G. CANNING**, *Canning, Sisson and Associates, Los Angeles, California (Chapter 4)*
- J. W. CARR, III**, *University of North Carolina (formerly with the University of Michigan), Chapel Hill, North Carolina (Chapter 2)*
- R. B. CONN**, *Space Technology Laboratories, Inc., Los Angeles, California (Co-Editor, Part C)*
- E. D. COWLES**, *The Detroit Edison Company, Detroit, Michigan (Chapter 8)*
- H. L. ENGEL**, *Thompson Ramo Wooldridge Inc., Los Angeles, California (Chapter 18)*
- E. M. GRABBE**, *Thompson Ramo Wooldridge Inc., Los Angeles, California (Co-Editor, Part D; Chapter 1)*
- B. M. GORDON**, *EPSCO, Inc., Boston, Massachusetts (Chapter 29)*
- W. J. KARPLUS**, *University of California at Los Angeles (Editor, Part E; Chapters 21, 25, and 27)*
- W. KINDLE**, *Electronic Associates, Inc., El Segundo, California (Chapter 21)*

- R. T. KOLL**, *Creole Petroleum Corporation, Caracas, Venezuela (Chapter 10)*
- J. F. LA FONTAINE**, *EPSCO, Inc., Boston, Massachusetts (Chapter 29)*
- H. T. LARSON**, *Aeronutronic Systems, Inc., Santa Ana, California (Co-Editor, Part C)*
- C. T. LEONDES**, *University of California at Los Angeles (Chapter 28)*
- H. S. LEVIN**, *Arthur Young & Company, New York City, New York (Chapter 7)*
- H. LOW**, *Space Technology Laboratories, Inc., Los Angeles, California (Chapter 26)*
- R. C. MACKEY**, *University of California at Los Angeles (Chapter 24)*
- M. E. MARON**, *Thompson Ramo Wooldridge Inc., Los Angeles, California (Chapter 11)*
- M. J. MENDELSON**, *Norden Division of United Aircraft Corporation, Gardena, California (Chapter 3)*
- I. PFEFFER**, *Space Technology Laboratories, Inc., Los Angeles, California (Chapter 22)*
- J. I. RAFFEL**, *Massachusetts Institute of Technology, Lincoln Laboratory, Lexington, Massachusetts (Chapter 19)*
- C. W. SCHMIDT**, *Teleregister Corporation, Los Angeles, California (Chapter 9)*
- R. L. SISSON**, *Canning, Sisson and Associates, Los Angeles, California (Chapter 4)*
- N. H. TAYLOR**, *Itek Corporation, Boston, Massachusetts (formerly with Massachusetts Institute of Technology) (Chapter 14)*
- H. TELLIER**, *General Electric Company, Richland, Washington (Chapter 8)*
- E. TOMASH**, *Telemeter Magnetics, Inc., Los Angeles, California (Chapter 6)*
- L. L. VAN OOSTEN**, *Allstate Insurance Company, Skokie, Illinois (Chapter 8)*
- A. C. VANSELOW**, *The Franklin Life Insurance Company, Springfield, Illinois (Chapter 8)*
- R. L. VAN WINKLE**, *The Franklin Life Insurance Company, Springfield, Illinois (Chapter 8)*
- W. H. WARE**, *The RAND Corporation, Santa Monica, California (Chapters 12 and 13)*
- G. P. WEST**, *Thompson Ramo Wooldridge Inc., Los Angeles, California (Chapter 30)*
- J. H. YIENGER**, *Aeronutronic Systems, Inc., Santa Ana, California (Chapter 5)*

# FOREWORD

The proliferation of knowledge now makes it most difficult for scientists or engineers to keep ahead of change even in their own fields, let alone in contiguous fields. One of the fields where recent change has been most noticeable, and in fact exponential, has been automatic control. This three-volume Handbook will aid individuals in almost every branch of technology who must constantly refresh their memories or refurbish their knowledge about many aspects of their work.

Automation, computation, and control, as we know them, have been evolving for centuries, but within the last generation their impact has been felt in nearly every segment of human endeavor. Feedback principles were exploited by Leonardo da Vinci and applied by James Watt. Some of the early theoretical work of importance was contributed by Lord Kelvin, who also, together with Charles Babbage, pointed the way to the development of today's giant computational aids. Since about the turn of the present century, the works of men like Minorsky, Nyquist, Wiener, Bush, Hazen, and von Neumann gave quantum jumps to computation and control. But it was during and immediately following World War II that quantum jumps occurred in abundance. This was the period when theories of control, new concepts of computation, new areas of application, and a host of new devices appeared with great rapidity. Technologists now find these fields charged with challenge, but at the same time hard to encompass. From the activities of World War II such terms as servomechanism, feedback control, digital and analog computer, transducer, and system engineering reached maturity. More recently the word automation has become deeply entrenched as meaning something about the field on which no two people agree.

Philosophically minded technologists do not accept automation merely as a third Industrial Revolution. They see it, as they stand about where the editors of this *Handbook* stood when they projected this work, as a manifestation of one of the greatest *Intellectual Revolutions in Thinking* that has occurred for a long time. They see in automation the natural

consequence of man's urge to exploit modern science on a wide front to perform useful tasks in, for example, manufacturing, transportation, business, physical science, social science, medicine, the military, and government. They see that it has brought great change to our conventional way of thinking about the human use of human beings, to quote Norbert Wiener, and in turn about how our engineers will be trained to solve tomorrow's engineering problems. They even see that it has precipitated some deep thinking on the part of our industrial and union leadership about the organization of workers in order not to hold captive bodies of workmen for jobs that automation, computation, and control have swept or will soon sweep away.

Perhaps the important new face on today's technological scene is the degree to which the broad field needs codification and unification in order that technologists can optimize their role to exploit it for the general good. One of the early instances of organized academic instruction in the field was at The Massachusetts Institute of Technology in the Electrical Engineering Department in September 1939, as a course entitled Theory and Application of Servomechanisms. I can well recollect discussions around 1940 with the late Dr. Donald P. Campbell and Dr. Harold L. Hazen, which led temporarily to renaming the course Dynamic Analysis of Automatic Control Systems because so few students knew what "servomechanisms" were. But when the GI's returned from war everybody knew, and everyone wanted instruction. Since that time engineering colleges throughout the land have elected to offer organized instruction in a multitude of topics ranging from the most abstract mathematical fundamentals to the most specific applications of hardware. Textbooks are available on every subject along this broad spectrum. But still the practicing control or computer technologist experiences great difficulty keeping abreast of what he needs to know.

As organized instruction appeared in educational institutions, and as industrial activity increased, professional societies organized groups in the areas of control and computation to meet the needs of their members to tell one another about technical advances. Within the past five years several trade journals have undertaken to report regularly on developments in theory, components, and systems. The net effect of all this is that the technologist is overwhelmed with fragmentary, sometimes contradictory, redundant information that comes at him at random and in many languages. The problem of assessing and codifying even a portion of this avalanche of knowledge is beyond the capabilities of even the most able technologist.

The editors of the *Handbook* have rightly concluded that what each technologist needs for his long-term professional growth is *to have a body*

*of knowledge that is negotiable at par in any one of a number of related fields for many years to come.* It would be ideal, of course, if a college education could give a prospective technologist this kind of knowledge. It is in the hope of doing this that engineering curricula are becoming more broadly based in science and engineering science. But it is unlikely that even this kind of college training will be adequate to cope with the consequences of the rapid proliferation of technology as is manifest in the area of automation, computation, and control. Hence, handbooks are an essential component of the technical literature when they provide the unity and continuity that are requisite.

I can think of no better way to describe this *Handbook* than to say that the editors, in both their organization of material and selection of substance, have given technologists a unified work of lasting value. It truly represents today's optimum package of that body of knowledge that will be negotiable at par by technologists for many years to come in a wide range of disciplines.

GORDON S. BROWN  
Dean, School of Engineering  
*Massachusetts Institute of Technology*



## PREFACE

Accelerated advances in technology have brought a steady stream of automatic machines to our factories, offices, and homes. The earliest automation forms were concerned with doing work, followed by the controlling function, and recently the big surge in automation has been directed toward data handling functions. New devices ranging from digital computers to satellites have resulted from military and other government research and development programs. Such activity will continue to have an important impact on automation progress.

One of the pressures for the development of automation has been the growing complexity and speed of business and industrial operations. But automation in turn accelerates the tempo of whatever it touches, so that we can expect future systems to be even larger, faster, and more complex. While a segment of engineering will continue to mastermind, by rule of thumb procedures, the design and construction of automatic equipment and systems, a growing percentage of engineering effort will be devoted to activities that may be classified as *problem solving*. The activities of the problem solver involve analysis of previous behavior of systems and equipment, simulation of present situations, and predictions about the future. In the past, problem solving has largely been practiced by engineers and scientists, using slide rules and hand calculators, but with the advent of large-scale data processing systems, the range of applications has been broadened considerably to include economic, government, and social activities. Air traffic control, traffic simulation, library searching, and language translation, are typical of the problems that have been attacked.

This *Handbook* is directed toward the problem solvers—the engineers, scientists, technicians, managers, and others from all walks of life who are concerned with applying technology to the mushrooming developments in automatic equipment and systems. It is our purpose to gather together in one place the available theory and information on general mathematics, feedback control, computers, data processing, and systems

design. The emphasis has been on practical methods of applying theory, new techniques and components, and the ever broadening role of the electronic computer. Each chapter starts with definitions and descriptions aimed at providing perspective and moves on to more complicated theory, analysis, and applications. In general, the *Handbook* assumes some engineering training and will serve as an information source and refresher for practicing engineers. For management, it will provide a frame of reference and background material for understanding modern techniques of importance to business and industry. To others engaged in various ramifications of automation systems, the *Handbook* will provide a source of definitions and descriptive material about new areas of technology.

It would be difficult for any one individual or small group of individuals to prepare a handbook of this type. A large number of contributors, each with a field of specialty, is required to provide the engineer with the desired coverage. With such a broad field, it is difficult to treat all material in a homogeneous manner. Topics in new fields are given in more detail than the older, established ones since there is a need for more background information on these new subjects. The organization of the material is in three volumes as shown on the inside cover of the *Handbook*. Volume 1 is on *Control Fundamentals*, Volume 2 is concerned with *Computers and Data Processing*, and Volume 3 with *Systems and Components*.

In keeping with the purpose of this *Handbook*, Volume 1 has a strong treatment of general mathematics which includes chapters on subjects not ordinarily found in engineering handbooks. These include sets and relations, Boolean algebra, probability, and statistics. Additional chapters are devoted to numerical analysis, operations research, and information theory. Finally, the present status of feedback control theory is summarized in eight chapters. Components have been placed with systems in Volume 3 rather than with control theory in Volume 1, although any discussion of feedback control must, of necessity, be concerned with components.

The importance of computing in research, development, production, real time process control, and business applications, has steadily increased. Hence, Volume 2 is devoted entirely to the design and use of analog and digital computers and data processors. In addition to covering the status of knowledge today in these fields, there are chapters on unusual computer systems, magnetic core and transistor circuits, and an advanced treatment of programming. Volume 3 emphasizes systems engineering. A part of the volume covers techniques used in important industrial applications by examining typical systems. The treatment of components

is largely concerned with how to select components among the various alternates, their mathematical description and their integration into systems. There is also a treatment of the design of components of considerable importance today. These include magnetic amplifiers, semiconductors, and gyroscopes.

We consider this *Handbook* a pioneering effort in a field that is steadily pushing back frontiers. It is our hope that these volumes will not only provide basic information on new fields, but also will inspire work and further research and development in the fields of automatic control. The editors are pleased to acknowledge the advice and assistance of Dean Gordon S. Brown and Professor Jerome B. Wiesner of the Massachusetts Institute of Technology, and Dr. Brockway McMillan of the Bell Telephone Laboratories, in organizing the subject matter. To the contributors goes the major credit for providing clear, thorough treatments of their subjects. The editors are deeply indebted to the large number of specialists in the control field who have aided and encouraged this undertaking by reviewing manuscripts and making valuable suggestions. Many members of the technical staff and secretarial staff of Thompson Ramo Wooldridge Inc. and the Ramo-Wooldridge Division have been especially helpful in speeding the progress of the *Handbook*.

EUGENE M. GRABBE  
SIMON RAMO  
DEAN E. WOOLDRIDGE

*June 1959*



# CONTENTS

## A. COMPUTER TERMINOLOGY

<b>Chapter 1.</b>	<b>Computer Terminology and Symbols</b>	<b>1-01</b>
	1. Standardization	1-01
	2. Symbols	1-01
	3. Glossary of Terminology	1-02
	References	1-22

## B. DIGITAL COMPUTER PROGRAMMING

<b>Chapter 2.</b>	<b>Programming and Coding</b>	<b>2-01</b>
	1. Nature of Programming	2-01
	2. Numbers and Scale Factors	2-12
	3. Number Conversion Tables	2-26
	4. Program Structure and Flow Diagrams	2-44
	5. Machine Logic	2-53
	6. Instruction Logic of Common Computers	2-63
	7. Traditional Programming Techniques	2-128
	8. Automatic Programming: Development and Objectives	2-155
	9. Automatic Programming: Assembly Programs	2-163
	10. Automatic Programming: Subroutines, Subroutine Generators, Utility Programs and Integrated Systems	2-167
	11. Automatic Programming: Languages, Compilers, and Translators	2-186

## CONTENTS

12. Automatic Programming: The IT Translator; Translator Construction 2-200
13. Automatic Programming: A Soviet Algebraic Language Compiler 2-228
14. Automatic Programming: Interpreters 2-234
15. Automatic Programming: Recursive Languages 2-244
16. Logical Programming 2-246
17. Microprogramming 2-251
18. Programs for Maintenance of Equipment 2-258
19. Programming with Natural Language 2-259  
Literature, Acknowledgments, and References 2-260

**C. THE USE OF DIGITAL COMPUTERS AND DATA PROCESSORS****Chapter 3. Data Processing Operations . . . . . 3-01**

1. Introduction 3-01
2. Data Collection 3-02
3. Data Conversion, Transcription, and Editing 3-03
4. Data Output 3-04
5. On-Line Versus Off-Line Processing 3-04
6. Scientific Data Manipulation 3-05
7. Business Data Manipulation 3-06
8. Checking 3-13

**Chapter 4. Quantitative Characteristics of Data Processing Systems . . . . . 4-01**

1. Determining System Requirements 4-01
2. Basic System Characteristics 4-02
3. Basic Equipment Characteristics 4-04
4. Measurement of System Factors 4-04
5. Relating System Characteristics to Equipment Characteristics 4-09
- References 4-16

<b>Chapter 5.</b>	<b>Equipment Description</b>	5-01
	1. General Equipment Description 5-01	
	2. Characteristics of Electronic Data Processing Equipment 5-04	
	3. Input Equipment 5-09	
	4. Storage Equipment 5-24	
	5. Output Equipment 5-33	
	6. Arithmetic and Logic Unit 5-38	
	7. Control Equipment 5-40	
	8. Typical Electronic Digital Equipment 5-43	
	References 5-43	
<b>Chapter 6.</b>	<b>Facility Requirements</b>	6-01
	1. Physical Installation 6-01	
	2. Personal Requirements 6-09	
	References 6-13	
<b>Chapter 7.</b>	<b>Design of Business Systems</b>	7-01
	1. General System Requirements 7-01	
	2. Stages of System Evolution 7-02	
	3. Detailed Steps of System Design 7-03	
	4. Economic Impacts of System Changes 7-12	
	References 7-14	
<b>Chapter 8.</b>	<b>Accounting Applications</b>	8-01
	1. Life Insurance 8-01	
	2. Casualty Insurance 8-08	
	3. Public Utility Customer Billing 8-11	
	4. Payroll 8-15	
<b>Chapter 9.</b>	<b>Inventory and Scheduling Applications</b>	9-01
	1. Inventory Control 9-01	
	2. Aircraft Production Scheduling 9-07	
	References 9-12	
<b>Chapter 10.</b>	<b>Scientific and Engineering Applications</b>	10-01
	1. Introduction 10-01	
	2. Simultaneous Linear Algebraic Equations and Matrix Inversion 10-02	

	3. Characteristic Roots and Vectors	
	10-04	
	4. Linear Programming	10-06
	5. Differential Equations	10-08
	6. Statistical Analysis	10-10
	References	10-12
<b>Chapter 11.</b>	<b>Handling of Non-Numerical Information</b>	<b>11-01</b>
	1. Introduction	11-01
	2. Performing Logic on a Digital Computer	11-02
	3. Game Playing Machines	11-11
	4. The Machine Translation of Languages	11-13
	5. Automatic Literature Searching and Retrieval	11-16
	References	11-19
<b>D.</b>	<b>DESIGN OF DIGITAL COMPUTERS</b>	
<b>Chapter 12.</b>	<b>Digital Computer Fundamentals</b>	<b>12-01</b>
	1. Digital Computers and Control Systems	12-01
	2. Digital Computer Fundamentals	12-02
	3. Machine Construction	12-07
	4. Number Systems and Number Codes	12-12
	5. Machine Number Systems	12-18
	6. Computer Design Characteristics	12-25
	References	12-30
<b>Chapter 13.</b>	<b>Techniques for Reliability</b>	<b>13-01</b>
	1. Introduction	13-01
	2. Summary of Operating and Design Techniques	13-02
	3. Operating Techniques	13-04
	4. System Design	13-05
	5. Circuit Design	13-07
	6. Maintenance	13-08
	References	13-10

<b>Chapter 14.</b>	<b>Components and Basic Circuits</b>	<b>14-01</b>
	1. Designing for Reliability	14-01
	2. Components and Circuit Design	14-03
	3. Marginal Checking	14-05
	4. Reliable Computer Circuits	14-19
	5. Components, Characteristics, and Application Notes	14-43
	6. Transistors	14-51
	References	14-54
<b>Chapter 15.</b>	<b>Magnetic Core Circuits</b>	<b>15-01</b>
	1. Fundamentals	15-01
	2. Magnetic Cores	15-04
	3. Transfer Loops	15-09
	4. Magnetic Shift Registers	15-15
	5. Logical Function Elements	15-16
	6. Magnetic Core Storages	15-19
	7. Timing Control Circuits	15-21
	8. Arithmetic and Miscellaneous Applications	15-22
	9. Drivers for Magnetic Core Circuits	15-23
	References	15-24
<b>Chapter 16.</b>	<b>Transistor Circuits</b>	<b>16-01</b>
	1. Introduction	16-01
	2. Transistor Switching Properties	16-02
	3. Direct-Coupled Transistor Switching Circuits	16-05
	4. Point-Contact Transistor Pulse Amplifiers	16-15
	5. Transistorized Calculator	16-20
	References	16-30
<b>Chapter 17.</b>	<b>Logical Design</b>	<b>17-01</b>
	1. Computer Elements	17-01
	2. Algebraic Techniques of Logical Design	17-10
	3. Preliminary Design Considerations	17-24
	4. Detailed Logical Design	17-30

## CONTENTS

- 5. Direct Simulation of a Logical Design 17-38
- References 17-42

<b>Chapter 18.</b>	<b>Arithmetic and Control Elements</b>	<b>18-01</b>
	1. System Considerations	18-01
	2. Notation	18-02
	3. Binary Operations	18-03
	4. Decimal Operations	18-25
	5. Special Operations	18-30
	6. Control Elements	18-33
	References	18-40

<b>Chapter 19.</b>	<b>Storage</b>	<b>19-01</b>
	1. Basic Concepts	19-01
	2. Magnetic Drum Storage	19-04
	3. Magnetic Core Storage	19-13
	4. Other Storage Techniques	19-29
	References	19-33

<b>Chapter 20.</b>	<b>Input-Output Equipment for Digital Computers</b>	<b>20-01</b>
	1. The Input-Output System	20-01
	2. Printed Page	20-06
	3. Perforated Tape	20-19
	4. Punched Card Machines	20-30
	5. Magnetic Tape	20-33
	6. Analog-Digital Conversion	20-44
	References	20-66

## **E. DESIGN AND APPLICATION OF ANALOG COMPUTERS**

<b>Chapter 21.</b>	<b>Analog Computation in Engineering</b>	<b>21-01</b>
	1. Definition of Analog Computation	21-01
	2. Classification of Analog Computers	21-02
	3. Requirements of Analog Computers	21-05
	4. General Steps in the Solution of Engineering Problems	21-06

	5. Areas of Application of Analog Computers	21-09
	6. Symbols and Diagram Notation	21-11
	References	21-11
<b>Chapter 22.</b>	<b>Linear Electronic Computer Elements</b>	<b>22-01</b>
	1. Introduction and Computer Diagram Notation	22-01
	2. Passive Computer Elements	22-04
	3. Direct-Current Operational Amplifiers with Feedback	22-08
	4. Scale Factors	22-10
	5. Typical Problem Setup	22-12
	6. Representation of Complex Transfer Functions	22-13
	7. Operational Amplifier Design	22-16
	8. Errors in Linear Computer Elements	22-33
	References	22-37
<b>Chapter 23.</b>	<b>Nonlinear Electronic Computer Elements</b>	<b>23-01</b>
	1. Function Multipliers	23-01
	2. Function Generators	23-14
	3. Switching Devices	23-22
	4. Trigonometric Devices	23-31
	5. Time Delay Simulators	23-34
	References	23-39
<b>Chapter 24.</b>	<b>Analogs and Duals of Physical Systems</b>	<b>24-01</b>
	1. Electric Analogy of Dynamic System	24-01
	2. General Terminology	24-03
	3. Analysis of General Systems	24-03
	4. Energy Considerations	24-07
	5. Duality	24-08
	6. Construction of Duals	24-09
	7. Across and Through Variables in Physical Systems	24-12
	References	24-13

<b>Chapter 25.</b>	<b>Solution of Field Problems</b>	<b>25-01</b>
	1. Formulation of Engineering Problems as Partial Differential Equations 25-01	
	2. Continuous Type Electric Analogs 25-05	
	3. Discrete Element Type Electric Ana- logs 25-11	
	4. Nonelectric Field Analogs 25-22	
	References 25-23	
<b>Chapter 26.</b>	<b>Noise and Statistical Techniques</b>	<b>26-01</b>
	1. Introduction and Definition 26-01	
	2. Random Variable Concepts 26-02	
	3. Treatment of Linear Systems 26-06	
	4. Treatment of Nonlinear Systems 26-09	
	5. Noise Generators 26-12	
	References 26-20	
<b>Chapter 27.</b>	<b>Mechanical Computer Elements</b>	<b>27-01</b>
	1. Introduction 27-01	
	2. Basic Operations 27-02	
	3. Function Generation 27-05	
	4. Solution of Equations 27-09	
	5. Scale Factors 27-14	
	References 27-15	
<b>Chapter 28.</b>	<b>Digital Techniques in Analog Compu- tation</b>	<b>28-01</b>
	1. Introduction 28-01	
	2. Digital Differential Analyzer 28-02	
	3. Digital Operational Computers 28-11	
	4. Auxiliary Digital Computer Tech- niques 28-15	
	5. Auxiliary Digital Control Techniques 28-17	
	References 28-18	

**F. UNUSUAL COMPUTER SYSTEMS**

<b>Chapter 29.</b>	<b>Operational Digital Techniques</b>	<b>29-01</b>
	1. Introduction	29-01
	2. Basic Devices	29-05
	3. Applications of Operational Digital Techniques	29-14
	4. Incremental Computation	29-17
	References	29-29
<b>Chapter 30.</b>	<b>Combined Analog-Digital Computer Systems</b>	<b>30-01</b>
	1. Description and Applications	30-01
	2. System Components	30-02
	3. Control and Timing	30-08
	4. Modes of Operation	30-13
	References	30-15
<b>Chapter 31.</b>	<b>Simple Turing Type Computers</b>	<b>31-01</b>
	1. Basic Concepts	31-01
	2. Functional Requirements	31-02
	3. Machine Description	31-03
	4. Mechanization	31-07
	5. Programming	31-09
	6. Communication with No Auxiliary Storage	31-13
	7. Machine Comparison	31-15
	References	31-16

**INDEX**



# COMPUTER TERMINOLOGY

## A. COMPUTER TERMINOLOGY

### 1. Computer Terminology and Symbols, by *E. M. Grabbe*



# Computer Terminology and Symbols

*E. M. Grabbe*

1. Standardization	I-01
2. Symbols	I-01
3. Glossary of Terminology	I-02
References	I-22

## I. STANDARDIZATION

The growth of analog and digital computers as major components of modern computing and control systems has done much to encourage standardization of terminology and symbols. A sizable part of this effort has been directed toward the terminology of digital computers. Hence, the glossary of terminology given in Sect. 3 is largely concerned with digital terms. No attempt has been made to define the terms associated with computer usage in scientific computation, business data processing, and control applications.

## 2. SYMBOLS

**Diagram Symbols.** Several sets of symbols for schematic and circuit diagrams have been in use in the analog and digital fields. In Part E on analog computers, one set of symbols has been chosen and used throughout the chapters. The alternate notation is also listed for linear computing elements in Chap. 22 (see Table 1).

In the digital field, while terminology has been standardized to some degree, the use of symbols has not. A variety of symbols is employed

for programming, logic, and circuit diagrams, depending on the author's preference and the type of diagram. Some symbols are easier to use for some purposes than others. In all cases the symbols are clearly defined and usage is unambiguous.

Since symbols are not standardized, no detailed list is given, but they are described in the various chapters. The following is a list of the chapters in the Handbook where tables of symbols may be found:

Symbols	Chapters
Digital computer	
Programming	Chap. 2, Sect. 4
Logical design	Chap. 17, Sect. 4
Logical operations	Chap. 17, Sect. 1, Table 1
Magnetic cores	Chap. 15, Sect. 1
Analog computer	
Linear computing elements	Chap. 22, Sect. 1
Nonlinear computing elements	Chap. 23, Sect. 1
Mechanical computing elements	Chap. 27, Sect. 2
Analog and duals	Chap. 24, Sect. 2
Digital differential analyzers	Chap. 28, Sect. 2

**Letter Symbols.** Letter symbols are standardized to some extent in Part E, Design and Application of Analog Computers. (See Chap. 21, Sect. 1.) Elsewhere letter symbols are defined when they are used.

### 3. GLOSSARY OF TERMINOLOGY

Terminology from the Institute of Radio Engineers (Ref. 1) and the Association for Computing Machinery (Ref. 2) has been compiled in a glossary. The I.R.E. terminology is largely concerned with digital computer design, although some analog terms are included. The A.C.M. terminology is concerned with programming. Where an overlap exists, the I.R.E. terminology has been selected since it represents the later effort. For some terms, minor changes or additions have been made for clarity and explanatory notes and examples have been added. Some terms are included which have no official definition, and reference to the chapters where they are described and defined is given. For terms not listed in this glossary, please refer to the index.

Terminology is reproduced with the permission of the Institute of Radio Engineers and the Association for Computing Machinery.

#### Glossary of Terminology

**Access Time.** A time interval which is characteristic of a storage unit, and is essentially a measure of the time required to communicate with that unit. Many definitions of the beginning and ending of this interval are in common use.

**Accumulator.** A device which stores a number and which, on receipt of another number, adds it to the number already stored and stores the sum. *Note.* The term is also applied to devices which function as described but which have other facilities also.

**Accuracy.** The quality of freedom from mistake or error, that is, of conformity to truth or to a rule. Accuracy is distinguished from *precision*. *Example.* A six-place table is more precise than a four-place table. However, if there are errors in the six-place table, it may be either more or less accurate than the four-place table.

**Adder.** A device which can form the sum of two or more numbers or quantities.

**Address.** An expression, usually numerical, which designates a particular location in a storage or a memory device or other source or destination of information. See also *Instruction Code*.

**Absolute address,** an address assigned by the machine designer to a particular storage location.

**Relative address,** the address used to identify a word in a routine or subroutine with respect to its position in that routine or subroutine.

**Symbolic address** (floating address), an address chosen to identify a particular word, function, or other information in a routine, independent of the location of the information within the routine. Sometimes called *symbol* or *tag*.

**Address Part.** In an instruction, any part that is usually an address. See also *Instruction Code*.

**Analog** (in electronic computers). A physical system on which the performance of measurements yields information concerning a class of mathematical problems.

**Analog Computer.** A physical system together with means of control for the performance of measurements (upon the system) which yield information concerning a class of mathematical problems.

**And Circuit.** Synonym for *and gate*.

**And Gate.** A gate whose output is energized when and only when every input is in its prescribed state. Thus, this gate performs the function of the logical *and*.

**Arithmetic Element.** Synonym for *arithmetic unit*.

**Arithmetic Unit.** That part of a computer which performs arithmetic and logical operations.

**Assemble; Assembler, Assembly Routine; Assembly.** See *Routine*.

**Automatic Check.** See *Check, Automatic*.

**Band.** A group of tracks on a magnetic drum.

**Base.** See *Positional Notation*.

**Binary.** See *Positional Notation*.

**Binary Cell.** An elementary unit of storage which can be placed in either of two stable states.

**Binary-Coded-Decimal System.** A system of number representation in which each decimal digit is represented by a group of binary digits.

*Note.* Usually refers to the four position binary codes 0000 to 1001 (decimal 1 to 9). Another example is the excess-three code.

**Binary Number System.** See *Positional Notation*.

**Binary Point.** See *Point*.

**Biquinary.** See *Positional Notation*.

**Bit.** (1) An abbreviation of "binary digit." (2) A single character of a language employing exactly two distinct kinds of characters. (3) A unit of information capacity of a storage device. The capacity in bits is the logarithm to the base two of the number of possible states of the device. See also *Storage Capacity*.

**Block.** A group of words considered as a unit.

**Borrow.** See *Carry*.

**Branch.** A synonym for *conditional jump*.

**Break Point.** A point in a routine at which a special instruction is inserted which, if desired, will cause a digital computer to stop for a visual check of progress.

**Buffer.** (1) An isolating circuit used to avoid reaction of a driven circuit on the corresponding driving circuit. (2) A storage device used to compensate for a difference in rate of flow of information or time of occurrence of events when transmitting information from one device to another.

**Bus.** One or more conductors which are used as a path for transmitting information from any of several sources to any of several destinations.

**Calculator.** See *Computer*.

**Carry.** (1) A signal, or an expression, produced as a result of an arithmetic operation on one digit place of two or more numbers expressed in positional notation, and transferred to the next higher place for processing there. (2) Usually a signal, or an expression, as defined in (1) which arises, in adding, when the sum of two digits in the same digit place equals or exceeds the base of the number system in use. If a carry into a digit place will result in a carry out of the same digit place, and if the normal adding circuit is bypassed when generating this new carry, it is called a **standing-on-nines carry**, or **high-speed carry**. If the normal adding circuit is used in such a case, the carry is called a **cascaded carry**. If a carry resulting from the addition of carries is not allowed to propagate (e.g., when forming the partial product in one step of a multiplication process), the process is called a **partial carry**. If it

is allowed to propagate, the process is called a **complete carry**. If a carry generated in the most significant digit place is sent directly to the least significant digit place (e.g., when adding two negative numbers by using nines complements) that carry is called an **end-around carry**. (3) In direct subtraction, a signal or expression as defined in (1) which arises when the difference between the digits is less than zero. Such a carry is frequently called a **borrow**. (4) The action of forwarding a carry. (5) The command requesting a carry to be forwarded.

**Cascaded Carry.** See *Carry*.

**Cell.** An elementary unit of storage (e.g., binary cell, decimal cell).

**Channel.** That portion of a storage medium which is accessible to a given reading station. See also *Track*.

**Character.** One of a set of elementary marks or events which may be combined to express information. *Note.* A group of characters, in one context, may be considered as a single character in another, as in the *binary-coded-decimal system*.

**Check.** A process of partial or complete testing of (a) the correctness of machine operations, (b) the existence of certain prescribed conditions within the computer, or (c) the correctness of the results produced by a routine. A check of any of these conditions may be made automatically by the equipment or may be programmed. See also *Verification*.

**Check, Automatic.** A check performed by equipment built into the computer specifically for that purpose, and automatically accomplished each time the pertinent operation is performed. Sometimes referred to as a built-in check. **Machine check** can refer to an automatic check, or to a programmed check of machine functions.

**Check Digits.** See *Check, Forbidden Combination*.

**Check, Forbidden Combination.** A check (usually an automatic check) which tests for the occurrence of a nonpermissible code expression. A **self-checking code** (or **error-detecting code**) uses code expressions such that one (or more) error(s) in a code expression produces a forbidden combination. A **parity check** makes use of a self-checking code employing binary digits in which the total number of 1's (or 0's) in each permissible code expression is always odd or always even. A check may be made for either even parity or odd parity. A **redundancy check** employs a self-checking code which makes use of redundant digits called **check digits**.

**Check, Marginal.** A preventive maintenance procedure in which certain operating conditions, e.g., supply voltage or frequency, are varied about their normal values in order to detect and locate incipient defective units.

**Check Problem.** See *Check, Programmed.*

**Check, Programmed.** A check consisting of tests inserted into the programmed statement of the problem and accomplished by appropriate use of the machine's instructions. A **mathematical check** (or **control**) is a programmed check of a sequence of operations which makes use of the mathematical properties of that sequence. A **check routine** or **check problem** is a routine or problem which is designed primarily to indicate whether a fault exists in the computer, without giving detailed information on the location of the fault. See also *Diagnostic Routine* and *Test Routine* under *Routine.*

**Check Routine.** See *Check, Programmed.*

**Check, Selection.** A check (usually an automatic check) to verify that the correct register, or other device, is selected in the performance of an instruction.

**Check, Transfer.** A check (usually an *automatic check*) on the accuracy of the transfer of a word.

**Circulating Register (or Memory).** A register (or memory) consisting of a means for delaying information and a means of regenerating and reinserting the information into the delaying means.

**Clear.** To restore a storage or memory device to a prescribed state, usually that denoting zero.

**Clock.** A primary source of synchronizing signals.

**Code.** (1) A system of characters and rules for representing information. (2) Loosely, the set of characters resulting from the use of a code. (3) To prepare a routine in machine language for a specific computer. (4) To encode, to express given information by means of a code. See also *Instruction Code, Language, Operation Code, and Pseudo-code.*

**Coding.** The list, in computer code or in pseudo-code, of the successive computer operations required to solve a given problem.

**Absolute, relative, or symbolic coding,** coding in which one uses absolute, relative, or symbolic addresses, respectively.

**Automatic coding,** any technique in which a computer is used to help bridge the gap between some "easiest" form, intellectually and manually, of describing the steps to be followed in solving a given problem and some "most efficient" final coding of the same problem for a given computer. Two basic forms, defined under **Routine**, are *compilation* and *interpretation.*

**Collate.** To combine two or more similarly ordered sets of items to produce another ordered set composed of information from the original sets. Both the number of items and the size of the individual items in the resulting set may differ from those of either of the original sets and of their sums.

**Command.** (1) One of a set of several signals (or groups of signals) which occurs as a result of an instruction; the commands initiate the individual steps which form the process of executing the instruction. (2) Synonym for *instruction*.

**Comparator.** A device for comparing two different transcriptions of the same information to verify the accuracy of transcription, especially of one copy of tape from another.

**Compare.** To examine the representation of a quantity for the purpose of discovering its relationship to zero, or of two quantities for the purpose of discovering identity or relative magnitude.

**Comparison.** The act of comparing and, usually, acting on the result of the comparison.

**Compile; Compiler, Compiling Routine; Compilation.** See *Routine*.

**Complement.** (1) A number whose representation is derived from the finite positional notation of another by one of the following rules. (a) True complement: subtract each digit from one less than the base; then add 1 to the least significant digit and execute all carries required. (b) Base minus one's complement: subtract each digit from one less than the base (e.g., "9's complement" in the base 10 and "1's complement" in the base 2). (2) To form the complement of a number. (a) Complement on  $n$ : subtract each digit of the given quantity from  $n - 1$ , add unity to the least significant digit, and perform all resultant carries. For example, the *two's complement* of binary 11010 is 00110; the *ten's complement* of decimal 456 is 544. (b) Complement on  $n - 1$ : subtract each digit of the given quantity from  $n - 1$ . For example, the *one's complement* of binary 11010 is 00101; the *nine's complement* of decimal 456 is 543. *Note.* In many machines, a negative number is represented as the complement of the corresponding positive number.

**Complete Carry.** See *Carry*.

**Computer.** (1) A machine for carrying out calculations. (2) By extension, a machine for carrying out specified transformations on information.

**Conditional Jump.** See *Jump*.

**Conditional Transfer of Control.** Synonym for *conditional jump*.

**Control.** (1) To exercise directing, guiding, or restraining power over. (2) Power or authority to control. (3) Usually, those parts of a digital computer which effect the carrying out of instructions in proper sequence, the interpretation of each instruction, and the application of the proper signals to the arithmetic unit and other parts in accordance with this interpretation. (See Chap. 18.) (4) Frequently, one or more

of the components in any mechanism responsible for interpreting and carrying out manually initiated directions. Sometimes called manual control. (5) In some business applications of mathematics, a mathematical check.

**Convert.** See *Routine*.

**Copy.** To reproduce information in a new location by replacing whatever was previously stored there and leaving the source of the information unchanged. See also *Transfer*.

**Correction.** See *Error*.

**Counter.** (1) A device capable of changing from one to the next of a sequence of distinguishable states upon each receipt of an input signal. (2) Less frequently, an accumulator.

**Counter, Ring.** A loop of interconnected bistable elements such that all but one are in their normal (or abnormal) state at any one time, and so that, as input signals are counted, the position of the one abnormal (normal) state moves in an ordered sequence around the loop.

**Cycle.** (1) The sequence of events beginning with a particular event and including intervening events leading up to a recurrence of the original event. (2) The time interval which spans the sequence of events of (1). See *Loop, Major Cycle, Minor Cycle*.

**Cyclic Binary Code.** See Chaps. 11 and 20.

**Cyclic Shift.** See *Shift*.

**Decimal Number System.** See *Positional Notation*.

**Decimal Point.** See *Point*.

**Decoder.** A network or system in which a combination of inputs is excited at one time to produce a single output. Sometimes called *matrix*.

**Delay Line.** (1) Originally, a device utilizing wave propagation for producing a time displacement of a signal. (2) Commonly, any device for producing a time displacement of a signal.

**Delay-Line Memory.** Synonym for **delay-line storage**.

**Delay-Line Storage.** A storage or memory device consisting of a delay line and means for regenerating and reinserting information into the delay line.

**Diagnostic Routine.** See *Routine*.

**Differentiator.** A device, usually of the analog type, whose output is proportional to the derivative of an input signal.

**Digit.** See *Positional Notation*.

**Digital Computer.** A computer which operates with information, numerical or otherwise, represented in a digital form.

**Double-Length Number, Double-Precision Number.** See *Number, Double-Length*.

**Edit.** To rearrange information. Editing may involve the deletion of unwanted data, the selection of pertinent data, the insertion of invariant symbols such as page numbers and typewriter characters, and the application of standard processes such as zero suppression.

**Encoder.** A network or system in which only one input is excited at a time and each input produces a combination of outputs. Sometimes called *matrix*.

**End-Around Carry.** See *Carry*.

**Erase.** To replace all the binary digits in a storage device by binary zeros. In a binary computer, erasing is equivalent to clearing. While in a coded decimal computer where the pulse code for decimal zero may contain binary ones, *clearing* leaves decimal zero whereas *erasing* leaves all-zero pulse codes. Erasing of magnetic tapes and drums may leave all zeros or may remove all information, both ones and zeros.

**Error.** (1) In mathematics, the difference between the true value and a calculated or observed value. A quantity (equal in absolute magnitude to the error) added to a calculated or observed value to obtain the true value is called a **correction**. (2) In a computer or data processing system, any incorrect step, process, or result. Strictly speaking, "error" is a mathematical term, but in computer engineering the term is also commonly used to refer to machine malfunctions as "machine errors" and to human mistakes as "human errors." It is frequently helpful to distinguish between these as follows: *errors* result from approximations used in numerical methods, **mistakes** result from incorrect programming, coding, data transcription, manual operation, etc.; **malfunctions** result from failures in the operation of machine components such as gates, flip-flops, and amplifiers.

**Inherited error,** the error in the initial values, especially the error inherited from the previous steps in the step-by-step integration.

**Rounding error,** the error resulting from deleting the less significant digits of a quantity and applying some rule of correction to the part retained.

**Truncation error,** the error resulting from the use of only a finite number of terms of an infinite series, or from the approximation of operations in the infinitesimal calculus by operations in the calculus of finite differences.

**Error-Detecting Code.** See *Check, Forbidden Combination*.

**Excess-Three Code.** A number code in which the decimal digit  $n$  is represented by the four-bit binary equivalent of  $n + 3$ . See also *Binary-Coded-Decimal System*.

**Extract.** To form a new word by juxtaposing selected segments of given words.

**Field.** A set of one or more characters (not necessarily all lying in the same word) which is treated as a whole; a unit of information. See also *Item*; *Key*.

**Card field,** a set of visually consecutive card columns fixed as to number and position into which the same unit of information is regularly entered.

**File.** A sequential set of items (not necessarily all of the same size).

**Fixed-Point System.** See *Point*.

**Flip-Flop.** (1) A device having two stable states and two input terminals (or types of input signals) each of which corresponds with one of the two states. The circuit remains in either state until caused to change to the other state by application of the corresponding signal. (2) A similar bistable device with an input which allows it to act as a single-stage binary counter.

**Floating-Point System.** See *Point*.

**Flow Diagram.** A graphic representation of a routine.

**Forbidden Combination Check.** See *Check, Forbidden Combination*.

**Four-Address Code.** See *Instruction Code*.

**Gate.** A circuit having an output and a multiplicity of inputs so designed that the output is energized when and only when certain input conditions are met. See also *And Gate*; *Or Gate*.

**Generate; Generator, Generative Routine; Generation.** See *Routine*.

**Gray Code.** See Chaps. 11 and 20.

**Half Adder.** A circuit having two input and two output channels for binary signals (0, 1) and in which the output signals are related to the input signals according to the following table:

Input to		Output from	
<i>A</i>	<i>B</i>	<i>S</i>	<i>C</i>
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(So called because two half adders can be used in the construction of one binary *adder*.)

**Hexadecimal.** See *Positional Notation*.

**High-Speed Carry.** See *Carry*.

**Information.** A set of symbols or an arrangement of hardware that designates one out of a finite number of alternatives; an aggregation of data which may or may not be organized.

**Inhibiting Input.** A gate input which, if in its prescribed state, prevents any output which might otherwise occur.

**Instruction.** See *Instruction Code*.

**Instruction Code.** An artificial language for describing or expressing the instructions which can be carried out by a digital computer. In automatically sequenced computers, the instruction code is used when describing or expressing sequences of instructions, and each instruction word usually contains a part specifying the operation to be performed and one or more addresses which identify a particular location in storage. Sometimes an *address part* of an instruction is not intended to specify a location in storage but is used for some other purpose. If more than one address is used, the code is called a **multiple-address code**. In a typical instruction of a **four-address code** the addresses specify the location of two operands, the destination of the result, and the location of the next instruction in the sequence. In a typical **three-address code**, the fourth address specifying the location of the next instruction is dispensed with and the instructions are taken from storage in a preassigned order. In a typical **one-address** or **single-address code**, the address may specify either the location of an operand to be taken from storage, the destination of a previously prepared result, or the location of the next instruction. The arithmetic element usually contains at least two storage locations, one of which is an accumulator. For example, operations requiring two operands may obtain one operand from the main storage and the other from a storage location in the arithmetic element which is specified by the operation part.

**Breakpoint instruction**, an instruction which, if some specified switch is set, will cause the computer to stop, or proceed in a special way.

**Conditional breakpoint instruction**, a conditional jump instruction which, if some specified switch is set, will cause the computer to stop, after which either the routine may be continued as coded or a jump may be forced.

**One-plus-one, or three-plus-one address instruction**, a two- or four-address instruction, respectively, in which one of the addresses always specifies the location of the next instruction to be performed.

**Zero address instruction**, an instruction specifying an operation in which the location of the operands are defined by the computer code, so that no address need be given explicitly.

**Integrator.** (1) A device whose output is proportional to the integral of an input signal. (2) In certain digital machines, a device for numerically accomplishing an approximation to the mathematical process of integration.

**Interlock.** A device which prevents certain activities for the duration of certain other activities.

**Interpret, Interpreter, Interpretive Routine, Interpretation.** See *Routine*.

**Item.** A set of one or more fields containing related information; a unit of correlated information relating to a single person or object.

**Jump.** To (conditionally or unconditionally) cause the next instruction to be selected from a specified storage location.

**Conditional Jump.** An instruction which will cause the proper one of two (or more) addresses to be used in obtaining the next instruction, depending upon some property of one or more numerical expressions or other conditions.

**Unconditional Jump.** An instruction which interrupts the normal process of obtaining instructions in an ordered sequence, and specifies the address from which the next instruction must be taken.

**Key.** A set of characters, forming a field, used to identify an item.

**Language.** (1) A system consisting of (a) a well-defined, usually finite, set of characters, (b) rules for combining characters with one another to form words or other expressions, and (c) a specific assignment of meaning to some of the words or expressions, usually for communicating information or data among a group of people, machines, etc. (2) A system similar to (1) but without any specific assignment of meanings. Such systems may be distinguished from (1), when necessary, by referring to them as formal or uninterpreted languages. Although it is sometimes convenient to study a language independently of any meanings, in all practical cases at least one set of meanings is eventually assigned. See also *Machine Language*.

**Library.** An ordered set or collection of standard and proven routines and subroutines by which problems and parts of problems may be solved, usually stored in relative or symbolic coding. (A library may be subdivided into various volumes, such as floating decimal, double-precision, or complex, according to the type of arithmetic employed by the subroutines.)

**Logic.** See *Logical Design*.

**Logical Design.** (1) The planning of a computer or data processing system prior to its detailed engineering design. (2) The synthesizing of a network of logical elements to perform a specified function. (3) The result of (1) and (2), frequently called the **logic** of the system, machine, or network.

**Logical Diagram.** In logical design, a diagram representing the *logical elements* and their interconnections without necessarily expressing construction or engineering details.

**Logical Element.** In a computer or data processing system, the smallest building blocks which can be represented by operators in an

appropriate system of symbolic logic. Typical logical elements are the *and* gate and the flip-flop which can be represented as operators in a suitable symbolic logic.

**Logical Operation.** (1) Any nonarithmetical operation. Examples are: *extract*, logical (bit-wise) multiplication, jump, and data transfer. (2) Sometimes only those nonarithmetical operations which are expressible bit-wise in terms of the propositional calculus or a two-valued Boolean algebra.

**Logical Symbol.** A symbol used to represent a logical element graphically.

**Loop.** The repetition of a group of instructions in a routine. See also *Cycle*.

**Machine Check.** See *Check, Automatic*.

**Machine Language.** (1) A language, occurring within a machine, ordinarily not perceptible or intelligible to people without special equipment or training. (2) A translation or transliteration of (1) into more conventional characters but frequently still requiring special training to be intelligible.

**Major Cycle.** In a storage device which provides serial access to storage positions, the time interval between successive appearances of a given storage position.

**Malfunction.** See *Error*.

**Manchester Recording.** See Chap. 19, Sect. 2.

**Marginal Checking.** See *Check, Marginal*.

**Marginal Testing.** See *Check, Marginal*.

**Master Routine.** See *Routine, Executive*.

**Mathematical Check.** See *Check, Programmed*.

**Matrix (Switch).** (1) A network or system having a number of inputs and outputs and so connected that signals representing information expressed in a certain code, when applied to the inputs, cause output signals to appear which are representations of the input information in a different code. (2) A network or system in which a combination of inputs is excited at one time to produce a single output. (3) A network or system in which only one input is excited at a time and each input produces a combination of outputs.

**Memory.** See *Storage*.

**Merge.** To produce a single sequence of items, ordered according to some rule (i.e., arranged in some orderly sequence), from two or more sequences previously ordered according to the same rule, without changing the items in size, structure, or total number. Merging is a special case of collation.

**Memory Capacity.** Synonym for *storage capacity*.

**Minor Cycle.** In a storage device which provides serial access to storage positions, the time interval between the appearance of corresponding parts of successive words.

**Mistake.** See *Error*.

**Modified Binary Code.** See Chaps. 11 and 20.

**Modifier.** A quantity, sometimes the cycle index, used to alter the address of an operand.

**Modify.** (1) To alter in an instruction the address of the operand. (2) To alter a subroutine according to a defined parameter.

**Multiple-Address Code.** See *Instruction Code*.

**Multiplier.** A device which has two or more inputs and whose output is a representation of the product of the quantities represented by the input signals. (See Chap. 18.)

**NRZ, Non-Return to Zero Recording.** See Chap. 19, Sect. 2.

**NRZI, Non-Return to Zero, Invert Recording.** See Chap. 19, Sect. 2.

**Number.** (1) Formally, an abstract mathematical entity which is a generalization of a concept used to indicate quantity, direction, etc. In this sense a number is independent of the manner of its representation. (2) Commonly, a representation of a number as defined above (e.g., the binary number "10110," the decimal number "3695," or a sequence of pulses). (3) A word composed wholly or partly of digits, and perhaps a sign, which does not necessarily represent the abstract entity mentioned in the first meaning. *Note.* Whenever there is a possibility of confusion between meaning (1) and meaning (2) or (3), it is usually possible to make an unambiguous statement by using "number" for meaning (1) and "numerical expression" for meaning (2) or (3). See also *Positional Notation*.

**Number, Double-Length.** A number having twice as many digits as are ordinarily used in a particular computer.

**Number System.** See *Positional Notation*.

**Octal.** See *Positional Notation*.

**Octonary.** See *Positional Notation*.

**One-Address Code.** See *Instruction Code*.

**On-Line Operations.** See *Real-Time Operation*.

**Operation Code.** (1) The list of operation parts occurring in an instruction code, together with the names of the corresponding operations (e.g., "add," "unconditional transfer," and "add and clear"). (2) Synonym for operation part of an instruction.

**Arithmetical operations,** operations in which numerical quantities form the elements of the calculation (e.g., addition, subtraction, multiplication, division).

**Complete operation**, an operation which includes (a) obtaining all operands from storage, (b) performing the operation, (c) returning resulting operands to storage, and (d) obtaining the next instruction.

**Computer operation**, the electronic operation of hardware resulting from an instruction.

**Logical operations**, operations in which logical (yes-or-no) quantities form the elements being operated on (e.g., comparison, extraction). A usual requirement is that the value appearing in a given column of the result shall not depend on the values appearing in more than one given column of each of the arguments.

**Red tape operations**, operations which do not directly contribute to the result; i.e., arithmetical, logical, and transfer operations used in modifying the address section of other instructions, in counting cycles, and in rearranging data.

**Transfer operations (storage operations)**, operations which move information from one storage location or one storage medium to another (e.g., read, record, copy, transmit, exchange). Transfer is sometimes taken to refer specifically to movement between different media; storage to movement within the same medium.

Although many operations fit the above definitions of two or more of the terms arithmetical, logical, transfer, and red tape, these terms are frequently used loosely to divide the operations of a given routine or of a given instruction code into four mutually distinct classes depending on the primary function intended for the given operation in the case at hand.

**Operation Part.** In an instruction, the part that usually specifies the kind of operation to be performed, but not the location of the operands. See also *Instruction Code*.

**Or Circuit.** Synonym for *or gate*.

**Order.** (1) Synonym for **instruction**. (2) Synonym for **command**. (3) Loosely, synonym for **operation part**. *Note.* The use of "order" in the computer field as a synonym for terms similar to those above is losing favor owing to the ambiguity between these meanings and the more common meanings in mathematics and business.

**Or Gate.** A gate whose output is energized when any one or more of the inputs is in its prescribed state. Thus, this gate performs the function of the logical *inclusive-or*.

**Overflow.** (1) The condition which arises when the result of an arithmetic operation exceeds the capacity of the number representation in a digital computer. (2) The *carry* digit arising from this condition.

**Parallel.** Pertaining to simultaneous transmission of, storage of, or logical operations on the parts of a word, character, or other subdivision of a word, using separate facilities for the various parts.

**Parallel Digital Computer.** One in which the digits are handled in parallel. Mixed serial and parallel machines are frequently called serial or parallel according to the way arithmetic processes are performed. An example of a parallel digital computer is one which handles decimal digits in parallel, although it might handle the bits which comprise a digit either serially or in parallel.

**Parity Check.** See *Check, Forbidden Combination.*

**Partial Carry.** See *Carry.*

**Place.** In positional notation, a position corresponding to a given power of the base. A digit located in any particular place is a coefficient of a corresponding power of the base.

**Point.** In positional notation, the location or symbol which separates the integral part of a numerical expression from its fractional part. For example, it is called the **binary point** in binary notation and the **decimal point** in decimal notation. If the location of the point is assumed to remain fixed with respect to one end of the numerical expressions, a **fixed-point system** is being used. If the location of the point does not remain fixed with respect to one end of the numerical expressions, but is regularly recalculated, then a **floating-point system** is being used. *Note.* A fixed-point system usually locates the point by some convention, while the floating-point system usually locates the point by expressing a power of the base.

**Positional Notation.** One of the schemes for representing numbers, characterized by the arrangement of digits in sequence, with the understanding that successive digits are to be interpreted as coefficients of successive powers of an integer called the **base** or **radix** of the *number system*. In the **binary number system** the successive digits are interpreted as coefficients of the successive powers of the base two just as in the **decimal number system** they relate to successive powers of the base ten. In the ordinary number systems each **digit** is a character which stands for zero or for a positive integer smaller than the base. The names of the number systems with bases from 2 to 20 are: binary, ternary, quaternary, quinary, senary, septenary, octonary (also octal), novenary, decimal, undecimal, duodecimal, terdenary, quaterdenary, quindenary, sexadecimal (also hexadecimal), septendecimal, octodenary, novendenary, and vicenary. The sexagenary number system has a base of 60. The commonly used alternative of saying "base 3," "base 4," etc., in place of ternary, quaternary, etc., has the advantage of uniformity and clarity. *Note.* In the most common form of positional notation the expression

$$\pm a_n a_{n-1} \cdots a_2 a_1 a_0 . a_{-1} a_{-2} \cdots a_{-m}$$

is an abbreviation for the sum

$$\pm \sum_{i=-m}^n a_i r^i,$$

where the point separates the positive powers from the negative powers, the  $a_i$  are integers ( $0 \leq a_i \leq r - 1$ ) called "digits," and  $r$  is an integer, greater than one, called the **base**. *Note 1.* The base of a number is usually indicated by a vertical line following the number with the base,  $r$ , as a subscript. The decimal number 12 in octal and binary codes is written  $12|_{10} = 14|_8 = 1100|_2$ . *Note 2.* For some purposes special rules are followed. In one such usage, the value of the base  $r$  is not constant. In this case, the digits are coefficients of successive products of a non-constant sequence of integers.

**Precision.** The quality of being exactly or sharply defined or stated. A measure of the precision of a representation is the number of distinguishable alternatives from which it was selected, which is sometimes indicated by the number of significant digits it contains. See also *Accuracy*.

**Program.** (1) A plan for the solution of a problem. (2) Loosely, a synonym for *routine*. (3) To prepare a program.

**Automatic programming,** any technique in which the computer is used to help plan as well as to help code a problem. See *Coding*.

**Optimum programming,** improper terminology for minimal latency coding, i.e., for producing a minimal latency routine. See *Routine*.

**Programmed Check.** See *Check, Programmed*.

**Pseudo-Code.** An arbitrary code, independent of the hardware of a computer, which must be translated into computer code if it is to direct the computer.

**Radix.** See *Positional Notation*.

**Random Access.** Access to storage under conditions in which the next position from which information is to be obtained is in no way dependent on the previous one.

**Read.** To acquire information, usually by observing some form of storage. *Note.* Usually a process which can be called reading can also be called writing, depending on the point of view of the observer.

**Real-Time Operation, On-Line Operation, Simulation.** Processing data in synchronism with a physical process in such a fashion that the results of data processing are useful to the physical operation.

**Redundancy Check.** See *Check, Forbidden Combination*.

**Reflected Binary Code.** See Chaps. 11 and 20.

**Regeneration.** (1) In a storage device whose information storing state may deteriorate, the process of restoring the device to its latest undeteriorated state. (2) In a storage device whose information storing state may be destroyed by a readout, the process of restoring the device to its state prior to the readout. This process is commonly known as *rewrite* (after destructive readout).

**Register.** A device capable of retaining information, often that contained in a small subset (e.g., one *word*) of the aggregate information in a digital computer. *Example.* A register in an arithmetic unit as opposed to a cell in storage.

**Register Length.** The number of characters which a register can store.

**Reset.** (1) To restore a storage device to a prescribed state. (2) To place a binary cell in the initial or "zero" state. See also *Clear*.

**Rewrite.** See *Regeneration*.

**Ring Counter.** See *Counter, Ring*.

**Routine.** A set of instructions arranged in proper sequence to cause a computer to perform a desired operation or series of operations, such as the solution of a mathematical problem.

**Executive routine (master routine),** a routine designed to process and control other routines. A routine used in realizing "automatic coding."

**Compiler (compiling routine),** an executive routine which, *before* the desired computation is started, translates a program expressed in pseudo-code into machine code (or into another pseudo-code for further translation by an interpreter). In accomplishing the translation, the compiler may be required to:

**Decode,** to ascertain the intended meaning of the individual characters or groups of characters in the pseudo-coded program.

**Convert,** to change numerical information from one number base to another (e.g., decimal to binary) and/or from some form of fixed point to some form of floating-point representation, or vice versa.

**Select,** to choose a needed subroutine from a file of subroutines.

**Generate,** to produce a needed subroutine from parameters and skeletal coding.

**Allocate,** to assign storage locations to the main routines and subroutines, thereby fixing the absolute values of any symbolic addresses. In some cases allocation may require segmentation.

**Assemble,** to integrate the subroutines (supplied, selected, or generated) into the main routine, i.e., to **adapt**, to specialize to the task at hand by means of preset parameters; to **orient**, to change

relative and symbolic addresses to absolute form; to **incorporate**, to place in storage.

**Record**, to produce a reference record.

**Check Routine**. See *Check, Programmed*.

**Diagnostic Routine**, a specific routine designed to locate either a malfunction in the computer or a mistake in coding.

**General routine**, a routine expressed in computer coding designed to solve a class of problems, specializing to a specific problem when appropriate parametric values are supplied.

**Interpreter (interpretive routine)**, an executive routine which, as the computation progresses, translates a stored program expressed in some machine-like pseudo-code into machine code and performs the indicated operations, by means of subroutines, as they are translated. An interpretive routine is essentially a closed subroutine which operates successively on an indefinitely long sequence of program parameters (the pseudo-instructions and operands). It may usually be entered as a closed subroutine and exited by a pseudo-code exit instruction.

**Minimal latency routine**, especially in reference to serial storage systems, a routine so coded, by judicious arrangement of data and instructions in storage, that the actual latency is appreciably less than the expected random access latency.

**Rerun routine (rollback routine)**, a routine designed to be used in the wake of a computer malfunction or a coding or operating mistake to reconstitute a routine from the last previous **rerun point**, which is that stage of a computer run at which all information pertinent to the running of the routine is available either to the routine itself or to a rerun routine in order that a run may be reconstituted.

**Service routine**, a routine designed to assist in the actual operation of the computer. Tape comparison, block location, certain post mortems, and correction routines fall into this class. Also called *operator routine*.

**Specific routine**, a routine expressed in specific computer coding designed to solve a particular mathematical, logical, or data handling problem.

**Subroutine**. (1) In a routine, a portion that causes a computer to carry out a well-defined mathematical or logical operation. (2) A routine which is arranged so that control may be transferred to it from a master routine and so that, at the conclusion of the subroutine, control reverts to the master routine. Such a subroutine is usually called a closed subroutine. A single routine may simultaneously be both a subroutine with respect to another routine and a master routine with

respect to a third. Usually control is transferred to a single subroutine from more than one place in the master routine, and the reason for using the subroutine is to avoid having to repeat the same sequence of instructions in different places in the master routine.

**Test routine**, a routine designed to show that a computer is not functioning properly.

**RZ, Return to Zero Recording.** See Chap. 19, Sect. 2.

**Scale.** To change the scale (i.e., the units) in which a variable is expressed so as to bring it within the capacity of the machine or routine at hand.

**Selection Check.** See *Check, Selection*.

**Self-Checking Code.** See *Check, Forbidden Combination*.

**Serial.** Pertaining to time-sequential transmission of, storage of, or logical operations on the parts of a word, with the same facilities for successive parts.

**Serial Digital Computer.** One in which the digits are handled serially. Mixed serial and parallel machines are frequently called serial or parallel according to the way arithmetic processes are performed. An example of a serial digital computer is one which handles decimal digits serially although it might handle the bits which comprise a digit either serially or in parallel. Antonym: *Parallel Digital Computer*.

**Set.** (1) To place a storage device in a prescribed state. (2) To place a binary cell in the "one" state.

**Sexadecimal.** See *Positional Notation*.

**Shift.** Displacement of an ordered set of characters one or more places to the left or right. If the characters are the digits of a numerical expression, a shift may be equivalent to a multiplication by a power of the base.

**Cyclic Shift.** An operation which produces a word whose characters are obtained by a cyclic permutation of the characters of a given word.

**Sign Digit.** A character used to designate the algebraic sign of a number.

**Simulation.** See *Real-Time Operation*.

**Single-Address Code.** See *Instruction Code*.

**Sort.** To arrange items of information according to rules dependent upon a key or field contained by the items.

**Standing-on-Nines Carry.** See *Carry*.

**Storage.** (1) The act of storing information. (See also *Store*.) (2) Any device in which information can be stored, sometimes called a *memory* device. (3) In a computer, a section used primarily for storing information. Such a section is sometimes called a *memory* or a *store* (British). *Note:* The physical means of storing information may be

electrostatic, ferroelectric, magnetic, acoustic, optical, chemical, electronic, electrical, mechanical, etc., in nature.

**Storage Capacity.** The amount of information that can be simultaneously retained in a storage (or memory) device, often expressed as the number of words that can be retained (given the number of digits, and the base, of the standard word). When comparisons are made among devices using different bases and word lengths, it is customary to express the capacity in *bits*. This number is obtained by taking the logarithm to the base 2 of the number of distinguishable states in which the storage can exist. *Note.* The "storage (or memory) capacity of a computer" usually refers only to the principal internal storage section.

**Store.** (1) To retain information in a device from which it can later be extracted. (2) To introduce information into such a device. (3) British synonym for storage (3).

**Subroutine.** See *Routine*.

**Switch.** A device for effectively making, breaking, or changing the path of information flow. See also *Matrix (Switch)*.

**Ternary.** See *Positional Notation*.

**Test Routine.** See *Routine*.

**Three-Address Code.** See *Instruction Code*.

**Track.** That portion of a moving-type storage medium which is accessible to a given reading station; e.g., as on film, drum, tapes, or disks. See also *Band*.

**Transcriber.** Equipment associated with a computing machine for the purpose of transferring input (or output) data from a record of information in a given language to the medium and the language used by a digital computing machine (or from a computing machine to a record of information).

**Transfer.** (1) To transmit, or copy, information from one device to another. (2) To transfer control. (3) The act of transferring.

**Transfer Check.** See *Check, Transfer*.

**Transfer Control.** Synonym for *jump*.

**Translate.** To change information (e.g., problem statements in pseudo-code, data, or coding) from one language to another without significantly affecting the meaning.

**Translator.** A network or system having a number of inputs and outputs and so connected that signals representing information expressed in a certain code, when applied to the inputs, cause output signals to appear which are a representation of the input information in a different code. Sometimes called *matrix*.

**Trunk.** A path over which information is transferred; a bus.

**Unconditional Jump.** See *Jump*.

**Unconditional Transfer of Control.** Synonym for *unconditional jump*.

**Unit.** A portion or subassembly of a computer which constitutes the means of accomplishing some inclusive operation or function, as *arithmetic unit*.

**Verification.** The process of automatically checking the results of one data recording process against the results of another data recording process for the purpose of reducing the number of errors in data transcription. See also *Check*.

**Verifier.** A device on which a manual transcription can be verified by comparing a retranscription with it character by character as it is being retranscribed.

**Volatile.** A term descriptive of a storage medium in which information cannot be retained without continuous power dissipation. *Note.* Storage devices or systems employing nonvolatile media may or may not retain information in the event of planned or accidental power removal.

**Williams Tube Storage.** A type of electrostatic storage.

**Word.** An ordered set of symbols which is the normal unit in which information may be stored, transmitted, or operated upon within the computer.

**Word Time.** Synonym for *minor cycle*.

**Write.** To introduce information, usually into some form of storage. See also *Read*.

**Zero Suppression.** The elimination of nonsignificant zeros to the left of the integral part of a quantity before printing operations are initiated; a part of editing.

## REFERENCES

1. IRE; *Standards on Electronic Computers: Definitions of Terms, 1956*. 56 IRE 8.51. *Proc. I. R. E.*, 44, 1166-73 (1956).
2. *First Glossary of Programming Terminology*, Association for Computing Machinery, New York, 1954.

# DIGITAL COMPUTER PROGRAMMING

## B. DIGITAL COMPUTER PROGRAMMING

2. Programming and Coding, *by John W. Carr III*



## Programming and Coding

*John W. Carr III*

1. Nature of Programming	2-01
2. Numbers and Scale Factors	2-12
3. Number Conversion Tables	2-26
4. Program Structure and Flow Diagrams	2-44
5. Machine Logic	2-53
6. Instruction Logic of Common Computers	2-63
7. Traditional Programming Techniques	2-128
8. Automatic Programming: Development and Objectives	2-155
9. Automatic Programming: Assembly Programs	2-163
10. Automatic Programming: Subroutines, Subroutine Generators, Utility Programs, and Integrated Systems	2-167
11. Automatic Programming: Languages, Compilers, and Translators	2-186
12. Automatic Programming: The IT Translator, Translator Construction	2-200
13. Automatic Programming: A Soviet Algebraic Language Compiler	2-228
14. Automatic Programming: Interpreters	2-234
15. Automatic Programming: Recursive Languages	2-244
16. Logical Programming	2-246
17. Microprogramming	2-251
18. Programs for Maintenance of Equipment	2-258
19. Programming with Natural Language	2-259
Literature, Acknowledgments, and References	2-260

### I. NATURE OF PROGRAMMING

#### **Problem Solving**

**Characteristics of Problem Solving.** Basically, the general problem which the digital computer programmer must face is the *solution of*

*problems.* The solution of problems with automatic information processing machines employs two tools:

1. Arithmetic (basically elementary descriptive number theory or the manipulation of integers). The problem of evaluating and understanding the arithmetic portion of the problem-solving job is *numerical analysis* (see Vol. 1, Chap. 14).

2. Formal logic, or the making of decisions on the basis of elemental pieces of basic information. In this chapter the emphasis will be placed on the non-numerical portions of the problems.

**Limitations.** For each machine developed to solve problems, a new problem will be found to strain its resources. Problems tend to outgrow the ability of the present man-machine combinations to construct models of the problem, within the computers, for solution. The important limitations imposed by general purpose computers are:

1. *Logical Portion of the Solution Process.* Vast problems on classical standards, describing the interaction of men, machines, and nature in a generally unpredictable (except statistically) fashion, must be solved by mapping their description into digital information machines. *Example.* The problem of commercial or military aircraft traffic control.

2. *Multidimensional Arithmetic Problems.* *Example.* Multidimensional problems in partial differential equations, as described by numerical analysis.

3. *Succession of Related or Even Nonrelated Problems.* Many problems involve organizing the solution of not one problem (large or small), but of a number of problems coming from different disciplines, so as to make use of previous knowledge accumulated and stored in the automatic digital computer.

4. *Many Problems Are Not Well Defined.* These experimentally and theoretically undefined problems must nevertheless have solution processes developed for them. *Examples.* Air traffic control, the solution of partial differential equations, and the control of management and information functions in a large organization.

**Approaches to Programming.** The two approaches to problem solving with machines can be divided into (1) *hand programming*, in which programs are produced in detail by individual practitioners in a manner that is more an art than a science, and (2) *automatic programming*, with the machine taking over most of the routine decisions.

**Automatic Programming.** This area encompasses two basic problems:

1. *The Problem of Languages.* If humans are to keep up with the voracious input capacity of the digital computers, then languages built

for the humans, and not the machines, must be developed and put into operation. This involves the creation of translators, techniques for using them, and, finally, a theory of such formal translators. Multimachine installations, with each machine having a separate language of its own, require a unified language for most efficient use. The development of "automatic problem solution" requires formalism, interchangeability of procedures, and compatibility of languages if it is to become a true discipline in the scientific sense.

2. *The Problem of Files.* Each problem solved on a computer should be considered as part of the structure of a generalized body of knowledge, either in a certain domain, or in the overall domain of problem solving itself. The body of knowledge about automatic problem solving so laboriously collected by the users of these machines must be set up so that each machine itself can store, generate, and accept problem procedures generated not only by human beings, but also by the machine itself and by other machines. The final long-range goal in this problem of files and information retrieval, in relationship to what is presently known as "computer programming," is therefore not to generate programs or algorithms for digital computers by human beings, but instead to develop programs or algorithms for the generation, improvement, filing, retrieval, and combination of other algorithms by the computers themselves.

### **Basic Concepts of Programming**

The process of preparing a program or set of coded instructions to solve a problem on a high-speed electronic digital computer has been more of an art than a science.

**Programming Procedure.** A programmer is called upon to map an external problem from the field of science, engineering, business, or other field of origin, into the logical structure of a general purpose digital computer by preparing, or overseeing the preparation of, a list of instructions to be performed in sequence by the computer. He may do this himself, step by step; but if he is able to take advantage of certain procedures already set up, he may be aided in many cases by the use of the machine itself. This allows incorporation of similar or associated procedures that he or other persons have evolved previously.

The usual procedures for the programmer are:

1. *Determine the problem,* since usually it is posed in such an original form that it is necessary to determine the actual statement of the problem before proceeding further.

2. *Analyze the problem* in whatever external language it has been presented—algebra, pictorialized "road map," handbook of organizational

procedures, an actual physical or organizational structure, or perhaps only inputs allowed and outputs required.

3. *Break the problem down* into logical "atomic fragments," and synthesize a resultant program or algorithm in terms of a different language, that of the machine.

**Basic Computer Organization.** The heart of a scientific calculating system is the large-scale general purpose, high-speed stored program electronic digital computer (see Ref. 19). These computers are patterned after the traditional sequential use of arithmetic, with internal decision making based on partial results.

**Block Diagram of Computer Solution.** Figure 1 describes a flow diagram which may be considered an example of a typical process of

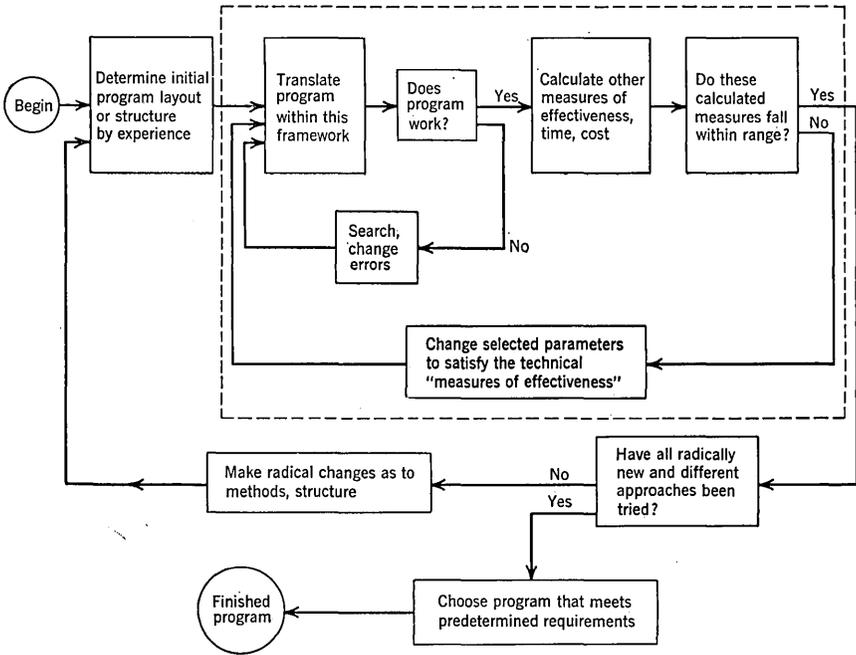


FIG. 1. Block diagram of computer solution.

solution of a scientific, business, or real-time control problem. In Fig. 1 all the blocks within the dotted lines can be handled automatically by a high-speed digital computer. Those blocks outside the dotted lines have not yet been successfully attacked by machine techniques. They require human intervention. Thus, any programming procedure embeds the electronic digital computer in a larger data processing system with interplay between the automatic computer on the one hand and the mathematician or engineer or business systems analyst on the other.

**Programming System.** The programmer, whatever his region of application, is faced with a classical problem, the same that faces either the computer designer or any designer: the formal description of a process or procedure. The steps are as follows:

1. Determination of a descriptive statement of the problem in a *formal language*, symbolic or pictorial, mathematical or graphical.

2. Determination of a basic (canonical) set of structures or pieces into which he will decompose his formal description. In computer programming these are called instructions, or subroutines, or statements, depending on the level of language involved.

3. Decomposition of the original problem into these basic pieces.

4. Optimization of the performance characteristics of the problem under some measure of effectiveness: fastest program, smallest amount of man and machine time, storage within some predetermined maximum.

5. Analysis of the performance. Determination of the behavior of the final program for a "satisfactory" selection of input parameters.

**Expansion of the Controlled System.** At the present time, at most the two boxes, "Translate program" and "Does program work," shown in Fig. 2, are being handled more or less automatically by the machines

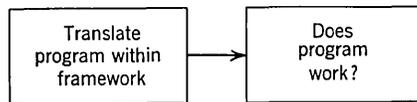


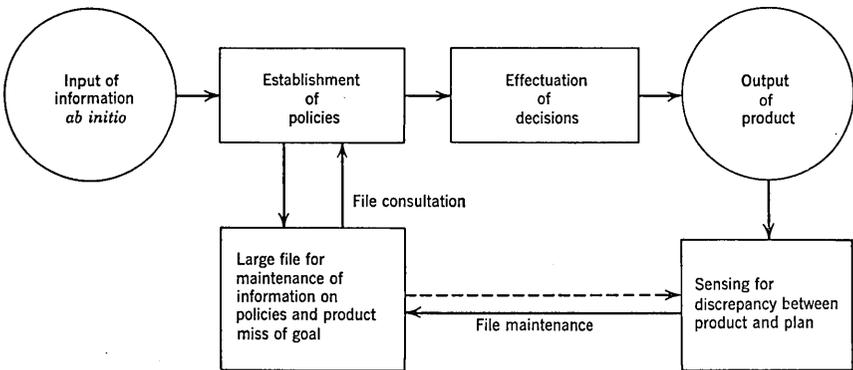
FIG. 2. Operations handled automatically by the machine.

themselves. The general trend of the discipline now being evolved is to extend the areas of the dotted lines of Fig. 1 as far as possible.

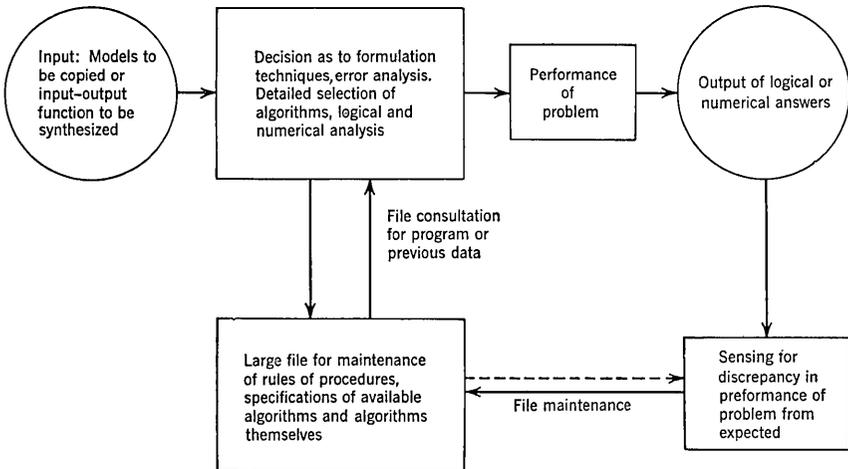
**Self-Improvement in Systems.** An ultimate desire in a system is that it be "self-improving." A general large control system is shown in Fig. 3a which is to improve its performance in time. Figure 3b shows an information machine system having similar objectives. The development of unified programming systems indicates recent trends toward complete mechanization (Ref. 97). One system stores on magnetic tape all programs being written, corrected, and performed *within the computer*.

**Measures of Performance.** If one therefore considers computer programming as embedded within a large structure, one can determine several pertinent categories of measurement of the performance of programs prepared for use on a machine.

1. Degree of generalization and abstraction of the program. Within bounds, a program that can handle a large variety of cases is much more useful than one that solves only one specific case. Experience has shown



(a)



(b)

FIG. 3. Computer system. Dotted lines indicate information flow is less heavy than for solid lines. (a) Large control system. (b) Information machine system.

that existence of a sufficiently general computer program will generate the solution of new cases not previously considered.

2. Addition to the body of knowledge of the system. A new computer program is worth much more if it adds new techniques to an organized laboratory programming system.

3. Standard conventions. By use of standard conventions the programmer can use the fruits of the labors of others who have obeyed the same conventions.

4. Cost, elapsed time, human time, machine time. These are the measures which are ordinarily uppermost in the computer user's mind

and on which many short-range decisions are based. In the long run, the three previous criteria may prove much more important.

### **Characteristics of Modern Problem Solving**

In using a general purpose digital computer there is a need for absolute preciseness required in transcription of data, writing of detailed instructions, operation of the machines. These areas are important, but represent the routine aspects of problem solving.

More fundamental are the characteristics of the formalized solution of problems presently required by the information machines as follows:

1. *Emphasis on structure rather than number.* Portions of the problem are arithmetic, but the most difficult part of the problem-solving procedure is the readying of the logical decision-making structure.

2. *Difficulty in making intuitive decision processes precise.*

3. *Requirement of the problem-solving system to adapt to the changing structure of the problem.* Most problems of modern science and industrial society are not static, but rather dynamic.

4. *Extreme combinatorial complexity.*

5. *Importance of correct manipulation of data.* Data processing machines can perform a far greater number of operations between errors than can a human being. This means that in smaller problems, many of the customary checks required to guarantee accuracy of solution can be ignored. However, since the size of problems has kept pace with the mean free path between errors, the need of thorough error-detecting and error-correcting procedures remains just as important on the larger problems (see Chap. 13). Until such procedures can be included automatically in an overall integrated system (as has been attempted, for example, by Carr *et al.*, Ref. 18), this property of machines of accuracy, valid for smaller problems, must still be a responsibility of the human programmer.

**The Programmer and Machine Design.** Onto the programmer falls the responsibility of bridging the gap between human and machine performance in the first three characteristics, as well as of taking advantage of the machine's penchant for spectacular performance in the last two cases. New programming procedures, when successful, have been built later into hardware. *Examples* are automatic indexing, automatic number conversion, simplified input-output, simplified and extended languages.

### **The Structure of Machine Programming**

It is important to understand just what is being done when a problem is "put on" the machine. This description may be formulated in mathematical terms.

**Mathematical Representation of the Machine.** As a simplification, consider first the electronic portion of the machine and assume a binary machine. At the beginning of a problem, there is a certain "binary function" stored in the machine, with each binary storage position containing either a 0 or 1. At the end of the problem, the machine contains a second binary function, which generally is not the same.

Thus, in mathematical language, one can consider the machine proper as a method of forming correspondence between one machine function and another (see also Chap. 17, Sect. 5). In mathematical notation label the binary storage positions in the machine by some index and call the total set of such positions  $X$ . If there are  $N$  storage positions in the machine, there are  $2^N$  possible "machine functions." One such machine function will be called  $f(x)$ . The set of all such machine functions will be called  $F$ . (See Vol. 1, Chap. 1, Sets and Relations.) Then the electronic elements of the machine form an operator  $M_E$  over the set of all functions  $f(x)$ . Figure 4 shows a picture of the machine's action.

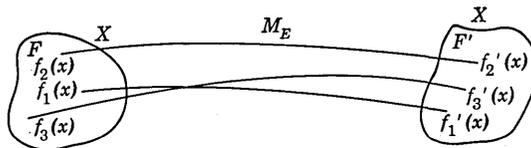


FIG. 4. Machine action in symbolic form.

The operator  $M_E$  can be extended to include the entire machine, with its input and output equipment. Such a function will be called the automatic machine or  $M$  operator, which transforms functions  $f(x)$ . The basic set  $X$  then includes not only all binary storage positions in the machine, but also binary storage positions on the paper, magnetic tape, and typewriters attached to it. The function space  $F$  now includes all functions over the extended set.

**Definition of Programming.** Problems to be solved by the machine must be similar in structure to the machine itself. Therefore, any problem to be solved on the machine must be in a sense "mathematically similar" to the machine structure just outlined. A problem to be put on the machine will be defined as some mapping or transformation or operator. A similar picture to the above can be drawn (see Fig. 5). Some set  $N$  with index  $\xi$ , a set of functions  $\phi(\xi)$  over  $N$ ,  $\Phi$ , and a transformation  $P$ , which makes a correspondence between the set of functions  $\Phi$  and another set  $\Phi'$ .

This "problem space" could be any type of function space. However, only for certain problems (in fact a very small minority of possible ones)

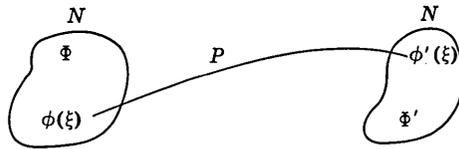


FIG. 5. Problem in symbolic form.

can one hope to get an exact reproduction of this problem space in a machine. This problem of the mapping of problems into a machine is called *programming*.

**The Art of Programming.** This is the job of translating the set of functions  $\Phi$  into  $F$ , the set of functions  $\Phi'$  into  $F'$ , and the problem operator  $P$  into the machine operator  $M$ . This is shown in Fig. 6.

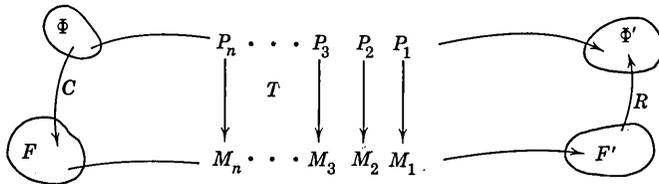


FIG. 6. Programming in symbolic form.

Programming a problem consists of forming a correspondence between the original problem functions and the machine functions  $f(x)$ , and between the problem operator  $P$  and the machine operator  $M$ . The mapping of functions is done by some prearranged convention, and the mapping of the operator  $P$  into  $M$  is done "in the small" or piecemeal. Thus the problem operator  $P$  is broken up into successive operators  $P_1, P_2, P_3, \dots P_n$ , as shown in Fig. 6.

In Fig. 6,

$C$  = a *conversion* operator converting problem functions into machine functions

$P_i$  = canonical components of the problem operator  $P$

$T$  = the operator transformation taking the  $P_i$  into corresponding  $M_i$

$R$  = a *reconversion* operator carrying machine functions back into problem functions.

Usually the operator  $C$  will carry  $\Phi$  into  $F$  in a fashion that is not 1:1. This means that a function in  $F$  may correspond to many functions in  $\Phi$ . *Example.* The mapping of irrational and rational numbers into finite length machine numbers. All the numbers within a certain interval

on the real line are made to correspond to one number in the machine. This is where the celebrated roundoff problem arises.

Although the numbers or functions in  $F$  may have exact counterparts in  $\Phi$ , the functions in  $\Phi'$  under the mapping  $R^{-1}$  in most cases do not have an exact counterpart in  $F'$ . Thus, except in special cases, the reconversion operator  $R$  cannot be the inverse mapping of  $C$ , that is,

$$R \neq C^{-1}.$$

This is due to the basic nature of any machine operator  $M$ , which must of necessity be restricted to a finite operator, while the problem operators of mathematics of most scientific models generally operate on an infinity of elements.

### Problem Solution

In most high-speed computing machines, the machine operator  $M$  and its canonical components  $M_i$  are stored as functions over the machine set  $X$ , and since they are therefore able to operate on each other, they introduce a further complication. This signifies merely that one natural canonical decomposition of  $M$  is into canonical operators where each is now a function of the preceding  $M_j$ ,  $j < i$ . Thus the state of the instruction portion of any program is a function of all the instruction program at each preceding stage in the machine's path of control.

The solution problem thus consists of a number of connected problems: encoding, programming, conversion, and reconversion.

**Encoding Problem.** The decision of just what functions  $f(x)$  will correspond to a given  $\phi(x)$  is called the encoding problem. Every mathematical problem can be broken up into infinitesimal or basic logical operations. The mathematics involved in encoding is essentially the same as the problem of encoding mathematical objects from a given group or algebraic structure over into binary form. Shannon, Wiener, and others (Ref. 110), have applied this theory to certain communication problems, but it has not been directly applied to machine problems.

There are two unsolved problems regarding encoding: minimization and self-encoding.

*Minimization.* For a given engineering criterion, can some function  $F$  for encoding  $P_i$  into  $M_i$  be minimized (either uniquely or not); i.e., does  $\min F(I, T_e, T_p, C, \dots)$  exist, where

- $I$  = amount of binary information storage
- $T_e$  = time required for encoding
- $T_p$  = time required in program operation
- $C$  = variable including engineering cost

It is very probable that there are some workable criteria that may improve the present methods of encoding.

*Self-Encoding.* Can an algorithm, or regular process, be developed by which one can arrive at even a near-minimal encoding system? If this can be done, it is a basic step toward the automation of problem solving. Once a proper algorithm is developed, it should be possible for the machine to decide on its own machine code; i.e., just how many and what combination of binary digits will give the most efficient machine representation of a basic canonical operator  $P_i$ . In other words (see Fig. 6), the machine should be allowed to engineer its own construction, once it knows what the decomposition of the operator  $P$  is to be in terms of some basic set of  $P_i$ 's.

Present machines are not self-encoding and can be called fixed code machines.

If better fixed-code machines or self-encoding machines are to be built, this is a combined engineering and logical problem, which has to do with a decision as to what is the best representation in the machine for the operators  $P_i$ , that is, just what shall be the form of the  $M_i$ .

**Programming.** This is the job of making the canonical decomposition ( $P_i$ ) and then actually performing the translation operation  $T$ . The decomposition of  $P$  into canonical parts has so far been a hit-or-miss proposition. There have been no overall logical studies of just how operations  $P$  should be decomposed and in what language they should be expressed. Mathematics has not so far been constructed for ease in translation to the basic binary functions of high-speed machines. Hence, programmers have turned to the much more easily translatable language of logic for translation purposes, mainly because the translation operation  $T$  has to be performed by human beings.

There is no reason why the machine cannot be "taught" to perform the operation  $T$ . The first steps are the writing and testing of the libraries of subroutines. However, it is apparent that since the structure of the problem  $P$  is going to be similar to the structure  $M$ , the previously mentioned theory of the best decomposition of  $M$  may turn out to be intimately related to a quantitative information theory of operational mathematics.

**Conversion and Reconversion.** The operations  $C$  and  $R$  are the conversion and reconversion programs that translate the functions  $\phi(\xi)$  into the functions  $f(x)$  of the machine. The operator  $C$  takes the outside language and translates its basic elements into binary functions inside the machine.

The operator  $R$  does the reverse job. If the translation  $C$  has been a many-to-one process, where there is ambiguity outside but not inside

the machine, the operator  $R$  will not be a true inverse. It is very probable then, that  $C$  and  $R$  will be made to be approximately 1 to 1 operators, the inverse of one another. In present machines, the beginnings of a workable operator  $C$  are the conversion and translation read-in programs, while the operator  $R$  is being approximated by the various print-out and post-mortem programs.

**Completely Automatic Programming.** One can now outline just what a completely automated machine will do when a particular problem is presented to it. This machine will perhaps contain a very flexible set of arithmetic and computational elements. It will contain stored in it, in some form or another, a great many rules and algorithms by which it is to perform the various operations. The following steps will occur.

1. It will be presented with a problem operator  $P$ , in some outside language which has been previously decided upon.

2. It will then translate  $P$  into machine language and, by some algorithm, decompose  $P$  into some predetermined set of  $P_i$ .

3. It will enumerate each of the  $P_i$  and determine what shall be the best encoded form for the corresponding  $M_i$ .

4. It will then translate the  $P_i$  into the  $M_i$  it has decided upon, by means of the operator  $T$ .

5. Then by a given rule  $C$ , it will translate whatever functions  $\phi(x)$  are to be operated on over into the machine form.

6. It will then perform the basic machine operation and obtain the corresponding functions  $f'(x)$ , which will be reconverted by the operator  $R$  into the "outside language" form.

Provided the proper rules of operation can be found, all this can be completely automatic. The implications on construction of machines themselves are the following:

1. The mathematical machine of the future may well be an even more general purpose machine than now, with a very large amount of high-speed storage and a very flexible set of arithmetical elements, which it will control and combine itself. (This allows for self-correcting by a machine.)

2. If the "programming" process is made nearly automatic, the largest of bottlenecks may come in the final transition of output information from the machine to the human.

## 2. NUMBERS AND SCALE FACTORS

**Logic and Numbers.** Mathematical logicians, such as Whitehead and Russell (Ref. 105), have shown that the concept of numbers can be built up from the fundamentals of logic. Turing (Ref. 101), a logician

and early computing machine designer, showed that a very simple machine, performing very elementary logical operations, can represent every automatic computing device within a large class (see Chap. 31). On the other hand, the relationship between the truth tables of the propositional calculus and the binary number system and the mapping of more advanced logical systems by Gödel and others into the integers shows that the concept of number and logical process are closely inter-related.

**Number Systems.** Number systems and numerical notations used, both by human beings and by general purpose digital computers, have several basic properties fundamental to their use: (1) base or radix of the system, (2) method of representation of negative numbers in the system, (3) precision or amount of information available in the representation of a number, and (4) range of application of the notation over the range of real numbers.

**Number System Base.** The *base* or *radix* of a number system is set by the number of digits or markers in the representation. Most number systems used by computers are of constant radix, such as the ordinary decimal system, as compared with variable radix systems, such as the Roman numeral notation. Each digit is a distinguishable symbol representing one of the first several consecutive integers or natural numbers, starting at zero. The number of such markers in the numerical notation gives the base or radix.

Most machines use number systems with base  $2^k$ , where  $k$  is most often 2, 3, 4, or 5, for ease in external or internal manipulation of numbers. Such bases are the base 2 (binary), base 8 (octal or octenary), base 16 (hexadecimal or sexadecimal), and base 32 systems. Machines usually use the binary notation as the fundamental internal number system, and the secondary binary-like bases as shorthand notations for input and output.

**Notation.** In any of these systems, digits appear in a series of consecutive positions or orders, which are usually presented on paper from left to right, proceeding from the highest order digit down to the lowest. Thus the notation for a number  $n$  expressed in the base  $r$

(1)  $n_r = a_j r^j + a_{j-1} r^{j-1} + \cdots + a_1 r^1 + a_0 r^0 + a_{-1} r^{-1} + \cdots + a_{-m} r^{-m}$ ,  
would be

$$a_j a_{j-1} \cdots a_1 a_0 . a_{-1} \cdots a_{-m},$$

where the  $a_i$  could be any of the  $r$  digits or markers of the notation, and the values of the integers  $j$  and  $m$  are dependent on the size of the number and the amount of information or precision it is to convey. The

period or radix point inserted between the digits is a device for noting the position of the  $a_0$  and  $a_{-1}$  markers.

Thus the integer representing the number of days in the year would be written in the decimal notation as

$$n = 365|_{10},$$

since 
$$n = 3 \times 10^2 + 6 \times 10^1 + 5 \times 10^0:$$

and in the binary notations as

$$n = 101101101|_2$$

since

$$n = 1 \times 2^8 + 0 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0.$$

### Conversion of Numbers from One System to Another

**Need for Conversion.** There is frequently a need to convert from decimal language to binary, octal, or hexadecimal number systems. Two conditions frequently encountered are:

1. Conversion from a first number system to a second number system by using the arithmetic of the first number system. *Example.* This occurs on output when conversion must be made from the internal machine number system to the external number system, with the machine instructed to perform the conversion in the internal number system.

2. Conversion from a first number system to a second number system by using the arithmetic of the second number system. *Example.* Computer output may be in its internal language, and a human being must convert to decimal. In this case the conversion must be performed in the external language, generally the decimal number system.

**Conversion in the Arithmetic of the Original System.** The number  $n \geq 0$  is expressed in an original number system in the following form:

$$(2) \quad n = \sum a_i r_1^i, \quad i = k, k-1, \dots, 1, 0, -1, -2, \dots$$

In the second number system

$$(3) \quad n = \sum a'_i r_2^i, \quad i = k', k'-1, \dots, 1, 0, -1, -2, \dots$$

It is to be converted by using the arithmetic of the first number system. To perform the conversion it is easiest to split the number given in eq. (2) into two parts, an integer part  $N$ , and a fractional part  $F$ , each one to be treated separately, i.e.,  $n = N + F$ .

Working first on the integer part, one divides  $N$  by  $r_2$  in the arithmetic of the first number system to obtain  $a'_0$  in the following fashion:

$$(4) \quad a'_0 = N - [N/r_2]r_2,$$

where  $[ \ ]$  indicates the "integral part of." Then

$$(5) \quad N^{(0)} = N,$$

$$(6) \quad a'_i = N^{(i)} - [N^{(i)}/r_2]r_2,$$

$$(7) \quad N^{(i+1)} = [N^{(i)}/r_2].$$

At each stage the reduced expression for the succeeding integer is given by the integer part of the quotient obtained by division by the radix  $r_2$ .

In converting the fractional part  $F$  of the number, instead of division, multiplication by the power of the radix  $r_2$  is performed at each step, and the integer part of the remainder yields the coefficient of the corresponding term. The general formula for this process is as follows:

$$(8) \quad F^{(0)} = a'_{-1}r_2^{-1} + a'_{-2}r_2^{-2} + a'_{-3}r_2^{-3} + \cdots,$$

$$(9) \quad a'_i = [r_2 F^{(i)}],$$

$$(10) \quad F^{(i+1)} = r_2 F^{(i)} - a'_{-1}.$$

**EXAMPLE 1.** Conversion of a mixed number given in the decimal notation into the corresponding binary number by using decimal arithmetic.

$$n = 97.975 = N^{(0)} + F^{(0)},$$

where  $N^{(0)} = 97$ ,  $F^{(0)} = 0.975$ .

$$\begin{aligned} a'_0 &= 97 - [97/2]2 = 1, \\ N^{(1)} &= [97/2] = 48, \\ a'_1 &= 48 - [48/2]2 = 0, \\ N^{(2)} &= [48/2] = 24, \\ a'_2 &= 24 - [24/2]2 = 0, \\ N^{(3)} &= [24/2] = 12, \\ a'_3 &= 12 - [12/2]2 = 0, \\ N^{(4)} &= [12/2] = 6, \\ a'_4 &= 6 - [6/2]2 = 0, \\ N^{(5)} &= [6/2] = 3, \\ a'_5 &= 3 - [3/2]2 = 1, \\ N^{(6)} &= [3/2] = 1, \\ a'_6 &= 1 - [1/2]2 = 1, \\ N^{(6)} &= [1/2] = 0. \end{aligned}$$

Therefore

$$N^{(0)} = 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0,$$

or as it is commonly written in the binary notation

$$N^{(0)} = 1100001|_2.$$

Similarly,

$$\begin{aligned} F^{(0)} &= 0.975, \\ a'_{-1} &= [2(0.975)] = 1, \\ F^{(1)} &= [2(0.975)] - 1 = 0.95, \\ a'_{-2} &= [2(0.95)] = 1, \\ F^{(2)} &= [2(0.95)] - 1 = 0.90, \\ a'_{-3} &= [2(0.90)] = 1, \\ F^{(3)} &= [2(0.90)] - 1 = 0.80, \\ a'_{-4} &= [2(0.80)] = 1, \\ F^{(4)} &= [2(0.80)] - 1 = 0.60, \\ a'_{-5} &= [2(0.60)] = 1, \\ F^{(5)} &= [2(0.60)] - 1 = 0.20, \\ a'_{-6} &= [2(0.20)] = 0, \\ F^{(6)} &= [2(0.20)] - 0 = 0.40, \\ a'_{-7} &= [2(0.40)] = 0, \\ F^{(7)} &= [2(0.40)] - 0 = 0.80. \end{aligned}$$

At this point the computation can be terminated, since  $F^{(3)} \equiv F^{(7)}$ , and the result is therefore a repeating binary fraction. Hence,

$$F = 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} + 1 \cdot 2^{-5} + 0 \cdot 2^{-6} + 0 \cdot 2^{-7} \dots$$

The number  $n$  can therefore be written

$$n = 97.975|_{10} = 1100001.11111001111100 \dots |_2$$

**EXAMPLE 2.** Conversion of a binary number, the integer  $I$ , into a decimal by using binary arithmetic inside a computer. The binary numbers obtained that are equivalent to one of the ten decimal digits 0 through 9 upon completion must be printed out of the computer by some procedure such as the use of binary-coded decimal through an output typewriter device.

Given  $I = N^{(0)} = 10110010101|_2$ , divide by  $10|_{10} = 1010|_2$ , successively, [ ] again mean the integral part of.

$$\begin{aligned} a'_0 &= 10110010101 - [10110010101/1010]1010 \\ &= 10110010101 - (10001110)(1010) \\ &= 1001|_2 = 9|_{10}, \\ N^{(1)} &= [10110010101/1010] = 10001110, \end{aligned}$$

$$\begin{aligned}
 a'_1 &= 10001110 - [10001110/1010]1010 \\
 &= 10001110 - (1110)1010 \\
 &= 0010|_2 = 2|_{10}, \\
 N^{(2)} &= [10001110/1010] = 1110, \\
 a'_2 &= 1110 - [1110/1010]1010 \\
 &= 0100|_2 = 4|_{10}, \\
 N^{(3)} &= [1110/1010] = 1, \\
 a'_3 &= 1 - [1/1010]1010 \\
 &= 0001|_2 = 1|_{10}.
 \end{aligned}$$

Therefore  $I = 10110010101|_2 = 1429|_{10}$ .

A similar example could be carried out for a binary fraction.

**Conversion in the Arithmetic of the Second System.** When  $n$  is expressed in the form of eq. (1), perform the operations on the right-hand side of the equation in the  $r_2$  system. This can be done by two methods: (a) simple multiplication of the powers which may be stored inside the computing machine, followed by addition or (b) the synthetic division or polynomial multiplication process which is given in the following form:

$$(11) \quad N|_{r_2} = (\cdots ((a_k r_1 + a_{k-1})r_1 + a_{k-2})r_1 + \cdots + a_0)|_{r_2}$$

or

$$(12) \quad p_1 = a_k, p_{i+1} = p_i r_1 + a_{k-i}, p_{k+1} = N \quad (p_i = \text{successive iterants}),$$

and

$$(13) \quad F|_{r_2} = (\cdots ((a_{-j}/r_1 + a_{-(j-1)})/r_1 + a_{-(j-2)})/r_1 + \cdots + a_{-1})/r_1|_{r_2}$$

where the value  $j$  is picked at the beginning.

EXAMPLE 1. Conversion of a number from binary to decimal by using the decimal number system.

$$n = 10110010101|_2.$$

(a) Adding successive powers of 2 in the decimal system, eq. (6),

$$\begin{aligned}
 n &= 2^0 + 2^2 + 2^4 + 2^7 + 2^8 + 2^{10} \\
 &= 1 + 4 + 16 + 128 + 256 + 1024 = 1429|_{10}.
 \end{aligned}$$

(b) Using the synthetic division process of eq. (7),

$$n = ((((((((((1(2)+0)2+1)2+1)2+0)2+0)2+1)2+0)2+1)2+0)2+1)2+0)2+1,$$

with successive iterants  $p_i$

$$2, 5, 11, 22, 44, 89, 178, 357, 714, 1429, n = 1429|_{10}.$$

In the first procedure, eq. (a), it was assumed that the numerical equivalents of the power of 2 were stored in some fashion in the external decimal system.

EXAMPLE 2. For fractions, the two procedures are, for

$$n = 0.10111001|_2:$$

(a) Adding successive powers,

$$\begin{aligned} n &= 1(2^{-1}) + 0(2^{-2}) + 1(2^{-3}) + 1(2^{-4}) + 1(2^{-5}) + 0(2^{-6}) + 0(2^{-7}) + 1(2^{-8}) \\ &= 0.5 \quad + 0.125 \quad \quad \quad + 0.0625 + 0.03125 \quad \quad \quad + 0.00390625 \\ &= 0.72265625|_{10}. \end{aligned}$$

(b) Synthetic division,

$$n = ((((((1(2^{-1}) + 0)2^{-1} + 0)2^{-1} + 1)2^{-1} + 1)2^{-1} + 1)2^{-1} + 0)2^{-1} + 1)2^{-1},$$

which yields as iterants

$$\begin{aligned} 0.5, 0.25, 0.125, 0.5625, 0.78125, 0.890625, 0.4453125, 0.72265625, \\ n = 0.72265625|_{10}. \end{aligned}$$

*Note.* For fractional numbers the second procedure of synthetic division, necessitating successive divisions by powers of two, does not work as well because of the difficulties of the fractions involved, and because the procedure moves from the lowest power of the radix upward.

EXAMPLE 3. The same procedure can be used in the conversion of decimal numbers to binary by using the binary system, as in the case of conversion of straight binary-coded decimal numbers to binary inside the machine. Convert  $n = 1429|_{10}$ .

(a) Adding successive powers,

$$\begin{aligned} n &= (1 \cdot 10^3 + 4 \cdot 10^2 + 2 \cdot 10^1 + 9 \cdot 10^0)|_{10} \\ &= 1(1111101000) + 100(1100100) + 10(1010) + 1001(1) \\ &= 10110010101|_2. \end{aligned}$$

(b) Synthetic division,

$$\begin{aligned} n &= ((1(1010) + 100)1010 + 10)1010 + 1001 \\ &= 10110010101|_2. \end{aligned}$$

**A Trick Method.** A trick method of conversion can be used when conversion is done in the system having the lower radix. The procedure makes use of a "magic number" which is the difference between the two bases, determined as follows:

1. If  $r_1 < r_2$ , the magic number is  $r_2 - r_1$ , and conversion is in the  $r_1$  arithmetic.

2. If  $r_1 > r_2$ , the magic number is  $r_2 - r_1$  (in this case always negative) and conversion is in the  $r_2$  arithmetic.

*Octal to Decimal.* Here  $r_1 < r_2$ , i.e.,  $8 < 10$ . The rule in going from an octal number to decimal is as follows. Use the magic number  $2 = 10 - 8$ , multiply it decimally by the first left-hand octal digit considered as a decimal, and subtract the resulting number decimally from the first two left-hand octal digits, again considered as a decimal. The procedure is then repeated by multiplying the first two digits of the remainder and subtracting them again, again treating all numbers as decimals. Repeat the process, multiplying by successively larger groups of digits by 2 in each case and subtracting until the number in that column vanishes. Repeat the same procedure in succeeding columns until a group of  $k - 1$  digits has been used, where  $k$  is the number of digits in the original number.

EXAMPLE. Convert  $1077|_8$  and  $678|_8$  to decimal

$$\begin{array}{rcl}
 n & = & 1077|_8 \\
 & & 1077 \\
 - (2 \times 1) & = & \frac{-2}{877} \\
 - (2 \times 8) & = & \frac{-16}{717} \\
 - (2 \times 71) & = & \frac{-142}{575|_{10}} \\
 n & = & 575|_{10}
 \end{array}
 \qquad
 \begin{array}{rcl}
 n & = & 678|_8 \\
 & & 678 \\
 - (2 \times 6) & = & \frac{-12}{558} \\
 - (2 \times 55) & = & \frac{-110}{448|_{10}} \\
 n & = & 448|_{10}
 \end{array}$$

*Hexadecimal to Decimal.* Here  $r_2 > r_1$ , i.e.,  $16 > 10$ . A similar procedure may be used for converting from hexadecimal to decimal via decimal. Here the magic number is negative,  $-6 = 10 - 16$ . The procedure is given directly without any further explanation.

EXAMPLE.

$$\begin{array}{rcl}
 n & = & 6A7|_{16} = 6 \ 10 \ 7|_{16} \\
 - (-6 \times 6) & = & \begin{array}{r} + \ 3 \ 6 \\ \hline 10 \ 6 \ 7 \end{array} \quad (\text{carry occurs at 16, i.e., } 10 + 6) \\
 - (-6 \times 106) & = & \begin{array}{r} + \ 6 \ 3 \ 6 \\ \hline 17 \ 0 \ 3|_{10} \end{array} \\
 n & = & 17 \ 0 \ 3|_{10}
 \end{array}$$

**Conversion with Scale Factors.** *Decimal to Binary.* In general, when converting from arbitrary external numbers into an internal machine language, scale factors may be introduced into the original numbers to complicate the procedure. The procedure most often used in the conversion of decimal numbers to binary by using binary arith-

metic is that given by eqs. (11) to (13). In the computer with internal binary arithmetic the scale factors allowable will generally consist of a decimal scale factor and a binary scale factor. In the MAGIC notation for use with the MIDAC computer, and in similar notations for other binary computers such as the IBM 704 and Univac Scientific, the following is the standard form of a digital number as applied on input. In many cases the binary and decimal scale factors will be zero so that the number itself will be given by the fractional part.

$$n = 97.975 \times 2^{-7},$$

$$n = 0.97975 \times 2^{-7} \times 10^2,$$

$$\text{MAGIC form} = 0.97975 - 7b + 2d.$$

The general procedure for conversion of these numbers is as follows. The decimal digits themselves are input into the machine using binary-coded decimal and shifted in the proper direction by binary shifts of multiples of four, either left or right, depending upon the decimal scale factor. The result is then split into integer and fractional parts. These are then converted by the methods given by eqs. (11) to (13) above.

When the resulting binary number is obtained, it is shifted finally by the corresponding binary scale factor to give the internal machine result.

If the machine is translating digital numbers, that is, numbers with absolute values less than one, it is required that the number itself, multiplied by its various scale factors, be less than one in absolute value. In case the number is being translated into a floating point form with a fractional part less than one in absolute value in binary and an accompanying exponential part also given in binary, the number is standardized with its fractional part less than one in absolute value but greater than one-half.

EXAMPLE.  $n = 1.55 \times 2^{-5} \times 10^1$   
becomes upon conversion

$$n = 31/32 \times 2^{-1}.$$

In some cases in order to save space, the fractional part and exponent are "packed" into one register or machine location with the binary fractional part and the exponential integer part stored together.

**Binary to Decimal Procedure.** In the MAGIC system on MIDAC the inverse procedure of binary to decimal conversion can also be applied, although in most cases the decimal scale factor is not stored in the machine, but instead only a binary scale factor is used. Output conversion requires a fractional part in absolute value less than one, and a corresponding binary scale factor to obtain a composite external decimal number. The scale factor is used to shift the fraction either left or

right depending upon the value of the binary scale factor, before conversion of the corresponding binary integer and fraction, with a method such as that given by eqs. (4) to (10). The resultant series of binary-coded-decimal digits is then printed out of the machine by using the binary-coded-decimal notation, or a "single character" notation depending upon the particular type of printout being used.

**Roundoff in Conversion.** With such systems as the Michigan MAGIC system or the MIT CSSR (Comprehensive System of Service Routines) (Ref. 24) with their complete conversion of numbers from binary to decimal and decimal to binary handled automatically by the machine itself, it is entirely possible for a binary machine to work satisfactorily in the external decimal language, but there are still difficulties involved. Exact binary numbers inside the machine will not be translated in every case into exact decimal numbers. Unless special provisions for a decimal type of roundoff are made, the user of a binary machine will find that his decimal numbers are being output in an incorrect form. Such an error can be corrected by performing the roundoff of numbers inside the machine in decimal rather than binary arithmetic.

**Internal Decimal Scale Factors.** Because of the complexities of the use of binary scale factors, some computer installations with binary machines have nevertheless made use of internal decimal scale factors. With this procedure the number conversion problem can be made much easier; however, it does not take advantage of special operations in the binary machines which allow the binary numbers to be shifted to the right or left, meanwhile saving the number of shifts that have been accomplished, thus allowing automatic binary scaling. Use of internal decimal scale factors also requires the storage of multiprecision constants in some cases, if precision is to be retained. The Illinois computer (ILLIAC) (Ref. 56), which makes use of decimal scale factoring in many of its routines and in its floating decimal point interpretive program, in its original mode of operation did not have a comprehensive system of input-output and therefore did not have as easy a system of converting by using a binary scale factor.

The effort that many mathematicians and programmers have expended on the problems of converting from one number system into another has convinced many of them that computers with internal decimal arithmetic are more useful to the external user. Many of the commercial business-oriented machines make use of some type of binary-coded-decimal representation. Many of the users of binary machines have converted them by internal programming into equivalent decimal machines on the outside. This indicates a trend toward the use of decimal arithmetic

inside the machine even though the actual circuitry may be of the binary type and the storage may be of a binary-coded-decimal nature. The IBM 709 design is an exception to this trend.

### **Representative of Negative Numbers**

Since electronic adders and complementers are much easier to design and require less hardware than subtractors, many digital computers make use of the technique of complementation, by which negative numbers are mapped into positive numbers outside of the usual range. Complemented numbers, under the proper conventions, may be added to a second number to yield the equivalent of the subtraction of the first number from the second.

**Complementation.** Two basic complementation techniques, each based on hardware considerations, must be kept in mind by the computer programmer if the machine he is using should be designed to make use of one or the other. These are (1) the "tens" (decimal), "twos" (binary), or *radix* complement; and (2) the "nines" (decimal), "ones" (binary), or *diminished radix* complement.

In the case of the radix complement, negative numbers less than one in absolute value may often be stored inside the computer in the range from 9 to 10 (for the tens complement) or one to two (for the ones complement). The diminished radix complementation scheme performs a similar mapping, with the added difficulty in the latter case that there are two representations of zero (negative and positive zero) which often must be treated as a special case. Problems arise with these complements in the shifting of numbers (multiplication by powers of the base) and in logical operations on stored negative numbers, since they do not follow the usual conventions. However, on those machines which follow the usual notation, of sign followed by absolute value, none of these difficulties can arise.

### **Precision or Amount of Information Available**

**Fixed Word Length.** Machines, called fixed-word-length computers, have standardized the length of their numbers in the range from about nine to twelve decimal digits, or 36 to 45 binary digits. The number of digits used to represent a number gives an indication of the precision, or amount of information that the number itself contains. Rather than make only one number length available, several machines have offered the user the alternative of (1) longer word lengths as listed above, or (2) shorter numbers, for example, half the number of digits.

**Multiprecision Operations.** For machines with a fixed word length, it is possible to obtain the equivalent of operating with longer numbers of

twice or three times the number of digits by making use of multiprecision types of operations, in which combinations of programmed instructions perform the equivalent of an arithmetic unit handling numbers of higher precision.

**Variable Word Length.** Computers developed particularly for business applications have a variable word length which allows numbers and groups of alphanumeric characters to be operated upon flexibly. With such internal computer structures, numbers of almost any digit length may be operated upon arithmetically, at a cost in time approximately proportional to the number of digits in each operand.

### Range of Numbers

**Fixed Point Machines.** A machine in which the arithmetic circuitry is so designed that the decimal point for operands and result remains in a corresponding fixed position, is called a fixed point computer. The fixed point may be located so as to make the numbers involved either fractions less than one in absolute value, integers, or mixed numbers.

**Location of Radix Point.** In general, a digital computer in which the radix point is located at the extreme left of the number deals with numbers of absolute value less than one, i.e.,  $|n| < 1$ . Such numbers have been called "digital" numbers (Ref. 19), and the term is used throughout this chapter. The product of two such numbers gives as a result another "digital" number, i.e.,  $|n'| < 1$ . On the other hand, division, addition, or subtraction of two "digital" numbers does not necessarily result in another "digital" number, and the problem of overflow arises.

Any other position of fixing the radix point can be made equivalent to fixing it at the extreme left by the use of appropriate scale factors. These scale factors are usually powers of the base, with their main purpose to keep partial results within the range of the computer and at the same time prevent these results being crowded together into a small region of the interval of the machine's range of application. In the case of "digital" numbers the latter interval would generally be close to zero, and numbers there would be said to have lost significance, since they have few information-containing digits.

The optional position for the radix point at the extreme right so as to make numbers inside the computer all integers is somewhat less satisfactory, since the product of two integers of  $n$  digits yields an integer containing  $2n$  significant digits. In this case scale factors are required for every multiplication, and overflows occur as before in many cases for addition, subtraction, and division. Placing of the radix point somewhere midway between these two extremes does not apparently gain any

further advantage, since scale factors or equivalent multiplication by powers of the base must be inserted after each multiplication.

**Floating Point Machines.** As an alternative to one or the other of the above fixed point notations, machines have been built with a variable radix point or, as it is better known, a floating point number system. With such a scheme, numbers are represented by a fractional part or mantissa, containing the significant digit portion of the number, and an exponent representing the power of the base by which the fraction must be multiplied in order to obtain the number itself. For example, one standard notation for floating point numbers  $n$  is

$$n = f \cdot r^e,$$

where  $f$  is the fractional or significant part of the representation, with  $|f| < 1$ , and  $e$  is the power of the base  $r$  needed to bring  $f$  up or down to the value of  $n$ .

**Number System Triad.** Floating point numbers are generally carried inside a digital computer with  $m$  significant digits representing the fractional part  $f$ , and  $n$  digits representing the exponent  $e$  (which may be complemented to represent negative as well as positive integer exponents). The integers  $m$  and  $n$ , along with a third integer  $p$  representing the number of digits to the left of the radix point, may be combined in a number system triad  $(m, n, p)$  indicating the number structure of a computer. Fixed point machines have  $n = 0$ . Machines using "digital" numbers (absolute value less than 1) have  $p = 0$ .

**EXAMPLES.** A (10, 0, 0) decimal computer would be one with 10 significant decimal digits in the fractional part of a number, no digits in the exponent, and no digits to the left of the decimal point. Numbers in a (10, 0, 0) decimal computer, therefore, would be fixed point "digital" numbers containing ten decimal digits, and with the decimal point therefore located at the extreme left. A (30, 6, 0) binary computer would be one with 30 binary digits in the fractional part, and six in the exponent part of a floating point number, and with the binary point at the extreme left. Finally, a (10, 2, 10) decimal computer would be one with the mantissa expressed in ten-decimal integers, and the exponential part of the number given by two-decimal digits.

### Scale Factors in Fixed Point Computations

The recent appearance of built-in floating point arithmetic on many machines, as well as the use of standard translator-compilers which provide automatic floating point programming via subroutines has decreased the need for emphasis on fixed point computation. However, as Burks, von Neumann, and Goldstine (see Ref. 19) point out, computations with

floating point can lead to violent arithmetic errors unless thoroughly analyzed. Hence, fixed point computations will continue, either as part of automatically scaled programs turned out by translators, or else as hand-tailored programs written for fixed point computers.

**Procedure.** The easiest technique for introduction of scale factors into a program is as follows.

1. Determine on the basis of values of input parameters, supposing their range is known at the beginning of a problem, the desired scale factor needed to bring each computed value down to absolute value less than one.

2. Insert shift operations at each stage of the computation which will guarantee that over the range of inputs the resulting numbers will remain digital.

With the PACT system (see Ref. 71), the programmer performs step 1 and the assembler step 2. The latter can also be done by hand.

**EXAMPLE.** Suppose the problem to be calculated is as follows:

Compute the value of the polynomial

$$p(x) = 2.42x^3 + 10.05x^2 - 1024.1x - 20327$$

for

$$x = -10, -9, \dots, 0, 1, \dots, 9, 10.$$

The computation used will be the nested sequence procedure

$$p(x) = ((2.42x + 10.05)x - 1024.1)x - 20327,$$

which may be decomposed into the following set of arithmetic operations where bounds on the absolute values of the partial result are also listed. Arrows indicate transfer of information (storage).

- |                                  |                             |
|----------------------------------|-----------------------------|
| 1. $p_0 \leftarrow 2.42$         | $( p_0  = 2.42, x \leq 10)$ |
| 2. $p_1 \leftarrow p_0x$         | $( p_1  < 25)$              |
| 3. $p_2 \leftarrow p_1 + 10.05$  | $( p_2  < 35)$              |
| 4. $p_3 \leftarrow p_2x$         | $( p_3  < 350)$             |
| 5. $p_4 \leftarrow p_3 - 1024.1$ | $( p_4  < 1400)$            |
| 6. $p_5 \leftarrow p_4x$         | $( p_5  < 15,000)$          |
| 7. $p_6 \leftarrow p_5 - 20327$  | $( p_6  < 36,000)$          |

The bounds on these partial results could be improved by taking the signs of the summands at each stage into consideration. From the list of bounds, the scale factors required to bring the partial results into "digital" number range are:

$$x \cdot 10^{-2}, p_0 \cdot 10^{-1}, p_1 \cdot 10^{-2}, p_2 \cdot 10^{-2}, p_3 \cdot 10^{-3}, p_4 \cdot 10^{-4}, p_5 \cdot 10^{-5}, p_6 \cdot 10^{-5}.$$

Therefore, the sequence of arithmetic statements would be listed stepwise, starting at the first, proceeding downwards, and inserting scale factors as required:

1.  $(p_0 \cdot 10^{-1}) \leftarrow (2.42 \cdot 10^{-1})$
2.  $(p_1 \cdot 10^{-2}) \leftarrow (p_0 \cdot 10^{-1})(x \cdot 10^{-2})(10^1)$
3.  $(p_2 \cdot 10^{-2}) \leftarrow (p_1 \cdot 10^{-2}) + (10.05 \cdot 10^{-2})$
4.  $(p_3 \cdot 10^{-3}) \leftarrow (p_2 \cdot 10^{-2})(x \cdot 10^{-2})(10^1)$
5.  $(p_4 \cdot 10^{-4}) \leftarrow (p_3 \cdot 10^{-3})(10^{-1}) - (1024.1 \cdot 10^{-4})$
6.  $(p_5 \cdot 10^{-5}) \leftarrow (p_4 \cdot 10^{-4})(x \cdot 10^{-2})(10^1)$
7.  $(p_6 \cdot 10^{-5}) \leftarrow (p_5 \cdot 10^{-5}) - (20,237 \cdot 10^{-5})$

There is a redundant left shift followed by a right shift in steps 4 and 5 that should be canceled for coding efficiency.

*Computer Instructions.* The translation from the latter sequence into a sequence of fixed point arithmetic instructions for any computer should now be a simple process. The program, as written for the Datatron 205, (see Sect. 6) upon ignoring use of any special features, such as the index registers and input-output, and coding only the arithmetic sequence, would be, when  $x = -10$ :

0.	ca 0101	$2.42 \cdot 10^{-1} \rightarrow \text{ACC}$
1.	mr 0100	$(2.42 \cdot 10^{-1})(x \cdot 10^{-2})$
2.	sl 0001	$(2.42 \cdot 10^{-1})(x \cdot 10^{-2})(10^1) = p_1 \cdot 10^{-2}$
3.	ad 0102	$(p_1 \cdot 10^{-2}) + (10.05 \cdot 10^{-2}) = p_2 \cdot 10^{-2}$
4.	mr 0100	$(p_2 \cdot 10^{-2})(x \cdot 10^{-2}) = p_3 \cdot 10^{-4}$
5.	ad 0103	$(p_3 \cdot 10^{-4}) + (1024.1 \cdot 10^{-4}) = p_4 \cdot 10^{-4}$
6.	mr 0100	$(p_4 \cdot 10^{-4})(x \cdot 10^{-2})$
7.	sl 0001	$(p_4 \cdot 10^{-4})(x \cdot 10^{-2})(10^1) = p_5 \cdot 10^{-5}$
8.	su 0104	$(p_5 \cdot 10^{-5}) - (20237 \cdot 10^{-5})$
9.	tmc 0105	$(p(x) \cdot 10^{-5}) \rightarrow \text{Location 105}$
100.	-1000000000	$(x \cdot 10^{-2})$
101.	2420000000	$(2.42 \cdot 10^{-1})$
102.	1005000000	$(10.05 \cdot 10^{-2})$
103.	1024100000	$(1024.1 \cdot 10^{-4})$
104.	2023700000	$(20237 \cdot 10^{-5})$
105.		$(p(x) \cdot 10^{-5})$

### 3. NUMBER CONVERSION TABLES

Most number conversion in binary computers is now done completely by the machine itself. However, for completeness and easy access to the

programmer who from time to time has to check a number obtained from lights on the console of the computer, or printed out in nonstandard notation, Tables 1 to 5 are included from Carr and Scott (Ref. 24) and the IBM 704 manual (Ref. 54).

<i>Table</i>	<i>Page</i>
Table 1. Octal-Decimal Integer Conversion	2-28
Table 2. Octal-Decimal Fraction Conversion	2-36
Table 3. Hexadecimal-Decimal Conversion (Two-Way)	2-42
Table 4. Powers of 2 (Positive and Negative) Expressed in Decimal	2-43
Table 5. Hexadecimal Multiplication	2-44

TABLE 1. OCTAL-DECIMAL INTEGER CONVERSION

0000 | 0000  
to | to  
0777 | 0511  
(Octal) | (Decimal)

Octal | Decimal  
10000 - 4096  
20000 - 8192  
30000 - 12288  
40000 - 16384  
50000 - 20480  
60000 - 24576  
70000 - 28672

	0	1	2	3	4	5	6	7
0000	0000	0001	0002	0003	0004	0005	0006	0007
0010	0008	0009	0010	0011	0012	0013	0014	0015
0020	0016	0017	0018	0019	0020	0021	0022	0023
0030	0024	0025	0026	0027	0028	0029	0030	0031
0040	0032	0033	0034	0035	0036	0037	0038	0039
0050	0040	0041	0042	0043	0044	0045	0046	0047
0060	0048	0049	0050	0051	0052	0053	0054	0055
0070	0056	0057	0058	0059	0060	0061	0062	0063
0100	0064	0065	0066	0067	0068	0069	0070	0071
0110	0072	0073	0074	0075	0076	0077	0078	0079
0120	0080	0081	0082	0083	0084	0085	0086	0087
0130	0088	0089	0090	0091	0092	0093	0094	0095
0140	0096	0097	0098	0099	0100	0101	0102	0103
0150	0104	0105	0106	0107	0108	0109	0110	0111
0160	0112	0113	0114	0115	0116	0117	0118	0119
0170	0120	0121	0122	0123	0124	0125	0126	0127
0200	0128	0129	0130	0131	0132	0133	0134	0135
0210	0136	0137	0138	0139	0140	0141	0142	0143
0220	0144	0145	0146	0147	0148	0149	0150	0151
0230	0152	0153	0154	0155	0156	0157	0158	0159
0240	0160	0161	0162	0163	0164	0165	0166	0167
0250	0168	0169	0170	0171	0172	0173	0174	0175
0260	0176	0177	0178	0179	0180	0181	0182	0183
0270	0184	0185	0186	0187	0188	0189	0190	0191
0300	0192	0193	0194	0195	0196	0197	0198	0199
0310	0200	0201	0202	0203	0204	0205	0206	0207
0320	0208	0209	0210	0211	0212	0213	0214	0215
0330	0216	0217	0218	0219	0220	0221	0222	0223
0340	0224	0225	0226	0227	0228	0229	0230	0231
0350	0232	0233	0234	0235	0236	0237	0238	0239
0360	0240	0241	0242	0243	0244	0245	0246	0247
0370	0248	0249	0250	0251	0252	0253	0254	0255

	0	1	2	3	4	5	6	7
0400	0256	0257	0258	0259	0260	0261	0262	0263
0410	0264	0265	0266	0267	0268	0269	0270	0271
0420	0272	0273	0274	0275	0276	0277	0278	0279
0430	0280	0281	0282	0283	0284	0285	0286	0287
0440	0288	0289	0290	0291	0292	0293	0294	0295
0450	0296	0297	0298	0299	0300	0301	0302	0303
0460	0304	0305	0306	0307	0308	0309	0310	0311
0470	0312	0313	0314	0315	0316	0317	0318	0319
0500	0320	0321	0322	0323	0324	0325	0326	0327
0510	0328	0329	0330	0331	0332	0333	0334	0335
0520	0336	0337	0338	0339	0340	0341	0342	0343
0530	0344	0345	0346	0347	0348	0349	0350	0351
0540	0352	0353	0354	0355	0356	0357	0358	0359
0550	0360	0361	0362	0363	0364	0365	0366	0367
0560	0368	0369	0370	0371	0372	0373	0374	0375
0570	0376	0377	0378	0379	0380	0381	0382	0383
0600	0384	0385	0386	0387	0388	0389	0390	0391
0610	0392	0393	0394	0395	0396	0397	0398	0399
0620	0400	0401	0402	0403	0404	0405	0406	0407
0630	0408	0409	0410	0411	0412	0413	0414	0415
0640	0416	0417	0418	0419	0420	0421	0422	0423
0650	0424	0425	0426	0427	0428	0429	0430	0431
0660	0432	0433	0434	0435	0436	0437	0438	0439
0670	0440	0441	0442	0443	0444	0445	0446	0447
0700	0448	0449	0450	0451	0452	0453	0454	0455
0710	0456	0457	0458	0459	0460	0461	0462	0463
0720	0464	0465	0466	0467	0468	0469	0470	0471
0730	0472	0473	0474	0475	0476	0477	0478	0479
0740	0480	0481	0482	0483	0484	0485	0486	0487
0750	0488	0489	0490	0491	0492	0493	0494	0495
0760	0496	0497	0498	0499	0500	0501	0502	0503
0770	0504	0505	0506	0507	0508	0509	0510	0511

1000 to 1777  
(Octal) | 0512 to 1023  
(Decimal)

	0	1	2	3	4	5	6	7
1000	0512	0513	0514	0515	0516	0517	0518	0519
1010	0520	0521	0522	0523	0524	0525	0526	0527
1020	0528	0529	0530	0531	0532	0533	0534	0535
1030	0536	0537	0538	0539	0540	0541	0542	0543
1040	0544	0545	0546	0547	0548	0549	0550	0551
1050	0552	0553	0554	0555	0556	0557	0558	0559
1060	0560	0561	0562	0563	0564	0565	0566	0567
1070	0568	0569	0570	0571	0572	0573	0574	0575
1100	0576	0577	0578	0579	0580	0581	0582	0583
1110	0584	0585	0586	0587	0588	0589	0590	0591
1120	0592	0593	0594	0595	0596	0597	0598	0599
1130	0600	0601	0602	0603	0604	0605	0606	0607
1140	0608	0609	0610	0611	0612	0613	0614	0615
1150	0616	0617	0618	0619	0620	0621	0622	0623
1160	0624	0625	0626	0627	0628	0629	0630	0631
1170	0632	0633	0634	0635	0636	0637	0638	0639
1200	0640	0641	0642	0643	0644	0645	0646	0647
1210	0648	0649	0650	0651	0652	0653	0654	0655
1220	0656	0657	0658	0659	0660	0661	0662	0663
1230	0664	0665	0666	0667	0668	0669	0670	0671
1240	0672	0673	0674	0675	0676	0677	0678	0679
1250	0680	0681	0682	0683	0684	0685	0686	0687
1260	0688	0689	0690	0691	0692	0693	0694	0695
1270	0696	0697	0698	0699	0700	0701	0702	0703
1300	0704	0705	0706	0707	0708	0709	0710	0711
1310	0712	0713	0714	0715	0716	0717	0718	0719
1320	0720	0721	0722	0723	0724	0725	0726	0727
1330	0728	0729	0730	0731	0732	0733	0734	0735
1340	0736	0737	0738	0739	0740	0741	0742	0743
1350	0744	0745	0746	0747	0748	0749	0750	0751
1360	0752	0753	0754	0755	0756	0757	0758	0759
1370	0760	0761	0762	0763	0764	0765	0766	0767

	0	1	2	3	4	5	6	7
1400	0768	0769	0770	0771	0772	0773	0774	0775
1410	0776	0777	0778	0779	0780	0781	0782	0783
1420	0784	0785	0786	0787	0788	0789	0790	0791
1430	0792	0793	0794	0795	0796	0797	0798	0799
1440	0800	0801	0802	0803	0804	0805	0806	0807
1450	0808	0809	0810	0811	0812	0813	0814	0815
1460	0816	0817	0818	0819	0820	0821	0822	0823
1470	0824	0825	0826	0827	0828	0829	0830	0831
1500	0832	0833	0834	0835	0836	0837	0838	0839
1510	0840	0841	0842	0843	0844	0845	0846	0847
1520	0848	0849	0850	0851	0852	0853	0854	0855
1530	0856	0857	0858	0859	0860	0861	0862	0863
1540	0864	0865	0866	0867	0868	0869	0870	0871
1550	0872	0873	0874	0875	0876	0877	0878	0879
1560	0880	0881	0882	0883	0884	0885	0886	0887
1570	0888	0889	0890	0891	0892	0893	0894	0895
1600	0896	0897	0898	0899	0900	0901	0902	0903
1610	0904	0905	0906	0907	0908	0909	0910	0911
1620	0912	0913	0914	0915	0916	0917	0918	0919
1630	0920	0921	0922	0923	0924	0925	0926	0927
1640	0928	0929	0930	0931	0932	0933	0934	0935
1650	0936	0937	0938	0939	0940	0941	0942	0943
1660	0944	0945	0946	0947	0948	0949	0950	0951
1670	0952	0953	0954	0955	0956	0957	0958	0959
1700	0960	0961	0962	0963	0964	0965	0966	0967
1710	0968	0969	0970	0971	0972	0973	0974	0975
1720	0976	0977	0978	0979	0980	0981	0982	0983
1730	0984	0985	0986	0987	0988	0989	0990	0991
1740	0992	0993	0994	0995	0996	0997	0998	0999
1750	1000	1001	1002	1003	1004	1005	1006	1007
1760	1008	1009	1010	1011	1012	1013	1014	1015
1770	1016	1017	1018	1019	1020	1021	1022	1023

TABLE 1. OCTAL-DECIMAL INTEGER CONVERSION (Continued)

		0	1	2	3	4	5	6	7										
2000 to 2777 (Octal)	1024 to 1535 (Decimal)	2000	1024	1025	1026	1027	1028	1029	1030	1031	2400	1280	1281	1282	1283	1284	1285	1286	1287
		2010	1032	1033	1034	1035	1036	1037	1038	1039	2410	1288	1289	1290	1291	1292	1293	1294	1295
		2020	1040	1041	1042	1043	1044	1045	1046	1047	2420	1296	1297	1298	1299	1300	1301	1302	1303
		2030	1048	1049	1050	1051	1052	1053	1054	1055	2430	1304	1305	1306	1307	1308	1309	1310	1311
		2040	1056	1057	1058	1059	1060	1061	1062	1063	2440	1312	1313	1314	1315	1316	1317	1318	1319
		2050	1064	1065	1066	1067	1068	1069	1070	1071	2450	1320	1321	1322	1323	1324	1325	1326	1327
		2060	1072	1073	1074	1075	1076	1077	1078	1079	2460	1328	1329	1330	1331	1332	1333	1334	1335
		2070	1080	1081	1082	1083	1084	1085	1086	1087	2470	1336	1337	1338	1339	1340	1341	1342	1343
		2100	1088	1089	1090	1091	1092	1093	1094	1095	2500	1344	1345	1346	1347	1348	1349	1350	1351
		2110	1096	1097	1098	1099	1100	1101	1102	1103	2510	1352	1353	1354	1355	1356	1357	1358	1359
		2120	1104	1105	1106	1107	1108	1109	1110	1111	2520	1360	1361	1362	1363	1364	1365	1366	1367
		2130	1112	1113	1114	1115	1116	1117	1118	1119	2530	1368	1369	1370	1371	1372	1373	1374	1375
		2140	1120	1121	1122	1123	1124	1125	1126	1127	2540	1376	1377	1378	1379	1380	1381	1382	1383
		2150	1128	1129	1130	1131	1132	1133	1134	1135	2550	1384	1385	1386	1387	1388	1389	1390	1391
		2160	1136	1137	1138	1139	1140	1141	1142	1143	2560	1392	1393	1394	1395	1396	1397	1398	1399
		2170	1144	1145	1146	1147	1148	1149	1150	1151	2570	1400	1401	1402	1403	1404	1405	1406	1407
		2200	1152	1153	1154	1155	1156	1157	1158	1159	2600	1408	1409	1410	1411	1412	1413	1414	1415
		2210	1160	1161	1162	1163	1164	1165	1166	1167	2610	1416	1417	1418	1419	1420	1421	1422	1423
		2220	1168	1169	1170	1171	1172	1173	1174	1175	2620	1424	1425	1426	1427	1428	1429	1430	1431
		2230	1176	1177	1178	1179	1180	1181	1182	1183	2630	1432	1433	1434	1435	1436	1437	1438	1439
		2240	1184	1185	1186	1187	1188	1189	1190	1191	2640	1440	1441	1442	1443	1444	1445	1446	1447
		2250	1192	1193	1194	1195	1196	1197	1198	1199	2650	1448	1449	1450	1451	1452	1453	1454	1455
		2260	1200	1201	1202	1203	1204	1205	1206	1207	2660	1456	1457	1458	1459	1460	1461	1462	1463
		2270	1208	1209	1210	1211	1212	1213	1214	1215	2670	1464	1465	1466	1467	1468	1469	1470	1471
		2300	1216	1217	1218	1219	1220	1221	1222	1223	2700	1472	1473	1474	1475	1476	1477	1478	1479
		2310	1224	1225	1226	1227	1228	1229	1230	1231	2710	1480	1481	1482	1483	1484	1485	1486	1487
		2320	1232	1233	1234	1235	1236	1237	1238	1239	2720	1488	1489	1490	1491	1492	1493	1494	1495
		2330	1240	1241	1242	1243	1244	1245	1246	1247	2730	1496	1497	1498	1499	1500	1501	1502	1503
		2340	1248	1249	1250	1251	1252	1253	1254	1255	2740	1504	1505	1506	1507	1508	1509	1510	1511
		2350	1256	1257	1258	1259	1260	1261	1262	1263	2750	1512	1513	1514	1515	1516	1517	1518	1519
		2360	1264	1265	1266	1267	1268	1269	1270	1271	2760	1520	1521	1522	1523	1524	1525	1526	1527
		2370	1272	1273	1274	1275	1276	1277	1278	1279	2770	1528	1529	1530	1531	1532	1533	1534	1535

2000 to 2777 (Octal) | 1024 to 1535 (Decimal)

Octal Decimal  
 10000 - 4096  
 20000 - 8192  
 30000 - 12288  
 40000 - 16384  
 50000 - 20480  
 60000 - 24576  
 70000 - 28672

3000     1536  
 to        to  
 3777     2047  
 (Octal) (Decimal)

	0	1	2	3	4	5	6	7
3000	1536	1537	1538	1539	1540	1541	1542	1543
3010	1544	1545	1546	1547	1548	1549	1550	1551
3020	1552	1553	1554	1555	1556	1557	1558	1559
3030	1560	1561	1562	1563	1564	1565	1566	1567
3040	1568	1569	1570	1571	1572	1573	1574	1575
3050	1576	1577	1578	1579	1580	1581	1582	1583
3060	1584	1585	1586	1587	1588	1589	1590	1591
3070	1592	1593	1594	1595	1596	1597	1598	1599
3100	1600	1601	1602	1603	1604	1605	1606	1607
3110	1608	1609	1610	1611	1612	1613	1614	1615
3120	1616	1617	1618	1619	1620	1621	1622	1623
3130	1624	1625	1626	1627	1628	1629	1630	1631
3140	1632	1633	1634	1635	1636	1637	1638	1639
3150	1640	1641	1642	1643	1644	1645	1646	1647
3160	1648	1649	1650	1651	1652	1653	1654	1655
3170	1656	1657	1658	1659	1660	1661	1662	1663
3200	1664	1665	1666	1667	1668	1669	1670	1671
3210	1672	1673	1674	1675	1676	1677	1678	1679
3220	1680	1681	1682	1683	1684	1685	1686	1687
3230	1688	1689	1690	1691	1692	1693	1694	1695
3240	1696	1697	1698	1699	1700	1701	1702	1703
3250	1704	1705	1706	1707	1708	1709	1710	1711
3260	1712	1713	1714	1715	1716	1717	1718	1719
3270	1720	1721	1722	1723	1724	1725	1726	1727
3300	1728	1729	1730	1731	1732	1733	1734	1735
3310	1736	1737	1738	1739	1740	1741	1742	1743
3320	1744	1745	1746	1747	1748	1749	1750	1751
3330	1752	1753	1754	1755	1756	1757	1758	1759
3340	1760	1761	1762	1763	1764	1765	1766	1767
3350	1768	1769	1770	1771	1772	1773	1774	1775
3360	1776	1777	1778	1779	1780	1781	1782	1783
3370	1784	1785	1786	1787	1788	1789	1790	1791

	0	1	2	3	4	5	6	7
3400	1792	1793	1794	1795	1796	1797	1798	1799
3410	1800	1801	1802	1803	1804	1805	1806	1807
3420	1808	1809	1810	1811	1812	1813	1814	1815
3430	1816	1817	1818	1819	1820	1821	1822	1823
3440	1824	1825	1826	1827	1828	1829	1830	1831
3450	1832	1833	1834	1835	1836	1837	1838	1839
3460	1840	1841	1842	1843	1844	1845	1846	1847
3470	1848	1849	1850	1851	1852	1853	1854	1855
3500	1856	1857	1858	1859	1860	1861	1862	1863
3510	1864	1865	1866	1867	1868	1869	1870	1871
3520	1872	1873	1874	1875	1876	1877	1878	1879
3530	1880	1881	1882	1883	1884	1885	1886	1887
3540	1888	1889	1890	1891	1892	1893	1894	1895
3550	1896	1897	1898	1899	1900	1901	1902	1903
3560	1904	1905	1906	1907	1908	1909	1910	1911
3570	1912	1913	1914	1915	1916	1917	1918	1919
3600	1920	1921	1922	1923	1924	1925	1926	1927
3610	1928	1929	1930	1931	1932	1933	1934	1935
3620	1936	1937	1938	1939	1940	1941	1942	1943
3630	1944	1945	1946	1947	1948	1949	1950	1951
3640	1952	1953	1954	1955	1956	1957	1958	1959
3650	1960	1961	1962	1963	1964	1965	1966	1967
3660	1968	1969	1970	1971	1972	1973	1974	1975
3670	1976	1977	1978	1979	1980	1981	1982	1983
3700	1984	1985	1986	1987	1988	1989	1990	1991
3710	1992	1993	1994	1995	1996	1997	1998	1999
3720	2000	2001	2002	2003	2004	2005	2006	2007
3730	2008	2009	2010	2011	2012	2013	2014	2015
3740	2016	2017	2018	2019	2020	2021	2022	2023
3750	2024	2025	2026	2027	2028	2029	2030	2031
3760	2032	2033	2034	2035	2036	2037	2038	2039
3770	2040	2041	2042	2043	2044	2045	2046	2047

TABLE 1. OCTAL-DECIMAL INTEGER CONVERSION (Continued)

4000 | 2048  
to | to  
4777 | 2559  
(Octal) | (Decimal)

Octal Decimal  
10000 - 4096  
20000 - 8192  
30000 - 12288  
40000 - 16384  
50000 - 20480  
60000 - 24576  
70000 - 28672

	0	1	2	3	4	5	6	7
4000	2048	2049	2050	2051	2052	2053	2054	2055
4010	2056	2057	2058	2059	2060	2061	2062	2063
4020	2064	2065	2066	2067	2068	2069	2070	2071
4030	2072	2073	2074	2075	2076	2077	2078	2079
4040	2080	2081	2082	2083	2084	2085	2086	2087
4050	2088	2089	2090	2091	2092	2093	2094	2095
4060	2096	2097	2098	2099	2100	2101	2102	2103
4070	2104	2105	2106	2107	2108	2109	2110	2111
4100	2112	2113	2114	2115	2116	2117	2118	2119
4110	2120	2121	2122	2123	2124	2125	2126	2127
4120	2128	2129	2130	2131	2132	2133	2134	2135
4130	2136	2137	2138	2139	2140	2141	2142	2143
4140	2144	2145	2146	2147	2148	2149	2150	2151
4150	2152	2153	2154	2155	2156	2157	2158	2159
4160	2160	2161	2162	2163	2164	2165	2166	2167
4170	2168	2169	2170	2171	2172	2173	2174	2175
4200	2176	2177	2178	2179	2180	2181	2182	2183
4210	2184	2185	2186	2187	2188	2189	2190	2191
4220	2192	2193	2194	2195	2196	2197	2198	2199
4230	2200	2201	2202	2203	2204	2205	2206	2207
4240	2208	2209	2210	2211	2212	2213	2214	2215
4250	2216	2217	2218	2219	2220	2221	2222	2223
4260	2224	2225	2226	2227	2228	2229	2230	2231
4270	2232	2233	2234	2235	2236	2237	2238	2239
4300	2240	2241	2242	2243	2244	2245	2246	2247
4310	2248	2249	2250	2251	2252	2253	2254	2255
4320	2256	2257	2258	2259	2260	2261	2262	2263
4330	2264	2265	2266	2267	2268	2269	2270	2271
4340	2272	2273	2274	2275	2276	2277	2278	2279
4350	2280	2281	2282	2283	2284	2285	2286	2287
4360	2288	2289	2290	2291	2292	2293	2294	2295
4370	2296	2297	2298	2299	2300	2301	2302	2303

	0	1	2	3	4	5	6	7
4400	2304	2305	2306	2307	2308	2309	2310	2311
4410	2312	2313	2314	2315	2316	2317	2318	2319
4420	2320	2321	2322	2323	2324	2325	2326	2327
4430	2328	2329	2330	2331	2332	2333	2334	2335
4440	2336	2337	2338	2339	2340	2341	2342	2343
4450	2344	2345	2346	2347	2348	2349	2350	2351
4460	2352	2353	2354	2355	2356	2357	2358	2359
4470	2360	2361	2362	2363	2364	2365	2366	2367
4500	2368	2369	2370	2371	2372	2373	2374	2375
4510	2376	2377	2378	2379	2380	2381	2382	2383
4520	2384	2385	2386	2387	2388	2389	2390	2391
4530	2392	2393	2394	2395	2396	2397	2398	2399
4540	2400	2401	2402	2403	2404	2405	2406	2407
4550	2408	2409	2410	2411	2412	2413	2414	2415
4560	2416	2417	2418	2419	2420	2421	2422	2423
4570	2424	2425	2426	2427	2428	2429	2430	2431
4600	2432	2433	2434	2435	2436	2437	2438	2439
4610	2440	2441	2442	2443	2444	2445	2446	2447
4620	2448	2449	2450	2451	2452	2453	2454	2455
4630	2456	2457	2458	2459	2460	2461	2462	2463
4640	2464	2465	2466	2467	2468	2469	2470	2471
4650	2472	2473	2474	2475	2476	2477	2478	2479
4660	2480	2481	2482	2483	2484	2485	2486	2487
4670	2488	2489	2490	2491	2492	2493	2494	2495
4700	2496	2497	2498	2499	2500	2501	2502	2503
4710	2504	2505	2506	2507	2508	2509	2510	2511
4720	2512	2513	2514	2515	2516	2517	2518	2519
4730	2520	2521	2522	2523	2524	2525	2526	2527
4740	2528	2529	2530	2531	2532	2533	2534	2535
4750	2536	2537	2538	2539	2540	2541	2542	2543
4760	2544	2545	2546	2547	2548	2549	2550	2551
4770	2552	2553	2554	2555	2556	2557	2558	2559

5000 to 5777 (Octal) | 2560 to 3071 (Decimal)

	0	1	2	3	4	5	6	7
5000	2560	2561	2562	2563	2564	2565	2566	2567
5010	2568	2569	2570	2571	2572	2573	2574	2575
5020	2576	2577	2578	2579	2580	2581	2582	2583
5030	2584	2585	2586	2587	2588	2589	2590	2591
5040	2592	2593	2594	2595	2596	2597	2598	2599
5050	2600	2601	2602	2603	2604	2605	2606	2607
5060	2608	2609	2610	2611	2612	2613	2614	2615
5070	2616	2617	2618	2619	2620	2621	2622	2623
5100	2624	2625	2626	2627	2628	2629	2630	2631
5110	2632	2633	2634	2635	2636	2637	2638	2639
5120	2640	2641	2642	2643	2644	2645	2646	2647
5130	2648	2649	2650	2651	2652	2653	2654	2655
5140	2656	2657	2658	2659	2660	2661	2662	2663
5150	2664	2665	2666	2667	2668	2669	2670	2671
5160	2672	2673	2674	2675	2676	2677	2678	2679
5170	2680	2681	2682	2683	2684	2685	2686	2687
5200	2688	2689	2690	2691	2692	2693	2694	2695
5210	2696	2697	2698	2699	2700	2701	2702	2703
5220	2704	2705	2706	2707	2708	2709	2710	2711
5230	2712	2713	2714	2715	2716	2717	2718	2719
5240	2720	2721	2722	2723	2724	2725	2726	2727
5250	2728	2729	2730	2731	2732	2733	2734	2735
5260	2736	2737	2738	2739	2740	2741	2742	2743
5270	2744	2745	2746	2747	2748	2749	2750	2751
5300	2752	2753	2754	2755	2756	2757	2758	2759
5310	2760	2761	2762	2763	2764	2765	2766	2767
5320	2768	2769	2770	2771	2772	2773	2774	2775
5330	2776	2777	2778	2779	2780	2781	2782	2783
5340	2784	2785	2786	2787	2788	2789	2790	2791
5350	2792	2793	2794	2795	2796	2797	2798	2799
5360	2800	2801	2802	2803	2804	2805	2806	2807
5370	2808	2809	2810	2811	2812	2813	2814	2815

	0	1	2	3	4	5	6	7
5400	2816	2817	2818	2819	2820	2821	2822	2823
5410	2824	2825	2826	2827	2828	2829	2830	2831
5420	2832	2833	2834	2835	2836	2837	2838	2839
5430	2840	2841	2842	2843	2844	2845	2846	2847
5440	2848	2849	2850	2851	2852	2853	2854	2855
5450	2856	2857	2858	2859	2860	2861	2862	2863
5460	2864	2865	2866	2867	2868	2869	2870	2871
5470	2872	2873	2874	2875	2876	2877	2878	2879
5500	2880	2881	2882	2883	2884	2885	2886	2887
5510	2888	2889	2890	2891	2892	2893	2894	2895
5520	2896	2897	2898	2899	2900	2901	2902	2903
5530	2904	2905	2906	2907	2908	2909	2910	2911
5540	2912	2913	2914	2915	2916	2917	2918	2919
5550	2920	2921	2922	2923	2924	2925	2926	2927
5560	2928	2929	2930	2931	2932	2933	2934	2935
5570	2936	2937	2938	2939	2940	2941	2942	2943
5600	2944	2945	2946	2947	2948	2949	2950	2951
5610	2952	2953	2954	2955	2956	2957	2958	2959
5620	2960	2961	2962	2963	2964	2965	2966	2967
5630	2968	2969	2970	2971	2972	2973	2974	2975
5640	2976	2977	2978	2979	2980	2981	2982	2983
5650	2984	2985	2986	2987	2988	2989	2990	2991
5660	2992	2993	2994	2995	2996	2997	2998	2999
5670	3000	3001	3002	3003	3004	3005	3006	3007
5700	3008	3009	3010	3011	3012	3013	3014	3015
5710	3016	3017	3018	3019	3020	3021	3022	3023
5720	3024	3025	3026	3027	3028	3029	3030	3031
5730	3032	3033	3034	3035	3036	3037	3038	3039
5740	3040	3041	3042	3043	3044	3045	3046	3047
5750	3048	3049	3050	3051	3052	3053	3054	3055
5760	3056	3057	3058	3059	3060	3061	3062	3063
5770	3064	3065	3066	3067	3068	3069	3070	3071

TABLE 1. OCTAL-DECIMAL INTEGER CONVERSION (Continued)

6000	3072
to	to
6777	3583
(Octal)	(Decimal)

Octal	Decimal
10000 -	4096
20000 -	8192
30000 -	12288
40000 -	16384
50000 -	20480
60000 -	24576
70000 -	28672

	0	1	2	3	4	5	6	7
6000	3072	3073	3074	3075	3076	3077	3078	3079
6010	3080	3081	3082	3083	3084	3085	3086	3087
6020	3088	3089	3090	3091	3092	3093	3094	3095
6030	3096	3097	3098	3099	3100	3101	3102	3103
6040	3104	3105	3106	3107	3108	3109	3110	3111
6050	3112	3113	3114	3115	3116	3117	3118	3119
6060	3120	3121	3122	3123	3124	3125	3126	3127
6070	3128	3129	3130	3131	3132	3133	3134	3135
6100	3136	3137	3138	3139	3140	3141	3142	3143
6110	3144	3145	3146	3147	3148	3149	3150	3151
6120	3152	3153	3154	3155	3156	3157	3158	3159
6130	3160	3161	3162	3163	3164	3165	3166	3167
6140	3168	3169	3170	3171	3172	3173	3174	3175
6150	3176	3177	3178	3179	3180	3181	3182	3183
6160	3184	3185	3186	3187	3188	3189	3190	3191
6170	3192	3193	3194	3195	3196	3197	3198	3199
6200	3200	3201	3202	3203	3204	3205	3206	3207
6210	3208	3209	3210	3211	3212	3213	3214	3215
6220	3216	3217	3218	3219	3220	3221	3222	3223
6230	3224	3225	3226	3227	3228	3229	3230	3231
6240	3232	3233	3234	3235	3236	3237	3238	3239
6250	3240	3241	3242	3243	3244	3245	3246	3247
6260	3248	3249	3250	3251	3252	3253	3254	3255
6270	3256	3257	3258	3259	3260	3261	3262	3263
6300	3264	3265	3266	3267	3268	3269	3270	3271
6310	3272	3273	3274	3275	3276	3277	3278	3279
6320	3280	3281	3282	3283	3284	3285	3286	3287
6330	3288	3289	3290	3291	3292	3293	3294	3295
6340	3296	3297	3298	3299	3300	3301	3302	3303
6350	3304	3305	3306	3307	3308	3309	3310	3311
6360	3312	3313	3314	3315	3316	3317	3318	3319
6370	3320	3321	3322	3323	3324	3325	3326	3327

	0	1	2	3	4	5	6	7
6400	3328	3329	3330	3331	3332	3333	3334	3335
6410	3336	3337	3338	3339	3340	3341	3342	3343
6420	3344	3345	3346	3347	3348	3349	3350	3351
6430	3352	3353	3354	3355	3356	3357	3358	3359
6440	3360	3361	3362	3363	3364	3365	3366	3367
6450	3368	3369	3370	3371	3372	3373	3374	3375
6460	3376	3377	3378	3379	3380	3381	3382	3383
6470	3384	3385	3386	3387	3388	3389	3390	3391
6500	3392	3393	3394	3395	3396	3397	3398	3399
6510	3400	3401	3402	3403	3404	3405	3406	3407
6520	3408	3409	3410	3411	3412	3413	3414	3415
6530	3416	3417	3418	3419	3420	3421	3422	3423
6540	3424	3425	3426	3427	3428	3429	3430	3431
6550	3432	3433	3434	3435	3436	3437	3438	3439
6560	3440	3441	3442	3443	3444	3445	3446	3447
6570	3448	3449	3450	3451	3452	3453	3454	3455
6600	3456	3457	3458	3459	3460	3461	3462	3463
6610	3464	3465	3466	3467	3468	3469	3470	3471
6620	3472	3473	3474	3475	3476	3477	3478	3479
6630	3480	3481	3482	3483	3484	3485	3486	3487
6640	3488	3489	3490	3491	3492	3493	3494	3495
6650	3496	3497	3498	3499	3500	3501	3502	3503
6660	3504	3505	3506	3507	3508	3509	3510	3511
6670	3512	3513	3514	3515	3516	3517	3518	3519
6700	3520	3521	3522	3523	3524	3525	3526	3527
6710	3528	3529	3530	3531	3532	3533	3534	3535
6720	3536	3537	3538	3539	3540	3541	3542	3543
6730	3544	3545	3546	3547	3548	3549	3550	3551
6740	3552	3553	3554	3555	3556	3557	3558	3559
6750	3560	3561	3562	3563	3564	3565	3566	3567
6760	3568	3569	3570	3571	3572	3573	3574	3575
6770	3576	3577	3578	3579	3580	3581	3582	3583

7000 | 3584  
to | to  
7777 | 4095  
(Octal) | (Decimal)

	0	1	2	3	4	5	6	7
7000	3584	3585	3586	3587	3588	3589	3590	3591
7010	3592	3593	3594	3595	3596	3597	3598	3599
7020	3600	3601	3602	3603	3604	3605	3606	3607
7030	3608	3609	3610	3611	3612	3613	3614	3615
7040	3616	3617	3618	3619	3620	3621	3622	3623
7050	3624	3625	3626	3627	3628	3629	3630	3631
7060	3632	3633	3634	3635	3636	3637	3638	3639
7070	3640	3641	3642	3643	3644	3645	3646	3647
7100	3648	3649	3650	3651	3652	3653	3654	3655
7110	3656	3657	3658	3659	3660	3661	3662	3663
7120	3664	3665	3666	3667	3668	3669	3670	3671
7130	3672	3673	3674	3675	3676	3677	3678	3679
7140	3680	3681	3682	3683	3684	3685	3686	3687
7150	3688	3689	3690	3691	3692	3693	3694	3695
7160	3696	3697	3698	3699	3700	3701	3702	3703
7170	3704	3705	3706	3707	3708	3709	3710	3711
7200	3712	3713	3714	3715	3716	3717	3718	3719
7210	3720	3721	3722	3723	3724	3725	3726	3727
7220	3728	3729	3730	3731	3732	3733	3734	3735
7230	3736	3737	3738	3739	3740	3741	3742	3743
7240	3744	3745	3746	3747	3748	3749	3750	3751
7250	3752	3753	3754	3755	3756	3757	3758	3759
7260	3760	3761	3762	3763	3764	3765	3766	3767
7270	3768	3769	3770	3771	3772	3773	3774	3775
7300	3776	3777	3778	3779	3780	3781	3782	3783
7310	3784	3785	3786	3787	3788	3789	3790	3791
7320	3792	3793	3794	3795	3796	3797	3798	3799
7330	3800	3801	3802	3803	3804	3805	3806	3807
7340	3808	3809	3810	3811	3812	3813	3814	3815
7350	3816	3817	3818	3819	3820	3821	3822	3823
7360	3824	3825	3826	3827	3828	3829	3830	3831
7370	3832	3833	3834	3835	3836	3837	3838	3839

	0	1	2	3	4	5	6	7
7400	3840	3841	3842	3843	3844	3845	3846	3847
7410	3848	3849	3850	3851	3852	3853	3854	3855
7420	3856	3857	3858	3859	3860	3861	3862	3863
7430	3864	3865	3866	3867	3868	3869	3870	3871
7440	3872	3873	3874	3875	3876	3877	3878	3879
7450	3880	3881	3882	3883	3884	3885	3886	3887
7460	3888	3889	3890	3891	3892	3893	3894	3895
7470	3896	3897	3898	3899	3900	3901	3902	3903
7500	3904	3905	3906	3907	3908	3909	3910	3911
7510	3912	3913	3914	3915	3916	3917	3918	3919
7520	3920	3921	3922	3923	3924	3925	3926	3927
7530	3928	3929	3930	3931	3932	3933	3934	3935
7540	3936	3937	3938	3939	3940	3941	3942	3943
7550	3944	3945	3946	3947	3948	3949	3950	3951
7560	3952	3953	3954	3955	3956	3957	3958	3959
7570	3960	3961	3962	3963	3964	3965	3966	3967
7600	3968	3969	3970	3971	3972	3973	3974	3975
7610	3976	3977	3978	3979	3980	3981	3982	3983
7620	3984	3985	3986	3987	3988	3989	3990	3991
7630	3992	3993	3994	3995	3996	3997	3998	3999
7640	4000	4001	4002	4003	4004	4005	4006	4007
7650	4008	4009	4010	4011	4012	4013	4014	4015
7660	4016	4017	4018	4019	4020	4021	4022	4023
7670	4024	4025	4026	4027	4028	4029	4030	4031
7700	4032	4033	4034	4035	4036	4037	4038	4039
7710	4040	4041	4042	4043	4044	4045	4046	4047
7720	4048	4049	4050	4051	4052	4053	4054	4055
7730	4056	4057	4058	4059	4060	4061	4062	4063
7740	4064	4065	4066	4067	4068	4069	4070	4071
7750	4072	4073	4074	4075	4076	4077	4078	4079
7760	4080	4081	4082	4083	4084	4085	4086	4087
7770	4088	4089	4090	4091	4092	4093	4094	4095

TABLE 2. OCTAL-DECIMAL FRACTION CONVERSION

OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.
.000	.000000	.100	.125000	.200	.250000	.300	.375000
.001	.001953	.101	.126953	.201	.251953	.301	.376953
.002	.003906	.102	.128906	.202	.253906	.302	.378906
.003	.005859	.103	.130859	.203	.255859	.303	.380859
.004	.007812	.104	.132812	.204	.257812	.304	.382812
.005	.009765	.105	.134765	.205	.259765	.305	.384765
.006	.011718	.106	.136718	.206	.261718	.306	.386718
.007	.013671	.107	.138671	.207	.263671	.307	.388671
.010	.015625	.110	.140625	.210	.265625	.310	.390625
.011	.017578	.111	.142578	.211	.267578	.311	.392578
.012	.019531	.112	.144531	.212	.269531	.312	.394531
.013	.021484	.113	.146484	.213	.271484	.313	.396484
.014	.023437	.114	.148437	.214	.273437	.314	.398437
.015	.025390	.115	.150390	.215	.275390	.315	.400390
.016	.027343	.116	.152343	.216	.277343	.316	.402343
.017	.029296	.117	.154296	.217	.279296	.317	.404296
.020	.031250	.120	.156250	.220	.281250	.320	.406250
.021	.033203	.121	.158203	.221	.283203	.321	.408203
.022	.035156	.122	.160156	.222	.285156	.322	.410156
.023	.037109	.123	.162109	.223	.287109	.323	.412109
.024	.039062	.124	.164062	.224	.289062	.324	.414062
.025	.041015	.125	.166015	.225	.291015	.325	.416015
.026	.042968	.126	.167968	.226	.292968	.326	.417968
.027	.044921	.127	.169921	.227	.294921	.327	.419921
.030	.046875	.130	.171875	.230	.296875	.330	.421875
.031	.048828	.131	.173828	.231	.298828	.331	.423828
.032	.050781	.132	.175781	.232	.300781	.332	.425781
.033	.052734	.133	.177734	.233	.302734	.333	.427734
.034	.054687	.134	.179687	.234	.304687	.334	.429687
.035	.056640	.135	.181640	.235	.306640	.335	.431640
.036	.058593	.136	.183593	.236	.308593	.336	.433593

.037	.060546	.137	.185546	.237	.310546	.337	.435546
.040	.062500	.140	.187500	.240	.312500	.340	.437500
.041	.064453	.141	.189453	.241	.314453	.341	.439453
.042	.066406	.142	.191406	.242	.316406	.342	.441406
.043	.068359	.143	.193359	.243	.318359	.343	.443359
.044	.070312	.144	.195312	.244	.320312	.344	.445312
.045	.072265	.145	.197265	.245	.322265	.345	.447265
.046	.074218	.146	.199218	.246	.324218	.346	.449218
.047	.076171	.147	.201171	.247	.326171	.347	.451171
.050	.078125	.150	.203125	.250	.328125	.350	.453125
.051	.080078	.151	.205078	.251	.330078	.351	.455078
.052	.082031	.152	.207031	.252	.332031	.352	.457031
.053	.083984	.153	.208984	.253	.333984	.353	.458984
.054	.085937	.154	.210937	.254	.335937	.354	.460937
.055	.087890	.155	.212890	.255	.337890	.355	.462890
.056	.089843	.156	.214843	.256	.339843	.356	.464843
.057	.091796	.157	.216796	.257	.341796	.357	.466796
.060	.093750	.160	.218750	.260	.343750	.360	.468750
.061	.095703	.161	.220703	.261	.345703	.361	.470703
.062	.097656	.162	.222656	.262	.347656	.362	.472656
.063	.099609	.163	.224609	.263	.349609	.363	.474609
.064	.101562	.164	.226562	.264	.351562	.364	.476562
.065	.103515	.165	.228515	.265	.353515	.365	.478515
.066	.105468	.166	.230468	.266	.355468	.366	.480468
.067	.107421	.167	.232421	.267	.357421	.367	.482421
.070	.109375	.170	.234375	.270	.359375	.370	.484375
.071	.111328	.171	.236328	.271	.361328	.371	.486328
.072	.113281	.172	.238281	.272	.363281	.372	.488281
.073	.115234	.173	.240234	.273	.365234	.373	.490234
.074	.117187	.174	.242187	.274	.367187	.374	.492187
.075	.119140	.175	.244140	.275	.369140	.375	.494140
.076	.121093	.176	.246093	.276	.371093	.376	.496093
.077	.123046	.177	.248046	.277	.373046	.377	.498046

TABLE 2: OCTAL-DECIMAL FRACTION CONVERSION (Continued)

OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.
.000000	.000000	.000100	.000244	.000200	.000488	.000300	.000732
.000001	.000003	.000101	.000247	.000201	.000492	.000301	.000736
.000002	.000007	.000102	.000251	.000202	.000495	.000302	.000740
.000003	.000011	.000103	.000255	.000203	.000499	.000303	.000743
.000004	.000015	.000104	.000259	.000204	.000503	.000304	.000747
.000005	.000019	.000105	.000263	.000205	.000507	.000305	.000751
.000006	.000022	.000106	.000267	.000206	.000511	.000306	.000755
.000007	.000026	.000107	.000270	.000207	.000514	.000307	.000759
.000010	.000030	.000110	.000274	.000210	.000518	.000310	.000762
.000011	.000034	.000111	.000278	.000211	.000522	.000311	.000766
.000012	.000038	.000112	.000282	.000212	.000526	.000312	.000770
.000013	.000041	.000113	.000286	.000213	.000530	.000313	.000774
.000014	.000045	.000114	.000289	.000214	.000534	.000314	.000778
.000015	.000049	.000115	.000293	.000215	.000537	.000315	.000782
.000016	.000053	.000116	.000297	.000216	.000541	.000316	.000785
.000017	.000057	.000117	.000301	.000217	.000545	.000317	.000789
.000020	.000061	.000120	.000305	.000220	.000549	.000320	.000793
.000021	.000064	.000121	.000308	.000221	.000553	.000321	.000797
.000022	.000068	.000122	.000312	.000222	.000556	.000322	.000801
.000023	.000072	.000123	.000316	.000223	.000560	.000323	.000805
.000024	.000076	.000124	.000320	.000224	.000564	.000324	.000808
.000025	.000080	.000125	.000324	.000225	.000568	.000325	.000812
.000026	.000083	.000126	.000328	.000226	.000572	.000326	.000816
.000027	.000087	.000127	.000331	.000227	.000576	.000327	.000820
.000030	.000091	.000130	.000335	.000230	.000579	.000330	.000823
.000031	.000095	.000131	.000339	.000231	.000583	.000331	.000827
.000032	.000099	.000132	.000343	.000232	.000587	.000332	.000831
.000033	.000102	.000133	.000347	.000233	.000591	.000333	.000835
.000034	.000106	.000134	.000350	.000234	.000595	.000334	.000839
.000035	.000110	.000135	.000354	.000235	.000598	.000335	.000843
.000036	.000114	.000136	.000358	.000236	.000602	.000336	.000846

.000037	.000118	.000137	.000362	.000237	.000606	.000337	.000850
.000040	.000122	.000140	.000366	.000240	.000610	.000340	.000854
.000041	.000125	.000141	.000370	.000241	.000614	.000341	.000858
.000042	.000129	.000142	.000373	.000242	.000617	.000342	.000862
.000043	.000133	.000143	.000377	.000243	.000621	.000343	.000865
.000044	.000137	.000144	.000381	.000244	.000625	.000344	.000869
.000045	.000141	.000145	.000385	.000245	.000629	.000345	.000873
.000046	.000144	.000146	.000389	.000246	.000633	.000346	.000877
.000047	.000148	.000147	.000392	.000247	.000637	.000347	.000881
.000050	.000152	.000150	.000396	.000250	.000640	.000350	.000885
.000051	.000156	.000151	.000400	.000251	.000644	.000351	.000888
.000052	.000160	.000152	.000404	.000252	.000648	.000352	.000892
.000053	.000164	.000153	.000408	.000253	.000652	.000353	.000896
.000054	.000167	.000154	.000411	.000254	.000656	.000354	.000900
.000055	.000171	.000155	.000415	.000255	.000659	.000355	.000904
.000056	.000175	.000156	.000419	.000256	.000663	.000356	.000907
.000057	.000179	.000157	.000423	.000257	.000667	.000357	.000911
.000060	.000183	.000160	.000427	.000260	.000671	.000360	.000915
.000061	.000186	.000161	.000431	.000261	.000675	.000361	.000919
.000062	.000190	.000162	.000434	.000262	.000679	.000362	.000923
.000063	.000194	.000163	.000438	.000263	.000682	.000363	.000926
.000064	.000198	.000164	.000442	.000264	.000686	.000364	.000930
.000065	.000202	.000165	.000446	.000265	.000690	.000365	.000934
.000066	.000205	.000166	.000450	.000266	.000694	.000366	.000938
.000067	.000209	.000167	.000453	.000267	.000698	.000367	.000942
.000070	.000213	.000170	.000457	.000270	.000701	.000370	.000946
.000071	.000217	.000171	.000461	.000271	.000705	.000371	.000949
.000072	.000221	.000172	.000465	.000272	.000709	.000372	.000953
.000073	.000225	.000173	.000469	.000273	.000713	.000373	.000957
.000074	.000228	.000174	.000473	.000274	.000717	.000374	.000961
.000075	.000232	.000175	.000476	.000275	.000720	.000375	.000965
.000076	.000236	.000176	.000480	.000276	.000724	.000376	.000968
.000077	.000240	.000177	.000484	.000277	.000728	.000377	.000972

TABLE 2. OCTAL-DECIMAL FRACTION CONVERSION (Continued)

OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.
.000400	.000976	.000500	.001220	.000600	.001464	.000700	.001708
.000401	.000980	.000501	.001224	.000601	.001468	.000701	.001712
.000402	.000984	.000502	.001228	.000602	.001472	.000702	.001716
.000403	.000988	.000503	.001232	.000603	.001476	.000703	.001720
.000404	.000991	.000504	.001235	.000604	.001480	.000704	.001724
.000405	.000995	.000505	.001239	.000605	.001483	.000705	.001728
.000406	.000999	.000506	.001243	.000606	.001487	.000706	.001731
.000407	.001003	.000507	.001247	.000607	.001491	.000707	.001735
.000410	.001007	.000510	.001251	.000610	.001495	.000710	.001739
.000411	.001010	.000511	.001255	.000611	.001499	.000711	.001743
.000412	.001014	.000512	.001258	.000612	.001502	.000712	.001747
.000413	.001018	.000513	.001262	.000613	.001506	.000713	.001750
.000414	.001022	.000514	.001266	.000614	.001510	.000714	.001754
.000415	.001026	.000515	.001270	.000615	.001514	.000715	.001758
.000416	.001029	.000516	.001274	.000616	.001518	.000716	.001762
.000417	.001033	.000517	.001277	.000617	.001522	.000717	.001766
.000420	.001037	.000520	.001281	.000620	.001525	.000720	.001770
.000421	.001041	.000521	.001285	.000621	.001529	.000721	.001773
.000422	.001045	.000522	.001289	.000622	.001533	.000722	.001777
.000423	.001049	.000523	.001293	.000623	.001537	.000723	.001781
.000424	.001052	.000524	.001296	.000624	.001541	.000724	.001785
.000425	.001056	.000525	.001300	.000625	.001544	.000725	.001789
.000426	.001060	.000526	.001304	.000626	.001548	.000726	.001792
.000427	.001064	.000527	.001308	.000627	.001552	.000727	.001796
.000430	.001068	.000530	.001312	.000630	.001556	.000730	.001800
.000431	.001071	.000531	.001316	.000631	.001560	.000731	.001804
.000432	.001075	.000532	.001319	.000632	.001564	.000732	.001808
.000433	.001079	.000533	.001323	.000633	.001567	.000733	.001811
.000434	.001083	.000534	.001327	.000634	.001571	.000734	.001815
.000435	.001087	.000535	.001331	.000635	.001575	.000735	.001819
.000436	.001091	.000536	.001335	.000636	.001579	.000736	.001823

.000437	.001094	.000537	.001338	.000637	.001583	.000737	.001827
.000440	.001098	.000540	.001342	.000640	.001586	.000740	.001831
.000441	.001102	.000541	.001346	.000641	.001590	.000741	.001834
.000442	.001106	.000542	.001350	.000642	.001594	.000742	.001838
.000443	.001110	.000543	.001354	.000643	.001598	.000743	.001842
.000444	.001113	.000544	.001358	.000644	.001602	.000744	.001846
.000445	.001117	.000545	.001361	.000645	.001605	.000745	.001850
.000446	.001121	.000546	.001365	.000646	.001609	.000746	.001853
.000447	.001125	.000547	.001369	.000647	.001613	.000747	.001857
.000450	.001129	.000550	.001373	.000650	.001617	.000750	.001861
.000451	.001132	.000551	.001377	.000651	.001621	.000751	.001865
.000452	.001136	.000552	.001380	.000652	.001625	.000752	.001869
.000453	.001140	.000553	.001384	.000653	.001628	.000753	.001873
.000454	.001144	.000554	.001388	.000654	.001632	.000754	.001876
.000455	.001148	.000555	.001392	.000655	.001636	.000755	.001880
.000456	.001152	.000556	.001396	.000656	.001640	.000756	.001884
.000457	.001155	.000557	.001399	.000657	.001644	.000757	.001888
.000460	.001159	.000560	.001403	.000660	.001647	.000760	.001892
.000461	.001163	.000561	.001407	.000661	.001651	.000761	.001895
.000462	.001167	.000562	.001411	.000662	.001655	.000762	.001899
.000463	.001171	.000563	.001415	.000663	.001659	.000763	.001903
.000464	.001174	.000564	.001419	.000664	.001663	.000764	.001907
.000465	.001178	.000565	.001422	.000665	.001667	.000765	.001911
.000466	.001182	.000566	.001426	.000666	.001670	.000766	.001914
.000467	.001186	.000567	.001430	.000667	.001674	.000767	.001918
.000470	.001190	.000570	.001434	.000670	.001678	.000770	.001922
.000471	.001194	.000571	.001438	.000671	.001682	.000771	.001926
.000472	.001197	.000572	.001441	.000672	.001686	.000772	.001930
.000473	.001201	.000573	.001445	.000673	.001689	.000773	.001934
.000474	.001205	.000574	.001449	.000674	.001693	.000774	.001937
.000475	.001209	.000575	.001453	.000675	.001697	.000775	.001941
.000476	.001213	.000576	.001457	.000676	.001701	.000776	.001945
.000477	.001216	.000577	.001461	.000677	.001705	.000777	.001949

TABLE 3. HEXADECIMAL-DECIMAL CONVERSION (TWO-WAY)

Dec.	Hex.		Dec.
$10^0$	1	$16 \times 1$	16
$10^1$	a	2	32
$10^2$	64	3	48
$10^3$	3e8	4	64
$10^4$	2710	5	80
$10^5$	186a0	6	96
$10^6$	f4240	7	112
$10^7$	989680	8	128
$10^8$	5f5e100	9	144
$10^9$	3b9aca00	10	160
$10^{10}$	2540be400	11	176
$10^{11}$	174876e800	12	192
$10^{12}$	e8d4a51000	13	208
		14	224
		15	240
		16	256
	Dec.		Dec.
$16^0$	1	$16^7$	268435456
$16^1$	16	$16^8$	4294967296
$16^2$	256	$16^9$	68719476736
$16^3$	4096	$16^{10}$	1099511627776
$16^4$	65536	$16^{11}$	17592186044416
$16^5$	1048576		
$16^6$	16777216		

TABLE 4. POWERS OF 2 (POSITIVE AND NEGATIVE)  
EXPRESSED IN DECIMAL

$n$	$2^n$	$n$	$2^{-n}$
0	1	0	1.0
1	2	1	0.5
2	4	2	0.25
3	8	3	0.125
4	16	4	0.062 5
5	32	5	0.031 25
6	64	6	0.015 625
7	128	7	0.007 812 5
8	256	8	0.003 906 25
9	512	9	0.001 953 125
10	1 024	10	0.000 976 562 5
11	2 048	11	0.000 488 281 25
12	4 096	12	0. <sup>3</sup> 244 140 625
13	8 192	13	0. 122 070 312 5
14	16 384	14	0. 061 035 156 25
15	32 768	15	0. 030 517 578 125
16	65 536	16	0. <sup>3</sup> 015 258 789 062 5
17	131 072	17	0. 007 629 394 531 25
18	262 144	18	0. 003 814 697 265 625
19	524 288	19	0. 001 907 348 632 812 5
20	1 048 576	20	0. <sup>6</sup> 953 674 316 406 25
21	2 097 152	21	0. 476 837 158 203 125
22	4 194 304	22	0. 238 418 579 101 562 5
23	8 388 608	23	0. 119 209 289 550 781 25
24	16 777 216	24	0. <sup>6</sup> 059 604 644 775 390 625
25	33 554 432	25	0. 029 802 322 387 695 312 5
26	67 108 864	26	0. 014 901 161 193 847 656 25
27	134 217 728	27	0. 007 450 580 596 923 828 125
28	268 435 456	28	0. <sup>6</sup> 003 725 290 298 461 914 062 5
29	536 870 912	29	0. 001 862 645 149 230 957 031 25
30	1 073 741 824	30	0. 000 931 322 574 615 478 515 625
31	2 147 483 648	31	0. 000 465 661 287 307 739 257 812 5
32	4 294 967 296	32	0. <sup>9</sup> 232 830 643 653 869 628 906 25
33	8 589 934 592	33	0. 116 415 321 826 934 814 453 125
34	17 179 869 184	34	0. 058 207 660 913 467 407 226 562 5
35	34 359 738 368	35	0. 029 103 830 456 733 703 613 281 25
36	68 719 476 736	36	0. <sup>9</sup> 014 551 915 228 366 851 806 640 625
37	137 438 953 472	37	0. 007 275 957 614 183 425 903 320 312 5
38	274 877 906 944	38	0. 003 637 978 807 091 712 951 660 156 25
39	549 755 813 888	39	0. 001 818 989 403 545 856 475 830 078 125
40	1 099 511 627 776	40	0. <sup>12</sup> 909 494 701 772 928 237 915 039 062 5
41	2 199 023 255 552	41	0. 454 747 350 886 464 118 957 519 531 25
42	4 398 046 511 104	42	0. 227 373 675 443 232 059 478 759 765 625
43	8 796 093 022 208	43	0. 113 686 837 721 616 029 739 379 882 812 5
44	17 592 186 044 416	44	0. <sup>12</sup> 056 843 418 860 808 014 869 689 941 406 25

TABLE 5. HEXADECIMAL MULTIPLICATION

1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
2	4	6	8	a	c	e	10	12	14	16	18	1a	1c	1e
3		9	c	f	12	15	18	1b	1e	21	24	27	2a	2d
4			10	14	18	1c	20	24	28	2c	30	34	38	3c
5				19	1e	23	28	2d	32	37	3c	41	46	4b
6					24	2a	30	36	3c	42	48	4e	54	5a
7						31	38	3f	46	4d	54	5b	62	69
8							40	48	50	58	60	68	70	78
9								51	5a	63	6c	75	7e	87
a									64	6e	78	82	8c	96
b										79	84	8f	9a	a5
c											90	9c	a8	b4
d												a9	b6	c3
e													c4	d2
f														e1

#### 4. PROGRAM STRUCTURE AND FLOW DIAGRAMS

##### Logical Structure of a Program

The general purpose stored program digital computer requires the preparation of an external set of instructions which may be formulated in terms of a language similar to the data language of the computer. For example, the instructions may also be binary numbers if numbers are operated on in the binary notation, decimal numbers if the internal number structure is decimal, or alphanumeric characters if the basic data language is alphanumeric. These instructions must be listed in detail, one after another, to provide the procedure or algorithm by which the machine is to perform the problem. Instructions must be exact; only a few machines have even rudimentary built-in mistake detection circuits by which programmer mistakes in the use of machine or instruction conventions can be caught.

**Control Function.** That portion of the computer which automatically executes the previously prepared program instructions, generally stored before performance in the same storage locations as the numbers, is called the *control* or *control unit* (see Chap. 18). In general, the sequence of instruction through the program is called the path of control. It may be considered a one-dimensional path, and it can often be represented on paper in the form of a sequence or flow diagram, where the path of control is designated by a directional line.

The control serves as a sort of instructional interpreter, which selects the pertinent instruction, decodes it into its fundamental components,

which in general represent one or more operands and a particular operation, and then proceeds onto the next instruction, which may be located by convention in the next machine position in sequence, or else may be given by the previous instruction itself.

**Sequential and Concurrent Control.** In the simplest case, the control will decode and perform one instruction at a time. Such computer operation is called *sequential* and is generally used in digital computers. *Concurrent* computer operation is a more complex one in that two or more different machine instructions, for example, reading and adding or adding and multiplying, may be going on at the same time. Most computers are either entirely sequential, or have only input-output operations performing at the same time as internal computational instructions. The intermeshing of such input-output operations with calculations for such computers can become an intricate part of the programming.

**Iteration Loops.** The passage of control through the sequence of instructions will not necessarily be a linear one, but will jump back and forth among the instructions. Most problems are not expressed in a linear fashion, but rather recursively, by the use of recursive functions or iteration loops. An example is the addition of a column of numbers.

1. *Group Summation.* Addition directly by human beings generally may be considered addition of all of a group of  $n$  numbers  $x_i$  ( $i = 1, 2, \dots, n$ ) at once, since most persons add column by column. Such a procedure for obtaining the sum  $S$  may be written symbolically as

$$(14) \quad S = \sum_{i=1}^n x_i,$$

where the  $\sum$  may be considered a simultaneous operator on all the  $x_i$ .

2. *Stepwise Addition.* On the other hand, in using a desk calculator, or an automatic stored program digital computer, the addition is done two numbers at a time, and may be expressed thus symbolically:

$$(15a) \quad s_1 = x_1,$$

$$(15b) \quad s_{i+1} = s_i + x_{i+1} \quad (i = 1, 2, \dots, n - 1),$$

$$(15c) \quad S = s_n.$$

Since very few automatic calculators possess a summation instruction in the sense of eq. (14), most additions of numbers are performed recursively, as in eqs. (15), with an iteration loop technique.

3. *Straight Line Coding.* The sum of a sequence of numbers may be obtained by a linear or sequential technique, which consists of "unwind-

ing" the iteration loop to produce a sequence of approximately  $n + 1$  instructions. These may be given symbolically by

$$\begin{aligned}
 (16) \quad & s_1 = x_1 \\
 & s_2 = s_1 + x_2 \\
 & s_3 = s_2 + x_3 \\
 & \cdot \\
 & \cdot \\
 & \cdot \\
 & s_{n-1} = s_{n-2} + x_{n-1} \\
 & s_n = s_{n-1} + x_n \\
 & S = s_n.
 \end{aligned}$$

**Comparison.** The summation procedure requires only one high-powered built-in machine instruction; the linear addition technique requires one stored instruction per addition. The iteration loop procedure occupies a compromise position between these two extremes. Actually, of the three equations given in (15), (15a) and (15c) are expressions which indicate the start and finish of the overall procedure, and correspond to the first and last equations of (16). The sequence of values given for  $i$  in the parentheses to the right of (15b) indicates that that equation must be performed for each value of  $i$  from 1 through  $n - 1$ . In the iteration loop procedure usually only one computer instruction is stored to correspond to (15b), but it is considered a function of  $i$ , and modified to perform the addition over and over again the required number of times. The machine must perform this modification of the addition instruction and at the same time decide when it has performed the operation corresponding to (15b) the proper number of times.

**Loop Control.** Such behavior of a general purpose computer is called loop control, and requires that the machine have three basic properties. It should be able to (1) modify its own instructions by some method, (2) change the sequence of the path of control through the stored instructions, and (3) make decisions on the basis of its partial results. In particular, in case of the iteration procedure given in eqs. (15), the computer must be able to decide when it has performed (15b)  $n - 1$  times. A diagram of the iterative procedure in (15) is given in Fig. 7. Theoretically, the third requirement is included in the other two. Practical machines have all three requirements fulfilled.

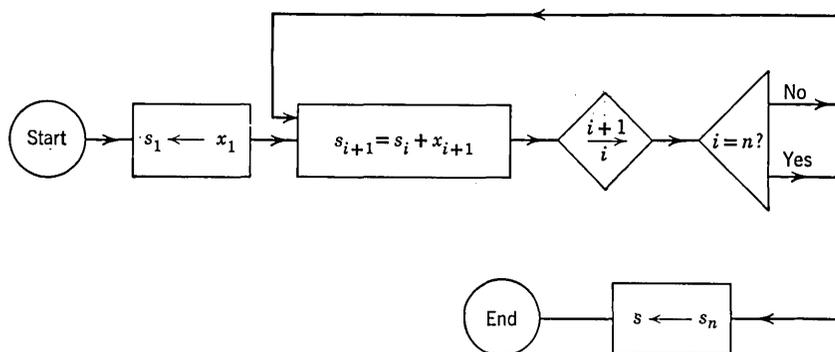


Fig. 7. Diagram of iterative procedure.

### Flow Diagrams

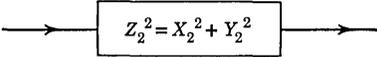
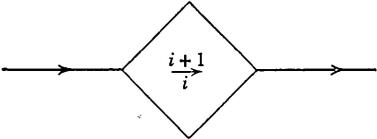
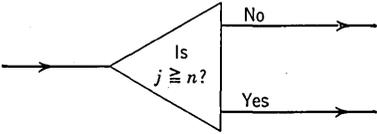
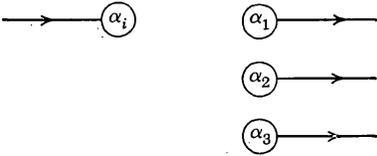
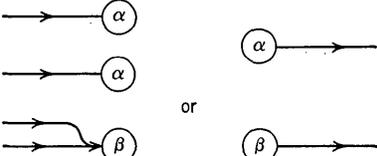
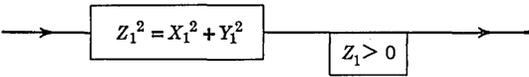
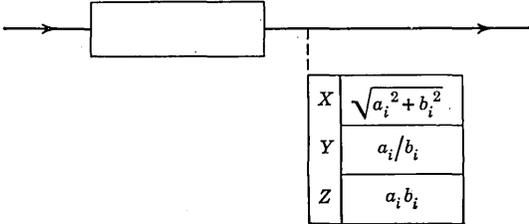
The diagram of Fig. 7 gives a geometric picture of the path of control for eqs. (15), represented by linear path, which in this case intersects itself. The presence of a closed loop in the diagram indicates a recursion or iteration. Such diagrams can be extremely helpful, as they have previously been with chemical processes, analog computers, and business procedures analyses, in presenting an overall picture of the structure of the process described.

The flow diagram is an attempt at a graphical portrayal of the course of the problem's solution, later to be approximated by the path of the machine's control through the instruction information stored in the machine during the course of solution. The flow diagram shall consist of a sequence of directed line segments or curves, called the *control line*, extending from one "box" or "block" on paper to the other boxes or blocks. Each box shall indicate an arithmetic, instructional, or index change, or an assertion about the status of the problem.

**Notation for Flow Diagrams.** Several different notations have been devised for giving as exact a picture of the problem process as possible. Such diagrams are called *flow* or *sequence diagrams*; the original notation was introduced by Burks, Goldstine, and von Neumann (see Ref. 19), and is particularly applicable to the computers with single-address logic of the type first designed at the Institute for Advanced Study. The flow diagram procedure developed by the Princeton group was adapted to one particular machine; a generalized notation is shown in Table 6. The notation of this table is aimed at providing a general language useful for all machines. Individual users may change the notation given here to suit particular machines for which they are preparing problems.

**Advantages of a General Notation.** The flow diagram is the closest to a "universal machine language" that has been developed. Use of such

TABLE 6. PROGRAMMING FLOW DIAGRAM NOTATION

Box	Notation						
(a) Calculation							
(b) Indicial operations							
(c) Alternative							
(d) Variable remote connections							
(e) Equivalent fixed remote connections							
(f) Assertion or information							
(g) Table of variables	 <table border="1" data-bbox="709 1395 865 1545"> <tr> <td>X</td> <td><math>\sqrt{a_i^2 + b_i^2}</math></td> </tr> <tr> <td>Y</td> <td><math>a_i/b_i</math></td> </tr> <tr> <td>Z</td> <td><math>a_i b_i</math></td> </tr> </table>	X	$\sqrt{a_i^2 + b_i^2}$	Y	$a_i/b_i$	Z	$a_i b_i$
X	$\sqrt{a_i^2 + b_i^2}$						
Y	$a_i/b_i$						
Z	$a_i b_i$						

a common language, not aimed at a particular machine, (1) in the formative stages of problem preparation delays narrowing down problem expression to one particular machine to the later stages, allows easy use of a particular flow diagram on any machine, and (2) permits easier exchange of methods between users of different machines. The value of the flow diagram is its use as a representation of a general solution of the problem, not of any *particular machine's* solution.

**Structure of Flow Diagrams.** The control line of the flow diagram and its various parts may come together at any point, indicating that two possible courses of the path of control through the instructional machine information have formed a junction; one line segment may split into two or more to form a disjunction or branch point. Every branch point must be immediately preceded by a box or boxes indicating the cause of the branching or the method by which control is to decide which path to take.

A linear sequence of operations, with no decisions to be made by the control, and therefore no branching, will consist of a continuous control line interrupted only by boxes. In the case of iterative calculations or data processing sequences in which operations will be performed in a repetitive or near-repetitive fashion, loops will appear in the flow diagram. Each such loop will consist of a curve equivalent to a circle, interrupted by one or more boxes.

Ordinarily, most flow diagrams will be two dimensional, and can be laid out on a two-dimensional surface without crossovers. However, it is possible for problems to require flow diagrams with crossovers in the control lines. For this reason, and for the purpose of providing uncrowded layouts in two-dimensional problems, the flow diagram structure can be broken up into a set of smaller diagrams, with particular notation indicating the connections between the various diagrams.

**Flow Diagram Boxes.** There are two types of *operation boxes*, indicial operations boxes, connected with the formation of indices and machine addresses ("red tape" boxes), and calculational operation boxes, representing actual numerical calculations as shown in Table 6a, b. Any operation box of either type must have only one input and one output path. *Alternative boxes* (see Table 6c) usually ask questions, answered by a "yes" or "no," but they may often indicate other types of binary decisions.

*Remote Connections.* Because a problem may require a multichoice rather than binary disjunction or branching at a particular stage, and since the machine, by assumption, has the property of modifying its own sequence of instructions, a particular representation is needed for multiple disjunctions. Such representation is given by a *variable remote*

*connection.* This is represented by termination of the control line by a circle marked with a Greek letter with a Roman letter subscript, indicating a variable terminal. Alternative continuations from this circle will be originated at other circles located at a distance, each marked with the same Greek letter, properly indexed by successive integers. (See Table 6d.)

In some cases, need of space on paper, or lack of two dimensionality in the diagram may require a *fixed remote connection*. Such a connection shown in Table 6e consists of a terminating circle on the control line followed by an originating circle some distance away, beginning a new control line. Fixed remote connections will be denoted by Greek letters without subscripts, indicating their nonchanging value. It is sometimes helpful, because of the geometry of the diagram, to have two terminating circles on a diagram referring to one originating circle, indicating the equivalent of a junction preceding one terminating circle.

An *assertion or information* block (see Table 6f) contains no information of operational significance, but it is only present in order to assert the validity of certain relationships in the problems, during the course of control, or to provide useful information for the formulator or reader. Such boxes will be rectangles "hung" from the control line with the line itself serving as one side of the rectangle and will contain one or more equalities, inequalities, or similar relations, as in Table 6f.

A *table of variables* is purely an explanatory and storage device, similar to the assertion boxes. It can take two forms; an overall table, giving sequential values of each floating variable as functions of the bound variables for each constancy interval, and a running table, in which portions of the overall table are stored directly adjacent to each constancy interval. A combination of both may be used, if desired. The running table of variables is usually attached by a dashed line to the control line somewhere on the constancy interval, as in Table 6g.

In many practical cases, neither assertion boxes nor tables of variables will be used by a problem formulator. Nevertheless, in the case of complicated problems, their value cannot be overlooked.

**Notation for Constants and Variables.** Standard practice in labeling is as follows:

1. Constants and variables: Roman letters followed by decimal integer subscripts.
2. Constants: *c*'s.
3. Arithmetic variables: end-of-the-alphabet letters such as *w, x, y, z*.
4. Indices: middle-alphabetical characters such as *i, j, k, l, m*, and *n*.

The calculation variables are most often *free variables*, able to be changed at the beginning of a problem. In the sequence diagram lan-

guage, free variables offer no difficulty; they belong to the same class as constants as far as all except initial behavior is concerned, and are generally changed by operation boxes. *Floating variables* are not assignable from the beginning. Once they are defined in terms of free or previously defined floating variables, they have the same behavior. When the values of the *bound variables* (usually indices) that assume a sequence of different values during the course of a problem solution change, however, such variables are changed in actual, although not notational value. By a change in such an index, therefore, no change is made in the flow diagram notation, although the actual value of free or floating variables will change.

A *transition point* is defined as a point on the control line where the value or domain of variation of any bound variable changes. The interval between two consecutive points along the control line will then be called a *constancy interval*. A constancy interval will be marked by transition points at its beginning and end (sometimes the same point) as in Fig 8.

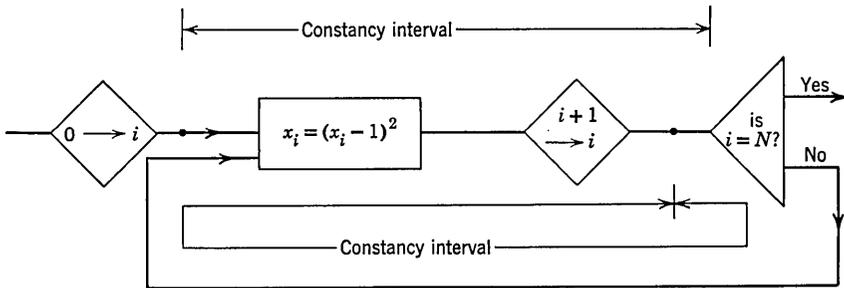


FIG. 8. Constancy intervals. Problem, form  $(x_0)^{2N}$ .

Floating variables within such an interval will remain unchanged, at least when considered as direct, but not indirect, functions of the bound variable in question.

By virtue of the more generalized definition given here, transition points will occur at indicial operations boxes. Finally, constancy intervals can be enumerated by a sequential decimal notation similar to the standard library Dewey decimal system.

*Contents of Boxes:* The contents of the various operational boxes shall in all cases express floating or bound variables in terms of other such variables. Indicial operational boxes, which mark transition points, if expressing a bound variable as a function of the same bound variable (in a previous iteration), will use an arrow to indicate that a function of the value held during the immediately preceding constancy interval for purposes of calculation is to replace the old value (see Table 6b). A

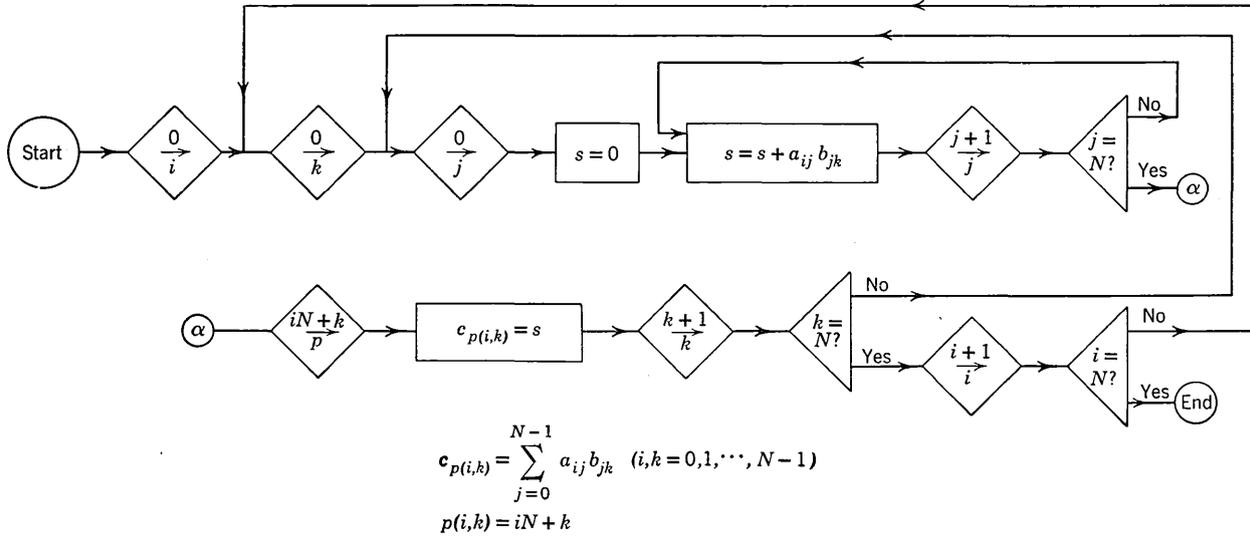


FIG. 9. Flow diagram for multiplication of two matrices.

calculational operational box will be a functional equality expressing the new value as a function of the previous value of the variable being obtained. No confusion will arise here, since floating variables will be direct functions of indices, whenever they are functions also of themselves. In such cases, the relationship will be expressed as an equation, with the new variable to the left of the equality sign.

**EXAMPLE.** A flow diagram, drawn with the above conventions, is given for the multiplication of two matrices in Fig. 9.

The problem required is, given two matrices, each stored in a one-dimensional sequence, to form their product and store it in a similar one-dimensional storage. The product matrix

$$c_{ik} = \sum_{j=0}^{n-1} a_{ij}b_{jk} \quad (i, k = 0, 1, \dots, n-1)$$

is to be formed so that each element  $c_{ik}$  is stored in a one-dimensional position  $c_p(i, k)$ . This is a practical problem in that all present-day storage can be considered, for practical purposes, one dimensional.

**Machine Application.** As the drawing of Fig. 9 is shown it represents a formulation that may be translated in a *static* fashion into any machine instructional code that obeys the assumptions of this chapter. Certain difficulties will arise in performing such a translation to any one particular machine, particularly if an attempt is made to make the coding process most efficient time-wise or space-wise. For example, in order to save space, more than one floating variable (temporary value) should perhaps be stored in one storage position in the particular machine. Moreover, since there are variants in the formulation of most problems that will still give perfectly correct solutions, it is probable that some variants will be particularly suited to direct static coding on one machine, while other variants will be suitable for others. A programmer who has gained experience on one particular machine will soon learn to develop those flow diagram formulation variants that will prove suited to the particular code required by his machine.

## 5. MACHINE LOGIC

The modern electronic computer or information processor consists of the following parts: the control, the storage, the arithmetic and logic unit, and the input-output unit. The control unit is responsible for the initiation and monitoring of all computer operations (see Chap. 18). Instructions from the storage are *translated* or *decoded* in the control unit and as a result the computer performs transfers, computing, logic, and input-output operations. Hence, transfers into and out of the computer, transfers between the various computer units, and appropriate

processing of information is automatically carried out as a result of instruction translation.

### Internal Machine Structure

**Loop Control.** Digital computer programming, in the usual sense, is the formalization of problem solution for general purpose stored program electronic digital computers. Such machines are said to have loop control if they have the following characteristics:

1. Numbers and instructions are physically indistinguishable as to storage position and method of storage. The storage may be shared in any ratio between numbers and instructions.

2. The ability to modify instructions in the computer's arithmetic unit. This enables the machine to *modify its own syntax*, that is, change its own logical structure.

3. The passage of control through the sequence of instructions can be modified on the basis of the contents of one or more accessible positions inside the machine's storage, arithmetic, or control units. Usually, the only decision producing information is the sign of the number in a specified location in the arithmetic unit (accumulator) or equivalently, the relative size of two numbers in the storage. Other portions of computers that have been used in the past for decision making are external human-operated switches, electronic switches operated by other machines (interrupt features), internal switches operated by overflow and other error detection devices, etc.

**Variable Instruction Computers.** The presence of these loop control features requires that a machine belong to the class of *variable* instruction computers, in which instructions may be changed externally and are not built into the hardware. This class may be divided into two types, depending on how instructions are modified:

1. *Indicial instruction* logic, with a separate, special instruction arithmetic element in addition to the standard arithmetic element for operating on both numbers and instructions.

2. *Ambiguous-word* logic, with both number and instruction words sharing the same storage and handled only by the same (arithmetic) processing element.

**Indicial Instruction Machines.** This design resulted from the need for two levels of arithmetic, (a) the actual computations themselves and (b) computations necessary to determine the values of indices and therefore addresses of data. Indices are always integers; numerical data will be manipulated generally as fixed or floating point numbers with a much larger number of significant figures. Index and address computations, or "red tape" operations are performed on integers requiring far fewer significant digits.

Separate arithmetic elements for numbers and instructions work best on computers with serial storage and transfer of information, since in such machines arithmetic elements can be inserted in instruction-handling paths without loss of operating time or large increase in cost. Many computers with parallel storage and data transfer, however, now have such separate arithmetic elements.

It should be noted that the computer with purely indicial instruction logic is virtually nonexistent. Under the definitions given here, most modern computers, especially those of general purpose, are variable instruction computers which are hybrids of the indicial instruction and ambiguous-word types. Most have ambiguous-word logic and many of these, in addition, have special capability such as the B box (see below) which gives them an indicial instruction character.

**B Box or Index Register.** A B box or automatic instruction modifier is an example of such a separate arithmetic unit. Many artificial instruction codes (abstractions) use such index registers to speed up hand programming. A certain portion of every instruction word is used to designate just how that particular instruction is to be modified with respect to one or more such special locations, which have been filled with specified values of an index by previously performed instructions. Usually the contents of such a special register is added to (or subtracted from) the address(es) of the instruction. Loop control is obtained by special instructions performing complex index comparisons combined with control transfer. Often arithmetic operations themselves can be performed on the (usually integer) numbers stored in the index registers, by means of special operations. The proper use of index registers can save both time and storage space over the methods required when they are not available.

A rather exceptional case is the "repeat instruction" of the Univac Model 1103-A (see Sect. 6). This instruction specifies the number of times the following instruction is to be repeated and how the addresses are to be modified.

**Ambiguous-Word Manipulations.** In this case, instructions are treated as numbers and operated upon in the common arithmetic unit. Indices must be stored with the proper scale factors so that they can be added directly to instructions. Comparisons are now made with respect to certain comparison instructions which specify the state at the end of an entire iteration process of an instruction being varied.

**Relative Advantages.** The use of separate indices and their separate manipulation in index registers is mathematically more satisfying with respect to hand programming. With respect to machine-composed programs, the optimum use of B registers may make the automatic programming technique itself more complicated. For hand programming, the

pedagogical difficulties inherent in considering numbers as instructions and instructions as numbers can raise difficulties in the learning process.

**Machine-Level Languages.** The instruction code of a digital computer is indeed a command language, obeying all the rules of formal logical languages, and one that can be studied as such. Machine-level languages were basically designed for two purposes: (1) to allow control of machine performance by human beings and (2) to require as little complex circuitry as possible for mechanizing this control.

The latter condition arose because the original computer designers were more concerned with building operable machines than the ease of operation by humans.

**Basic Units.** The basic unit in a digital computing machine is called a *word*. Such a word is an informational unit of more than one digit, which can be transported as a whole from one part of the machine to another. Most storage systems, to be used most efficiently, require manipulation of fixed length groups of digits at one time. Thus most machines have fixed word lengths and fixed instruction formats. The advent of variable word length data processing equipment (IBM 702 and 705, Bizmac) have shown designers that storage used ingeniously can allow variable data groupings. Variable groupings of storage digits into instructions is planned for new computers such as Stretch and Gamma 60. In some machines instructions are multiples of one, two, or three standard lengths.

Each word is given an *address* or location number. The instruction words that the machine uses contain one or more addresses or datum numbers, one or more *operations*, and numerous modifiers, which can modify both the address and operation portions of such an instruction. Machine languages have so far been *positional*, with location in the instruction being the identifying tag as to the fashion in which an address is to be interpreted.

**Multi-Address Codes.** The instructions with which machines are supplied serve only to express manipulations on numbers (usually arithmetic) or in some cases on instructions themselves. The most general such instructions require a set of basic information:

1. The addresses of the operands or numbers to be operated on.
2. The location or address to which the result is to be sent.
3. The numerical (or logical) operation to be performed.
4. The location or address of the next instruction.

Figure 10 summarizes the structures of the various instruction types discussed below.

**Four-Address Codes.** In the simpler arithmetic operations such as addition, subtraction, multiplication, and division, there are only two

Operation code	First operand address	Second operand address	Result address	Next instruction address
----------------	-----------------------	------------------------	----------------	--------------------------

Four-address instruction

Operation code	First operand address	Second operand address	Result address
----------------	-----------------------	------------------------	----------------

Three-address instruction

Operation code	First operand address	Second operand address
----------------	-----------------------	------------------------

Two-address instruction

Operation code	Operand address	Next instruction address
----------------	-----------------	--------------------------

One + one-address instruction

Operation code	Operand address
----------------	-----------------

One-address instruction

FIG. 10. Instruction types.

operands and one result. Thus, including the location of the next instruction as an additional address, the most general type of instruction for the elementary operations mentioned should contain four addresses. Such a four-address code was originally devised for the EDVAC and SEAC (Ref. 92). The fourth or subsequent instruction address has the specific function of allowing minimal access machine performance if the programmer so desires and it also simplifies changing or correcting a program in machine language.

**Three-Address Codes.** With parallel machines, instructions are normally assumed to be stored in *sequential* address positions. Random

access noncyclic storage machines, therefore, can without loss in efficiency dispense with one of the four addresses, if provision is made for special instructional operations to provide a change of control if desired. It is therefore possible for a three-address code in such a machine to be highly efficient. In addition, numerous cyclic machines have sacrificed the subsequent address in their instruction codes to obtain simpler formats. The NORC is a random-access three-address instruction parallel machine (see Ref. 111); the MIDAC is a machine with cyclic storage and a three-address instruction code with no subsequent address in its instruction (see Ref. 25).

**Two-Address Codes.** If again, by convention, some of the operand or result addresses required by a three-address code are considered as fixed locations, it is possible to eliminate one or two other addresses. Several two-address machines have been designed. Such instruction logic offers instructions with addresses of one operand and a result, or an operand and a subsequent address, or combinations of the other possible variants, with a standard location again being used. The Univac 1103, 1103A, and 1105 are two-address instruction machines (Ref. 103).

**Single-Address Instructions.** The standard location mentioned above has most often been called an *accumulator* by the designers, after the nomenclature of the ordinary desk calculator. In parallel machines, it is usually a part of the adder and multiplier, and thus serves an operational as well as storage purpose. With such a storage position, the number of addresses required in an instruction can be cut to one. In such a one-address or single-address code, in the simpler arithmetic orders one operand is by convention in the accumulator, and the result is left in the accumulator. Separate instructions are needed to store results obtained in the accumulator, when desired.

The trend has been, on the grounds of machine simplicity and simplicity of coding, toward single-address machines. Examples are the Univac I and II, the SAGE System Air Defense Computers, the IBM 700 Series, and the "Princeton Class" of equipments.

Machines have even been proposed with no addresses, but rather operands themselves, stored in the instructions.

**One-Plus-One Address Codes.** In the one-plus-one addressing procedure, each instruction has a basic single-address format, but also includes a second address to be used to designate the location of the next instruction to be performed. This allows minimal access programming with the cyclic storage system on a magnetic drum. The IBM 650 is a machine with such a structure (Ref. 102).

**Circulating Loops.** A method used with single-address magnetic drum computers to provide efficiency comparable to the one-plus-one

addressing system is the use of the circulating loop which provides a smaller, more rapid access storage region where instructions may be performed, and data stored and retrieved. The presence of such a smaller, speedier storage system, surrounded by a larger, slower storage region, provides an example of the storage hierarchy problem in programming. (See Ref. 32.)

**A "Best" Addressing System.** Programmers and designers have long argued the merits of one or the other instruction format for programming ease and ease of machine construction and design. A formal study was made by Elgot (Ref. 33), who compared single-address versus three-address codes with results interpretable in favor of either system depending upon the measures of effectiveness to be used. The advent of newer storage systems and automatic programming techniques involving translation from extremely flexible external languages has generally eliminated the controversy. Future designs will depend on: (1) ease of design compatibility of instructions with hardware and (2) ease of machine programming of itself (automatic programming, see Sects. 8 to 15) in the most efficient manner, rather than the ease of human hand programming. Experience with all systems of instruction logic would indicate that extreme flexibility would probably be most compatible with (2), if not with (1).

**Operations.** Usually, each instruction contains only one operation, although attempts have been made on some machines (such as the Illinois computer, ILLIAC) (Ref. 56), to systematize the instructions, in that they are actually made up of a number of separate commands which can be combined in a straightforward manner into one operation. There are six different types of operations that are usually available, as given below.

### **Operating Types.**

1. *Arithmetic Operations.* These include the four basic operations of addition, subtraction, multiplication, and division, with several variants; scale-factoring or numerical normalization operations; number shifting by multiplication by powers of the radix, and special purpose combinations of all of these.

2. *Digital Logical Operations.* These operations include means of replacing specific digits of one number by those of another, circular shifting of numerical storage, and various operations to aid in the modification of instructions, such as address substitution, and storage of the present address of control for later use.

3. *Decision Operations.* Most computers make use of the sign of numbers or a comparison with zero to effect assignment of the next step

of control to one of two locations in the storage. Such assignment may be made after forming differences of numbers or of the absolute values of numbers in the multiaddress machines. Other instructions will cause the machine to repeat subsequent instructions one or more times dependent on contents of certain registers, etc.

4. *Change of Control Operations.* In addition to the conditional change of control operations given under (3), it is usually useful to have an unconditional change of control operation available to provide for jumps in the instructional sequence. Even so, conditional change of control operations can be used by means of forced comparisons to provide unconditional control transfers.

5. *Storage and Input-Output Operations.* In machines with accumulators, it is usually necessary to provide operations permitting storage of the accumulator contents. Similarly, with machines with more than one (high-speed and medium-speed, for example) type of storage, instructions are required to transfer information back and forth. Access to the external read and write devices must also be controlled by special input-output operations.

6. *Indexing Operations.* These are used with the index registers to alter, in the control unit, just prior to execution, the address or addresses of the instruction to be performed. Generally the instruction in storage itself is not changed. Indexing operations may be involved with decisions (ending an iteration after a count down in an index register), and with input-output (using the contents of an index register to specify where the next input is to be stored, etc.). Included in this category are operations specifically designed to alter instruction operands in machines without index registers.

**Complexity of Operations.** Square roots and trigonometric functions can be built in as operations in any machine, but this is not usually done. An exception is the Soviet computer Strela, described in Sect. 6. Such functions, and others like them, can be formed satisfactorily by purely coded methods. The gain in shorter performance time and in decreased instructional storage over coded methods for such built-in operations has not overcome the increased cost of equipment.

Finally, the use of synthetic instructions as part of assembly programs provides a method to effectively increase the computer's repertoire of instructions without the necessity of building in further machine "hardware." Such coded tools may tend to provide more operations in future machines, without added machine cost.

**Address Modification Methods.** In the indicial instruction machines, some of which also allow ambiguity in the words used, a special portion of each instruction is used as an address modifier, to indicate whether

or not an address or addresses are to be modified before the instruction is performed. This modification usually consists of (a) addition or subtraction of a B line, base counter, or index register to or from the address before it is used to select a particular word, or (b) the use of an index register to perform automatic counting for inductive loops, or to store return addresses for use with side passages through subroutines or pre-coded, pre-tested standard coded functional operations.

More than one index register may be included to allow numerous loop paths of control. If only one is available, provisions are usually made to exchange its contents with that of an address in the other available storage. If considered in index notation, such B line devices provide an easy means for almost direct static translation for the notation of numerical analysis, without the intervention of a flow diagram.

**Breakpoint Notations.** Some machines provide breakpoint information in an instruction. This usually consists of one binary digit or a decimal digit value that can be used to indicate special action—usually special printing of several of the significant pieces of information in a particular instruction. The latter process, termed “automonitoring” by some groups, and “mistake or error diagnosis,” or “checking” by others, combined with the special breakpoint behavior provides useful running information of a program’s progress. Most machines do not include this feature built-in, but use programmed “trace” routines to provide a running account of the performance of the problem.

**Word Length and Instruction Types.** Most computers, especially those for scientific and engineering calculations, have a word length which depends on the number length desired for arithmetic operations. Usually this is 10–13 decimal digits or 36–44 binary digits. With this requirement met, instructions are fitted into the basic structure as appropriate.

The number of digits required to describe the operations and the addresses in a single-address instruction of course depends on the number of operations and addresses involved. In early single-address computers two instructions were placed in one 36-bit word, 18 bits being sufficient to describe, say, 128 operations (7 binary digits) and 2048 addresses (11 binary digits). When greater instruction repertoires and greater storage capacities become important, only one instruction per word could be provided for in a 36-bit word. At the same time the need grew for other binary digits in the word for address modification, breakpoint definition, etc. Thus, the single-address instruction “grew into” the 36-bit word.

Greater instruction flexibility and greater storage systems are placing more pressure on increasing the word length of computers now under design. These factors have now replaced the considerations of number

size in arithmetic operations in providing the principal criterion for word length.

Various instruction system formats are as follows:

1. *The Princeton design* (Ref. 19) places two instructions in one word, leaving the number as the standard unit of information, and enumerating instructions by the word address plus a separate designator for right- or left-hand instructional position. Such a scheme leads to unusual conventions to be used with instructions, such as requirements that changes of control must be made to left-hand instructions only, etc.

2. *The EDSAC design* (Ref. 108), later used on the IBM 701 (Refs. 71 and 87), treats the shorter single-address instruction as the basic element with standard addresses, and may provide for two types of numbers, short and long, the first the length of the instruction, the second twice as long. Here conventions are again necessary to meet the physical nature of the structure; for example, long numbers must always have *even* addresses. Type 701 machines using larger storage units must use an even more complex address-instruction-word numbering relationship.

3. *The Whirlwind I design* (Ref. 57) arbitrarily made the instruction and number length the same, 16 binary (approximately 4 to 5 decimal) digits. For useful computation it is necessary to combine two or more words through programmed methods to provide a satisfactory number length. Only high speed, however, allows such programmed operation without a disastrously low final speed.

4. *Later designs*, such as the IBM 704, 709, STRETCH, UNIVAC-LARC, and Datatron 205-220 machines, have kept the single-address structure in each instruction word, but have used the large number of digits available to allow for larger amounts of storage to be addressed, many index registers, access to more than one accumulator, and other special control features.

**Use of Instructions.** Many problems solved on scientific calculators make little use of other than the arithmetic operations.

*Examples.* (1) In a study of three typical small scientific problems performed on the MANIAC (a Princeton type computer) at the Los Alamos Scientific Laboratory of the Atomic Energy Commission, Herbst, Metropolis, and Wells (Ref. 112) noted that no input-output transfer to magnetic drums or magnetic tapes occurred. In a hydrodynamics problem (the numerical solution of a multidimensional partial differential equation), almost 65% of the total computing time was devoted to multiplication and division.

(2) In business data processing programs, about 60% of the time is spent in data manipulation not involving arithmetic.

Small scientific problems, termed *process limited*, or *computer limited*, are limited by calculating speed and the effectiveness of arithmetic coding. As the size of the scientific problem increases relative to available storage, the storage hierarchy problem enters in, and the quantity of input-output instructions goes up. Such scientific problems like the business data processing problems, called *tape limited*, or *input-output limited*, are limited by the efficiency of the input-output devices and the coding used in that area.

## 6. INSTRUCTION LOGIC OF COMMON COMPUTERS

This section describes the instruction logic of some of the common computers. The set of descriptions includes scientific and business machines of both large and small scale. Table 7 shows some of the features of common computers. The instruction logic is given for the following computers:

<i>Computer</i>	<i>Page</i>
IBM 704	2-63
Univac Scientific 1103A	2-77
Univac II	2-83
IBM 650	2-93
Datatron 205	2-98
Bendix G-15D	See Intercom example, Interpreters, Sect. 14
Royal-McBee LGP-30	2-109
Soviet Strela	2-111
MIDAC	2-115
EASIAC	2-122

MIDAC is included as an illustration of a three-address instruction machine. EASIAC is an interpretive routine for MIDAC and is included as an example of that form of programming.

### **Instruction Logic of the IBM 704**

The IBM 704 is a binary digital computer with fixed or floating point arithmetic and either a (27, 7, 0) or (35, 0, 0) "digital" (absolute value less than one) number system. Numbers are stored in absolute value form. There are available from 4096 to 32,768 locations of high-speed (magnetic core) as well as up to 16,384 words of nonaddressable magnetic drum storage. Up to ten magnetic tape units with up to 900,000 machine words (each of 36 bits) are available as secondary storage, which can be transferred in or out, after the tape is in motion, at the rate of 2500 words per second.

Input-output is via 80-column punched cards either direct or through peripheral equipment to and from magnetic tape. A high-speed printer and a cathode ray tube display are also available for output.

TABLE 7. FEATURES OF COMMON COMPUTERS

Computer	Number System, Operation	Word Size	Number Triad ( $m, n, p$ ) <sup>a</sup>	Instructions <sup>b</sup>	Operating Speed <sup>c</sup>	Maximum Storage
IBM 704	Binary, parallel	36 bits	(27, 7, 0) or (35, 0, 0)	Single address two per word; 86 operations, index register	+ : 84 $\mu$ sec × : 200 $\mu$ sec	Cores, 32,768 words, drums, 16,384 words, tapes, 10
IBM 705	Alphanumeric, serial	Variable	( $m, 0, 0$ ) $m$ variable	Single address, 35 operations	+ : 200 $\mu$ sec × : 2480 $\mu$ sec (10 dig. dec. nos.)	Cores, 20,000 characters, tapes, 10
Univac 1103A	Binary, parallel	36 bits	(35, 0, 0) or (27, 8, 0)	Two address, 50 operations	+ : 48 $\mu$ sec × : 266 $\mu$ sec	Cores, 4096 words, tapes, 10
Univac II	Alphanumeric, serial	12 decimal digits	(12, 0, 0)	Single address, two per word; 63 operations	+ : 400 $\mu$ sec × : 2100 $\mu$ sec	Cores, 10,000 words, tapes, 16
IBM 650	Decimal, serial	10 decimal digits	(10, 0, 0)	One-plus-one address, 70 operations, 1 to 3 index registers	2.4 msec average access time	Drum, 2000 words, tapes, 6
Datatron 205	Decimal, serial	10 decimal digits	(10, 0, 0)	Single address, 55 operations, index register	8.5 msec or 0.85 msec average access time	Drum, 4000 words, tapes, 10
Bendix G-15	Binary, serial	29 bits	(28, 0, 0)	Two-plus-one address	14.5 msec or 0.54 msec average access time	Drum, 2176 words, tapes, 4
Royal McBee LGP-30	Binary, serial	32 bits	(30, 0, 0)	Single address	9 msec average access time	Drum, 4096 words
Soviet Strela	Binary, parallel	43 bits	(35, 6, 0)	Three address	220-500 $\mu$ sec	Cores, 2048 words Drums, 5120 words
MIDAC	Binary, serial	45 binary digits	(44, 0, 0)	Three address, 19 operations, index registers	+ : 192 $\mu$ sec × : 2304 $\mu$ sec	Acoustic, 512 words Drum, 24,576 words

<sup>a</sup> Number triad ( $m, n, p$ ):

$m$  = significant digits representing the fractional part of a number,

$n$  = digits in the exponent of the number ( $n = 0$  is a fixed point computer,  $n > 0$  is a floating point machine),

$p$  = digits to the left of the radix point ( $p = 0$  means all numbers have absolute value less than 1; such numbers are called "digital" numbers).

<sup>b</sup> One instruction per word unless noted.

<sup>c</sup> For computers with circulating storage only the average access time is given.

**Instruction Description.** This machine has three index registers and has only one single-address instruction stored per word. There are two types of instructions, A and B. Type A instructions contain a fifteen-bit address portion and a fifteen-bit "decrement" portion (the latter for index use in certain cases) as well as a three-bit operation portion and a three-bit "tag." Type B instructions contain an eleven-bit operation portion, a three-bit tag, and a fifteen-bit address portion.

Instructions are either indexable or nonindexable. Presence of one or more of the three tag bits automatically adds in the contents of the corresponding index registers into the instruction address. In certain type B instructions this process will change the operation code as well.

There are two arithmetic elements, an accumulator (AC) of 37 bits plus sign and a multiplier-quotient register (MQ) of 35 bits plus sign. The two extra AC bits are for special overflow information. Instruction sequencing is of the standard single-address type. The IBM 704 has a special operational mode, the "Trapping Mode," into which it may be transferred by a special instruction. This allows machine tracing of programs by observing the flow of control automatically without detailed interpretation.

Type A instructions are written in the form:

**TNX 02301 B 03449**

where the three-letter code indicates the operation, the first five-digit decimal code the decrement to be used with the index registers, the next letter, one or more of the three index registers (A, B, or C), and the last five-digit decimal code the address.

Type B instructions are written usually as

**SLW C 2644**

where the decrement field is now omitted. In some cases, type B instructions do not have tag locations.

The following list of instructions for the IBM 704 is taken from a "Manual of Operation" (Ref. 54). Notation is standard, C(Y) indicating contents of location Y, and subscripts corresponding to specific digits. S, P, Q are the sign and two overflow digits of AC, respectively.

All instructions are indexable except those with an X in the operation code, designating an index register. The four digits alongside each mnemonic operation indicate the octal operation equivalent. In cases where the operation code extends into the address, there may be more than four digits. The first integer gives the number of cycles (each of 12 microseconds) for performance of the instruction. The detailed handling of overflow and floating point zero is not included in this account, nor is a complete account of the problems of normalization.

**Fixed Point Arithmetic Operations.**

**Clear and Add. 2 CLA Y +0500.** The C(Y) replace the C(AC)<sub>S,1-35</sub>. Positions Q and P of the AC are cleared. The C(Y) are unchanged.

**Add. 2 ADD Y +0400.** This instruction algebraically adds the C(Y) to the C(AC) and replaces the C(AC) with this sum. The C(Y) are unchanged. AC overflow is possible.

**Add Magnitude. 2 ADM Y +0401.** This instruction algebraically adds the magnitude of the C(Y) to the C(AC) and replaces the C(AC) with this sum. The C(Y) are unchanged. AC overflow is possible.

**Clear and Subtract. 2 CLS Y +0502.** The negative of the C(Y) replaces the C(AC)<sub>S,1-35</sub>. Positions Q and P of the AC are cleared. The C(Y) are unchanged.

**Subtract. 2 SUB Y +0402.** This instruction algebraically subtracts the C(Y) from the C(AC) and replaces the C(AC) with this difference. The C(Y) are unchanged. AC overflow is possible.

**Subtract Magnitude. 2 SBM Y -0400.** This instruction algebraically subtracts the magnitude of the C(Y) from the C(AC) and replaces the C(AC) with this difference. The C(Y) are unchanged. AC overflow is possible.

**Multiply. 20 MPY Y +0200.** This instruction multiplies the C(Y) by the C(MQ). The 35 most significant bits of the 70-bit product replace the C(AC)<sub>1-35</sub> and the 35 least significant bits replace the C(MQ)<sub>1-35</sub>. The Q and P bits are cleared. The sign of the AC is the algebraic sign of the product. The sign of the MQ agrees with the sign of the AC.

**Multiply and Round. 20 MPR Y -0200.** This instruction executes a multiply followed by a round. (The latter operation is defined below.) AC overflow is not possible.

**Round. 2 RND +0760 . . . 010.** If position 1 of the MQ contains a 1, the magnitude of the C(AC) is increased by a 1 in position 35. If position 1 of the MQ contains a zero, the C(AC) remain unchanged. In either case, the C(MQ) are unchanged. AC overflow is possible.

**Divide or Halt. 20 DVH Y +0220.** This instruction treats the C(AC)<sub>S,Q,P,1-35</sub> and the C(MQ)<sub>1-35</sub> as a 72-bit dividend plus sign, and the C(Y) as the divisor. If  $|C(Y)| > |C(AC)|$ , division takes place, a 35-bit quotient plus sign replaces the C(MQ) and the remainder replaces the C(AC)<sub>S,1-35</sub>. The sign of the remainder always agrees with the sign of the dividend.

If  $|C(Y)| \leq |C(AC)|$ , division does not take place and the calculator stops with the divide-check indicator and light on. Consequently, if po-

sition Q or P of the AC contains a 1, division does not take place since  $|C(Y)| < |C(AC)|$ . The dividend remains unchanged in the AC.

**Divide or Proceed. 20 DVP Y +0221.** This instruction executes a division (as defined above) if  $|C(Y)| > |C(AC)|$ . If  $|C(Y)| \leq |C(AC)|$ , division does not take place, the divide-check indicator and light are turned on, and the calculator proceeds to the next instruction. The dividend remains unchanged in the AC.

**Load MQ. 2 LDQ Y +0560.** The C(Y) replace the C(MQ). The C(Y) are unchanged.

**Store MQ. 2 STQ Y -0600.** The C(MQ) replace the C(Y). The C(MQ) are unchanged.

**Store Left-Half MQ. 2 SLQ Y -0620.** The  $C(MQ)_{s,1-17}$  replace the  $C(Y)_{s,1-17}$ . The  $C(Y)_{18-35}$  and the C(MQ) are unchanged.

**Store. 2 STO Y +0601.** The  $C(AC)_{s,1-35}$  replace the  $C(Y)_{s,1-35}$ . The C(AC) are unchanged.

**Store Zero. 2 STZ Y +0600.** The C(Y) are replaced by zeros and the sign of Y is made plus.

**Store Prefix. 2 STP Y +0630.** The  $C(AC)_{p,1,2}$  replace the  $C(Y)_{s,1,2}$ . The  $C(Y)_{3-35}$  and the C(AC) are unchanged.

**Store Decrement. 2 STD Y +0622.** The  $C(AC)_{3-17}$  replace the  $C(Y)_{3-17}$ . The  $C(Y)_{s,1,2,18-35}$  and the C(AC) are unchanged.

**Store Address. 2 STA Y +0621.** The  $C(AC)_{21-35}$  replace the  $C(Y)_{21-35}$ . The  $C(Y)_{s,1-20}$  and the C(AC) are unchanged.

**Clear Magnitude. 2 CLM +0760...000.** The  $C(AC)_{q,p,1-35}$  are cleared. The AC sign is unchanged.

**Change Sign. 2 CHS +0760...002.** If the AC sign bit is negative, it is made positive, and vice versa.

**Set Sign Plus. 2 SSP +0760...003.** A positive sign replaces the  $C(AC)_s$ .

**Set Sign Minus. 2 SSM -0760...003.** A negative sign replaces the  $C(AC)_s$ .

### Logical Operations.

**Clear and Add Logical Word. 2 CAL Y -0500.** This instruction replaces the  $C(AC)_{p,1-35}$  with the C(Y). Thus the sign of the C(Y) appears in position P of the AC, and the S and Q bits are cleared. The C(Y) are unchanged.

**Add and Carry Logical Word. 2 ACL Y +0361.** This instruction adds the  $C(Y)_{s,1-35}$  to the  $C(AC)_{p,1-35}$ , respectively, and replaces the  $C(AC)_{p,1-35}$  with this sum (position S of register Y is treated as a numerical bit, and the sign of the AC is ignored). A carry out of the P

bit adds into position 35 of the AC, but does not add into Q. Q is not changed. The C(Y) are unchanged. No overflow is possible.

**Store Logical Word. 2 SLW Y +0602.** The C(AC)<sub>P,1-35</sub> replace the C(Y)<sub>S,1-35</sub>. The C(AC) are unchanged.

**AND to Accumulator. 3 ANA Y -0320.** Each bit of the C(AC)<sub>P,1-35</sub> is matched with the corresponding bit of the C(Y)<sub>S,1-35</sub>, the C(AC)<sub>P</sub> being matched with the C(Y)<sub>S</sub>. When the corresponding bit of both the AC and location Y is a one, a one replaces the contents of that position in the AC. When the corresponding bit of either the AC or location Y is a zero, a zero replaces the contents of that position in the AC. The C(AC)<sub>S,Q</sub> are cleared. The C(Y) are unchanged.

**AND to Storage. 4 ANS Y +0320.** Each bit of the C(AC)<sub>P,1-35</sub> is matched with the corresponding bit of the C(Y)<sub>S,1-35</sub>, the C(AC)<sub>P</sub> being matched with the C(Y)<sub>S</sub>. When the corresponding bit of both the AC and location Y is a one, a one replaces the contents of that position in location Y. When the corresponding bit of either the AC or location Y is a zero, a zero replaces the contents of that position in location Y. The C(AC) are unchanged.

**OR to Accumulator. 2 ORA Y -0501.** Each bit of the C(AC)<sub>P,1-35</sub> is matched with the corresponding bit of the C(Y)<sub>S,1-35</sub>, the C(AC)<sub>P</sub> being matched with the C(Y)<sub>S</sub>. When the corresponding bit of either the AC or location Y is a one, a one replaces the contents of that position in the AC. When the corresponding bit of both the AC and location Y is a zero, a zero replaces the contents of that position in the AC. The C(Y) and the C(AC)<sub>S,Q</sub> are unchanged.

**OR to Storage. 2 ORS Y -0602.** Each bit of the C(AC)<sub>P,1-35</sub> is matched with the corresponding bit of the C(Y)<sub>S,1-35</sub>, the C(AC)<sub>P</sub> being matched with the C(Y)<sub>S</sub>. When the corresponding bit of either the AC or location Y is a one, a one replaces the contents of that position in location Y. When the corresponding bit of both the AC and location Y is a zero, a zero replaces the contents of that position in location Y. The C(AC) are unchanged.

**Complement Magnitude. 2 COM +0760 . . . 006.** All ones are replaced by zeros and all zeros are replaced by ones in the C(AC)<sub>Q,P,1-35</sub>. The AC sign is unchanged.

**Shift Operations.** Shift instructions are used to move the bits in a word to the right or left of their original positions in the AC or MQ register or both. With the exception of the RQL instruction, zeros are automatically introduced in the vacated positions of a register. Thus, a shift larger than the bit capacities of the registers involved in the shifting will have no significance after the capacities of the registers are exceeded. When an instruction is interpreted as a shift instruction, the extent of the shift is

determined by the least significant eight bits of the address of the instruction. Since the maximum possible shift is 255, a number larger than 255 in the address part of a shift instruction is interpreted modulo 256.

**Accumulator Left Shift. 2-1 ALS Y +0767.** The  $C(AC)_{Q,P,1-35}$  are shifted left Y modulo 256 places. If a nonzero bit is shifted into or through position P, the AC overflow indicator and light are turned on. Bits shifted past position Q are lost. Positions made vacant are filled in with zeros.

**Accumulator Right Shift. 2-1 ARS Y +0771.** The  $C(AC)_{Q,P,1-35}$  are shifted right Y modulo 256 places. Bits shifted past position 35 of the AC are lost. Positions made vacant are filled in with zeros.

**Long Left Shift. 2-1 LLS Y +0763.** The  $C(AC)_{Q,P,1-35}$  and the  $C(MQ)_{1-35}$  are shifted left Y modulo 256 places. Bits enter position 35 of the AC from position 1 of the MQ. If a nonzero bit is shifted into or through position P, the AC overflow indicator and light are turned on. Bits shifted past position Q are lost. Positions made vacant are filled in with zeros. The sign of the AC is replaced by the same sign as that of the MQ.

**Long Right Shift. 2-1 LRS Y +0765.** The  $C(AC)_{Q,P,1-35}$  and the  $C(MQ)_{1-35}$  are shifted right Y modulo 256 places. Bits enter position 1 of the MQ from position 35 of the AC. Bits shifted past position 35 of the MQ are lost. Positions made vacant are filled in with zeros. The sign of the MQ is replaced by the same sign as that of the AC.

**Logical Left. 2-1 LGL Y -0763.** The  $C(AC)_{Q,P,1-35}$  and the  $C(MQ)_{S,1-35}$  are shifted left Y modulo 256 places. Bits enter position S of the MQ from position 1 of the MQ, and enter position 35 of the AC from position S of the MQ. If a nonzero bit is shifted into or through position P of the AC, the AC overflow indicator and light are turned on. Bits shifted past position Q are lost. Positions made vacant are filled in with zeros. The sign of the AC is unchanged.

**Rotate MQ Left. 2-1 RQL Y -0773.** The  $C(MQ)_{S,1-35}$  are rotated left Y modulo 256 places. The bits rotate from position 1 to position S of the MQ, and from position S to position 35 of the MQ.

### Floating Point Arithmetic Operations.

**Floating ADD. 7-11 FAD Y +0300.** The  $C(Y)$  are algebraically added to the  $C(AC)$ , and this sum replaces the  $C(AC)$  and the  $C(MQ)$ . The  $C(Y)$  are unchanged. The fractional part of the product is normalized to between  $\frac{1}{2}$  and 1 in absolute value.

During execution of a floating point addition, the AC or MQ overflow indicator and the corresponding light on the operator's console are turned

on by too large a characteristic or too small a characteristic in the AC or the MQ, respectively.

**Unnormalized Floating Add. 6-11 UFA Y -0300.** Same as floating add except the result is not normalized.

**Floating Subtract. 7-11 FSB Y +0302.** Same as floating add except that the negative of the C(Y) is added.

**Unnormalized Floating Subtract. 6-11 UFS Y -0302.** Same as floating subtract except that the result is unnormalized.

**Floating Multiply. 17 FMP Y +0260.** The C(Y) are multiplied by the C(MQ). The most significant part of the product appears in the AC and the least significant part appears in the MQ.

The product of two floating point numbers is in normalized form if the multiplier and multiplicand are in this form. If either the multiplier or multiplicand is not in normalized form, the product is in normalized form only if a shift of one place is sufficient to normalize it.

During execution of floating point multiplication, too large or too small a characteristic in the AC or the MQ, respectively, turns on the AC or the MQ overflow indicator and the corresponding light on the operator's console.

**Unnormalized Floating Multiply. 17 UFM Y -0260.** This operation is the same as floating multiply except that no shifting is included.

**Floating Divide or Halt. 18-IV FDH Y +0240.** The C(AC) are divided by the C(Y), the quotient appears in the MQ and the remainder appears in the AC. The MQ is cleared before actual division takes place.

If positions Q or P of the AC are not zero, division may take place and either or both of the AC and/or MQ overflow indicators may be turned on. When division by zero is attempted, the divide-check indicator and light are turned on and the calculator stops, and the dividend is left unchanged in the AC. The quotient is in normalized form if both divisor and dividend are in that form. If divisor or dividend or both are not in normalized form, the quotient is in normalized form if

$$2|C(Y)_{9-35}| > |C(AC)_{9-35}| \geq \frac{1}{2}|C(Y)_{9-35}|$$

During execution of a floating point division, the AC or MQ overflow indicator and the corresponding light on the operator's console are turned on for too large or too small a characteristic in the AC or MQ, respectively.

**Floating Divide or Proceed. 18-IV FDP Y +0241.** This operation is the same as floating divide or halt except for division by zero.

When division by zero is attempted, the divide-check indicator and light are turned on, division does not take place and the calculator proceeds to the next instruction. If the magnitude of the fraction in the AC is greater

than (or equal to) twice the magnitude of the fraction in the SR, the divide-check indicator and light are turned on, division does not take place and the calculator proceeds to the next instruction. The dividend in the AC is unchanged.

### Control Operations.

**No Operation. 2 NOP +0761.** The calculator takes the next instruction in sequence.

**Halt and Proceed. 2 HPR +0420.** This instruction causes the calculator to stop. If the start key on the operator's console is depressed, the calculator proceeds to the next instruction in sequence.

**Enter Trapping Mode. 2 ETM +0760 . . . 007.** This instruction turns on the trapping indicator and also the trap light on the operator's console. The calculator operates in the trapping mode until a leave trapping mode operation is executed or until either the clear or reset key is pressed on the console.

**Leave Trapping Mode. 2 LTM -0760 . . . 007.** This instruction turns off the trapping indicator and the trap light on the operator's console. The calculator will not operate in the trapping mode until another enter trapping mode operation is executed.

*Note.* When the calculator is operating in the trapping mode, the location of every transfer instruction (except trap transfer instructions) replaces the address part of location 0000, whether or not the conditions for transfer of control are met. If the conditions are met, the calculator takes the next instruction from location 0001 and proceeds from that point. The location of each transfer instruction replaces the address part of location 0000.

**Halt and Transfer. 2 HTR Y +0000.** This instruction stops the calculator. When the start key on the operator's console is depressed, the calculator starts again, taking the next instruction from location Y and proceeding from there.

**Transfer. 2 TRA Y +0020.** This instruction causes the calculator to take its next instruction from location Y, and to proceed from there.

**Transfer on Zero. 2 TZE Y +0100.** If the  $C(AC)_{Q,P,1-35}$  are zero, the calculator takes its next instruction from location Y and proceeds from there. If they are not zero, the calculator proceeds to the next instruction in sequence.

**Transfer on No Zero. 2 TNZ Y -0100.** If the  $C(AC)_{Q,P,1-35}$  are not zero, the calculator takes its next instruction from location Y and proceeds from there. If they are zero, the calculator proceeds to the next instruction in sequence.

**Transfer on Plus. 2 TPL Y +0120.** If the sign bit of the AC is positive, the calculator takes the next instruction from location Y and proceeds from there. If the sign bit of the AC is negative, the calculator proceeds to the next instruction in sequence.

**Transfer on Minus. 2 TMI Y -0120.** If the sign bit of the AC is negative, the calculator takes the next instruction from location Y and proceeds from there. If the sign bit of the AC is positive, the calculator proceeds to the next instruction in sequence.

**Transfer on Overflow. 2 TOV Y +0140.** If the AC overflow indicator and light are on as the result of a previous operation, the indicator and light are turned off and the calculator takes the next instruction from location Y and proceeds from there. If the indicator and light are off, the calculator proceeds to the next instruction in sequence.

**Transfer on No Overflow. 2 TNO Y -0140.** If the AC overflow indicator and light are off, the calculator takes the next instruction from location Y and proceeds from there. If the indicator and light are on, the calculator proceeds to the next instruction in sequence after turning off the indicator and light.

**Transfer on MQ Plus. 2 TQP Y +0162.** If the sign bit of the MQ is positive, the calculator takes the next instruction from location Y and proceeds from there. If the sign bit of the MQ is negative, the calculator proceeds to the next instruction in sequence.

**Transfer on MQ Overflow. 2 TQO Y +0161.** If the MQ overflow indicator and light have been turned on by an overflow or underflow in the MQ characteristic during a previous floating point operation, the indicator and light are turned off, the calculator takes the next instruction from location Y and proceeds from there. If the indicator and light are not on, the calculator proceeds to the next instruction in sequence.

**Transfer on Low MQ. 2 TLQ Y +0040.** If the C(MQ) are algebraically less than the C(AC), the calculator takes the next instruction from location Y and proceeds from there. If the C(MQ) are algebraically greater than or equal to the C(AC), the calculator proceeds to the next instruction in sequence.

**Transfer and Set Index. 2 TSX Y +0074.** Not indexable. This instruction places the 2's complement of the location of this instruction in the specified index register. The calculator takes the next instruction from location Y and proceeds from there.

The 2's complement is used in this instruction because indexing is a subtractive process on the IBM 704 and subtracting the 2's complement of a number is equivalent to adding the number.

**Transfer with Index Incremented. 2 TXI Y +1000.** Not indexable. Contains a decrement part. This instruction adds the decre-

ment to the number in the specified index register and replaces the number in the index register with this sum. The calculator takes the next instruction from location Y and proceeds from there.

**Transfer on Index High. 2 TXH Y +3000.** Not indexable. Contains a decrement part. If the number in the specified index register is greater than the decrement, the calculator takes the next instruction from location Y and proceeds from there.

If the number in the specified index register is less than or equal to the decrement, the calculator proceeds to the next instruction in sequence.

**Transfer on Index Low or Equal. 2 TXL Y -3000.** Not indexable. Contains a decrement part. If the number in the specified index register is less than or equal to the decrement, the calculator takes the next instruction from location Y and proceeds from there.

If the number in the specified index register is greater than the decrement, the calculator proceeds to the next instruction in sequence.

**Transfer on Index. 2 TIX Y +2000.** Not indexable. Contains a decrement part. If the number in the specified index register is greater than the decrement, the number in the index register is reduced by the amount of the decrement, and the calculator takes the next instruction from location Y and proceeds from there.

If the number in the specified index register is equal to or less than the decrement, the number in the index register is unchanged, and the calculator proceeds to the next instruction in sequence.

**Transfer on No Index. 2 TNX Y -2000.** Not indexable. Contains a decrement part. If the number in the specified index register is equal to or less than the decrement, the number in the index register is unchanged, the calculator takes the next instruction from location Y and proceeds from there.

If the number in the specified index register is greater than the decrement, the number in the index register is reduced by the amount of the decrement and the calculator proceeds to the next instruction in sequence.

**Trap Transfer. 2 TTR Y +0021.** This instruction causes the calculator to take its next instruction from location Y and to proceed from there *whether in the trapping mode or not*. This makes it possible to have an ordinary transfer even when in the trapping mode.

**P Bit Test. 2 PBT -0760...001.** If the  $C(AC)_P$  is a one, the calculator skips the next instruction and proceeds from there. If position P contains a zero, the calculator takes the next instruction in sequence.

**Low Order Bit Test. 2 LBT +0760...001.** If the  $C(AC)_{35}$  is a one, the calculator skips the next instruction and proceeds from there.

If position 35 contains a zero, the calculator takes the next instruction in sequence.

**Divide Check Test. 2 DCT +0760 ··· 012.** If the divide-check indicator and light are on, the indicator and light are turned off, and the calculator takes the next instruction in sequence. If the indicator and light are off, the calculator skips the next instruction and proceeds from there.

**Redundancy Tape Test. 2 RTT -0760 ··· 012.** If the tape-check indicator and light are on, the indicator and light are turned off and the calculator takes the next instruction in sequence. If the indicator and light are off, the calculator skips the next instruction and proceeds from there.

**Compare Accumulator with Storage. 3 CAS Y +0340.** If the  $C(Y)$  are algebraically less than the  $C(AC)$ , the calculator takes the next instruction in sequence. If the  $C(Y)$  are algebraically equal to the  $C(AC)$ , the calculator skips the next instruction and proceeds from there. If the  $C(Y)$  are algebraically greater than the  $C(AC)$ , the calculator skips the next two instructions and proceeds from there. Two numbers are algebraically equal when the magnitude of the numbers and the sign are both equal. A plus zero is algebraically larger than a minus zero.

### Indexing Operations.

**Load Index from Address. 2 LXA Y +0534.** Not indexable. The address part of the  $C(Y)$  replaces the number in the specified index register. The  $C(Y)$  are unchanged.

**Load Index from Decrement. 2 LXD Y -0534.** Not indexable. The decrement part of the  $C(Y)$  replaces the number in the specified index register. The  $C(Y)$  are unchanged.

**Store Index in Decrement. 2 SXD Y -0634.** Not indexable. The  $C(Y)_{3-17}$  are cleared and the number in the specified index register replaces the decrement part of the  $C(Y)$ . The  $C(Y)_{S,1,2,18-35}$  are unchanged. The contents of the index register are unchanged if one index register is specified. If a multiple tag is specified, the *logical or* of the contents of these index registers will replace the  $C(Y)_{3-17}$  and will also replace the contents of the specified index registers.

**Place Address in Index. 2 PAX +0734.** Not indexable. The address part of the  $C(AC)$  replaces the number in the specified index register. The  $C(AC)$  are unchanged.

**Place Decrement in Index. 2 PDX -0734.** Not indexable. The decrement part of the  $C(AC)$  replaces the number in the specified index register. The  $C(AC)$  are unchanged.

**Place Index in Decrement. 2 PXD -0754.** Not indexable. The  $AC$  is cleared and the number in the specified index register is placed

in the decrement part of the AC. The contents of the index register are unchanged if one index register is specified. If a multiple tag is specified, the *logical or* of the contents of these index registers will replace the  $C(AC)_{3-17}$  and will also replace the contents of the specified index registers.

**Input-Output Operations.** The identifying numbers for the various input-output components appear in the address part of an instruction whenever the programmer wants to operate one of these units. Whether the address part of an instruction refers to a storage location or to one of the components depends on the operation part of the instruction. Some operations make no sense if the address is interpreted as a location in storage; other operations make no sense if the address is interpreted as a component identification. Thus, an address is automatically interpreted by the calculator in the light of what it is asked to do by the operation part of the instruction.

The addresses of the input-output units are given below.

<i>Component</i>	<i>Octal</i>	<i>Decimal</i>
Cathode ray tube	030	024
Tapes		
Binary coded decimal	201-212	129-138
Binary	221-232	145-154
Drum	301-310	193-200
Card reader	321	209
Card punch	341	225
Printer	361	241

**Read Select. 2-111 RDS Y +0762.** This instruction causes the calculator to prepare to read one record of information from the component specified by Y. If Y specifies a tape unit, the MQ is cleared by this instruction.

**Write Select. 2-111 WRS Y +0766.** This instruction causes the calculator to prepare to write one record of information on the component specified by Y. WRS 333<sub>8</sub> is used to *delay* the execution of any instruction until the MQ is available for computing after reading information from a tape.

**Backspace Tape. 2-111 BST Y +0764.** This instruction causes the tape designated by Y to space one record in a backward direction. If the tape designated by Y is positioned at the load point, the BST Y instruction is interpreted as no operation.

**Write End of File. 2-111 WEF Y +0770.** This instruction causes the tape unit designated by Y to leave an end-of-file space, an end-of-file mark, and a redundancy character on its tape.

**Rewind. 40ms-111 REW Y +0772.** This instruction causes the tape unit designated by Y to rewind its tape to the load point.

**End of Tape Test. 2 ETT -760...011.** This instruction must be given while the tape unit is selected (i.e., after a WRS or WEF instruction and before the tape disconnects; no more than 744 microseconds after the last CPY if WRS instruction; and any time up to 40 milliseconds, if WEF). Failure to program this instruction may cause the tape to be pulled from its reel. If the tape indicator and the tape indicator light are off, the calculator skips the instruction immediately following the ETT and proceeds from that point. If the tape indicator and the tape indicator light are on, they will be turned off and the calculator will take the next instruction in sequence (no skip).

If tape instructions are given to a tape while the tape indicator is on, they will operate normally.

**Locate Drum Address. 2 LDA Y +0460.** This instruction follows a read select or write select instruction referring to a drum unit, and the address part of the C(Y) specifies the first location of the record to be read from or written on the drum. Not giving this instruction is equivalent to giving the instruction with the address part of the C(Y) equal to zero.

**Copy and Skip. -111 CPY Y +0700.** This instruction is used after an RDS, WRS, or another CPY instruction to transfer a word of information between location Y in storage and an input-output component specified by the address part of the preceding RDS or WRS instruction. When this instruction is executed, the 36-bit word is formed in the MQ and then is transmitted to storage or to the component. If the CPY instructions are not given within specific time ranges (found in the descriptions of these components), the calculator stops, and a read-write check light on the operator's console is turned on.

If an additional CPY instruction is given after the last word of a unit record has been copied from a card or a record of tape, the CPY is not executed, and the calculator skips the two instructions immediately following the CPY and proceeds from there. If an additional RDS instruction is given for which there is no corresponding record, the calculator sets up an end-of-file condition. The first CPY instruction given after this RDS is not executed; instead, the calculator skips the instruction immediately following the CPY and proceeds from there.

**Plus Sense. 2 PSE Y +0760.** This instruction provides a means of testing the status of sense switches (and of turning on or off the sense lights on the operator's console), thus providing the programmer with flexible means of altering the sequence of instructions being executed. This instruction also permits the transmission of an impulse to or from the exit or entry hubs on the printer or card punch.

The address part of this instruction determines whether a light, switch, printer, or card punch is being sensed, and it further determines which light, switch, or hub is being sensed.

**Minus Sense. 2 MSE Y -0760.** This instruction provides a means of testing the status of sense lights on the operator's console. The addresses of the four sense lights are 141-144. If the corresponding sense light is on, the light is turned off, the calculator skips the next instruction and proceeds from there. If the light is off, the calculator takes the next instruction in sequence.

### Univac Scientific (1103A) Instruction Logic

The Univac Scientific computer is a (35, 0, 0) binary machine, with option of (27, 8, 0). The arithmetic unit contains two 36-bit X (exchange) and Q (quotient) registers and one 72-bit A register (accumulator). Negative numbers are represented in one's complement notation.

Input-output is via high-speed paper tape reader and punch, direct card reader and punch, and Uniservo magnetic tape units, which may be connected to peripheral punched card readers and punches and a high-speed printer. In addition, information may be recorded on magnetic tape directly from keyboards by the use of Unitypers. Communication with external equipment is via an 8-bit (IOA) register and a 36-bit (IOB) register. Information sent to these registers controls magnetic tapes as well as other input-output equipment. The program address counter (PAK) contains the present instruction address. Storage is in up to 12,288 locations of magnetic core storage, along with a directly addressable drum of 16,384 locations. Instructions are of the two-address form, with six bits for the operation code and two fifteen-bit addresses (u and v).

The following information is taken from a Univac Scientific Manual (Ref. 103).

### Definitions and Conventions.

#### *Instruction Word*

<b>oc</b> 6 bits	<b>u</b> 15 bits	<b>v</b> 15 bits	
135 ...	129 ...	114 ...	10

- oc** Operation code.
- u** First execution address.
- v** Second execution address.

For some of the instructions, the form jn or jk replaces the u address; for others the form k replaces the v address.

- j** One-digit octal number modifying the instruction.
- n** Four-digit octal number designating number of times instruction is to be performed.

**k** Seven-digit binary number designating the number of places the word is to be shifted to the left.

*Address Allocations (Octal)*

<b>MC</b>	{	<b>00000-07777</b>	4096,
		<b>00000-17777</b>	8192, or
		<b>00000-27777</b>	12,288 36-bit words.
<b>Q</b>		<b>31000-31777</b>	1 36-bit word.
<b>A</b>		<b>32000-37777</b>	1 72-bit word.
<b>MD</b>		<b>40000-77777</b>	16,384 36-bit words.

*Fixed Addresses*

<b>F<sub>1</sub></b>	<b>00000</b> or <b>40001</b>
<b>F<sub>2</sub></b>	<b>00001</b>
<b>F<sub>3</sub></b>	<b>00002</b>
<b>F<sub>4</sub></b>	<b>00003</b>

*Arithmetic Section Registers*

<b>A</b>	72-bit accumulator with shifting properties.
<b>A<sub>R</sub></b>	Right-hand 36 bits of A.
<b>A<sub>L</sub></b>	Left-hand 36 bits of A.
<b>Q</b>	36-bit register with shifting properties.
<b>X</b>	36-bit exchange register.

*Note.* Parentheses denote *contents of*. For example, (A) means contents of A (72-bit word in A); (Q) means contents of Q (36-bit word in Q).

*Input-Output Registers*

<b>IOA</b>	8-bit in-out register.
<b>IOB</b>	36-bit in-out register.
<b>TWR</b>	6-bit typewriter register.
<b>HPR</b>	7-bit high-speed punch register.

*Word Extension*

**D(u)** 72-bit word whose right-hand 36 bits are the word at address u, and whose left-hand 36 bits are the same as the leftmost bit of the word at u.

**S(u)** 72-bit word whose right-hand 36 bits are the word at address u, and whose left-hand 36 bits are zero.

**D(Q)** 72-bit word—right-hand 36 bits are in register Q, left-hand 36 bits are same as leftmost bit in register Q.

**S(Q)** same as D(Q) except left 36 bits are zero.

**D(A<sub>R</sub>), S(A<sub>R</sub>)** are similarly defined.

**L(Q)(u)** 72-bit word—left-hand 36 bits are zero, right-hand 36 bits are the bit-by-bit product of corresponding bits of (Q) and word at address u.

$L(Q')(v)$  72-bit word—left-hand 36 bits are zero, right-hand 36 bits are the bit-by-bit product of corresponding bits of the complement of  $(Q)$  and word at address  $v$ .

### Transmit Instructions.

**11\* Transmit Positive TPuv†:** Replace  $(v)$  with  $(u)$ .

**13 Transmit Negative TNuv:** Replace  $(v)$  with the complement of  $(u)$ .

**12 Transmit Magnitude TMuv:** Replace  $(v)$  with the absolute magnitude of  $(u)$ .

**15 Transmit U-address TUuv:** Replace the 15 bits of  $(v)$  designated by  $v_{15}$  through  $v_{29}$ , with the corresponding bits of  $(u)$ , leaving the remaining 21 bits of  $(v)$  undisturbed.

**16 Transmit V-address TVuv:** Replace the right-hand 15 bits of  $(v)$  designated by  $v_0$  through  $v_{14}$ , with the corresponding bits of  $(u)$ , leaving the remaining 21 bits of  $(v)$  undisturbed.

**35 Add and Transmit ATuv:** Add  $D(u)$  to  $(A)$ . Then replace  $(v)$  with  $(A_R)$ .

**36 Subtract and Transmit STuv:** Subtract  $D(u)$  from  $(A)$ . Then replace  $(v)$  with  $(A_R)$ .

**22 Left Transmit LTjkv:** Left circular shift  $(A)$  by  $k$  places. If  $j = 0$  replace  $(v)$  with  $(A_L)$ ; if  $j = 1$  replace  $(v)$  with  $(A_R)$ .

### Q-Controlled Instructions.

**51 Q-controlled Transmit QTuv:** Form in  $A$  the number  $L(Q)(u)$ . Then replace  $(v)$  by  $(A_R)$ .

**52 Q-controlled Add QAuv:** Add to  $(A)$  the number  $L(Q)(u)$ . Then replace  $(v)$  by  $(A_R)$ .

**53 Q-controlled Substitute QSuv:** Form in  $A$  the quantity  $L(Q)(u)$  plus  $L(Q')(v)$ . Then replace  $(v)$  with  $(A_R)$ . The effect is to replace selected bits of  $(v)$  with the corresponding bits of  $(u)$  in those places corresponding to 1's in  $Q$ . The final  $(v)$  is the same as the final  $(A_R)$ .

### Replace Instructions.

**21 Replace Add RAuv:** Form in  $A$  the sum of  $D(u)$  and  $D(v)$ . Then replace  $(u)$  with  $(A_R)$ .

**23 Replace Subtract RSuv:** Form in  $A$  the difference  $D(u)$  minus  $D(v)$ . Then replace  $(u)$  with  $(A_R)$ .

**27 Controlled Complement CCuv:** Replace  $(A_R)$  with  $(u)$  leaving  $(A_L)$  undisturbed. Then complement those bits of  $(A_R)$  that correspond to ones in  $(v)$ . Then replace  $(u)$  with  $(A_R)$ .

\* Octal notation.

† Mnemonic notation.

**54 Left Shift in A LAuk:** Replace (A) with  $D(u)$ . Then left circular shift (A) by  $k$  places. Then replace (u) with  $(A_R)$ . If  $u = A$ , the first step is omitted, so that the initial content of A is shifted.

**55 Left Shift in Q LQuk:** Replace (Q) with (u). Then left circular shift (Q) by  $k$  places. Then replace (u) with (Q).

### Split Instructions.

**31 Split Positive Entry SPuk:** Form  $S(u)$  in A. Then left circular shift (A) by  $k$  places.

**33 Split Negative Entry SNuk:** Form in A the complement of  $S(u)$ . Then left circular shift (A) by  $k$  places.

**32 Split Add SAuk:** Add  $S(u)$  to (A). Then left circular shift (A) by  $k$  places.

**34 Split Subtract SSuk:** Subtract  $S(u)$  from (A). Then left circular shift (A) by  $k$  places.

### Two-Way Conditional Jump Instructions.

**46 Sign Jump SJuv:** If  $A_{71} = 1$ , take (u) as NI. If  $A_{71} = 0$ , take (v) as NI. (NI means next instruction.)

**47 Zero Jump ZJuv:** If (A) is not zero, take (u) as NI. If (A) is zero, take (v) as NI.

**44 Q-Jump QJuv:** If  $Q_{35} = 1$ , take (u) as NI. If  $Q_{35} = 0$ , take (v) as NI. Then, in either case, left circular shift (Q) by one place.

### One-Way Conditional Jump Instructions.

**41 Index Jump IJuv:** Form in A the difference  $D(u)$  minus 1. Then if  $A_{71} = 1$ , continue the present sequence of instructions; if  $A_{71} = 0$ , replace (u) with  $(A_R)$  and take (v) as NI.

**42 Threshold Jump TJuv:** If  $D(u)$  is greater than (A), take (v) as NI; if not, continue the present sequence. In either case, leave (A) in its initial state.

**43 Equality Jump EJuv:** If  $D(u)$  equals (A), take (v) as NI, if not, continue the present sequence. In either case leave (A) in its initial state.

### One-Way Unconditional Jump Instructions.

**45 Manually Selective Jump MJjv:** If the number  $j$  is zero, take (v) as NI. If  $j$  is 1, 2, or 3, and the correspondingly numbered MJ selecting switch is set to "jump," take (v) as NI; if this switch is not set to "jump," continue the present sequence.

**37 Return Jump RJuv:** Let  $y$  represent the address from which CI was obtained. Replace the right-hand 15 bits of (u) with the quantity  $y$  plus 1. Then take (v) as NI.

**14 Interpret IP:** Let  $y$  represent the address from which CI was obtained. Replace the right-hand 15 bits of  $(F_1)$  with the quantity  $y + 1$ . Then take  $(F_2)$  as NI.

### Stop Instructions.

**56 Manually Selective Stop MSjv:** If  $j = 0$ , stop computer operation and provide suitable indication. If  $j = 1, 2$ , or  $3$  and the correspondingly numbered MS selecting switch is set to "stop," stop computer operation and provide suitable indication. Whether or not a stop occurs,  $(v)$  is NI.

**57 Program Stop PS**—Stop computer operations and provide suitable indication.

### External Equipment Instructions.

**17 External Function EF-v:** Select a unit of external equipment and perform the function designated by  $(v)$ .

**76 External Read ERjv:** If  $j = 0$ , replace the right-hand 8 bits of  $(v)$  with  $(IOA)$ ; if  $j = 1$ , replace  $(v)$  with  $(IOB)$ .

**77 External Write EWjv:** If  $j = 0$ , replace  $(IOA)$  with the right-hand 8 bits of  $(v)$ ; if  $j = 1$ , replace  $(IOB)$  with  $(v)$ . Cause the previously selected unit to respond to the information in  $IOA$  or  $IOB$ .

**61 PRint PR-v:** Replace  $(TWR)$  with the right-hand 6 bits of  $(v)$ . Cause the typewriter to print the character corresponding to the 6-bit code.

**63 PUnch PUjv:** Replace  $(HPR)$  with the right-hand 6 bits of  $(v)$ . Cause the punch to respond to  $(HPR)$ . If  $j = 0$ , omit seventh level hole; if  $j = 1$ , include seventh level hole.

### Arithmetic Instructions.

**71 Multiply MPuv:** Form in  $A$  the 72-bit product of  $(u)$  and  $(v)$ , leaving in  $Q$  the multiplier  $(u)$ .

**72 Multiply Add MAuv:** Add to  $(A)$  the 72-bit product of  $(u)$  and  $(v)$ , leaving in  $Q$  the multiplier  $(u)$ .

**73 Divide DVuv:** Divide the 72-bit number  $(A)$  by  $(u)$ , putting the quotient in  $Q$ , and leaving in  $A$  a non-negative remainder  $R$ . Then replace  $(v)$  by  $(Q)$ . The quotient and remainder are defined by:  $(A)_i = (u) \cdot (Q) + R$ , where  $0 \leq R < |(u)|$ . Here  $(A)_i$  denotes the initial contents of  $A$ .

**74 Scale Factor SFuv:** Replace  $(A)$  with  $D(u)$ . Then left circular shift  $(A)$  by 36 places. Then continue to shift  $(A)$  until  $A_{34} \neq A_{35}$ . Then replace the right-hand 15 bits of  $(v)$  with the number of left circular shifts,  $k$ , which would be necessary to return  $(A)$  to its original position. If  $(A)$  is all ones or zeros,  $k = 37$ . If  $u$  is  $A$ ,  $(A)$  is left unchanged in the first step, instead of being replaced by  $D(A_R)$ .

### Sequenced Instructions.

**75 RePeat RPjnw:** This instruction calls for the next instruction, which will be called NIuv, to be executed  $n$  times, its  $u$  and  $v$  addresses being modified or not according to the value of  $j$ . Afterwards the program is continued by the execution of the instruction stored at a fixed address  $F_1$ . The exact steps carried out are:

- (a) Replace the right-hand 15 bits of  $(F_1)$  with the address  $w$ .
- (b) Execute NIuv, the next instruction in the program,  $n$  times.
- (c) If  $j = 0$ , do not change  $u$  and  $v$ .  
     If  $j = 1$ , add one to  $v$  after each execution.  
     If  $j = 2$ , add one to  $u$  after each execution.  
     If  $j = 3$ , add one to  $u$  and  $v$  after each execution.

The modification of the  $u$  address and  $v$  address is done in program control registers. The original form of the instruction in storage is unaltered.

(d) On completing  $n$  executions, take  $(F_1)$ , as the next instruction.  $F_1$  normally contains a manually selective jump whereby the computer is sent to  $w$  for the next instruction after the repeat.

(e) If the repeated instruction is a jump instruction, the occurrence of a jump terminates the repetition. If the instruction is a Threshold Jump or an Equality Jump, and the jump to address  $v$  occurs,  $(Q)$  is replaced by the quantity  $j$ ,  $(n - r)$ , where  $r$  is the number of executions that have taken place.

### Floating Point Instructions.

**64 Add FAuv:** Form in  $Q$  the normalized rounded packed floating point sum  $(u) + (v)$ .

**65 Subtract FSuv:** Form in  $Q$  the normalized rounded packed floating point difference  $(u) - (v)$ .

**66 Multiply FMuv:** Form in  $Q$  the normalized rounded packed floating point product  $(u) \cdot (v)$ .

**67 Divide FDuv:** Form in  $Q$  the normalized rounded packed floating point quotient  $(u) \div (v)$ .

**01 Polynomial Multiply FPuv:** Floating add  $(v)$  to the floating product  $(Q)_i \cdot (u)$ , leaving the packed normalized rounded result in  $Q$ .

**02 Inner Product FIuv:** Floating add to  $(Q)_i$  the floating product  $(u) \cdot (v)$  and store the rounded normalized packed result in  $Q$ . This instruction uses MC location  $F_4 = 00003$  for temporary storage, where  $(F_4)_f = (Q)_i$ . The subscripts  $i$  and  $f$  represent "initial" and "final."

**03 Unpack UPuv:** Unpack  $(u)$ , replacing  $(u)$  with  $(u)_M$  and replacing  $(v)_C$  with  $(u)_C$  or its complement if  $(u)$  is negative. The characteristic portion of  $(u)_f$  contains sign bits. The sign portion and mantissa

portion of  $(v)_f$  are set to zero. *Note.* The subscripts M and C denote the mantissa and characteristic portions.

**04 Normalize Pack NPuv:** Replace  $(u)$  with the normalized rounded packed floating point number obtained from the possibly unnormalized mantissa in  $(u)_i$  and the biased characteristic in  $(v)_e$ . *Note.* It is assumed that  $(u)_i$  has the binary point between  $u_{27}$  and  $u_{26}$ ; that is, that  $(u)_i$  is scaled by  $2^{-27}$ .

**05 Normalize Exit NEj-:** If  $j = 1$  normalize without rounding until a master clear or until the instruction is again executed with  $j = 0$ .

### Univac II Instruction Logic

The Univac I and II are successive models of one of the earliest computers produced successively by the Eckert-Mauchly Corporation and later the Remington-Rand Univac Division of the Sperry-Rand Corporation. The latter machine is compatible with programs written for the former. Univac II is a (12, 0, 0) decimal machine.

The basic storage of the Univac II is 2000 words on magnetic cores of 12 alphanumeric characters each. Larger memories up to 10,000 words are available. The secondary storage is on magnetic tape units called Uniservos. All input-output, except through a console keyboard, is via the magnetic tape units, which may be loaded directly from typewriter (Unityper) or from punched cards (card-to-tape converter) and unloaded by typewriter, high-speed printer, or tape-to-card converter. The Univac II has a "variable field length" property which is designed to be of use in data processing. All arithmetic is in fixed point "digital" numbers (absolute value less than one).

**Registers.** The following are the registers in the Univac II computer:

1. **rA** Accumulator
2. **rX** X register
3. **rL** Quotient register
4. **rF** Extraction register
5. **rW** 9-word transfer register
6. **rZ** 60-word transfer register
7. **rI** 60-word input register

The following Univac II instrumentation code is taken from a Univac II manual (Ref. 34).

**Conventions.** In describing the actions caused by Univac II system instructions, the phrase "transfer the contents of \_\_\_\_\_ to \_\_\_\_\_" is taken to mean: the information in the component following the "of" is duplicated in the component following the "to." The component *to* which information is transferred is erased of its original contents just before the new informa-

tion is entered into it. *Unless otherwise specified, the contents of the component from which information is transferred remains unaltered.*

**Symbology.** A symbolic notation is used to designate the operations caused by the execution of an instruction. This symbolic notation, as described below, is shown for each instruction. In addition a verbal description is given.

<i>Symbol</i>	<i>Meaning</i>
$\rightarrow$	is transferred to
<b>m</b>	a main storage location whose address is m
( )	the contents of the element within the parentheses. Thus, (m) = the contents of a main storage location where address is m
<b>r</b>	register, the letter of the register follows the r. Thus, rA = register A
$\bar{X}$	those characters of any element, X, which correspond to character positions in rF containing extractors
<b>X</b>	those characters of any element, X, which correspond to character positions in rF containing nonextractors.

An instruction is symbolized by three characters—two characters specify the operation code, the third character, always m, stands for the four-digit address portion of the instruction.

### One-Word Transfer Instructions.

**BOm (m)  $\rightarrow$  rA, rX.** Transfer the contents of the storage location specified by m to both register A and register X.

**FOm (m)  $\rightarrow$  rF.** Transfer the contents of the storage location specified by m to register F.

**LOm (m)  $\rightarrow$  rL, rX.** Transfer the contents of the storage location specified by m to both register L and register X.

**HOm (rA)  $\rightarrow$  m.** Transfer the contents of register A to the storage location specified by m.

**COm (rA)  $\rightarrow$  m; 0  $\rightarrow$  rA.** Transfer the contents of register A to the storage location specified by m. Clear register A to a word of decimal zeros.

**GOm (rF)  $\rightarrow$  m.** Transfer the contents of register F to the storage location specified by m.

**IOm (rL)  $\rightarrow$  m.** Transfer the contents of register L to the storage location specified by m.

**JOm (rX)  $\rightarrow$  m.** Transfer the contents of register X to the storage location specified by m.

**KOm**  $(rA) \rightarrow rL; 0 \rightarrow rA$ . Transfer the contents of register A to register L. Clear register A to a word of decimal zeros.

### Arithmetic Instructions.

**AOm**  $(m) \rightarrow rX; (rX) + (rA) \rightarrow rA$ . Transfer the contents of the storage location specified by m to register X. Then send the contents of register X and register A to the Adder and add them. Return the sum to register A.

**AHm**  $(m) \rightarrow rX; (rX) + (rA) \rightarrow rA, m$ . Transfer the contents of the storage location specified by m to register X. Then send the contents of register X and register A to the Adder and add them. Return the sum to both register A and the storage location specified by m.

**XOm**  $(rX) + (rA) \rightarrow rA$ . Send the contents of register X and register A to the Adder and add them. Return the sum to register A.

**SOm**  $-(m) \rightarrow rX; (rX) + (rA) \rightarrow rA$ . Transfer the contents of the storage location specified by m to register X. In transit reverse the sign of the word being transferred (from 0 to -, or from - to 0). Then send the contents of register X and register A to the Adder and add them. Return the sum to register A. (Note: plus signs are represented by 0.)

**SHm**  $-(m) \rightarrow rX; (rX) + (rA) \rightarrow rA, m$ . Transfer the contents of the storage location specified by m to register X. In transit reverse the sign of the word being transferred (from 0 to -, or from - to 0). Then send the contents of register X and register A to the Adder and add them. Return the sum to both register A and the storage location specified by m.

**MOm**  $(m) \rightarrow rX; (rL) \times (rX) \rightarrow rA$  (rounded). Transfer the contents of the storage location specified by m to register X. Then multiply the contents of register L by the contents of register X. Return an 11-digit product, with the least significant digit rounded, to register A. The previous contents of rX and rF are destroyed.

**NOm**  $-(m) \rightarrow rX; (rL) \times (rX) \rightarrow rA$  (rounded). Transfer the contents of the storage location specified by m to register X. In transit reverse the sign of the word being transferred. Then multiply the contents of register L by the contents of register X. Return an 11-digit product, with the least significant digit rounded, to register A. The previous contents of register X and register F are destroyed.

**POm**  $(m) \rightarrow rX; (rL) \times (rX) \rightarrow rA, rX$  (22 digits). Transfer the contents of the storage location specified by m to register X. Then multiply the contents of register L by the contents of register X. Return the sign and first 11 digits of the product to register A. Return the sign and second 11 digits of the product to register X. The previous contents of register F are destroyed.

**DOm** ( $m$ )  $\rightarrow rA$ ; ( $rA$ )  $\div$  ( $rL$ )  $\rightarrow rA$  (**rounded**),  $\rightarrow rX$  (**unrounded**). Transfer the contents of the storage location specified by  $m$  to register A. Then divide the contents of register A by the contents of register L. Return a rounded quotient to register A, and an unrounded quotient to register X.

**The Second-Digit Modifier F.** The second-digit modifier F allows the isolation and independent treatment of groups of characters, which are stored as a part of a word. Two memory-to-register one-word transfer instructions and six arithmetic instructions may be written with the second instruction digit, F (instead of a zero as they have been shown). The second instruction digit F modifies the instruction and causes it to be operative on only a portion of the word. Those characters of the word upon which it is desired to have the instruction operate are indicated to the computer by an "extract pattern."

**The Extract Pattern.** Each of the 63 Univac II system characters is either an extractor or a nonextractor. Those characters whose seven-place code representations have a binary zero in their rightmost bit positions are extractors; e.g., decimal 1. Those characters whose seven-place Univac II system code representations have a binary one in their rightmost bit positions are nonextractors; e.g., decimal 0.

An extract pattern is a word so arranged that characters which are extractors occupy the same character positions within the extract pattern word as are occupied within their word by those characters that it is desired to enter an operation. All other character positions in the extract pattern are nonextractors. The extract pattern governing an operation must be in register F at the time of execution of the operation. Extract patterns are stored in the main storage and brought to register F (by an FOm instruction) as needed.

The second instruction digit F can modify the BOm and LOm one-word transfer instructions. As modified by the second instruction digit F, the BFm and LFm instructions direct the central computer to:

1. Transfer to corresponding character positions in the appropriate registers (the same register affected by the unmodified instruction), only those characters of the storage location specified by  $m$  whose positions in the word in  $m$  correspond to positions in register F containing extractors.
2. Place zeros in those positions in the receiving registers which correspond to nonextractors in register F. Thus,

$$\text{BFm} \quad (\overline{m}) \rightarrow \overline{rA}, \overline{rX}; 0 \rightarrow \underline{rA}, \underline{rX}.$$

$$\text{LFm} \quad (\overline{m}) \rightarrow \overline{rL}, \overline{rX}; 0 \rightarrow \underline{rL}, \underline{rX}.$$

The second instruction digit F can modify the AOm, SOm, MOm, NOm, POm, and DOm arithmetic instructions. As modified by the F the



are found to be equal take the next instruction word from the storage location specified by  $m$  of the right-hand instruction of the instruction word containing the  $QOm$  rather than the next sequential storage location. Then continue sequential operation from  $m$ . If the contents of register  $A$  do not equal the contents of register  $L$  continue on with uninterrupted sequential operations.

**TOm** If  $(rA) > (rL)$ , **Jump to  $m$** . Send the contents of both register  $A$  and register  $L$  to the comparator, and compare them. If the contents of register  $A$  are found to be algebraically greater than the contents of register  $L$ , take the next instruction word from the storage location specified by  $m$  of the right-hand instruction of the word containing the  $TOm$  instruction rather than the next sequential storage location. Then continue sequential operation from  $m$ . If the contents of register  $A$  are not greater than the contents of register  $L$  continue on with uninterrupted sequential operations.

**OOm SKIP**. Proceed to the next instruction.

**90m STOP**. Stop processing.

**ROm 000000 U0 ( $c + 1$ )  $\rightarrow m$** . Place in the storage location specified by  $m$  the following word: 000000 U0 ( $c + 1$ ), where  $c$  = the storage location of the instruction word of which  $ROm$  order is a part.

### Shift Instructions.

**.nm Shift ( $rA$ ) right, with sign,  $n$  places.\*** Shift all twelve characters of the word in register  $A$  the number of places specified by  $n$  to the right. (The  $n$  least significant digits of the word are destroyed by this shift.) Place zeros in the  $n$  left-hand character position of register  $A$ .

**;nm Shift ( $rA$ ) left, with sign,  $n$  places.\*** Shift all twelve characters of the word in register  $A$  the number of places specified by  $n$  to the left. (The  $n$  left-hand characters of the word are destroyed by this shift.) Place zeros in the  $n$ -least significant digit positions of register  $A$ .

**-nm Shift ( $rA$ ) right, excluding sign,  $n$  places.\*** Shift the eleven significant digits (excluding sign) the number of places specified by  $n$  to the right. (The  $n$  least significant digits of the word will be destroyed by this shift.) Place zeros in the  $n$ -most significant digit positions excluding sign of register  $A$ .

**Onm Shift ( $rA$ ) left, excluding sign,  $n$  places.\*** Shift the eleven significant digits (excluding sign) the number of places specified by  $n$  to the left. (The  $n$ -most significant digit of the word will be destroyed by this shift.) Place zeros in the  $n$ -least significant digit positions of register  $A$ .

\*  $n$  = may be any digit from 1 to 9.

**Multiword Transfer Instructions.**

**Vnm** ( $m, m + 1, \dots, m + n - 1$ )  $\rightarrow$  **rW**. If  $n = 0$ , skip. If  $n$  equals 0, treat this instruction as a skip instruction. If  $n$  equals any other digit, transfer  $n$  consecutive words beginning with the word in  $m$  from the main storage to register  $W$ .

**Wnm** (**rW**)  $\rightarrow m, m + 1, \dots, m + n - 1$ . If  $n = 0$ , skip. If  $n$  equals 0, treat this instruction as a skip instruction. If  $n$  equals any other digit, transfer  $n$  consecutive words from register  $W$  to  $n$  consecutive storage locations in the main storage beginning with the storage location specified by  $m$ .

**Ynm** ( $m, m + 1, \dots, m + 10n - 1$ )  $\rightarrow$  **rZ**. If  $n = 0, 7, 8, 9$ , skip. If  $n = 0, 7, 8, 9$  treat this instruction as a skip instruction. If  $n = 1, 2, 3, 4, 5$ , or 6, transfer  $10n$  consecutive words beginning with the word in storage location  $m$  from the main storage to register  $Z$ .

**Znm** (**rZ**)  $\rightarrow m, m + 1, \dots, m + 10n - 1$ . If  $n = 0, 7, 8, 9$ , skip. If  $n = 0, 7, 8$ , or 9 treat this instruction as a skip instruction. If  $n$  equals 1, 2, 3, 4, 5, or 6, transfer  $10n$  consecutive words from register  $Z$  to  $10n$  consecutive storage locations in the main storage beginning with the storage location specified by  $m$ .

**Input-Output Instructions.**

**Inm** **60 words from tape to rI, forward**. Read the next block from tape mounted on Uniservo  $n$ , with the tape moving in the forward direction. Place the block as it is being read from tape, into register  $I$ .

**2nm** **60 words from tape to rI, backward**. Read the next block from the tape mounted on Uniservo  $n$ , with the tape moving in the backward direction. Place the block, as it is being read from tape, into register  $I$  in the same word order as it would have been placed in register  $I$  had it been read with a forward read.

**30m** (**rI**)  $\rightarrow m, m + 1, \dots, m + 59$ . Transfer the contents of register  $I$  to 60 consecutive storage locations beginning with storage location  $m$ .

**40m** (**rI**)  $\rightarrow m, m + 1, \dots, m + 59$ . Same as 30m instruction, above.

**3nm** (**rI**)  $\rightarrow m, m + 1, \dots, m + 59$ ; **60 words from tape  $\rightarrow$  rI, forward**. Transfer the contents of register  $I$  to 60 consecutive storage locations beginning with storage location  $m$ . Then read the next block from the tape mounted on Uniservo  $n$ , with the tape moving in the forward direction. Place the block, as it is being read from tape, into register  $I$ .

**4nm** (**rI**)  $\rightarrow m, m + 1, \dots, m + 59$ ; **60 words from tape to rI, backward**. Transfer the contents of register  $I$  to 60 consecutive storage

locations beginning with storage location  $m$ . Then read the next block from the tape mounted on Uniservo  $n$  with the tape moving in the backward direction. Place the block, as it is being read from tape, into register I in the same word order as it would have been placed in register I had it been read with a forward read.

**10m Supervisory Control Keyboard**  $\rightarrow m$ . Stop processing until a word is typed on the supervisory control keyboard. When a word has been typed, and the word release key has been depressed, place the typed word into storage location  $m$ .

**5nm ( $m, m + 1, \dots, m + 59$ )**  $\rightarrow$  **Tape, 250 characters per inch.** Transfer the contents of the 60 consecutive storage locations beginning with storage location  $m$  to register O. Then write the contents of register O onto the tape mounted on Uniservo  $n$  at a recording density of 250 characters per inch.

**7nm ( $m, m + 1, \dots, m + 59$ )**  $\rightarrow$  **Tape, 50 characters per inch.** Transfer the contents of the 60 consecutive storage locations beginning with storage location  $m$  to register O. Then write the contents of register O onto the tape mounted on Uniservo  $n$  at a recording density of 50 characters per inch.

**50m ( $m$ )**  $\rightarrow$  **S.C.P.** Write the contents of storage location  $m$  on the supervisory control printer. This instruction can be modified by a set of buttons on the supervisory control panel to cause the contents of:

1. Register A.
2. Register L.
3. Register X.
4. Register F.
5. The Control Counter, or
6. The Control Register.
7. Successive storage locations beginning with a specified storage location to be printed on the supervisory control printer.

**6nm Rewind tape.** Rewind the tape mounted on Uniservo  $n$ .

**8nm Rewind tape with interlock.** Rewind the tape mounted on Uniservo  $n$ . Set an interlock on Uniservo  $n$  which will cause the computer to stop if any other order is directed to that Uniservo.

The input-output instructions which direct Uniservo operations are symbolized with a second instruction digit of  $n$ . The  $n$  specifies the particular Uniservo to which the instruction is directed and is usually written as 1, 2, 3, 4, 5, 6, 7, 8, 9, —, A, B, C, D, E, or F. It may, however, be written as a delta ( $\Delta$ ). If it is written as a  $\Delta$ , the Uniservo to which the order is directed is determined by a set of 16 buttons (one for each Uniservo) on the supervisory control console. Only one of these buttons may be depressed at any one time. The Uniservo corresponding to the depressed button becomes Uniservo  $\Delta$ .

**Breakpoint Instructions.**

**,m Breakpoint stop.** If the comma breakpoint switch on the supervisory control panel is in the breakpoint position, stop. If the comma breakpoint switch is not in the breakpoint position, skip.

**Qnm** If the button in the breakpoint section of the supervisory control panel corresponding to  $n$  is not depressed, treat as a **OOm** instruction. If the button in the breakpoint section of the supervisory control panel corresponding to  $n$  is depressed, perform the comparison between the contents of register A and register L, light a neon on the supervisory control panel indicating whether or not a jump would normally take place, and stop. The operator can then force a jump or no jump to take place and continue processing.

**Tnm** If the button in the breakpoint section of the supervisory control panel corresponding to  $n$  is not depressed, treat as a **TOm** instruction. If the button is depressed, perform the comparison between the contents of register A and register L, light a neon on the supervisory control panel indicating whether or not a jump would normally take place, and stop. The operator can then force a jump or no jump to take place and continue processing.

**Summary Table.** A summary of the instructions for Univac II is given in Table 8.

TABLE 8. SUMMARY OF UNIVAC II INSTRUCTIONS

Instruc- tion	Execution Time, <sup>a</sup> microseconds	Description
<b>AOm</b>	200	$(m) \rightarrow rX; (rX) + (rA) \rightarrow rA$
<b>AFm</b>	200	$(\bar{m}) \rightarrow r\bar{X}; 0 \rightarrow r\bar{X}; (rX) + (rA) \rightarrow rA$
<b>AHm</b>	240	$(m) \rightarrow rX; (rX) + (rA) \rightarrow rA \rightarrow m$
<b>BOm</b>	120	$(m) \rightarrow rA, rX$
<b>BFm</b>	120	$(\bar{m}) \rightarrow rA, r\bar{X}; 0 \rightarrow rA, r\bar{X}$
<b>COm</b>	120	$(rA) \rightarrow m; 0 \rightarrow rA$
<b>DOm</b>	(3700)	$(m) \rightarrow rA; (rA) \div (rL) \rightarrow rA$ (rounded) $rX$ (unrounded)
<b>EOm</b>	120	$(\bar{m}) + (rA) \rightarrow rA$
<b>EFm</b>	200	$(rA) + (\bar{m}) \rightarrow rA \rightarrow m$
<b>FOm</b>	120	$(m) \rightarrow rF$
<b>GOm</b>	120	$(rF) \rightarrow m$
<b>HOm</b>	120	$(rA) \rightarrow m$
<b>IOm</b>	120	$(rL) \rightarrow m$
<b>JOm</b>	120	$(rX) \rightarrow m$
<b>KOm</b>	120	$(rA) \rightarrow rL; 0 \rightarrow rA$
<b>LOm</b>	120	$(m) \rightarrow rL; rX$

<sup>a</sup> Times shown in parentheses are statistical averages. The exact times for execution depends upon the data upon which these orders are operating.

TABLE 8. SUMMARY OF UNIVAC II INSTRUCTIONS (*Continued*)

Instruction	Execution Time, <sup>a</sup> microseconds	Description
LFm	120	$(\overline{m}) \rightarrow \overline{rL}; 0 \rightarrow \overline{rL}$
MOm	(1900)	$(m) \rightarrow \overline{rX}; (rL) \times (rX) \rightarrow rA$ (rounded)
MFm	(1900)	$(\overline{m}) \rightarrow \overline{rX}; 0 \rightarrow \overline{rX}; (rL) \times (rX) \rightarrow rA$ (rounded)
NOm	(1900)	$-(m) \rightarrow \overline{rX}; (rL) \times (rX) \rightarrow rA$ (rounded)
NFm	(1900)	$-(\overline{m}) \rightarrow \overline{rX}; 0 \rightarrow \overline{rX}; (rL) \times (rX) \rightarrow rA$ (rounded)
POm	(1900)	$(m) \rightarrow \overline{rX}; (rL) \times (rX) \rightarrow rA, rX$ (22 digits unrounded)
PFm	(1900)	$(\overline{m}) \rightarrow \overline{rX}; 0 \rightarrow \overline{rX}; (rL) \times (rX) \rightarrow rA, rX$ (22 digits unrounded)
Qnm	200	If $(rA) = (rL)$ , jump to m; stop if breakpoint n is selected
ROm	120	Record 000000 UO [c + 1] in m
SOm	200	$-(m) \rightarrow \overline{rX}; (rX) + (rA) \rightarrow rA$
SHm	280	$-(m) \rightarrow \overline{rX}; (rX) + (rA) \rightarrow rA \rightarrow m$
SFm	200	$-(\overline{m}) \rightarrow \overline{rX}; 0 \rightarrow \overline{rX}; (rX) + (rA) \rightarrow rA$
Tnm	200	If $(rA) > (rL)$ , jump to m; stop if breakpoint n is selected
UOm	120	Jump to m
Vnm	80 + 40n	$(m), (m + 1), \dots, (m + n - 1) \rightarrow rW$
Wnm	80 + 40n	$(rW) \rightarrow m, m + 1, \dots, m + n - 1$
XOm	120	$(rX) + (rA) \rightarrow rA$
Ynm	80 + 405n	$(m), (m + 1), \dots, (m + 10n - 1) \rightarrow rZ$
Znm	80 + 405n	$(rZ) \rightarrow m, m + 1, \dots, m + 10n - 1$
OOm	120	SKIP
.nm	80 + 40n	Shift (rA) right, with sign, n places
;nm	80 + 40n	Shift (rA) left, with sign, n places
-nm	80 + 40n	Shift (rA) right, without sign, n places
Onm	80 + 40n	Shift (rA) left, without sign, n places
90m	120	STOP
,Om	120	Stop if comma breakpoint is selected
1nm	(3500)	60 words from tape n to rI, forward
2nm	(3500)	60 words from tape n to rI backward
30m	2675	$(rI) \rightarrow m, m + 1, \dots, m + 59$
40m	2675	Same as 30m, above
3nm	3500	$(rI) \rightarrow m, m + 1, \dots, m + 59$ ; 60 words from tape n to rI, forward
4nm	(3500)	$(rI) \rightarrow m, m + 1, \dots, m + 59$ ; 60 words from tape n to rI, backward
5nm	(3500)	$(m), (m + 1), \dots, (m + 59) \rightarrow$ tape n, 250 characters/inch
6nm	299	Rewind tape n
7nm	(3500)	$m, (m + 1), \dots, (m + 59) \rightarrow$ tape n, 50 characters/inch
8nm	200	Rewind tape n, with interlock

<sup>a</sup> Times shown in parentheses are statistical averages. The exact times for execution depends upon the data which these orders are operating.

### IBM 650 Instruction Logic

The basic IBM 650 is a magnetic drum (10, 0, 0) decimal computer with one-plus-one address instruction logic. It has a storage of 1000 or 2000 10-digit words (plus sign) with addresses 0000-0999 or 0000-1999. More extended versions of the equipment have built-in floating point arithmetic and index accumulators, but the basic machine will be described here. There are three arithmetic registers in addition to the standard program register and program counter. All information from the drum to the arithmetic unit passes through a signed 10-digit *distributor*. A twenty-digit *accumulator* is divided into a lower and upper part, each of 10 digits with sign. Each of these is addressable (distributor 8001, lower accumulator 8002, and upper accumulator 8003). Each accumulator may be cleared to zero separately (in IBM 650 terminology, "reset"). The entire 20-digit register can be considered as a unit, or each part separately (but affecting the other in case of carries). The 10-digit instruction is broken down into the following form:

10	9	8	7	6	5	4	2	3	1	0
Op. Code		Data Address				Next Instruction Address				Sign

One particular instruction, Table Look-Up, allows automatic table search for one particular element in a table, which can be stored with a corresponding functional value. Input-output is via 80-digit numerical punched cards. An "alphabetic device" allows limited alphabetical entry on cards. Only certain 10-word groups on the magnetic drum are available for input and output. The following information is taken from an IBM 650 manual (Ref. 102). Much of the input-output is handled via board wiring, which is not described in detail below. The two-digit pair represents the machine code. The BRD (Branch on Digit) operation is used with special board wiring to tell when certain specific card punches exist.

#### Input-Output Instructions.

**70 RD (Read).** This operation code causes the machine to read cards by a two-step process. First, the contents of the 10 words of read buffer storage are automatically transferred to one of the 20 (or 40) possible 10-word groups of read general storage. The group selected is determined by the D address of the Read instruction. Secondly, a card is moved under the reading brushes, and the information read is entered into buffer storage for the next Read instruction.

**71 PCH (Punch).** This operation code causes card punching in two steps. First the contents of one of the 20 (or 40) possible 10-word groups of punch storage are transferred to punch buffer storage. The group selected is specified by the D address of the Punch instruction. Secondly, the card is punched with the information from buffer storage.

**69 LD (Load Distributor).** This operation code causes the contents of the D address location of the instruction to be placed in the distributor.

**24 STD (Store Distributor).** This operation code causes the contents of the distributor with the distributor sign to be stored in the location specified by the D address of the instruction. The contents of the distributor remain undisturbed.

### **Addition and Subtraction Instructions.**

**10 AU (Add to Upper).** This operation code causes the contents of the D address location to be added to the contents of the upper half of the accumulator. The lower half of the accumulator will remain unaffected unless the addition causes the sign of the accumulator to change, in which case the contents of the lower half of the accumulator will be complemented. Also, the units position of the upper half of the accumulator will be reduced by one.

**15 AL (Add to Lower).** This operation code causes the contents of the D address location to be added to the contents of the lower half of the accumulator. The contents of the upper half of the accumulator could be affected by carries.

**11 SU (Subtract from Upper).** This operation code causes the contents of the D address location to be subtracted from the contents of the upper half of the accumulator. The contents of the lower half of the accumulator will remain unaffected unless the subtraction causes a change of sign in the accumulator, in which case the contents of the lower half of the accumulator will be complemented. Also, the units position of the upper half of the accumulator will be reduced by one.

**16 SL (Subtract from Lower).** This operation code causes the contents of the D address location to be subtracted from the contents of the lower half of the accumulator. The contents of the upper half of the accumulator could be affected by carries.

**60 RAU (Reset and Add into Upper).** This operation code resets the entire accumulator to plus zero and adds the contents of the D address location into the upper half of the accumulator.

**65 RAL (Reset and Add into Lower).** This operation code resets the entire accumulator to plus zero and adds the contents of the D address location into the lower half of the accumulator.

**61 RSU (Reset and Subtract into Upper).** This operation code resets the entire accumulator to plus zero and subtracts the contents of the D address location into the upper half of the accumulator.

**66 RSL (Reset and Subtract into Lower).** This operation code resets the entire accumulator to plus zero and subtracts the contents of the D address location into the lower half of the accumulator.

#### **Accumulator Store Instructions.**

**20 STL (Store Lower in Memory).** This operation code causes the contents of the lower half of the accumulator with the accumulator sign to be stored in the location specified by the D address of the instruction. The contents of the lower half of the accumulator remain undisturbed.

It is important to remember that the D address for all store instructions must be 0000-1999. An 8000 series D address will not be accepted as valid by the machine on any of the store instructions.

**21 STU (Store Upper in Memory).** This operation code causes the contents of the upper half of the accumulator with the accumulator sign to be stored in the location specified by the D address of the instruction. If STU is performed after a division operation, and before another division, multiplication, or reset operation takes place, the contents of the upper accumulator will be stored with the sign of the remainder from the divide operation (Op-Code 14). The contents of the upper half of the accumulator remain undisturbed.

**22 STDA (Store Lower Data Address).** This operation code causes positions 8-5 of the distributor to be replaced by the contents of the corresponding positions of the lower half of the accumulator. The modified word in the distributor with the sign of the distributor is then stored in the location specified by the D address of the instruction.

**23 STIA (Store Lower Instruction Address).** This operation code causes positions 4-1 of the distributor to be replaced by the contents of the corresponding positions of the lower half of the accumulator. The modified word in the distributor with the sign of the distributor is then stored in the location specified by the D address of the instruction. The contents of the lower half of the accumulator remain unchanged, and the sign of the accumulator is not transferred to the distributor. The modified word remains in the distributor upon completion of the operation.

#### **Absolute Value Instructions.**

**17 AABL (Add Absolute to Lower).** This operation code causes the contents of the D address location to be added to the contents of the lower half of the accumulator as a positive factor regardless of the

actual sign. When the operation is completed, the distributor will contain the D address factor with its actual sign.

**67 RAABL (Reset and Add Absolute into Lower).** This operation code resets the entire accumulator to zeros and adds the contents of the D address location into the lower half of the accumulator as a positive factor regardless of its actual sign. When the operation is completed, the distributor will contain the D address factor with its actual sign.

**18 SABL (Subtract Absolute from Lower).** This operation code causes the contents of the D address location to be subtracted from the contents of the lower half of the accumulator as a positive factor regardless of the actual sign. When the operation is completed, the distributor will contain the D address factor with its actual sign.

**68 RSABL (Reset and Subtract Absolute into Lower).** This operation code resets the entire accumulator to plus zero and subtracts the contents of the D address location into the lower half of the accumulator as a positive factor, regardless of the actual sign. When the operation is completed, the distributor will contain the D address factor with its actual sign.

### **Multiplication and Division.**

**19 MULT (Multiply).** This operation code causes the machine to multiply. A 10-digit multiplicand may be multiplied by a 10-digit multiplier to develop a 20-digit product. The multiplier must be placed in the upper accumulator prior to multiplication. The location of the multiplicand is specified by the D address of the instruction. The product is developed in the accumulator beginning in the low-order position of the lower half of the accumulator and extending to the left into the upper half of the accumulator as required.

**14 DIV (Divide).** This operation code causes the machine to divide without resetting the remainder. A 20-digit dividend may be divided by a 10-digit divisor to produce a 10-digit quotient. In order to remain within these limits, the absolute value of the divisor must be *greater than* the absolute value of that portion of the dividend that is in the upper half of the accumulator. The entire dividend is placed in the 20-position accumulator. The location of the divisor is specified by the D address of the divide instruction.

**64 DIV RU (Divide and Reset Upper).** This operation code causes the machine to divide as explained under operation code 14 (DIV). However, the upper half of the accumulator containing the remainder with its sign is reset to zeros.

### **Branching Instructions (Decision Operations).**

**44 BRNZU (Branch on Non-Zero in Upper).** This operation code causes the contents of the upper half of the accumulator to be exam-

ined for zero. If the contents of the upper half of the accumulator is nonzero, the location of the next instruction to be executed is specified by the D address. If the contents of the upper half of the accumulator is zero, the location of the next instruction to be executed is specified by the I address. The sign of the accumulator is ignored.

**45 BRNZ (Branch on Non-Zero).** This operation code causes the contents of the entire accumulator to be examined for zero. If the contents of the accumulator is nonzero, the location of the next instruction to be executed is specified by the D address. If the contents of the accumulator is zero, the location of the next instruction to be executed is specified by the I address. The sign of the accumulator is ignored.

**46 BRMIN (Branch on Minus).** This operation code causes the sign of the accumulator to be examined for minus. If the sign of the accumulator is minus, the location of the next instruction to be executed is specified by the D address. If the sign of the accumulator is positive, the location of the next instruction to be executed is specified by the I address. The contents of the accumulator are ignored.

**47 BROV (Branch on Overflow).** This operation code causes the overflow circuit to be examined to see whether it has been set. If the overflow circuit is set, the location of the next instruction to be executed is specified by the D address. If the overflow circuit is not set, the location of the next instruction to be executed is specified by the I address.

**90-99 BRD 1-10 (Branch on 8 in Distributor Position 1-10).** This operation code examines a particular digit position in the distributor for the presence of an 8 or 9. Codes 91-99 test positions 1-9, respectively, of the test word; code 90 tests position 10. If an 8 is present, the location of the next instruction to be executed is specified by the D address. If a 9 is present, the location of the next instruction to be executed is specified by the I address. The presence of other than an 8 or 9 will stop the machine.

### Shift Instructions.

**30 SRT (Shift Right).** This operation code causes the contents of the entire accumulator to be shifted right the number of places specified by the units digit of the D address of the shift instruction. A maximum shift of nine positions is possible. A data address with units digit of zero will result in no shift. All numbers shifted off the right end of the accumulator are lost.

**31 SRD (Shift Round).** This operation causes the contents of the entire accumulator to be shifted right the number of places specified by the units digit of the D address of the instruction. A 5 is added (-5 if the accumulator is negative) in the twenty-first (blind) position of the amount in the accumulator. A data address units digit of zero will shift 10 places right with rounding.

**35 SLT (Shift Left).** This operation code causes the contents of the entire accumulator to be shifted left the number of places specified by the units digit of the D address of the instruction. A maximum shift of nine positions is possible. A data address with a units digit of zero will result in no shift. All numbers shifted off the left end of the accumulator are lost. However, the overflow circuit will not be turned on.

**36 SCT (Shift Left and Count).** This operation code causes (1) the contents of the entire accumulator to be shifted to the left until a nonzero digit is in the most significant place, (2) a count of the number of places shifted to be inserted in the two low-order positions of the accumulator. This instruction is to aid fixed-point scaling.

#### **Table Look-Up Instructions.**

**84 TLU (Table Look-Up).** This operation code performs an automatic table look-up using the D address as the location of the first table argument and the I address as the address of the next instruction to be executed. The argument for which a search is to be made must be in the distributor. The address of the table argument equal to, or higher than (if no equal exists) the argument given is placed in positions 8-5 of the lower accumulator. The search argument remains, unaltered, in the distributor.

#### **Miscellaneous Instructions.**

**00 No-Op (No Operation).** This code performs no operation. The data address is bypassed, and the machine automatically refers to the location specified by the instruction address of the **No-Op** instruction.

**01 Stop.** This operation code causes the program to stop provided the *programmed* switch on the control console is in the stop position. When the *programmed* switch is in the run position the 01 code will be ignored and treated in the same manner as **00 (No-Op)**.

#### **Datatron 205 Instruction Logic**

A typical magnetic drum computer of the single-address type is the Datatron 205 computer manufactured by the ElectroData Division of the Burroughs Corporation. The following description of the machine is obtained from a Datatron manual (Ref. 32). This computer, in its simplest form, is a fixed point, decimal (10, 0, 0) computer. The numbers are "digital" numbers, in absolute value less than one. Each instruction (and number) occupies one machine word. The address part of an instruction occupies the four least significant digits, with a two-digit operation code immediately preceding.

Storage is on a magnetic drum with 4000 words capacity. A quick

access storage (recirculating loop) has a capacity of 80 words in four separate blocks of 20 words each. Addresses in the main storage range from 0000 through 3999; addresses in the quick access storage are 4000, 5000, 6000, and 7000. Numbers block-transferred back and forth between main and quick access storage can ordinarily be altered in one while remaining unchanged in the other.

### Registers.

**A register.** This register contains 11 decimal-digit positions. Ten are for the number or instruction and one for the algebraic sign. This register acts as an accumulator, and the results of most operations appear here at the end of the operation.

**R register.** This register contains 10 decimal-digit positions (no algebraic sign). In several operations, it serves as an extension of the A register and holds the 10 least significant digits of the number contained in the combined A and R registers.

**B register.** This register contains 4 decimal-digit positions, and is used to facilitate the modification of commands and for tallying. As each command is received from memory, it is checked to determine whether its address part is to be modified or not. This is determined by a 1 or a 0 in the algebraic sign position. If the sign digit is 1, the four-digit number contained in the B register is added absolutely to the four least significant digits (the address part) of the command, and the command is then executed as modified by the contents of the B register. It is important to note that the commands with negative sign, as stored in the memory, are not altered by the operation of this register; they are temporarily modified in the electronic registers immediately before execution. Thus, the same command may be executed many times during a computation, temporarily modified each time by a different number in the B register.

The R register may be loaded from the drum by means of a right shift through the A register.

Information to be punched out on paper tape or printed on the typewriter comes from the A register.

**Arithmetic Instructions.** In all arithmetic operations one operand is stored in the A register, the other having been fetched from the storage location specified in the address part of the command word. If the sign digit of the command word is 1, the B register will be added to the address before the command arrives at the C register. If on input the sign digit of the command is 3, the B register will be added to the address before the command reaches the drum; and the word will be stored with a 1 in the sign digit, to produce B modification on execution. If on input the sign digit of the command word is 2, the B register will be added to the address

before the command reaches the drum and the word will be stored with a 0 in the sign digit. In the following description the alphanumeric codes and two-digit numbers in parentheses after the name of the command are those used on standard code sheets.

### Addition.

**ADD (ad x, 74).** Add to the number in the A register the number in storage location x. This command and associated add commands do not affect the contents of the R register.

**SUBTRACT (su x, 75).** Subtract from the contents of the A register the number in storage location x.

**ADD ABSOLUTE VALUE (ada x, 76).** Add to the A register the number in storage location x with a positive sign attached.

**SUBTRACT ABSOLUTE VALUE (sua x, 77).** Subtract from the number in the A register the value of the number in storage location x with a positive sign attached.

**CLEAR AND ADD (ca x, 64), CLEAR AND SUBTRACT (cs x, 65), CLEAR AND ADD ABSOLUTE VALUE (caa x, 66), and CLEAR AND SUBTRACT ABSOLUTE VALUE (csa x, 67).** These commands clear the A register before the operation, and then have the same effect as the corresponding commands without the clearing.

### Multiplication.

**MULTIPLY AND HOLD (mh x, 60).** Bring the number from storage location x into the D register; multiply it by the number in the A register; and hold the 20-digit product in the A and R registers. In the MULTIPLY-HOLD command, R is considered to be an extension of A. Before any multiplication command the R register is automatically cleared.

**MULTIPLY AND ROUND OFF (mr x, 70).** Bring the number from storage location x into the D register; multiply it by the number in the A register and round off. If the first digit of R is 5 or greater, increase the absolute value of the number in the A register by  $10^{-10}$ . If the first digit of R is less than 5, do not change A. In either case, the R register is cleared after the command has been executed. There is no possibility of overflow upon execution of a MULTIPLY AND ROUND OFF command.

**ROUND OFF (ro, 23).** If the first digit of R is 5 or greater, add  $10^{-10}$  to the absolute value of the number in the A register and clear R. If the first digit of R is less than 5, clear R. Overflow is possible if A contains all 9's and the first digit of R is 5 or greater.

### Division.

**DIVIDE (div x, 61).** Divide the number in the A and R registers by the number in storage location x. In this case the R register is considered

as an extension of A. If the number in A and R is less in absolute value than the number in x, the division is performed; the A register will contain the 10-digit quotient; and the R register will contain the true remainder after the DIVIDE command has been executed. If, however, the number in A and R is greater in absolute value or equal to the number in the D register, the overflow toggle is set; and A and R are cleared.

**CLEAR R (cl R, 33).** Set all digits of the R register to zero.

### Logical Instructions.

#### Unit Adjust.

**UNIT ADJUST (ua, 06).** Set the "1" toggle in the first position of the A register. This command effectively increases an even digit in A1, making it odd; and leaves an odd digit unchanged. This command is normally used after a right shift to compensate for overflow on an addition. It allows the coder to compensate for the overflow without testing for sign.

A1 before ua	A1 after ua
0	1
1	1
2	3
3	3
4	5
5	5
6	7
7	7
8	9
9	9

**Storage.** The results of arithmetic operations are stored in the A register. The transfer of these results to storage is effected by two commands which do not affect the R register. They are:

**TO MEMORY AND HOLD (tmh x, 12).** Store the number in the A register in storage location x, and retain the number in the A register.

**TO MEMORY AND CLEAR (tmc x, 02).** Store the number in the A register in storage location x, and clear the A register, including the sign position.

#### Shift.

**SHIFT RIGHT (sr n, 13).** Shift the number in the A and R registers, not including sign, n places to the right (n is interpreted modulo 20). The n rightmost digits are lost. The n leftmost digits become zero. In this command as well as the following command, the number of shifts (n) is indicated by the address part of the command word.

**SHIFT LEFT (sl r, 14).** Circulate the number in the A and R registers, not including sign,  $n$  places to the left ( $n$  is interpreted modulo 20). The  $n$  leftmost digits in A become the  $n$  rightmost significant digits in R.

**LOGICAL CYCLE LEFT (lcl n, 01).** Circulate the A register, including the sign digit,  $n + 1$  places to the left ( $n$  is interpreted modulo 20). The R register is not affected by this command.

**SPECIAL LEFT SHIFT (spl y, 15).** If the number in the A register is not zero, shift the A and R registers to the left until the first nonzero digit in A is in the first position of the A register. Change of control does not take place. If the A register contains zero, replace the contents of the A register by the contents of the R register and change control to storage location  $y$ .

This command *normalizes* the number in A and R and, therefore, is frequently used in floating point operations.

Since it is important to know the number of shifts that occur as a result of the SPECIAL LEFT SHIFT, a count of these shifts is stored in the special counter.

**Special Counter.** The special counter is a four-toggle binary counter recycling on 15 (1111). It is used in the machine for three purposes:

1. To count the shifts during multiplication.
2. To count the shifts during division.
3. To count the shifts necessary to normalize a number with a special left. The number in the special counter after the completion of a MULTIPLY or DIVIDE command is 9 (except when division sets the overflow toggle). After a SPECIAL LEFT (spl  $y$ ) command the special counter shows the number of zeros that were to the left of the first nonzero digit in A before the shift. In the case where  $A = 0$ , the special counter counts to 10. The number in the special counter is retained until a multiply, divide, or other special left has been executed. In all these cases it is cleared before the new number is added to it. Information may be obtained from the special counter by means of two commands.

**ADD SPECIAL COUNTER (spc+, 16).** Add (algebraically) the number in the special counter scaled  $10^{-10}$  to the number in the A register.

**SUBTRACT SPECIAL COUNTER (spc-, 17).** Subtract (algebraically) the number in the special counter scaled  $10^{-10}$ , from the number in the A register.

As in the addition commands, there is a possibility of overflow. Also, the special counter can contain a forbidden combination (when  $A = 0$  before spl). In such a case, the execution of the special counter (either add or subtract) command will signal the forbidden combination alarm, and the computer halts.

**EXTRACT (ex x, 63).** In normal operation, storage location  $x$  will contain a combination of 0's and 1's. This number is referred to as an extract combination. For the digits of the extract combination that are zeros, replace the corresponding digits in the A register including the sign by zeros. For the digits of the extract combination that are ones, leave the corresponding digits in the A register, including the sign, unchanged.

### Loop Transfer.

**BLOCK TRANSFER INTO A LOOP (bl (4)  $\leftarrow$  x, 34).** Block transfer into the 4000 loop the 20 words at storage location  $x$  through storage location  $x + 19$ , the main storage and loop addresses corresponding modulo 20. The commands  $\text{bl}(5) \leftarrow x$ ,  $\text{bl}(6) \leftarrow x$ ,  $\text{bl}(7) \leftarrow x$  are executed in the same way as the above command, the 20 words being blocked into the 5000, 6000, and 7000 loops respectively. Because it is not possible to block transfer from one loop to another, all blocking commands are interpreted modulo 4000. Information transferred from main storage into a high-speed loop remains in main storage.

**BLOCK TRANSFER INTO MAIN MEMORY (bl(4)  $\rightarrow$  x, 24).** Block transfer into 20 main storage positions starting at storage location  $x$  the words in the 4000 loop. The words will go into main storage locations corresponding modulo 20 to their loop addresses. The  $\text{bl}(5) \rightarrow x$ ,  $\text{bl}(6) \rightarrow x$ ,  $\text{bl}(7) \rightarrow x$  commands are defined as above with the exception that 5000, 6000, and 7000, respectively, are substituted for 4000. After execution of the command, the information is retained in the loop until new information is written in.

### Change Control.

**UNCONDITIONAL CHANGE OF CONTROL (cu y, 20).** Change control unconditionally to storage location  $y$ . This command replaces the contents of the control counter with the address part of the **cu** command.

**CONDITIONAL CHANGE OF CONTROL (cc y, 28).** If the overflow toggle is set, change control to  $y$  and reset the overflow toggle to 0. If the overflow toggle is not set, ignore this command.

**Control Block Transfer.** The two commands listed below and two of the commands under Control-Record are a combination of change of control commands and block transfer.

**BLOCK AND UNCONDITIONAL CHANGE OF CONTROL (cub y, 30).** Block transfer into the 7000 loop the words at 20 consecutive locations starting at main storage location  $y$ , and change control to the image  $y$  in the 7000 loop;  $y$  is interpreted modulo 4000. It is important to note that the words occupy locations in the 7000 loop which are congruent

modulo 20 to their previous main storage addresses. Change of control is effected by replacing the first two digits in the address part of the command word by 70 and placing this in the control counter. This command, therefore, automatically changes control to the 7000 loop.

**BLOCK AND CONDITIONAL TRANSFER OF CONTROL (ccb y, 38).** If the overflow toggle is set, this command acts in the same way as the cub y command described in the previous paragraph. If the overflow toggle is not set, this command is ignored.

Since the control counter performs the operation of counting up one only after the command word has been fetched, it can be seen readily that the cub command must replace the first two digits by 70, not merely replace the first digit by 7. If, say, a cub 2999 were executed and only the first digit were replaced by 7, one command execution would cause a control change from the 7000 loop into main storage.

#### **Control-Record.**

**UNCONDITIONAL CHANGE OF CONTROL AND RECORD (cuR y, 21).** Clear the R register; replace its four most significant digits with the contents of the control counter; and change control unconditionally to storage location y. This command records the address from which a departure into a subroutine has been made and makes it possible to provide in advance for a control change to the next address in the main routine.

**CONDITIONAL CHANGE OF CONTROL AND RECORD (ccR y, 29).** If the overflow toggle is set, clear the R register, replace its four most significant digits with the contents of the control counter, and change control to storage location y. If the overflow toggle is not set, ignore this command.

**BLOCK, UNCONDITIONAL CHANGE OF CONTROL AND RECORD (cubR y, 31).** Clear the R register and replace its four most significant digits with the contents of the control counter; block transfer the words in storage location y and the following 19 locations into the 7000 loop and change control unconditionally to the number corresponding to y in the 7000 loop.

**BLOCK, CONDITIONAL TRANSFER OF CONTROL AND RECORD (ccbR y, 39).** If the overflow toggle is set, effect a cubR y command. If the overflow toggle is not set, ignore this command.

All block transfer commands are interpreted modulo 4000. The y's in the commands that perform a blocking operation are interpreted modulo 4000, while those in other commands are interpreted modulo 8000. These commands do not disturb the contents of any register.

**ZERO CHECK (z y, 04).** If the A register does not contain  $\pm 0$  before the execution of this command, change control to storage location

y. If the A register contains  $\pm 0$ , ignore this command ( $-0$  is changed to  $+0$ ).

**SIGN COMPARE (sgc x, 73).** If the sign of the A register is the same as the sign of the word in storage location x, do not set the overflow toggle. If the sign of the A register is not the same as the sign of the word in storage location x, set the overflow toggle.

The SIGN COMPARE command is usually followed by a cc, ccR, ccb, or ccbR command. If it is not, the setting of the overflow toggle will cause the machine to stop. It is only because of the sgc x command that the programmer need know the sign of zero.

### **B Register.**

**REPLACE B (B x, 72).** Set the B register to the last four digits in storage location x. The sign of the number in x has no bearing on the setting of B.

**INCREASE B (B+, 32).** Add 1 to the contents of the B register. If the B register before the execution of a B+ command contains 9999, it will contain 0000 after the execution of the command.

**B TO A (B  $\rightarrow$  A, 11).** Clear the A register and then replace its four rightmost digits with the contents of the B register. The B register is not changed by this command.

**DECREASE B (B- y, 22).** If the B register before the execution of this command did not contain zero, subtract 1 from the contents of the B register and change control to storage location y. If the B register before the execution of this command contained zero, do not change control. In this case the command leaves the B register set to 9999.

**STOP (s, 08).** Halt the machine. The operator can continue his program by pressing the CONTINUOUS button.

### **Input-Output Instructions.**

**Paper Tape and Keyboard Input.** The INPUT SELECTOR switch may be set to OPTICAL READER, MECHANICAL READER (on the Flexowriter) or KEYBOARD.

**INPUT (in x 00).** Start reading the first word on the tape or received from the keyboard into storage location x. Read following words into the next consecutive locations. (See Special Use of the Sign Column below.)

**SINGLE DIGIT ADD INPUT (dA 10).** Algebraically, add to the contents of the A register in the A10 position the positive value of the number punched on the keyboard or read by the mechanical reader or photoelectric reader. When the command is executed, the machine will stop until a digit has been entered, after which it resumes operation.

**SPECIAL USE OF THE SIGN COLUMN.** Words containing numbers other than 0 or 1 in the sign column are used for control of the input

device and for B modification of words during input. A 2 or 3 placed in the sign column will cause the address part of the word to be modified by the B register as the word is read into storage. In storage, the word will have a positive sign if there was a 2 in the sign column, a negative sign if there was a 3 in the sign column. In both cases, the word will be stored with the contents of the B register added to its address part. This is summarized in Table 9.

TABLE 9. SPECIAL USE OF SIGN COLUMN, INPUTS 0 TO 3

Sign Column on Input	B Modification before Execution	B Modification on Input	Comments
0	No	No	} May be a command or part of the data
1	Yes	No	
2	No	Yes	Sign appears in storage as 0
3	Yes	Yes	Sign appears in storage as 1

An input sign 4 is used only to control input and causes the command to be read directly into the C register. The effect of each of these numbers with cu or STOP appears in Table 10.

TABLE 10. SPECIAL USE OF SIGN COLUMN, INPUTS 4 TO 7

Sign Column	Tape Stops or Keyboard is Deactivated	B Modification before Execution
4	No	No
5	No	Yes
6 <sup>a</sup>	Yes	No
7 <sup>a</sup>	Yes	Yes

<sup>a</sup> With the IN command, a 6 or 7 in the sign column does not stop the input device.

The commands most often used with an input control digit in the sign column are cu, cub, in, and stop.

**Flexowriter and Paper Tape Output.** There are two printout commands. With either of them, the second digit of the address part is a coded format instruction to the Flexowriter, Flexowriter punch or console punch. (The two punches carry along the format instruction as an information digit for the typewriter control. It does not affect format until the printing operation is carried out.) These instructions, which are sensed and carried out before any other part of the command is executed, are as follows:

<i>Second Address Digit</i>	<i>Instruction</i>
0	None.
1	Feed out one character-space of blank tape (10 to the inch). (This instruction has no effect on the Flexowriter.)
2	Print decimal point and suppress sign digit.
3	Suppress sign digit and substitute a space for sign digit when the word is printed.
4	Translate alphanumerically, two A register digits per alphanumeric character.
5	Actuate carriage return.
6	Actuate tab key.
7	Stop printout. Idle the computer if any printout command comes up before the typewriter control RESET is pressed.
8	Actuate space bar.
9	None.

**PRINT OUT (po n, 03).** This command prints out the sign and n digits of the A register (unless the sign has been suppressed by a 2 or a 3 format instruction), n being interpreted modulo 20. The R register is not affected. The A register, including the sign, is circulated  $n + 1$  places left. This operation differs from that of the shift commands in that they do not shift the sign.

The format instruction digits make it possible to control format completely by proper construction of the computer command word.

**PRINT OUT (po f, 07).** The po f command carries out the format instruction contained in the second address digit.

**PUNCH OUT (po n, 03; po f, 07).** The console punch uses the same two commands as the Flexowriter and typewriter control with the difference that the OUTPUT SELECTOR is set to TAPE. The console punch has no connection with the typewriter control except that its output tape may be an input to various arrangements of the typewriter control patch panel.

**ALPHANUMERIC CODE.** Typewriter action corresponding to the various two-digit combinations read out from the A register is given in Table 11. Alphanumeric information comes out on the console punch as pairs of decimal digits. The format instruction (4) accompanies the information so that when the tape is read for printing later it will be translated by the typewriter control into alphanumeric Flexowriter action.

Since it takes two digits to represent one alphabetic character, a po 0406, written 0.0000030406, will print out three alphabetic characters from the A register. *Example.* If  $(A) = +.7065464646$ , and the command mentioned in the above paragraph is executed, the following characters will be printed out: if6.

TABLE 11. TYPEWRITER ACTION CORRESPONDING TO VARIOUS TWO-DIGIT CODES

Typewriter Action		Alphanumeric Code	Typewriter Action		Alphanumeric Code
L.C.	U.C.		L.C.	U.C.	
a	A	20	0	)	40
b	B	61	1	1/2	41
c	C	62	2	&	42
d	D	63	3	/	43
e	E	64	4	\$	44
f	F	65	5	%	45
g	G	66	6	?	26
h	H	67	7	!	47
i	I	70	8	*	40
j	J	71	9	(	51
k	K	72	+	=	54
l	L	36	-	—	25
m	M	73	;	:	26
n	N	74	,	,	31
o	O	75			32
p	P	76	;	;	33
q	Q	77	Lower case		27
r	R	21	Upper case		30
s	S	22	Space		34
t	T	23	Color shift		35
u	U	52	Ignore		00
v	V	53	Back space		01
w	W	54	Tab		06
x	X	55	Carriage return		05
y	Y	56	Stop		07
z	Z	57			

### Punched Card Input-Output.

**CARD INPUT** ([1000 - m] ci x, 44). Read in m cards, starting at storage location x. The number of words per card, from 1 to 8, is set on a selector switch. The first three digits of the command word, excluding sign, are 1000 - m. The x is interpreted as follows:

0000 = 0000	4000 = 0000	8000 = 4000
1000 = 1000	5000 = 1000	9000 = 4000
2000 = 2000	6000 = 2000	
3000 = 3000	7000 = 3000	

After the command, the storage cells which have received new information are x through  $x + mk - 1 + (20 - k)$ , where m is the number of cards read, and k is the setting of the input WORDS PER CARD switch.

**CARD/TABULATOR OUTPUT** ([1000 - m] co x, 54). Punch out m cards, starting at storage location x. The number of words per card, from 1 to 8, is set on a selector switch. The first three digits of the command word, excluding sign, are 1000 - m. The x is interpreted as for card input. After the command, the 4000 loop contains the words transferred from  $x + mk - 1 + (20 - k)$  and the 19 preceding cells, where m is the number of cards punched, and k is the setting of the output WORDS PER CARD switch.

For tabulator output, the command is the same except that the words "line" and "printed" are substituted for the words "card" and "punched," respectively.

### Magnetic Tape.

**TAPE SEARCH** (ts x, 42). In preparation for a TAPE READ or WRITE command, search for block x on the tape unit designated by the third digit of the command word. As with word addresses on the drum, x is the last four digits of the command word. If a TAPE READ or WRITE command is fetched before the addressed block is found, skip the EXECUTE cycle of the tr or tw command and set the overflow toggle.

**TAPE READ** (tr x, 40). From the tape unit designated by the third digit of the command word, read as many consecutive blocks of 20 words each as are indicated by the first two digits of the command word. Write these words in successive storage locations beginning with storage location x.

**TAPE WRITE** (tw x, 50). On the tape unit designated by the third digit of the command word, write as many consecutive blocks of 20 words each as are indicated by the first two digits of the command word. These words will be read from successive storage locations, beginning with storage location x.

**REWIND** (rw, 52). Rewind the tape in the tape unit, designated by the third digit of the command word, to the 0000 end. If the rewind switch on the designated tape unit is in the normal position upon completion of the rewind, the unit will be locked out of the system. If the switch is in the rewind-ready position, the unit can again be called upon for a subsequent search, read, or write operation.

### Royal-McBee LGP-30 Instruction Logic

A typical small machine is the LGP-30, built by Librascope and marketed by Royal-McBee. It is desk size, its input-output is a Flexowriter, and it has a single-address instruction code with a 30 binary digit word length using digital numbers (absolute value less than one). The instruction code for this magnetic drum (30, 0, 0) binary machine is exceedingly

simple, lacking the complex optimization features of the "next instruction address" or the recirculating loop. There is only one input and one output instruction.

**List of Instructions.** The following list of commands is taken from an LGP-30 instruction manual (Ref. 66).

<i>Instruction<sup>a</sup></i>	<i>Effect</i>
<b>B m</b>	<b>Bring.</b> Clear the accumulator, and add the contents of m to it.
<b>A m</b>	<b>Add</b> contents of m to the contents of the accumulator, and retain the result in the accumulator.
<b>S m</b>	<b>Subtract</b> the contents of m from the contents of the accumulator, and retain the result in the accumulator.
<b>M m</b>	<b>Multiply</b> the number in the accumulator by the number in memory location m, and terminate the result at 30 binary places.
<b>N m</b>	<b>Multiply</b> the number in the accumulator by the number in m, and retain the least significant half of the product.
<b>D m</b>	<b>Divide</b> the number in the accumulator by the number in memory location m, and retain the rounded quotient in the accumulator.
<b>H m</b>	<b>Hold.</b> Store contents of the accumulator in m, and retain the number in the accumulator.
<b>C m</b>	<b>Clear.</b> Store contents of the accumulator in m and clear the accumulator.
<b>Y m</b>	<b>Store</b> only the <i>address</i> part of the word in the accumulator in memory location m, while leaving the rest of the word undisturbed in memory.
<b>R m</b>	<b>Return address.</b> Add one to the address held in the counter register (C) and record in the address portion of the instruction in memory location m. The counter register normally holds the address of the next instruction to be executed.
<b>E m</b>	<b>Extract,</b> or logical product order, i.e., clear the contents of the accumulator to zero in those bit positions occupied by zeros in m.
<b>U m</b>	<b>Transfer control</b> to m unconditionally, i.e., get the next instruction from m.
<b>T m</b>	<b>Test,</b> or conditional transfer. Transfer control to m only if the number in the accumulator is negative.
<b>I 0</b>	<b>Input.</b> Fill the accumulator from the Flexowriter.
<b>P x</b>	<b>Print</b> a Flexowriter symbol. The symbol is denoted by the track number part of the address (x).
<b>Z t</b>	<b>Stop.</b> Contingent on five switch ( $T_1 \cdots T_5$ ) settings on the control panel.

<sup>a</sup> The address part of the instruction is denoted by m when it refers to a memory location.

### Instruction Logic of the Soviet Strela (Arrow)

A typical general purpose digital computer using three-address instruction logic is the Strela (Arrow) constructed in quantity under the leadership of Iu. Ia. Basilevskii of the Soviet Academy of Sciences, and described in detail by Kitov (Ref. 61). This computer uses a (35, 6, 0) binary floating point number system. Its instruction word, of 43 digits, contains a six-digit operation code, and three 12-digit addresses, with one breakpoint bit. In octal notation, two digits represent the operation, four each the addresses, and one bit the breakpoint. This machine operates with up to 2048 words of high-speed cathode ray tube storage.

Input-output is ordinarily via punched cards and punched paper tape. A "standard program library" is attached to the computer as well as magnetic tape units (termed "external accumulators" below). *Note.* This computer is different from both the BESM described by Lebedev (Ref. 65) and the Ural reported by Basilevskii (Ref. 7). Apparently, it is somewhat lower in performance than BESM.

Since all arithmetic is ordinarily in floating point, "special instructions" perform fixed point computations for instruction modifications.

Ordinarily instructions are written in an octal notation, but external to the machine operation symbols are written in a mnemonic code. The notation used is similar to that described below for the EASIAC. The two-digit numerals are the octal instruction equivalent.

#### Arithmetic and Logical Instructions.

01. +  $\alpha$   $\beta$   $\gamma$ . Algebraic addition of ( $\alpha$ ) to ( $\beta$ ) with result in  $\gamma$ .
02. +<sub>1</sub>  $\alpha$   $\beta$   $\gamma$ . Special addition, used for increasing addresses of instructions. The command ( $\alpha$ ) or ( $\beta$ ) is added to the number ( $\beta$ ) or ( $\alpha$ ) and the result sent to the cell with address  $\gamma$ . As a rule, the address of the instruction being changed corresponds to the address  $\gamma$ .
03. -  $\alpha$   $\beta$   $\gamma$ . Subtraction with signed numbers. From the number ( $\alpha$ ) is subtracted the number ( $\beta$ ) and the result sent to  $\gamma$ .
04. -<sub>1</sub>  $\alpha$   $\beta$   $\gamma$ . Difference of the absolute value of two numbers  $|(\alpha)| - |(\beta)| = (\gamma)$ .
05.  $\times$   $\alpha$   $\beta$   $\gamma$ . Multiplication of two numbers ( $\alpha$ ) and ( $\beta$ ) with result sent to  $\gamma$ .
06.  $\wedge$   $\alpha$   $\beta$   $\gamma$ . Logical multiplication of two numbers in cells  $\alpha$  and  $\beta$ . This instruction is used for extraction from a given number or instruction a part defined by the special number ( $\beta$ ).
07.  $\vee$   $\alpha$   $\beta$   $\gamma$ . Logical addition of two numbers ( $\alpha$ ) and ( $\beta$ ) and sending the result to cell  $\gamma$ . This instruction is used for forming numbers and commands from parts.

10. **Sh  $\alpha \beta \gamma$ .** Shift of the contents of cell  $\alpha$  by the number of steps equal to the exponent of the ( $\beta$ ). If the exponent of the ( $\beta$ ) is positive then the shift proceeds to the left, in the direction of increasing value; if negative, then the shift is right. In addition, the sign of the number, which is shifted out of the cell, is lost.

11.  **$-_2 \alpha \beta \gamma$ .** Special subtraction, used for decreasing the addresses of instructions. In the cell  $\alpha$  is found the instruction to be transformed, and in cell  $\beta$  the specially selected number. Ordinarily addresses  $\alpha$  and  $\gamma$  are identical.

12.  **$\neq \alpha \beta \gamma$ .** Comparison of two numbers ( $\alpha$ ) and ( $\beta$ ) by means of digital additions of the numbers being compared *modulo* two. In the cell  $\gamma$  is placed a number possessing ones in those digits in which inequivalence results in the numbers being compared.

### Control Instructions.

13. **C  $\alpha \beta$  0000.** Conditional transfer of control either to instruction ( $\alpha$ ) or to instruction ( $\beta$ ), depending on the results of the preceding operation. With the operations of addition, subtraction, and subtraction of absolute values, it appraises the sign of the result: for a positive or zero result it transfers control to the command ( $\alpha$ ), for negative results to the command ( $\beta$ ).

The result of the operation of multiplication is dependent on the relationship to unity. Transfer is made to the command ( $\alpha$ ) in the case where the result is greater than or equal to one, and to command ( $\beta$ ), if it is smaller than one.

For conditional transfer after the operation of comparison, transfer to the instruction ( $\alpha$ ) is made in the case of equality of binary digits, and to ( $\beta$ ) when there is any inequivalence.

After the operation  $\wedge$  (logical sequential multiplication) the conditional transfer command jumps to the instruction ( $\alpha$ ) when the result is different from zero, and to instruction ( $\beta$ ) when it is equal to zero.

A forced comparison is given by

**C  $\alpha \alpha$  0000**

The third address in this command is not used and in its place is put zero.

14. **I-O  $\alpha$  0000 0000.** This instruction is executed parallel with the code of the other operations, and guarantees bringing into working position in good time the zone of the external accumulator (magnetic tape unit) with the address  $\alpha$ .

15. **H 0000 0000 0000.** This instruction executes an absolute halt.

**Group Transfer Instructions.** Special instructions for group transfer serve for the accomplishment of a transfer of numbers to and from the

accumulators. In the second address in these instructions stands an integer, designating the quantity of numbers in the group which must be transferred. Group transfers always are produced in increasing sequence of addresses of cells in the storage.

**16.  $T_1$  0000 n  $\gamma$ .** The instruction  $T_1$  guarantees transfer from a given input unit (with punched cards, perforated tape, etc.) into the storage. In the third address  $\gamma$  of the instruction is indicated the initial address of the group of cells in the storage where numbers are to be written. With punched paper tape or punched cards the variables are written in sequence, beginning with the first line.

**17.  $T_2$  0000 n  $\gamma$ .** The instruction  $T_2$  guarantees transfer of a group of  $n$  numbers from an input unit into the external accumulator in zone  $\gamma$ .

**20.  $T_3$   $\alpha$  n  $\gamma$ .** This instruction guarantees a line-by-line sequence of transfers of  $n$  numbers from zone  $\alpha$  of the external accumulator into the cells of the storage beginning with the cell with address  $\gamma$ .

**21.  $T_4$   $\alpha$  n 0000.** This instruction guarantees the transfer to the input-output unit (to punched paper tape or punched cards) of a group of  $n$  numbers from the storage, beginning with address  $\alpha$ . The record on punched paper tape or punched cards as a rule will begin with the first line and therefore a positive indication of the addresses of the record is not required.

**22.  $T_5$   $\alpha$  n  $\gamma$ .** Instruction  $T_5$  guarantees transfer of a group of  $n$  numbers from one place in the storage with initial address  $\alpha$  into another place in the storage with initial address  $\gamma$ .

**23.  $T_6$   $\alpha$  n  $\gamma$ .** Instruction  $T_6$  guarantees transfer of a group of  $n$  numbers from the storage with initial address  $\alpha$  into the external accumulator with address  $\gamma$ .

**24.  $T_7$   $\alpha$  n 0000.** Instruction  $T_7$  serves for transfer of  $n$  numbers from the zone of the external accumulator with address  $\alpha$  into the input-output unit.

Instructions  $T_2$  and  $T_7$  cannot be performed concurrently with other machine operations.

**Standard Subroutine Instructions.** Certain instructions in the Strela, although written as ordinary instructions are actually "synthetic" instructions which call on a subroutine for computation of the function involved. The amount of machine time (number of basic instruction cycles) for an iterative process depends on the required precision of the computed function. The figures given below are based on approximately ten-digit decimal numbers with desired precision one in the tenth place.

**25. D  $\alpha$   $\beta$   $\gamma$ .** This standard subroutine serves for execution of the operation of division: The number ( $\alpha$ ) is divided into the number ( $\beta$ ) and the quotient is sent to cell  $\gamma$ .

The actual operation of division is executed in two steps: the initial obtaining of the value of the inverse of the divisor, by which the dividend is then multiplied. The computation of the inverse is given by the usual Newton formula, originally used with the EDSAC (Ref. 108).

$$y_{n+1} = y_n(2 - y_n x).$$

For  $x = d \cdot 2^p$ , where  $\frac{1}{2} < d < 1$ , the first approximation is taken as  $2^{-p}$ . The standard subroutine takes 8 to 10 instructions and can be executed in 18–20 machine cycles (execution time for one typical command).

**26.  $\sqrt{\phantom{x}}$  a 0000  $\gamma$ .** This instruction guarantees obtaining the value  $\sqrt{x}$  from the value  $x = (\alpha)$  and sending the result to cell  $\gamma$ . Initially  $1/\sqrt{x}$  is computed by the iteration formula

$$y_{n+1} = \frac{1}{2}y_n(3 - xy_n^2),$$

where the first approximation is taken as

$$y_0 = 2^{\lfloor p/2 \rfloor},$$

the bracket indicating “integral part of.” After this the result is multiplied by  $x$  to obtain  $\sqrt{x}$ . This standard subroutine contains 14 instructions and is executed in 40 cycles.

**27.  $e^x$  a 0000  $\gamma$ .** This instruction guarantees formation of  $e^x$  for the value  $x = (\alpha)$  and sending the result to cell  $\gamma$ . The computation is produced by means of expansion of  $e^x$  in a power series. The standard subroutine contains 20 instructions and is executed in 40 cycles.

**30.  $\ln x$  a 0000  $\gamma$ .** This instruction guarantees formation of the function  $\ln x$  for the value  $x = (\alpha)$  and sending the result to location  $\gamma$ . Computation is produced by expansion of  $\ln x$  in series. The subprogram contains 15 instructions and is executed in 60 cycles.

**31.  $\sin x$  a 0000  $\gamma$ .** This instruction guarantees execution of the function  $\sin x$  and sending the result to location  $\gamma$ . The computation is produced in two steps: initially the value of the argument is translated into the first quadrant, then the value of the function is obtained by a series expansion. The subroutine contains 18 instructions and is executed in 25 cycles.

**32. DB a n  $\gamma$ .** This instruction performs conversion of a group of  $n$  numbers, stored in locations  $\alpha, \alpha + 1, \dots$  from binary-coded decimal into binary and sending of the result to locations  $\gamma, \gamma + 1, \dots$ . The subroutine contains 14 instructions and is executed in 50 cycles (for each number).

**33. BD a n  $\gamma$ .** This instruction performs the conversion of a group of  $n$  numbers stored in locations  $\alpha, \alpha + 1, \dots$  from the binary system into binary-coded decimal and sends them to locations  $\gamma, \gamma + 1, \dots$ .

The subroutine contains only 30 instructions and is executed with 100 cycles (for each number).

**34. MS  $\alpha$  n  $\gamma$ .** This is an instruction for storage summing. This instruction produces the formal addition of numbers, stored in locations beginning with address  $\alpha$ , and the result is sent to location  $\gamma$ . Numbers and instructions are added in fixed point. This sum may be compared with a previous sum for control of storage accuracy.

### Instruction Logic of the MIDAC

The MIDAC, Michigan Digital Automatic Computer (Ref. 25), was constructed on the basis of the design of the SEAC at the National Bureau of Standards. Its instruction code is particularly of interest because it incorporates the index register concept into a three-address binary instruction. Numbers in this machine are (44, 0, 0) fixed points. The word length is 45 binary digits with serial operation.

**Word Structure.** The data or address positions of an instruction are labeled the  $\alpha$ ,  $\beta$ , and  $\gamma$  positions. Each contains twelve binary digits represented externally as three hexadecimal digits. Four binary digits, or one hexadecimal digit, are used to convey the instruction modification or relative addressing information. The next four binary digits or single hexadecimal digit represents the operation portion of the instruction. The final binary digit is the halt or breakpoint indicator for use with the instruction.

For example, the 45-binary-digit word,

000001100100000011001000000100101100000001011

considered as an instruction would be interpreted as

$\alpha$	$\beta$	$\gamma$	abcd	Op	halt
000001100100	000011001000	000100101100	0000	0101	1

In external hexadecimal form this would be written

**064 0c8 12c 0 5 -**

The above binary word is the equivalent machine representation of the following instruction: "Take the contents of hexadecimal address 064, add to it the contents of hexadecimal address 0c8, and store the result in hexadecimal address 12c. There is *no* modification of the 12-binary-digit address locations given by the instruction. Upon completion of the operation, stop the machine if the proper external switches are energized." The binary combination represented by 5 is the operation code for addition.

**Data or Addresses.** The addresses given by the twelve binary digits in each of the three locations designate in the machine the individual acoustic storage cells and blocks of eight magnetic drum storage cells. The

addresses from 0 to 1023 (decimal) or 000 to 3FF (hexadecimal) correspond to acoustic storage cells. The addresses from 1024 to 4095 (decimal) or 400 to FFF (hexadecimal) correspond to magnetic drum storage blocks. In certain operations, however, the addresses 0 to 15 (decimal) or 0 to F (hexadecimal) represent input-output stations rather than storage locations.

These twelve-binary-digit groups will in some cases be modified by the machine in order to yield a final twelve-binary-digit address. The method of processing will depend on the values of the instruction modification digits. After modification, the final result will then be interpreted by the control unit as a machine address.

In some instructions, namely those that perform change of control operations, which involve cycling and counting rather than simple arithmetic operations on numbers, the  $\alpha$  and  $\beta$  positions in an instruction are not considered as addresses. In those cases, they are used instead as counters or tallies. In other instructions, which do not require three addresses, but only one or two, the  $\beta$  position is not considered as an address. In these cases, the oddness or evenness of the  $\beta$  address is used to differentiate between two operations having the same operation code digits. That is, the parity of binary digit P22 is used as an extra function designator.

**Instruction Modification Digits.** The four binary digits P9-P6 are used as instruction modification or relative addressing digits. Their normal function is relatively simple; nevertheless, the possible exceptions to the general rule can make their behavior complicated. These four digits are labeled the a, b, c, and d digits. Ordinarily the a digit is associated with the  $\alpha$  position, the b digit with the  $\beta$  position, and the c digit with the  $\gamma$  position in an instruction.

When binary digit P22 (or the  $\beta$  position) is used in an instruction to represent extra operation information, the instruction modification digit b is ignored. In the case of input and output instructions, when the various address positions represent machine address locations on the drum, input-output stations, or block lengths, and modification of these addresses is not desired in any case, the corresponding relative addressing digits are ignored.

The purpose of the instruction modification digits is to tell the machine whether or not to modify the twelve binary digits making up the corresponding address position in an instruction by addition of the contents of one or the other of two counters. In the normal case, if the a, b, or c digit is a zero, the twelve binary digits in the corresponding position are interpreted, unchanged, as the binary representation of the machine address of the number word to be processed by the instruction.

If one or more of the a, b, or c digits is a one, the contents of one of two

auxiliary address counters is added to the corresponding twelve binary digits to yield a final address usually different from that given by the original twelve-digit portion of the instruction word. The addresses are then said to be relative to the counter.

The two counters involved in the address modification feature of the MIDAC are known as the instruction counter and the base counter. In the normal case, if the fourth instruction modification or d digit is a zero, the contents of the instruction counter will be added to the contents of the various twelve-digit addresses (dependent on the values of the a, b, and c digits) before further processing of the instruction. If the a digit is one and the d digit zero, the contents of the instruction counter will be added to the  $\alpha$  address; similarly for b and d digits and  $\beta$  address, etc.

If the d digit is a one, the contents of the base counter will be normally added to the contents of the twelve digits in the  $\alpha$ ,  $\beta$ , and  $\gamma$  positions (again dependent on the values of the a, b, and c digits), before further processing of the results. If the a digit is one and the d digit one, the contents of the base counter will be added to the  $\alpha$  address, etc.

The effect of the instruction modification digits may be summarized as follows:

The contents of the two counters will be designated by  $C_d$  ( $d = 0, 1$ ).

$C_0$  = contents of the instruction counter,

$C_1$  = contents of the base counter.

Then the modified addresses  $\alpha'$ ,  $\beta'$ , and  $\gamma'$  are related to the  $\alpha$ ,  $\beta$ , and  $\gamma$  addresses appearing in the instruction by the following:

$$\alpha' = \alpha + aC_d, \quad \beta' = \beta + bC_d, \quad \gamma' = \gamma + cC_d \quad (a, b, c, d = 0, 1).$$

In certain instructions addresses relative to one of the two counters may be prohibited. Thus, if in a particular instruction  $\alpha$  may be relative only to the instruction counter, then for that instruction

$$\alpha' = \alpha + aC_0,$$

no matter whether the d digit is a 0 or a 1.

The notation  $(\alpha')$ ,  $(\beta')$ , or  $(\gamma')$  is used to indicate the word stored in the location whose address is  $\alpha'$ ,  $\beta'$ , or  $\gamma'$ .

**Instruction Counter.** The instruction counter is a twelve-binary digit (modulo 4096) counter which contains the binary representation of the address of the instruction which the control unit is processing or is about to process. In normal operation when no change of control operation is being processed, the contents of the instruction counter is increased by one at the completion of each instruction. Thus, normally the next instruction to

be processed is stored in the acoustic storage cell immediately following the cell which contains the present instruction.

A change of control operation is one which selects a next instruction not stored in sequence in the acoustic storage. That is, at the completion of such instructions the contents of the instruction counter is not increased by one, but instead is changed entirely.

**Base Counter.** The base counter is a second twelve-binary-digit counter (modulo 4096), physically identical to the instruction counter, which contains the binary representation of a base number or tally. Unlike the instruction counter, however, the base counter does not sequence automatically, but remains unchanged until a change of base instruction is processed. This counter serves two primary purposes, dependent on the usage to which it is put:

1. It may contain the address of the initial word in a group, thus serving as a base address to which integers representing the relative position of a given word in the group of words may be added by using the address modification digits.

2. It may contain a counter or tally which can be increased by a base instruction. This instruction makes use of the address modification digits to change the counter so as to count the number of traversals of a particular cycle of instructions.

**Instruction Types.** Instructions used in MIDAC can be divided into three categories: change of information, change of control, and transfer of information. The first category can be further subdivided into arithmetic and logical instructions. In the arithmetic instructions are included addition, subtraction, division, various forms of multiplication; power extraction, number shifting; and number conversion instructions. The sole logical instruction is extract, which modifies information in a non-arithmetic fashion.

The transfer of information or data transfer instructions include transfers of individual words or blocks of words into and out of the acoustic storage and drum and magnetic tape control.

The possible change of control instructions includes two comparisons that provide different future sequences dependent on the differences of two numbers. In the compare numbers or algebraic comparison, the difference is an algebraic, signed one. In the compare magnitudes or absolute comparison, the difference is one between absolute values. Two other instructions, file and base, perform other tasks beside transferring control. The file instruction transfers control unconditionally. The file instruction files or stores the contents of the base or instruction counter in a specific address position of a particular word in the storage. The base or tally instruction

provides a method for referring addresses automatically relative to the address given by the base counter, irrespective of its contents. The base instruction also gives a conditional transfer of control.

The nineteen MIDAC instructions can be described functionally as follows:

### Change of Information.

1. **Add.**  $(\alpha') + (\beta')$  is placed in  $\gamma'$ . Result must be less than 1 in absolute value.

2. **Subtract.**  $(\alpha') - (\beta')$  is placed in  $\gamma'$ . Result must be less than 1 in absolute value.

3. **Multiply, Low Order.** The least significant 44 binary digits of  $(\alpha') \times (\beta')$  are placed in  $\gamma'$ .

4. **Multiply, High Order.** The most significant 44 binary digits of  $(\alpha') \times (\beta')$  are placed in  $\gamma'$ .

5. **Multiply, Rounded.** The most significant 44 binary digits of  $(\alpha') \times (\beta') \pm 1 \cdot 2^{-45}$  are placed in  $\gamma'$ . The  $1 \cdot 2^{-45}$  is added if  $(\alpha') \times (\beta')$  is positive, and subtracted if  $(\alpha') \times (\beta')$  is negative.

6. **Divide.** The most significant 44 binary digits of  $(\beta')/(\alpha')$  are placed in  $\gamma'$ . (Note the inversion of order of  $\alpha$  and  $\beta$ .) Result must be less than 1 in absolute value.

7. **Power Extract.** The number  $n \cdot 2^{-44}$  is placed in  $\gamma'$  where  $n$  is the number of binary 0's to the left of the most significant binary 1 in  $(\alpha')$ . The  $b$  digit is ignored;  $\beta$  may be any even number. If  $(\alpha')$  is all zeros, zero is placed in  $\gamma'$ .

8. **Shift Number.** The 44 binary digits immediately to the right of the radix point in  $(\alpha') \cdot 2^{(\beta')} \cdot 2^{44}$  are placed in  $\gamma'$ . The result, in  $\gamma'$ , is the equivalent of shifting  $(\alpha')$   $n$  places, where  $n \cdot 2^{-44} = (\beta')$  and  $n$  positive indicates a shift left,  $n$  negative a shift right. If  $|n| \geq 44$ , zero is placed in  $\gamma'$ .

9. **Extract or Logical Transfer.** Those binary digits in  $(\gamma')$ , including the sign digit, whose positions correspond to 1's in  $(\beta')$  are replaced by the digits in the corresponding positions of  $(\alpha')$ .

10. **Decimal to Binary Conversion.** This operation may be interpreted in two ways: (a)  $(\alpha')$  is considered as a binary-coded-decimal integer times  $2^{-44}$ . It is converted to the equivalent binary integer times  $2^{-37}$  and the result is placed in  $\gamma'$ , or (b)  $(\alpha')$  is considered as a binary-coded-decimal fraction,  $D$ . It is converted into an intermediate binary fraction,  $B_i$ , such that  $B_i = D \times 10^{11} \times 2^{-37}$  and the result placed in  $\gamma'$ . To obtain  $B$ , the true binary equivalent of  $D$ ,  $B_i$  must be multiplied by  $(10^{-11} \times 2^{37})$ . However, since this factor is greater than 1 and therefore

cannot be represented in the machine, two operations must be performed. For example,

$$B_i \times (10^{-11} \times 2^{37} - 1) = B_j,$$

$$B = B_i + B_j.$$

Here the  $b$  digit is ignored, and  $\beta$  may be any *even* number.

**11. Binary-to-Decimal Conversion.** ( $\alpha'$ ), considered as a binary fraction, is converted into the equivalent eleven-digit binary-coded-decimal fraction. The result is placed in  $\gamma'$ . The  $b$  digit is ignored, and  $\beta$  may be any odd number.

### Change of Control.

**12. Compare Numbers.**  $\gamma$  can be relative only to the instruction counter. If  $(\alpha') \geq (\beta')$ , the contents of the instruction counter are increased by one as is normally done at the end of each instruction. If  $(\alpha') < (\beta')$ , the contents of the instruction counter are set to  $\gamma'$ .

**13. Compare Magnitudes.**  $\gamma$  can be relative only to the instruction counter. If  $|(\alpha')| \geq |(\beta')|$ , the contents of the instruction counter are increased by one as is normally done at the end of each instruction. If  $|(\alpha')| < |(\beta')|$ , the contents of the instruction counter is set to  $\gamma'$ .

**14. Base or Tally.** The  $d$  digit is ignored.  $\alpha$  and  $\beta$  may be relative only to the base counter,  $\gamma$  only to the instruction counter. If  $\alpha' \geq \beta'$ , the contents of the base counter are set to zero and the contents of the instruction counter increased by one as usual. If  $\alpha' < \beta'$ , the contents of the base counter are set to  $\alpha'$  and the contents of the instruction counter to  $\gamma'$ . (*Note.* The comparisons made here are of addresses themselves, not their contents.)

**15. File.**  $\beta$  may be any odd number.  $\alpha$  and  $\gamma$  may be relative only to the instruction counter.

If  $d = 0$ , the contents of the instruction counter increased by one is placed in the  $\gamma$  position of  $(\alpha')$ , and the instruction counter is set to  $\gamma'$ .

If  $d = 1$ , the contents of the base counter is placed in the  $\alpha$  position of  $(\alpha')$ , and the instruction counter is set to  $\gamma'$ . In addition, if  $b = 1$ , the contents of the base counter is set to zero; if  $b = 0$ , the contents of the base counter is not changed.

### Transfer of Information.

**16. Read In.** The  $a$  digit must be 0; the  $b$  digit is ignored. If  $\beta$  is in the range 0 to 7 (decimal) or 000 to 007 (hexadecimal)  $\alpha$  words are read into the acoustic storage from input-output station  $\beta$ . The first word read

in is placed in  $\gamma'$ , the second in  $\gamma' + 1$ , etc. If  $\beta$  is in the range 1024 to 1791 decimal (400 to 6FF hexadecimal),  $\alpha$  words are read into the acoustic storage from the drum starting with the first word in the drum block whose address is  $\beta$ . The first word is placed in  $\gamma'$ , the second in  $\gamma' + 1$ , etc.

**17. Read Out.** The a digit must be 0, the c digit is ignored. Starting with ( $\beta'$ ), read out  $\alpha$  consecutive words from the acoustic storage to input-output station  $\gamma$ , if  $\gamma$  is in the range 0 to 7 decimal (000 to 007 hexadecimal), or to the drum starting at the beginning of the drum block whose address is  $\gamma$ , if  $\gamma$  is in the range 1024 to 1791 decimal (400 to 6FF hexadecimal).

**16. Alphanumeric Read In.** The a digit must be 1; the b digit is ignored. If  $\beta$  is in the range 0 to 7 (decimal) or 000 to 007 (hexadecimal)  $\alpha$  characters are read into the acoustic storage from input-output station  $\beta$ . The first character read in is placed in  $\gamma'$ , the second in  $\gamma' + 1$ , etc. Each character occupies the six most significant digit positions of the register into which it is read; the other positions are set to zero. This operation may not be used to read words from the drum into the acoustic storage.

**17. Alphanumeric Read Out.** The a digit must be 1; the c digit is ignored. Starting with ( $\beta'$ ), read out  $\alpha$  consecutive characters from the acoustic storage to input-output station  $\gamma$ ;  $\gamma$  must be in the range 0 to 7 (decimal) or 000 to 007 (hexadecimal). This operation may not be used to read words from the acoustic storage onto the drum.

**18. Move Tape Forward.** (a, b, c, and d digits are ignored.)  $\beta$  may be any *even* number;  $\gamma$  must be in the range 0 to 15 decimal (000 to 00F hexadecimal). The magnetic tape at input-output station  $\gamma$  is moved forward  $n$  blocks where

$$n = \left[ \frac{\alpha - 1}{8} \right] + 1,$$

that is, one plus the integral part of  $\alpha - \frac{1}{8}$ , or the number of blocks that include  $\alpha$  words.

**19. Move Tape Backward.** (a, b, c, and d digits are ignored.)  $\beta$  may be any *odd* number;  $\gamma$  must be in the range 0 to 15 decimal (000 to 00F hexadecimal). The magnetic tape at input-output station  $\gamma$  is moved backward  $n$  blocks where

$$n = \left[ \frac{\alpha - 1}{8} \right] + 1,$$

that is, one plus the integral part of  $\alpha - \frac{1}{8}$ , or the number of blocks that include  $\alpha$  words.

### EASIAC Instruction Logic

As an example of a typical three-address machine a description of the EASIAC (Easy Instruction Automatic Computer) is given here. This machine, an abstraction, made use of the MIDAC (Michigan Digital Automatic Computer) as the host computer. EASIAC is an interpretive routine (see Sect. 13), and is included as an example of that form of automatic programming. Its internal structure has been described thoroughly by Perkins (Ref. 80). It performs its arithmetic operations on floating point numbers (actually binary, but printed out in standard decimal form with properly located decimal point. If any one of eight specific programming errors occurs (given by the codes listed below under EASIAC Error Printout), then a printout occurs automatically. Such errors as division by zero, taking the square root of a negative number, or using an operand of the wrong type of information (an instruction) are some of the causes of such printouts. In addition the contents of seven index registers are printed, along with a "jump table" listing the instructions where transfers of control were made, and how many times each loop was performed.

The addresses in the three-address instructions are symbolic or floating addresses. The first two digits in an address give the position in a region for which the letter and last two digits are the name. This "computer" was used for two years for undergraduate instruction purposes at the University of Michigan without any kind of error diagnosis other than the automatic printouts described.

The EASIAC reads all the alphabetical and numerical characters listed, and, as noted in the Summary of Operations (see Table 12), contained very thorough alphabetical input-output instructions with punched paper tape high-speed input and typewriter and high-speed punch output.

A demonstration problem, calculation of a polynomial for a number of values, is included. The notation "fa--a01" indicates that the given instruction or number has been assigned the floating address "a01." The signal "end" notes the end of information to be read in. Index accumulator (tally) modification is noted by the suffix ( $T_i$ ) added on to an address, where  $i$  is the tally number.

**SYMBOLS.**

<	less than
>	greater than
$\geq$	greater than or equal to
$\neq$	is not identical to
	absolute value
$\alpha$	the cell whose floating address appears in the first component of an instruction
$\beta$	the cell whose floating address appears in the second component of an instruction
$\gamma$	the cell whose floating address appears in the third component of an instruction
$\alpha'$	the cell whose address is obtained by modifying the address of $\alpha$ by the contents of the proper tally
$\beta'$	the cell whose address is obtained by modifying the address of $\beta$ by the contents of the proper tally
$\gamma'$	the cell whose address is obtained by modifying the address of $\gamma$ by the contents of the proper tally
$(\alpha')$	the contents of $\alpha'$
$(\beta')$	the contents of $\beta'$
$(\gamma')$	the contents of $\gamma'$
$\rightarrow$	becomes the new contents of
$C_i$	instruction sequencer

**Characters.**

0	G	g	W	w	—
1	H	h	X	x	-
2	I	i	Y	y	:
3	J	j	Z	z	;
4	K	k	&		
5	L	l	/		$\frac{1}{2}$
6	M	m	\$		$\frac{1}{4}$
7	N	n	%		$\frac{3}{4}$
8	O	o	?		(Back space)
9	P	p	!		(Space)
A	a	Q	q	*	(Upper case shift)
B	b	R	r	(	(Lower case shift)
C	c	S	s	)	(Tab)
D	d	T	t	"	(Carriage return)
E	e	U	u	'	(Color shift)
F	f	V	v	¢	(Back space)

TABLE 12. EASIAC, SUMMARY OF OPERATIONS

Operation Code	Operation	Symbolic Notation
<b>add</b>	Add	$(\alpha') + (\beta') \rightarrow \gamma'$
<b>sub</b>	Subtract	$(\alpha') - (\beta') \rightarrow \gamma'$
<b>mul</b>	Multiply	$(\alpha') \times (\beta') \rightarrow \gamma'$
<b>div</b>	Divide	$(\alpha') \div (\beta') \rightarrow \gamma'$
<b>xfr</b>	Transfer	$(\alpha') \rightarrow \gamma'$
<b>cmp</b>	Compare	if $(\alpha') \geq (\beta')$ : $(C_i) + 1 \rightarrow C_i$ if $(\alpha') < (\beta')$ : $\gamma' \rightarrow C_i$
<b>cav</b>	Compare absolute values	if $ (\alpha')  \geq (\beta')$ : $(C_i) + 1 \rightarrow C_i$ if $ (\alpha')  < (\beta')$ : $\gamma' \rightarrow C_i$
<b>jmp</b>	Jump	$\gamma' \rightarrow C_i$
<b>lev</b>	Leave	$\gamma' \rightarrow C_i$
<b>ret</b>	Return	
		Address of order immediately following last unpaired lev order $\rightarrow C_i$
<b>set-x</b>	Set tally x	$(\alpha') \rightarrow T_x$
<b>ndx-x</b>	Index tally x	$(\alpha') + (T_x) \rightarrow T_x$
<b>eye-x</b>	Cycle tally x	if $(T_x) + 1 \geq (\beta')$ : $0 \rightarrow T_x$ , $(C_i) + 1 \rightarrow C_i$ if $(T_x) + 1 < (\beta')$ : $(T_x) + 1 \rightarrow T_x$ , $\gamma' \rightarrow C_i$
<b>fil-x</b>	File tally x	$(T_x) \rightarrow \gamma'$
<b>stp</b>	Stop	Halt computer. Upon pushing start button, process next instruction and continue
<b>pno</b>	Print out numbers	Print $(\alpha')$ numbers starting at $\beta'$ with $(\gamma')$ digits to the right of the decimal point. Print carriage returns after each number but the last
<b>rno</b>	Read in numbers	Read in $(\alpha')$ numbers from station $(\beta')$ and store, starting at $\gamma'$
<b>pch</b>	Print out alphanumeric characters	If $(\alpha')$ is a number, print $(\alpha')$ characters starting at $\gamma'$ If $(\alpha')$ is a character, print characters starting at $\gamma'$ to, but not including, the first occurrence of $(\alpha')$
<b>rch</b>	Read in alphanumeric characters	If $(\alpha')$ is a number, read in $(\alpha')$ characters from station $(\beta')$ and store, starting at $\gamma'$ If $(\alpha')$ is a character, read characters from station $(\beta')$ and store, starting at $\gamma'$ to, and including, the first occurrence of $(\alpha')$
<b>sqr</b>	Square root	$\sqrt{(\alpha')} \rightarrow \gamma'$
<b>s-c</b>	Sine-cosine	$\sin(\alpha') \rightarrow \beta'$ , $\cos(\alpha') \rightarrow \gamma'$
<b>atn</b>	Arctangent	$\arctan \frac{(\alpha')}{(\beta')} \rightarrow \gamma'$

**EASAC Error Printout.**

<i>Code No.</i>	<i>Error</i>
1.	Trying to interpret a noninstruction
2.	More than 5 unpaired lev instructions A ret instruction with no unpaired lev preceding In a simple loop more than 250 times
3.	Attempt to jump to, or to obtain an instruction or operand from a nonexistent cell (i.e., a cell before the first instruction or more than 250 cells beyond the first)
4.	Attempt to divide by 0, take the arctan of 0/0, or take the square root of a negative number
5.	Not used
6.	Not used
7.	Operand not right type of information
8.	Operand not integer, or not in required range
9.	Attempt to read, print, or generate a number too large
10.	Attempt to put result or to read in to a cell already containing an instruction

002a07		Address where computation stopped
8		Error code (see above)
2		Contents of: T <sub>1</sub>
0		“ “ T <sub>2</sub>
0		“ “ T <sub>3</sub>
1		“ “ T <sub>4</sub>
0		“ “ T <sub>5</sub>
0		“ “ T <sub>6</sub>
0		“ “ T <sub>7</sub>
000a00	004a00	1
001a00	000b21	14
000a07		Jump table

**Demonstration Problem.**

*Problem.* Evaluate  $f(x) = a_0x^4 + a_1x^3 + a_2x^2 + a_3x + a_4$  for  $x = 1, 2, 3, \dots, 25$ .

After the 25 values of  $x$  have been evaluated, print out the 25 values of  $f(x)$ .

The values of the coefficients are:

$$a_0 = 1, \quad a_1 = 1.5, \quad a_2 = 0.02, \quad a_3 = -23, \quad a_4 = -56.$$

$$f(x) = x^4 + 1.5x^3 + 0.02x^2 - 23x - 56.$$

For easier computation let

$$f(x) = \{(a_0x + a_1)x + a_2\}x + a_3\}x + a_4.$$

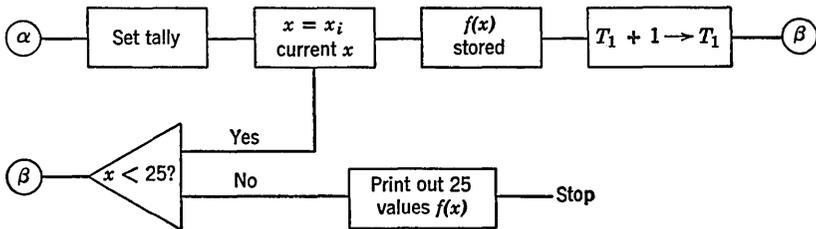
*Analysis.* For one value of  $x$  the problem would consist of:

1. Calculating  $a_0x$ .
2. Calculating  $a_0x + a_1$ .
3. Calculating  $(a_0x + a_1)x$ .
4. Calculating  $(a_0x + a_1)x + a_2$ .
5. Calculating  $[(a_0x + a_1)x + a_2]x$ .
6. Calculating  $[(a_0x + a_1)x + a_2]x + a_3$ .
7. Calculating  $\{(a_0x + a_1)x + a_2\}x + a_3\}x$ .
8. Calculate  $f(x) = \{(a_0x + a_1)x + a_2\}x + a_3\}x + a_4$ .

*Additional Steps.*

9. Setting any tallies needed.
10. Reading in a new value of  $x$  for the iterative process.
11. Storing the value of  $f(x)$ .
12. Counting the number of times an iteration takes place.
13. Printing out the 25 values of  $f(x)$ .
14. Stopping the computer.
15. Writing *all* the constants and temporaries which will be needed during computation.

*Flow Diagram.*



PROGRAM FOR EVALUATION OF A POLYNOMIAL

Instructions					
fa--a00	c02	0	0	set-1	$0 \rightarrow T_1$
fa--a01	c00*(T <sub>1</sub> )	c02	d00	add	$x = x_i$ , current $x$
	d00	b00	e00	mul	$a_0x$
	e00	1b00	e00	add	$a_0x + a_1$
	e00	d00	e00	mul	$(a_0x + a_1)x$
	e00	2b00	e00	add	$(a_0x + a_1)x + a_2$
	e00	d00	e00	mul	$\{(a_0x + a_1)x + a_2\}x$
	e00	3b00	e00	add	$\{(\quad) + a_2\}x + a_3$
	e00	d00	e00	mul	$\{\{\quad\}x + a_3\}x$
	e00	4b00	e00	add	$f(x) = \{\quad\}x + a_4$
	e00	0	c01*(T <sub>1</sub> )	xfr	$f(x) \rightarrow c00*(T_1)$
	1c02	0	0	ndx-1	$1 + T_1 \rightarrow T_1$
	d00	2c02	a01	cmp	$x < 25?$ Yes, to a01; no, go on
	2c02	c01	1c02	pno	Print out 25 values of $f(x)$ beginning with cell c01.
Numbers	0	0	0	stp	Stop computer
fa--b00	1				$a_0$
	1.5				$a_1$
	.02				$a_2$
	-23.				$a_3$
	-56.				$a_4$
fa--c00	1				} coefficients  } values of $x$
	2				
	3				
	4				
	5				
	6				
	7				
	8				
	9				
	10				
	11				
	12				
	13				
	14				
	15				
	16				
	17				
	18				
	19				
	20				
	21				
	22				
	23				
	24				
	25				
fa--c01					} Answers $f(x)$
empty--24					} 25 temporaries
fa--c02	0				Constant 0
	1				Constant 1
	25				Constant 25
fa--d00					Current $x$ , empty cell
fa--e00					Partial answers, empty cell
end					

## 7. TRADITIONAL PROGRAMMING TECHNIQUES

**Method.** The traditional hand-programming method for a scientific or engineering problem for a high-speed digital computer occurs in the following sequence:

1. Selection of a numerical method of solution, a priori appraisal of errors, selection of finite difference step size, and decision as to digit length of numbers (single or multiple precision).

2. Preparation of a flow diagram using the symbology of Sect. 4 or a similar one.

3. Static translation from the flow diagram into a sequence of instructions and listing of the constants, both in the original language of the machine (generally octal or hexadecimal notation for binary machines, decimal or alphanumeric decimal for equipment using that machine notation).

4. Entry into the machine of this sequence of information, now considered merely as a string of machine *words*, on punched cards or punched paper tape.

5. Checking or *debugging* of the written procedure by comparison of contents of the machine registers during and after performance of the problem, with previously obtained partial results computed by hand.

6. Upon obtaining deviations between the hand-computed results and those read out on lights or by printer from the machine, a complete search of the pertinent portion of the program to determine the error or errors.

7. Correction of the errors by changing the set of coded instructions, with or without the corresponding change in the flow diagram, and then a return through steps 3 through 7 until all results check with the hand-computed values. (Mistakes may occur in the hand computation.)

8. Upon complete satisfaction that the program performs as it should, entry of supplementary parameters in machine language and performance of all necessary cases.

**Programming Errors.** The above sequence is precisely that of Fig. 1, Sect. 1. However, the repetition of steps 3 through 7 are the most routine, detailed, and time-consuming part of the process. These steps are most prone to error and at the same time require the lowest level of basic skills. One minor mistake in transcription, hand conversion, or data punching, if not caught, can cause major mistakes in output, or a frustrating search requiring vast outlays of programmer and computer time.

Once a programming error is discovered, instructions must be changed, and often inserted or deleted. In the latter case, succeeding instructions will acquire new addresses, and any instructions referring to them must be changed to refer to the new address. One minor error can therefore cause a chain reaction of corrections. This may be avoided by patching or

inserting transfer of control instructions to remote unused locations, where insertions may be made without complete renumbering of addresses. Similar insertions of transfer of control instructions can be used to accomplish deletions. These procedures tend to cause further programming errors.

### **Physical Restrictions on Programming**

One ever present set of restrictions on digital computer programming is that set of measures of magnitude (human effort, elapsed time, computer time, computer storage) that describe the programming process and its relation to the external practical world. Unfortunately, evaluation of most of these measures is mainly a matter of experience.

**Human Time.** No formulas are available that can predict the amount of human time required to program a particular problem, given an original general description of a problem. Experience indicates that such time estimates are usually underestimated. The advent of automatic programming has generally decreased the amount of programmer time needed.

If an estimate of the (static) number of instructions is available (this may be obtained by comparison with previously written programs), then an estimate of human cost (and therefore time) may be based on the common estimate for cost of hand-coded programs of \$5.00 per checked out instruction. This figure compares very unfavorably, of course, with corresponding programming costs using the translator-compilers of the IT, Fortran, Math-matic types.

**Elapsed Time.** Overall elapsed time is a function of the previously discussed variables and is particularly a function of the machine aids to the programming process available. Regular routine program debugging procedures, such as described by Pietrasanta (Ref. 82), can aid markedly in decreasing elapsed time. Combined use of translators with hand-coded insertions, if easily available (such as with the IT system, see Sect. 12), may cut elapsed programming time markedly.

General discussions of the programming process from this point of view are available, for scientific problems, in Carr (Ref. 20), and for data processing problems, in Gottlieb and Hume (Ref. 113).

**Computer Time.** The original estimate was made by Burks, Goldstine, and von Neumann (Ref. 19) that in most scientific problems the multiplication time of a computer would be the dominant factor, and therefore an estimate of the number of such operations, multiplied by the time per multiplication, would give a reasonable time estimate. The most satisfactory method of such estimations at present, however, is still an experimental one for any particular problem.

**Computer Storage.** Sooner or later almost every computer will find its primary storage completely saturated by a problem, which must then

be broken up into component parts and fed into the main storage in smaller blocks. If there is no secondary storage, this process is dependent on the flexibility and speed of external input-output equipment, such as punched cards or punched paper tape. If there is secondary internal storage, such as magnetic drum storage or magnetic tape units, the amount of time that a problem requires will depend very strongly on the method of division of a problem into pieces, and the routing of these pieces in and out of main working storage in the most efficient sequence. Some of the obvious procedures possible are:

1. Storage of data in main storage and bringing in programs in blocks from secondary storage.
2. Storage of program in main storage and bringing in data in blocks from secondary storage.
3. Mixture of (1) and (2).

A discussion of the third process, with the inclusion of built-in checks, is given in Brown *et al.* (Ref. 18). The most experience with such hierarchy transfer of information has been by users of the Univac I, which had a relatively small main storage in the form of acoustic delay lines and a large secondary storage in the form of magnetic tape units. A discussion of an automatic system which faces the problem of *segmenting* a program, either data or instructions, into pieces is given by this group (see Ref. 2). The general conclusions of these and other workers is that while rules may be set up to prescribe the storage hierarchy manipulation process so that a computer may do it automatically, it is imperative that a programmer be allowed to *override* any automatic segmenting and allocation system in order to provide increased efficiency.

**Minimal Latency Programming.** For those computers with a one-plus-one address instruction scheme, on the other hand, machine allocation of storage seems satisfactory in a large majority of cases. Most work of this type has been done for the IBM 650. Gordon (Ref. 41) first wrote a program assigning next instruction addresses automatically by machine for this computer; this was later incorporated into SOAP (Symbolic Optimal Assembly Program) (Ref. 83).

### Examples of Computer Programming

The most straightforward way to describe the process of digital computer programming is to give a sequence of equivalents for each type of element in the flow diagram notation already discussed. Two "target" machine languages will be described in the list of equivalents: a computer with a single-address (actually "one-plus-one") instruction logic, the IBM 650; and a computer with a three-address instruction logic, the MIDAC (Michigan Digital Automatic Computer). (See Sect. 6.)

In the examples that follow, the first few will be in the original languages of these machines, so that the results will be the actual ones that might be used. The later ones will use symbolic addresses in place of the usual decimal or hexadecimal integer addresses. This allows much easier understanding of the routines. These symbolic addresses will generally use five or fewer alphanumeric symbols, such as are used in the SOAP (Symbolic Optimal Assembly Program) for the IBM 650. The MIDAC notation used on that machine with the MAGIC system is somewhat simpler, but comparable, and for uniformity the same addressing system will be used. For the IBM 650, when the next instruction address (NI Add) is not written, it means that the next instruction follows directly below in sequence. See Sect. 6 for instruction codes of the IBM 650 and MIDAC computers.

**Notation.**

- "store"
- A<sub>U</sub> Upper accumulator
- A<sub>L</sub> Lower accumulator
- C(n) Contents of location n
- Loc(a) Address of the location containing a

Parentheses surrounding an address mean it is modified during the program. Dotted lines drawn underneath instructions indicate conditional transfer of control. Solid lines indicate unconditional transfer of control.

**Arithmetic Boxes.** A typical arithmetic box would be that of Fig. 11

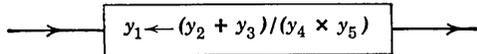


FIG. 11. An arithmetic box.

where the values of  $y_1, y_2, \dots$ , etc., are "digital numbers." The corresponding sequence of IBM 650 instructions, using the SOAP assembly language notation, is given in Table 13. It is assumed that numbers are so

TABLE 13. IBM 650 PROGRAM FOR PROBLEM OF FIG. 11

SOAP Symbolic Program	Machine Language Program				NI Address	Explanation
	Loca- tion	Opera- tion	Data Address			
RAU Y0004	0100	60	0204	0101	$y_4 \rightarrow A_U$	
MPY Y0005	0101	15	0205	0102	$C(A_U) \times y_5 \rightarrow A_L$	
STU T0001	0102	21	0301	0103	$C(A_U) \rightarrow t_1$	
RAU Y0002	0103	60	0202	0104	$y_2 \rightarrow A_U$	
AUP Y0003	0104	10	0203	0105	$C(A_U) + y_3 \rightarrow A_U$	
DIV T0001	0105	14	0301	0106	$C(A_U)/t_1 \rightarrow A_L$	
STL Y0001	0106	20	0201	0107	$C(A_L) \rightarrow y_1$	

scaled so that overflow would not occur. The next instruction address of the IBM 650 may be omitted in the SOAP program, since it is filled in automatically. The translation of this program, with both data and next instruction address included, is given alongside the original sequence, with explanation at right. It is supposed that the following decimal address storage assignments have been made:

Program:	100	(instructions follow in sequence)
Y0001:	201	(other Y's follow in sequence)
T0001:	301	(other T's follow in sequence)

(Generally, if minimum latency or optimal coding were used, the location of successive instructions and the corresponding next instruction addresses would not appear in sequence.)

On the MIDAC, the corresponding program in the MAGIC system might be used (see Table 14). It is assumed that the same storage assign-

TABLE 14. MIDAC PROGRAM FOR PROBLEM OF FIG. 11

MAGIC Symbolic Program				Machine Language Program (Hexadecimal)					Explanation
				Hexa- decimal Location	$\alpha$ Ad- dress	$\beta$ Ad- dress	$\gamma$ Ad- dress	Oper- ation	
Y04	Y05	T01	MU	064	OCC	OCD	12D	08	$y_4 \times y_5 \rightarrow t_1$
Y02	Y03	T02	AD	065	OCA	OCB	12E	05	$y_2 + y_3 \rightarrow t_2$
T01	T02	Y01	DV	066	12E	12D	OC9	0B	$t_2/t_1 \rightarrow y_1$

ments hold as above. Note that a computer with a three-address instruction logic does not use an accumulator and does not need as many machine words to perform the same problem in this case. (Hereafter, machine language translations will be omitted.)

**Comparison Boxes.** The act of comparison can generally be accomplished by one instruction using either type of logic. A typical comparison box would be that of Fig. 12. Now assign the same locations for  $y_1$  and program as before, and assume in addition that the number .15 is in location  $t_1$ . All numbers are "digital." The IBM 650 program is given in

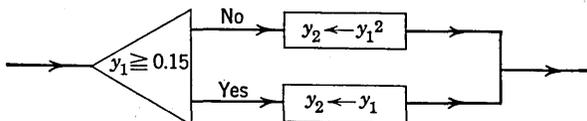


FIG. 12. A typical comparison.

Table 15. The MIDAC program is given in Table 16. The "base" instruction (BA) is a simple transfer of control instruction to 104, which corresponds to the next address "0108" of instruction 0104 of the IBM 650 program above. In Table 16  $C(y_2)$  is assumed originally zero.

TABLE 15. IBM 650 SOAP PROGRAM FOR COMPARISON OF FIG. 12

Location	Operation	Data Address	Next Instruction Address	Explanation
0100	RAL	Y0001		$y_1 \rightarrow A_L$
0101	SLO	T0001		$C(A_L) - t_1 \rightarrow A_L$
0102	BMI	0105		$C(A_L) < 0 \rightarrow$ NI Add = 105
-----				
0103	RAL	Y0001		$y_1 \rightarrow A_L$
0104	STL	Y0002	0108	$C(A_L) \rightarrow y_2$
0105	RAU	Y0001		$y_1 \rightarrow A_U$
0106	MPY	Y0001		$C(A_U) \times y_1 \rightarrow A_U$
0107	STU	Y0002		$C(A_U) \rightarrow y_2$
0108	(continue)			

TABLE 16. MIDAC MAGIC PROGRAM FOR COMPARISON OF FIG. 12

Decimal Location	$\alpha$ Ad- dress	$\beta$ Ad- dress	$\gamma$ Ad- dress	Oper- ation	Explanation
100	Y1	T1	103	CN	$y_1 < t_1 \rightarrow$ NI Add = 103
-----					
101	Y1	Y1	Y2	EX	$y_1 \rightarrow y_2$
102	000	001	104	BA	$000 < 001 \rightarrow$ NI Add = 104
103	Y1	Y1	Y2	MU	$y_1 \times y_1 \rightarrow y_2$
104	(continue)				

**Indicial Boxes.** Indicial boxes in a flow diagram are most often part of a more elaborate loop or induction structure. Index modification is usually accomplished in two ways: (1) in the arithmetic unit; (2) by means of an index register.

Performance of an inductive process usually involves four separate functions:

1. Initial setting of an index or counter to an initial value (often, but not always, zero or one).
2. Modification of an address of arithmetic (or other) instruction as a function of the index.
3. Increasing the value of, or incrementing, the index (in some cases this may be a decrementing process).

4. Testing the value of the index to see if the induction has been completed.

An example of a process containing all four of these functions is the computation of the vector inner product  $\sum_{i=1}^n a_i b_i$ . It is assumed that the numbers are "digital" and that they are so scaled that no overflow will occur. The flow diagram is as in Fig. 13.

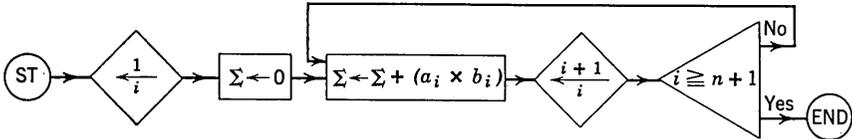


FIG. 13. Vector inner product,  $\sum_{i=1}^n a_i b_i$ .

Programmers are generally advised, even if it costs more instructions, to preset all counters, instructions, etc., to original conditions as is shown on the flow diagram of Fig. 13 rather than afterwards, as could possibly be done with "loops within loops." It is possible to make use of input of

TABLE 17. IBM 650 SOAP PROGRAM FOR VECTOR INNER PRODUCT, FIG. 13

Loca- tion	Oper- ation	Data Address	Next Instruction Address	Explanation
0100	RAL	T0000		$0 \rightarrow A_L$
0101	STL	SIGMA		$C(A_L) \rightarrow \Sigma$
0102	RAL	T0001		$1 \times 10^{-6} \rightarrow A_L$
0103	STL	I0000		$C(A_L) \rightarrow i \times 10^{-6}$
0104	RAL	0120		"RAU A0000" $\rightarrow A_L$
0105	ALO	I0000		Form "RAU Loc ( $a_i$ )"
0106	STL	0110		$C(A_L) \rightarrow 0110$
0107	RAL	0121		"MPY B000" $\rightarrow A_L$
0108	ALO	I0000		Form "MPY Loc ( $b_i$ )"
0109	STL	0111		$C(A_L) \rightarrow 0111$
0110	RAU	(A0000)		$a_i \rightarrow A_U$
0111	MPY	(B0000)		$C(A_U) \times b_i \rightarrow A_U$
0112	AUP	SIGMA		$\Sigma + C(A_U) \rightarrow A_U$
0113	STU	SIGMA		$C(A_U) \rightarrow \Sigma$
0114	RAL	I0000		$i \times 10^{-6} \rightarrow A_L$
0115	ALO	T0001		$(1 \times 10^{-6}) + C(A_L) \rightarrow A_L$
0116	STL	I0000		$(i+1) \times 10^{-6} \rightarrow i \times 10^{-6}$
0117	SLO	N0001		$C(A_L) - (n+1) \times 10^{-6} \rightarrow A_L$
0118	BMI	0104		$C(A_L) < 0 \rightarrow NI \text{ Add} = 0104$
0119	HLT	0000		Stop
0120	RAU	A0000		Base instruction for $a_i$
0121	MPY	B0000		Base instruction for $b_i$

information from outside the computer to do the initial setting. However, in this case, one cannot start over at the initial internal instruction, but must start over at the point of reading in of instructions. The process of "resetting" counters to their original condition after the testing process is completed should not be used unless one is willing to take the consequences of possible improper runs upon starting over.

Note that a temporary location  $\Sigma$  (SIGMA) is used to hold the partial sum. The coding for the IBM 650, using address modification in the arithmetic unit, is given in Table 17. (It is assumed that location T0000 contains 0, T0001 contains  $1 \times 10^{-6}$ , T0002 contains  $n \times 10^{-6}$ ,  $a_1$  is in A0001,  $a_2$  in A0002,  $b_1$  in B0001,  $b_2$  in B0002,  $\dots$ , etc., and  $(n + 1) \times 10^{-6}$  in location N0001.)

In this program (Table 17) locations 100-103 preset the initial conditions, locations 104-109 modify the instruction addresses, locations 114-116 perform the incrementing, and locations 117-118 perform the comparison. This program may be rewritten as in Table 18 to use the two instructions being modified themselves as counters. (Hereafter symbolic addresses will be used as instruction location addresses as well as data.) Note that now the flow diagram of Fig. 13 is not followed precisely.

TABLE 18. ALTERNATE IBM 650 SOAP PROGRAM FOR VECTOR INNER PRODUCT

Location	Operation	Data Address	Next Address	Explanation
BEGIN	RAL	T0000		} $0 \rightarrow \Sigma$
	STL	SIGMA		
	RAL	INST1		} RAU A0000 $\rightarrow$ MULT1
	STL	MULT1		
	RAL	INST2		} MPY B0000 $\rightarrow$ MULT2
	STL	MULT2		
ENTRY	RAL	MULT1	←	} Generate RAU Loc ( $a_i$ )
	ALO	T0001		
	STL	MULT1		
	RAL	MULT2		
	ALO	T0001	} Generate MPY Loc ( $b_i$ )	
	STL	MULT2		
	SLO	TEST1	} Is $i = n + 1$ ?	
	BMI	MULT1		
	HLT	0000		
MULT1	RAU	(A0000)	MULT2	} $\Sigma + (a_i \times b_i) \rightarrow \Sigma$
MULT2	MPY	(B0000)	NEXT	
NEXT	AUP	SIGMA		
	STU	SIGMA	ENTRY	
INST1	RAU	A0000	MULT2	Initial instruction
INST2	MPY	B0000	NEXT	Initial instruction
TEST1	MPY	B( $n + 1$ )	NEXT	Test instruction

In certain cases the latter technique may prove quicker or may require less storage. (Above, the second technique requires twelve instead of fifteen instructions in the loop itself.) Only the first procedure will be coded for the MIDAC as shown in Table 19.

TABLE 19. MIDAC MAGIC PROGRAM FOR VECTOR INNER PRODUCT, FIG. 13

Location	$\alpha$ Address	$\beta$ Address	$\gamma$ Address	Operation	Explanation
BEGIN	INCRE	T0000	I0000	AD	$1 \rightarrow i$
	T0000	T0000	SIGMA	AD	$0 \rightarrow \Sigma$
	INSTR	T0000	MULT1	AD	Set instruction
MULT1	(A0000)	(B0000)	T0002	MU	$a_i \times b_i \rightarrow t_2$
	T0002	SIGMA	SIGMA	AD	$t_2 + \Sigma \rightarrow \Sigma$
	I0000	INCRE	I0000	AD	$i + 1 \rightarrow i$
	INCRE	MULT1	MULT1	AD	Increase addresses
	I0000	TEST1	MULT1	CN	$i \leq n \rightarrow NI = \text{MULT1}$
	000	000	000	RI	Stop
INCRE	001	001	000	00	Increment
INSTR	(A0000)	(B0000)	T0002	MU	Base instruction
TEST1	$(n + 1)$	$(n + 1)$	000	00	End of cycle test

**Use of Index Registers.** The augmented IBM 650, with index registers, has three index registers, A, B, and C, of four decimal digits each. Modification of an address at execution time by using an index register will be indicated by one of these letters following the address in a "tag" position. The modified program for vector inner product given in Table 20 requires some new IBM 650 instructions not previously described, and uses a value of  $n = 100$ .

**RSA.** *Reset and subtract from index accumulator A.* Index accumulator A will be reset to zero and the data address will be subtracted from it.

**AXA.** *Add to index accumulator A.* Add the data address to index accumulator A.

**NZA.** *Branch on nonzero index accumulator A.* If the contents of the index accumulator A is nonzero, take the next instruction from the data address. Otherwise take the next instruction from the instruction address.

**RAA.** *Reset and add to index accumulator A.* Index accumulator A will be reset to zero and the data address will be added to it.

Note that the program of Table 20 is much shorter when using the index registers, but the flow diagram has been slightly altered so as to count down from  $i = -100$  to  $i = 0$ . The instructions in location ENTER and its successor still perform their operations in an increasing sequence, however.

The MIDAC has one index register, the base counter, which is added to any address in an instruction; at the time of execution, when that address

TABLE 20. IBM 650 (AUGMENTED) SOAP PROGRAM FOR VECTOR INNER PRODUCT WITH INDEX REGISTERS

Location	Operation	Data Address	Tag	Next Instruction Address	Tag	Explanation
	RSA	0100				$-100 \rightarrow i$
	RAL	T0000				} $0 \rightarrow \Sigma$
	STL	SIGMA				
→ ENTER	RAU	A0101	A			} $\Sigma + a_i \times b_i \rightarrow \Sigma$
	MPY	B0101	A			
	AUP	SIGMA				
	STU	SIGMA				
	AXA	0001				
-----	NZA	ENTER		NEXT		$i > 0 \rightarrow NI = ENTER$
NEXT	HLT	0000				Stop

is "tagged" with a negative sign. One operation, the *base* operation, is used to set, increase and test the counter. The instruction

$$-\alpha \quad \beta \quad \gamma \quad BA$$

performs all of the following operations in sequence:

- C (Base Counter) +  $\alpha \rightarrow$  Base Counter;
- if C (Base Counter) <  $\beta \rightarrow$  Next Instruction Address equals  $\gamma$ ;
- if C (Base Counter)  $\geq \beta \rightarrow$  Next Instruction Address is in sequence, and  $0 \rightarrow$  Base Counter.

Thus the base counter is ordinarily set to zero by an instruction

$$000 \quad 000 \quad 000 \quad BA$$

The above vector inner product program for MIDAC would now be that of Table 21.

TABLE 21. MIDAC MAGIC PROGRAM FOR VECTOR INNER PRODUCT WITH AN INDEX REGISTER

Location	$\alpha$ Address	$\beta$ Address	$\gamma$ Address	Operation	Explanation
BEGIN	000	000	000	BA	Clear base counter
	SIGMA	SIGMA	SIGMA	SU	$0 \rightarrow \Sigma$
→ ENTER	-A0001	-B0001	T0002	MU	$a_i \times b_i \rightarrow t_2$
	T0002	SIGMA	SIGMA	AD	$\Sigma + t_2 \rightarrow \Sigma$
-----	-001	n	ENTER	BA	$i < n \rightarrow NI = ENTER$
	000	000	000	RI	Stop

Note that here the counter  $i$  ranges from 0 to  $n - 1$ . The comparison in efficiency between the IBM 650 and MIDAC is not a completely fair one, since some of the features of the IBM 650 (performance of instructions from the accumulator, use of the distributor, etc.) have not been used. Nevertheless, these examples do show the marked advantage in this type of cyclical problem of a three-address instruction logic with index register over a comparable single-address instruction logic.

**Multiway Switch.** The flow diagram notation for the variable remote connector, or multiway switch, provides another example of address modification. Such a multiway switch might be used with a table look-up process, such as is required in many function evaluation processes, interpretive programs, and other problems in which performance is dependent upon a value of a function. The flow diagram is given in Fig. 14, where,

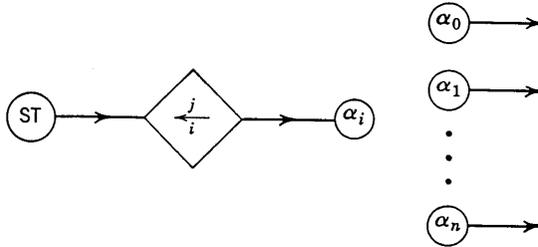


FIG. 14. A multiway switch.

dependent on the value of  $j$ , control jumps to one of  $n + 1$  remote connections.

Four possible programs are described below, two for a single-address (IBM 650) and two for a three-address (MIDAC) instruction logic, with and without the use of index registers.

EXAMPLE 1. IBM 650 SOAP program for multiway switch without use of index accumulators. Note performance of an instruction from the lower accumulator.

Location	Operation	Data Address	Instruction Address	Explanation
BEGIN	RAL	JUMP		} Set JUMP to } "NOP 0000 (ALPHA + j)"
	ALO	J0000	8002	
JUMP	NOP	0000	(ALPHA)	} Jump to $\alpha_j$ (performed in $\Lambda_L$ )
J0000	00	0000	$j$	} Locations $\alpha_0, \alpha_1,$ } $\dots, \alpha_n$
ALPHA				

EXAMPLE 2. MIDAC MAGIC program for multiway switch without use of base counter, see Table 23.

Location	$\alpha$ Address	$\beta$ Address	$\gamma$ Address	Operation	Explanation
BEGIN	J0000	JUMP	JUMP	AD	Set JUMP to "000 000 (ALPHA + j) BA"
JUMP J0000 ALPHA	000 000	000 000	(ALPHA) $j$	BA 00	
					Locations $\alpha_0, \alpha_1,$ $\dots, \alpha_n$

EXAMPLE 3. IBM 650 (Augmented) SOAP program for multiway switch with index register. Note that for the special case where the address of the RAA instruction is greater than 7999, the index register A is loaded directly.

Location	Operation	Data Address	Tag	Instruction Address	Tag	Explanation
BEGIN	RAL	J0000				$j \rightarrow$ Index Acc A; jump to $\alpha_j$
	RAA	8002		ALPHA	A	
J0000 ALPHA	00	0000		$j$		Locations $\alpha_0,$ $\alpha_1, \dots, \alpha_n$
.	.	.	.	.	.	

EXAMPLE 4. MIDAC MAGIC program for multiway switch with an index register. Again in this case the use of the base counter is hindered because there is no direct way to store an integer in it.

Location	$\alpha$ Address	$\beta$ Address	$\gamma$ Address	Operation	Explanation
BEGIN	000	000	000	BA	Clear base counter
	J0000	JUMP1	JUMP2	AD	Store $j$ in $\alpha$ of JUMP2
JUMP2	(000)	999	-ALPHA	BA	$j \rightarrow$ base counter, jump to $\alpha_j$
J0000 JUMP1 ALPHA	$j$ 000	000 999	000 -ALPHA	00 BA	
.	.	.	.	.	Locations $\alpha_0, \alpha_1,$ $\dots, \alpha_n$
.	.	.	.	.	

**Dynamic Stop.** It is possible to code a transfer to the same instruction to give a *tight loop* that accomplishes the equivalent of stopping the machine but allows it to run on at high speed. For computers with electrostatic storage, where the "read-around" or "consultation ratio" is important, this is definitely not recommended. It has mainly been used on computers without built-in halt instructions. On the two machines being used in examples, the following would constitute "dynamic stops."

EXAMPLE 1. IBM 650 SOAP program for dynamic stop.

Location	Operation	Data Address	Instruction Address	Explanation
LOOP	NOP	0000	LOOP	NI Add = LOOP

EXAMPLE 2. MIDAC MAGIC program for dynamic stop.

Location	$\alpha$ Address	$\beta$ Address	$\gamma$ Address	Operation	Explanation
LOOP	000	001	LOOP	BA	NI Add = LOOP

**Subroutine Linkages.** Entry to subroutines must accomplish the following:

1. Store the address of the next word (which may contain the next main program address, or else a program parameter to be used in the subroutine).
2. Transfer control to the first address of the subroutine.

This is given by the flow diagram of Fig. 15.

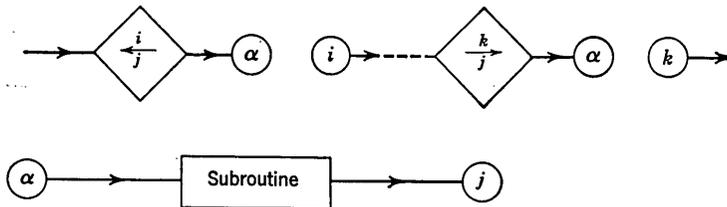


Fig. 15. Two main program entries to the same subroutine.

On the IBM 650, the transfer of information about the position to which control is to be returned in the main program in one technique makes use of the one-plus-one address features of the computer. The next instruction to which control is to be returned in the main program is loaded in an available machine register (the distributor) and then, after transfer to the subroutine, the latter stores the next-instruction in an exit-instruction location (Table 22). Note that the exit instruction originally is loaded with a halt instruction, so that if control should be transferred improperly to the

TABLE 22. IBM 650 SOAP PROGRAM FOR SUBROUTINE ENTRY

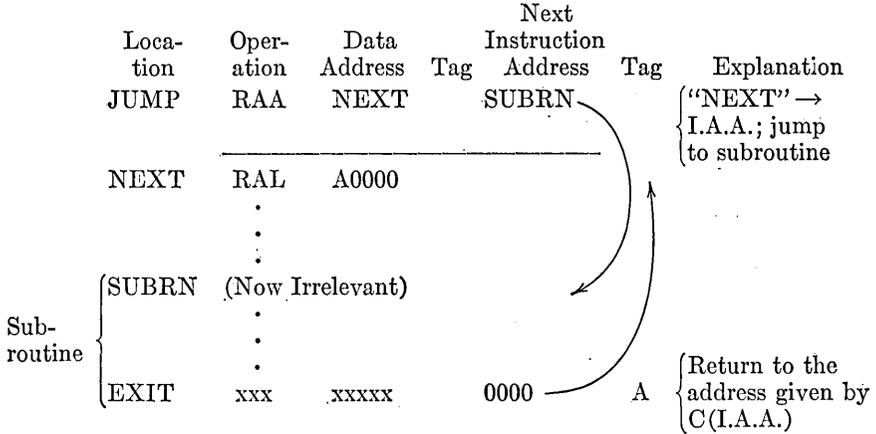
	Loca- tion	Oper- ation	Data Address	Next Instruction Address	Explanation
		·			
		·			
		LDD	RJUMP	SUBRN	{"RAL A0000" to Distr., jump to subroutine
	RJUMP NEXT	RAL	A0000	NEXT	
		·			
		·			
Sub- routine	SUBRN	STD	EXIT		{Store next instr. in EXIT
		·			
		·			
	EXIT	(HLT	0000	0000)	EXIT instruction

subroutine, the computer would stop. Such safeguards are sometimes useful in debugging programs.

If the IBM 650 did not have a next instruction address and no special subroutine entry instruction were available, the following so-called Wheeler entry (Ref. 108) could be used. The instruction JMP is not an actual IBM 650 instruction.

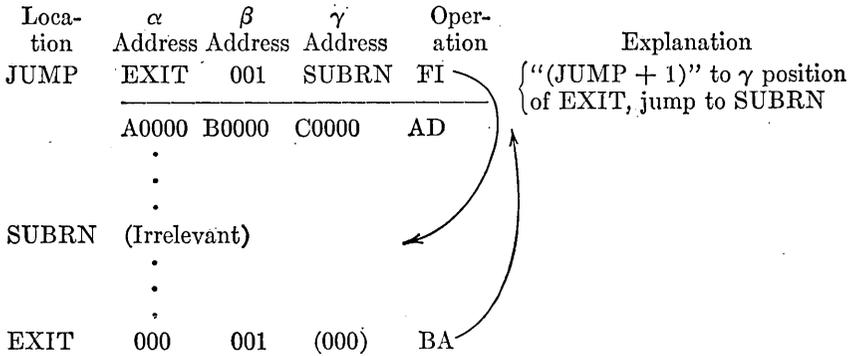
	Loca- tion	Oper- ation	Data Address	Next Instruction Address	Explanation
	SELF	RAL	SELF		"RAL SELF" → A <sub>L</sub>
		ALO	THREE		"RAL (SELF + 3)" → A <sub>L</sub>
		JMP	SUBRN		Jump to subroutine
		RAL	A0000		Next instruction
		·			
		·			
Sub- routine	SUBRN	SDA	EXIT		Store NIAdd in EXIT
		·			
		·			
	EXIT	JMP	(0000)		This becomes "JMP (SELF + 3)"
		·			
		·			
	THREE	00	0003	0000	

With an index accumulator, the following subroutine sequence could be used to provide a return jump from a subroutine on the IBM 650.



(Here the sequences of x's indicate the operation and data address can be anything. I.A.A. stands for index accumulator A.)

In the MIDAC's typical three-address instruction logic, one instruction, the "file" (FI) operation, performs the same function as the instruction labeled JUMP for the modified IBM 650.



*Note.* If program parameters (variables needed in the subroutine) are required, they are generally stored in (1) the accumulator and other positions, except in the Wheeler entry method, and (2) registers following the JUMP instruction in the main program. In the latter case, the subroutine entries or the subroutines themselves must be altered in an obvious fashion.

**Table Look-Up.** In many cases, it is desired to find the value of a function stored in a table. Since the process of finding an inner product described above obviously requires looking up  $a_i$  (and  $b_i$ ) in a table with

argument  $i$ , a similar procedure can also be used, as long as the arguments occur at equal intervals. The flow diagram for one approach is shown in Fig. 16; no actual coding is included. Here the argument is  $x$  at equal

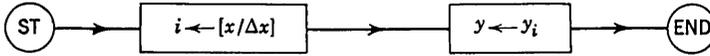


FIG. 16. Table look-up of  $y = f(x)$ .

intervals  $\Delta x$ , and the function values are  $y_i$ . Again  $[\dots]$  means "integral part of." The result will be the value in the table corresponding to  $[x]$ .

The table look-up instruction on the IBM 650 (see Sect. 6) provides a similar technique using only one instruction on the IBM 650. Several hardware restrictions render this instruction less useful, but it is still a very powerful device.

A binary table look-up procedure may often prove most efficient when an equal interval table or table look-up operation is not available. Suppose there exist 16 arguments,  $x_0, \dots, x_{15}$  in a table. A value of  $x$  is given, and it is desired again to find the approximation  $y = f(x)$  from a table of  $y_i (i = 0, \dots, 15)$ . The flow diagram is shown in Fig. 17. Again, no coding is included.

This process may obviously be recorded in a recursive (loop) structure. The number of comparisons in this process is  $C(N) \approx \log_2 N$ , where  $N$  is the number of elements in the table.

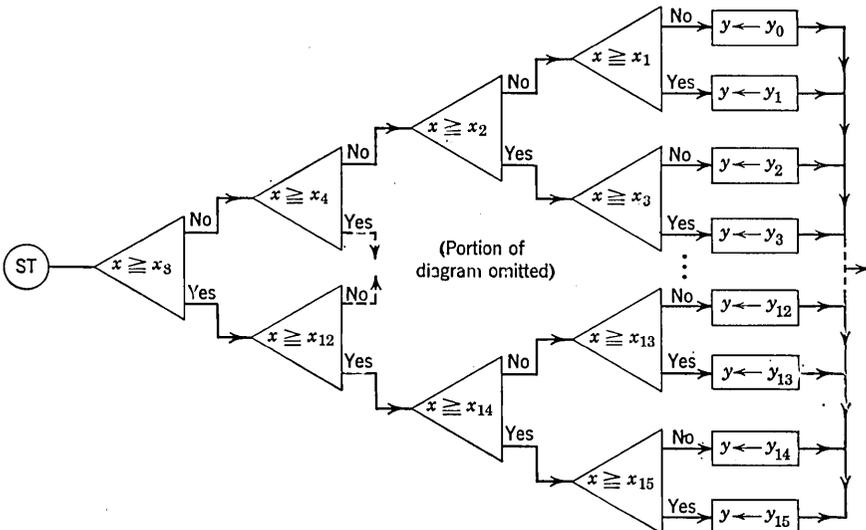


FIG. 17. Binary table look-up procedure.

### Programming with Secondary Storage

Since secondary storage varies from digital computer to computer, it is difficult to give specific rules for its usage. The various devices which have been attached to general purpose computers as secondary storage include: (1) magnetic drums, (2) magnetic tape units, and (3) large-scale random access devices (bin type magnetic tape units, magnetic disks, large-size drums).

All these devices employ magnetic methods of recording, and are therefore storage of a general nonvolatile nature. They nevertheless have the capacity for malfunctions; dust on a magnetic surface, improper relay closure, etc., may cause an incorrectly read or written digit. It is therefore necessary, if satisfactory reliability or built-in checks are not available, to include programmed checks, usually by using storage-summing techniques, to guarantee proper performance. These techniques are described below under Integrated Systems (see Sect. 10).

Some magnetic drum systems are integrated into the high-speed storage unit; here the only programming requirement is to provide economy of performance either by minimal latency programming or an interlace feature. (See Ref. 103.) Use of magnetic drums in this fashion causes no basic problems. Drum equipment used as a secondary storage, however, entails a scheduling problem that generally can be solved exactly only by a computer itself, a procedure which has not been followed. Simpler methods of approximate solution are needed. Programs for such hierarchy transfer are discussed in Sect. 11.

The use of magnetic tape units is very dependent on the presence or absence of built-in checking, ability to read both forward and backward, presence of fixed or variable block length. The reader is urged to consult Sect. 6, and then the various manufacturer's operation manuals or reports (see Refs. 34, 50, 54, 93, 103, and 148) for a fuller discussion of the instructions that govern magnetic tape equipment.

Large-scale, so-called random access storage, as embodied in tape bin storage (see Ref. 119) and magnetic disk storage (see Ref. 118) basically require methods of mapping call words of long digit length (for example, inventory parts numbers) into a smaller number of digits giving the address in the random access storage. A parts number, ten digits in length, may correspond to a five-digit address in a random access unit. How can a unique correspondence be made? Certainly if there are more than 100,000 different parts, this is impossible; but if there are fewer than that, some method of randomization may allow an almost one-to-one mapping from the set of parts numbers (scattered thinly throughout the entire ten-digit range) into the set of storage addresses (most of which would be used).

One popular technique is a variation of the so-called mid-square procedure (see Ref. 74), to produce (in this case) the desired five-digit address.

**EXAMPLE.** Suppose an inventory parts number were 1122305151. The twenty-digit product of the number with itself is:

12595688518611232801

If one uses digits 8 through 12 to provide a five-digit address, one obtains 85,186. With high probability, out of a group of 100,000 parts numbers each with ten digits, no two of them will have the same set of five mid-square digits. If more than one number does have a duplicate address under this mapping, the address can be tagged as an "exception" and either a second mid-square process based on the center ten digits of the resultant square, or another group of five digits in the square, may be used to generate a new address, which again can be tested for duplications, etc.

**Sorting and Merging.** One primary problem that involves the use of secondary storage is the problem of rearrangement of input data in an ordered fashion. This problem can occur on one hand in assembly programs where symbolic addresses are to be arranged in an easily entered, ordered list or, on the other hand, in any sort of business file maintenance problem where inquiries or changes that are not externally ordered in sequence are to be compared with a main file. Goldstine and von Neumann (see Ref. 19, Part II, Vol. II) developed the first theoretical analysis of two of the main methods of information rearrangement and compared the use of a general purpose digital computer for these purposes with standard punched card equipment, with some advantage in favor of the former. Later studies, as listed in Seward's dissertation (Ref. 140), produce a better "informational advantage" as far as use of a general purpose digital computer is concerned, but still indicate that this present machine structure is far from dominant in such performance.

There are two general classes of information rearrangement:

1. *Merging.* The act of taking two (or more) previously numerically increasing (or decreasing) ordered sequences of numerical information and combining them in one numerically increasing (or decreasing) sequence.

2. *Sorting.* The act of taking an arbitrarily ordered sequence of (numerical) information and arranging it in a numerically increasing (or decreasing) sequence. Since alphabetical information in a computer is most often encoded in some numerical form that is ordered analogous to the position in the alphabet, these definitions also cover merging and sorting of alphabetical and other nonnumerical information.

Such blocks of information (called *items*) are usually sorted with respect to a *key*, a sequence of one or more symbols (digits) which are pertinent to

the position of the information in the sequence. In this discussion, it will be assumed, without loss of generality, that the key is numeric and the ordering desired is generally increasing.

**Use of Main Storage in Sorting.** If the blocks of information to be sorted each contain few enough computer words, many of them may be stored in the high-speed storage of a computer. During the sorting process, the relative values of the keys (usually located at the beginning of a block) may be used to exchange entire blocks. A simpler and often more efficient process, however, is to move only the *addresses* of the blocks rather than the blocks themselves after a comparison of the keys has been made. Thus, if  $n$  blocks of  $m$  words are to be sorted, to be stored in  $nm$  positions, space must be also left for  $n$  addresses, which will be shuffled into an order corresponding to the order into which the blocks should be moved. After this process of rearranging the  $n$  addresses is completed, the corresponding blocks may then be read out onto secondary storage in the proper sorted order. This technique may obviously be extended to use of magnetic drums as well, since they in general have a relatively small access time.

### Sorting Methods

There are two main types of sorting: (1) digital sorting and (2) merge sorting.

**Digital Sorting.** This method uses successive digits (or groups of digits) in the key to arrange the sequence being sorted into an ascending order. This is the method usually used on punched card equipment, where the information is passed through the sorter one time for each digit in the key, and the cards are collected in one of a number of output units (10 in a decimal sorter) at the end of each pass. If one starts at the least significant digit and proceeds upward in sequence, ordering the entire stack by digits after each pass, the entire process requires  $d$  passes, where  $d$  is the number of digits in the key.

The logical extreme of the digital sort technique is the so-called address-sorting technique on a stored-program computer. If, for example, in a decimal computer single words are to be sorted on a two-digit key, each value of which is to appear only once, this key may be used as an index to modify a storage address for each word in turn. Thus, if the resultant storage block for ordered information is in locations 1900 to 1999, and if the two-digit key is 65, the machine word corresponding should be sent to location 1965. Even if the information being sorted is in larger blocks, if the keys are unique and occur densely within the entire possible range of key values, a similar technique may be used, either with the blocks themselves or with their addresses. Duplicate keys, if very few occur, may be handled by signals designating an exceptional case.

With magnetic tape units, (decimal) digital sorting may be achieved by reading from one tape and storing the output on one of ten tapes, each corresponding to a possible digit of the key. With two banks of ten-tape units, the previous output can be used as the next input, with the next successive significant digit being sorted on in order.

The time required for such a digital sort is

$$T = t \times n,$$

where  $T$  is total time,  $t$  is time for one passage of the entire information through the storage, and  $n$  the number of digits, or as Seward (Ref. 140) has noted, approximately

$$T = NA[\log_r R]$$

where  $N$  is the number of items being sorted,  $A$  the access time to read or write items in the storage, the range of the key is from 0 to  $R$ , and  $r$  is the base or radix used in representing the key.

**Sorting by Merging.** This technique is that recommended by Goldstine and von Neumann for internal sorting. Sorting by merging consists in taking two or more ordered groups of items and merging them into one ordered group (usually called a *string*). Figure 18 shows an example of such a merging process.

Goldstine and von Neumann (Ref. 19) have discussed merging in detail in the case where information is stored in the main storage. The flow diagram is shown in Fig. 19. In this case, an item  $X^i$  consists of a one-word

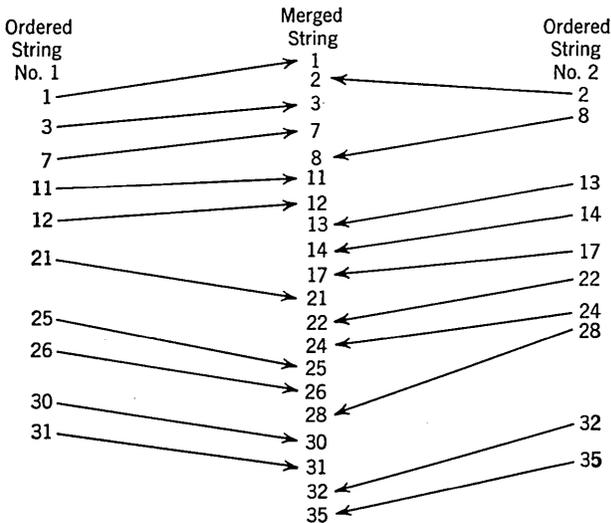
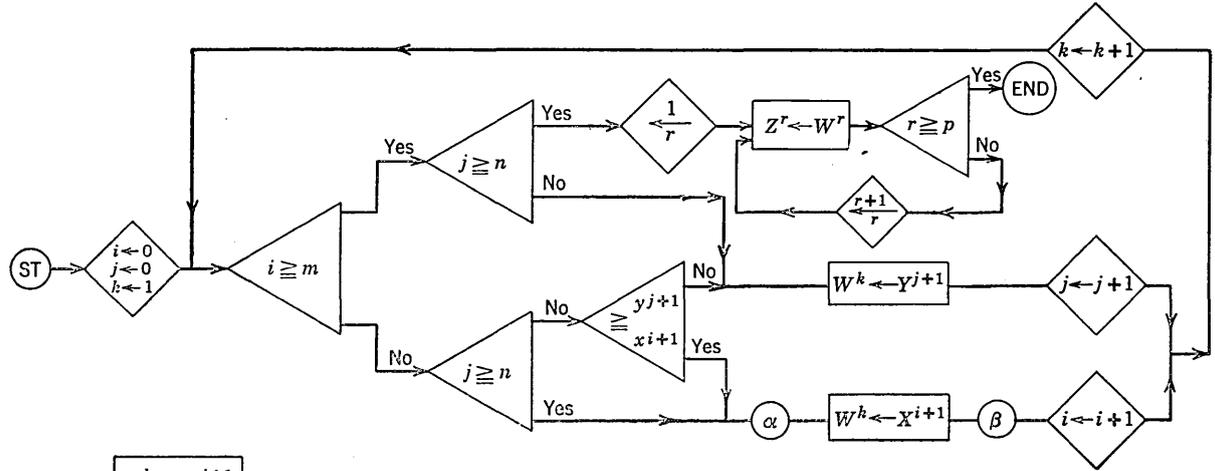
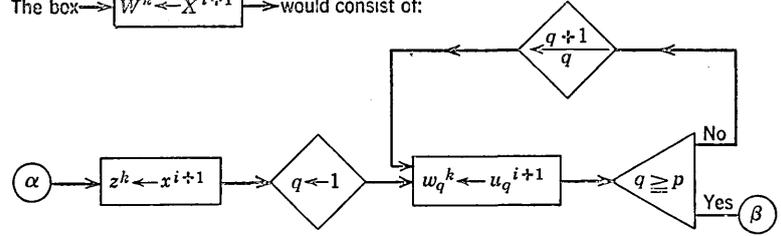


FIG. 18. Example of sorting by merging.



The box  $W^k \leftarrow X^{i+1}$  would consist of:



and similarly for  $W^k \leftarrow Y^{j+1}$  and  $Z^r \leftarrow W^r$

FIG. 19. Merging two strings of  $m$  and  $n$  items, each of order  $p$ .

key  $X^i$  followed by  $p$  other words ( $p$  is the *order* of the item). A string  $S$  will consist of  $n$  items, where  $n$  is the *length* of the string. A string would then contain  $n(p + 1)$  words. The flow diagram describes the merging of two strings  $S = (X^1, X^2, \dots, X^n)$  (of  $n$  items) and  $T = (Y^1, Y^2, \dots, Y^m)$  (of  $m$  items) to produce an ordered string  $R = (Z^1, Z^2, Z^3, \dots, Z^{n+m})$  where each  $Z^i$  is one of the previous  $X$ 's or  $Y$ 's such that the keys  $x^i$  and  $y^i$  are now arranged in increasing order. The string  $X^i$  would be given by  $X^i = (x^i, u_1^i, \dots, u_p^i)$ ,  $Y^i$  by  $Y^i = (y^i, v_1^i, \dots, v_p^i)$ , and  $Z^i$  by  $Z^i = (z^i, w_1^i, \dots, w_p^i)$ .

In the general case of merging two strings from two tapes into a third string on a third tape, there would need to be further storage boxes included in the flow diagram.

Sorting of an arbitrarily ordered string of length  $N$  can now be accomplished by successive merging. This can be accomplished as follows:

1. Each pair of items in sequence is merged by considering each a string of length one to form a string of length two.
2. Each pair in sequence of strings of length  $2^\nu$  ( $\nu = 1, \dots$ ) is merged, using the merging process described above, to yield a merged string of length  $2^{\nu+1}$ .
3. When  $2^{\nu+1} \geq N$  and there is only one string, the process is complete.

The fact that  $N$  is not exactly equal to  $2^\nu$  can be disregarded by considering the remaining  $r_\nu = N - 2^\nu$  elements as a separate string which may or may not be merged at each stage of the process.

An example of the merge-sort process is given for a general string of 35 items (here merely keys) in Fig. 20.

A nonordered string of items  $\{A^1, A^2, A^3, \dots, A^N\}$ , each of order  $p$ , to be sorted may be manipulated by the flow diagram of Fig. 21. Here the merge routine of Fig. 19, with parameters  $m$  and  $n$ , is the heart of the process. There will be two indices involved,  $\nu$ , indicating the number of overall mergings that have been completed, and  $\omega$ , the number of mergings of strings of length  $2^\nu$  that have been completed for this value of  $\nu$ . To use the subroutine, the main program furnishes values  $m_\omega$  and  $n_\omega$ , which are used as  $m$  and  $n$  in the merge routine, and the addresses of  $X^1$ ,  $Y^1$ , and  $Z^1$ . The resulting merged sequence  $\{Z^k\}$  is stored beginning at the address of  $X^1$ . (The manipulation of these addresses is not included in the merge routine, for the sake of simplicity, but it is an obvious extension of the flow diagram of Fig. 21.)

For the worst possible case, when the keys are present in exactly the reverse order, the number of comparisons required for sorting  $n$  items,  $C(N)$ , which may be considered a measure of the amount of effort needed to sort using this method, can be shown to be bounded from above:

$$C(N) \leq N \log_2 N$$

<i>Initial String</i>	<i>Sequences of 2</i>	<i>Sequences of 4</i>	<i>Sequences of 8</i>	<i>Sequences of 16</i>	<i>Sequences of 32</i>	<i>Sequences of 64 (or less)</i>
50	27	2	2	2	2	2
27	50	27	20	3	3	3
39	2	39	21	5	4	4
2	39	50	27	11	5	5
21	21	20	39	15	9	9
46	46	21	46	16	11	11
65	20	46	50	20	12	12
20	65	65	65	21	15	15
3	3	3	3	27	16	16
5	5	5	5	29	17	17
61	29	29	11	31	18	18
29	61	61	15	39	19	19
16	16	11	16	46	20	20
31	31	15	29	50	21	21
15	11	16	31	61	25	25
11	15	31	61	65	26	26
48	28	4	4	4	27	27
28	48	28	17	9	28	28
4	4	44	28	12	29	29
44	44	48	44	17	31	31
49	17	17	48	18	33	33
17	49	49	49	19	39	39
62	55	55	55	25	43	42
55	62	62	62	26	44	43
18	18	9	9	28	46	44
43	43	18	12	33	48	46
9	9	25	18	43	49	48
25	25	43	19	44	50	49
33	12	12	25	48	55	50
12	33	19	26	49	61	55
26	19	26	33	55	62	56
19	26	33	43	62	65	61
56	56	42	42	42	42	62
63	63	56	56	56	56	63
42	42	63	63	63	63	65

FIG. 20. An example of merge-sorting by pairs.

(It can be easily shown where  $N = 2^v$ ,  $v$  an integer, that:

$$C(N) = N \log_2 N - N + 1 \quad (N = 2^v)$$

and the bound can be extended by somewhat more complicated analysis.)

For the most favorable case, when the items are already sorted by key, the number of comparisons can be made as low as

$$C(N) \leq N.$$

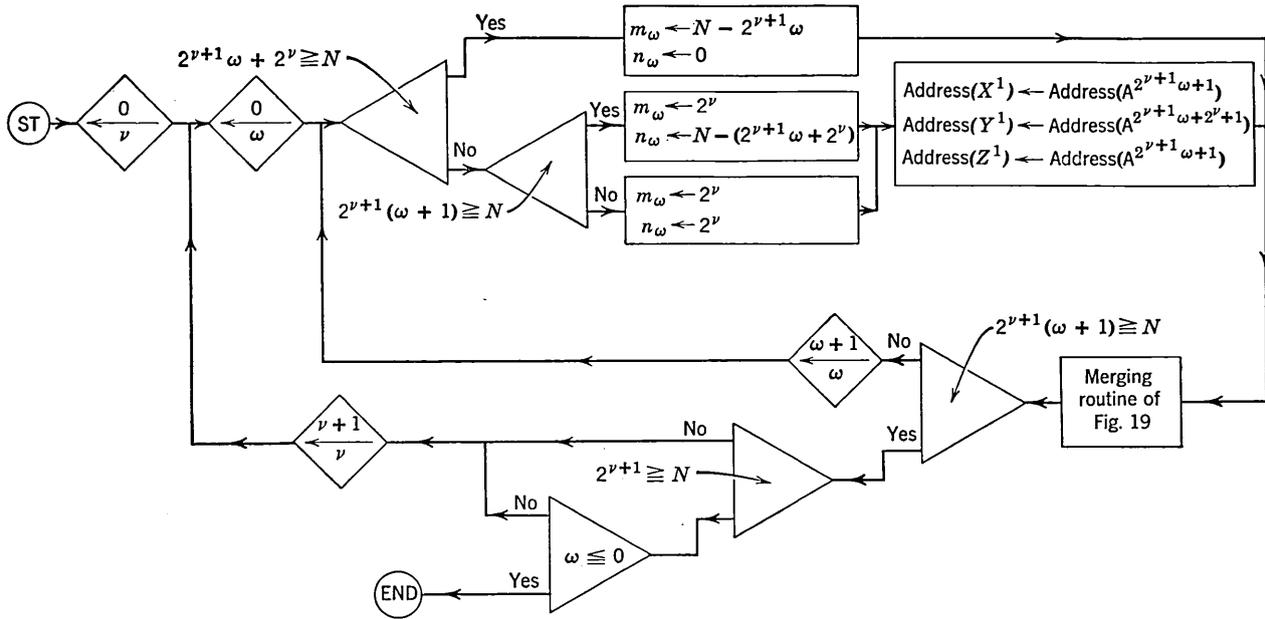


Fig. 21. Sorting a sequence of  $N$  items using merging.

This requires that the merging routine of Fig. 21 be reprogrammed in a more complicated fashion to take advantage of the possibility of an original string that is almost ordered.

For sorting external to a high-speed storage, where the number of comparisons is no longer dominant, but rather the amount of input and output, a similar process can be used, but here the strings resulting at the various stages of the merging process would be divided approximately equally on two (or more) output tapes. Upon passage through the entire data, the tape units being used for the output strings could now be rewound, and their information considered the input for a new merging process. If one item is read successively from each tape, keys compared, and the item with the smaller key sent to one of several output tapes, successively larger strings can be built up as with the internal storage procedure. If there are  $2^b$  tape units available, the number of passages through the entire information will be

$$\log_b N,$$

where  $b$  units are used for input and  $b$  for output. The detailed procedure in this case depends upon how many items can be held in the internal storage at one time. In general, it is best to perform internal sorts whenever possible and in as large a string as possible. If it is possible to hold only two items in the internal storage at once, it may pay to modify the von Neumann-Goldstine procedure to allow merging to continue to produce a string as long as a sequence of keys is monotone. This string would be read out on one of the output tapes. The next string, of arbitrary length depending on the sequence of keys read in, would be put out on another tape, etc. Upon exhaustion of the input information, the role of input and output tapes would be reversed.

### Other Internal Sorting Methods

Seward (Ref. 140) has collected statistics for various other methods of internal sorting:

**Finding the Smallest.** Find the smallest of a group  $N$ , and store it. Find the smallest of the group  $N - 1$ , and store it. Continue the process until the group is exhausted. The number of comparisons is

$$C(N) = \frac{N(N - 1)}{2}.$$

**Interchanging Pairs.** Compare, and interchange if necessary, the pairs beginning with an odd-numbered item (1, 3, 5,  $\dots$ ,  $N$ ). Repeat for the pairs beginning with even-numbered items. Alternate this process until no interchanges occur. The number of comparisons in the worst possible case is

$$C(N) \approx N^2/2.$$

**Sorting by Sifting.** Compare items in sequence, moving them forward in the list until an item with smaller key is reached. When the last item is "sifted" the items are ordered. The number of comparisons in the worst case is

$$C(N) \approx \frac{N(N-1)}{2}.$$

**Partial Sorting.** Items 1 and 2, 2 and 3,  $\dots$ ,  $j$  and  $j+1$ ,  $\dots$ , are compared, interchanging where necessary. The process is repeated until no interchanges occur. Again

$$C(N) \approx \frac{N(N-1)}{2}.$$

It is apparent from a more detailed study that the bounds on number of comparisons given by digital sorting or sorting by merging with

$$C(N) \approx N \log_2 N$$

are by far the best. Except in rare cases where much information is known about the a priori sequence of the items, only these two methods should be used.

### A Practical Example

As a practical example of a process in use with magnetic tape units on an actual equipment, the following procedure taken from a description of the use of magnetic tapes on the IBM 650 is given. (See Ref. 148.) A tape record on this machine (block stored on or read from tape) ranges from 1 to 60 words. It is assumed first that one item may be stored in a record. In this procedure:

1. The original sequence of items (here written on two tapes) is processed and written on two output tapes.
2. The two output tapes, considered as new input tapes, are merged to write two other output tapes. The process is repeated until completion.

At each step, two new records are compared with each other and with the last output record written out. If either one of these is in proper sequence with the last output record written out, it is written out on that tape. If not, a new sequence is started on the second tape using the item with the smallest key. The process is shown by an example, and in the flow diagram of Fig. 22. When no new sequence has been set up during a merging pass,  $\omega = 0$ , and the entire process is complete. In the case below, strings are separated by vertical lines.



EXAMPLE.

*First Pass*

Initial Tape 1      23, 13, 11, 18, 29, 4, 5, 30, 15, 10  
 Initial Tape 2      16, 6, 24, 2, 17, 22, 33, 9, 7, 28  
 Output Tape 3      16, 23 | 2, 11, 17, 18, 22, 29, 33 | 7, 15, 28  
 Output Tape 4      6, 13, 24 | 4, 5, 9, 30 | 10  
 These tapes now become input tapes 1 and 2.

*Second Pass*

Output Tape 3      6, 13, 16, 23, 24 | 7, 10, 15, 28  
 Output Tape 4      2, 4, 5, 9, 11, 17, 18, 22, 29, 30, 33 |  
 These tapes now become input tapes 1 and 2.

*Third Pass*

Output Tape 3      2, 4, 5, 6, 9, 11, 13, 16, 17, 18, 22, 23, 24, 29, 30, 33  
 Output Tape 4      7, 10, 15, 28  
 These tapes now become input tapes 1 and 2.

*Fourth Pass*

Output Tape 3      2, 4, 5, 6, 7, 9, 10, 11, 13, 15, 16, 17, 18, 22, 23, 24, 28,  
 29, 30, 33  
 Output Tape 4

8. AUTOMATIC PROGRAMMING: DEVELOPMENT AND OBJECTIVES

**Definition.** *Automatic programming* can be defined as all those methods which attempt to shift the burden of formulation and programming of problems for automatic computers onto the machines themselves.

**Automatic Coding Systems.** Bemer (Ref. 141) has collected a list of various automatic coding systems for computers which is reproduced in Table 23. This list shows that:

1. The amount of effort put into automatic coding systems has been large.

*Note.* Fig. 22 explanation.

Given four tape units numbered 1, 2, 3, 4:  
 Let  $m, m + 1$  be the numbers of the input tape units,  
 $n, p$  be the numbers of the output tape units,  
 $i$  be the index on tape  $m$ ,  
 $j$  be the index on tape  $m + 1$ ,  
 $k$  be the index on tape  $n$ ,  
 $l$  be the index on tape  $p$ .  
 Let  $I_{m,i}$  be the  $i$ th item on tape  $m$ , etc.,  
 $k(A)$  be the key for item  $A$ , etc.,  
 $i_0$  be the number of items on tape  $m$ ,  
 $i_0$  be the number of items on tape  $m + 1$ .

TABLE 23. AUTOMATIC CODING SYSTEMS (REF. 141)

Computer	System Name or Acronym	Developed by	Code	M.L.	Assem.	Inter.	Comp.	Oper. Date	Index- ing	Fl Pt	Symb.	Algeb.
IBM 704	AFAC	Allison G.M.	C				X	Sept. 57	M2	M	2	X
	CAGE	General Electric			X		X	Nov. 55	M2	M	2	
	FORC	Redstone Arsenal					X	June 57	M2	M	2	X
	FORTRAN <sup>a</sup>	IBM	R				X	Jan. 57	M2	M	2	X
	NYAP	IBM			X			Jan. 56	M2	M	2	
	PACT IA <sup>a</sup>	Pact Group <sup>b</sup>					X	Jan. 57	M2	M	1	
	REG-SYMBOLIC	Los Alamos			X			Nov. 55	M2	M	1	
	SAP <sup>a</sup>	United Aircraft	R		X			Apr. 56	M2	M	2	
	NYDPP	Serv. Bur. Corp.			X			Sept. 57	M2	M	2	
	KOMPILER 3	UCRL Livermore					X	March 58	M2	M	2	X
	IBM 701	ACOM	Allison G.M.	C			X		Dec. 54	S1	S	0
BACAIC		Boeing Seattle	A		X		X	July 55	—	S	1	X
BAP		Univ. of Calif., Berk.		X	X			May 57	—	—	2	
DOUGLAS		Douglas SM			X			May 53	—	S	1	
DUAL		Los Alamos		X		X		March 53	—	S	1	
607		Los Alamos			X			Sept. 53	—	—	1	
FLOP		Lockheed Calif.		X	X	X		March 53	—	S	1	
JCS 13		Rand Corp.			X			Dec. 53	—	—	1	
KOMPILER 2		UCRL Livermore					X	Oct. 55	S2	—	1	X
NAA ASSEMBLY		N. Am. Aviation				X						
PACT I <sup>a</sup>		Pact Group <sup>b</sup>	R				X	June 55	S2	—	1	
QUEASY		NOTS Inyokern				X		Jan. 55	—	S	—	
QUICK		Douglas ES				X		June 53	—	S	0	
SHACO		Los Alamos				X		Apr. 53	—	S	1	
SO 2		IBM			X			Apr. 53	—	—	1	
SPEEDCODING		IBM	R		X	X		Apr. 53	S1	S	1	
IBM 705-1, 2		ACOM	Allison G. M.	C			X		Apr. 57	S1	—	0
	AUTOCODER <sup>a</sup>	IBM	R	X	X		X	Dec. 56	—	S	2	
	ELI	Equitable Life	C			X		May 57	S1	—	0	
	FAIR	Eastman Kodak				X		Jan. 57	—	S	0	
	PRINT I <sup>a</sup>	IBM	R	X	X	X		Oct. 56	S2	S	2	
	SYMB. ASSEM.	IBM			X			Jan. 56	—	S	1	
	SOHIO	Std. Oil of Ohio		X	X	X		May 56	S1	S	1	
	FORTRAN	IBM-Guide	A				X	Nov. 58	S2	S	2	X
	IT	Std. Oil of Ohio	C				X		S2	S	1	X
	AFAC	Allison G.M.	C				X		S2	S	2	X

Computer	System Name or Acronym	Developed by	Code	M.L.	Assem.	Inter.	Comp.	Oper. Date	Index- ing	Fl Pt	Symb.	Algeb.
IBM 705-3	FORTRAN AUTOCODER	IBM-Guide	A				X	Dec. 58	M2	M	2	X
		IBM	A		X		X	Sept. 58	—	S	2	
IBM 702	AUTOCODER ASSEMBLY SCRIPT <sup>a</sup>	IBM		X	X		X	Apr. 55	—	S	1	
		IBM			X			June 54	—	—	1	
		G. E. Hanford	R	X	X	X	X	July 55	S1	S	1	
IBM 709	FORTRAN SCAT	IBM	A				X	Jan. 59	M2	M	2	X
		IBM-Share	R		X		X	Nov. 58	M2	M	2	
IBM 650	ADES II	Naval Ord. Lab					X	Feb. 56	S2	S	1	X
	BACAIC	Boeing Seattle	C		X	X	X	Aug. 56	—	S	1	X
	BALITAC	M.I.T.		X	X		X	Jan. 56	S1	—	2	
	BELL L1 <sup>a</sup>	Bell Tel. Labs		X		X	X	Aug. 55	S1	S	0	
	BELL L2, L3	Bell Tel. Labs		X		X		Sept. 55	S1	S	0	
	DRUCO I	IBM				X		Sept. 54	—	S	0	
	EASE II	Allison G.M.			X		X	Sept. 56	S2	S	2	
	ELI	Equitable Life	C			X		May 57	S1	—	0	
	ESCAPE	Curtiss-Wright			X	X	X	Jan. 57	S1	S	2	
	FLAIR	Lockheed MSD, Ga.		X			X	Feb. 55	S1	S	0	
	FOR TRANSIT <sup>a</sup>	IBM—Carnegie Tech.	A				X	Oct. 57	S2	S	2	X
	IT <sup>a</sup>	Carnegie Tech.	C				X	Feb. 57	S2	S	1	X
	MITILAC	M.I.T.		X		X		July 55	S1	S	2	
	OMNICODE	G. E. Hanford				X	X	Dec. 56	S1	S	2	
	RELATIVE	Allison G.M.			X			Aug. 55	S1	S	1	
	SIR	IBM				X	X	May 56	—	S	2	
	SOAP I	IBM			X			Nov. 55	—	—	2	
	SOAP II <sup>a</sup>	IBM	R		X			Nov. 56	M	M	2	
	SPEED CODING	Redstone Arsenal		X		X		Sept. 55	S1	S	0	
	SPUR	Boeing Wichita		X	X	X		Aug. 56	M	S	1	
FORTRAN (650T)	IBM	A				X	Jan. 59	M2	M	2	X	
Sperry Rand 1103A	COMPILER I <sup>a</sup>	Boeing Seattle			X		X	May 57	—	S	1	X
	FAP	Lockheed MSD		X		X		Oct. 56	S1	S	0	
	MISHAP	Lockheed MSD			X			Oct. 56	M1	S	1	
	RAWOOP-SNAP	Ramo-Wooldridge			X	X		June 57	M1	M	1	
	TRANS-USE	Holloman A.F.B.			X			Nov. 56	M1	S	2	
	USE <sup>a</sup>	Ramo-Wooldridge	R		X		X	Feb. 57	M1	M	2	
	IT	Carn. Tech.-R-W	C				X	Dec. 57	S2	S	1	X
	UNICODE	R Rand St. Paul	R				X	Jan. 59	S2	M	2	X

TABLE 23. AUTOMATIC CODING SYSTEMS (REF. 141) (Continued)

Computer or Acronym	System Name or Acronym	Developed by	Code	M.L.	Assem.	Inter.	Comp.	Oper. Date	Index- ing	Fl Pt	Symb.	Algeb.
Sperry Rand 1103	CHIP	Wright A.D.C.		X		X		Feb. 56	S1	S	0	
	FLIP/SPUR	Convair San Diego		X		X		June 55	S1	S	0	
	RAWOOP	Ramo-Wooldrige	R		X			March 55	S1	—	1	
	SNAP	Ramo-Wooldrige	R		X	X		Aug. 55	S1	S	1	
Sperry Rand Univac I and II	A0	Remington Rand		X	X		X	May 52	S1	S	1	
	A1	Remington Rand		X	X		X	Jan. 53	S1	S	1	
	A2	Remington Rand		X	X		X	Aug. 53	S1	S	1	
	A3, ARITHMATIC <sup>a</sup>	Remington Rand	C	X	X		X	Apr. 56	S1	S	1	
	AT3, MATHMATIC <sup>a</sup>	Remington Rand	C		X		X	June 56	S1	S	2	X
	B0, FLOWMATIC <sup>a</sup>	Remington Rand	A	X	X		X	Dec. 56	S2	S	2	
	BIOR	Remington Rand		X	X		X	Apr. 55	—	—	1	
	GP	Remington Rand	R	X	X		X	Jan. 57	S2	S	1	
	MJS (UNIVAC I)	UCRL Livermore		X	X			June 56	—	—	1	
	NYU, OMNIFAX	New York Univ.					X	Feb. 54	—	S	1	
	RELCODE	Remington Rand		X	X			Apr. 56	—	—	1	
	SHORT CODE	Remington Rand		X			X	Feb. 51	—	S	1	
	X-1	Remington Rand	C	X	X			Jan. 56	—	—	1	
	IT	Case Institute	C				X		S2	S	1	X
MATRIX MATH	Franklin Inst.					X	Jan. 58					
Sperry Rand File Comp.	ABC	R Rand St. Paul						June 58				
Sperry Rand Larc	K5	UCRL Livermore			X		X		M2	M	2	X
	SAIL	UCRL Livermore			X				M2	M	2	
Burroughs Datatron 201. 205	DATACODE I	Burroughs					X	Aug. 57	MS1	S	1	
	DUMBO	Babcock and Wilcox				X	X					
	IT <sup>a</sup>	Purdue Univ.	A				X	July 57	S2	S	1	X
	SAC	Electrodata		X	X			Aug. 56		M	1	
	UGLIAC	United Gas Corp.					X	Dec. 56	—	S	0	
STAR	Dow Chemical Electrodata			X		X						
Burroughs UDEC III	UDECIN-I	Burroughs				X		57	M/S	S	1	
	UDECOM-3	Burroughs					X	57	M	S	1	

Computer	System Name or Acronym	Developed by	Code	M.L.	Assem.	Inter.	Comp.	Oper. Date	Index- ing	Fl Pt	Symb.	Algeb.
M.I.T.	ALGEBRAIC	M.I.T.	R				X		S2	S	1	X
Whirlwind	COMPREHENSIVE SUMMER SESSION	M.I.T. M.I.T.		X	X	X X		Nov. 52 June 53	S1 S1	S S	1 1	
Midac	EASIAC MAGIC	Univ. of Michigan Univ. of Michigan		X	X		X X	Aug. 54 Jan. 54	S1 S1	S S	1 1	
Datamatic	ABC I	Datamatic Corp.					X					
Ferranti	TRANSCODE	Univ. of Toronto	R		X	X	X	Aug. 54	M1	S	1	
Illiac	DEC INPUT	Univ. of Illinois	R		X			Sept. 52	S1	S	1	
Johnniac	EASY FOX	Rand Corp.	R		X			Oct. 55	—	S	1	
Norc	NORC COMPILER	Naval Ord. Lab			X			Aug. 55	M2	M	1	
Seac	BASE 00 UNIV. CODE	Natl. Bur. Stds. Moore School		X		X						
							X	Apr. 55				

<sup>a</sup> Indicates present heavy usage.

<sup>b</sup> Pact group contains Douglas SM, ES, LB, Lockheed, NOTS, N. Am., Rand.

*Chart Symbols*

- Code
  - R = Recommended for this computer, sometimes only for heavy usage.
  - C = Common language for more than one computer.
  - A = System is both recommended and has common language.
- Indexing
  - M = Actual Index registers or B boxes in machine hardware.
  - S = Index registers simulated in synthetic language of system.
  - 1 = Limited form of indexing, either stopped unidirectionally or by one word only, or having certain registers applicable to only certain variables, or not compound (by combination of contents of registers).
  - 2 = General form, any variable may be indexed by any one or combination of registers which may be freely incremented or decremented by any amount.
- Floating point
  - M = Inherent in machine hardware.
  - S = Simulated in language.
- Symbolism
  - 0 = None.
  - 1 = Limited, either regional, relative or exactly computable.
  - 2 = Fully descriptive English word or symbol combination which is descriptive of the variable or the assigned storage.
- Algebraic
  - A single continuous algebraic formula statement may be made. Processor has mechanisms for applying associative and commutative laws to form operative program.
- M.L.
  - = Machine language.
- Assem.
  - = Assemblers.
- Inter.
  - = Interpreters.
- Compl.
  - = Compilers.

2. Most early efforts were in the direction of interpreters, but most efforts now are in the design of compiler-translators.

3. Many systems have been replaced by newer, more efficient languages.

### Development

Some of the most important steps in development of automatic programming are listed below. The ideas listed here were proposed by many persons; among those particularly to be mentioned are M. V. Wilkes, D. J. Wheeler, and S. Gill of Cambridge University, C. W. Adams of M.I.T., Grace Hopper of Remington Rand, and N. Rochester of IBM.

**1. Understandable Language. Translation, Mnemonic Codes, Compiling, and Interpretation.** The ambiguity of instructions and numbers inside machines makes instructions appear only as numerical sequences.

(a) Programs were devised which automatically translated both numbers and instructions from an external decimal language (more useful to human beings) into an internal binary language (more useful to the machine). The important step was the recognition of the principle of a dual language system and a programmed translation between the two languages, with the computer to perform the translation.

(b) Along with the numerical translations came the use of "mnemonic"—easy to remember—instruction operations, such as standard algebraic notations, or abbreviations of the corresponding English words.

These are now known as "input translation programs." The general idea of translation leads to two basic techniques: pretranslation or *compiling*, and running translation or *interpretation*.

(c) Compiling requires a large amount of medium access storage (generally magnetic tape). The compiling process usually occurs only once.

(d) Interpretation is the only feasible translation method in machines with small amounts of storage; this process is less efficient since the same translation may occur over and over again during performance of a problem.

The first attempts to improve the procedures of coding in machine language resulted in a set of input orders that changed alphanumerical sequences representing instructions on teletype tape over into internal binary machine notation. (Ref. 108.) Today, most of the automatic programming schemes make use of the pretranslation idea.

**2. Easy-to-Correct and Easy-to-Use Input Languages. Symbolic Addresses and Control Instructions.** The use of external languages that were easily understandable, such as mnemonic codes, does not prevent arithmetic, logical, or clerical programming mistakes. The translation process, developed to handle the requirement of an understandable input language, can also be applied to the use of so-called *sym-*

*boldic* or *floating* addresses, which have no permanent absolute machine internal counterparts. With these addresses, which are almost completely equivalent to an algebraic notation, assignment of addresses can be made completely automatic by the computers themselves.

The use of such automatic address assignment, it turns out, requires two "passes" or traverses through the input information in order (1) to find out what algebraic addresses are present and what their internal equivalent absolute addresses are, and (2) to assign these absolute addresses wherever the floating addresses occur (Ref. 21). (D. J. Wheeler has shown, for the EDSAC II, that two passes are not always necessary.) The assignment of absolute addresses cannot take place until *all* algebraic addresses are known. The existence of these two passes through information immediately presents opportunities to perform all sorts of other transformations on the input information. From this grew the concept of floating or symbolic address (Ref. 106). A similar system was developed shortly thereafter for the IBM 701 (Ref. 87), using a Dewey decimal type of symbolic addressing system.

In order to control the processes of correction, reassignment, and deletion, and to handle the process of translation, a new kind of instruction is required. So-called *control combinations*, which tell the translation program, rather than the computer hardware, what is to be done, give another dimension of latitude of expression between the programmer and machine. These new "tag words" allow the instruction process on input to expand indefinitely.

**3. Elimination of Repetitious Coding. Subroutines, Assembly Programs, and Synthetic Instructions.** To eliminate duplication of effort that occurs when similar problems are repeated often, routines were developed which performed standard operations. Such routines could be called on by any programmer without the necessity of being rewritten from the beginning. To make the most of such routines, complete generality was required. The floating addresses developed for correction purposes helped provide this feature; similar so-called *preset* and *program* parameters were devised to make such routines flexible. Such assembly programs alleviated much of the effort in making corrections to instructions by insertion or deletion. Because of the "floating" nature of the symbolic addresses which were retranslated at each new machine input, insertions and deletions caused no further requirements for changing other instruction addresses. With the advent of secondary storage in the form of magnetic drums and magnetic tape, the need for two inputs of the program punched tape or cards was eliminated; the two passes could be made completely inside the machine.

The combination of these standard subroutines (see Subroutines below), as they came to be called, with the input translation (compiling) techniques

brought forth the concept of *automatic assembly* of programs. Here code words called *pseudo instructions* or *synthetic instructions* are used to call in and store precoded subroutines in a main program. Such *open* subroutines, without automatic entry and exit, worked best with compiling techniques; *closed* subroutines, which act in the same unitized manner as an ordinary instruction, fitted most easily into the interpretive schemes.

Extension of assembly programs to the use of pseudo instructions that replaced one line of coding external to the machine by more than one inside, allowed an increase in the number and variety of instructions available to the programmer. Such efforts probably reached their peak in the PACT system (Ref. 6) developed for the IBM 701 by a group of Los Angeles users of that equipment. This system included automatic insertion of scaling instructions for fixed-point computations as well as a long list of pseudo instructions. A more recent example of such an assembly is the Share Assembly Program (SAP), and the X-1 assembly system for Univac (Ref. 100). Details of SAP are given in Sect. 9.

**4. Easy Mistake Discovery. Utility Programs.** Relatively straightforward methods of correcting mistakes did not speed up program checkout as much as would be expected, since before programming mistakes can be corrected, they must be found. Unless complete retranslation of stored information is made available to a programmer, he is forced to know and use *both* external and internal languages, which nullifies some of the advantages discussed under 1 above. Such retranslation adds to the complexity of the translation procedure, as well as causing possible ambiguities because of many-to-one input translations. Nevertheless, such retranslation can contribute to aiding the discovery of human programming mistakes.

However, there are many difficulties attendant with a retranslation to symbolic language in mistake diagnosis. One of these difficulties is that a complete directory or dictionary must be retained in the machine and consulted. Often, retranslation becomes difficult because a machine cell is not uniquely identifiable with a symbolic address. These difficulties have led, in many cases, to performing diagnosis in machine language. Unfortunately, this often leads the programmer to another difficulty: making corrections in machine language.

Diagnostic procedures fall into two categories: static or dynamic. Static procedures give results only at specific points during solution of a problem, usually only at the beginning, end, or both. "Sieved" or "changed-word post-mortems," or storage printouts, have proved a welcome use of the principle of machine screening of unnecessary information.

Dynamic mistake diagnosis, on the other hand, has often suffered from the fact that it has of necessity been interpretive and, hence, machine-time consuming, or else that it has put out information in an unretranslated

form. A new concept in dynamic procedures, which give results as the problem is being performed, combines built-in automatic switching with programmed retranslation to speed up this process and make it competitive in time. This is discussed in Sect. 10 under Utility Programs.

**5. Prevention of Mistakes before They Occur. Generators.** The widespread use of precoded subroutines, which had been checked thoroughly for both arithmetic, logical, and clerical mistakes, is an example of a preventative technique. Instead of storing complete programs, however, a more efficient method appears to store programs called *generators* that can generate large classes of programs. If a correct algorithm can be developed by which a computer can generate a general class of small problems, mistake-free codes can be produced directly. Similarly, automatic assembly and automatic subroutine call-in techniques, if correct themselves, will generate mistake-free programs.

**6. Unification of Techniques. Combined Systems.** Systems using portions of the techniques listed above have been developed (Ref. 8). Certain of the categories are directly opposed. Hence, the useful combinations are employed with the objective of a self-sufficient method of machine operation that requires a minimum of human intervention.

**7. Universal Computer Language.** The progress through the steps listed above leads, of necessity, toward some sort of standardization of the basic input language of all computers, as it looks to the users, before the computer matches the internal language to the external human language. Present day lack of compatibility between algorithms or programs developed on one computer and another is causing as much undue duplication of effort in space as occurred in time before the advent of subroutines. Universal languages, more or less standardized, exist for certain types of problems, for example, those that can be expressed entirely algebraically.

## 9. AUTOMATIC PROGRAMMING: ASSEMBLY PROGRAMS

### Structure and Objectives

Basically an assembly program accommodates programs written in machine language. However, it allows the programs to be written with certain flexibilities and conveniences not available in pure computer language. The assembly program usually provides for the following:

1. Specification of all numerical constants in convenient decimal form. (Provision is usually made, however, to allow the writing of binary numbers in appropriate cases.)
2. The use of symbolic addresses, addresses with mnemonic content.
3. The use of mnemonic two- or three-letter pairs to describe machine instructions.

4. The use of free addresses for instructions to allow the arbitrary naming of an instruction without stating its location in the storage.

5. The automatic assembly of precoded programs (subroutines) into the program, often by the use of pseudo instructions.

Two kinds of assembly programs are possible:

1. *One-pass assemblers*, where the computer, in translation, makes one pass through the data to perform the required translation.

2. *Two-pass assemblers*, where the computer, in translation, first makes a pass through the data to discover all symbolic addresses and to determine subroutines to be used. The symbolic addresses are assigned machine addresses, and the subroutines are included in the second pass.

If symbolic addresses are used, and they are in almost every assembly program, a *directory* must precede the data in the case of a one-pass compiler. This directory is simply a list of symbolic addresses with corresponding machine addresses; the directory tells the assembly program how to assign the machine addresses. The directory is developed during the first pass through the data, through the technique involving the use of a *location counter* (see Assembly Procedure, below).

The difference between an assembly program and a compiler, discussed in following sections, is tenuous. A compiler is considered to allow more complex operations. It usually expands a convenient problem-oriented language (such as algebraic formulas) into machine language. However, the difficulty in terminology arises when the assembly program allows many powerful pseudo operations which result in appropriate subroutines included into the program. The compiler, in its truest sense, usually makes no provision for machine instructions, or at least deemphasizes them, while the assembly program, as the name implies, assembles pieces of machine instruction programming.

Two assemblers are prominent; the USE Compiler (Ref. 152) prepared for the Univac Scientific Exchange (USE) for the Univac 1103A by The Ramo-Wooldridge Corporation; and SAP, Share Assembly Program, prepared for the Share Cooperative Programming Group for the IBM 704 by the United Aircraft Corporation. Both these compilers are of the two-pass type, as are nearly all modern assembly programs. An assembly program of the one-pass variety was RAWOOP (Ramo-Wooldridge One-Pass) assembler (Ref. 8). The SAP program is described in some detail below.

### Assembly Procedure

The procedure of assembly is in two parts:

1. Examination of the program to be assembled in order to define each symbol used in writing the program. A location counter *L* is used to specify the absolute location of each word (number or instruction) in the program. *L* is set initially to an integer common to all programs, or in exceptional

cases, supplied to the assembly program by the program being assembled. Thereafter, each new instruction input increases the contents of  $L$  by one. Simultaneously a table or directory is constructed. Each directory entry gives an equivalence relationship between a symbolic address  $S$  and the corresponding absolute machine address  $L_S$  assigned by the counter at the time of input. Entries in the table may also be made by means of certain pseudo operations. The order of the absolute instructions produced by the assembly program is governed only by the order in which information is input.

2. During the second assembly pass, the value of the counter  $L$  is computed in the same manner as on the first pass. In addition, replacements are made for symbolic addresses by integers stored in the directory as a result of the first pass.

### Share Assembly Program (SAP)

This assembly program was written for the IBM 704 computer (Ref. 143). Instructions for this system are written with addresses expressed as combinations of symbols and decimal integers. A typical IBM 704 instruction has operation, address, and a tag and decrement to be used with that machine's index registers. In addition to instructions, data in decimal, octal (the IBM 704 is a binary machine internally), or Hollerith alphanumeric code may be read from punched cards. Previously coded library routines may be conveniently inserted into a program whenever desired.

A number of pseudo operations which help carry out the assembly process are as follows:

1. **Origin specification (ORG)**. The location counter  $L$  is set to the value of the expression (previously defined symbol) in the address portion of the instruction. (This allows gaps in the sequence of input assignment at any stage.)

2. **Equality (EQU)**. The symbol appearing to the left of the operation code is assigned the integer value given by the previously defined expression in the address portion of the instruction. (This allows programmer assignment of preset parameters.)

3. **Synonym (SYN)**. The symbol appearing to the left of the operation code is assigned the integer value given by the previously defined expression in the address portion of the instruction. (This allows programmer assignment of blocks of storage.)

4. **Decimal data (DEC)**. The decimal data following are to be converted to binary and assigned to consecutive locations  $L, L + 1, \dots$ . Successive words of data on a card are separated by commas. Depending on whether or not each number contains an exponent or not, it will be translated into a floating point, or scaled fixed point number, or fixed point integer.

5. **Octal data (OCT)**. The octal data following are to be converted to binary integer form, and assigned to consecutive storage locations,  $L, L + 1, \dots$ .

6. **Hollerith data (BCD)**. The 10 six-character words of Hollerith (binary-coded-decimal) information are read and assigned to locations  $L, L + 1, \dots$ .

7. **Block started by symbol (BSS)**. The block of storage from  $L$  to  $L + N - 1$ , where  $N$  is the value of the expression following the operation, is reserved, and any symbol preceding the operation is assigned the value  $L$ .

8. **Block ended by symbol (BES)**. Similar to BSS, except any symbol preceding this operation is assigned the value  $L + N$ .

9. **Repeat (REP)**. Two expressions, separated by a command, following this operation, define integers  $M$  and  $N$  such that the block of instructions or data preceding the REP operation in locations  $L, L + 1, \dots, L + M - 1$  is repeated  $N$  times, stored in locations  $L + M, L + M + 1, \dots, L + (MN) - 1$ .

10. **Library search (LIB)**. The library routine of  $k$  words, identified by the symbol preceding the operation, is obtained from a library tape and inserted in the program being assembled at locations  $L, L + 1, \dots, L + k - 1$ . Any symbols appearing in the library routine are entered in the directory and properly defined.

11. **Heading (HED)**. If two or more programs use the same symbolic addresses, they may be combined with this heading pseudo instruction which prefixes each symbol used in the following program by the single character given in the usual address position. Thus nonunique designations are made unique.

12. **Define (DEF)**. This pseudo instruction assigns the value of the expression in the address position to any subsequent undefined symbols in successive integer order.

13. **Remarks (REM)**. Any Hollerith (alphanumeric) characters following this operation will be printed in the output listing of the assembled program, but not processed in any other way.

14. **End of program (END)**. The value of the expression in the address position is punched as the transfer (starting) address in an IBM 704 binary transfer card, the last in the output deck.

The input to this assembly program, on the IBM 704, is either a binary-coded-decimal (alphanumeric) magnetic tape, previously generated by peripheral card-to-magnetic-tape equipment, or punched cards.

Output is on binary nonrelocatable (fixed address) or relocatable punched cards (in machine storage) along with a printed copy of the entire program with or without library routines suppressed.

Additions to an already assembled program can be made if the table of symbols (directory) punched out during assembly is available. Upon

reloading this table, additional new parts may be assembled. Any change to the original program which does not require relocation of any part of the program or reassignment of any symbols, may be made by assembly of only those parts of the program requiring change.

**An Example of a SAP Program.** A SAP program for evaluation of a quadratic form

$$P_n(x, y) = \sum_{j=0}^n \left[ \sum_{i=0}^{i+j=N} a_{ij} X^i \right] y^j$$

is given in Table 24. The first column (*a*) gives the octal machine location, the next four columns (*b*) (12 octal digits with sign) give the translation, the next column (*c*) the symbolic location, operation or pseudo-operation, and symbolic address, tag, and decrement. The last column (*d*) explains the instruction. The reader is referred to the instruction list for the IBM 704 for the meaning of the operation codes. (Sect. 6.)

TABLE 24. A SAP PROGRAM FOR EVALUATION OF QUADRATIC FORM

<i>a</i>	<i>b</i>				<i>c</i>		<i>d</i>
				04000		ORG 2048	
04000	-0	53400	5	04011		LXD P1, J + K	Initialize index registers
04001	-0	63400	4	04020	P4	SXD P2, K	Store K
04002	0	50000	1	04022		CLA A + 1, J	Obtain first element
04003	1	77777	1	04004		TXI P6, J, -1	X
04004	-2	00001	4	04017	P6	TNX P5, K, 1	X
04005	0	76500	0	00043	P3	LRS 35	Form polynomial
04006	0	26000	0	04046		FMP X	In X
04007	0	30000	1	04022		FAD A + 1, J	X
04010	1	77777	1	04011		TXI P1, J, -1	Step coefficient
04011	2	00001	4	04005	P1	TIX P3, K, 1	Test reduced K
04012	0	60100	0	04051		STO S	Store partial sum
04013	0	56000	0	04050		LDQ Z	Form polynomial
04014	0	26000	0	04047		FMP Y	In Y
04015	0	30000	0	04051		FAD S	X
04016	-3	77754	1			TXL OUT, J, -R/2 + 1	X
04017	0	60100	0	04050	P5	STO Z	X
04020	1	00000	4	04001	P2	TXI P4, K	X
				00005		N EQU 5	
				00052		R EQU N*N + 3*N + 2	
				04021	A	BSS R/2	
04046	0	00000	0	00000		X	[Note that the $a_{ij}$ 's are stored in the order $a_{05}$ , $a_{14}$ , $a_{04}$ , $a_{23}$ , $a_{13}$ , $a_{03}$ , $\dots$ , $a_{00}$ from location A on.
04047	0	00000	0	00000		Y	
04050	0	00000	0	00000		Z	
04051	0	00000	0	00000		S	
				00001	J	EQU 1	
				00004	K	EQU 4	
				04000		END P4 - 1	
				00000		OUT	

## 10. AUTOMATIC PROGRAMMING: SUBROUTINES, SUBROUTINE GENERATORS, UTILITY PROGRAMS, AND INTEGRATED SYSTEMS

*Subroutines* are precoded, pretested, conventional coded programs, which can be used over and over again in many different programs and by many different machine users. Subroutines were first used with the Bell relay

computers (Ref. 96); their use with variable stored program machines was first expounded by Burks, Goldstine, and von Neumann (Ref. 19). The EDSAC group first experimented successfully with subroutines on a machine (Ref. 108).

**Synthetic Instructions.** A *synthetic* or *programmed instruction* is a subroutine or precoded combination of real instructions that may be treated conventionally as an instruction. For example, some machines have built-in instructions for square rooting. Such instructions are real instructions; in other machines exactly the same operations can be performed only by a subroutine.

**Structure of Subroutines.** In structure any subroutine is nothing more than a function of one or more variables. These variables, which are considered to be the parameters of the subroutine, may be numbers, addresses, or very infrequently, actual instructions. The variables for a subroutine can be either free or floating variables (see Sect. 4). A *parameter* to be used with a subroutine or any coded program is a variable of the program. A *program parameter*, as defined by Wilkes *et al.* (Refs. 108 and 109) is the machine counterpart of the *floating variable*. Such a parameter is thus a number calculated elsewhere in a program, whose value is not known until just before the performance of the subroutine, when it must be assigned. A *preset parameter*, on the other hand, corresponds to a *free variable*, assigned at the beginning of a problem and remaining constant from one performance of the subroutine to the next.

For example, if a subroutine is to be used to calculate a function to the same precision each time it is performed, the constant determining the precision of the result could be assigned at the beginning as a preset parameter. On the other hand, if the tolerance must be changed at each performance of the subroutine, it is actually a free variable, and must be inserted as a program parameter.

**Open Subroutines.** An *open subroutine* is a combination of instructions which must be inserted directly into a larger program. For each successive operation of the subroutine another copy or set of instructions must therefore be stored. The open subroutine has been most thoroughly investigated by the programmers of the Univac staff, including Hopper and Ridgway (Ref. 2), in the larger framework of the compiler or "compiling routine." Its advantage is its simplicity of entrance and exit (control moves sequentially in succession into, through, and out of the subroutine), which eliminates the need for red tape operations. The chief disadvantage is that successive copies must be stored for successive operation of the subroutines, which require a large amount of storage. The success of the Univac compiler schemes came because of the availability of large amounts of magnetic tape storage that could be handled efficiently. A sample flow diagram for successive open subroutines is shown in Fig. 23.

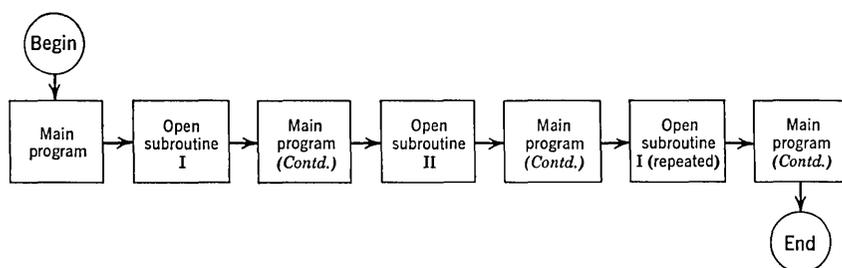


FIG. 23. Use of open subroutines interspersed with hand-coded program.

**Closed Subroutines.** The *closed subroutine* eliminates the duplications of storage required for open subroutines. One closed subroutine can perform the same function of many copies of an open subroutine. A closed subroutine is not stored in its proper place in the linear operational sequence in storage, but occupies a storage position away from the main coded program which is to refer to it. Such a subroutine is entered by a *change of control* operation which sends control to the entrance (often the first) instruction of the subroutine. After completion of the operations required by the subroutine, it is then necessary for it to return control to the instruction following the change of control instruction which transferred control to it originally. If this process is done automatically, as it generally is, the subroutine is called a *closed automatic subroutine*. The flow diagram for such a subroutine used several times during a program is shown in Fig. 24.

**Automatic Exit from a Subroutine.** The methods providing for automatic exit from a subroutine are in general two, depending on the absence or presence of a built-in instruction to aid the process. For an automatic return of control, the subroutine needs to know, as one of its floating variables, the setting of the control counter or instruction counter at the change of control instruction that originally transferred control. On the machines which have no special operation for this purpose, this contents of control must be considered a program parameter. The process of entering such parameters into a subroutine has been labeled *preliminary preparation* of the subroutine for use.

*Wheeler Method* (Ref. 108). This standard method of preliminary preparation makes use of the following preparatory sequence of operations:

1. An instruction is stored in a standard location (usually in a machine with a single-address code, the accumulator). The instruction stored is that one which performed the storage.
2. Addition of an instruction considered a constant (usually containing *three* in the address portion) to the stored instructions so as to make the result a change of control instruction.

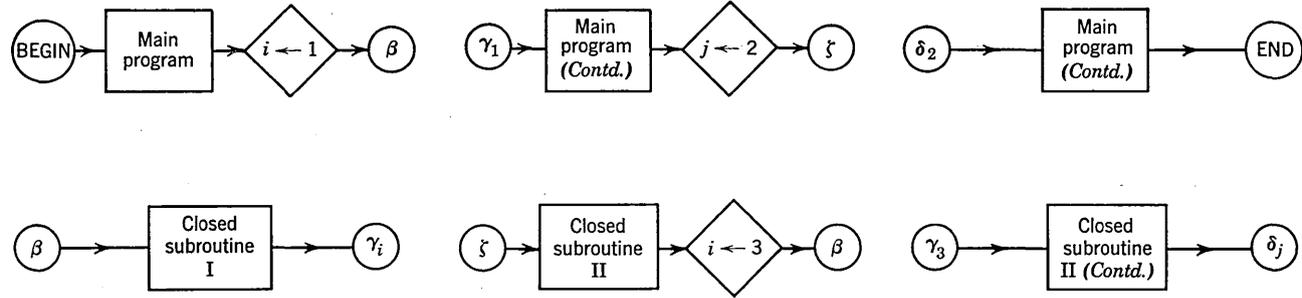


FIG. 24. Use of closed subroutines in a program. (Here the main program calls on subroutine I, and also on subroutine II, which in turn calls on subroutine I.)

3. Entry into the subroutine by a standard change of control instruction.

This procedure usually leaves a change of control instruction containing as its address portion the proper return address for the subroutine. The first instruction of the subroutine must then be to store this instruction in the proper exit location of the subroutine.

*Deferred Preparation.* This method can be used only when the instruction code for the computer contains an operation designed especially to accomplish this deferred preparation (Ref. 17). Two machines with such an instruction are the Whirlwind I (single-address) and the MIDAC (three-address). The Whirlwind I change of control instructions *subprogram* (sp) and *conditional program* (cp) automatically transfer the present location of control (contents of the control counter), increased by one, to a standard position, the machine's A register (Ref. 57). The first instruction of the subroutine must be a *transfer address*, ta n instruction (where n is a storage location), which stores the address given in the A register in the address position of storage location n, which is usually the exit change of control instruction. When this instruction is processed, control will transfer to the location following the original call-in instruction.

On the MIDAC, one three-address instruction, *file* (fi), performs the deferred preparation (Ref. 25). This change of control operation transfers control to the subroutine entrance, at the same time storing the return address in the proper position in the exit instruction or in a previously agreed upon conventional location.

**Subroutine Specifications Manuals.** How subroutines are used depends on the storage medium on which they are stored. Generally subroutines have been stored in two or more fashions at once. Usually a written copy of the subroutine is stored in a subroutine specifications manual compiled in a looseleaf notebook form. Each subroutine is usually explained thoroughly on a subroutine specification sheet which gives a thorough statement of its characteristics, including:

1. Preset and program parameters to be assigned.
2. Method of such assignment.
3. Length of subroutine.
4. Entry of subroutine.
5. Exit of subroutine.
6. Type (open or closed).
7. Time for execution expressed as a function of the parameters.
8. Precision of the results.
9. Description of the numerical process involved.

A second sheet, containing the actual coded program for the subroutine, is also kept on file in the subroutine specifications manual. The programmer who desires to know the actual behavior of a subroutine may use this

manual as a reference. An example of a subroutine library is given in Table 25. Examples of typical storage specifications forms for the MIDAC and Univac 1103 computers are given in Tables 26 and 27.

TABLE 25. SUBROUTINE LIBRARY FOR THE MIDAC COMPUTER

The following is a list of all the subroutines available for use on MIDAC. All subroutines are available on paper tape and seven have also been stored permanently on the drum. The specification sheets for the subroutines marked with an asterisk have been included under Tape Storage Subroutines and those marked "Drum" have been included under Drum Storage Subroutines.

- A. Printout
  - 1. Printout subroutine: Binary to decimal (adjustable format), drum
  - 2. Printout subroutine: Binary to decimal (fixed format)
  - 3. Graphical output
- B. Number conversion
  - 1. Fixed point number conversion: Binary to decimal (integers), drum
  - 2. Fixed point number conversion: Decimal to binary
  - 3. Standardized number conversion: Binary to decimal (approximate)
  - 4. Standardized number conversion: Binary to decimal (exact)
  - 5. Fixed point double precision conversion: Decimal to binary (fractions)
  - 6. Floating point number conversion: Binary to decimal (approximate)
- C. Interpretive routines
  - \*1. Floating point interpretive routines
  - \*2. Interpretive routine for operations on complex numbers
- D. Logarithm and exponential routines
  - 1. Natural logarithm, drum
  - 2. Exponential (for fractional exponents only), drum
  - \*3. Exponential (for fractional and integral exponents)
- E. Square root
  - 1. Square root (fixed point, single precision), drum
  - \*2. Square root (floating point numbers)
- F. Trigonometric routines
  - 1. Sine
  - 2. Cosine
  - 3. Sine, cosine, drum
  - 4. Arc tangent (polynomial approximation)
  - 5. Arc tangent (Taylor's series), drum
- G. Random numbers
  - \*1. Normal random deviates
- H. Integration
  - \*1. Integration by Simpson's rule
- I. Matrix routines
  - 1. Transpose of a square matrix
  - \*2. Transpose of a rectangular matrix
  - 3. Matrix multiplication for square matrix
  - \*4. Matrix multiplication for rectangular matrices
  - 5. Jacobi diagonalization
- J. Linear systems
  - 1. Seidel code
  - \*2. Jordan elimination

TABLE 26. MIDAC SUBROUTINE SPECIFICATION SHEET

Title	Normal Random Deviates	Tape Classification Machine Code	14D20 m2 Closed Relative Address
No. of consecutive registers required	19 (exclusive of registers 510 and 511)		
No. of instructions and constants	19		
Parameter (see description)	$D_i$	Location 510	Range $0 < D_i < 2^{44}$ Scaled $2^{-44}$
Results	$D_{i+1}$ $N_i$	510 511	$0 < D_{i+1} < 2^{44}$ $N_i \leq 6$ $2^{-44}$
Method of entry	Standard		
No. of modified instructions	2		
Error halts	None		
Description			

When a correct value of  $D_i$  (as explained below) is placed in register 510, the result,  $N_i$ , is a pseudo random number from a normal distribution with mean 0 and a standard deviation of 1. The routine takes  $R_{i,j} = R_{i,j} \cdot 5^{17}$  (low order 44 bits). To obtain  $N_i$  the high order 14 bits of the ( $j = 1, \dots, 12$ ) are extracted, shifted, added together and the result subtracted from 6.  $2^{-4} \cdot D_0 = 5^{17}$  and the  $D$ 's are defined by  $D_{i:1} = R_{i,j+12}$ . A correct value of  $D$  must be placed in register 510 before each entry to the subroutine. If the contents of cell 510 are not changed between successive files to this subroutine a correct value of  $D$  will already be in that cell. Otherwise  $D_{i:1}$  must be transferred and saved by the program and replaced in cell 510 the next time this subroutine is to be used. For programs requiring more than one period on the machine the last  $D_{i:1}$  may be printed out and read in again at the beginning of the next run. However, since an error in a single bit or digit during readout, transcription, or read in might destroy the validity of all further runs, the following table of correct starting values of  $D$  is supplied.  $D_{1000n}$  ( $n = 0, 1, 2, \dots$ ) are tabulated.

Coded by Bauer and Perkins  
 Checked by  
 Date March 11, 1954

*Hexadecimal listing*

$n$	$D_{1000n}$
0	0b1a2bc2ec5
1	0e35a475d45
2	3237a9fbc5
3	2156d657a45
4	9a6ec3868c5
5	0f800b89745
6	843948605c5
7	ac4a140b445
8	4a3d088a2c5
9	cd9dbfdd145
10	31a5d403fc5
11	1bb3defee45
12	39cb7acdce5
13	e1154170b45
14	e65eccc79c5
15	da9ab732845
16	2d609a516c5
17	076d1044545
18	0b21b30b3c5
19	79051ca6245
20	8e4237150c5
21	232bac57f45
22	89b5066edc5
23	abf98f59c45
24	6ab8e118ac5
25	7bd795ab945
26	08df47127c5
27	4d7e8f4d645
28	7609085c4c5
29	7df74c3f345
30	ce66f4f61c5

=  $5^{17}$

TABLE 27. A UNIVAC 1103 SUBROUTINE SPECIFICATION SHEET

Linear Matrix Equation Solver ( $AX = B$ )

	Specifications	
Identification tag:	MTI-0	
Type:	Subroutine available on cards for assembly	
Storage:	217 instructions, addresses	
	10M00 (00M00) thru 10M51 (0M51)	
	11M00 (01M00) thru 11M37 (01M37)	
	12M00 (02M00) thru 12M63 (02M63)	
	13M00 (03M00) thru 13M62 (03M62)	
	12 constants in program, addresses	
	C1N00 (C0N00) thru C1N11 (C0N11)	
	Temporary storage used, but not stored in program (see text).	
	229 words total program storage.	
	The constant pool and temporary storage pool are used by this routine.	
Program entrance:	Address 10M02	
Program exit:	Address 10M01	
Alarm exit:	The alarm exit is used by this routine.	
Machine time:	For all storage in ES time is approximately (in milliseconds):	
	$.3n^3 + .9n^2m + 1.7n^2 + .3m^2 + 2.5nm$	
	$+ 1.8n + 1.6m + 2.7(\text{matrix size} = n \times m)$	
	For temporary storage (see text) on drum add approximately (in milliseconds):	
	$.04[n^3 + 4n^2m + 3n^2 + 10mn] + 51$	
Mode of operation:	Fixed point	
Coded by:	W. L. Frank	October 25, 1955
Code checked by:	W. L. Frank	November 15, 1955
Machine checked by:	W. L. Frank	November 17, 1955
Approved by:	W. F. Bauer	November 30, 1955

*Description*

This subroutine solves the linear matrix equation  $AX = B$ , where  $A$  is a non-singular matrix of size  $n \times n$  and  $B$  has the dimensions  $n \times m$ . The solution,  $X = A^{-1}B$ , is a matrix of size  $n \times m$ . For the special case, when  $B$  is the identity matrix ( $I$ ), one obtains the inverse of the matrix  $A$ . Otherwise, one can solve  $m$  sets of  $n$  simultaneous linear equations in  $n$  unknowns.

Considerable flexibility is afforded the programmer with respect to the storage of the matrices  $A$ ,  $B$ , and the answer  $X$ . The programmer *must* code two auxiliary routines as follows:

- (a) The first must provide successive rows of the augmented matrix  $[A, B]$ . (When  $B = I$ , one only need supply rows of  $A$ .) Each row, consisting of  $(n + m)$  elements (or  $n$  elements when  $B = I$ ), must be set up in the fixed location immediately following the subroutine. This data must be scaled at  $2^{35}$  and be such, that for all elements  $a_{ij}$  of  $[A, B]$

$$|a_{ij} \cdot 2^{35}| \leq 2^{34}$$

In the general case, for  $B \neq I$ , the rows of  $[A, B]$  may be scaled independently. However, in the case of inverting a matrix, it is necessary that the entire matrix be scaled by the same factor.

- (b) The second auxiliary must take the successive columns of  $X$ , found in the  $n$  cells immediately following the routine, and either store them internally or punch them out. Since the columns of  $X$  are independently calculated, each has an associated scale factor (scaled at  $2^p$ ). This parameter positions the binary point (assuming the input matrices are scaled at  $2^{35}$ ) and is to be found in the  $(n + 1)$ st cell following the routine. If one has inverted a matrix, and if the input rows were originally scaled by  $10^p$  (or  $2^p$ ), then the output columns must be rescaled by  $10^p$  (or  $2^p$ ). These auxiliary routines are automatically entered  $n$  and  $m$  times respectively by RJ instructions. The subroutine sets up these two RJ instructions from information gleaned from the parameters of the entry. This procedure allows storage of  $A, B$ , and  $X$  on ES, MD, magnetic tape or externally on cards or tape. It is also possible to generate the elements of successive rows when a functional relation exists. In addition to the 229 words of storage needed by the subroutine, it is necessary to provide  $2(n + m)$  cells temporary storage immediately following the subroutine, and a block of  $[n(n + 1)/2] + nm$  cells, *either all on ES or all on MD*.

#### Operating Instructions

- Entrance to the subroutine is made by the following orders ( $B \neq I$ ):

$p$	RJ	00M01	00M02
$p + 1$	00	00X00	00Y01
$p + 2$	—	uuuuu	vvvvv
$p + 3$	—	—	xxxxx

where 00M00 is the location of the first word of the subroutine

00X00 is the location of the first word of the first auxiliary

00Y01 is the location of the second word of the second auxiliary

uuuuu =  $m$  (number of columns of  $B$ )

vvvvv =  $n$  (number of rows of  $A$ )

xxxxx = is the location of the first cell of the block of  $[n(n + 1)/2] + nm$  cells *all in ES or all in MD*.

- For the case when  $B = I$ , the  $p + 1$  word must be 40 00X00 00Y01
- The auxiliary routines must be available and coded so that they can be entered with

RJ 00X00 00X01

and

RJ 00Y00 00Y01 respectively.

This implies that the first and second words of both auxiliaries are exit and entrances respectively.

#### Alarm Conditions

Two alarm conditions can result:

- A test is made to see that all elements,  $a_{ij}$  of the input rows are within the limits

$$|a_{ij} \cdot 2^{35}| \leq 2^{34}$$

If this is violated the alarm routine ALR-1 is entered and "alarm-xxxx" is printed where xxxx-3 is the address of the cell from which the subroutine was entered.

2. If a singular matrix is detected in the process of inversion, the alarm routine ALR-1 is entered and "singul-wwww" is printed where wwwww-3 is the address of the cell from which the subroutine was entered. The routine can not, however, detect all singularities due to roundoff errors (see below).

Starting again at xxxx + 1 will cause the rest of the main program to be obeyed.

#### *Machine Time*

The machine time is as indicated on the first page when all operations are carried on in ES. This time is exclusive of the times taken by the auxiliaries. In case the block of  $[n(n+1)/2] + nm$  words are stored on MD, the time must be increased by the terms indicated.

These times are approximate and will be a minimum in most cases. Sample computation times for matrices of order 27 and 99 were respectively 53 seconds and 30 minutes.

#### *Mathematical Method* (Gauss elimination method)

Elementary row operations are performed on the matrix  $A$  reducing it to an upper triangular matrix  $\bar{A}$ . At the same time, these operations are performed on the matrix  $B$  giving a new matrix  $\bar{B}$ . A partial floating point arithmetic is maintained, in that the rows of the augmented matrix  $[A, B]$  are always kept within the limits such that the largest element of the row (in absolute value) lies in the interval

$$2^{34} > |a_{ij} \cdot 2^{35}| \geq 2^{33}$$

In addition, before eliminating, leading elements of two rows are compared and the element of largest magnitude becomes the pivotal point. Next, successive columns of  $\bar{B}$  are taken and the equation  $\bar{A}X = \bar{B}$  is solved by the back substitution procedure.

Singularities in  $A$  are detected if a zero appears on the diagonal of  $\bar{A}$ . Since roundoff errors can prevent this from occurring, one must inspect the size of the scale factor if  $A$  is suspected of being singular. Ill-conditioned matrices will cause the scale factors to be very small. That is, the elements of  $X$  will be very large.

#### *Accuracy*

The accuracy in the result is a function of the condition of the matrix  $A$ . Seven to eight decimal place accuracy was obtained for matrices of order 10 to 16. A matrix of order 39 and 99 yielded 7 and 6 place accuracy respectively.

The program for this problem on the Univac 1103 is given in Table 28.

TABLE 28. UNIVAC 1103 PROGRAM FOR LINEAR MATRIX EQUATION SOLVER

D	10M00	00100			144	00	00000	00000	
D	11M00	00152			230	00	00000	00000	
D	12M00	00190			276	00	00000	00000	
D	13M00	00254			376	00	00000	00000	
D	00M00	00100			144	00	00000	00000	
D	01M00	00152			230	00	00000	00000	
D	02M00	00190			276	00	00000	00000	
D	03M00	00254			376	00	00000	00000	
D	C1N00	00317			475	00	00000	00000	
D	C0N00	00317			475	00	00000	00000	
10M00	37	75701	75702	B	ALARM AND	144	37	75701	75702
10M01	MJ	00000	0		NORMAL EXIT	145	45	00000	00000
10M02	54	00M01	20017	BRB	ENTRY	146	54	00145	20017
10M03	TU	A0000	00M11		P-1	147	15	20000	00157
10M04	TU	A0000	01M09			150	15	20000	00241
10M05	AT	00015	A0000		P-2	151	35	00017	20000
10M06	TU	A0000	00M21			152	15	20000	00171
10M07	AT	00015	A0000		P-3	153	35	00017	20000
10M08	TU	A0000	00M19			154	15	20000	00167
10M09	TU	A0000	01M02			155	15	20000	00232
10M10	TP	A0000	01M03			156	15	20000	00233
10M11	TP	00000	A0000		SET	157	11	00000	20000
10M12	TU	A0000	01M06		A	160	15	20000	00236
10M13	TV	A0000	03M48		U	161	16	20000	00456
10M14	AT	00015	A0000		X	162	35	00017	20000
10M15	SS	00016	00015		I	163	34	00020	00017
10M16	TU	A0000	03M48		L	164	15	20000	00456
10M17	LA	A0000	00042			165	54	20000	00052
10M18	TV	A0000	01M06			166	16	20000	00236
10M19	TV	00000	02M50		SET F	167	16	00000	00360
10M20	TN	00016	C0N10			170	13	00020	00507
10M21	TP	00000	A0000			171	11	00000	20000
10M22	TV	A0000	C0N06		SET N	172	16	20000	00503
10M23	TP	A0000	C0N06		SET M	173	15	20000	00505
10M24	AT	02M10	A0000		Y	174	35	00310	20000
10M25	TV	A0000	01M13			175	16	20000	00245
10M26	TV	A0000	01M14			176	16	20000	00246
10M27	TV	A0000	03M02			177	16	20000	00400
10M28	54	C0N08	20071	BRB		200	54	00505	20071
10M29	AT	C0N06	A0000		SET M-N	201	35	00503	20000
10M30	TV	A0000	C0N07		T	202	16	20000	00504
10M31	AT	02M10	A0000			203	35	00310	20000
10M32	TV	A0000	C0N01			204	16	20000	00476
10M33	TV	A0000	02M01			205	16	20000	00277
10M34	LA	A0000	00015			206	54	20000	00017
10M35	TU	A0000	02M06			207	15	20000	00304
10M36	TU	A0000	02M50			210	15	20000	00360
10M37	TU	A0000	02M11			211	15	20000	00311
10M38	TU	A0000	02M17			212	15	20000	00317
10M39	TP	C0N01	A0000			213	11	00476	20000
10M40	AT	C0N06	A0000		Z	214	35	00503	20000
10M41	TV	A0000	03M16			215	16	20000	00416
10M42	TP	00021	00000			216	11	00025	10000
10M43	QS	C0N08	01M12			217	53	00505	00244
10M44	TP	C0N10	A0000		SET	220	11	00507	20000
10M45	AT	C0N06	C0N09		NO1	221	35	00503	00506
10M46	54	C0N07	20017	BRB		222	54	00504	20017
10M47	TU	A0000	01M25		M-N	223	15	20000	00261
10M48	RA	00M01	C0N05		P-4 EXIT	224	21	00145	00502
10M49	TV	03M02	03M47			225	16	00400	00455
10M50	TU	02M62	01M17			226	15	00374	00251
10M51	TU	02M62	02M07			227	15	00374	00305
11M00	TV	02M50	01M01		SET 0 FOR	230	16	00360	00231
11M01	TP	00013	00000		INTERCHANGE	231	11	00015	00000
11M02	TV	00000	02M50		RESET	232	16	00000	00360
11M03	STU	A0000	02M01	BRB	TO	233	64	00000	00177
11M04	STU	A0000	02M01		F ADDRESS	234	15	20000	00277
11M05	TP	C0N02	C0N11		SET SF INDEX	235	11	00477	00510
11M06	RJ	00000	00		TO AUX 1	236	37	00000	00000
11M07	TP	C0N07	00023		SET INDEX	237	11	00504	00027
11M08	TP	00021	00000			240	11	00025	10000
11M09	TP	00000	A0000		TEST FOR	241	11	00000	20000
11M10	SJ	01M11	01M16		INVERSION	242	46	00243	00250
11M11	RS	C0N11	00016			243	23	00510	00020
11M12	75	10000	01M14		AUGMENT	244	75	10000	00246
11M13	TP	00013	00000	BRB	ROW OF	245	11	00015	00000
11M14	TP	C0N04	00000		UNIT	246	11	00501	00000
11M15	RA	01M14	00016		MATRIX	247	21	00246	00020
11M16	TP	C0N04	A0000		CHECK IF ALL	250	11	00501	20000
11M17	TM	00000	00024		ELEMENTS IN	251	12	00000	00030

TABLE 28. UNIVAC 1103 PROGRAM FOR LINEAR MATRIX EQUATION SOLVER  
(Continued)

11M18	TJ	00024	00M00		ROW ARE	252	42	00030	00144
11M19	RA	01M17	00015		SCALED	253	21	00251	00017
11M20	IJ	00023	01M16		CORRECTLY	254	41	00027	00250
11M21	RA	C0N10	00016		ADVANCE AND	255	21	00507	00020
11M22	TP	C0M10	00026		SET INDEX	256	11	00507	00032
11M23	QS	01M25	02M00			257	53	00261	00276
11M24	QS	01M25	02M45			260	50	00261	00353
11M25	MJ	00000	01M28			261	45	00000	00264
11M26	RS	02M00	00015			262	23	00276	00017
11M27	RS	02M45	00015			263	23	00353	00017
11M28	TU	02M00	02M49		RESET	264	15	00276	00357
11M29	TV	C0N01	02M30		T	265	16	00476	00334
11M30	TU	02M06	02M22		ADVANCE	266	15	00304	00324
11M31	RA	02M07	00015		X	267	21	00305	00017
11M32	TU	02M07	02M12			270	15	00305	00312
11M33	TU	02M07	02M16			271	15	00305	00316
11M34	TU	02M07	02M21			272	15	00305	00323
11M35	TU	02M07	02M46			273	15	00305	00354
11M36	SS	02M07	10025	BRB		274	55	00305	10025
11M37	TV	00000	02M26			275	16	10000	00330
12M00	TS	30000	02M02	BBR	TRANSMIT ITH	276	76	30000	00300
12M01	TP	00000	0		ROW TO ES	277	11	00000	00000
12M02	TV	03M26	02M21		SET FOR	300	16	00430	00323
12M03	TV	03M26	02M22		INVERSION	301	16	00430	00324
12M04	TP	00013	00024			302	11	00015	00030
12M05	TP	00013	00028			303	11	00015	00034
12M06	TM	00000	00029		COMPARE LEAD	304	12	00000	00035
12M07	TM	00000	A0000		ELEMENTS	305	12	00000	20000
12M08	TJ	00029	02M16			306	42	00035	00316
12M09	ZJ	02M11	02M13			307	47	00311	00313
12M10	00	00000	C0N12		CONSTANT	310	00	00000	00511
12M11	MP	00000	C0N03		R INTER	311	71	00000	00500
12M12	DV	00000	00024		O	312	73	00000	00030
12M13	RA	02M21	00016		W CHANGE	313	21	00323	00020
12M14	TN	C0N04	00027			314	13	00501	00033
12M15	MJ	00000	02M20			315	45	00000	00322
12M16	S4	00000	20043	B	NO ROW	316	54	00000	20043
12M17	DV	00000	00024		INTER	317	73	00000	00030
12M18	RA	02M22	00016		CHANGE	320	21	00324	00020
12M19	TP	00013	00027			321	11	00015	00033
12M20	TN	00024	00024		OK	322	13	00030	00030
12M21	TP	00000			L	323	11	00000	00000
12M22	TP	00000			I	324	11	00000	00000
12M23	MA	00030	20043	BRB	N	325	54	00036	20043
12M24	MA	00031	00024		E	326	72	00037	00030
12M25	NJ	00000	03M58		A	327	45	00000	00470
12M26	TP	80000	00000		R	330	11	30000	00000
12M27	TM	80000	A0000		L	331	12	30000	20000
12M28	TJ	00028	02M30		Y	332	42	00034	00334
12M29	TP	A0000	00028		COMBINE	333	11	20000	00034
12M30	TP	00031	00000			334	11	00037	00000
12M31	RA	02M21	00015		R	335	21	00323	00017
12M32	RA	02M22	00015		O	336	21	00324	00017
12M33	RA	02M30	00016		W	337	21	00334	00020
12M34	RA	02M26	00016		S	340	21	00330	00020
12M35	ST	C0N01	A0000			341	36	00476	20000
12M36	ZJ	02M21	02M37			342	47	00323	00343
12M37	EJ	00028	02M47		R	343	43	00034	00355
12M38	TV	00013	02M46		E	344	16	00015	00354
12M39	SS	00028	10001	BRB	S	345	55	00034	10001
12M40	QJ	02M44	02M41		C	346	44	00352	00347
12M41	QJ	02M47	02M42		A	347	44	00355	00350
12M42	RA	02M46	00016		L	350	21	00354	00020
12M43	QJ	02M45	02M42		E	351	44	00353	00350
12M44	TV	03M56	02M46			352	16	00466	00354
12M45	TS	20000	02M47	BBR	R	353	75	20000	00355
12M46	LA	00000			W	354	54	00000	00000
12M47	TP	00027	A0000		REPLACE ROW	355	11	00033	00000
12M48	SJ	02M49	02M51		ON DRUM IF	356	46	00357	00361
12M49	S5	30000	02M51	BBR	INTERCHANGE	357	75	30000	00361
12M50	TP	00000			TOOK PLACE	360	11	00000	00000
12M51	TP	00021	00000			361	11	00025	10000
12M52	TP	02M01	A0000			362	11	00277	20000
12M53	QA	02M00	02M01			363	52	00276	00277
12M54	LA	A0000	00057			364	54	20000	00071
12M55	TV	A0000	02M50			365	16	20000	00360
12M56	IJ	00026	01M26			366	41	00032	00262
12M57	IJ	C0N09	00M50		I TIMES	367	41	00506	00226
12M58	RS	C0N07	C0N06		N-1 TIMES	370	23	00504	00503
12M59	RS	C0N07	00016		SET MOI	371	23	00504	00020
					FOR INDEX				

TABLE 28. UNIVAC 1103 PROGRAM FOR LINEAR MATRIX EQUATION SOLVER  
(Continued)

12M60	54	C0N06	20017	BRB	372	54	00503	20017
12M61	QS	A0000	03M55		373	53	20000	00465
12M62	TP	C0N11	00029		374	11	00510	00035
12M63	TV	03M16	03M14		375	16	00416	00414
13M00	TP	C0N10	00026	SET INDEX	376	11	00507	00032
13M01	TU	02M01	03M14		377	15	00277	00414
13M02	TP	C0N04	00000	SET SCALE	400	11	00501	00000
13M03	TP	00021	00000		401	11	00025	10000
13M04	QS	00013	03M13		402	53	00015	00413
13M05	TP	00013	00028	COUNTERS TO	403	11	00015	00034
13M06	TP	00013	00027	ZERO	404	11	00015	00033
13M07	RA	00028	00015	ADVANCE	405	21	00034	00017
13M08	RA	03M13	00015	COUNT	406	21	00413	00017
13M09	RS	03M14	C0N08		407	23	00414	00505
13M10	TU	03M14	03M16	TRANSFER	410	15	00414	00416
13M11	RS	03M14	00016	ROWS	411	23	00414	00020
13M12	RS	03M14	00028	OF UPPER	412	23	00414	00034
13M13	RP	30000	03M15	TRIANGULAR	413	75	30000	00415
13M14	TP	00000	0	MATRIX TO	414	11	00000	00000
13M15	RA	03M16	C0N09	ES	415	21	00416	00506
13M16	TN	00000	0		416	13	00000	00000
13M17	54	03M16	20017	BRB	417	54	00416	20017
13M18	TU	A0000	03M25		420	15	20000	00427
13M19	TV	03M02	03M25		421	16	00400	00427
13M20	TP	00027	00032	SET INDEX	422	11	00033	00040
13M21	TP	00013	A0000		423	11	00015	20000
13M22	MJ	00000	03M25		424	45	00000	00427
13M23	54	00030	20043	BRB	425	54	00336	20043
13M24	CC	00031	00013		426	27	00037	00015
13M25	MA	00000	0	B	427	72	00000	00000
13M26	TP	80000	00030	A	430	11	30000	00036
13M27	TP	A0000	00031	C	431	11	20000	00037
13M28	TM	00030	A0000	K	432	12	00036	20000
13M29	TJ	C0N04	03M31	S	433	42	00501	00435
13M30	MJ	00000	03M54	B	434	45	00000	00464
13M31	RS	03M25	00017	S	435	23	00427	00021
13M32	IJ	00032	03M23	T	436	41	00040	00425
13M33	TU	03M25	03M36	I	437	15	00427	00442
13M34	TV	03M25	03M44	T	440	16	00427	00452
13M35	TU	03M25	03M44	U	441	15	00427	00452
13M36	TM	00000	00024	T	442	12	00000	00030
13M37	TN	00030	00030	E	443	13	00036	00036
13M38	TN	00031	00031		444	13	00037	00037
13M39	TM	00030	A0000		445	12	00036	20000
13M40	TJ	00024	03M42		446	42	00030	00450
13M41	MJ	00000	03M54	RESCALE	447	45	00000	00464
13M42	54	00030	20043	BRB	450	54	00036	20043
13M43	CC	00031	00013		451	27	00037	00015
13M44	DV	00000	0		452	73	00000	00000
13M45	RA	00027	00016	ADVANCE	453	21	00033	00020
13M46	IJ	00025	03M07	NOI TIMES	454	41	00032	00405
13M47	TP	00029	00000		455	11	00035	00000
13M48	RJ	00000	0	TO AUX 2	456	37	00000	00000
13M49	RA	C0N09	00015		457	21	00506	00017
13M50	IJ	C0N07	02M62	M-1 TIMES	460	41	00504	00374
13M51	MJ	00000	00M01	EXIT	461	45	00000	00145
13M52	11	C0N00	75756	SET ALARM	462	11	00475	75756
13M53	MJ	00000	00M00	WORD	463	45	00000	00144
13M54	RS	00029	00016	RESCALE	464	23	00035	00020
13M55	RP	20000	03M57		465	75	20000	00467
13M56	LA	C0N13	00071		466	54	00512	00107
13M57	SJ	03M52	03M17		467	46	00462	00417
13M58	TP	C0N04	00032		470	11	00501	00040
13M59	SJ	03M60	03M61		471	46	00472	00473
13M60	TN	C0N04	00032		472	13	00501	00040
13M61	AT	00032	A0000		473	35	00040	20000
13M62	MJ	00000	02M26		474	45	00000	00330
C1N00	24	14061	33411	B	475	24	14061	33411
C1N01	TP	80000	000	SINGUL	476	11	30000	00000
C1N02	00	00000	00 42	C	477	00	00000	00042
C1N03	37	77777	77777	O	500	37	77777	77777
C1N04	20	00000	000 0	N	501	20	00000	00000
C1N05	00	00000	000 3	S	502	00	00000	00003
C1N06	00	0		TEMP	503	00	00000	00000
C1N07	00	0		A	504	00	00000	00000
C1N08	00	0		STORAGE	505	00	00000	00000
C1N09	00	0		N	506	00	00000	00000
C1N10	00	0		T	507	00	00000	00000
C1N11	00	0		S	510	00	00000	00000

### Methods of Call-In

*Momentary Call-In.* In magnetic drum storage, a selection of subroutines could be stored permanently on the drum ready for use when desired. With most magnetic drum machines (the ERA 1103 and 1103A being the exceptions) the drum and high-speed storage are separate logical entities; in general instructions can be formed only in the high-speed storage. The Manchester computer group (Ref. 15) developed a method of use of these two storages which shall be defined here as *momentary call-in*. In this scheme subroutines are called into the primary storage from the secondary storage only whenever their actual operation is required. Thus, generally no more than one subroutine is ever stored in the high-speed storage at once, and enough space must be saved for subroutines to hold the largest one to be performed. On some machines this method provides a satisfactory match between drum to high-speed storage transfer speed and the operating speed of the computer's arithmetic element.

*Permanent Call-In.* On other computers with a greater ratio between arithmetic operation and storage transfer speeds, a method of handling subroutines called *permanent call-in* seems desirable. In the permanent call-in scheme, as many subroutines as possible that are required in a program are stored in the high-speed storage at the beginning of the program, and performed when desired without any further need for call-in. The advantage of this method is: instead of a subroutine being called in each time it is to be performed, it is called in only once, thereby saving time. Here, as in many other computer applications, space is traded for time. The permanent call-in system requires storage space for all subroutines used, not just the longest one.

*Combined Methods.* In some cases where there is not enough storage for all such subroutines, a compromise must be made in using both methods. The decision as to just what subroutines shall be called in only once and which ones for each performance is usually a matter of guess, since a priori a programmer cannot say just how many times a particular subroutine will be used during the course of a program, and therefore cannot solve the empirical time-storage relationship required for a best solution.

**Arithmetic Classification of Subroutines.** The Whirlwind subroutine specifications manual originally listed seventeen different categories included in the Whirlwind Library of Subroutines (Ref. 57). The EDSAC library contains a smaller number of classifications (see Ref. 108). The tendency, with the advent of synthetic instructions, has been not to list subroutines by classifications but rather by the operation itself. The following types of subroutines appear to have been most useful in scientific and engineering calculations.

1. Elementary Functions: (a) sine, (b) cosine, (c) sine and cosine, (d) tangent, (e) arc sine, (f) arc cosine, (g) arc tangent, (h) exponential, (i) logarithm, and (j) hyperbolic functions.
2. Roots: (a) square roots, (b) cube roots, (c)  $N$ th roots,  $N$  arbitrary, and (d) solution of  $N$ th degree algebraic equations.
3. Integrators for ordinary differential equations.
4. Functional summation methods (Gauss's, etc.).
5. Matrix manipulations (including some simultaneous equation solutions).
6. Interpolation routines.

**Red Tape Subroutines.** In addition to the calculational subroutines listed here, there are numerous other red tape data handling routines which are usually stored and filed by various computer organizations along with such computational subroutines. The trend is to classify these red tape routines with the utility programs (see Utility Programs in this section); in many cases a complete integration of noncalculational subroutines into master automatic organizational schemes is made.

### Numerical Methods (See Vol. 1, Chap. 14.)

Most of the subroutines listed above have as their purpose the determination of some sort of approximation in a finite number of steps to an algebraic or transcendental function, with a resulting error that is to be kept small. The machine operations to be used are not the actual operations, but in general certain pseudo operation approximations to them. Whether codes for subroutines should be written to fit in the least storage space in a machine, to operate in as short a time as possible, or to minimize some complicated function of these two variables, will vary from machine to machine. No constituted theory or procedure for any particular method of approximation in terms of the standard machine operations on any computer has been established. For this reason, most of the numerical methods used with subroutines have been ones similar to those used previous to the advent of the high-speed computer.

However, methods of overall approximation, such as those making use of the Tchebysheff polynomials, have begun to replace the Taylor-McLaurin series "approximations in a neighborhood," which often have very slow ratio of convergence.

Similar deviations exist between the standard hand calculational techniques and those used with machine subroutines. The need for workable approximations will open up entire new fields in analysis and numerical methods. (See Vol. 1, Chap. 14.)

### Subroutine Generators

As Rutishauser first noted (Ref. 89), a *stretched* program eliminating all red tape operations or instructions will run much faster on a computer than does the corresponding inductive, loop-controlled program. However, it may also occupy much more storage space. A general program, prepared to accept all possible variables as program parameters, will run more slowly because of the time spent on the red tape modification of instructions to agree with the parameters. In this case, however, a program written specifically for inverting a matrix of order 10, although faster than a general  $n$ th order routine, may also occupy more space than a general program for inversion of matrices of degree  $n$ .

Judicious combination of the techniques of stretching or linearizing program loops, plus constructing programs for specific values of key parameters, may produce subroutines that minimize storage, performance time, or some combination of both. *Subroutine generators* are computer programs that do this automatically, taking in as input the parameters of the problem as it is to be performed. The concept of the subroutine generator is apparently first due to Hopper (Ref. 49), and her co-workers.

**EXAMPLE 1.** Suppose a particular problem required a sine subroutine with a precision of approximation  $\epsilon$  over a given range (for example,  $-1 < x < 1$ ). A sine subroutine generator would take this given value of  $\epsilon$ , determine the coefficients of the Tchebysheff approximation needed to provide this precision, and then punch out the most efficient program for calculating a polynomial of that degree.

**EXAMPLE 2.** Suppose that a large quantity of numbers were to be printed out with  $n$  numbers per line,  $m$  lines between spaces, each number to have  $d$  digits before the decimal point. The numbers  $m$ ,  $n$ , and  $p$  would be fed as parameters to a subroutine *print generator* which would then produce a printing subroutine tailor-made to perform printout in this format as efficiently as possible.

**EXAMPLE 3.** In analysis of propagation of error in a computational process, bounds may be determined in terms of certain maxima of derivatives of the functions involved. As shown by Kahrimanian (Ref. 150), formal differentiation may be performed on a computer. A simple recursive procedure for differentiation may be used more efficiently using techniques discussed by Kantorovich (Ref. 60). A *differentiation generator* could be used to proceed through a machine (or more likely, an algebraic language) program, constructing the corresponding derivatives of the functions involved, and combining the whole in a rational error analysis program that could be run immediately following the actual computation.

## Utility Programs

Similar to a subroutine, but not generally used in the same fashion, the *utility program* provides testing, program debugging, and general auxiliary aid to the computer programmer. Utility programs are mainly of the following types: (1) storage printouts, (2) selective (changed-word) post-mortem programs, (3) tracing programs, (4) input programs, (5) assembly programs, (6) storage transfer programs, (7) interpretive programs, and (8) compiler-translators.

**Storage printouts** may be used as standard subroutines, controlled by the programmer, or as programs controlled by the computer operator in case of unexpected program behavior. Such printouts generally provide optional formats ("octal words, decimal fixed point numbers, assembly language instructions, decimal floating point numbers," might be one sequence of options for a binary machine). Under programmer or operator control any selected portion of high-speed storage could be printed out in any one of these formats.

**Selective post-mortems** (called in only after a program is *dead*) allow only those portions (instructions and numbers) of storage that are different from the original program previously read in to be printed out. Thus a programmer need study only that portion of the program that has changed since original input. Thus the *changed-word post-mortem* compares the contents of storage *after* a problem has been performed, with input information stored on an external medium, such as punched paper tape or cards, or else with a copy of the previous input stored on a secondary storage, such as magnetic drum or magnetic tape.

**Tracing programs** are devices for following the course of a program during its performance. Their mechanism is described in detail under Interpretive Programs (see Sect. 14). Such programs usually allow detailed printouts of location, instruction, one or more operands, and result for each instruction performed. If desired, only instructions with certain selected operations, in certain selected regions of the storage, or with certain selected addresses may have this tracing information output; all else will be bypassed. Tracing of closed subroutines may also be eliminated if desired.

**Input programs** often, dependent on the machine, provide simple translation from decimal number input to internal binary, from standard numerical notation to internal coded floating point, or from external mnemonic instructions to internal machine language.

**Assembly Programs.** When such routines are combined into a program-controlled whole, it is usually called an *assembly* or *translation program* (see Sect. 11). Control of each process in turn on input translation makes

use of so-called *control combinations*, which are special instructions describing to the translator the information to be translated (see Ref. 108). More recently, many translators have determined information type from context, thus limiting the need for many control combinations. (See Sect. 9 for a description of the Share Assembly Program.)

**Storage transfer programs** are usually coded to move information back and forth from a main storage to secondary storage, and may be considered subroutines. However, they are also often used under control of an operator in a semiautomatic fashion. They usually contain automatic input and output storage-summing routines, which allow checking of all transfers, and may also contain *rollback* features by which the contents of any "volatile" main storage may be stored on a more permanent secondary storage so that a program may be restarted at the last rollback point in case of any sort of machine failure. Such a procedure is described by Brown *et al.* (Ref. 18). Some programs provide a complete magnetic tape titling and checking procedure to prevent incorrect mounting of reels.

**Interpretive programs** usually provide methods for calculating with types of arithmetic not built into machine hardware (such as floating point, complex numbers, and matrix-vector arithmetic) using machine-like instructions. Their mechanism is described in more detail in Sect. 14. Such programs generally are provided with input and output instructions, the former in machine language and the latter in the interpreter language, so that machine language and interpreter language portions of a program may be interconnected at will.

**Compiler-translators** are often not considered utility programs, but since they perform the same general function, they may well be considered so. These programs generally translate from an external (usually algebraic) nonmachine-oriented language into machine language, assign storage, call in and orient subroutines, and usually provide an assembly language program as translated output. Their mechanism is described in Sect. 11. They may also store programs automatically in secondary storage to be called in later.

### **Integrated Systems**

Interconnection of some or all these different utility programs into an organized, programmer-controlled, semiautomatic or automatic whole is usually called an *integrated system*. The first such systems, described by Adams and Laning (Ref. 1), and Brown and Carr (Ref. 17), were apparently the MIT CSSR (Comprehensive System of Service Routines) and the MAGIC (Michigan Automatic General Integrated Computation) systems. These provided either operator control through typing in English code words through an external typewriter: "translate," "com-

pute," "trace," "post-mortem," etc., or similar code words placed on the program tape or punched cards.

Such systems have spread to include almost all the classes of utility programs listed above, and have been extended more and more to provide completely automatic, nonstop running of a problem (the General Motors-North American System for the IBM 704), or complete program storage inside a machine (the Corbie system of the National Bureau of Standards on the same type of machine). Bauer (Ref. 8) has described a program allowing manual operator as well as program control for the Univac 1103. Swift (Ref. 97) has proposed certain machine hardware changes including interlocks and timers that would aid in preventing programming errors from causing major catastrophes in these new automatic systems.

Among the present or proposed automatic features of these integrated systems are the following:

1. Complete time-keeping, machine time billing, and bookkeeping for machine operation.
2. Storage of programs during check out and production.
3. Scheduling of program priorities.
4. Thorough checking of input programs or data to catch logical or transcription errors.
5. Automatic recovery procedures after all recognized programming errors (machine overflow, division by zero, improper operations, etc.).
6. Automatic recovery procedures after all recognized machine malfunctions (storage, arithmetic, or input-output failure).
7. Keeping track of previous errors, types of operations used, etc., so as to improve future performance.

Incorporation of these processes would allow continuous machine operation without programmer, operator, or engineer intervention, save at long intervals for program or hardware maintenance. Recent proposals, still in the proposal stage, have called for complete computer buffering to and from a large number of programmer input-output stations, each containing a typewriter. Programmers could each independently communicate at their leisure with the computer, which could process several programs in the checking-out stage at the same time that main production problems would also be in operation. A machine *interrupt* feature, now available in a simplified form on several commercially available computers, would allow high-speed performance of the production problems except at widely spaced intervals when programmers' requests would be answered. Such a system awaits more programming effort and built-in hardware at the present time.

The final goal of such a process is a completely automatic file system, containing all information acquired by the programming system during

operation. Development of file-searching and correlation procedures, provided with inductive *artificial intelligence* processes, might allow computer generation of programs on the basis of past processes that had proved successful.

## 11. AUTOMATIC PROGRAMMING: LANGUAGES, COMPILERS, AND TRANSLATORS

Three basic elements are involved in the solution of a problem with a computer: (1) The language of the problem, (2) preparation of the problem in terms of building blocks, and (3) the recursive use of these building blocks. The recursive use of building blocks is covered in Sect. 15. An automatic computer program for performing a translation from one machine language to another is called a *translator*. A program which can call in, connect into, and feed parameters into previously prepared sub-routines, is a *compiler*. Translation is usually part of this step.

### Language

The first experiments to use algebraic languages to describe problems of necessity depended on satisfactory symbolic and alphabetic input and were made in 1951-52. It was not until 1956 that development of a sequence of languages began, including Fortran (Formula Translator) for the IBM 704; IT (Internal Translator) for the Datatron 205, IBM 650, Univac Scientific 1103A, IBM 701, and Univac I; Math-Matic for the Univac I and II; Unicode for the Univac Scientific 1103A; as well as others for the IBM 709, Ferranti Mercury (United Kingdom), PERM (Western Germany), and Strela (Soviet Union).

An Ad Hoc Committee on Common Algebraic Languages was set up in 1957 by the Association for Computing Machinery to propose standards for such languages. The activity of this committee was extended to joint meetings with the GAMM (German-Swiss Applied Mathematics Society) in 1958. The purpose of these activities has been to standardize definitions and symbology so as to set up a *class* of languages that will be easily translatable by machine from one to another, and also easily recognizable to the ordinary human user.

**Translators.** Such languages form the input to a class of automatic computer programs called *translators*, which perform a translation from the algebraic language over into a second or *target* language. The latter may be either (1) an assembly language, such as SOAP, SAP, or MAGIC (see Sects. 7 and 9), or (2) a straight machine language, in pure decimal, binary (or in some cases such as the Univac I and II), alphanumeric.

**One-Pass and Two-Pass Systems.** Translation directly from an algebraic language into machine language is generally called a *one-pass*

system. Translation from an algebraic language into an assembly language and then into machine language is generally called a *two-pass* system.

The advantages of the two-pass system are: (a) it allows the programmer to see the results of the first translation stage in the assembly language, which he may add to, delete from, or change easily in any fashion he desires. This combination mates the abilities of automatic programming with hand-tailored programming in the most flexible fashion possible, and (b) it allows the programmer to use the usual techniques of program checkout such as tracing programs and post-mortems that have been developed for use with programs in the assembly language.

The advantage of the one-pass system is that it eliminates the need for the user to know anything about a machine-oriented assembly language. If he can obtain a correctly tested program without the necessity for using the classical checkout tools of the hand programmer, he will have sped up the programming process considerably. This, however, is not always possible.

The Fortran program for the IBM 704 is a one-pass translator, in that while it provides as output of its translation phase a listing in IBM 704 assembly language (SAP), the latter may not be used or altered by the programmer and is in many cases incomprehensible.

**The IT Translator.** The IT (Internal Translator) program for the various machines on which it has been developed is generally available in both one-pass and two-pass versions. The Internal Translator, as first written by Perlis and Smith (Ref. 81), is described in Sect. 12.

### Preparation of Problems in Terms of Building Blocks

**Extensions to the Language.** The second important element in the solution of any problem on a computer is the use of primary building blocks (subroutines), pieced together to form a program. In the IT language, such building blocks are called extensions, because they extend the range of the language beyond the simple algebraic operations originally defined. In the IT language, extensions are included by two techniques:

1. *Extension Operands.* These are basic subroutines which have one or more input parameters but only one numerical output variable. They may be inserted bodily in a statement like any other type of operand. Such extension operands are typified by the standard square root and trigonometric functions.

2. *Extension Statements.* These are basic algorithms or subroutines that perform much more complex transformations on larger masses of data, for which there may be many input parameters and many output results. Typical of this type of statement is an extension statement that performs an ordering of a string of numbers in storage. It might be described in

terms of the *name* of the first variable and the *number* of variables to be ordered. Certainly such an algorithm is not a simple function-operand such as the first type.

The addition of the ability to handle such building blocks of already prepared programs makes the IT language translator into a true compiler. The IT language, as described in Sect. 12, does all this in terms of a notation again very similar to (but not precisely) that of ordinary mathematical notation. An example of an extension of the IT language is given in Sect. 12.

Some computing machines, instead of using a compiler to tie into previously coded subroutines, actually have the subroutines wired into the computer, and have specific entry instructions set up which may be used on a machine language level to call in the subroutines automatically and perform them, just as with any other instruction. Kitov (Ref. 61) describes such a system with three-address instructions on the BESM and Strela computers of the Academy of Sciences, of the Soviet Union. The following subroutines are listed as being performed automatically as the result of one written three-address instruction: square root, sine, cosine, natural logarithm, exponential, conversion of  $n$  numbers from binary to decimal, conversion of  $n$  numbers from decimal to binary, and fixed point memory summation of  $n$  machine words.

The method of the Academy of Sciences of the Soviet Union is powerful as long as such relatively simple functions are used, but use of a translation technique such as that of the IT language to provide automatic subroutine interconnection allows much more complex subroutines to be handled easily. The next step for the Academy of Sciences machines is to combine a translation procedure along with the built-in subroutine entry technique. Once hand coding is eliminated, the need for one-instruction subroutine entries is no longer so important.

**A Flexible Filing System.** One of the weaknesses of such translator-compilers as IT and Fortran is the inability of the programmer to develop his own building blocks *easily*. As has been seen, it is entirely possible to write extensions or subroutines in machine or assembly language—the Runge-Kutta extension described in Sect. 12 is a case in point—but there is no way in the present versions of these languages to take portions of the algebraic language programs, define them as an extension operand or extension statement, and then be easily able to call on these newly defined pieces of coding by name, with automatic interconnection of entry, exit, and parameters.

What is needed in the solution of scientific problems is an application of the data processing techniques of the automatic file system to the writing of programs. Several programs are at present being used, although none has made use of the simplified algebraic language structure that has been used for the translators such as that for the IT system.

A programming system developed by the National Bureau of Standards, the Corbie system (Ref. 153), is just such a file system, basically built to handle entire programs, not subroutines. In order to save input-output card reading time on an IBM 704 lacking peripheral tape-to-card and card-to-tape converters, Wegstein and others have developed a file system by which entire programs may be stored on magnetic tape inside the machine. Corrections may be inserted from outside via a small number of cards, changes made, and the programs tested, all without the necessity of giant input-output activity. This extends the policy of control by exception, basic to the heart of all automatic machine activity, to the process of program testing.

**Generalized Programming.** Manipulation of entire programs is not enough, however, for the building block construction necessary for most efficient automatic and semiautomatic construction of complicated programming algorithms. Again, as with the Corbie system without a connection to an algebraic language, GP or *Generalized Programming* constructed by Holt and Turanski (Ref. 154) for the Univac I and II and later as GPX (Generalized Programming Extended) for the Univac-LARC, provides the building block procedure necessary for handling subroutines. With this programming system, any sequence of coding, regardless of its size, may be named, its entries and exits and input and output parameters added to lists, and the name and coding itself added to a magnetic tape file semiautomatically. Thus, many portions of the process performed by the subroutine writer for the Runge-Kutta-Gill IT subroutine in adding it to the library of subroutines for that system can be performed by the machine using the GP program. For GP, programming using a subroutine hierarchy system, with its many advantages of pretesting individual blocks before combining them, is apparently most natural for the user. The use of subroutines within subroutines is simple. The chief attributes of such a system may be listed as follows:

1. Natural coding in small blocks.
2. Possibility of automatic filing of any portion of a problem.
3. Possibility of automatic call-in and use of any portion of a previously developed portion of the same or different problem.
4. Ease of program testing because of the building block principle.
5. Possible conjunction of work of different persons easily.

The major difficulty of the GP systems is their requirement of knowledge of a nonmnemonic, nonrational code to describe to the compiler the filing system activities listed above. GP is in the full sense of the word a "programmer's" compiler, whereas the language of IT (and the other languages analogous to it) is tailor-made for the novice.

**Future Trends.** The merger of the two philosophies of algebraic language and simple command structure on the one hand, and generalized

filing systems on the other, seems to be the obvious trend. Already specifications for changes in such languages as IT and Fortran to allow some or all of the features of GP have been announced. A similar merger of GP with Univac algebraic languages may be underway. The next stage of the compiler-translator development is: (1) a machine-independent language of the IT algebraic type, allowing (2) definition of program sequences in the algebraic language as functions or more complex algorithms, combined with (3) a filing system for storing such building block pieces for automatic call-in by algebraic type function statements.

This is not the limit of present day compiler-translator goals. The initial successes in the use of the IT language for more than one computer, followed by the standardization efforts of the ACM Ad Hoc Committee on a common algebraic language, indicate that the multimachine language is the target of today's automatic programming efforts. Such a language has been proposed for many different computers and data processors now in use for the United States Army. Such a language would allow definition of existing assembly or compiler languages as sublanguages of the overall language and would provide machine aided techniques for actually constructing the translating and compiling programs themselves.

Perlis and Smith (Ref. 156) describe a "string language" for writing compiler programs and natural language as well as algebraic translators. Included as the core of an overall compiler system, this device would allow easy description of symbol-manipulation algorithms necessary for construction of new definitions in any overall language. Other papers by Graham (Ref. 156) and Bemert (Ref. 11) described language translations between IT and Fortran and Fortran and IT.

**The Theory of Algorithms.** The most thorough description of this symbol manipulation structure is given in a monograph by the Russian mathematician Markov (Ref. 67) now being translated into English. In this, Markov, from the point of view of a theoretical logician, describes a notation and develops a series of processes for complete manipulation of symbols. With some changes in symbology this language could be used directly as an input to compilers which are to perform symbol manipulation (such as compiler-producing compilers). The extension of such language application to such other indirect reasoning processes as theorem proving, natural language translation, and complex decision making is discussed in papers by Razumovski (Ref. 151), Ianov (Ref. 52), Kantorovich (Ref. 60), and other Soviet mathematicians. The extent of actual Soviet computer programs now in operation is uncertain. In the United States, the most advanced work so far described has been done by Newell, Simon, and Shaw (Ref. 76), to be described in Sect. 15.

**Automatic Instruction Modification.** The popularization of the B line or index register on the Ferranti Mark I and later the Datatron 205

and IBM 704 brought an increasing use of subroutines, even without the advantages of algebraic languages. But these automatic indexing devices did not solve the overall subroutine usage problem. The presence of only one index register on the first two named machines required a thorough complement of instructions for loading, unloading, and incrementing this special register, as can be seen by examination of the Datatron 205 code list (see Sect. 6). Even the three index registers of the IBM 704 prove insufficient in many problems, and much of the complexity of the Fortran compiler-translator program is caused by necessity for the planning by the compiler of the sequence of use of this limited number of index registers. More advanced computers such as the Univac-Larc, the IBM-STRETCH, and the Univac M-460 have 5 to 15 index registers.

It remained for the "one-and-a-half address" instruction code, described in Kitov (Ref. 61), but detailed in full by Schecher (Ref. 91) in his design for the Munich Technische Hochschule PERM computer, to carry the idea of index registers to its logical conclusion. He proposed that every storage location inside a computer be made available as an index register, and that the ordinary computer instruction word be provided with a second "half" address along with various modification digits to allow various types of modification procedures to be described below. Most important to the Schecher design was an indirect addressing "digit" in each word to provide for recursive indexing if required.

**Types of Instruction Modification.** In the use of subroutines and programming in general, four types of changes of instruction information are generally required:

1. Subroutine instruction orientation, so that a computer subroutine may be written with nonfixed addresses, yet located on call-in in an arbitrary location in the storage. (Subroutine orientation.)

2. At the time of subroutine execution, closed subroutines, upon transfer to them from the main computer program or a higher subroutine in a subroutine hierarchy assemblage, must have their exit instructions changed to give the proper return jump. (Return jump modification.)

3. During the performance of loops, addresses must be changed as new data are to be used. (Inductive modification.)

4. At the time of subroutine execution, the main program or higher subroutines must feed either (a) the parameters required or (b) the addresses of required parameters to subroutines lower in the hierarchy. (Indirect addressing.)

The automation of types 2 and 3 is handled most rationally by index registers. For a return jump, the location of the present address of the subroutine entry instruction may be stored in an index register, and this later used to modify the return jump exit instruction in the subroutine. For subroutines within subroutines, one index register is not enough, or else

instructions must be made available to exchange the contents of the index registers with other locations, etc. For a complex subroutine hierarchy, if this switching process is to be avoided, the number of index registers required is equal to the level of penetration into the hierarchy.

For induction changes in which an operation such as

$$Z_i \leftarrow X_i + Y_i \quad (i = 1, \dots, 100)$$

is to be performed for such a range of  $i$ , the index register is again a rational answer. However, if "loops within loops" are programmed (as occurs, for example, in the simplest case in matrix multiplication) then more than one index register is required, or else complete and efficient facilities are needed for exchanging the contents of the sole such device. Again, if there is to be a sequence of inductions in depth, as can occur in many recursive processes, the number of index registers must correspond to the furthest penetration of loops in the hierarchy.

Type 1 changes, subroutine orientation, can be handled by several different inefficient processes, each of which requires tagging all subroutine instruction addresses to be modified, or else all those not to be modified. A B box can be used, as on the Datatron 205 and PERM, to store the initial subroutine address, or else some other idle register in the arithmetic unit. The Datatron and PERM facilities were designed to orient automatically subroutines stored on an external medium, in this case paper tape.

**Present Address Relative.** Until 1958 when the Gamma 60 machine was announced, only two computers, the MIDAC and the FLAC (a sister computer to the MIDAC), provided automatic subroutine orientation from secondary storage (in this case a magnetic drum) (Ref. 25). In these computers subroutines are written with all addresses *relative to the present instruction location*, as opposed to the usual technique of coding subroutine addresses relative to the first location in the subroutine. The MIDAC and FLAC instruction systems allow direct performance of these "present address relative" instructions. This technique is equivalent to the index register technique (but requires one less counter in the computer, since only an instruction counter is required). A difficulty in the MIDAC-FLAC design is that only one register can modify an instruction; instructions modified by the present address cannot be modified by the index register, thus making loop operation in subroutines difficult.

Present day techniques of subroutine orientation, therefore, may use a B box orientation process, provided enough index registers are available. Since most subroutines require the use of one or more loops for induction purposes, the most widely used technique apparently at present is to forego the use of any automatic indexing and to orient subroutines with standard instruction modification techniques using the arithmetic unit.

**Indirect Addressing.** The indirect addressing technique, with its proposed recursive indexing and index addition, eliminates this requirement and others. Portions of the PERM scheme have been incorporated in the IBM 709, and the National Advisory Committee on Aeronautics (NACA) in Cleveland has modified its Univac 1103 for indirect addressing. Therefore, a complete discussion of instruction modification as planned with the PERM should be very useful in describing future trends.

Schecher proposed that digits be included in the instruction address to signify changes of types 2, 3, and 4. (With a decimal machine language one digit would provide 10 possible combinations.)

In Fig. 25 at the top is the main program. After location 516, the subroutine entry location, appear four hatched locations, which are the location of information about the program parameters for the subroutine. These can contain either (a) the parameters of the subroutine, or (b) their addresses, or in unusual cases (c) the addresses of their addresses. At the bottom of the figure is shown a subroutine with the following typical add instruction in location 834:

add 002' +.

This contains two signals, the prime, and the plus. At the time of subroutine entry, the address 516 is stored in an auxiliary location in the control unit. At the time of performance of the instruction in location 834, the prime in the instruction will cause this value to be added in automatically to the data address of that location giving the instruction

add 518 +.

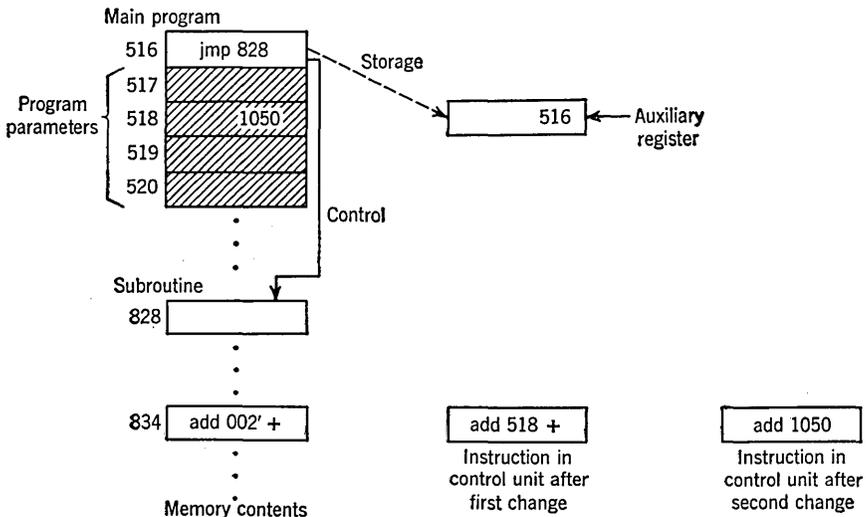


FIG. 25. Simple indirect addressing without "half-addresses."

The plus sign performs a second purpose. If it is present, the control unit will not take the contents of location 518 to the arithmetic unit for addition, but will instead substitute the address portion of location 518 into the address portion of the instruction. In the case of the figure, this would yield

add 1050,

and the instruction now would be performed in the usual fashion to supply the contents of location 1050 to the subroutine.

If, however, the contents of location 518 should contain as its address portion

1050+

then the instruction would again be delayed, and the address portion of location 1050 used to replace the instruction address. Thus a recursive "address of an address of an address . . ." similar to the IT language subscripts on subscripts is available.

The addition of a second or half-address to each instruction now makes possible multiple index modification. The "auxiliary register" of the original subroutine entry can now be any storage location designated by that instruction's half-address. The prime signal can now represent addition of the contents of an instruction's half-address location rather than the contents of a fixed location. Now if one wanted to bring  $x_{i+j+k}$  into a

TABLE 29. INDIRECT ADDRESSING

Location	Operation	Data Address (516) <sup>a</sup>	Half-Address
000		(516) <sup>a</sup>	000
⋮			
516	jmp	828	000
517			
518		<i>i</i> '	519
519		<i>j</i> '	001
520		<i>k</i> '	001
521		1000	000
522		Continued Program	
⋮			
828			
⋮			
834	add	002'+	000
⋮			
840	ret	006'	000

<sup>a</sup> Parentheses indicate contents after performance of instruction 516 which stores the latter address in the data address portion of 000.

subroutine, where  $x_0$  is stored in location 1000, one could proceed as in Table 29. The sequence of changes of instruction 834 in the control unit would be:

add	$002' +$	000
add	518+	000
add	$i'$	519
add	$(i + j)'$	520
add	$(i + j + k)'$	521
add	$(1000 + i + j + k)$	521

which would then bring the contents of location  $(1000 + i + j + k)$  into the accumulator for action by the subroutine.

The exit entry from this subroutine would be as in location 840 which would return control to  $006 + 516 = 522$ , the next location in the main program.

The use of such indirect addressing procedure allows efficient machine control of recursive subroutine hierarchy structures. Instead of requiring actual transfer of data from one subroutine to the next below it in the hierarchy, as is described in the next section on subroutine hierarchy programming, the program now has only to pass the *names* (addresses) of variables along by using the indirect addressing scheme. The use of such built-in techniques will probably grow markedly as the need for recursive subroutine programming becomes more evident.

### Use of Subroutine Hierarchy Counters

Suppose that now two special machine counters, telling how deep into a hierarchy of subroutines the path of control has penetrated, are added to the control unit of the half-address machine. These will be called the Permanent Subroutine Hierarchy Counter (PSHC) and the Temporary Subroutine Hierarchy Counter (TSHC). Signal for their use in modifying an address, as written here, will be the asterisk (\*), for the PSHC and exclamation point (!) for the TSHC.

At the occurrence of each *jmp* instruction to a subroutine *lower* in the overall hierarchy of subroutines, the contents of both counters will be increased by one. At the occurrence of each *ret* instruction back from a lower subroutine to a higher subroutine, the contents of both counters are decreased by one. The original value of their contents at the start of a program is zero. Thus, at any stage the value of PSHC will indicate the depth into the subroutine hierarchy that has been penetrated. Even if such a structure is recursive (a lower subroutine enters one higher above it in the hierarchy), the PSHC contents will indicate the number of levels, by whatever complex route, to the top or main program level.

The TSHC, in any normal situation, will have the same value as the

PSHC. However, when a lower level subroutine is calling for an operand to be fed to it from a higher level, at each process of address substitution the TSHC will be decreased by one before the next cycle of address substitution or actual instruction operation begins. After an instruction is actually performed the computer resets TSHC to correspond to PSHC.

The contents of the PSHC correspond to the index  $j$  in the flow diagram of Fig. 26.

The computer is now able to perform automatically all the subroutine entry, exit, and parameter transfer manipulations given in Fig. 26. However, due to the automatic address modification and indirect address substitution, the programmer needs to follow only a very simple set of rules.

As an example, suppose a main program (MP) is to refer to a subroutine (SRI), which in turn is to refer to a lower subroutine (SRII), which is to refer to a third (SRIII), which is to refer recursively to the original main program (see Table 30). Such a process will "loop" infinitely often unless some sort of stopping criterion is included. The *tr neg* (transfer control on negative accumulator) instruction in location 101 can be considered to serve such a purpose, since if the number being tested is negative, the return trip up the subroutine hierarchy chain will begin.

As the path of control travels down through these subroutines, the main program is to feed a parameter  $p_0$  to SRI, SRI a parameter  $p_1$  to SRII and on to SRIII, SRII a parameter  $p_2$  to the subroutine below SRIII (which turns out to be SRI, etc.).

Suppose that the instruction counter in the machine is directly addressable at location 001. This means that the half-address of a primed instruction referring inside the same subroutine can use the instruction counter location 001 as its half-address and thus all such instructions can be completely independent of all absolute machine addresses. (This is the "present address relative" addressing system mentioned earlier.) Location 000 will contain zero.



TABLE 30. PROGRAM USING SUBROUTINE HIERARCHY COUNTERS

Symbol and Absolute Location	Regional Location	Operation	Data Address	Half-Address	Explanation
MP = 50	0	read	002'	001	
	1	jmp	SRI	RET*	$p_0 \rightarrow 002 + C(IC) = 052$
	2	storage for Parameter $p_0$			$C(PSHC) + 1 \rightarrow PSHC$ and TSHC $100 \rightarrow IC$
SRI = 100	3	stop			$51 \rightarrow 1000 + C(PSHC)$
	0	read	004'	001	$p_1 \rightarrow 004 + C(IC) = 104$
	1	cla	001'	RET!	$C(001 + C(1000 + C(TSHC))) \rightarrow ACC$
	2	tr neg	003'	001	tr neg to 003 + $C(IC) = 105$
	3	jmp	SRII	RET*	$C(PSHC) + 1 \rightarrow PSHC$ and TSHC $200 \rightarrow IC, 103 \rightarrow 1000$ $+ C(PSHC)$
SRII = 200	4	storage for Parameter $p_1$			
	5	ret	002'	RET*	$002 + C(1000 + C(PSHC)) = 002$ $+ 051 = 053 \rightarrow IC$
	0	read	005'	001	$p_2 \rightarrow 005 + C(IC) = 205$
	1	cla	001'	RET*	$C(001 + C(1000 + C(PSHC))) \rightarrow ACC$
	2	sto			
SRIII = 300	3	jmp	SRIII	RET*	$C(PSHC) + 1 \rightarrow PSHC$ and TSHC $300 \rightarrow IC, 203 \rightarrow 1000$ $+ C(PSHC)$
	4		001' +	RET!	Name of parameter $p_1$
	5	storage for Parameter $p_2$			
	6	ret	002'	RET*	$002 + C(1000 + C(PSHC)) = 002 + 103 = 105 \rightarrow IC$
	0	cla	001'	RET*	$C(001 + C(1000 + C(PSHC))) \rightarrow ACC$
	1	sto			
SRIII = 300	2	cla	002'	RET*	$C(002 + C(1000 + C(PSHC))) \rightarrow ACC$
	3	sto			
	4	jmp	SRI	RET*	$C(PSHC) + 1 \rightarrow PSHC$ and TSHC $100 \rightarrow IC, 305 \rightarrow 1000$ $+ C(PSHC)$
	5		002' +	RET!	Name of parameter $p_2$
	6	ret	003'	RET*	$003 + C(1000 + C(PSHC)) = 003 + 203 = 206 \rightarrow IC$

The storage in region RET will be as follows during the course of operation:

RET = 1000	
0	000
1	051
2	103
3	203
4	304

One can now trace through the performance of those instructions of the above program that are pertinent to the demonstration of the techniques of indirect addressing, use of the half-address, and the PSHC and TSHC. Initially these latter two counters will contain 00 (see Table 31).

TABLE 31. INSTRUCTIONS OF TABLE 30 PROGRAM INVOLVED IN INDIRECT ADDRESSING

IC Before	Instruction	IC After	PSHC		TSHC	
			Before	After	Before	After
50	read 002' 001	50	00	00	00	00
	read 052 000					
51	jmp 100 1000*	100	00	01	00	01
	jmp 100 1001					
100	read 004' 001	101	01	01	01	01
	read 104 000					
101	add 001' 1000!		01		01	
	add 001' 1001					
	add 052 000	102		01		01
102		103	01	01	01	01
103	jmp 200 1000*		01	02	01	02
	jmp 200 1002	200		02		02
200		201	02	02	02	02
201	add 001' 1000*		02		02	
	add 001' 1002					
	add 104 000	202		02		02
202		203			02	
203	jmp 300 1000*		02	03	02	03
	jmp 300 1003	300		03		03
300	add 001' 1000*		03		03	
	add 001' 1003					03
	add 204 000			03	03	03
	add 001' 1000!		03	03	03	02
	add 001' 1002		03		02	02
	add 104 000	301		03	02	02
301		302	03	03	03	03
302	add 002' 1000*					
	add 002' 1003					
	add 205 000	303		03		03
303		304		03		03
304	jmp 100 1000*		03	04	03	04
	jmp 100 1004	100	04	04	04	04

TABLE 31. INSTRUCTIONS OF TABLE 30 PROGRAM INVOLVED IN INDIRECT ADDRESSING (*Continued*)

IC Before	Instruction	IC After	PSHC		TSHC	
			Before	After	Before	After
100		101		04		04
101	add 001' 1000!			04		04
	add 001' 1004					
	add 305			04		04
	add 002' 1000!		04	04	04	03
	add 002' 1003					
	add 205			04		03
102	tr neg 003 001		04		04	
	tr neg 105 001	105		04		04
	(Suppose ACC is negative.)					
105	ret 002' 1000*		04	04	04	04
	ret 002' 1004		04	03	04	03
	ret 306 000	306		03		03
306	ret 003' 1000*		03	03	03	03
	ret 003' 1003		03	02	03	02
	ret 206 000	206		02		02
206	ret 002' 1000*		02	02	02	02
	ret 002' 1002		02	01	02	01
	ret 105 000	105		01		01
105	ret 002' 1000*		01	01	01	01
	ret 002' 1001		01	00	01	00
	ret 053 000	053		00		00
053	stop					

In this program one has transferred to any one subroutine from only one sequence above in the hierarchy, but there could also be many entries from many different routines into a lower level sequence, and the process would work in the same fashion.

Until more machines are built with this half-address feature and hierarchy storage counters, users will have to content themselves with compiler-translator processes or direct interpreters that will perform the same actions.

## 12. AUTOMATIC PROGRAMMING: THE IT TRANSLATOR, TRANSLATOR CONSTRUCTION

The IT program (Internal Translator) was first written by Perlis and Smith (Ref. 81) for the IBM 650 and was later modified and enhanced at the University of Michigan Digital Computation Unit. The IT language, with its general translation techniques, was the first adopted over a wide range of computers representing a number of manufacturers. In this sense, it was one of the first *universal programming* languages. Because of its

preeminent position, the IT program is described in detail here. It is available in both the one-pass and two-pass versions (see Sect. 11).

### The IT Translator

Two versions of this program are available. The input language in both cases is based on the symbols available on the Share standard (Fortran) IBM 026 Key Punch. The same symbols are used to print the programs on the IBM 407 printer. Standard symbols are used in every case possible, although the number of symbols available with input equipment to the IBM 650 is not enough to express algebraic manipulation satisfactorily.

The command language for IT is machine independent in that it has not been adapted to any of the vagaries of any of the target language machines with which it is used. Programs are written in the form of *statements*. Each statement is a string of letters and algebraic symbols (limited by readability, input medium flexibility, and the power of the translator itself). In the IT translator for the IBM 650, this number varies from translator version to version, but the number of characters per statement is of the order of magnitude of 100.

The output of the one-pass version of the translator is in straight machine language, ten decimal digits per word, output in general on punched cards, with information stored on the card in a standard five-word-per-card random load format that allows storage of any one of the five words in an arbitrary position inside the computer storage.

The output of the two-pass version of the translator is first to an assembly routine, SOAP (Symbolic Optimal Assembly Program) which is generally the standard assembly language for the IBM 650. The SOAP language is written in the form of one instruction per line, with mnemonic three-digit operation code and symbolic addresses for the IBM 650 one-plus-one address system. The latter uses two addresses, one to indicate the data to be operated, the other to indicate the next instruction to be used.

As opposed to other languages such as Fortran used on larger computers, where a more powerful translator is available, the IT language for the IBM 650 uses three different classes of variables, I variables (fixed point ten-digit integers) and Y and C variables (floating point (8, 2, 0) decimal numbers). Any variable may be subscripted by an integer or an integer variable (I variable) so that subscripting in depth (for example,  $i_{17}$ ) is entirely possible, as well as computed subscripts, for example,  $i_{i_1+(i_7 \times i_3)/i_2}$ . Subscripts are obtained by direct concatenation, since they are not on the key punch, for example YI1, Y7.

**Rules for IT Language.** A complete set of rules for the IT language, as adapted from the original specifications by Perlis and Smith (Ref. 81), are given here for completeness. The mechanics of key punching, input-output, card format, and computer operation are omitted.

1. **Characters of the Language.** The language as described is applicable to the IBM 650 with Fortran characters which admits the digits 0 through 9, the (Roman capital) letters A through Z, and the special characters (,), +, -, =, ., \*, /, and , (comma). Certain standard symbols are represented by alphabetical characters at particular portions of the language translation process because of printer limitations. Except when it occurs in an English word, each alphabetical character has one and only one meaning in this language.

*Punctuation Characters.*

<i>Symbol</i>	<i>Name</i>	<i>Representation</i>
(	Left parenthesis	( or L
)	Right parenthesis	) or R
.	Decimal point	. or J
←	Substitution	= or Z
=	Relational equality	U
>	Greater than	V
≧	Greater than or equal	W

The following punctuation characters will be introduced as they arise:

,	Comma	, or K
' ,	Quotation marks	Q
	Type	T
	Finish	F
	Extension identifier	E

*Variable Characters.*

I      Y      C

*Digit Characters.*

The integers 0 through 9  
B

In the sequel lower case letters, such as k, l, m, and n, will be used to represent arbitrary positive integers.

*Operator Characters.*

<i>Symbol</i>	<i>Name</i>	<i>Representation</i>
<b>Binary Operators</b>		
+	Addition	+ or S
-	Subtraction	- or M
×	Multiplication	* or X
/	Division	/ or D
exp	General exponentiation	P

Note.  $Y_1 P Y_2$  means  $Y_1^{Y_2}$

Unary Operators

$ \dots $	Absolute value	A
$(-\dots)$	Negative of	$(-\dots)$ or LM... R

**2. Admissible Variables.**

*Problem (floating point) Variables.*

$Y_n$	$YIn$	$YII_n$	For example:	$Y_3$	$YI_{47}$	$YII_{21}$
$C_n$	$CIn$	$CII_n$		$C_3$	$CI_0$	$CII_7$

These two classes of variables have the same logical significance in the language. They aid in the (external) differentiation between two classes of data or problem variables. The numerical value of any of these variables is always represented in floating point form.

*Problem (fixed point) Variables.*

$I_n$	$II_n$	$III_n$	For example:	$I_8$	$II_{36}$	$III_2$
-------	--------	---------	--------------	-------	-----------	---------

These variables take on integral values only and are used primarily as indices.

*Composite Variables.*

$Y(\dots)$	For example:	$Y(I_1 + 6)$
$I(\dots)$		$I(II_3 \times 19)$
$C(\dots)$		$C(I(I_1 + 2))$

The parenthesized quantities must be *fixed point expressions* (see 4. Admissible Operands).

*Matrix (floating point) Variables.*

$YN(\dots, \dots)$
$CN(\dots, \dots)$

These variables are general elements of matrices (listed row-wise) whose components are  $Y_0, Y_1, \dots$ ; and  $C_0, C_1, \dots$ , respectively.

The parenthesized quantities, which must be *fixed point expressions*, specify the row and column location respectively of the matrix variable.

The row dimensions of  $YN$  and  $CN$ , i.e., the number of columns in the matrices, must be specified by assigning them to  $I_1$  and  $I_2$ , respectively.

The row number and the column number always range from 0 to their respective dimensions less one.

For example, a rectangular matrix may be assigned as

$Y_0$	$Y_1$	$Y_2$
$Y_3$	$Y_4$	$Y_5$
$Y_6$	$Y_7$	$Y_8$
$Y_9$	$Y_{10}$	$Y_{11}$

Thus YN (0, 0) refers to Y0, and YN (1, 2) refers to Y4, while I1 and I2 must be assigned the value 3 and 4, respectively.

### 3. Admissible Constants.

*Fixed Point Constants (Integers).*

$n_1n_2 \cdots n_k \quad k \leq 10$  For example: 1066; 10; 1292345566

However 123. is not such a constant since it contains a decimal point.

*Floating Point Constants.*

a.  $n_1n_2 \cdots n_t . n_{t+1} \cdots n_k \quad k \leq 8$  For example: 14.92; .11; 13.

b.  $n_1n_2 \cdots n_k B_m \quad k \leq 8$  which means  $n_1n_2 \cdots n_k \times 10^m$ , where a.  $n_1n_2 \cdots n_k$  is either a fixed or a floating point constant, and b.  $m$  is either  $m_1m_2$  or  $-m_1m_2$ , where  $m_1m_2$  is a fixed point constant.

*Rule.* If  $m$  is of the form  $-m_1m_2$  then the *entire* constant must be enclosed in parentheses.

For example. 14.92B3; (1066B-11); and (-727B-5) mean  $14.92 \times 10^3$ ;  $1066 \times 10^{-11}$ ; and  $-727 \times 10^{-3}$ ; respectively.

*Note.* Floating point constants used within statements (see Admissible Statements) must be of the above form and *not* in standard floating point form. Thus 14.92 would be used as 14J92 but *not* as 1492000051.

### 4. Admissible Operands.

The following rules apply:

a. Any variable or constant is an operand.

b. If  $v_1$  and  $v_2$  are operands, then  $(v_1 \Delta v_2)$  is an operand, where  $\Delta$  is an admissible operator character.

c. Subroutines, themselves functions of one or more operands, are operands. They are represented as " $nE, v_1, v_2, \dots, v_j$ " which means the subroutine whose identification number is  $n$  ( $n$ th extension) and which is a function of the variables  $v_1, v_2, \dots, v_j$ . Here  $n$  must be a fixed point constant less than 626. For example, if the sine subroutine were the subroutine number 21, then  $\sin(Y_1 + Y_2)$  would be represented by 21 E,  $Y_1 + Y_2$ . The statement "1 E, "21 E,  $Y_1 + Y_2$ " " would represent the  $\log_{10}(\sin(Y_1 + Y_2))$  if the subroutine number 1 were the log routine.

d. If  $v_1$  and  $v_2$  are operands, then  $v_1 \Delta v_2$  is an *expression*, where  $\Delta$  is an admissible operator character. Owing to the method by which the compiler examines strings of symbols, some expressions will not be treated as operands. However, all operands are expressions. The *norm* of an expression is the number of symbols, exclusive of spaces, making up the expression. Although not necessary for correct interpretation, users are urged to parenthesize all statements fully to avoid mistakes.

e. If an operand is a variable or a constant its arithmetic is that of the variable or constant.

f. If any operand, with the exception of subroutines, is composite and at least one of its members is floating point, the entire operand is floating point.

g. The arithmetic of subroutines is determined by their extension number, according to the following:

$$n < 500 \text{ floating point} \quad n \geq 500 \text{ fixed point}$$

### 5. Admissible Statements.

Each statement is identified by a nonnegative integer  $k \leq 626$ . The execution sequence of a set of statements is not determined by this identifier, but rather by the physical ordering.

A natural correspondence exists between the types of processes found in flow charts and the kinds of statements in the language. Statements may be considered as sentences—correctly formed strings in the characters of the language. A description of the various statement forms follows.

#### *Substitution Statement.*

The statement

$$k: \quad v1 \leftarrow v2,$$

where  $v1$  is a variable,  $v2$  is an expression, and  $k$  is the statement identifier (number), has the following effect: the value of  $v1$  is set equal to that of  $v2$  in the arithmetic of  $v1$ . For example,

$$7: \quad YI2 \leftarrow I1 + (I3 \times (Y3 - Y4))$$

sets the value of  $YI2$  equal to that of  $I1 + I3 \times (Y3 - Y4)$  in statement 7.

#### *Unconditional Linkage Statement.*

Any of

$$\begin{aligned} k: & \quad G_n, \\ k: & \quad GI \cdots I_n, \\ k: & \quad G(\cdots), \end{aligned}$$

where  $k$  is the statement number, have the following effect: the next statement is defined in the execution sequence as that one having its statement number equal to the value of  $n$ , or the value of  $I \cdots I_n$ , or the value of the parenthesized fixed point expression (operand), whichever the case may be.

#### *Relational (Conditional) Linkage Statement.*

$$\begin{aligned} \text{Any of} \quad k: & \quad G_n \\ k: & \quad GI \cdots I_n \quad \text{IF} \quad v1 \gamma v2 \\ k: & \quad G(\cdots) \end{aligned}$$

where  $v_1$  is an operand,  $v_2$  an expression, and  $\gamma$  is one of the three relations  $= : > \geq$ , is interpreted as: if  $v_1$  is in the relation  $\gamma$  to  $v_2$ , the effect is that of the G portion of the statement; if not, the execution sequence is unaltered. Thus,

4: G I3 IF (Y1 + Y2)  $\geq$  9

transfers control to the statement having a number which is the value of I3 if  $Y1 + Y2 \geq 9$ ; otherwise the sequence is unaltered. Removing the parentheses would make the statement inadmissible since Y1 would not then be an operand for relational statements. In these statements when the left operator is compound, it must be delimited by parentheses.

*Halt Statement.*

k: H.

The effect is to suspend computer operation.

*Input Statement.*

k: READ.

The effect is to initiate the input of one or more data cards. The format is dependent on the particular structure of the input subroutine, which may differ from installation to installation. Reading ceases upon presentation of a terminating symbol (hole in a certain column, for example). Read statements can call in numbers, alphanumerical characters, or both.

*Output Statement (Type Statement).*

k: T v1 T v2 T v3 T v4 T v5.

The effect is to punch a card, whose format is that of an input card, containing the names and current values of  $v_1, v_2, \dots, v_5$ . Here the  $v$ 's must be variables and from one to five may be listed in the statement. If a variable is composite, e.g., YI6, its current name is punched, e.g., if, at punching, the value of I6 is 4 then the value of Y4 will be punched along with the name Y4. The statement number is always punched on the card.

Conditional output statements have the same form as output statements except that the statement number must be zero. They provide output conditional on the IBM 650 sign storage entry switch as follows. Storage entry switch set to minus (-) activates output; positive (+) setting causes output to be by-passed.

*Iteration Statement.*

k: j, v1, v2, v3, v4.

Here  $j$  is an integer which *must* be positive;  $v_1$  is a variable;  $v_2$  is an expression;  $v_3$  is an expression; and  $v_4$  is an expression which must not

contain the operators X, D, or P. The norm of each  $v$  must not exceed five. (Here *norm* is defined as the number of characters.) The effect is to construct an iteration of the set of statements interposed between  $k$  and (including)  $j$ —called the scope of the iteration statement—on the variable  $v_1$  as it varies from  $v_2$  to  $v_4$  in increments of  $v_3$ . Thus,

```

15:   19, I1, I3 + 4, I4, I5 + 1.
21:   Y5 ← CI1 + 2
19:   YI1 ← Y5 - 7
    
```

causes statements 21 and 19 to be executed sequentially for all values of  $I1$  from  $I3 + 4$  to  $I5 + 1$  in steps of  $I4$ . That statement which follows an iteration statement must name a unique nonzero statement number.

If  $v_4 - v_2$  is not divisible by  $v_3$  the iteration stops before  $v_1$  assumes a value greater than  $v_4$ . If  $v_3$  is to be taken an actual decrement it must be of the form  $-v$  where  $v$  takes on only non-negative values, and must be a constant or variable.

A hierarchy of nesting of iterations is permitted, i.e., an iteration statement may be included in the scope of a prior iteration statement. However, no particular nesting may exceed a depth of four.

In general, fixed point quantities should be used in iteration statements. If floating point quantities are used, care should be taken to guarantee the desired number of iterations.

*Extension Statement.*

```

k:   "nE, . . .".
    
```

The effect is to accomplish the subroutine before passing to the next statement in sequence. Such a statement is used wherever compound sequences of operations, not necessarily leading to the definition of a single variable, are required, for example, sorting, solving a system of differential equations, packing and unpacking data, etc.

**Flow Diagrams in Compiler Language.** The following will indicate how a particular flow chart may be represented in compiler language. The flow chart of Fig. 27 evaluates the polynomial

$$y = \sum_{i=0}^{10} a_i x^i.$$

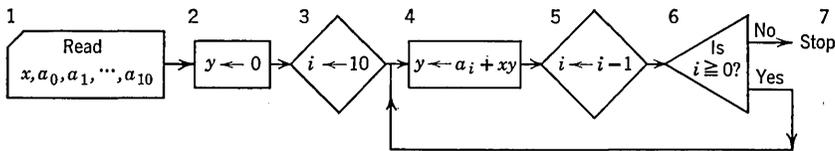


FIG. 27. Flow chart for evaluating the polynomial  $y = \sum_{i=0}^{10} a_i x^i$ .

Using the notation for variables in the compiler language, the flow chart is given in Fig. 28. The corresponding program (I) in the compiler language is given in Table 32.

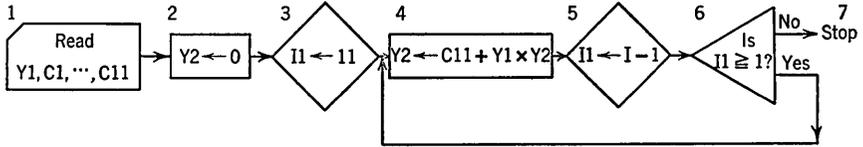


FIG. 28. Flow chart of Fig. 27 in compiler language.

TABLE 32. PROGRAM (I) IN COMPILER LANGUAGE

1:	READ
2:	$Y2 \leftarrow 0$
3:	$I1 \leftarrow 11$
4:	$Y2 \leftarrow C11 + (Y1 \times Y2)$
5:	$I1 \leftarrow I1 - 1$
6:	G4 IF $I1 \geq 1$
7:	H

Using an iteration statement and letting the degree be variable by assigning it to the variable I5, the program is given in Table 33.

TABLE 33. PROGRAM (II) IN COMPILER LANGUAGE

1:	READ
2:	$Y2 \leftarrow 0$
3:	4, I1, I5, -1, 1,
4:	$Y2 \leftarrow C11 + (Y1 \times Y2)$
7:	H

Using the required representation for all symbols, the alternate program (II) for evaluating a polynomial of any degree is given in Table 34.

TABLE 34. PROGRAM (II) FOR EVALUATING A POLYNOMIAL OF ANY DEGREE

1:	Y1 C1 through C11	READ	F	DATAREAD
2:	$Y2 = 0$		F	SET P
3:	4, I1, I5, -1, 1		F	I LOOP
4:	$Y2 \leftarrow C11 + (Y1 \times Y2)$		F	NEW P
7:	H		FF	STOP

This program, punched one statement to a card, is then translated within the computer to yield a machine program in SOAP language.

The following remarks concerning the representation of the flow chart are pertinent:

a. Each process in the flow chart leads to one and only one statement in (I). However (II) encompasses in one statement, that numbered 3, the processes numbered 3, 5, and 6 in the flow chart. These three processes are typical of certain iterations where a process  $f(\cdot \cdot \cdot)$  is to be carried out for all  $n_1 \leq i \leq n_2$ . Process 3 sets the parameter; process 5 increments it; and process 6 determines whether its upper limit has been surpassed. These responsibilities are combined in the iteration statement.

b. Since Y1 is a floating point variable it is convenient, but not necessary, to use a zero which is a floating point constant.

c. The degree of the polynomial is, properly speaking, a problem variable. Treating it as such in (II) consequently makes for a more general program.

d. The F and FF mark the statement end and program end respectively.

*The Compiled Program.* Compilation on the IBM 650 two-pass system proceeds in two phases: (1) translation (into a symbolic program), and (2) assembly (into a specific machine code). The result of the translation phase is a symbolic program in SOAP language; i.e., one instruction per card in standard alphanumeric SOAP format.

The compiled output (translation) consists of four parts: (1) the main (symbolic) program; (2) the statement dictionary, to be described in the sequel; (3) the constants (abcons) used within statements; (4) ten reservation cards.

The code for each statement is punched out as soon as the statement has been translated. A statement dictionary entry is the first card punched for each statement. This dictionary provides the linkages for transfer statements. In addition to the program and the statement dictionary, a list of constants (those found in the statements together with several required by the finished program and furnished by the compiler itself) is punched together with ten cards which reserve space in the machine for these constants, the statement dictionary, and the problem variables.

For the program (II) which evaluates the polynomial of degree 10, the output of the translation phase is given in Table 35.

**General Remarks.** The following remarks concerning the translation produced are pertinent:

1. S0001 is the first location of the statement dictionary and contains a link to the (symbolic) location given to the statement whose identifier is 1.

2. Locations within statements are of the form L----, e.g., LAAAC or LBCAA, which refer to the third instruction within the first statement and the first instruction within the twenty-ninth statement respectively. The first instruction of each statement is always assigned a symbolic location.

3. Symbolic locations of the form E---- refer to the entry locations of subroutines (extensions). Those listed in the example are to floating point

## DIGITAL COMPUTER PROGRAMMING

TABLE 35. OUTPUT OF TRANSLATION PHASE FOR PROGRAM (II)  
WHICH EVALUATES A POLYNOMIAL OF DEGREE 10

1	S0001	00	0000	LAAAA	
2	LAAAA	LDD	LABAA	E00AQ	READ
3	S0002	00	0000	LABAA	
4	LABAA	RAL	A0007		Y2 Z 0J
5		STL	Y0002	LACAA	
6	S0003	00	0000	LACAA	
7	LACAA	RAL	I0005		I1 Z
8		STL	I0001	LADAA	I5
9	S0004	00	0000	LADAA	
10	LADAA	RAL	Y0002	LADAC	Y2 Z CI
11	LADAB	RAL	Y0001	LADAD	1 S Y1 X Y
12	LADAC	STL	ACC	LADAB	2 F
13	LADAD	LDD		E00AJ	
14		RAL	I0001		
15		SLT	I0003		
16		ALO		8002	
17		RAL	C		
18		LDD		E00AI	
19		STL	Y0002	LAEAA	
20	S0000	00	0000	LAEAA	
21	LAEAA	RSL	A0008		I1 Z
22		ALO	I0001		I1 S
23		STL	I0001	LAFAA	LM1R
24	S0000	00	0000	LAFAA	
25	LAFAA	RAL	I0001		G 0004
26		STL	W		IFL 1
27		RAL	A0008		RW I1
28		SLO	W		
29		BMI	LAFAG		
30		NOP	S	S0004	
31	LAFAG	NOP	LAFAG	LAGAA	
32	S0007	00	0000	LAGAA	
33	LAGAA	HLT	LAGAA	LAHAA	H
33	A	00	0000	0022	
34	A0008	00	0000	0001	
35	A0007	00	0000		
36	A0006	00	0000	3000	
37	A0005	00	0000	2000	
38	A0004	00	0000	1000	
39	A0003	00	0000	0010	
40	A0002	00	0000	0007	
41	A0001	00	0000	0001	
41 4			I	U0001	
41 3			I0002	0006	
41 4			Y	U0007	
41 3			Y0008	0009	
41 4			C	U0010	
41 3			C0011	0021	
41 4			S	U0022	
41 3			S0023	0029	
41 4			A	U0030	
41 3			A0031	0038	

arithmetic subroutines, e.g., E00AI is the entry to the floating point addition routine.

4. ACC is the floating point accumulator. *W* refers to a temporary storage location. The symbolic locations of the variables are their names.

5. All constants appearing in a statement, after being converted into the appropriate fixed or floating point format, are assigned an absolute constant location. In addition, certain constants needed during program operation are assigned, as required, by the compiler. The same constant will never be assigned more than one symbolic location. After the last statement has been translated, the abcon locations with their respective values are punched out, one per card.

6. Appearing as (SOAP) comments in the first and ensuing instructions of each statement is the original (untranslated) statement. A maximum of ten contiguous characters of the original statement may appear as comments with a single SOAP instruction.

Table 36 gives the name and meaning of all symbolic addresses or locations which may be found in a translated program.

TABLE 36. NAMES AND MEANINGS OF SYMBOLIC ADDRESSES  
OR LOCATIONS IN TRANSLATED PROGRAM

Name	Meaning
$L\alpha_1\alpha_2\beta_1\beta_2$	An instruction location
I 0000 + $n$	A fixed point variable, $I_n$
Y 0000 + $n$	A floating point variable, $Y_n$
C 0000 + $n$	A floating point variable, $C_n$
W 0000 + $n$ ( $0 \leq n \leq 9$ )	Temporary storage required by parentheses and/or quotation nesting
A 0000 + $n$	An absolute constant
ACC	Floating point accumulator
P 0000 + $n$ ( $0 \leq n \leq 9$ )	Subroutine parameter storage
E 00 $\alpha_1\alpha_2$	Name of (entry to) a subroutine
S 0000 + $n$	Entry in the statement dictionary

P0-P9 is a temporary storage block used only for temporary storage of subroutine parameters, including those used by TYPE statements. The floating point accumulator, ACC, is required for the floating point arithmetic subroutines. The first four abcons, A0 through A3 will contain the machine locations of S0, I0, Y0, C0 respectively. Contiguous blocks of storage are assigned, in the order given, to (a) variables (I, Y, C), (b) statement dictionary (S), (c) absolute constants (A), (d) the compiled program (L), including extensions (E).

The actual assignments are indicated on (SOAP) reservation cards, which are the last ten cards punched during compilation. The first instruction of the first statement, LAAAA, always appears in 650 location 1999.

**Examples of IT Language Problems.** The two examples given here are problems as written for the IT language, a root determination process

and a simple differential-equation integration. These are adapted from problems given in the M173 course at Michigan by B. A. Galler and the author.

**EXAMPLE 1. Problem.** Solve the equation  $ax^3 + bx^2 + cx + d = 0$  by Newton's method by using various initial values ( $x_0$ ) and various criteria for accuracy of the solution.

*Analysis.* Newton's method for the equation  $f(x) = 0$ .

$$x_0 \text{ given: } x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}.$$

For the equation of this problem,

$$x_0 \text{ given: } x_{i+1} = x_i - q_i,$$

where

$$q_i = \frac{ax_i^3 + bx_i^2 + cx_i + d}{3ax_i^2 + 2bx_i + c}.$$

Criterion for stopping iteration:

$$|q_i| = |x_{i+1} - x_i| < \epsilon$$

*Flow Diagram of Solution Procedure.* See Fig. 29.

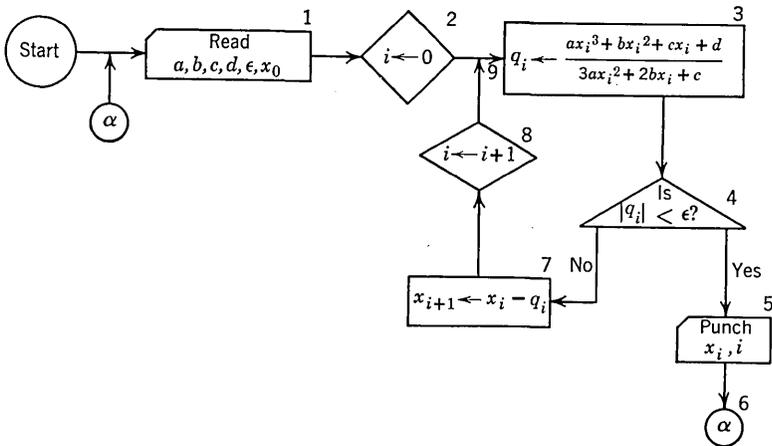


FIG. 29. Flow diagram for Example 1.

*Rough Program.*

- 1: Read  $a, b, c, d, x_0, \epsilon$
- 2:  $i \leftarrow 0$
- 3:  $q \leftarrow (ax^3 + bx^2 + cx + d) / (3ax^2 + 2bx + c)$
- 4: Go to 7 if  $|q| \geq \epsilon$
- 5: Punch  $x, i$
- 6: Go to 1

```

7:  x ← x - q
8:  i ← i + 1
9:  Go to 3
    
```

*Correct Compiler Program: Note.* We make the following storage assignments:

i:	I1	a:	C1	d:	C4
x:	Y1	b:	C2	ε:	C5
q:	Y2	c:	C3		

```

1:  a, b, c, d, ε, x0 READ                                F
2:  I1 ← 0                                                F
3:  Y2 ← ((C1 · Y1 · Y1 · Y1) + (C2 · Y1 · Y1) + (C3 · Y1)
      + C4)/((3 · C1 · Y1 · Y1) + (2 · C2 · Y1) + C3)    F
4:  G7 IF |Y2| ≥ C5                                       F
5:  TY1  TI1                                             F
6:  G1                                                    F
7:  Y1 ← Y1 - Y2                                         F
8:  I1 ← I1 + 1                                          F
9:  G3                                                    FF
    
```

*Note.* To produce a more efficient and accurate program, Statement 3 could be replaced by

$$Y2 \leftarrow \frac{((((((C1 \cdot Y1) + C2) \cdot Y1) + C3) \cdot Y1) + C4)}{(((3 \cdot C1 \cdot Y1) + 2 \cdot C2) \cdot Y1) + C3}.$$

**EXAMPLE 2.** *Problem.*  $y' = x - y^2$  to be solved from  $x = x_0$  to  $x = z$  with initial conditions  $y(x_0) = y_0$ .

*Method of Solution.* Step-by-step integration (Euler process).

$$\Delta y = (x - y^2) \Delta x.$$

*Initial Conditions and Parameters.*

$$\Delta x = 0.1, \quad x = 1, \quad y = 1, \quad z = 1.5.$$

*Rough Program.*

*Final Program.*

1:	$\Delta x, x, y, z$ READ	1:	C1, Y0, Y1, Y2 READ
2:	Tx Ty	2:	TY1 TY0
3:	$\Delta y \leftarrow (x - y^2) \Delta x$	3:	$C0 \leftarrow (Y1 - (Y0 \times Y0)) \cdot C1$
4:	$y \leftarrow y + \Delta y$	4:	$Y0 \leftarrow Y0 + C0$
5:	$x \leftarrow x + \Delta x$	5:	$Y1 \leftarrow Y1 + C1$
6:	Tx Ty	6:	TY1 TY0
7:	G3 IF $x < z$	7:	G3 IF $Y1 < Y2$
8:	H	8:	H

*Storage Assignments.*  $y$ : Y0,  $x$ : Y1,  $z$ : Y2,  $\Delta y$ : C0,  $\Delta x$ : C1.

Flow Diagram. See Fig. 30.

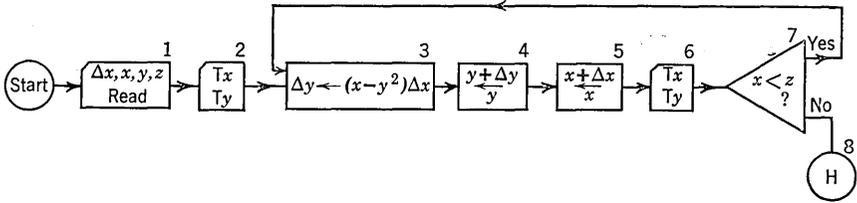


FIG. 30. Flow diagram for Example 2.

**An Example of an Extension of the IT Language.** An example of a typical extension to the IT language is given by the subroutine for the solution of a system of ordinary differential equations using the Runge-Kutta-Gill technique. The complete version, as available to the programmer, is given below. As with all compiler subroutines, the written outline includes: (1) number, (2) title, (3) number of locations occupied, (4) storage needed in addition to subroutine, (5) description, including method of entry, (6) flow diagram, (7) assembly language (SOAP) version of the subroutine code, (8) test example in compiler language.

Of course, entry in the IT language is relatively easy, since all the storage of parameters, setting up of return addresses, etc., is handled by the translator-compiler and the subroutine itself.

### University of Michigan IT Subroutine Library.

*Extension No.* 26.

*Title.* Runge-Kutta solution of differential equations.

*No. of locations.* 101.

*Storage needed in addition to subroutine.*  $Y(j), Y(j+1), \dots, Y(j+3n-1)$ , where  $j$  is any base subscript and  $n$  is the order of differential equation systems to be solved.

*Error indication.* Program Register 69xxxx9271 indicates floating point overflow or underflow.

*Description.* This is a floating point program based on the Gill version of the fourth order Runge-Kutta process. The equations are assumed to have the form

$$y'_i = f_i(y_0, y_1, \dots, y_{n-1}) \quad (i = 0, 1, \dots, n-1),$$

where  $y_0$  is the independent variable and is regarded as the solution to the equation

$$y'_0 = f(y_0, y_1, \dots, y_{n-1}) \equiv 1.$$

The user must reserve for the subroutine a block of  $3n$  consecutive  $Y$  variables, starting with  $Y(j)$ , for some  $j$ , and ending with  $Y(j+3n-1)$ . The first  $n$  of these locations will contain the current values of  $y_0, y_1, \dots, y_{n-1}$ ,

the next  $n$  locations will contain the values of  $k_0, k_1, \dots, k_{n-1}$  (where  $k_i = \Delta y_i = y'_i \Delta x$ ), and the next  $n$  locations will be used by the subroutine for the so-called bridging  $q$ 's.

The subroutine is entered by means of a single extension statement (with a nonzero statement number) of the form

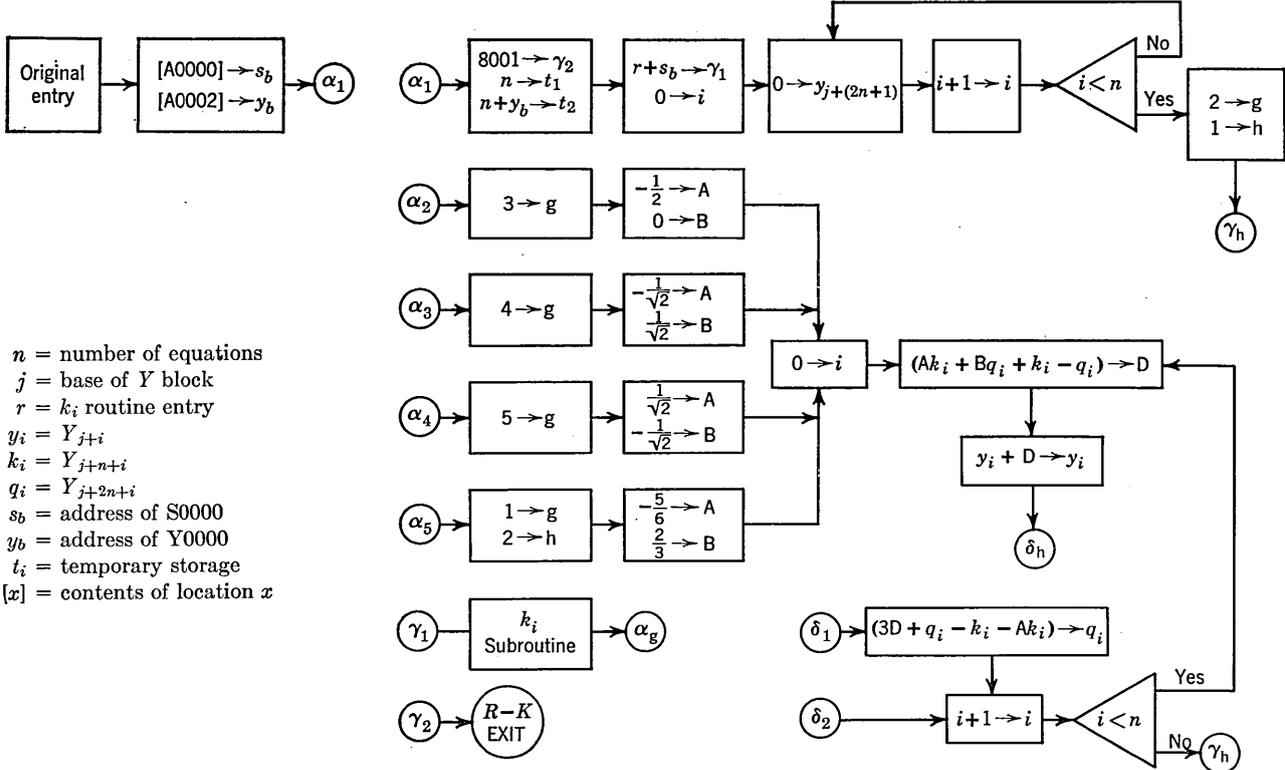
h:    "26E,  $n, j, r$ ",

where  $n$  is a fixed point expression whose value is the order of the equation system to be solved,  $j$  is a fixed point expression whose value is the base subscript of the block of  $Y$  variables provided for the subroutine, and  $r$  is a fixed point expression whose value is the nonzero statement number of the first statement of the  $k$  program described below. The expressions for  $n, j,$  and  $r$  may not involve extensions. Moreover, the statement immediately following the extension statement must also have a nonzero statement number.

Before entering the subroutine the user must (a) set  $Y(j), Y(j+1), \dots, Y(j+3n-1)$  to the initial values of  $y_0, y_1, \dots, y_{n-1}$ , (b) set  $n, j,$  and  $r$  to their respective values. The subroutine will itself clear  $q_0, q_1, \dots, q_{n-1}$ , to zero.

Since the  $f_i$  must be evaluated several times during the computation of the Runge-Kutta subroutine, a part of the subroutine must be supplied by the user. (It should be regarded as a subroutine for the Runge-Kutta program.) This small program, called the  $k$  program, computes new values of  $k_0, \dots, k_{n-1}$  from the current values of  $y_0, y_1, \dots, y_{n-1}$ , by using the definition  $k_i = y'_i(x)$ . The  $k$  program will actually be entered by the Runge-Kutta program four times during the computation of a single point of the solution. The only restrictions of the  $k$  program are that (1) it must begin with a nonzero statement number  $r$  (see description of extension statement above), and (2) it must end with a statement Gh, where h is the statement number of the extension statement.

At the end of the Runge-Kutta computation, the initial values of  $y_0, y_1, \dots, y_{n-1}$  will have been replaced by their values at the next solution point. These new values may then serve immediately as the initial values in the computation of the next set of values. The user must provide his own test to determine the end of the computation, and must also provide for punching out any of the  $y_i$  desired. The testing and punching portion of the program must follow immediately the extension statement calling for the Runge-Kutta subroutine. If after the process has been completed (i.e., the equations have been solved over the entire interval specified), the user seeks to reenter the subroutine again with different initial values, a different step size, or a different set of equations, he need only reinitialize  $n, j,$  and  $r$ . The accuracy of the solution is approximately  $(\Delta x)^5 y_i$ . See Fig. 31 for a flow diagram of the Runge-Kutta IT compiler and Table 37 for the SOAP language version of the program.



$n$  = number of equations  
 $j$  = base of Y block  
 $r = k_i$  routine entry  
 $y_i = Y_{j+i}$   
 $k_i = Y_{j+n+i}$   
 $q_i = Y_{j+2n+i}$   
 $s_b$  = address of S0000  
 $y_b$  = address of Y0000  
 $t_i$  = temporary storage  
 $[x]$  = contents of location  $x$

FIG. 31. Flow diagram of Runge-Kutta IT compiler extension 26.

TABLE 37. SOAP LANGUAGE VERSION

E00BA	STD	BAEX	BAN9		LDD	BAA1	BAN5
BAN9	STL	BAN		BAN5	STD	E00BA	
	-RAL	1809			STU	BACA	
	LDD	BAN1			RSL	8002	
	SDA	BAN1			STL	BACB	
	LDD	BAN2			RAL	BAYJ	
	SDA	BAN2			RAA	8001	
	-RAB	0000	BAN1		ALO	BAN	
BAN1	LDD	A0000			RAB	8002	
	STD	BASB			ALO	BAN	
	-AXB	0002	BAN2		RAC	8002	BAN6
BAN2	LDD	A0002		BAN6	RAU	BACA	
	STD	BAYB	BAN3		-FMP	0000	B
BAA1	STD	BAEX	BAN9		STU	ACC4	
	STL	BAN	BAN3		RAU	BACB	
BAN3	RAL	BAYB			-FMP	0000	C
	-ALO	P0002			FAD	ACC4	
	STL	BAYJ			-FAD	0000	B
	ALO	BAN			-FSB	0000	C
	ALO	8001			STU	ACC5	
	RAA	8001			-FAD	0000	A
	RAB	8002			-STU	0000	A
	STL	BAYJN	BAN4	BAD	RAU	BA8	BAN7
BAN4	-STU	0000	B	BAN7	FMP	ACC5	
	-AXB	0001			-FAD	0000	C
	-SXA	0001			-FSB	0000	B
	NZA	BAN4			FSB	ACC4	
	RAL	BASB			-STU	0000	C
	-ALO	P0001		BAN8	-AXA	0001	BAN8
	STL	BAKE			-AXB	0001	
	LDD	BAA2			-AXC	0001	
	STD	E00BA			RAL	8006	
	LDD	BA1			SLO	BAYJN	BAG
	STD	BAD		BAG	BMI	BAN6	BAKE
	LDD	BA2		BA1	RAU	BA8	BAN7
	STD	BAG	8001	BA2	BMI	BAN6	BAKE
BAA2	RSU	BA4		BA3	BMI	BAN6	BAEX
	LDD	BAA3	BAN5	BASB	00	0000	
BAA3	RSU	BA5		BAYB	00	0000	
	ALO	8003		BAEX	00	0000	
	LDD	BAA4	BAN5	BAKE	00	0000	
BAA4	RAU	BA5		BACA	00	0000	
	ALO	8003		BACB	00	0000	
	LDD	BAA5	BAN5	BAYJ	00	0000	
BAA5	LDD	BAN8		BAYJN	00	0000	
	STD	BAD		BA4	50	0000	0050
	LDD	BA3		BA5	70	7106	7850
	STD	BAG		BA6	83	3333	3350
	RSU	BA6		BA7	66	6666	6750
	SLO	BA7		BA8	30	0000	0051
				BAN	00	0000	

Negative Instruction Means Fixed Data Address.

*Compiler Program for Runge-Kutta Test Case.* Given  $d^2y/dx^2 = 6x$  with initial conditions  $(y)_{x=0} = 0$ ,  $(dy/dx)_{x=0} = 0$ , determine  $y$  in the interval  $0 \leq x \leq 3$ , for  $h = 0.1$ . Written as a system of three equations, the problem becomes,

$$\begin{aligned}y'_0 &= 1, \\y'_1 &= y_2, \\y'_2 &= 6y_0,\end{aligned}$$

with initial conditions,

$$\begin{aligned}y_0 &= 0, \\y_1 &= 0, \\y_2 &= 0.\end{aligned}$$

For this problem the following compiler variables are defined.

$$\begin{aligned}y_0, y_1, y_2 &= Y5, Y6, Y7, \\k_0, k_1, k_2 &= Y8, Y9, Y10, \\q_0, q_1, q_2 &= Y11, Y12, Y13, \\h &= C1 = 0.1, \\n &= I1 = 3, \\j &= I2 = 5, \\r &= I3 = 8.\end{aligned}$$

$n$ ,  $j$ , and  $r$  could appear in the statements as constants if only one equation is to be solved by the subroutine.

For the header card

$$\begin{aligned}n_I &= 5, \\n_Y &= 15, \\n_C &= 2, \\n_S &= 15, \\N &= (2000 - 772) = 1228, \\n_E &= 750 \text{ (Package 3)}.\end{aligned}$$

Notice that the values of the  $n$ 's may be greater than the actual number of locations needed by the program.\*

Statements:

1.	C1, I1, I2, I3, READ	F
2.	3, I4, 0, 1, I1-1	F
3.	Y(I2 + I4) = 0	F
4.	Q26E, I1, I2, I3Q	F
5.	TYI2 TY(I2 + 1)	F
6.	G4 IF 3.0 V YI2	F
7.	H	F

\* Program author: Bruce W. Arden.

8.	$Y(I2 + 3) = C1$	F
9.	$Y(I2 + 4) = Y(I2 + 2) \cdot C1$	F
10.	$Y(I2 + 5) = 6.0 \cdot YI2 \cdot C1$	F
11.	G4	FF

**Run Request.** For use with the IT language and its translation on the IBM 650 at the University of Michigan, a Run Request of punch-card size is used to indicate operations to be performed. The programmer indicates what parts of the one-step or two-step system he wants performed. He may indicate one or all of the options.

- a. Compile (translate by using the IT language compiler-translator).
- b. Translate from SOAP II to IBM 650 five-word-per-card basic language, with possible options as to SOAP subroutines and reservation decks for IT extensions (IT language subroutines).
- c. Perform an actual run on a basic machine language problem, obtained either from hand coding (straight) or compiled. With this, he indicates any special subroutines, the particular package of the various IT options to be used, the kind and amount of data to be read in, conditional punch output for diagnostic purposes, output kind and quantity expected, and any special utility routines to be used or other information to be obtained by the operator.

During the translation process the IT translator will punch error cards to indicate to the programmer that he has violated the conditions of the IT language [either the formation rules, based on two-letter pairs, to be described below, or the stop rules, which give the limits on the number characters per statement or per card, the number of nested subscripts or extensions (subroutines), etc.]. Such an error card will consist of the word "ERROR" followed by the offending two-character combination, or else a corresponding two-character code to indicate the violation of the stop rules.

This process, at the option of the computer operator, can be carried on with either the one-step or two-step process. Again at the option of the operator output may continue to be translated after an error has occurred or has been terminated. Generally, because of the nature of the two-step target language (SOAP) which uses symbolic addresses, an error in one statement will not affect the correctness of a later translation. The one-step target language (basic machine language) is an absolute address language which, if an error is made, will be incorrect thereafter. In the latter case, occurrence of an error always terminates the translation, although the translator may continue to scan input data so as to locate any other mistakes, without putting out any translation.

At the time of running a problem, one of several "subroutine packages," depending on what has been asked for by the programmer on the Run Request, will be stored in the machine by the operator. These packages will contain successively longer lists of subroutines starting with floating



allows the indication of the large number of options from which the programmer can choose in compiling, testing, and running his program.

**Translator Construction**

**The Symbol Pair Technique.** The IT compiler, as constructed originally by Perlis, Smith, and Van Zoeren, had built into it a set of checks for the formation rules of the language. This is accomplished by scanning statements from left to right, two characters at a time. Each symbol pair actuates a machine or assembly language generator which produces a corresponding sequence of one or more instructions in either basic IBM 650 or SOAP language dependent on whether or not this is a one-pass or two-pass compiler. Any symbol pair which is not admissible in the sense of occurring in a meaningful string of symbols in the language will cause an error alarm in the compiler itself.

The list of admissible symbol pairs, among the more than 1000 that could possibly occur, can be generated by laying out formation rules, such as that described in the example below. The syntax or grammar of the language is a slightly simplified version of that given earlier, with all references to norms of variables, statements, etc., omitted. Each rule in the language allows certain admissible symbol pairs. Each entity, when substituted into the succeeding rule, allows other admissible pairs formed by juxtaposition of the last symbol of certain entities combined with the first symbol of others.

Note that the symbol pair method of checking and translating allows errors to occur. For example, unless some type of memory is included in the compiler generators, such a sequence as

123.45.67

with two decimal points in a number, may be translated incorrectly, as well as passed by the translator as a perfectly well-formed string of symbols.

*Example of Formation Rule: Floating Point Constants.*

$n_1n_2 \dots n_k Bm \quad k \leq 8$  Examples: 14.92B3; 1066B(-11)

$n_1n_2 \dots n_k$ : fixed or floating points constant of type A with  $m = n_1n_2$  or  $(-n_1n_2)$

First symbols:  $n, . ;$  Last symbols:  $n, )$

Admissible pairs.

Bn      n)      -n      (-      B(      nB      .B

Restrictions: Upon occurrence of nB, counter set to 8. Decimal point has no effect. (Note: “(+)” is not considered admissible.)

**Permissible Symbol Pairs and Compiler Representation.** The end result of this rational sequence of generating all possible symbol pairs on the basis of the original rules describing the language can be given in Table 38.

TABLE 38. PERMISSIBLE SYMBOL PAIRS AND COMPILER REPRESENTATION

AC	AC	.X	JX	"X	QX	>C	WC
AI	AI	.n	Jn	"n	Qn	>I	WI
A(	AL	,A	KA	)/	RD	>.	WJ
A"	AQ	,C	KC	)F	RF	>(	WL
AY	AY	,F	KF	)I	RI	>"	WQ
B(	BL	,I	KI	),	RK	>Y	WY
Bn	Bn	,.	KJ	)—	RM	>n	Wn
CI	CI	,(	KL	)P	RP	XA	XA
C(	CL	,"	KQ	)"	RQ	XC	XC
Cn	Cn	,Y	KY	)	RR	XI	XI
/A	DA	,n	Kn	)+	RS	X.	XJ
/C	DC	(A	LA	)T	RT	X(	XL
DF	DF	(C	LC	)=	RU	X"	XQ
/I	DI	(I	LI	)>	RV	XY	XY
./	DJ	(.	LJ	)≥	RW	Xn	Xn
/(	DL	((	LL	)X	RX	YI	YI
/"	DQ	(—	LM	)←	RZ	Y(	YL
/Y	DY	("	LQ	+A	SA	Yn	Yn
/n	Dn	(Y	LY	+C	SC	←A	ZA
E,	EK	(n	Ln	+I	SI	←C	ZC
FA	FA	-A	MA	+	SJ	←I	ZI
FC	FC	-C	MC	+(	SL	←.	ZJ
FF	FF	-I	MI	+"	SQ	←(	ZL
FI	FI	-J	MJ	+Y	SY	←"	ZQ
F.	FJ	-(	ML	+n	Sn	←Y	ZY
F(	FL	-"	MQ	TC	TC	←n	Zn
F"	FQ	-Y	MY	TI	TI	n←	nZ
Fn	Fn	-n	Mn	T(	TL	nB	nB
GI	GI	PA	PA	TY	TY	n/	nD
G(	GL	PC	PC	=A	UA	nE	nE
Gn	Gn	PI	PI	=C	UC	nF	nF
HF	HF	P.	PJ	=I	UI	n.	nJ
IF	IF	P(	PL	=.	UJ	nI	nI
II	II	P"	PQ	=(	UL	n,	nK
I(	IL	PY	PY	="	UQ	n—	nM
In	In	Pn	Pn	=Y	UY	nP	nP
./	JD	"/	QD	=n	Un	nn	nn
.F	JF	"F	QF	>A	VA	n"	nQ
..	JK	"	QK	>C	VC	n)	nR
.—	JM	"—	QM	>I	VI	n+	nS
.P	JP	"P	QP	>.	VJ	nT	nT
."	JQ	""	QQ	>(	VL	n=	nU
.)	JR	")	QR	>"	VQ	n>	nV
.+	JS	"+	QS	>Y	VY	n≥	nW
.=	JU	"=	QU	>n	Vn	nX	nX
.>	JV	">	QV	>A	WA	n	nZ
>	JW	">	QW				

**A More Rational Procedure for Translator Construction.** The weaknesses of the symbol pair process are apparent. Errors when symbols like decimal points are repeated indicate that all recursive definitions cannot be checked without the complicated process of adding memory to the generators, which becomes completely an *ad hoc* process.

A multiple scan process is therefore necessary to produce a complete set of checks on the formation rules, as well as rational setting up of generator entries. In describing this, a notation of *productions*, or language definition rules, as described in Rosenbloom (Ref. 88), will be used. The list of productions that follows below can be seen to parallel the original set of rules set up for the IT language (again questions of variable statement norms are omitted for simplicity). (Production 1 below can be read as "If  $\alpha$  is a number, then  $\alpha$  is a fixed point integer constant." Production 2 would read, "If the string  $\alpha$  is a fixed point integer constant, and if the string  $\beta$  is a fixed point integer constant, then the concatenated string  $\alpha\beta$  is a fixed point integer constant." Production 27 would read, "If the string  $\alpha$  is an operand,  $\beta$  is an operator, and the string  $\gamma$  is an operand, then the string  $(\alpha\beta\gamma)$  is an operand.")

From the set of 46 productions, each of which defines a new string which can be developed in terms of previous older ones, a "production tree" will be set up that describes this definition process graphically.

**Formal Representation of the IT Language.** Writing the formation rules in a symbolic fashion, one defines the following classes:

- $\mathcal{N}$  numbers
- $\mathcal{J}$  fixed point integer constants
- $\mathcal{A}$  floating point constants type *a*
- $\mathcal{B}$  floating point constants type *b*
- $\mathcal{C}$  all constants
- $\mathcal{X}$  fixed point integer variables
- $\mathcal{F}$  floating point variables
- $\mathcal{V}$  all variables
- $\Delta$  operations
- $\mathcal{E}$  extensions
- $\mathcal{D}$  fixed point operands
- $\mathcal{O}$  operands
- $\mathcal{P}$  subroutine parameter sets
- $\mathcal{S}$  statements
- $\mathcal{U}$  unconditional transfers
- $\mathcal{R}$  relationships
- $\mathcal{X}$  conditional decisions

Then, by adding these symbols to the language one can obtain a formal system based on a set of axioms, and a set of productions, that produce all

possible constants, variables, extensions, operands, and statements. One can consider that the extra symbols carried above are used in the machine to designate an object of the particular type. Small Greek letters will indicate strings of symbols.

*Axioms.* A0.  $\mathcal{X}I, \mathcal{F}Y, \mathcal{F}C, \mathcal{U}0, \mathcal{U}1, \dots, \mathcal{U}9, \mathcal{R} =, \mathcal{R} >, \mathcal{R} \geq, \text{Send HF, Send READ F, } \Delta +, \Delta -, \Delta x, \Delta /, \Delta \text{ exp}$

*Productions:*

- P1.  $\mathcal{U}\alpha \rightarrow \mathcal{J}\alpha$
- P2.  $\mathcal{J}\alpha, \mathcal{J}\beta \rightarrow \mathcal{J}\alpha\beta$
- P3.  $\mathcal{J}\alpha, \mathcal{J}\beta \rightarrow \mathcal{A}\alpha.\beta$
- P4.  $\mathcal{J}\alpha, \mathcal{J}\beta \rightarrow \mathcal{B}\alpha\mathcal{B}\beta$
- P5.  $\mathcal{A}\alpha, \mathcal{J}\beta \rightarrow \mathcal{B}\alpha\mathcal{B}\beta$
- P6.  $\mathcal{J}\alpha, \mathcal{J}\beta \rightarrow \mathcal{B}\alpha\mathcal{B}(-\beta)$
- P7.  $\mathcal{A}\alpha, \mathcal{J}\beta \rightarrow \mathcal{B}\alpha\mathcal{B}(-\beta)$
- P8.  $\mathcal{J}\alpha \rightarrow \mathcal{C}\alpha$
- P9.  $\mathcal{A}\alpha \rightarrow \mathcal{C}\alpha$
- P10.  $\mathcal{B}\alpha \rightarrow \mathcal{C}\alpha$
- P11.  $\mathcal{J}\alpha \rightarrow \mathcal{X}I\alpha$
- P12.  $\mathcal{X}\alpha \rightarrow \mathcal{X}I\alpha$
- P13.  $\mathcal{X}\alpha \rightarrow \mathcal{F}Y\alpha$
- P14.  $\mathcal{X}\alpha \rightarrow \mathcal{F}C\alpha$
- P15.  $\mathcal{J}\alpha \rightarrow \mathcal{F}Y\alpha$
- P16.  $\mathcal{J}\alpha \rightarrow \mathcal{F}C\alpha$
- P17.  $\mathcal{D}\alpha \rightarrow \mathcal{F}Y\alpha$
- P18.  $\mathcal{D}\alpha \rightarrow \mathcal{F}C\alpha$
- P19.  $\mathcal{D}\alpha \rightarrow \mathcal{X}I\alpha$
- P20.  $\mathcal{F}\alpha \rightarrow \mathcal{V}\alpha$
- P21.  $\mathcal{X}\alpha \rightarrow \mathcal{V}\alpha$
- P22.  $\mathcal{V}\alpha \rightarrow \mathcal{O}(\alpha)$
- P23.  $\mathcal{C}\alpha \rightarrow \mathcal{O}\alpha$
- P24.  $\mathcal{V}\alpha \rightarrow \mathcal{O}(-\alpha)$
- P25.  $\mathcal{O}\alpha \rightarrow \mathcal{O}(\alpha)$
- P26.  $\mathcal{O}\alpha \rightarrow \mathcal{O}A\alpha$
- P27.  $\mathcal{O}\alpha, \Delta\beta, \mathcal{O}\gamma \rightarrow \mathcal{O}(\alpha\beta\gamma)$
- P28.  $\mathcal{X}\alpha \rightarrow \mathcal{D}\alpha$
- P29.  $\mathcal{J}\alpha \rightarrow \mathcal{D}\alpha$
- P30.  $\mathcal{D}\alpha, \Delta\beta, \mathcal{D}\gamma \rightarrow \mathcal{D}(\alpha\beta\gamma)$
- P31.  $\mathcal{D}\alpha \rightarrow \mathcal{O}\alpha$
- P32.  $\mathcal{O}\alpha \rightarrow \mathcal{P}\alpha$
- P33.  $\mathcal{O}\alpha, \mathcal{O}\beta \rightarrow \mathcal{P}\alpha, \beta$
- P34.  $\mathcal{P}\alpha, \mathcal{O}\beta \rightarrow \mathcal{P}\alpha, \beta$
- P35.  $\mathcal{J}\alpha, \mathcal{P}\beta \rightarrow \mathcal{E} \text{ "}\alpha\mathcal{E}, \beta\text{"}$

- P36.  $\mathcal{E}\alpha \rightarrow \mathcal{O}\alpha$   
 P37.  $\mathcal{V}\alpha, \mathcal{V}\beta \rightarrow \text{Send } \alpha \leftarrow \beta\text{F}$   
 P38.  $\mathcal{V}\alpha, \mathcal{O}\beta \rightarrow \text{Send } \alpha \leftarrow \beta\text{F}$   
 P39.  $\mathcal{D}\alpha \rightarrow \mathcal{U}\text{G}\alpha$   
 P40.  $\mathcal{X}\alpha \rightarrow \mathcal{U}\text{G}(\alpha)$   
 P41.  $\mathcal{U}\alpha \rightarrow \text{Send } \alpha\text{F}$   
 P42.  $\mathcal{O}\alpha, \mathcal{R}\beta, \mathcal{O}\gamma \rightarrow \mathcal{K}\alpha\beta\gamma$   
 P43.  $\mathcal{U}\alpha, \mathcal{K}\beta \rightarrow \text{Send } \alpha\text{IF}\beta\text{F}$   
 P44.  $\mathcal{V}\alpha, \mathcal{V}\beta, \mathcal{V}\gamma, \mathcal{V}\delta \rightarrow \text{Send } \text{T}\alpha\text{T}\beta\text{T}\gamma\text{T}\delta\text{F}$   
 P45.  $\mathcal{E}\alpha \rightarrow \text{Send } \alpha\text{F}$   
 P46.  $\mathcal{J}\alpha, \mathcal{P}\beta \rightarrow \text{Send } \alpha, \beta, \text{F}$

Now invert the process of productions in order to develop an inverse process that will allow us to check any arbitrary string  $\alpha$  in our original symbols. Speaking intuitively, we are looking for a method by which we can guarantee that any statement in the Perlis IT language obeys the rules for generating such statements that we have set up. Formally, we should add two new symbols,  $\tau$  (true) and  $\phi$  (false), to our present language to indicate intuitively that a string obeys the rules or does not, and then develop an entirely new set of productions (computer program!) that will produce the string  $\tau\alpha$  whenever our original string obeys the rules of the language, and  $\phi\alpha$  whenever our original string does not obey the rules.

Such a procedure is a *decision process*, one which arises over and over again in formal mathematics. In terms of the description here, a decision process is nothing but a set of productions which separates out a particular class of strings in a language. Other decision processes pertinent to computer programming include the following:

1. Decide whether or not a given computer program runs to completion (does not end in a loop) without ever running it.
2. Decide whether a given iterative process converges, without ever actually calculating with it.

Instead of writing out a set of such productions, we will concentrate here on the process by which they might be obtained. In particular we include a production tree, a graph describing the definition sequence given of the productions, and the inverse of such a graph, a similar tree describing the decision process itself.

By starting at the symbol  $\mathcal{S}$ , which is the end point in the production process of producing all possible IT statements, one can proceed backward up the tree of Fig. 33, testing each path in the maze in turn. The results are shown in Fig. 34. Here, if a translator proceeds along the paths in the direction of the arrows, based on subsequent decisions as to whether or not

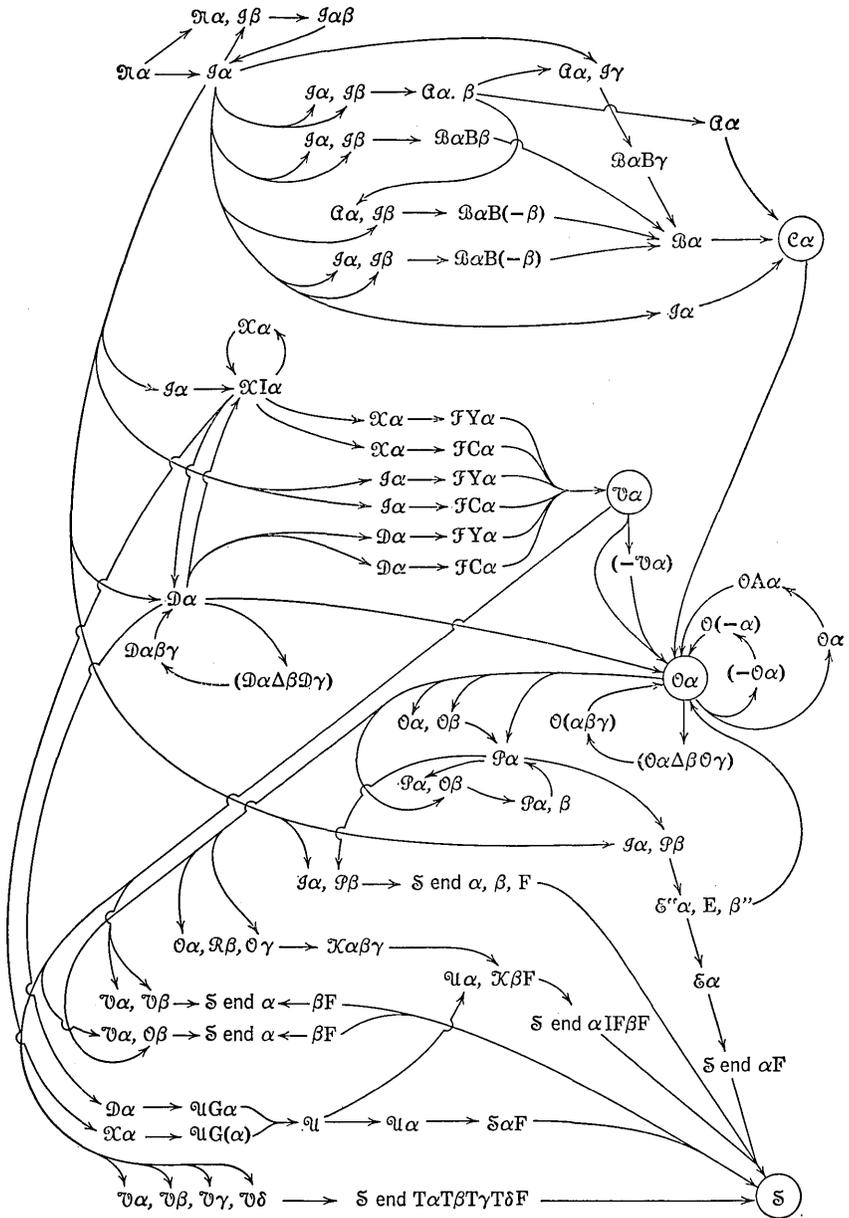


FIG. 33. Pattern for computing all possible statements in the IT language.

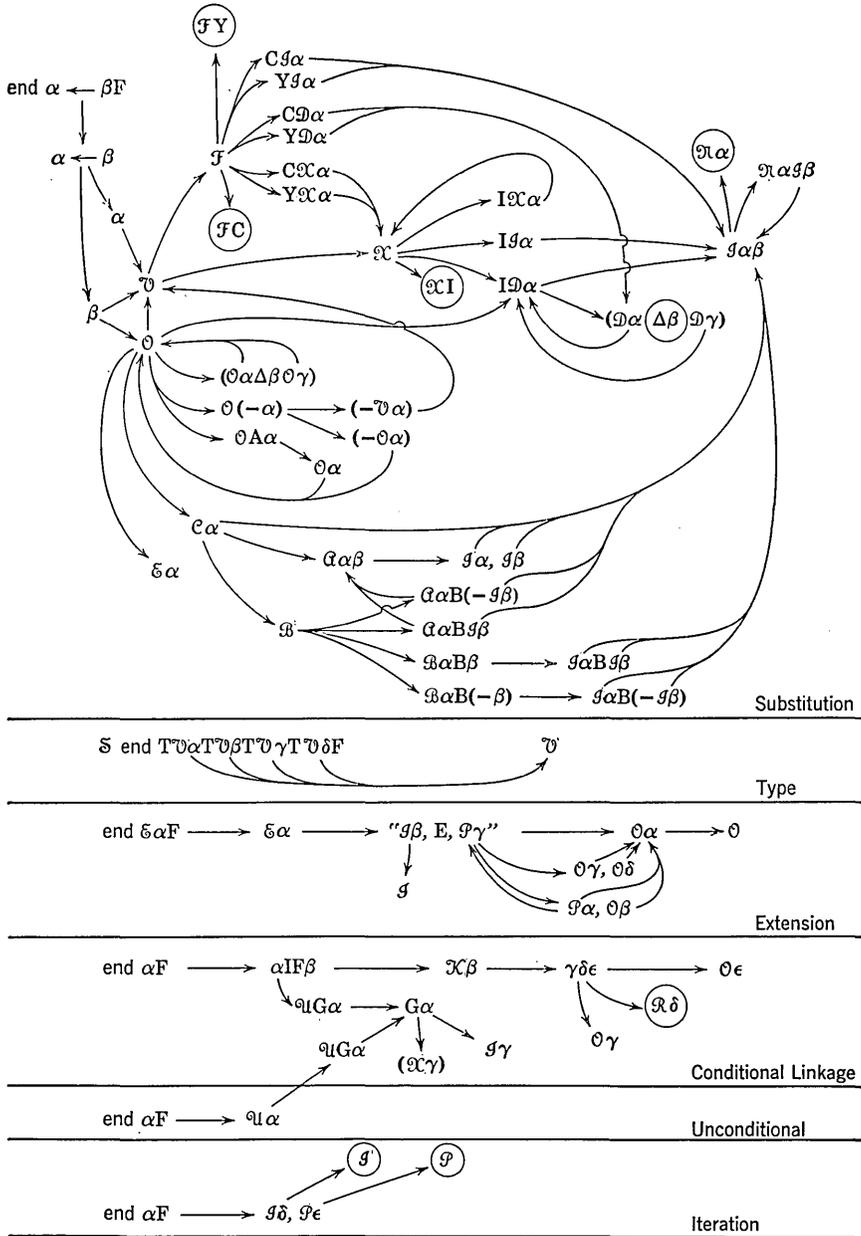


FIG. 34. Production tree for checking arbitrary strings.

the string has the form in question, it will arrive at a decomposition of the original language string into a hierarchical structure. For example,

$$Y2 \leftarrow (Y(I1 + 7) - C3)/(Y(I3 + 6) \times CI2)$$

will become decomposed into the structure of Fig. 35. Now if the translator starts at the bottom of this tree structure and performs a bottom-to-

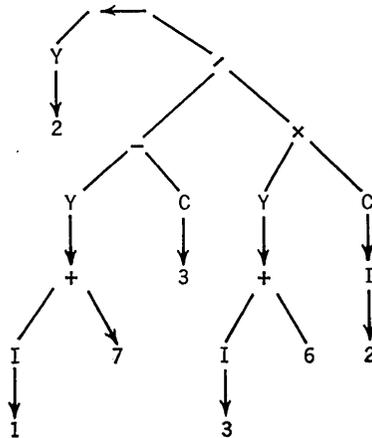


FIG. 35. Decomposition example:  
 $Y2 \leftarrow (Y(I1 + 7) - C3)/(Y(I3 + 6) \times CI2)$ .

top dictionary translation, a completely rational translation will be the end result, with no possibility for errors as in the symbol pair method. As Kantorovich (Ref. 60) has pointed out, this method of decomposition allows application of syntactical transformation rules (commutativity, associativity, etc.) to be applied before translation to produce more efficient hierarchical structures. Such procedures are reported by Kitov (Ref. 61) with reference to the compiler-translator for the Strela and in the Fortran description (Ref. 35).

This technique of formal decomposition also applies to natural languages, as might be expected. The difficulty here, of course, is not in the development of the decomposition process, but in the listing of the original definitions of acceptable strings (original productions) which are far more than the approximately 50 required by the IT language.

### 13. AUTOMATIC PROGRAMMING: A SOVIET ALGEBRAIC LANGUAGE COMPILER

#### Description

Mathematicians of the Soviet Union V. A. Steklov Mathematical Institute have developed a programming technique similar to that of the

algebraic language compilers in the United States and Western Europe, but enough different so that the technique should be included here separately. This technique, that of "operational programming," is due to Liapounov, Ianov, and others, and is described by Kitov and the originators (see Refs. 51, 52, and 61).

The Soviet programs, analogous to compiler-translators, translated literally as "programs that program," are based on an external notation that is written linearly across the page.

EXAMPLE. Adams' method for integrating an ordinary differential equation  $dy/dx = f(x, y)$ , requires three *arithmetic operators*:  $A_1$ , setting up initial data;  $A_2$ , computing the three starting ordinates of the integral curve;  $A_3^i$ , computing the ordinate in the next  $i$ th point in terms of the already known previous three ordinates. If these are combined with an iteration operator

$$\prod_{i=3}^{n-1}$$

the sequence describing the program performance for  $n$  steps would be

$$A_1 A_2 \prod_{i=3}^{n-1} A_3^i$$

If there is need to calculate  $m$  integral curves rather than one, the program description would become

$$\prod_{j=1}^m (A_1^j A_2^j \prod_{i=3}^{n-1} A_3^{(ij)})$$

The symbol  $\prod$  is one of a class of *logical operators*. (Note the similarity of this notation to the vertical statement language of the Internal Translator (see Ref. 81 and Sect. 12) with each arithmetic or logical "operator" replaced by an arithmetic or iteration "statement.")

A second type of logical operator is the logical condition, an arithmetic proposition that may be either true or false and is denoted by the forms

$$p(x = y)$$

$$p(x \cong y), \text{ etc.}$$

with the proposition being given within the parentheses. Ianov (Ref. 52) has suggested the use of brackets to indicate possible program sequence. Brackets would occur in pairs with the same subscripts. Ordinarily the sequence of operators in a "program scheme," as a line of operators describing a program is called, will indicate their sequence of performance. Following each logical condition a subscripted left bracket will indicate

possible change of control out of the usual left-to-right sequence. If the condition is *false*, the next operator to be performed will be that operator following the correspondingly subscripted right bracket. If the condition is *true*, the next operator to be performed will be the next operator in sequence.

EXAMPLE.  $A_1 p(x = y) \underset{1}{[} A_2 A_3 \underset{1}{]} A_4$

Interpretation. 1. Perform  $A_1$ . 2a. If  $x = y$ , perform  $A_2$ ,  $A_3$ , and  $A_4$ . 2b. If  $x \neq y$ , perform  $A_4$ .

Note. Indexing operators, including both prestoring, restoring, and incrementing of addresses, will be designated by the letter I.

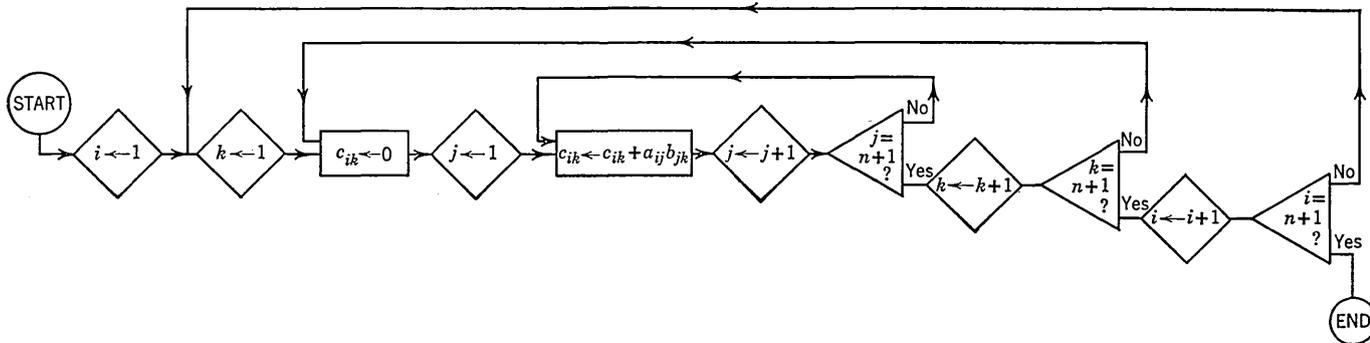
**Comparison with Other Methods.** Three comparable notations for describing the process of matrix multiplication are shown in Fig. 36. The first is the standard flow diagram notation already described. The second is the original string notation of Liapounov, designed for use with a computer with a three-address instruction logic. Here the bracket superscript indicates the *number of the operator* to which control is to be transferred when the logical condition is false. The bracket subscript indicates the *number of the operator* to which control is to be transferred when the logical condition is true.

The third notation is that of Ianov described above. The advantage of the third notation, as he and Kantorovich noted (Refs. 51 and 60), is that such an algebra of string transformations allows development of a set of transformation rules by which a "string language" of the Ianov type of notation may be manipulated to produce a "more efficient" program in the sense of some measure of effectiveness. Perlis and Smith (Ref. 155) have developed a string language manipulator for the IBM 704, based partially on the work of Markov (Ref. 67) on the theory of algorithms, which performs such manipulations easily. The string language manipulator is proposed to be used not only with such symbolic operational programs, but also algebraic language compiler-translators and natural language translators.

**Rules.** Ianov gives a complete set of transformation rules for programs. These rules, when applied to any program written in terms of operators, logical conditions, and left and right strokes, may be used to generate all possible equivalent variants of a program.

Let us call an *expression*, every finite line composed of various symbols, of operators, logical conditions and right strokes so that not more than one left and not more than one right stroke with the subscript  $i$  is found therein for every natural number 1. A logical condition  $\alpha \underset{i}{[}$  is subordinate, for a given set of parameters, to the logical condition  $\beta$  if  $\alpha$  is performed only when  $\beta$  is performed for the given set of parameters.

$$c_{ik} = \sum_{j=1}^n a_{ij}b_{jk} \quad (i = 1, \dots, n) \\ (k = 1, \dots, n)$$



Comparable Liapounov (three-address) string notation

$$I_1 \begin{array}{|c|} \hline 11 \\ \hline \end{array} \begin{array}{|c|} \hline 9 \\ \hline \end{array} A_3 I_4 \begin{array}{|c|} \hline 7 \\ \hline \end{array} A_5 I_6 P_7 (j = n + 1) \begin{array}{|c|} \hline 5 \\ \hline \end{array} I_8 P_9 (k = n + 1) \begin{array}{|c|} \hline 3 \\ \hline \end{array} I_{10} P_{11} (i = n + 1) \begin{array}{|c|} \hline 2 \\ \hline \end{array}$$

Comparable Ianov (single-address) string notation

$$I_1 \begin{array}{|c|} \hline 1 \\ \hline \end{array} I_2 \begin{array}{|c|} \hline 2 \\ \hline \end{array} A_3 I_4 \begin{array}{|c|} \hline 3 \\ \hline \end{array} A_5 I_6 P_7 (j = n + 1) \begin{array}{|c|} \hline 3 \\ \hline \end{array} I_8 P_9 (k = n + 1) \begin{array}{|c|} \hline 2 \\ \hline \end{array} I_{10} P_{11} (i = n + 1) \begin{array}{|c|} \hline 1 \\ \hline \end{array}$$

FIG. 36. Matrix multiplication (method 1). Various operational programming notations.

The following system of axiom schemes and derivation rules (where  $\mathcal{M}$ ,  $\mathcal{N}$  are arbitrary expressions;  $\mathcal{Q}(\mathcal{M})$  is an expression containing the expression  $\mathcal{M}$  formally;  $A_k$  are arbitrary operators;  $\alpha$ ,  $\beta$ , 0, 1 are the formulas of classical propositional calculus) is complete in the sense of the deducibility of every true formula of the form  $\mathcal{A} = \mathcal{B}$  for a given set of values of parameters, where  $\mathcal{A}$  and  $\mathcal{B}$  are program schemes.

*Note.* In the following equations the symbols  $\&$  and  $\vee$  are used for logical *and* and *inclusive or*.

$$\text{I. (1) } 0 \underset{i}{\lfloor} A_k \underset{i}{\rfloor} = 0 \underset{i}{\lfloor} \underset{i}{\rfloor};$$

$$(2) 1 \underset{i}{\lfloor} \mathcal{M} \underset{i}{\rfloor} = \mathcal{M};$$

$$(3) \underset{i}{\rfloor} \mathcal{M} 1 \underset{i}{\lfloor} = \mathcal{M}.$$

$$\text{II. (1) } (\alpha \& \beta) \underset{i}{\lfloor} \mathcal{M} \underset{i}{\rfloor} = \alpha \underset{i}{\lfloor} \beta \underset{j}{\lfloor} \mathcal{M} \underset{i}{\rfloor} \underset{j}{\rfloor};$$

$$(2) \underset{i}{\rfloor} \mathcal{M} (\alpha \& \beta) \underset{i}{\lfloor} = \underset{i}{\rfloor} \underset{i}{\rfloor} \mathcal{M} \alpha \underset{i}{\lfloor} \beta \underset{j}{\lfloor};$$

$$(3) \alpha \vee \beta \underset{i}{\lfloor} = \bar{\alpha} \underset{j}{\lfloor} \beta \underset{i}{\rfloor} \underset{j}{\rfloor}.$$

$$\text{III. } \mathcal{M}\mathcal{N} = 0 \underset{i}{\lfloor} \underset{j}{\rfloor} \mathcal{N} 0 \underset{k}{\lfloor} \underset{i}{\rfloor} \mathcal{M} 0 \underset{j}{\lfloor} \underset{k}{\rfloor}.$$

$$\text{IV. } \underset{i}{\rfloor} \underset{j}{\rfloor} = \underset{j}{\rfloor} \underset{i}{\rfloor}.$$

$$\text{V. } \alpha \underset{i}{\lfloor} \underset{i}{\rfloor} = \Lambda, \text{ where } \Lambda \text{ is an empty scheme (null-program).}$$

$$\text{VI. (1) } \alpha \underset{i}{\lfloor} \mathcal{M} \underset{i}{\rfloor} \alpha \underset{j}{\lfloor} \mathcal{N} \underset{j}{\rfloor} = \alpha \underset{i}{\lfloor} \mathcal{M} \alpha \underset{j}{\lfloor} \mathcal{N} \underset{j}{\rfloor} \underset{i}{\rfloor};$$

$$(2) \alpha \underset{i}{\lfloor} \mathcal{M} \underset{j}{\rfloor} \mathcal{N} \underset{i}{\rfloor} \alpha \underset{j}{\lfloor} = \alpha \underset{i}{\lfloor} \mathcal{M} \underset{j}{\rfloor} \underset{i}{\rfloor} \mathcal{N} \alpha \underset{j}{\lfloor};$$

$$(3) \underset{j}{\rfloor} \mathcal{M} \alpha \underset{i}{\lfloor} \mathcal{N} \underset{i}{\rfloor} \alpha \underset{j}{\lfloor} = \underset{j}{\rfloor} \underset{j}{\rfloor} \mathcal{M} \alpha \underset{i}{\lfloor} \mathcal{N} \alpha \underset{j}{\lfloor};$$

$$(4) \underset{i}{\rfloor} \alpha \underset{j}{\lfloor} \mathcal{M} \alpha \underset{i}{\lfloor} \mathcal{N} \underset{j}{\rfloor} = \alpha \underset{j}{\lfloor} \mathcal{M} \alpha \underset{i}{\lfloor} \mathcal{N} \underset{j}{\rfloor} \underset{i}{\rfloor};$$

$$(5) \underset{i}{\rfloor} \alpha \underset{j}{\lfloor} \mathcal{M} \underset{j}{\rfloor} \mathcal{N} \alpha \underset{i}{\lfloor} = \alpha \underset{j}{\lfloor} \mathcal{M} \underset{j}{\rfloor} \underset{i}{\rfloor} \mathcal{N} \alpha \underset{i}{\lfloor};$$

$$(6) \underset{j}{\rfloor} \mathcal{M} \underset{i}{\rfloor} \alpha \underset{j}{\lfloor} \mathcal{N} \alpha \underset{i}{\lfloor} = \underset{j}{\rfloor} \underset{j}{\rfloor} \mathcal{M} \alpha \underset{j}{\lfloor} \mathcal{N} \alpha \underset{i}{\lfloor};$$

$$\text{VII. } \underset{i}{\rfloor} \alpha \underset{i}{\lfloor} \mathcal{M} \underset{j}{\rfloor} \alpha \underset{j}{\lfloor} = \alpha \underset{i}{\lfloor} \mathcal{M} \underset{i}{\rfloor} \underset{j}{\rfloor} \alpha \underset{j}{\lfloor};$$

$$\text{VIII. } \frac{\alpha \equiv \beta}{\mathcal{M}(\alpha) \equiv \mathcal{M}(\beta)} \quad \text{(This implies that the bottom expression holds whenever the top one does.)}$$

$$\text{IX. } \frac{\mathcal{M} = \mathcal{N}, \mathcal{Q}(\mathcal{M}) \equiv \mathcal{R}}{\mathcal{Q}(\mathcal{N}) \equiv \mathcal{R}}, \text{ where } \mathcal{Q}(\mathcal{N}) \text{ is also an expression.}$$

X. The pair of corresponding strokes  $\lfloor_i, \rfloor_i$ , in one expression can be replaced by any other pair  $\lfloor_j, \rfloor_j$  but so that the expression would remain an expression.

XI. If the logical condition  $\alpha \lfloor_i$  for a given set of parameter values is subordinate to  $\beta$ , then  $\alpha \lfloor_i$  can be replaced by  $(\alpha \& \beta) \lfloor_i$ .

**Mechanization of This Notation**

The translation of the Liapounov-Ianov string programming for the Strela (see Sect. 6) has been performed semiautomatically. Each symbol in a string would be coded into a corresponding number. The strings of numbers would then be read into the computer and the corresponding subroutines generated and compiled, as in the IT compiler. However, the algebraic translation of operators written as statements, if desired, could be relatively easily accomplished.

**Example of Use of the Ianov Transformation Rules**

Given the program scheme of Fig. 36, in the Ianov notation, one may obtain the following sequence of transformations. It is assumed that duplication of any operators requires duplicate storage facilities.

$$1. \quad I_1 \lfloor_1 I_2 \rfloor_2 A_3 I_4 \rfloor_3 A_5 I_6 P_7 \lfloor_3 I_8 P_9 \lfloor_2 I_{10} P_{11} \lfloor_1 \quad (\text{Given}).$$

$$2. \quad I_1 \lfloor_1 (I_2 \& I_2) \rfloor_2 A_3 I_4 \rfloor_3 A_5 I_6 P_7 \lfloor_3 (I_8 P_9 \& I_8 P_9) \lfloor_2 I_{10} P_{11} \lfloor_1.$$

(Propositional calculus:  $\alpha \& \alpha \equiv \alpha$ )

$$3. \quad I_1 \lfloor_1 I_2 \rfloor_2 I_2 \rfloor_4 A_3 I_4 \rfloor_3 A_5 I_6 P_7 \lfloor_3 I_8 P_9 \lfloor_4 I_8 P_9 \lfloor_2 I_{10} P_{11} \lfloor_1.$$

(Rule II.2)

$$4. \quad I_1 \lfloor_1 I_2 \rfloor_2 A_3 I_8 P_9 \lfloor_2 I_2 \rfloor_4 I_4 \rfloor_3 A_5 I_6 P_7 \lfloor_3 I_8 P_9 \lfloor_4 I_{10} P_{11} \lfloor_1.$$

(Arithmetic independence of  $A_3$  and  $I_4 \rfloor_3 A_5 I_6 P_7 \lfloor_3$ .)

$$5. \quad (I_1 \& I_1) \lfloor_1 I_2 \rfloor_2 A_3 I_8 P_9 \lfloor_2 I_2 \rfloor_4 I_4 \rfloor_3 A_5 I_6 P_7 \lfloor_3 I_8 P_9 \lfloor_4 (I_{10} P_{11} \& I_{10} P_{11}) \lfloor_1.$$

(Propositional calculus)

$$6. \quad I_1 \lfloor_5 I_1 \rfloor_1 I_2 \rfloor_2 A_3 I_8 P_9 \lfloor_2 I_2 \rfloor_4 I_4 \rfloor_3 A_5 I_6 P_7 \lfloor_3 I_8 P_9 \lfloor_4 I_{10} P_{11} \lfloor_5 I_{10} P_{11} \lfloor_1.$$

(Rule II.2)

$$7. \quad I_1 \begin{array}{|c} \hline \text{ } \\ \hline \end{array} \begin{array}{|c} \hline \text{ } \\ \hline \end{array} I_2 \begin{array}{|c} \hline \text{ } \\ \hline \end{array} A_3 I_8 P_9 \begin{array}{|c} \hline \text{ } \\ \hline \end{array} I_{10} P_{11} \begin{array}{|c} \hline \text{ } \\ \hline \end{array} I_1 \begin{array}{|c} \hline \text{ } \\ \hline \end{array} I_2 \begin{array}{|c} \hline \text{ } \\ \hline \end{array} I_4 \begin{array}{|c} \hline \text{ } \\ \hline \end{array} A_5 I_6 P_7 \begin{array}{|c} \hline \text{ } \\ \hline \end{array} I_8 P_9 \begin{array}{|c} \hline \text{ } \\ \hline \end{array} I_{10} P_{11} \begin{array}{|c} \hline \text{ } \\ \hline \end{array}.$$

(Arithmetic independence of  $I_2 \begin{array}{|c} \hline \text{ } \\ \hline \end{array} A_3 I_8 P_9 \begin{array}{|c} \hline \text{ } \\ \hline \end{array}$ )

and

$$I_2 \begin{array}{|c} \hline \text{ } \\ \hline \end{array} I_4 \begin{array}{|c} \hline \text{ } \\ \hline \end{array} A_5 I_6 P_7 \begin{array}{|c} \hline \text{ } \\ \hline \end{array} I_8 P_9 \begin{array}{|c} \hline \text{ } \\ \hline \end{array}; \quad \prod_i MN = \prod_i M \prod_i N$$

if  $M$  and  $N$  are independent.)

$$8. \quad I_1 \begin{array}{|c} \hline \text{ } \\ \hline \end{array} I_2 \begin{array}{|c} \hline \text{ } \\ \hline \end{array} A_3 I_8 P_9 \begin{array}{|c} \hline \text{ } \\ \hline \end{array} I_{10} P_{11} \begin{array}{|c} \hline \text{ } \\ \hline \end{array} I_1 \begin{array}{|c} \hline \text{ } \\ \hline \end{array} I_4 \begin{array}{|c} \hline \text{ } \\ \hline \end{array} I_2 \begin{array}{|c} \hline \text{ } \\ \hline \end{array} A_5 I_8 P_9 \begin{array}{|c} \hline \text{ } \\ \hline \end{array} I_6 P_7 \begin{array}{|c} \hline \text{ } \\ \hline \end{array} I_{10} P_{11} \begin{array}{|c} \hline \text{ } \\ \hline \end{array}.$$

$(\prod_i \prod_j A_5 = \prod_j \prod_i A_5.)$

$$9. \quad I_1 \begin{array}{|c} \hline \text{ } \\ \hline \end{array} I_2 \begin{array}{|c} \hline \text{ } \\ \hline \end{array} A_3 I_8 P_9 \begin{array}{|c} \hline \text{ } \\ \hline \end{array} I_{10} P_{11} \begin{array}{|c} \hline \text{ } \\ \hline \end{array} I_4 \begin{array}{|c} \hline \text{ } \\ \hline \end{array} I_1 \begin{array}{|c} \hline \text{ } \\ \hline \end{array} I_2 \begin{array}{|c} \hline \text{ } \\ \hline \end{array} A_5 I_8 P_9 \begin{array}{|c} \hline \text{ } \\ \hline \end{array} I_{10} P_{11} \begin{array}{|c} \hline \text{ } \\ \hline \end{array} I_6 P_7 \begin{array}{|c} \hline \text{ } \\ \hline \end{array}.$$

$$\prod_j (\prod_k (I_2 \begin{array}{|c} \hline \text{ } \\ \hline \end{array} A_5 I_8 P_9 \begin{array}{|c} \hline \text{ } \\ \hline \end{array})) = \prod_k (\prod_j (I_2 \begin{array}{|c} \hline \text{ } \\ \hline \end{array} A_5 I_8 P_9 \begin{array}{|c} \hline \text{ } \\ \hline \end{array}))$$

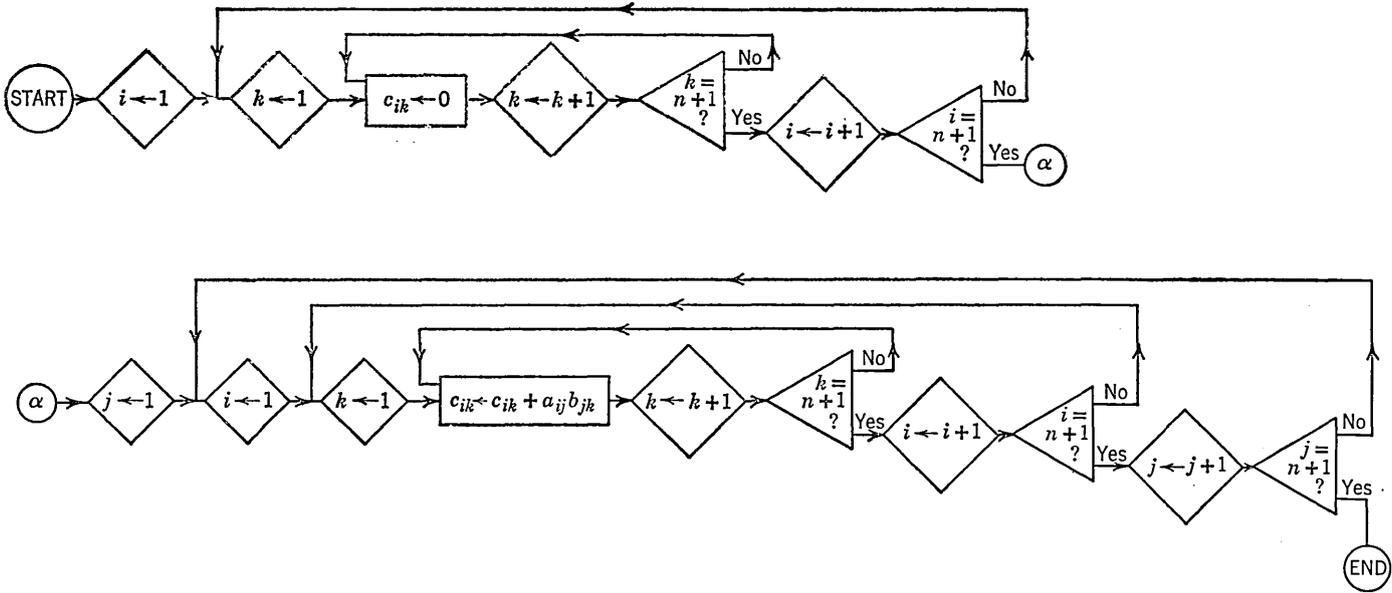
The resulting flow diagram for matrix multiplication is given in Fig. 37, and shows a less efficient, but at the same time less standard method of performing matrix multiplication. The reader is urged to draw the intermediate flow diagrams to picture the sequence of the program transformation. A similar discussion of such program equivalences, but not in this formal fashion, is given by Jeanel (Ref. 113).

## 14. AUTOMATIC PROGRAMMING: INTERPRETERS

### Interpretive Routines

Turing (Ref. 101) originally described a simple computer based on paper tape storage, the ability to read and write on it, and to move right or left along it, dependent on the symbols written on the tape. Turing showed that such a machine, restricted to reading and writing binary digits (1's and 0's) could nevertheless simulate the behavior of *any other* such machine, provided the first machine had a large enough set of different internal states (different configurations). Any machine that has this capacity of universal simulation of another computer, irrespective of symbols read or written and numbers of internal states in the machine being simulated, is now called a Universal Turing Machine. Moore (Ref. 73) describes a similar proof of Turing's original results. The design of Turing type machines is discussed in Chap. 31.

$$c_{ik} = \sum_{j=1}^n a_{ij}b_{jk} \quad (i = 1, \dots, n) \\ (k = 1, \dots, n)$$



*Ianov notation*

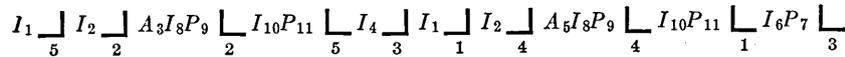


Fig. 37. Matrix multiplication (method 2). The transformed flow diagram of Fig. 36, obtained by using the Ianov method.

It can be shown that any general purpose digital computer with the facilities to read from, and transfer into its storage, is such a Universal Turing Machine. Therefore, any general purpose digital computer, given enough storage, can simulate in precise detail the internal behavior of any other such machine.

The key to this *interpretation* process is the use of a set of closed subroutines to represent the individual instructions of the computer being simulated. The instructions of the simulated machine may be considered *parameters* of these subroutines which tell which subroutine is to be used along with which operands. Interpretive routines first came into extensive use with the desire to perform floating point arithmetic on fixed point arithmetic machines. Arithmetic programs interpreted as floating point programs on the fixed point IBM 701, for example, were used extensively.

**EXAMPLE.** *Simulation of one computer on another computer.* Given a very simple single-address instruction digital computer with the following six instructions (the two-digit pairs at left are the operation code for the computer).

- (01) **inp n** Input one number from the external input-output unit.
- (02) **add n** Add the contents of location n into the accumulator.
- (03) **sub n** Subtract the contents of location n from the accumulator.
- (04) **sto n** Store the contents of the accumulator in location n.
- (05) **cmp n** Compare the contents of the accumulator with zero; if less than zero, take next instruction from location n<sub>1</sub>. Otherwise proceed in a normal fashion.
- (06) **out n** Output one number from location n through the input-output unit.

Figure 38 gives a flow diagram describing the process of simulating this computer on any other digital computer of sufficiently large storage capacity.

The following notation is used:  $I_j$  designates the instruction in the simulated computer being stored at position j of that computer. Each instruction and each number considered as an integer contains five digits. The instruction form is

$$I_j = c \times 10^3 + n,$$

where c is the two-digit operation and n the three-digit address.

The process of performance of interpretation is as follows. Instructions in the language of the computer being simulated are stored in the usual single-address sequence. The address of the first simulated instruction

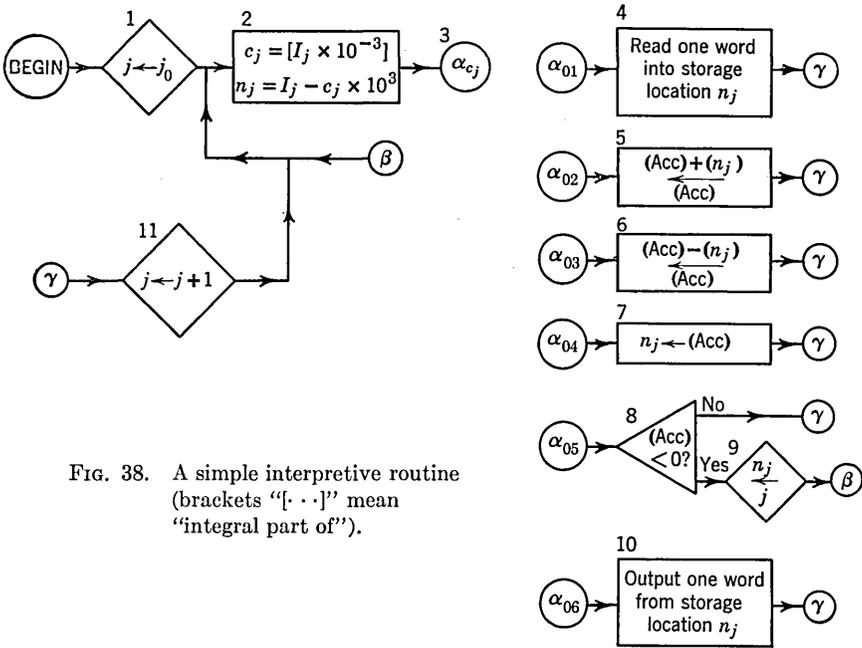


FIG. 38. A simple interpretive routine (brackets "[ · · · ]" mean "integral part of").

to be performed is given to the "host" machine via its external facilities: input-output, console switching, etc.

The following steps are involved in the interpretation (execution) of an instruction.

1. In box 1, this value replaces  $j$ , which serves as an "instruction counter" for the simulated machine.

2. In box 2, the five-decimal digit instruction  $I_j$  is decomposed into a two-digit integer  $c_j$ , representing the operation, and a three-digit integer  $n_j$ , representing the address of  $I_j$ .

3. In box 3, a remote connector, control is transferred to the subroutine corresponding to  $c_j$ .

4. Boxes 4, 5, 6, 7, and 10 perform the standard actions of the corresponding instruction, relative to a host machine location which has been labeled "Acc" (accumulator).

5. Upon completion of this subroutine, in box 11 the counter  $j$  is increased by one, corresponding to usual single-address computer behavior, and the process now returns to box 2 and step 2 above.

6. If on the other hand the simulated operation is *compare* (05), then control is transferred at step 4 to box 8. This now makes a decision on the basis of the contents of the location Acc. If that number is non-negative, the procedure is to box 11 and then step 2 above. If (Acc) is negative, the

“next instruction address” is placed into the “instruction counter” in box 9, and control is transferred back to box 2 and step 2 above.

This process will continue around the overall “loop” as long as instructions in the machine being simulated are available and behave “properly.” The reader is urged to follow through the sequence of operations for the following test program on the simulated machine. At the beginning of the program,  $j_0$  is 003.

<i>Location</i>	<i>Number or Instruction</i>	<i>Stored Word</i>	<i>Explanation</i>
000	00000	00 000	Temporary
001	00001	00 001	
002	-00001	-00 001	Constant for forced jump
→ 003	sto 000	04 000	(Acc) → (000)
004	sub 000	03 000	Clear Acc
005	add 001	02 001	$x_i$ to Acc
006	out 001	06 001	Print $x_i$
007	add 001	02 001	$x_i + x_i$
008	sto 001	04 001	$2x_i \rightarrow 001$
009	sub 001	03 001	Clear Acc
010	add 002	02 002	Make Acc Neg
011	cmp 003	05 003	Jump to 003

This program of the simulated machine will print the following integers in sequence:

$$1, 2, 4, 8, 16, \dots$$

until it overflows, performing the problem

$$x_0 = 1,$$

$$x_{i+1} = x_i + x_i.$$

The sequence taken through the boxes of the flow diagram of Fig. 38 will be as follows:

<i>Instruction Location</i>	<i>Instruction</i>	<i>Sequence through Boxes of Flow Diagram</i>
003	04 000	1, 2, 3, 7, 11
004	03 000	1, 2, 3, 6, 11
005	02 001	1, 2, 3, 5, 11
006	06 001	1, 2, 3, 10, 11
007	02 001	1, 2, 3, 5, 11
008	04 001	1, 2, 3, 7, 11
009	03 001	1, 2, 3, 6, 11
010	02 002	1, 2, 3, 5, 11
011	05 003	1, 2, 3, 8, 9
003	04 000	2, 3, 7, 11, etc.

This computer may be simulated on any general purpose digital computer that has a reasonable complement of operations, whether it be single-address, two-address, three-address, decimal, binary, etc. The further removed the host and simulated computers are in structure, the more complex the interpreter will be in the coding required; but otherwise there is no problem.

### Simulation of One Machine by Another

During the process of transferring work from one digital computer to another, users or manufacturers have made use of the interpretation principle to simulate a new not-yet produced digital computer. For example, the first such simulation was the preparation of an interpretative program simulating the IBM 704 on the IBM 701 before the former computer was completed. The list of such simulations includes:

1. IBM 704 on IBM 701 (IBM Programming Research)
2. IBM 650 on IBM 704 (Corporation for Economic and Industry Research)
3. Univac II on Univac I (Remington Rand Univac)
4. IBM 650 on Datatron 220 (Burroughs Corporation)
5. RW 300 on ERA 1103A (Ramo-Wooldridge Corporation)
6. IBM 704 (Share Assembly Program) on IBM 701 (University of Michigan)

Some of these simulating programs were not interpretive procedures, but rather compiling programs that compiled open subroutines dependent on the program (not the machine) being simulated. Simulation of a *program*, by a compilation technique, rather than a machine by an interpretive technique, can make a large difference in efficiency of performance. Some of the interpretive processes were slower than the original computer by several orders of magnitude.

**Acceptance Tests.** The simulation of one machine on another has been used extensively for equipment and program acceptance tests, especially by the armed services. If a special purpose computer is to be delivered, along with a certain program, by a contractor, a program is prepared for a general purpose computer to simulate the special purpose machine. After this simulation program is prepared, the contractor-prepared program for the special purpose computer can be run on the general purpose computer. The same program can be run on the special purpose computer. If the results agree, this constitutes an acceptance test for the special purpose machine. In addition, if both programs produce the desired results, this constitutes an acceptance test for the special purpose computer program. Figure 39 shows the process where the special purpose computer is "embedded," by the interpretive process, in the general purpose machine.

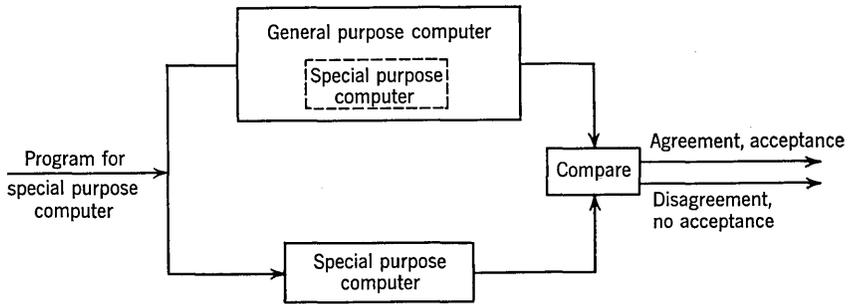


FIG. 39. Block diagram of acceptance test for a special purpose computer using an interpretive process.

### Tracing Programs

If one universal machine can simulate any other machine of a somewhat smaller storage capacity (which is what Turing's statement on universal machines means), it should therefore be possible for a computer to simulate a version of itself with a smaller amount of storage. Such self-interpretation is called "automonitoring," "checking," "tracing," "mistake diagnosis," etc., by different groups. The procedure is basically simple. Suppose the machine of the example above actually existed, and one wanted to be able to follow through every step of its action, with values of its internal locations printed out in detail. A box could be inserted between boxes 2 and 3 of Fig. 38 to obtain complete information about every instruction as it was performed. This is shown in Fig. 40 with the new box marked 2a.

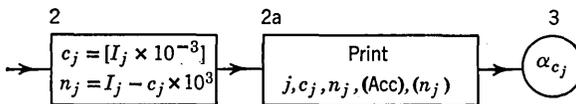


FIG. 40. Box added for print out.

Now if this flow diagram was coded on the original machine, which is now the one being simulated, programs could be run (using a smaller section of storage) that would perform in the standard fashion, and at the same time print out pertinent information such as: location, instruction, previous contents of Acc, and the previous contents of the address of the instruction. If two output print stations were available, one could be used for regular program output, the other for information derived from the tracing process.

The SEAC computer of the National Bureau of Standards had the

tracing ability built in as a hardware design feature. The computer could be operated in the "automonitor mode" and would print out the instruction counter, the instruction, and the contents of the three operand registers. It would do this optionally on each instruction or on each instruction designated as a *breakpoint* by a digit in the instruction. This feature has not been included on later machines; the interpretive program is always used. However, the IBM 704 does have the *trapping mode* defined in Sect. 6, which causes a transfer of control to a standard cell on each transfer of control instruction.

Most practical tracing programs involve techniques of decision-making on the part of the program to trace: (a) only certain instruction regions, (b) only certain specified types of operations (for example, transfers of control), and (c) only operations with certain addresses, etc.

Tracing may be done on computers with any address structure, although the example here is of a single-address instruction.

**EXAMPLE.** *Use of the Intercom system on the Bendix G-15.* The Bendix G-15 computer is a binary computer with 2160 words in its main storage and 16 words on a rapid-access magnetic drum loop (see Ref. 143). This computer has been designed to use a minimum of circuitry, with the logical structure based on transfers to and from the storage. For this reason, although the address system in an instruction resembles a three-address system, it is more complex in that it is described in terms of a source and timing number, destination and timing number, and next command timing number. The "sources" and "destinations" in many cases play the part of operations in the more common instruction logic.

To make programming for the G-15 easier, an interpretive routine has therefore been written that simulates a decimal, floating point computer with index registers (see Ref. 56). The simulated machine has available to it 864 locations as the "interpretive memory." An accumulator, or "A Register," is simulated by the interpretive routine. This holds one number and is used in a fashion analogous to the accumulators of the IBM 704, Univac 1103A, and IBM 650 computers already described.

*Index Registers.* There are eight sets of three-digit index registers in the Intercom system. Each set contains the following:

1. B register. This register stores a number, as usual, to be added to the address of an instruction at the time of performance.
2. D register. This register is used to store a number which may be added to the contents of the corresponding B register.
3. L register. This register contains a "limit" to which the B register may be compared in determining whether or not to transfer control.

*Numbers.* All numbers  $n$  are entered into the computer in a form equivalent to:

$$n = N \times 10^{t-50},$$

where  $N$  is a five-digit decimal fraction between 0.10000 and 0.99999 and  $t$  is a two-digit decimal integer

$$12 \leq t \leq 88$$

A number is therefore represented by the seven-digit couple  $(t.N)$  with a decimal point separating. Arithmetic operations are on six-digit numbers; results are rounded to five digits upon being typed out.

*Instructions.* Instructions are written in a six-digit single-address form, with the first two digits an operation code. The next three digits represent a location in the interpretive memory from 000 to 863. The sixth digit is ordinarily 0, except when one of the corresponding index register sets  $(B_i, D_i, L_i)$ , where  $i = 1, \dots, 8$ , is to be involved in the instruction either as an address modification or as an operand.

The instruction list of the Intercom system is given in Table 39. Instructions are normally taken in sequence from storage.

Of these instructions, the arithmetic operations in the first group correspond to the usual standard single-address instructions. The five operations affecting only the accumulator include four functions that are not usually found in a computer instruction code, and are here available because they are actually performed by closed subroutines in the basic G-15 language.

The "marked transfer" and "return transfer" instructions allow automatic entry to and return from subroutines coded in the *Intercom language*.

The index register instructions are standard except for the following:

<i>Instruction</i>	<i>Address</i>	<i>Explanation</i>
Decrement	$B_i, n$	$(B_i) - (D_i) \rightarrow B_i$
		if $(B_i) \geq (L_i), n \rightarrow IC$
		if $(B_i) < (L_i), (IC) + 1 \rightarrow IC$
Increment	$B_i, n$	$(B_i) + (D_i) \rightarrow B_i$
		if $(B_i) \leq (L_i), n \rightarrow IC$
		if $(B_i) > (L_i), (IC) + 1 \rightarrow IC$

The "replace" index register instruction allows the contents of any index register to be transferred into any other index register.

The output operations, in order to save computer time, allow information to be output in a standard three numbers per line format and to be "stacked" in an output stack, so that they can be typed out in a group. Typing of a fixed point number allows integer labeling of output.

The following operations are improper:

1. Division by zero.
2. Log of zero.
3. Log of negative number.
4. Square root of negative number.
5. Exponential of a number greater than  $128 \ln 2$ .

6. Any arithmetic operation resulting in a number whose absolute value is greater than  $2^{128}$ .

7. When an improper order is sensed by the machine, "xxxxxxx" is typed out, the bell rings continuously for about 12 seconds and computations are halted. The A register will contain a meaningless result. When the compute switch is reset to "Go," computations are resumed with the order that is next in sequence.

TABLE 39. INTERCOM INSTRUCTION LIST

## Arithmetic operations

Operations involving a memory position and the A register

- 49 Clear and subtract
- 4v Clear and add
- 4z Divide
- 59 Subtract
- 5v Add
- 5x Store
- 67 Multiply

Operations involving the A register only

- 07 Square root
- 11 Natural logarithm
- 13 Absolute value
- 16 Exponential
- 29 Negate

## Transfers of control operations

- 0v Transfer if A register is negative
- 10 Transfer if A register is non-negative
- 19 Unconditional transfer
- 42 Halt
- 43 Marked transfer (unconditional)
- 44 Return transfer (unconditional)

## Operations on index registers

- 3x Set B register
- 40 Set D register
- 41 Set L register
- 57 Decrement B register
- 65 Increment B register
- 1y Replace index register

## Output operations

- 1z Type and tab
- 20 Type and carriage return
- 21 Stack
- 22 Type the stack
- 23 Space
- 24 Type fixed point number
- 33 Ring bell

## Input operations

- 09 Read punched tape

### Comparisons

It is useful to compare the Intercom system with the other interpreter described, the EASIAC (see Sect. 6). The EASIAC, which simulated a three-address instruction logic on a three-address logic machine, also had a large number of index registers. Since the MIDAC, its "host" machine, had a large storage (6, 144 words of secondary storage), larger EASIAC programs were possible, but the EASIAC storage was also decreased over the original MIDAC storage.

Compared with the Intercom system, which uses almost 1300 locations, out of about 2200, the final stages of the IT compiler on the IBM 650 uses only about 400 out of 2000 locations to accomplish the same features. This difference is due to the noninterpretive nature of the IT compiler. The speed of EASIAC and Intercom, although they can perform address modifications rapidly and quite efficiently, is limited by the interpretation cycle which causes a drop in speed, with the need for floating point subroutines, that may be as much as an order of magnitude.

The following conclusions may be drawn from the discussion:

1. Interpreters may be very useful in simulating another computer on an original host machine.
2. Interpretation will slow down speed of operation markedly.
3. The use of a preliminary translator with closed subroutines will generally produce much more effective results (but note the exceptions below under Recursive Languages, Sect. 15).

## 15. AUTOMATIC PROGRAMMING: RECURSIVE LANGUAGES

### Recursive Use of Subroutines

One of the most powerful techniques that has been developed for using digital computers is that first attempted by Newell, Simon, and Shaw (Ref. 143). The basis of their technique is to free the programmer from all dependence upon machine characteristics, including storage allocation. They have made use of the concept of indirect addressing (described in Sect. 11) to build upon an "associative memory." The latter is a list-like structure that allows one or more "lists" (the counterpart of the usual programming "region") to be built up in storage—added to or deleted from, either at the ends or in the middle—without the need for the user's keeping track of any storage positions. This, in a sense, is an extension of the symbolic or floating address technique from *instruction* addresses to *data* addresses, and moreover it allows the usual program changes of deletion and insertion to be made *during* actual performance of the problem, rather than merely *before* its operation.

The first problem that this group of researchers attacked was that of instructing a machine to play chess. This was later changed to proof of the first listed theorems of the propositional calculus in the *Principia Mathematica* (Ref. 105). The chess player's performance, as has been shown by de Groot (Ref. 152), is a recursive one, in that he will attack a portion of the problem, move onward through a chain of successive subproblems, and then return to the original problem as a result of some decision made in the subproblem investigations, proceed once more to the subproblems, etc.

A similar performance was discovered in the behavior of human beings attempting to solve the theorems of the propositional calculus. The need is apparent, in such problems, to have available complete facility for construction of a completely flexible hierarchy of subroutines, so that any one may call on any other (even one above it in the chain) without loss of control of the process, and without introduction of any of the standard logical paradoxes.

To a certain extent this has been realized in the Holt and Turanski GP compiler (Ref. 149); but without indirect addressing, associative memory features of the IPL (Interpretive Programming Language) of Newell, Simon, and Shaw, use of the techniques on any machine is limited because of storage assignment and reassignment requirements that must be met during the course of performance. The IT, Unicode, and Fortran compiler languages (Sects. 11 and 12), which allow subscripting, are general enough to permit such a recursive subroutine description. As Schecher (Ref. 91) has shown, one way of handling subroutines in a recursive fashion is for each subroutine to supply all variables and return addresses to the subroutine one level in the hierarchy above or below it. A "level index" ( $j$  in Fig. 26, see Sect. 11) must be kept, which is increased at each time of subroutine entry and decreased at each time of subroutine exit.

Figure 26, Sect. 11, describes a problem involving three subroutines, entered at  $\alpha$ ,  $\gamma$ , and  $\epsilon$ , and controlled by a master routine. If each subroutine, as a closed type, is required to return to the level from which it was entered, and if enough storage space is provided so that the variables can be stored for every entry of a subroutine, then this process will work. Since in such a problem as that of Fig. 26 the knowledge of how many times a subroutine is to be performed is unavailable at the start of the problem, some device such as the associative memory described above must be used to guarantee that storage is used to the fullest. Similarly, the computer can be programmed to check the time that has occurred in any area of the problem so that it can stop that portion whenever time is up.

The problem in Fig. 26 is not an actual one, but is typical of the type of problems involved with this technique. A flow diagram, command compiler language, or machine code is necessary for description of such prob-

lems. Whether this problem ever terminates or not is dependent on the three functions  $f(a_i, b_i, c_i)$ ,  $g(a_i, b_i, c_i)$ , and  $h(a_i, b_i, c_i)$  and the values read in, and in the general case cannot be predicted.

## 16. LOGICAL PROGRAMMING

The standard uses for a digital computer are in computation of problems in numerical analysis. However, binary machines can be used for many nonarithmetical problems involving logical predicates. (See Chap. 11.) As an example of such problems the checker game first analyzed by Strachey (Ref. 15) and later improved by Samuel in a program for the IBM 704 makes use of machine words with binary zeros and ones representing the absence or presence of checker men on the board. This technique has been used by Ulam and Kister (Ref. 114) also, apparently, in their first try at a solution of a game of chess.

Below is given a simple problem, that of evaluation of truth tables in the propositional calculus, along with an explanation of the instructions available on four binary computers (Univac Scientific 1103A, IBM 701 and 704, and MIDAC).

**Problem in the Propositional Calculus.** The following statement is drawn from Copi (Ref. 115, p. 52, problem 24).

"If old Henning wants to retire, then he will either turn the presidency over to his son or sell the business. If old Henning needs money then he will either sell the business or borrow additional capital. Old Henning will never sell the business. Therefore if he neither turns the presidency over to his son nor borrows additional capital, then he neither wants to retire nor needs money."

With the notation  $R, T, S, N, B$ , and the symbols  $\supset$  (*material implication*),  $+$  (*or*),  $\cdot$  (*and*), and  $\bar{\phantom{x}}$  (*negation*), one has

$$[R \supset (T + S)] \cdot [N \supset (S + B)] \cdot [\bar{S}] \supset [(\bar{T} \cdot \bar{B}) \supset (\bar{R} \cdot \bar{N})]$$

One would like to know whether the statement is true or false. The problem can easily be programmed to be solved on the computer. The truth tables for all possible combinations of  $R, T, S, N, B$  are stored by columns (5 columns of 32 bits each) in a binary machine. Starting at the innermost parenthesis, one may perform operations on columns in pairs and the results stored. Working from the inner parenthesis out by such two-valued functional procedures, one finally obtains the truth value of the statement for all possible combinations.

If the computer operations do not permit *or* and *material implication*, for example, they can be replaced as follows:

$$A + B \equiv \overline{\bar{A} \cdot \bar{B}}$$

$$A \supset B \equiv \bar{A} + B \equiv \overline{\bar{A} \cdot \bar{B}} \equiv \overline{A \cdot \bar{B}}$$

**Logical Programming for the Univac 1103A**

The (u) will be represented by  $U_{j,n}$ , where  $j$  will indicate the digit ( $j = 0, \dots, 35$ ) and  $n$ , the time of operation of the instruction, with a similar notation for  $v$ .

$V_{j,n} \equiv 1$  will indicate that all digits ( $j = 0, \dots, 35$ ) of  $V$  contain a binary one.

$U_{j,n} \equiv 0$  will indicate that all digits ( $j = 0, \dots, 35$ ) of  $U$  contain a binary zero.

Five Univac 1103A instructions are particularly pertinent for use with the basic two-valued logical functions:

**27. Controlled Complement (CC).** Replace  $A_R$  with (u) leaving  $A_L$  undisturbed. Then complement those bits of ( $A_R$ ) that correspond to ones in (v). Then replace (u) with  $A_R$ .

*Truth Table*

$U_{j,n}$	$V_{j,n}$	$U_{j,n+1}$
0	0	0
0	1	1
1	0	1
1	1	0

Result:

$$U_{j,n+1} \equiv (U_{jn} \neq V_{jn}) \equiv \bar{U}_{jn} \cdot V_{jn} + U_{jn} \cdot \bar{V}_{jn} \quad (j = 0, \dots, 35),$$

$$A_{Rj,n+1} \equiv (U_{jn} \neq V_{jn}) \quad (j = 36, \dots, 71),$$

$$A_{Lj,n+1} \equiv A_{Lj,n} \quad (j = 0, \dots, 35).$$

**51. Q-Controlled Transmit (QT).** Form in A the number  $L(Q)(u)$ . Then replace (v) by ( $A_R$ ). ( $L(Q)(u)$  has leftmost bits zero and righthand bits given by individual bit product.)

$$A_{Lj,n+1} \equiv 0 \quad (j = 0, \dots, 35),$$

$$A_{Rj,n+1} \equiv U_{jn} \cdot Q_{jn} \quad (j = 36, \dots, 71),$$

$$V_{j,n+1} \equiv U_{jn} \cdot Q_{jn} \quad (j = 0, \dots, 35).$$

**53. Q-Controlled Substitute (QS).** Form in A the quantity  $(Q)(u)$  plus  $L(Q)'(v)$ . Then replace (v) with ( $A_R$ ). The effect is to replace selected bits of (v) with the corresponding bits of (u) in those places corresponding to 1's in Q.

$$A_{Lj,n+1} \equiv 0,$$

$$A_{Rj,n+1} \equiv V_{jn} \cdot \bar{Q}_{jn} + U_{jn} \cdot Q_{jn} \quad (j = 36, \dots, 71),$$

$$V_{j,n+1} \equiv V_{jn} \cdot \bar{Q}_{jn} + U_{jn} \cdot Q_{jn} \quad (j = 0, \dots, 35).$$

*Special Case I*

If  $U = 1$ ,

$$\begin{aligned}
 A_{R_{j,n+1}} &\equiv (V_{jn} \cdot \bar{Q}_{jn}) + (1 \cdot Q_{jn}) & (j = 36, \dots, 71) \\
 &\equiv V_{jn} \cdot \bar{Q}_{jn} + Q_{jn} \\
 &\equiv Q_{jn} + V_{jn} \cdot \bar{Q}_{jn} \\
 &\equiv (Q_{jn} + V_{jn}) \cdot (Q_{jn} + \bar{Q}_{jn}) & \text{(Distributive law)} \\
 &\equiv (Q_{jn} + V_{jn}) \cdot 1 \\
 &\equiv Q_{jn} + V_{jn}.
 \end{aligned}$$

Therefore if  $U = 1$  (or  $= Q$ ), the  $Q$ -controlled substitute yields the logical *or*.

*Special Case II*

If  $V = 1$ ,

$$\begin{aligned}
 A_{R_{j,n+1}} &\equiv 1 \cdot \bar{Q}_{jn} + U_{jn} \cdot Q_{jn} \\
 &\equiv \bar{Q}_{jn} + U_{jn} \cdot Q_{jn} \\
 &\equiv (\bar{Q}_{jn} + U_{jn}) \cdot (\bar{Q}_{jn} + Q_{jn}) \\
 &\equiv (\bar{Q}_{jn} + U_{jn}) \cdot 1 \\
 &\equiv \bar{Q}_{jn} + U_{jn} \\
 &\equiv Q_{jn} \supset U_{jn} \text{ by definition } (j = 0, \dots, 35).
 \end{aligned}$$

Therefore if  $V \equiv 1$  (or  $\bar{Q}$ ), then  $Q$ -Controlled Substitute yields the logical *material implication*.

**11. Transmit Positive (TP).** Replace (v) with (u)

$$V_{j,n+1} \equiv U_{jn} \quad (j = 0, \dots, 35).$$

**12. Transmit Negative (TN).** Replace (v) with the complement of (u).

$$V_{j,n+1} \equiv \bar{U}_{jn} \quad (j = 0, \dots, 35).$$

If one therefore performs (in some cases) proper preliminary storage of 0 and 1, one can express all the logical operations  $\supset$ ,  $\cdot$ ,  $+$ ,  $\bar{\phantom{x}}$ ,  $\neq$  immediately with at most two instructions.

**Logical Programming for the IBM 704**

Unlike the Univac 1103A, the IBM 704 has one instruction for each of the common logical operations *and*, *or*, and *not*. In the description below digits P of the accumulator and S of the storage location are labeled digit zero (0). Digits S and Q of the accumulator are labeled  $-2$  and  $-1$  respectively.

**AND to Accumulator ANA Y.** Each bit of the  $C(AC)_{P,1-35}$  is matched with the corresponding bit of the  $C(Y)_{S,1-35}$ , the  $C(AC)_P$  being matched with the  $C(Y)_S$ . When the corresponding bit of both the AC and location Y is a one, a one replaces the contents of that position in the AC. When the corresponding bit of either the AC or location Y is a zero, a zero replaces the contents of that position in the AC. The  $C(AC)_{S,Q}$  are cleared. The  $C(Y)$  are unchanged.

$$A_{j,n+1} \equiv A_{jn} \cdot Y_{jn} \quad (j = 0, \dots, 35),$$

$$A_{j,n+1} \equiv 0 \quad (j = -2, -1),$$

$$Y_{j,n+1} \equiv Y_{jn} \quad (j = 0, \dots, 35).$$

**AND to Storage ANS Y.** Each bit of the  $C(AC)_{P,1-35}$  is matched with the corresponding bit of the  $C(Y)_{S,1-35}$ , the  $C(AC)_P$  being matched with the  $C(Y)_S$ . When the corresponding bit of both the AC and location Y is a one, a one replaces the contents of that position in location Y. When the corresponding bit of either the AC or location Y is a zero, a zero replaces the contents of that position in location Y. The  $C(AC)$  are unchanged.

$$A_{j,n+1} \equiv A_{jn} \quad (j = 2, -1, 0, \dots, 35),$$

$$Y_{j,n+1} \equiv A_{jn} \cdot Y_{jn} \quad (j = 0, \dots, 35).$$

**OR to Accumulator ORA Y.** Each bit of the  $C(AC)_{P,1-35}$  is matched with the corresponding bit of the  $C(Y)_{S,1-35}$ , the  $C(AC)_P$  being matched with the  $C(Y)_S$ . When the corresponding bit of either the AC or location Y is a one, a one replaces the contents of that position in the AC. When the corresponding bit of both the AC and location Y is a zero, a zero replaces the contents of that position in the AC. The  $C(Y)$  and the  $C(AC)_{S,Q}$  are unchanged.

$$A_{j,n+1} \equiv A_{jn} + Y_{jn} \quad (j = 0, \dots, 35),$$

$$A_{j,n+1} \equiv A_{jn} \quad (j = -2, -1),$$

$$Y_{j,n+1} \equiv Y_{jn} \quad (j = 0, \dots, 35).$$

**OR to Storage ORS Y.** Each bit of the  $C(AC)_{P,1-35}$  is matched with the corresponding bit of the  $C(Y)_{S,1-35}$ , the  $C(AC)_P$  being matched with the  $C(Y)_S$ . When the corresponding bit of either the accumulator or location Y is a one, a one replaces the contents of that position in location Y; when the corresponding bit of both the AC and location Y is a zero, a zero replaces the contents of that position in location Y. The  $C(AC)$  are unchanged.

$$A_{j,n+1} \equiv A_{jn} \quad (j = -2, -1, 0, \dots, 35),$$

$$Y_{j,n+1} \equiv A_{jn} + Y_{jn} \quad (j = 0, \dots, 35).$$

**Complement Magnitude COM.** All ones are replaced by zeros and all zeros are replaced by ones in the  $C(AC)_{Q,P,1-35}$ . The AC sign is unchanged.

$$A_{j,n+1} \equiv \bar{A}_{j,n} \quad (j = -1, 0, \dots, 35),$$

$$A_{j,n+1} \equiv A_{jn} \quad (j = -2).$$

### Logical Programming for the IBM 701

The IBM 701 has easily available only three logical operations particularly pertinent for use with the basic two-valued logical functions.

#### Extract Y EXTR.

$$A_{j,n+1} \equiv A_{jn} \quad (j = 0, \dots, 35),$$

$$Y_{j,n+1} \equiv Y_{jn} \cdot A_{jn} \quad (j = 0, \dots, 35).$$

#### Add Y ADD.

#### Subtract Y SUB.

If position Y contains the binary number  $-.111 \dots 1$  (all ones), then if the accumulator is positive or positive zero, addition of (Y) will give the negation of ACC in all 36 positions. If the accumulator is negative or negative zero, then subtraction of (Y) will give the negation of ACC in all 36 positions.

If  $A_0 \equiv 0$ , then  $A_{j,n+1} \equiv \bar{A}_{jn}$  after ADD Y.

If  $A_0 \equiv 1$ , then  $A_{j,n+1} \equiv \bar{A}_{jn}$  after SUB Y.

### Logical Programming on the MIDAC

On the MIDAC, a 44-bit computer built at the University of Michigan as a modification of the National Bureau of Standards SEAC design, there exists a special *three-address* instruction that combines several of the logical operations in one machine order. The "extract" operation is defined as follows:

**ex     $\alpha$      $\beta$      $\gamma$ .** Whenever  $\beta$  has a digit "one," replace the corresponding digit of  $\gamma$  by the digit of  $\alpha$ ; otherwise, leave  $\gamma$  unchanged. The truth tables for this instruction could be constructed as follows.

$\alpha$	$\beta$	$\gamma_{n-1}$	$\gamma_n$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Forming the Boolean equivalent of the table, one gets

$$\begin{aligned}\gamma_n &\equiv \bar{\alpha} \cdot \bar{\beta} \cdot \gamma_{n-1} + \alpha \cdot \bar{\beta} \cdot \gamma_{n-1} + \alpha \cdot \beta \cdot \bar{\gamma}_{n-1} + \alpha \cdot \beta \cdot \gamma_{n-1} \\ &\equiv (\bar{\alpha} + \alpha) \cdot \bar{\beta} \cdot \gamma_{n-1} + \alpha \cdot \alpha \cdot (\bar{\gamma}_{n-1} + \gamma_{n-1}) \\ &\equiv \bar{\beta} \cdot \gamma_{n-1} + \alpha \cdot \beta.\end{aligned}$$

Now, if the 44-bit word (ignoring sign)  $\alpha$  is given the value of all "one's,"

$$\begin{aligned}\gamma_n &\equiv \bar{\beta} \cdot \gamma_{n-1} + 1 \cdot \beta \\ &\equiv \bar{\beta} \cdot \gamma_{n-1} + \beta \\ &\equiv \beta + \gamma_{n-1}.\end{aligned}$$

On the other hand, if  $\gamma_{n-1} \equiv 1$ , then

$$\gamma_n \equiv \bar{\beta} \cdot 1 + \alpha \cdot \beta \equiv \bar{\beta} + \alpha \cdot \beta \equiv \bar{\beta} + \alpha \equiv \beta \supset \alpha.$$

Certainly, if  $\gamma_{n-1} \equiv 0$ , then

$$\gamma_n \equiv \alpha \cdot \beta.$$

Finally, if  $\gamma_{n-1} \equiv 1$  and  $\alpha \equiv 0$ , then

$$\gamma_n \equiv \bar{\beta} \cdot 1 + 0 \cdot \beta \equiv \bar{\beta}.$$

Thus one instruction contains *and*, *or*, *not*, and *material implication*.

## 17. MICROPROGRAMMING

Many programmers have thought that the technique of programming should allow greater internal control by the programmer of the detailed behavior of the computer. This direction of procedure would move the programmer in the direction of the logical designer, rather than in the direction of the numerical analyst and mathematician as the use of translators and compilers had been carrying him. At a meeting of computer designers and programmers at the Massachusetts Institute of Technology, Cambridge, Massachusetts, in 1956 (proceedings not published), it was agreed to call this technique, still not clearly defined, *microprogramming*. Several formal definitions of this technique were proposed, two of which are listed.

1. Wheeler of Cambridge University described the technique used in the construction of the EDSAC II there as a useful example for programmers. This machine's designers had used a magnetic core storage matrix of large size as the heart of the control element. By passing leads through various cores in sequence, the standard instruction code of the computer was being developed.

2. Others proposed an extension of the Cambridge technique to allow continuous control by the programmer of the computer's instruction code or at least a subset of it. This would be accomplished by transferring

certain words (equivalent in meaning to the wiring sequence used with the EDSAC II) to specified storage locations in the control unit of the computer. These would, in essence, then change the actions and time sequence of a given instruction. By this technique a computer could theoretically be changed quickly from single to double precision arithmetic or to floating point. The input-output instructions could be changed to suit the problem at hand.

### The TX-0 Development

Hardware development on microprogramming has been done by a group including Clark, Farley, Gilmore, Peterson, and Frankovich (Ref. 39) in connection with the instruction logic for the TX-0, an experimental computer designed at the Lincoln Laboratory of the Massachusetts Institute of Technology. This computer was constructed to test a large  $256 \times 256$  (65,536-word) magnetic core storage system. The logical design, therefore, was simple in that it had only four operations in two bits of the standard 18-bit word. The remaining 16 bits of each word were used to address any one of the  $2^{16}$  storage positions.

The computer, in addition to the standard program counter and instruction register, contains intervention registers, as well as four internal registers which can be used in the arithmetic process:

1. Memory buffer register (MBR) with 18 bits plus one parity check bit.
2. Accumulator (AC) with 18 bits, used to store results of numerical operations as well as an input-output buffer. It is constructed as a one's complement ring adder modulo  $2^{18}$ , with no overflow alarm.
3. Memory address register (MAR) with 16 bits, which selects information in storage and "operate class commands." (See below.)
4. Live register (LR) with 18 bits, a high-speed flip-flop rapid access register.

Three of the four instructions are similar to those used in any single-address instruction code:

**Sto X.** Replace the contents of register X with the contents of AC. Leave AC unchanged.

**Add X.** Add the word in register X to the contents of the AC and leave the sum in AC.

**Trn X.** If the sign digit of the accumulator ( $AC_0$ ) is negative (i.e., a one) take the next instruction from register X. If the sign is positive (i.e., a zero) proceed as usual.

The one unusual instruction of this computer which may be considered as the first announced true microprogramming instruction is the following:

**Opr X.** Execute one of the operate class commands indicated by the number X.

The operate class commands all have the same binary combination in the two-bit instruction position, but the remaining 16 bits selected 21 possible microoperations dependent upon the bit combination. Each microoperation has a specific time pulse during control on which it was scheduled to operate. Thus a given sequence of address bits in any "opr" instruction gives a unique sequence of microoperations.

The basic microoperations, in time pulse sequence, with the octal address corresponding, include:

*At pulse time 0.8:*

**CLL.** Clear the left nine digits of AC—100,000 (octal).

**CLR.** Clear the right nine digits of AC—40,000 (octal).

**IOS.** In-Out Stop. Stop machine so that an in-out command (specified by bits 6, 7, 8 of the MAR) may be executed—20,000 (octal).

**P7H.** Punch holes 1–6 in punched tape from AC positions 2, 5, 8, 11, 14, 17, with an added seventh hole—7000 (octal).

**P6H.** Same as previous, no seventh hole—6000 (octal).

**PNT.** Print on typewriter one six-digit character from positions 2, 5, 8, 11, 14, 17—4000 (octal).

**RIC.** Read one line of punched tape into AC positions, 0, 3, 6, 9, 12, 15—1000 (octal).

**R3C.** Read three characters into 2, 5, 8, 11, 14, 17; 1, 4, 7, 10, 13, 16; and 0, 3, 6, 9, 12, 15 of AC—3000 (octal).

**DIS.** Intensify a point on oscilloscope output with one's complement  $x$  coordinate given by AC digits 0–8 and one's complement  $y$  coordinate given by AC digits 9–17—2000 (octal).

*At pulse time 1.1:*

**PEN.** Read light pen flip-flops (set by viewing the intensity of the output oscilloscope at a point by a light pen including a phototransistor as its penpoint) into  $AC_0$  and  $AC_1$ —100 (octal).

**TAC.** Insert a one in each digital position of the AC wherever there is a one in the corresponding position of the TAC (18-toggle switch accumulator)—4 (octal).

*At pulse time 1.2:*

**COM.** Complement every digit in AC—40 (octal).

**AMB.** Store contents of AC in MBR—1 (octal).

**TBR.** Store contents of TBR in MBR—3 (octal).

*At pulse time 1.3:*

**MLR.** Store the contents of MBR in LR—200 (octal).

**LMB.** Store the contents of the LR in MBR—2 (octal).

*At pulse time 1.4:*

**PAD.** Partial add AC to MBR, that is, for  $AC_{\text{after}} \equiv \{(AC_{\text{before}} \neq MBR) - 20$  (octal) $\}$ .

**SHR.** Shift AC right one place (multiply by  $2^{-1}$ )—400 (octal).

**CYR.** Cycle AC right one position *modulo* 18 ( $AC_n \rightarrow AC_{n+1 \text{ modulo } 18}$ )  
—18 ( $AC_n \rightarrow AC_{n+1 \text{ modulo } 18}$ )—600 (octal).

*At pulse time 1.7:*

**CRY.** Carry partial add (see above) the 18 digits of the AC to the corresponding 18 digits of the carry (shifted left one place modulo 18, with an end-around carry included as usual with the one's complement addition)—10 (octal).

**Macroinstructions from Microinstructions.** Many macroinstructions can be constructed from properly sequenced microinstructions. For example, an instruction CYL (cycle left) can be represented as

$$\text{CYL} \equiv \text{opr } 31$$

since opr 31 combines

$$\text{AMB (1.2), PAD (1.4), and CRY (1.7)}$$

in that order. Cycle left will perform a sequence analogous to CYR listed above. The address 31 (octal) is obtained by the logical disjunction of the bits in the three octal addresses 1, 20, and 10.

Among the list of combinations which were found to be useful in forming macrooperations were many useful ones available on more standard computers plus others that ordinarily would not have been in the hardware. A translation program was written to translate these operations, considered as ordinary instructions, into sequences of operate instructions intermixed with the three standard instructions. Among the list of instructions found to be useful, which continued to grow with the use of the computer, were the following:

0.8    0.8  
CLL + CLR = opr 140,000 = Clear the AC (CLA).

1.2    1.4    1.7  
AMB + PAD + CRY = opr 31 = Cycle the AC left one digital position (CYL).

0.8    0.8    1.2  
CLL + CLR + COM = opr 140,040 = Clear and complement AC (CLC).

0.8    0.8  
IOS + DIS = opr 22,000 = Display (this combination was included as a reminder that with every in-out command the IOS must be included) (DIS).

0.8    0.8    0.8  
IOS + CLL + CLR = opr 160,000 = In out stop with AC cleared.

0.8    0.8    1.4  
IOS + P7H + CYR = opr 27,600 = Punch 7 holes and cycle AC right.

0.8    0.8    1.4  
IOS + P6H + CYR = opr 26,600 = Punch 6 holes and cycle AC right.

0.8    0.8    0.8    0.8  
 IOS + CLL + CLR + P6H = opr 166,000 = Clear the AC and punch a blank space on tape.

0.8    0.8    0.8  
 IOS + PNT + CYR = opr 24,600 = Print and cycle AC right.

0.8    0.8    1.2    1.4  
 IOS + P7H + AMB + PAD = opr 27,021 = Punch 7 holes and leave AC cleared.

0.8    0.8    1.2    1.4  
 IOS + P6H + AMB + PAD = opr 26,021 = Punch 6 holes and leave AC cleared.

0.8    0.8    1.2    1.4  
 IOS + PNT + AMB + PAD = opr 24,021 = Print and leave AC cleared.

0.8    0.8    0.8  
 CLL + CLR + R1C = opr 141,000 = Clear AC and start photoelectric tape reader running (notice no IOS, which means computer has not stopped to wait for information).

0.8    1.2    1.4    1.7  
 R1C + AMB + PAD + CRY = opr 1,031 = Start petr running and cycle AC left.

0.8    0.4  
 R1C + CYR = opr 1600 = Start petr running and cycle right.

0.8    0.8    0.8    0.8  
 CLL + CLR + IOS + R3C = opr 163,000 = Clear AC, read 3 lines of tape.

0.8    0.8    0.8    0.8  
 CLL + CLR + IOS + R1C = opr 161,000 = Clear AC and read one line of tape.

0.8    0.8    0.8    0.8    1.4    1.7  
 CLR + CLR + IOS + R1C + PAD + CRY = opr 161,031 = Read 1 line of tape and cycle AC left.

0.8    0.8    0.8    0.8    1.4  
 CLL + CLR + IOS + R1C + CYR = opr 161,600 = Read one line of tape and cycle right.

0.8    0.8    1.1  
 CLL + CLR + TAC = opr 140,004 = Put contents of TAC in AC.

1.4    1.7  
 PAD + CRY = opr 30 = Full-add the MBR and AC and leave sum in AC.

0.8    0.8    1.3    1.4  
 CLL + CLR + LMB + PAD = opr 140,022 = Clear the AC, store LR contents in memory buffer register, add memory buffer to AC, i.e., store live register contents in AC (LAC).

1.2    1.3  
 AMB + MLR = opr 201 = Store contents of AC in MBR, store contents of MBR in LR, i.e., store contents of AC in LR (ALR).

1.3    1.4  
 LMB + PAD = opr 22 = Store contents of LR in MBR, partial add AC and MBR, i.e., partial add LR to AC (LPD).

1.3

MLR = opr 200 = Since MLR alone will have a clear MBR, this is really clear LR (LRO).

1.3      1.4      1.7

LMB + PAD + CRY = opr 32 = Full-add the LR to the AC (LAD).

0.8      0.8      1.3      1.4

CLL + CLR + TBR + PAD = opr 140,023 = Store contents of TBR in AC.

**An Example of a Program.** As an example of a program (coded with octal addresses) and using the microprogramming technique, an input program that reads in a following program in standard binary form, three 6-bit paper tape characters, making up an 18-bit word, is shown in Table 40. This routine is automatically punched out by the translation program at the head of the binary output tape.

The basic read in mode of the TX-0 computer allows read in of this standard input routine by merely pushing the machine's Read In button, which automatically reads the input routine in and automatically transfers control to octal location 177744. Words of the succeeding binary program are punched on the tape in a block whose first word is a store instruction ( $stoW_1$ ) which contains the address  $W_1$  into which the block is to be stored. The second word is the complement of a store instruction ( $stoW_n$ ), where  $W_n$  is the address of the  $n$ th (last) word in the block. Following this comes a sequence of binary words (the program), terminated by a word (sum check) containing the complement of the sum of all the preceding words in the block including the first two control words.

The starting instruction for this block is given in the address of an instruction following the last block of words in a program to be input. If that instruction is *add Z* ( $Z$  the starting address), the computer stops *before* transferring control. If the instruction is *trn Z* then the transfer of control is immediate.

The three microprogrammed instructions used in the input routine are:

Opr 160,000 — CLL + CLR + IOS + R3C. Clear AC and read three lines on the paper tape, cycling each time so that they are assembled as an 18-bit word in AC.

Opr 140,000 — CLL + CLR (CLA). Clear both halves of the accumulator.

Opr 30,000—Halt.

TABLE 40. INPUT ROUTINE FOR THE TX-0 COMPUTER

	177741	Temporary storage
	177742	add 177773
	177743	trn 177772
Enter →	177744	opr 163,000 (R3C)
	177745	sto 177756
	177746	trn 177756
	177747	add 177774
	177750	trn 177775
	177751	opr 163,000 (R3C)
	177752	sto 177777
	177753	add 177756
	177754	sto 177741
→	177755	opr 163,000 (R3C)
	177756	(sto $W_i$ )
		(add $Z$ ) (Not used)
		(trn $Z$ ) ←
	177757	add 177741
	177760	sto 177741
	177761	opr 140,000 (CLA)
	177762	add 177756
	177763	add 177773
	177764	sto 177756
	177765	add 177777
	177766	trn 177755
	177767	opr 163,000 (R3C)
	177770	add 177741
	177771	trn 177742
	177772	opr 30,000 (HLT) ←
	177773	1
	177774	200,000
	177775	sto 177777 ←
	177776	opr 30,000 (HLT)
	177777	trn $Z$ (or the comple- ment of the address of the last word in a block).

Partial sum of block  
If the preceding block's sum is correct, go on to next block or transfer control word. If not, go to 177772 and stop computer. Read in the first word of a block or the transfer control word (add  $Z$  or trn  $Z$ ) and store it in register 177756.

Is it st  $W_1$ , add  $Z$ , or trn  $Z$ ? If trn  $Z$  go directly to register 177756.

It is either st  $W_1$  or add  $Z$ ; add 200,000 to the AC. If it was add  $Z$ , the AC is now neg. (= trn  $Z$ ), so go to 177775.

Read in the complement of the address of the last word in the block and store it in the register 177777.

Add the first two control words of the block together and store in 177741 to initiate the partial sum.

Read in the  $i$ th word and store it in its assigned memory location.

Add the  $i$ th word to the partial sum of the block.

Index the address section of the register 177756 by one.

Has the  $n$ th word been transferred to storage? If AC is negative-no, return to 177755. Read in the sum of the block. Is it the same as the sum in register 41? If it is, AC = minus zero, go to 177742. If it is positive... stop the computer. The sum check is wrong. Constants

The last block has been stored and the transfer control word was add  $Z$ . Put trn  $Z$  in register 177777 and stop the computer.

Upon restarting, transfer control to register  $Z$ .

## 18. PROGRAMS FOR MAINTENANCE OF EQUIPMENT

A very useful type of programming is used to test a computer, either in the construction and design stages or later during operation. This type of program has been discussed by Graney (Ref. 44) and Brock and Rock (Ref. 16).

Such maintenance programs must be used periodically with or without marginal testing (see Chap. 13). Acceptance tests are those evolved for testing a computer in its earliest stages, before it has ever been used in productive operation.

Among the criteria that have been stressed for programs to do these two jobs are the following.

1. Severe operating conditions for the computer must be produced.
2. A sufficiently large set of random (actually pseudo random) numbers must be operated on and the results verified, so that a satisfactory sampling of machine conditions will have been tested; or, alternatively,
3. Elements of the computer must assume all possible states and be checked for correctness.

Regular maintenance programs are usually bootstrap programs stored originally in a small, pretested area of storage (for example, on Whirlwind I, in a given bank of hand switches for storage) that call in the next portion of the program which tests a wider area of machine structure before proceeding. Usually the simplest arithmetic, control, and storage operations will be first verified. After this a storage summing operation will be used to check a larger portion of storage. Automatic procedures are usually available that will print out a notation of a failing storage position or a failing machine operation.

After the main storage unit has been satisfactorily checked, then the more elaborate instructions may be verified, including secondary storage (drums and magnetic tapes).

Input-output equipment using punched paper tape or punched cards cannot be verified automatically, and with its mechanical components is a major source of failures. Generally, information is printed out and then read back in after having been physically handled by human beings.

The presence of automatic checking devices such as parity checks, forbidden character checks, and duplicate arithmetic checks, and casting out nines (see Chap. 13), *built into the machine*, can be useful, but they also cause delays in the automatic performance of such checking, since upon finding its own errors the computer usually stops. This requires human intervention and the difficult task of finding the actual error. Automatic recovery, by which upon discovering an error, a computer notes it, corrects it, and then proceeds, has been used mainly on magnetic tape han-

dling equipment. While maintenance testing programs built to use such features are more difficult to program, they improve efficiency of the equipment.

Random numbers for such programs may be generated by the techniques discussed by Moshman (Ref. 74) for decimal number systems.

## 19. PROGRAMMING WITH NATURAL LANGUAGE

**Language Commands.** The possibility of programming digital computers to respond to simple English commands has already been realized on a low level by a group working under Hopper (Ref. 98) which produced the Flow-matic compiler for the Univac I and II. Important advances in this field have come from linguistic studies, either in the field of mechanical translation (see Ref. 120 and Chap. 11), or in linguistic structure of natural languages. The results quite surprisingly correspond to the analyses described for the Fortran and IT compilers (Ref. 144). The most thorough such analysis has been described in a series of papers by Chomsky (Refs. 26-28). Chomsky follows the pattern first laid out by Post (Ref. 84) and described in Rosenbloom (Ref. 88).

**Language Decomposition.** Appendix II of Ref. 26 lists a set of eleven basic phrase structure rules for a set of productions for decomposing English sentences into constituent parts. These must in certain cases be preceded in a certain order by a set of fifteen more transformations, involving such relationships as the passive voice, interrogative mood, conjunctions, gerunds, and participles. To these morphemic rules would be adjoined a set of phonomorphic rules relating to the changes in word structure caused by phonetic rules of performance of English-speaking persons.

A sentence decomposition in the earliest stages of this analysis structure is similar to the tree structure developed for the simpler algebraic-programming languages already described (see Sect. 12). For example, the sentence "The man hit the ball," when decomposed into constituents such as noun phrase (NP), verb phrase (VP), noun (N), verb (V), and article (T), decomposes into the tree of Fig. 41.

**Storage Requirements.** This is not much more involved than the standard parsing of high school grammar. However, the overall set of rules developed by Chomsky allows complete explanation and decomposition of such ambiguous sentences as "I found the boy studying in the library," and such phrases as "the shooting of the hunters." Based on Chomsky's analysis, or one carried even further, a complete program can be written to give complete structural decomposition of the English language upon entry into a computer. The semantic or meaning portion of the machine language problem is dependent on further theoretical investi-

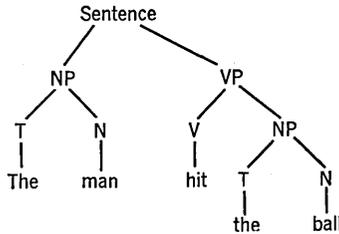


FIG. 41. Tree structure resulting from decomposition of the sentence, "The man hit the ball."

gations as well as further experiments with large-scale storage devices such as the RAMAC (see Ref. 118) and Tapefile (Ref. 119). The problem of pattern recognition to recognize structures such as that of Fig. 41 has been studied for simpler problems by Dineen (Ref. 116), and Minsky (Ref. 117). The construction of automatic dictionaries has already been begun with such algebraic languages as Fortran, Unicode, and GP at one end, and with the language translation experiments at the other. By the time successful translation of scientific documents will have been accomplished, a programmed knowledge of the structure of English and other natural languages will have had to be acquired. The pattern recognition procedures and compiling will fill the gap between such decomposition and actual machine instructions which are to correspond to the original English input.

The use of any large magnetic tape storage should be sufficient for large portions of a natural language (for example, scientific vocabularies combined with a basic vocabulary). The need for speed, however, determines the requirement for the more rapid large-scale storage devices noted above.

## LITERATURE, ACKNOWLEDGMENTS, AND REFERENCES

### LITERATURE

**Books.** For the elementary approach to programming, the texts by McCracken (Ref. 68) and Gottlieb and Hume (Ref. 113) may be consulted. For a detailed discussion of programming for a specific machine, described in detail, Wilkes, Wheeler, and Gill (Refs. 108 and 109) is a thorough documentation of the work on EDSAC. A compilation of a series of articles in *Control Engineering* (Ref. 99) provides many practical examples not found elsewhere. In the area of mechanical translation the book edited by Locke and Booth (Ref. 120) gives an account pertinent to many programming problems. The University of Michigan Summer Session Notes edited by Carr and Scott (Refs. 24, 25) contain

practical accounts of recent procedures. Books on the general structure of digital computers, such as those by Bowden (Ref. 15), Booth and Booth (Ref. 121), Richards (Ref. 122), Wilkes (Ref. 124), Eckert and Jones (Ref. 111), the ERA Staff (Ref. 125), Stibitz and Larrivee (Ref. 96), and Berkeley and Wainwright (Ref. 126) usually include one or more chapters on elementary programming.

From the point of view of business problems, such books as Kozmetsky and Kircher (Ref. 127), Canning (Ref. 128), Chapin (Ref. 129), Berkeley (Ref. 130), and a small pamphlet by Gorn and Manheimer (Ref. 131) give simplified accounts tailored to the nontechnical reader. Practical examples in this area are available in publications of the Joint Computer Conferences of the American Institute of Electrical Engineers, Institute of Radio Engineers, and Association for Computing Machinery (Ref. 136), and those of the American Management Association (Ref. 132), and the Office Management Association of Great Britain (Ref. 134).

Perhaps the most thorough account of digital computer programming, containing detailed examples and analyses of both hand processes and automatic programming, is in the Russian language book by Kitov (Ref. 61). Portions of this have been translated into English and may become available through the Association for Computing Machinery. In the area of automatic programming the Proceedings of Office of Naval Research Symposia (Ref. 98) are important. The only book available in the area of artificial intelligence as it might be applied to general purpose computer programming is that of Shannon and McCarthy (Ref. 94).

In addition to the books, numerous publications are available from the manufacturers. They range from simple descriptions of particular computers to automatic programming schemes for a particular computer. A particularly thorough bibliography is that of Remington Rand Univac (Ref. 137), which includes listings from outside that organization as well as inside.

**Journals.** These might be catalogued as follows:

1. Theory of programming and examples of general programming techniques:

*Journal of the Association for Computing Machinery*, published by the Association, 2 East 63 Street, New York 23, New York.

*Communications of the Association for Computing Machinery*, published by the Association, 2 East 63 Street, New York 23, New York.

*Journal of Research and Development of the International Business Machines Corporation*, published by International Business Machines Corporation, 590 Madison Avenue, New York.

*Transactions of the Professional Group on Electronic Computers (I.R.E.)*, published by the PGEC of I.R.E., 1 East 79 Street, New York 21, New York.

*Systems* (Remington Rand Univac), published by Remington Rand Univac, Philadelphia, Pennsylvania.

*Journal of the Franklin Institute*, published by the Franklin Institute, Philadelphia, Pennsylvania.

*The Computer Journal*, published by the British Computer Society, London, England.

2. Examples of problem formulation and programming in the areas of scientific computation:

*Journal of the Society for Industrial and Applied Mathematics*, published by the Society, Box 7541, Philadelphia, Pennsylvania.

*Journal of the Association for Computing Machinery*, published by the Association, 2 East 63 Street, New York 23, New York.

*Control Engineering*, published by McGraw-Hill, 330 West 42nd Street, New York 36, New York.

*Mathematical Tables and Other Aids to Computation*, published by the National Academy of Sciences, National Research Council, Washington 25, D. C.

*Computing News*, published by Jackson Granholm, 12805 Sixty-fourth Avenue, South, Seattle 88, Washington.

*The Computer Journal*, published by the British Computer Society, London, England.

3. Examples of problem formulation in the areas of business and industrial applications:

*Journal of the Operations Research Society of America*, published by the Operations Research Society of America at Mount Royal and Guilford Avenues, Baltimore 2, Maryland.

4. Applications to business problems:

*Data Processing Digest*, published by Canning, Sisson, and Associates, 914 South Robertson Boulevard, Los Angeles 35, California.

5. Popular discussions of programming and applications:

*Computers and Automation*, published monthly by Berkeley Enterprises, Inc., 815 Washington Street, Newtonville 60, Massachusetts.

6. Foreign journals:

*Zeitschrift für angewandte Mathematik und Physik*

*Doklady, Akad. Nauk S.S.S.R.* (*Proceedings of the Academy of Sciences of the U.S.S.R.*)

*Numerisches Mathematik*

7. General reference and current reviews:

*Mathematical Reviews*, published by the American Mathematical Society, Providence, Rhode Island.

*Matematicheskii Referaty*, published by the Academy of Sciences of the U.S.S.R.

#### ACKNOWLEDGMENTS

The cooperation of the following organizations is gratefully acknowledged in granting permission to reproduce material in this chapter.

##### *General*

University of Michigan. Material from courses and summer sessions.

##### *Section 3*

University of Michigan. Number conversion tables from summer session notes.

International Business Machines. Number conversion tables from IBM 704 manual.

*Section 6*

International Business Machines. Instruction codes from the IBM 704 and IBM 650 manuals.

Remington Rand Univac. Instruction codes from the Univac II and Univac Scientific 1103A manuals.

Electrodata, a division of the Burroughs Corporation. Instruction codes from the manual for the Datatron 205.

University of Michigan. Instruction codes from manuals for the MIDAC computer and the Easiac interpretive system.

Librascope Corporation. Instruction codes from manuals for the Royal-McBee LGP-30 computer.

*Section 8*

International Business Machines. Symbolic optimum assembly programming (SOAP), from IBM 650 programming bulletins.

University of Michigan. The MAGIC system used for the MIDAC computer.

*Section 9*

General Motors Corporation. Description of the SHARE assembly program.

International Business Machines. SHARE assembly programs from IBM 704 bulletins.

*Section 10*

The Ramo-Wooldridge Corporation. Subroutines from the Subroutine Library for the Univac Scientific 1103A.

*Section 12*

Bendix Computer Division. The Intercom 101 system for the Bendix G-15 computer.

Carnegie Institute of Technology. The IT system for the IBM 650 computer, from a manual by A. J. Perlis, J. W. Smith, and Harold Van Zoeren.

*Section 17*

Massachusetts Institute of Technology, Lincoln Laboratory. Instruction codes for the TX-0 computer from reports.

## REFERENCES

1. C. W. Adams, and J. H. Laning, Jr., The M.I.T. systems of automatic coding: Comprehensive, summer session, and algebraic, *Symposium on Automatic Programming for Digital Computers*, Office of Naval Research, Washington, D. C., May 1954.

2. R. Ash, E. Broadwin, V. Della Valle, M. Greene, A. Genny, C. Katz, and L. Yu, *Preliminary Manual for Math-Matic and Arith-Matic Systems for Algebraic Translation and Compilation for Univac I and II*, Automatic Programming Development, Remington-Rand Univac, Philadelphia, Pa., April 1957.
3. *Automatic Coding for the Univac Scientific System (A Progress Report)*, Remington-Rand Univac, Philadelphia, Pa., 1955.
4. Automatic programming the A-2 compiler system, Pts. 1 and 2, *Computers and Automation*, **4**, 25-31, Sept. 1955, 15-23, Oct. 1955.
5. J. W. Backus, The IBM speed coding system, *J. Assoc. Comp. Mach.*, **1**, 4-6 (1954).
6. C. L. Baker, The PACT I coding system for the IBM Type 701, *J. Assoc. Comp. Mach.*, **3**, 272-278 (1956).
7. Iu. Ia. Basilwskii, The universal electronic digital machine (URAL) for engineering research, *J. Assoc. Comp. Mach.*, **4**, 511-519 (1957).
8. W. F. Bauer, An integrated computation system on the ERA-1103, *J. Assoc. Comp. Mach.*, **3**, 181-185 (1956).
9. I. K. Belksaja, Machine translation of languages, *Research*, **10**, 383-389 (1957) (in Russian).
10. R. W. Bemer, What the engineer should know about programming, *Automatic Control*, **6**, 66-69, March 1957.
11. R. W. Bemer, E. Krasnow, G. Jira, A. Hoggatt, and R. Levitan, *IBM 705 Print I Manual*, Programming Research Dept., International Business Machines Corporation, New York.
12. *Bendix G15 Preliminary Operations Manual*, Bendix Computer Division, Los Angeles, Calif.
13. E. K. Blum, *Automatic Digital Encoding System II (ADES II)*, Navord Report 4209, U. S. Naval Ordnance Laboratory, White Oak, Md., Feb. 8, 1956.
14. Bendix Intercom 101-G15 Interpretive Routines, Bendix Computer Division, Los Angeles, Calif., July 1956.
15. B. V. Bowden, Editor, *Faster than Thought*, Pitman, London, 1953.
16. P. Brock, and S. Rock, Problems in acceptance tests of digital computers, *J. Assoc. Comp. Mach.*, **1**, 92-87 (1954).
17. J. H. Brown, and J. W. Carr III, Automatic programming and its development on the MIDAC, *Symposium on Automatic Programming for Digital Computers*, Office of Naval Research, Washington, D. C., May 1954.
18. J. H. Brown, J. W. Carr III, B. Larowe, and J. R. McReynolds, Prevention of propagation of machine errors in long problems, *J. Assoc. Comp. Mach.*, **3**, 309-313 (1956).
19. A. Burks, H. H. Goldstine, and J. von Neumann, *Preliminary Discussion of the Logical Design of an Electronic Computing Instrument*, Institute for Advanced Study, Princeton University, Princeton, N. J., 1946 (published under contract W360340RD7481, can be obtained from Ballistics Research Laboratory, Aberdeen Proving Grounds, Md.).
20. J. W. Carr III, Solving scientific problems, *Control Eng.*, **3**, 63-70, January 1956.
21. J. W. Carr III, Towards an automatic programming procedure, *Proc. Assoc. Computing Machinery*, Meetings, May 1952, Pittsburgh, Pa.
22. J. W. Carr III, and A. J. Perlis, A comparison of large scale calculators, *Control Eng.*, **3**, 83-92, Feb. 1956.

23. J. W. Carr III, and A. J. Perlis, Small scale computers as scientific calculators, *Control Eng.*, **3**, 99-104, March 1956.
24. J. W. Carr III, and N. R. Scott, Editors, *Notes on the Special Summer Conference on Digital Computers*, Special Summer Conferences on Digital Computers, University of Michigan, Ann Arbor, Mich., 1955.
25. J. W. Carr III, and N. R. Scott, Editors, *Notes on Digital Computers and Data Processors*, Special Summer Conferences on Digital Computers, University of Michigan, Ann Arbor, Mich., 1956.
26. N. Chomsky, *Syntactic Structures*, Mouton, 's-Gravenhage, 1957.
27. N. Chomsky, Three models for the description of language, *I.R.E. Trans. on Information Theory*, **IT-2**, Proc. Symposium on Information Theory, 113-124, Sept. 1956.
28. N. Chomsky, *Transformational Analysis*, Ph.D. Dissertation, University of Pennsylvania, 1955.
29. W. A. Clark, The Lincoln TX-2 computer development, *Proc. Western Jt. Comp. Conf.*, 143-145, Feb. 1957.
30. H. B. Curry, On the composition of programs for automatic computing, *Naval Ordnance Laboratory Memorandum 9805*, Naval Ordnance Laboratory, White Oak, Silver Springs 19, Md.
31. J. I. Derr, and R. C. Luke, Semi-automatic allocation of data storage for PACT I, *J. Assoc. Comp. Mach.*, **3**, 299-308 (1956).
32. *Datatron Programming and Coding Manual*, Electro Data Corporation, Pasadena, Calif.
33. C. Elgot, On single vs. triple address and computing machines, *J. Assoc. Comp. Mach.*, **1**, 118-123 (1954).
34. Features of the new Univac II data automation system, *Manual U23*, Remington-Rand Univac Division of the Sperry Rand Corporation, Philadelphia, Pa., 1957.
35. *Fortran Automatic Coding System for the IBM 704 EDPM*, Applied Science Division and Programming Research Dept., International Business Machines Corporation, New York.
36. J. M. Frankovich and H. P. Peterson, A functional description of the Lincoln TX-2 computer, *Proc. Western Jt. Comp. Conf.*, 146-155, Feb. 1957.
37. General Motors Research Staff, *704 Operations Manual-Share Assembler*, Detroit.
38. J. T. Gilmore, Jr., TX-0 direct input utility system, *Memorandum 6M-5097*, Lincoln Laboratories, M.I.T., Cambridge, Mass., Apr. 10, 1957.
39. J. T. Gilmore, Jr., and H. P. Peterson, A functional description of the TX-0 computer, *Memorandum 6M-4789*, Lincoln Laboratories, M.I.T., Cambridge, Mass., Nov. 20, 1956.
40. H. T. Glantz, A note on microprogramming, *J. Assoc. Comp. Mach.*, **3**, 77-84 (April 1956).
41. B. Gordon, An optimizing program for the IBM-650, *J. Assoc. Comp. Mach.*, **3**, 3-5 (1956).
42. S. Gorn, An experiment in universal coding, *BRL Rept. No. 953*, Ballistics Research Laboratories, Aberdeen Proving Grounds, Md., Aug. 1955.
43. S. Gorn, Standardized programming methods and universal coding, *J. Assoc. Comp. Mach.*, **4**, 254-273 (1957).
44. E. P. Graney, Maintenance and acceptance tests used on the MIDAC, *J. Assoc. Comp. Mach.*, **2**, 95-98 (1955).

45. M. Grems, and R. E. Porter, *A Digest of the Boeing Airplane Co. Algebraic Interpretive Coding System*, Physical Research Staff, Boeing Airplane Co., Seattle, Wash., July 1955.
46. I. D. Greenwald and H. G. Martin, Conclusions after using the PACT I advanced coding technique, *J. Assoc. Comp. Mach.*, **3**, 309-313 (1956).
47. G. Hempstead and J. Schwartz, PACT loop expansion, *J. Assoc. Comp. Mach.*, **3**, 292-298 (1956).
48. G. M. Hopper, *Report to the Association for Computing Machinery—First Glossary of Programming Terminology*, Association for Computing Machinery, New York, 1954.
49. G. M. Hopper, The education of a computer, *Proc. Assoc. Comp. Mach.*, Meetings, May 1952, Pittsburgh, Pa.
50. J. H. Hughes, Tape file maintenance, *Programmer*, Sperry-Rand Corporation, July 1957.
51. Yu. I. Ianov, On matrix schemes, *Doklady Akad. Nauk S.S.S.R.*, **113**, 283-286 (1957) (in Russian; translation available).
52. Yu. I. Ianov, On the equivalence and transformation of program schemes, *Doklady Akad. Nauk S.S.S.R.*, **113**, 39-42 (1957) (in Russian; translation available).
53. *The Fortran Automatic Coding System for the IBM 704 EDPM* (Programmer's Reference Manual), Applied Science Division and Programming Research Department, International Business Machines Corporation, New York, Oct. 15, 1956.
54. *IBM 704 Electronic Data Processing*, Form 24-6661-2, International Business Machines Corporation, New York.
55. *IBM 705-III Data Processing System*, Form 22-6730-0, International Business Machines Corporation, New York.
56. *Illiac Programming Manual*, University of Illinois, Urbana, Ill.
57. D. R. Israel, Introduction to coding, *Engineering Note E-2000-1*, Servomechanism Laboratory (later Project Whirlwind), M.I.T., Cambridge, Mass., Sept. 26, 1950.
58. D. R. Israel, *The Application of a High-Speed Digital Computer to the Present-Day Air Traffic Control System*. Master's Thesis EE, M.I.T., Cambridge, Mass., 1949.
59. J. L. Jones, *A Survey of Automatic Coding Techniques for Digital Computers*, Master's Thesis, M.I.T., Cambridge, Mass., May 1954.
60. L. V. Kantorovich, On a mathematical symbolism convenient for performing machine calculations, *Doklady Akad. Nauk S.S.S.R.*, **113**, 738-741 (1957) (in Russian; translation available).
61. A. I. Kitov, *Elektronnie Tsifrovie Mashiny* (Electronic Digital Machines); Izdatelstvo "Sovetskoe Radio," Moscow, 1956 (partial translation available).
62. S. C. Kleene, *Introduction to Metamathematics*, Van Nostrand, Princeton, N. J., 1952.
63. L. N. Korolev, Coding and code compression, *Doklady Akad. Nauk S.S.S.R.*, **113**, 746-747 (1957) (in Russian; translation available).
64. J. H. Laning and N. Zierler, A program for translation of mathematical equations for Whirlwind I, *Engineering Memorandum E-364*, M.I.T. Instrumentation Laboratory, Cambridge, Mass., January 1954.
65. S. A. Lebedev, The high-speed calculating machine of the Academy of Sciences of the USSR, *J. Assoc. Comp. Mach.*, **3**, 129-133 (1956).

66. Librascope, Inc., *Librascope LGP-30, Manual*, Librascope Inc., 80 Western Ave., Glendale, Calif.
67. A. A. Markov, The theory of algorithms, *Trudy V.A. Steklov Math. Inst.*, **42**, 3-374 (1954) (partial translation available).
68. D. D. McCracken, *Digital Computer Programming*, Wiley, New York, 1957.
69. W. S. Melahn, A description of a cooperative venture in the production of an automatic coding system, *J. Assoc. Comp. Mach.*, **3**, 266-271 (1956).
70. R. J. Mercer, Micro-programming, *J. Assoc. Comp. Mach.*, **4**, 157-171 (1957).
71. R. C. Miller, Jr., and B. Oldfield, Producing computer instructions for the PACT I compiler, *J. Assoc. Comp. Mach.*, **3**, 288-291 (1956).
72. O. Mock, Logical organization of the PACT I compiler, *J. Assoc. Comp. Mach.*, **3**, 279-287 (1956).
73. E. F. Moore, A simplified universal Turing machine, *Proc. Assoc. Comp. Mach.*, Toronto, 1952.
74. J. Moshman, The generation of pseudo-random numbers on a decimal calculator, *J. Assoc. Comp. Mach.*, **1**, 88-91 (1954).
75. F. J. Murray, Univac acceptance tests, *Mathematical Tables and Aids to Computation*, **5**, 176-177 (1951).
76. A. Newell, J. C. Shaw, and H. A. Simon, Empirical explorations of the logic theory machine: A case study in heuristic, *Proc. Western Jt. Comp. Conf.*, 218-239, Feb. 1957.
77. A. P. Newell, *Notes for the Lectures on Heuristic Programs*, delivered at the University of Michigan Summer Session on Advanced Programming, August 1957.
78. A. Newell and H. A. Simon, The logic theory machine, *I.R.E. Trans. on Information Theory*, **IT-2**, 61-79, Sept. 1956.
79. A. Oettinger, Simple learning by a digital computer, *Proc. Assoc. Comp. Mach.*, Toronto, 1952.
80. R. Perkins, *EASIAC*, a pseudo-computer, *J. Assoc. Comp. Mach.*, **3**, 65-72 (1956).
81. A. J. Perlis, J. W. Smith, and H. R. Van Zoeren, *Internal Translator (IT)*—*A Compiler for the 650*, Carnegie Institute of Technology, Computation Center, Pittsburgh, Pa., 1956.
82. A. M. Pietrasanta, *Debugging Programs on the IBM-650*, New York Scientific Computation Center, International Business Machines Corporation, New York.
83. S. Poley and G. Mitchell, *Symbolic Optimum Assembly Programming (SOAP)*, 650 Programming Bulletin 1, Form 22-6285-1, International Business Machines Corporation, New York.
84. E. L. Post, Formal reductions of the general combinatorial decision problem, *Am. J. Math.*, **65**, 197-215 (1943).
85. *Proposed Specifications for Fortran for the Type 704*, Programming Research Department, International Business Machines Corporation, New York.
86. F. Ragusa and S. Zucker, The NYU Omnifax compiler and library of sub-routines, *NYU-7692 Physics Rept.*, Institute of Mathematical Sciences, New York University, New York, Oct. 15, 1956.
87. N. Rochester, Symbolic programming, *Trans. I.R.E.*, **EC-2**, 10-15, 1953.
88. P. Rosenbloom, *Elements of Mathematical Logic*, Dover, New York, 1950.
89. H. Rutishauser, Automatische Rechenplanfertigung bei programmgesteuerten Rechenmaschinen, *Mitteilung aus dem Institut für angewandte Mathematik*, Basel, 1952, pp. 1-45.

90. H. Rutishauser, Some programming techniques for the ERMETH, *J. Assoc. Comp. Mach.*, **2**, 1-4 (1955).
91. H. Schecher, Programmierung für eine Maschine mit erweiterten Adressenrechenwerk; *International Kolloquium in. Probl. d. Rechentechnik*, Dresden, 1955, pp. 69-81.
92. *SEAC Operating and Programming Notes I-VIII*, RP 1612, 1807, 1873, 1917, 1957, 2342, 2461, and 2918; Office of Scientific Publications, National Bureau of Standards, Washington, D. C.
93. *704 Electronic Data Processing Machine*, Manual of Operation (Form 24-6661-3), International Business Machines Corporation, New York, 1955.
94. C. Shannon and J. McCarthy, Editors, *Automata Studies*, Princeton University Press, Princeton, N. J., 1956.
95. R. J. Solomonoff, An inductive inference machine, *1957 I.R.E. National Convention Record*, Pt. 2 (Circuit Theory and Information Theory), pp. 56-62.
96. G. Stibitz and J. Larrivee, *Mathematics and Computers*, McGraw-Hill, New York, 1956.
97. C. J. Swift, Machine features for a more automatic monitoring system on digital computers, *J. Assoc. Comp. Mach.*, **4**, 172-173 (1957).
98. *Symposium on Automatic Programming for Digital Computers*, May 13-14, 1954; Office of Naval Research, Department of the Navy, Washington, D. C.
99. *The Uses of Digital Computers*, a series of reprints by Grabbe, Carr, Perlis, and others from *Control Eng.*, 1955-1957, McGraw-Hill, New York, 1957.
100. *The X-1 Assembly System*, Remington-Rand Univac Division, Sperry Rand Corporation, New York, 1956.
101. A. M. Turing, On computable numbers, with an application to the entscheidungs problem, *Proc. London Math. Soc.*, ser. 2, **42** (1936-37).
102. *Type 650 Magnetic Drum Data-Processing Machine* (Manual of Operations), Form 22-60 60-1, International Business Machines Corporation, New York, 1955.
103. *Univac Scientific Electronic Computing System Model 1103A*, Form EL 338, Remington-Rand Corporation, 1902 West Minnehaha Ave., St. Paul W4, Minn.
104. M. Weik, *A Second Survey of Domestic Electronic Computing Machines*, BRL Rept. No. 1010, Aberdeen Proving Grounds, Md., 1957.
105. A. N. Whitehead and B. Russell, *Principia Mathematica*, Vol. I, 1st edition, Cambridge, 1910, 2nd edition, 1925.
106. M. V. Wilkes, The use of a "floating address" system for orders in an automatic digital computer, *Proc. Camb. Phil. Soc.*, **49**, Pt. I, 84 (1953).
107. M. V. Wilkes and J. B. Stringer, Micro-programming and the design of the control circuits in an electronic digital computer, *Proc. Camb. Phil. Soc.*, April 1953.
108. M. V. Wilkes, D. J. Wheeler, and S. Gill, *The Preparation of Programs for a Digital Computer*, Addison-Wesley, Cambridge, Mass., 1952.
109. M. V. Wilkes, D. J. Wheeler, and S. Gill, *Programs for an Electronic Digital Computer*, 2nd edition, Addison-Wesley, Reading, 1957.
110. C. E. Shannon, *The Mathematical Theory of Communication*, University of Illinois Press, Urbana, Ill., 1949.
111. W. J. Eckert and R. Jones, *Faster, Faster*, McGraw-Hill, New York, 1952.
112. E. Herbst, N. Metropolis, and M. B. Wells, Analysis of problem codes on the MANIAC, *Mathematical Tables and Aids to Computation*, **9**, 14-20 (1955).
113. C. C. Gottlieb and J. N. P. Hume, *High-Speed Data Processing*, McGraw-Hill, New York, 1958.
114. J. Kister, P. Stein, S. Ulam, W. Walden, and M. Wells, Experiments in chess, *J. Assoc. Comp. Mach.*, **4**, 174-177 (1957).

115. I. Copi, *Symbolic Logic*, Macmillan, New York, 1954.
116. G. P. Dineen, Programming pattern recognition, *Proc. Western Jt. Comp. Conf.*, Los Angeles, Calif., 1955.
117. M. Minsky, Heuristic aspects of the artificial intelligence problem, *Group Rept. 34-55*, Lincoln Laboratories, M.I.T., Cambridge, Mass.
118. *Manual of Operation, IBM 650 Data Processing System with 355 Random Access Memory*, Form 22-6270-1, International Business Machines Corporation, New York.
119. *Handbook, Model 560 DATAFILE*, Multiple Bin Tape Unit, Bulletin 3026, Electro-Data Corporation, Pasadena, Calif., 1956.
120. W. N. Locke and A. D. Booth, *Machine Translation of Languages*, The Technology Press, Boston, and Wiley, New York, 1956.
121. A. D. Booth and R. H. V. Booth, *Automatic Digital Calculators*, Butterworth, London, 1953.
122. R. K. Richards, *Arithmetic Operations in Digital Computers*, Van Nostrand, Princeton, N. J., 1955.
123. S. Langer, *Introduction to Symbolic Logic*, Dover, New York, 1955.
124. M. V. Wilkes, *Automatic Digital Computers*, Wiley, New York, 1956.
125. ERA Staff, *High-Speed Computing Devices*, McGraw-Hill, New York, 1952.
126. E. C. Berkeley and L. Wainwright, *Computers, Their Operation and Applications*, Reinhold, New York, 1956.
127. G. Kozmetsky and P. Kircher, *Electronic Computers and Management Control*, McGraw-Hill, New York, 1956.
128. R. Canning, *Electronic Data Processing for Business and Industry*, Wiley, New York, 1956.
129. N. Chapin, *An Introduction to Automatic Computers: A Systems Approach for Business*, The Technology Center, Chicago, Ill., 1955.
130. E. C. Berkeley, *Giant Brains*, Wiley, New York, 1951.
131. S. Gorn and W. Manheimer, *The Electronic Brain and What It Can Do*, Science Research Associates, Inc., Chicago, Ill.
132. Pioneering in data processing, *Special Rept. No. 9*, American Management Association, New York.
133. *Electronic Data Processing in Industry*, American Management Association, New York, Aug. 1955.
134. *Scope for Electronic Computers in the Office, A Symposium*, Office Management Association, London, 1955.
135. *Impact of Computers on Office Management*, Office Management Series No. 136, American Management Association, New York, 1954.
136. Trends in computers: Automatic control and data processing, *Proc. Western Jt. Comp. Conf.*, Am. Inst. Elec. Engrs., Feb. 11-12, 1954.
137. *Large Scale Digital Computers—An Annotated Bibliography*, EL 335, Remington-Rand Univac Division of the Sperry Rand Corporation.
138. G. M. Hopper, Chairman, First glossary of programming terminology, *Committee on Nomenclature of the ACM*, The Association for Computing Machinery, New York, June 1954.
139. *Unicode, Preliminary Reference Manual*, Remington-Rand Univac Division of the Sperry Rand Corporation, U 1451, New York.
140. H. Seward, *Information Sorting in the Application of Electronic Digital Computers to Business Operations*, M.S. Thesis, M.I.T., Cambridge, Mass., 1954.
141. R. W. Bemer, The status of programming for scientific problems, *Proc. 4th Comp. Appl. Symp.*, Armour Research Foundation, Chicago, Oct. 24-25, 1957.

142. *Introduction to Transac S-2000 Data Processing System*, Philco Corporation, Government and Industrial Division, Philadelphia, Pa., May 1957.

143. A. Newell, J. C. Shaw, and H. A. Simon, Empirical explorations of the logic theory machine: A case study in heuristic, *Proc. Western Jt. Comp. Conf.*, Feb. 26-28, 1957, Los Angeles, Calif., pp. 218-230.

144. *Automatic Coding*, Proc. of the symposium held January 24-25, 1957, at the Franklin Institute, Philadelphia, Monograph No. 3, *J. Franklin Institute*, Philadelphia, Pa.

145. *Univac II Programming Manual*, Remington-Rand Univac, Philadelphia, Pa.

146. *Principal Characteristics of Bendix G-15D General Purpose Digital Computers*, Bendix Computer Division, Los Angeles, Calif.

147. J. Jeanel, Programs as a tool for research in systems organizations, *IBM J. Research and Development*, **2**, 105-122 (1958).

148. *IBM 650 Manual of Additional Features—Magnetic Tapes, High-Speed Storage, Printer*, Form 22-6265-1, International Business Machines Corporation, New York.

149. *Univac Generalized Programming*, Management Services and Operations Research Department, Remington-Rand Univac Division of Sperry Rand Corporation, 1957, New York.

150. H. Kahrmanian, Univac Programming Report, Sperry Rand Corporation, Philadelphia, Pa.

151. S. N. Razumovskii, On the question of automatization of programming of problems of translation from one language to another, *Doklady Akad. Nauk S.S.S.R.*, **113**, 760-762 (1957).

152. A. D. de Groot, Über das Denken des Schachspielers, *Rev. Psicol.*, **50**, 73-104 (1956).

153. J. Wegstein and J. Cooper, Corbie system for the 704, BSCRB, National Bureau of Standards, Washington, D. C., 1957.

154. J. W. Carr III, Editor, Notes on applications of logic to automatic programming, Engineering Summer Conference, University of Michigan, Ann Arbor, Mich., 1957.

155. R. M. Graham, Translation between algebraic coding languages, *Paper 29*, *Assoc. Comp. Mach.*, 13th Natl. Meeting, Urbana, Ill., June 11-13, 1958.

156. A. J. Perlis and J. W. Smith, A command language for handling strings of symbols, *Paper 30*, *Assoc. Comp. Mach.*, 13th Natl. Meeting, Urbana, Ill., June 11-13, 1958.

157. R. W. Bemer and D. A. Hemmes, Computer language compatibility through multilevel processors, *Paper 31*, *Assoc. Comp. Mach.*, 13th Natl. Meeting, Urbana, Ill., June 11-13, 1958.

# THE USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

## C. THE USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

H. T. Larson and  
R. B. Conn, Editors

3. *Data Processing Operations*, by *M. J. Mendelson*
4. *Quantitative Characteristics of Data Processing Systems*, by *R. L. Sisson and R. G. Canning*
5. *Equipment Description*, by *J. W. Busby and J. H. Yienger*
6. *Facility Requirements*, by *E. Tomash*
7. *Design of Business Systems*, by *H. S. Levin*
8. *Accounting Applications*, by *A. C. Vanselow*  
*R. L. VanWinkle*  
*L. L. van Oosten*  
*E. D. Cowles*  
*H. Tellier*
9. *Inventory and Scheduling Applications*, by *C. E. Ammann*  
*C. W. Schmidt and R. Bosak*
10. *Scientific and Engineering Applications*, by *R. T. Koll*
11. *Handling of Non-Numerical Information*, by *M. E. Maron*



## Data Processing Operations

*M. J. Mendelson*

1. Introduction	3-01
2. Data Collection	3-02
3. Data Conversion, Transcription, and Editing	3-03
4. Data Output	3-04
5. On-Line Versus Off-Line Processing	3-04
6. Scientific Data Manipulation	3-05
7. Business Data Manipulation	3-06
8. Checking	3-13

### 1. INTRODUCTION

Data processing begins the instant any action takes place which will generate information whose content will be a subject for subsequent analysis, and terminates only when all such analyses have been completed, records up-dated, and reports generated. In some instances it may never be completed since these outputs may themselves become new sources of data or evoke new pieces of information for processing. In any event, since data processing is a continuous and not a discrete process, planning for an efficient data processing system must begin at some point earlier than the generation of the first information and extend through all subsequent operations. This chapter will delineate and define the more important and basic operations which permeate all data processing situations. These definitions will serve to clarify the content of the succeeding chapters.

## 3-02 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

### 2. DATA COLLECTION

It is important for the efficiency of the data processing operation that data be recorded in some form which admits of subsequent automatic processing. Generally, it is equally desirable that this recording be effected at the earliest possible point in the process since it is only at this time that all the contributing factors are simultaneously present, the adequacy and correctness of the record may be verified, and errors of recording rectified. In addition, the earlier the recording, the more the processing can be automated. Because of this desire, it is convenient to break data collection situations into two categories: (1) Processor controlled collection systems are those which are processor oriented. (2) Collection controlled collection systems are those which are collection oriented. This split is chosen since the subsequent processing may be profoundly effected by the category to which its data collection system belongs.

**Language, Medium, and Structure.** These are important aspects of data collection systems. *Language* is the particular set of marks and symbols which are used to represent data in any particular consistent system. These may take such forms as printed characters, holes punched in particular places on a document, or groups of magnetic markings on a magnetizable surface. *Medium* is the physical unit on which the information is recorded and retained. Typical examples are the card which is punched, the paper which is printed, the magnetic tape which is magnetized. *Structure* pertains to the organization of the data, with respect to such factors as sequence, groupings, and special punctuations.

**Processor Controlled Collection Systems.** These are systems in which the language, medium, and structure of the data collected are made to conform with the requirements of the input system of the data processor which will perform the subsequent analyses. Processor controlled collection systems represent the ideal situation from the viewpoint of data processing system efficiency, and should be achieved if the operation and the data collection process permit. Achievement of this goal eliminates time-consuming and possibly error-producing intermediate operations.

*Manual Systems.* Intermediate operations are most common when data generation is manual and there is no alternative but for human operators to transcribe the data to machine language by a subsequent second manual operation of equipment which produces suitable media. In this instance, since the second manual operation is required, the characteristics of the processor input system can in great part determine the

nature of the transcription equipment and the structure of the transcription process.

*Automatic Systems.* When data collection is automatic, it may still be possible to define the characteristics of the data collection system on the basis of the processor input system and produce input data suitable for processor assimilation. Two conditions must generally be met before this can be true.

1. The collection process and instruments must be such that they permit the collection of the data in a suitable form on a suitable media.

2. There exists no requirement that the primary data be available in a form for direct and immediate human utilization.

**Collection Controlled Collection Systems.** Many data collection situations are not readily adapted to the above conditions. Where any combination of the process, the instrumentation, or human requirements determine any particular combination of the media, or language, or structure of the data collected, the system is called *collection controlled*. Data may also be forced into a particular structure by virtue of the requirements of remote transmission equipment.

**EXAMPLES.** Scientific data collection is often analog in form and not suitable for digital processor assimilation. Business data must often be in a form suitable for human consumption and therefore they may not be in proper language, or on a proper media, or in a proper structure for direct processor use.

### 3. DATA CONVERSION, TRANSCRIPTION, AND EDITING

Where data collection is collection controlled, an intermediate step consisting of a combination of conversion, transcription, and editing processes is generally required.

*Conversion* is the process of changing data from one language form to another. Analog data may be converted to digital data; binary data may be converted to decimal data; data in one coded system may be converted to data in a second coded system.

*Transcription* is the process of changing data from one medium of recording to a second. Written records may be transcribed to punched cards; punched paper tape data may be transcribed to magnetic tape recording.

*Editing* is the process of changing data from one structure to another. This may consist of changing the sequence of the data on the storage medium and of adding or deleting information. These functions are also usually required at the output end of the system where data must be transformed appropriate to its end use.

### 3-04 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

#### 4. DATA OUTPUT

At the other end of the data processing stream lies the problem of output data preparation. Here the end use of the data will determine their language, medium, and structure. Data output preparation may be conveniently categorized on the basis of this end use. In general, data are prepared by the processing system for use by (a) humans, (b) for later reuse by itself, or (c) for use by machines other than itself.

**Output Data for Human Use.** The most common form of output use for human consumption is the printed document. This may take several forms:

1. A printed report whose entire content is generated within the processor, or a document generated from the insertion of information on a partially preprinted form such as an invoice or a check.

2. A graph or pictorial representation of a set of information.

**Output Data for Machine Reuse.** The nature of data generated for machine reuse is often dependent on whether the information is retained internal to the machine's system or is to be transmitted outside the machine's domain for subsequent return and reentry. Magnetic tape files are an example of the former, punched card checks of the latter.

**Output Data for Other Machine Use.** Data may often be generated by one machine for subsequent use by another. This may occur where a piece of processing equipment also acts as a data collecting device for later processing operations.

**EXAMPLES.** The point of sale type of data processor which prepares punched tape for subsequent processing. Installations where remote data transmission by teletype is a system requirement.

#### 5. ON-LINE VERSUS OFF-LINE PROCESSING

Once the data have been organized in a suitable language, are on a suitable input medium, and have a structure suitable for assimilation by automatic processing equipment, they may be subjected to a number of data manipulative processes. Of fundamental importance for consideration at this point is the time position of the data processor in the data processing sequence. A data processor is called *off-line* if data are collected on a storage medium and brought to the processor for processing at a later time. The data processor is called *on-line* if it participates directly in the data processing operation, manipulating the data as they are produced.

In a second sense, if the output of the data processor is made available at practically the same time as the input data which called for the process, it is said to be an *on-line* processor, but if its output can satis-

factorily be produced with a significant time delay, it is said to be an *off-line* processor.

**EXAMPLES.** A department store in which the daily sales activity is collected from a set of point of sale recorders and daily analyses compiled at the end of the day would be using the point of sale recorders in an on-line fashion and the data processor in an off-line fashion. An automatic airline reservation system in which space availability, sales of seats, and reservation cancellations are automatically processed as they occur is an example of a system operating in an on-line fashion.

Where the processor is in an on-line status and is required to keep pace with some physical phenomena, it is often referred to as a *real-time* processor. *Example.* A digital autopilot and control system which must accept current status signals from an aircraft's sensing system, process them, and provide proper control signals would be an example of a real-time processor.

The determination of which status is required of a processor in a given data processing situation will have a profound effect on the entire data processing system to be employed.

## 6. SCIENTIFIC DATA MANIPULATION

Scientific data manipulation functions can be characterized only in a broad sense. Most scientific problem solutions consist of a central processing scheme peculiar to the problem being solved working in conjunction with a set of standardized suboperations.

**Central Processing Methods.** The central processing methods tend to be so highly individualistic as to defy classification. The alternative seems to be, therefore, to define classes of problems rather than classes of operations. See Chap. 10 for a detailed description of these classes of problems.

**Suboperations.** Although the central process in a scientific calculation is unique, the suboperations it employs are generally drawn from a "library" of such operations which are common to many processing situations. The suboperations of such a library are of three major types:

1. *Function Generation.* Function generation, as its name implies, consists of the evaluation of a function for a given value of a variable. Obtaining the sine of a given angle or the square root of a given number is a typical suboperation of the function generation class.

2. *Computing Sequences.* Operations such as the integration step in the solution of a set of differential equations or the multiplication of two matrices in a coordinate transformation operation are common to many different scientific data processes.

## 3-06 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

3. *Modification Operations.* These are suboperations which are designed to have the effect of modifying the machine program to one more suitable to a given application. Typical routines of this class provide for such facilities as floating point arithmetic operations or complex number manipulation.

### 7. BUSINESS DATA MANIPULATION

Since the initial applications of data processors in the business field have been largely of a record keeping and statistic reporting nature, it becomes quite possible to characterize business data manipulation on an operation basis. That is, rather than each application being unique unto itself as in the scientific field, business data problems tend to show a strong similarity of structure and generally can be shown to consist of combinations and sequences of some fundamental operations.

As the modern data processors are applied to the business field, more and more of the data analysis will be turned over to the processor and, therefore, an increasing proportion of the business applications will tend to match their scientific brothers in individuality.

**Units of Information.** The description of business data processing operations which follow will be greatly simplified by a definition of a few fundamental terms describing the basic units of information involved in a processing situation. The unit of information in any data processor is the *binary digit*. A binary digit may be defined as the amount of information described by a single yes-no decision. (See Vol. I, Chap. 16.) In electronic systems, all information is ultimately expressed in terms of binary digits.

Although the binary digit is a fundamental unit of information within a data processor, it is of little use in communicating information to the human users of a system. The human user communicates by means of combinations of symbols each of which represents far more information than a single bit. The fundamental element of this set of symbols is defined to be the *character*. The characters of a typical business system consist of the numbers from zero to nine, the alphabetic symbols from A to Z, punctuation marks, and special symbols such as dollar signs, percentage symbols, and asterisks. A group of characters used to describe a piece of information is called a *field*. A field may contain any number of characters. A unit of information which may be described only by a combination of a number of fields of information will be termed an *item*. The particular field of an item by which the item is identified in any processing situation is often termed the *key*. An assemblage of items, usually ordered with respect to a key, necessary to describe a

complete system of information will be termed a *file*. The items of a file are often referred to as *records*.

**EXAMPLE.** One example will suffice to clarify these definitions. An inventory file consists of a set of items each describing one unit of merchandise, and ordered according to the field containing the stock number as a key. Other fields of each item contain descriptive and quantitative information, such as color or amount on hand, expressed in terms of the basic characters available within the system. If such a file is recorded on a magnetic tape, the ultimate form of data representation is in terms of binary digits of information as defined by magnetized areas on the surface of the tape.

### **Data Rearranging**

**Editing.** In addition to the editing functions required in order that data may be initially assimilated by a processor, there are a number of other data manipulations which may be grouped under the general category of data rearranging. Because the initial input medium is quite often one of the slower communication elements in the processor system, it is generally desirable to minimize the number of times that data are transferred from it to the processor. However, the initial input data quite often contain information which will ultimately be distributed to, and have an effect on, a large variety of different processing activities. It is expedient, then, to absorb the data from the input medium only once, and, in the process, create a number of different standard items of information each peculiar to its own end use. *Example.* A punched paper tape from a point-of-sale recorder may contain such diverse information as clerk number, customer number, stock number, quantity sold, sales price, state tax, and federal tax. From this information, reports and records such as clerk sales analyses, buyers reports, inventory records, customer billing, and tax reporting are generated. For each report, a specific subset of information recorded on the paper tape must be extracted, regrouped, and appropriately organized for the data processes which it will undergo. This type of processing thus constitutes a second kind of editing problem.

**Sorting.** If it is assumed that a set of standard items relating to a given reporting function has been generated, it is generally required that the items be placed in some orderly sequence. The process of arranging a set of items of information into a sequence dependent upon a specific set of characters within each item is called *sorting*. The need for sorting information stems from two principal sources. First, many auxiliary storage systems used for the recording of files of information

### 3-08 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

are of a serial access nature. That is, the time for acquisition of a given item in a file is dependent on the relationship of its position in the file to the position of the last preceding item acquired. In such a situation, it is absolutely essential that the file be used in an orderly fashion in order to minimize the time delays of data acquisition. This is most easily accomplished by keeping the file in some fixed sequence and sorting the input information to that sequence before file processing begins. When suitable random access auxiliary storages (i.e., each data acquisition time is essentially independent of any preceding data acquisition) become available, file updating processes will cease to be a reason for sorting. However, the advent of such pieces of equipment will by no means eliminate the need for sorting since a second requirement for sorting exists which is independent of such equipment. Most reports generated for human use must be presented in an orderly fashion, and this order is quite frequently determined by the input data themselves. The buyer's report of our previous example is a record of the sales made in the departments which come under his jurisdiction. This record is only of use to him if it is presented in a logical sequence that can be determined only by sorting the input information, no preorganizing being possible since there is no control over what items will be sold on a given day.

**Merging and Collating.** It is frequently required that two sets of items, each independently sorted according to a common key, be combined into a single set of items sorted with respect to that same key. This operation is called *merging* when the two sets of items have an identical item structure, and it is called *collating* when the two sets of items have independent structures (except for the common key, of course). Merging operations are frequently used in obtaining a sort on a large set of data. They are often conveniently used when several different activities may affect a common file. For example, separate ordered reports may be required to indicate sales of merchandise to customers, receipt of merchandise from vendors, and returns of merchandise from customers. Each of these reports would require a separate sorting operation. Since each of these activities will affect a common inventory file, it is desirable to perform a merge or collation of these sets of data before the file updating process begins.

**Match-Merge.** In some processing systems, a number of variations of the merge operation are provided. One such variation is called the match-merge. In this operation, one set of items is considered as a control set and the second as a master set. The result of the match-merge operation is to produce an output set of items which consists of an ordered combination of the control set and those items of the master

set whose keys exactly match the keys of the control set. Such an operation is useful in selecting from a large file only those items which are actually to be operated on in a subsequent process. This is most frequently done where special equipment is available to accomplish the match-merge process since it relieves the equipment involved in the subsequent process of the task of recognizing, and time delay involved in bypassing, inactive items in the master file. Another variation of the merge process uses the control set to cause the insertion or deletion of items in the master set.

**Scanning.** Scanning is the process of moving systematically through a set of data in search of the items whose key meets a specified set of criteria. Since in summarizing a set of sorted data, one wishes to accumulate all those items with a common key, it is necessary to scan the item keys until one is found which is different from its predecessors. The results of such a scan thus determine the exact point at which the current accumulation should be terminated and a new scan operation begun.

Another common scan application is that of *table-look-up*. Many business functions can only be expressed in a tabular form whose entry arguments are nonuniformly spaced. A utility rate table where the rate changes do not occur at equal increments of consumption is an example of such a function. In order to obtain the correct function value for an arbitrary argument, it is necessary to scan the table until one finds an entry argument which is greater than or equal to the specified arbitrary argument. The result of the scan determines the exact point at which the correct function value can be found. A scanning operation also occurs when it is required to sift out of a set of items all those which have a common set of characteristics. The scanning of a set of loan payment records to determine all those which are delinquent is an example of such an operation.

### **File Maintenance**

Almost without exception, every business application requires the maintenance of files. The addition or deletion of records and the addition, deletion, or modification of information within a record constitute the major elements of the file maintenance problem. The removal of the records of no longer stocked items from an inventory file or the deletion of the records of subscribers who have failed to renew a magazine subscription are typical examples of a record deletion operation. Similarly, the introduction of records pertaining to items that have just become active in a system constitutes the bulk of the addition of records type of operation.

### 3-10 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

**Nature of the File.** An important determinant in the file maintenance process is the physical nature of the file system employed. If the information storage system of the file enforces, either by its physical structure or logical organization, a "nextness" relationship between records (i.e., no record may be inserted between any two adjacent records) it will be termed a *continuous* system. When no nextness relationship is implied, the file system will be termed *discrete*. Magnetic tapes and drums (without special index devices) are typical examples of continuous file systems. Punched cards represent the most common example of a discrete file system. If the system is such that a particular record may be modified, at least in content if not in space occupied, and replaced in its original position in the file without having any effect on the remaining contents of the file, the system is said to admit of *selective recording*. All discrete systems and some continuous systems admit of selective recording.

**Continuous File Processing.** In a continuous file system which does not admit of selective recording, it is necessary to make a completely new copy of the entire file whenever any portion of the file is to be modified; with an obvious concomitant time cost. Where the number of records to be modified is a small percentage of the total file, this time cost can become quite excessive. If a continuous file system has the ability to locate a particular record independent of the data processor and its medium admits of selective recording, a large percentage of this time may often be saved provided the modified record does not require more storage space than is available. Any continuous system faces a severe problem when the file maintenance procedure requires addition and deletion of information in the file. Here, even though selective recording may be available, the nextness characteristics of the continuous system preclude the possibility of its use. The entire file must be recopied in such an operation.

**Discrete File Processing.** Discrete systems generally obtain their characteristics in one of two fashions.

1. The actual storage medium may be divided into discrete pieces with a single record of information occupying one or more such pieces, and not more than one record occupying a single piece. In such a situation, it is trivially possible to record selectively, to expand, or to contract the file without disturbing unmodified data.

2. The actual storage medium is physically continuous but it is broken into discrete pieces logically by means of an index which is used to define where each piece of stored information is located. Such a system always admits of selective recording and of expansion and contraction of information. It is to be pointed out, however, that the index is a special

file itself and has many of the problems of file maintenance inherent in its own structure. Further, many file modifications also require index modifications thus doubling the work required in these instances. After a reasonable period of use, the indexing system tends to become quite complex and cumbersome in its attempt to describe the locations of new records added, new pieces of old records stored in spaces not physically near their parent record, and spaces made available by the deletion of old records. In such systems, a periodic housecleaning and revamping of the file and its index is required for any reasonable efficiency in file processing.

### **Retrieval of Information from Storage**

Most files are organized with respect to some key characteristic of the data they contain, with direct access to the file being available only through the specification of the correct key. For example, an inventory file is organized with respect to the item stock number as a key, and no item may be located directly without specific knowledge of its stock number. Thus, even though a verbal or technical description completely defining a single item may be available, direct access to the item is not possible without its stock number. In view of this fact, it is convenient to break information retrieval operations into two classes, one which is externally controlled and the other internally controlled.

**Externally Controlled Retrieval.** The first class is termed *external* because sufficient information is available external to the file itself to determine the precise identification of the desired record. A request for the current status of a particular item whose stock number is known is an example of an external file operation.

**Internally Controlled Retrieval.** On the other hand, it is frequently required that an item be located with respect to some characteristic other than the key with respect to which it is filed. This situation gives rise to the second class, which is internally controlled. *Internal* situations require that a scanning operation be applied to a file in order to accomplish the location of the desired information. (Many file systems have this as a requirement for locating information even when the key is known.) This may be further clarified by considering a particular application and a particular file system. Consider a loan account file, ordered according to account number, and stored on a selectively recordable magnetic tape in a file system which has the ability to locate any record if it is given the account number. The posting of payments in such a system would become an example of an externally controlled operation since it would be trivially possible to identify the proper account number when the payment is received. The process of determin-

### 3-12 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

ing delinquencies is an example of an internally controlled operation since the file must be scanned to locate those records in which no payment has been posted.

**Categorical Types of Information Retrieval.** This operation brings up another characteristic internal operation of fundamental importance in automatic file systems. Information to be obtained from a given file often consists of locating all those records whose characteristics match a prespecified set of characteristics. An inventory file might be scanned in order to produce a list of all those items which are used in the manufacture of a specific product, which are in short supply, and which require a purchase lead time of six weeks or more as an example of such an operation. Manual systems have provided for such information searches (which the author terms *category search*), on a limited basis, provided the particular set of classifying characteristics were known at the time the system was installed. Edge-punched cards are a common example of such a system.

Automatic systems for the first time provide the user with the ability to provide economically for internal file operations of this type on a large scale and not necessarily predetermined basis. It is apparent that giving the business man the ability to scan his records for information matching any prescribed set of characteristics provides him with one of the most powerful analytical tools available in modern data processing equipment, and that this feature will be one of the first to be exploited after the first rush of straight data reporting jobs have been taken care of.

**The Interrogation Problem.** One of the most important and at the same time difficult problems facing the user of automatic data processing equipment in the business field is the necessity for making file information available to both the processing equipment in its language, structure, and medium and to the human users in theirs. Unfortunately, situations arise where the people using a data processing system simply must be able to obtain specific information from the records contained within the system on a random and reasonably rapid basis. A department store customer who wishes to pay up his account before leaving town on an extended trip must be provided with his current balance within a short time of his request. One simply does not tell him that he cannot leave town until the twenty-fifth because that is when his bill is due to be computed according to the current machine billing procedure. Yet such data as this must be contained within the automatic auxiliary storage of the processing equipment and cannot be simultaneously and economically duplicated in a form satisfactory for human use. At the present time, no really satisfactory solution for this problem exists. Present

methods generally require a printing of a minimum amount of information for each record in the file to serve as a basis for a manual interrogation process. This problem can be readily handled by a system with a random access storage, but no such device is today capable of providing sufficient processing speed for the bulk data processing activity to justify its inclusion for the interrogation problem. In certain on-line data processing systems with reasonably slow input-output requirements such a solution is provided. The automatic reservation devices employed by some airlines and railways are examples of such systems.

## 8. CHECKING

Few systems can tolerate the introduction, generation, or perpetuation of erroneous information. Of course this statement applies to a greater or lesser degree depending on the particular situation. One erroneous data point in a missile tracking situation will not have profound effect on the final analysis. A mistake in an inventory record of a few hundred washers would not upset a hardware warehouse, but a similar mistake might be catastrophic to a supplier of household laundry equipment. Error detection and correction takes on a considerably more important aspect in automatic data processing situations than it has in manual systems for several reasons. The data within an automatic system does not pass under human scrutiny where reasonability judgments may readily take place. The sale of five thousand fur coats might be readily accepted by an automatic business system, whereas it would be immediately questioned by its human operators. Just as important perhaps is the fact that the user has a considerable investment in his data processing system and that location and error correction may entail the execution of a difficult, time-consuming, and costly operation. It is therefore extremely important that provision be made to detect and provide correction procedures for all errors which may be introduced in any phase of operation of an automatic system.

Checking procedures may be described as being either *automatic* or *programmed*. Automatic checking systems provide for the verification of the correctness of all operations by special equipment built into the automatic processing system. As such, automatic checking systems do not introduce any new function for the system user. Programmed checks, however, are basic data processing functions because they consist of the verification of the correctness of system operation by means of special procedures introduced by the system user.

**Input Data Checking.** Three techniques are in common use for the checking of input data.

### 3-14 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

1. Where the sequence with which data are entered is important, each input item is made to contain a sequence number, and a programmed check for the continuity of this sequence number is provided.

2. Where the correctness of a particular set of input characters is especially important, a redundancy check may be employed. In this case, the set of input characters is augmented by the addition of one or more characters whose nature is determined in a well-defined fashion by the input characters with which they are associated. Subsequent to the input operation, a programmed check is performed to verify that the input characters and their associated check characters satisfy the correct relationship. An example for numerical data entry verification is to provide a "mod 11" check character. This character is obtained by alternately adding and subtracting the digits of the data, casting out any multiples of 11 which may accrue during the process. (The mod 11 check digit for the number 52719 is obtained as  $9 - 1 + 7 - 2 + 5 = 18$ ,  $18 - 11 = 7$ .) This digit is then entered along with the data (thus 52719 would be entered as 752719), and a programmed check provided to ensure that the check character matched the data entered. The mod 11 check ensures that no single digit was entered incorrectly and that no transposition of digits occurred, and is, therefore, an excellent check against the most common types of operator transcription errors.

3. The third common form of input data check is the sum check. In this instance, a set of input items are arbitrarily chosen and each field of each item is added into a common total. This total is then entered together with the set of input items to which it refers and a similar summing operation is performed subsequent to the entry process.

**Checking of Data Processing.** Operations internal to an automatic data processor often permit programming checks. Internal operations may be repeated and results compared or a given result may be obtained by two independent methods. It is often the case that two independent and necessary results can act as a crosscheck against each other because some fundamental relationship is known to exist between them. It is sometimes possible to provide reasonability type checks on the results of computations. In scientific problems differencing techniques will reveal out of line results. In business data processing situations, reasonable activity levels may be established and recorded for checking purposes. Such a check would remove the possibility of recording a sale of ninety-five mink coats because of the entry of an incorrect stock number in an inventory updating procedure.

**Output Data Checking.** Checking of data transmitted from the internal processor to output equipment is usually accomplished by automatic checking equipment since it is generally impossible to acquire

any information on output results for programmed checks. One major exception to this is the checking of magnetic recording. No automatic check of magnetic recordings is satisfactory that is not obtained by a direct reading subsequent to writing of the actual magnetic record produced. Since such a system is not generally provided in current equipment, the only satisfactory check of magnetic recording is a programmed check in which the data are read back from the magnetic medium and compared with those which were transmitted originally. It is sometimes possible to circumvent this requirement by delaying the read back check until the current recording operation has been entirely completed, and then executing some useful processing operation utilizing the recorded information. By this means a writing error from the previous operation can be caught and corrected during the current operation provided the magnetic system admits of selectable recording.



## Quantitative Characteristics of Data Processing Systems

*Roger L. Sisson and Richard G. Canning*

1. Determining System Requirements	4-01
2. Basic System Characteristics	4-02
3. Basic Equipment Characteristics	4-04
4. Measurement of System Factors	4-04
5. Relating System Characteristics to Equipment Characteristics	4-09
References	4-16

### I. DETERMINING SYSTEM REQUIREMENTS

The objectives of this section are to point out the basic characteristics or factors which exist in any data processing system, and to discuss methods of (a) measuring the system characteristics and (b) relating the system characteristics to the possible equipments for performing the systems functions.

The boundaries of the system under consideration are set to exclude areas over which the designer has no control. The information flowing into the system across these boundaries determines the input characteristics, and the type and nature of the inputs can usually be measured. The information that flows out of the system is produced in accordance with clearly defined requirements which are the output characteristics. It is assumed that the designer has complete control over the structure within the boundaries, being limited only by (1) the techniques known, (2) equipment, (3) the economics of the situation (in military systems

## 4-02 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

values other than dollar economy might be defined), and (4) policy restrictions.

The steps in designing a data system are:

1. Determine and clearly state requirements.
2. Determine the sources of data (outside the system) which allow the realization of these requirements.
3. Determine, from an analysis of the requirements, the sequences of data manipulations and transmissions to meet the requirements.

### 2. BASIC SYSTEM CHARACTERISTICS

#### **Input Factors.**

1. The rate at which data come into the system and the fluctuations in this rate for each source of data. See Vol. I, Chap. 16.
2. The significance of the data. Each quantity must refer to some entity in the physical or economic system, either explicitly, by symbols or descriptive matter, or implicitly, by its location in relation to other data.

#### **Output Factors.**

1. The rate and variations in rate.
2. The significance of each output specified explicitly (e.g., by a heading), implicitly (by format), or by data previously supplied. An output might be a variable which is a function of one or more others, the rate of change (trend) of a variable (occasionally higher order rates of change are required), or the past history of a variable (sometimes expressed as the integral).

**Internal System Characteristics.** The internal structure of a system may vary, depending upon the job being done. However, many data systems fall into the following pattern:

1. Data collected. Input recorded, ready for use by the system when required. Conversion, transcription, and editing might occur here. See Chap. 7.
2. Input data stored (filed), often by storage of the recording medium.
3. Data transmitted to a central point (data gathering).
4. Data batched, i.e., collected into groups for ease in processing.
5. Data stored or filed, to make it possible: (a) to hold for processing or until external processes affecting the data occur; (b) to maintain records of past events (historical or integral data); (c) to combine data for processing; (d) to determine rates of change of data (trend analysis).
6. Computations and logical manipulations performed on the data, such as: (a) maintenance of the files noted in step 5; (b) computations required for the outputs. The inputs for these steps consist both of incoming data and data extracted from the central files.

7. Conversion and transcription of the data to a usable form.  
 Table 1 illustrates these steps in typical systems.

TABLE 1. EXAMPLES OF DATA PROCESSING SYSTEM CHARACTERISTICS

Function	System			
	Business System (Inventory Control)	Data Reduction (Guided Missile)	Computation (Linear Programming)	Control (Tape Driven Machine)
1. Record data	Record transactions on keyboard	Not recorded in missile; transmit to ground where recorded on tape	Record parameters on paper	Record on magnetic tape during preparatory process
2. File data (temporary)	(Not done)	Store tape	Hold data (paper)	Hold tapes
3. Gather data	Transmit electrically to paper tape punch	Send tapes to processing center	Organize parameters in form required by program	Receive error signals, and signals from tape
4. Batch data	Combine paper tapes	Put tapes in proper sequence	Organize parameters in form required by program	None (on line)
5. Store data	File on magnetic tapes	Not required	File on magnetic tape	None (on line)
6. Manipulate File maintenance	Bring master files up to date	Not required	Linear program computation	None (on line)
Output preparation	Prepare reports, orders, etc.	Edit, calibrate, compute (data reduction)	Linear program computation	None (on line)
7. Transcribe	Print out	Print out, plot	Print out answers	Convert to servo signals to guide machine tool

**General System Factors.**

1. Accuracy, correspondence of results to reality (error rate).
2. Precision, repeatability.
3. Flexibility, ability to solve new problems (meet new situations).
4. Cost (or value, e.g., to the military).
5. Installation time.
6. Reliability, probability of the machine being available when needed.

## 4.04 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

### 3. BASIC EQUIPMENT CHARACTERISTICS

Table 2 outlines briefly the factors which characterize data processing devices.

TABLE 2. BASIC EQUIPMENT CHARACTERISTICS

Equipment	Factors Defining Data Processing Characteristics
Communication or transmission devices	Peak capacity to transmit information Error rates
Storage (storage filing) elements	Storage capacity Maximum rate of data entry or extraction Access time to individual elements of data, minimum, average, and maximum
Arithmetic and logical manipulators	Basic operations the equipments can perform Rate of performance of basic operations Input rate Output rate
Data converters, including input and output units	Form of input to converter Rate of conversion of data from input form to converter output form Form of output from converter

### 4. MEASUREMENT OF SYSTEM FACTORS

This section describes how the characteristics described above are measured and discusses them in more detail.

**Location of Sources of Data.** Table 3 gives examples of sources of data and typical data rates.

TABLE 3. EXAMPLES OF DATA SOURCES

Type of Data Processing	Location of Source	Typical Data Rates
Guided missile data reduction	Instruments in the missile	0.1-10,000 binary digits per second
Scientific computation	Person who defines the problem	A few binary digits per minute
Business systems	Devices (or persons) that record data about transactions (sales, purchases, work done, etc., and the results of management decisions)	From a few binary digits per day to 200 binary digits per second

### Source Data Units

Source data rate is measured in number of binary digits per second (as a function of time) entering the system. If data are recorded in a definite format, it is possible to use other units such as: (1) documents per unit of time, where all documents have approximately the same number of digits; (2) transactions per unit of time, where the same data are recorded about each transaction, e.g., points between which a machine tool is directed to travel.

The data may enter the system at a uniform rate; but more often they enter at random intervals so that a choice must be made between a system handling the average rate but introducing delays, and the more expensive system which is capable of handling peak loads.

**Measurement of Source Data Rates.** This is usually performed by one of the following methods.

1. Estimating from knowledge of the type of source. For example, knowing the limits of performance of a guided missile, it is possible to estimate the maximum rate of data transmission required to record faithfully the variation in the various factors, e.g., fuel pressure.

2. Determining the existing rate of input, e.g., counting incoming sales orders over a period of time in a business system.

3. Knowledge of the problem. For example, by knowing the size of the matrix problem to be solved, it is possible to determine the number of coefficients to be fed into the computing system per iteration.

### Storage of Data

The two key factors in data storage systems are (1) storage capacity and (2) rate at which data must enter and leave storage.

**Storage Capacity.** This quantity is basically measured in binary digits. However, a particular system may use derived measures. For example, in a decimal system, storage may be measured in decimal digits. Often a system handles data in groups of digits because of the storage media used (e.g., cards) or because of the equipment design (many equipments handle small groups of digits called *words*). Then the storage capacity is stated in terms of the number of words in the storage or the number of cards it will hold.

In every storage or filing system, data within the storage are located by an *address*. The address is a number (or code) which the access or scanning equipment recognizes, and it is used to locate the data for insertion or withdrawal.

**Data Entry.** Entry of data into a storage system can occur in two ways: (1) the data can already be arranged into address sequence; (2)

## 4-06 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

data arrive at the storage randomly with respect to the address at which a given item will be stored. These are referred to respectively as *sequential* and *random* arrivals. The requirements imposed on the system may determine the sequence in which data arrive at the storage. Very often, however, this choice is left to the system designer. For example, the designer may allow data which are arriving in a random fashion to enter storage as they arrive, or he may first collect a group of the incoming data and order them on a unit which sorts them into the desired sequence of address numbers. Then this set of data can be fed into a storage sequentially. Thus the system for entering data into storage cannot be divorced from the types of equipment available.

**Collation Ratio.** This is defined as the number of unit records (of a file) used in a given period of time to the total number of unit records in the file. With low collation ratios (few records affected per cycle time) the use of sequential data storage units becomes costly. If the collation ratio is high, and if the data can be arranged in the proper sequence, sequential storage may be more practical.

**Size of Files.** The process of determining the size of files and the access type and time required is an integral part of the system study and involves a detailed analysis of the problem. The file size depends upon: (1) instantaneous rates of data coming into and out of the file and (2) the number of items which the file represents.

**Access Requirements.** These are determined by: (1) the rate of information arrival and withdrawal, both average and peak; (2) the maximum permissible delay between the arrival of an inquiry of the file and final response to the inquiry; (3) the size of the file.

### Computation and Logical Manipulation

The basic operations performed by a data processing system are described in Chap. 3. The job of the system designer is to arrange these basic operations so that the required outputs are obtained in the most economical manner within the permissible time limits. An important question is how to measure computation and manipulation. Almost every system has basic items or transactions which it is processing (e.g., payroll records, points in space—missile position, iterations of a relaxation computation).

**Manipulation Rate.** This can be measured as items or transactions processed per unit time. Examples of the items or transactions that define the size of a given data processing task may be obtained by studying Chaps. 8, 9, and 10.

**Computation Rate.** As with storage, the computational characteristics of a system often cannot be completely measured without relating

the problem to the equipment which can perform the job. If a general purpose computing device is used in part of a system, it can be programmed to perform any computational procedure required. On the other hand, a special purpose computer might be designed to perform only that sequence of operations required of a particular system. The problem in either case consists of estimating the economics and time involved.

### Output Factors

The form of the output is determined by the requirements of the elements, external to the system, which are to utilize the information from the system. The factors to consider when determining these requirements are: (1) the *language* or *form* of data presented, e.g., analog or digital; (2) the *format*, that is, the arrangement of data; (3) the *medium* (see Chap. 5); (4) the *rate* at which data will be produced and the fluctuations in the rate.

**The Rate of Output Data.** This can be derived by knowing the input data rates from the data sources, the computational procedures used, and the requirements for output information content. Since this process is rather lengthy in complex systems, it is sometimes sufficient, for quick estimates of the output factors, to count or to measure the output rates in an existing system. These data must be used carefully, however, because the new system is often being designed to provide some new and different outputs. In some cases the new system will result in more output data; in other cases, the output rate is reduced because the new system performs more of the decision-making functions internally (see Vol. I, Chap. 15).

### Errors and Reliability

Important characteristics which must be considered at every point in a system are error rate and reliability.

**Error Rate.** All the characteristics mentioned above (for instance, rate of input and rate at which computations proceed) must also include information on rate at which errors may occur. With good design it is possible to reduce the probability of undetected errors to such a low value that it is effectively zero in actual operation of the system. But when error detecting and correcting methods are included (see Chaps. 5 and 13), it becomes important to recognize that the effective rate at which data are arriving or computation is proceeding is reduced because part of the information is redundant (i.e., involved with the error correction procedures).

## 4-08 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

**Reliability.** The reliability requirements of the system are also extremely important. There are some systems (for instance, the control of air traffic) in which, ideally, no failures can be tolerated. On the other hand, there are other systems such as certain business systems in which machine failures would merely delay the overall process, but would cost little money and certainly no lives. Where extremely high reliability is required, the cost of the system must be expected to be higher.

### **Flexibility**

A data processing system is a tool designed to process information. As with any tool, there may be "tool design" and also "setup" time and effort required, if the tool is expected to perform a variety of specific jobs. The tool design aspect of a data processing system includes such things as programming a computer, or wiring the plugboard of punched-card or analog computing equipment. Setup includes such steps as entering the program into the computer memory and inserting the plugboard.

Where a large variety of jobs must be processed, the system must be flexible, and the ease and rapidity of designing and setting up for a new job become vital. In high volume applications, the programming (design) occurs less often (usually only to make improvements) so that flexibility is less important. In this case, special purpose machines designed to perform only one specific job can be effective.

Flexibility is measured by comparing the time and cost for preparing for a new job required by the systems under consideration.

### **Cost**

Given specific system characteristics, there generally are specified economic limits within which the equipment costs, installation costs, and operating costs must fall, if the system is to be satisfactory.

**Equipment Costs.** These include the cost of data transducers and recorders, transmission equipment, data converters, input devices, computers and data processors, output devices, auxiliary data storage equipment, office furniture, supply storage equipment, air conditioning machinery, and maintenance equipment.

**Installation Costs.** These include three major items:

1. The planning of the new data processing system, organizing for its inception, and training personnel to operate the new system (see Chap. 7).
2. The preparation of the physical facilities for the new equipment, and the actual installation and check out (see Chap. 6).
3. The operation of the new system in parallel with existing systems

for a period of time, to establish its reliability and shake out the system bugs before depending upon it (see Chap. 7).

**Operating Costs.** These costs include the types of costs associated with operating any system of men and machines. The major items of operating costs are depreciation or rental, salaries and wages and associated overhead, maintenance, space, supplies, and power.

### **Installation Time**

An important overall system requirement or factor is the time required to put the system into operation. The major steps which account for the installation time are those listed under installation costs in the previous paragraph.

## **5. RELATING SYSTEM CHARACTERISTICS TO EQUIPMENT CHARACTERISTICS**

The equipment characteristics are described in detail in Chap. 5. This section will attempt to show how the system designer must relate various system factors in selecting equipment.

### **Recording, Transmission, and Temporary Storage**

The input source factors are determined by location, rate, and content. These affect equipment characteristics in several ways:

1. Devices may be needed for converting physical phenomena (pressure, shaft position) or informational inputs (quantity of items processed, item of a sales order) to a form which the data processing equipment can then handle, such as electric impulses, punched cards, and mechanical motions.

2. Devices for recording these data for temporary storage may be required.

3. The temporary storage media itself must be selected.

4. Communication links to bring the data from sources to intermediate and to central processing centers are usually involved.

**Location.** The location of data sources in relation to each other and to the central processing units of the system determines the need for the recording, temporary storage, and communication links in the input part of the system.

**Rate and Content.** The rate at which data are generated at the sources in relation to each other and to the processing rate determines the need for temporary storage within the input system. If data are generated at peak rates which differ from processing rates or the rate at which data are generated at other sources, some temporary storage is required. Often data are generated intermittently and/or processed

## 4-10 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

intermittently, so that storage is required to hold data during inactive periods. The need for and size of temporary input storage is related to the type and size of storage facilities in the central processing unit and to the type of communications available. For example, broad-band links which handle peak loads of information may require little storage; links available only intermittently may create a need for more input storage.

Temporary input storage is generally sequential, since any rearranging to be performed is accomplished at the main processing center; that is, data are read out in the same sequence as received but sometimes at a different rate.

The need for conversion is brought about when, for example, the source data are in analog form and the processing or communications are digital. In almost every system the data as they arrive from the source must be converted to some other form or medium for further processing into the system. For examples, see Table 4.

TABLE 4. EXAMPLES OF DATA CONVERSION

Source Form	Required Form
Pressure	Electric voltage
Shaft position	Digital signals (pulses representing numbers)
Handwritten figures on paper	Punched cards
Coded information on magnetic tape	The same information in a different code on paper tape

EXAMPLES. *Relation of Data Source and Input Equipment.*

1. *Guided missile system.* The data are generated at various rates at points throughout the missile (see Fig. 1). The data are converted

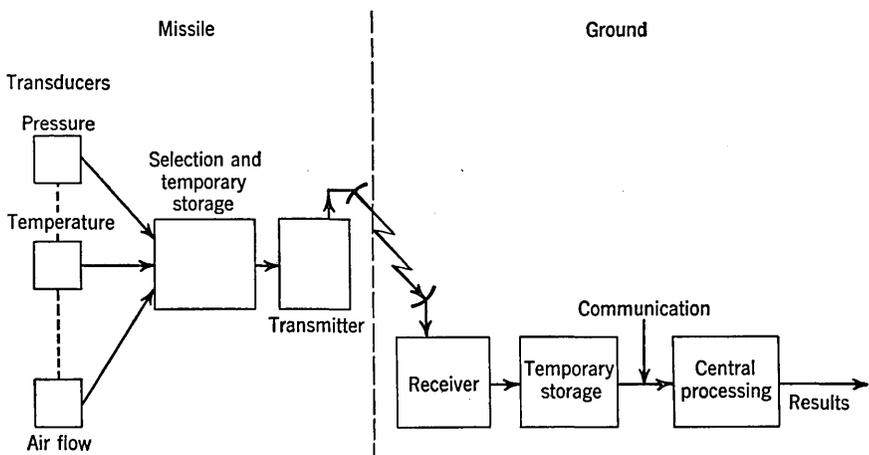


FIG. 1. Relation of data source and input equipment, guided missile system.

by transducers from the physical form to a form convenient for transmission. They are transmitted to a central air-to-ground transmission link by communications systems within the missile. Some temporary storage may be required here as the internal transmission rate may differ from the rate at which air-to-ground communications occur. The data are then transmitted to the ground. Here they may be communicated to a processing center, or may be stored (e.g., on magnetic tape) and then processed later.

2. *Processing sales orders.* The data originate at random times with the customers and are communicated by mail to a branch office of the firm (see Fig. 2). Here they are stored, by holding the media, and then

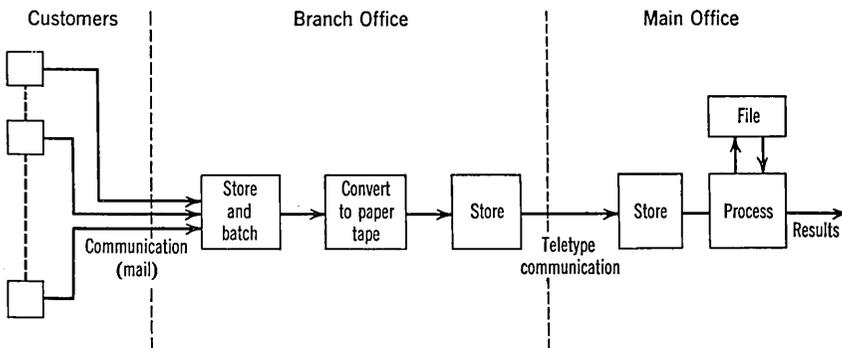


FIG. 2. Relation of data source and input equipment, sales order processing.

eventually further translated by retyping into standard forms and punched on paper tape. Data are then communicated (by teletype, for example) to the main office where they may be stored again, and finally processed on an intermittent basis.

**Storage Equipment (Memory Units)**

The storage capacity of the required equipment relates directly to the storage requirements in the central processing system. There must be provision for equipment to hold the number of binary digits (basically) that must be stored.

The rate and sequence in which data will arrive partially determine the type of access or scanning system required. In a sequential storage the access mechanism locates the desired item by scanning one item after another, usually in order of the address numbers. In a random access storage, the access mechanism can go directly to the desired item (address). Most storage systems are a compromise. Table 5 shows that many access systems are partly random and partly sequential.

## 4-12 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

TABLE 5. SCALE OF DATA STORAGE ACCESS METHODS

		Examples
Scale of Randomness (Arbitrary)	Completely Sequential	Tape (magnetic and paper)
	↑	
		Tape drum and tape loops
		Magnetostrictive lines- Acoustic lines
		Magnetic drum
	Semirandom	Disk array
		Ledger card
	↓	
	Completely Random	Magnetic cores

The functional difference between random and sequential access systems is great, and the system design will vary appreciably, depending upon which is used. With a sequential storage; data may have to be batched and sorted by address numbers before entering the file or storage.

The choice between random and sequential methods and equipment is affected by the collation ratio. See Sect. 4. With sequential equipment it is often nearly as easy to examine all unit records of a file as it is to examine a few, so that systems with a high ratio can be processed almost as cheaply as those with a low ratio.

In a sequential system access time is measured for two cases: (1) time to read or write the next item in sequence; (2) rate at which data are transferred when transferring continuously.

The access time is measured as the time required to locate the desired item, sometimes termed the search time. If not a completely random system, the average access time is given.

If a naturally sequential system (e.g., magnetic tape) is used as a random access system (which is usually possible) then the system would have a long mean access time with a large variance or range. See the magnetic tape entry in Table 6. The naturally "random" system is

TABLE 6. ACCESS TIMES OF STORAGE DEVICES

Storage Device	Representative Capacity	Representative Access Times		
		Min.	Mean	Max.
Magnetic cores <sup>a</sup>	5000-600,000 binary digits	10 $\mu$ sec	10 $\mu$ sec	10 $\mu$ sec
Mercury delay lines <sup>a</sup>	20,000-50,000 binary digits	50 $\mu$ sec	200 $\mu$ sec	400 $\mu$ sec
Magnetic drums				
Univac file computer drum	180,000 alpha-numeric characters	1 msec	18.5 msec	36 msec
Univac Larc drum	3,000,000 decimal digits	3 msec	1.25 sec	2.5 sec
Datatron drum <sup>a</sup>	40,000 decimal digits	0.5 msec	8.5 msec	17 msec
Magnetic tapes	1,000,000-28,000,000 characters per reel	10 msec	2 min	10 min
Tape strips (data file)	20,000,000 decimal digits per unit	2 sec	16 sec	48 sec
Magnetic disks	6,000,000 decimal digits	150 msec	500 msec	800 msec
Photographic disks	4,000,000 decimal digits	0.1 msec	12.5 msec	25 msec

<sup>a</sup> Internal storage devices.

characterized by a short mean access time with a small range or variance (e.g., see magnetic drums and magnetic cores in Table 6). Mixed systems often have a short mean time but a wider variance than the naturally random system. For example, a group of short tapes might have a mean access time of 100 milliseconds and a range of 5 to 200 milliseconds. See Chap. 20.

### Computational and Logical Processors

The computational and manipulation facilities required are determined by the processing which must be performed. Table 7 reviews some of the characteristics of digital computing equipment and their relation to system factors.

The size of storages (2 and 3) will be influenced in part by whether the system is on line or off line. On-line systems require large internal memory to hold programs for all possible occurrences, but small auxiliary memories. Off-line system must have large auxiliary storages to hold batches of data, but can be operated with smaller internal storages.

The flexibility required by the system determines the requirements for ease in reprogramming or rearranging the equipment. Analog computing equipment is often used when a number of different problems must be

## 4-14 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

Table 7. EQUIPMENT FACTORS AND RELATED SYSTEM FACTORS

Equipment Factors	Related System Factors
1. Internal operation, including index registers, buffers, special commands for sorting, merging, floating point	1. a. Manipulations required b. Flexibility and ease of programming and setup required
2. Internal storage	2. The need for storage of factors and programs during computation
3. Auxiliary storage	3. The requirements for storage of all information in the system at any one time
4. Capacity: number length, alphabetic facilities	4. Language and size of the fields in the data processed
5. Reliability	5. Freedom from undetected errors required; efficiency required (effective computing speed, including redundancy for error detection and correction, and down time)
6. Cost	6. System economics
7. Physical size	7. Space available
8. Availability	8. System installation plans
9. Auxiliary features: converters, "file processors," sorters	9. Auxiliary system requirements: Conversion, transcription, editing, sorting and file manipulation requirements (not to be performed in a computer)

solved quickly, since it is easy to program or design the system for a particular problem. Programming for a digital computer is more costly, although special features and techniques are available to make programming easier. Programming ease can be "bought" by sacrificing computing time through the use of automatic programming techniques. See Chap. 2.

### Output Equipment

The form and media on which the output must be presented determine the type of output equipment. The equipment must accept the data in the form and on the media available from the processor proper. The output equipment must translate or convert these data to a required output form, including inserting the format arrangement or position data inserted to make interpretation easier. The equipment must handle the data at the required rate. Details such as requirements for multiple copies, reproduction copies, instantly visible copies, and other factors not necessarily explicit in the data themselves must be considered.

The control of format may be performed in two places, in the computational facilities or within the output units themselves. In either case

the format control must be able to provide the foreseeable range of formats.

The output media required, of course, determine the type of output units which can be used.

If the computations are performed digitally and an analog output is required, a digital-analog converter is required having the proper characteristics.

In many output systems it is common to measure the output rate in terms of the output form rather than basic binary digit rate. Thus, we have units such as decimal digits per second, points plotted per minute, lines printed per second, and cards punched per minute. When output information rates are given in terms of these larger units, additional information is often needed to determine the actual information rate. For example, the information per line depends on the number of characters in each line.

The output requirements also can affect the computational procedures. They can affect the sequence in which computations are performed, so that data arrive at the output in the correct order.

TABLE 8. TYPICAL REDUNDANCY CHECKING METHODS

Type	Typical Use	Redundancy
Parity	Checking magnetic tape data transfers	One binary digit of 7
Redundant digits	Checking internal computer transfer and arithmetic operations, e.g., in Datamatic 1000	4 binary digits of 40
	Self-checking number systems Calculating and checking "batch" or control totals	One decimal digit of 7 Typically 10 digits of 1000
Duplication	Repeating complete message transmission (duplication in time)	100%
	Duplicate arithmetic logic, unit, e.g., in Univac	100%
Reasonableness	Checking to see if data deviate by a given percentage from an extrapolation of previous data	The computation involved in extrapolation, comparison and storage of limits <sup>a</sup>
Consistency	Checking agreement of stock number and name	Storage of correct combination and checking calculation time <sup>a</sup>
Completeness	Checking to see if a field is missing or if an item is skipped in a sequence	Checking calculation time <sup>a</sup>

<sup>a</sup>Such checking typically consumes 10% of the time for a complete calculation.

### **Equipment Error Rate and Reliability**

The undetected error rate permissible directly affects the type of equipment which can be selected for a given system. Where extremely low undetected error rates are required a large amount of redundancy must be introduced into all phases of equipment. Redundancies of about 17 per cent (e.g., a parity bit for every six information binary digits) are common. Redundancy as high as 100 per cent, that is, complete duplication, is used (e.g., equipment duplication as in the Univac arithmetic unit, or a time duplication as in the IBM transceiver). Communication channels must carry extra data, in the form of error-correcting codes, for instance. Computational procedures must include a large number of checks in the form of redundancy (duplication, perhaps) and consistency checks. See Table 8.

### REFERENCES

1. R. G. Canning, *Electronic Data Processing for Business and Industry*, Wiley, New York, 1956.
2. C. W. Adams, Automatic data-processing equipment: a survey, *Electronic Data Processing in Industry*, pp. 125-138, American Management Association, New York, 1955.

## Equipment Description

*J. W. Busby and J. H. Yienger*

1. General Equipment Description	5-01
2. Characteristics of Electronic Data Processing Equipment	5-04
3. Input Equipment	5-09
4. Storage Equipment	5-24
5. Output Equipment	5-33
6. Arithmetic and Logic Unit	5-38
7. Control Equipment	5-40
8. Typical Electronic Digital Equipment	5-43
References	5-43

### I. GENERAL EQUIPMENT DESCRIPTION

#### **Organization of Electronic Digital Equipment**

All electronic digital computers and data processors have five functional elements, namely input, storage, arithmetic and logic, output, and control, as shown in Fig. 1.

The input unit is the means of getting both the data to be processed and the directions for processing this data (control instructions) into the equipment. The storage unit provides a resting place for all data placed into the equipment, for data developed during the processing operations, and for the instructions that direct the operations. The arithmetic and logic unit processes the data according to the rules of arithmetic and a predetermined logic. The output unit accepts the results of the processing and passes them out of the equipment. The

## 5-02 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

control unit directs the processing operations by informing the various units when and how they should perform. The arithmetic and logic, storage, and control units are referred to as the *processing unit*.

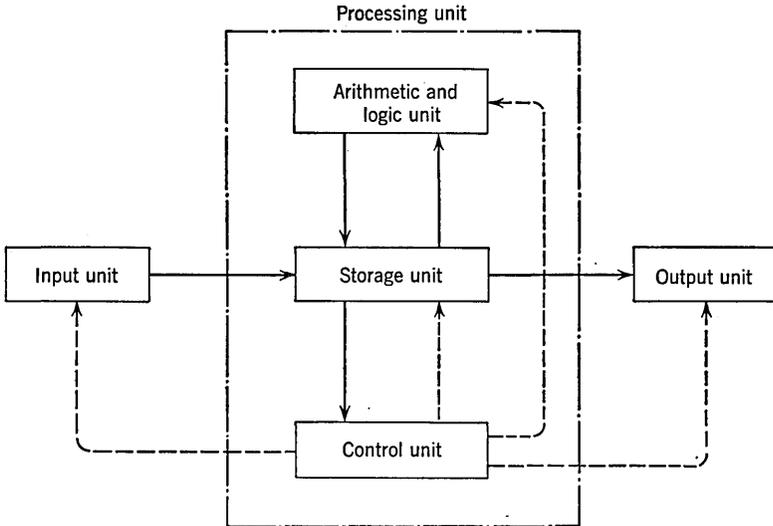


FIG. 1. Organization of electronic digital equipment.  
Transmission: information — ; control - - -.

**Major Components.** The major components that make up electronic digital equipment are illustrated in Fig. 2.

*Input Media.* The most widely used input media are punched cards and magnetic tape. They contain the data to be processed and the directions for the processing in a coded form.

*Storage Media.* The most widely used storage media are magnetic cores, magnetic drums, and magnetic tapes. They hold the data to be processed and the instructions for the processing in magnetic form. Magnetic drums and magnetic cores are permanent components of the storage unit, whereas magnetic tapes may be disconnected from the processor and replaced with other reels.

*Arithmetic and Logic and Control Units.* The arithmetic and logic and control units consist of electronic circuits.

*Output Media.* The most widely used output media are punched cards, magnetic tapes, and printed reports. These media are used to record the results of the data processing. Punched paper tape, electric typewriters, visual displays and special electronic equipment can also be used for output.

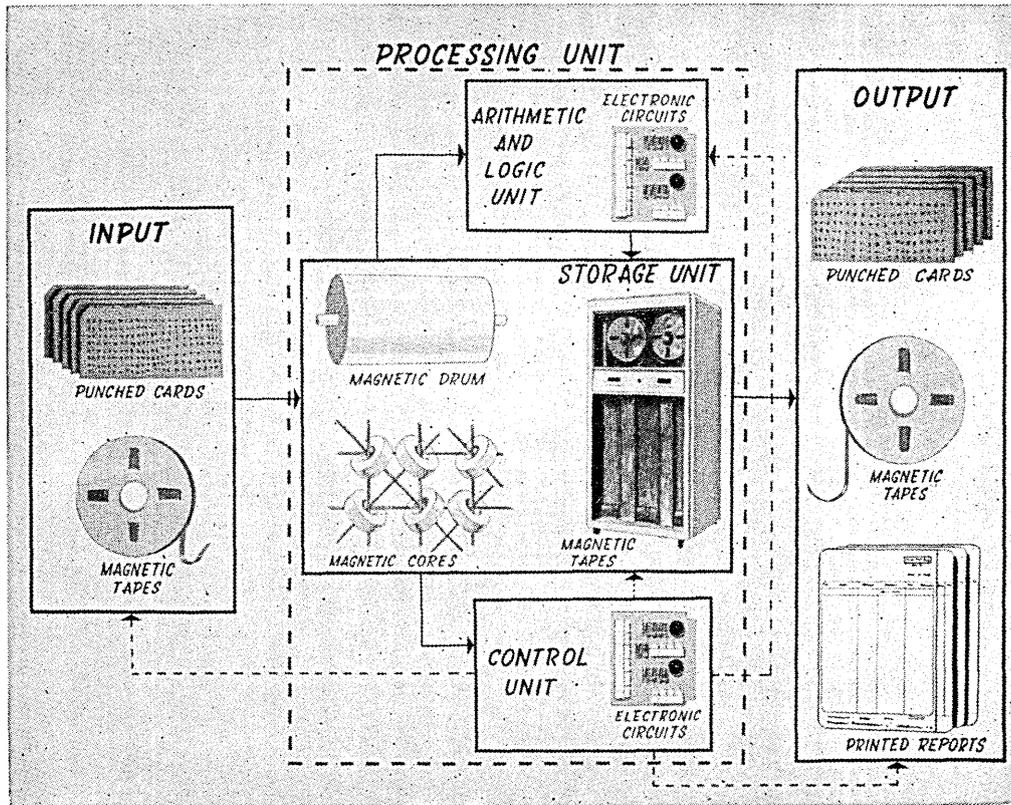


FIG. 2. Components of electronic digital equipment.

## 5-04 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

**General Purpose and Special Purpose Equipment.** *General purpose* electronic digital equipment has the ability to perform a large variety of problems. This equipment is designed to perform a set of basic arithmetic and logical operations. The sequence of the operations executed can be readily changed by changing the stored program. This feature makes general purpose equipment very versatile and results in a flexibility that permits it to be used in a variety of applications.

*Special purpose* electronic digital equipment have the following characteristics:

1. They are limited in the types of problems they can solve.
2. Most special purpose equipment is designed for only one particular application.
3. The sequence of operations which special purpose machines execute is often fixed into the control unit. To change this sequence, it is necessary to make an engineering change within the equipment.
4. Special input media are usually employed.

EXAMPLE. Reservation machines (see Chap. 9).

**Distinction between Scientific and Business Machines.** There is no sharp dividing line between scientific and business electronic digital equipment. In general, the two kinds of machines have the same functional organization and can do the same things. Electronic digital equipment for scientific applications is designed with emphasis primarily on the speed of arithmetic operations and the ability to perform complicated arithmetic processes. Scientific computations are characterized by generally low volume of input and output, little or no reference file requirements, and extensive repetition of internal arithmetical operations.

Electronic digital equipment for business applications is designed with emphasis primarily on fast means of transferring data into and out of the equipment, large capacity file storage, and the ability to manipulate data. Business data processing is characterized by voluminous file storage requirements, high volume input and output of data, and generally nominal amounts of internal computations.

New designs tend to combine the features of scientific and business machines.

## 2. CHARACTERISTICS OF ELECTRONIC DATA PROCESSING EQUIPMENT

**Number Systems.** Numbers may be described by the digits composing them and the number system on which they are based. The number system used in everyday living is the decimal system, based on the number 10. Electronic computers use some form of the binary number system (see Chaps. 2 and 12).

The conversion of a decimal number to a binary number and vice versa can be performed by the electronic digital equipment. The basic arithmetic operations of addition, subtraction, multiplication, and division can also be performed in the binary system. See Chaps. 12 and 18.

**Coding of Information.** The numerical data that have to be processed are usually in decimal form on the input documents, and the results of the processing have to be in the same form on the output documents.

Electronic digital equipment has been constructed to represent numbers in the decimal system. However, this is a very inefficient and costly method of number representation in electronic form. Hence, the binary number system or a binary coded system is usually employed. (See Chaps. 12 and 13 for other codes.)

**Binary Code.** The *binary system* has been widely used in electronic digital equipment, especially computers for scientific and engineering work. The major disadvantage to the binary number system is that it is difficult for human beings to interpret without training.

**Binary Coded Decimal.** Another method of coding information in electronic digital equipment is the binary coded decimal system or the 8421 system shown here:

<i>8 4 2 1 Binary Coded Decimal</i>	<i>Decimal Equivalent</i>
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9

A decimal number such as 267 is represented as (0010) (0110) (0111), while 1959 is represented as (0001) (1001) (0101) (1001). The advantages of this system are that it is fairly easy to interpret and there is room for additional symbols because the ten-decimal digits use only ten of the sixteen possible combinations. A disadvantage is that it is not as efficient as the binary system if the six additional code combinations are not used.

**Alphanumeric Codes.** The coding of decimal digits can be extended to include alphabetic characters and special symbols. Six binary digits make it possible to accommodate sixty-four different combinations. Therefore, six binary digits can handle the ten-decimal digits, twenty-six

## 5-06 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

alphabetic characters, and up to twenty-eight additional characters such as punctuation marks and special symbols. One adaptation of the use of six binary digits to represent various characters is as follows:

<i>Character</i>	<i>Binary Coded Character</i>
1	000001
2	000010
3	000011
4	000100
.	.
.	.
.	.
9	001001
0	001010
A	110001
B	110010
C	110011
D	110100
.	.
.	.
.	.
Z	011001
,	011011
%	011100
\$	101011
*	101100

This type of binary coded character representation is used in the alphanumeric machines. Its advantages are that (a) it is fairly easy to interpret, and (b) twenty-eight possible combinations left after the ten-decimal digits and twenty-six alphabetic characters provide room for additional symbols.

**Information Representations.** Number systems are represented within electronic digital equipment by discrete electric signals. These signals are interpreted by establishing a correspondence between their characteristics and the characteristics of the number system used, as shown in Fig. 3.

Electronic digital equipment is designed to treat a specific group of digits as a single item called a "word." It can treat a sequence of binary digits as a binary word, binary number, a binary coded decimal word, or a binary coded decimal number, as illustrated in Fig. 3.

**Words.** A word within electronic digital equipment is a grouping or a collection of digits and/or characters that represents a specific number or another piece of information. Most arithmetical and logical opera-

tions are performed upon the word as a whole. In fixed word length equipment, the size of all words, such as ten-decimal digits, is constant. In variable word length equipment, the size of the words is not necessarily constant.

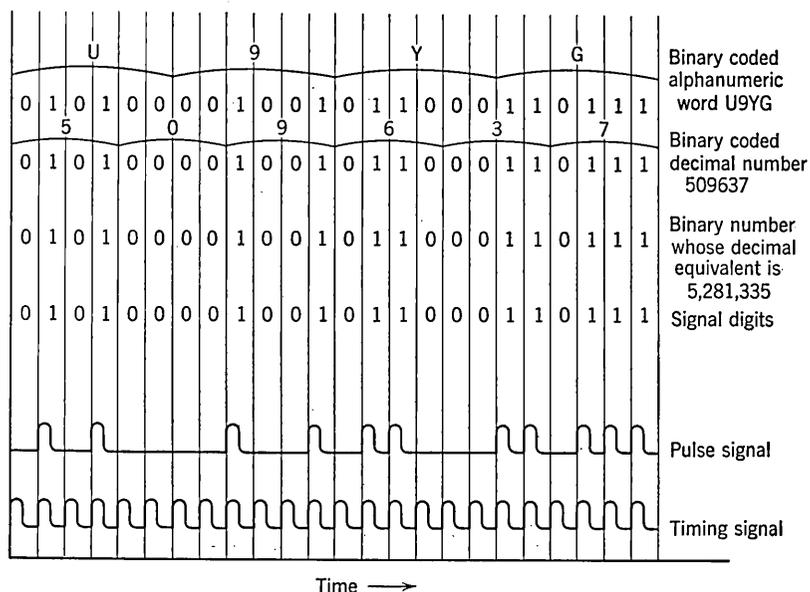


FIG. 3. Discrete electric signals.

**Operating Speeds.** Elementary switching operations in computers can be performed at rates of the order of 1,000,000 per second. However, in most electronic digital equipment, the arithmetic and logical data processing operations require several hundred switching times or periods and, therefore, are performed at rates of the order of 1000 to 10,000 per second.

**Control Signals and Instruction Code.** The collection of digits or characters representing a control signal is interpreted as a control word. The complete list of control words, together with an identifying description, is called the instruction code of the data processing equipment. This instruction code is the list of the different operations that the equipment can perform. (See Chap. 2.)

**Addressing.** Distinguishing one storage location from another is accomplished in certain electronic digital storage devices, such as magnetic cores and magnetic tapes, by assigning an address to each storage location.

## 5-08 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

In some magnetic tape storage units, the individual storage locations on magnetic tape are not assigned addresses. The storage locations are distinguished by the pieces of information stored in the locations. The pieces of information contain distinguishing labels such as account numbers or part numbers and are stored in a particular sequence on the magnetic tape. When a particular piece of information is required in this addressing scheme, the equipment has to search for it because it does not know exactly where it is located.

**Types of Instructions.** The various types of instructions used are shown in Fig. 4.

Operation code	First operand address	Second operand address	Result address	Next instruction address
----------------	-----------------------	------------------------	----------------	--------------------------

(a)

Operation code	First operand address	Second operand address	Result address
----------------	-----------------------	------------------------	----------------

(b)

Operation code	First operand address	Second operand address
----------------	-----------------------	------------------------

(c)

Operation code	Operand address	Next instruction address
----------------	-----------------	--------------------------

(d)

Operation code	Operand address
----------------	-----------------

(e)

FIG. 4. Instruction types: (a) four-address instruction, (b) three-address instruction, (c) two-address instruction, (d) one + one address instruction, (e) one-address instruction.

**Programming and Coding.** Before electronic data processing equipment can solve a problem or process data, it is necessary for human

beings to prepare a program for the equipment. This program is a specific, predetermined sequence of instructions that the equipment can perform. The program is prepared by the human operations of problem definition, programming, coding, and debugging. (See Chap. 2.)

### 3. INPUT EQUIPMENT

**Scope of Input Function.** The input unit connects the electronic digital equipment with the external world. It supplies the data that are to be processed as well as the instructions that guide the processing. It converts the data from the form used in the external world into machine language.

The complete input function starts with the preparation of the media that carry the required data into the system from the source documents. It ends with the actual placing of the data and control instructions into the equipment. The input preparation equipment is not connected to the electronic digital equipment, but it is included so as to bring out the full scope of the input function. The preparation of the input media is usually accomplished by operating a keyboard and manually keying the information contained in the source documents into the input media. In general, the same amount of data to be processed will require the same number of operators working for the same length of time to prepare the input media, no matter what input media are used.

#### **Requirements of a Good Input Medium.**

1. Machine language that can be fed automatically into electronic digital equipment or can be automatically converted to such a medium.
2. Simple and inexpensive recorder. Most applications require much recording equipment; therefore, it is necessary to minimize cost and maintenance.
3. Obtainable as by-product. Input medium should be obtained as a by-product of another machine operation, thereby eliminating duplication.
4. Compact, inexpensive, and lightweight.
5. Visually auditable, for checking and editing.
6. Reusable.
7. High-speed input; approaching internal speed of electronic digital equipment.
8. Automatically convertible to high-speed input, if not a high-speed input itself.

**Comparison of Input Media.** Based on the above requirements, Table 1 gives a comparison of punched cards, punched paper tape, paper documents, and magnetic tape. Table 2 compares speeds and costs (1958 data) for these input media.

## 5-10 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

TABLE 1. COMPARISON OF INPUT MEDIA

Requirements	Punched Cards	Punched Paper Tape	Paper Documents	Magnetic Tape
Machine language	Yes	Yes	Yes	Yes
Simple and inexpensive recorder	No	Yes	Yes	No
Obtainable as by-product	Yes	Yes	Yes	No
Compact and inexpensive	No	Yes	No	Yes
Visually auditable	Yes	Yes	Yes	No
Reusable	No	No	No	Yes
High-speed input	No	No	No	Yes
Convertible to high-speed input	Yes	Yes	Yes	—

TABLE 2. SPEEDS AND COSTS OF INPUT MEDIA

Speed and Cost <sup>a</sup>	Punched Cards	Punched Paper Tape	Paper Documents	Magnetic Tape
Speeds, char/sec	150-2,500	400-2,000	500-1,000	6,000-60,000
Costs, approximate				
Input media, per char	\$0.00001	\$0.0001	—	\$0.00001
Preparation equipment				
Key unit	\$2,500	\$2,250	\$500	\$4,500
By-product	\$3,000	\$500	—	N.A.
Input units	\$15,000	\$4,000	—	\$20,000

<sup>a</sup> Based on 1958 costs.

**Speed and Costs.** Magnetic tape is by far the fastest input medium available. The cost of a magnetic tape input unit is the greatest, but the cost per character in information transferred per unit of time is much less. This accounts for the extensive use of magnetic tape as an input medium.

The cost of the input media is very small and does not play a large part in the overall cost problem. The cost of the equipment required to obtain punched cards as a by-product is much greater than the cost for punched paper tape. This is one of the reasons punched paper tape has been more extensively used as a common language medium.

### Input Preparation Equipment

**Key-Driven.** The most basic method of preparing input media is that of finger-driven keyboard devices. The two most prevalent keyboards available are the typewriter keyboard and the ten-key numeric keyboard. The typewriter keyboard is a general purpose board for handling the ten-decimal digits, twenty-six alphabetic characters, punctuation marks,

and special symbols. In cases where the data are, or can be made to be, all numeric, the ten-key numeric decimal keyboard is frequently used.

Card punches, paper tape punches, and magnetic tape preparation devices are the most widely used key-driven input preparation equipment. These devices prepare the input media as the operator keys the data onto the keyboard. For example, as the operator keys the data onto the keyboard of a card punch, the data are permanently recorded in the card by means of the punched holes.

**Speed.** The speed with which information can be keyed onto an input medium depends upon the type and form of the data to be recorded, the operator's familiarity with the data, and the equipment being employed. Operators using typewriter keyboards can average 7000 to 12,000 key depressions per hour. A ten-key numeric keyboard can be operated 30 to 50 per cent faster because of its smaller size.

**By-Product.** Input media can be produced as a by-product during the preparation of other material, such as a bill, invoice, or check. In this operation, the input preparation equipment is directly connected to another machine, such as a typewriter, bookkeeping, or accounting machine. Then as this machine is operated, the data are not only recorded on a document but are also recorded on the input medium. This eliminates one entire keying operation.

Equipment, such as couplers, makes it possible to connect card punches, and paper tape punches to standard office machines, such as typewriters, bookkeeping machines, adding machines, and desk calculators. This office equipment uses either the typewriter keyboard or the ten-key numeric keyboard. The speed with which these devices can be operated is equivalent to the operating speed of the regular key-driven devices. However, these by-product machines require more time for set up.

**Automatic.** The objective of this equipment is to accomplish the recording of input data in machine readable form as a by-product of other necessary automatic functions. This reduces the time expended and cost involved in preparing an input medium. Also, it increases the accuracy of the input data.

In the process control field, special gages and meter readers can convert their sensings to digital data. These digital data can then be either recorded on an input medium or transmitted from the sensors to a central recording station. This eliminates one human operation of reading the gages and meters and another human operation of recording these data on an input medium.

In the manufacturing field, time clocks that punch time cards are used to record the working hours of employees automatically. Other devices

## 5-12 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

can count, measure, and weigh products coming off assembly lines. Then this information is automatically recorded in machine readable form for further processing.

**Verifiers.** When preparing an input medium, especially by a human operation, there is an opportunity for errors to be made in the recordings. If these errors are not corrected, they will enter the processing and thereby affect all succeeding operations. Therefore, it is a necessary part of the input function to locate and correct errors that have been made during the original recordings.

Several techniques are available for verifying input data: (1) systems or logical techniques, such as proofreading or control totals, are frequently used but do not require any special equipment; (2) machine verification.

Machine verifiers are used to check the accuracy of the original recordings. After the input medium has been originally prepared, it is sent to the verifier. Here a second operator keys in the same information as the original operator. This is compared with the recorded data. If there is a disagreement, an indication is given, and the proper steps can be taken to correct the mistake. Machine verifiers can be operated at a rate equivalent to that of the original key-driven equipment.

### **Directly Connected Manual Input Equipment**

There are several manual means of getting information directly into electronic digital equipment, but all these methods depend upon human operations. Since they are therefore relatively slow, they are not used to introduce any large amount of data into the processing. They are used primarily for: (1) control over the computer, (2) basic communications between the operator and the equipment, (3) the insertion of small amounts of input data.

**Consoles.** A console is the primary means of communication between the human operators and the electronic digital equipment. It is a panel with a series of switches, lights, and buttons that are used manually (1) to control the machine, (2) to correct errors, (3) to determine the status of machine circuits, registers, and counters, (4) to determine the contents of storage locations and accumulator storage, and (5) to revise the contents of a storage location.

**Electric typewriters.** Most electronic digital equipment has an electric typewriter associated with it. In a great many cases, the electric typewriter is a part of the console. It is used primarily for input and output of very small amounts of data, correctional information, and control data. Human operator speed is about 8 to 10 characters per second.

**Inquiry Units.** One of the primary drawbacks of electronic storage units is that the information contained therein cannot be visually examined. The information has to be read, interpreted, or translated by the electronic digital equipment, and then printed out. This can be accomplished by the main processor, but such use is inefficient. Because of this, several electronic digital equipments have special input units associated with them that make it possible to search for and print out the information contained in the electronic storage units. These special input units are called inquiry units or interrogation units.

**Special Keyboards.** Special keyboards have been developed for use with particular applications. Examples are the bank savings window machine for use in the savings account application and the reservation agent's set for use in the airline's reservation systems. These special keyboards have been designed, for special applications and are not usable for any other purposes.

### **Punched Card Equipment**

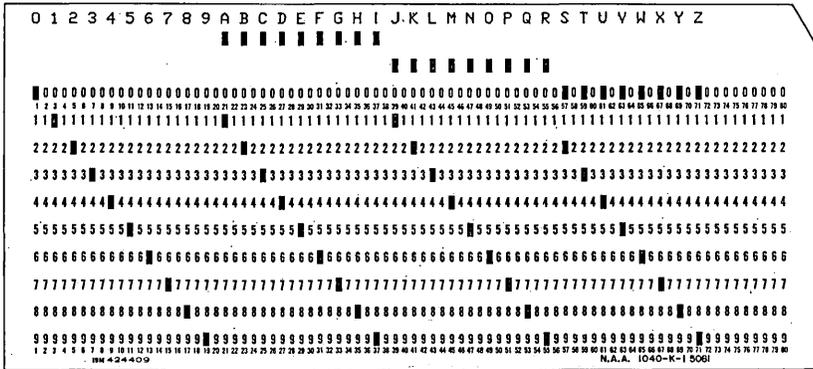
**Kinds of Punched Cards.** The three types of punched cards used are the 80-column card (International Business Machines), the 90-column card (Remington Rand), and the punched price tags. The 80- and 90-column cards are used to feed data directly into electronic digital equipment. The Kimball and Dennison punched price tags are used primarily in the retail business and are not directly fed into electronic digital equipment.

The International Business Machine card is  $3\frac{1}{4}$  inches high and  $7\frac{3}{8}$  inches wide. It is divided horizontally into 80 vertical columns, each capable of storing one character of information. Each vertical column is divided into 12 row positions, numbered from 0 to 9 and 11 and 12. Decimal digits are represented by a single punch in the appropriate row 0 to 9. Alphabetic characters are coded by two punches in the same column; one in one of the rows 1 to 9 and another in either row 0, 11, or 12. See Fig. 5a.

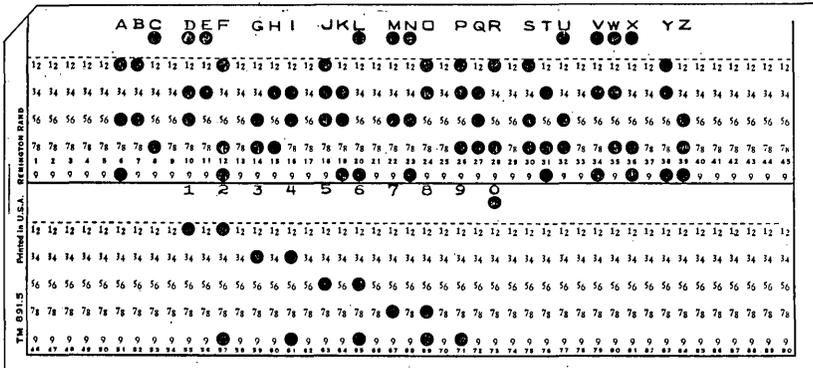
The Remington Rand card is  $3\frac{1}{4}$  inches high and  $7\frac{3}{8}$  inches wide. It is divided vertically into an upper half and a lower half. Each half is divided horizontally into 45 vertical columns, each capable of storing one character of information. This gives a total capacity of 90 columns per card. Information is represented within each column by a combination of one, two, or three punched holes. See Fig. 5b.

The Kimball and Dennison punched price tags are used primarily in the retail trade. The tag is detached from an article when it is sold (see Fig. 5c, d). These tags are then read by a ticket converter and the information contained in the tag is converted to punched cards or punched

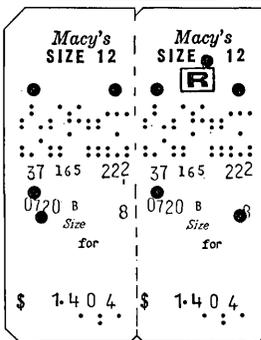
# 5-14 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS



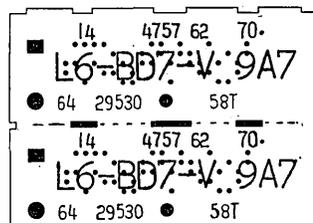
(a)



(b)



(c)



(d)

Fig. 5. Punched cards: (a) IBM punched card (80 columns), (b) Remington Rand punched card (90 columns), (c) Kimball punched tag, (d) Dennison punched tag.

paper tape. Then the punched cards or punched paper tape are used as input media to the electronic digital equipment.

**Preparation Equipment.** Card punches are the basic machines used to prepare punched cards. They operate from a keyboard, either typewriter or ten-key numeric, and permanently record information in cards by means of punched holes. Reproducing punches are capable of producing a duplicate set of cards or many duplicates from a single master card. Punched cards can also be produced as a by-product of another machine operation such as a typewriter or bookkeeping machine. Punched card verifiers are available to check the accuracy of the original punching. Punched price tickets are prepared by a dial set machine.

**Reading Techniques.** There are three basic techniques for reading the information contained in punched cards, namely electric contact, mechanical probes, and photoelectric cells.

In the *electric contact method* the card passes over a contact roller under a set of brushes. These brushes are kept from touching the contact roller by the card, which acts as an insulator. However, when a punched hole is reached, the brush drops into the hole and touches the contact roller. This completes an electric circuit and electric current flows until the contact is broken by an unpunched portion of the card. Information is recognized by the time spacing of the holes from the edge of the card.

With *mechanical probes*, punched cards can be read by moving a set of probes against the card. If there is a hole, then the probes protrude through the hole. The movement of the probe may be used to control other mechanical devices or to generate an electric impulse.

*Photoelectric card reading* is accomplished by moving the punched card under a beam of light. The card acts as an opaque substance until a hole is reached. Then the beam of light travels through the card, impinges upon a photoelectric cell, and thereby generates an electric impulse.

Punched cards are generally read a row at a time. In this manner, all card columns are read simultaneously. Reading speeds are 100 to 900 cards per minute. Punched cards can be read a column at a time by a photoelectric reader. Speeds up to 2000 cards per minute have been achieved with this technique by the National Cash Register Company.

**Input Units.** Punched card readers of various kinds are used as input units for electronic digital equipment. Standard electrical accounting machine equipment such as reproducing punches, summary punches, and collators operate in the range from 100 to 240 cards per minute. Special card readers have been developed for specific digital equipments that operate in the range from 100 to 2000 cards per minute.

## Paper Tape Equipment

**Kinds of Paper Tape.** There are two kinds of paper tape, namely printed and punched.

*Printed paper tape* has the information represented by dot patterns or bar patterns. Printed paper tape is not used as extensively as punched paper tape; however, it does offer several advantages such as compactness and simple, inexpensive, mechanical recorders.

*Punched paper tape* can be either fully perforated or chadless. Fully perforated paper tape has the hole punched completely through. With chadless paper tape the holes are not punched all the way through so that no chad or confetti is produced; this permits legible overprinting. With electronic digital equipment fully perforated tape is most extensively used.

There are four types of fully perforated paper tape, namely the 5, 6, 7, and 8 channel. See Fig. 6. On all these tapes there is a sprocket hole which, when meshed with a small gear in the reading equipment, provides a means of advancing the tape.

**Five-Channel Tape.** The 5-channel punched paper tape has been used since the last century with the telegraph. The 5-channel tape with holes and no holes provides 32 combinations. These 32 combinations are sufficient to designate the 26 letters of the alphabet. Then by using one of the remaining combinations as a line signal to shift, the 32 combinations can be used over again to designate the ten decimal digits, punctuation marks, and other special characters.

*Disadvantages of the Five-Channel Tape.*

1. It does not provide a self-checking channel.
2. In electronic digital equipment it is sometimes advantageous to have the coding on the tape correspond to the coding used to represent information within the electronic equipment.

As a result of these disadvantages, 6-, 7-, and 8-channel punched paper tapes have been developed for use in local or private communications networks and with electronic digital equipment. Punched paper tapes range in width from  $1\frac{1}{16}$  inch to 1 inch, and the holes are usually spaced ten to the inch along the length of the tape.

**Preparation Equipment.** Tape perforators are the basic machines used to prepare punched paper tape. They operate from a typewriter keyboard and record information in the paper tape by means of punched holes. Punched paper tape can also be prepared as a by-product of another machine operation. For example, it is possible to connect the tape perforator to a typewriter, calculator, bookkeeping machine, or cash register designed for such use. Then as these machines are being oper-

ated, selected information can automatically be transferred to punched paper tape. Punched paper tape verifiers are available to check the accuracy of the original recording.

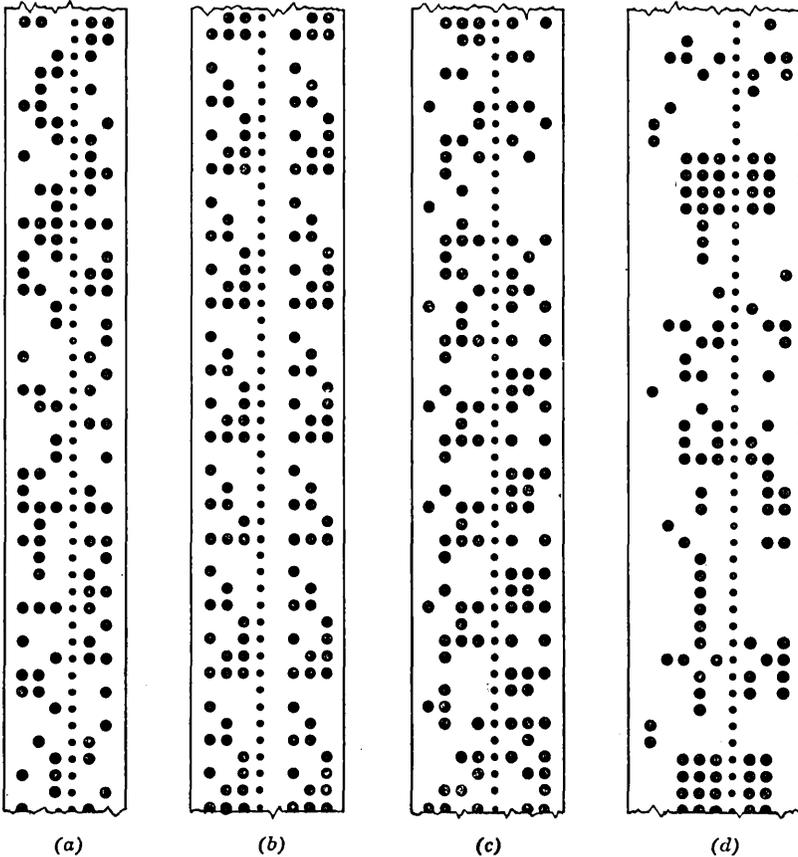


FIG. 6. Punched paper tape: (a) 5-channel, (b) 6-channel, (c) 7-channel, (d) 8-channel.

**Reading Techniques.** Punched paper tape can be read by *mechanical* means. A set of probes, one for each channel, is moved against the tape. If there is a hole in the tape, a probe protrudes through the hole and an electric impulse is generated. If there is no hole in the paper tape, the probe does not protrude and no electric impulse is generated. Since this is a mechanical process, it operates at relatively slow speeds of about 60 to 100 characters per second.

A *photoelectric system* can also be used to read the holes in punched

## 5-18 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

paper tape. Light from a single source is projected through the holes in the paper tape onto a series of photocells. If there is a hole, then light is projected on that channel's photocell and an electric impulse is produced. If there is no hole, no electric impulse is produced. Reading speeds are in the range of 200 to 2000 characters per second. All punched paper tapes are read a character (column) at a time, so that reading speed is proportional to linear tape speed.

Brush contact readers such as those used for punched cards have not been developed for paper tape.

**Input Units.** Several types of punched paper tape readers are used as input units for electronic digital equipment. Most electric typewriter-perforators are also mechanical punched paper tape readers. They operate at a maximum of 10 characters per second. There are several types of photoelectric punched paper tape readers. They operate in the range from 200 to 2000 characters per second.

### Magnetic Tape Equipment

**Kinds of Magnetic Tape.** Magnetic tape is a strip of either plastic or metal that is coated with a ferromagnetic substance. See Magnetic Tape under Sect. 4 for equipment description.

**Preparation Equipment.** Magnetic tape used as an input is generally prepared by a process of converting information from another medium such as punched cards or punched paper tape to the magnetic tape. The punched cards and punched paper tape are verified before the data are placed on magnetic tape.

A typewriter keyboard device has been developed that directly places information onto magnetic tape as it is keyed onto the keyboard. A magnetic tape verifier operates on the same principle as the punched card verifier. The primary advantage of directly produced magnetic tape is the elimination of the conversion operation. The disadvantages are that the recording density is much less than on magnetic tape produced by other methods, and there is no off-line method of producing batch totals often used in checking the accuracy of input data.

**Reading and Recording Techniques.** Information is usually recorded on magnetic tape in a series of records or blocks as shown in Fig. 7.

The size of the records can be fixed or variable, depending upon the specific electronic digital equipment. The records can have specific addresses or the identifying labels of the information stored in each record can serve as distinguishing marks. The gaps between the records serve as a nonusable area to allow for starting and stopping of the tape. When the tape is motionless, the reading and recording head rests over the middle of the gap. One-half the length of the gap is required to

accelerate the tape motion up to full reading or recording speed. Also, one-half the length of the gap is required for the tape to decelerate to a complete stop.

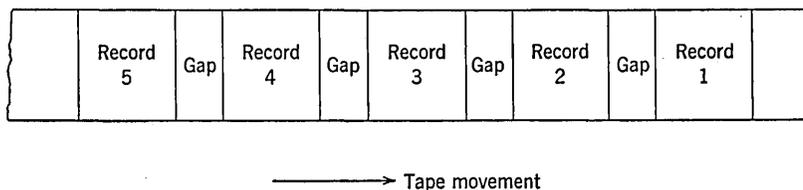


FIG. 7. Recording on magnetic tape.

Reading and recording are accomplished by moving the tape past the stationary head (see Chap. 19). Binary digits are recorded in the range 50-534 per linear inch. The magnetic tapes can be transported up to several hundred inches per second.

**Input Units.** Magnetic tape units contain the magnetic tape reels, one for feeding and another for take-up, the reading and recording heads, and the motors required to move the tape. All magnetic tape units contain some system to prevent tension from being placed upon the magnetic tape during its sudden starts and stops. Magnetic tape units operate in the range from 6000 to 60,000 characters per second and can be accelerated to full speed in 2 to 50 milliseconds. A common size of tape reel is 10 inches in diameter, and the reel holds 2400 feet of tape. For additional information on tape handling mechanisms, see Sect. 4.

### Document Reading Equipment

Electronic digital equipment can also accept input data in the language of the business world as it is recorded on the original documents. This eliminates the battery of typists needed to translate the input information from the source documents to the input medium. Automatic character recognition techniques make this possible. These techniques make use of either special codes to represent the characters, or they can read the printed characters themselves. These special codes or printed characters are recorded on the original documents themselves.

Character recognition techniques can be classified as:

1. Codes or patterns: (a) mark sensing, (b) electrically conductive spots, (c) fluorescent ink spots, (d) magnetic bar codes.
2. Printed characters: (a) printer's ink, (b) magnetic ink.

**Codes and Patterns.** *Mark sensing* uses special electrical conducting marks placed in various positions on punched cards. These marks are



Some of this reading equipment is directly connected to the electronic digital equipment. Other reading equipment produces other input media, such as punched cards or punched paper tape, which are used to transfer the information into the electronic digital equipment.

**Printed Characters.** It would be advantageous for a machine to read not codes but conventional alphabetic characters and arabic numerals. Characters and numbers are easier to print, are visually auditable, and do not require additional space (for codes and patterns) on the document. The following techniques have been developed to read and interpret printed characters.

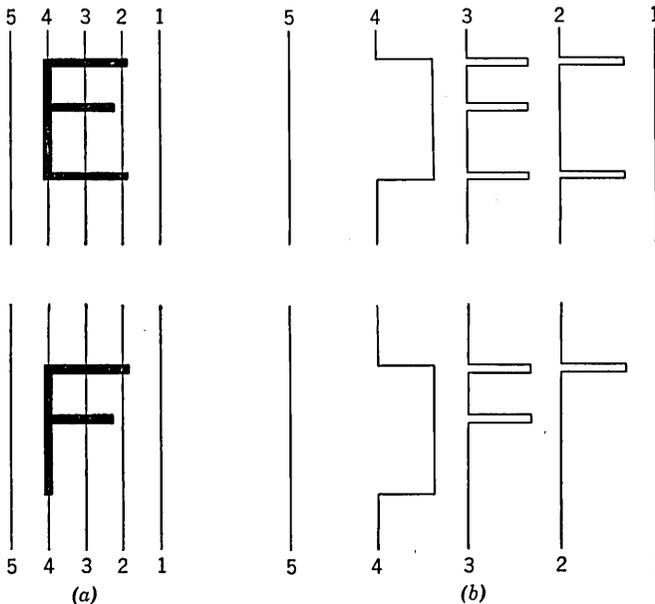


FIG. 9. Photoelectric reading of printed characters: (a) five scanning lines, (b) pulse output of scanning circuits.

With *printer's ink*, information is recorded on the document in the normal fashion. Photoelectric scanning techniques have been developed to read and interpret the printed characters. As the characters move past the reading station, they are continually scanned by a beam of light. A photoelectric cell is used in the operation, and the output of this cell is a pattern of pulses. Each character will produce its own unique pattern of pulses. See Fig. 9.

*Magnetic ink* has an advantage over printer's ink whose primary disadvantage is obliteration and mutilation. Extraneous markings are

## 5-22 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

sensed and cause incorrect interpretations of the characters. Some of this mutilation disadvantage can be overcome by using magnetic ink because the printing can be performed when the document is originally prepared or the magnetic ink characters can be recorded at a later time by special ribbons or carbon paper. The characters printed with magnetic ink are read by passing the document beneath a magnetic reading head which produces an electric signal that is unique for each different character. See Fig. 10.

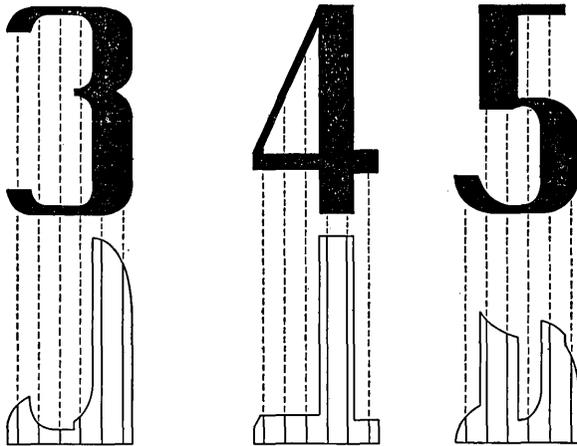


FIG. 10. Magnetic reading of printed characters. As the printed numbers pass horizontally beneath the read head, the head sums up the total magnetic ink covered in a given time interval and produces a proportional electric signal. Signal forms for several numbers are shown.

Special reading equipment is required to read either printer's ink or magnetic ink. Either it can be connected directly to the electronic digital equipment, or it can be used to prepare another input medium, such as punched cards.

### Auxiliary Input Equipment

In business data processing applications, large amounts of input data generally have to be placed into the processing. Since the input operations are generally much slower than the internal operations of the electronic digital equipment, the machine will be tied up for long periods of time while it performs the time-consuming operation of inserting the data. This utilizes the electronic equipment's abilities inefficiently. Several things have been done to speed up the input operations.

**Conversion Equipment.** Conversion is the process of transcribing information from one input medium to another. The primary purpose for the transcription is to take advantage of the inherently higher speeds of certain input media (see Table 2). This is the case where information on punched cards, punched paper tape, or paper documents is converted to information on magnetic tape. These transcription operations are performed automatically by special machines, and they are performed independently of the main electronic digital equipment.

Transcription of information from one medium to another is often performed for other reasons, such as compatibility or ease of preparation. For example, punched cards to punched paper tape conversion is required for transmission over a telegraph system. In other cases, punched paper tape to punched cards conversion is required for computer input. Punched cards to magnetic tape and punched paper tape to magnetic tape conversions are used because in a great many cases cards and paper tapes are easier to prepare and verify than the magnetic tapes.

**Buffering Equipment.** Most input devices handle data at a much slower rate than the electronic digital equipment does. Since, in the normal chain of processing, each component normally must wait before beginning its operations until the preceding operations performed by other units have finished, the overall speed of the processing is considerably reduced by the input operations. To overcome this drawback, buffer storage devices are used. They permit internal processing operations to be performed during input operations. Thus, by enabling all the components to operate simultaneously, buffer storage devices reduce waiting times and increase the overall speed of the data processing.

The buffer has some or all of the following features:

1. It is usually capable of operating at two speeds, slow, dealing with input-output, and fast, dealing with computer.
2. It serves as a kind of synchronizer, by taking over from the control unit the control functions associated with input.
3. It can assist in rearranging data. For example, a word can go into the buffer most significant digit first and come out least significant digit first.
4. Code conversion from language of input medium to language of digital equipment can be performed. Additional equipment is required in this case.

Many buffers have some computing capability.

**Multiplexing Equipment.** Another way of speeding up the input of information into electronic digital equipment is to connect multiple input units to the machine. Each one of these units has its own buffer associated with it. Then by having multiplexing equipment, it is pos-

## 5-24 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

sible to connect these various input units to the central computer in a definite sequence. While one input unit is connected to the computer, the remaining units are free to be loading their input buffers. The multiplexing unit scans the various input devices, searching for one that is ready to transmit to the central electronic equipment. When it finds one, the computer is connected to that input device. As soon as the necessary information has been transferred, the multiplexing unit searches for another input unit that is ready to transmit to the central equipment. Multiplexing equipment is very important with in-line equipment having many manual input stations.

### 4. STORAGE EQUIPMENT

#### Types of Storage

**Internal.** Internal storage forms an integral physical part of the equipment and is directly controlled by the equipment. It holds the control instructions which direct the processing, the data to be processed, the intermediate results of the processing, and the final results of the processing. Internal storage usually consists of one or both of the following types of storage: (1) fast access, low capacity storage, such as magnetic cores; (2) medium speed access, medium capacity storage, such as magnetic drums.

**External.** External storage is divorced from the equipment itself but holds the information in the form prescribed by the electronic digital equipment. It is a bulk storage medium used for long-range filing of reference data. External storage must be low in cost because of the relatively large volumes required. It is usually a long access, large capacity type such as magnetic tape. Punched cards and punched paper tape can also serve as external storage media, but their use in this capacity is relatively restricted.

#### General Characteristics of Storage Devices

1. *Capacity.* The upper limit of information that can be stored within the storage unit.

2. *Access time.* The time interval between the instant at which information is called for from the storage location and the instant at which delivery is complete. There are three types of access to information, namely random, sequential (or serial), and a combination of these two. Random access implies an access time which is constant, regardless of storage location. In equipment with this characteristic, it is possible to move from one randomly selected storage location to another randomly chosen storage location in the same amount of time as is required to go

from one random storage location to its adjacent storage location. Sequential access occurs when it is necessary to go sequentially through all intervening storage locations to get from one storage location to another. As a result of this, sequential access time is directly proportional to the number of storage locations between the starting storage location and the ending storage location. A combination of random and sequential access is obtained when it is possible to have random access to a particular group of information, but then it is necessary to proceed sequentially through that group to locate the particular piece of information desired.

3. *Volatility.* Nonpermanence of the recording when the applied electric power is cut off.

4. *Erasability.* Capable of being erased and reused.

5. *Physical.* Size, shape, heat generated, power required, sensitivity, etc.

6. *Cost.* Cost per unit of information stored.

### Spectrum of Electronic Storage Devices

Table 3 indicates the characteristics of various types of storage devices used in electronic digital equipment. Some general relations can be observed from this table:

1. *Capacity and access time.* As capacity increases the access time also increases.

2. *Capacity and cost.* As capacity increases the cost per unit of information stored decreases.

3. *Access time and cost.* As access time increases the cost per unit of information stored decreases.

**Operation and Uses.** Table 4 describes the various storage devices, principles of operation, and their uses. See Chap. 19 for details of design and operations of magnetic drums and cores. Magnetic tapes and random access devices are discussed below.

### Magnetic Tape

**Recording.** Magnetic tape is a thin, nonmagnetic base material coated with a magnetic material on which information is recorded. See Chap. 19 for details on magnetic coatings. The base material can be either a metallic ribbon or a plastic (acetate or Mylar) ribbon. Magnetic tape is usually several thousandths of an inch thick, from  $\frac{1}{2}$  to 3 inches wide, up to 2400 or 2700 feet in length, and wound on 10-inch diameter reels.

Information is recorded on magnetic tape in a series of records or blocks shown in Fig. 7. Tape records have the following features.

TABLE 3. SPECTRUM OF ELECTRONIC STORAGE DEVICES

Storage Device	Representative Capacity	Representative Access Time	Volatile	Erasable	Physical Characteristics	Cost per Unit <sup>a</sup>
Vacuum tubes and transistors	50-100 binary digits	Random 0.000001 sec	Yes	Yes	Electronic circuits	\$60 per character
Magnetic cores	10,000-100,000 characters	Random 0.00001 sec	No	Yes	Small cores, large units, complicated circuits	\$7 per character
Electrostatic tubes	5,000-50,000 characters	Random 0.00001 sec	Yes	Yes	Very sensitive, delicate, complicated circuits	\$7 per character
Mercury delay lines	5,000-12,000 characters	Combination av. 0.001 sec	Yes	Yes	Sensitive, bulky	\$20 per character
Magnetic drums	20,000-2,000,000 characters	Combination av. 0.01 sec	No	Yes	Large size, very reliable	\$0.1 per character
Random access devices	5,000,000-75,000,000 characters	Combination av. 0.5 sec	No	Yes	Large size, complicated mechanical features	\$0.01 per character
Magnetic tapes	2,000,000-24,000,000 characters per reel	Serial 0.01 sec Random 2 min	No	Yes	Compact, lightweight removable	\$0.01 per character (attached) \$0.00001 per character per reel

<sup>a</sup> Based on 1958 costs.

TABLE 4. STORAGE DEVICES

Storage Device	Principle of Operation	Use
Vacuum tubes and transistors	Bistable circuits	Registers, temporary storage
Magnetic cores	Two opposite states of magnetization of cores	High-speed internal storage
Electrostatic tubes	Positively or negatively charged spots of dielectric tube face	Early high-speed internal storage
Mercury delay lines	Ultrasonic sound pulses traveling in a column of mercury	Early high-speed internal storage
Magnetic drums	Rotating cylinder, magnetic coating, areas of magnetization	Medium-speed internal storage
Random access devices	Jukebox—many magnetic disks Tape strips—open or closed loops Multiple magnetic drums	Large capacity storage under computer control
Magnetic tapes	Magnetic coated tape	Usually external bulk storage, computer input

1. The size of the records can be fixed or variable, depending upon the electronic digital equipment.

2. The records can have specific addresses or the identifying labels of the information stored in each record can serve as distinguishing marks.

3. The gaps between the words are unused areas that are required for starting and stopping the tape.

4. When the tape is motionless, the reading and recording head rests over the middle of the gap.

5. One-half the length of the gap is required to accelerate the tape up to full reading and recording speed. Also, one-half the length of the gap is required for the tape to decelerate to a complete stop.

Operation details are as follows.

1. Information is recorded within the records by magnetizing small areas. Each such area can represent a binary digit.

2. These small areas are magnetized by a reading and recording head, which is a tiny electromagnet. See Chap. 19.

3. When reading information stored on magnetic tape, this same reading and recording head is used to determine the magnetic state of these small areas.

## 5-28 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

4. Reading and recording are accomplished by moving the tape past the stationary reading and recording head.
5. Binary digits can be recorded with densities up to many hundred per linear inch.
6. The magnetic tapes can be transported up to several hundred inches per second.
7. The time required to accelerate the tape from standstill to full speed ranges from several milliseconds to 50 milliseconds.
8. A number of channels are recorded across the tape. See Fig. 11.

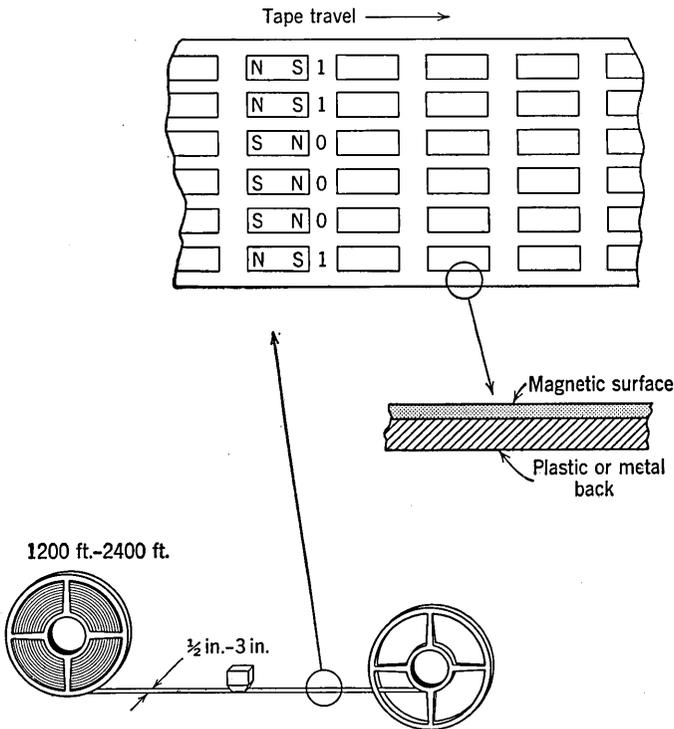


FIG. 11. Recording channels on magnetic tape.

**Capacity.** The size of the interrecord gap has to be considered when determining the capacity of one reel of magnetic tape.

**EXAMPLE 1.** Consider a magnetic tape, 2400 feet long, with recording density of 200 characters per inch and a  $\frac{3}{4}$ -inch interrecord gap. Each reel has a theoretical capacity of:

$$\begin{aligned} 2400 \text{ feet} &= 28,800 \text{ inches,} \\ (28,800 \text{ inch})(200 \text{ characters/inch}) &= 5,760,000 \text{ characters.} \end{aligned}$$

However, it is necessary to take into consideration the size of the inter-record gap. If the information to be stored is a fixed record of 80 characters, the capacity of one reel of magnetic tape is developed as follows:

$$\begin{aligned} \text{Record length} &= \frac{80 \text{ characters}}{200 \text{ characters/inch}} = 0.40 \text{ inch} \\ \text{Interrecord gap} &= \underline{0.75} \\ \text{Record plus gap} &= 1.15 \text{ inches} \end{aligned}$$

$$\frac{28,800 \text{ inches}}{1.15 \text{ inches/record}} = 25,000 \text{ records,}$$

$$(25,000 \text{ records})(80 \text{ characters/record}) = 2,000,000 \text{ characters.}$$

**EXAMPLE 2.** If the fixed record contained 600 characters, the capacity of one reel of magnetic tape would be 4,620,000 characters contained in 7700 records. Record length = 3 inches, gap = 0.75 inch.

In general, the larger the size of the record, the greater amount of information that can be stored per reel of tape because of the proportionally smaller space wasted with the interrecord gaps.

**Speed.** Start-stop time has to be considered when determining effective tape speed.

**EXAMPLE 1.** Consider a magnetic tape with start-stop time of 10 milliseconds and a transfer rate of 15,000 characters per second. For a fixed record of 150 characters, then the effective tape speed is:

$$\frac{150 \text{ characters}}{15,000 \text{ characters/record}} = 0.01 \text{ second to read}$$

$$\text{Start time} = \underline{0.01}$$

$$\text{Time required per record} = 0.02 \text{ second}$$

$$\frac{150 \text{ characters}}{0.02 \text{ second}} = 7500 \text{ characters/second}$$

**EXAMPLE 2.** If the fixed record contained 600 characters, the effective tape speed would be 12,000 characters per second. Time to read = 0.04 second, and start time = 0.01 second.

In general, the larger the size of the record, the greater the effective tape speed because of the proportionally less time wasted in accelerating the tape to full speed. The total time required to read the length of a reel ranges from 3 minutes to 10 minutes.

**Tape Handling Mechanism.** The tape transport mechanisms must on demand be able:

## 5-30 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

1. To start the tape motion rapidly,
2. To maintain a uniform speed over the heads,
3. To stop the tape,
4. To maintain continuously proper tape guidance and control.

Other requirements on the equipment are:

5. The tape bearing surfaces must be smooth and free of sharp edges to reduce wear to a minimum and protect the tape from tears.
6. The read-write heads of all tape mechanisms must be aligned to a standard so that tapes written on one tape mechanism may be mounted and read by the other tape mechanisms.

Operation is as follows:

1. A typical tape mechanism consists of a feed reel and a takeup reel which can hold the entire length of magnetic tape.
2. The tape is threaded from the feed reel through some kind of surge tank containing a few feet of tape, over a reverse tape drive capstan, over the read-write and erase heads, over a forward tape drive capstan, through a second surge tank and onto the takeup reel. See Fig. 12.
3. The tape drive capstans rotate continuously in opposite directions, always ready to drive the tape when a pressure roller forces the tape against one of the capstans.
4. The surge tanks are used to isolate the relatively slow acting reel drives from the capstan drives which must actuate tape motion as rapidly as possible.
5. When the amount of tape in a surge tank becomes either longer or shorter than the nominal amount, the associated reel servo is signaled by vacuum or photoelectric sensing to adjust the tape in the surge tank accordingly.

**Use of Magnetic Tape for Storage.** Practically all modern business data processors have adopted the magnetic tape for large capacity storage.

### *Advantages*

1. By using many reels of tape, it is possible to store almost any desired amount of information.
2. It has proved to be a very reliable storage medium.
3. It has made possible the solution of a great many business applications where random access to the file information is not an absolute requirement.

### *Disadvantages*

1. Access to items on the tape is sequential and hence extremely slow compared with internal operating speeds. It is more efficient to run through a file and update it in a sequential fashion rather than a random fashion.
2. Updating the files also presents some problems. To modify an entry

on magnetic tape without rewriting the entire tape, it is necessary to read the information from the tape, and, while the entry is being modified, back-space the tape so that the modified record can be recorded back into the original space. This is a very time consuming operation.

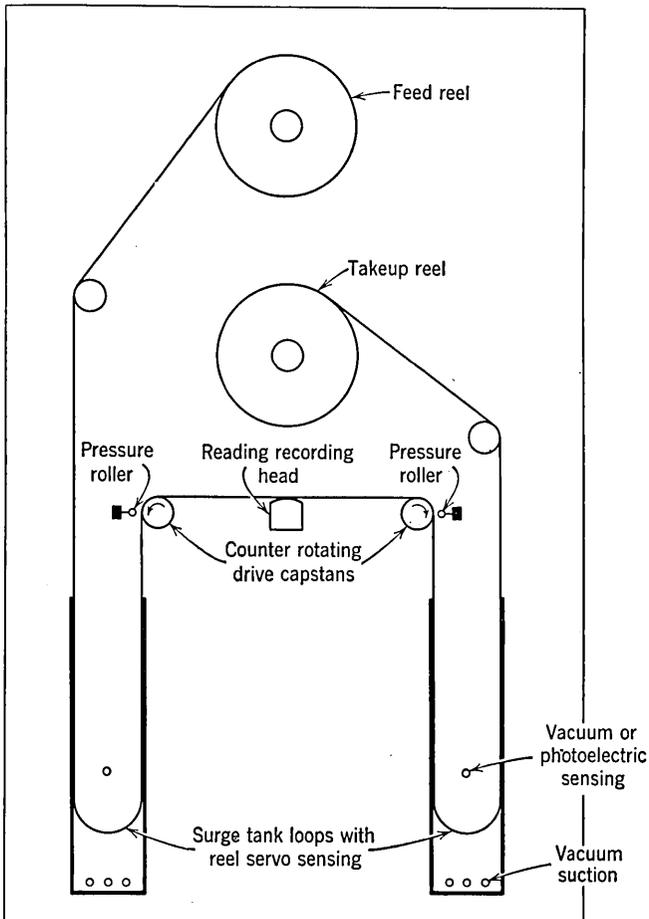


FIG. 12. Typical magnetic tape handling mechanism.

To add a new entry to an ordered file it is necessary to provide empty spaces between each original entry in the file, or else rewrite the entire file and put the new entry in its correct location. Neither of these possibilities is attractive. Leaving blank spaces in the original file means that storage space is wasted with no guarantee that the blank spaces will be properly arranged for the additions that have to be made. Re-

## 5-32 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

writing the entire file to make additions and changes is costly and introduces the possibility of errors.

### **Random Access Devices**

In most business applications, very large capacity storage units are required to maintain the files of business records. The most economical method of storing this information is magnetic tape, which has a very long average access time. To overcome this serious drawback, random access devices have been developed. These devices have large storage capacities and relatively short average access times. (See Chap. 4.)

These devices do not have full random access ability (see above). In general, they have an average access time in the range of one-half to several seconds. However, the term *random access* has been associated with these devices and is commonly accepted.

**Large Capacity Magnetic Drums.** Several types of magnetic drums have been developed with a storage capacity of up to several million characters per drum. Many drums can be connected to the electronic digital equipment to bring the total storage capacity up to the order of 100,000,000 characters. These drums operate on the same principles as the regular magnetic drums. They have an average random access time in the range from 0.01 to 0.05 second.

**Magnetic Disks.** Magnetic disks are flat circular shaped records that have a coating of magnetic material on both faces. These disks are mounted on a shaft. A movable reading and recording arm is used to read and record information on both sides of the disk. These units look and operate like a jukebox (see Chap. 19). It is possible to store up to 5,000,000 characters of information in one magnetic disk unit. The average random access to information is on the order of 0.5 second.

**Magnetic Tape Devices.** Several devices have been constructed to use magnetic tape in a different manner so that the average random access time to information stored in these devices is in the range from 1 to 20 seconds. One device makes use of strips (250 feet) of magnetic tape. There is a series of bins with one strip of tape in each bin. A movable reading and recording head can be moved from bin to bin. Then the strip of magnetic tape within that particular bin is moved, in either direction, to the particular storage location desired.

Another device makes use of short strips of magnetic tape mounted on movable frames. These frames are stored within a bin. They are selected from the bin and brought against a reading and recording head. After reading or recording, the frames are returned to their bin.

## 5. OUTPUT EQUIPMENT

The output unit connects the electronic digital equipment with the external world. It delivers the results of the data processing in a form that can be used outside of the equipment. It converts data from the form of discrete electric signals used inside the equipment to a form required by the output medium. Output devices are compared in Table 5.

TABLE 5. OUTPUT DEVICES

Output Media	Equipment	Speeds
Low speed		
Punched cards	Reproducing punches	100-200 cards/min
Punched paper tape	Flexowriter, teletype punch, etc.	6-60 characters/min
Printed page	Electric typewriter	6-12 characters/sec
Visual display	Mechanical annunciators	Audio alarm
	Electric picture tube	several words/sec
Plotters	Analog plotters	Depends on equipment (see Chap. 23)
	Typewriters	6-12 points/sec
Line printers		
Printed page	Accounting machine, wheel or bar printer	100-150 lines/min
Printed page	High-speed wheel	300-900 lines/min
Printed page	High-speed matrix	500-1000 lines/min
Magnetic tape	Tape units	6000-60,000 characters/sec
Electronic printers		
Photographic copy } Xerography copy }	Picture tube	Up to 20,000 characters/sec

### Types of Output

**Direct and Indirect.** Output from electronic digital equipment is either direct or indirect. The direct manner produces data in a form that is directly usable, e.g., printed data from an electric typewriter or line printer. The indirect manner produces data in a form that is not directly usable, e.g., punched cards or magnetic tape. In this case, the information on the output medium is usually reproduced into a usable form independent of the main equipment. The information on paper tape is usually reproduced into a report form by an electric typewriter whereas the information on punched cards and magnetic tapes is reproduced by line printers.

## 5-34 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

**Reports.** The primary means of communicating the results of the data processing with human beings is the printed report. These reports contain results of the processing for use by management, employees, operators, and users of electronic digital equipment. In business applications, these reports can be classified as legal records, management information reports, and operating results. In scientific and engineering applications, these reports will generally be in the form of tables or graphs.

**Machine Readable Outputs.** In many applications, the output of one processing cycle is required as input to another. When this occurs, it is important to record the results of the first processing cycle in machine readable form that can be used automatically in the next processing cycles. This can be achieved by producing the results on machine readable media such as punched paper tape, punched cards, and magnetic tape. Then the results can be used as automatic input to the next processing cycle and can be used to produce printed reports.

### Low-Speed Outputs

Several slow-speed methods are available for output. These methods have operating speeds in the range from 10 to 200 characters per second. The equipment required with these methods operates at these low speeds because it is primarily mechanical in its operation. Table 5 shows a comparison of the various low-speed outputs. Punched card machines are standard electric accounting machines. Punched paper tape data can be printed out by an electric typewriter. Electric typewriters are used primarily for low volume output and operating instructions. Volatile visual displays are useful where temporary and short-term indication of limited amounts of data is required. Electrically actuated mechanical annunciators are used on supervisory and monitor consoles, as well as electronic picture tubes which can be used to display rapidly several words such as the contents of a specific storage location. Graph plotters eliminate human transcription of the results from digital to graphical form. Digital-to-analog converters are required for analog plotters.

### Line Printers

Line printers may be defined as those that print an entire line at one time. The line may be composed of from 25 to 120 printed positions spaced approximately 10 characters per inch and with approximately six lines per inch vertically. These printers may also skip blank lines (slew) at 3 to 5 times their line printing speed. Some printers have a means for horizontal and vertical format control through plugboards and paper tape control loops located on the printer. In other printers,



## 5-36 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

The *wheel printer* utilizes a continuously spinning printer wheel with the characters embossed on the periphery and a printing hammer which is timed to strike the paper against the ribbon and the wheel as the desired character passes. Each printing wheel must contain each character to be printed. Most wheel type printers utilize a rotating drum containing a "wheel" of print for each print column, such as 120 wheels for printing a line containing 120 character positions. See Fig. 13b.

In the *wire matrix printer*, the characters are formed by selectively actuating appropriate wires in a rectangular matrix usually containing five columns of seven wires each. The printed character then is an array of dots which is readable but is not as plain as the characters printed from a wheel type printer. See Fig. 13c.

### **Magnetic Tape**

Magnetic tape also can be used as an output medium. The main advantage of using it arises from its much higher speed compared with low-speed media or line printers. With these other media, the maximum possible output speed is about 2000 characters per second, but with magnetic tape output speed is in the range from 6000 to 60,000 characters per second. See Table 5. This means that when magnetic tape output is used, the ratio of output time to computing time is lower than when other mediums are used.

### **Electronic Printers**

**Principles of Operation.** Electronic printers are those in which the character image is generated in the form of an electron beam with a cross section in the shape of the desired character or is generated by line scanning electron beams such as in television. These devices are capable of translating binary coded information used within the electronic digital equipment into characters and displaying them on the face of a cathode ray tube. In the charactron tube, this is accomplished by directing an electron beam through a miniature matrix which contains cutout letter and decimal digit shapes. The electron beam is then deflected into the proper position on the face of the picture tube and thus creates an image in the shape of the desired character. See Fig. 14.

**Producing Hard Copy.** The picture tube may be directly viewed. However, most applications require the results to be produced in report or other printed form. One method of producing a printed report uses a photographic process in which pictures of the face of the tube are taken. In some cases, the film can be developed by darkroom techniques, while in other cases the film is exposed, developed, and fixed automatically.

Another method of producing hard copy of the face of the tube utilizes

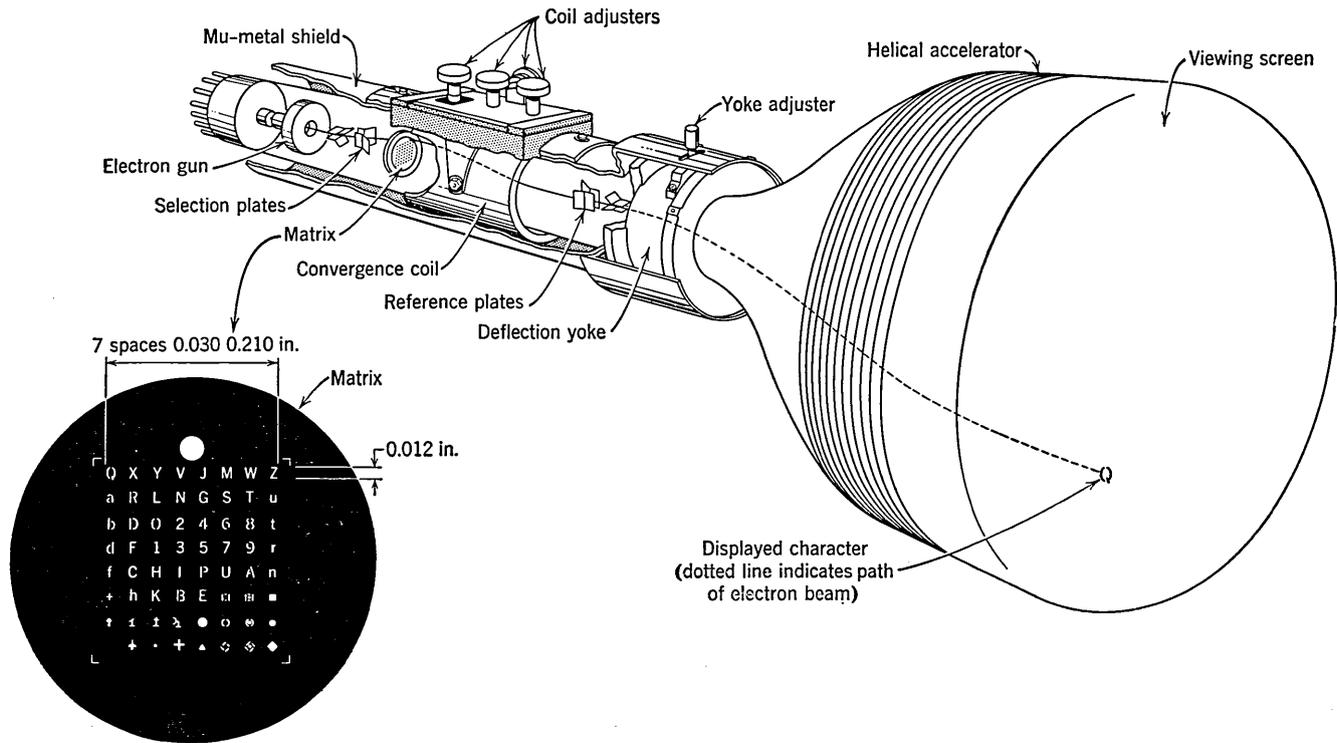


FIG. 14. Charactron shaped beam tube. (Courtesy of Stromberg-Carlson, a division of General Dynamics.)

## 5-38 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

an xerographic printing process. This process uses an electrostatic method whereby tiny, dustlike particles are attracted and adhere to the surface of a selenium-coated drum that has been exposed to the face of the tube. A subsequent step, after these particles have been transferred to paper, fuses the character shapes permanently to the paper by means of heat and pressure.

These electronic printers operate much faster than electromechanical printers. They can operate as high as 20,000 characters per second.

### **Auxiliary Output Equipment**

In business data processing applications, extensive amounts of output data are generated. Since the output operations are much slower than the internal operations of electronic digital equipment, the ratio of output time to computing time may be too high. This is inefficient utilization of the electronic equipment. Several things have been done to speed up the output operations.

**Conversion Equipment.** Conversion is the processing of transcribing information from one output medium to another. (See Sect. 3.) Conversion equipment such as magnetic tape to card and tape to printer is used to take advantage of the inherently higher speeds. Because of the speeds at which information can be put out on magnetic tape, the electronic digital equipment is not tied up for as extensive a period of time during the output functions and thereby has more time available for data processing.

**Buffer Equipment.** The advantages of buffering equipment in performing output operations are similar to the advantages of buffering in input operations, as described above under Input (see Sect. 3).

## 6. ARITHMETIC AND LOGIC UNIT

**Description.** The arithmetic and logic unit performs the arithmetical and logical operations necessary in the processing of data. It has a set of basic operations that it can perform, such as addition, subtraction, multiplication, division, shifting, comparison of two numbers, and the transfer of data to and from the storage unit. Design of arithmetic and logic units is covered in Chap. 18. This section considers equipment aspects.

**Arithmetic.** Both serial and parallel adders are used. The advantage of the parallel mode arises from its very fast speed. For example, to add two pure binary numbers, each 36 binary digits in length, in the serial mode would require about thirty-six times as long as it would to add two one-digit binary numbers. In the parallel mode, it would require only two to four times as long, depending upon the internal design of the

equipment, thereby making a substantial savings in time. As a result of this, most large-scale electronic computers of today operate in the parallel mode.

Electronic digital equipment that codes its information in one of the binary coded representations performs addition in a combination of the serial and parallel modes. Consider the binary coded decimal number representation. In a mode of operation known as the serial-parallel mode, commonly called serial, there are four adders that operate in parallel on the bits of each binary coded decimal digit. Then the binary coded decimal digits are added successively in time by the same set of four adders. In the parallel-serial mode, there is one adder for each binary coded decimal digit, the four binary digits representing each binary coded decimal digit being added successfully through time by the one adder.

Subtraction is usually performed by adding the complement. Multiplication is usually performed by making repeated additions. Division is performed in electronic data processing equipment by the combined operations of subtracting, shifting, and counting the number of times subtraction is possible before shifting is necessary.

**Logical Operations.** The comparison of two numbers, which is an example of a logical operation, may be performed by subtracting one number from the other, and determining whether the remainder is zero or not. As a result of this determination, the equipment can choose between two alternative courses of action, and in effect can make a logical decision. Operations of this type are often called logical operations.

**Alphabetic Information.** The handling of alphabetic and special character information is accomplished in two ways.

1. In the large electronic data processors, alphabetic information is handled by a six binary digit coded character representation similar to the one presented earlier. When this type of code is used and the equipment is to treat the character as a decimal digit, the first two binary digits are ignored in arithmetic operations. The remaining four binary digits are then treated as a binary coded decimal digit and are operated upon in a manner entirely analogous to the operations upon a binary coded decimal digit.

2. Another way of handling alphabetic information is to use two decimal digits to designate one letter or special character. For example, 61 would represent A and 62 would represent B.

**Special Operations.** The requirements of a great many scientific and engineering applications are such that special operations, such as double precision, automatic floating point, and base counter or tally are built into the equipment. These operations are used either to increase

## 5-40 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

the accuracy of computations or facilitate logical operations. They may also be programmed.

### 7. CONTROL EQUIPMENT

**Description.** The control unit receives the instructions from the storage unit, interprets them, and directs the components to perform the appropriate operations. It can direct the arithmetic and logical unit to manipulate the data and the input-output units to communicate with the outside world. A control unit makes the electronic digital equipment accomplish what the human programmer wishes the equipment to do, within the limits designed into the equipment.

**Instructions Classified by Functions Performed.** There are three basic functional groups of instructions required in the operations of electronic digital equipment, namely arithmetic, data movement, and decision. In each of these groups, there are several general instruction requirements, shown in the Table 6.

**Sequencing of Instructions.** The sequencing of the control instructions can be accomplished in two ways, namely by an instruction counter or by the previous control instruction.

#### Control Console

The control console is a panel composed of switches, lights, and buttons that provides the human operators with the means of communication and controlling the electronic digital equipment. The control console includes indicators and controls for power, maintenance, monitoring, and operations.

**Power.** Controls are available for turning a-c power onto the power supply and then turning the various d-c levels onto the computer circuits in the proper order. There are meters for monitoring the voltages and other controls for changing the applied voltages during checking procedures.

**Maintenance.** There are logical circuit indicators, alarm lights, and specialized operating controls which are helpful in preventive maintenance as well as in trouble shooting when the machine stops on account of some alarm condition or malfunction. Examples of such controls are switches which can be set to stop the computer on specified instructions, and others that permit the equipment to execute only one cycle of operation for each push of the starter button.

**Monitoring.** Indicators that show the contents of internal storage locations, principal registers, and counters are provided. In addition, there is a monitor printer (usually an electric typewriter) which can make periodic reports on the progress of an operation.

**Operations.** Start, stop, and off buttons.

TABLE 6. BASIC INSTRUCTION TYPES AND REQUIREMENTS

Order Type	Functions	Required Address Information (Explicit or Implicit)
Arithmetic		
Add, subtract, multiply, divide	To perform basic arithmetic operations	Storage location of operands
Shift	To move the digits of a number right or left within a word	Direction of shift and the number of digit positions to shift
Data Movement		
Read in	Movement of data from input media or external storage to internal storage	Identification of input unit or external storage unit, amount of data and internal storage location to place data in
Transfer data	Selective rearrangement within internal storage	Storage location of data to be transferred, amount of data, final storage location of data
Clear	Internal destruction of data no longer desired	Storage location of data to be cleared
Write out	Movement of data from internal storage to external storage or output media	Identification of external storage or output unit, amount of data, and internal storage location from which to write out
Decision		
Compare	Two words for equality, greater than or less than	Storage locations of two words and location of next instructions for each of three possible outcomes
Conditional transfer of control	Transfer of control based upon prescribed conditions <sup>a</sup>	Storage locations of next instruction for each of the possible prescribed conditions
Unconditional transfer of control	Transfer of control under all conditions	Storage location of next instruction
Stop	Halt all machine operations	No address information required

<sup>a</sup> Conditional transfer of control refers to choosing one of two or more logical paths in a program based upon the condition of a machine calculated result or of a data word supplied as input for processing. For example, when two identification numbers are subtracted for matching, one of three conditions will exist: (1) result zero if identifications are equal, (2) result positive, (3) result negative. Conditional transfer orders either jump to another sequence of instructions or permit normal program sequencing, depending upon the condition of the result.

TABLE 7. TYPICAL ELECTRONIC DIGITAL EQUIPMENTS

General Characteristics	Royal McBee LGP-30	Electrodata Datatron	IBM 705
Arithmetic and logic unit			
Number system	Binary	Binary coded decimal	Binary coded alphanumeric
Word size	30 binary digits plus sign	10 decimal digits plus sign	Variable word length
Operating mode	Serial, fixed point	Serial—parallel, fixed or floating point	Serial, fixed point
Arithmetic speeds	Add, 8.75 msec; mult., 24.00 msec	Add, 1.1 msec; mult., 9.3 msec	Add, 0.119 msec (five-digit numbers); mult., 0.799 msec (five-digit numbers)
Control unit			
Instruction type	One address	One address	One address
Checking features	Accumulator overflow	Overflow, read-write	Character check, overflow, read-write
Internal storage			
Type	Magnetic drum	Magnetic drum	Magnetic cores   Magnetic drum
Capacity	4096 words	4080 words	40,000 char   60,000 char
Access time	7.5 msec	0.85 msec	0.017 msec   8.0 msec
Magnetic tape equipment			
Number	None	Up to 10	Up to 100
Start time	—	6 msec	10 msec
Transfer rate	—	6000 digits/sec	15,000 char/sec
Input equipment			
Type	Paper tape   Keyboard	Punched cards   Paper tape	Punched cards
Speed	10 char/sec   Manual	240/min   540 digits/sec	250/min
Output equipment			
Type	Paper tape   Printed page	Punched cards   Paper tape	Punched cards
Speed	10 char/sec   10 char/sec	100/min   60 digits/sec	100/min
Printing equipment			
Type	Flexowriter	Wheel	Wheel   Matrix
Speed	10 char/sec	150 lines/min	150 lines/min   500 lines/min
Use	On-line or off-line	On-line	On-line or off-line
Approximate costs <sup>a</sup>			
Rental	\$1100/month	\$4,000–\$10,000/month	\$25,000–\$60,000/month
Purchase	\$50,000	\$140,000–\$350,000	\$1,250,000–\$3,000,000

<sup>a</sup> Based on 1958 costs.

### Checking Features

In order to ensure accurate and reliable results, electronic digital equipment has checking features built in as extra hardware. The checking features commonly used are the parity check, duplication, and indicator checks. See Chap. 13.

**Parity Checking.** In some parts of electronic digital equipment, a redundant check bit is added to the code for each character so that each character always has either an even number or an odd number of binary one bits. The characters are checked for parity each time they are read, transferred, or written. This type of check detects the most common type of malfunction that occurs in electronic digital equipment, an error in one binary digit of a code. See Chaps. 4 and 13.

**Duplication.** Duplicate electronic units or dual operations can be used to ensure the accuracy of all calculations. Dual recording on magnetic tape is sometimes used to increase the reliability of magnetic tape reading and writing.

**Other Checks.** Other built-in checks are used to indicate undesirable conditions that can occur within the electronic digital equipment. Such checks can indicate conditions like accumulator overflow, incorrect sign, and invalid instruction codes and address.

## 8. TYPICAL ELECTRONIC DIGITAL EQUIPMENT

There are several commercially available data processing systems for business data processing and scientific computation. Shown in Table 7 are comparative features, speeds, and capabilities of typical available equipment. In general, the equipment described in this table represents a basic system, which can usually be expanded in capacity.

### REFERENCES

1. E. C. Berkeley, *Giant Brains, or Machines That Think*, Wiley, New York, 1949.
2. A. D. Booth and K. H. V. Booth, *Automatic Digital Computers*, Butterworths, London, 1953.
3. B. V. Bowden, Editor, *Faster than Thought*, Pitman, London, 1953.
4. W. J. Eckert and R. Jones, *Faster, Faster*, International Business Machines Corporation, New York, 1955.
5. R. K. Richards, *Arithmetic Operations in Digital Computers*, Van Nostrand, Princeton, N. J., 1955.
6. N. Chapin, *An Introduction to Automatic Computers*, Illinois Institute of Technology, Chicago, 1955.
7. E. C. Berkeley and L. Wainwright, *Computers, Their Operation and Applications*, Reinhold, New York, 1956.

## 5-44 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

8. R. G. Canning, *Electronic Data Processing for Business and Industry*, Wiley, New York, 1956.
9. G. Kozmetsky and P. Kircher, *Electronic Computers and Management Control*, McGraw-Hill, New York, 1956.
10. M. V. Wilkes, *Automatic Digital Computers*, Wiley, New York, 1956.
11. G. Kozmetsky and P. Kircher, *Electronic Computers and Management Control*, McGraw-Hill, New York, 1956.
12. J. W. Carr and A. J. Perlis, A comparison of large-scale calculators, *Control Eng.*, **3**, 84-92 (February 1956).
13. R. K. Richards, *Digital Computer Components and Circuits*, Van Nostrand, Princeton, N. J., 1957.
14. W. D. Bell, *A Management Guide to Electronic Computers*, McGraw-Hill, New York, 1957.
15. E. M. Grabbe, Editor, *Automation in Business and Industry*, Wiley, New York, 1957.
16. Haskins and Sells, *Introduction to Electronic Data Processing*, Haskins and Sells, Los Angeles, Calif., 1957.
17. D. D. McCracken, *Digital Computer Programming*, Wiley, New York, 1957.
18. M. H. Weik, *A Second Survey of Domestic Electronic Digital Computing Systems*, Office of Technical Services, U. S. Department of Commerce, Washington, D. C., 1957.
19. M. Phister, Jr., *Logical Design of Digital Computers*, Wiley, New York, 1958.

## Facility Requirements

*Erwin Tomash*

1. Physical Installation	6-01
2. Personnel Requirements	6-09
References	6-13

### 1. PHYSICAL INSTALLATION

#### **Overall Planning**

The installation of an electronic data processing system will pose unique problems not encountered in the installation of other types of equipment, and careful consideration must be given to these problems if an efficient, effective operation is to result. Planning the actual physical installation, which can be a major item of expense, is an often neglected phase of the overall make-ready program.

This chapter is concerned primarily with the installation of large-scale equipment. The installation of smaller equipment will pose problems similar in nature, but to a lesser degree. Often, these smaller installations may be treated as extensions of existing punch card installations.

**General Engineering Services.** Experience has shown that it is important to seek the advice and assistance of those experienced in installing and operating data processing systems. Two approaches are:

1. Prospective users having available the services of a plant engineering department (or equivalent) will be wise to avail themselves of this

## 6-02 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

service, since careful supervision and coordination of the technical details is mandatory.

2. Prospective users not having such services available will find it advantageous to obtain the services of an industrial engineering consultant to work with the equipment supplier and the various subcontractors, and thus assure proper coordination of all aspects.

**Equipment Manufacturers' Services.** Most suppliers make available an experienced installation department prepared to work with the customer's staff. They also furnish installation and specification manuals, which provide the technical information needed. However, the installation of a large-scale system poses many special problems requiring individual customer review and decision.

The supplier's fundamental responsibility ends with his submission of complete specifications and installation requirements. Usually these are furnished together with particular recommendations for the installation at hand. The supplier is primarily concerned that proper power and refrigeration facilities be provided in such a manner as to make possible the trouble-free operation of his equipment. He is much less concerned about such matters as physical layout, operational convenience, minimization of installation expense, and standby facilities.

The user wishes to comply with the supplier's installation specifications in order to have trouble-free operation of the equipment, and also to be able to hold the supplier responsible for equipment malfunction. The user must concern himself with certain installation details, particularly physical layout, so as to obtain proper and efficient utilization of the equipment. He should give careful study to work flow, traffic control, personnel housing, appearance, etc. Most users have chosen to use the equipment supplier's installation department on an informal consultant basis. In this manner, the user avails himself of the supplier's service and yet is able to retain control of planning to take into account his own individual requirements.

### **Physical Layout**

From the user's point of view the most important aspect of the installation planning is to provide adequate, properly laid out working space. It is safe to assume that power, refrigeration, and structural problems will be resolved and that these problems, however difficult of solution, once handled may be forgotten (except for routine maintenance). On the other hand, problems which arise during operation due to improper area and equipment layout remain with the user as long as the equipment is in use, or must be corrected at great expense.

**Main Equipment Room.** Important factors to consider are:

1. *Space.* A main equipment room is required to house the central data processing and computing equipment and any intimately associated input-output devices. The space recommended by most suppliers is around 3000 to 3500 square feet. This should be regarded as a minimum. Several installations have used as much as 4500 square feet and have found the additional space more than welcome. The exact layout of the operational area is dependent upon the individual operational requirements and type of equipment installed.

2. *Control Console.* All data processing systems have some sort of monitoring and control desk, and this control station should be located so that while seated at the console the operator in charge can view all equipment directly connected to the machine, such as magnetic tape units, card readers, printers, and typewriters. All elements of the system which require starting or stopping, loading or reloading and which can be controlled from the monitoring station should be located adjacent to the control station and in clear view of it.

3. *Accessibility.* The various elements which compose the data processing system must be placed to permit accessibility for maintenance and ease of operation. It should be possible to replace or change control panels, operate all controls, load and unload units, etc., without having to move these units. Generally, for commercial systems now in field use, suppliers recommend that a service area extending 3 or 4 feet from the unit in all directions be provided to permit ready access to the front, sides, and rear. If space is severely limited, this figure can be reduced somewhat by careful checking of control panel and access door clearances. Prior to installation, this checking can be done against manufacturer's specifications and drawings.

4. *Viewing Area.* Another important aspect of machine location results from the interest in equipment appearance. Most installations have provided viewing rooms for visitors. It has been found desirable to place the more dramatic and impressive units of equipment, such as the printing devices, monitoring panels, and magnetic tape units close to and facing out toward the viewers.

5. *Magnetic Tape Reels.* As a by-product of a large-scale data processing operation, a large active file of magnetic tape reels will be established and maintained. Suitable shelves, racks, files, or carts will have to be provided, generally adjacent to the magnetic tape units, for storage and effective control of these tape reels. Both file drawers and open shelves are in common use, with the latter somewhat more popular. Specially designed office style units to house these reels are available from the office

## 6-04 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

equipment manufacturers, as are tape dollies, etc. It will also be necessary to develop a very complete tape reel control and identification system, so that the wrong tapes are not inadvertently used or an excessive inventory of tapes is not developed.

**Maintenance Room.** In addition to the central equipment area, the supplier usually requires that an adjacent maintenance area be provided. An area of 300 to 400 square feet is recommended. The maintenance crew for a large-scale system will be permanently assigned to the user's installation, and will occupy this area. Suppliers generally request that facilities such as desks, chairs, and coat racks be provided for these personnel. If floor area immediately adjacent to the machine is at a premium, it will also be necessary to provide a small bulk-storage area for spare parts. This may be a separate room at some other convenient location, or it may be adjacent to the maintenance area. Ordinarily, for leased equipment, the manufacturer supplies all special equipment relating to the maintenance work itself, such as work benches, test equipment, tools, drawing racks, and spare parts drawers. For purchased machines it is necessary for the customer to provide equivalent facilities. From the appearance and control point of view it has been found preferable to establish the maintenance room with its own separate entrance and doorway, rather than as an area simply adjacent to the computer machine itself.

**Central Supply Room.** A small supply room approximately 100 square feet in area and suitably equipped with shelves or cabinets should be provided. This room is used to stock supplies associated with the data processing center, such as forms and paper, punch cards, and magnetic tape. This room should be convenient to the central processing room, if not adjacent to it. If the former, then a set of shelves or a cabinet should be provided in the main equipment room for small amounts of supplies.

**Power Room.** A wall area to mount controls for power and refrigeration equipment is usually all that is required in the central equipment room. Special provisions are often made so that large switches, circuit breakers, etc., are hidden behind panels or cover plates. Large transformers, motor generators, compressors, etc., to which limited accessibility is required, are in some instances located near the central installation in "power rooms," or they are distributed throughout the building, as convenient.

**Office Space.** It has rarely been possible to provide offices adjacent to the central equipment for all personnel associated with the operation. Certain personnel should be housed adjacent to the area, if at all possible.

The department manager or other administrative official who is in day-to-day charge of the operation should, by all means, have his office adjacent to the computer area. In addition, the chief operator should have his office adjacent to the central equipment area. Additional operating personnel usually share common office space near that of the chief operator.

Systems analysts and programming staff will require housing accessible, but not necessarily adjacent, to the central equipment room. Experience has indicated that, for the bulk of the programming staff, it is not efficient to use a common working room. The usual solution is to establish two-man offices, or use eye height partitions to create these. Systems analysis and programming require close and continued concentration, and the privacy and quiet resulting from the two-man office arrangement seems to be conducive to greater work output.

**Visitors' Area.** Most users have found it advantageous to establish and maintain a visitors' viewing room which permits ready view of all operations within the equipment room. These generally have been designed to accommodate ten to twelve people comfortably and to include glass panels for viewing and a suitable display and poster area where models and explanatory charts may be shown. It has been found necessary at most installations to supplement the operations being shown with a brief explanatory talk, and also to furnish diagrams, charts, and reprints of articles.

**Staff Conference Area.** During planning, initial cutover, and regular operation, frequent staff conferences will be needed, and a conference room seating at least six to eight people should be provided.

### **Power Requirements**

**General.** The user has little latitude in the matter of power requirements. The manufacturer's specifications must be closely followed. Local building codes and type of power available will effect installation detail; power control apparatus of various manufacturers may be used; however, any differences will be in detail rather than function, and the switchgear supplier may be relied upon to provide complete and satisfactory details for its installation.

**Regulation.** The major problem encountered from the electrical point of view is to provide adequate regulation of voltages entering the machine. Local power companies are usually cooperative in providing information as to regulation of power lines. If given sufficient advance notice, they will monitor and record data on the very lines which will be used. These regulation data provide the magnitude and frequency

## 6-06 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

of the variations likely to be encountered. A new installation represents a major increase in load and any advance notice which can be provided to the power company is greatly appreciated.

Since it is not possible to use unregulated voltages, some sort of regulating apparatus is required. Two approaches have been used:

1. Provide a buffering motor-generator set. Some manufacturers provide this motor-generator set as an integral part of the system. Others specify that such a motor-generator set be provided to ensure proper operation.

2. Other manufacturers recommend use of stabilizing voltage regulators, rather than the use of motor-generator sets.

In each case, manufacturers specifications should be followed carefully. The power lines for the compressors, blowers, etc., required to provide proper cooling of the system should be kept independent of those supplied to the computer proper in order to minimize further regulation problems.

Experience has indicated that electrical contractors accustomed to doing standard electrical work cannot be relied upon exclusively because of the special nature of this work. It will be necessary for the user to maintain close liaison between equipment manufacturer and electrical contractor to assure that the system is properly and efficiently wired and powered.

**Load Requirements.** The user should take precaution during planning to make certain that the primary source of power is greater than that amount exactly required to operate the equipment presently being installed. Future expansion of the system should be regarded as almost inevitable and as new or more modern units are added, these may require additional amounts of power. In addition, when estimating total power requirements, it is essential to remember that a large amount of power will be required for the refrigeration system. In fact, the power required by the refrigeration system very nearly equals that required for the data processing system itself.

**Cabling and Wiring.** Interconnecting cabling to connect the various elements of the system with the main power supply and with each other is provided by the manufacturer. The customer is expected to provide wiring from the primary source of power to the switchgear and voltage regulating system and to power supplies of the units. Usually interconnecting cables are run underneath the system. A false floor is often used to handle these cables and conduits. An alternate approach is to build or dig trenches in the floor between units. Both methods are in common use and have proved satisfactory, the false floor providing somewhat more flexibility and being somewhat more expensive.

A third alternative, which is the least attractive, the least expensive, and the least satisfactory, has also been used. This is to build a ramp over the cables and run the cables along the floor. The ramp acts as a protective cover to the cables but it also impedes the use of dollies in the machine area and provides a traffic obstacle.

**Convenience Outlets and Maintenance Area Power.** It will be necessary to install a number of convenience outlets all along the central equipment room walls for powering of maintenance test equipment and for operation of mechanic's tools, etc. In addition to convenience outlets at the work benches in the maintenance area, special power facilities are often needed for specialized maintenance test equipment.

Complete information regarding these special facilities is furnished by the manufacturer.

### Refrigeration Requirements

**General.** Proper equipment cooling has been a cause of great concern to both equipment manufacturers and equipment users. Equipment manufacturers have attempted in various ways to handle this problem in a convenient and economical manner. Simply stated, these large systems dissipate large amounts of power (from 50 to 150 kilowatts), and this heat must be removed from the equipment and the room housing the equipment.

Equipment manufacturers state their cooling requirements in different ways. Refrigeration equipment suppliers also provide specifications with varying terminology. Therefore, the following table will be found useful:

1 kilovolt-ampere	= 3400 British thermal units per hour
1 ton	= 12,000 British thermal units per hour
1 kilovolt-ampere	= 0.283 ton
1 ton	= 3.6 kilovolt-amperes

A ton (of refrigeration) means the refrigeration required to remove an amount of heat equal to the heat of fusion of one ton of ice per day.

**System Requirements.** The general type and amount of refrigeration required is usually specified in fairly complete detail by the supplier. The amount of refrigeration required is determined by the amount of power dissipated by the system. The exact type of refrigeration required depends in large part upon the design of the data processing equipment.

Essentially two types of systems are in common use today. These may be referred to as the open-ended and closed-loop systems.

1. *Open-Ended Systems.* The heat dissipated by the equipment is exhausted into the room, and the refrigerating system is required to

## 6-08 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

remove this hot air and provide incoming cool air to the machine. This type of system is almost universally used in smaller machines. The room air conditioning system is required to handle the machine heat load very much as it would handle any other room heat load.

Open-ended systems have been used quite effectively for large systems as well as smaller units. Quite obviously, when very large heat loads are installed in a few thousand square feet of floor space, then large capacity, carefully ducted air cooling systems are required.

2. *Closed-Loop Systems.* The closed-loop approach has also found considerable acceptance and a number of variations have been used in commercial data processing equipment. In this system, the cooling air is circulated but kept within the confines of the room or kept within the machine itself. The machine being cooled may be considered as a sealed unit. A set of internal cooling coils at the bottom of the unit cool the air; it is then blown past the heat dissipating elements; finally, the warm air is returned to the cooling coils.

The internal cooling coils are usually fed from a chilled water source, and these systems are often referred to as "chilled water" systems. The heat is carried away from the machine by the warm water return. The machine, therefore, contains within its own casework the blowers and heat exchangers which transfer the heat developed by the equipment to the water system, which is connected in turn to a refrigeration system. The user is required to furnish the necessary chilled water to such units. This type of system has an important advantage in that it is independent to a large extent of the heat ambient of the room in which the equipment is installed. The heat load of other units in the room, humidity conditions, etc., and other conditions external to the machine have no effect on the operation.

It should be pointed out, however, that during maintenance periods, access doors must be open, and there is, of necessity, an interchange of air between equipment and the room. Thus, depending on the humidity, condensate may form on the heat exchangers, and drains must be provided to remove this condensate if it should occur.

An advantage of the closed-loop system is that the refrigeration load is constant during summer or winter and, as stated before, is independent of external influences. A disadvantage of this system is that in addition to the chilled water system, it is generally necessary to provide a room air conditioning system for personnel comfort and to accommodate the heat load of small input-output units, such as magnetic tape units, card units, and key punch machines, which are not connected to the central sealed-air chilled-water system. Thus two air conditioning units may be needed, the second, however, of much lesser capacity.

### **Floor Loading**

Specifications for sizes and weights of the various elements in the system will be provided by the equipment supplier. It is important to note that many units are on casters, legs, or wheels; therefore, floor loading and structural plans must be checked for capacity to withstand concentrated loads. It is often desirable to insert metal floor plates at the load points, and thus distribute the load over a greater area. This also prevents damage to composition floors, such as linoleum, which otherwise would be damaged by pressure over a period of time.

### **Soundproofing**

The noise level associated with most commercial large-scale systems is considerably lower than that found in large punch card installations, but it is still somewhat higher than that encountered in most office operations. This is due in part to the noise created by input-output devices such as card readers, card punches, and printing units, and in part to blower noise, air conditioning equipment, and electric motors. Generally speaking, no special soundproofing is required beyond the use of acoustic tiles, such as those now commonly used in modern office construction, for ceiling and walls.

### **Lighting Control**

For most commercial equipment available today, there are no particular special lighting requirements. Light levels used in normal office operation are satisfactory.

## **2. PERSONNEL REQUIREMENTS**

**General.** Unlike problems in physical installation, which may be overlooked, a great deal of attention has been focused upon personnel training and procurement. Although there is no unanimity of opinion with regard to the number and type of people required, certain common patterns seem to be emerging.

In general, it seems to be considerably easier to teach data processing to company personnel experienced in the operation than it is to teach operational and systems background to data processing experts. Based upon this philosophy, most users have tried to train and use personnel from within their own organization and to hire a minimum of specialized experience. This is particularly true during the first phase of conversion where attention is focused on mechanizing present procedures.

An important development has been the use of specialists trained in the scientific disciplines who have been working with company systems

## 6-10 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

and procedures personnel to develop new approaches to the operational problems. This is particularly true in the upper echelons of data processing work where operations research teams and systems analysis departments are quite common.

A number of users have also chosen to employ the services of consulting firms to assist with the conversion problems. Management consulting firms have on their staffs trained individuals capable of assisting users in more effective utilization of the equipment.

**Training.** The primary source of education in the fundamentals of data processing is found in training courses offered by equipment manufacturers. Colleges and universities are now starting to provide generalized data processing training and each year more and more courses are being offered at universities throughout the country. The true source of practical training in data processing has been at the user's installation. Practical experience on an operating system remains the best source of training available today. Most users have found it not only advantageous but necessary to maintain their own continuing training program. This has been done on a more or less formal basis, often in cooperation with the equipment supplier. This on-the-job training has provided, of course, better control, emphasis, and format, and, not unimportantly, it serves as an excellent training vehicle for those individuals conducting the courses.

The training required by the various members of the data processing center staff varies with their function. The amount and type of training required are discussed below under the various staff classifications.

### Staffing

**Management Staff.** In general, the data processing center reports to a member of the middle management group who is responsible for results obtained in relation to the established objectives. This individual should be familiar with the general principles and concepts of electronic data processing. He should, in all probability, attend one of the shorter training courses of the equipment suppliers, as well as university seminars, management conferences, etc. He should have sufficient time available and sufficient interest in data processing to visit other installations, act on industry-wide committees, and in general represent the using organization in the field of data processing. He should, of course, act as the primary communication link to the top management of the user organization and to other using organizations with similar interests.

Reporting to the executive level position described above, most users have designated a direct managing head of the data processing center. This individual usually has had extensive experience with the using

organization and has a detailed knowledge of the organization, its systems and procedures, and its objectives. He should be able to devote full time to the management of the data processing operation. In addition to training in principles and concepts, attendance at seminars and meetings, he should attend programming and coding courses and be completely familiar with the properties of the equipment to be installed.

**Systems Planning Staff.** This is by far the most important activity in the operation of the data processing center, and success or failure of the entire operation will hinge largely on the effectiveness of the systems planning work. All too often no distinction has been made between "systems planning" and "programming." The importance of the distinction should not be disguised by the fact that good systems analysts are usually experienced programmers and as such often do some programming or coding work themselves.

The systems analysis portion of the work load, especially during initial conversion, is perhaps 80 per cent of the total effort to be expended. The detailed programming and coding which follows may require a greater man-hour expenditure, but they can be accomplished by lower level, less highly trained individuals following careful rules of procedure delineated by the systems design.

The typical systems analyst is a senior staff individual familiar with the application to be mechanized and its *objectives*. In addition, he has been thoroughly trained in the application and programming of the particular equipment being used. Most users have found that a group of six to eight systems analysts are required to accomplish a major conversion. The primary source of personnel for this group is from within the using organization. Experience has indicated, however, that it is wise to add to this group one or two very experienced analysts from other installations who will make up for their lack of knowledge of systems detail by their superior knowledge of equipment utilization. These experienced analysts very often may be used successfully as a consulting group to the other systems analysts and to the programmers. The equipment supplier is often able to furnish one or two analysts of this calibre to assist the customer in a consulting capacity.

The systems analyst should take a complete programming, coding, and applications course for a total of about three months formal training, and should have several years of systems and procedures experience.

**Programmers and Coders.** The detail work of carrying out the systems design and preparing it for running on the data processing machine is accomplished by the programming group. Very often, a distinction is drawn between programmers and coders, the former sometimes being considered more senior and more experienced than the latter.

## 6-12 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

Programmers are considered capable of taking systems flow diagrams and preparing these completely for use on the machine. Coders are considered capable of taking individual runs within an operation and preparing these for use on the equipment. The number of programmers and coders required is dependent upon the size of the application being converted and the rate of progress desired. In general, a minimum size programming group consists of about fifteen individuals. Groups of thirty programmers and coders are not uncommon.

The training provided for programmers and coders should include at least three months of formal programming training. After this training program, about nine months of experience will be necessary before their skills may be considered adequately developed.

**Instructional Staff.** As mentioned heretofore, most users have found it advantageous to establish their own training program and assign at least one individual to work with new employees, conduct training courses, etc. Experience has indicated that the user will require a continuing source of programmers and coders to take care of normal attrition, promotions, etc. Usually a member of the programming and coding group, or a junior systems analyst, is selected for this training function.

**Operators.** One of the major advantages of an electronic data processing system and its attendant centralization of processing is the large reduction in number of machine operators required. However, a small number of skilled individuals will be necessary.

A chief operator, preferably having extensive experience in another installation, will be required. This individual can be considered a member of the administrative staff and will be useful in training, program planning, machine scheduling, etc. In addition, he will supervise the activity of his staff of from four to eight operators. Naturally, the number of operators will depend upon the complement of equipment in use and the number of shifts the equipment is operated. Two classes of operators are found in most large installations. These are computer operators and auxiliary machine operators, the former being more highly trained and skilled than the latter.

Computer operators should have basic programming training, although certainly not as completely as programmers. In addition, they will require a rudimentary knowledge of the equipment logic and, of course, a detailed knowledge of its operation. After this training period, which generally requires about three months, time should be allowed for the operating skills to be developed. The timing of computer operator training is somewhat critical in that operators should continue to operate equipment in order to keep their skills fully developed. Training more than six months in advance of equipment installation has been found

to be wasteful. Auxiliary machine operators, on the other hand, require very little formal training beyond an understanding of the control of the particular units they will operate. A training program consisting of a one- or two-week course and a few weeks to acquire operating skill is generally satisfactory for these individuals.

**Maintenance.** Maintenance services are provided by the manufacturers for users leasing equipment and poses no particular problem. Many users of large electronic data processing programs have found it advisable to hire a senior engineer trained and experienced in data processing to act as a staff consultant on equipment procurement, installation, etc. This individual also serves in a liaison capacity, with the maintenance engineers provided by the equipment supplier.

For those users purchasing equipment, it will be necessary to hire and train a maintenance staff. Equipment suppliers provide their customers with thorough and detailed training courses similar to those provided for their own maintenance crews. A chief service engineer should be employed as soon as possible after the decision to purchase equipment. This person should have a formal education in electronic engineering and should be experienced in the maintenance of the particular equipment being purchased.

At least one year before installation of the equipment, a group of maintenance technicians should be hired and training started. It will not be necessary for these maintenance technicians to have completed formal education at the university level. The graduates of technical institutions, servicing schools, and armed forces training schools have been found to be adequately prepared to take manufacturers' training courses.

A period of six months should be allowed for the formal training on the computer system and auxiliary units. In addition, six months will be required for sufficient skill to be developed on the part of the maintenance crew to provide adequate maintenance services.

## REFERENCES

1. R. G. Canning, *Installing Electronic Data Processing Systems*, Wiley, New York, 1957.
2. W. C. Bell, *Management Guide to Electronic Computers*, McGraw-Hill, New York, 1957.
3. G. Kozmetsky and P. Kircher, *Electronic Computers and Management Control*, McGraw-Hill, New York, 1956.
4. American Management Association, *Electronics in Action: The Current Practicality of Electronic Data Processing*, New York, 1957.
5. American Management Association, *Establishing an Integrated Data Processing System: Blueprint for a Company Program*, New York, 1956.



## Design of Business Systems

*Howard S. Levin*

1. General System Requirements	7-01
2. Stages of System Evolution	7-02
3. Detailed Steps of System Design	7-03
4. Economic Impacts of System Changes	7-12
References	7-14

### I. GENERAL SYSTEM REQUIREMENTS

Digital computers offer new opportunities to the systems planner. Computers allow development of business information systems characterized by high-speed computation, rapid search, and procedural rigor. These factors, when coupled with new input and output devices and with scientific techniques for administrative control, enable design of a truly effective, integrated information system attuned to the needs of the firm.

The information needs of a firm are not necessarily the output of present procedures. Current methods are often a patchwork imposed by machine limitations, supervisory inflexibility, and reaction to business emergencies. As a consequence, the information required for policy guidance and day-to-day operation of the business may not be clearly reflected by current office activity. Nevertheless, present procedures provide a starting point for redesign of a business information system. They picture the way in which the firm is currently operated, reveal the emphasis presently placed on various aspects of the office work cycle, and

## 7-02 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

provide a basis for viewing office costs. With present procedures as a starting point, the system analyst can work toward a statement of basic information needs. Initially he will gain only an approximation of such needs, but this approximation is valuable in judging the feasibility of computers and choosing among competitive machines.

The impact of information handling on management effectiveness provides ample justification for careful design of information systems. An information system should be regarded as a growing thing in which no final or optimum form is ever achieved. The information output of the system, once established, should be used to reevaluate and reshape the information system to serve management purposes better.

### 2. STAGES OF SYSTEM EVOLUTION

The effective use of computing equipment and management sciences in meeting the information needs of business is brought about in several distinct stages. These are (1) analysis of present information flow patterns, (2) system definition and selection of equipment, and (3) system implementation. These stages are discussed first in general terms and then are detailed in outline form.

**Analysis of Present Information Flow Patterns.** The level of detail reached in analysis of present procedures is a basic problem. A careful balance must be sought between cursory survey of current procedures and detailed recital of clerical operations which may obscure fundamentals. The emphasis should be on the major paths of information flow rather than on the processing of paper.

Real effort is necessary at this stage to "see through" present procedures and gain insight into their basic structure. Preparation of charts showing major paths of information flow and important exceptions can provide insight to planners starting redesign of an information system. Questions regarding current information flow are much in order. Why? For whom? What does he do with it? What would happen if he didn't do it? What other information outputs would be desirable if available when needed at attractive prices? The answers are meaningful in determining the information requirements which underlie present systems.

**System Definition and Selection of Equipment.** Fundamentals of present activities are distilled and used to structure an idealized information processing system in this stage. This idealized system serves as a basis for equipment selection. To choose among competing data processors, it is necessary to know the kinds and volumes of information inputs and outputs, size of reference files, frequency of file use, sorting requirements, and computational formulation. Totaling these elements for many application areas builds a picture of requirements that data

processing equipment must satisfy. These requirements can then be weighed against equipment specifications and costs to choose among available equipments and to select the most economic basis of acquisition.

**System Implementation.** Computer system implementation has the prerequisite of careful problem definition. There must be an absolutely complete statement of the job to be done. Each step of the procedure must be explicitly stated with a precision not generally found in business information handling.

Programming a complex data processing operation is a task for which firm guides are not yet established. Automatic coding will shorten the actual job of machine instruction, but automatic coding is no substitute for the planning that must precede computer application. The system planner through problem definition must establish the framework in which automatic coding can be used.

The administrative problems of system implementation are not trivial. This is particularly true of the transition from one system to another. Careful advance planning is needed in such areas as file conversions, code changes, operational cutover, and personnel relocation.

**Management Review.** The three stages of system evolution take place consecutively and the completion of each affords an opportunity for top management review and evaluation. At each stage, understanding and support at the upper levels of company administration are required since far-reaching changes in company procedure may prove desirable. Positive attitudes toward change, particularly in the middle management group, must be won through both top management support and the persuasiveness of the system planner.

**Scope of Study.** The number of people and the investment required in design of an information system vary with the scope of the work undertaken. For limited areas, system design can be a modest undertaking; but where the total information system of a large and complex business is under study, the jobs of system design and implementation rise sharply in magnitude. Yet the design of an information system on the widest possible basis is important to satisfaction of management needs and efficient utilization of computer potential.

### 3. DETAILED STEPS OF SYSTEM DESIGN

The detailed steps of system design are as follows:

- I. **Analysis of Present Information Flow Patterns**
  - A. Establish system design program
    1. Objectives and scope of study
    2. Reporting status of study group

## 7-04 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

3. Group selection on basis of
    - a. Management perspective
    - b. Knowledge of company operations
    - c. Knowledge of information handling technology
  - B. Review present data processing activities placing emphasis on:
    1. Flow of information within the company
    2. Isolation of exceptions to general routines
  - C. Develop information flow patterns as a basis for system design. These patterns contain:
    1. Major paths of information flow and important exceptions
    2. System parameters for each information processing area
      - a. Input data
      - b. Search requirements
      - c. File maintenance
      - d. Computation
      - e. Output requirements
      - f. Data transmission requirements
  - D. Analyze information use within the company to discover:
    1. Motivation for output of present systems
      - a. Management decisions
      - b. Operating practices
      - c. External requirements
        - 1) Federal, state, and local governments for tax and regulatory purposes
        - 2) Independent auditors
        - 3) Equity and credit financing
        - 4) Security exchanges
    2. Needs not met by current procedures
    3. Time factors bearing on usefulness of information
  - E. Measure present costs in data processing areas
  - F. Determine basic direction of further systems study
    1. Basic technical and economic feasibility of electronic data processing
    2. Practical levels of automaticity in data processing
    3. Potential for operations research tools
- II. System Definition and Selection of Equipment**
- A. Develop specifications for a business information system in which:
    1. Information needs of management for policy guidance and operational control are adequately met

2. Computational and search abilities of electronic computers are employed where useful
  3. Applicable operations research techniques are employed
  4. Related data processing activities are integrated
- B. Project system requirements against various general purpose and special purpose computers. Determine the system implications of each data processor and its auxiliaries in terms of:
1. Equipment items necessary
    - a. Computing equipment
    - b. Input and output devices
    - c. Communications equipment
    - d. Data conversion equipment
  2. Operating personnel required
    - a. Input transcription clerks
    - b. Machine operators
    - c. Clerical workers
    - d. Maintenance crews
  3. Information output of the system
- C. Select equipment on the basis of:
1. System implications stemming from use of equipment items in various combinations
    - a. Information outputs
    - b. Automaticity of data handling operations
    - c. Capacity for growth in volume of data and system complexity
  2. Machine characteristics
    - a. Computing equipment
      - 1) Compatibility with various input and output equipments
      - 2) Machine logic and speed
      - 3) Status of programming development
      - 4) Error detection and correction features
      - 5) User experience
      - 6) Availability of computer, auxiliaries, and spare parts
      - 7) Manufacturer training facilities for programmers and maintenance personnel
      - 8) Manufacturer reputation and financial responsibility
    - b. Auxiliary storages
      - 1) Sequential access: magnetic tapes
      - 2) Random access: drums, disks, etc.
      - 3) Extent of buffering

## 7-06 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

- c. Input and output devices
  - 1) Compatibility with computing equipment
  - 2) Program features
  - 3) Operation speed
  - 4) Reliability
  - 5) Extent of automatic input sensing
  - 6) Input verification features
  - 7) Extent of buffering
- d. Communications equipment
  - 1) Kind of data transmission (i.e., keyboard, page copy, teletype tape, punched cards, magnetic tape)
  - 2) Kind of data receipt
  - 3) Program features
  - 4) Transmission speed
  - 5) Transmission reliability
  - 6) Selection and switching features
- e. Data conversion equipment
  - 1) Speed of conversion
  - 2) Program features
  - 3) Reliability
- 3. Economic factors
  - a. Projected cost reductions, if any
  - b. Value of improved information output
  - c. Equipment, programming, and transition costs
  - d. Rate-of-return calculations to determine desirability of equipment acquisition and whether purchase or rental is most economical
- D. Verify key system design assumptions and capabilities of selected equipment
  - 1. Develop selected routines and computer test critical runs
  - 2. Confirm estimated usefulness of library routines and automatic coding
  - 3. Validate computer timing estimates on the basis of tests

### III. System Implementation

- A. Organize implementation effort
  - 1. Select and train personnel for detailed system design and operation
    - a. Supervisors
    - b. Analysts versed in business systems, statistics, or operations research
    - c. Programmers
    - d. Operating technicians

- e. Maintenance engineers and technicians (if required)
- 2. Establish implementation group
  - a. Resolve reporting status to senior management
  - b. Provide for coordinating decisions which cut across departmental lines
  - c. Announce system implementation effort to company employees
  - d. Set up group administration
    - 1) Group leadership
    - 2) Working level contacts throughout the company
    - 3) Organization of balanced teams containing system analysis skills, programming knowledge, statistical know-how, and operations research experience in model building and optimization
- B. Plan equipment acquisition
  - 1. Contract arrangements
    - a. Delivery of computer and other data processing equipment
    - b. Training arrangements
    - c. Maintenance contracts
    - d. Legal and tax implications of contract terms
    - e. Acceptance tests on customer site
    - f. Insurance on equipment
  - 2. Site preparation
    - a. Space for computer, auxiliaries, and staff
    - b. Communications facilities
    - c. Power
    - d. Air conditioning as required
    - e. Special housings as required
  - 3. Purchase of spare parts and test equipment if necessary
- C. Set priorities for implementation of various application areas
- D. Define input, output, data processing, and data transmission requirements within each application area through:
  - 1. Further development of system requirements and their adaptation to the specifics of the selected equipment
  - 2. Formulation of policy and operating decisions which will meet management objectives and contribute to effective system design
  - 3. Coordination of policy and operating decisions within the company
  - 4. Development of models reflecting company operation for study of:
    - a. Interdependence of operating variables within the firm

## 7-08 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

- b. Optimization of cost, service, or production objectives
- c. Intra-firm communications
- E. Design procedures for auxiliary equipment
  - 1. Input engineering
    - a. Forms design
    - b. Explicit procedures for completing each input document
    - c. Equipment programming for keypunches, typewriter tape punches, etc.
  - 2. Communications engineering
  - 3. Programming of conversion devices required by the system
  - 4. Design of system outputs
- F. Program computer operations
  - 1. Broad computer planning: *process charts* projecting computer runs
  - 2. Specification of computer logic through *block diagrams*
  - 3. Detailing of exact procedural steps through computer *flow charts*
    - a. Use of automatic coding techniques
    - b. Basic computation routines
      - 1) Problem formulation routines
      - 2) Error detection routines
    - c. Housekeeping routines
      - 1) Input routines
      - 2) Output routines
      - 3) Rerun procedures
  - 4. Preparation of machine instructions by *computer coding*
- G. Debug computer routines
  - 1. Prepare sample problem for each routine which tests flow chart branches and uses subroutines
  - 2. Hand compute answers to sample problem and contents of selected storage locations
  - 3. Run sample problem and compare with hand computations
  - 4. Analyze computer routine if computer hangs up or if computer calculations are in error
    - a. Stop computer at preselected check points
    - b. Read out selected storage locations and compare with expected contents
    - c. Correct routines where errors are discovered
- H. Debug system operation by running individual routines successively as planned in actual use
  - 1. Combinations of routines with test data
  - 2. Parallel operation with existing system



## 7-10 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

- I. Plan transition to new system operation
  - 1. Train personnel in new duties
  - 2. Phase over to new procedures
    - a. Preparing records in the form required by the new system
    - b. Converting to new codes and designators where necessary
    - c. Adjusting to policy changes embodied in the new system
    - d. Labor scheduling to provide for expanding work load during changeover
    - e. Scheduling of equipment operation
  - 3. Relocation of personnel as older procedures are superseded
- J. Evaluate system operation
  - 1. Utilization of information outputs
  - 2. Analysis of exceptions requiring manual handling
    - a. Input errors
    - b. Unplanned occurrences
  - 3. Programming inefficiencies
  - 4. Programming errors
  - 5. Justification of various special handling routines through analysis of their use
- K. Research further into information needs and the means of satisfying them

EXAMPLE. Charting techniques are helpful in both setting down present information flow patterns and defining system requirements as a basis for machine selection. A payroll example will be used to illustrate the procedure.

*Existing System.* Figure 1 charts information flow in a public utility payroll procedure. The figure represents an existing system before redesign in the light of computer potentials. It is intended to show major paths of information flow and to deemphasize paper handling.

The operations and controls noted by the system analyst are shown. Key-driven payroll accounting machines were used in the procedure and the characteristics and limitations of this equipment are reflected in Fig. 1. Consequently, a computer system would not necessarily duplicate the processing shown. Figure 1 should be considered as providing raw material for redesign of the information system.

*Redesigned System.* Figure 2 suggests redesign for three related areas: personnel records, payroll, and labor distribution. From the standpoint of present information flow these areas are separate procedures, each under different administrative control. These related areas have been integrated as one step of system definition. Figure 2 should not, however, be interpreted as specifying computer processing. The processing indi-

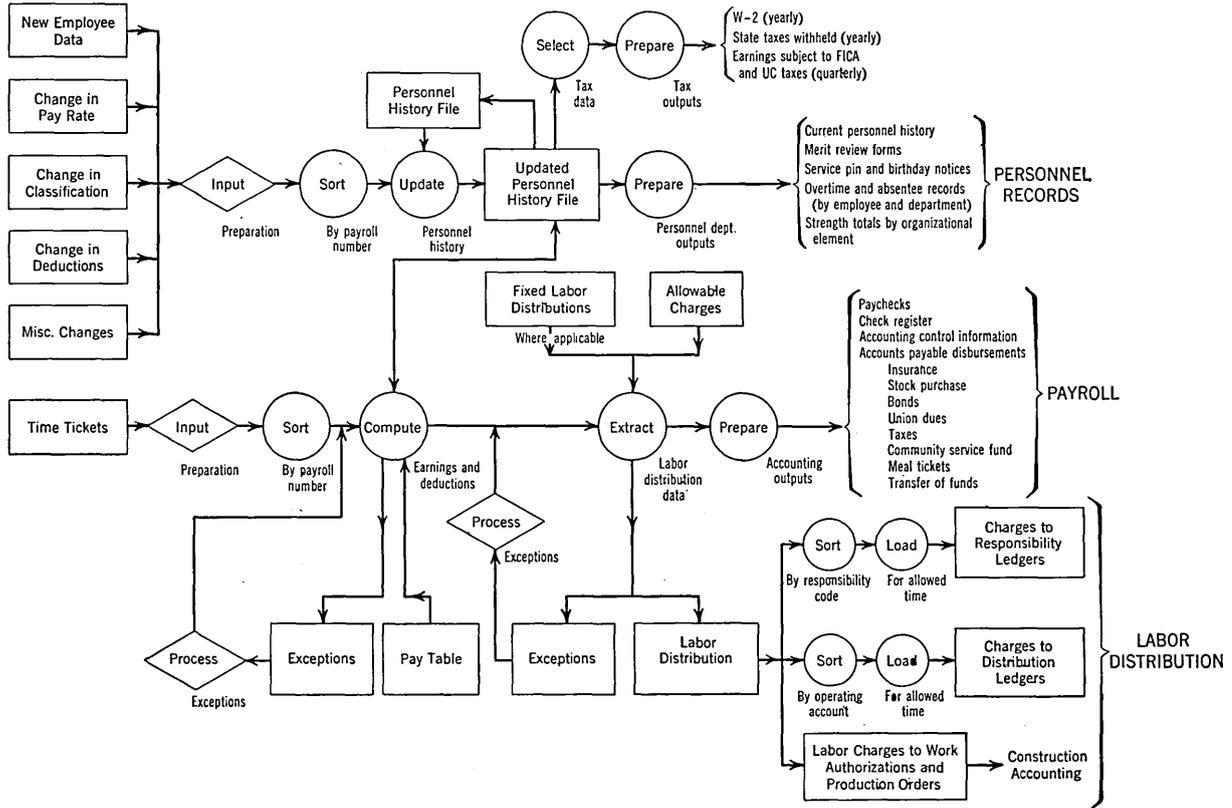


FIG. 2. A broad view of information processing in three related areas.

## 7-12 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

cated should be modified in the light of eventual equipment selections. Rather, Fig. 2 provides an idealized system as a starting point for equipment evaluation.

### 4. ECONOMIC IMPACTS OF SYSTEM CHANGES

**Economic Evaluation.** The decision to acquire a computer and associated data processing equipment is very largely a dollar and cents matter. Both reduction of clerical costs and the value of better information can be used to estimate the net dollar effect of computer system utilization. This net figure results from considering all cash costs incurred in computer system operation and all benefits which result. The flow of cash over the useful life of the system allows the desirability of computer acquisition to be measured in definitive terms.

**Return on Investment.** Computers and allied equipment represent just one area in which management may have opportunity for investment. Acquisition of data processing equipment must compete for funds with other worth-while projects within the firm. This acquisition can be judged in terms of *rate of return* on invested capital. Installing a computer system through either rental or purchase necessitates a considerable investment. The return on this investment is calculated from the cash flows generated through system use. This analysis requires an assumption of useful life for the computer, associated equipment, and computer routines. An estimate of useful life for these elements plays an important role in economic analysis. While computer routines may undergo substantial revision after operational experience is gained with the computer system, the computer itself should have relatively long life. It does not matter if cheaper, more powerful, or otherwise more enticing machines are developed; a given computer can provide productive service over an extended period. Estimates of useful computer life range from five to fifteen years with the writer tending to accept the higher figure as a reasonable estimate.

Another important factor in economic analysis is the depreciation of capitalized facilities taken for tax purposes. Such depreciation makes possible an estimate of the net cash effect of the system over the useful life of the system. This net cash after taxes allows computation of the rate of return on the project. The rate of return on invested capital is defined as the rate at which future earnings of a project must be discounted so that their present worth equals the investment. Such analysis provides a sound basis for evaluating the economic desirability of computer acquisition. Choices among competitive machines and between rental and purchase are facilitated through comparison of rates of return on invested capital.

TABLE 1. RATE-OF-RETURN CALCULATIONS FOR EQUIPMENT PURCHASE AND RENTAL

<i>Out-of-Pocket Investment</i>			
A. Equipment Purchase		B. Equipment Rental	
Equipment cost	\$ 700,000	Programming and transition cost	\$400,000
Programming and transition costs	400,000	Site preparation and additional capitalized costs	100,000
Site preparation and additional capitalized costs	100,000		\$500,000
	<u>\$1,200,000</u>		
Income tax saving through expensing of programming and transition cost	200,000	Income tax saving through expensing of programming and transition cost	200,000
	<u>\$1,000,000</u>		<u>\$300,000</u>

*Rates of Return*

Items Determining Cash Flow	A. Equipment Purchase		B. Equipment Rental	
	Life of System		Life of System	
	Five Years	Ten Years	Five Years	Ten Years
1. Out-of-pocket investment in data processing equipment, computer routines, and other costs	\$1,000,000	\$1,000,000	\$300,000	\$300,000
2. Net annual cost reduction before cost of equipment maintenance	\$ 325,000	\$ 325,000	\$325,000	\$325,000
3. Less equipment rental (including maintenance)			220,000	220,000
4. Less depreciation on equipment, site, and other capitalized costs	160,000	80,000	20,000	10,000
5. Less cost of equipment maintenance	60,000	60,000		
6. Net taxable savings	\$ 105,000	\$ 185,000	\$ 85,000	\$ 95,000
7. Income tax, 50% of item 6	52,500	92,500	42,500	47,500
8. Added annual cash generated (item 2 minus items 3, 5, 7)	\$ 212,500	\$ 172,500	\$ 62,500	\$ 57,500
9. Pay-back period (length of time needed to get the out-of-pocket investment back—item 1 divided by item 8)	4 $\frac{3}{4}$ years	5 $\frac{3}{4}$ years	4 $\frac{3}{4}$ years	5 $\frac{1}{4}$ years
10. Return on investment <sup>a</sup>	2%	12%	12%	15%

<sup>a</sup> Calculated by

$$P = \frac{1}{r} \left( 1 - \frac{1}{e^{nr}} \right),$$

where  $P$  is the pay-back period,  $n$  the number of years of economic life, and  $r$  the per cent return on investment.

## 7-14 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

EXAMPLE. Table 1 contains sample rate-of-return calculations. These have been determined for equipment rental and purchase under two assumptions of system useful life. Several simplifying assumptions have been made for illustrative purposes: level rather than accelerated depreciation, equal useful life for all system elements, income tax at 50 per cent, equal annual savings over the useful life of the system.

These rates of return, when viewed with the amount invested in system implementation, allow management to make knowledgeable decisions about the desirability of equipment acquisitions.

### REFERENCES

1. W. D. Bell, *A Management Guide to Electronic Computers*, McGraw-Hill, New York, 1957.
2. R. G. Canning, *Electronic Data Processing in Business and Industry*, Wiley, New York, 1956.
3. R. G. Canning, *Installing Electronic Data Processing Systems*, Wiley, New York, 1957.
4. C. W. Churchman, R. L. Ackoff, E. L. Arnoff, *et al.*, *Introduction to Operations Research*, Wiley, New York, 1957.
5. M. P. Doss, Editor, *Information Processing Equipment*, Reinhold, New York, 1955.
6. L. H. Hattery and G. P. Bush, Editors, *Electronics in Management*, The University Press of Washington, D. C., 1956.
7. I. A. Herrmann, *Office Methods, Systems and Procedures*, Ronald, New York, 1950.
8. G. Kozmetsky and P. Kircher, *Electronic Computers and Management Control*, McGraw-Hill, New York, 1956.
9. H. S. Levin, *Office Work and Automation*, Wiley, New York, 1956.
10. D. D. McCracken, *Digital Computer Programming*, Wiley, New York, 1957.
11. F. Wallace, *Appraising the Economics of Electronic Computers*, Controllership Foundation, New York, 1956.
12. C. C. Gotlieb and J. N. P. Hume, *High-Speed Data Processing*, McGraw-Hill, New York, 1958.

## Accounting Applications

1. Life Insurance Accounting, by A. C. Vanselow and R. L. VanWinkle	8-01
2. Casualty Insurance Accounting, by L. L. van Oosten	8-08
3. Public Utility Customer Billing, by E. D. Cowles	8-11
4. Payroll and Salary Distribution, by H. Tellier	8-15

### I. LIFE INSURANCE ACCOUNTING

*A. C. Vanselow and R. L. VanWinkle*

#### Overall System Description

A centralized life insurance accounting system requires speed in processing large volumes of data for the preparation of all reports and records generated for many policyowners and agency representatives. These reports are a result of the combination of a brought-forward file and new input data.

To handle such applications as premium billing, premium accounting, dividend accounting, agents' commission accounting, and valuation of policy reserves with maximum efficiency requires an integrated data processing system.

Franklin Life's integrated system is built around a master tape file of 240 digit items and a name and address tape file of a variable item size with basic items of 120 digits. These two files replace five major punched card files, an Addressograph system of 650,000 plates, and seven ledger and index card files.

**Equipment.** The equipment employed for the integrated data processing consists of: 1 large data processor (Univac I), 18 typewriter

## 8-02 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

inputs for magnetic tapes (Unitypers), 12 magnetic tape units (includes 2 for high-speed printers), 2 high-speed printers (600 lines/minute).

**Installation and Operation.** The planning and preparation of the programs necessary to convert our entire system to E.D.P. required 504 man-months. This was accomplished by 14 men. Five persons attended computer manufacturer's programming courses. The data processing system is used 580 hours per month for production and debugging. Scheduled and unscheduled downtime are 13 per cent and 3 per cent, respectively.

The operating staff required for the integrated system consists of 4 analysts, 10 programmers, 4 operators, and 12 maintenance personnel.

### **Master Tape File**

The master file contains 20 Univac words for each policy item. This includes all the policy data necessary for the five major accounting systems except the policyowner's name and address, which is maintained on a separate tape file. The sequence of the master file is policy number within each premium billing due day (1-31 inclusive).

There are many policy changes, including premium payments, terminations, data changes, policy loans, partial surrender of dividends or coupons, and new business, each reflecting a change in the status of the policy. Daily application of all such changes to the entire tape file, approximately one hundred twenty reels of 6000 items each, is not justified economically. As a result, accumulated changes are applied for a particular due day on a monthly cycle basis when the due day is scheduled to be processed for the selection of premiums due for billing, dividends, policy loan interest, etc.

All policy changes are untyped daily and verified. The verified tape is purified for legitimate coding and merged with the previous accumulated change file simultaneously selecting off all changes for the due day to be processed. These changes are then applied and the master file is now current. Updating the master file for one particular due day takes about 30 minutes to process external changes. Total time spent each month in updating the entire file is approximately 20 hours.

### **Dividend Accounting**

The following is a description of the dividend accounting procedure.

**Objectives.** The dividend accounting system is designed to calculate the amount of current dividends and coupons, calculate interest on the savings fund, prepare dividend checks, and record which of the five dividend options a policyowner has chosen each year. The options

are: (1) part payment of premium due, (2) purchase of additional paid-up insurance payable at maturity or death, (3) deposit with the company as an interest-bearing savings account, (4) shorten the premium paying period, (5) cash.

In contrast to most life companies, Franklin allows the policyowner to make this selection with each dividend payment. Our system is unique in that we prepare in excess of 1200 checks daily for the majority of the current dividends which are mailed to our agency representatives for delivery to the policyowner. All checks are returned directly to the home office for processing with the selected option indicated except those which are cashed. Certain plans of insurance contain coupons which are guaranteed endowments and are credited to the policyowner's account automatically at the rate of 600 items daily as an interest-bearing fund subject to the clipping of the coupons in the policy if the cash is desired when earned.

#### **Input and Output.**

1. Input.
  - a. Master file magnetic tape.
  - b. Data on dividend options on magnetic tape (prepared by Uni-typer, electric typewriter to magnetic tape).
  - c. Supervisory control panel type-in: check numbers.
2. Outputs.
  - a. Magnetic tape, new master file dividend, coupon result tape.
  - b. Printed reports.

#### **Major Processing Steps.**

1. Determine if dividend and/or coupon is due.
2. Calculate dividend and/or coupon.
3. Prepare dividend, coupon item.
4. Determine if dividend and/or coupon interest earned.
5. Calculate dividend and/or coupon interest earned.
6. Prepare output tapes using name and address file.
7. Accumulate control totals.
8. Print out reports.

#### **Quantity of Data Processing Performed.**

- A. Input and output.
  1. Input.
    - a. 181,440,000 digits of information stored in master file.
    - b. 103,680,000 digits of information stored in name and address file.
  2. Output.
    - a. 5,685,000 digits printed.
    - b. 6,500,000 digits recorded on magnetic tape.

## 8-04 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

- B. Each day's portion of the master file is processed as a self-contained group; daily result tapes are processed when eight days' tapes have been accumulated.
- C. File storage requirements.
  - 1. Magnetic tape master file contains 120 reels, 6000 items per reel, and twenty words per item.
  - 2. High-speed storage.
    - a. 300 words temporary storage.
    - b. 700 words of program storage.

### **Processing Steps Performed in Dividend Accounting**

#### **Selection of Dividends and/or Coupons Due.**

- 1. Check for presence of dividend/coupon code. Absence of dividend coupon code will cause computer to proceed to next policy.
- 2. If code present, examine policy issue year to determine if policy is old enough to earn a dividend or a coupon.
- 3. If policy is old enough, is dividend or coupon due the current working month?
- 4. If dividend due, determine if policyowner has returned previous dividend for deposit as an interest bearing savings fund.
- 5. If deposit field is negative, an error has occurred and the master item key is written out for manual checking.
- 6. If deposit field is zero, no interest is due.
- 7. If deposit field is positive, calculate interest earned.
  - a. Store interest amount for future notification to policyowner.
  - b. Add interest to amount of deposit and return to storage to be inserted into new master item.
- 8. Write new master item on new master tape (also, retain in storage).
- 9. Calculate the dividend or coupon duration year.
- 10. Prepare from the new master item a dividend/coupon item.
  - a. Indicative information.
  - b. Dividend interest earned.
  - c. Amount of insurance.
  - d. Coupons and interest on deposit.
  - e. Paid-up additions on deposit.
- 11. Write dividend/coupon item on a result tape.
- 12. This procedure is repeated for each policy on each tape for the selected due day.
- 13. The result tapes are retained for a period of eight days as an arbitrary procedure to save computer time.
- 14. The eight tapes are then sorted by the key (fund, mortality table, duration year, kind of policy, and age at issue).

15. The key of the sorted dividend/coupon items is matched for equality against the dividend rate index.
16. If equality is not found, the item is written out for manual coding of an index item so that the dividend/coupon can be run through the next eight-day cycle.
17. When equality is found, is a dividend, a coupon, or both earned?
18. If a dividend is earned, calculate the current dividend earned and the amount of additional paid up insurance this dividend would purchase.
19. If a coupon is earned, calculate the current coupon earned.
20. If previous coupon and interest on deposit, add to current coupon and calculate one year's interest.
21. If no previous coupon on deposit, calculate one year's interest on the current coupon.
22. Prepare and write out an item on the dividend/coupon result tape.
23. The result tape of dividends and coupons are now sorted by their new key which is due day and policy number.
24. Match this sorted result tape with the policyowner's name and address file for a selected due day.
25. When equality is found with the name and address file prepare output tapes with items:
  - a. Outstanding dividends.
  - b. Dividends, coupons, and interest due.
  - c. Dividends, coupons, and interest not due (due days other than selected, controlled by name and address tapes).
  - d. Dividends.
26. If equality is not found, follow same procedure with no name and address on the dividend check (manually typed).
27. Accumulate totals for dividend amount, coupon amount, interest on coupons, and interest on dividends.
28. Assign a check number to each dividend check (controlled by supervisory control panel type-in).
29. The accumulated totals are entered at the end of the dividend, coupon, and interest due tape.
30. The tape of dividends, coupons, and interest due is listed on the high-speed printer for the check register.
31. The dividend total on check register represents the amount of the checks written and is used as the basis to prepare manually a check requisition.
32. Coupons, interest on coupons, and interest on dividends are automatically placed on deposit for the policyowners by manually

## 8-06 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

preparing a journal voucher from totals printed on the check register.

33. List the dividend checks on the high-speed printer.
34. Manually select all dividend checks with a permanent option indicated.
35. All other dividend checks are mailed to the agency representatives for delivery to the policyowners.

Permanent options are dividends which are automatically placed on deposit at the request of the policyowner. Checks are prepared with "Void" in the amount so they may be processed in the same manner as returned checks.

Specific conditions require that certain options are not available, therefore, a series of X's is printed on the check form in this option field. For example, the premiums paid by bank draft, salary deduction, government allotment, or any automatic premium payment plan, special authorizations are necessary each time the amount is changed; therefore, the option of applying the current dividend to reduce the next premium payment is deleted.

The calculation of the dividend and coupon interest is different because the dividend is credited at the beginning of the policy year whereas the coupon, the coupon interest, and dividend interest are credited at the end of the policy year.

**Processing Returned Dividend Checks and Automatic Option Voided Checks.** This procedure is for the voided permanent option checks, the checks returned from the policyowner with a selected option, and the cashed checks returned from the bank.

1. Tally checks by selected option code with an adding machine.
2. Attach adding machine tape to each group of returned checks.
3. Unitype 20 digits of information for each check.
  - a = check number
  - b = policy number
  - c = selected option code
  - d = amount of dividend
4. Unitype selected option code control totals from attached adding machine tape into the first sentinel item.
5. Tally unityped tapes on Univac by selected option code and balance with control totals in sentinel block.
6. If there is no balance, a manual determination must be made as to whether the checks must be re-unityped or the error is of a nature which will later be detected by Univac and ejected.
7. If the checks balance or it is decided to continue the processing

- of the checks which are out of balance, they are now sorted by check number and policy number.
8. Match the sorted returned checks against the outstanding check file by check number and policy number.
  9. If no equality is found with the returned check items, they are rejected for manual consideration due to a unityping error.
  10. If no equality is found with the outstanding items, they are tested for determination as to whether six months outstanding.
  11. If the checks have been outstanding six months, they are removed from the outstanding check file and classified as an "automatic transfer."
  12. These automatic transfers are credited by Univac to be left on deposit as an interest-bearing savings or to purchase additional paid up insurance contingent upon the provisions of the policy contract, whether Franklin or of a company acquired by Franklin.
  13. The automatic transfers are tallied by option code and written on the same result tape as the returned checks.
  14. If the checks have not been outstanding six months, but are 90 days outstanding, they are written on a follow-up tape and the new outstanding file. Follow up notices are run on all checks outstanding over 90 days reminding the policyowner to return the check to the home office with a selected option indicated or to cash it.
  15. If the outstanding checks are not 90 days outstanding, they are written on the new outstanding check file.
  16. If equality is found with returned check and the outstanding check items, the outstanding is selected for further processing and written on the dividend result tape.
  17. When the match and select run is completed, the dividend result tape is tallied by trial balance account (determined by the dividend option) within state, county, and fund and added to the previous dividend liability summary tape for the year to date.
  18. The dividend result tape is listed for a detailed listing of transactions.
  19. The next program contains four input tapes:
    - a. Monthly coupon liability brought forward.
    - b. Unityped dividends and coupons surrendered.
    - c. Current day dividend and coupon interest and coupons.
    - d. Returned dividend checks and automatic transfers.
  20. The current coupons and surrenders of dividends and coupons are tallied by trial balance accounts within state, county, and fund.

## 8-08 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

21. These tallies are added to the coupon liability summary file tape preparing a new summary tape.
22. At the same time master file change items are prepared:
  - a. Dividends and coupons surrendered.
  - b. Coupon interest and coupons.
  - c. Returned dividend checks and automatic transfers.

These changes will be merged into the accumulated master change file to be applied to next billing cycle by due day.

23. The dividend liability and coupon liability tapes are merged and summarized monthly producing figures to be used in the annual statement.

## 2. CASUALTY INSURANCE ACCOUNTING

*L. L. van Oosten*

### Introduction

Casualty insurance accounting requires the rapid generation of many reports. These reports are generated by combining brought-forward data with those on source documents representing premiums, losses, and expense. In the past, cards representing the source documents were punched and balanced at each of Allstate's 28 branch offices. These cards were sent to the zone office (there are five zones) for summarizing and balancing and these summary cards were then sent to the home office. These summary cards were used then as inputs to punched card machines to develop the necessary reports.

To provide the smallest changeover transient and retain the desirable features of the punched card system and yet take advantage of electronic data processing machines it was decided to keep the above method of providing inputs and do the final processing with a stored program electronic data processor. Thus once the summary card information has been fed into the data processor, one pass produces the desired reports, comparisons, and ratios. This one pass does the job formerly requiring many passes through different types of punched card equipment.

### Remarks on Overall System

The data processor used in this installation is a Datatron. It has a 4080 word storage, an IBM 528 reproducer for punched card input, an optical tape reader (540 decimal digits/second) for punched paper tape input, an IBM 407 line printer for output, and three magnetic tape drives for auxiliary storage. Each tape drive holds a 2500-foot reel of magnetic tape on which can be stored 4,000,000 decimal digits grouped in 20,000 blocks of 200 digits each.

The planning and preparation of the first ten programs was accom-

plished by five men, four of whom attended a two-week programming course provided by ElectroData. This group then trained four men in programming and these men do any programming required now. The computer system is currently used approximately 230 hours per month and is operated by a staff of three operators and two service engineers. During 1955 there was 1.7 per cent unscheduled down time and about 7 per cent scheduled down time. The figures for 1956 are 3 per cent and 3 per cent respectively.

The accuracy of the Datatron system is greater than any other method ever used by Allstate to accomplish the same results. This does not mean that other equipment is necessarily inaccurate. However, by obtaining required results from one system capable of doing all the work involved in reporting instead of using many machines, plus several manual operations, the vulnerability to transpositions, sorting errors, lost cards, etc., is greatly reduced.

### Application Example

- I. **The System.** The following is a description of how one of Allstate's reports is prepared with the Datatron. The name of the report is "Analysis of Claims Closed by Duration."
  - A. This report is an analysis by branch of the claims closed during a given accounting month by claim report month, that is, the months in which the claims were originally reported. The past history of the claims closed for these report months is combined with the current closed claims to provide management with experience for the report months on a to-date basis, a 12-month moving average basis, and a 12-month-to-date moving average basis. Only claim cards are used in this particular report with no comparison to policies written.
  - B. Input and output.
    1. Input.
      - a. Magnetic tape. Brought forward information from previous report. Stored by branch.
      - b. Punched cards. Claims closed for current accounting month and closures for the accounting month one year prior to current month.
    2. Output.
      - a. Magnetic tape. Carry forward information for next period report.
      - b. Printed report on IBM 407 line printer.
  - C. Major processing steps.
    1. Punched cards are sorted together by branch code only.

## 8-10 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

2. Punched cards are read into the computer by an IBM 528.
3. Brought forward information is read from magnetic tape and adjusted by the information in the punched cards.
4. Calculated report is printed out directly on the IBM 407.
5. New carry forward information is created and stored on magnetic tape. Each block of data is summed, stored on tape, and checked for control purposes on future reports.

### II. Quantity of Data Processing Performed.

#### A. Input and output.

##### 1. Input.

- a. 4000 cards (as noted, sorted by branch only).
- b. 21,840 digits of information stored on magnetic tape in order by branch.

##### 2. Output.

- a. 174,720 digits printed.
- b. 218,400 digits recorded on tape.

#### B. Each of 26 branches is processed as a self-contained group. Within each branch there are five possible coverages (A, B, C, D, H). Coverages A and C are further broken down by 24 report months, coverages B, D and H by 12 report months.

#### C. File storage requirements (one word equals ten digits and sign).

1. Tape. 840 computer words for each of 26 branches for the brought forward and also for the carry forward, or a total of 1680 words for each branch.
2. Computer proper, for each branch.
  - a. 840 words to store brought forward.
  - b. 840 words to store information taken from the cards.
  - c. 672 words to store the computations.

### III. Processing Steps Performed.

- A. The program deck in the form of punched cards is read into the computer and a check sum or total of the program codes is created.
- B. "Digit add" the current accounting month, used to check if correct decks are being used.
- C. Punched cards are automatically read into computer in groups of 60 by the IBM 528.
- D. Control totals are read onto the drum from the beginning of the brought forward magnetic tape and checked. Later they are used to check the brought forward for each branch.
- E. After reading in the 60 cards on the drum, each card is tested.
  1. If there is a change in branch in the group of cards just

read, the magnetic tape is searched for brought forward information.

2. If all cards are for the same branch as previously read in, the coverage and report date determine where the information is accumulated on the drum (pseudo sorting).
  3. When the last card of a branch being processed is sensed, the brought forward information is read onto the drum from magnetic tape and checked against the control total stored on the drum at the beginning of the job (III, D). The carry forward tape is in position for storage of the new carry forward. Calculations are performed, report is printed out, new carry forward information is rearranged for use next month, and then stored on magnetic tape. The carry forward information is then read back onto the drum and checked against the control total.
- F. At end of each run the control totals are recorded on the new carry forward tape with a check sum. The information thus stored on magnetic tape then becomes next month's brought forward.

Within the computer, sorting as explained in III, E, 2 is done. Eleven averages are also computed for each line of the report (924 for each branch). Tapes are searched, read from, and written on.

Total time is 4 hours of which 4.3 minutes are used in reading and writing from tape.

Data on tape are stored so that for any one branch one read order brings in all the information needed and in the form needed. The calculations are stored so that only five output instructions will cause all the calculations to be printed in the proper form.

### 3. PUBLIC UTILITY CUSTOMER BILLING

*E. D. Cowles*

*Note.* This section describes the billing operations at The Detroit Edison Company in effect from 1955-1957. Conversion to IBM 705 was started in March 1957, and was completed in May 1958. This description is still valid for systems requiring only a medium speed machine.

#### I. Task Performed.

- A. **General Description of Task.** Every public utility has the job of preparing customer billing. This description of customer billing at the Detroit Edison Company is an example of data processing in a public utility. The meters are read, a calculation is performed to determine each customer's use, then com-

## 8-12 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

pared to previous use to determine if this billing is reasonable, the bill amount calculated, accumulations for revenue and statistics are made, and each customer's bill is prepared.

### B. **Inputs.**

1. Meter reading card (mark sense with provisions for four readings). The economy and usefulness of a multiple use card must be weighed against manual and machine complications due to multiple codes, control of reading field, errors and corrections. There is a minimum of one card for each meter on the customer's premises. If a meter has been changed during the billing period there will be a card for each meter used during the billing period.
2. Minimum charge card. This card gives the special (other than normal) minimum charge for the customer under consideration.
3. Rate information.

### C. **Outputs.**

1. Customer's ledger record (billing card). This card is used in preparing the bill form with a punched coupon as part of the customer's bill. The ledger record card is used as the accounts receivable record where payments are posted and from which overdue statements are rendered if required.
2. Outgoing meter card. The customer's meter change notice is prepared from this card.
3. Ledger total cards. These cards contain the accumulated total of kilowatthours and net billing. They are also used to check the numeric bill printer operation.
4. Detail revenue cards for statistical purposes. There are usually one or more of these cards depending on the number of municipalities, revenue classes, and tax classes within the ledger. These are accumulations of customer count, kilowatthours use, and net billing.

### D. **Major Processing Steps.**

1. Calculate customer billing.
2. Prepare revenue accumulations.
3. Prepare billing statistics.
4. Prepare bills.

## II. **Quantity of Information Processed.**

- A. **Input and Output.** There are one or more input cards per account and normally one output card per account. However, more than one output card may be required because of estimated

billing, meter change, and multiple meter accounts. In addition, ledger total cards, detail revenue cards, and statistics cards are prepared. Anywhere from one to fifty of these cards per ledger are prepared depending on the required breakdown.

**B. Number of Accounts.**

1,080,000 residential customers; billed bimonthly.

120,000 commercial customers; billed monthly.

110,000 off-peak water heater accounts; billed separately from residence accounts.

1,800 steam accounts.

35,000 accounts are processed daily.

**C. File Storage Required.** Detroit Edison has file storage requirements for:

2 million name and address cards.

2 million 400 thousand reading and historical record cards.

800 thousand outstanding accounts receivable cards.

2 million cards representing three months of paid records.

**III. Equipment and Staff.**

An IBM 650 is operated 12 hours per day, to process the 35 thousand accounts handled daily. The only output equipment used is the card punch associated with the machine.

Five key people were trained for the original planning; 3 of these were first line supervisors, and 2 were staff personnel. 325 man-days were used in programming the customer billing job. This was accomplished in 15 calendar months.

Two people are now sufficient to operate the processor when it is used on the customer billing application.

**IV. How Processing Is Performed.**

**A. Processing Steps.**

1. Read-in rate information.
2. Read-in other supplementary information.
3. Read-in meter reading and account information.
4. Calculate billing charges for a period.
5. Check billing for a divergence from expected amount.
6. Punch results into output card.
7. Prepare supplementary output card or cards as required.
8. Sort output cards, segregating various types.
9. Reproduce bill forms.
10. Print the billing side of the post card bill.
11. Address other side.
12. Perform various control and quality checks.

## 8-14 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

13. Customer bill form to post card size.
14. Mail bills.

### B. Types of Data Processing Operations.

1. Transcription.
2. Calculation.
3. Sorting.

### Programming

**Coded Information.** It is very desirable to keep the amount of information coded into machine language to a minimum. On the other hand, the machine must be told everything necessary to bill the account properly. Codes must be designed for the particular billing system at hand. No rule can be set down that would apply generally. The following are only some of the many conditions that require codes: (1) outgoing meter, (2) disconnected account, (3) skip billing, (4) estimated billing, (5) prorated billing for first, special, or final bills, (6) multiple meters, (7) rates, (8) tax applications, (9) revenue classification, (10) town distribution.

**Programming Techniques.** Many tricks may be developed as familiarity is gained with the machine programming techniques. It takes two instructions to test for zero whereas other values, requiring the subtraction of a number, require at least three instructions and an additional storage location for the "constant." When space is at a premium, these things become valuable, and sometimes a requirement, if the job is to be done at all.

The first several times through the programming, no regard should be given to efficient placement of instructions or stored data. Location of errors and testing can be done more efficiently if coding is sequential. When making corrections or adding small subroutines, to take care of some oversight, a check should be made for possible interference with other operations. This is very easy to overlook. Quite a few routines are seldom used, and it may take a long time to prove them out in actual practice. It is well to build up a "test" deck of cards for input that will prove every routine programmed. This may be quite difficult to do, but it may pay off in "trouble shooting" when in production. Once the program has been "debugged," then optimum coding can be undertaken.

Optimum programming of the 650 raised the card output speed from 48 cards per minute to 92 cards per minute for calculations on residence billings. (Maximum output rate is 100 cards per minute.)

**Instruction Breakdown.** A breakdown of instructions for the billing application is about as follows:

Input instructions	150
Output instructions	200
Processing instructions	
To pass inactive records	90
To process 81% of active records	1260
To process remainder of active records (involving exceptions)	<u>3300</u>
Total instructions	5000

## 4. PAYROLL AND SALARY DISTRIBUTION

*H. Tellier*I. **Task Performed.**

A. **General Description.** Preparation of the weekly payroll and the distribution of salary costs for cost accounting.

B. **Inputs and Outputs.** The basic information required each week is in the following form:

1. Punched card payroll master file. Contains a card for each authorized person with such data as hourly rate, occupation, cost center, and number of dependents.
2. Punched card deduction files. Each file contains a card for each person who has authorized that particular type of deduction.
3. Weekly time card for each employee. Daily attendance is printed by clock recordings. Hours worked or absent each day and a weekly summarization are entered manually on the card. Daily time distribution and weekly summarization are recorded manually.
4. Vacation payment authorization cards.
5. Earnings adjustment and correction cards.
6. Previous week's up-dated payroll record tape. Contains year-to-date earnings and taxes.

The information produced each week is listed in detail in Sect. IV below.

C. **Major Processing Steps.**

1. Prepare new time cards.
2. Process time cards.
3. Up-date files.
4. Calculate payroll and prepare payroll reports.
5. Prepare salary distribution reports.

II. **Quantity of Information Processed.**

The payroll represents some 7000 persons who distribute their hours worked to some 18,000 entries. The output of payroll and

## 8-16 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

salary distribution represents some 250,000 printed lines of records and reports. Payroll preparation and salary distribution represent less than 20 per cent of the electronic data processing work load.

The time card itself is considered to be the official record, and all time cards are retained in a records center for an extended period. The payroll record magnetic tape file for each week is retained for two years. All other payroll and salary distribution reels are retained only through the next payroll cycle. The input data cards are retained for two months.

### III. Equipment and Staff.

A. **Data Processing Equipment Used.** The central data processing center is equipped with an IBM Type 702 electronic data processing machine. In addition to the central processor (which contains a 10,000 character storage), the following peripheral equipment is available:

- 1 card reader
- 2 printers (150 lpm)
- 1 card punch
- 1 magnetic storage drum ((60,000 character storage)
- 9 magnetic tape units

The following IBM auxiliary punched card equipment is also available and used for payroll and labor distribution, in addition to key punches and verifiers:

- 2 Type 077 collators
- 3 Type 082 sorters
- 1 Type 101 electronic statistical machine
- 1 Type 407 accounting machine
- 1 Type 519 document originating machine
- 1 Type 552 alphabetical interpreter

### B. Staff Necessary.

1. *Conversion to the 702.* Payroll preparation and salary distribution were already mechanized by conventional punched card equipment. In converting these operations to the 702, no changes were made in input data fed into the data processing operation, and no changes were made in output reports prepared by the data processing operation. This policy was established so that no changes had to be negotiated with the customers, the personnel of the data processing organization did not have to learn a new system as well as the new machine, and the two machine systems could operate in parallel with identical output to test the accuracy of the new machine system.

Some 10,000 man-hours by analysts, programmers, operators, and control clerks were spent on programming, debugging, parallel runs, and comparison of the two machine systems through the first independent 702 processing.

2. *Weekly Routine Operation.* Owing to the variety of work performed by the data processing operation staff, requirements for payroll preparation and salary distribution are best expressed in average man-hours per week.

115 man-hours keypunch and verify hours worked and distribution of hours to cost codes

36 man-hours clerical control

98 man-hours punched card machine operations

36 man-hours operating the 702

#### IV. How Processing Is Performed.

**Time Card Preparation.** Time cards for each week are prepared from the payroll master file and sorted to clock location.

**Prepayroll Processing.** Payroll master file, appropriate deduction files, and key-punched payroll adjustments are translated from cards to tape. These tape files and last week's up-dated payroll file are then subjected to a file maintenance and editing procedure which produces an up-dated payroll file plus the necessary tapes to produce the following reports: submitted deductions by type and employee master list.

**Time Card Processing.** At the end of the payroll period, time cards are sorted to payroll number and matched with the master file to identify missing time cards.

Time cards are reviewed manually and a summary of hours is entered on each card. From the summary on each time card, a detail hours card is key-punched.

Vacation payment authorizations are also key-punched. These two types of cards are sorted to payroll number and processed card to tape.

**Payroll Calculation and Report Preparation.** Tape records of vacation, detail hours and the previously up-dated payroll status are used to perform the calculation from hours to net pay and continue through several additional computer operations to prepare various output reels.

#### **Output.**

1. Pay checks, earnings statements, reconciliation cards, and check transmittal receipts.
2. Payroll register.
3. Used and unused deductions by type.
4. Overtime summary by employee for the year.
5. Reports of hours worked, vacation, absence, year-to-date earnings, taxes, etc.

## 8-18 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

6. Force reports by occupation organization, working area, etc.
7. Accumulated deductions for savings bonds, bonds to be purchased and bond distribution transmittal lists.
8. Control totals on hours, earnings, etc.

**Time Card Processing for Salary Distribution.** After detail hours cards have been punched from the time cards, the time cards are sorted to cost center and manually checked for time distribution by cost code. Salary distribution cards are punched for time distribution by cost code, sorted to payroll number and processed card to tape.

**Salary Distribution Calculation.** Tape records of hours by cost code and the current week's payroll record are processed to calculate salary distribution. Additional passes are made to sort and arrange information for the following salary distribution reports.

### **Outputs.**

1. Distribution of regular earnings by cost code for each employee.
2. Individual charges to cost codes in payroll number sequence with cost code. Summary totals by cost code are computed and used later in month-end cost reports. Individual cost entries charged to work orders are later used as partial input to a work order cost system.
3. Listings of premiums and allowances by payroll number within cost center. Summary totals are computed and used later in month-end cost reports.

### **V. Processing Time Scale.**

Table 1 shows for each phase of the application the average time required for each type of operation in the data processing center. The following facts should be noted:

1. The actual 702 processing from hours worked to net pay requires only one 702 pass taking 1.75 hours. However, to manipulate the data into output reels for payroll reports calls for six additional passes.
2. The relative large number of punched card machine hours is needed to process punched card checks and earnings statements for distribution and the printing of numerous payroll reports from card output from 702 processing.
3. 702 auxiliary operations include card-to-tape, tape-to-card, and tape-to-printer operations.

A word of caution to persons developing electronic data processing systems is in order. Do not overlook:

1. The time it takes to sort data in the processor.
2. The time required to print the voluminous quantities of reports required for an integrated payroll system. Calculated answers are of no value as long as they are still on magnetic tape. Their value is not realized until they have been printed in report format.

TABLE 1. AVERAGE TIME REQUIRED FOR DATA PROCESSING OF PAYROLL AND SALARY DISTRIBUTION

Phase	702 Central Processing Unit		Punched Card Machine Processing, hrs.	702 Auxiliary, hrs. <sup>a</sup>	Key Punch Verification, hrs.	Control Clerk, hrs.
	No. of Passes	Hours				
Time card preparation	—	—	6	—	—	—
Prepayroll processing	4	2.75	1	4	9	4
Time card processing	—	—	10	—	46	—
Payroll calculation and report preparation	7	7.50	67	37	—	26
Time card processing for salary distribution	—	—	6	—	60	—
Salary distribution calculation	5	7.50	8	10	—	6
Total	16	17.75	98	51	115	36

<sup>a</sup> Card-to-tape and tape-to-printer operations.

## 8-20 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

### VI. Accuracy Control.

The same general types of control over input data are used with the 702 as were previously used in punched-card applications: control totals in the form of number of entries, sums of quantities or sums of dollars are established at time of original input. When control factors change from one element to another, the original element is verified, and the new element is established for further operation.

For example, attendance hours are the original control element for payroll. As soon as earnings expressed in dollars have been computed, if the number of hours is still under control, dollars then become the control element. Each processing step ties back to the prior control element.

Into the 702 processing we programmed such control elements and the console typewriter records control totals, both to tie back into prior processing and to establish new controls for the next processing.

Built into the 702 system are certain systems checks which test the validity of the machine's operation. From an operational standpoint, the major concern is that units of data are not lost in processing because of errors in programming logic.

## Inventory and Scheduling Applications

1. Inventory Control, by <i>Charles E. Ammann</i>	9-01
2. Aircraft Production Scheduling, by <i>C. W. Schmidt and R. Bosak</i>	9-07
References	9-12

### I. INVENTORY CONTROL

*Charles E. Ammann*

#### Application Description

**General.** The major task performed in this use of electronic data processing equipment is the maintenance of a timely record of the inventory of passenger seats on American Airlines. Inventory information is quickly disseminated to airline personnel upon request, and the inventory is instantly adjusted as sales and cancellations occur at many sales outlets (Ref. 1). Other types of inventory control systems are discussed in Refs. 2 and 3.

A second purpose of the system described here is to provide arrival and departure information, giving such information as arrival time, departure time, and delayed departure status with the reason for delay.

The system consists of a centrally located special purpose processing unit operated remotely by input-output devices referred to as *agent sets*. The agent sets are located in various airline reservation and ticket offices located in some instances hundreds of miles away from the processing equipment.

**Inputs.** The major inputs for this data processing system are the following (unless otherwise specified, these inputs are via the agent sets):

1. An initial inventory of seats available for sale on each flight leg.

## 9-02 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

(A flight is often made up of several portions known as "flight legs.") This inventory is entered by using a master agent set, a separate entry for each flight leg of a given day.

2. An initial inventory of seats allowed on each flight leg, also entered via the master agent set. "Seats allowed" refers to the capacity of the plane when flown over the particular leg, while "seats available" are those available for sale after deducting any which may be set aside for extra gas or other load purposes.

3. Requests for information as to the availability of seats, specifying the date, the number of seats, and destination.

4. Sales of seats, specifying the same information as in (3) but with a specific indication of the leg or legs required.

5. Cancellations of sales, specifying the same information as in (4).

6. Flight status information, entered via a special agent set located in flight dispatch. Arrival information is given in 5-minute increments, and departure information is given as routine or in terms of reason for delay. Sixteen conditions can be entered for designating flight status. If the flight information condition is complicated, the answer signal to the agent set indicates "call," requiring the agent to obtain details elsewhere.

7. Requests for information on flight status specifying the information as in (4).

8. Information on changes in operating schedules which introduce or delete flights or flight legs, change the number of seats available, etc. This input is made via the master agent set.

**Outputs.** The outputs of the system are the following (via the agent sets, unless otherwise specified):

1. Availability of seats, in response to specific requests defined as in (3) under inputs.

2. Sale completion information, telling the seller that the computer has effected the changes required by the sale.

3. Indication that insufficient seats exist to make a sale stated by an agent set.

4. Cancellation completion information.

5. Indication that inventory is at zero when a cancellation is attempted, rejecting the cancellation input.

6. Date and flight leg on which inventory has been exhausted. This output is by a teletype printer and punch.

7. Indication of improperly conditioned input information, produced whenever input information is incomplete, inconsistent, or requests information not in the present inventory records. Input is not accepted.

8. Indication of erroneous operation within the computer or unavailability of the computer due to malfunction.

9. Flight status information.

10. Total inventory of unsold seats, sold seats, and number of seats set aside for load purposes on any flight leg, via master agent set.

**Major Processing Steps.** The central processing equipment is capable of performing several fixed programs. A program is provided for each of the several modes of operation required of this system, such as sale, availability inquiry, and cancellation.

1. *Availability Program.* The date, number of seats desired, and coded reference to the particular group of flights indicated on the destination plate are transmitted to the processing equipment along with a signal indicating that this is an availability inquiry. The computer compares the inventory remaining on each of the eight flights or flight legs in the group with the number of seats requested. Signals are then transmitted to the agent to show which flights or flight legs have the number of seats requested.

2. *Sale Program.* The input steps are similar to those in an availability request except that they are confined to selected flights or legs in the group. The computer subtracts the number of seats sold from the inventories on the designated flight legs. One of several procedures then takes place: (1) If the inventory or inventories contain enough seats to cover the sale, the reduced inventory is stored in the processing equipment and the fact that the computer has taken the desired action is signaled to the agent set. (2) If any inventory is reduced to zero by this sale, this event is also signaled to the teletype output. (3) If an inventory is insufficient to cover the sale, this condition is signaled to the agent set, and the inventory is not reduced.

3. *Cancellation Program.* Here the computer usually increases the designated inventory by the number of reservations canceled and signals the completion of the action to the agent set. If, however, the inventory is zero at the time the computer attempts to increase inventory, the action will depend on the type of agent set we are using. On agent sets designated as "limited cancel," the inventory remains at zero and a signal is sent to the agent set that the cancellation was not accepted. Such cancellations are then routed to a sales agent who maintains a wait list of people who previously were unable to obtain space on the flight of their choice. If no demand exists for the space these positions can input the cancellation on their "unlimited cancel" agent sets even though the inventory is at zero.

### **Quantity of Information Processed**

**Daily Volume of Transactions.** In a normal business day, the equipment will handle over 45,000 calls. This includes all the major types of

## 9-04 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

transactions: availability inquiries, sales, and cancellations. Most of this volume occurs in one ten-hour period. Peak loads of 70,000 calls a day occur.

Once per day the inventory information for the day just passed is deleted and the starting inventories for another day are inserted automatically. Inventories for approximately 1300 flight legs are entered into the storage.

**Input-Output Stations.** Over 170 agent sets are connected to the central processing equipment.

**Storage Requirements.** An inventory of up to 255 seats for 31 days on approximately 2000 flight legs can be stored. Two days designated as X Day and Y Day provide availability information for any special period, for example, a future holiday. Storage is also provided for flight information on today's flights and the weekly frequency pattern of all flights which limits all action to days of the week each flight will operate.

### Data Processing Equipment

**General.** The electronic equipment performing this inventory control task is the Magnetronic Reservisor, manufactured by The Teleregister Corporation specifically for this job. The central processing unit has a memory capacity suitable for the volume of information described above, plus storage for certain tables that aid in the location of specific inventories in the storage. The machine performs fixed wired-in programs. The computer has a limited arithmetic and logical capability appropriate for this task.

The computer is in duplicate, each unit performing all operations. The operation of the two units is compared, and when a discrepancy occurs the call in process is printed out on a maintenance teletypewriter and the inventory retained in the original state prior to the call. A signal is also returned to the originating input unit and indicates that the call has been rejected. The system can operate at full speed with only one computer, in the event the other is down for repair.

The agent sets resemble a small adding machine in size and appearance (see Fig. 1). The central processing unit is connected to these input-output devices via multiple wire cables for on-premise extensions and by telephone circuits for off-premise points. Calling agent sets are connected serially to the computer and only one call is processed at a time. The computer remains connected to a given agent set long enough to complete a transaction, and then steps on to the next calling unit. The average waiting time experienced by an agent is less than one second, with a maximum of 5 seconds.

**Destination Plates.** The designation of the flight or flight legs desired on a given transaction is performed with the aid of steel destination plates. Each plate has printed material on it presenting in abbreviated

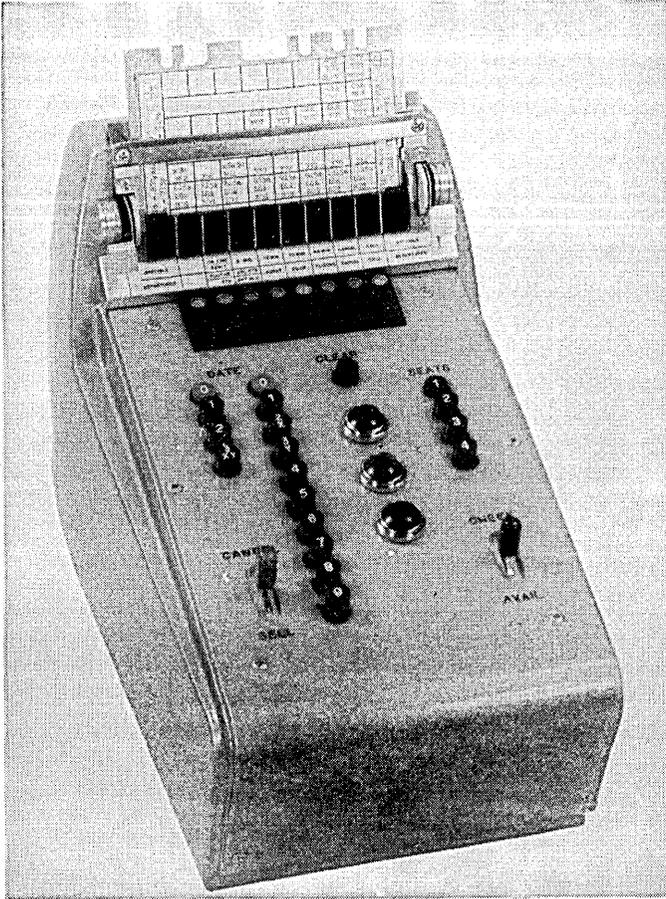


Fig. 1. Reservisor agent set.

form schedules for 64 flight legs. As shown in Fig. 2, the printed material on each side is in four rows of eight boxes; each box represents one leg. The two top rows are printed upside down. The top and bottom edges of the plate are notched.

When inserted into the agent set, the notches along the bottom edge of the plate control a series of switches. These switches set up a code which designates the two rows of eight legs visible right side up on the

9-06 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

lower half of the plate. A movable shutter in front of the destination plate modifies the input code to indicate the upper or lower row.

**Agent Sets.** The form of the inputs and outputs which take place via the agent sets is shown in Fig. 1. The travel date is indicated by

1 To - BOS/ALB (NSTP) 815p - 145a 615a - 715p 1								ALB LGA 420p	ALB LGA 620p	1 To - BOS/ALB (NSTP) 815p - 145a 615a - 715p 1
								752	750	
								BOS LGA 815p	BOS LGA 815p	
				145a	1115p	1015p	915p	BOS LGA 915p	BOS LGA 915p	
				312	510	346	368	318	318	
		xSu 382	308	xSaSu 302	300	372	332	310	344	
		615a LGA BOS	715a LGA BOS	745a LGA BOS	815a LGA BOS	915a LGA BOS	945a LGA BOS	1015a LGA BOS	1115a LGA BOS	
		374	340	394	338	352	388	320	398	
		1215p LGA BOS	115p LGA BOS	215p LGA BOS	315p LGA BOS	415p LGA BOS	515p LGA BOS	615p LGA BOS	715p LGA BOS	

FIG. 2. Reservisor destination plate.

depressing the proper tens and units buttons in the two columns of buttons on the left. The number of seats is indicated by depressing the 1, 2, 3, or 4 button on the right.

The desired group of flights or flight legs is designated by the destination plate, inserted into the agent set. Of the two rows of flight legs appearing right side up, the desired row is selected by flipping a shutter up or down and exposing the desired row.

When availability is requested, a small lamp lights (in the combination lamp-buttons below the destination plate) directly below each leg on the destination plate where the required number of seats is available. In

a sale or cancellation action, the specific flight legs in question are indicated by depressing the lamp buttons aligned directly under the appropriate boxes on the destination plate.

The indicator lights on the face of the agent set serve several operating and output functions. A green check bulb lights whenever a given transaction has been successfully processed by the system and assures the agent that the desired action has taken place and has been checked for accuracy. A yellow bulb lights when an insufficient number of seats is available to satisfy a sell transaction, or when a cancellation is attempted on a leg containing zero inventory. A red error light signals when incomplete or impossible input data are entered into the agent set, such as when the transaction concerns a flight leg inventory not stored in the computer, or when the computer does not successfully crosscheck itself.

**Performance.** In the small hours of the morning the Reservisor is taken off the line for a period not exceeding two hours when the entire equipment undergoes a marginal check. During the first year, the central equipment operated 99.8 per cent of its scheduled time of 22 hours per day, 7 days per week. Unscheduled down time runs less than 0.1 per cent with outages so short that only minor disruption of service is incurred.

### **Staff**

Aside from the persons operating the agent sets, the only personnel required is the maintenance staff. This staff is composed of five maintenance technicians and one electronic engineer. At least one technician is on duty at all times. The equipment is operated 24 hours per day.

## **2. AIRCRAFT PRODUCTION SCHEDULING**

*C. W. Schmidt and R. Bosak*

### **Requirements of a Good Schedule**

A good aircraft production schedule is one developed in a short time with a minimum of human effort and having a smooth acceleration from zero to peak production. An organization large and complex enough to manufacture aircraft can best develop in the manner of the well-known "growth" curve. This curve can be expressed mathematically as the integral of the normal error curve. Schedule development involves evaluation of this function and is thus facilitated by use of an electronic computer (see Refs. 3 and 4).

It is impossible to analyze properly and evaluate a production schedule without knowing its effect on manpower. Consequently, manpower forecasts are a basic part of this computer-developed schedule.

### Characteristics of Aircraft Production

Unitwise, aircraft production rates are very low compared with those of television sets and automobiles; however, the manhours required are extremely high. Since a limited number of people can work on an aircraft at one time, it has an unusually long production cycle. During this time it goes through three different types of manufacturing operations. These are fabrication, assembly, and production flight.

**Fabrication.** A large aircraft may have 20,000 detail parts which are produced on a lot basis during fabrication. The time between releases on successive lots is called the "lot cycle" and is relatively constant at approximately five months. The lot size varies with the production rate. Scheduling of fabricated parts is a separate problem and is not considered here.

**Assembly.** Major subassemblies are brought together and the airplane takes on a recognizable form. After completion of the airframe, the functional items (electrical, hydraulic, etc.) are installed. The work is broken down into line positions, and the airplanes physically move along an assembly line. All work must be done in its proper sequence.

**Production Flight.** This involves installing electronic gear, checking controls, systems and engines, inspection, test flights and working off "squawks." Airplanes can move to work stations somewhat at random so it is not a strict assembly line operation. Thus only the start and completion dates are significant. The completion of production flight operations is the delivery date to the customer.

### Approach to Problem

This application of data processing provides data for management decisions. The two major considerations that differentiate this from ordinary data processing are first, the relatively small amount of input and output, and secondly, the irregularity with which the work is done. The input will rarely exceed 200 cards, and the output 1000 cards. The job is done only when a new design is being considered for production or modification of existing schedules is necessary.

**Input.** The input cards contain the program, the basic parameters which represent management decisions and can be varied to suit the problem, and the time cuts which represent the times at which manpower requirements are to be determined.

**Output.** The output cards contain the schedule of time into each assembly station and the average manpower required between successive time cuts.

**Method.** The essential method consists, for each plane, in (1) determining the delivery date; (2) calculating the amount of elapsed time in production flight, thereby determining the time into production flight; (3) calculating the elapsed time in each assembly line position and the time into that position.

### Symbols.

$i$	plane number
$j$	assembly station number; $j = 1$ denotes the final assembly station
$t_k$	time cut
$T_{ij}$	time into station $j$ for plane $i$
$H_{jk}$	cumulative manhours expended in station $j$ up to $t_k$
$M_{jk}$	average manpower required in station $j$ from $t_{k-1}$ to $t_k$
$m$	total number of stations
$n$	total number of planes
$T_{iD}^*$	delivery time of plane $i$ according to contractual commitments
$T_{iD}$	scheduled delivery time

### Equations

The following equations refer to portions of the flow diagram (see Fig. 4):

$$(1) \quad T_{iD} = a + bi - ce^{-di},$$

where  $b$  = the time between successive completions when full production rate is achieved and  $a = T_{nD}^* - bn$ . The quantities  $c$  and  $d$  are determined so that

$$T_{iD} \leq T_{iD}^* \quad \text{and} \quad \sum_{i=1}^n (T_{iD}^* - T_{iD})$$

is a minimum.

$$(2) \quad T_{io} = T_{iD} - f - ge^{-hi},$$

where  $f$  is elapsed time required in production flight to complete work on a plane after full production rate has been reached. The quantities  $g$  and  $h$  are determined by considering the elapsed time required on the first plane and the rate at which this elapsed time is to be reduced to  $f$ .

$$(3) \quad T_{ij} = T_{ij-1} - r - \frac{s(\gamma + 1)}{(\gamma^i + 1)} [T_{io} - T_{i-1,0} - r] \quad \begin{array}{l} \text{before a full production} \\ \text{line is reached} \end{array}$$

$$= T_{ij-1} - r = T_{i-1, j-1} \quad \begin{array}{l} \text{after production} \\ \text{line is filled,} \end{array}$$

## 9-10 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

in which  $r$  is the elapsed time required in each station after full production rate has been reached,

$s$  is a constant determined by considering the amount of elapsed time required on the first plane,

$\gamma$  is called a span decrease rate factor.

By assigning  $\gamma$  various values, control can be achieved of the rate at which the elapsed time required for successive planes in each station is diminished.

Let  $V_{ijk}$  represent the fractional portion of the work completed on aircraft  $i$  in station  $j$  at time  $t_k$ , assuming that the manhours are expended evenly during the time the aircraft remains in that station. Then

$$V_{ijk} = 0, \quad \text{if } t_k \leq T_{ij+1};$$

$$V_{ijk} = \frac{t_k - T_{ij+1}}{T_{ij} - T_{ij+1}}, \quad \text{if } T_{ij+1} < t_k < T_{ij};$$

$$V_{ijk} = 1 \quad \text{if } T_{ij} \leq t_k.$$

$$(4) \quad H_{jk} = \sum_{i=1}^m V_{ijk} K i^{-p}.$$

where  $K$  is the number of manhours required to complete the work on the first airplane at station  $j$ , and  $p$  is the learning curve factor for the

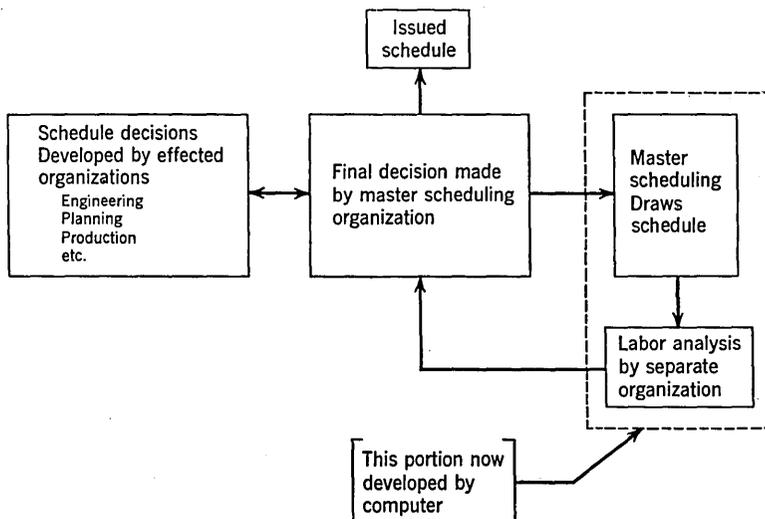


Fig. 3. Relation of production scheduling to overall management.

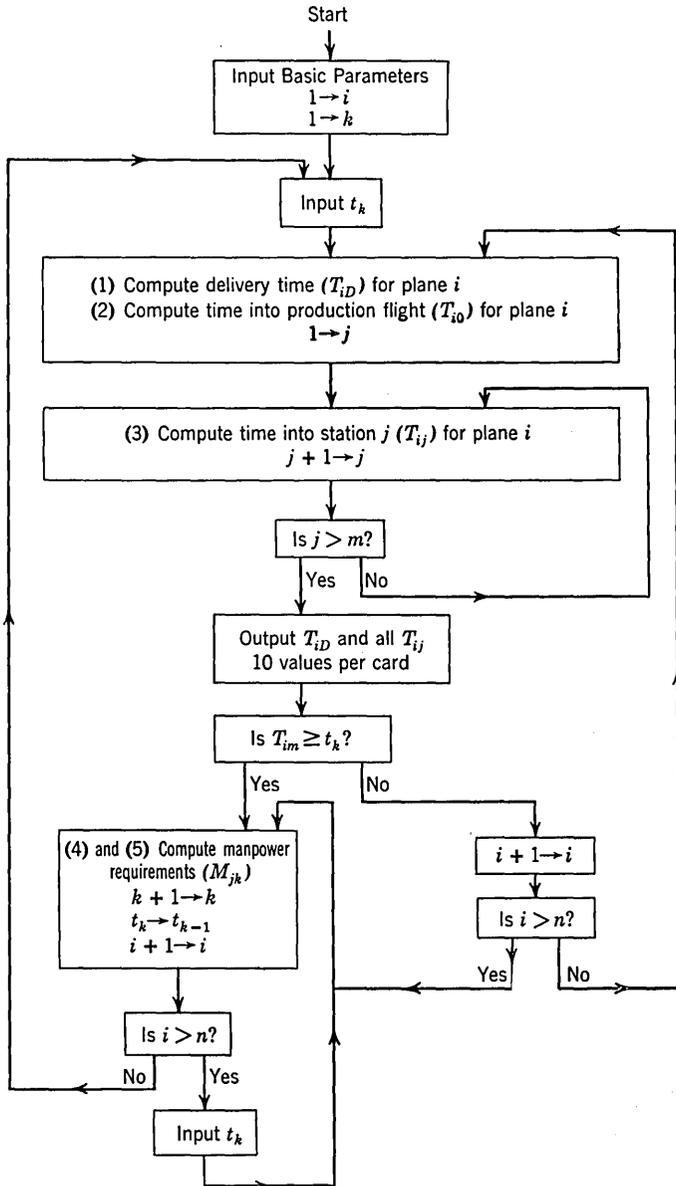


FIG. 4. Simplified flow diagram for production schedule computer run.

## 9-12 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

type of work performed in station  $j$ .

$$(5) \quad M_{jk} = \frac{H_{jk} - H_{jk-1}}{t_k - t_{k-1}}.$$

**Computation Time.** After the schedule is determined the average manpower requirements are calculated. Computation time on the IBM Type 650 computer is approximately one second per airplane per assembly station. Only one person is needed to run the problem. Programming and checkout required about 100 manhours.

### Flow Diagrams

Figure 3 shows how the computations fit into the overall management decision-making process. To determine a satisfactory schedule, it may be necessary to rerun the problem several times, each time modifying one or more of the basic parameters until a satisfactory schedule is produced. Figure 4 shows a simplified flow diagram of the actual computation run.

**Advantages.** The major advantage of using electronic data processing for schedules is that it is feasible to develop several schedules and labor forecasts within a period of four or five days and allow management to choose the schedule as desired. Formerly, only one schedule would be developed and that would be modified and reworked until a suitable schedule had been determined. This process took a considerably longer period of time.

### REFERENCES

1. M. L. Haselton and E. L. Schmidt, Automatic inventory system for air travel reservations, *Elec. Eng.*, **73**, 641-646 (1954).
2. J. L. Hill, Design features of Remington Rand speed tally, *Proc. Western Computer Conf.*, S-59, 155-162, April 1954.
3. W. L. Martin, A merchandise control system, *Proc. Western Computer Conf.*, S-59, 184-191, April 1954.
4. Electronic data processing in industry, Special Rept. No. 3, 206-214, American Management Association, New York.
5. Frank J. Andress, The learning curve as a production tool, *Harvard Bus. Rev.*, **32** [1], 87-97 (1954).

## Scientific and Engineering Applications

*R. T. Koll*

1. Introduction	10-01
2. Simultaneous Linear Algebraic Equations and Matrix Inversion	10-02
3. Characteristic Roots and Vectors	10-04
4. Linear Programming	10-06
5. Differential Equations	10-08
6. Statistical Analysis	10-10
References	10-12

### I. INTRODUCTION

Scientists and engineers from almost every technical field have learned to program and run their own problems. As a consequence of the close relationship that has developed between the source of problems and the computer, a large variety of problems has been placed on computers.

This chapter describes some of the mathematical methods that are frequently used and which are usually available to the problem originator in digital computer program libraries. The methods were selected for discussion because of their applicability to a large number of engineering and scientific areas. Some of the advantages and disadvantages of specific techniques are mentioned, and a rough estimate of the required solution time on a typical computer is given.

Formulas and more details on these techniques are covered in Vol. I, Chap. 14.

## 2. SIMULTANEOUS LINEAR ALGEBRAIC EQUATIONS AND MATRIX INVERSION (See Vol. I, Chap. 14)

**Direct Methods.** Of the numerous methods of solving systems of linear algebraic equations, certain ones are particularly adaptable to stored program computers. The well-known Gaussian elimination method that leads to a readily solvable triangular system of equations is easily coded and theoretically can solve any nonsingular system. The calculation is complicated by the accumulation of roundoff errors which are inherent in the computation and common to all methods of solution. Multiple precision arithmetic and other techniques are often adopted to preserve a sufficient number of significant digits in the final results.

*Matrix inversion*, in one sense, is a generalization of solving a system of linear algebraic equations that, by methods analogous to Gaussian elimination, requires about three times as much computation. Direct methods of solution are generally used to invert matrices of high order.

The *Jordan method* simply extends Gaussian elimination to the formation of an auxiliary matrix that, when the original matrix has been reduced to an identity matrix, is the inverse of the original matrix. The accumulation of roundoff errors in inverting a matrix by the Jordan method may be so rapid that the results obtained are of little value. The use of floating point arithmetic is practically required for inverting by the Jordan method high order matrices whose elements differ significantly in magnitude, and it eliminates the need for scaling the data.

The *von Neumann-Goldstine method* (Ref. 1) is an adaptation of the elimination scheme that combines optimum scaling of data and positioning for size by interchanging rows and columns of the matrix, to insure maximum accuracy. This technique involves a considerable amount of nonproductive data handling and requires lengthy and complex coding. It is a useful method for inverting poorly conditioned matrices in fixed point arithmetic. Simpler schemes involving only row and column interchanges are generally adequate for floating point operations.

Table 1 indicates the order of matrices that can be inverted by various techniques on several computers without using any form of auxiliary storage, as well as the time required for computing the inverse.

**Iterative Methods.** In contrast to Gaussian elimination and other direct methods, iterative methods assume an approximation to the solution and then repeatedly improve it in some mathematical sense. Each iteration consists of a fixed number of steps. Iterations are performed until a desired degree of accuracy is reached, the number of iterations required depending upon the particular problem. Roundoff errors may limit the degree of accuracy that can be obtained. An advantage of

TABLE 1. MATRIX INVERSION WITH COMMERCIALY AVAILABLE COMPUTERS

Method		Type of Matrix	Computer		Highest Order (approx.)	Approximate Time to Invert a Matrix of Order $n$
Mathematical Method	Arithmetic Mode		Type	No. of Words of Storage		
Jordan	Floating point	Arbitrary real	Magnetic drum	2000	40	$0.072n^3$ sec $n = 20, 10$ min
			High speed	4096	60	$0.5n^3$ msec $n = 20, 4$ sec
von Neumann-Goldstine	Fixed point	Real positive definite symmetric	High speed	4096	50	$n = 20, 4$ sec $n = 50, 8$ min
Elimination combined with positioning for size	Floating point	Arbitrary real	High speed	4096	60	$n = 20, 30$ sec $n = 50, 1$ min
	Double precision floating point	Arbitrary real	High speed	4096 8192	50 80	$n = 20, 4$ sec $n = 50, 5$ min $n = 20, 20$ sec

## 10-04 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

many iterative methods is that calculating errors at intermediate steps do not affect the final results. For iterative processes in general, miscalculated results of one iteration are equivalent to a new set of approximations to begin improving by successive iterations. For the iterative methods that are discussed here and for all others that may initially assume any arbitrary approximations, such errors are eliminated from the final results.

Two iterative methods for solving systems of linear algebraic equations are the Gauss-Seidel and the conjugate gradient. The *Gauss-Seidel method* in effect uses each equation of the system to compute a better approximation of one variable by substituting successively improved values of the other variables into the equation. The *conjugate gradient method* (Ref. 2) obtains the roots of the system by minimizing a specific function of the variables. In both these methods the original matrix of coefficients is unaltered by the computation. This permits eliminating unnecessary calculations when zero coefficients appear in the system of equations. For both methods it is a simple matter to restart the computation by using intermediate results.

The Gauss-Seidel method converges to the roots of the system when the system is positive definite symmetric or when each diagonal element is greater than the absolute value of the sum of the off-diagonal ones in the same equation. The larger the diagonal elements relative to the off-diagonal ones, the more rapidly the improved values converge. For systems particularly suited to the method, a solution may be obtained to the desired degree of accuracy by the Gauss-Seidel method in less than  $n$  iterations, where  $n$  is the order of the system. For other systems, many times  $n$  iterations may be required. The conjugate gradient method guarantees convergence in at most  $n$  iterations in the absence of roundoff errors. In many cases, roundoff errors necessitate that a few more than  $n$  iterations be performed. The conjugate gradient method may be applied to a symmetric system or in a somewhat different form to a non-symmetric system without altering the original coefficient matrix.

Table 2 indicates the order of systems of simultaneous equations that can be solved by various methods with several computers and the time of computation for a representative system.

### 3. CHARACTERISTIC ROOTS AND VECTORS

Most of the common methods of finding the characteristic roots and vectors of a matrix are subject to an excessive accumulation of roundoff errors for even relatively low order matrices and fail completely when repeated roots are encountered.

**Danielewsky's Method.** An iterative method that has been used with success on a magnetic drum computer for matrices of the fifth order

TABLE 2. SOLVING SYSTEMS OF SIMULTANEOUS LINEAR EQUATIONS WITH COMMERCIALY AVAILABLE COMPUTERS

Method		Type of System	Computer		Highest Order (approx.)	Approximate Time to Make Representative Computation <sup>a</sup>
Mathematical Technique	Arithmetic Mode		Type	Words of Storage		
Gaussian elimination	Double precision floating point	Arbitrary real	High speed	4096	600 <sup>b</sup>	3.6n <sup>3</sup> msec, regardless of N
Gauss-Seidel	Fixed point	Real positive definite symmetric or predominately diagonal	Magnetic drum	2000	100 <sup>c</sup>	n = 200, N = 8 hrs
	Fixed or floating point		High speed	4096	200 <sup>c</sup>	n = 90, N = 600, 40 sec/iteration
Conjugate gradient	Double precision floating point	Real symmetric	High speed	4096	200 <sup>c</sup> or more <sup>b</sup>	n = 150, N = 1500, 0.4 sec/iteration: 1 sec/iteration
		Nonsymmetric	High speed	4096	200 <sup>c</sup> or more <sup>b</sup>	n = 200, N = 12,000, 1 min/iteration <sup>b</sup>
						n = 200, N = 12,000, 2 min/iteration <sup>b</sup>

<sup>a</sup> n, order of system; N, number of nonzero coefficients.  
<sup>b</sup> Using magnetic tape for auxiliary storage.  
<sup>c</sup> A large number of coefficients being zero.

## 10-06 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

is Danielewsky's, which performs a series of transformations on the matrix so that the coefficients of the characteristic equation are obtained. By using double precision floating point arithmetic, results for matrices of the twelfth order can be satisfactorily obtained by this method.

**Jacobi's Method.** Considerably greater accuracy at the expense of a greater amount of computing at each iteration is obtained by using Jacobi's method, which, by a sequence of orthogonal transformations to diagonalize the matrix, yields the characteristic roots directly. Experience indicates that roundoff errors are kept reasonably low by Jacobi's method and are practically independent of the order of the matrix. Jacobi's method has the additional advantage of finding all roots even when roots are repeated. With Jacobi's method for real symmetric matrices, computations for matrices of the forty-fifth order can be made by a high-speed computer with 4096 words of storage; of the sixty-eighth order, with 8192 words of storage. About fourteen seconds of computing per iteration are required for a twentieth order matrix. The time per iteration is roughly proportional to the cube of the order of the matrix. The number of iterations required is usually close to the order of the matrix, although it may be significantly less for matrices of order greater than ten.

**EXAMPLES.** The complete processing times for four representative matrices, including the time required to get all the necessary information into and out of the computer as well as the time spent performing the computation itself, follow:

8th order matrix:	1.2 minutes
12th order matrix:	2.4 minutes
20th order matrix:	6.0 minutes
40th order matrix:	20 minutes

**Greenstadt's Method** (Ref. 3). From a computing point of view, this method is similar to Jacobi's. It has been developed as a method for finding the roots of arbitrary complex matrices. Its general applicability, however, has not been rigorously demonstrated.

### 4. LINEAR PROGRAMMING

**Problem Statement.** A problem in linear programming is defined mathematically as finding the values of  $x_j$  satisfying conditions of the form (see Refs. 4 and 6):

$$(1) \quad \begin{aligned} \sum a_{ij}x_j &= b_i, \\ \sum a_{ij}x_j &\leq b_i, \\ \text{or} \quad \sum a_{ij}x_j &\geq b_i, \end{aligned}$$

as well as the condition that

$$(2) \quad x_j \geq 0 \text{ for all values of } j,$$

such that some function

$$(3) \quad f(x) = \sum c_j x_j \text{ is a maximum,}$$

where  $a_{ij}$ ,  $b_i$ , and  $c_j$  are given constants. Problems requiring minimization rather than maximization of  $f(x)$  and problems not requiring restriction (2) for some or all of the variables can be reduced to the same form.

**Simplex Method** (see Refs. 4 and 5, and Vol. I Chap. 15 for details of the simplex technique). This is a general method of solving problems in linear programming. In the simplex method the coefficients and constants appearing in the constraint eq. (1) and in the function being maximized (3) are incorporated in a matrix array that also includes entries corresponding to slack and artificial variables introduced into the problem for computing convenience. A series of transformations is performed that ultimately leads to the optimum solution. Various calculations and tests are made to determine whether or not the optimum solution has been obtained, and to indicate successive transformations that lead to the optimum solution. The modified simplex method is basically the same mathematical technique altered in such a way as to facilitate efficient computation of large problems by means of electronic computers using some form of auxiliary storage, such as magnetic tape, for a large part of the data.

**Problem Size.** The size of a problem in linear programming is usually defined by  $m$ , the number of rows, and  $n$ , the number of columns, of the matrix array;  $m$  is the number of the constraint eqs. (1) in the problem, and  $n$  is the total number of variables in the problem plus the number of slack and artificial variables introduced for computational purposes. A magnetic drum computer, with 2000 words of storage, can solve by the simplex method problems where  $m \leq 30$ ,  $n < 60$ , and  $m(n + 1) \leq 1400$ . With the modified simplex method and storing data on magnetic tape, there is effectively no limit to the number of variables involved in problems the drum computer can accommodate. The approximate time,  $T$ , required by the drum computer to perform an iteration is

$$T = 90m \times n \text{ milliseconds.}$$

A high-speed computer with 4096 words of storage can solve by the modified simplex method problems involving 255 restrictions on a virtually unlimited number of unknowns. The approximate time required per iteration by the high-speed computer is

$$T = 1.5m \times n \text{ milliseconds.}$$

## 10-08 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

**The Transportation Problem.** The transportation problem is a specific type of linear programming problem that can be solved more economically by a special technique than by the simplex method. The technique is not limited to transportation problems in the literal sense, and it is applicable to any problem that can be completely defined by essentially the same tabular array of data.

*Tabular Form.* Unit shipping costs are presented in a rectangular table, each row associated with a specific destination, each column associated with a specific origin. Amounts to be shipped from each origin to each destination must be determined for a similar table that obeys the restrictions that each column total equals the amount of material available at the location associated with that column, and that each row total equals the delivery requirement of the destination associated with that row.

**Solution.** A large number of such tables will fulfill the requirements. Starting with one such table that is easily calculated, a special technique ascertains whether a more economical solution exists, and points out which way the table should be modified to decrease total shipping costs. This modified table is then formed, and the process is repeated until the test for determining whether a best solution has been obtained is satisfied. From this best solution a similar technique determines all equally good solutions.

**Problem Size.** A magnetic drum computer with 2000 words of storage can be used to solve transportation problems involving 100 origins and 450 total number of origins plus destinations. With the drum computer, an iteration for a problem having 7 origins and 85 destinations requires about 1.5 minutes. Most problems of this size are completely solved in less than an hour.

A high-speed computer with 4096 words of storage can be used to solve transportation problems involving 600 origins and as many as 5600 total number of origins plus destinations. The approximate time,  $T$ , required per iteration by the high-speed computer is approximately

$$T = 0.8m \times n \text{ milliseconds,}$$

where  $m$  is the number of destinations and  $n$  is the number of origins. For a problem with 7 origins and 85 destinations, the time per iteration is about one-half second.

## 5. DIFFERENTIAL EQUATIONS

### Ordinary Differential Equations

**Methods.** The numerical methods usually employed to solve ordinary differential equations evaluate the solution of a first order equation step

by step through a series of intervals in the independent variable. Most of the common methods are readily extended to a system of simultaneous differential equations of the first order, and they can thus be adopted to solve ordinary differential equations of any order. The accuracy of the solution depends upon the fineness of the interval in the independent variable and upon the order of the derivatives preserved by the method. To obtain accurate results over a wide range of values, the usual digital computer practice is to maintain fourth order accuracy in the method, and to choose an interval that will give the desired degree of accuracy for each particular problem. The methods that are subsequently described are generally applied in this way.

**Numerical Integration.** There are a number of methods for the numerical solution of ordinary differential equations based on formulas for numerical integration, using at each new interval values of the function obtained in preceding steps. These methods require determining the first few values of the functions by Taylor's series or some other suitable technique, using known initial values of the variables. These "starting values" must be computed to the number of significant figures desired in the solution. Their determination is an important and laborious part of the computation. In methods of this type it is not a very simple matter to change the integration interval except by doubling it. Two such methods are those of Adams and Milne. Of the two, Adams' is simpler to apply; its chief disadvantage is the lack of a suitable check on the accuracy of the computations and of the process itself throughout the calculation. A measure of the error at each step in Milne's method (Ref. 7) is easy to compute, and it can be used to control the integration interval, doubling the interval and reducing the number of integrations performed whenever the error becomes sufficiently small. When the error becomes too large, it is usually necessary to restart the computation with a smaller interval, by using the most recent computed values as new initial conditions.

**Increment Evaluation.** In contrast to the methods described above, the methods of Runge-Kutta and Gill (Ref. 8) are based on a set of formulas that evaluate the increment in the dependent variable for some increment in the independent variable, using only the values at the preceding step. Hence initial conditions suffice to begin the computation, and no special techniques are required to evaluate "starting values." Furthermore, the length of the interval may be modified in any way at any step in the computation. However, at each step four evaluations of the function appearing in each differential equation must be made. For complicated equations this may demand an excessive amount of computation. Another disadvantage is that a

## 10-10 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

measure of the error in either method requires computations independent of the process itself. The Gill method insures greater accuracy than the Runge-Kutta. The latter is subject to computational errors that do not recommend the method for general use in solving large systems, or for integrating over a wide range of values.

**Solution.** Large-scale computers can solve by Adams', Milne's, Runge-Kutta's, or Gill's methods, systems of simultaneous first order differential equations of at least the fiftieth order. The time required to compute a new value for each variable is about 4 milliseconds with Milne's method and about 3 milliseconds with Adams'. Since the bulk of the computation at each step in the methods of Runge-Kutta and Gill is devoted to evaluating the functions appearing in the system of differential equations, the time required by these methods is largely dependent upon the particular problem.

### **Partial Differential Equations**

The most general numerical method for solving partial differential equations that is applicable to digital computers replaces the partial derivatives by finite differences and then solves the difference equation. By this method the differential equation is effectively reduced to a set of simultaneous algebraic equations. During an iterative solution of these equations, successively improved values of the function are evaluated at discrete points of a grid that includes known boundary values of the function. This method gives good results for boundary value problems involving quasilinear partial differential equations of elliptic, parabolic, and hyperbolic types. For examples of applications of large-scale computers in solving partial differential equations, see Refs. 9 and 10.

## 6. STATISTICAL ANALYSIS

Digital computers are used in a wide variety of statistical analyses including simple correlation, first and second order partial correlation, autocorrelation and power spectrum analysis, analysis of variance, seasonal adjustment of time series, and factor analysis. One of the most fruitful statistical applications of digital computers is multiple linear regression and correlation analysis. This application, along with factor analysis, is singled out to demonstrate the ability of digital computers in the field of statistics.

**Multiple Linear Regression and Correlation Analyses.** A basic objective in this analysis is the expression of a dependent variable, for purposes of prediction or fitting, as a suitable linear function of various

independent variables. The regression function which expresses this linear relationship is so derived that it best approximates the sample observations of the dependent variable in a least squares sense. The regression function is considered suitable if, apart from what may reasonably be attributed to chance, the values predicted by it for the dependent variable account for a large part of the dependent variable's sample variance.

The first step in the analysis is to calculate the sample means, standard deviations, and simple correlation coefficients. A symmetric matrix is formed whose elements are the simple correlation coefficients. Partial and multiple correlation coefficients, the regression coefficients and their standard errors, and the standard error of estimate are readily computed from the inverse of the simple correlation matrix.

A high-speed computer can perform such an analysis for as great a total number of observations as 900,000. The number of independent variables carried throughout the entire computation is restricted by the matrix inversion program used. The complete processing for a problem consisting of 200 observations of each of 30 variables requires about 10 minutes. Multiple regression analysis can be performed by a magnetic drum computer for problems involving as many as 30 variables.

**Factor Analysis.** Another type of statistical analysis often entailing computation of such volume that the use of a high-speed computer is desirable is factor analysis. Factor analysis is a branch of statistical theory concerned with resolving a set of descriptive variables in terms of a small number of categories or factors by an analysis of their inter-correlation. The aim of factor analysis is to account for the observed correlation among the variables in terms of the smallest number of factors with the smallest possible residual errors. Its purpose, as contrasted with multiple regression and correlation analysis, is economy of description rather than prediction.

Various mathematical methods of performing factor analysis have been developed. A method commonly applied to high-speed computers is the principal components method. As in the case of multiple linear regression and correlation analysis, the principal components method of factor analysis requires the calculation of elementary statistics and the formation of a simple correlation matrix. The characteristic roots of the simple correlation matrix are then computed and interpreted to identify the significant factors. Since the most significant part of the computation is finding the characteristic roots of a matrix, the reader is referred to Sect. 3 for an estimate of the time required by a high-speed computer to perform factor analysis by the principal components method.

## 10-12 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

### REFERENCES

1. J. von Neumann and H. H. Goldstine, Numerical inverting of matrices of high order, *Bull. Am. Math. Soc.*, **53** (11), 1021-1099 (1947).
2. M. R. Hestenes and E. Stiefel, Methods of conjugate gradients for solving linear systems, *J. Research Natl. Bureau Standards*, **49** (6), 409-436 (1952).
3. J. Greenstadt, A method for finding roots of arbitrary matrices, *Math. Tables and Other Aids to Computation*, **IX** (50), 47-52 (1955).
4. A. Charnes, W. W. Cooper, and A. Henderson, *An Introduction to Linear Programming*, Wiley, New York, 1953.
5. G. B. Dantzig, Wm. Orchard-Hays, *et al.*, various papers on the simplex method, Rand Corporation, Los Angeles, Calif.
6. K. Eisemann, Linear programming, *Quart. Appl. Math.*, **XIII** (3), 210-232 (1955).
7. W. E. Milne, *Numerical Solution of Differential Equations*, Wiley, New York, 1953.
8. S. Gill, A process for the step-by-step integration of differential equations in an automatic computing machine, *Proc. Cambridge Phil. Soc.*, 1951.
9. J. Sheldon and L. H. Thomas, The use of large scale computing in physics, *J. Appl. Phys.*, **24**, 235-242 (1953); The numerical solution of a partial differential equation on the IBM Type 701 electronic data processing machine, *Proc. Assoc. Computing Machinery*, 1952.
10. W. J. West, W. W. Garvin, and J. W. Sheldon, Solution of the equations of unsteady state two-phase flow in oil reservoirs, *Trans. Am. Inst. Mech. Eng.*, 1953.
11. F. S. Beckman and D. A. Quarles, Jr., Multiple regression and correlation analysis on the IBM Type 701 and Type 704 electronic data processing machines, *Am. Statistician*, **10**, 6-9 (1956).

## Handling of Non-Numerical Information

*M. E. Maron*

1. Introduction	11-01
2. Performing Logic on a Digital Computer	11-02
3. Game Playing Machines	11-11
4. The Machine Translation of Languages	11-13
5. Automatic Literature Searching and Retrieval	11-16
References	11-19

### I. INTRODUCTION

**Characteristics of Non-Numerical Problems.** Digital computers may be used to handle problems for which strict mathematical methods of solution are not known. Such problems involve many more logical operations than arithmetic operations and allow a spectrum of results rather than a single exact solution. These problems are, by and large, problems involving non-numerical information such as logic, games, language translation, and literature searching. This chapter will describe (a) some of the problems, (b) methods of attack, and (c) results involved in the computer handling of non-numerical information.

**Brain Behavior.** One of the really exciting facets of work on non-numerical computer applications lies in the insights that are to be gained from the handling of complex problems—insights into the nature of automatic problem solving. Perhaps the primary characteristic of brain behavior is the ability to solve problems, not just to solve problems in

## 11-02 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

the sense of finding answers by using old methods, but to solve new and different problems by using new and different methods of solutions. Human beings solve many problems daily without knowing how the solutions were found, certainly without executing a rigidly prescribed sequence of logical and arithmetic operations. The task of programming complex non-numerical problems for a computer could lead to insights into the nature of problem solving and to an understanding of the functioning of the brain.

Computers are useful tools for research in brain behavior because brain models can be simulated on a digital computer, or better still, special purpose machines can be built such as those of Grey Walter (Ref. 1) and Ashby (Ref. 2), which behave in ways resembling brain activity. This means that complex models of the brain, which cannot be analytically evaluated, could be simulated by a digital computer and could thereby allow the machine's behavior to make explicit those features of the model which are only implicit in the design.

This sort of work also will lead eventually to new concepts of computer design, to new principles of organizing digital computers so that they will become even more useful tools both in automatic control and in the automatic handling of non-numerical information.

### 2. PERFORMING LOGIC ON A DIGITAL COMPUTER

**Development of Logic** (Ref. 3). The subject of logic, which dates back 2200 years to Aristotle's theory of syllogistic reasoning, remained essentially unchanged for over twenty centuries. Then, beginning with the work of Boole, Schröder, Frege, and Peano in the years around 1850, and accelerated with the publication of the monumental *Principia Mathematica* by Whitehead and Russell in the early part of this century, the field of logic mushroomed and today it encompasses a very wide area, including all classical mathematics.

**Mechanization of Logic** (Ref. 4). The mechanization of logic does not have such a long history; the first logic machine dates back only 150 years, although Leibnitz, in the seventeenth century, envisioned a universal symbolism and a calculus of reasoning for the mechanical manipulation of non-numerical information. The British statesman Charles Stanhope first constructed a machine for solving syllogisms at the close of the eighteenth century and in 1947, two Harvard students, Kalin and Burkhardt (Ref. 5), built the first electromechanical logic machine for the solution of logical problems more complex than the syllogism. Since that time several varieties of logic machines have appeared, including one by McCallum and Smith (Ref. 6) in England, one by Burke,

Warren, Wright, and Michle (Refs. 7, 8) and one by Maron (Ref. 23). The last machine will be described as an illustration.

**The Nature of Logic.** The following factors characterize logic:

(a) The expressions of logic are tautological, i.e., true by virtue of their form alone and completely empty of content.

(b) All logic is reducible to sentences of the type "*A or not A*," "*A is equivalent to A*," etc., and they are true, regardless of whether the sentence that is substituted for *A* is true or false.

(c) The primary occupation of the logician is to find expressions which have this tautological character and to prove that the expressions in question are, in fact, tautological. Such tautological expressions are theorems of logic.

**Methods of Proof.** Basically, there are two ways of proving that an expression is a theorem. Each method has given rise to a different kind of logic machine.

1. *Evaluation Procedure.* The evaluation method consists in using a set of rules which describe precisely and in complete detail how to determine whether an expression is a theorem. These rules function in much the same way as recipes in that they dictate step by step the operations to be performed, starting with an arbitrary expression whose validity is to be analyzed and ending with a yes or no answer to whether the expression is a theorem. Such a set of rules is called a *decision procedure*. Decision procedures have been found for only some parts of modern logic, and for other parts of logic it has been proved that no decision procedures are possible. Decision procedures, where they do exist, are capable in principle of being mechanized by means of digital circuitry. Hence, one may build or program a computer to determine automatically whether or not an expression is a theorem of logic.

2. *Free Derivation.* This method consists in deriving theorems from axioms or from other theorems which are themselves tautological. See section entitled The Mechanization of Deductive Procedures below.

## The Sentential Calculus and Its Decision Procedures

**Definitions.** The *sentential calculus* is that part of logic whose basic symbols are two-valued variables which may be thought of as sentences whose truth values must be either true or else false. An *atomic sentence* is one which contains no proper part which is itself a sentence. The atomic sentences designated as *p*, *q*, *r*, etc., may be connected by certain logical connectives to form molecular sentences. The logical connectives are designated by such familiar terms as *not*, *and*, *or*, and *if, then*. The simplest example of obtaining a molecular sentence from

## 11-04 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

an atomic sentence is by means of negation (e.g., *not p* is obtained from *p* by negation). Similarly, given any two sentences *p* and *q*, one can connect them with such terms as *and*, *or*, and *if, then* and thereby obtain the following molecular sentences: *p and q*, *p or q*, and *if p, then q*.

The process of constructing molecular sentences by means of the logical connectives can be continued indefinitely, giving rise to such expressions as: *(p or not q) and [(not r and not s) if and only if t]*. If the truth values of the atomic components are known, there are rules of logic by means of which the truth value of molecular sentences can be determined.

**Truth Table Analysis.** Tables 1 and 2, called truth tables, give some of the rules for calculating the truth values of molecular sentences given the truth values of their atomic components (see Chap. 17). The symbols T for true and F for false correspond to the 1 and 0 used in logical design. The Boolean notation for various logical operations is given in parenthesis under the column headings.

TABLE 1. TRUTH TABLE FOR ATOMIC VARIABLE *A*

<i>A</i>	Not <i>A</i> ( $\bar{A}$ )
T	F
F	T

TABLE 2. TRUTH TABLE FOR FUNCTIONS OF ATOMIC VARIABLES *A* AND *B*

<i>A</i>	<i>B</i>	<i>A or B</i> ( $A + B$ )	<i>A and B</i> ( $A \cdot B$ )	<i>If A, then B</i> ( $\bar{A} + B$ )	<i>A if and only if B</i> ( $A \cdot B + \bar{A} \cdot \bar{B}$ )
T	T	T	T	T	T
T	F	T	F	F	F
F	T	T	F	T	F
F	F	F	F	T	T

**EXAMPLE.** Use of truth tables to determine the truth value of a molecular sentence. Consider the sentence *p or (if q then r)*. If *p* and *q* are true and *r* is false, what is the truth value of the above sentence? The tables tell us that *if q then r* is false since *q* is true and *r* is false, but since *p* is true, the entire sentence is true.

Truth table analysis can be used to determine whether an expression is a theorem of logic. An expression in the sentential calculus is a theorem if it is true regardless of the individual truth values of its atomic components. If an expression contains *n* different variables, there are  $2^n$  possible combinations of truth and falsity which the variables might

assume. This then is the decision procedure for the sentential calculus; in order to determine whether an arbitrary molecular sentence of  $n$  different variables is a theorem, examine with the aid of truth tables each of the  $2^n$  combinations of truth values of the atomic sentences. If the molecular sentence is never false for any of the  $2^n$  combinations, then it is a theorem of logic.

### The Mechanization of the Decision Procedures for the Sentential Calculus

Shannon (Ref. 9) showed that the logical structure of electric switching circuits (Chap. 17) is essentially identical to the logic (viz., sentential calculus) discussed above. With only series and parallel connections and relays it is possible to construct the electrical analog for any complex expression in the sentential calculus. This makes it possible to construct an electromechanical "analog" decision machine.

The following requirements must be met in order to mechanize physically the decision procedure.

1. There must be a flexible way of combining elementary circuits to form complex circuits, i.e., to form the electrical equivalent of the expression to be tested.

2. There must be some way of automatically examining all  $2^n$  possible cases and testing each of the cases to see whether the circuit is open or closed.

3. There must be some controls so that the machine may stop if it detects a false case and some way of indicating whether or not there were any false cases.

**A Decision Machine.** A decision machine constructed in 1952 (Ref. 23) can handle problems involving a maximum of eight variables. Each variable is represented by a wire contact relay with eight pairs of contact points. Seven of the eight pairs of relay points are brought out to a control panel and the eighth pair of points is used to control a light on the face panel of the machine. This light indicates whether the relay is *on* or *off*; that is, it indicates whether the corresponding sentence is *true* or *false*. Adjacent to the input variable hubs (jacks) on the control panel are the hubs, which connect with the inputs and outputs of the logical connectives. Each *or* connection is simply a set of hubs connected in parallel and each *and* connection is a set of hubs connected in series. The connectives for these circuits are wired on the underside of the control panel assembly as indicated in Figs. 1 and 2.

Thus, if one takes the normally open and common terminals from the  $p$  variable and the corresponding points from the  $q$  variable and connects

## 11-06 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

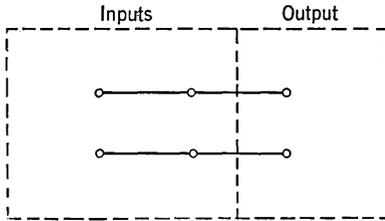


FIG. 1. *Or* circuit. Each input has two connection wires.

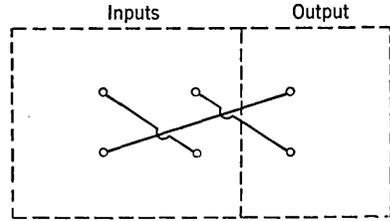


FIG. 2. *And* circuit. Each input has two connecting wires.

them to the input hubs of an *or* circuit, one obtains the compound electric circuit corresponding to the expression  $p$  or  $q$ . In order to program *not p* or *not q*, the common and normally closed points of each of the

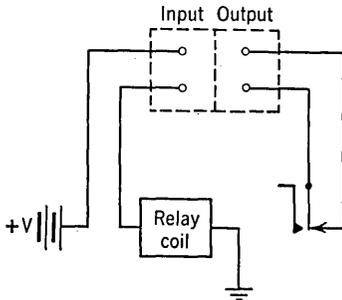


FIG. 3. Negation circuit. If the input hubs are connected by a closed circuit (true sentence), then the relay is energized and the output hubs constitute an open circuit (false sentence).

relays are taken. The hubs representing the input and output for the negation circuit of Fig. 3 also appear on the control panel. By combining the above connections the electrical equivalents for *if*, *then*, and *if and only if* were wired and the corresponding input and output points were brought out to the control panel.

The completed control panel has eight input variables each of which has seven common, normally open and normally closed points. It has fourteen *or* circuits, fourteen *and* circuits, ten negation circuits, six *if*, *then*, and four *if and only if* circuits. Thus, given the input variables and the connectives, any expression which contains no more than eight variables (and no more than about 50 connectives) can be programmed on the control panel by connecting the correct hubs with connecting wires.

For any complex expression programmed on the machine, there is always a single output of two leads. These are two ends of the relay circuit which represents the logical expression being tested. One end is connected to a 40-volt source and the other end is connected to a relay which will be energized and pick up if the circuit is closed (expression true), and it will not pick up if the circuit is open (expression false). A light on the face of the machine indicates the state of this indicator relay. Automatic examination of each of the  $2^n$  possible combinations of truth values of the  $n$  atomic sentences which make up the complex expression

is accomplished by connecting an eight-stage binary counter to the eight input variable relays, one stage to each variable. The binary counter receives impulses from a set of relays which pick up and drop and thereby send a regular sequence of impulses to the counter. The output of the final stage of the counter is connected to the end of a latch relay. When the counter runs through all  $2^n$  cases the latch relay picks up and cuts off the power to the pulse generator and thus stops the instrument. By touching a reset switch the latch relay drops out and the unit can begin to operate again.

The truth value of a complex expression is automatically tested simply by determining whether the corresponding electric circuit is open or closed. The solution time on this machine is a function solely of the number of basic variables and is independent of the kinds or number of logical connectives.

**Application of the Decision Machine.** Whenever information (whether it be in law or business) is unambiguous and can be formulated in terms of sentences connected by the truth functional connectives, then that information can be processed automatically on a logic machine for the following kinds of analysis:

1. To determine whether or not the information is logically consistent, that is, whether or not it contains a contradiction.

2. To make decisions about logical implication; that is, to decide whether some arbitrary information is logically implied by other given information. The machine can do this because logic tells us that " $B$ " is logically implied by " $A$ ," if " $A$  implies  $B$ " is a tautology.

3. To determine whether two classes of information are logically equivalent. Since there is a sameness of structure between the type of logic that this machine can handle and the electrical properties of switching circuits, the machine can be programmed to determine whether two different switching circuits are electrically equivalent.

**EXAMPLE.** To illustrate one type of problem which the decision machine can handle, consider the following. Assume that information  $A$ , which is contained in sentences 1 through 5, is true.

1. If the 1958 financial report for the Acme Electronics Co. indicates a net gain, then basic research at the company will be increased, and either a stock dividend will be declared or employee benefits will be increased.

2. If, however, the 1958 financial report does not indicate that Acme has enjoyed a net gain, then either plant  $P$  will be closed or a government contract will be obtained for Acme.

## 11-08 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

3. There will be no government contract without an increase in basic research at Acme, and basic research cannot be expanded without an increase in scientific research personnel.

4. If it is the case that either employee benefits are not increased or the company prestige drops, then it will be impossible to obtain new scientific personnel.

5. The prestige of the Acme Electronics Co. will fall if and only if the stockholders do not obtain a stock dividend.

Given this information, determine whether the following information is true:

$D_1$  If plant  $P$  is closed, then the financial report did not indicate a net gain.

$D_2$  If there is no stock dividend, then basic research at Acme will not be increased.

One notes that the above information is made up of eight basic or atomic sentences. These sentences are the following:

$p$  = The 1958 financial report for the Acme Electronic Co. indicates a net gain.

$q$  = Basic research at Acme Electronic Co. will be increased.

$r$  = A stock dividend will be declared.

$s$  = Employee benefits will be increased.

$t$  = Plant  $P$  will be closed.

$u$  = A government contract will be obtained for Acme.

$v$  = Scientific research personnel at Acme will be increased.

$w$  = The prestige of the Acme Electronics Co. will fall.

These elementary sentences are put together by means of logical connectives such as *not*, *or*, *and*, and *if, then* so as to make the information previously stated. The information merely has to be programmed on the control panel, and in approximately 30 seconds, the machine tells us that  $D_2$  is true since it logically follows from the input information  $A$ , whereas  $D_1$  does not logically follow from  $A$ . Many other decisions are also possible on the basis of the input information  $A$ —it is not limited to  $D_1$  and  $D_2$ .

**Automatic Programming** (Ref. 24). In 1953 a small multiplier-accumulator was built which was controlled automatically by a logic machine so as to solve elementary probability problems (see Ref. 6). It can compute the probability of occurrence of any logical combination of  $n$  independent events  $E_1$  to  $E_n$ , for example ( $E_1$  and not  $E_2$ ) if and only if ( $E_3$  and not  $E_4$  or  $E_5$ ), given the probabilities for the individual events.

The initial probability values are set up by means of selector switches on the multiplier. Each switch is gang-connected so that if a probability  $p$  is set up on one switch, its complement  $1 - p$  is automatically set up. The combination logic machine and multiplier-accumulator operate in the following manner. Assume the problem is to compute the probability of occurrence of the event  $E_1$  and either  $E_2$  or not  $E_3$ . The problem is programmed on the machine, and the probabilities  $p$ ,  $q$ , and  $r$  associated with the events  $E_1$ ,  $E_2$ , and  $E_3$  are set on the multiplier selector switches. The logic machine begins to examine all  $2^n$  combinations of the  $n$  variables.

A single built-in program for the multiplier consists of the operation "multiply and accumulate." As the logic machine goes through the  $2^n$  cases, it gives a signal for all cases that are logically compatible with the original expression, that is, for all cases where the programmed expression is *true*. When the multiplier-accumulator receives the *true* signal, it multiplies the corresponding probabilities (or their complements) and accumulates the result, as shown in Table 3. When the machine has gone

TABLE 3. PROGRAMMING OF PROBABILITY PROBLEM

Case	Logical Value	Action
1. $E_1$ and $E_2$ and $E_3$	True	Multiply $p$ , $q$ , $r$ and accumulate
2. $E_1$ and $E_2$ and not $E_3$	True	Multiply $p$ , $q$ , $1 - r$ and accumulate
3. $E_1$ and not $E_2$ and $E_3$	False	Do nothing
4. $E_1$ and not $E_2$ and not $E_3$	True	Multiply $p$ , $1 - q$ , $1 - q$ and accumulate
5. Not $E_1$ and $E_2$ and $E_3$	False	Do nothing
6. Not $E_1$ and $E_2$ and not $E_3$	False	Do nothing
7. Not $E_1$ and not $E_2$ and $E_3$	False	Do nothing
8. Not $E_1$ and not $E_2$ and not $E_3$	False	Do nothing

through  $2^n$  cases the final probability value in the accumulator is displayed.

**Limitations.** A logic machine has limitations for handling ordinary language. The larger part of the language that is used in science and everyday life is so complex and vague that it cannot be mapped into the language of the sentential calculus for subsequent handling. In many cases, the logical structure within a sentence must be analyzed in order to determine the logical relationships of consistency, implication, equivalence, validity, etc., and for such linguistic situations this type of logic machine would be inadequate.

Because most of logic and mathematics involves problems in which there are no decision procedures, and because in those cases where they do exist they are so complex as to be impractical, the usual proof pro-

## 11-10 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

cedure consists in free derivations rather than the evaluations by decision procedures.

### **Proof by Free Derivation**

**Method.** Starting with (1) an initial set of axioms and those theorems already proved and (2) a set of inference rules, the inference rules are then applied sequentially until the desired expression results. Thus the proof is the sequence leading from the initial axioms to the final result or theorem via continued application of the few inference rules.

**Inference Rules.** The most common inference rules in logic are those of modus ponens and of substitution. *Modus ponens* states that if  $A$  is an axiom or theorem and " $A$  implies  $B$ " is an axiom or theorem, then  $B$  is a theorem. *Substitution* permits the substitution of any expression for any variable in any theorem, provided that the substitution is made uniformly throughout the theorem wherever the variable appears.

Inference rules are not strict prescriptions as to how to proceed, but rather descriptions of operations that are allowable, so consequently, there are no explicit rules dictating in what sequence they are to be used. The inference rules are used in a groping fashion, by trial and error, by intuition, by creative insight, until the desired expression turns up. Even though an expression is a theorem there is no guarantee that it will be found, with the technique of free derivation. The only guarantee is that, if a chain is found leading from axioms and theorems via the inference rules to the desired expression, the expression in question must be a theorem.

### **The Mechanization of Deductive Procedures**

Newell, Shaw, and Simon (Ref. 10) have programmed this method of free derivation on a high-speed digital computer, the Johnniac for which the sentential calculus is used as the logical language. Programs have been written for the Johnniac for deriving theorems by the technique of free derivation. The machine is supplied (1) an axiom set for the sentential calculus (taken from Whitehead and Russell (Ref. 11)), (2) a set of three inference rules, and (3) some general operating guides called heuristics, which describe to the machine how to grope for proofs. For example, in attempting to find a proof for some expression  $B$ , the machine is instructed to look among its list of axioms and theorems for " $A$  implies  $B$ " and if that can be found, then to look for  $A$  in the list of theorems. If, however, the machine can find " $A$  implies  $B$ ," but cannot find  $A$ , then it executes a routine which searches for some other expression  $C$  which is similar to  $A$ ; the machine has rules instructing it how to transform  $C$  in an attempt to obtain  $A$ .

Thus a machine can be supplied with strategies to follow in order to be efficient in groping. With improved heuristics and machine strategies it is hoped that the "proving" capabilities of the machine will be improved.

### 3. GAME PLAYING MACHINES

**Approaches.** During the past decade a variety of small-scale game machines has been developed. There have been Nim machines (Ref. 12), Tic, Tac, Toe machines, a machine to play Hex, and coin matching machines (Ref. 13). The last machines will play the game of matching coins with a human opponent by using a strategy that searches for patterns in their human opponents' betting behaviors. They have racked up a remarkable ratio of wins to losses. There also has been much interest in programming a general purpose digital computer to play other more complex games, in particular, checkers and chess (Refs. 14, 15). Strachey, in England (Ref. 16), has programmed checkers on the Manchester machine, which played a very good game, and in this country a modified form of chess has been programmed on the Maniac with good results.

**The Analogy between Logic and Games.** Many games and puzzles are reducible to simple problems in logic, and consequently there is an analogy between the two. The analogy between chess and sentential calculus is shown in the following table:

<i>Sentential Calculus</i>	<i>Chess</i>
Inference rules of logic	Rules for moves
Initial axioms	Initial position of pieces
Theorems	Subsequent arrangement of pieces
Theorem derivation	Allowable sequences of moves

To complete the analogy the following two problems might be contrasted: (1) determining whether some arbitrary arrangement of pieces on the chess board could be arrived at by use of legal moves and (2) determining whether an arbitrary expression in sentential calculus is a theorem. For example, any position with both black bishops on squares of the same color would be an impossible arrangement unless a pawn had been promoted.

**Machine Methods of Solution.** Most games have a discrete nature and the positions and moves can be described precisely. Hence, it is relatively easy to (1) build a special purpose game machine with digital equipment or (2) program games on a general purpose digital computer. Such machines can be made to execute legal moves for almost any game.

## 11-12 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

The real problem is to have the machine play a competent game, that is, to use a good strategy of play. For many games a complete mathematical theory has been developed which describes an optimal strategy, and in such cases it is possible in principle to have a machine play an optimal game. For games whose mathematical theory is relatively simple as, for example, the game of Nim, it is relatively simple to build a machine which will play the best possible game against any opponent. Thus, just as a machine can decide automatically whether an expression is a theorem given a decision procedure, so also a machine can play an optimal strategy given a mathematical theory of the game. Because of the enormous storage requirements and the lengthy computing time, it is not practical to build machines for games such as chess whose optimal strategy is very complicated, even though this is possible in principle.

Consider now those games for which no mathematical theory has been developed and games with complicated optimum strategies. In such cases it is possible to develop heuristic methods which will guide the machine along the path of a reasonably good strategy. Such general principles of play correspond to the general principles of derivation that Newell *et al.* (Ref. 10) have used to derive theorems on the Johnniac. Again, there is no guarantee that the machine will win a game, but if the designer (programmer) has a good insight into the game, he will be able to formulate a reasonably good machine strategy.

**Programming Chess on a General Purpose Computer.** Kister, Stein, Ulam, Walden, and Wells (Ref. 17) performed a series of experiments in which a modified form of chess was played on the Maniac I. The game is played on a 6 by 6 board, omitting the bishops and using 6 pawns on each side. Castling is not allowed, but the promotion of a pawn is allowed. The machine is programmed to "look ahead" two moves, that is, two by white and two by black. The machine considers for each legal move of its pieces the possible consequences in terms of the opponent's counter moves, and for each of those subsequent arrangements it considers again the consequences of its very next move. The number of chains that must be examined is very large even for a 6 by 6 game with only two looks ahead, and it requires an average of 12 minutes for the Maniac to consider all such chains.

The real problem is to program the computer to make an evaluation for each chain it examines, so as to be able to select the most promising alternative. Two basic criteria have been used for evaluating the position for each chain of four moves: (1) material advantage and (2) mobility (see Ref. 17). Material advantage involves an evaluation of the kinds and numbers of pieces that result after any exchanges for each chain. A minimax method is used to evaluate mobility in terms of the

number of legal moves available to the player after the first, second, and third moves of the chain. Although these evaluation techniques might appear crude, the above machine could be compared to a human player who has average aptitude for the game and experience amounting to 20 or so full games played. Clearly the machine could play an even better game if it were able to look "three deep" instead of two, and this would be quite feasible with a faster machine.

If the machine is to play a better game, an improved method of evaluation might be programmed, but the problem of describing evaluation criteria in precise and unambiguous language is extremely difficult. A competent chess player can make an evaluation of a chess move, but it is not something that he can easily describe in precise language; such a description is required if a computer is to be programmed. A chess master is able to look many many moves deep (sometimes as much as 20 moves in depth), but he certainly does not consider all possible chains in such detail. Rather he is able to weight the chains intuitively as he proceeds to examine them. He quickly rejects those which do not look promising and examines only the most promising chains in detail. Perhaps this method of weighting the chains could be formulated so that a machine also could concentrate preferentially on the highly weighted alternatives while quickly rejecting the others.

The use of a general purpose computer as a chess playing device seems to be one good way of empirically verifying the effectiveness of possible chess strategies. However, the really valuable aspect of mechanical chess is that it provides a wonderful tool with which to experiment and obtain data concerning problem solving.

#### 4. THE MACHINE TRANSLATION OF LANGUAGES (Refs. 18 to 20)

**Code Conversion.** The problem of machine translation of languages is essentially the problem of code conversion, i.e., the problem of converting information which is represented in one code to identical information represented in a different code. (See Chap. 2.) When dealing with strict codes, the conversion problem is relatively simple and straightforward to mechanize because there are strict, definite, and explicit rules that describe which code patterns are equivalent.

Much work has already been done in clarifying the problem of machine translation and in obtaining a set of linguistic rules for making a translation by machine. Most of this work concerns the translation from Russian to English. The prospects for good, intelligible, accurate translation by means of a digital computer (forgetting for the moment considerations of cost, speed, etc.) depend upon the type of information to be translated. The primary function of ordinary language as it is

## 11-14 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

used in scientific discourse is to convey information, and the prospects of obtaining good machine translations of scientific literature are excellent.

In the case of so-called natural languages, such as English, Russian, and German, there is no exact one-to-one correspondence between the words of different languages; even if there were, there still would be a problem because the conversion must keep *meaning* invariant and meaning is not a function of words in isolation. Strictly speaking, only sentences have meaning, and the meaning of a sentence is a function of the semantic content of its individual words, and the syntactical form of the individual words. The content and grammatical function of many words vary with their position or order relative to the other words in the sentence. Since the "code conversion" must keep the meaning invariant, and since meaning is a function of the form, content, and order of individual words, a number of linguistic problems must be solved in order to have a machine translate from one natural language to another. Three of the key linguistic problems are the *recognition* of individual words, the *translation* of recognized words, and the *transposition* of translated words.

**The Recognition Problem.** In a strongly inflected language such as Russian, the nouns, verbs, and adjectives may appear in a multitude of forms (Ref. 21). That is to say, there are many different kinds of endings which may be attached to the stem of a word in order to indicate such grammatical information as number, gender, case, and tense. Therefore, when a word appears in any of a variety of forms, the primary machine problem is that of recognition, i.e., of determining what the word actually is, independent of its inflectional ending.

One computer solution to the problem of recognition is to strip the endings off the word in question, letter by letter, until the stem is recognized by comparing it with entries in a "stem dictionary." The endings then are compared with the entries in an "ending dictionary" so that one has not only the information content of the word but also its grammatical function.

Another solution to the recognition problem concerns the use of a *paradigm dictionary*, one which lists all words with all their inflectional forms. This solution requires more machine storage since a paradigm dictionary would be much greater in size than a stem and ending dictionary together. However, it might well involve fewer machine operations in order to recognize any given word.

**The Translation Problem.** Once a word has been recognized it must be translated, at which point the problem of semantic ambiguity arises. Ambiguity arises out of the fact that the same word may have several

meanings. For example, the word "fair" as an adjective may mean beautiful and as a noun may mean festival. It may be argued that this ambiguity is resolved once the grammatical function of the term in question is identified, as for example, when it is known that the word "fair" is an adjective. However, an even more subtle ambiguity remains. *Webster's New Collegiate Dictionary* lists fourteen separate and distinct meanings for the word "fair" as an adjective (it may mean beautiful, plausible, gracious, ample in size, desirable, elegant, light, blond as opposed to brunet, impartial, clean, etc.).

Studies have indicated that the semantic ambiguity of a word can be reduced considerably when the word is viewed in context. Usually knowledge of the preceding word or the subsequent word helps to reduce the ambiguity enormously. In order for a machine to eliminate semantic ambiguity it would have to "know" hundreds of rules concerning the kinds of nouns which follow certain prepositions, the kinds of adjectives which give specific meaning to certain nouns, etc., and these rules have not yet been formulated. In order to determine how context reduces ambiguity in particular instances, it would be necessary to do an elaborate analysis of language as it is ordinarily used. But given such rules a machine could be instructed to select the proper meaning for a word in a particular instance as a function of the other words used with it.

**The Transposition Problem.** It is often necessary to transpose the linear order of terms so as to preserve the meaning of a sentence when translating from one language to another. In the case of a highly inflected language such as Russian, this transposition (at least in the case of scientific Russian) is not critical since a straight word-by-word translation is usually quite intelligible. However, in the case of German, for example, meanings will be lost in the translation of words alone; it is necessary to transpose the order of the German into the usual word order of English (i.e., subject before verb, adjective before noun, and direct object after verb).

This problem of automatically reordering the words from the German to grammatically correct English can be best carried out clause by clause (Ref. 22). That is, the individual terms must be unscrambled within their respective clauses. Thus there are two aspects to this problem: (1) recognizing the beginning of a clause, the end of a clause, and the type of clause; (2) recognizing the grammatical function of each word in the clause and rearranging the words so that they will conform to the standard order in which they would appear in an English clause of the same type.

## 11-16 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

### 5. AUTOMATIC LITERATURE SEARCHING AND RETRIEVAL

**Library Problem** (Ref. 25). With the increasing rate of publications of all sorts, libraries are growing at an exponential rate. It is becoming increasingly difficult to find all and only the relevant information on a specific subject matter in a relatively short time. Automatic search and retrieval systems would be a boon not only to technical and scientific libraries but also for patent searching, legal searching (i.e., to find a precedent), medical searching, etc. The possible applications are manifold, and they are all variations on the same theme, viz., the use of machines to store, search for, and retrieve information.

**Need for an Indexing System.** The central linguistic problem in automatic literature searching is that of identifying similar meanings in information usually formulated in ordinary language with all its vagueness and ambiguity. At the library, both the man who requests the material and the author who produces it have formulated their meanings in the same ordinary language. The problem is to determine automatically whether the subject of the request is the same as, or close to, that contained in the various books, journals, etc., of a library. Because, at least for some time to come, no machine can actually read a document and decide whether or not its subject matter relates to some given requested subject, it is necessary to use some intermediate identifying tags, viz., an indexing system.

An *index* to a document is a code which provides a tag by means of which one can identify the subject content of the document. The indexing system of a library is the *common language* which the library uses to match the language of its documents with the language of the users. Then the only remaining task is to compare the indexes associated with the documents with those associated with requests and to retrieve those documents for which there is a match.

**Nature of an Index.** What is the best method of indexing? One approach is to divide all information (knowledge) into subject classes and then continually subdivide these classes until one arrives at very specific subject classes; then assign codes (addresses) to the subject classes and search for specific information by searching within these groups. This is classification indexing as represented by the Dewey decimal system, Library of Congress system, universal decimal system, etc.

It is extremely difficult to divide all knowledge into exclusive and exhaustive classes, and therefore an alternate indexing system known as coordinate indexing has developed. *Coordinate indexing* assumes that any subject matter can be identified as a point in an  $n$ -dimensional

meaning space where the coordinates of the point in question are one or many freely chosen terms (coordinates). For example, with the method of coordinate indexing, one might index Shannon's book on communication theory with the following set of terms (meaning coordinates): information theory, coding, error control, noise, entropy, Shannon, etc.

Both classification and coordinate indexing have serious limitations as to their capability for adequately representing the subject content of a document.

**Mechanized Systems.** Once a set of indexes has been assigned to each document it can be encoded into digital form, put on a suitable machine medium, and scanned automatically. For example, a punched card could contain indexes along with an accession number for the document. A duplicate card could be prepared for each different index of a document, and these could be sorted and stored. Then given any request for documents on some specific subject, one could drop the proper pile of punched cards into a sorter and search for all those cards which satisfy the request. Those cards which are thus retrieved could be fed into a tabulator which, in turn, would list the resulting accession numbers. Then the actual process of retrieving the documents would be a hand process.

One could put the same information on a continuous medium such as magnetic tape (in which case it might not be necessary to prepare a duplicate for each different index of a document) and scan the information at high speed with a high-speed digital computer (Ref. 26). The computer would be programmed to print out all those accession numbers which corresponded to a match of their indexes and the given request.

A further step in this type of mechanization would consist in using photographic film, instead of magnetic tape, to allow the document to be photographed beside its indexes and thereby save much space and provide a copy of the document immediately. In this way the machine could scan the film, match indexes with request, photocopy, and thereby supply a hard copy of the desired documents as a result of a search. Such a system, called the Rapid Selector, has been developed (see Ref. 25).

A recent variation on the Rapid Selector theme is the Minicard system developed by Eastman Kodak Company (Ref. 27). In the Minicard system the documents are reduced by a factor of 60 to 1 and copied on 16-mm film. The film is cut into units (instead of a continuous strip) and sorted into cells according to the digital information associated with each document image.

**Irrelevant and Incomplete Retrieval.** In each of the cases considered above, a machine searches and retrieves (either a copy of the document, an abstract, or an accession number) by matching indexes or some

## 11-18 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

logical combinations of indexes with each request. Thus the actual matching procedure is a go or no-go affair. A set of indexes either satisfies a request or it does not. There is no middle ground. On the other hand, the correspondence between the information content of a document and its set of indexes is not exact, for it is extremely difficult to specify precisely the subject content of a document by means of one or a set of indexes. Furthermore, the correspondence between the user's request, as formulated in terms of one or many indexes, and his real need (intention) is not exact. Thus there is "semantic noise" in both the index and in the user's request. The machine, however, is used to match the indexes and the request exactly. As a consequence of this the result of a search provides documents which are irrelevant to the real need and, even worse, some really relevant documents are not retrieved.

If one "broadens" his request so as to reduce the probability of missing a document, he increases the probability of obtaining irrelevant material and, conversely, if he "narrows" his request in order to avoid irrelevant material, he increases the probability of missing relevant information.

**Other Techniques.** One way of reducing the noise which results from inadequate indexing is to put a human user in the retrieval loop to function as a filter. This might be accomplished as follows. A request is programmed on a machine and it proceeds to scan its storage to find all documents whose indexes satisfy the request in question. At the same time, the machine would search a stored cross-reference system and print out indexes closely associated with the indexes in the original request. Then the machine would display to the user, abstracts of those documents which resulted from the original retrieval. On the basis of both the resulting abstracts which he could scan quickly on a convenient display and the associated indexes that the machine would supply, the user could modify his initial request and make a second "pass" on the machine. Hopefully, the result of this second retrieval would converge much more closely with his real information needs.

An automatic searching and retrieval system which employed a much improved indexing language might eliminate the need for a human in the loop. This indexing language would consist of a standard vocabulary and linguistic rules describing which words should be used in which combinations to obtain specific meanings. Thus the index to a document would consist not merely of sets of words, but, rather, sets of sentences formulated in a standard language. In principle, a system that included a print reader automatically could read documents, abstract them, and translate them to the standard language.

Given an improved indexing language one would hope to generate computer programs which, in effect, would mechanize the notion of rele-

vance so that a machine would not only look for an exact match of indexes, but would also retrieve items which were not explicitly requested but which are relevant to the user's request.

## REFERENCES

1. G. Walter, *The Living Brain*, Duckworth, London, 1953.
2. W. R. Ashby, *Design for a Brain*, Wiley, New York, 1951.
3. E. W. Beth, Hundred years of symbolic logic, *Dialectica*, **1**, 311-346 (1947).
4. W. Mays, C. E. M. Hansel, and D. P. Henry, Note on the exhibition of logical machines at the joint session, July 1950, *Mind*, **60**, 262 (1951).
5. E. C. Berkeley, *Giant Brains*, Wiley, New York, 1949, p. 144.
6. D. M. McCallum and J. B. Smith, Mechanical reasoning, *Electronic Eng.*, **23** (April 1951).
7. A. W. Burks, D. W. Warren, and J. B. Wright, An analysis of a logical machine using parenthesis-free notation, *Mathematical Tables and Other Aids to Computation*, **VIII** (April 1954).
8. W. Miehle, Burroughs truth function evaluation, *J. Assoc. Comp. Mach.*, **4** [2], 189-192 (1957).
9. C. E. Shannon, A symbolic analysis of relay and switching circuits, *Trans. Am. Inst. Elec. Eng.*, **57**, 713-723 (1937).
10. A. Newell, J. C. Shaw, and H. A. Simon, Empirical explorations of the logic theory machine: A case study in heuristics, The Rand Corp., 15 March 1957. Also *Proc. Western Joint Computer Conf.*, 218-229, Feb. 1957, Los Angeles, Calif.
11. A. N. Whitehead and B. Russell, *Principia Mathematica*, Cambridge University Press, Cambridge, England, Vol. I, 1910.
12. B. V. Bowden, *Faster than Thought*. Pitman, London, 1953, Chap. 15, Machines for the Solution of Logical Problems; Chap. 16, Special-Purpose Automatic Computers; Chap. 25, Digital Computers Applied to Games.
13. D. W. Hagelbarger, SEER, a sequence extrapolating robot, *I.R.E. Trans. Electronic Computers*, **EC-5**, 1-6, March 1956.
14. A. Newell, The chess machine: An example of dealing with a complex task by adaptation, *Proc. Western Joint Computer Conf.*, 101-107, March 1955, Los Angeles, Calif.
15. C. E. Shannon, Programming a computer for playing chess, *Phil. Mag.*, **41**, 256-275 (1950).
16. C. S. Strachey, Logical or non-mathematical programmer, *Proc. Assoc. Comp. Mach.*, 46-49, Toronto, 1952.
17. J. Kister, P. Stein, S. Ulam, W. Walden, and M. Wells, Experiments in chess, *J. Assoc. Comp. Mach.*, **4** [2], 174-177 (1957).
18. Y. Bar-Hillel, Can translation be mechanized? *Am. Scientist*, **42** [2] (1954).
19. W. N. Locke and A. D. Booth, *Machine Translation of Languages*, Technology Press of M.I.T., Cambridge, Mass., 1955.
20. "Design Study for an Integrated USAF Intelligence Data Handling System Appendix A, Machine Translation of Languages," Submitted by The Ramo-Wooldridge Corporation, Los Angeles, Calif. to Rome Air Development Center, Griffiss Air Force Base, Rome, N. Y.

## 11-20 USE OF DIGITAL COMPUTERS AND DATA PROCESSORS

21. K. E. Harper, Translating Russian by machine, *J. Commun.*, **5** [2], 41-46 (1955).
22. V. A. Oswald, Jr., and S. L. Fletcher, Jr., Proposals for the mechanical resolution of German syntax patterns, *Modern Language Forum*, **36** [3-4], 1-24 (1951).
23. M. E. Maron, A decision machine for the sentential calculus, *IBM Engr. Lab.*, San Jose, Calif., Tech. Rept. 205.002.018, March 6, 1953.
24. M. E. Maron, Some remarks concerning automatic programming, *IBM Engr. Lab.*, San Jose, Calif., Tech. Memo. 201.071.002, July 27, 1954.
25. L. Ridenour *et al.*, *Bibliography in An Age of Science*, University of Illinois Press, Urbana, Ill., 1951.
26. R. H. Bracken and H. E. Tillitt, Information searching with the 701 calculator, *J. Assoc. Comp. Mach.*, **4** [2], 131-136 (1957).
27. J. W. Kuipers, A. W. Tyler, and W. L. Myers, A Minicard system for documentary information, *Am. Documentation*, **8** [4], 246-268 (1957).

### Other References

- F. P. Brooks, Jr., A. L. Hopkins, Jr., P. G. Neumann, and W. V. Wright, An experiment in musical composition, *I.R.E. Trans. Electronic Computers*, **EC-6**, 175-181, Sept. 1957.
- C. Cherry, *On Human Communication*, The Technology Press of M.I.T., Wiley, N. Y., 1957.
- M. E. Maron, Logic, discovery and the foundations of computing machinery, *I.R.E. Trans. Electronic Computers*, **EC-3**, 2-7, June 1954.
- A. Oettinger, Programming a digital computer to learn, *Phil. Mag.*, **43**, 1243-62 (1952).
- C. E. Shannon, Presentation of a maze-solving machine, *Trans. Eighth Cybernetics Conf.*, Josiah Macy, Jr. Foundation, New York, 1952.
- C. E. Shannon and E. F. Moore, Machine aid for switching circuit design, *Proc. I.R.E.*, **41** [10], 1348-51 (1953).
- C. E. Shannon, Computers and automata, *Proc. I.R.E.*, **41** [10], 1235-41 (1953).
- N. Wiener, *Cybernetics*, Wiley, New York, 1948.

# DESIGN OF DIGITAL COMPUTERS

## D. DESIGN OF DIGITAL COMPUTERS

E. E. Bolles and E. M. Grabbe, Editors

12. Digital Computer Fundamentals, *by W. H. Ware*
13. Techniques for Reliability, *by W. H. Ware*
14. Components and Basic Circuits, *by N. H. Taylor*
15. Magnetic Core Circuits, *by I. L. Auerbach*
16. Transistor Circuits, *by I. L. Auerbach*
17. Logical Design, *by L. D. Amdahl*
18. Arithmetic and Control Elements, *by H. L. Engel*
19. Storage, *by D. R. Brown and J. I. Raffel*
20. Input-Output Equipment, *by J. K. Brigden*



## Digital Computer Fundamentals

*Willis H. Ware*

1. Digital Computers and Control Systems	12-01
2. Digital Computer Fundamentals	12-02
3. Machine Construction	12-07
4. Number Systems and Number Codes	12-12
5. Machine Number Systems	12-18
6. Computer Design Characteristics	12-25
References	12-30

### I. DIGITAL COMPUTERS AND CONTROL SYSTEMS

The digital computer is a device which operates on information, numerical or otherwise, represented in digital form, and it may be regarded as an extremely versatile, general purpose information or data handling device. The digital computer may be applied to many problems in the automation and control field as well as to scientific computation and general data processing. However, the choice in a particular application between the digital computer or analog computer must always be resolved on the basis of suitability, economics, reliability, and similar factors. Generally, the digital computer will represent more equipment than the analog, but in return will provide more precise results, may achieve a solution in shorter time, may be a more versatile and powerful computer, will be more readily adaptable to a wider range of problems, and may, because of its speed, be able to handle sequentially in time many problems in a sampled data fashion.

## 2. DIGITAL COMPUTER FUNDAMENTALS (Refs. 1, 2)

**Block Diagram.** A typical general purpose digital computing system will include the following sections: an input unit, an output unit, an arithmetic and logical unit, one or more storage units, a control unit. Figure 1 illustrates a typical set of interconnections between these sections.

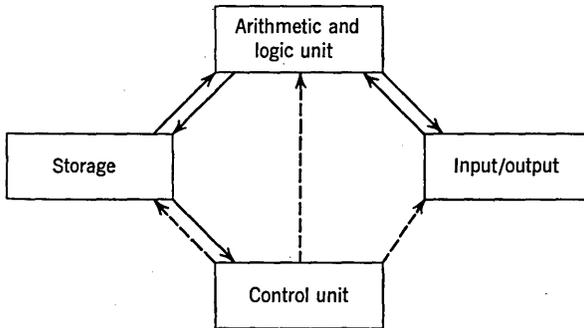


FIG. 1. Block diagram of a digital computing system.

**Input and Output Units.** These serve as communication channels between the computing system and the external world, which in an automation or control application may represent people or other machines, of a computing nature or otherwise. The input-output section may be thought of as a converter, which speaks the language of digital computing machines on the one hand, and speaks the language of the external world on the other hand.

**Arithmetic and Logical Unit.** This provides the facility for performing (1) any required arithmetic operations, e.g., add, subtract, divide, multiply, sometimes square root; and (2) certain logical operations, e.g., transfer of data, other manipulations of data which are nonarithmetic in nature.

**The Storage Unit.** This serves as a repository for (1) initial information, (2) intermediate results during a computation, and (3) completed results before they are returned via the output unit to the external world. The storage may be thought of as an array of pigeonholes, each of which is identified to the machine system by an *address* (Ref. 3). The address of a storage location permits repeated reference to the same pigeonhole to make use of any information which may be stored there, or to put new information into that location. The address (a number) signifies nothing about the contents of a storage location, but merely identifies the location.

**Control Section.** This is the master mind of the entire system and serves to interpret the statement of the problem to all other parts of the

system. Machine systems at their conception are endowed with the ability to understand and execute a certain set of *instructions*. Typical instructions are: multiply, add, bring in information through the input, put out information through the output, transfer information from arithmetic unit to storage, etc.

**Statement of the Problem to the Machine System.** This is a list of instructions used in such an assortment and arranged in such a sequence as to cause the computational process to proceed as planned by the user. The plan of solution for a problem is called the *program* and its graphical representation, a *flow diagram*. The statement of this program as a list of instructions for the machine to execute is called the *routine*. An instruction specifies to the computer system what is to be done and also specifies the location of the data upon which the operation is to be performed. The *what* of an instruction is called the *operation part* and the location of the information to be operated upon is specified by one or more *addresses*. A machine whose instructions contain one operation part plus one address is called a *single-address machine*. Instructions which contain more than one address are referred to as *two-address*, *three-address*, etc. Figure 2 illustrates the format of an instruction.

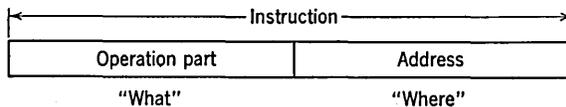


FIG. 2. Instruction format.

An instruction must be specified to a machine system in a language understandable to the system. Since a machine has been organized to operate on numeric data, it follows that instructions will also be in numerical form. The normal unit of information with which a machine system deals is called a *word*; a word may then represent either a piece of data or it may represent one or more instructions, depending upon the length of an instruction relative to the length of a word. Some machines have one instruction per word whereas others have two instructions per word.

### Method of Operation

**Information Input.** Two distinct kinds of information must be given to a digital computing system: the *routine*, or list of instructions which describes the problem, and the data with which the machine must deal in solving this problem. In some machines the data and the routine are placed in the same storage, whereas in other machines the data and the

routine are in separate storage devices. In either case the sequence of operations which a system goes through is, in broad outline, the same.

**Execution of Instructions.** A machine normally executes the list of instructions sequentially; hence, given the routine and the data, the control section when started generally proceeds through the following steps: (1) obtains the first instruction in the routine from the storage; (2) inspects the operation part of the instruction and interprets it to determine what function is to be performed; (3) routes one or more addresses to the storage device to transfer data to/from the indicated location(s) from/to an appropriate source/destination; (4) causes the appropriate arithmetic, logical, input, output or other system operation to be executed; (5) remains quiescent until all parts of the system report back that their individual functions have been completed; (6) again refers to storage for the next instruction in the routine. This basic sequence is repeated for each instruction in the routine, although deviations occur under some circumstances. The control routes appropriate signals to all other parts of the machine so that the ensemble acts together as a unit to carry out the instruction indicated.

If a machine is a single-address machine, each instruction can procure only one piece of data from this single address. A series of three single-address instructions is therefore necessary to carry out any one of the four fundamental arithmetic processes, since each process must obtain two factors and store the result. In a three-address machine, the three addresses may be used to indicate the locations of the two factors entering into the arithmetic operation and the third address may be used to indicate the location to which the result is to be returned.

**Jump Instructions.** Although the control generally proceeds sequentially down the ordered list of instructions, under some circumstances it is necessary to depart from the ordered sequence and go to an entirely different part of the routine. If this departure is to be executed regardless of conditions in the machine at the time of this instruction, such an instruction is called an *unconditional transfer* or *jump*. However, machine systems generally have the ability to detect simple conditions, e.g., to detect whether a number is zero or not, to detect whether a number is positive or negative. A *conditional transfer* or *branch* instruction calls for a jump to another part of a routine only if a specified condition is satisfied; if unsatisfied, the branch instruction is ignored and the control proceeds sequentially. This effectively permits the machine to make simple yes-no decisions. The address part of the jump or branch instruction (if satisfied) specifies the location of the next instruction.

**Computation Loops.** Generally, a program for a problem may be regarded as loops of computation (perhaps loops within loops within loops, etc.) connected by segments of *straight line* or *linear* routine. The jump and branch instructions are the means by which the machine circles each loop the proper number of times and then escapes from the loop. For instance, the computation of the sine from its infinite series expansion would contain a loop designed to compute the  $j^{\text{th}}$  term of this series. After the  $j^{\text{th}}$  term is completed and the new partial sum of the series is formed, an auxiliary loop adjusts the instructions in the first loop so that when the computation returns to the first loop, it computes the  $(j + 1)^{\text{st}}$  term. Meanwhile, another loop keeps track of the number of terms computed and decides when the job is completed.

Machine decisions are typically used for (1) recognizing the status of a problem, (2) controlling the path of computation, frequently on the basis of previously computed results, (3) answering arbitrarily complicated questions, such as identification of characters or symbols.

#### **Typical Machine Instructions.**

1. *Arithmetic*, consisting of add, subtract, divide, multiply, occasionally square root.

2. *Input-output*, which govern the input-output equipment both electrically and mechanically, and cause a flow of information via these converters into or out of the machine system.

3. *Logical*, which include the jump and branch operation, the transfer of data from one location to a second, the *extract* or *mask* instructions which form a new word by combining selected segments of other words, shift instructions which shift the position of data columnwise and thus may scale the data. In a shift instruction the address part is not used to specify a location in storage, but is used to specify the number of places that the data is to be shifted.

4. *Console*, which permit manual control of the computing system and operator intervention into the automatic operation of the system.

### **Computer Equipment**

**Input-Output Section** (see Chap. 20). If the machine application is that of scientific computing or data processing, the input-output will consist of devices for communicating with people; e.g., punched card reader, card punch, photoelectric paper tape reader, typewriter, printers, and plotters. If the application is one of control or automation, the input-output may also contain devices for communicating with other machines; e.g., analog-digital converters, digital-analog converters, magnetic tape, and digital servos. There may also be associated with this section certain

error-detecting or verification equipment to monitor the correctness of translation of information from one "language" to another.

**The Arithmetic and Logical Section** (see Chap. 18). This typically contains (1) an *adder* to effect the arithmetic process of addition and (2) three *registers* to store the two factors for and the result of an arithmetic or logical process. A *register* is a storage device for one word and frequently also possesses the ability to step (or shift) its contents columnwise in either direction. The shift feature is valuable for scaling data and for some logical manipulations of information. The adder and one register are frequently associated to form the *accumulator*. There is no standard terminology for labeling the registers of the arithmetic unit. In some machines they are referred to as the *accumulator*, the *M-Q* (this register receives the multiplier in multiplication, the quotient in division) and the *number register* (this register receives the multiplicand in multiplication and the divisor in division); in others they are merely designated by letters or numbers. Some machines also contain additional arithmetic or logical equipment for special purposes, e.g., more than one accumulator, additional registers for the storage of special information, and special equipment to permit extremely high-speed multiplication.

**The Storage Units** (see Chap. 19). This contains some device which possesses a property capable of storing information and also whatever electronics may be necessary to control and manipulate appropriately this physical means of storing and retrieving information. Typically, the associated electronics for the storage will be amplifiers, pulse control equipment, pulse sources, etc. For economic reasons the storage will normally be organized in a hierarchy in which a relatively small but fast principal storage is supported by a larger but slower secondary storage which may in turn be supported by a tertiary larger and slower storage, to as many levels as required.

**The Control Section** (see Chap. 18). This is the most heterogeneous part of the machine. It contains a large amount of equipment to make decisions and to remember that certain events in machine operation have occurred. The decision-making elements are called *gates* (Chap. 14), which in turn are referred to as *and gates*, or *or gates*, according as the logical process which the gate accomplishes is the *and* operation, or the *or* operation of formal logic. The element which remembers that some particular event has occurred stores a single *yes-no* piece of information and is called a *flip-flop* or a *toggle* (see Chap. 14). An *instruction counter* keeps a running tally of the address in which the instruction next to be executed is to be found. If the machine system is executing a linear routine placed in consecutive addresses, the counter proceeds

through a consecutive sequence of numbers; if a jump instruction occurs, the counter will be forced to commence counting at some new origin. There must also be a counter to tally the several steps in a multiplication or division process or the number of shifts required in a shift instruction.

**The Power Supplies.** For large digital electronic computers these frequently become heavy-current units. It is frequently necessary that close regulation be maintained, both under long-term drifts and under severe transient load conditions. Although rotating machines have been used, magnetically regulated, gas-tube, or solid state power supplies are most common.

**The Console.** This portion of a digital computing machine system serves two functions, which are sometimes physically separated: (1) the *operating console* presents to the operator those manual controls and displays of information which an operator needs to control the machine adequately and observe its status; (2) the *maintenance console* presents a complete picture of conditions within the machine and provides the maintenance engineer with the ability to effect changes in any of the information stored within the machine system.

Optional parts of the console system are: (1) a rather extensive supervisory protective system which monitors all power supply voltages to verify that these voltages are not too far from nominal values, (2) an elaborate system of fusing which often takes such a form that a blown fuse automatically turns off machine power and indicates the location of trouble, (3) *marginal checking* facilities by means of which the operating environment of the system (e.g., supply voltages, temperature, pulse rates) can be systematically varied to detect incipient failures and to investigate the system's tolerance to drifting components.

### 3. MACHINE CONSTRUCTION

A functional description of a computing system can be realized in terms of a number of distinctly different techniques, e.g., mechanical, electromechanical, photoelectric, electronic. The nature of the engineering technique in which the machine is realized will bear heavily on the reliability of the completed system and on the speed at which the completed system operates.

**Mechanical Computers.** Desk calculators are not included because they do not have the facility for being automatically sequenced through a long series of arithmetic and logical operations.

The Mark I machine, constructed and designed jointly by the International Business Machines (IBM) Corporation and Harvard University, consisted principally of mechanical arithmetic devices and it may

be regarded as a mechanical computer. It was automatically sequenced from a routine contained on paper tapes; motive power was supplied from a central rotating drive shaft.

**Punched Card Machines.** Several manufacturers offer lines of equipment in which the unit record of information is a paper card containing punched holes. In the equipment offered by IBM, the card is  $7\frac{3}{8}$  in. by  $3\frac{1}{4}$  in. by 0.0065 in. thick, and contains 80 *columns* by 12 *rows* of hole positions. The card code (Hollerith code) in which information is represented depends on the position and number of punches. Although  $2^{12}$  or 4096 distinct characters in principle could be represented in each column, in practice, a maximum of three punches per column is permitted and only a total of 47 characters per column is available. Numeric digits are indicated by a punched hole in one of ten possible rows, whereas alphabetic and special characters are indicated by two or three punches, one or two occurring in the ten *numeric* positions and one in an additional two rows of *zone* positions.

In the Remington-Rand equipment line the unit record is again a paper card, of the same dimensions, but which is effectively 90 columns by 6 rows although this is physically arranged on the card as 12 rows by 45 columns. The code in which information is represented is more sophisticated and utilizes combinations of holes in 6 rows to represent all the numeric and alphabetic symbols. The IBM and Remington-Rand card codes are tabulated in Fig. 3.

For each of these kinds of cards there is available a line of machinery for originating, reading, and manipulating the cards: (1) key punches which produce cards from keyboard information; (2) card readers which translate punched information into electrical or mechanical signals; (3) card punches which are actuated by electrical or mechanical signals to produce a new card; (4) sorters which inspect a specified column of the card and, on the basis of the information contained therein, physically move the card to one of several collection pockets; (5) devices for producing a new card identical to an original card; (6) certain other kinds of special equipment. Some equipment is available for manipulating cards which are physically shorter than full width. In general, card speeds vary from 100 to 1000 cards per minute depending on the particular equipment.

The IBM Card Program Calculator (CPC) and the Remington-Rand 409-2 machine are typical of the punched card computers. In each, the instruction specifying the operation to be performed and the data to be operated upon is contained in punched cards. Each machine provides a certain amount of storage and contains a plugboard by means of which a large amount of internal electromechanical equipment may be organ-

ized to effect a wide variety of logical and arithmetic operations. Such punched card computers are relatively slow devices, being paced by the card-handling equipment which normally operates in the vicinity of 100 to 150 cards per minute.

Frequently, punched card equipment, particularly readers and punches, finds application in the input-output section of a high-speed electronic digital computer.

**Relay Computers.** Some general purpose computing machines have been built that make use of relay techniques. A series of relay computers known as the Bell relay computers were designed and constructed by the Bell Telephone Laboratories. Harvard University designed and constructed the Mark II computer with relay techniques for the arithmetic and logical operations of the machine. Such machines are limited in speed by the relay components.

**Electronic Computers.** After 1946 a large number of computing machine systems appeared which extensively use various electronic circuit techniques. These machines are generally the fastest and most powerful of the digital computers. The first was the Eniac machine, designed and constructed at the University of Pennsylvania. Subsequently, a family of machines patterned after the developments at the Institute for Advanced Study of Princeton, New Jersey, appeared: the Illiac, the Ordvac, the Avidac, the Johnniac, the Maniac, and the Oracle. Commercially, IBM introduced its 700 series of machines; Sperry-Rand introduced its Univac and Univac-Scientific series of machines. A number of small, medium-priced machines were also introduced on the market, among them the series by National Cash Register, the Datatron machine by the Electro Data division of the Burroughs Corporation, the machines by Librascope, Bendix Computer Division, and others. A number of companies have also introduced special purpose machines intended to accommodate a restricted class of problems.

### **Three Categories of Electronic Machines.**

1. The *arithmetic machine*, whose principal function is to accomplish arithmetic processes, but which in addition must also perform many logical types of operations. This class of machines includes most existing business and scientific computers and is the type previously discussed here.

2. The *logical machine* whose function is to handle problems that are stated in the language of formal logic, e.g., the Kalin-Burkhart or Truth computer. (See Chap. 11.)

3. The *special purpose machines*, which include the digital differential analyzer, the inventory control machines, the check sorting machines, and the ticket reservation machines.



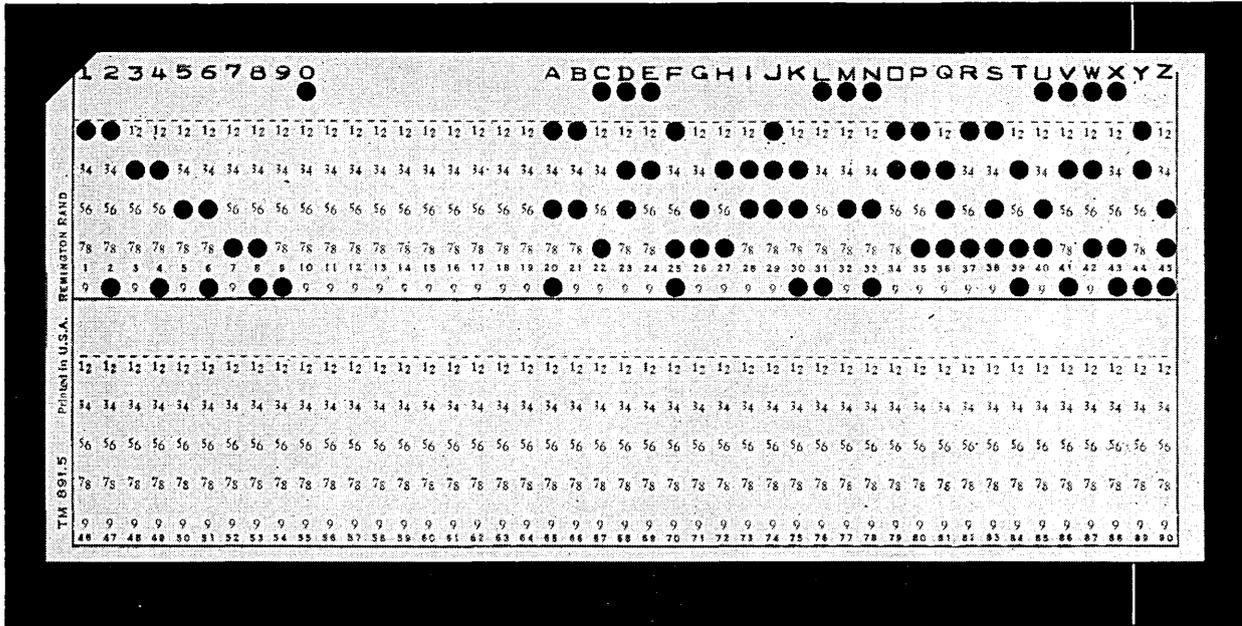


Fig. 3b. Remington-Rand card and code.

**The Digital Differential Analyzer** (see Chap. 28). This differs from the general purpose arithmetic machine in that it is very highly organized internally so that efficient use is made of equipment. This machine accomplishes the mathematical operation of integration by making use of a numerical approximation to it. It is therefore competitive with analog types of differential analyzers and accommodates the same class of problems. It provides for more precision in the results, and, generally, for a given amount of equipment it provides a larger number of integrators than the analog system. As many as 100 integrators are typical.

The digital differential analyzer also differs from the general purpose arithmetic machine in that it represents information in a different form. Frequently, information in the digital differential analyzer is represented in incremental form as a train of pulses, whereas in arithmetic machines digital information is usually represented by the relative position of an electric signal.

#### 4. NUMBER SYSTEMS AND NUMBER CODES (Refs. 2 and 4)

**Number Representation.** Care must be taken to distinguish between two different meanings of number. Formally, a *number* is an abstract mathematical entity which is a generalization of a concept used to indicate quantity, direction, etc. In this sense a number is independent of the manner of its representation. Commonly number is taken to mean a representation in a specific number system of a quantity as above defined. In the following discussion number is used to refer solely to the abstract mathematical entity, whereas a representation of a number in a particular number system will be called a *numerical expression*.

A number is independent of its manner of representation, and therefore, since counting may be accomplished in any number system, rules must exist for transforming a numerical expression in one system to the equivalent numerical expression in any other system.

Modern notation for a numerical expression makes use of the concept of *positional notation*. In a given number system there is a specified number of permitted symbols which may be used to indicate entries in the system, e.g., the decimal number system contains 10 digits. The number of symbols permitted in a system is called the *base* or *radix* of the system. A number  $N$  may be expressed in the number system of radix  $R$  in the following form:

$$N = \cdots + a_4R^4 + a_3R^3 + a_2R^2 + a_1R^1 + a_0R^0 + a_{-1}R^{-1} + a_{-2}R^{-2} + a_{-3}R^{-3} + \cdots,$$

—————Integral part—————
—————Fractional part—————

where the  $a_i < R$  and it is agreed that this expression is to be written as

$$\cdots a_4 a_3 a_2 a_1 a_0 . a_{-1} a_{-2} a_{-3} \cdots .$$

The magnitude represented by a particular value of  $a_i$  depends not only on  $a_i$  itself but also upon its position with respect to the *point* which is the mark separating the integral and fractional parts.

EXAMPLE.

$$\begin{aligned} 555 &= 500 + 50 + 5 \\ &= 5 \times 10^2 + 5 \times 10^1 + 5 \times 10^0. \end{aligned}$$

Here  $a_2 = a_1 = a_0 = 5$ , but the same symbol represents magnitudes of 500, 50, and 5 respectively. Since the digits traditionally used in the decimal number system (0, 1, 2, 3,  $\cdots$ , 9) are also commonly used to express the digits in any number system of radix less than 10, frequently a subscript (written in decimal notation) at the right end of the numerical expression is used to indicate the base in use, e.g.,  $1793_{10}$ ,  $1765_8$ ,  $1110_2$ .

A comparison of the first  $20_{10}$  entries for the binary, octal, decimal, and hexadecimal systems is shown in Table 1. In the hexadecimal system, six additional symbols are necessary, and letters have been used.

TABLE 1. A COMPARISON OF NUMBER SYSTEMS

Decimal	Binary	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13
20	10100	24	14

Note that the symbol  $10$  (not to be pronounced *ten*) specifies a magnitude representing the radix of the system. The correctness of this table is easily verified by writing any entry in positional notation. In base  $10_{10}$ :

$$\begin{aligned} 20_{10} &= 2 \times 10^1 + 0 \times 10^0 \Big|_{10} = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 \\ &\qquad\qquad\qquad + 0 \times 2^1 + 0 \times 2^0 \Big|_{10} \rightarrow 10100_2 \\ &= 2 \times 8^1 + 4 \times 8^0 \Big|_{10} \rightarrow 24_8 \\ &= 1 \times 16^1 + 4 \times 16^0 \Big|_{10} \rightarrow 14_{16}. \end{aligned}$$

Care must be taken because of the equality sign to see that a uniform number system is used on both sides of an equation; e.g.,  $8_{10}$  could not appear in an octal expression nor  $2_{10}$  in a binary expression.

### Conversion from One Base to Another (Ref. 5)

#### Whole Numbers.

*Base R to Decimal.* To convert a number in base  $R$  to its equivalent in decimal, write the number in its base  $R$  form but with decimal notation. Expand the resulting decimal expression.

EXAMPLES.

$$\begin{aligned} 23_8 &= 2 \times (10)^1 + 3 \times (10)^0 \Big|_8 \\ &\rightarrow 2 \times (8)^1 + 3 \times (8)^0 \Big|_{10} \\ &= 16 \quad + \quad 3 \Big|_{10} \\ &= 19_{10}. \end{aligned}$$

$$\begin{aligned} 14_{16} &= 1 \times (10)^1 + 4 \times (10)^0 \Big|_{16} \\ &\rightarrow 1 \times (16)^1 + 4 \times (16)^0 \Big|_{10} \\ &= 16 \quad + \quad 4 \quad \Big|_{10} \\ &= 20_{10}. \end{aligned}$$

*Decimal to Base R.* To convert a decimal numerical expression to its equivalent in base  $R$ : (1) divide the given numerical expression by the new base, performing the arithmetic in the decimal system; (2) the remainder is  $a_0$  in the new expression; (3) with only the integral part from the previous operation, again divide by the new base; (4) the remainder is  $a_1$  in the new expression; (5) continue this process until the division produces only a remainder; (6) this will be the final  $a_i$  in the new expression.

EXAMPLE.

$$55_{10} = \dots a_i \times 2^i + \dots + a_3 2^3 + a_2 2^2 + a_1 2^1 + a_0 2^0 |_{10}$$

$$\left. \begin{array}{l} \frac{55}{2} = 27 + \frac{1}{2} : a_0 = 1 \\ \frac{27}{2} = 13 + \frac{1}{2} : a_1 = 1 \\ \frac{13}{2} = 6 + \frac{1}{2} : a_2 = 1 \\ \frac{6}{2} = 3 + \frac{0}{2} : a_3 = 0 \\ \frac{3}{2} = 1 + \frac{1}{2} : a_4 = 1 \\ \frac{1}{2} = 0 + \frac{1}{2} : a_5 = 1 \end{array} \right|_{10}$$

or

$$\begin{aligned} 55_{10} &\rightarrow a_5 a_4 a_3 a_2 a_1 a_0 \\ &= 1 1 0 1 1 1_2 \\ &\rightarrow 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 |_{10} \\ &= 32 + 16 + 0 + 4 + 2 + 1 |_{10} \\ &= 55_{10}. \end{aligned}$$

*Arithmetic Operations.* If the arithmetic operations have to be carried out in the new base, the same processes work but the expressions must be transliterated into the new base before performing arithmetic.

EXAMPLE. (Refer to Table 1 for equivalences.)

$$\begin{aligned} 75_{10} &= 7 \times 10^1 + 5 \times 10^0 |_{10} \\ &\rightarrow (0111) \times (1010)^1 + (0101) \times (1010)^0 |_2 \\ &= 1000110 + 0101 |_2. \\ &= 1001011 |_2. \end{aligned}$$

Conversely,

$$\begin{aligned} 1001011/1010 &= 0111 + 0101/1010 : a_0 = 0101_2 \rightarrow 5_{10} \\ 0111/1010 &= 0 + 0111/1010 : a_1 = 0111_2 \rightarrow 7_{10} \\ &\rightarrow 75_{10}. \end{aligned}$$

It should be noted that integers of finite length always convert to integers of finite length.

**Fractional Numbers.** The powers of the radix now appear in the denominator, i.e.,

$$\begin{aligned} 0.345_{10} &= \frac{3}{10} + \frac{4}{100} + \frac{5}{1000} \Big|_{10} \\ &= 3 \times 10^{-1} + 4 \times 10^{-2} + 5 \times 10^{-3} \Big|_{10}. \end{aligned}$$

*Decimal to Base R.* Accordingly, to convert from a decimal fraction to one in another base: (1) multiply the decimal fraction by the new base, performing the multiplication in the decimal system; (2) the integral part is the  $a_{-1}$  in the new expression; (3) with only the fractional part from the previous operation, again multiply by the new base; (4) the integral part is the  $a_{-2}$  in the new expression; (5) continue until the process terminates or until sufficient precision is achieved. Conversely, to convert a base  $R$  fractional into decimal, express the fractional in its base  $R$  form but use decimal notation and expand the resulting decimal expression.

EXAMPLES.

$$\begin{aligned} 0.1101_2 &\rightarrow 1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} \Big|_{10} \\ &= \frac{1}{2} + \frac{1}{4} + 0 + \frac{1}{16} \Big|_{10} \\ &= \frac{13}{16}_{10}; \end{aligned}$$

but  $0.6_{10} = a_{-1} \times 2^{-1} + a_{-2} \times 2^{-2} + a_{-3} \times 2^{-3} \dots \Big|_{10}$

$$0.6 \times 2 = 1.2 : a_{-1} = 1$$

$$0.2 \times 2 = 0.4 : a_{-2} = 0$$

$$0.4 \times 2 = 0.8 : a_{-3} = 0$$

$$0.8 \times 2 = 1.6 : a_{-4} = 1$$

$$0.6 \times 2 = 1.2 : a_{-5} = 1$$

etc.

$$0.6_{10} \rightarrow 0.10011 \dots_2.$$

*Rational fractions* will convert to repeating fractions where a finite fraction is considered a degenerate case of a repeating fraction. Irrational fractions will convert to infinite fractions.

EXAMPLES.

$$\frac{6}{10} = 0.6_{10} \rightarrow 0.1001 \ 1001 \ 1001 \dots_2$$

$$\frac{1}{3} = 0.333 \dots_{10} \rightarrow 0.1_3$$

$$\pi = 3.14159 \dots_{10} \rightarrow 3.110374 \dots_8$$

**Mixed Numbers.** The integral and fractional parts must be separated, converted individually, and reassembled.

**Bases Related as Powers or Roots.** In the special case where the old and new bases are related one as the power of the other, conversion is particularly easy. To convert from the larger base to the smaller transliterate digit by digit from the high base to the low base. Conversely, use the power which relates the two bases to determine the length of groups into which the digits of the lower-based expression is formed, and interpret these groups as digits in the higher base.

EXAMPLE. Base 2 and Base 8:  $2^3 = 8$ , and therefore the group length is 3:

$$001\ 110\ 111\ 010_2 \rightarrow 1674_8$$

and

$$103_8 \rightarrow 001\ 000\ 011_2.$$

### Reflected Number Systems (Ref. 6)

In some applications, e.g., analog-digital converters (see Chap. 20), it is desirable to avoid the characteristic of normal number systems that in counting more than one digit may change at a time, e.g., the transition from 999 to 1000. Reflected number systems avoid this difficulty by counting first upwards and then downwards in each digit position in turn. Table 2 demonstrates the method of constructing reflected systems. Of particular importance is the reflected binary system or Gray code. If the true binary number is represented as  $A_n A_{n-1} \cdots A_1$  and the re-

TABLE 2. A TABLE OF NORMAL AND REFLECTED NUMBER SYSTEMS

Decimal	Reflected Decimal	Normal Octal	Reflected Octal	Normal Binary	Reflected Binary
0	0	0	0	0	0
1	1	1	1	1	1
2	2	2	2	10	11
3	3	3	3	11	10
4	4	4	4	100	110
5	5	5	5	101	111
6	6	6	6	110	101
7	7	7	7	111	100
8	8	10	17	1000	1100
9	9	11	16	1001	1101
10	19	12	15	1010	1111
11	18	13	14	1011	1110
12	17	14	13	1100	1010
13	16	15	12	1101	1011
14	15	16	11	1110	1001
15	14	17	10	1111	1000
16	13	20	20	10000	11000

flected binary number as  $a_n a_{n-1} \cdots a_1$ , then one set of rules for conversion is:

1. To find  $a_k$ , add mod 2 (i.e., the carry is discarded), the digits  $A_k$  and  $A_{k+1}$ . Notice that  $a_n$  always equals  $A_n$ .

2. To find  $A_k$ , add  $a_k$  through  $a_n$ , divide by 2, and the remainder is  $A_k$ .

Other rules can be formulated which avoid the arithmetic operations, but which may require the new number to be formed sequentially starting from the most significant digit.

## 5. MACHINE NUMBER SYSTEMS

**Binary Coded Systems** (Ref. 2). Storage devices are not presently available which are wholly satisfactory for number base systems higher than 2. Computing machines are therefore commonly organized to do arithmetic in the binary number system. If necessary to design a machine which is internally decimal or higher base than 2, the high base information is encoded in terms of binary digits. Although only 3.32 binary digits are required to express one decimal digit, practically, this must be rounded to four, and the resulting inefficiency of approximately 20 per cent must be accepted.

For the decimal system, ten distinct symbols are necessary. A tetrad of binary digits has  $2 \times 2 \times 2 \times 2$  or sixteen possible configurations and may be used to encode the decimal digits. Some ten of the sixteen possible combinations are selected to represent the decimal digits, and the balance is discarded. Of the approximately  $10^{10}$  such codes (the permutations of 16 things taken 10 at a time) only a small number have found practical application.

Machines that are so organized that decimal information is internally represented in terms of binary tetrads are referred to as *binary coded decimal machines*. If decimal information as well as alphabetic information must be accommodated, such a machine is called *alphanumeric* or *alphameric* and represents information in sextuples of binary digits. One alphanumeric code would be to use the first 10 natural binary numbers for the decimal digits, the next 26 to represent the alphabet and as many more as necessary to represent punctuation and special characters. The unused of the  $2^6$  (= 64) sextuples are discarded.

**Factors Determining the Choice of a Code.** Such things are considered as efficiency of storage requirements, ability to detect errors, ability to correct errors, convenience in arithmetic operations such as complementing, no code groups without at least a single 1, the same number of 1's in each code group, the minimum number of 1's in the complete code, and convenience to human beings.

**Weighted Codes.** The ten combinations to be used may be so selected that if weights are assigned to each column of the binary tetrad, the decimal value of that tetrad may be found by adding up the weights of those columns in which a binary 1 appears. A simple example of a weighted code (see Table 3) is one in which the four columns of the

TABLE 3. WEIGHTED CODE

Decimal	8, 4, 2, 1 Code
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1

tetrad have weights 8, 4, 2, 1. This expresses the decimal digits as the first ten binary natural numbers. A list of 70 weighted 4-bit codes is given by Richards (Ref. 2). Other common weighted codes are the 5, 4, 2, 1; the 2, 4, 2, 1; the 5, 3, 1, 1. It is not necessary that the weight be a positive integer, e.g., 7, 4, -2, -1. The representation of the decimal digits in these codes is exhibited in Table 4. The sum of the positive weights must be 9 or more.

TABLE 4. EXAMPLES OF WEIGHTED BINARY CODES

Decimal	5, 4, 2, 1	2, 4, 2, 1	5, 3, 1, 1	7, 4, -2, -1
0	0000	0000	0000	0000
1	0001	0001	0001	0111
2	0010	0010	0011	0110
3	0011	0011	0100	0101
4	0100	0100	0101	0100
5	1000	1011	1000	1010
6	1001	1100	1001	1001
7	1010	1101	1011	1000
8	1011	1110	1100	1111
9	1100	1111	1101	1110

**Nonweighted Codes.** Sometimes it is desirable to use a 4-bit code to which weights cannot be assigned. One such code which has received fairly extensive application (see Self-Complementing Codes) is the *excess 3 code* shown in Table 5. In it the decimal digit  $D$  is represented as  $D + 3$  in the 8, 4, 2, 1 code. Thus the group 0000 which might

TABLE 5. EXCESS 3 CODE

Decimal	Excess 3
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

be represented by absence of all signal does not appear; thus absence of all signal is readily detected as an error.

**Self-Complementing Codes.** The arithmetic process of subtraction may be performed by making use of complement representation of negative numbers (see Negative Numbers). In a binary coded decimal machine it is then necessary to form the *decimal* complement (as opposed to *binary* complement) of the binary tetrad. Generally the decimal complement is not the binary complement of the tetrad, except that some codes have the property that if 1's and 0's are interchanged, i.e., form the binary 1's complement, then the 9's complement of the decimal digit is obtained. Such codes must have the property that the algebraic sum of the weights is 9. The 8, 4, 2, 1 code is not a self-complementing code, but the 2, 4, 2, 1 is. A detailed listing of this particular code and its complements are exhibited in Table 6. It should be noted that other than weighted codes may exhibit this self-complementing property as, for instance, the excess 3 code.

**Higher Than 4-Bit Codes.** Groups of binary digits longer than four may be used to represent decimal or other information under certain

TABLE 6. SELF-COMPLEMENTING PROPERTY OF THE 2, 4, 2, 1 CODE

Decimal Digit	Number 2, 4, 2, 1		Complement 2, 4, 2, 1	Decimal 9's Complement
0	0 0 0 0	Changing 1's and 0's →	1 1 1 1	9
1	0 0 0 1		1 1 1 0	8
2	0 0 1 0		1 1 0 1	7
3	0 0 1 1		1 1 0 0	6
4	0 1 0 0		1 0 1 1	5
5	1 0 1 1		0 1 0 0	4
6	1 1 0 0		0 0 1 1	3
7	1 1 0 1		0 0 1 0	2
8	1 1 1 0		0 0 0 1	1
9	1 1 1 1	0 0 0 0	0	

circumstances. The use of extra bits not used for character representation adds redundancy to the code. This redundancy may be utilized to provide error-detecting or error-correcting features (see Refs. 7 and 8). One such error-detecting code is generated by adding a fifth bit to any of the 4-bit codes, the value of the fifth bit being chosen in such a way that the total number of 1's present in the resulting 5-bit group is even (or odd). Thus a *parity check* of any code group will detect an odd number of errors. See Table 7. (See also Vol. I, Chap. 16.)

TABLE 7. THE 8, 4, 2, 1 CODE WITH AN ODD PARITY CHECK

Decimal	8, 4, 2, 1 Code	8, 4, 2, 1 Code and Parity Bit
0	0000	10000
1	0001	00001
2	0010	00010
3	0011	10011
4	0100	00100
5	0101	10101
6	0110	10110
7	0111	00111
8	1000	01000
9	1001	11001

A 7-bit weighted code is the *biquinary code*, shown in Table 8. Five of the bits are grouped together to form a representation of the decimal digits 0 through 4. The other pair of bits indicates the presence of

TABLE 8. BIQUINARY CODE

Decimal	5, 0, 4, 3, 2, 1, 0
0	0 1 0 0 0 0 1
1	0 1 0 0 0 1 0
2	0 1 0 0 1 0 0
3	0 1 0 1 0 0 0
4	0 1 1 0 0 0 0
5	1 0 0 0 0 0 1
6	1 0 0 0 0 1 0
7	1 0 0 0 1 0 0
8	1 0 0 1 0 0 0
9	1 0 1 0 0 0 0

decimal 5 or the presence of decimal 0. Each code group contains two, and only two binary 1's. This characteristic may be utilized to inspect code groups for errors.

**Arithmetic Processes in Binary Coded Machines.** Not all combinations of tetrads (or larger groups) are used, e.g., 10 of 16, and since

the arithmetic section will treat these groups as true binary numbers, special measures will need to be taken when performing arithmetic operations in a coded machine to prevent the appearance of these invalid groups. The corrections are peculiar to the code in question and can be deduced by a study of the code. The difficulty of performing arithmetic limits the usefulness of some weighted codes and most of the nonweighted codes.

In the 8, 4, 2, 1 code, the first 10 of the 16 code groups are used. The corrective procedure for addition is to add 6 (0110) to the preliminary sum if it exceeds 9.

EXAMPLES.

3	0011	6	0110
+5	0101	+7	0111
8	1000	(1)3	1101 > 1001
		↑	0110    Corrective 6
		Carry	(1)0011
			↑
			Carry

### Negative Numbers

Negative numbers may be represented in a digital computing system in two ways. The conventional way in which people deal with a negative number (representing it as a magnitude plus algebraic sign) is referred to as *sign and magnitude notation*. The alternative to this representation is the *complement representation* by means of which the arithmetic process of subtraction is replaced by the arithmetic process of addition.

**Complement Representation.** In subtracting two numbers  $N_1$  and  $N_2$  in base  $R$ , the process may be rearranged as follows:

$$N_1 - N_2 = N_1 - N_2 + R^n - R^n = N_1 + (R^n - N_2) - R^n,$$

where  $n$  has been chosen so that  $R^n$  exceeds the largest expected  $N$ . The expression  $(R^n - N_2)$  is the *complement of  $N_2$  with respect to  $R^n$*  or commonly, the *complement of  $N_2$* . (See Sign Digit below for representation of the sign of a complement.) The usefulness of this technique will be exhibited by the following examples.

EXAMPLES. Let

$$N_1 = 0.769,$$

$$N_2 = 0.345.$$

Then  $(10 - N_2) = 10.000 - 0.345 = 9.655$

and  $(10 - N_1) = 10.000 - 0.769 = 9.231$

Sign and Magnitude	As Carried Out	Actual Meaning
0.769	0.769	0.769
<u>-0.345</u>	<u>+9.655</u>	<u>+9.655 - 10</u>
0.424	(1)0.424	10.424 - 10

↑  
Carry is discarded

but:

0.345	0.345	0.345
<u>-0.769</u>	<u>+9.231</u>	<u>9.231 - 10</u>
-0.424	9.576	9.576 - 10

↑  
Indicates this is a complement.  
Its sign and magnitude representation is found by again complementing and affixing a negative sign.

10.000	
<u>-9.576</u>	
0.424	-0.424

The  $(-R^n)$  which is not carried along in the arithmetic manipulations is canceled by a carry from the most significant place of the addition process when required. The value of complementing is the replacement of subtraction by addition. The arithmetic section need thus have only adding properties. Implicit is the assumption that the complement can be formed without actually performing subtraction. Consider forming the complement of 69 with respect to 100.

$$\begin{aligned} (100 - 69) &= (99 - 69) + 1 \\ &= 30 + 1 = 31. \end{aligned}$$

Since each digit column of 99 contains the largest digit of the decimal system, i.e., 9, it can be guaranteed that no borrows can arise in forming  $(99 - 69)$  and hence, each column of this operation can be handled independently. This replaces subtraction with a columnwise logical operation. Thus, even the formation of the complement is reduced to a logical operation plus an addition. In this example,  $(99 - 69) = 30$  is called the 9's complement whereas 31 is the true or 10's complement.

In a binary system, the 1's *complement* is a process of swapping 1's and 0's. The complement of 0101 is 1010 as shown below:

$$\begin{aligned}(10000 - 0101)_2 &= (1111 - 0101) + 1|_2 \rightarrow (16 - 5)_{10} = (15 - 5) + 1|_{10} \\ &= 1010 + 1 = 1011|_2 & & = 10 + 1 = 11|_{10}.\end{aligned}$$

In this example 1010 is the 1's complement and 1011 the 2's complement.

In general, the complement with respect to some power of the base is referred to as the *true complement*, whereas the complement with respect to (some power of the base  $-1$ ) is referred to as the (*radix*  $-1$ )'s *complement*.

Although complementing simplifies addition and subtraction, it complicates multiplication and division since spurious terms will arise in the result. Certain corrective operations must be performed at the end of the process in order to eliminate them.  $(N_1)(-N_2)|_2$  would be done as:

$$(N_1)(2^r - N_2) = 2^r N_1 - N_1 N_2,$$

whereas the desired result is  $(2^r - N_1 N_2)$ .

**Sign and Magnitude Representation.** Here the multiplication and division proceed smoothly. Multiply or divide the magnitudes of the factors and form the sign of the result from an inspection of the signs of the factors. Difficulty with respect to addition and subtraction may arise. It is not known a priori whether the difference of two numbers will be positive or negative. If the minuend enters the arithmetic section first and is larger, the sign of the result will be positive and no difficulty occurs. If the subtrahend enters first and is larger, the result will not appear in its true sign and magnitude representation, but will appear in complement form. This behavior is illustrated by noting the action of the dials in the carriage of a desk calculator. Upon subtracting a larger number from a smaller one, all the dials rotate through zero to produce the complement of the sign-magnitude form of the result.

If a complement form appears in the arithmetic element in this way, it must be complemented to return it to the sign and magnitude notation for which the rest of the machine system is organized.

### Roundoff (Ref. 9)

Since multiplication of two numbers  $n$  places long may yield a product  $2n$  places long, a machine may be forced to truncate a number. In so doing, it is hoped that a roundoff process can be chosen so that the resultant error statistically cancels in an extended computation.

Rounding must be separately investigated for the number system of interest. Generally, it consists of *half-adjusting* which in decimal adds 5 (in binary, 1) in the column to the right of the column to be rounded on,

### Sign Digit

An algebraic sign must be attached to each number. In binary machines the leftmost bit is generally reserved for this feature and a common choice is 0 for positive, 1 for negative. In decimal machines, the leftmost decimal digit is often used for sign information, and the other 8 values of the digit (only two are needed for sign information) may be used for special purposes.

The treatment of the sign digit depends on the means of representing negative numbers. In sign-magnitude representation, the sign digit does not enter into the arithmetic process but is treated separately. In complement representation, the sign digit enters into the arithmetic operation as though it were a numerical digit. The examples of the paragraph on Complement Representation demonstrate this; the sign digit is the 0 or 9 to the left of the point.

### Overflow

Within a machine, the result of an arithmetic operation can exceed the range of numbers which it can accommodate. Most machines detect such an overflow situation. A machine reaction to an overflow can be a console indication, a halt, or an automatic correction procedure. Rules for detecting overflow depend upon the manner of representing negative numbers. For sign and magnitude machines, it is sufficient to detect a carry from the most significant digit column. For a complement machine, if numbers  $N_1$  and  $N_2$  are being added, the rules are:

1. If the sign digits of  $N_1$  and  $N_2$  differ, there cannot be an overflow.
2. If the sign digits of  $N_1$  and  $N_2$  are alike, there is or is not an overflow according as the sign digit of the answer disagrees or agrees with the like sign digits of  $N_1$  and  $N_2$ .

## 6. COMPUTER DESIGN CHARACTERISTICS

Computing machines may be classified according to a number of characteristics. A representative list of the more pertinent, with a short discussion of each, follows.

**General Purpose vs. Special Purpose.** Digital computing machines intended to accommodate the broadest possible class of problems are categorically known as *general purpose* machines. On the other hand, one designed to be particularly efficient for a special class of problems is known as a *special purpose* machine. An example of the latter is the digital differential analyzer for which the special class of problems is ordinary total differential equations.

**Serial vs. Parallel.** The particular property for deciding whether a particular system is serial or parallel is not generally agreed upon. Frequently it is the mode in which the arithmetic unit operates. If successive digits of a number are handled in time sequence with the same equipment for each digit, the machine is said to perform *serial arithmetic*. If sufficient equipment is provided that all digit columns of a number may be operated upon essentially simultaneously, the machine is said to perform *parallel arithmetic*. Binary coded machines may be hybrid *serial-parallel* machines in that the 4 bits of the coded tetrad may be operated upon in parallel, but the successive decimal digits operated on in series; *serial-serial* machines also exist. Serial and parallel may also be applied to transmission processes within a machine system. If the digits of a word are transmitted essentially simultaneously over parallel transmission channels, the machine is said to utilize *parallel transmission*; a machine having *serial transmission* will transmit the successive digits of a word in time sequence over a single transmission channel. Hybrid systems may exist.

**Binary vs. Decimal vs. Alphanumeric.** A machine system organized to perform arithmetic operations in the binary number system is known as a *binary machine*. A machine organized to perform arithmetic processes in terms of some binary coded decimal format is known as a *decimal* or *binary-coded-decimal machine*. Frequently a decimal machine also includes some binary instructions to provide flexibility in manipulating data in logical processes. Some machines have been referred to as *octal* or *hexadecimal machines* but they are actually true binary machines, in which the binary digits are grouped in triads or tetrads respectively and are interpreted as base 8 or base 16 digits.

A machine organized to deal with 6-bit or larger groups in order to accommodate alphabetic and numeric information is referred to as an *alphanumeric* or *alphameric machine*.

**Type of Storage.** A machine system is frequently characterized by the nature of the principal internal storage. Generally, a machine system will have a hierarchy of storage devices where the most rapid form of storage may be thought of as the principal internal working storage, supported by slower and more capacious levels of storage. A machine whose principal storage is a magnetic drum is termed a *magnetic drum machine*. Alternatively, a machine whose principal internal storage is electrostatic might be termed an *electrostatic storage machine*, and so forth. The nature of the secondary storage, tertiary storage, and such other levels of storage as may exist, is often not specified in the short description of a system.

Storage may be *random access*, which implies that the time required

to consult any location is essentially the same; or it may be *serial access*, which implies that the locations appear in an ordered sequence and hence, the time to consult a location is variable and depends upon when the request is received by the storage and possibly upon past history. These characteristics may be likened to a pigeonhole file (any hole reachable in substantially the same time) and to a tub file respectively (the information comes sequentially and waiting time is a function of the position of the tub). By extension, random access has come to be used to describe storages whose access time may vary but whose maximum access time is short compared to the interval between successive consultations of the storage. Thus, a given storage device may be random access for one machine system but serial access for another. Further, for a fixed machine system, a given storage may be random access for one problem but serial access for another.

With respect to the rest of the machine, the store may exhibit a *destructive* or *nondestructive read* process. These imply respectively that the act of consulting the storage for a piece of information destroys or does not destroy that piece of information within the storage. The nondestructive storage has the advantage that a given location may be repeatedly read for the same information without any special care in programming.

**Word Length.** A word is the unit of information usually handled within a digital computing system and is measured in terms of the number of characters it contains. If the *word* is constrained to be numeric only, then its length will be measured as so many binary digits, or so many octal digits, or so many decimal digits, and so forth. If, on the other hand, a word may be either numeric or alphabetic or mixed, then its length will be measured as so many characters, where a character may be numeric, alphabetic, or other special symbols.

A machine may be organized as a *fixed word length machine* in which case all words are of constant length. Alternatively, it may be a *variable word length machine* in which case the beginning and ending of each word must be indicated, or an *adjustable word length machine* in which the length of the word may be adjusted to suit the problem.

If insufficient precision is available from numbers one word in length, a machine may be programmed to compute with numbers which are actually  $n$  words in length. Usually this is done for numbers two words long, and such numbers are called *double length numbers*.

**Number of Addresses per Instruction.** A machine system whose instructions contain one address per instruction is known as a *single-address machine*. Machines whose instructions contain two addresses are known as *two-address machines*, and so forth. It may be that all

of the several addresses in an instruction are not of the same length. Commonly an address in the instruction will be capable of referring to the entirety of the principal internal storage. Sometimes an address within an instruction is used to refer to other parts of the machine as, for instance, the registers of the arithmetic unit, a particular one of many input-output units, and so forth. Under these circumstances the address need not be as long as one intended to refer to the entire principal storage, and is sometimes called a *degenerate address*. Machines with this structure are sometimes designated as *one and one-half address machines*, *two and one-half address machines* and so forth.

Sometimes an address does not refer to an absolute location in storage, but indicates an incremental change from the last-used address. This arrangement is referred to as *relative address organization*.

**Nature of the Control Section.** It is possible to organize the control section of a digital computer in two broad ways. The *synchronous* machine has a *clock* within the control section which provides the source of all timing signals needed by the machine. All electronic activity in the machine is paced by these regularly recurring clock signals, and no event occurs within the machine except at one of the clock signals. Between signals, transient phenomena are allowed to decay. A synchronous machine is sometimes called a *clocked machine*.

The control may also be organized so that each event within the entire system is permitted to proceed at a rate which is governed only by the natural time constants of that event. All other events are effectively interlocked so that no other may occur until the stated one is complete. At that time, the completing event indicates the termination of its cycle and invites, so to speak, the beginning of the next event in sequence. This type of organization is referred to as *asynchronous control*.

If the events in section A of a computing machine are occurring at essentially random intervals or at a rate which is unrelated to the occurrence of events in section B, then the term *asynchronous* is sometimes used to state that section A is asynchronous with respect to section B. In this usage there is no implication as to whether section A or section B is in itself either synchronously or asynchronously organized. An example of this case is that of a magnetic drum running at its own clock rate but having to communicate with a synchronous machine running at its own clock rate. At the time of communication between them, the two devices must be synchronized for the interval during which information is being transmitted.

**Fixed Point vs. Floating Point.** If the position of the radix point, e.g., binary point or decimal point, is always fixed in position with respect to the machine word, the machine is said to be a *fixed-point*

*machine*. The two most common cases are: the point at the right end of a word, in which case the machine performs integral arithmetic; the point at the left end of the word, in which case the machine performs fractional arithmetic. The position of the point may not be fixed with respect to the word but may be indicated by additional information contained within the word. This type of operation is known as *floating-point operation* and is similar to the widely used scientific notation. The number 636,107 may be written as  $0.636107 \times 10^6$  or the number 0.00053 may be written as  $0.53 \times 10^{-3}$ . A floating-point word would then contain a *magnitude* which in the above examples would be 0.636107 or 0.530000 and an *exponent* which in the above examples is +6 or -3. Floating point accommodates an extremely wide range of size in the numbers that can be handled, but careless use can give rise to serious errors or can be misleading as to the number of significant digits in the result.

In a fixed-point computation there sometimes is considerable difficulty in adjusting the parameters of a problem so that all the numerical information fits within the range of numbers which the machine system can accommodate. Fitting a problem to a machine so far as number size is concerned is referred to as *scaling*.

**Internally vs. Externally Programmed Machines.** If the list of instructions directing a machine system is in the principal internal storage of the system, the machine is known as an *internally programmed* or *stored program* machine. The advantage of this method is that the storage can be allocated as much to routine and as much to data as a particular problem demands. A further advantage is that the machine system can have access to its own instructions and can therefore alter them by performing arithmetic or logical operations upon these instructions; thereby it can govern its future course of progress on the basis of its past results. Most of the electronic computing machines in existence today are the internally programmed type. Alternatively, a machine may be organized so that its list of instructions is outside its principal operating storage, in which case the machine generally cannot have access to these instructions to change them. Such a machine is referred to as an *externally programmed machine* and is typified by the IBM Card Program Calculator or the Harvard Mark I machine. The latter machine stores its list of instructions on punched paper tape.

**Real Time or On-Line Operation.** A computer designated as a *real time machine* is one in which the operation rate is sufficiently rapid that it can perform the problem solution in the same time that a parallel physical process occurs. In machine simulation of physical processes, real time implies that the machine accomplishes the simulation in the

same time as that required by the original physical process. An *on-line machine* is one which processes data in synchronism with some physical process such that the results of the computation are useful to this physical operation. Many of the applications of computing machines in automation or in control fall into either of these categories, particularly applications to process control.

### REFERENCES

1. Engineering Research Associates, *High Speed Computing Machines*, McGraw-Hill, New York, 1950.
2. R. K. Richards, *Arithmetic Operations in Digital Computers*, Van Nostrand, Princeton, N. J., 1955.
3. Standards on Electronic Computers, *Proc. I.R.E.*, **44** (9) (1956).
4. Staff of Harvard Computational Laboratory, Synthesis of electronic computing and control circuits, *Ann. Harvard Computation Lab.*, **27**, Harvard Univ. Press, Cambridge, Mass., 1951.
5. J. V. Uspensky and M. A. Heaslet, *Elementary Number Theory*, McGraw-Hill, New York, 1939.
6. I. Flores, Reflected number systems, *I.R.E. Trans. Elec. Computers*, **EC-5**, 79-82, June 1956.
7. R. W. Hamming, *Bell System Tech. J.*, **29**, 147 (1950).
8. W. Keister, A. E. Ritchie, and S. H. Washburn, *Design of Switching Circuits*, Van Nostrand, Princeton, N. J., 1951.
9. F. B. Hildebrand, *Introduction to Numerical Analysis*, McGraw-Hill, New York, 1956.

## Techniques for Reliability

*Willis H. Ware*

1. Introduction	13-01
2. Summary of Operating and Design Techniques	13-02
3. Operating Techniques	13-04
4. System Design	13-05
5. Circuit Design	13-07
6. Maintenance	13-08
References	13-10

### I. INTRODUCTION

Digital computer systems must exhibit "instantaneous reliability," e.g., a single failure could (1) change an instruction into a completely different kind, (2) cause the machine to consult a wrong address, (3) cause the machine to execute a meaningless sequence of instructions, or (4) introduce an undetected error which unbeknown to the user invalidates the final solution.

Contributions to the overall operational reliability of a digital system can be made at three levels: (1) operating techniques, (2) design techniques, and (3) maintenance techniques.

The importance of the type of error that will occur will depend, to some extent, on the use to which the digital computer is applied. In general, computer applications can be divided into three categories: (1) scientific and engineering computations, (2) business data processing, and (3) on-line control operations. In the first two cases, low reliability

will increase the cost and time of operation. In the third case, computer malfunction can result in loss of product or plant shutdown. Hence, for on-line control, higher reliability is required, and computer reliability must be considered as a system parameter.

In the typical electronic analog computing device, there exist two distinct kinds of feedback (see Refs. 1 and 4).

1. Electronic circuit feedback techniques are employed to stabilize the various parts of the analog system against fluctuation in performance and parameter values of the circuit components.

2. Feedback loops exist by virtue of the way in which the analog computer mechanizes the solution of the differential or other type of equation.

Because of these two kinds of feedback protection the analog computer is, in a large sense, proof against relatively major fluctuations in the performance of its components. It need, therefore, exhibit only "reliability on the average." In the electronic digital computing system the situation is radically different. Here, because of the high speeds of operation of the all-electronic circuits, it is often difficult to make use of feedback techniques in circuit design. More importantly, however, the methods of problem preparation for digital machines are not such as to exhibit the second, or "logical" kind of feedback. Alternate techniques that can be used are discussed in Sect. 3.

## 2. SUMMARY OF OPERATING AND DESIGN TECHNIQUES

Table 1 summarizes the characteristics of various operating and design checks.

**Evaluation Criteria.** In determining which programmed checks are appropriate, the user will consider such things as:

1. The cost in machine time for the various checks.
2. The behavior pattern and the idiosyncrasies of the machine in use, especially where and how it tends to give trouble.
3. The consequences in time or money if the solution proves wrong.
4. The nuisance of having to rerun part or all of the problem.

The programmer will consider the trade-off between an investment in some amount of machine time for programmed checks versus the expectation that some or all of the problem may have to be rerun because of machine malfunctions.

In determining which checks should be built into a machine, the designer will consider such things as:

1. The cost of the equipment which must be added for the checking features.
2. His faith in the ability of his components and his techniques to perform satisfactorily.

TABLE 1. SUMMARY OF OPERATING AND DESIGN CHECKS

Name of Check	Programmed	Automatic	Errors Likely to be Detected	Check Usually for:			Remarks
				Trans. Oper.	Arith. Oper.	Store Oper.	
Duplication	Usually	Sometimes	Systematic or transient errors in a machine operation.	x	x	x	When automatic, only part of machine may be duplex (Univac I) or entire machine (Sage AN/FSQ-7).
Reasonableness	Usually	Unlikely	Gross mistakes in the solution.	x	x	x	Yields fairly rough measures of confidence in details of solution.
Mathematical	Usually	Unlikely	Depending on the mathematical properties employed, gross or detailed, transient or systematic machine malfunctions.	x	x	x	Restricted generally to scientific and engineering problems.
Check sum	Usually	Possible	Any which are not compensating with respect to addition.	x		x	May be applied to any transfer of a block of data.
Parity	Possible	Usually	Any odd number of errors which are not compensating with respect to this check.	x		x	Very common with stores, especially magnetic tapes.
Self-check code	Possible but unlikely	Usually	Any errors giving rise to a forbidden combination in the code.	x		x	Similar to forbidden combination check except code groups here are usually longer. Examples are: Bell Relay Calculators, IBM-650.
Error detecting-correcting code	Possible but unlikely	Usually	Any for which code is designed; e.g., detect single and double but correct only single.	x		x	Rarely used unless expense of large amount of additional equipment is justified.
Weighted check	Possible	Usually	Probably transpositions and any which are not compensating under the arithmetic or logical technique used.	x		x	Examples are: Raydac, Datamatic-1000.
Forbidden combination	Possible but unlikely	Usually	Any which produce an illegal combination of the code in question.	x		x	Relatively common in binary coded decimal machines.
Casting out	Possible	Usually	All except those which make the result of the operation wrong by: (any integer times the number cast out).		x		Examples are: Raydac, NORC, Datamatic-1000.

3. The environment in which the machine may have to operate, especially as the environment may affect the performance of the equipment; e.g., dirt and dust on the performance of magnetic tapes.

4. The possibility that, since the checking equipment increases the total quantity of equipment in the system, the checked system may be no more reliable than the smaller unchecked system.

5. The possibility that a less carefully designed basic machine supported by an adequate checking system may result in a more economic but equally reliable design.

6. The class of problems for which the machine system is intended; e.g., business data processing may be more demanding of machine performance than scientific problems.

### 3. OPERATING TECHNIQUES (Ref. 2)

**Programmed Checks.** The programmer, in preparing a problem for a digital system, has available several possible schemes for incorporating into a routine additional operations which serve as checks on the proper functioning of the machine. The most valuable techniques will be those which check large segments of machine operation, and which serve to catch transient as well as systematic malfunctions of the equipment. Such checks are called *programmed checks*.

Programmed checks may also be used to improve the operation of the system of which the computer is a part. Examples are reasonableness checks on input data and mathematical error minimizing techniques. Such programming techniques, which reject, smooth, or filter errors due to malfunction of input equipment, measuring equipment, or other segments of the system, will not be included in this discussion.

**Duplication of Machine Operations.** A possible technique is duplication of an operation, either arithmetic or data transfer type. This, however, will fail to catch systematic malfunctions of the system unless, in the checking operation, the equipment is used in a completely different fashion.

#### EXAMPLES.

1. Form  $ab$  the first time, but form  $ba$  for the check multiplication.
2. Form  $a + b = c$ , but form  $c - a$  and compare the result with  $b$  for the check.
3. Extract a square root by an iterative process but check by squaring the result and comparing with the original operand.

**Reasonableness Checks.** In many problems there are bounds to certain problem parameters or to certain solutions of the problem. The existence of any supplementary information about the problem can be

used to increase the confidence in a solution. When applied to the input data, this is often called a *consistency check*.

EXAMPLES. (1) The total payroll for a given period is less than some maximum amount. (2) No individual payroll check is larger than some maximum amount. (3) The specific gravity in a physical problem must always be a positive number. (4) The size of inventory for each item falls within some bound. (5) The magnitudes or rates of change of certain physical quantities fall within some bound.

**Mathematical Checks.** Frequently such checks are available only in the scientific problem where the problem is one of mathematical as opposed to data processing nature. This check depends on the mathematical properties of a sequence of operations.

EXAMPLES. (1) If the sine must be computed, also compute the cosine, and verify that the sum of the squares is unity. (2) A sufficiently high order difference of the solution of a differential equation exhibits relative constancy. (3) Upper and lower bounds are known for the variables of the problem.

**Check Sums** (Ref. 5). A block of data and/or instructions is summed as though it were a sequence of numbers. This check sum is carried with the block as a tag. At any subsequent time the block may be verified by again summing and checking the new sum against the previous tag.

#### 4. SYSTEM DESIGN (Refs. 2, 4)

**Choice of System Organization.** By adding redundancy to a digital system in appropriate ways, reliability may be increased by endowing the system with an ability to detect, or possibly to detect and correct either single or multiple errors. A check of this kind, if it involves additional equipment in the system, is referred to as an *automatic* or, sometimes, a *machine check*. If the normal machine instructions in an appropriate routine are used to accomplish the check, this is another form of a programmed check. Different checks must be used for data transfer and arithmetic processes.

#### Data Transfer Checks

These techniques are useful for verifying the correctness of data transfers or for verifying the proper functioning of a storage device. They do not behave properly under arithmetic processes. Such a check could be used to (a) check data transfer from the internal store to an output device, (b) check data transfer from a magnetic tape to the internal store, (c) check that information retrieved from a storage device is without errors of the kind which the particular check can detect.

**Parity Check** (Ref. 6). In the parity check (see Chap. 12), an additional binary digit is added to a message in such a way that the number of 1's in the message is, say, even. The parity check could also be made odd rather than even, or on the number of 0's rather than the number of 1's. An extension of this scheme would add redundant information about both the number of 0's and number of 1's. The message to which a parity or check digit is added might be a word, a character, or a block of data. A simple parity check will detect only an odd number of errors per message.

**Self-Checking Codes.** In such codes some characteristic of each code group is a priori known; e.g., each code group contains a specified number of 1's, no code group contains all 0's. An example of such a code is the biquinary code first utilized in early relay calculators (see Chap. 12). A particular kind of self-checking code is the error-detecting and error-correcting code (see Ref. 3). In such a code additional bits of information are added in such a way that the presence of single or multiple errors is indicated. If sufficient additional information is added, it is possible to correct for single or multiple errors automatically.

**Weighted Check.** A check tag is derived from the message to be checked and carried with the message. At any subsequent time, the message may be verified by again deriving the tag and comparing it to the tag accompanying the message. This technique may detect transpositions in the message.

**EXAMPLE.** (1) Divide the message by an appropriate factor (often a prime number) and use the remainder as the tag. (2) Sum the digits of the message "casting out" multiples of an appropriate factor [often the (base - 1)] (see Arithmetic Checks: Casting Out).

**Forbidden Combinations.** A kind of redundancy check which is applicable to several of the above schemes is the *forbidden combination check*. Here a simple test is made to determine that a given code group is one of the allowable set of code groups which a machine utilizes; e.g., in a binary coded decimal machine, the six discarded combinations of the sixteen binary tetrads would be regarded as forbidden combinations, and their presence would therefore indicate an error.

### Arithmetic Checks

An automatic check for the arithmetic operation is considerably more difficult to implement.

**Duplication of the Arithmetic Section.** In such a system all arithmetic operations are performed twice, simultaneously by different sets of equipment.

**Casting Out.** For this process additional equipment over and beyond that of the normal arithmetic section is provided to perform the casting out operation.

**EXAMPLE.** In the check procedure multiples of 9 might be discarded.

<i>Operation</i>	<i>Check</i>
1978	$1 + 9 + 7 + 8 = 25; \quad 25 - 18 = 7$ <span style="padding-left: 100px;">(2×9)</span>
<u>×2156</u>	$2 + 1 + 5 + 6 = 14; \quad 14 - 9 = \underline{\times 5}$ <span style="padding-left: 100px;">(1×9)</span>
11868	
9890	
1978	
<u>3956</u>	
4264568	$4 + 2 + 6 + 4 + 5 + 6 + 8 = 35; \quad 35 - 27 = \boxed{8}$ <span style="padding-left: 100px;">(3×9)</span>

The equality of the two check results gives a high probability that the original operation was done correctly.

Other than 9's might be cast out, but the total number of possible errors checked for will be different; e.g., if 2's are cast out, only the parity (evenness or oddness) of the final digit of the product will be verified; if 99's are cast out, a larger class of errors will be detected but the equipment or time required to perform the check increases.

## 5. CIRCUIT DESIGN

The typical computing system represents, relatively speaking, a substantial quantity of electronic components. The parameters that specify the characteristics of the components will be statistically distributed; these statistical distributions will be a function of (1) the manufacturing process which produced these components, and (2) time through various aging effects, through temperature and humidity effects, and through fabrication techniques, such as those which might induce temperature cycling. (See Chap. 14.)

**Component Variations.** It is necessary at the outset that the designer incorporate in his design technique means for accommodating such statistical fluctuation of component parameters. This generally means, for instance, that a given circuit design must operate satisfactorily, even though, say, all resistors deviate from normal values by 5 per cent; all active elements deviate from normal parameters by perhaps 25 per cent; supply voltages deviate from normal values by perhaps 10 per cent; and

all the deviations must be assumed to occur simultaneously and in the worst possible directions. By definition, the tolerance of any component parameter for which the circuit fails, specifies *end of life* for the component in question. In order to maintain a circuit near the center of its region of operability, components which reach end of life (as here defined) will usually be periodically replaced, even though such components may have considerable additional life expectancy for other applications.

**Environment.** The operating environment of a component may drastically affect its life expectancy. Frequently components designed for commercial application are rated in such a way that the life expectancy is considerably less than that required by a digital application; therefore, it has become customary in digital circuit design to apply derating factors to the commercially established characteristics. For example: (1) resistors may be operated at 50 per cent dissipation rating; (2) capacitors may be operated at 80 per cent voltage rating; (3) active elements may be operated at 50 per cent of current rating.

**Design Philosophy.** A conservative attitude on the part of the designer toward the performance of his circuit and the performance which he expects from his components is valuable. In some cases this may require additional components in order to absorb parameter variations of the basic parts of the circuit. In other cases it may require additional circuits in order to avoid possible faults which the designer might foresee in the performance of his basic circuit. The designer attempts to "design away" from all the difficulties of which he can conceive.

## 6. MAINTENANCE

**Scheduled.** Maintenance for the typical digital system usually consists of regularly scheduled *preventive maintenance* periods, in which the system is carefully checked for its compliance with design center performance. In such preventive maintenance periods, there may be typically a systematic:

1. Cleaning of relay contacts.
2. Checking of parameters of active elements.
3. Visual inspection of solder joints, or condition of plugs.
4. Inspections by means of an oscilloscope of signals throughout the system.
5. Replacement of components whose parameters are approaching end-of-life values.
6. Cleaning and lubrication of mechanical equipment.
7. Inspection of mechanical components for loose connections.
8. Verification of system performance by use of test problems.

9. Verification of system performance by marginal checking.

**Unscheduled.** In addition to scheduled maintenance, there also will occur occasional intervals of unscheduled maintenance, in which an unexpected failure causes catastrophic or transient machine malfunction.

**Maintenance Techniques.** For performing maintenance, the following specific features typically are available.

1. *Marginal Checking Facilities.* This is an arrangement for systematically varying the operating environment of a machine, e.g., basic pulse widths and repetition rate, supply voltages, temperature of cooling air. Generally, facilities are provided for varying all aspects of the operating environment in all combinations of ways. Sometimes it is possible for the machine itself to control a systematic search through all combinations of its possible marginal situations. This technique is valuable in indicating the location or existence of incipient difficulties.

2. *A Carefully Designed Maintenance Console.* From this facility it is usually possible to monitor a major part of the internal operation of the system; e.g., all flip-flops may be reported by suitable indicators; supply voltages may be directly indicated; various alarm conditions may be reported; internal conditions may be manually controlled.

3. *Diagnostic Problems.* These are of the nature of a test problem for the machine, usually arranged so that the nature of the failure in such a problem will provide information about the existence and/or the location of trouble. Sometimes such problems are referred to as *trouble location* problems if their prime purpose is to indicate the location of difficulty, or *trouble detection* problems if their prime purpose is to indicate the presence of some difficulty. Such diagnostic routines generally grow to cover a more exhaustive set of possible difficulties with a machine, as maintenance experience with the machine accumulates. Care must be taken that the maintenance personnel do not "groom" the machine to do its diagnostic problems while not really fixing a fault. Often the best kind of diagnostic problem is the problem itself on which the machine system failed.

4. *Miscellaneous.* As a measure of the ability of an operating system to survive unpredicted environmental influences, techniques such as the following are frequently employed.

(a) Subjection of the equipment to a low level of mechanical shock; as, for instance, by tapping with a rubber mallet. This tends to cause incipient difficulties to appear as permanent so that they may be remedied during scheduled periods of maintenance, e.g., poor solder joints, loose tube elements, faulty socket connections.

(b) Subjection of the equipment to extraneous electrical interference as, for instance, might be generated by a spark discharge. This technique

tends to measure the ability of a system to reject noise interference, which is always present in a machine environment but usually difficult to locate as a source of trouble, since most often it would produce malfunctions of a transient nature.

### REFERENCES

1. W. H. Ware, *Proc. West. Jt. Comp. Conf.*, 27-31, February, 1957.
2. Ref. 1, Chap. 1, and definition of *check*, Chap. 1.
3. R. Hamming, *Bell System Tech. J.*, **29**, 147 (1950).
4. A. W. Boldyreff, *Proc. West. Jt. Comp. Conf.*, 18-20, February, 1957.
5. D. D. McCracken, *Digital Computer Programming*, Wiley, New York, 1957, p. 154.
6. D. D. McCracken, *Digital Computer Programming*, Wiley, New York, 1957, p. 151.

## Components and Basic Circuits

*Norman H. Taylor*

1. Designing for Reliability	14-01
2. Components and Circuit Design	14-03
3. Marginal Checking	14-05
4. Reliable Computer Circuits	14-19
5. Components, Characteristics, and Application Notes	14-43
6. Transistors	14-51
References	14-54

### I. DESIGNING FOR RELIABILITY

*Note.* The research described in this chapter, carried out at Lincoln Laboratory, was supported jointly by the Army, Navy, and Air Force under contract with the Massachusetts Institute of Technology.

**Design Philosophy.** Circuits have been designed and constructed using the design philosophy described in this chapter. A typical circuit including its tube has an operating lifetime of over 100,000 hours. A routine maintenance scheme in which the marginal-checking principle is properly exploited can predict failures of over 90 per cent of the total to be expected. System operating efficiency (useful time/total time) over 95 per cent is attainable even in systems where 20,000 circuits are employed.

The achievement of reliability is a goal that must be pursued from the very beginning of the system design project.

*Step 1.* Consider each individual component to be used in the system and critically analyze its capabilities.

*Step 2.* Employ the data from Step 1 to evaluate applications of these components that avoid their worst limitations. This chapter lists analyses of components and the resulting component applications that have been made by the staff of Lincoln Laboratory over a period of several years.

*Step 3.* The final phase of the design project is the electronic circuit design, based on the component analyses and applications notes derived earlier, with the objective of the achievement of high reliability. The thorough design method developed by Lincoln Laboratory is described in detail. This method provides reasonable component tolerances and adequate safety margins, and incorporates marginal checking throughout the design process.

The method is illustrated in a detailed example of a high-speed vacuum tube flip-flop. Additional tube and transistor basic building-block circuits are described and discussed in Sects. 5 and 6 of this chapter. These basic circuits have been used in the assembly of large data processing systems.

**Reliability in Control Systems.** In approaching the problem of reliability in computing and control systems, the concept changes considerably from the commonly accepted rules of the radio, television, and home appliance field. In the area of automatic control, the system under control is often a very costly one. It places a premium on its controlling parts; and, in military operations, human life itself is sometimes dependent on the reliability of the controlling electronics. Therefore, the electronic designer is faced with an extremely stringent design requirement and he must "design for reliability" from the start, even changing the systems concept, if necessary, to insure the desired result.

Three vulnerable areas influence reliability: components, component application, and design.

**Components.** Choice of components has a first order effect on system reliability. Two factors in component manufacture must be considered:

*Component Stability.* Components are never absolutely stable. They drift in value with time, temperature, humidity, and altitude. Stability factors must be known and considered before design work is undertaken.

*Component Reproducibility.* This is a factor of production tolerances. The 1 per cent resistor is now common, but the tube with 1 per cent tolerance in plate current has never been built. Tolerances must be known to the circuit designer and taken into account before design work is started.

**Component Application.** The way in which a component is used, with consideration given to the problems of stability and reproducibility, is the second major factor contributing to reliability. Ideally, the natural

properties of the component are exploited, and its inherent weaknesses are avoided or bypassed by careful design.

**Design Considerations.** The design of specific circuits is the third requirement for reliable electronics. Of the three factors in reliability, the design phase is the more difficult, because it must encompass the decisions and account for the boundary conditions imposed by the other two factors.

## 2. COMPONENTS AND CIRCUIT DESIGN

**Component Failures.** A good method of appraising a particular application of a component is an evaluation based on the four major types of failures. These are:

1. *Deterioration.* Deterioration is a disease that the designer must face. "How fast is this component going to wear out in this application?" is the question which must be answered.

2. *Sudden Failures.* Sudden failure is self-explanatory. A vacuum tube loses its vacuum, a resistor opens up, a fuse blows—these are complete, sudden failures.

3. *Intermittent Failures.* Means are available for coping with almost all types of failures except intermittents. Eventually, intermittents must be designed out of the components wherever possible.

4. *Maladjustments.* These are related to proper maintenance. The best possible solution is to create a design without adjustments. This, of course, is not always possible, but attention must be given to permissible adjustments and recommended procedures.

**EXAMPLE. Magnetic Cores.** A good example of a component application that meets the basic requirements of reliable design is the storage core in a magnetic storage plane. Failure characteristics are:

1. The component deteriorates slowly; in fact, present data indicate practically no change in characteristics with time.

2. Sudden failures are rare, limited to broken cores and usually occurring only in assembly.

3. Intermittents are rare; adequate insulation on the wires through the cores is all that is required.

4. No adjustments are necessary.

In addition, cores are stable with respect to temperature and humidity, and are reproducible to close tolerances.

The core storage itself is not useful without driving and sensing circuits. However, these auxiliary circuits cannot be made as reliable as the cores themselves. Magnetic core storage units have run for weeks and even months with complete freedom from error. They are presently the most reliable part of high-speed computers.

**Component Vulnerability.** Although a core is a good component, current techniques do not allow systems to be built entirely of cores. The designer is faced with the choice and use of many other components that are more subject to deterioration, intermittents, sudden failures, maladjustments, instability, and limitations in reproducibility. The following list suggests an order of vulnerability, with the most vulnerable components at the head of the list: (1) vacuum tubes, (2) diodes, (3) connectors, (4) relays, (5) resistors, (6) condensers, (7) transformers, (8) inductors, (9) cores. Transistors have not been included since there is not an equivalent amount of supporting information. On the basis of present units, they probably rank either above or below diodes.

The individual characteristics of tubes and other components as these affect reliability are discussed in detail in Sect. 5, and notes on applications of these components are also included there. Transistors are treated, briefly, in Sect. 6. (See also Chap. 16.)

**Design Considerations. Safety Margins.** The electronic designer is not accustomed to provide safety margins adequate to allow for the various disturbances that may occur in a circuit during its lifetime. The increasing use of electronic controls in larger and more critical areas forces the circuit engineer to pay greater attention to these factors.

The circuit engineer is usually asked to make a circuit perform some specific function as a portion of a system. In the computer field, such a requirement may be for a high-speed switcher: Performance specifications are given on the limits on the speed of switching, the voltage swing that the switch should deliver, the resolution time, and perhaps some power limitation. The *circuit designer* must know about component stability and tolerances, and must build his design around such knowledge.

Thus, the problem that really confronts the circuit designer becomes one of designing highly reliable and stable circuitry made up of components that are not reproducible and which are subject to deterioration, intermittents, maladjustment, and sudden failure. "Components," as used above, include power supply voltages, diode characteristics, tube characteristics, and resistors, anything that can change to the detriment of circuit performance.

**Design Criteria.** Three criteria are effective guides for the recognition of component tolerances and their effect on adequate safety margins.

**DESIGN CRITERION 1.** *The circuit must meet its performance specifications with all components at their worst initial tolerances, and with any component at its worst end-of-life tolerance.*

By "worst" is meant deviation of the components in whatever direction is least favorable for the circuit. Usually, several worst combinations of components must be evaluated. The initial tolerance on a composition

resistor might be  $\pm 5$  per cent, and end-of-life  $\pm 15$  per cent. The numbers to use for these tolerances must be the result of a component study, as discussed in Sect. 5.

In some simple circuits, it might be possible to have all components deteriorate to end-of-life at once, and still have satisfactory performance; this capability is desirable only if it requires no increase in circuit complexity. Added components result in a greater probability of intermittents; the increased complexity results in more difficult servicing. Statistically, one of a group of components will reach end-of-life while most of the others are still good; therefore, criterion 1 above is suggested as being a good engineering compromise in the search for circuit longevity.

When a circuit geometry has been found that shows promise of meeting criterion 1 above, criteria 2 and 3 must be considered and met.

**DESIGN CRITERION 2.** *The circuit must be able to withstand the loss of any one of the supply voltages in itself or in any circuit connected to its input or output without component damage.*

When diode logic is used, criterion 2 will usually necessitate use of protection diodes to prevent excessive back voltages.

**DESIGN CRITERION 3.** *Since all components (especially vacuum tubes) degenerate with life, circuit design should include means for detecting significant changes in component values during use of the circuit, and soon enough to insure replacement of the component before failure occurs.*

Provision of means for detecting deterioration allows near failures of components to be discovered and eliminated during routine, scheduled maintenance periods. Costly, unscheduled downtime can be minimized. Slow deterioration of tubes and other components may be observed over a period of time, and replacement can be predicted and even scheduled through the application of marginal checking.

### 3. MARGINAL CHECKING

**Requirements.** To determine whether or not a circuit design meets the given performance specifications with the desired reliability, there is need for a method of evaluation that will (1) make graphically clear, in an explicit quantitative way, what tolerance a given circuit has to variations in its components, and (2) provide a method, usable in the later systems phase, of preventive maintenance that will adequately cope with the problems of component deterioration. Such a method for *marginal checking* was developed by Lincoln Laboratory; it has been extensively used in the design phases of large real time control systems, as well as in day-by-day operation of such systems.

In the design phase, the allowable variation of a component is determined as a function of a selected circuit parameter, usually a supply

voltage. This measures the margins of circuit performance in terms of the marginal-checking parameter.

**Tolerance Plots.** In practice, the tolerance of one of the components in the circuit is plotted against the variation in this marginal-checking parameter, as illustrated in Fig. 1. The intersection of mean-value and normal marginal-checking parameter lines near the center of the parabola indicates the operating point of the circuit, i.e., normal voltage on the

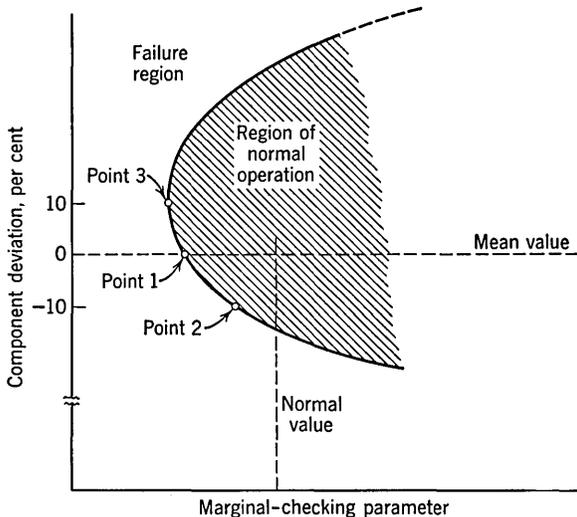


FIG. 1. Locus of failure points in a typical circuit.

circuit and mean value of the components. By considering the supply voltage as the marginal-checking parameter and lowering it, a point is plotted on the contour line where the circuit fails to perform. This failure can be defined as the point at which the function of the circuit deviates from that prescribed in the specification. In an oscillator, for instance, the point at which the frequency shifts out of tolerance can be considered failure; in a flip-flop, the point at which some standard pulse fails to switch the position may be failure. Changing the tolerance on the component by some factor such as 10 per cent in a negative direction and again varying the marginal-checking voltage will result in a different failure point, such as Point 2 on the curve. By raising the tolerance of the component 10 per cent, another failure point, Point 3, can be plotted. When this study is continued, a contour line representing the locus of the failure point of the circuit to tolerance in componentry, as a function of some marginal-checking parameter, can be drawn enclosing an area of reliable operation. The result is that the contour often

is not symmetrical about the operating point, and that wide safety margins occur on one side but very narrow margins occur on the other.

In most cases, the contour would be a closed loop if the marginal-checking parameters could be varied far enough without damaging the components.

Plotting the curves and varying each of the components in even a moderately complex circuit represents a rather long and tedious study. However, the reader should remember that the circuits under discussion are to be subjected to all sorts of variations; failure in such circuits may cause losses of life or of large sums of money. With these factors in mind, the acceptability of the circuit to the system can be based only on the knowledge of tolerance plots.

### **Marginal Checking: Components**

The application of marginal-checking procedures to components involves a variety of techniques which can be tailored for the particular problem at hand. The discussion in Sect. 1 assumed a one-dimensional (one-parameter) tolerance plot; whereas, of course, in reality there are many dimensions that vary simultaneously. Therefore, numerous experiments are usually needed to verify the initial studies.

Ideally, one would like to plot a curve for each component of the circuit, with the component's deviation from nominal value on one axis and the marginal-checking parameter on the other, resulting in points on the curve that represent the boundary between satisfactory and unsatisfactory operation. To ease this job, engineering judgment must be used to determine which data shall be taken. Enough data *must* be taken so that the effect of change in each component may be deduced.

All the component variations (or branch supply voltages where applicable) must be plotted against the marginal-checking voltage to determine that the required component tolerance is not prohibitive, and that the normal operating point is centered in the area of operation. If a component tolerance turns out to be +1 per cent and -30 per cent, clearly the design is not centered, and a different nominal value for the component is indicated. Common sense indicates that all margins will not be symmetrical. *Example.* The plate supply voltage of a cathode follower can be lowered only until grid current loads the input circuit too much, but it may be possible to raise it until arc-over occurs.

**Resistors and Tubes.** The effect of change of characteristics in resistors and in tubes may often be determined by varying the supply voltage for individual branches of the circuit and plotting this branch supply voltage against the marginal-checking voltage to determine the

area of satisfactory operation. In the case of resistive voltage dividers, a variation of the supply voltage for the divider may be converted to an equivalent change in the divider resistors.

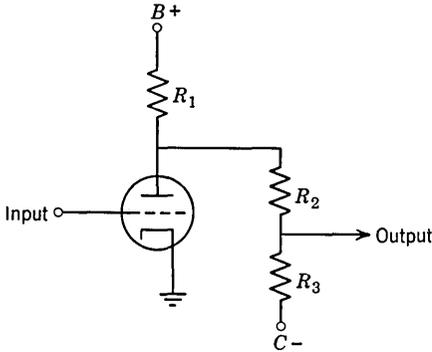


FIG. 2. Direct-coupled amplifier.

other parameters that would cause this same limiting change in output level. By employing this principle, one set of data can be used to determine the required tolerance of many components.

**The Pentode.** The effect of loss of emission in a pentode can be simulated by reducing the screen voltage—a drop in screen voltage being equivalent to a drop in available zero-bias plate current. A rise in screen voltage will increase the cutoff voltage required, and it is useful for checking the adequacy of the bias provided.

**The Triode.** Aging in triodes is much more difficult to simulate. The principle used is to reduce the plate supply voltage, so that the tube attempts to pass the same current at less plate voltage. Consider the circuit in Fig. 3, where the input will be either at ground or cutoff, and the output voltage swing will be dependent on the current the tube draws at zero bias.

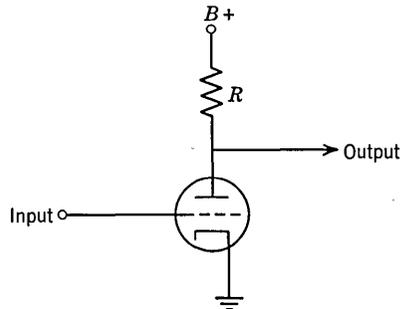


FIG. 3. Plate-loaded amplifier.

The load line is shown in Fig. 4 along with the zero-bias lines for both a new and an end-of-life tube. A new tube would operate at point *A* and an end-of-life tube at point *B*. An end-of-life tube is not generally available for checking the circuit's operation at this point, so the supply voltage is reduced from  $E_1$  to a voltage  $E_2$  so that with the new tube the

**EXAMPLE.** Consider the direct-coupled amplifier in Fig. 2. The output voltage is a function of the input,  $B+$ ,  $C-$ ,  $R_1$ ,  $R_2$ , and  $R_3$ , so that a change of any one of these parameters will affect the output level. If the permissible excursion of  $C-$  is determined experimentally (other parameters held fixed), the resulting change in the output level may be calculated. From this change in output, it is possible to calculate the equivalent change in any of the

operating point shifts to point *C*. The output voltage swing corresponds to an end-of-life tube with  $B+$  and  $E_2$ , since the same plate current is switched into the load resistor for each case. This type of analysis may be used to evaluate the supply voltage variations observed in a circuit, in order to determine the adequacy of a design. If the magnitude of the supply variation alone were considered in comparing two com-

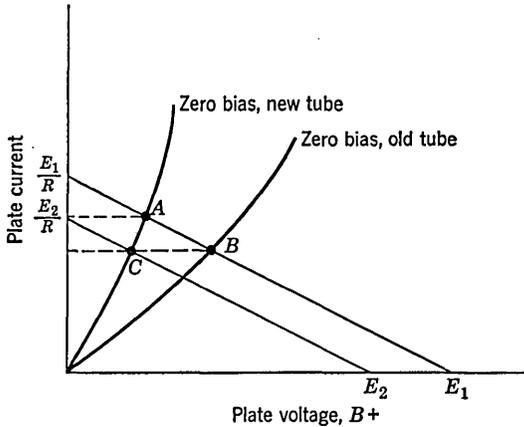


FIG. 4. Load line chart of circuit of Fig. 3.

peting circuits, the circuit chosen might be the one that is the least tolerant of an old tube. An analysis similar to that given above will enable the engineer to make the better choice.

It is often possible to substitute for an end-of-life tube a different type that will normally have characteristics close to those that would be expected from an end-of-life tube of the type to be used in the circuit. For example, a 6072 or a 12AY7 has approximately the same  $\mu$  as a 5965, but about one-third the zero-bias current. If a 6072 works in circuits designed for 5965's, one knows then that the circuit will tolerate an end-of-life 5965 tube.

**The Semiconductor Diode.** In general, diodes may deteriorate in the direction of lower back resistance or higher forward resistance. A diode with low back resistance may be simulated by a shunt resistor across a good diode. High forward resistance may be simulated by series resistors.

**L and C Components.** The variations in inductors and capacitors generally must be determined by replacing them with different values or with series-parallel combinations. Every variation that might affect the circuit should be tried.

### Marginal Checking: Circuits

The application of marginal checking to circuit design will be described in terms of an example, a high-speed flip-flop designed for use in a large, high-speed digital computer. The complete treatment of this circuit includes performance and component specifications, and an evaluation of the performance, including plots of pertinent margins of operation.

**High-Speed Flip-Flop.** The circuit for the high-speed flip-flop is shown in Fig. 5. Its complete specifications follow.

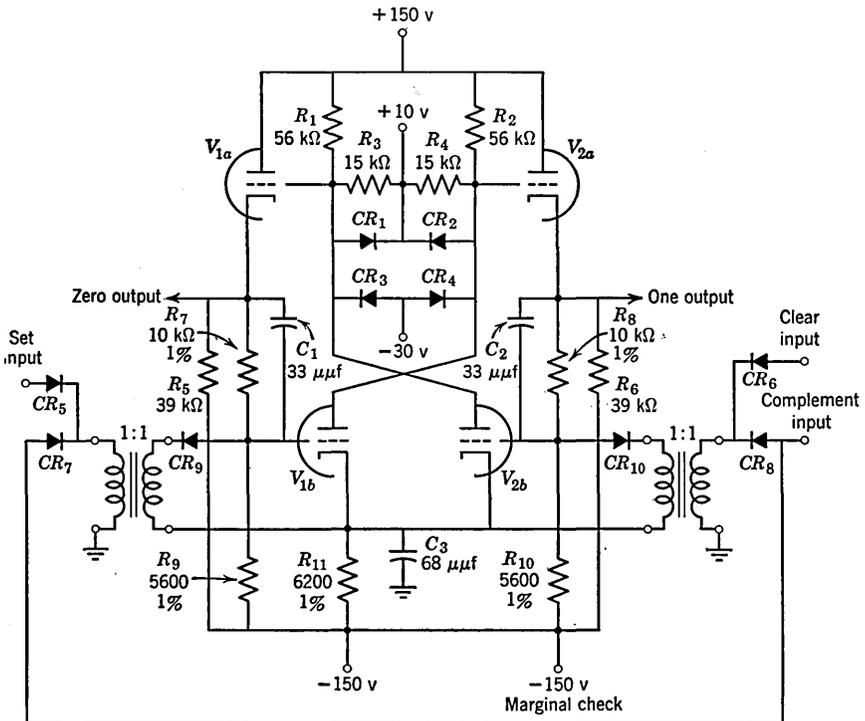


FIG. 5. High-speed flip-flop. Unless otherwise specified: (1) resistors are in ohms,  $1w \pm 5\%$ ; (2)  $V_1$  and  $V_2$  are 5965 tubes.

### Performance Specifications.

*Input.* 0.08 to 0.12- $\mu$ sec half-sine wave positive pulses, 20 to 40 volts amplitude, 0 to 2 megacycles/sec pulse repetition frequency (prf), set, clear, or complement.

*Output.* Upper level, +10 to +12.5 volts; lower level, -27 to -30.5 volts. Total transition time, less than 0.5  $\mu$ sec (measured from the beginning of the pulse until the new level is reached), and a delay

suitable for counting. (The same pulse is used to complement the flip-flop and to sense a gate tube connected to its output.)

*Load That Can Be Driven.* 90  $\mu\mu\text{f}$  maximum per output, with a total load of no more than 100  $\mu\mu\text{f}$ .

*Marginal Checking.* It must be possible to marginal-check the circuit from a remote point so that drift in the components can be detected before they cause failure of the circuit during system use.

### **Component Specifications.**

*Resistors.* *Composition resistors* used were nominal 1- and 2-watt size, had no more than 50% of the manufacturer's rated dissipation, an initial tolerance of  $\pm 5\%$ , and an end-of-life tolerance of  $\pm 15\%$ ; no more than 500 volts rms applied. Nominal half-watt composition resistors had no more than 25% of the manufacturer's rated dissipation, an initial tolerance of  $\pm 5\%$  and an end-of-life tolerance of  $\pm 15\%$ , and no more than 350 volts rms applied. Values were restricted to the 5% RETMA (now EIA) series.

*Precision film resistors* had no more than 50% of the manufacturer's rated dissipation, an initial tolerance of  $\pm 1\%$  and an end-of-life tolerance of  $\pm 5\%$ , and no more than 500 volts rms for the nominal 1-watt size or 750 volts rms for the nominal 2-watt size. Values were restricted to the 5% RETMA (now EIA) series.

*Capacitors.* Capacitors were ceramic-dielectric,  $\pm 5\%$  initial and  $\pm 15\%$  end-of-life tolerance, 50% of the manufacturer's rated voltage (which is 500), and available in the 10% RETMA (now EIA) series from 12  $\mu\mu\text{f}$  to 330  $\mu\mu\text{f}$ .

*Pulse Transformers.* A small hermetically sealed canned transformer, designed to pass the standard 0.1- $\mu\text{sec}$  pulses, was used.

*Germanium Diodes.* A special group of diodes was specified. The high points of these specifications follow.

*Type W diode* (intended for pulse mixing and clamping). With a low duty factor 0.1- $\mu\text{sec}$  half-sine 50-ma current pulse, the forward drop was not to exceed 3 volts; acceptance back resistance was to be at least 500  $\text{k}\Omega$  between  $-10$  and  $-50$ , 100  $\text{k}\Omega$  design value back resistance, except 50  $\text{k}\Omega$  design value for the first 0.5  $\mu\text{sec}$  after applying back voltage; for reverse recovery, after 5 ma forward for 1  $\mu\text{sec}$ , 40 volts reverse was to be applied, through a series resistance of 2  $\text{k}\Omega$  and the back current was to be less than 0.5 ma in 0.3  $\mu\text{sec}$ ; forward current was not to exceed 150 ma peak for 0.1  $\mu\text{sec}$  or 60 ma rms; reverse voltage was not to exceed 60 volts.

*Type Y diode* (a less expensive general purpose diode). At 1 volt forward voltage, the current was to be between 5 and 20 ma; back resistance specification was the same as for type W above, except

reverse current need decay to only 0.8 ma 0.3  $\mu\text{sec}$  after applying back voltage, and forward current was not to exceed 45 ma peak or 16 ma rms.

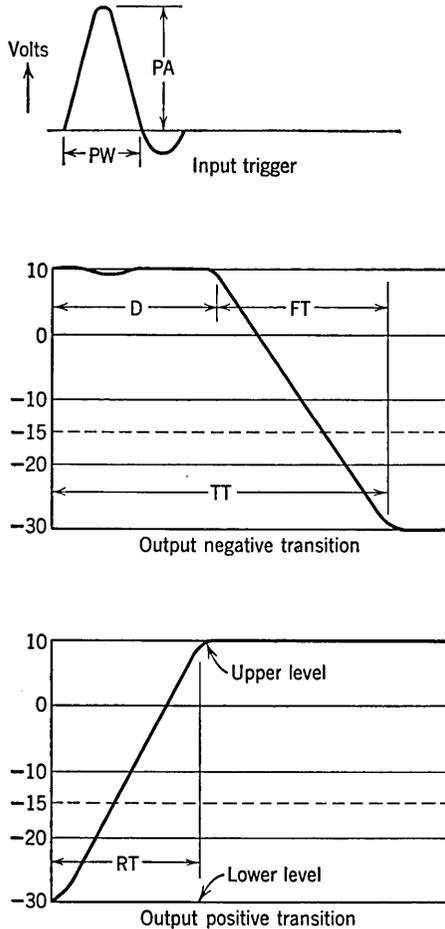


FIG. 6. Output waveforms. PA, pulse amplitude,  $20\text{ v} \leq \text{PA} \leq 40\text{ v}$ ; PW, pulse width,  $0.08\ \mu\text{sec} \leq \text{PW} \leq 0.12\ \mu\text{sec}$ ; D, circuit delay,  $\text{PW} \leq \text{D} \leq 0.20\ \mu\text{sec}$ ; TT, transition time from  $+10\text{ v}$  to  $-30\text{ v}$ ,  $\text{TT} \leq 0.5\ \mu\text{sec}$ ; FT, fall time,  $0.2\ \mu\text{sec} \leq \text{FT} \leq 0.3\ \mu\text{sec}$ ; RT, rise time,  $0.2\ \mu\text{sec} \leq \text{RT} \leq 0.5\ \mu\text{sec}$ .

*Tubes.* Only tubes that are acceptable as reliable types by manufacturers were specified. Triodes were preferred to pentodes since the triode has a simpler structure, and is less prone to intermittent shorts. Twin triodes were preferred over single triodes since fewer tube sockets are needed. Very high performance types were avoided on account of

their inherent close internal spacing and attendant high probability of intermittents.

*Power Supplies.* Only centralized power supplies were used; for this system, the voltages available were +250, +150, +90, +10, -15,

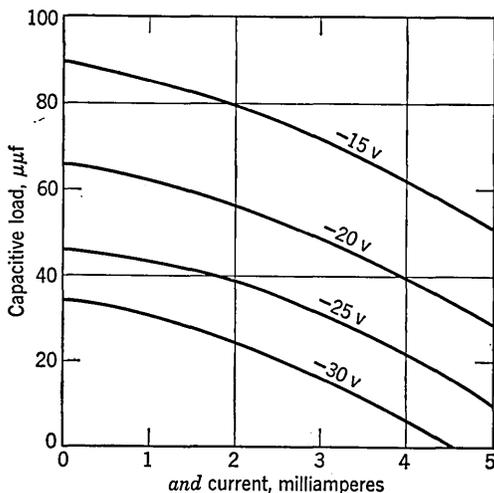


Fig. 7. Maximum load for less than 0.5  $\mu$ sec fall time.

-30, -150, and -300. All supplies had  $\pm 1\%$  regulation with  $\pm 1\%$  additional due to line drops, giving a total specification of  $\pm 2\%$ . The end-of-life figure used was  $\pm 5\%$ .

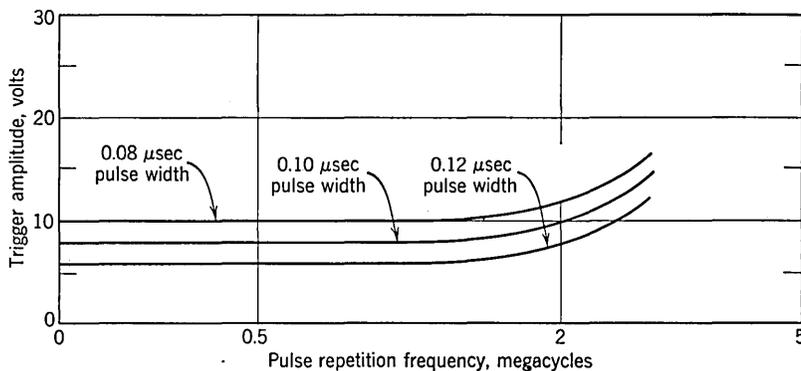


Fig. 8. Pulse repetition frequency response characteristics versus pulse width.

**Circuit Description.** The feature of the circuit of the high-speed flip-flop (Fig. 5) is low-performance triodes with cathode followers isolating the plates from both the external load and the capacitance of the

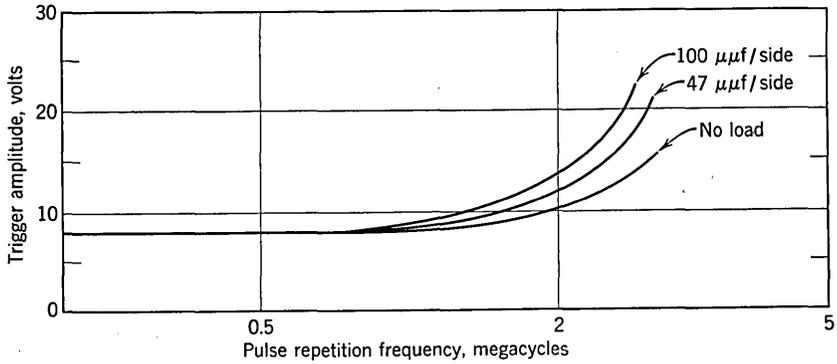


FIG. 9. Pulse repetition frequency response characteristics versus load.

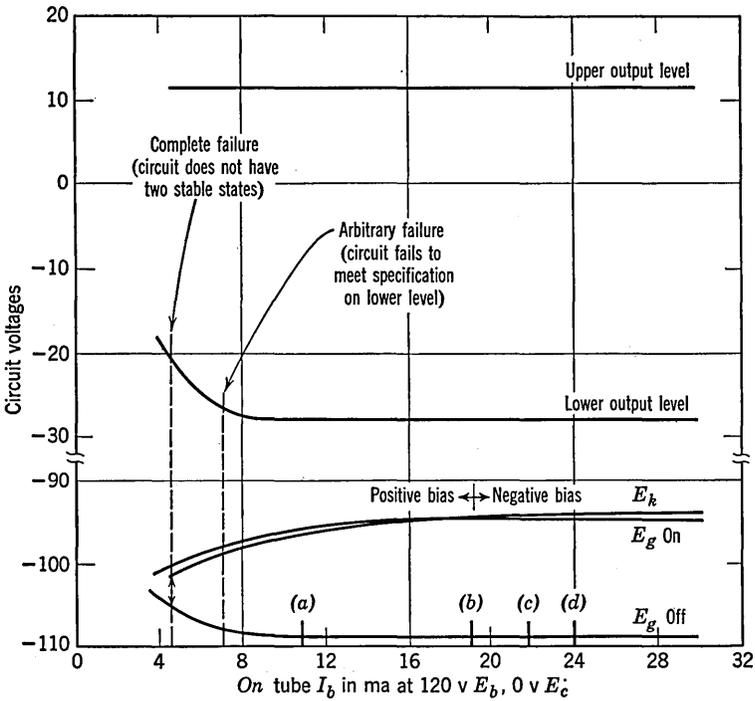


FIG. 10. Circuit voltages versus tube characteristics: (a) -40% average, (b) average tube, (c) Off tube, (d) +25% average.

opposite grid. The plate circuits are clamped through diodes to both +10 and -30, stabilizing both the output swing and the signal transmitted to the opposite grid to make the circuit less sensitive to variations in plate current of the triodes. The bias return of one grid divider is the marginal-checking point; this is moved above and below its nominal voltage of -150 to determine the circuit's margin. This simulates drift in the four voltage divider resistors directly ( $R_7$  through  $R_{10}$ ).

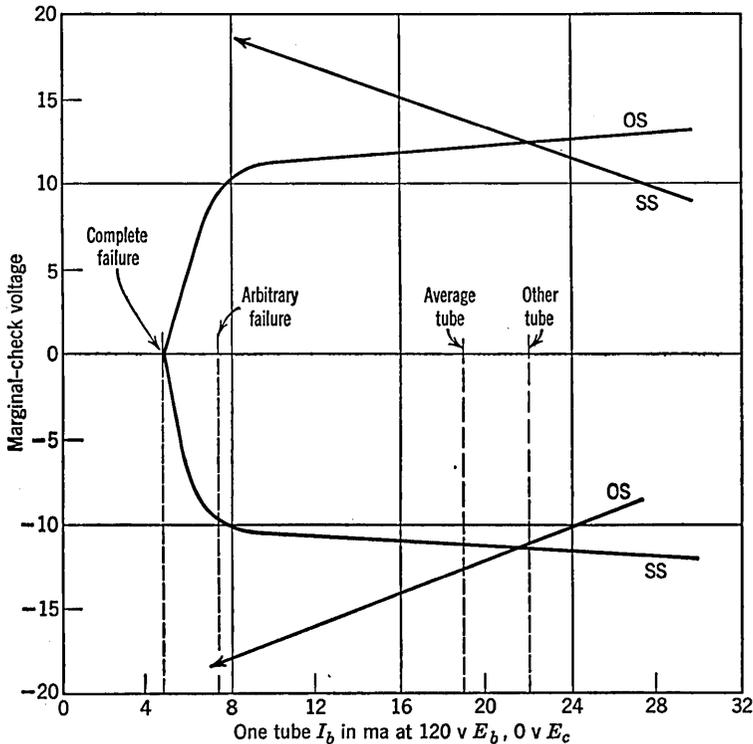


Fig. 11. Marginal-check voltages versus tube characteristics.

**Performance Data.** Figures 6 and 7 define the performance of the flip-flop for system timing analysis. Figure 6 shows output and input waveforms; the time taken for the output to reach -15 is of interest since that is the level required to cut off a gate tube (described later). Figure 7 shows the maximum amount of capacitance and/or current that the flip-flop can handle and still fall to the indicated voltage level within 0.5 microseconds. Rise time is inherently faster than fall time. *and* current is current drawn by a load connected to a positive voltage, measured when the flip-flop output is at its lower level.

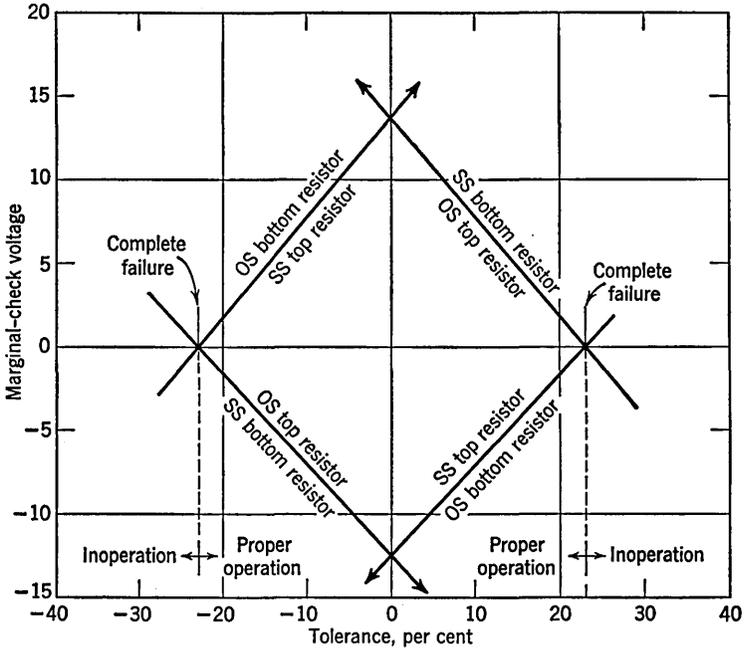


FIG. 12. Marginal-check voltages versus divider-resistor tolerance.

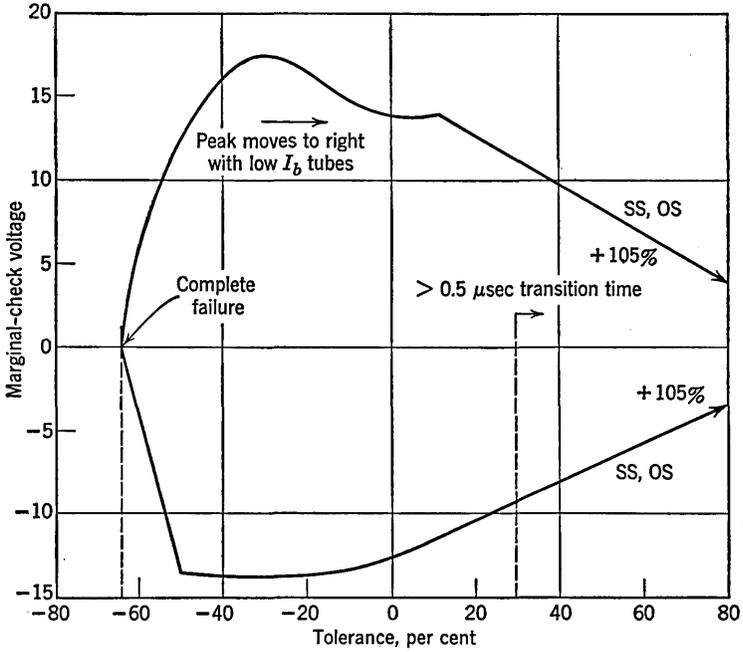


FIG. 13. Marginal-check voltages versus cathode-resistor tolerance. Tolerance with balanced tubes: +30%, -64%. Tolerance range:  $\approx$  90%.

Figure 8 shows the minimum complement trigger amplitude plotted against prf for various pulse widths, and Fig. 9 shows how the minimum complement trigger amplitude varies with prf for different capacitive loads.

### Reliability Data.

*Marginal Checking.* Figures 10 through 16 show reliability data, obtained by marginal checking, that indicate the component tolerances and safety margins of the circuit. In these diagrams, OS means that the curve represents circuit behavior when the component concerned was on the opposite side of the circuit from which the marginal-checking input was located; SS means the same side.

Figure 10 shows how critical voltages in the circuit vary as the tube ages. These data were taken by reducing the filament voltage to simulate the weak tube, a very touchy method. The tube was connected to a three-pole two-position switch, switched to a test position until the plate current stabilized, and then switched into the flip-flop circuit; the d-c voltages were then measured.

*Plate Current.* Figure 11 shows the circuit margins plotted against the test plate current of one tube while the other is held fixed. The tube was "aged" by filament variations as described above.

*Voltage Divider Resistors.* Figure 12 shows circuit margins plotted against the variations of the four voltage divider resistors,  $R_7$  through  $R_{10}$ . This linear relationship is expected, and could have been obtained analytically.

*Cathode Resistors.* Figure 13 shows margins plotted against tolerance of the cathode resistor,  $R_{11}$ . A note on the curve explains how it shifts with low-current tubes, so that the apparent unbalance in tolerance is not objectionable.

*Plate Load Resistors.* Figure 14 shows how the margins vary with the plate load resistors,  $R_1$  and  $R_2$ .

*Low Back Resistance.* Figure 15 shows the effect on margins of low back resistance in the  $-30$ -volt clamp diodes,  $CR_3$  and  $CR_4$ .

*Trigger Amplitude.* Figure 16 shows margins as a function of trigger amplitude. This plot is the most useful one for optimizing the final circuit, especially when the data are taken for both low and high (2-Mc) prf's, maximum and minimum loads, unbalanced tubes, etc.

Other data were taken to show margins versus the back resistance of the trigger diodes ( $CR_9$  and  $CR_{10}$ ), cathode-follower plate voltage, memory capacitors ( $C_1$  and  $C_2$ ), cathode bypass capacitor ( $C_3$ ), forward drop of input diodes ( $CR_5$  through  $CR_8$ ), etc.

**Summary.** The detailed analysis illustrated in this example gives the circuit designer quantitative figures for component tolerances and

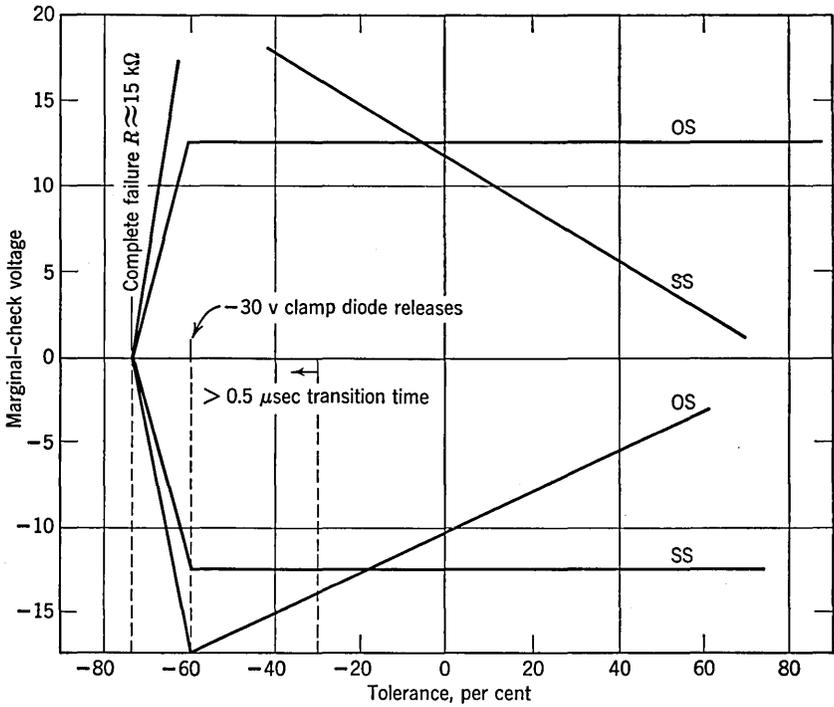


FIG. 14. Marginal-check voltages versus +150-volt plate-resistor tolerance.

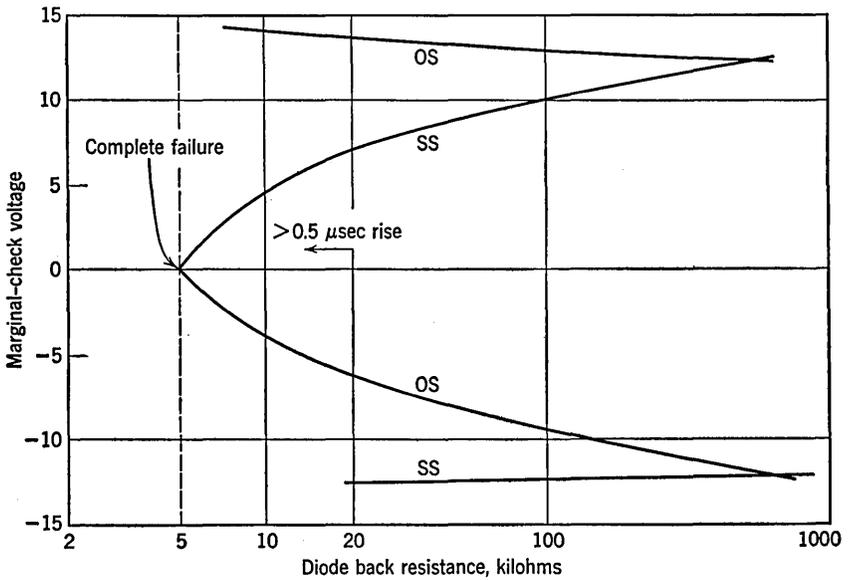


FIG. 15. Marginal-check voltage versus back resistance of -30-volt clamp diode.

safety margins in his circuit from which he can extrapolate the effect of simultaneous change in several components.

A large electronic computing system is made up of a number of basic circuits of which the high-speed flip-flop is one. Each of these basic

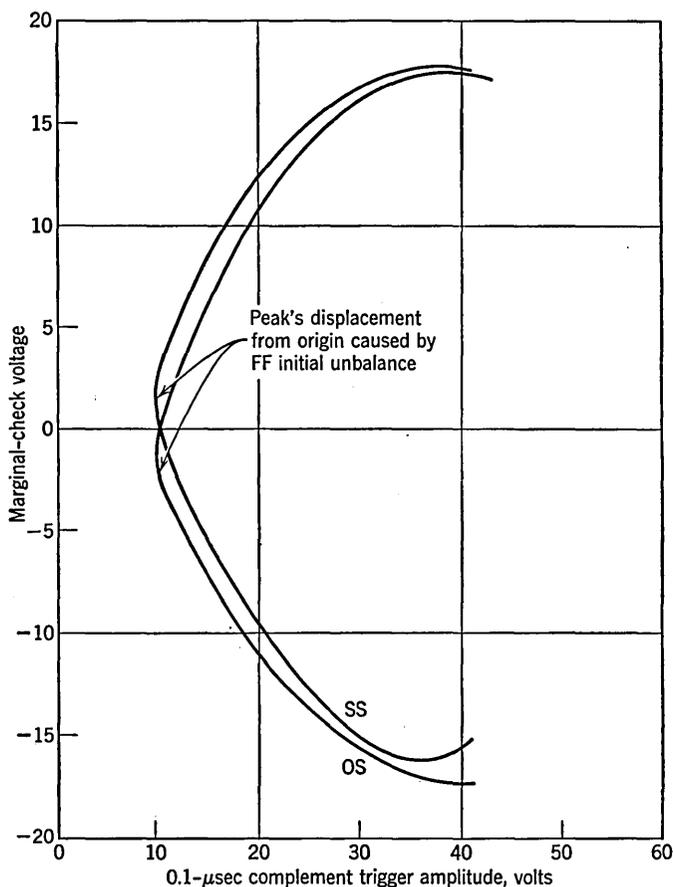


FIG. 16. Marginal-check voltage versus input-trigger amplitude.

circuits must be designed with the same thoroughness with respect to its performance, component tolerances, and safety margins.

#### 4. RELIABLE COMPUTER CIRCUITS

**General Considerations.** A group of typical and compatible computer circuits is presented to illustrate applications of the philosophy of reliable design. Each of the tube circuits was designed with the same care and study as the high-speed flip-flop example of the previous section.

The design of the transistor circuits has been slightly less rigorous because of the newness of the transistor.

The group of circuits includes pulse sources, pulse amplifiers, logical gate circuitry, flip-flops, cathode followers, indicators, and blocking oscillators. These basic units can be interconnected to perform most logic functions. Together with a storage system, this circuitry is adequate for reliable high-speed computer application.

In a large system, signal levels must, whenever possible, conform with a standard to make possible the interconnection of circuits, such as those listed below, in building-block fashion.

The use of a set of centralized d-c power supplies offers many advantages over the use of several small supplies. With only one set of supplies to build, it is advantageous to devote a sizable effort to them. Greater potential reliability accrues from the use of fewer components. Trouble detection equipment (such as low-voltage and overvoltage detectors), control circuitry, and emergency supply switch gear is far less extensive than the composite of analogous equipment used for many smaller supplies.

### Vacuum Tube and Associated Semiconductor Diode Circuits

The circuits to be described are designed to operate in a system using positive, nominally half-sinusoid (between triangular and square) pulses between 0.08 and 0.12  $\mu\text{sec}$  duration, and between 20 and 40 volts in amplitude, as standard control pulses. The prf can be between 0 and 2 Mc. Figure 17 shows such a standard 0.1- $\mu\text{sec}$  pulse. Circuits such as flip-

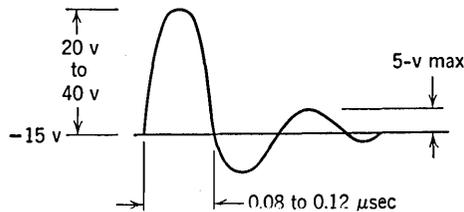


FIG. 17. Standard pulse.

flops generate outputs at +10 volts or -30 volts d-c levels, and are designed to be triggered by standard 0.1- $\mu\text{sec}$  pulses. Gate tube circuits used as *and* gates are designed for standard 0.1- $\mu\text{sec}$  pulse inputs to control grids and for levels to the suppressor grids of more positive than +10 volts (for selection) or more negative than -15 volts (for non-selection). Since the flip-flops generate +10 volts and -30 volts, and the gate tubes require only +10 volts and -15 volts, the signal level from the flip-flop may be allowed to climb and be attenuated through

cathode followers and diode logic on its way to a gate tube, without restandardizing the level.

Whenever possible, gate tubes and pulse amplifiers are designed to have less than unity gain with pulse inputs of less than 5 volts; greater than unity gain but less than 40 volts output for inputs between 20 and 25 volts; and, for inputs of from 25 to 40 volts, to have outputs of 25 to 40 volts (see Fig. 18). This transfer characteristic creates a tendency

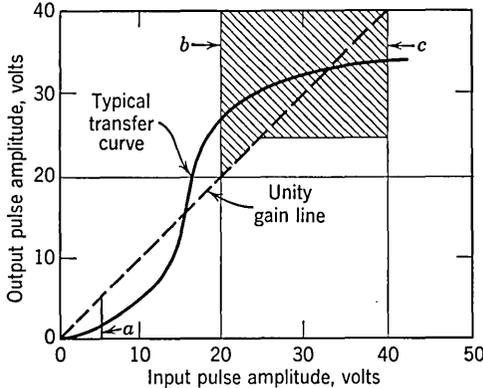


FIG. 18. Transfer characteristics. Specifications require that the transfer curve intersect vertical lines *a*, *b*, and *c*. Shading indicates normal operating region.

toward standardization of pulse amplitudes and alleviates the need for restandardization of pulses traveling through chains of gate tubes and pulse amplifiers. Marginal-checking facilities are included in each circuit to aid in detection of failing or deteriorating components.

The circuits described here have been designed to operate from centralized power supplies with outputs of  $-300$ ,  $-150$ ,  $-30$ ,  $+10$ ,  $+90$ ,  $+150$ , and  $+250$  volts. Particular attention has been paid to decoupling supply voltages to prevent interaction between circuits.

**Pulse Source.** The circuit of Fig. 19 is a typical clock pulse (standard 0.1-microsecond) generator. This circuit is designed to supply 0.1-microsecond pulses into a 93-ohm load at a 2-megacycle rate. The 2-Mc sine wave output from the first stage is clipped in stage 2 and amplified (now as 2-megacycle pulses), and re clipped and standardized in stage 3. The ability of stage 3, and circuits like it, to standardize pulses to 0.1-microsecond width is described in succeeding paragraphs.

Marginal checking of each stage is accomplished by decreasing screen grid voltage and sensing the output for failure.

**Pulse Amplifiers.** In many computer applications, the relatively large loads (long lines or many stages) that must be driven, require the use of pulse amplifiers, such as those shown in Figs. 20 and 23.

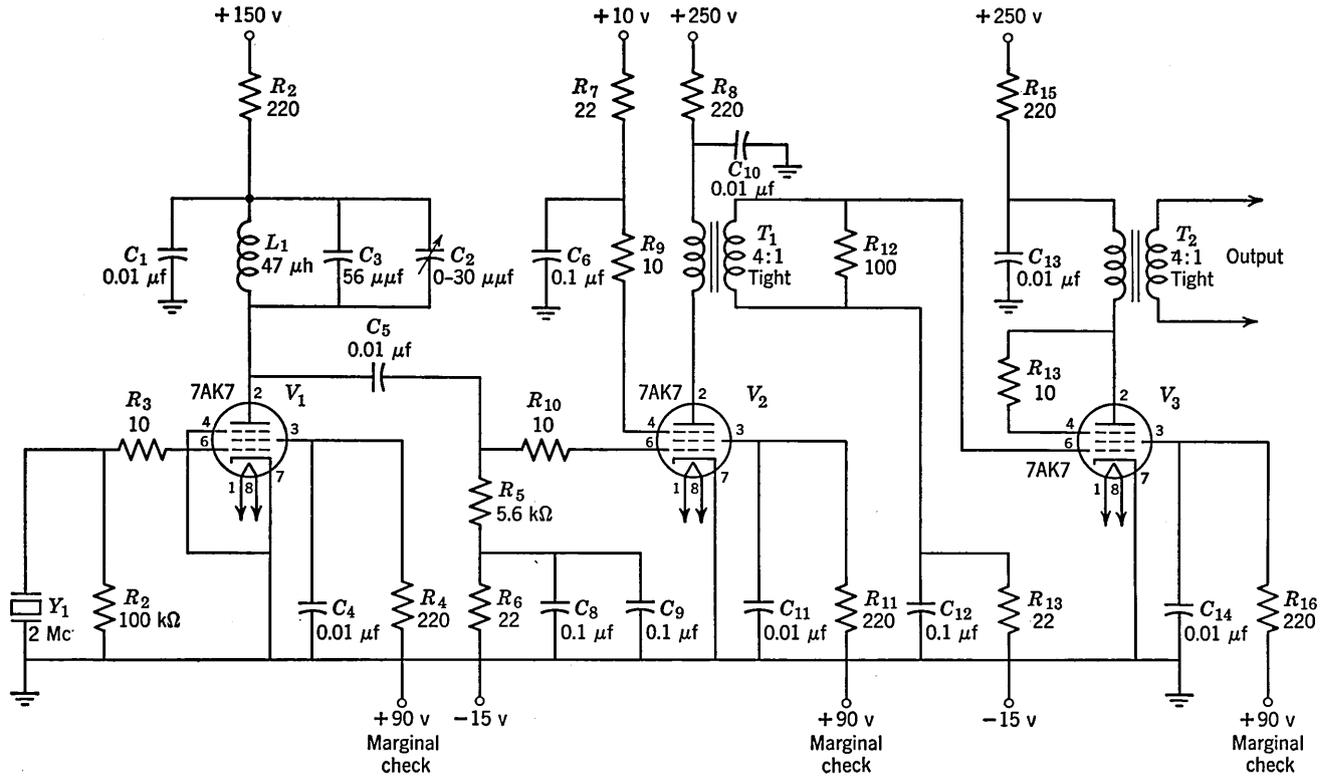


FIG. 19. Clock pulse generator.

The general purpose pulse amplifier of Fig. 20 has an output relatively independent of load and input voltage when load resistance is greater than 91 ohms and input greater than 20 volts. Figure 21 shows the transfer characteristics of this circuit for various loads. Partial amplitude "standardization" of output pulses is accomplished by "plate bot-

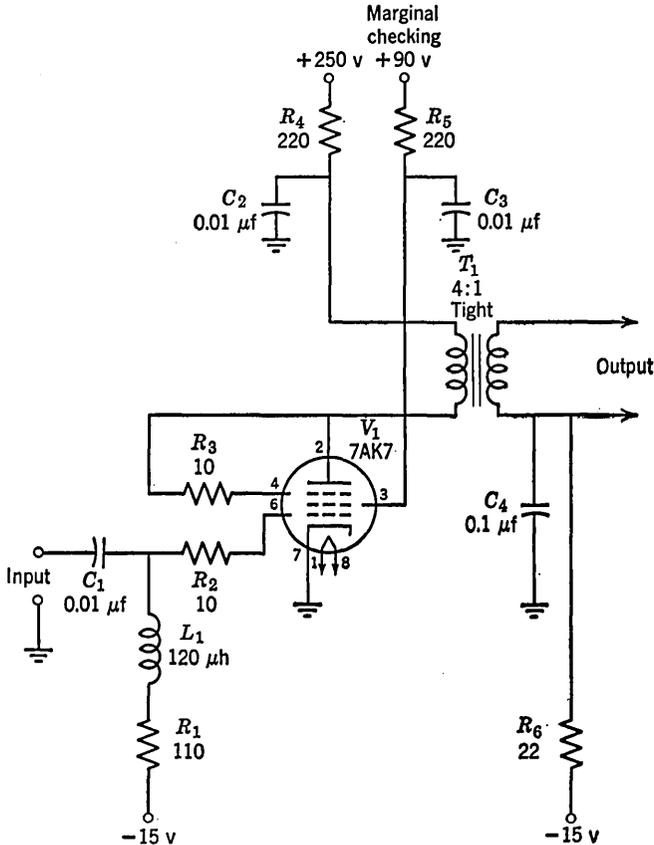


Fig. 20. General purpose pulse amplifier.

toming"; width standardization is accomplished by the characteristics of the pulse transformer. Pulse width standardization by this type of circuit is illustrated by Fig. 22, which shows the relationship between input pulse (width and amplitude) and output pulse width. It should be noted from the transfer characteristics (Fig. 21), that this circuit performs well for a load resistance of 91 ohms and above.

The input capacitor  $C_1$  is large enough to cause negligible loss of the input signal in spite of the grid current.  $L_1$  presents a high shunt im-

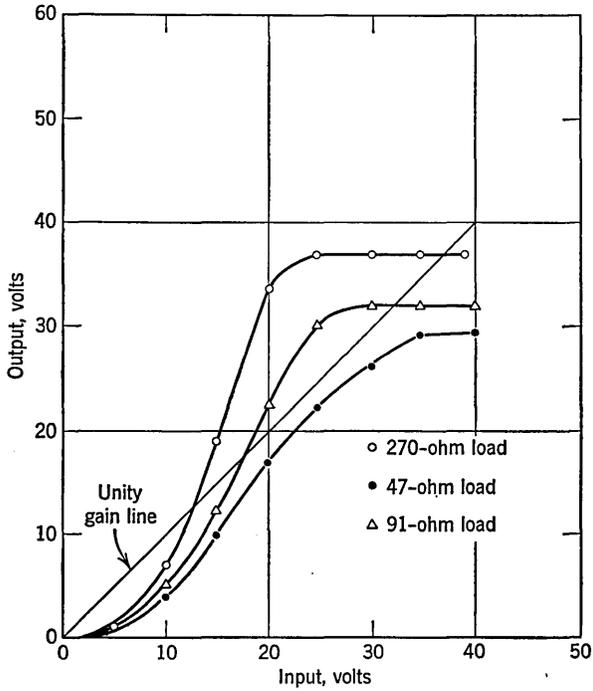


FIG. 21. Transfer characteristics, general purpose pulse amplifier.

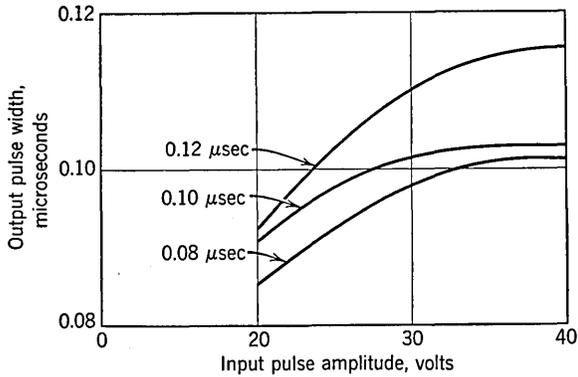


FIG. 22. Relationship between input and output pulse widths for typical pulse amplifier.

pedance to the input source and low impedance to d-c, and  $R_1$  critically damps the circuit. Since the pulses are generated by a transformer, the area of the pulse equals the area of the overshoot, and therefore they may be passed through such an a-c coupling circuit with no shift in relative baseline. The 4/1 pulse transformer matches the high-impedance plate circuit to a 91-ohm load. At this impedance level, pulses can be efficiently carried through coaxial cables.

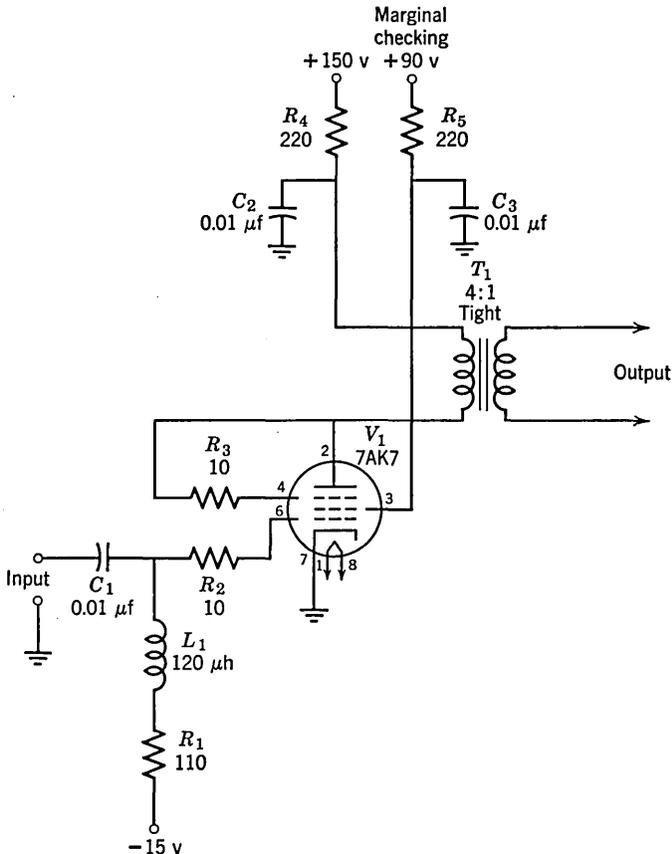


FIG. 23. High-power pulse amplifier.

For applications in which greater pulse amplitudes are desired (such as driving long lines) or in which larger loads must be driven, the high-power pulse amplifier of Fig. 23 is used. None of the transfer curves (Fig. 24) of this amplifier satisfies the standard transfer characteristic illustrated in Fig. 18. However, this high-power pulse amplifier ordi-

narily drives nonlinear loads involving large numbers of gate tubes or pulse amplifiers which enable the circuit to meet the standard transfer characteristic requirement easily.

**Vacuum Tube Gate.** The circuit for a gate tube is shown in Fig. 25. Inputs to this circuit are:  $G_1$ , 0.1-microsecond pulses, 20 to 40 volts amplitude;  $G_3$ , +10 volts for selection, -15 volts for nonselection. This

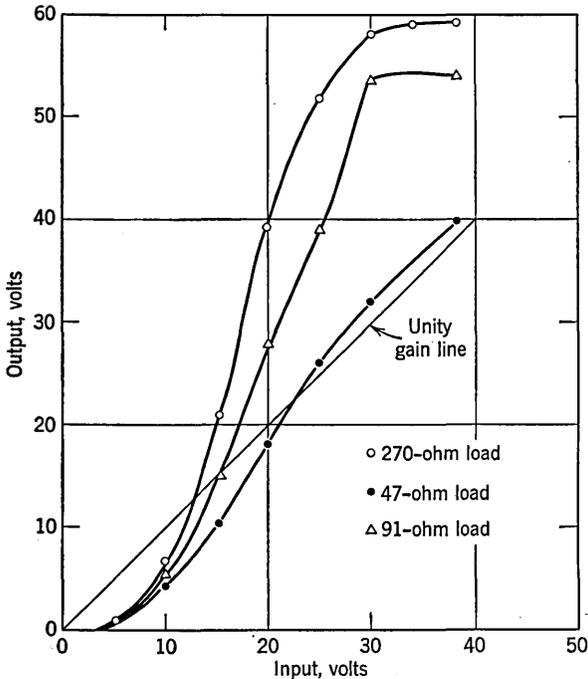


FIG. 24. Transfer characteristics, high-power pulse amplifier.

gate tube circuit, like the pulse amplifier circuits mentioned above, is capable of partially standardizing pulses. Standardization of pulse amplitude is not so good as in the general purpose amplifier, as can be seen from the transfer characteristics (Fig. 26). Selection in the gate tube is accomplished by applying a voltage greater than +10 at the suppressor grid input. Nonselection, that is, prevention of pulses applied to control grid from "getting through" to the plate circuit, is accomplished by applying a voltage more negative than -15 volts to the suppressor grid. By using +10 as the selection voltage, enough power can be delivered from the gate tube circuit to drive four flip-flops, or four gate tubes, or any combinations of these, without intermediate buffering. Under applications of light load to the gate tubes, the output of gates should

be resistance-loaded to give pulses of standard amplitudes. Marginal checking of the circuit is accomplished by varying the screen voltage: a drop in screen voltage makes the tube look older, and a rise in screen voltage detects dangerously high noise pulses at its input.

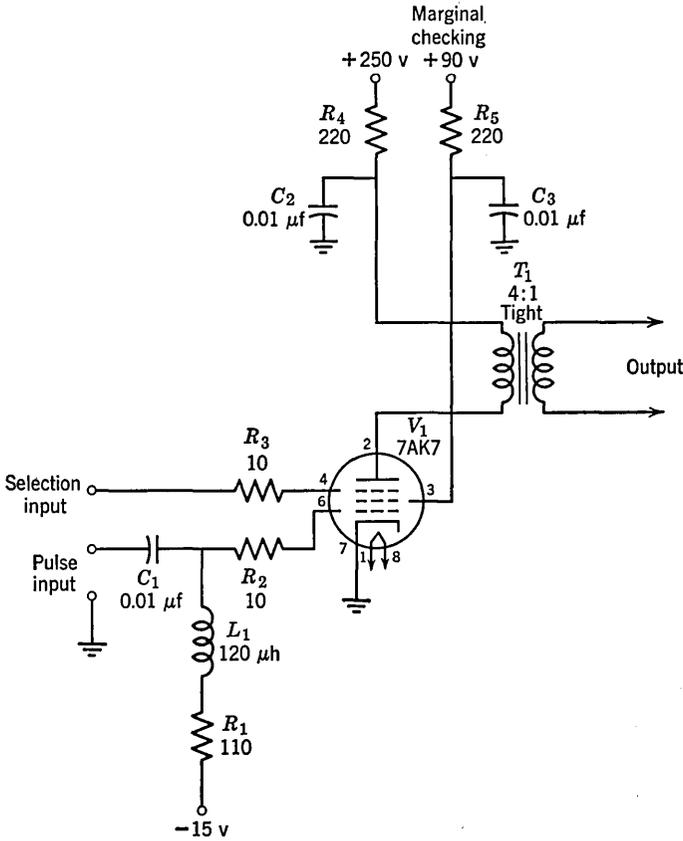


FIG. 25. Gate tube circuit.

**Diode Capacitor Gate.** A passive element pulse gating circuit is shown in Fig. 27a. Whereas the setup time of a vacuum tube gate is determined only by the transition time of its suppressor grid input, the diode capacitor circuit setup time (assuming a step function selection input) is determined by the charging time of the capacitor,  $C_1$ . If there is to be no feed through of pulses when the selection input is at  $-30$  volts, care must be exercised to keep input pulses always below 40 volts amplitude. This can be achieved by always driving sets of diode capacitor gates with the general purpose pulse amplifier mentioned earlier. Since no pulse

regeneration or partial standardization takes place within this type of gating circuit, long chains of these gates are not practical except when they are broken up by pulse amplifiers. In Fig. 27,  $L_1$  is chosen to give a high impedance to 0.1-microsecond pulses to isolate the pulse source from shunt capacitance of a possible long lead between  $R_1$  and  $L_1$ .  $R_1$  is large enough to isolate the source for the selection input (usually a

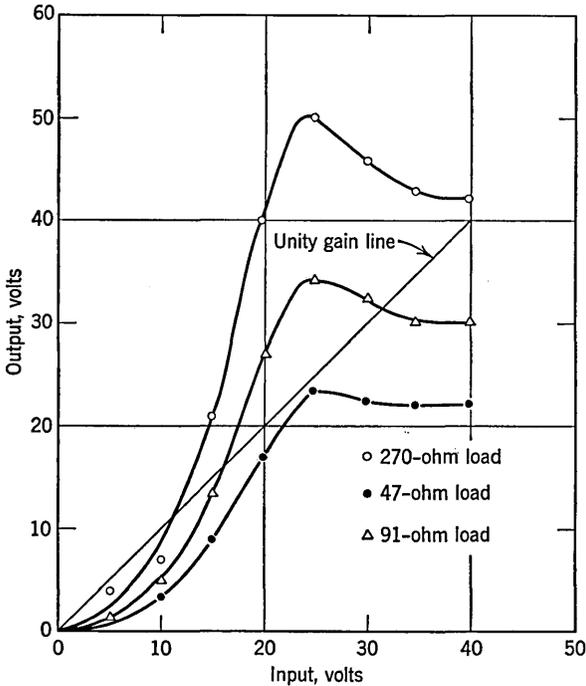


FIG. 26. Transfer characteristics, gate tube.

flip-flop) from the same shunt capacitance, as well as from  $C_1$ . At the same time,  $R_1$  must be small enough to prevent loss of pulse amplitude, but small enough to allow reasonable values for  $R_1$ .

An alternate termination for the gate is shown in Fig. 27b.  $L_2$ ,  $R_2$ , and the stray capacitance comprise a damped  $RLC$  circuit that reshapes the pulse which is then a-c coupled to the grid, where a clamp diode is used to establish a baseline.

**Low-Speed Flip-Flop.** In applications where compactness is desirable and high speed is unnecessary, a flip-flop such as that shown in Fig. 28 is useful. It is capable of being complemented at a 200-kilocycle rate, and has one tube (two cathodes) as compared to the 2-megacycle capability and the two tubes (four cathodes) of the high-speed flip-flop

described in Sect. 3. Rise and fall times of the outputs are on the order of 5 microseconds, depending on the load.

Basically, this flip-flop is an Eccles-Jordan circuit. A d-c coupling path is provided between each plate and the opposite grid. The 10,000-ohm resistor in the common cathode provides a large amount of d-c

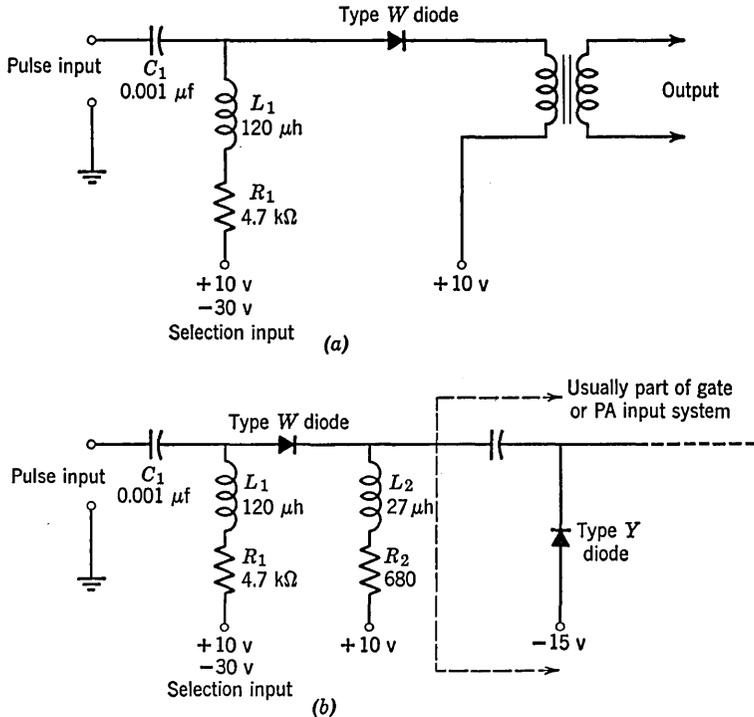


FIG. 27. Diode capacitor gate circuits.

degeneration and thus stabilizes the circuit against changes in tube characteristics. A 70-micromicrofarad cathode condenser stabilizes cathode voltage during transition of the flip-flop from one state to another.

The circuit is capable of driving up to 320 micromicrofarads of load capacitance on each output. Transformers are provided at each input to invert pulses for flip-flop triggering.

**Cathode Followers.** The limited load-driving capabilities of most flip-flops restrict loading to little more than a few gate tubes or their equivalent. Diode logic, if it is at all extensive, or long coaxial lines require the use of flip-flop output buffering. The cathode follower of Fig. 29 is adequate for moderate loads. (Little can be said about rise

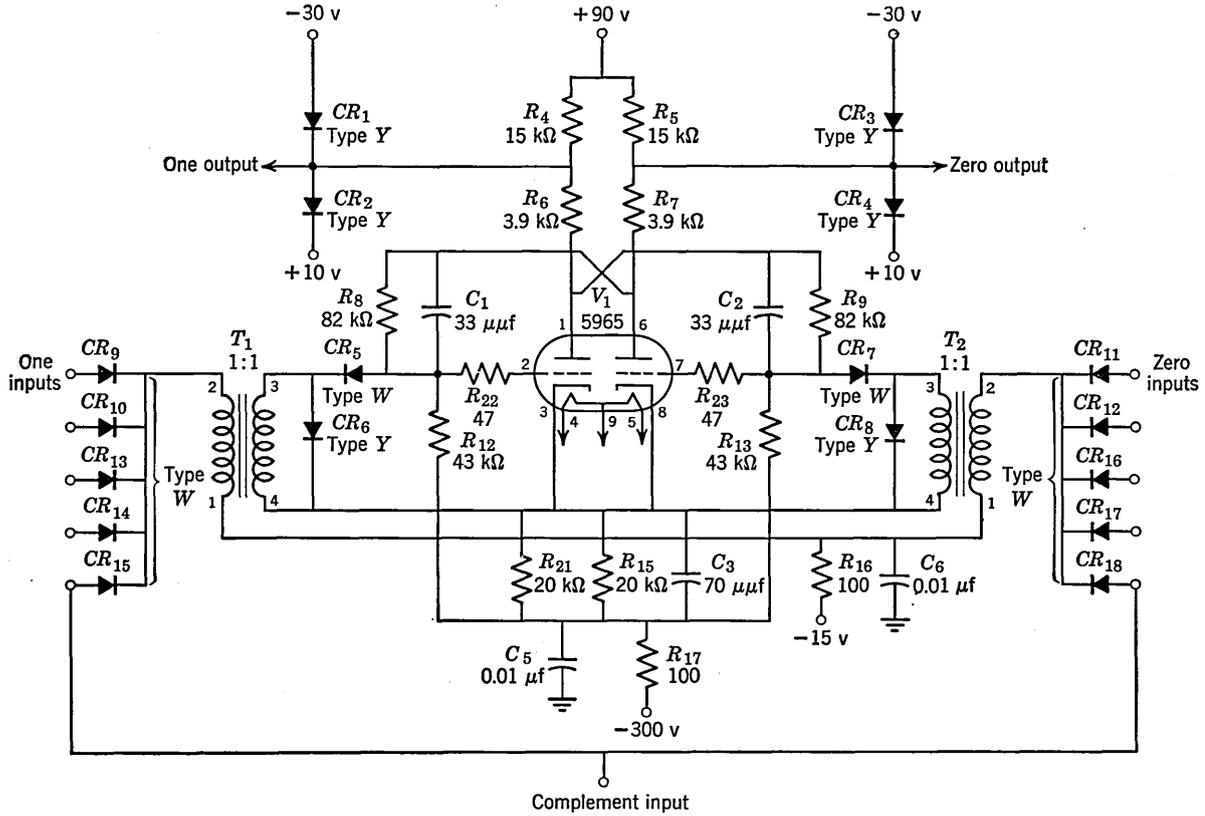


FIG. 28. Low-speed flip-flop.

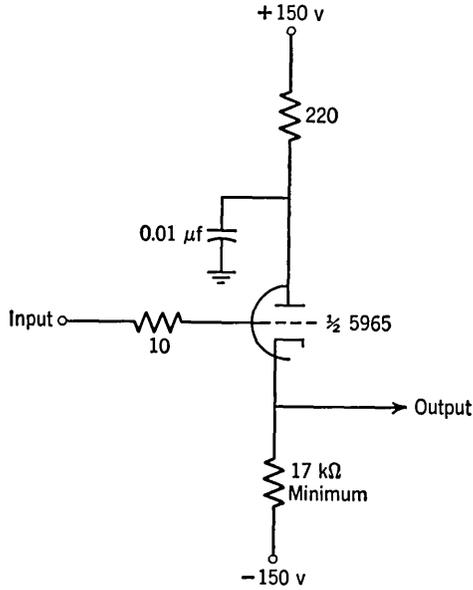


FIG. 29. Cathode follower.

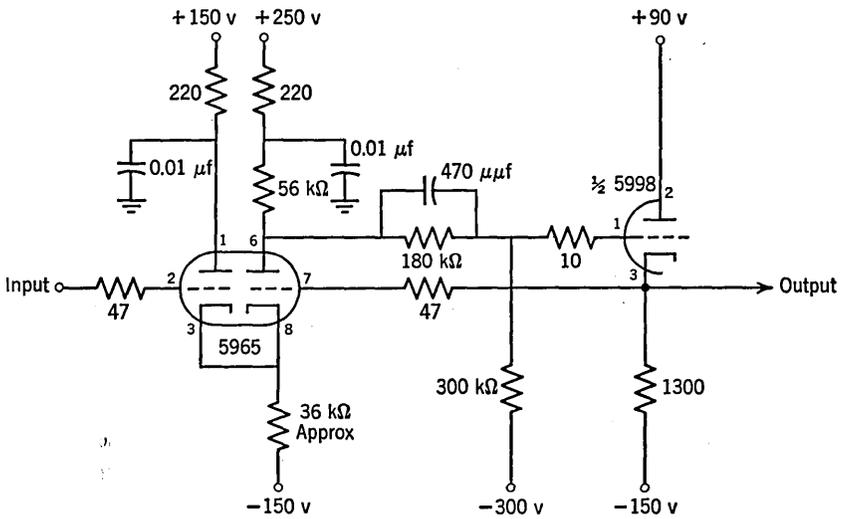


FIG. 30. Power cathode follower.

and fall times without specifying load impedance.) Variations of this circuit include paralleling of the circuit for driving heavier loads.

For applications involving heavy loads or where cathode follower bias buildup must be kept to a minimum, the feedback circuit of Fig. 30 is used. Comparison of input and output levels and amplification of the difference signal takes place in stage 1. Unity gain can be achieved by attenuating the feedback signal. With an input rise or fall time of  $0.3 \mu\text{sec}$ , output rise or fall time will be  $0.7$  microsecond for the nominal  $+10$ -volt to  $-30$ -volt signal. Power output capabilities of this circuit may be increased by paralleling other sections of 5998's to the present half-section of 5998.

**Flip-Flop Indicators.** Indicator neon lamps on flip-flops are useful in general systems operation and are particularly valuable in troubleshooting.

The indicator circuit of Fig. 31 is often used. Its operation depends on the difference between the outputs of the flip-flop to fire the appro-

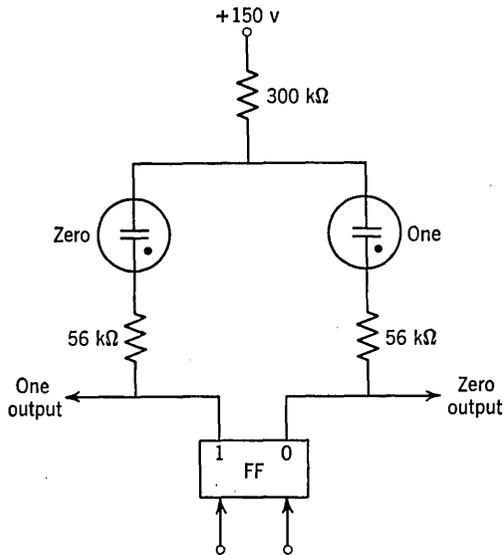


FIG. 31. Twin-neon flip-flop indicator.

ropriate neon lamp. The indicators may be separated from the flip-flop by hundreds of feet. In such a case,  $56$ -kilohm resistors in the flip-flop prevent loading of the flip-flop by the interconnecting cable to the neon lamps in a remote indicator unit.

A circuit providing one neon per flip-flop saves space in the indicator unit, since only one lamp is necessary for each flip-flop, and saves cable

since only one line is required from each flip-flop to the indicator unit (see Fig. 32). The pulse power supply is an integral part of the indicator

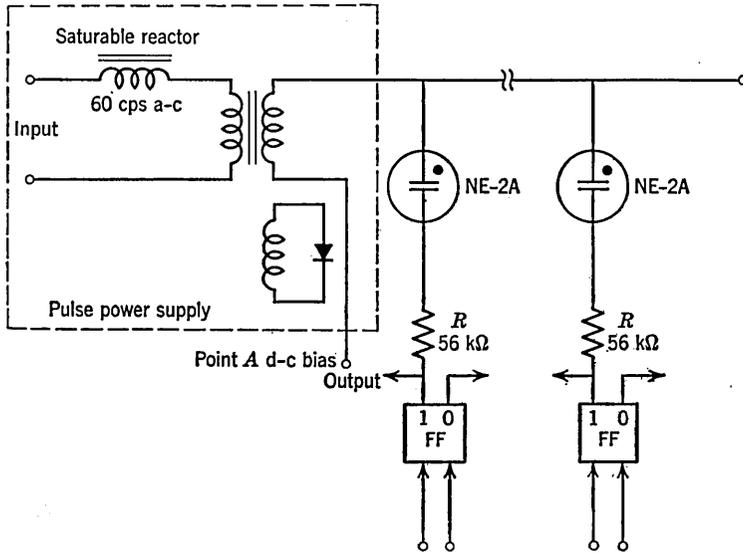


FIG. 32. Single-neon indicator circuit.

unit assembly. The pulse ignites all neons regardless of the states of the flip-flops. By proper adjustment of the d-c bias (point A in Fig. 32), the neons may be made to remain ignited or turn off according to the states of the flip-flops.

Measurements have shown the ranges of ignition and extinction voltages of a large sample of new and aged NE-2A indicator lamps to be as shown in Fig. 33. The single neon indicator system of Fig. 32 depends on the fact that a 15-volt change will bring any lamp from conduction to below extinction.

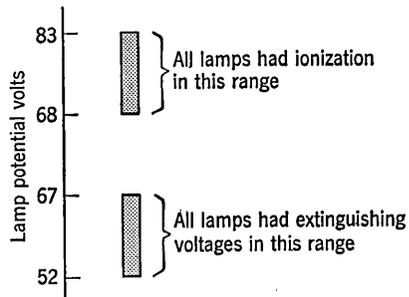


FIG. 33. Summary of measurements on NE-2A.

**Diode Logic.** Circuits of nonlinear passive elements such as diodes can perform many of the logical *or* and *and* functions in computing systems. The circuit of Fig. 34a is an *or* circuit. Its inputs are mixed so that its output is +10 volts if any one of the inputs is at +10. The value of  $R$  is chosen to give the circuit adequate fall time. (The smaller

the  $R$ , the closer the output will follow input fall time.) The circuit of Fig. 34b is a diode *and* circuit. Output level will be +10 volts (as opposed to -30 volts) if and only if all inputs are at +10. The choice of  $R$  here is governed by the desired rise time of the circuit output.

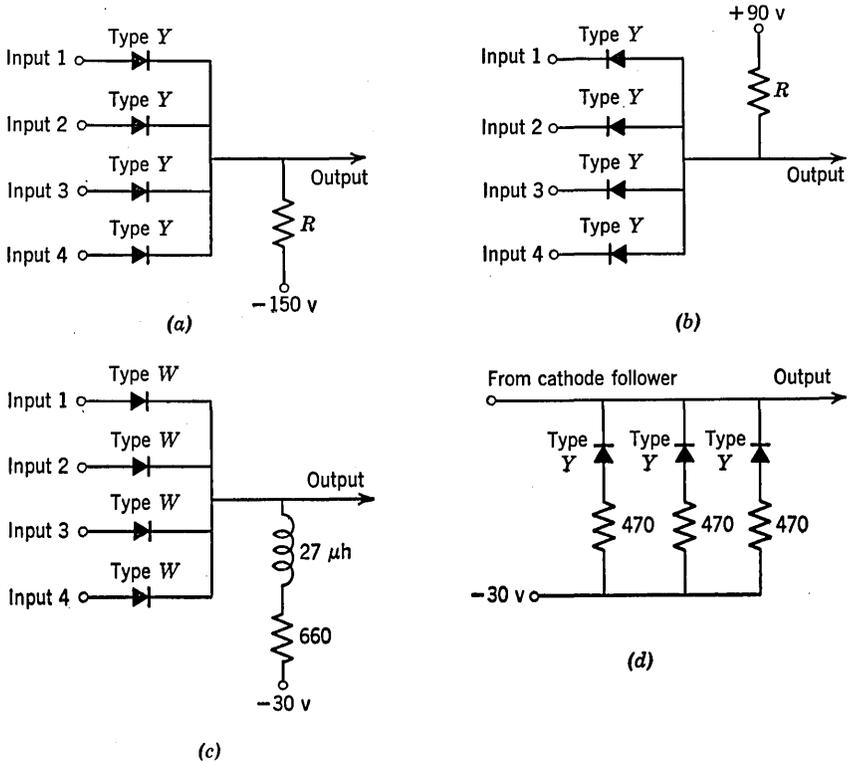


FIG. 34. Diode logic: (a) diode *or* circuit, (b) diode *and* circuit, (c) pulsed *or* circuit, (d) diode protection.

The circuit of Fig. 34c is an *or* circuit similar to that of Fig. 34a. This circuit is designed to mix 0.1-microsecond pulses. Although only four inputs are shown, any number of inputs (up to the point where the back resistance of the diodes loads the driving circuits) can be used. The shape of the leading edge of the output pulse is determined by that of the input, whereas the shape of the trailing edge is determined by  $R$ ,  $L$ , and the shunt (stray) capacitance.  $R$  and  $L$  are so chosen that the trailing edge of the pulse returns to the baseline, with no overshoot.

These diode logic circuits are usually driven by cathode followers such as that shown in Fig. 29. If the tube were pulled from its socket in this circuit, the output line would drop to -150 volts and apply a large back

voltage to diodes connected to the cathode follower. Failure of a tube, removal of a plug-in unit, or loss of certain supply voltages can similarly damage diodes such as those of the circuits of Figs. 34*a, b*. Figure 34*d* shows a protection circuit that prevents the output of the cathode follower from going very much below  $-30$  volts (normal back voltage for the diodes in the logic circuits). Enough diode-resistor combinations must be used to clamp the cathode follower output near  $-30$  volts without exceeding the current rating of the protection diodes. The 470-ohm resistors insure equitable current distribution through the diodes.

**Blocking Oscillators.** Blocking oscillators normally subject tubes to extremely hard usage, for example, high peak cathode current. The circuit of Fig. 35 has been designed to provide near-standard 0.1-micro-

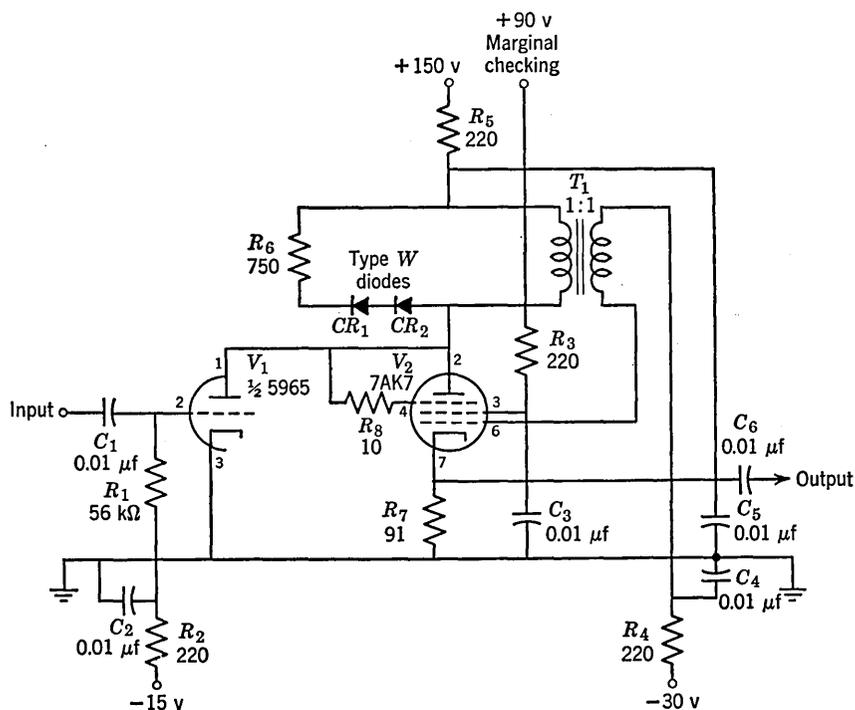


FIG. 35. Blocking oscillator.

second pulses from an input waveform of 20-volt rise at the minimum rate of 100 volts per microsecond. This circuit will operate between 0 and 100 kilocycles prf. Marginal checking is performed by varying the 90-volt screen supply voltage. Degeneration supplied by the load in parallel with  $R_7$  limits cathode current of the 7AK7 to a safe value.

### Transistor Circuits

The reliability of transistor circuits is affected by the same basic factors that determine reliability in tube circuits and some of the techniques of tube circuit design are adaptable to transistor circuits. (This parallel must not be carried too far—in no case can a transistor be “plugged” into a circuit in place of a tube and have the circuit operate equally well.) In general, transistor circuits operate at an impedance level an order of magnitude below that of similar tube circuits. This reduces the problem of accidental triggering from external noise but magnifies the problem of driving several parallel stages from a single transistor stage.

The group of transistor circuits to be discussed were designed around the high-speed Philco surface barrier transistor (SBT). These circuits are intended for use in a system with standardized supply voltages of +10, -3, and -10 volts, negative pulses of 3-volt amplitude and 80-millimicrosecond duration, and flip-flop levels of 0 and -3 volts.

**Problem of Transistor Circuit Design.** The problems of practical transistor circuit design involve in order of ease of solution (1) static operating margins, (2) dynamic operating margins, (3) temperature stability, (4) transient response.

1. *Static Operating Margins.* Design of transistor circuits with large static operating margins follows essentially the same procedures used with tubes. In general, however, the problem is easier with transistors since power consumption is so low compared to tube circuits that the power efficiency of the resulting network is less critical.

2. *Dynamic Operating Margins.* The achievement of acceptable dynamic margins is aided by the very sharp cutoff characteristics of transistors which may be utilized to fix limits on pulse amplitudes. However, the problem is complicated by the fact that the transistor is a low-input-impedance, high-output-impedance device and, therefore, susceptible to loading effects.

3. *Temperature Stability.* Both static and dynamic circuit margins are sensitive to temperature. The current gain and frequency response vary slowly with temperature, but the exponential variation of the leakage current is such that it approximately doubles for each 8°C temperature increment.

In general, a large dynamic margin may be designed into the circuitry by taking into consideration the effects of static operating point drift with temperature. Stabilizing devices, such as emitter degeneration resistors and heavy bleeder networks, should be used. The effects of temperature on the frequency response of the input and output networks,

as well as on the gain and frequency response of the transistor, must be considered.

4. *Transient Response.* The major adverse influence on circuit transient response results from temperature-induced transistor impedance variations which load the input network and alter the available "drive" power at the input to the transistor. The causes of temperature variations include both external ambients and internal self-heating caused by power dissipation at the transistor junctions and in the bulk material. Since present day units have fairly high thermal resistance (0.2–0.5°C per milliwatt), the circuit must be designed to guard against the effects of power dissipation.

**High-Speed Flip-Flop Group.** The basic flip-flop and its associated circuits are illustrated in Fig. 36. Transistors  $Q_3$  and  $Q_4$  are the flip-flop transistors;  $Q_1$  and  $Q_2$  are input pulse amplifiers for the two sides of the flip-flop;  $Q_5$  and  $Q_6$  are buffer inverter amplifiers;  $Q_7$ – $Q_9$  and  $Q_8$ – $Q_{10}$  are cascode power amplifiers which drive the output lines. This group of circuits is usually constructed on two etched cards, mounted together in one miniature pluggable unit, to minimize lead lengths between the individual circuits.

*Marginal Checking.* The circuits are marginal checked with the two +10-volt supply buses as indicated in Fig. 36. The +10-volt lines for the individual circuits may be separated if it is desired to marginal-check the circuits independently.

The *basic flip-flop* is a direct-coupled Eccles-Jordan circuit in which triggering is done with input pulse amplifiers in series with the emitters of transistors  $Q_3$  and  $Q_4$ , as shown in Fig. 36.

The *input-amplifiers*  $Q_1$  and  $Q_2$  are grounded emitter pulse amplifiers that are normally saturated or turned *on*. A positive pulse at the input terminal is coupled to the base of the input amplifier, turning this pulse amplifier *off*. The output is directly coupled from the collectors of the amplifiers to the emitters of the flip-flop transistors.

The *inverters* are grounded-emitter amplifiers that serve as buffers between the flip-flop transistors and the output cascode power amplifiers.

The *designs* of the flip-flop, input amplifier, and inverter are conventional and straightforward transistor circuit design, but the cascode power amplifier is unique. Transistor  $Q_{10}$  in the zero-side cascode amplifier is basically an emitter follower which drives the zero output line in the negative direction. Since these are saturating circuits with the inherent difficulties of "hole storage," the rise time of these circuits would deteriorate if these emitter followers were required to drive the output lines in the positive direction as well. Hole storage in the emitter follower and succeeding stages has the same effect on the output rise

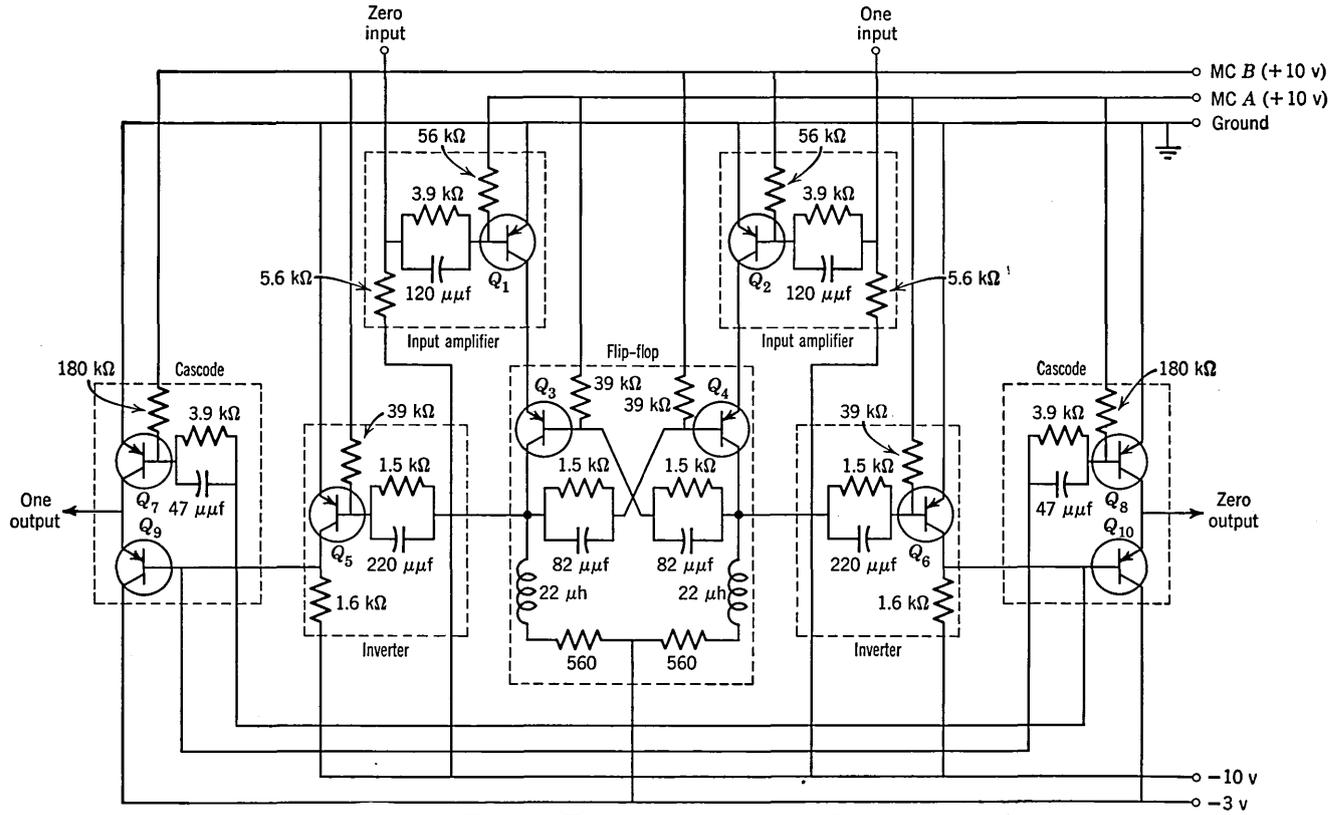


FIG. 36. High-speed transistor flip-flop.

time as line capacity. Transistor  $Q_8$  serves as a positive driver in this cascode circuit to make it possible to drive greater true or hole storage capacity. These cascode power amplifiers work so well that the rise time from 10 to 90 per cent on unloaded output lines is about 20 millimicroseconds, whereas the fall time is about 30 millimicroseconds unloaded.

*Input and output waveforms* of the flip-flop group are shown in Fig. 37. With 50-millimicrosecond trigger pulses as illustrated in the figure, the

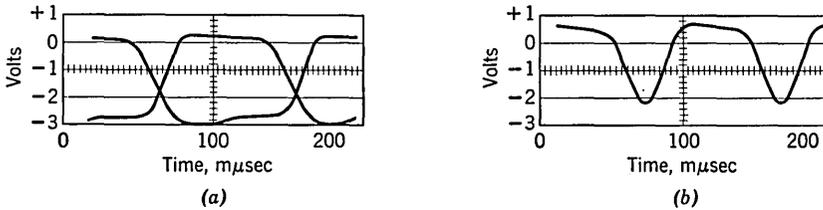


FIG. 37. High-speed flip-flop waveforms; (a) output, (b) trigger input.

maximum prf of the flip-flop group is approximately 10 megacycles. Maximum prf and minimum pulse widths are limited primarily by the lack of ability to generate and handle very narrow pulses. The circuit operates very reliably at the design goal of 5 megacycles prf.

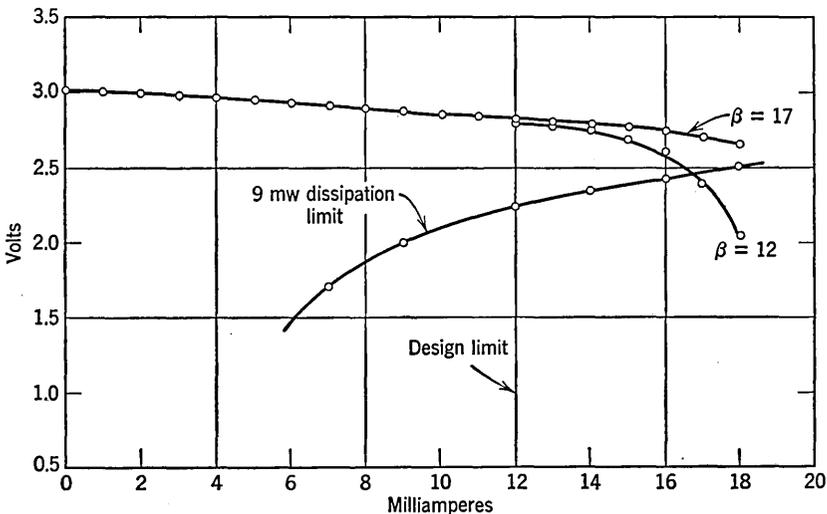


FIG. 38. Cascode d-c output characteristic.

The *d-c output characteristic* of the cascode circuit is shown in Fig. 38. The maximum output current is 16.8 milliamperes at the intersection of the characteristic curve for  $\beta$  near end of life and the 9-milliwatt tran-

istor dissipation limit. Design maximum for the output current is 12 milliamperes. The slope of the output characteristic or the internal equivalent resistance of the circuit is about 16 ohms.

The capacity-driving capability of the cascode is shown in Fig. 39. More than 400 micromicrofarads of output capacity can be driven at a

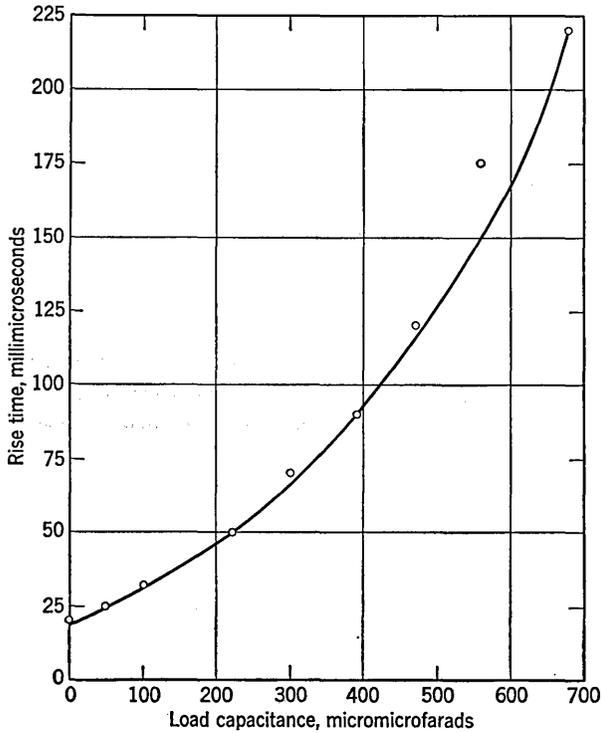


FIG. 39. Cascode output rise time versus load capacitance.

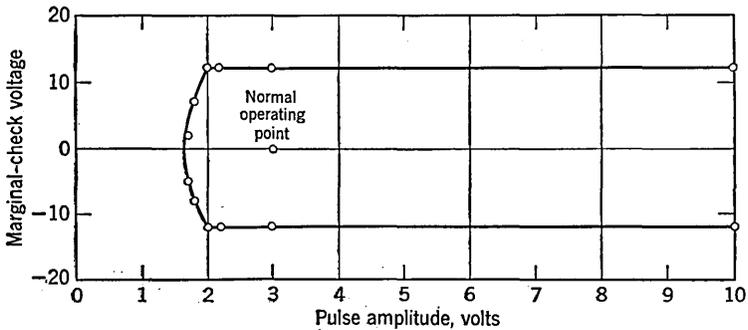


FIG. 40. High-speed flip-flop marginal-check voltage versus input pulse amplitude.

rise time of 0.1 microsecond. One cascode can drive up to eight inverter circuits with a normal amount of interconnecting wiring.

The *specifications* for these SBT transistors are summarized as follows:

$$I_{co} = 0.2 \text{ to } 3\mu\text{a}; \quad I_{eo} = 0.2 \text{ to } 3\mu\text{a}$$

$$\text{Initial } \alpha > 0.94 \quad \text{or} \quad \beta > 15.6;$$

or

$$\text{End-of-life } \alpha = 0.92 \quad \text{or} \quad \beta = 11.5.$$

$$V_{\text{max}} \text{ (punch through)} = 6 \text{ to } 20 \text{ volts}$$

$$V_{CE} \text{ (saturation)} \leq 0.1 \text{ volt at } I_{\text{base}} = 2.5 \text{ ma } I_c = 8 \text{ ma}$$

$$\tau = 30 \text{ to } 70 \mu\text{sec.}$$

The factor  $\tau$  is a measure of hole storage in a standard circuit.

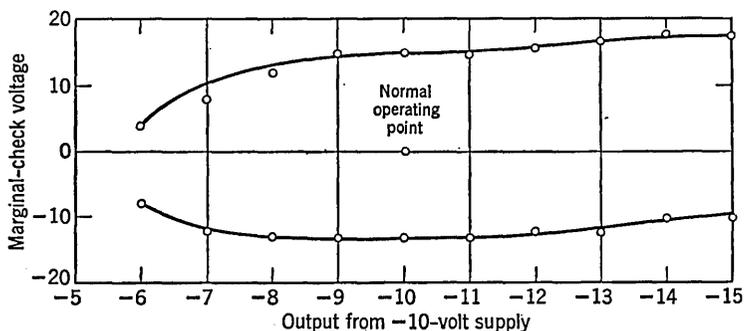


FIG. 41. High-speed flip-flop marginal-check voltage versus -10-volt supply.

The *marginal-checking curves* for the flip-flop group are illustrated in Figs. 40, 41, 42, and 43. Variation of the marginal-checking voltage from its nominal +10-volt value as a function of input pulse amplitude is

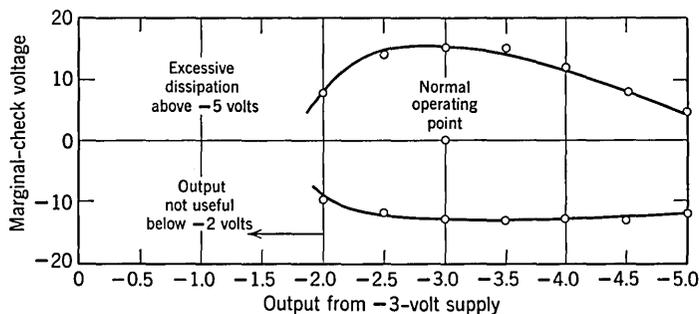


FIG. 42. High-speed flip-flop marginal-check voltage versus -3-volt supply.

shown in Fig. 40. The margins with respect to the output of the  $-10$ -volt supply are shown in Fig. 41 and with respect to the output of the  $-3$ -volt supply in Fig. 42. The margins in pulse amplitude are plotted in Fig. 43 for variations in  $\beta$  and  $\tau$  in one of the basic flip-flop transistors.

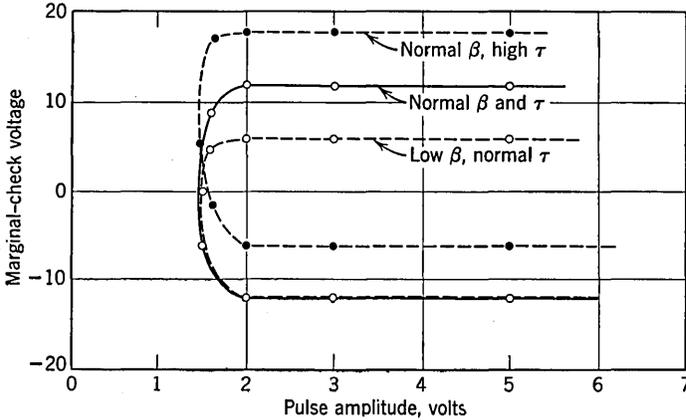


FIG. 43. Margins versus pulse amplitude for variations in  $\beta$  and  $\tau$  in one of the flip-flop transistors.

For normal values, the curve is symmetrical about the marginal-check axis. For either low  $\beta$  or high  $\tau$ , the curve is asymmetrical, thereby indicating variation of these transistor parameters.

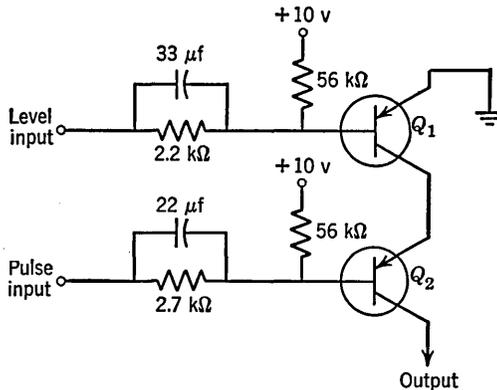


FIG. 44. Transistor pulse gate. Transistors: Philco SBT L-5122; resistors:  $\frac{1}{2}$ w 5%.

**High-Speed Pulse Gate.** A pulse level *and* gate for use with the high-speed flip-flop, described above, is shown in Fig. 44. This gate is equivalent in function to the circuit of Fig. 25 in which a pulse input

and a level input are gated together. The output of the pulse gate is usually connected directly to one of the inputs to the flip-flop group. The components in these inputs constitute collector load and supply for the pulse gate.

The emitter of transistor  $Q_2$  may be returned directly to ground to make  $Q_2$  a simple pulse amplifier instead of a pulse level *and* gate. The outputs of several such gates and/or such pulse amplifiers may be tied directly together at flip-flop inputs for logical *or* functions.

## 5. COMPONENTS, CHARACTERISTICS, AND APPLICATION NOTES

The discussion of design methods in the body of the report has specified many components and their applications in particular circuits. Certain component characteristics and notes on applications that have led to reliable electronic design are discussed in this section.

### Tubes

**Construction Characteristics.** The best tubes are assembled in air-conditioned, lint-free rooms, since the cause of many intermittent shorts has proved to be carbonized particles of lint and foreign matter within the tube envelope. Residual gases in the tube are reduced by a better-than-average vacuum. A tube designed for simplicity and ease of assembly will be more dependable than a complex construction.

Metal evaporated from the cathode and deposited on the structural mica insulators tends to lower the leakage resistance between the electrodes. To counter this, slots are cut in the insulators to lengthen these leakage paths.

Grid wires are plated to minimize secondary emission. Plates are designed to maximize the heat-radiating area, and are made of materials with minimum gas content.

**Grid Spacing.** A short, rugged mount structure with a minimum of 0.005- to 0.006-inch interelectrode spacing will minimize the incidence of shorts and troubles resulting from rough handling. For mount structures longer than 1 inch, even wider spacings are necessary. A requirement for high transconductance is basically opposed to the requirement for high tube reliability.

**Cathode Temperature.** The operating temperature of the cathode must be chosen as a best compromise among several factors that affect tube life. Lower cathode temperature may reduce emission below usable levels early in life, may increase the susceptibility of the cathode to poisoning by gases, and may increase cathode interface (an impedance that develops between the cathode base and the emitting material).

High cathode temperature increases the evaporation of the coating and sublimation of the base metal (with consequent increase in grid emission and depositing of metal coatings on structural insulators), increases the probability of heater burnout, and may accelerate the growth of cathode interface.

**Notes on Applications.**

1. Vibration and shock must be avoided during tube operation, since long-life pulse tubes are not usually designed to minimize microphonics.

2. Bulb temperature must be reduced to 80 to 100°C by forced-air circulation or air conditioning, to avoid evolution of gas from the glass envelope and to reduce the incidence of electrolysis of the glass between base pins under the influence of d-c potential.

3. Heater voltage regulation within 2 per cent of rated value has been demonstrated to give increased tube life.

4. D-c cathode current should be limited to 50 to 60 milliamperes per square centimeter of active cathode area.

5. Pulse cathode current should be limited to about 10 times this value for short pulses (under 1  $\mu$ sec) and low duty factors (under 10 per cent).

*Note.* Although these numbers have been proved safe in operation, they are not presented as limiting boundary values.

6. For longest tube life, design ratings for plate current and dissipation may have to be reduced from those given in manufacturers' literature. It is well to remember that manufacturers base their ratings both on the demand for particular tube characteristics and on the results of extensive life tests. For any particular tube application, the components engineer should work with the manufacturer's applications engineer to determine the basis for the tube's ratings, and the life that can be expected from the tube in this application.

7. Tests to determine compliance with specifications must be performed by the manufacturer, the user, or both. The manufacturer should give a preburning test to every tube to detect weak heaters. Tests to pick up transient shorts are particularly important if intermittent failures in service are to be minimized; these tests should be sufficiently sensitive to detect shorts of as little as 1-microsecond duration. Pulse and steady-state life tests on a sample basis, if continued sufficiently long, will give an indication of expected life. Manufacturers' tests should weed out potentially unserviceable tubes.

8. Standardization on a minimum number of tube types, and cooperation with the manufacturer's tube applications engineer to assure best use of these types, will materially reduce the incidence of interrupting tube failures.

## Semiconductor Diodes

**Characteristics.** Semiconductor diodes are most often used in switching networks or as blocking or clamping devices, to hold d-c levels in the presence of pulses, and to clip unwanted tails from pulses. All these applications presuppose that the diode will have nearly perfect rectifier characteristics—an open circuit in one direction, and a short circuit in the other. Every realizable diode falls short of this ideal. Furthermore, the diode acts differently when subjected to pulses than it does under static d-c conditions.

The principal characteristics of diodes are of importance to the pulse circuit designer. The first three of these are relatively self-explanatory, but forward recovery and reverse recovery are important factors that may be overlooked.

1. *Static (d-c) Forward Voltage Drop.* The diode volt-ampere curve is nonlinear, and electrical description of this characteristic should include a maximum value (or a tolerable range of values) of voltage drop at a specified forward current.

2. *Static Back Current.* A maximum value of back current should be specified at a fixed back voltage.

3. *Reverse Voltage Breakdown.* At sufficiently high reverse voltages, the back resistance of semiconductor diodes decreases rapidly, so much so that the diode may destroy itself. This reverse breakdown voltage point should be well above the highest back voltage that will be applied to the diode in any circuit application.

4. *Forward Recovery.* When a forward current is suddenly forced through some diodes, the voltage drop across them initially is as much as 200 per cent greater than the steady-state value and decays to the steady-state value in a fraction of a microsecond.

5. *Reverse Recovery.* The reverse resistance of a diode requires a finite time to build up after the diode has been switched from the forward to the back direction. In some junction diodes, the back resistance, immediately after switching, may be as low as the forward resistance before switching, and the recovery time may be hundreds of microseconds. In the faster whisker diodes, the initial back resistance is often not less than one-tenth the static back resistance and has a time constant of the order of 0.1 microsecond.

The characteristics described above should be specified at values close to those existing in the circuit application. If a diode is to be used under conditions different from those in the diode manufacturer's specifications, the tests performed should duplicate as much as possible the actual operating conditions.

Since diode characteristics are adversely altered by the effects of moisture and other contaminants, a sealed housing should be specified. Glass-enclosed diodes have been proved more satisfactory than diodes that depend upon some waxy filling compound for moisture resistance.

#### **Notes on Applications.**

1. Silicon and tungsten-whisker germanium diodes do not exhibit the very low forward pulse voltage drop needed for core driving and other high-current applications. Gold-bonded and indium-plated whisker diodes are better suited for such uses.

2. Junction diodes, some gold-bonded, and some tungsten-whisker diodes exhibit a long switching time (slow reverse recovery). Diodes for use in fast-pulse computers should be specified for fast reverse recovery.

3. If a diode is exposed to high temperatures during soldering, its characteristics may change, and life may be impaired. It is wise to solder quickly, and to use a heat shunt (a copper alligator clip would do) on the diode lead while heat is being applied.

4. The back resistance of a diode may change markedly during life, particularly when traces of contaminants introduced during manufacture remain within the diode case. For circuits intended to give long service, the designer should use for back resistance a design figure that is considerably less than the one given in the manufacturer's specifications. A factor of 5 for degradation of back resistance from initial to end-of-life specifications has been used successfully.

5. Diodes should not be used indiscriminately in series to increase the back voltage rating, or in parallel to increase current-handling ability. In series connection, a larger fraction of the applied back voltage will appear across the diode with the higher back resistance, increasing the probability of failure. Likewise, in the parallel connection, the diode with lower forward resistance will carry most of the current. Balancing resistors across each diode in the series connection, and in series with each diode in the parallel connection, will help to equalize the voltages and currents. The effect of the balancing resistors is to reduce back resistance, and to increase effective forward resistance, respectively.

#### **Connectors**

Connectors often give rise to intermittent contact troubles. These usually result from inadequate contact pressure, which allows oxide or other insulating films to develop on the contact surfaces. After being flexed many times, spring members of the contact may become fatigued or distorted, and so reduce the contact pressure. If plugs are left dis-

engaged for long periods, oxide or other insulating layers may develop on contact surfaces. The following suggestions may help prevent these difficulties.

1. Contact spring members should be relatively long, to avoid stress concentration when the spring is bent, and should be made of a non-fatiguing material such as beryllium-copper.

2. When a plug is mated with its socket, one contact surface should wipe over the mating surface for a considerable distance, to break down any insulating films.

3. Contacts should be plated with a nontarnishing metal. Gold and rhodium have been satisfactory.

4. Contact pressure from surface to mating surface should be high, whereas the force required to mate the connector should be moderate to avoid damaging the connector parts during mating.

## Relays

**Construction Characteristics.** Relay difficulties center around contact erosion, failure to make contact through oxide or foreign matter on contacts, and mechanical failure of contact springs and armature supports. For low-current, signal-handling relays, the following suggestions apply:

1. Extreme miniaturization should be avoided. Relatively long spring members will prevent stress concentration. A relatively large, conservatively rated coil should be used to avoid overheating and burnout.

2. Bifurcated (twin) contacts on each contact spring will ensure that if one contact of the pair is blocked by dust or foreign matter the other one will still provide continuity.

3. A dust cover with a gasket seal should be provided.

4. Cotton, paper, or other such organic insulations will, under the influence of d-c potential and moisture, contribute to electrolytic corrosion of the windings. Nylon, cellulose acetate, and similar synthetic insulations are satisfactory. The potential of the coil should be negative with respect to the relay frame.

5. Arc-suppression networks across contacts that break current-carrying circuits should be provided to absorb the inductive voltage produced when the contacts open.

**Enclosed Relays.** In high-altitude, moist or saline conditions, and in situations where tampering can occur, the hermetically sealed relay is essential. However, since the products of arcing, or varnish products produced when the coil overheats, may collect within the relay case and interfere with contact operation, sealing is not a cure-all. Further, the

relay cannot be adjusted if its parts should fatigue after many operations, and it must therefore be discarded. Where expert maintenance personnel are available, as is the case at a large computer installation, relays can be equipped with removable dust covers and thus be accessible for servicing.

**Time Delays.** There is always a time delay between application of voltage to the relay coil and the actual closing of the contacts. Likewise, there is a delay between removal of the voltage and the actual openings of the contacts. This timing may change during the life of the relay, as a consequence of fatigue of spring members. The initial timing, and that after a large number of operations on a repetitive life test, should be studied if close timing is required by the circuit application.

Various means are used to increase the delay in the closing or opening of a relay, including mounting copper slugs on the relay core (which act as a short-circuited turn to delay the buildup or decay of the magnetic field), provision of fluid-filled or air dashpots, and use of external *RC* timing circuits.

**Power Supply Transients.** Relay operation may create power supply transients in other parts of the system when large currents are interrupted. Such transients may be minimized in some cases by operating the relays when minimum current is flowing through their contacts. In other cases, it is necessary to resort to decoupling networks or filters.

## Resistors

If intelligently selected and applied, resistors are a particularly dependable class of components. To assure long life, all standard resistors should be derated from their nominal power and voltage ratings.

**Composition Carbon Resistors.** Available in 5, 10, and 20 per cent initial tolerances, composition resistors are not so stable as other types during long life. A resistor considered by the manufacturer as a  $\pm 5$  per cent unit may vary as much as  $\pm 20$  per cent during several years use. The nominal power ratings of these resistors should be cut by 50 per cent to assure stability and long service.

**Film Resistors.** These include deposited carbon, precious metal, and electrically conductive glass film types. They are considerably more stable than composition resistors, but a resistor nominally of 1 per cent tolerance should be included in the power rating of deposited carbon types: they should be derated to 50 per cent of nominal power rating. The metal and glass films are not so critical. If deposited carbon resistors are used, they should be hermetically sealed. Varnish coatings have been shown to admit moisture, causing early failures. Glass housings, and ceramic housings with solder-sealed ends, are satisfactory.

**Precision Wire-Wound Resistors.** These resistors are quite reactive, and thus are not highly useful in fast-pulse circuitry. They may be of great value in digital-to-analog decoders, since they are more stable than any other kind of resistor. A 1 per cent tolerance resistor will stay within 5 per cent during life if it is properly derated (50 per cent of nominal rating) and is not subjected to violent temperature excursions. Wire smaller than 0.002 inch should not be used in critical equipment. The construction that employs a cast or molded plastic bobbin, with an outer housing of the same plastic, is far superior to the older ceramic-bobbin, paper-wrap, varnish-impregnated construction.

**Power Wire-Wound Resistors.** These are useful in power supply bleeder and other high-power applications. Only the vitreous-enameled varieties are recommended. Power dissipated in the resistor should not exceed 50 per cent of the nominal power rating. Resistance wire smaller than 0.002 inch should not be used. A metal mounting bracket to conduct heat to a metal panel or chassis should be used to carry off most of the heat dissipated in the resistor.

**Potentiometers.** For low-power noncritical uses, the carbon potentiometer with a molded resistance element is satisfactory. As with composition carbon resistors, potentiometers should be derated, and consideration should be given to end-of-life tolerances. Vitreous-enameled power rheostats have proved reliable in values up to 5000 ohms.

## Capacitors

If adequately specified and properly applied, capacitors need cause little trouble in electronic equipment. Capacitor failures result principally from poor construction techniques and from application of excessive voltage.

**Paper Capacitors.** These should be hermetically sealed in metal cans, should have at least three layers of dielectric paper, and use a fluid impregnant such as mineral oil or one of the complex hydrocarbons. Applied voltage should not exceed 50 per cent of the manufacturer's rated voltage, and should be even less for use at temperatures greater than 55°C. Alternating-current ripple voltage impressed on the capacitor may cause dielectric heating, which would require further derating. The capacitance can vary by 12 per cent over the initial tolerance during life. Paper capacitors should not be used in circuits where day-to-day stability of capacitance is important. Tubular, bathtub, and rectangular can cases are popular.

**Mica Capacitors.** Mica capacitors, particularly those using silver paint as the electrode material, are quite stable and trouble-free. Silvered micas, though, are subject to migration of the silver across insulat-

ing boundaries under the influence of moisture and d-c voltage. Ordinary molded phenolic housings are not sufficiently moistureproof to prevent silver migration. The design tolerance should be taken as  $\pm 10$  per cent beyond the nominal purchase tolerance for nonsilvered units, and  $\pm 5$  per cent beyond purchase tolerance for silvered micas; 50 per cent voltage derating should be applied. Some trouble has been encountered in capacitors where the termination of the pigtail lead is crimped rather than soldered in a slot in the end plate inside the body.

**Ceramic Capacitors.** These, in the disk and tubular forms, are quite dependable. Units over 0.001 micromicrofarad use ceramic with high dielectric constant, and are quite unstable; these should be used only for bypass and decoupling applications where exact values of capacitance are of no concern. In smaller values, the design tolerance should be considered to be  $\pm 10$  per cent beyond the purchased tolerance.

**Electrolytic Capacitors.** The best electrolytic capacitors are "telephone quality," originally developed and produced by several suppliers for the Bell System; they are intended to give ten years or more service life, and two years nonoperating shelf life. A 10 per cent voltage-derating factor should be applied, and surge voltages seen by the capacitor should never exceed the manufacturer's rated *operating* voltage. Capacitors that have been out of service for some time should be reformed by connecting each in series with a 1000-ohm resistor to a source sufficient to develop rated voltage across the capacitor for a period of one hour.

### Transformers and Inductors

The principal troubles encountered in transformers stem from the effects of moisture. Electrolytic corrosion, plating away the copper wire onto the transformer frame in the presence of d-c potential and moisture, may cause open-circuit failures. Decomposition of insulation, as evidenced by low insulation resistance, is also caused by moisture.

Such troubles can be remedied by insisting upon adequate varnish impregnation of the winding, and hermetically sealing the case. Very small wires used in miniature transformers may also be subject to failure from corrosion. High operating temperature of the windings may accelerate failure of the insulation. The maximum operating temperature of the windings recommended for long, reliable service is 70°C.

Small pulse transformers using toroidal ferrite cores have been most satisfactory for short-pulse circuits. When hermetically sealed and operated within the manufacturer's d-c ratings, they have proved extremely dependable.

RF chokes have given very little trouble in service. Vacuum-varnish-impregnated coils, wound on ferrite, powdered iron, or passive cores have been most satisfactory.

## 6. TRANSISTORS

**Introduction.** The transistor does not have the long history of use accruing to most other electronic components. The transistor field is a fast-moving one in which the picture is continually changing; this means that practices and procedures may change as the units themselves change. Although various revolutionary devices are now on the horizon, this section is concerned with devices presently available in production quantities.

The original transistors were point-contact devices, and the theory of these was not completely developed. Mechanical and electrical stability of early units was poor. The appearance of the junction transistor in 1950–1951 brought about the big break-through of the transistor into equipment design. Mechanically, it is a rugged unit which can be made to handle power of the order of a few watts.

**Construction Characteristics.** Most of the problems of transistor reliability can be traced to contamination in manufacture and leakage of water vapor into the device after fabrication. After receiving its final clean-up etch, the transistor must be handled under extremely clean conditions and well-controlled humidity to prevent accidental contamination of the surfaces before or during encapsulation. For long life, a hermetic seal is a “must.”

Potting compounds—solid, liquid, or gaseous—may be used for protection against welding vapors or soldering fluxes, or they may be used to fix the surface potential at a stable point. Different treatments may be required for different types of units.

**Types of Junction Transistors.** Junction transistors are basically very small sandwiches of  $p$ -type and  $n$ -type semiconductor material. These sandwiches may be arranged as  $p-n-p$  or  $n-p-n$  in structure in which these layers are, respectively, the emitter-base-collector electrodes of the transistor. The sandwich structure may be built up by either an alloying process (alloy junction transistor) or a crystal-growing process (grown junction transistor).

The existence of the junction transistor in the two forms,  $n-p-n$  and  $p-n-p$ , using bias supplies of opposite polarity, makes possible some very useful circuits. The alloy junction transistor makes a practical switch, having an *off* resistance of about a megohm and an *on* or saturated resistance of about 10 ohms.

Most of the present day transistors are made from germanium. Silicon transistors, however, are available and can be used at temperatures about 100°C.

### **Junction Transistor Characteristics**

Since transistors have not yet been widely used in computer circuits, little life data have been accumulated. Therefore, transistor types cannot be selected on the basis of proven reliability. If the transistors are to be used in large numbers, the types chosen should be those that can be manufactured with reasonable yields by present fabrication techniques. Characteristics that are important in the design of reliable transistor circuits are frequency response, power dissipation, and storage time.

**Frequency Response.** The frequency response of a junction transistor is inversely proportional to the square of the base width. Unfortunately, in an alloy junction transistor, the maximum emitter-collector voltage, the punch-through voltage, varies directly as the square of the base width. Therefore, as the frequency response is raised, the maximum operating voltage is decreased. This punch-through voltage relationship does not hold in a grown junction transistor. However, with present methods of fabrication, it is difficult to obtain high production yields of conventional, high-frequency, grown junction transistors (15 to 20 megacycles).

Special jet-etching and plating techniques make possible extremely accurate base width control in the surface barrier transistor. These units are in the 60- to 70-megacycle cutoff range. Maximum voltage is usually 6 to 20 volts, with recommended operating voltage 4 to 5 volts.

**Power Dissipation.** The power-handling capability of a transistor depends on good thermal design. Heat must be conducted away from the collector junction which, in a germanium transistor, cannot be allowed to exceed about 85°C. Provision for external connection to a heat sink is desirable for high-power units.

Since maximum voltages are normally limited to less than 50 volts, high-power transistors must be capable of passing high currents. The tendency for current gain to decrease with increased emitter current (alpha crowding) can be alleviated by double-doping the emitter. The problem of self-cutoff for the center region of the emitter is eliminated by the use of an elongated rectangular emitter.

Most high-power units will have poor frequency response (in the audio range) because the large electrode area necessary for high currents means high junction capabilities and wider base widths (to allow for the possibility of uneven alloying). Most high-frequency transistors tend to have

low power ratings. The surface barrier transistor is rated at 10 milliwatts.

**Storage Time.** The collector current of a transistor that has been saturated or turned *on* does not stop flowing the instant a turn *off* signal is applied. The transistor remains at a low impedance for a short period of time until it recovers from the saturated condition. This delay or storage time is of considerable importance in high-speed saturating type switching circuits. Storage time can be reduced by making the base width small, that is, by designing a high-frequency transistor.

### Notes on Applications

1. *Temperature limits* must be rigidly observed and units should be operated close to or below room temperature if possible. Transistors good for tens of thousands of hours at room temperature may fail in hundreds of hours at 100°C. Since high power dissipations cause internal temperatures to rise, circuits must be designed to keep power dissipation low. Silicon transistors should be used where high temperatures are unavoidable.

2. *Aging.* In general, as transistors age, collector leakage current will increase, current gain will decrease, and breakdown voltage will decrease. Circuits must be designed to allow for these gradual variations in parameters.

3. *Soldering.* For best reliability, transistors should be soldered into circuits and a heat sink should be used to prevent excessive junction temperatures. The leakage in ordinary 110-volt a-c soldering irons may be sufficient to damage certain types of low-power transistors (such as the surface barrier transistor). Irons with on-off switches have been found to develop large momentary transients capable of destroying these units. A small 6-volt iron is recommended and will eliminate many of these difficulties.

4. *Testing.* Transistors should be tested individually in the grounded emitter configuration by using an automatic curve plotter. Grounded emitter displays magnify most of the important irregularities by the factor of the current gain  $\beta$ . The avalanche breakdown voltage occurs at a lower value for this configuration.

5. For *saturating switching applications*, the alloy junction transistor is usually superior to the grown junction transistor since it has a lower emitter-collector impedance.

6. *Life tests* made under conditions similar to those of the contemplated use are helpful in predicting reliability. These tests may possibly be accelerated by operation at increased ambient temperatures (with considerable reservations about the validity of such a practice since addi-

tional failure mechanisms may occur at elevated temperatures). This procedure is not recommended if reasonable time is available for room temperatures tests.

#### ACKNOWLEDGMENTS

In the preparation of this report, particular contributions are those of R. L. Best in the section on Basic Circuit Criteria and Marginal Checking Techniques, and W. J. Canty in typical tube computer circuitry. Transistor circuitry was contributed by K. H. Olsen and his section. The transistor as a component has been reported by D. J. Eckl. The component work has been reported by B. B. Paine.

Nolan Jones has carried the burden of collecting and clarifying the text. The Publications group of the Laboratory has made many helpful suggestions as to methods of organization and expression.

Much of this material is based on an article written by Norman H. Taylor and is used with the permission of the Institute of Radio Engineers.

#### REFERENCES

1. N. H. Taylor, Designing for reliability, *Proc. I.R.E.*, **45** (6), 811-822 (1957).
2. N. H. Taylor, Marginal checking as an aid to computer reliability, *Proc. I.R.E.*, **38**, 1418-1421 (1950).
3. H. W. Boyd, Design of a reliable flip-flop, *RETMA Symposium on Reliable Applications of Electronic Tubes*, Philadelphia, Pa., May 21, 1956.
4. J. J. Hoffer, Application techniques leading to reliable circuit performance, *RETMA Symposium on Reliable Applications of Electron Tubes*, May 21, 1956.
5. K. Henney *et al.*, *Reliability Factors for Ground Electronic Equipment*, McGraw-Hill, New York, 1956.
6. W. J. Canty, Designing for reliability, *Joint Military-Industry Guided Missile Reliability Symposium*, Huntsville, Ala., October 16, 1956.
7. N. L. Daggett and E. S. Rich, Diagnostic programs and marginal checking with the Whirlwind I computer, *1953 I.R.E. Convention Record*, Pt. 7, 48-54.
8. W. J. Canty, Experiences in setting up a SAGE prototype computer, *Am. Inst. Elec. Engrs. Summer General Meeting*, Paper No. CP 57-679, June 24-28, 1957.
9. H. F. Heath, Jr., R. E. Nienburg, M. M. Astrahan, and L. R. Waters, Reliability of an air defense computing system, *I.R.E. Trans. Elec. Computers*, **EC-5**, 224-237, December 1956.

## Magnetic Core Circuits

*Isaac L. Auerbach*

1. Fundamentals	15-01
2. Magnetic Cores	15-04
3. Transfer Loops	15-09
4. Magnetic Shift Registers	15-15
5. Logical Function Elements	15-16
6. Magnetic Core Storages	15-19
7. Timing Control Circuits	15-21
8. Arithmetic and Miscellaneous Applications	15-22
9. Drivers for Magnetic Core Circuits	15-23
References	15-24

### I. FUNDAMENTALS

Circuits employing bistable magnetic cores are characterized by compactness, low power consumption, and long life with reliable operation. They are capable of performing most of the functions of digital data processing systems, including storage, delay, control, and logical operations.

**Switching Concepts.** The idealized rectangular hysteresis loop of Fig. 1 illustrates the binary property of cores used in these circuits. The positive residual state  $+B_r$  is designated binary 1, the negative state  $-B_r$  binary 0. Application of a pulsed magnetizing force along the  $H$  axis of sufficient amplitude and duration will set the core in a desired binary state where it will remain until pulsed in the opposite sense.

Reapplication of magnetization in the same sense results in relatively little flux change, e.g., from point  $B_r$  to  $B_m$  and back to  $B_r$ .

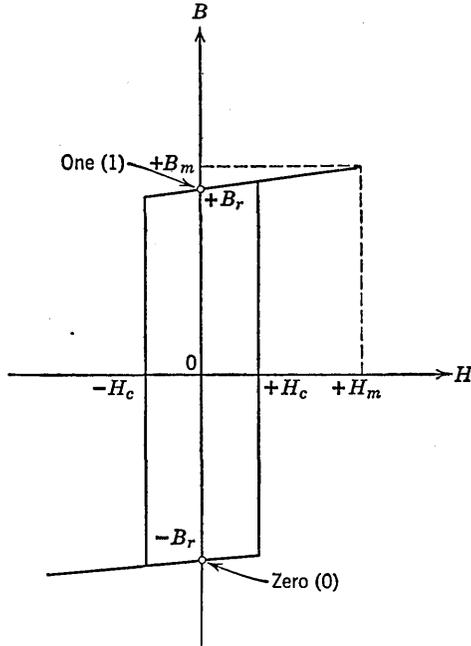


FIG. 1. Idealized rectangular hysteresis loop.  $H_m$ , applied magnetizing force;  $H_c$ , coercive force;  $B_r$ ,  $B_m$ , magnetic induction.

The core may be used as an *ampere-turn transformer* ( $T$  in Fig. 2) with output coupled to the input of another core or device. Conventional dot notation indicates winding polarity with the added definition that

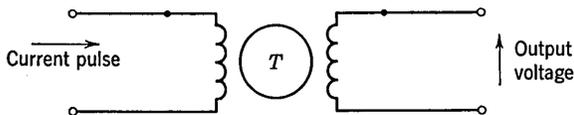


FIG. 2. Magnetic core transformer,  $T$ .

current *into* a *dot* terminal (positive end) will set the core to the 0 state. Typical waveforms of voltages induced in the output winding are illustrated in Fig. 3 together with total flux variations within the core material. Signal-to-noise voltage ratios are generally greater than twenty to one with commercially available cores.

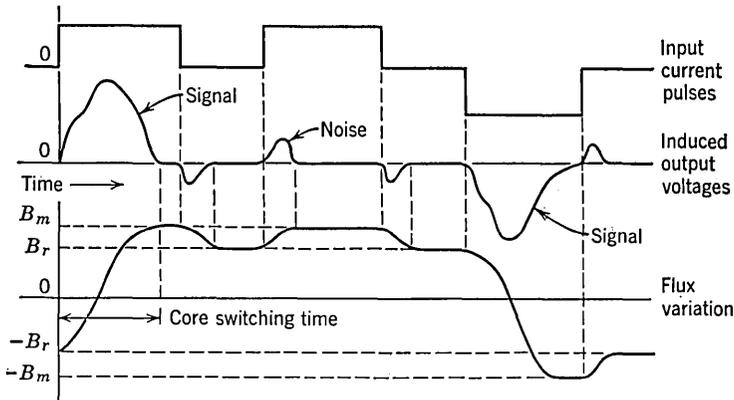


FIG. 3. Typical waveforms associated with bistable magnetic cores.

The core may also be regarded as a *variable impedance* (Fig. 4). If the core is in the 1 state when  $I$  is applied, the core will switch, a relatively large counter-emf,  $e$ , will be generated in the winding, and a high impedance will be presented to the driving source during the period of switching. If in the 0 state, the emf and presented impedance will be small. This bistable impedance characteristic is the basis for a number of circuits.

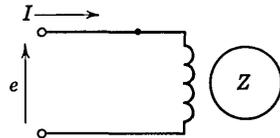


FIG. 4. Magnetic core as a variable impedance.

**Logical Representation.** A simplified functional representation is helpful in understanding the operation of magnetic core systems and as a basis for circuit design. A core is represented by a circle. A line with an arrow pointing to the circle (Fig. 5) denotes an input to the core, and the numeral within the circle the state to which the core is set by the input. Open arrows indicate pulses; closed arrows d-c signals. Two opposed signals applied simultaneously cancel and leave the core in the original state. Double arrows indicate overriding inputs that determine the state of the core regardless of opposing inputs.

A line originating at the circle denotes an output from the core. The output signals are generally limited to a single polarity by the use of unilateral elements; thus, the numeral within the circle at the origin of the line indicates the state *to which* the core must be switched from the other state to produce the output. If an output is produced by only one of several transfer inputs, the signal output is said to be conditional, indicated by an "eyebrow" drawn within the circle. Symbols along lines identify signals, and often include the times at which the data appears.

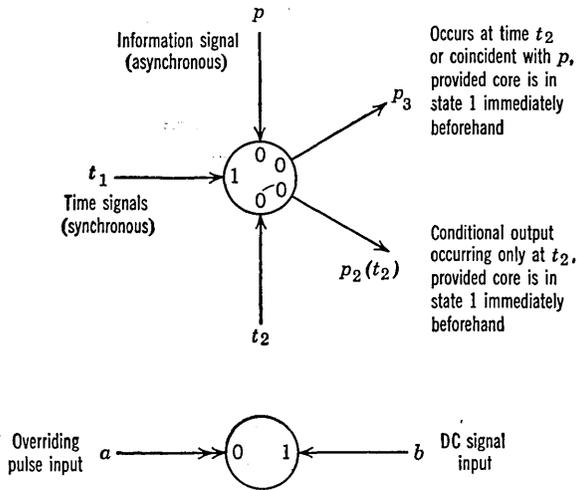


FIG. 5. Basic logical symbols.

## 2. MAGNETIC CORES

Magnetic cores have been developed in two basic forms, the metal strip types and the molded ferrites. The strip cores usually consist of a number of wraps of a microthin alloy tape wound on a ceramic bobbin, spot welded in place and annealed. They may be plastic covered, or potted in resin to afford greater mechanical and chemical protection. Ranging in nominal sizes from about  $\frac{1}{8}$  to 1 inch diameter, metal cores are generally employed in manipulative or logical types of circuits. Although considerably more expensive than ferrite cores, they are less affected by temperature change and require less power under comparable operating conditions.

The ferrites are fabricated from various mixtures of pure metals, metallic oxides, and binding materials pressed into toroid shapes and fired at high temperatures—a process familiar to the ceramics industry. They may be obtained in sizes smaller than is possible with wound strip, diameters of 80 mils (0.080 inch) being quite common. Because of size and low cost, the ferrites are generally preferred in static memory applications where the number of cores required is likely to be large.

**Magnetic Cores as Circuit Elements.** Windings are applied to the core in conformity with the design requirements—either by the supplier or, more commonly, at circuit assembly. So fitted, the core becomes a circuit element in the more usual sense, but one which lends itself to flux-current considerations rather than voltage-resistance analysis.

**Technical Data for Circuit Design.** While the basic properties of magnetic cores are described by the hysteresis curves, of more practical importance are the relationships between magnetomotive forces  $NI$  and switching flux  $\Phi_s$ , switching time  $T_s$ , and squareness ratio  $S$ . The curves are generally shown, in the case of  $\Phi_s$  and  $T_s$ , as upper and lower limits over the recommended operating range. Squareness ratio  $S$ , defined as  $B_r/B_m$ , is important as a figure of merit of the particular core alloy. The curves of Fig. 6 are typical of a type of metal strip core used in magnetic core manipulative circuits.

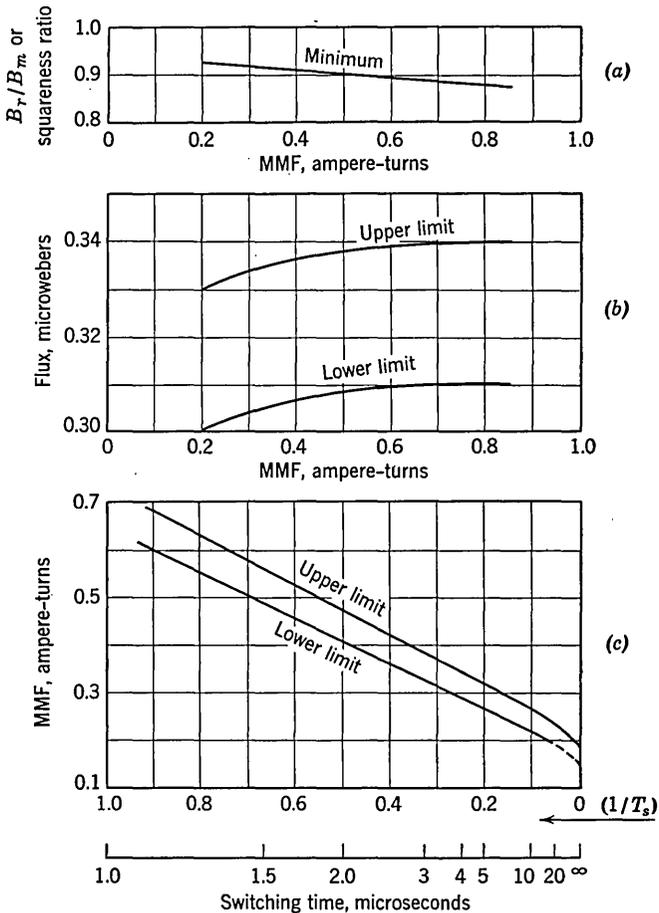
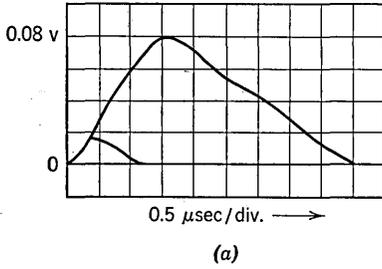
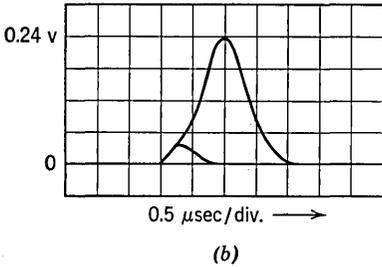


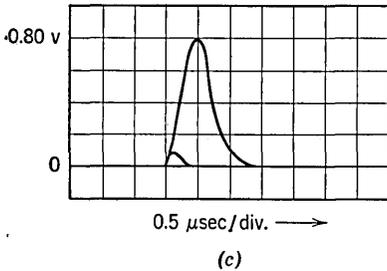
FIG. 6. Characteristic curves for a typical metal strip magnetic core. Material, 4-79 Permalloy; tape thickness,  $\frac{1}{8}$  mil; tape width,  $\frac{1}{8}$  in.; mean diameter, 0.200 in.; number of wraps, 20. (a) Squareness ratio  $S$ . (b) Switching flux  $\Phi_s$ . (c) Switching time  $T_s$ ; current pulse rise time  $0.5 \mu\text{sec}$ . (Courtesy the Burroughs Research Center.)



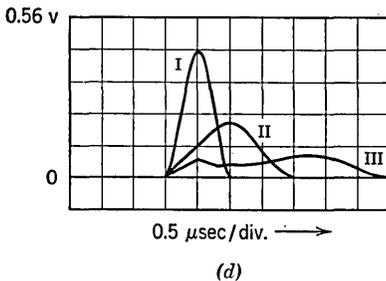
(a) Material, 48% Ni-Fe; tape thickness,  $\frac{1}{2}$  mil; tape width,  $\frac{1}{8}$  in.; mean diameter, 0.5 in.; squareness ratio, 0.90; number of wraps, 2;  $H_m$ , applied magnetizing force equals 1.0 oersted.



(b) Material, 4-79 Permalloy; tape thickness,  $\frac{1}{4}$  mil; tape width,  $\frac{1}{8}$  in.; mean diameter, 0.5 in.; squareness ratio, 0.91; number of wraps, 9;  $H_m$  equals 0.5 oersted.



(c) Material, MF-1118 (ferrite); cross-sectional area, 0.0026 sq in.; mean diameter, 0.58 in.; squareness ratio, 0.95;  $H_m$  equals 3.0 oersteds.



(d) Switching voltages only for different values of applied magnetizing force  $H_m$ , core (b). I, 1.0 oersted; II, 0.5 oersted; III, 0.25 oersted.

FIG. 7. Switching and nonswitching voltages of various bistable magnetic cores.

**Core Constants.** Two useful constants may be derived from the switching time-mmf characteristic or "core curve," shown in Fig. 6c. The pulse threshold mmf ( $F_0$ ), which corresponds to the intercept at  $1/T_s = 0$ , is defined as that value of magnetomotive force below which no switching is possible. It is important from the standpoint of noise immunity. Coefficient  $G$ , the slope of the core curve referred to the  $1/T_s$  base, is constant over the useful operating range. With the aid of the upper and lower limits of  $\Phi_s$ ,  $F_0$ , and  $G$ , it is possible to make quantitative calculations of a core's performance under many conditions without resort to the complete characteristic curves.

**Noise in Bistable Magnetic Cores.** A source of noise inherent in magnetic cores is nonsquareness of the hysteresis loop. The amount of flux change produced when a core is driven from remanence to the corresponding point of saturation can be considered as false information; the noise source theoretically would vanish if a unity squareness ratio were attainable in actual square-loop magnetic materials.

Figure 7 illustrates the switching waveforms of three representative bistable cores. The larger of the waveforms is the information signal; the smaller, the noise signal. Figure 7d shows a comparison of the three information signals for the types of core material noted.

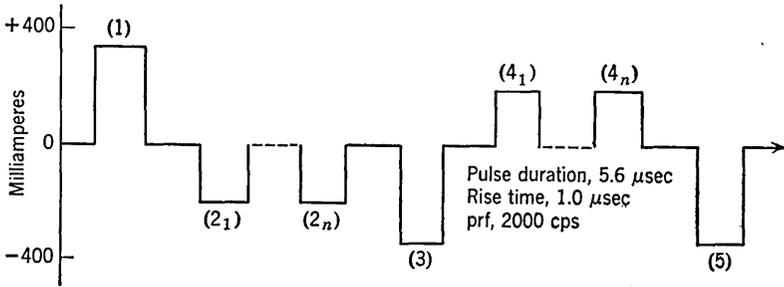
**Ferrite Storage Cores.** Ferrite cores which are carefully manufactured and tested for uniform characteristics are generally required for coincident-current storage applications. These minute cores must change state in response to a current pulse of specified waveform, but they remain unaffected by a series of pulses one-half the amplitude. See Chap. 19 for a general discussion of storage.

To evaluate the cores properly for the storage application, test specifications generally are formulated which subject a single core to conditions that might be encountered in an operating array. Figure 8 shows a current pulse pattern which:

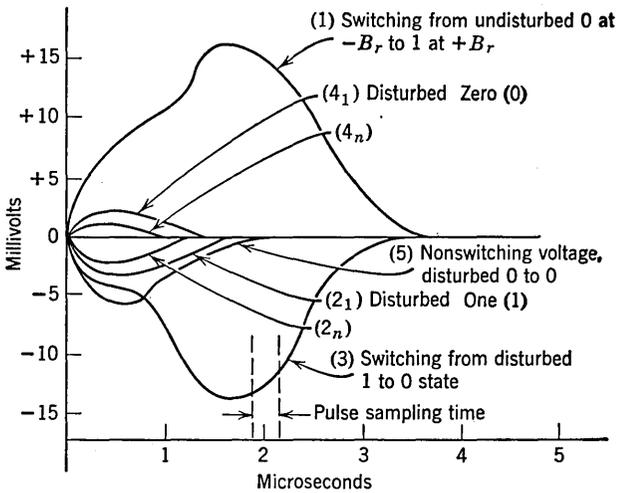
1. Sets a typical storage core in the positive state of magnetizing.
2. Disturbs the state by a series of partial pulses that tend to reduce the magnitude of residual flux.
3. Switches the core from the disturbed positive state to the negative state.
4. Disturbs the negative state toward the positive.
5. Restores the core from the disturbed negative state to the normal negative state.

In the case of a satisfactory core, the amount of flux change during the disturbance is not a function of the number of partial pulses, provided the number of such pulses is greater than some small number, usually

2 or 3. The flux changes, moreover, should be as small as possible since the noise voltages thus induced may cause error in the information readout.



(a)



(b)

Fig. 8. Determination of minimum signal and maximum noise in the output of a ferrite core for storage matrix application. Core of ferrite, type S3 (grade B); outside diameter, 0.080 in.; inside diameter, 0.050 in.; thickness, 0.025 in. (a) Applied current pulse pattern. (b) Observed output voltages at  $77 \pm 2^\circ \text{F}$ .

Signal-to-noise ratios in the outputs of individual cores accordingly are considerably less favorable, on a peak amplitude basis, than in applications where partial pulses need not be taken into account. These ratios, however, usually may be improved by employing sampling methods in the matrix output equipment.

**Temperature Stability.** Normal changes in operating temperature have negligible effect upon the characteristics of metal strip cores. Variations of less than  $\pm 2$  per cent over the range of 0 to 150°F are not uncommon in the case of many quality components. The molded ferrites, on the other hand, are considerably less stable, particularly the special grades supplied for coincident current matrix applications. Reliable operation of these systems usually requires control of core temperatures within a few degrees of a specified ambient value.

### 3. TRANSFER LOOPS

Transfer loops are circuits connecting two or more cores for the purpose of information transfer. The loop includes basically an output winding of the transmitting core and an input winding of the receiving core.

**Single Diode Transfer Loop.** The single diode transfer loop is the simplest form of loop. It is illustrated schematically in Fig. 9, together with the logical representation. The diode isolates core  $M_b$  when 1 is

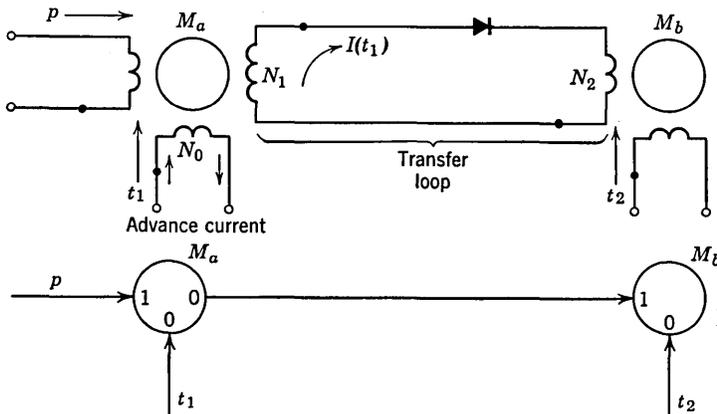


FIG. 9. Single diode transfer loop. Typical values: cores,  $\frac{1}{4}$  mil, 4-79 Permalloy,  $\frac{1}{8}$  in. wide; 10 wraps,  $\frac{3}{16}$  in. diameter; diode, T6; windings:  $N_1$ , 21 turns;  $N_2$ , 5 turns;  $N_0$ , 15 turns; current waveforms, trapezoidal; rise times, current pulse, 1  $\mu$ sec;  $p = I(t_1)$ , 70 ma;  $t_1$ , 190 ma; core switching times:  $M_a$ , 8.0  $\mu$ sec,  $M_b$ , 3.5  $\mu$ sec.

being set into  $M_a$ . A turn ratio  $N_1/N_2$  of about 4/1 and the nonlinear characteristic of the diode, discriminating against the smaller signals, prevent an undesired backward flow of information at time  $t_2$ .

The use of this type of loop in logical circuits is limited to applications where it is desired to switch the core *whenever* switching current exists in  $N_0$  or any similar winding on the core. In these instances the transfers are said to be unconditional.

**Split-Winding Transfer Loop.** (See Refs. 1, 2, and 9.) The split-winding loop, one form of which is shown in Fig. 10, provides for conditional transfer of information while permitting isolated operations on either the transmitting or the receiving core. Unlike the single diode loop, it is immune to the backward flow of information and makes practicable the simultaneous switching of as many as five or six receiving cores. By proper design the impedance of  $N_1$  will be large relative to

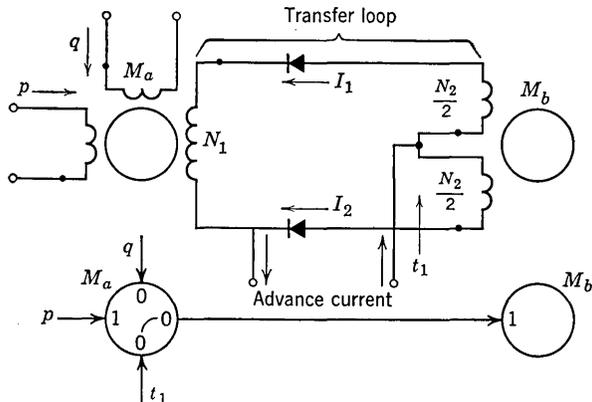


FIG. 10. Split-winding transfer loop. Typical values: cores,  $\frac{1}{4}$  mil, 4-79 Permalloy,  $\frac{1}{8}$  in. wide; 10 wraps,  $\frac{3}{16}$  in. diameter; diodes, T6, windings:  $N_1$ , 31 turns;  $N_2$ , 6 turns; coil resistance,  $N_2$ , 8 ohms; current waveforms, trapezoidal; rise times, current pulse, 1  $\mu$ sec;  $t_1$ , 125 ma;  $I_1$ , 8 ma;  $I_2$ , 117 ma; core switching times:  $M_a$ , 10.4  $\mu$ sec;  $M_b$ , 4.0  $\mu$ sec.

the other impedances in the transfer loop (see Fig. 10), and branch current  $I_1$  will be much smaller than  $I_2$ . The resultant ampere-turns difference is sufficient to switch  $M_b$  to 1. Branch current  $I_1$  clears  $M_a$  to the 0 state.

Another type of conditional transfer loop is shown in Fig. 11. As in the case of the split-winding loop, the two diodes insure immunity to the backward flow of information and permit isolated operations upon either core. The function of conditional transfer from  $M_a$  to  $M_b$  in response to  $I(t_1)$ , while analogous to that of the preceding circuit, is accomplished in a somewhat different manner. Current pulse  $I(t_1)$  is steered through either the upper or the lower branch of the loop, depending upon the state of core  $M_a$  immediately prior to  $t_1$ . If in the 0 state, except for effects traceable to a nonideal core, all the applied current will flow through the high-conductance diode  $D_2$ , bypassing  $M_2$  and leaving core  $M_b$  in the 0 state. When  $M_a$  previously has been set to 1,  $I(t_1)$  acting through  $N_0$  switches the transmitting core toward 0. The changing flux induces a voltage in the output loop such that nearly all the current flows through

$D_1$  and  $N_2$ , switching  $M_b$  to 1. In practical circuits, parameters are generally so chosen that approximately 10 per cent of the current flows through the lower loop during transfer to stabilize the switching action of core  $M_a$ .

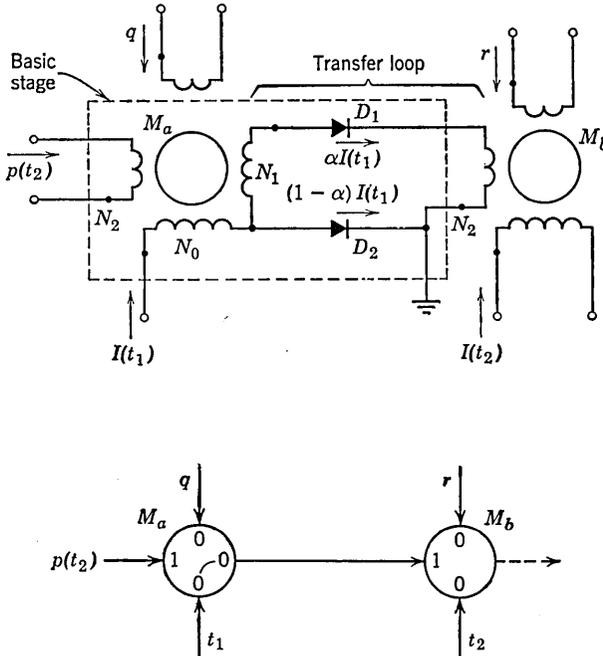


Fig. 11. Another form of transfer loop. Typical values: cores, Ferrite S3, General Ceramics; diodes, T25G; windings:  $N_0$ , 23 turns;  $N_1$ , 20 turns;  $N_2$ , 8 turns;  $I(t_1)$ , 100 ma;  $\alpha$ , 0.9; core switching time:  $M_a$ , 3  $\mu$ sec;  $M_b$ , 2  $\mu$ sec.

*Note 1.* This means effectively that the current signal generated in the loop does not completely cancel  $I(t_1)$  in the lower branch of the loop.

*Note 2.* To insure a complete forward transfer of data, the switching time of the driving core must be longer than that of the driven core.

**Condenser Delay.** In certain data-handling systems it is often desirable to transfer information both *from* and *to* the same core by application of a single timing pulse. Such operation is possible if provision is made in the transfer loop to delay the input to the core until readout has been accomplished. Assuming the cores of Fig. 12 contain 1's, both are switched to 0 at time  $t$ . The delayed output of  $M_a$  then resets  $M_b$  to 1 before arrival of the next timing pulse (see Ref. 3).

Figure 13 illustrates a practical method for accomplishing the above. If the forward resistance of the diode is small compared with  $R$ , the

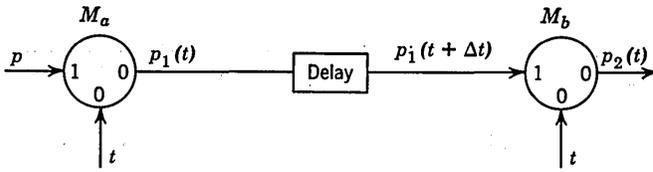


FIG. 12. Transfer of information into and out of core  $M_b$  by a single pulse, by using a condenser delay.

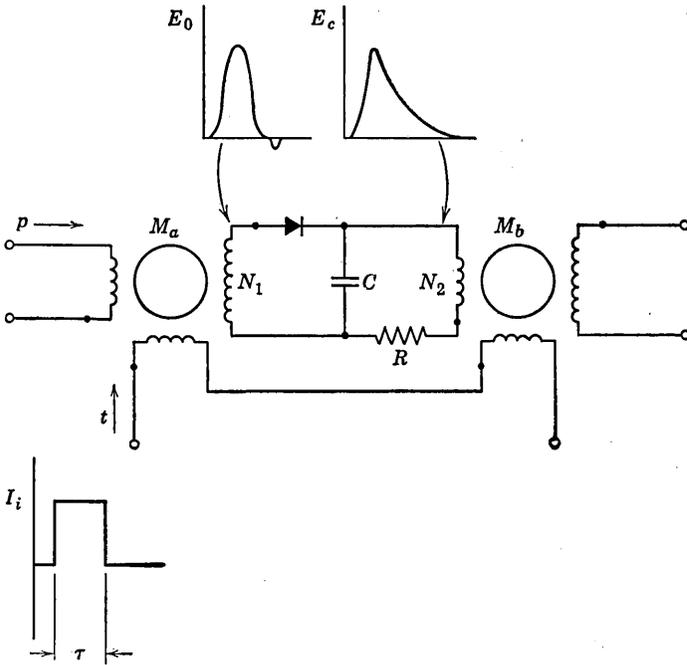


FIG. 13. Circuit corresponding to Fig. 12. Approximate circuit requirements: core switching times:  $M_a = \tau$ ;  $M_b = 1.75RC$ ; turns ratio:

$$\frac{N_1}{N_2} = \frac{\tau}{1.6ZC} (1 + 2m),$$

where  $Z$  equals average impedance of  $N_2$  during read-in of 1, and  $m$  is ratio of average drop across diode during charge to  $E_c$  (max.); maximum operating frequency:

$$f_m = \frac{1}{\tau + 2.5RC}.$$

capacitor charges to nearly the peak value of the induced voltage as  $M_a$  is switched by  $t$ , and sufficient charge is present to switch  $M_b$  to 1 after termination of  $t$ . These loops have been operated at frequencies as high as 500 kilocycles in cascades up to 50 stages with high stability. The technique adds to the flexibility of magnetic core circuit design.

**Transistor-Magnetic Core Loops.** (See Refs. 4 and 5.) The basic magnetic core transfer loop can be extended to include a transistor as a power-supplying element within the loop. Generally a grounded emitter configuration is employed with at least one core winding in the collector branch. Owing to the element of gain, input signals need only trigger the transistor to effect switching of the cores, and possible speeds of operation exceed those attainable with passive loops. Power consumption is characteristically low, and wide tolerances in component values and transistor parameters insure good reliability of operation.

A basic loop, applied successfully in circuits operating in the neighborhood of 100 to 150 kilocycles, is shown in Fig. 14. The transistor is biased at cutoff and is normally nonconducting. Owing to coupling

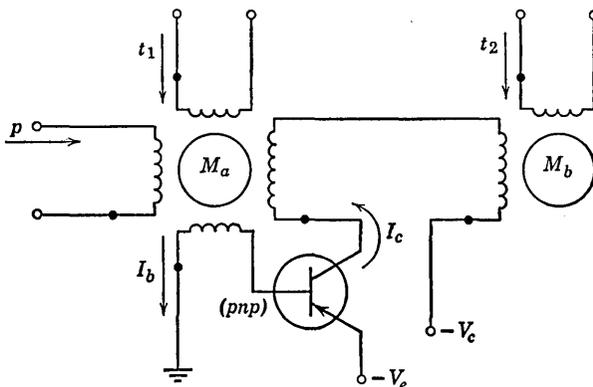


FIG. 14. Transistor magnetic core information transfer loop.

through the core, a fractional microsecond current pulse ( $t_1$ ) is sufficient to initiate a regenerative switching process, provided  $M_a$  contains a 1. When switching of the cores is completed, the feedback loop gain falls below unity and the transistor again ceases to conduct. Reset of  $M_a$  to 1 results in the appearance of a positive voltage at the base, tending to drive the transistor further into cutoff. The buffering action of the transistor eliminates backward flow of information in the loop (noise) and permits multiple branching between cores without difficulty.

The loop of Fig. 15 illustrates the addition of a diode for improved stability and an intermediate storage capacitor to enable both readout

and readin of core  $M_b$  to take place within the same pulse time. Current pulse  $t$  switches both cores (if previously in state 1) to 0, and at the same time the inhibit pulse maintains the transistor cutoff. At the termina-

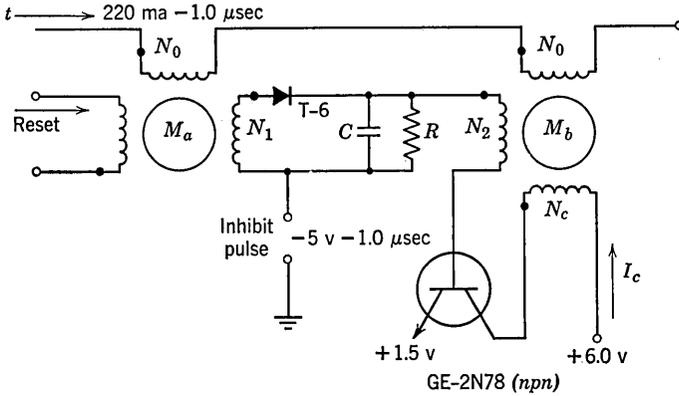


Fig. 15. Magnetic core transistor information transfer loop with a diode for improved stability. Circuit values, 200-ke operation:  $N_0$ , 5 turns;  $N_1$ , 15 turns;  $N_2$ , 10 turns;  $N_c$ , 30 turns;  $C$ , 0.001  $\mu$ f;  $R$ , 4.7 k $\Omega$ .

tion of the pulses, the voltage across the capacitor is sufficient to initiate switching of  $M_b$  to 1. Operational frequencies up to 500 kilocycles are possible by the use of fast surface barrier transistors and very low flux cores.

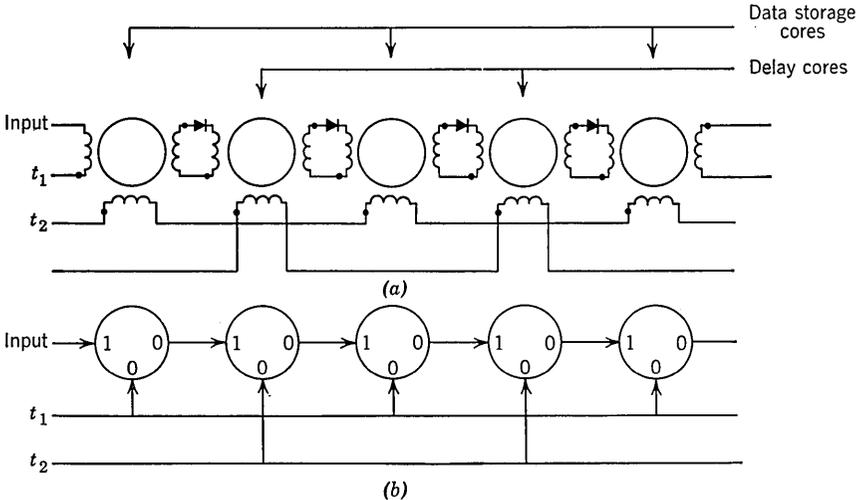


Fig. 16. Shift register, serial input, and output. Cores per binary digit of register capacity, 2; number of timing pulses per cycle, 2; direction of shift, right; data insertion, one binary digit between  $t_1$  pulses. (a) Schematic; (b) logical.

4. MAGNETIC SHIFT REGISTERS

Shift registers for handling binary data are synthesized by means of magnetic cores and transfer loops. By suitable design and choice of

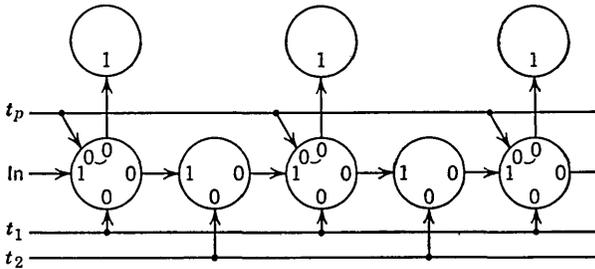


FIG. 17. Shift register, serial input-parallel output. Similar to Fig. 16 with addition of conditional transfer loops. Parallel read-out accomplished by  $t_p$  applied as required at times other than  $t_1, t_2$ , or during input.

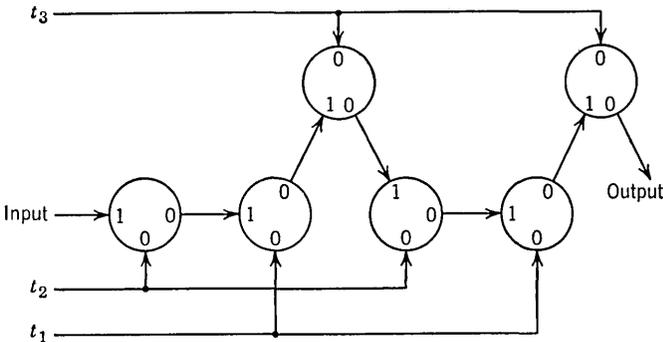


FIG. 18. Serial shift register, reduced core complement. Cores required, 3 for each 2 binary digits of register capacity; number of timing pulses per cycle, 3; direction of shift, right. The use of  $n + 1$  cores and drivers for each  $n$  binary digits of register capacity may be extended indefinitely.

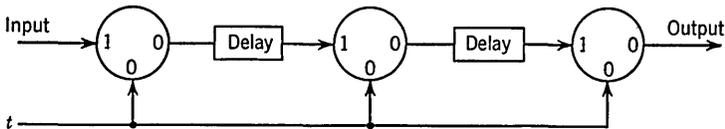


FIG. 19. Serial shift register, with delay network. Cores per binary digit of register capacity, 1; number of timing pulses per cycle, 1; direction of shift, right.

coupling determined from preliminary logical diagrams, registers may be constructed that (a) accept information serially or in parallel, (b) step the recorded sequence along the register in either direction, and (c) furnish a serial or parallel output. One complete timing cycle

$(t_1, t_2, \dots, t_n)$  accomplishes a shift by one binary place, for example, 100100 to 010010 in a six-bit register. Figures 16 to 20 show various forms of shift registers.

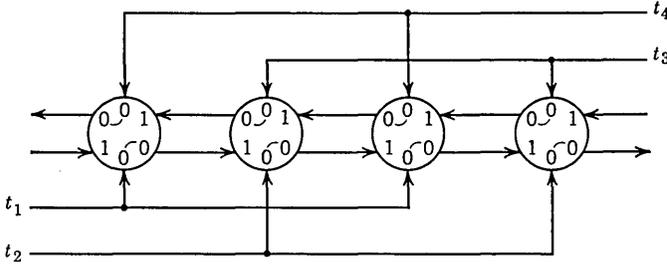


FIG. 20. Serial shift register, reversible. Cores per binary digit of register capacity, 2; number of timing pulses per cycle, each direction, 2; direction of shift, right or left.

5. LOGICAL FUNCTION ELEMENTS

Bistable magnetic cores are combined with conditional and unconditional transfer loops in a variety of unit arrangements capable of performing the basic logical operations of *and*, *or*, *exclusive or*, and *negation*.

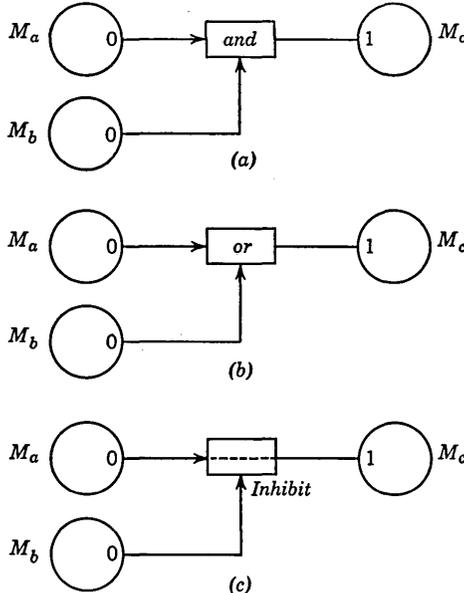


FIG. 21. *And*, *or* and *inhibit* logical symbols. Requirements for input to  $M_c$ : (a) *and* ( $\cdot$ ), coincident outputs from  $M_a$  and  $M_b$ ,  $M_a \cdot M_b$ ; (b) *or* ( $+$ ), output from  $M_a$  or  $M_b$  or coincident  $M_a$  and  $M_b$ ,  $M_a + M_b$ ; (c) *inhibit*, output from  $M_a$  only; transfer inhibited by coincident output from  $M_b$ .

By means of these elements, the more complex digital data processing functions can be synthesized (see Ref. 7).

The unit functions in many instances involve the combination or mixing of the outputs of a number of cores. *And*, *or*, and *inhibit* (negate) symbols are required to indicate the nature of the combination and to assist in the interpretation of the logical diagrams. The symbols, illustrated in Fig. 21, are employed only when the particular function is to be obtained by a single transfer loop in the implementing physical circuit.

**And Circuits.** (Conjunction or union, see Vol. I, Chap. 11.) Figure 22 shows block diagrams for an *and* circuit implementation for three

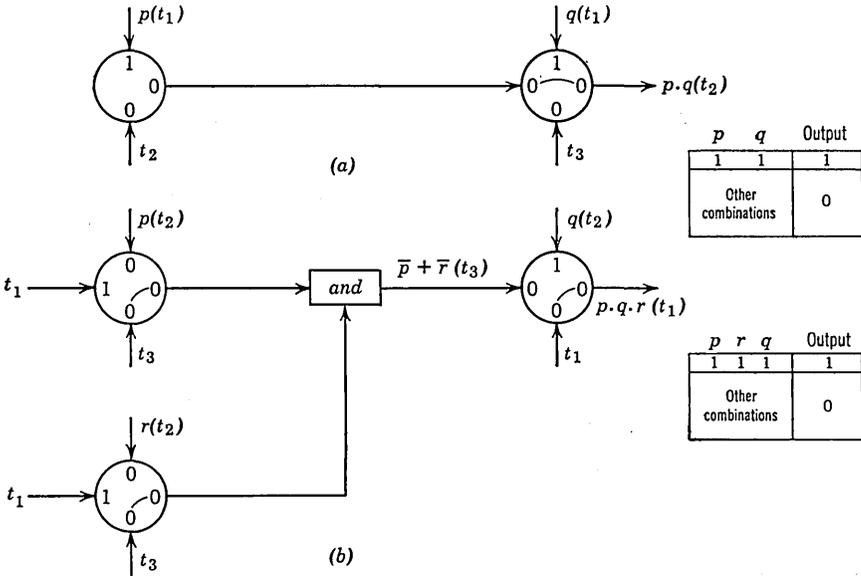


FIG. 22. *And* circuits, truth tables and block diagrams for (a) two inputs and (b) three inputs.

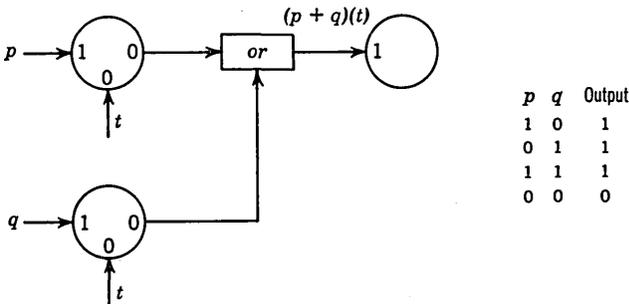


FIG. 23. *Or* circuit and truth table.

inputs and two inputs. Note that for three inputs an intermediate or circuit is used. The truth tables are also given.

**Or Circuits.** (Intersection or inclusive disjunction, see Vol. I, Chap. 11.) Figure 23 shows an or circuit with the corresponding truth table.

**Exclusive or Circuits.** (Symmetric difference or exclusive disjunction, see Vol. I, Chap. 11.) Figure 24 shows a block diagram for an exclusive or circuit. A circuit diagram and truth table is also given in Fig. 24.

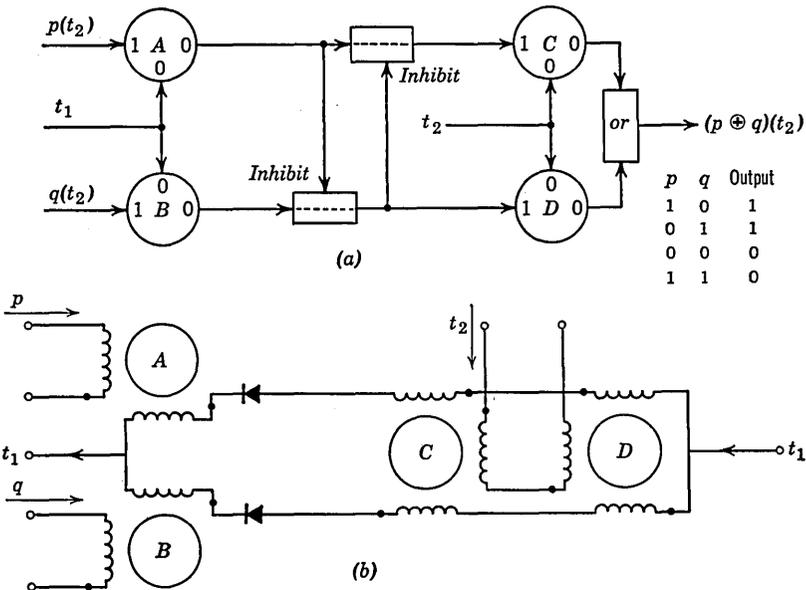


FIG. 24. Exclusive-or ( $\oplus$ ) circuits. (a) Block diagram and truth table. (b) Circuit diagram (output winding, C and D, not shown).

**Negation.** The circuit shown in Fig. 25 provides a means for obtaining the complementary functions of other elements. Output  $\bar{p}$  occurs in consequence to absence of input  $p$  and vice versa. For example,

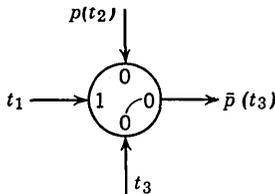


FIG. 25. Negation circuit.

the *material equivalence* function can be realized by negating the output of an *exclusive or* circuit, i.e., combining the logical elements of Figs. 24 and 25.

*Material Equivalence Function*

Inputs		Output
<i>p</i>	<i>q</i>	
1	0	0
0	1	0
0	0	1
1	1	1

## 6. MAGNETIC CORE STORAGES

Important use of bistable magnetic cores is found in storage, or memory, devices of digital data processing systems. Such devices may have very short access times, require no power to maintain storage of information, and are not dependent upon mechanical movement of parts for operation. Circuits and techniques of the present chapter are directly applicable in a number of arrangements.

*Note.* See also the coincident-current magnetic core storage, Chapter 19.

**Magnetic Shift Registers.** These registers serve often to provide buffer storage. Information from a tape, drum, or other source can be inserted in the register as available, and delivered to the output whenever desired. Because of input and output versatility and high speed of operation, these registers are further employed in transformation of mode of data transmission (serial to parallel, and vice versa), and matching of systems components of different repetition rates.

*Note.* Refer to Figs. 16 to 20 for technical details of this type of register.

**Large-Scale Static Storages.** Figure 26 illustrates a method by which a storage of desired word capacity may be realized with a single core and diode per bit. Insertion and extraction of data are accomplished in parallel (Ref. 8). A write pulse, applied to a selected row (address) of cores, inserts 1's in those digit positions where an appropriate voltage, impressed at the control point, enables diode conduction. Reading is accomplished by interrogating all cores of a selected address. In this operation, a voltage appears at each control point where the corresponding core switches to 0 from the 1 state.

**Selection Circuits.** Selection of data location (address) in a storage system involves switching circuits that are digital in character. The basic requirement of such a network is that each of the outputs, equal to the number of word locations, may be obtained by a prescribed com-

bination of input signals. Suitable circuit arrangements are numerous and may employ such components as semiconductors, electron tubes, or magnetic cores. In general, the number of switching functions is equal to the total possible input combinations, or  $2^N$ , where  $N$  is the number of input variables.

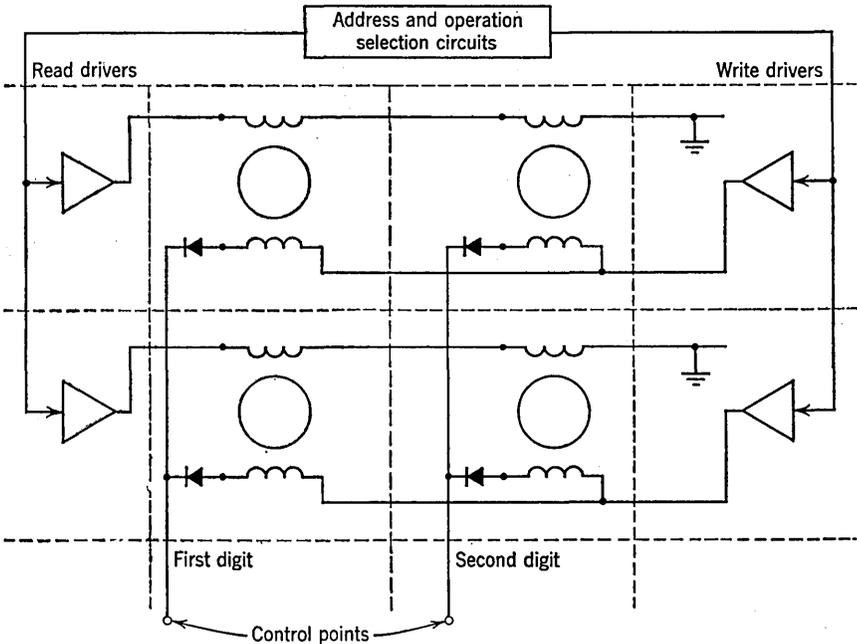


FIG. 26. Two-by-two magnetic storage (see Ref. 8).

Figure 27 illustrates a logical configuration by which four distinct outputs are obtained by combinations of the presence and absence of two inputs. The *and* circuit of each channel, shown functionally, requires the simultaneous arrival of two signals to generate an output. The occurrence of each of these signals in turn is dependent upon a specific condition of input to the core by which it is produced, e.g.,  $A$ ,  $\bar{A}$ ,  $B$ , or  $\bar{B}$ .

*Note.* Cf. Fig. 22.  $\bar{A}$  and  $\bar{B}$  symbolize the absence (or negation) of the respective inputs, as opposed to their electrical presence. In terms of Boolean algebra, each of the four switching functions may be expressed in the form  $Z_1 = AB$ ,  $Z_2 = \bar{A}B$ , etc.

Two different core arrangements provide for sensing of the presence or absence of signal input, the latter being accomplished by the principle of negation. The symbols noted at each output in the figure indicate the input combination that will gate the particular network.

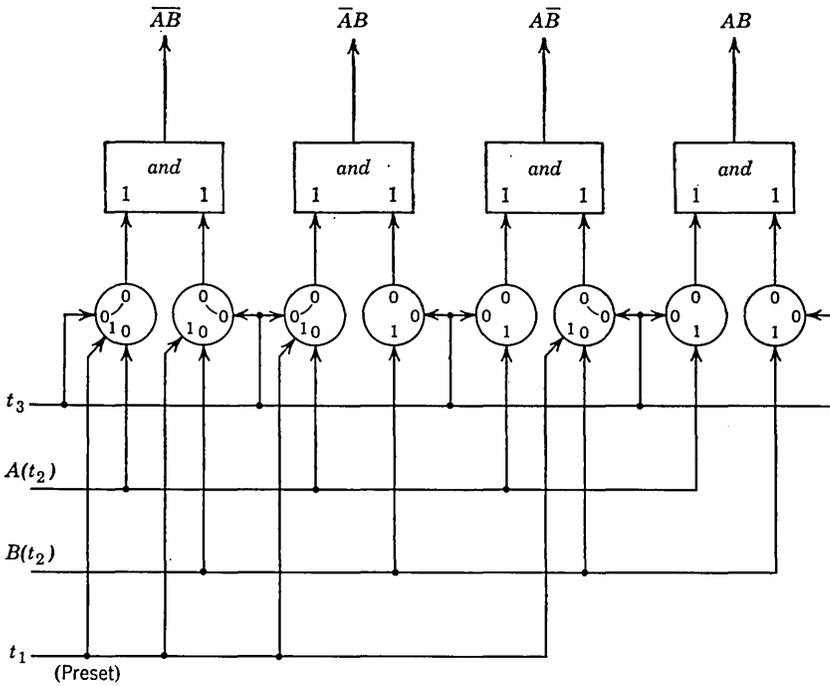


FIG. 27. Illustration of a logical circuit, four outputs for inputs  $A$  and  $B$ .

### 7. TIMING CONTROL CIRCUITS

In general, magnetic core functional units require the application of a number of unconditional signals in the execution of a single logical operation. Depending upon the circuitry employed, these signals trigger driving elements, or drive the cores directly to effect switching in an ordered sequence.

**Magnetic Cycle Distributor.** Figure 28 illustrates a cycle distributor frequently employed as a multioutput signal generator for timing and

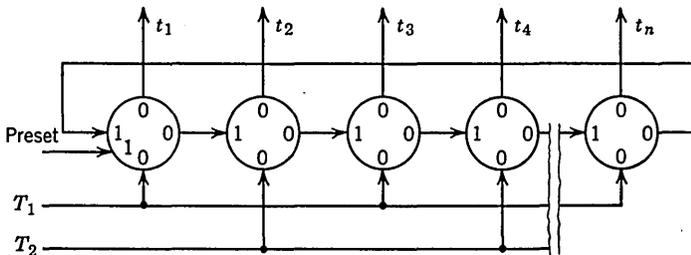


FIG. 28. Cycle distributor for timing control circuits.

control signals. Following initial preset, a time signal is derived on a separate control line each time the signal 1 is advanced one position along the closed end chain in response to  $T_1$  or  $T_2$ . These clock pulses usually are obtained from an external oscillator, such as a continuously running multivibrator.

Unequal spacing of signals in the output pattern may be achieved by disregarding appropriate cores in the control line connections. These cores assume simply the function of delay, but must be included in time and frequency considerations. For example, in the configuration shown, the repetition frequency of each of the line signals is  $2/n$  that of  $T_1$ , where  $n$  is the total number of cores.

## 8. ARITHMETIC AND MISCELLANEOUS APPLICATIONS

It previously has been shown how bistable magnetic cores and the basic transfer loops can be arranged to realize most of the essential functions of a digital data processing system. These functions have

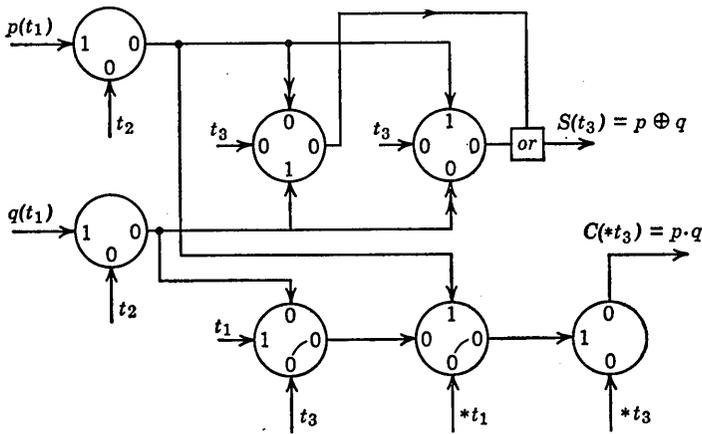


FIG. 29. Serial binary half adder. Serial input of digits of two binary numbers with in-phase entry of corresponding orders, starting with the least significant. *Note.* The last core has been added so that the carry output occurs at the same clock time as  $S$  but delayed by one cycle. This is necessary in order that the carry generated during any one cycle can be added to the sum bit generated during the next cycle when two half adders are placed in cascade to form a full serial binary adder.

included storage, delay, control, and the fundamental logical operations. With these building blocks and the principles delineated, virtually any function can be performed with the efficiencies characteristic of the new techniques. The diagrams of Figs. 29 through 31 are representative of the combinative possibilities.

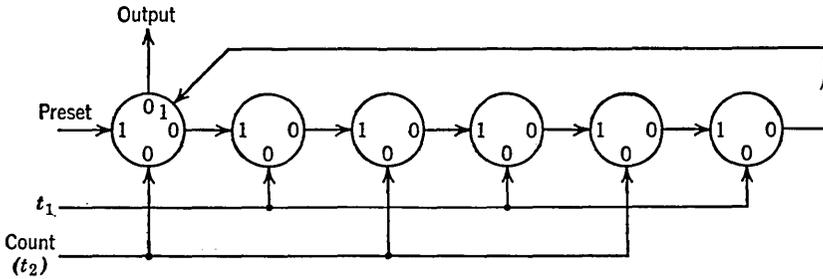


FIG. 30. Mod 3 ring counter. *Note.* This arrangement can be used in general to obtain any count. The number of cores required in each case is twice the count.

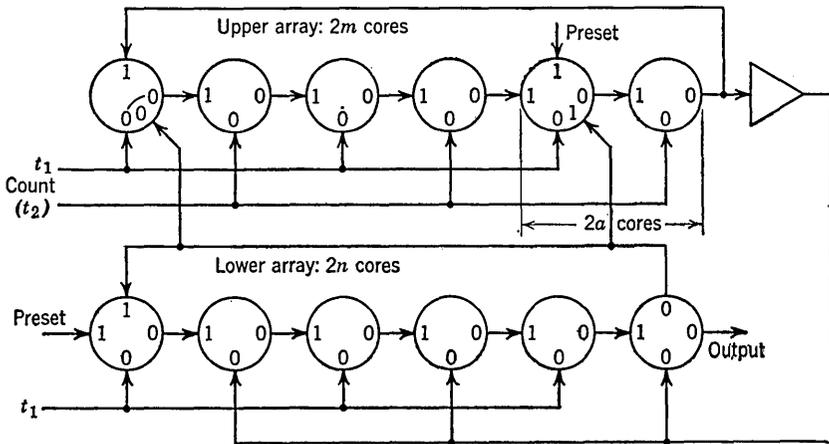


FIG. 31. Mod 7 counter, illustrating use of cascaded distributors and feedback techniques. *Note.* For long counts, this method requires fewer cores than in the preceding arrangement. A count of  $(n - 1)m + a$  is obtained with  $2(m + n)$  cores.

**Nondigital Applications** (Ref. 9). Although the present chapter is devoted mainly to digital processes, the use of bistable magnetic cores is not so limited, and numerous nondigital applications are to be found. The magnetic cycle distributor of the preceding paragraph, employed as a multioutput signal generator, is an example; others include devices for automatic phase control, and control of pulsing circuits to recording heads in a unique matrix printing process.

### 9. DRIVERS FOR MAGNETIC CORE CIRCUITS

Except for the more recently developed transfer loops containing active elements (transistors), pulse power from an external source is required to drive magnetic core networks and is commonly derived from electron

tube type generators. Vacuum and gas tubes, transistors, and magnetic amplifiers all are used for driving magnetic core circuits. A detailed description of the magnetic and transistor drivers lies outside the scope of the present chapter.

Constant-current sources are most commonly used, to simplify the analysis and the loop design equations. The pulses sometimes may be nearly rectangular, but more often are approximated by trapezoidal functions. Driver current amplitude must be stable over the range of impedances presented by the variable load conditions.

Readout drivers for magnetic core circuits include vacuum tube pentodes triggered by appropriate pulses, blocking oscillator type circuits, and self-extinguishing thyatron circuits. The last two can be triggered by the output from a single core and may be used as amplifiers where it is required that one transmitting core switch a large number of receiving cores. Frequencies of 500 kilocycles are obtainable without difficulty except for the thyatron drivers which are generally limited to 5 kilocycles due to deionization time or pulse-forming network delays.

Considerations in the development of transistor drivers are that, for good efficiency as a series switch, the collector saturation voltage should be low, and to drive many cores, the collector voltage and dissipation ratings should be high. As high-power junction transistors and driver circuits using them to best advantage are developed, these drivers may be used increasingly in place of electron tube drivers, in order to advance the reliability and reduce the size and power requirements of magnetic core data processing systems.

## REFERENCES

1. A. J. Meyerhoff, Application of magnetic cores to digital devices, *Auto. Con.*, April, May, and July, 1956.
2. W. Keister, A. E. Ritchie, and S. H. Washburn, *The Design of Switching Circuits*, Van Nostrand, Princeton, N. J., 1951.
3. R. D. Kodis, S. Ruhman, and W. D. Woo, Magnetic shift register using one core per bit, *Convention Record I.R.E.*, Pt. 7, 1953.
4. A. J. Meyerhoff and R. M. Tillman, A high-speed two-winding transistor-core oscillator, *I.R.E. Trans. on Circuit Theory*, Transistor Circuits Conference, February 15, 1956.
5. S. S. Guterman and W. M. Carey, Jr., A transistor-magnetic core circuit; a new device applied to digital computing techniques, *Convention Record I.R.E.*, Pt. 4, 1955.
6. H. Epstein and F. Innes, The electrographic recording technique, *Convention Record I.R.E.*, Pt. 4, 1955.

7. I. L. Auerbach and S. B. Disson, Magnetic elements in arithmetic and control circuits, *Elec. Eng.*, **74**, Pt. 2, 766-770, September, 1955.
8. I. L. Auerbach, A static magnetic memory system for the ENIAC, *Proc. Assoc. for Computing Machinery*, 213-222, May, 1952.
9. W. Miehle, J. Paivinen, J. Wylen, and D. Loev, BIMAG circuits for digital data processing systems, *Convention Record I.R.E.*, **3**, Pt. 4, 70-83, 1955.



## Transistor Circuits

*Isaac L. Auerbach*

1. Introduction	16-01
2. Transistor Switching Properties	16-02
3. Direct-Coupled Transistor Switching Circuits	16-05
4. Point-Contact Transistor Pulse Amplifiers	16-15
5. Transistorized Calculator	16-20
References	16-30

### I. INTRODUCTION

Transistors can be used to perform practically any of the basic gating, storage, and control functions in digital computers. As a logical element the transistor has the following advantages:

1. The transistor life span is several orders of magnitude greater than that of vacuum tubes.

2. The transistor because of switch contact like parameters may be d-c coupled. They may also be connected in parallel and series. This type circuitry can be exceptionally simple.

3. By using *npn*'s and *pnp*'s together in circuits possessing complementary symmetry, great design flexibility can be obtained.

4. High efficiency is possible in switching.

5. The transistor provides considerable economies in size, weight, and power requirements.

This chapter presents the design approach and basic circuits for three demonstrated techniques of transistor digital computer design: (a) direct-

coupled transistor logic (DCTL) or switching circuits, (b) point-contact transistor pulse amplifiers with diode gates, and (c) transistor circuitry connected by d-c paths with diode clamping. In each case a computer based on the technique is described; they are, respectively, (a) Transac (Philco, Ref. 1), (b) Tradic Phase 1 (Bell Telephone Laboratories, Ref. 2), and (c) IBM 608 (Ref. 3). The essential features of the machines are summarized in Table 1.

Particular attention is given to the logic and control circuit techniques. Storage techniques and auxiliary units such as clock supplies are covered elsewhere in the handbook.

## 2. TRANSISTOR SWITCHING PROPERTIES

**Transistor Types.** Point-contact transistors were used for Tradic because of their availability at the time Tradic was being developed. Of more general current interest are the various types of junction transistors. To a large extent the differences are determined by the method of manufacture. The various types are commonly referred to as: grown junction, alloy junction, surface barrier, diffused junction or graded base junction.

The diffused junction transistor is manufactured by a combination of techniques. Because of its high switching speed it is excellent for computing use.

### Switching Characteristics of Transistors

The junction transistors exhibit switching properties that may be likened to those of a relay: the base-to-emitter voltage or current may be considered as the actuating factor, and the collector-to-emitter current or voltage may be regarded as the output signal. The switching properties are given below.

**Signal Path Resistance When "Open."** For each of the transistors listed above the resistance in the operating region is measured in terms of hundreds of thousands of ohms or megohms.

**Signal Path Resistance When "Closed."** The alloy and surface-barrier transistors have "closed" resistances of 1 to 5 ohms, when driven to saturation; grown junctions usually have greater resistance.

**Speed of Switching.** The speed at which the transistor can be turned on (the switch closed) ranges from 10 microseconds for low-frequency alloy junction switching types to less than 10 millimicroseconds for surface-barrier transistors. The turn-on speed is a function of the application. See Refs. 4 and 5.

Turn-off time, i.e., opening the switch, runs from many microseconds to millimicroseconds, depending upon the particular circuits and voltages

TABLE 1. SUMMARY OF CHARACTERISTICS OF THREE TRANSISTOR COMPUTERS

	Power Requirements, Watts	Transistors	Diodes	Supply Voltages	Reliability	Circuit Philosophy
Transac <sup>a</sup> (DCTL)	100	2500 (surface barrier)		-3	Transistor failure of 0.01%/1000 hrs	Direct-coupled transistor logic. 0.25- $\mu$ sec delay per flip-flop stage
Tradic: phase 1 computer	75	700 (point contact)	11,000	6 1 0.5 -2 -8	15,000 hrs operation. Transistor failure of 0.07%/1000 hrs. No failure of capacitors, resistors, and diodes	Diode logic synchronous pulse amplifier 1-megacycle 4-phase clock
IBM 608 <sup>b</sup> transistor calculator	310	2165 (junction)	3600	15 -5 -8 -15	5% of transistors replaced under adverse operating conditions	Diode gating in conjunction with a 50-kc clock

<sup>a</sup> Housed in a 2 cu ft cabinet. Uses 780 resistors and 100 capacitors.

<sup>b</sup> Uses 95% less power than vacuum tube model.

used, even for the same transistor type. The turn-off time is related to storage time and decay time. These switching times are related to emitter and collector currents,  $\alpha$ , and  $\alpha$  cutoff frequency.

**Logical Gain Characteristics.** The ability of one transistor switch to drive many others is related to the direct-current gain, or "switching  $\beta$ " and to the leakage current,  $I_{co}$ .

The logical gain or treeing factor may run from 5 to about 15, depending upon the particular circuit and transistor. With each type of junction transistor listed above, the ability to drive others without buffer amplifiers decreases as they are pushed to their speed limit. At the current state of the art, the surface barrier affords somewhat lower logical gain, but greater frequency response, than the others.

**Reliability Characteristics.** Transistor characteristics vary with temperature and with the operating voltage: the direct-current gain,  $\beta$ , decreases with temperature rise, and the leakage current,  $I_{co}$ , increases with increase in the operating voltage. Available evidence indicates that the life span of the germanium transistor is several orders of magnitude greater than the life span of the vacuum tube. Failure rates of less than 0.01 per cent per thousand hours have been observed in tests with several hundred transistors for several thousand hours in computer circuits at room temperature.

### Design Factors

The essential *circuit* design considerations for computers are speed, reliability, logical gain, noise immunity, and power dissipation. These factors are discussed below.

**Speed.** From an analysis of the factors determining turn-on time and turn-off time, an important basic principle can be stated:

*In any given transistor, fast turn on is accomplished by using high initial base current, or by preventing saturation; fast turn off is accomplished by using a high inverse base current in relation to the steady-state forward base current.*

Further speed up depends upon gain and cutoff frequency, margins for gain and time constants, and the operation in saturated or unsaturated conduction.

**Reliability.** Reliability is often built into computers by means of "worst case" design. In worst case design, the circuits are built to perform even though all parameters are at their "end-of-life" values. This criterion is very conservative since it is highly improbable that *all* end-of-life values will be reached simultaneously, and since, in general, the circuit will operate when one or two parameters have exceeded their assigned end-of-life values.

Circuit parameters that must be controlled in worst case design of transistor computers include: supply voltage tolerances, component tolerances, transistor parameter tolerances, and temperature tolerances. Among the principal transistor parameters are the direct-current gain,  $\beta$ , and the operating collector leakage current.

**Logical Gain.** The establishment of worst case tolerances dictates rules for permissible logical interconnections between transistors. As noted previously, logical gain is a function of the direct-current gain,  $\beta$ , and  $I_{co}$ . In driving bases in parallel, the worst case is high base resistance and low  $\beta$ . Statistical spread in base resistance adds to the problem; to turn on a high-resistance base in parallel with a low-resistance base, the latter is made to draw more current than necessary.

**Noise Immunity.** Consideration of noise generated within the machine and from external sources enters into the safety margins allowed in the worst case design. The principal external noise factor affecting transistor circuitry is radio-frequency radiation. Internal sources fall into two categories: proximity coupling and common impedance coupling. Proximity coupling denotes spurious electromagnetic and electrostatic fields; common impedance coupling includes the usual factors of power supply common impedances and ground plane currents.

**Power Dissipation.** Transistor switching power dissipation in both static and transient operation must be held within ratings. Manufacturers' specifications may indicate a pulsed dissipation rating higher than the average limits, but caution should be observed. Thermal time constants can be much shorter than the overall time constant.

In the saturation state, static collector dissipation is simply the product of collector voltage and collector current; in the cutoff state, collector dissipation is a function of leakage current and collector voltage.

### 3. DIRECT-COUPLED TRANSISTOR SWITCHING CIRCUITS

**Description.** The Transistor Automatic Computer (Transac), developed by the Philco Corporation, is designed for "real time" control problems involving as many as 3000 single address instructions. Provisions are made for some 30 inputs and 30 outputs. Inputs generally take the form of quantized physical variables, and outputs are registered by indicators and power servos. Operations are performed in parallel to obtain the requisite speed, and the machine has been made asynchronous to avoid undue complexity in its design. Direct-coupled transistor logic (DCTL) circuitry is employed for the arithmetic and arithmetic control functions.

### Direct-Coupled Transistor Switching Circuits

Transistors can serve as bistable elements, the two stable states being saturation and cutoff. When the transistor is at saturation, the collector-to-emitter voltage is very low; when the transistor is at cutoff, the collector-to-emitter voltage is very high. With a low enough voltage applied to the base of a transistor the transistor can be maintained at cutoff;

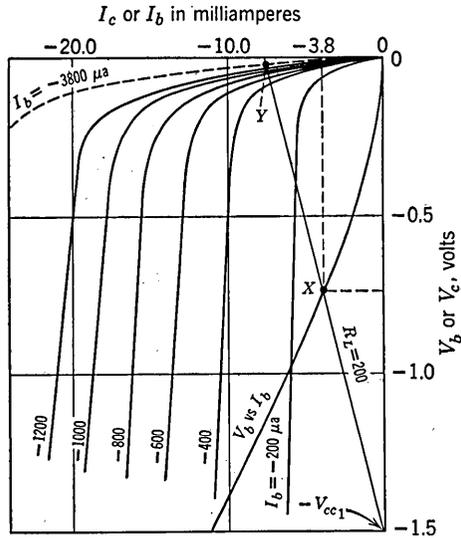


Fig. 1. Combined collector and base characteristics of surface barrier transistor.

with a high enough voltage applied to the base of a transistor, the transistor can be maintained at saturation. These characteristics make practical direct-coupled chains of common emitter switching circuits. In these chains, adjacent stages are in opposite states, and all stages switch when a signal is applied to the input stage.

Figure 1 shows the base characteristic and the collector characteristics for a surface-barrier transistor with a definite power supply and load resistance. Figure 2 shows three stages in a direct-coupled amplifier chain. If bias and leakage currents are minimized the load current of an *on* transistor (saturated) consists almost entirely of collector current (point Y in Figure 1); and the load current of an *off* transistor (cutoff) consists almost entirely of the base current of the transistor in the following stage (point X).

In DCTL some of the factors discussed in Sect. 2 are modified as described below.

**Logical Gain.** The logical gain in transistor circuits depends on  $\beta$  and  $I_{co}$ . For DCTL the gain also depends on the base-to-emitter resistance and the output impedance of the transistor in the closed position of the switch.

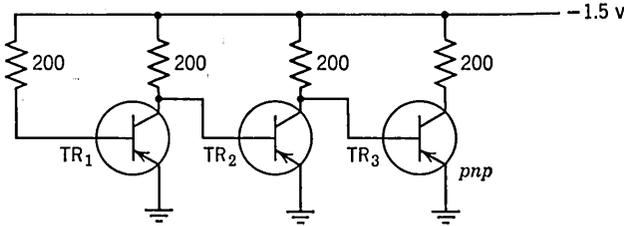


FIG. 2. Direct-coupled amplifier chain.

**Parameters in Worst Case Design.** In addition to all the parameters listed in Sect. 2, the base and collector voltages during the *on* and *off* states must be taken into account in worst case design in DCTL.

**Switching Speed.** DCTL provides fast turn on and turn off. Turn on is fast because a high initial base current is supplied the transistor through the load resistor of the previous stage. Turn off is fast because of the high reverse base current. The reverse base current is high because the base acts as a low negative voltage source of very low internal impedance, and because the base returns to ground through the low saturation resistance of the previous stage (now *on*).

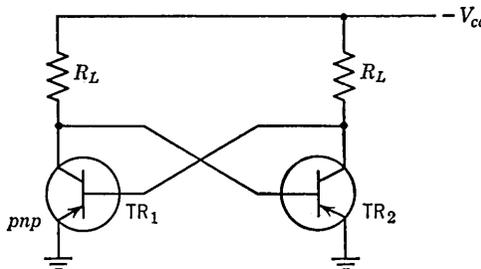


FIG. 3. Basic direct-coupled saturation flip-flop.

**Saturation Flip-Flop**

A closed two-stage chain, or saturation flip-flop, is shown in Fig. 3. In this circuit, the collector voltage of the cutoff transistor is negative, corresponding to point X in Fig. 1. This voltage is the same as the base

voltage of the saturated transistor. The collector of the saturated transistor is low or near ground, corresponding to point  $Y$ .

The flip-flop can be triggered to the opposite state by dropping the base voltage of the saturated transistor to or near ground. This causes the collector voltage to rise, energizing the transistor that was previously cut off. Two slightly different methods of triggering a saturation flip-flop

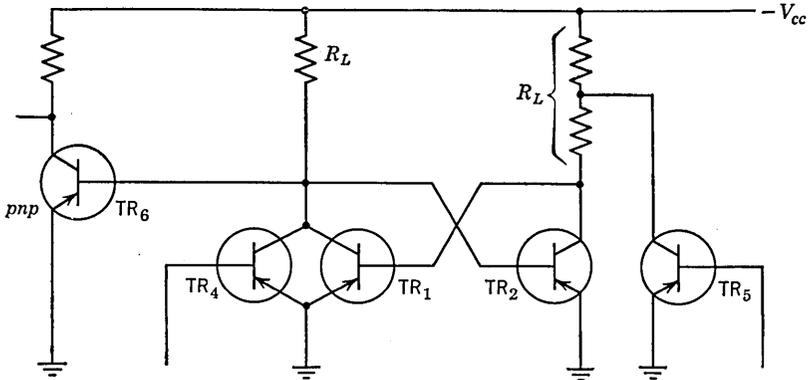


FIG. 4. Flip-flop triggering and sensing.

are shown in Fig. 4. A negative pulse of sufficient amplitude and duration applied to the base of TR<sub>4</sub> will saturate TR<sub>4</sub> and cut off TR<sub>2</sub>. A negative signal applied to the base of TR<sub>5</sub> will saturate TR<sub>5</sub> and cut off TR<sub>1</sub>. The connection of the TR<sub>5</sub> collector to a tap on the TR<sub>2</sub> load impedance permits the use of TR<sub>5</sub> in triggering other flip-flops at the same time by sharing the load impedance between the TR<sub>5</sub> collector and the power supply among all the flip-flops to be triggered simultaneously.

The state of the flip-flop can be sensed by connecting the input of a sensing element, such as TR<sub>6</sub>, to the appropriate base-collector-load tie point.

### One-Shot Multivibrator

A one-shot circuit or monostable flip-flop is essentially a flip-flop with an  $RC$  circuit inserted in one of the collector-to-base cross-coupling paths. Figure 5 shows a typical circuit. In the quiescent state, TR<sub>2</sub> is maintained saturated by the base current supplied through  $R_0$ . The collector of TR<sub>1</sub> is essentially at the supply voltage since negligible leakage current flows in TR<sub>1</sub> and TR<sub>3</sub>.

A negative trigger applied to the base of TR<sub>3</sub> causes it to conduct and charge capacitor  $C_0$ . The resultant positive going potential applied to

the base of  $TR_2$  causes  $TR_2$  to conduct less. This condition in turn makes the base of  $TR_1$  more negative, thus turning  $TR_1$  on. This action is cumulative:  $TR_1$  quickly reaches saturation and  $TR_2$  reaches cutoff.

As soon as this stable state is reached,  $C_0$  begins to discharge. After an interval of time determined by the circuit time constant, the voltage at the base of  $TR_2$  reaches the point where it causes  $TR_2$  to conduct

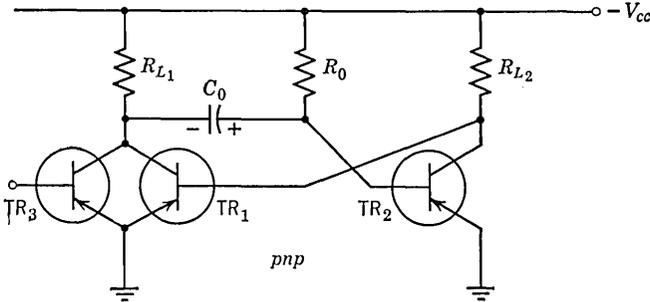


FIG. 5. Typical one-shot circuit.

again. Once  $TR_2$  starts conducting, cumulative action brings  $TR_2$  to saturation and  $TR_1$  to cutoff, and the circuit remains in this state until another trigger is applied.

At the collector of  $TR_2$  is produced a rectangular pulse, whose width is determined by the amount of time it takes  $C_0$  to bring the base potential of  $TR_2$  to the point where  $TR_2$  starts conducting. The width of the output pulse is independent of the width of the trigger. An approximation to the pulse width at the collector of  $TR_2$  is obtained by calculating the discharge time. The initial capacitor voltage is  $V_{cc} + (V_{cc} - V_{b2\text{ sat}})$ ; the final voltage is approximately  $V_{cc}$ ; and the time constant is  $R_0C_0$ . Therefore,

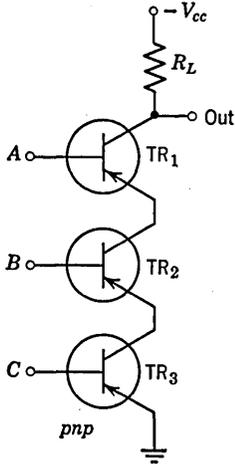
$$T = R_0C_0 \ln (2 - V_{b2\text{ sat}}/V_{cc}).$$

With SB-100 and similar transistors, this expression yields better than 5 per cent accuracy for pulse widths above 0.5 microsecond. For shorter pulses the rise time and fall time may add to the error. Other considerations include the exact values of saturation voltage and  $V_{b2}$  turn-on voltage, the effects of leakage currents, and variation of  $V_{cc}$  and  $R_0C_0$ .

### Gates

Various types of gates can be built: *and*'s and *or*'s, using direct-coupled transistors either paralleled or series-connected. General configurations are shown in Figs. 6 and 7. Depending upon whether a 1 at

any of the input bases is defined as a negative voltage or a near-zero voltage, the series circuit operates as an *and* or an *or*, and the parallel circuit functions as an *or* or an *and*. For convenience, let the negative



Parameters:

$$V_{cc} = -3 \text{ v}; R_L = 1 \text{ k}\Omega;$$

$$V_{out} = 100 \text{ mv}; \beta = 10$$

Currents when saturated:

$$I_{C1} = 2.900 \text{ ma}; I_{b1} = 0.290 \text{ ma},$$

$$I_{C2} = I_{R_L} + I_{b1} = 3.190 \text{ ma},$$

$$I_{b2} = 0.319 \text{ ma},$$

$$I_{C3} = I_{R_L} + I_{b2} + I_{b1} = 3.509 \text{ ma},$$

$$I_{b3} = 0.351 \text{ ma}.$$

FIG. 6. Representative series gate. Output,  $A \cdot B \cdot C$ .

input gate be termed a "negative gate" and the near-zero input gate be termed a "positive gate." The gates also invert the output signal.

**Series Gates.** If each base in Fig. 6 is negative in the absence of a 1 input, the collector load path through  $TR_1$ ,  $TR_2$ , and  $TR_3$  is then com-

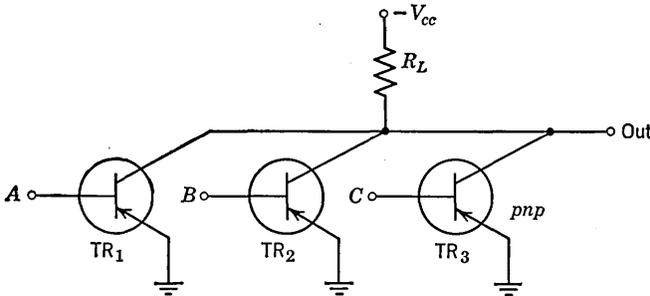


FIG. 7. Parallel gate. Output,  $A + B + C$ .

plete, and the output is "positive." If any or all of the bases go positive, the load path is interrupted and the output is negative. This is a "positive *or*" circuit. For the same circuit but reversing the 1 and 0 convention, the function is a negative *and*. The relationship among

currents in an *on* series stack are tabulated for a typical case in Fig. 6.

Design difficulty can result from the fact that the *on* output is the *sum* of the individual saturation voltages, and there is a maximum voltage above which the next stage will not be held off. Hence the number of series stages that can be stacked is limited, to not over two to four, using SB-100's, depending upon the worst case design stringency and operating parameters. The saturation voltage can be minimized by making  $R_1$  as large as possible, consistent with drive requirements.

There is a limit to the number of transistors that can be placed in series, because of base current adding. In Fig. 6, for instance,  $TR_3$  must be able to draw the base currents of  $TR_1$  and  $TR_2$  in addition to its own.

A disadvantage of the negative *and* circuit is that rise times of the series-connected transistors add together (in an rms manner), and thus slow their operation.

**Parallel Gates.** A parallel gate circuit is shown in Fig. 7. If the inputs are normally near ground, the output is negative; if any or all inputs are made negative, the output goes to ground. This constitutes "negative *or*" operation. Similarly, if the inputs are normally negative and the output near ground, all the inputs must approach ground to cause the output to swing negative, and the circuit functions as a "positive *and*."

The parallel arrangement can be used with more inputs than the series circuit since there is very little change of output levels or rise time as more stages are added. Ultimately, the number of stages that can be paralleled is dependent on the effective leakage current ( $I_{co}$ ) in the *off* state. Since all individual  $I_{co}$ 's flow through the common  $R_L$ , the  $(\Sigma I_{co})(R_L)$  drop may be sufficient to turn *off* the following stage. To some extent this can be counteracted by lowering  $R_L$ , but without exceeding the maximum  $I_c$  rating during conduction. The  $I_{co}$ 's increase appreciably with temperature, and the practical maximum number of inputs in parallel is approximately ten, for SB-100's operating at 3 volts supply and 40°C. Exact limits depend also on  $\beta$ , on  $R_L$ , and on the operating point, and number of stages driven.

### Combined Gates

Representative combined gate circuits performing addition are shown in Figs. 8, 9, and 10.

**Half-Adder.** All the gates in the half-adder circuit shown in Fig. 8 are negative. The complementary pairs of inputs,  $(A, \bar{A})$  and  $(B, \bar{B})$ , are the outputs of opposite sides of flip-flops; for error protection the states of both sides are carried through the logic gates. In like manner

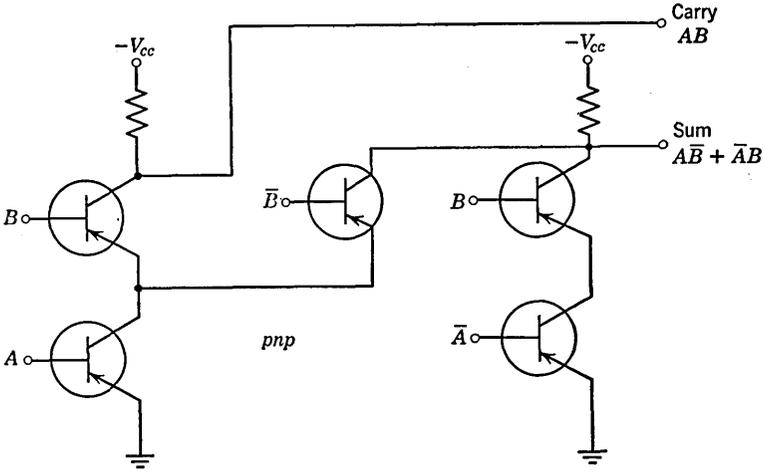


FIG. 8. Half-adder.

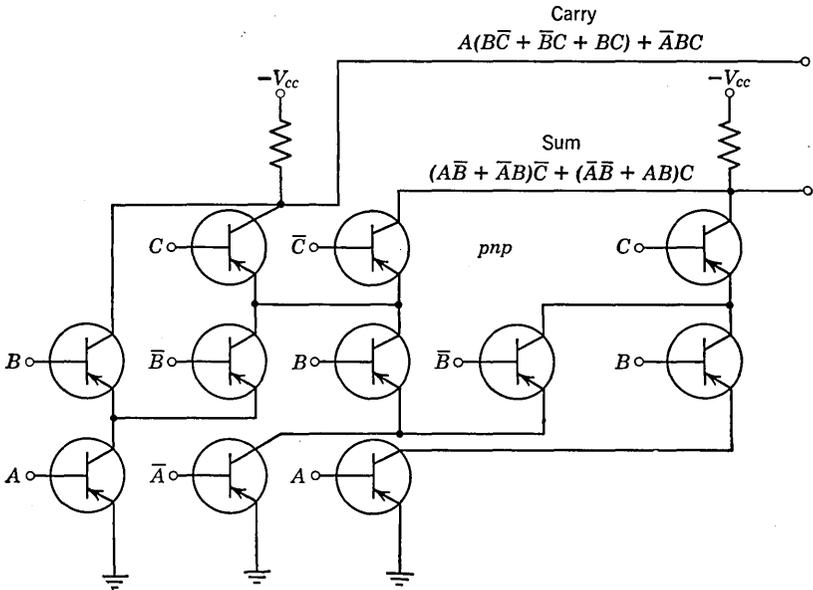


FIG. 9. Full adder.

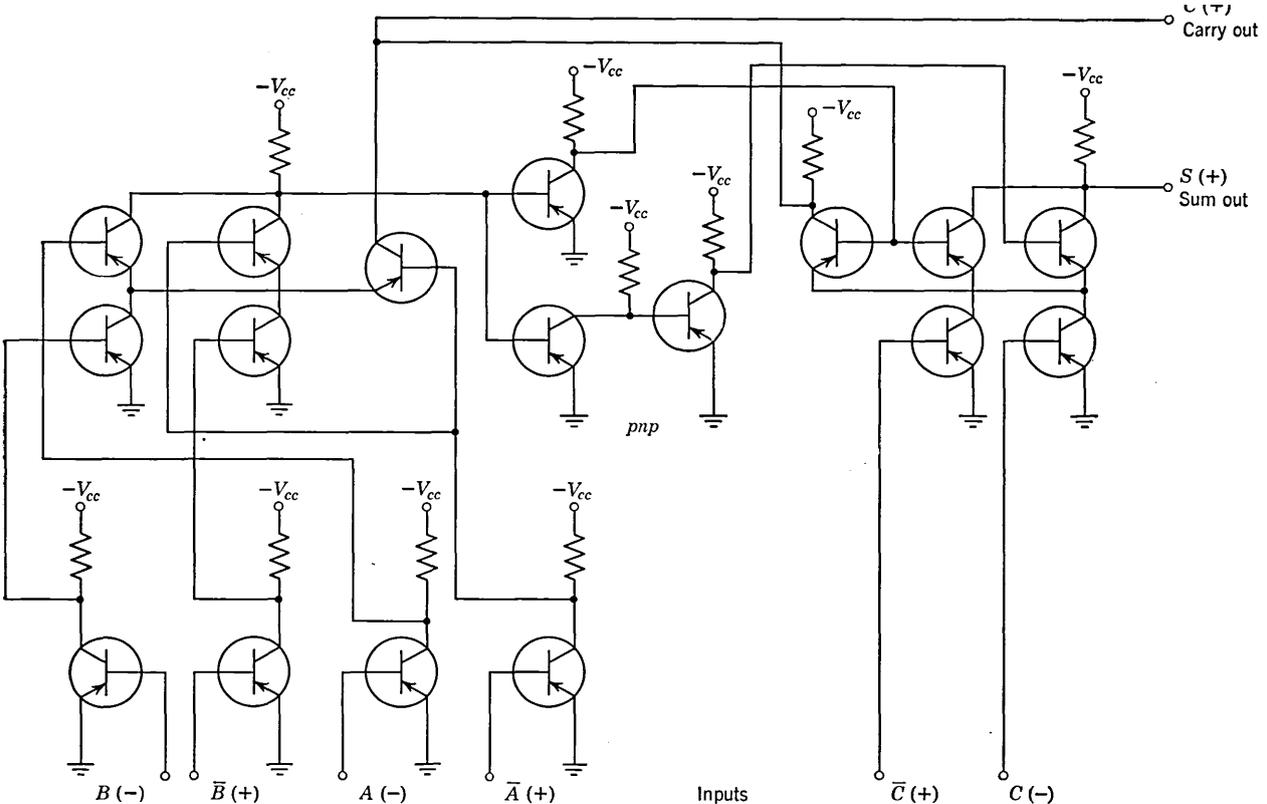


FIG. 10. Full adder with not over two transistors in series. Symbols: (-), complement; (+), indicated logic state corresponds to 0-volt level; (-), indicated logic state corresponds to minus volt level.

the single outputs shown may be supplemented by inverters to provide double outputs for each sum and carry.

**Full Adders.** Figure 9 shows a three-input or full adder. In Fig. 10 is given a special full-adder circuit with series stacks limited to two transistors in height. This restriction is derived, under a worst case design philosophy, from the limitations pointed out previously. Input buffer inverters are included in the full-adder circuit of Fig. 10. Throughout a computer, the inverter buffers may be used, singly or in trees, as a pulse distribution system.

**RC Speedup.** For some transistors the gates and buffers can be speeded in turn off by the addition of a parallel resistor-capacitor combination in the base. The same technique is applicable to the flip-flops. It is sometimes also advantageous to return the base to a positive bias voltage, and thus hold it off more firmly and improve noise rejection. The *RC* and bias are depicted in Fig. 11. (See also Design Factors under Sect. 2.)

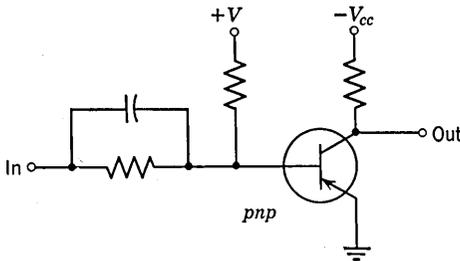


FIG. 11. Inverter with *R-C* speedup and base bias.

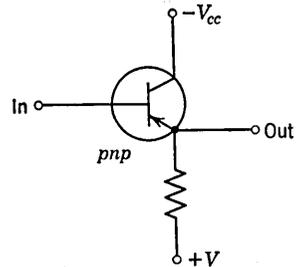


FIG. 12. Emitter follower.

**Emitter Followers.** The emitter follower, shown in Fig. 12, is a current amplifier that works in the nonsaturation region. This means that hole storage time has no effect on the speed. The use of the emitter follower is analogous to the use of cathode followers in vacuum tube computing circuits. The loss in voltage swing between input and output limits the number of emitter followers that can be put in a direct-coupled chain without compensation.

### Transac Packaging and Power Requirements

For the Transac computer designed and built by the Philco Corporation, packaging is in the form of plug-in cards containing from 50 to 70 transistors and their associated components. Particular attention has been paid to the problem of maintenance with unskilled personnel.

Exclusive of the input devices, indicators, and output converters, Transac requires approximately 2500 transistors, 700 resistors, and 100 capacitors. All transistors are surface barrier except for 72 power transistors in the storage. The computer can be economically contained in approximately 2 cubic feet and requires less than 100 watts, including 35 watts for driving the storage drum. For some environments the computer must be maintained below the ambient temperature; the 2-cubic-foot volume is cooled without difficulty.

#### 4. POINT-CONTACT TRANSISTOR PULSE AMPLIFIERS

**Description.** The Phase 1 Tradic computer, developed by the Bell Telephone Laboratories, was built to demonstrate the feasibility of a transistor computer for military airborne service. A binary serial machine with a complete arithmetic unit, it operates at a pulse repetition frequency of 1 megacycle per second. Point-contact transistors are used for pulse regeneration, semiconductor diodes for logic, and electromagnetic delay lines for storage and incidental delay.

**Active Element.** The elemental block around which the Tradic phase 1 computer is designed is the transistor pulse amplifier. This circuit contains a single point-contact transistor and associated resistors, capacitors, and diodes; its function is to reshape and retiming half-microsecond digit pulses at appropriate intervals in the logic networks. One amplifier will drive one to seven similar amplifiers, depending on the intervening logic.

The point-contact transistor is employed in a common base configuration to take advantage of its negative resistance emitter characteristic, shown in Fig. 13. This property, not generally shared by junction type transistors, occurs when the current gain of a transistor is greater than unity and the input and output signal voltages are in phase.

**Pulse Amplifier.** The  $RC$ -coupled pulse amplifier circuit is given in Fig. 14. Input, output, and clock waveforms are represented in Fig. 15. The circuit normally rests in the low-current state, where current flow through  $R_2$ ,  $X_2$ , and  $X_3$  holds the junction of the diodes at point  $A$  on the emitter characteristic, a voltage just below the peak point. A positive input signal (1) raises the emitter voltage to the negative resistance region. With sufficient bandwidth, the transistor is unstable in this region and snaps out on a load line provided by capacitor  $C_c$  to point  $D$ . The capacitor then discharges from  $D$  to  $B$ , at which point diode  $X_2$  conducts and the transistor is locked in its high-current state. The following positive clock pulse at the base in effect lowers the emitter voltage into the unstable region, and the circuit returns to its initial state.

Only the positive half-cycles of the clock sine wave appear at the base

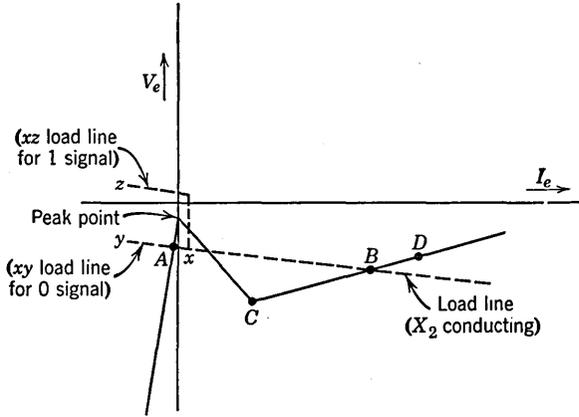
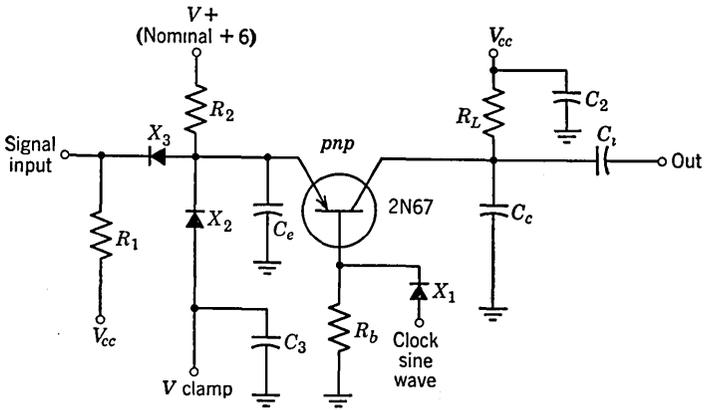


FIG. 13. Idealized point-contact emitter  $V$ - $I$  characteristics and operating load line



$R_1$	12 k $\Omega$	$V_{cc}$	-8 volts
$R_2$	22 k $\Omega$	$V$ clamp	-1 volt
$R_b$	470 $\Omega$	$V+$	+6 volts
$R_L$	470 $\Omega$	Clock	10-volt peak, symmetrical re- spect to ground
$C_1$	0.01 $\mu$ f coupling	$X_1$	Base diode
$C_2$	0.01 $\mu$ f filter	$X_2$	Emitter clamp diode
$C_3$	0.01 $\mu$ f filter	$X_3$	Input diode
$C_e$	15 $\mu$ f		
$C_c$	50 $\mu$ f		

FIG. 14.  $RC$ -coupled pulse amplifier circuit.

of the transistor, and these recur regularly at 1-megacycle rate. Input information signals are timed to arrive toward the end of the clock pulse, and the transistor triggers as the sine wave goes through ground potential. Since turn on and turn off are controlled, respectively, at the termination of one base pulse and the beginning of the next, the timing and duration

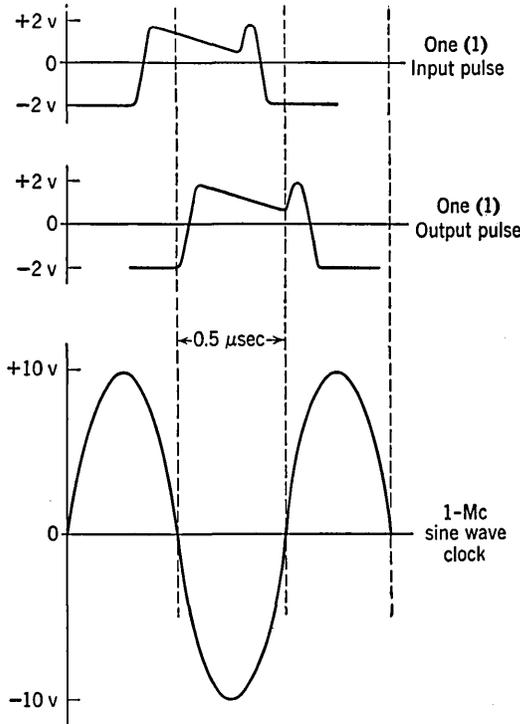


FIG. 15. Typical waveforms, Tradic packages.

of the square wave output is determined by the clock rather than the input signal. In the case of a 0 input signal (no signal) during a cycle of operation, the clock tends merely to drive the transistor further into cutoff, and no output occurs.

**Transformer Coupling.** Improved amplifier performance may be obtained with transformer coupling and the use of additional circuit components. The *RC* circuit, previously described, and the transformer-coupled circuit operate fundamentally in the same way. The circuit schematic of the basic transformer-coupled amplifier may be obtained from Fig. 16, the schematic of the four-input *or* package, by omitting diodes  $CR_{14}$ ,  $CR_{21}$ ,  $CR_{22}$ ,  $CR_{23}$ , and  $CR_{24}$ , and all but one of the signal

input leads. The outstanding advantages of the transformer-coupled amplifier are a higher and more uniform output signal level and a sizable reduction in clock input power.

**Phase I Packages.** For convenience, the amplifiers are packaged singly with the appropriate diode logic gates included. Each type of package has a simple function and can drive several packages in parallel. The *or*, *and*, *inhibit*, and storage packages are representative of those employed in the arithmetic unit of the computer. Schematics of the packages are shown in Figs. 16, 17, 18, and 19. A number of features are common to all packages.

*Strappable Load.* With the collector circuit heavily loaded, safety margins are improved due to deepening of the valley of the negative resistance curve. Diodes  $CR_{17}$  and  $CR_{18}$ , and resistor  $R_{10}$  are provided as a strappable load in order that packages may always be connected with a minimum of four loads.

*Bypass Capacitors.* Capacitors  $C_3$ ,  $C_4$ ,  $C_5$ , and  $C_6$  prevent transients through the common voltage supplies from affecting amplifier operation.

*Waveforms.* Typical input and output voltage waveforms relating to the functional packages are illustrated in Fig. 15.

**Four-Terminal Or Circuit.** The *or* circuit, shown in Fig. 16, will have an output if there is a signal (a positive pulse) on any of the input leads. The purpose of  $CR_{21}$ ,  $CR_{22}$ ,  $CR_{23}$ , and  $CR_{24}$  is to isolate the inputs from one another.  $CR_{14}$  is required because the series diode reduces the effect of the clamp of the driving package.

**Four-Terminal And Circuit.** The *and* circuit, shown in Fig. 17, will have an output only if there is a simultaneous signal on all its input leads. The logic circuit consists of resistors  $R_6$ ,  $R_7$ ,  $R_8$ , and  $R_9$  and diodes  $CR_7$ ,  $CR_8$ ,  $CR_9$ , and  $CR_{10}$ . All these diodes must be cut off if the amplifier is to trigger and provide an output signal. The absence of a signal on any one of the leads permits the corresponding input diode to clamp the transistor emitter below the peak point. Unused *and* inputs are connected to ground to provide a continuous signal at these terminals.

**Inhibitor Circuit.** The *inhibitor* circuit, shown in Fig. 18, will have an output if there is a simultaneous signal on both of the *and* input terminals, provided there is *no* accompanying signal on the inhibit terminal. Diodes  $CR_{31}$  and  $CR_{32}$ , together with  $R_{14}$  serve the dual purpose of providing damping for  $T_2$ , and of establishing a current threshold to give protection against spurious inhibition. In the quiescent state, both  $CR_{31}$  and  $CR_{32}$  carry small currents. The inhibiting signal must replace this current in  $CR_{31}$  and cut it off before it can cause  $CR_8$  to conduct and thus inhibit the package.

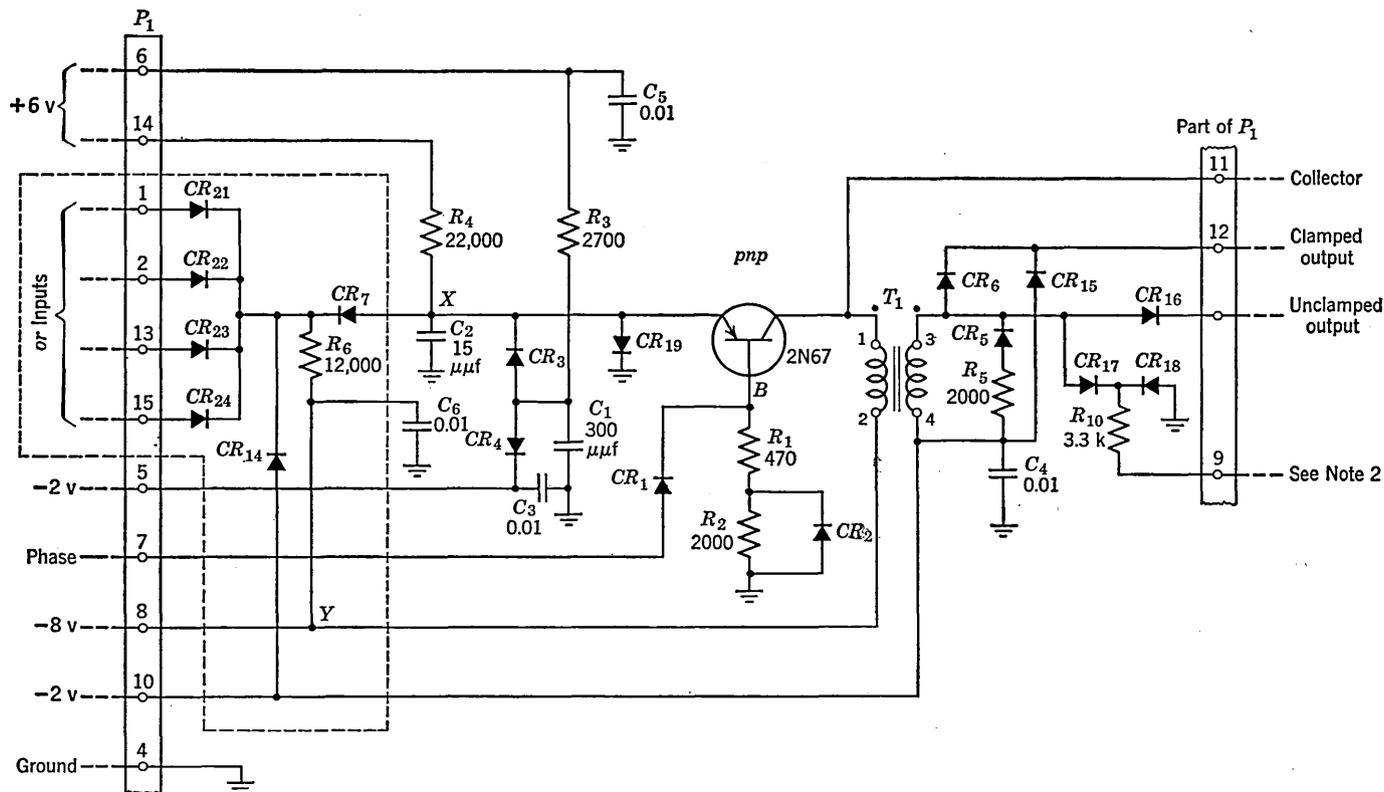


FIG. 16. Tradic four-input or circuit. Notes. (1) All values are expressed in ohms and microfarads unless otherwise indicated. (2) With one or two loads, strap 9 to 8. With three loads, strap 9 to 10. With more loads, leave 9 open.

The inhibit clock input circuit, consisting of  $CR_9$ ,  $CR_{22}$ , and  $R_8$ , is an additional *and* type input to prevent a false output should the input *and* signals last longer than the *inhibit* signal. The inhibit clock voltage, which lags the amplifier clock voltage by 90 degrees, goes negative at the time the input signals would normally disappear. This causes  $CR_9$  to conduct and clamp the emitter below the peak point.

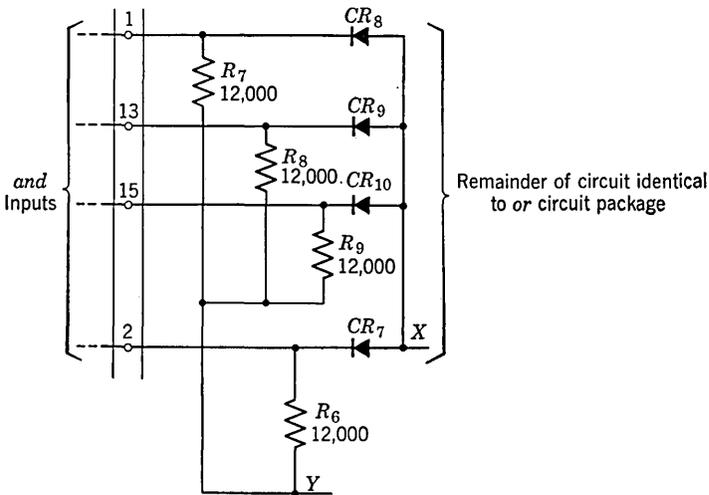


FIG. 17. Tradic four-input *and* circuit. Remainder of circuit identical to or circuit package.

**Storage Circuit.** When a signal is applied to the *Set 1* input of the circuit shown in Fig. 19, an output signal occurs  $\frac{1}{4}$  microsecond later, and every digit period thereafter until a signal is applied to the *Set 0* input. Simultaneous signals to both terminals result in no output signal.

An output signal is fed back through the delay circuit of  $T_3$ ,  $CR_{25}$ ,  $C_8$ , and  $CR_{24}$ , and into the amplifier input. The delay circuit is arranged so that the total loop delay is 1 microsecond, and recirculation of the pulse continues until the transistor emitter is clamped negative by the inhibiting *Set 0* input.

**Power Requirements.** The total power required, d-c and clock, is 75 milliwatts for a pulse amplifier and logic package. The requirements for the computer system, which contains approximately 700 transistors and 11,000 diodes, is less than 100 watts from all sources.

## 5. TRANSISTORIZED CALCULATOR

**Description.** This transistorized equipment is functionally identical to the IBM 604 electronic calculating punch, a vacuum tube machine in

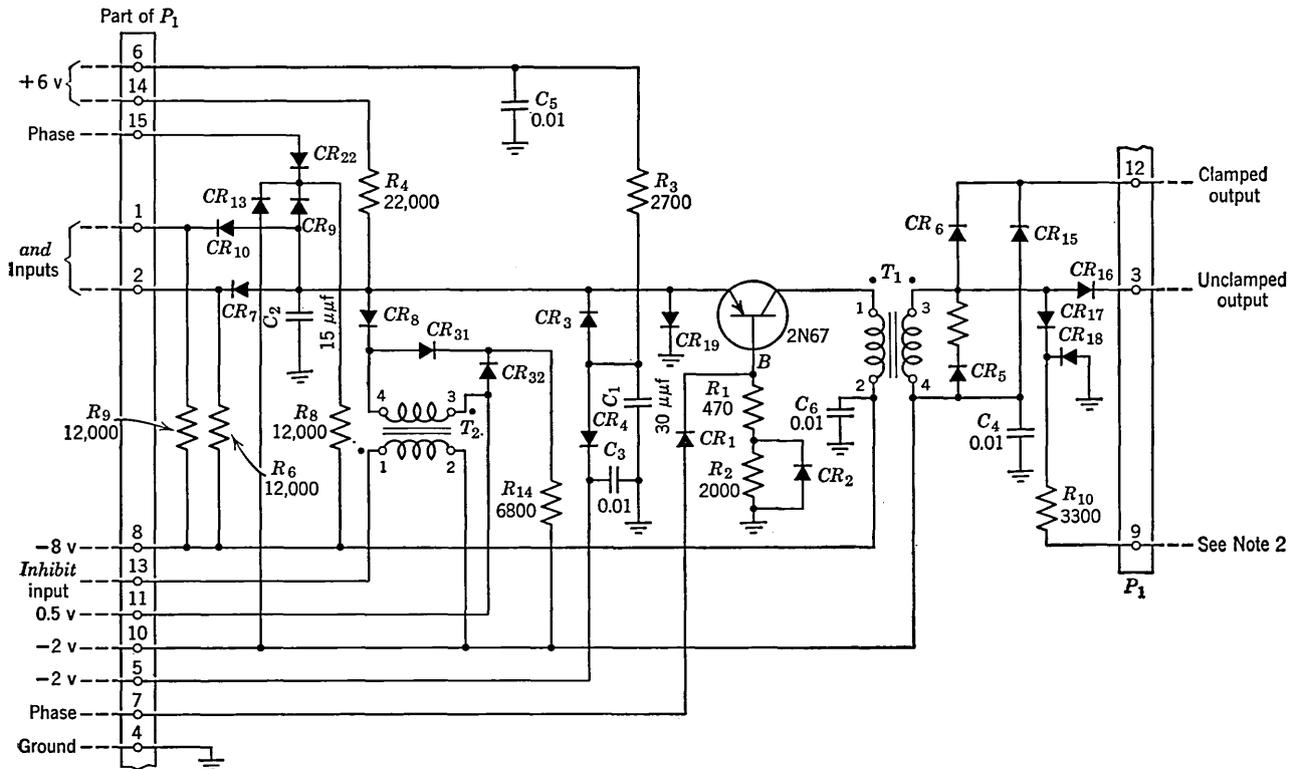


FIG. 18. Tradic inhibitor circuit. Notes. (1) All values are expressed in ohms and microfarads unless otherwise indicated (2) With one or two loads, strap 9 to 8. With three loads, strap 9 to 10. With more loads, leave 9 open.

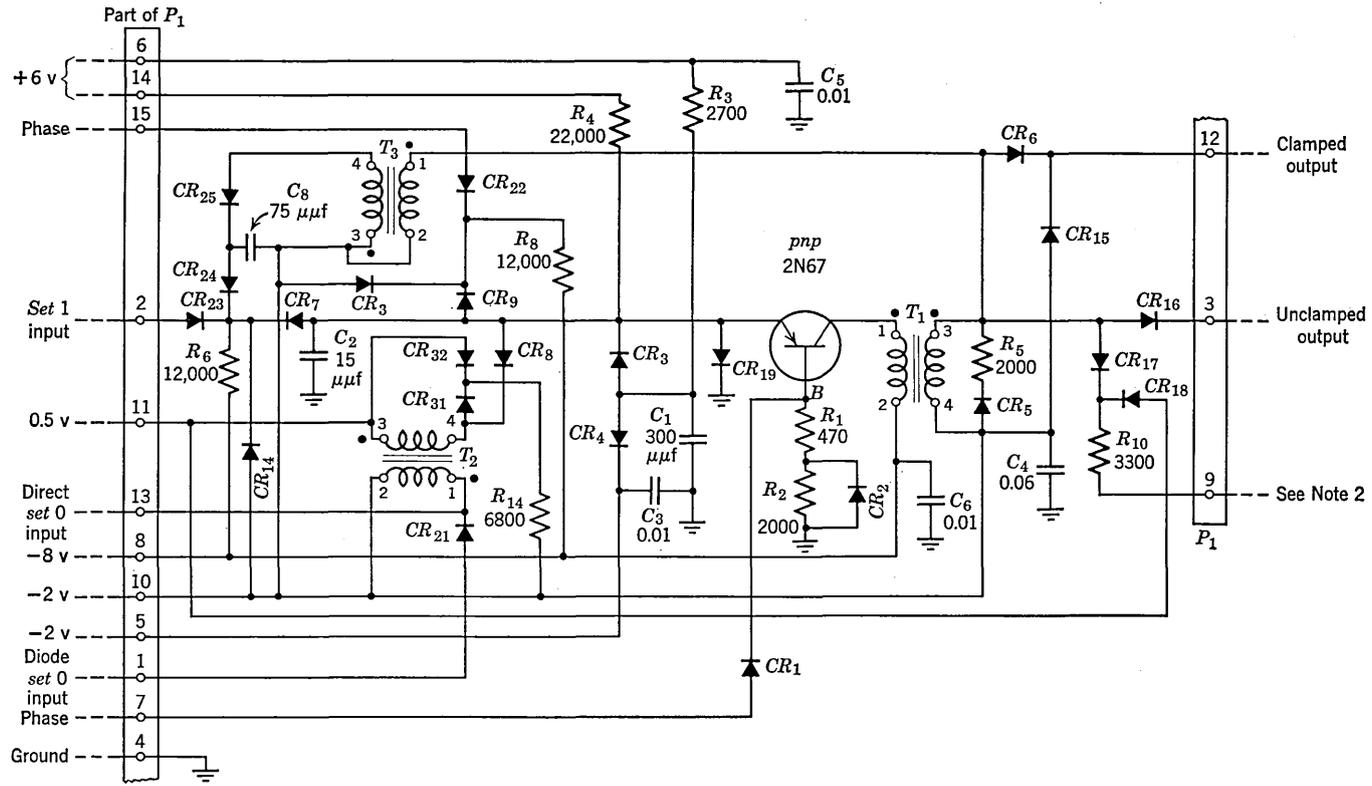


FIG. 19. Tradic storage circuit. Notes. (1) All values are expressed in ohms and microfarads unless otherwise indicated. (2) With one or two loads, strap 9 to 8. With three loads, strap 9 to 10. With more loads, leave 9 open.

widespread use. The calculator has a decimal arithmetic unit and decimal storage. It operates in the 1-2-4-8 binary coded decimal system, with numbers transferred in parallel by decimal digit, and uses an essentially d-c type of logic. Transistor logic circuits are employed where the speed requirements dictate; elsewhere diode logic elements are utilized. The circuitry is based on direct-coupled junction transistors with diode clamping (Ref. 6).

**Characteristics.** The input-output is by punched cards which are read and punched at the rate of 100 cards per minute. Calculations are performed at a basic pulse repetition frequency of 50 kilocycles. The computer cycle occurs between the time a card is first read and before it arrives at the punching station. Thus, the calculated result is punched in one card while the next is being read. As a service check, the calculator must operate without failures at 70 kilocycles.

**Basic Circuits.** Various types of basic circuits are employed to accomplish the necessary switching and control functions. Included among these circuits are inverters, emitter followers, flip-flops, *and* *or* transistor gates, and output drivers.

The germanium junction transistors which are used in most of the circuits are fairly low-frequency response units with alpha cutoff at 1 megacycle. Further specification includes a beta between 40 and 90, and a reverse current, at room temperature, not greater than 10 microamperes with 5 volts reverse bias.

The 5-volt signal excursion, from  $-5$  volts to ground, is considered optimum for a number of reasons, the most important of which is the need for reliability. In general, the cutoff current of a transistor has a higher increase with time at the higher collector voltages, and this would indicate that a low collector voltage, and hence a low signal swing should be maintained. The use of low signal levels, on the other hand, introduces problems relating to the speed of operation and the amount of energy required for switching. Alpha cutoff is directly related to the collector voltage, and collector capacitance is inversely related to the collector voltage. These factors would indicate that a high signal level is best for high-speed operation. The 5-volt swing employed in the circuits proves to be a satisfactory compromise.

**Inverter.** The inverter is shown in Fig. 20*a*, *b*, with *pnp* and *nnp* junction transistors. The *pnp* inverter is capable of charging rapidly a capacitive load when a positive output is required. Conversely, the *nnp* inverter can be used when a capacitive load must be driven rapidly in the negative direction.

The diode clamps,  $D_1$  and  $D_2$ , reduce by a factor of two the turn-off time of the output signal when the inverter is driving a capacitive load.

They also firmly establish the *off* level, and make this level less dependent on the type of load being driven.  $R_1$  and  $C_1$  serve to decrease turn-off delay caused by minority carrier storage effects. The value of load resistor  $R_c$  is chosen so as to limit the maximum collector current to 5 milliamperes.

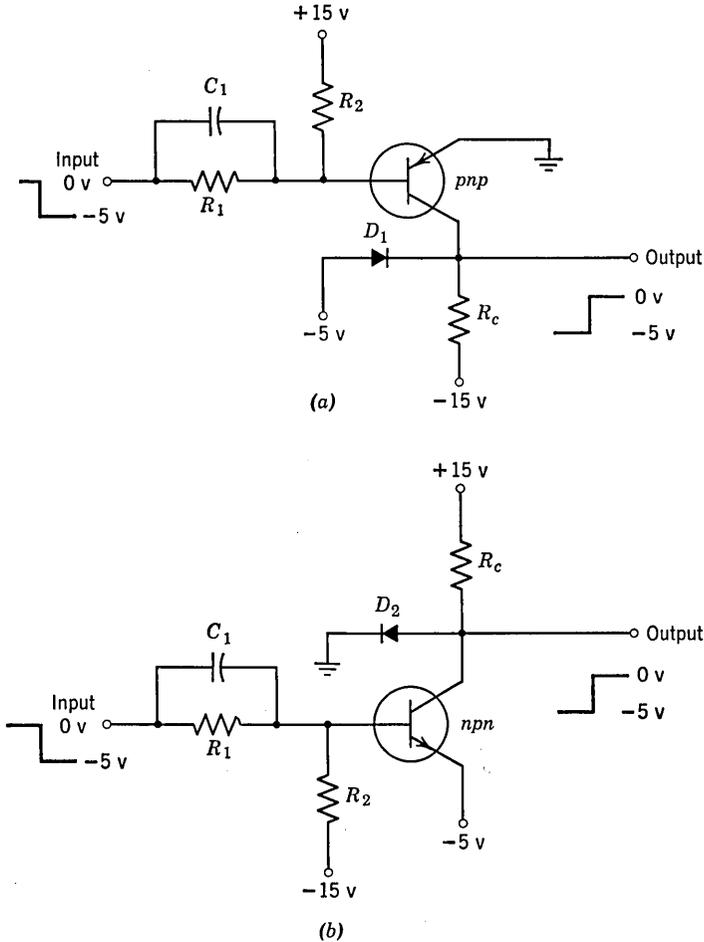


FIG. 20. Inverter. (a) *pnp* type:  $R_1$ , 10 k $\Omega$ ;  $R_2$ , 110 k $\Omega$ ;  $R_c$ , 3 k $\Omega$ ;  $C_1$ , 680  $\mu\text{f}$ .  
(b) *npn* type:  $R_1$ , 10 k $\Omega$ ;  $R_2$ , 75 k $\Omega$ ;  $R_c$ , 3.9 k $\Omega$ ;  $C_1$ , 680  $\mu\text{f}$ .

**Emitter Follower.** Emitter followers of types *pnp* and *npn* are shown in Fig. 21a, b. The characteristics of the basic arrangement cause the signal to suffer a loss in amplitude and a shift in level. After the signal is lowered by several stages of emitter followers or diode logic,

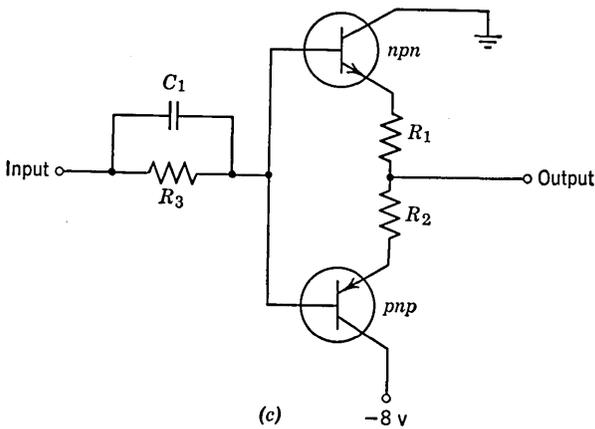
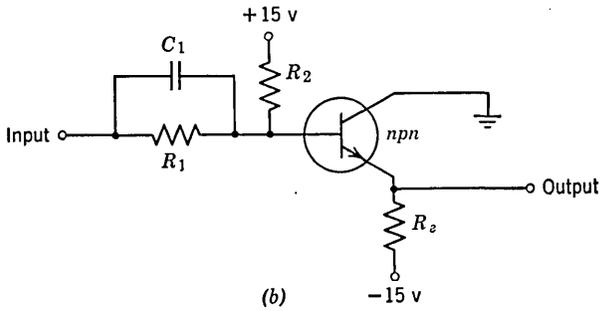
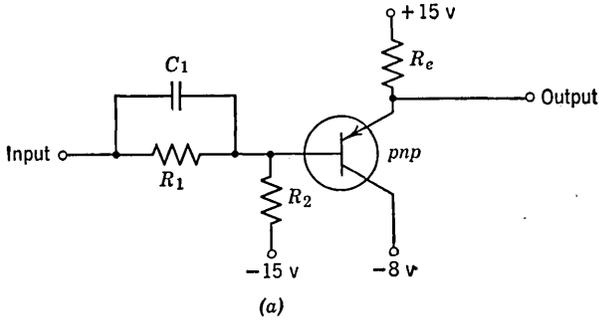


FIG. 21. Emitter follower. (a) *pnp* type:  $R_1$ , 1 k $\Omega$ ;  $R_2$ , 75 k $\Omega$ ;  $R_e$ , 3.9 k $\Omega$ ;  $C_1$ , 680  $\mu\text{mf}$ . (b) *npn* type:  $R_1$ , 1 k $\Omega$ ;  $R_2$ , 110 k $\Omega$ ;  $R_e$ , 3 k $\Omega$ ;  $C_1$ , 680  $\mu\text{mf}$ . (c) Complemented type:  $R_1$ , 20  $\Omega$ ;  $R_2$ , 20  $\Omega$ ;  $R_3$ , 510  $\Omega$ ;  $C_1$ , 1000  $\mu\text{mf}$ .

the loss in amplitude is corrected by passing the signal through an inverter. Compensation for shift in level, on the other hand, is made at each emitter follower by means of resistors  $R_1$  and  $R_2$ , and the voltage to which  $R_2$  is returned.

One serious difficulty encountered with an emitter follower, when driving a capacitive load, is an overshoot in the output waveform. This distortion is minimized by shunting the input resistor  $R_1$  with capacitor  $C_1$ .

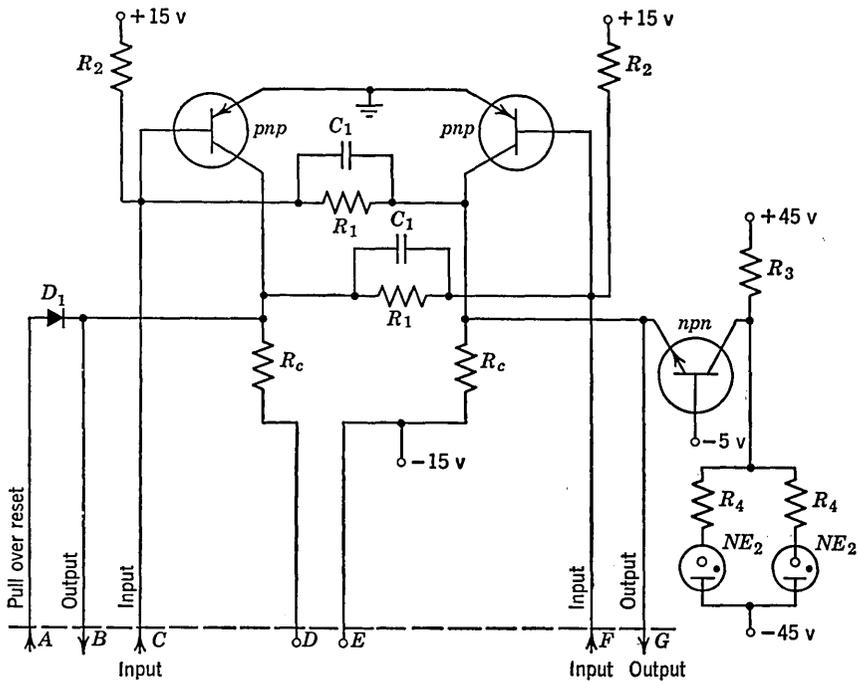
The collector of the *pn*p emitter follower is returned to  $-8$  volts rather than  $-5$  volts. This prevents minority carriers from being injected into the base region from the collector when the input is more negative than the latter value. Because the *npn* emitter followers in the calculator are used to drive loads with long time constants, the minority carrier effects in this case are not appreciable.

When fast rise and fall times are required with capacitive loads, a complemented emitter follower (Fig. 21c) is used. A positive-going input causes the *pn*p transistor to cut off and the *npn* transistor to supply the load current. As the input goes negative, the conditions are reversed and the *pn*p unit discharges the capacitive load. This circuit can drive as many as twelve flip-flops in parallel at the pulse repetition of 50 kilocycles.

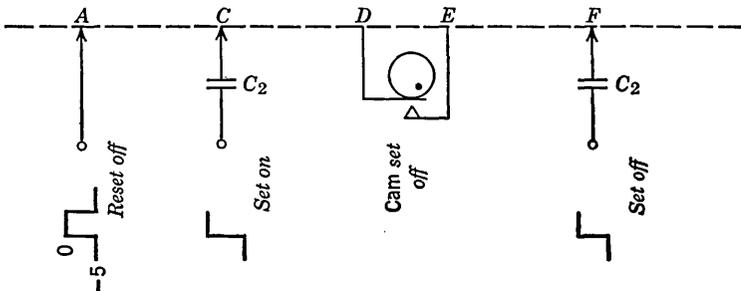
**Flip-Flops.** The basic (Eccles-Jordan) flip-flop is shown in Fig. 22a. The collector on the left side is clamped at  $-5$  by means of diode  $D_1$ ; the collector on the right side is similarly clamped by the emitter-base junction of the auxiliary *npn* transistor. This *npn* unit develops a voltage swing of 40 volts and is required for operation of the neon indicator lamps.

The basic drive circuits used in conjunction with the flip-flop circuit are shown in Fig. 22b, c. The type of input shown at (b) is used when the trigger is operated in a ring circuit. The flip-flop can be reset by means of a cam-driven contact which opens the collector supply on one side of the flip-flop. It can also be reset by means of diode  $D_1$  when the input is driven positive by a *pn*p inverter which is normally off. Positive going waveforms at the *Set on* or *Set off* terminals set the trigger in the *on* (right transistor conducting) or *off* (left transistor conducting) state, respectively.

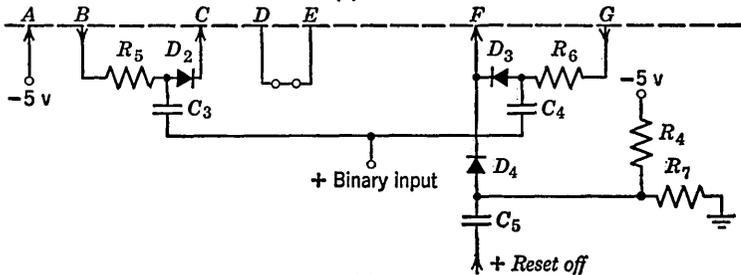
Figure 22c shows the type of input connection for flip-flops used in counter applications. The diode  $D_1$  is connected to  $-5$  volts, and no cam reset is required. The circuits comprising  $R_5$ ,  $D_2$ , and  $C_3$ , and  $R_6$ ,  $D_3$ , and  $C_4$  act as gates or steering circuits. Capacitor  $C_5$  and diode  $D_4$  provide a reset *off* input. These binary flip-flops may be cascaded, with blocking feedback from the fourth to the second stage, to form a binary-coded decimal counter as indicated in Fig. 23.



(a)



(b)



(c)

FIG. 22. (a) Basic flip-flop circuit:  $R_1$ , 10 k $\Omega$ ;  $R_2$ , 110 k $\Omega$ ;  $R_3$ , 39 k $\Omega$ ;  $R_4$ , 200 k $\Omega$ ;  $C_1$ , 680  $\mu\text{f}$ ;  $R_c$ , 3 k $\Omega$ . (b) Ring trigger drive:  $C_2$ , 680  $\mu\text{f}$ . (c) Counter trigger drive:  $R_5$ , 3 k $\Omega$ ;  $R_6$ , 3 k $\Omega$ ;  $R_7$ , 3 k $\Omega$ ;  $R_8$ , 27 k $\Omega$ ;  $C_3$ , 1000  $\mu\text{f}$ ;  $C_4$ , 1000  $\mu\text{f}$ ;  $C_5$ , 1000  $\mu\text{f}$ .

**Logic Elements.** Transistors are used for logic in the calculator wherever the requirements of speed dictate. *And* and *or* operations are performed by emitter followers, paralleled as shown in Fig. 24a, b. The output of the *and* circuit is at ground level if *X*, *Y*, and *Z* are simultaneously at ground. The *or* circuit output is at ground if any of the inputs are at ground.

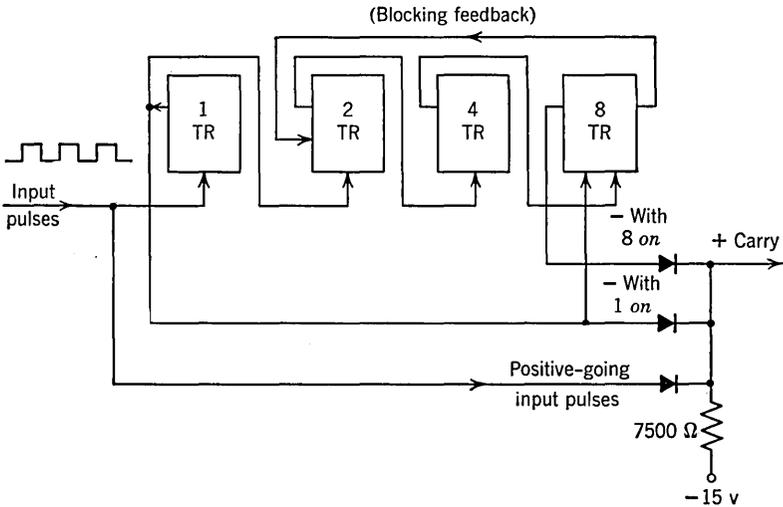


FIG. 23. Coded decimal counter.

**Output Drivers.** Relays and other electromagnetic devices are driven by a power transistor developed by IBM. This transistor exhibits a negative resistance characteristic, and its use is quite similar to that of a thyatron tube in a control application.

The relay driver circuit is shown in Fig. 25. The transistor latches on with a short input pulse of about 1 microsecond duration, and the collector circuit is opened mechanically to return the transistor to its low-conduction state. The average emitter to collector current gain is four; the average voltage drop is 2 volts at 100 milliamperes.

**Packaging.** The transistorized calculator occupies less than one-half the volume of the model 604 vacuum tube equivalent, and further miniaturization is practicable. Printed wiring techniques are used for assembly of the transistor circuitry. A pluggable unit arrangement is used to facilitate assembly and servicing of the machine. The calculator contains 2165 transistors and 3600 diodes. A typical package consists of circuit components soldered on a 3-inch by 5½-inch card with 18 terminals. The 595 cards in the calculator comprise a total of 40 different types.

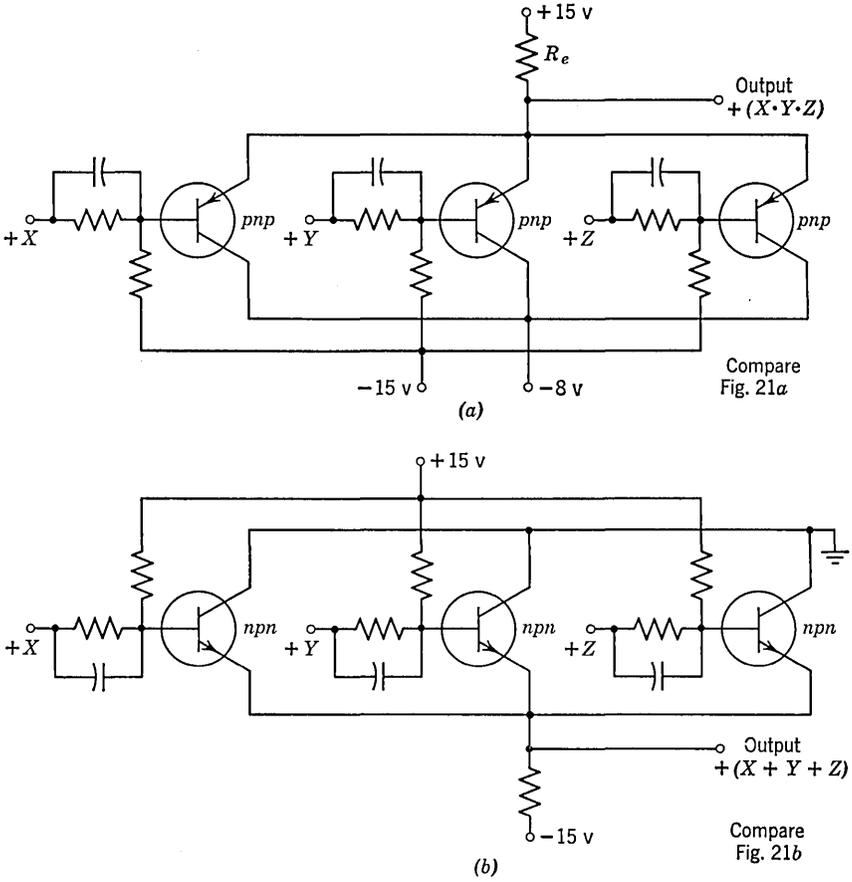


FIG. 24. Logic elements: (a) and circuit, (b) or circuit.

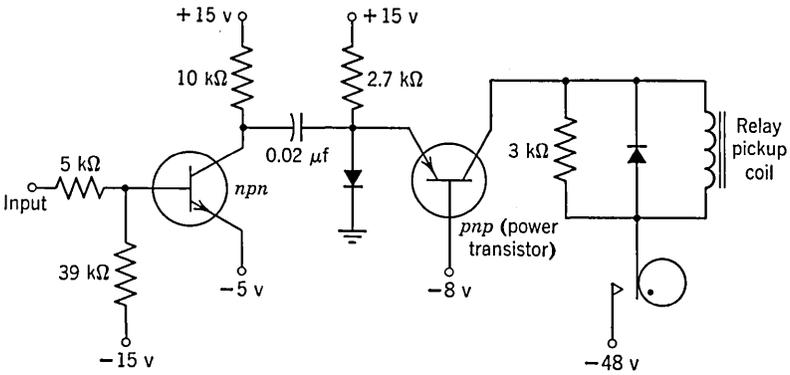


FIG. 25. Relay driver.

**Power Supplies.** The supply voltages are +15, -5, -8, and -15 volts for the transistor circuits; +45 and -45 volts are supplied for the neon indicators. The power supply has a tuned transformer input which enables the circuits to operate satisfactorily with a line variation from 90 to 125 volts at 60 cycles per second.

The total power requirements for the calculator is 310 watts. Compared with the requirements of the vacuum-tube machine, this represents a power reduction of the order of 95 per cent.

#### ACKNOWLEDGMENTS

The author wishes to express his appreciation for the permission received from the various companies to include the material contained in this chapter. He further gratefully acknowledges the assistance of those listed for their assistance in providing information included here: H. A. Affel, Jr., Philco Corporation, for material on Transac and DCTL circuitry; E. G. Clark and J. H. Dahms, Burroughs Corporation, for material on design factors and information on transistor switching properties and for their assistance in compiling the figures; J. R. Harris, Bell Telephone Laboratories, for his critical review of the text; R. M. Shultz, Auerbach Electronics Corporation, for his contribution on DCTL circuitry.

#### REFERENCES

1. R. H. Beter, W. E. Bradley, R. H. Brown, and M. Rubinoff, *Electronics*, **28**, 132-136 (June 1955).
2. Tradic Phase I Computer—Summary Engineering Report, Bell Telephone Laboratories, Whippany, N. J., under contract AF 33(600)-21536.
3. G. D. Bruce and J. C. Logue, The IBM 608 Calculator, International Business Machines Corporation, Poughkeepsie, N. Y.
4. J. J. Ebers and J. L. Moll, Large-Signal Behavior of Junction Transistors, *Proc. I.R.E.*, **42**, 1761-1772 (1954).
5. J. L. Moll, Large-Signal Transient Response of Junction Transistors, *Proc. I.R.E.*, **42**, 1773-1784 (1954).
6. G. D. Bruce and J. C. Logue, An Experimental Transistorized Calculator, *Elec. Engr.*, **74**, 1044-1048 (1955).
7. E. Wolfendale, L. P. Morgan, and W. L. Stephenson, The Junction Transistor as a Computing Element (Pt. 2), *Elec. Engr.*, **29**, 83-87 (1957).
8. R. H. Baker, Boosting Transistor Switching Speed, *Electronics*, **30**, 190-193 (1957).
9. G. J. Prom and R. L. Crosby, Junction Transistor Switching Circuits for High-Speed Digital Computer Applications, *Trans. I.R.E. Elec. Comps.*, **EL-5**, 192-196 (1956).
10. G. W. Booth and T. P. Bothwell, Basic Logic Circuits for Computer Applications, *Electronics*, **30**, 196-200 (1957).
11. W. D. Rowe and G. H. Royer, Transistor *Nor* Circuit Design, *Trans. Am. Inst. Elec. Engrs., Paper No. 57-196*.
12. E. V. Cobler, Steady State Conditions of a Junction Transistor Flip-flop, MIT Lincoln Laboratory Memo No. M-2686, March 1954.
13. A. E. McMochon, I. L. Lebon, and R. H. Baker, A Design Procedure for Junction Transistor Flip-flops, MIT Lincoln Laboratory Internal Memo No. M24-34, April 1954.

# Logical Design

*Lowell Amdahl*

1. Computer Elements	17-01
2. Algebraic Techniques of Logical Design	17-10
3. Preliminary Design Considerations	17-24
4. Detailed Logical Design	17-30
5. Direct Simulation of a Logical Design	17-38
References	17-42

## I. COMPUTER ELEMENTS

### Definitions

*Logical design* is the specification of the interconnection of computer elements to produce a computer with the desired operational properties. *Computer elements* are aggregates of components which, as independent units, perform elementary computing functions. *Examples.* Flip-flops, inverters, cathode followers, gates.

*Logical elements* are computer elements which perform a nontrivial logical function; that is, some input or combination of inputs produces something new as an output. The cathode follower is not a logical element, whereas an inverter is.

Two classes of logical elements are *decision elements* and *storage elements*. Outputs of decision elements respond to their input stimuli during the same time interval as the inputs. Decision elements are frequently referred to as *gates*. *Examples.* Inverters, *and* gates, *or* gates.

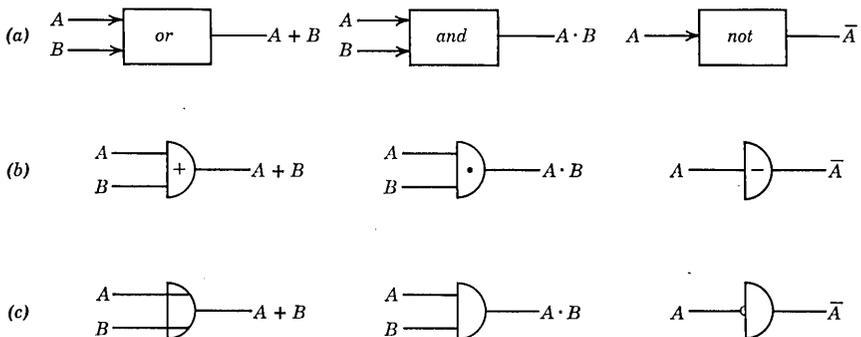
Outputs of storage elements respond to the input stimuli in a later time

interval. Such a time interval is a basic characteristic of a storage element. *Examples.* Flip-flops, delay lines, magnetic cores. For asynchronous computers, time intervals are determined by completion of events.

*Note.* It will be assumed in the following discussion that all logical elements have input or output values of 1 or 0 only, in accord with the widespread use of binary devices in electronic computers. Other terms used for the binary states are true or false, on or off, mark or space, open or closed, and up or down.

**Symbols.** The elements used in computer logical design may be represented by symbols. Three different representations in use are shown in Table 1. Representation (a) simply uses a box containing the operation name, (b) uses a segment of a circle containing the algebraic symbol, and (c) uses a purely symbolic representation. None of these has achieved universal adoption.

TABLE 1. BLOCK SYMBOLS FOR COMPUTER ELEMENTS



**Decision Elements** (See Ref. 1, Chap. 2; Ref. 2, Chap. 3; Ref. 3, Chap. 4; Ref. 4, Chaps. 3 and 4)

Inputs to decision elements will be designated by letters at the beginning of the alphabet,  $A$ ,  $B$ ,  $C$ , etc. Outputs of decision elements will be designated  $F_0$ ,  $F_1$ ,  $F_2$ , etc. Later it will be convenient to identify outputs by their functional relationships to inputs, since all outputs of decision elements are dependent only on the input variables.

**One Input.** A decision element with one input has two possible input configurations and four possible output responses. In general, an  $N$  input decision element can have  $2^N$  input combinations and  $2^{2^N}$  possible output responses. Some of these responses are trivial.

Table 2 shows the possible values of 0 or 1 for input  $A$ . Each of the remaining four columns indicates a possible response to this input, followed by its usual name and Boolean algebraic representation.

Functions  $F_0$  and  $F_3$  are trivial in that they provide constant outputs

of 0 and 1, respectively, completely unaffected by the input information. Function  $F_1$  repeats the input information, commonly performed in computers by a wire, a cathode or emitter follower, or a noninverting amplifier. Only one binary decision element with a single input and output exists—the inverter,  $F_2 = \bar{A}$ . This function is commonly referred to as *not*, *negation*, or *complementation*.

TABLE 2. DECISION ELEMENTS WITH ONE INPUT

Input	Possible Outputs			
	$F_0$	$F_1$	$F_2$	$F_3$
A				
0	0	0	1	1
1	0	1	0	1
Name	(Trivial)	Wire, cathode follower, amplifier	Inverter, complementer	(Trivial)
Algebraic function	0	A	$\bar{A}$	1

**Two Inputs.** Table 3 shows the four combinations of inputs  $A$  and  $B$ , and the sixteen possible output responses,  $F_0$  to  $F_{15}$ , often referred to as *switching functions*.  $F_0$  and  $F_{15}$  are trivial in that they have no dependence on the inputs.  $F_3, F_5, F_{10}$ , and  $F_{12}$  are special cases since they depend on only one of the two inputs.  $F_2$  and  $F_4$ , and also  $F_{11}$  and  $F_{13}$ , can be symmetrically paired since an interchange of the name of the inputs produces the paired function. There are eight functions of interest:

$$F_1, F_2 \text{ or } F_4, F_6, F_7, F_8, F_9, F_{11} \text{ or } F_{13}, F_{14}.$$

Because of their simple diode and resistor realization, the most prevalent of these are  $F_1$  and  $F_7$ , the *and* gate and the *or* gate.  $F_6$ , the *exclusive or*, and its complement  $F_9$ , occur frequently in arithmetic processes. The remaining functions are rather simply mechanized by magnetic core techniques. They are also of considerable theoretical interest (see the section on Completeness).

**Many Inputs.** Decision elements with many inputs are usually extensions or combinations of those already discussed. For example, a six-input *and* gate will produce an output of 1 only when all six inputs are simultaneously 1. Similarly, a three-input *or* gate produces an output of 0 only when all three inputs are simultaneously 0. Although not common in practice, a many input *exclusive or* gate produces an output of 1 if an odd number of inputs has a value of 1, and produces an output of 0 if an even number of inputs has a value of 0.



## Storage Elements

A binary storage element is capable of retaining one bit of information. For *synchronous computers* in which events are timed by clock signals, storage elements have the property of retaining information for one or more time intervals. For *asynchronous computers* in which events occur at irregularly spaced intervals of time, a storage element has the property of retaining information from one input event to the next. *Bistable* storage elements can remain indefinitely in a given state in the absence of input events; thus, these devices are suitable for either synchronous or asynchronous computers. Bistable storage elements used in the arithmetic and control sections of a computer are referred to as *flip-flops* or *toggles*. (See Chap. 14.) *Monostable* storage elements are binary devices which have only one stable state, but are capable of indicating a second state as a transient response to an input event. These elements are well suited to synchronous operation, and several computers that utilize them to a large extent have been built. There is no widely accepted nomenclature for monostable storage elements; however, *delay element* and *dynamic flip-flop* have been used.

For purposes of logical design, the properties of storage elements used for arithmetic and control are of especial importance. The outputs of flip-flops are dependent on the previous *state*, or *stored value*, of the flip-flop and the previous input configuration. The equation which relates the output to the input and previous state is a *difference equation* in time. (See Vol. I, Chap. 4.) It will be assumed that the output responds to the input in a later time interval as shown by the superscripts in the difference equations.

Flip-flops have been mechanized in vastly different ways with a large number of variations in their logical response to inputs. Some of the more usual logical configurations will be described. (See Ref. 1, Chap. 5; Ref. 3, Chap. 3.)

**Outputs of Storage Elements.** Almost all storage elements have a *normal*, or *1's*, or *true output* that is equivalent to its stored value (state). That is, if a flip-flop stores a value of 0, the output has a value of 0. It is usually desirable to have a *complement*, or *0's*, or *false output* which has the opposite value to that stored by the flip-flop. Some flip-flops provide a *differentiated output* which is activated by a transition in the stored value of the flip-flop, say a 1 to 0 transition. This type of output has particular utility in a binary counter. Table 4 shows the different types of outputs as related to an arbitrary sequence of stored values of a flip-flop for successive unit time intervals  $\tau$  to  $\tau + 4$ .

*Note.* Since these output values are directly derivable from the sequence of states, the subsequent discussion will consider only the relationship

TABLE 4. OUTPUTS AS RELATED TO AN ARBITRARY SEQUENCE OF FLIP-FLOP STATES

Time Interval	$\tau$	$\tau + 1$	$\tau + 2$	$\tau + 3$	$\tau + 4$
Flip-flop state	1	0	0	1	0
Normal (1's) output	1	0	0	1	0
Complement (0's) output	0	1	1	0	1
Differentiated ( $1 \rightarrow 0$ ) output	0	1	0	0	1

between inputs to storage elements and their states (or normal outputs).

**Single Input Storage Elements.** The two most common single input storage elements are called delays and triggers. The two differ markedly, since the state of a delay is not inherently a function of its previous state, whereas the state of a trigger is completely dependent on its previous state.

**Delay Elements.** A *delay element* is a storage device which delays its input information one or more time intervals before presenting it on its normal output terminal. The state of a single time interval delay storage element  $U$  at time  $\tau + 1$  depends only on its input  $D$  at time  $\tau$ , as shown by the following difference equation:

$$U^{\tau+1} = D^{\tau}.$$

The truth table specifying the relationship between the input and the state of the delay element is also extremely simple as shown in Table 5.

TABLE 5. STORAGE CHARACTERISTICS OF A SINGLE TIME INTERVAL DELAY

$D^{\tau}$	$U^{\tau+1}$
0	0
1	1

The delay element has a striking dissimilarity to other storage elements to be described in that its state is not intrinsically a function of previous states (that is,  $U^{\tau}$  does not appear on the right-hand side of the difference equation). From the standpoint of input gating, it is not surprising that the delay is best suited for applications where such dependence is not desired, e.g., shifting registers. However, by making the input a function of its own output, a delay element can be used to "latch" or hold information as shown in the following example.

**EXAMPLE.** A delay element is provided with external gating which enables it to hold information provided on the "set" input until a "reset" input of 1 occurs. This configuration, shown in Fig. 1, is sometimes referred to as a *latch*.

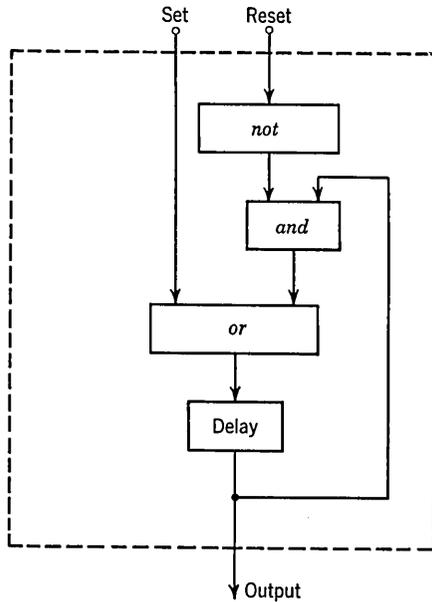


FIG. 1. Delay latch.

**Triggers.** A flip-flop that changes state when its input is 1 and retains its state when its input is 0 is called a *trigger* or a *scale-of-two counter*. The state of trigger  $U$  at time  $\tau + 1$  is a function of its own state and its input

TABLE 6. STORAGE CHARACTERISTICS OF A TRIGGER (SCALE-OF-TWO COUNTER)

$T^\tau$	$U^\tau$	$U^{\tau+1}$
0	0	0
0	1	1
1	0	1
1	1	0

$T$  at time  $\tau$  as described by the truth table of Table 6 and the following difference equation:

$$U^{\tau+1} = (T \cdot \bar{U} + \bar{T} \cdot U)^\tau.$$

Triggers are innately well suited for applications which involve counting or certain other arithmetic operations such as sum functions.

**Multiple Input Storage Elements.** A majority of flip-flops utilize two or more inputs. Additional inputs can provide “intrinsic” gating as indicated by the complexity of the difference equation defining the storage element. If this intrinsic gating is well adapted to the application, sub-

stantial economies in the required decision elements can be realized, as compared with that required in conjunction with a simple storage device such as a delay element.

**R-S Flip-Flop.** The *R-S* flip-flop has two inputs, *R* and *S*, which *reset* and *set* the flip-flop to 0 and 1 respectively. Sometimes these are also referred to as 0's and 1's inputs. If both inputs are 0 at time  $\tau$ , the flip-flop retains its previous state. The situation that both inputs are 1 is not normally permitted because the flip-flop does not respond in a uniform way to this input configuration. The truth table for the *R-S* flip-flop is

TABLE 7. STORAGE CHARACTERISTICS OF AN *R-S* FLIP-FLOP

$S^\tau$	$R^\tau$	$U^\tau$	$U^{\tau+1}$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	?
1	1	1	?

shown in Table 7, and is expressed by the following difference equation with its auxiliary condition:

$$U^{\tau+1} = (S + \bar{R} \cdot U)^\tau,$$

$$R \cdot S = 1 \text{ not permitted.}$$

In a number of important variations of the *R-S* flip-flop *R* and *S* inputs simultaneously equal to 1 are permitted. For example, a flip-flop which responds with a state of 1 to simultaneous inputs (that is, the last two entries for  $U^{\tau+1}$  of Table 7 are 1's) has an *overriding set* input. The "latch" made up of a delay flip-flop and external gating as described in the preceding example has an overriding set input.

**J-K Flip-Flop.** Another variation of the *R-S* flip-flop is one which changes state (triggers) when simultaneous inputs of 1 are applied. This flip-flop is termed a *J-K* flip-flop and is described by Table 8 and the following difference equation:

$$U^{\tau+1} = (J \cdot \bar{U} + \bar{K} \cdot U)^\tau.$$

**R-S-T Flip-Flop.** A flip-flop which has been widely used has three inputs, *R*, *S*, and *T*, which serve to *reset*, *set*, or *trigger* the flip-flop. In its most common form, no more than one input is permitted to have the value 1 during a given time interval. The truth table for the *R-S-T* flip-

TABLE 8. STORAGE CHARACTERISTICS OF A *J-K* FLIP-FLOP

$J^r$	$K^r$	$U^r$	$U^{r+1}$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

flop is shown in Table 9, and has the following difference equation subject to its auxiliary condition:

$$U^{r+1} = (S + T \cdot \bar{U} + \bar{R} \cdot \bar{T} \cdot U)^r,$$

$$R \cdot S + R \cdot T + S \cdot T = 1 \text{ not permitted.}$$

TABLE 9. STORAGE CHARACTERISTICS OF AN *R-S-T* FLIP-FLOP

$T^r$	$S^r$	$R^r$	$U^r$	$U^{r+1}$
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	?
0	1	1	1	?
1	0	0	0	1
1	0	0	1	0
1	0	1	0	?
1	0	1	1	?
1	1	0	0	?
1	1	0	1	?
1	1	1	0	?
1	1	1	1	?

### Selecting a Set of Computer Elements

A minimum requirement of a set of computer elements is that they form a complete set and operate with adequate reliability. Beyond this, it is desirable to have elements which permit flexibility and economy in mechanizing a computer system (see Ref. 5).

**Completeness.** A function or set of functions is *complete* if all other functions can be formed from them. Of the decision elements described

in Table 3, only the Sheffer function ( $F_{14}$ ) and Peirce function ( $F_8$ ) are of themselves complete. A less known fact (see Ref. 6) is that functions  $F_2$ ,  $F_4$ ,  $F_{11}$ , and  $F_{13}$  are each complete if the constants 0 or 1 are available. (These constants are usually a d-c voltage or a clock pulse.) Pairs of functions which are complete are *and* and *negation*, *or* and *negation*, *and* and *exclusive or* ( $F_1$  and  $F_6$ ), and *or* and *exclusive or* ( $F_7$  and  $F_6$ ). Note that *and* and *or* do not form a complete set. But *and*, *or* and *negation* form a complete set, the one most frequently used in logical design.

Decision elements which are complete, together with a nontrivial storage element, form a complete set of logical elements. Because of the intrinsic gating associated with flip-flops, it is also possible to have a storage element which is a universal logical element and which requires no external decision elements.

**Versatility.** In only rare instances have minimal sets of computer elements been chosen because of the versatility and economy of a wider array of decision elements. The decision elements *and*, *or*, and *negation* are commonly used because of their availability, and also because they are well adapted to the pattern of human thought. Usually *and* and *or* gates are mechanized with varying numbers of inputs to suit the need. Circuit limitations sometimes severely restrict the sequence in which gating and permissible cascading of gates can occur. In some computer designs, negation is provided only by storage elements having both "normal" and "complement" outputs, i.e., inverters are not used.

## 2. ALGEBRAIC TECHNIQUES OF LOGICAL DESIGN

Boolean algebra provides a mathematical framework for describing the design of computers utilizing binary computer elements (Vol. I, Chap. 11). An algebraic statement of the interrelation of binary variables is a *switching function*, referred to frequently as a function. Switching functions have been the object of much study, particularly with regard to their simplest representation. (See Ref. 1, Chap. 4; Ref. 2, Chap. 2; and Ref. 4, Chap. 5.)

This section describes the representation of switching functions and their simplification. Section 4 shows the application of these techniques to the particular problem of determining inputs to storage elements.

### Summary of Equivalences in Boolean Algebra

The relations in Table 10 are useful in working with logical expressions. They are postulates or theorems of Boolean algebra.

### Representation of Switching Functions

Switching functions are frequently derived from equivalent tabular or diagrammatic representations. Both of these representations to be described have their merits and limitations for a particular use.

TABLE 10. EQUIVALENCES IN BOOLEAN ALGEBRA

$A + 0 = A$	$A \cdot 0 = 0$
$A + 1 = 1$	$A \cdot 1 = A$
$A + A = A$	$A \cdot A = A$
$A + \bar{A} = 1$	$A \cdot \bar{A} = 0$
$A + B = B + A$	$A \cdot B = B \cdot A$
$(A + B) + C = A + (B + C)$	$(A \cdot B) \cdot C = A \cdot (B \cdot C)$
$A + (B \cdot C) = (A + B) \cdot (A + C)$	$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$
$\overline{(\bar{A})} = A$	$A + AB = A$
$\overline{A + B} = \bar{A}\bar{B}$	$A + \bar{A}B = A + B$
$\overline{A + B + C} = \bar{A}\bar{B}\bar{C}$	$AB + A\bar{B} = A$
$\overline{AB} = \bar{A} + \bar{B}$	$AB + AC + B\bar{C} = AC + B\bar{C}$
$\overline{ABC} = \bar{A} + \bar{B} + \bar{C}$	

$$(A + B)(A + C) = A + BC$$

$$(A + B)(\bar{A} + C) = AC + \bar{A}B$$

$$\overline{(AB + AC)} = \bar{A} + \bar{B}\bar{C}$$

$$\overline{(AB + \bar{A}C)} = A\bar{B} + \bar{A}\bar{C}$$

$$\overline{(A + B)(A + C)} = \bar{A}(\bar{B} + \bar{C})$$

$$\overline{(A + B)(\bar{A} + C)} = (A + \bar{B})(\bar{A} + \bar{C})$$

**Truth Tables.** The binary value of a switching function for each combination of binary variables can be tabulated in a truth table in the following manner.

1. Assign a column to each binary variable.
2. Assign a column to each binary function.
3. Assign a row to each different combination of values of the binary variables. If there are  $n$  variables, there will be  $2^n$  rows.
4. In each row, enter the appropriate functional value corresponding to the combination of variables in that row.

**EXAMPLE.** Consider a function  $F$  of the two binary variables  $A$  and  $B$  defined by the truth table of Table 11.

TABLE 11. A TRUTH TABLE IN TWO VARIABLES

$A$	$B$	$F$
0	0	0
0	1	1
1	0	1
1	1	0

$F$  has a value of 1 in two instances: when  $A$  is 0 and  $B$  is 1 (usually written  $\bar{A}B$  omitting the *and* dot between  $\bar{A} \cdot B$  and read as “not  $A$ , and  $B$ ”), or when  $A$  is 1 and  $B$  is 0 (written  $A\bar{B}$  and read “ $A$  and not  $B$ ”). The *and* operator is sometimes called the *logical product*.

In words,  $F$  has a value of 1 when  $A$  or  $B$  is 1, but not both. This is the *exclusive or* function, and is given the symbol  $\oplus$ ; that is,  $F = A \oplus B$  (see Table 3).

Had the last entry in the  $F$  column of Table 11 been 1, then  $F$  would have a value of 1 when either or both of  $A$  or  $B$  are 1. This is the *inclusive or* operator (also called the *logical sum*), and is given the symbol  $+$ . Hereafter, if the word *or* is used in the sense of a logical function, the *inclusive or* is meant.

**Minterm Canonical Form.** Each row of a truth table corresponds to a *minterm*, also called a *primitive*. The four minterms of two variables  $A$  and  $B$  are  $\bar{A}\bar{B}$ ,  $\bar{A}B$ ,  $A\bar{B}$ , and  $AB$ . Since any function can be described by a truth table, it can also be described by utilizing minterms. The general form for the representation of a function  $F$  of  $n$  variables by minterms is:

$$(1) \quad F = \sum_{i=0}^{2^n-1} a_i \cdot m_i,$$

where  $\sum$  implies the connective *or* ( $+$ ) between *terms* and  $\cdot$  is the connective *and* between the *factors*  $a_i$  and  $m_i$ . The  $a_i$ 's are the functional values which can be obtained from a truth table corresponding to a minterm,  $m_i$ . A function expressed in this manner is in *canonical* form.

**EXAMPLE.** The minterm canonical expression for function  $F$  of Table 11 is obtained as follows. Assign the minterms  $m_0$  to  $m_3$  to successive rows of Table 11 and substitute into eq. (1).

$$\begin{aligned} F &= \sum_{i=0}^3 a_i \cdot m_i \\ &= a_0 \cdot m_0 + a_1 \cdot m_1 + a_2 \cdot m_2 + a_3 \cdot m_3 \\ &= 0 \cdot \bar{A} \cdot \bar{B} + 1 \cdot \bar{A} \cdot B + 1 \cdot A \cdot \bar{B} + 0 \cdot A \cdot B \\ &= \bar{A} \cdot B + A \cdot \bar{B} \\ &= \bar{A}B + A\bar{B} \end{aligned}$$

The last step of dropping the *and* dot conforms to usual practice. Note that the final canonical form can be written directly from the truth table merely by connecting with *or*'s all minterms having functional values of 1.

**Maxterms.** Logical functions may also be expressed as maxterms. A *maxterm* is the logical sum of all of the variables with each variable or its complement appearing once. Hence, the maxterms of two variables are  $\bar{A} + \bar{B}$ ,  $\bar{A} + B$ ,  $A + \bar{B}$ , and  $A + B$ . Any function can be repre-

sented as a logical sum of minterms or a logical product of maxterms. (See Ref. 2, Chap. 3.)

**Normal Form.** A typical computing system will consist of several levels of cascaded decision elements of the *and*, *or*, and *not* type. A direct algebraic statement of this gating network may be quite complicated. Techniques are available to the logical designer to enable him to manipulate these complex functions for purposes of simplification or reorganization. In the previous section it was shown that any function can be written in canonical minterm form by consulting a truth table of that function. It is also possible to transform an arbitrary function into canonical form by algebraic manipulation only.

**DE MORGAN'S THEOREMS.** Manipulations of functions frequently involve the use of De Morgan's theorems:

$$(2) \quad \overline{A \cdot B \cdot C} = \bar{A} + \bar{B} + \bar{C}$$

$$(3) \quad \overline{A + B + C} = \bar{A} \cdot \bar{B} \cdot \bar{C}$$

A switching function composed of terms with an *or* connective, each term being composed of factors with an *and* connective is in *normal form*, also called *disjunctive form*. A normal function is closely related to a minterm canonical function; they differ only in that a term of a normal function need not be a minterm.

Functions are frequently expressed in normal form as a basis for comparison or simplification.

**EXAMPLE:** Put the following expression into normal and canonical form.

$$\begin{aligned} & \overline{(A + \bar{B} + \bar{D}) \cdot (\bar{B} + C + D)} \\ &= (A + \bar{B} + \bar{D}) + (\bar{B} + C + D) \\ &= (\bar{A} \cdot B \cdot D) + (B \cdot \bar{C} \cdot \bar{D}) \\ &= \bar{A}BD + B\bar{C}\bar{D} \quad \text{normal form} \\ &= \bar{A}BD(C + \bar{C}) + B\bar{C}\bar{D}(A + \bar{A}) \\ &= \bar{A}BCD + \bar{A}B\bar{C}D + AB\bar{C}\bar{D} \\ & \quad + \bar{A}B\bar{C}\bar{D} \quad \text{canonical minterm form.} \end{aligned}$$

It is possible to reverse this procedure and go from a normal expression to some other form which, for example, may be required by the available decision elements.

**Veitch Diagrams.** A very useful form of the truth table has been devised by E. W. Veitch which contains all the information of a tabulated truth table in a more compact form (see Refs. 7 and 8). Moreover, it is a convenient representation for simplifying functions by geometrical relationships.

The Veitch diagram has a unit square for each minterm of the binary

variables, each square corresponding to a row entry in a tabulated truth table. Figure 2 illustrates a Veitch diagram for two variables,  $A$  and  $B$ . It has the following characteristics:

1. All squares for which  $A$  has a value of 1 are in the column headed by  $A$ .
2. All squares for which  $A$  has a value of 0 are in the column headed by  $\bar{A}$ .
3. All squares for which  $B$  has a value of 1 are in the row headed by  $B$ .
4. All squares for which  $B$  has a value of 0 are in the row headed by  $\bar{B}$ .

Figure 2 shows the appropriate minterm entered into each square, this being the coordinates of the square by column and row heading. *Note.* In practice, headings  $\bar{A}$  and  $\bar{B}$  are omitted since they are implied by the absence of  $A$  and  $B$  headings respectively.

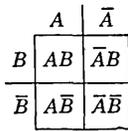


FIG. 2. A Veitch diagram in two variables.

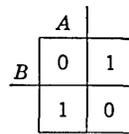


FIG. 3.  $F = A \oplus B$ .

A Veitch diagram of the *exclusive or* function of  $A$  and  $B$  is shown in Fig. 3, and corresponds to the tabulated truth table of Table 11. Here the functional values of 0 or 1 have been entered into the appropriate minterm square.

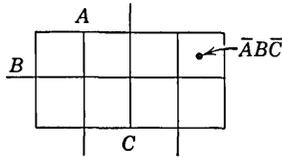


FIG. 4. A diagram in three variables.

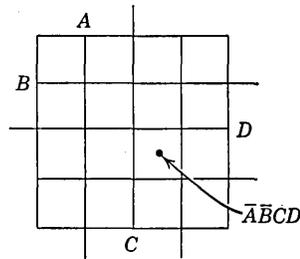


FIG. 5. A diagram in four variables.

The Veitch diagram for three and four variables is a straightforward extension of that shown for two variables, and is illustrated in Figs. 4 and 5. Like tabulated truth tables, Veitch diagrams can be extended to any number of binary variables; however, their utility becomes marginal

for a large number of variables. Beyond four variables, variable groups must be repeated. Figures 6 and 7 show diagrams for five and six variables.

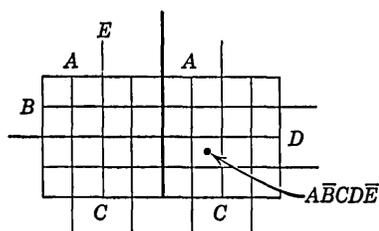


FIG. 6. Five variables.

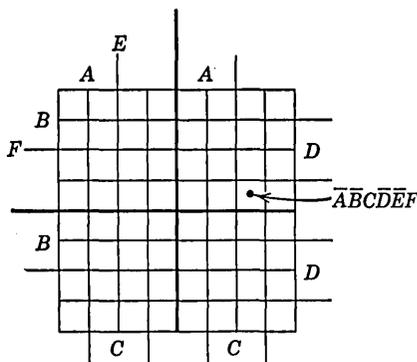


FIG. 7. Six variables.

### Simplification of Switching Functions

Once a tentative logical design for a computer has been prepared, it is important to attempt to simplify it. There are two approaches to this problem:

1. Achieving the same or equivalent operation by a different approach.
2. Reducing the gating network by algebraic techniques.

The first of these approaches to simplification requires a thorough reconsideration of the logical design. The following questions should be asked: What are the alternatives? Has time-sharing of equipment been properly emphasized? Are the chosen timing signals unique? Particular study should be given to repetitious parts of the computer, since savings in equipment are multiplied.

The second approach to simplification is formal in nature and can therefore be applied in a systematic way. This section is devoted to simplification by algebraic techniques.

**Simplification by Inspection.** The following four equivalences are given to aid the simplification process by inspection. A logical designer will find these very useful, particularly if he can perceive these simplifications in a subtle multivariable form.

$$(4) \quad A + AB = A,$$

since  $A + AB = A1 + AB = A(1 + B) = A(1) = A.$

$$(5) \quad AB + A\bar{B} = A,$$

since  $AB + A\bar{B} = A(B + \bar{B}) = A(1) = A.$

$$(6) \quad A + \bar{A}B = A + B,$$

since  $A + \bar{A}B = A + AB + \bar{A}B = A + B$ .

$$(7) \quad AB + AC + B\bar{C} = AC + B\bar{C},$$

since  $AB + AC + B\bar{C} = ABC + AB\bar{C} + AC + B\bar{C} = AC(B + 1) + B\bar{C}(A + 1) = AC + B\bar{C}$ .

Following each of the above equivalences is the proof that it holds, assuming  $A(1) = A$ ,  $A + 1 = 1$ ,  $A + \bar{A} = 1$ , and  $AB + AC = A(B + C)$ .

EXAMPLE 1.

$$\begin{aligned} AB + BC + \bar{A}B\bar{C} &= AB + BC + \bar{A}B && \text{from (6),} \\ &= B + BC && \text{from (5),} \\ &= B && \text{from (4).} \end{aligned}$$

EXAMPLE 2.

$$\begin{aligned} A\bar{B} + A\bar{C} + B\bar{C} + \bar{A}\bar{B}C &= A\bar{B} + B\bar{C} + \bar{A}\bar{B}C && \text{from (7),} \\ &= A\bar{B} + B\bar{C} + \bar{B}C && \text{from (6).} \end{aligned}$$

*Note.* For Example 2 there is an equivalent result,  $A\bar{C} + B\bar{C} + \bar{B}C$ .

**Quine Simplification Method.** Any switching function can be reduced to its simplest normal form. Therefore, any switching circuit composed of *or*'s and *and*'s (normal form) can be reduced to its most economical form. This is true for any reasonable way of evaluating the cost of the decision elements. In its simplest form, a function is *irredundant*. A method of obtaining the simplest normal representation of a function has been devised by W. V. Quine. (See Refs. 9 and 10.)

The Quine method consists of three steps: (1) Find the prime implicant terms. (2) Find the essential prime implicant terms. (3) If the essential terms do not complete the function, find the simplest combination of remaining prime implicant terms required for completion.

The expressions *prime implicants* and *essential prime implicants* are defined later by the processes which obtain them. However, it is evident from the brief statement of the three steps that the simplest function contains *only* prime implicant terms and *at least all* of the essential prime implicants.

*Step 1. Finding the Prime Implicant Terms.* It is assumed that the switching function to be simplified is in normal form. (See preceding paragraphs for techniques for transforming functions into normal form.) The prime implicants are obtained by two distinct processes of reduction and expansion of the function being simplified. The process of reduction is simplification by inspection, by using the first three rules given by eqs. (4), (5), and (6). These rules are repeated here as they apply to normal functions, where  $A$  is a single factor of a term and  $X$  and  $Y$  are either single or multiple factors of a term: (a) Replace  $X + XY$  by  $X$ . (b) Re-

place  $A + \bar{A}X$  by  $A + X$  and  $\bar{A} + AX$  by  $\bar{A} + X$ . (c) Replace  $AX + \bar{A}X$  by  $X$ . The third of these simplifications is not necessary, but it is frequently convenient. The next process, expansion of the simplified function, consists of adding *terms of consensus* obtained in the following manner: For any two terms  $AX + \bar{A}Y$ , add the term  $XY$ . Of course, if a factor in  $X$  appears also in  $Y$  in complementary form, then  $XY$  would be zero and no term would be added.

These two processes, simplification by inspection and adding terms of consensus, must be repeated until no more changes occur, or until an "oscillation" sets in, that is, all new terms of consensus would be removed by the simplification process. The resulting terms are the *prime implicants*.

*Step 2. Finding the Essential Prime Implicants.* The essential prime implicant terms can be found by forming a table. The rows of the table are defined by the prime implicants. The columns of the table are defined by all minterms of the function (and hence, all minterms of the prime implicants). The procedure for finding the essential terms consists of entering a check mark under each minterm column included in each prime implicant. When all check marks for each row have been entered, encircle all checks that occur singly in a column. All prime implicants in rows containing an encircled check are *essential terms*. Any simplest function must include the essential prime implicants.

Draw a line through each row having an encircled check; then draw a line through every column containing a check mark with a line through it. If all the columns are lined out by this process, the simplest equivalent function has been found—the *or* of the essential terms is the simplest equivalent. If there are columns not ruled out by this process, Step 3 must be performed.

*Step 3. Finding the Simplest Combination of Remaining Prime Implicants to Complete the Function.* The remaining minterms are to be expressed by the simplest combination of remaining prime implicants. It is possible to get several equally simple expressions at this point.

In most situations the function can be completed by inspection. Alternatively, a new table can be constructed consisting only of the remaining minterm columns, and the rows of prime implicants which are not essential, but having a check mark in one of the remaining minterm columns. This table is referred to as the *reduced table of prime implicants*. The problem of selecting reduced prime implicants to most simply represent the minterms, nevertheless, is a cut-and-try procedure. References 11 and 12 are a more thorough investigation of this problem.

### Examples of Quine Simplification.

EXAMPLE 1. Function:  $AB + \bar{A}\bar{B} + A\bar{B}C$ .

Step 1.  $AB + \bar{A}\bar{B} + A\bar{B}C$  reduced

$AB + \bar{A}\bar{B} + AC (+\bar{B}C)$  added term of consensus

These are the prime implicants.

Step 2.

Table of Prime Implicants for Example 1

Prime Implicants	Minterms				
	$ABC\bar{C}$	$ABC$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}C$	$A\bar{B}C$
$AB$	⊗	✓			
$\bar{A}\bar{B}$			⊗	✓	
$AC$		✓			✓
$\bar{B}C$				✓	✓

The core is  $AB + \bar{A}\bar{B}$  identified by rows containing encircled checks.

Step 3. The remaining minterm to be included by the function is  $A\bar{B}C$ .

Reduced Table of Prime Implicants for Example 1

	$A\bar{B}C$
$AC$	✓
$\bar{B}C$	✓

Either of the terms  $AC$  and  $\bar{B}C$  complete the function, hence there are two equally simple equivalent functions to that above. The result is:

$$AB + \bar{A}\bar{B} + AC \quad \text{or} \quad AB + \bar{A}\bar{B} + \bar{B}C.$$

EXAMPLE 2. A common design situation is to obtain a function in canonical minterm form. This example shows the complete Quine simplification procedure for the following function:

$$\begin{aligned} &\bar{A}\bar{B}C\bar{D} + \bar{A}\bar{B}CD + \bar{A}B\bar{C}\bar{D} + \bar{A}B\bar{C}D + \bar{A}BCD \\ &\quad + A\bar{B}\bar{C}\bar{D} + A\bar{B}C\bar{D} + AB\bar{C}\bar{D} + ABC\bar{D} + ABCD. \end{aligned}$$

Step 1. The first reduction process in finding the prime implicants is not unique, but one such reduction is shown.

$$\begin{aligned} &\bar{A}\bar{B}C \qquad \qquad \bar{A}B\bar{C} \qquad \qquad BCD \\ (\bar{A}\bar{B}C\bar{D} + \bar{A}\bar{B}CD) &+ (\bar{A}B\bar{C}\bar{D} + \bar{A}B\bar{C}D) + (\bar{A}BCD + ABCD) \\ &\qquad \qquad \qquad A\bar{B}D \qquad \qquad AB\bar{D} \\ &\quad + (A\bar{B}\bar{C}\bar{D} + A\bar{B}C\bar{D}) + (AB\bar{C}\bar{D} + ABC\bar{D}) \\ &\qquad \qquad \qquad = \bar{A}\bar{B}C + \bar{A}B\bar{C} + BCD + A\bar{B}D + AB\bar{D}, \end{aligned}$$



Essential terms:  $\bar{A}\bar{B}C$ ,  $A\bar{B}D$ .

Step 3.

*Reduced Table of Prime Implicants for Example 2*

	$\bar{A}\bar{B}\bar{C}\bar{D}$	$\bar{A}\bar{B}C\bar{D}$	$\bar{A}BCD$	$AB\bar{C}\bar{D}$	$ABC\bar{D}$	$ABCD$
$CD$			✓			✓
$\bar{A}\bar{B}\bar{C}$	✓	✓				
$A\bar{B}\bar{D}$				✓	✓	
$\bar{A}BD$		✓	✓			
$B\bar{C}\bar{D}$	✓			✓		
$ABC$					✓	✓

Since each of these prime implicants represents only two minterm columns, it requires at least three prime implicants to represent all six minterms. If only three are required, and if  $CD$  can be one of these three, it will represent the simplest solutions since it has one fewer factors than the other prime implicants. Such a choice is possible with the first three prime implicants,  $CD$ ,  $\bar{A}\bar{B}\bar{C}$ ,  $A\bar{B}\bar{D}$ . By combining these terms with the essential terms, the simplest equivalent function is

$$\bar{A}\bar{B}C + A\bar{B}D + CD + \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{D}.$$

**Veitch Diagram Simplification.** The Veitch diagram (see Veitch Diagrams) is a convenient representation for the simplification of functions because many of the rules of simplification can be deduced by geometrical consideration (see Ref. 1). The relationship between these rules and the geometry can best be learned from examples. The following four functions of two variables can be readily simplified:

$$\bar{A}B + AB = B,$$

$$\bar{A}\bar{B} + A\bar{B} = \bar{B},$$

$$A\bar{B} + AB = A,$$

$$\bar{A}\bar{B} + \bar{A}B = \bar{A}.$$

The Veitch diagrams of these four functions are shown in Fig. 8a, b, c, d. There are two other functions of two variables which are composed of two minterms which do not reduce:

$$\bar{A}B + A\bar{B},$$

$$\bar{A}\bar{B} + AB.$$

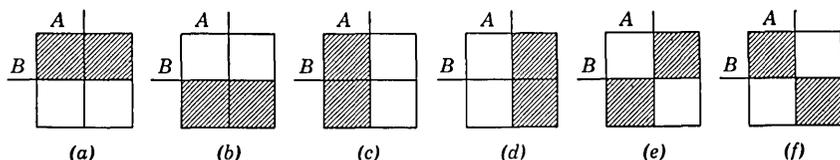


FIG. 8. Geometric relationships.

The Veitch diagrams of these two functions are shown in Fig. 8e, f. It is evident from the figure that those normal functions which can be simplified (a, b, c, and d) occupy adjoining squares on the Veitch diagram; those which cannot be simplified (e and f) are not adjoining.

These relationships can be used to simplify systematically any function in normal form. The function with a Veitch diagram of Fig. 9 can be written by minterms as  $AB + A\bar{B} + \bar{A}B$ ; but from the geometry it can be deduced that  $B + A\bar{B}$  and  $A + \bar{A}B$  also represent this function; and a simplest normal representation of the function is given by  $A + B$ .

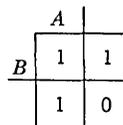


FIG. 9. Veitch diagram for  $AB + A\bar{B} + \bar{A}B$ .

The rules of geometrical relationships do not always require that the related squares in the Veitch diagram form a cluster, since for more than two variables different arrangements of the terms are possible. An example

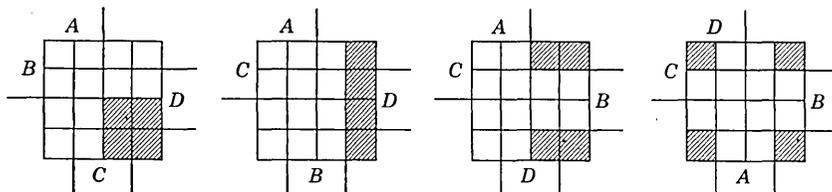


FIG. 10. Equivalent arrangements of the minterms of  $\bar{A} \cdot \bar{B}$ .

of the term  $\bar{A}\bar{B}$  of a function of four variables is shown in Fig. 10, in which the same relationship is involved, but the geometrical arrangement differs in each instance. For four variables, any four minterms on a Veitch diagram can be associated if they form a 2 by 2 square, a 1 by 4 rectangle, a pair of 1 by 2 rectangles symmetrically located on opposite edges, or 4 squares in opposite corners.

**EXAMPLE 1.** A full binary adder has three inputs ( $A, B, C$ ) and two outputs ( $S, K$ ). The sum output  $S$  is 1 whenever exactly one or all three of the inputs are 1. The carry output  $K$  is 1 whenever any two or all

three of the inputs are 1.

$$S = A\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}C + ABC,$$

$$K = AB\bar{C} + A\bar{B}C + \bar{A}BC + ABC.$$

Veitch diagrams of these two functions are shown in Fig. 11 and Fig. 12. It is evident that the sum function cannot be reduced in normal form since

	A			
B	0	1	0	1
	1	0	1	0
	C			

FIG. 11. Sum function ( $S$ ).

	A			
B	1	1	1	0
	0	1	0	0
	C			

FIG. 12. Carry function ( $K$ ).

the 1's entered into the Veitch diagram are geometrically unrelated. However, the carry function can be simplified, its simplest normal representation being

$$K = AB + AC + BC.$$

**EXAMPLE 2.** The function described by the Veitch diagram of Fig. 13 has as its simplest normal equivalent function

$$ABC + A\bar{B}D + \bar{A}\bar{B}C + \bar{A}BD.$$

Note that the center cluster of four 1's ( $CD$ ) is not a term in the simplest representation. This example points up a need for systematically obtaining the simplest normal function from the Veitch diagram.

	A			
B	0	1	0	0
	0	1	1	1
	1	1	1	0
	0	0	1	0
	C			

FIG. 13. Veitch diagram for Example 2.

	A			
B	0	0	0	0
	0	1	0	0
	0	1	1	1
	0	0	1	1
	C			

FIG. 14. Veitch diagram illustrating the Quine simplification.

Such a systematic approach can be obtained by adapting Quine simplification to the Veitch diagram. This adaptation results in a two-step procedure: (1) Find the essential terms. (2) Find the best representation for the minterms which are not included in the essential terms.

*Step 1.* The essential terms on a Veitch diagram are determined by the following criteria:

(a) Any minterm which is geometrically unrelated to any other minterm is an essential term. *Example.* All minterms of the sum function of Fig. 11 are essential terms.

(b) Any minterm which is geometrically related to *only one* other minterm or group of minterms has the largest such grouping as an essential term.

The Veitch diagram of Fig. 14 shows a function completely defined by the two essential terms in the shaded areas,  $ACD + \bar{A}\bar{B}$ . Those minterm squares containing dots require that the associated term be essential.

*Step 2.* Finding the best representation of minterms not included in the essential terms is sometimes obscured because often there may be more than one equally simple function. In this respect, it is like working with the reduced table of prime implicants in the Quine method. To show this, Example 2 of the Quine simplification procedure is repeated here; see Fig. 15.

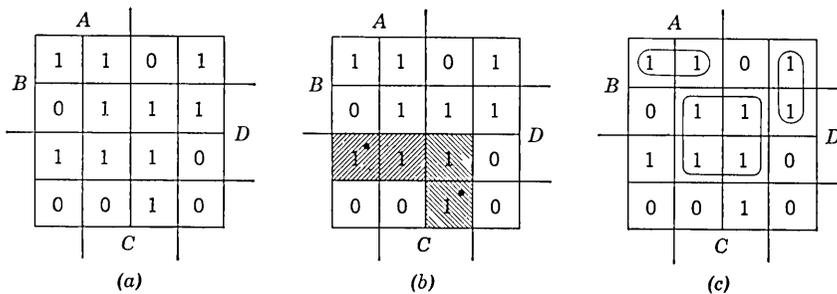


FIG. 15. Simplest normal equivalent function:  $A\bar{B}D + \bar{A}\bar{B}C + CD + AB\bar{D} + \bar{A}\bar{B}\bar{C}$ . (a) Function (see Example 2, Quine simplification procedure), (b) essential terms  $A\bar{B}D + \bar{A}\bar{B}C$ , (c) best representation of remaining minterms  $CD + AB\bar{D} + \bar{A}\bar{B}\bar{C}$ .

**Redundancies.** Many switching functions of interest have either (a) nonoccurring combinations of variables or (b) functional values which are not of interest for certain combinations of variables. In either case, it is often possible to simplify a function by properly utilizing these *redundancies* or “don’t care” conditions.

For Veitch diagram simplifications, all minterms which are redundant will have an X placed in their minterm square; all other squares will have functional values of 1 or 0. Redundancies are utilized as 1’s whenever their use will simplify a term, or they are ignored otherwise. Two examples with binary-coded decimal numbers will be given. Figure 16a shows a diagram with the decimal value of four binary bits  $A_1, A_2, A_4, A_8$  which

are weighted 1, 2, 4, and 8 respectively. For the decimal digits 0 through 9, this is referred to as binary-coded decimal representation. Assuming that representations for 10 through 15 do not occur, Fig. 16*b* shows the use of a redundancy to specify the digit 3 as  $A_1 \cdot A_2 \cdot \bar{A}_4$ .

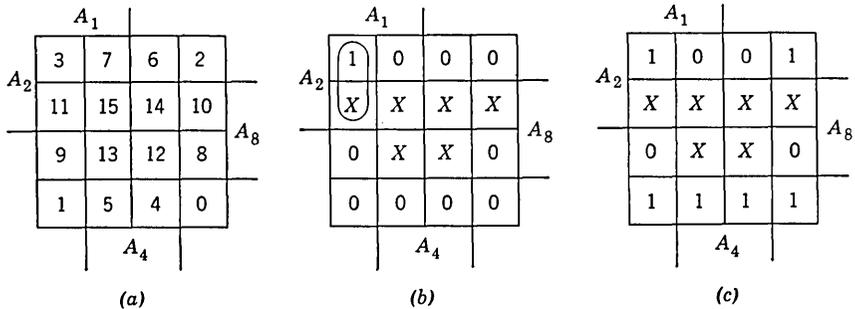


FIG. 16. Examples illustrating redundancies. (a) Decimal equivalents, (b) decimal digit 3, (c) decimal digits  $\leq 5$ .

Figure 16*c* shows the Veitch diagram of a function which detects all decimal digits less than, or equal to, five. There are three equally simple representations of this function; however, since the redundancies are used differently, the functions are not equivalent for all minterms.

$$\bar{A}_4 \cdot \bar{A}_8 + \bar{A}_2 \cdot \bar{A}_8; \quad A_2 \cdot \bar{A}_4 + \bar{A}_2 \cdot \bar{A}_8; \quad \bar{A}_4 \cdot \bar{A}_8 + \bar{A}_2 \cdot A_4.$$

**Simplification Including Storage Elements.** Techniques have also been developed for determining the minimum number of storage elements required for a given logical network (see Refs. 13 and 14). These techniques are usually more applicable to well-defined portions of a computer design than to the whole design.

### 3. PRELIMINARY DESIGN CONSIDERATIONS

#### System Aspects

A computing system is typically made up of a number of subsystems such as input, output, storage, and arithmetic and control. The integration of these subsystems must be considered early in the design effort. For example, input might consist of multiple magnetic tape units, punched card readers, analog-to-digital converters, and a keyboard for manual entry of data. Each of these input devices has its own peculiar characteristics—it may provide digital information at a different rate, on a different number of wires, in a different block size, and with a different word composition.

The following discussion assumes the usual situation in which the arithmetic and control form the hub of the system.

### Matching Input and Output.

*Direct Presentation.* One method of incorporating inputs with arithmetic and control is the direct presentation of "raw" digital information to one or more of the arithmetic registers of the computer at the rate at which it is being read. This technique has the advantage of being economical to mechanize, but suffers from two disadvantages. First, the computer usually is slowed down because it must operate at the speed of slower input devices. Second, the arithmetic and control unit becomes specialized with respect to the input devices. That is, the computer registers must be designed to accept the input information at the input rate, and perhaps in an undesirable format.

*Buffer Storage.* A second method of incorporating inputs is to provide a buffer storage. The buffer storage accepts information at a rate commensurate with the input devices, and the computer then takes the information from the buffer at a higher rate. It is also possible to reorganize data which normally are not read in the desired form. For example, serial information can be put into parallel form, or punched card information can be suitably organized into word size.

*Extension of Buffer Storage.* A third method of incorporating inputs is gaining favor, particularly for large-scale systems. This method is an extension of the buffer storage technique, and it involves additional equipment which makes the input buffer a computer in its own right. In addition to acting merely as buffer storage for input information, this device has the ability to search for information—a feature especially useful for magnetic tape. It is imperative that the input computer be suitably interlocked with the central computer. ▶

Computer output can be coordinated with arithmetic and control by techniques analogous to the three methods discussed for inputs: direct, buffer, or output computer.

**Storage.** Usually the arithmetic and control unit communicates directly with the storage unit through one or more of its registers. The transmission of information is ordinarily in one of these three forms: serial by bit, parallel by word, or serial-parallel (serial by character). Two aspects of the main storage bear heavily on the logical design.

1. Speed compatibility of the storage and the arithmetic circuits may influence the choice of serial or parallel operation.

2. The type of storage, serial or random access, may affect both arithmetic speed requirements and the type of instructions.

There is an increasing tendency in large-scale computers to provide a buffer storage for instructions and operands which are about to be used. Since the determination of which instructions and operands will be needed is rather complex (especially with automatic address modification and program branch points), such a buffer device is in itself a computer.

**Arithmetic and Control Aspects.** Before the logical design can proceed, decisions must be made as to the representation of numbers and instructions, the arithmetic operations to be performed, and the arithmetic techniques which will be employed (see Ref. 2).

**Instructions.** (See Chaps. 2 and 12 for greater detail.) The number of addresses of an instruction are generally specified for arithmetic operations. For example, a *three address* instruction specifies two storage addresses from which to obtain operands and an address at which the result is to be stored. The most prevalent address structures for instructions are one address, two addresses, and three addresses. In addition to operand and result addresses, the address from which the next instruction is to be obtained must be specified. Most large-scale computers use a *program address counter* for this purpose, selecting instructions from sequential addresses in storage. Deviations from this pattern are referred to as *branching* or *program transfers*. Program transfer instructions therefore explicitly contain an instruction address. Computers with a serial main storage (such as a magnetic drum) often have in each instruction an explicit address from which the next instruction will be obtained. Alternatively, a modified form of a program address counter is used.

For computers with a fixed word length, it is desirable to have numbers and instructions either the same length, or one having a length that is an integral multiple of the other. This permits the efficient use of storage locations for either instructions or numbers, and allows arithmetic operations on instructions.

**Number Representation.** (See Chaps. 2 and 12 for greater detail.) The categories of number representation commonly employed in electronic computers are *alphanumeric*, *decimal*, and *binary*. Alphanumeric coding is a representation of an alphabetic or decimal symbol by combinations of six or more binary bits. The decimal symbol is a combination of four or more bits, the most common representation being *binary-coded decimal* which has four bits with "weights" of 1, 2, 4, and 8. Some decimal codes employ different weights which are capable of expressing every decimal digit 0 through 9; other decimal codes, such as the *excess three code*, have no column weighting associated with them. The "pure" binary number system is usually used for internal computing because it is economical. Negative numbers are commonly represented in one of several different ways in binary computers, for example, *minus sign and magnitude*, *2's complement*, and *1's complement*. The last two are useful because of their properties in subtraction.

**Arithmetic Operations.** (See Chap. 18 for greater detail.) The choice of techniques for performing arithmetic operations will in a large measure be determined by the operational speed requirements and the way in

which numbers are presented to the arithmetic unit. If operations are performed on operands as they appear serially in time, the operation is *serial*; if operands are utilized in parallel, the operation is *parallel*; if one operand is used serially and the other in parallel (for example, in multiplication), the operation is *serial-parallel*. Generally, the more parallel the operation, the greater the speed achieved and the more equipment required. Operands may themselves appear in serial-parallel form. For example, decimal codes may be transmitted parallel by bit, serial by decimal digit.

### **Preliminary Design of the Arithmetic and Control Unit**

When the required capabilities of the computer have been specified, the logical design can begin.

**Separation.** If possible, a breakdown of the logical design into independent parts should be made. This is especially important for complex computers which will require a parallel effort of several logical designers, since such a breakdown will enable them to work independently. There usually are other benefits of such an approach: the education of others on the workings of the computer is simplified, the isolation of malfunctioning parts of the completed computer is simpler, and the layout of the computer for packaging purposes is improved.

Certain portions of a computer quite naturally separate out. For instance, arithmetic and instruction registers with their associated control flip-flops can usually be singled out. Also, arithmetic equipment for various arithmetic operations may be separable. The most difficult part of the arithmetic and control unit to sectionalize is that associated with control. Three techniques in common use are:

1. Group the controls which are needed for similar types of instructions.
2. Break up the execution of each instruction into *suboperations*. (Usually different instructions will have some suboperations in common.) Group the controls for a suboperation.
3. Determine the different control *states* required of the computer to execute all types of instructions. Group the controls for a given state.

**Manual Control and Display.** These items, unfortunately, are often dealt with as afterthoughts. If these requirements are specified and borne in mind during the initial design effort, it is often possible to include them with little additional electronic equipment. There are few generalizations that can be made concerning manual control. However, one universal problem is the ability to start a computer. This is usually accomplished by providing a suitable starting instruction or instruction address manually, or by having an implied instruction or instruction address. In order to start satisfactorily, it is necessary to have some of the computer flip-flops

in known states. Considerable ingenuity can be used to determine which flip-flops *must have* known states, and of these, which *must be forced* into these states.

A word of warning with respect to manual control—manual intervention into the operation of a high-speed computer is expensive in time and is fraught with possibilities of introducing errors. The need for these controls for program correction or maintenance should be clearly established.

### Representation of Logical Design

Diagrammatic techniques of recording logical design are used to a large extent, however, symbolic (algebraic) design is becoming increasingly popular because of advantages in a number of areas.

**Block Diagrams.** The interconnection of computer elements, which constitutes the logical design of a machine, can be depicted by equivalent interconnections of blocks or circuit symbols representing these elements. Although several sets of circuit symbols have been employed, none has achieved universal adoption, see Table 1, Sect. 1.

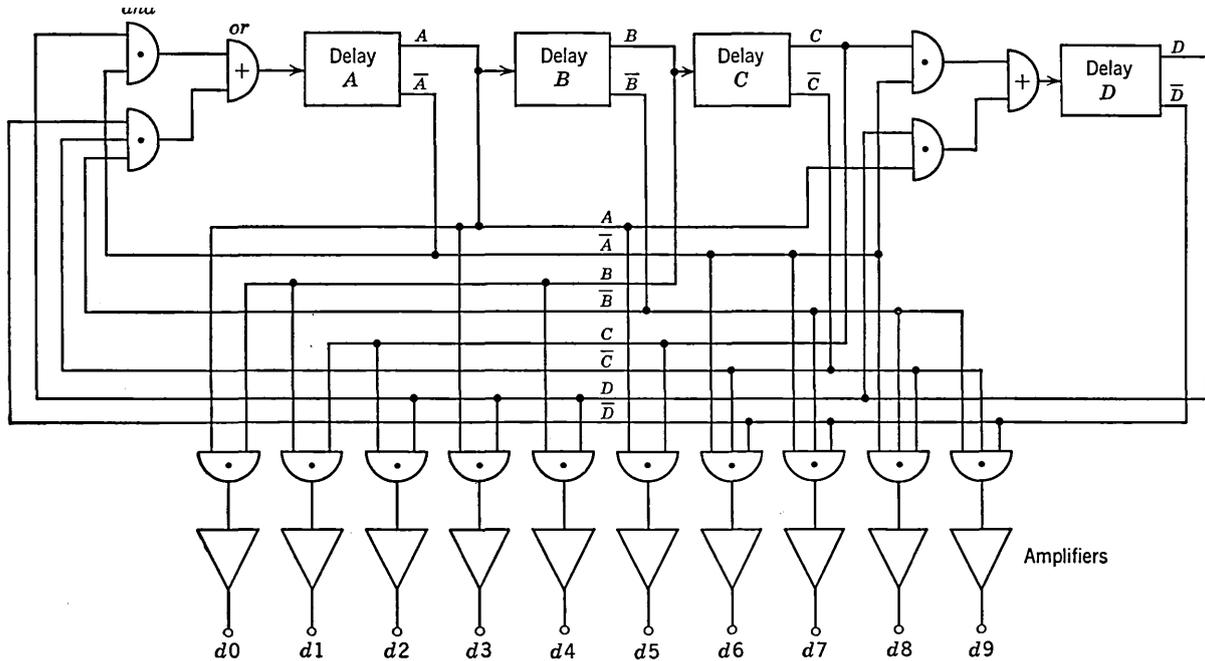
Usually, a block is chosen to correspond to a physical unit in the computer so that the input and output terminals also represent plug connections. This permits the use of the logical diagram as a wiring diagram also (see Refs. 5 and 15). Figure 17*a* gives the block diagram for the ten-stage counter used as a design example in Sect. 4. The block diagram uses gates, delay elements with two outputs, and amplifiers. For comparison the logical equations for the counter are given in Fig. 17*b*.

*Advantages of the block diagram:* (1) It affords a "picture" of the logical network which often enhances intuitive understanding. (This is particularly true of certain highly developed representations of contact networks.) (2) It usually serves as a wiring diagram. (3) It is easily understood by maintenance personnel.

*Disadvantages of the block diagram:* (1) Complex control circuits present a formidable layout problem that requires a substantial amount of time, complicates changes, and thereby discourages improvements. (2) The pictorial representation may obscure rather obvious formal simplifications.

**Symbolic Design** (Ref. 16). Algebraic representation of computer design is coming into widespread use for the following reasons: (1) Continued improvement in formal techniques for synthesis and simplification tends to generate a symbolic design. (2) The use of computers for the design of computers requires a machine language. *Examples.* Logical simulation to detect logical design errors, layout and wiring calculations, tabulations of loading on outputs. (3) Revisions in design are easily incorporated.

Symbols for computer elements should be chosen which are unique and,



(a)

*Storage Element Inputs*  
 $I(A) = \bar{A}D + \bar{B}\bar{C}\bar{D}$   
 $I(B) = A$   
 $I(C) = B$   
 $I(D) = \bar{A}C + AD$

*Amplifier Outputs*  
 $d0 = AB$   
 $d1 = BC$   
 $d2 = \bar{C}D$   
 $d3 = AD$   
 $d4 = BD$   
 $d5 = AC$   
 $d6 = \bar{A}\bar{C}\bar{D}$   
 $d7 = \bar{A}\bar{B}\bar{D}$   
 $d8 = \bar{A}\bar{B}\bar{C}$   
 $d9 = \bar{B}\bar{C}\bar{D}$

(b)

Fig. 17. Design of a ten-state master counter. (a) Block diagram, (b) logical equations corresponding to (a). (See Sect. 4.)

if possible, provide a mnemonic association. Inputs and outputs to an element are easily described by augmenting the symbol describing the basic element. For example, a  $J$ - $K$  flip-flop which occupies the thirteenth bit position of the  $B$  register of a computer could be identified by the symbols of Fig. 18. Here the inputs have  $J$  and  $K$  added to the flip-flop designations. The normal output has been assigned the same symbol as the flip-flop itself—usually there is no difficulty in distinguishing between the two. The output symbol,  $B13'$ , is frequently used rather than  $\overline{B13}$  because it is more easily typed.

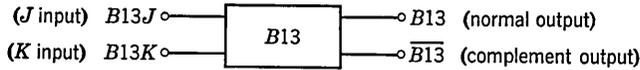


FIG. 18. Typical flip-flop symbolic notation.

It is extremely useful to have the completed design on unit records such as punched cards. This permits sorting, merging, duplication, and tabulation by standard card equipment, and also facilitates revisions. Punched card representation of symbols requires the use of capital letters only and may restrict the use of special symbols. For example, the complement output of Fig. 18 could be represented as  $B13C$  or  $B13-$ .

**The First Design Attempt.** A good place to start in the detailed design of a computer is to consider that operation which will require the most equipment. Then, as additional requirements are added, an attempt should be made to utilize as much of the equipment already in use as possible. It is important to determine the flip-flops that will be required and to specify their functional roles, since the detailed logical design will require it. As an example of this approach, consider the logical design of an arithmetic unit which will have serial-parallel multiplication but will perform the other arithmetic operations serially. It is quite likely that providing a minimum of equipment for the multiplication will provide nearly enough equipment to perform the rest of the arithmetic operations.

As the logical design proceeds, there may arise a need for miscellaneous control flip-flops. It is usually advantageous to carry these as separate flip-flops and attempt to time-share them only after the design has been quite well established, since minor changes can upset the economics of time-sharing. It may be desirable to redefine the roles of flip-flops as the logical design proceeds, or even change some of the system specifications to achieve a balanced design.

#### 4. DETAILED LOGICAL DESIGN

The preliminary planning of the logical design of a computer involves a determination of the storage elements to be used and their functional

roles. This plan provides a foundation on which the detailed logical design can be built. The detailed logical design is a statement of the interconnections of computer elements required to mechanize the desired computer. It will be assumed in the following discussion that the logical design is specified by input equations to flip-flops, although the complete design may require other specifications also. (See Ref. 1.)

The input equations must be in a form compatible with the computer elements used and their permissible arrangement.

### Time Difference Equations

If a storage element retains information for one time interval, its time difference equation relates its output at time  $\tau + 1$  to its input and its previous state at time  $\tau$ . If the difference equation of a flip-flop is known for all time (for any  $\tau$ ), the complete input gating required for this flip-flop can be determined. Since the functional roles of all flip-flops are known, their difference equations (and hence their inputs) can be defined. Knowledge of the role of a flip-flop usually defines its output (or state) in one of the following three ways: (1) The output as a time sequence. *Examples.* Sequencing control, timing. (2) The output as a combination of inputs. *Examples.* Accumulator, shifting register, error detector. (3) The output as time sequences of combinations. *Examples.* Time-shared registers, operation control.

**Outputs Given as a Time Sequence.** It is typical of many of the control flip-flops of a computer that the preliminary design provides a description of the outputs as 1's or 0's during each time interval. For example, a timing flip-flop may be required to produce a 1 during the fifth time interval of the execution of each instruction, and a 0 at all other times. This output is stimulated by its input during the fourth time interval, and the problem becomes that of uniquely specifying the fourth time interval. If the preliminary design does not provide a unique identification of this time interval, then it must be modified accordingly.

Knowledge of the output sequence  $F$  of a flip-flop  $U$  does not provide a difference equation directly, but requires specification of the input sequence  $G$ :

$$\begin{aligned} U^{\tau+1} &= F^{\tau+1} \quad (\text{output sequence}) \\ &= G^{\tau} \quad (\text{input sequence}). \end{aligned}$$

Frequently the inputs for a flip-flop with defined outputs can be written directly; in the example above, the simplest unique identification for the fourth time interval would provide the simplest input equation. Input equations can also be derived from difference equations. The following example of a ten-state counter shows the complete transition from a known output sequence to input equations for various types of flip-flops.

If it does not matter what the outputs of a flip-flop are for certain time intervals, it may be possible either to simplify the input equations or to *time-share* the flip-flop, that is, to assign it some other role during this period of time.

**Outputs Given as a Combination of Inputs.** It is typical of flip-flops in the arithmetic portion of a computer to have outputs which depend on the information being processed. It is not known in advance whether the state of such a flip-flop will be 0 or 1 at a particular time, since it will depend on the instructions or numbers of the computer program. The difference equation for flip-flop  $U$  is immediately obtained from the input function  $G$ :

$$U^{r+1} = G^r.$$

The input equations for flip-flops are derived directly from their difference equations as shown in the example of a ten-state counter which follows.

**Outputs Given as Time Sequences of Combinations.** Most control flip-flops have a different sequence of outputs in executing different instructions, and hence depend on particular operation code combinations. Similarly, most arithmetic flip-flops have a somewhat different role during particular time intervals. These cases are a combination of the two previously described, and the flip-flop input equations can either be written directly or be deduced from the difference equations.

### Design of a Ten-State Counter

The following example will illustrate the following points: (1) the use of the Veitch diagram as an aid to creative logical design; (2) obtaining time difference equations for flip-flops; (3) obtaining equations for flip-flop inputs.

**The Problem.** It is desired to have a four flip-flop counter with any ten unique states which repeat. The states of the counter will be used in many places for timing; hence, it is desirable to reduce the gating required to sense each count, and thereby also reduce the output loading of the counter flip-flops.

**Determination of the Ten States.** The four flip-flops will be called  $A$ ,  $B$ ,  $C$ , and  $D$ . It should be noted that no count can be specified by just one of the four flip-flops, since this would permit at most nine counts. At least one count can be specified by two flip-flops. Figure 19a shows a Veitch diagram with an arbitrarily placed count which, by virtue of its associated three redundant minterms (hatched), allow it to be specified by  $AB$ . *Note.* Redundant minterms are usually marked by an  $x$ ; however, in these examples such squares are shaded to show the symmetrical configurations. Figure 19b shows an efficient extension of the redundant nucleus such that six counts are achieved which are specified by only

two of the four flip-flops. In order to obtain ten counts, only one more minterm of the remaining five can be selected as a redundancy.

Figure 19c shows the final choice which again has maximum efficiency, but requires sensing three of the four flip-flops to obtain the remaining

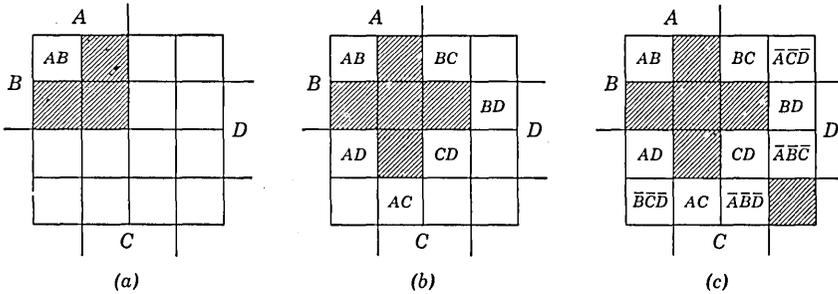


FIG. 19. Determining counts which require limited sensing. (a) One count, (b) six counts, (c) ten counts.

four counts. In Fig. 20 the ten counts have been labeled  $d0$  to  $d9$ , and the table of the figure shows the values of the flip-flops for all counts, and the coincidence required to recognize a particular count. The choice of

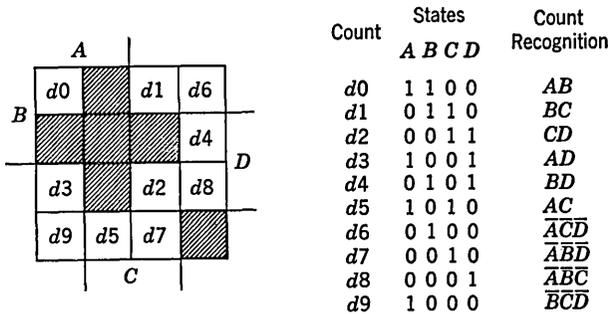


FIG. 20. Assignment of counter sequence.

the sequence of counts bears heavily on the input gating that will be required to generate it. There are no simple techniques which give the best choice of sequence. This choice will, of course, depend on the properties of the storage element used.

**Inputs for Flip-Flop A, B, C, and D.** The inputs for the flip-flops can be obtained directly from the time difference equations.

**Difference Equations for A, B, C, and D.** In Fig. 20 it can be seen that A has a value of 1 for  $d0$ ,  $d3$ ,  $d5$ , and  $d9$ ; thus for an arbitrary time

interval  $\tau + 1$ :

$$A^{\tau+1} = (d0 + d3 + d5 + d9)^{\tau+1}.$$

Note that this is not a difference equation since the time superscripts are the same on both sides of the equation. To obtain a difference equation, the right-hand side of the equation is replaced by its preceding count:

$$A^{\tau+1} = (d9 + d2 + d4 + d8)^{\tau}.$$

A substitution for  $d9$ ,  $d2$ ,  $d4$ , and  $d8$ , in terms of  $A$ ,  $B$ ,  $C$ , and  $D$  provides the desired function (together with the "don't care" conditions for simplification). This is readily shown in the diagram of Fig. 21, where the symbol  $\tau$  has been dropped from  $A^{\tau+1}$ . The remaining difference equations can be obtained in the same manner. The results are summarized below:

$$A^{+1} = \bar{A}D + \bar{B}\bar{C}\bar{D},$$

$$B^{+1} = A,$$

$$C^{+1} = B,$$

$$D^{+1} = \bar{A}C + AD.$$

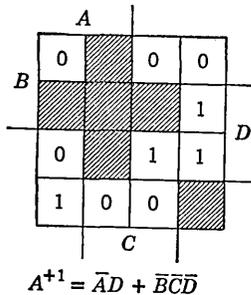


FIG. 21. Diagram of  $A^{+1}$ .

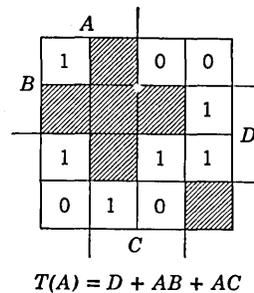


FIG. 22. Diagram of  $T(A)$ .

**Delay Flip-Flops.** If  $A$ ,  $B$ ,  $C$ , and  $D$  are delay flip-flops, their inputs  $I$  are the same as the difference equation.

$$I(A) = \bar{A}D + \bar{B}\bar{C}\bar{D},$$

$$I(B) = A,$$

$$I(C) = B,$$

$$I(D) = \bar{A}C + AD.$$

Time does not appear in the total input equations for flip-flops, since the gating structure of the computer is fixed.

**Trigger Flip-Flops.** If flip-flop  $A$  is a trigger flip-flop, its input  $T(A)$  is defined by the diagram of Fig. 22. Note that it differs from the differ-

ence equation for  $A$  (Fig. 20) in that 1's and 0's are interchanged in the columns for  $A = 1$ . This is because the flip-flop is in the 1's state, and it is desired that it remain in the 1's state for the next time interval, requiring an input of zero.

The inputs required for trigger flip-flops  $A$ ,  $B$ ,  $C$ , and  $D$  are:

$$T(A) = D + A(B + C)$$

$$T(B) = \bar{A}B + A\bar{B}$$

$$T(C) = \bar{B}C + B\bar{C}$$

$$T(D) = \bar{A}(\bar{C}D + C\bar{D})$$

**R-S Flip-Flops.** If flip-flop  $A$  is an  $R$ - $S$  flip-flop, its inputs  $S(A)$  and  $R(A)$  are diagrammed in Fig. 23. Note that additional redundancies have been added to those of the difference equation of Fig. 21. This

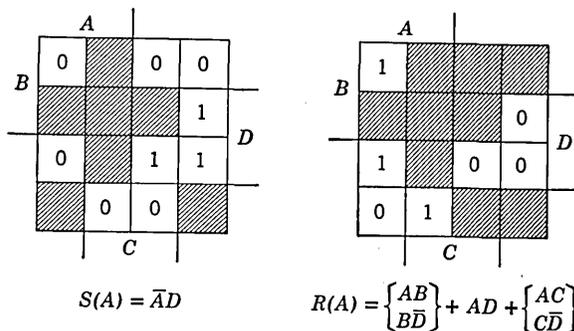


Fig. 23. Inputs to  $R$ - $S$  flip-flop  $A$ .

happens because there is not always a unique input sequence for a given output sequence. For example, set inputs do not have to be applied a second time if an  $R$ - $S$  flip-flop is to remain in the 1's state two successive time intervals. The diagram for  $R(A)$  has entries which are the complement of these in the difference equation, and all 0's for  $A = 0$  become redundant. The  $R(A)$  input equation is not unique, as shown by the equivalent terms within braces.

Inputs for  $R$ - $S$  flip-flops  $A$ ,  $B$ ,  $C$ , and  $D$  are:

$$S(A) = \bar{A}D, \quad R(A) = \left\{ \begin{matrix} AB \\ B\bar{D} \end{matrix} \right\} + AD + \left\{ \begin{matrix} AC \\ C\bar{D} \end{matrix} \right\},$$

$$S(B) = A, \quad R(B) = \bar{A},$$

$$S(C) = B, \quad R(C) = \bar{B},$$

$$S(D) = \bar{A}C. \quad R(D) = \bar{A}\bar{C}.$$

**J-K Flip-Flops.** If flip-flop  $A$  is a  $J$ - $K$  flip-flop, its inputs  $J(A)$  and  $K(A)$  are diagrammed in Fig. 24. Note that  $J$  inputs can be anything if the state of the flip-flop is 1 in the same time interval. Similarly,  $K$

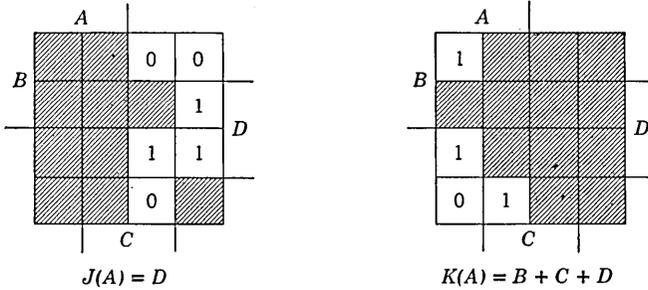


FIG. 24. Inputs to  $J$ - $K$  flip-flop  $A$ .

inputs can be anything if the state of the flip-flop is 0 in the same time interval. Minimal equations for  $J$  and  $K$  inputs never include the output of the same flip-flop.

Inputs for  $J$ - $K$  flip-flops,  $A$ ,  $B$ ,  $C$ , and  $D$  are:

$$\begin{aligned} J(A) &= D, & K(A) &= B + C + D, \\ J(B) &= A, & K(B) &= \bar{A}, \\ J(C) &= B, & K(C) &= \bar{B}, \\ J(D) &= \bar{A}C. & K(D) &= \bar{A}\bar{C}. \end{aligned}$$

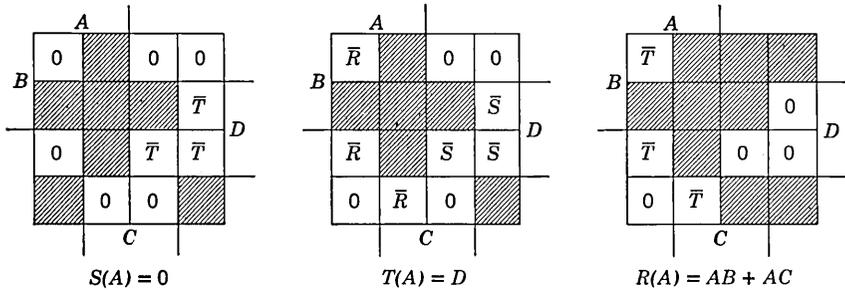


FIG. 25. Inputs to  $R$ - $S$ - $T$  flip-flop  $A$ .

**R-S-T Flip-Flops.** If flip-flop  $A$  is an  $R$ - $S$ - $T$  flip-flop, its inputs  $R(A)$ ,  $S(A)$ , and  $T(A)$  are diagrammed in Fig. 25. In several instances  $\bar{R}$ ,  $\bar{S}$ , or  $\bar{T}$  appear as entries rather than 0 or 1; for example,  $\bar{T}$  appears in  $S(A)$  and  $\bar{S}$  in the same position of  $T(A)$ . This implies that equivalent operation can be obtained by a set or a trigger input, but both are not

allowed simultaneously. The equations shown are not unique—the terms shown for  $R(A)$  could be put in  $T(A)$  instead.

Inputs for  $R$ - $S$ - $T$  flip-flops  $A$ ,  $B$ ,  $C$ , and  $D$  are:

$$\begin{array}{lll} S(A) = 0, & R(A) = A(B + C), & T(A) = D, \\ S(B) = A, & R(B) = \bar{A}, & T(B) = 0, \\ S(C) = B, & R(C) = \bar{B}, & T(C) = 0, \\ S(D) = \bar{A}C. & R(D) = \bar{A}\bar{C}. & T(D) = 0. \end{array}$$

### Further Considerations

To complete the design of this counter, three more items should be considered: (1) the initiation of each count, (2) presetting of the count, (3) behavior of the counter in unused states.

**Count Initiation.** The counter as described by the final equations will produce a count each time interval. That is, the storage or decision elements are internally acted upon by clock signals so that the counter continually sequences through its ten states.

If it is desired to count external pulses, these counting signals must be introduced into the input equations for the storage elements. For the  $T$ ,  $R$ - $S$ ,  $J$ - $K$ , and  $R$ - $S$ - $T$  flip-flops, this is easily done by introducing the count signal into an *and* gate with each input equation. This is sufficient since all these flip-flops retain their states if all of their inputs have a value of zero. The delay element does not have this implicit holding property, hence, holding of a given state requires external feedback of the type shown in Fig. 1.

**Count Preset.** Presetting the counter to a particular count will be required if it must be slaved to other events in the computer. If, however, it is a master counter, it may serve to bring all other devices into step with it, without requiring a preset. The external signal which causes the preset is incorporated in the equations in such a way as to force each storage element into the desired state.

**Unused States.** A final consideration is the possibility of the counter going into a state that is normally not used. This might happen, for example, when power is applied to the computer, or in the event of a transient failure. If the preset condition is not adequate to take care of this situation, then a “clearing” signal can be provided. This produces the same effect as a preset, but it may be a part of a general clearing signal provided to all parts of the computer.

For a master timing counter which can begin its counting cycle at any time, it is sufficient to determine that the counter will not lock into an unused state. Figure 26 is a study of the unused states of the ten-state

counter using delay elements. The unused states are shown in rectangles. The arrows indicate the transitions made in advancing the counter by a count of one. This shows that the six unused states all eventually move into one of the ten desired states. The result of this analysis depends on

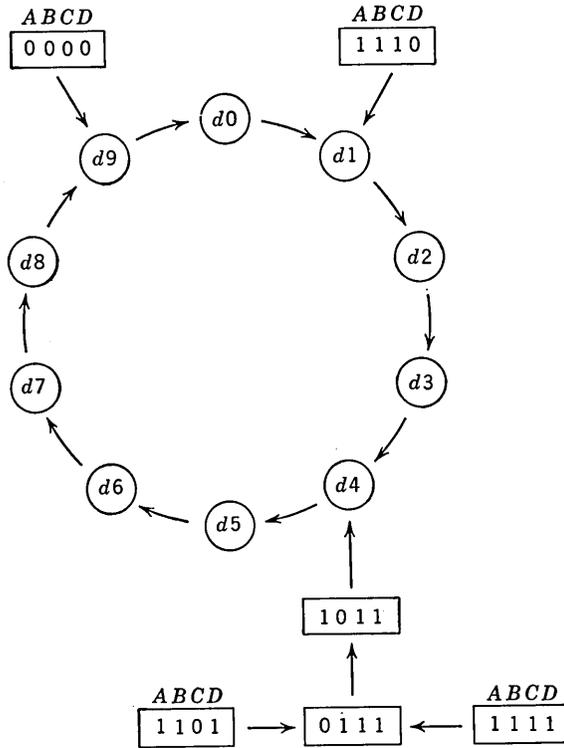


FIG. 26. Analysis of unused counter states.

the storage element since the unused states were employed as redundancies for simplification. For example, the above variations of this counter that employ storage elements other than delays would lock into the state  $ABCD = 0000$ . Additional gating logic would be required to release the counter from a transient excursion into this configuration.

### 5. DIRECT SIMULATION OF A LOGICAL DESIGN

Little use has been made of digital computers as tools in the design of other digital systems. This section discusses an approach to direct simulation of the logical behavior of a system from its definitive logical design. See Ref. 17.

**Objectives.** The prime objective of logical simulation is the detection of mistakes in the logical design of the simulated computer. There are several aspects of this problem.

1. These mistakes are costly in checkout time of first-of-a-kind computers because detection of the error must be followed by a correction process which inevitably requires some rewiring of the computer, and very likely requires more computer elements. In addition, the detection of wiring mistakes and circuit malfunctions is complicated when errors in logical design are present or suspected.

2. Logical simulation facilitates changes. Just as the first designs have mistakes, frequently changes which are proposed have mistakes in them. Thus a simulation program written for a first design is available to check changes in subsequent models.

3. In applications of digital systems to control problems, the amount of digital output may be small. This lack of voluminous digital outputs may indefinitely conceal subtle mistakes in logical design. The resulting infrequent computational errors might be attributed to transient conditions rather than to its true cause.

### Initial Considerations

The success or failure of a simulation program depends heavily on the initial approach. It should be borne in mind that in all but the simplest cases a limited objective must be sought.

**The Simulator.** It is assumed that the design group has a general purpose digital computer available to them. This computer (the simulator) need not be larger than the computer undergoing simulation in the sense of word length, storage capacity, or speed.

A special purpose computer could also be designed to simulate the logical design of a digital system (see Ref. 18).

**Initial States.** The behavior of a digital system is completely described by (1) the initial states of all storage elements, (2) the states of all inputs—initially and thereafter, (3) the logical design of the system.

The initial state of all storage elements can be set by storing 1's or 0's in their proper places within the simulator. These initial states may not be known, however. In this case it is good practice to emulate computer behavior from "power on" to "running." For example, all simulated flip-flops can be set to a random state to typify their condition after power has been applied. Thereafter, the steps which the operator would go through to get the machine "cleared" could be taken.

**Simulated Inputs.** It is desirable to minimize the number of inputs which must be simulated. Also, it is desirable for these inputs to be closely related to the programmed variables of the machines—instructions

and numbers. Usually, both of these ends will be served if the simulated portion (if not all) of the system is chosen as one of the subsystems or a combination of the subsystems. For example, the arithmetic and control portion of the computer can usually be separated from the rest of the system. The inputs which would require simulation then would typically be numbers and instructions from the storage unit, control signals and keyboard information from the console, etc. Note that this separation would not, for example, test a program address counter in the arithmetic and control unit directly by actually obtaining a number or instruction from the storage unit.

**Test Parameters.** The parameters of the test are the sequences of values which are applied to each of the inputs. It becomes apparent that there are far too many sequences of input combinations to test the logical design for all possibilities.

Consider those inputs which correspond to instructions and numbers. It is not reasonable to test each arithmetic instruction for all combinations of numbers. Instead, one should make an intelligent choice of the combinations which represent differences in the logical design. For example, the choice of arithmetic instruction-number combinations would be influenced by (1) combinations of signs of the operands, (2) operand combinations to yield all possible rounding effects, (3) operands resulting in overflow, (4) special cases such as operands or results which are zero, (5) various choices of operands for adder input combinations.

For a parallel arithmetic device, it ordinarily would not be necessary to test the last of the above conditions for every adder position. Symmetry usually exists which can easily establish the equivalence of operation of many such adders; thus, checking one adder checks the rest.

### Programming the Simulation

The simulation program performs the required functions of setting initial values of simulated flip-flops (or other storage elements), and thereafter updating these flip-flops. In each succeeding simulated time interval, the logical design defines the influence that the current flip-flop and input states shall exert upon the next state of a simulated flip-flop. Part of the working storage of this program is therefore the present value of each simulated storage element.

**Compiled Logic.** The simulation program requires a logical description of the simulated digital system. It would be possible to use the symbolic logical design directly. However, the use of this design in updating simulated storage elements is the heart of the program and it is important that this part be as efficient of time as possible.

The transcription of the symbolic logical design into an effective simulator process is called the *compiled logic*. The nature of the compiled

logic will depend on the computer to be used for simulation. A typical procedure would evaluate an equation for the  $S$  (set) input of an  $R$ - $S$  flip-flop which consists of *and*'s, of *or*'s (normal form), of other flip-flop outputs.

**EXAMPLE.** Consider the logical equation:

$$A12S = A11 \cdot AX1 + A13 \cdot \overline{AX1} \cdot AX2.$$

The symbolic design might not contain the symbols  $=$ ,  $\cdot$ ,  $+$  explicitly. For example, in punched card representation the leftmost columns might always be reserved for the input designation,  $A12S$ .

The compiled logic for this equation would be a short sequence of instructions which would evaluate  $A12S$  as a 0 or 1 depending on the present values of  $A11$ ,  $A13$ ,  $AX1$ , and  $AX2$ . Thus, if  $A11$  and  $AX1$  are both 1,  $A12S$  would have the value 1. The compiled logic contains instructions with addresses corresponding to the location in the simulator storage of these simulated flip-flop values. The instructions would be chosen in such a manner that the indicated logical operation of *and*, *or*, or *not* would take place. This is easily achieved with a general purpose simulator having logical instructions. Ordinary arithmetic instructions can also be used. For example, multiplication of single 1's or all 0's have the same property as the logical *and*.

After the evaluation of all simulated storage element inputs, the next state of each element can be determined. At this point, a condition such as simultaneous  $R$  and  $S$  inputs could cause an alarm indication.

**Driver Program.** The program which causes sequential simulation is called the *driver program*. The *compiled logic* is used in a subroutine to evaluate the states of all storage elements for each time interval. Another subroutine places new values on the inputs each simulated time interval from a list of instructions and numbers, and switch and keyboard settings. The driver program also provides a subroutine to calculate independently the expected result produced by the simulated computer. These calculations are made only when results are supposed to be available from the simulated computer. Rounding, negative number representation, and word length of the computer undergoing simulation must be considered.

**Simulation Output.** The driver program provides a comparison between the simulated computer outputs and the expected outputs. If these are not identical, the simulator indicates that a malfunction has occurred. In this event, it is desirable also to provide sufficient information for the logical designer to pinpoint the source of the error. Such information would be available, for example, from a sequential recording of simulated storage element states and inputs. These would be provided for all simulated time intervals in the execution of the instruction in error.

REFERENCES

1. M. Phister, Jr., *Logical Design of Digital Computers*, Wiley, New York, 1958.
2. R. K. Richards, *Arithmetic Operations in Digital Computers*, Van Nostrand, Princeton, N. J., 1955.
3. Engineering Research Associates, *High-Speed Computing Devices*, McGraw-Hill, New York, 1950.
4. Staff of the Computation Laboratory, *Synthesis of Electronic Computing and Control Circuits*, Annals of the Computation Lab. of Harvard Univ., Harvard University Press, Cambridge, Mass., Vol. 27, Chaps. 3, 4, and 5, 1951.
5. S. Greenwald, R. C. Harter, and S. N. Alexander, SEAC, *Proc. I.R.E.*, **41**, 1300-1313 (1953).
6. N. M. Martin, On completeness of decision element sets, *J. Computing Systems*, **1** [3], 150-154 (1953).
7. E. W. Veitch, A chart method for simplifying truth functions, *Proc. Assoc. Computing Machinery*, 127-132, May 2-3, 1952.
8. M. Karnaugh, The map method for synthesis of combinational logic circuits, *Trans. Am. Inst. Elec. Engrs.*, Pt. I, **72**, 539-598 (1953).
9. W. V. Quine, The problem of simplifying truth functions, *Am. Math. Monthly*, **59**, 521-531 (1952).
10. W. V. Quine, A way to simplify truth functions, *Am. Math. Monthly*, **62**, 627-631 (1955).
11. M. J. Ghazala, Irredundant disjunctive and conjunctive forms of a Boolean function, *IBM J. Research and Dev.*, **1** [2], 171-176 (1957).
12. E. J. McCluskey, Jr., Minimization of Boolean functions, *Bell System Tech. J.*, **35** [6], 1417-1444 (1956).
13. D. A. Huffman, The synthesis of sequential circuits, *J. Franklin Inst.*, Pt. I, **257** [3], 161-190; Pt. II, **257** [4], 275-304 (1954).
14. G. H. Mealy, A method for synthesizing sequential circuits, *Bell System Tech. J.*, **34** [5], 1045-1079 (1955).
15. J. H. Felker, Typical block diagrams for a transistor digital computer, *Communications and Electronics*, No. 1, 175-182, July 1952.
16. E. C. Nelson, An algebraic theory for use in digital computer design, *Trans. I.R.E.*, PGEC, EC-3 [3], 12-21 (1954).
17. S. R. Cray and R. N. Kisch, A progress report on computer applications in computer design, *Proc. Western Joint Computer Conf.*, 82-85, Feb. 7-9, 1956.
18. W. E. Smith, A digital system simulator, *Proc. Western Joint Computer Conf.*, 31-36, Feb. 26-28, 1957.

## Arithmetic and Control Elements

*H. L. Engel*

1. System Considerations	18-01
2. Notation	18-02
3. Binary Operations	18-03
4. Decimal Operations	18-25
5. Special Operations	18-30
6. Control Elements	18-33
References	18-40

### 1. SYSTEM CONSIDERATIONS

The arithmetic element of a digital computer should be designed together with the computer system of which it is a portion. Some of the important system considerations that influence the design of an arithmetic element are listed in the following table:

<i>Feature</i>	<i>Consideration</i>
Input and output	Decimal or binary numbers, analog, combinations.
Speed	The importance of speed. Speed can be increased by adding equipment or by increasing the basic computer rate. A parallel or partially parallel arithmetic element will be faster than a serial element using the same clock rate, but it will require more hardware.
Circuitry	Transistors, cores, tubes, etc.
Flip-flops	Static or dynamic.
Gates	Levels of gating that can be used; tube, transistor, diode, core, resistor, etc.
Arithmetic operations	Only addition, subtraction and multiplication; possibly also division and square root.
Numbers	Fixed or floating point, binary or decimal, location of sign bit (most significant or least significant end).

All these factors greatly influence the design of the arithmetic element. In the work to follow, the techniques of logical design described in Chap. 17 are used to design portions of arithmetic elements. An attempt has been made to include a large variety of possible implementations in order to demonstrate that logical design techniques can be adapted to the hardware available.

The design of an arithmetic element is partly art and partly science. The design of a control element involves more art and less science. Thus design of a control element is best illustrated by exhibiting the techniques used in design. This is done in this chapter by designing the control element of a simple computer, showing many of the devices available to the logical designer and the many choices he must make.

## 2. NOTATION

**Logical Operations.** A plus (+) indicates logical sum, *or*, and a dot (·) logical product, *and*. Dots may be omitted in logical products. Where these symbols are used for arithmetic sums and products instead, this is indicated. A bar ( $\bar{\phantom{x}}$ ) indicates a logical complement; a prime (') is frequently substituted in typing. The summation sign ( $\Sigma$ ) indicates an arithmetic sum.

**Flip-Flop.** The name of a flip-flop is an upper case letter or an upper case letter and a number, for example,  $F$ ,  $X$ ,  $F1$ ,  $F3$ ,  $D14$ . The normal output of the flip-flop has the same name as the flip-flop and provides a signal only when the flip-flop is in the 1's state. The complementary output, if there is one, carries the name of the flip-flop with a bar over it ( $\bar{\phantom{x}}$ ). It provides a signal only when the flip-flop is in the 0's state. Conventionally, each output is assigned the value 1 when it provides a signal, and 0 when it does not.

**Flip-Flop Logic.** Three of the many logically different flip-flops are used in this chapter. They are shown with their truth tables in Fig. 1.

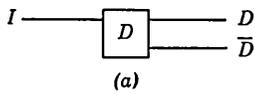
1. The first is the *delay flip-flop* with the name  $D$ , outputs  $D$  and  $\bar{D}$ , and the single input  $I$ . In terms of the input at digit time  $n$ , the output at digit time  $n + 1$  is

$$(1) \quad D_{+1} = I.$$

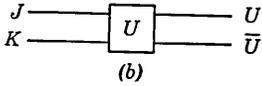
2. The second flip-flop is the *trigger flip-flop*, named  $U$ , with outputs  $U$  and  $\bar{U}$ , and inputs  $J$  and  $K$ . The state of this flip-flop at digit time  $n + 1$  in terms of the state and inputs at digit time  $n$  is

$$(2) \quad U_{+1} = J\bar{U} + \bar{K}U.$$

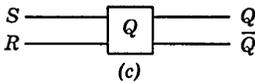
3. The third flip-flop is the *set-reset flip-flop*, named  $Q$ , having outputs  $Q$  and  $\bar{Q}$ , and inputs  $S$  and  $R$ . The state of the flip-flop at digit time



$I$	$D_{+1}$	$\bar{D}_{+1}$
0	0	1
1	1	0



$J$	$K$	$U_{+1}$	$\bar{U}_{+1}$
0	0	$U$	$\bar{U}$
0	1	0	1
1	0	1	0
1	1	$\bar{U}$	$U$



$S$	$R$	$Q_{+1}$	$\bar{Q}_{+1}$
0	0	$Q$	$\bar{Q}$
0	1	0	1
1	0	1	0
1	1	Unknown	Unknown

FIG. 1. Symbols and truth tables. (a) Delay flip-flop, (b) trigger flip-flop, (c) set-reset flip-flop.

$n + 1$  is given in terms of the state and inputs at digit time  $n$  by

$$(3) \quad Q_{+1} + RS = S\bar{Q} + \bar{R}Q + RS.$$

Here  $RS$  is written on both sides of the equation so that it is not possible to find  $Q_{+1}$  if both  $R$  and  $S$  are 1. This describes the behavior of the flip-flop; the next state cannot be predicted if there are two simultaneous inputs. Set-reset flip-flops are used in Sect. 6. Different flip-flop names are employed there to indicate the flip-flop usages.

*Note.* The name  $F$  is applied to any flip-flop for which the logic has not yet been selected.

**Combined Elements.** Boolean polynomials may be assigned names consisting of single upper-case letters.

The outputs of a half adder are named  $S_{\frac{1}{2}}$  and  $C_{\frac{1}{2}}$  (for sum and carry). See Sect. 3. The outputs of a half subtractor bear the same names.

The outputs of a full adder are named  $S$  and  $C_1$  for sum and carry. See Sect. 3. The outputs of a full subtractor are similarly named.

### 3. BINARY OPERATIONS

#### Count

The number  $N$  in an  $M$ -stage binary counter is given by

$$N = \sum_{m=1}^M 2^{m-1} Fm$$

where  $Fm$  is the name applied to the  $m$ th stage and has the value 1 when that flip-flop is in the 1's state, and 0 when that flip-flop is in the 0's state.

(a)

$N$	$D1$	$D2$	$D3$	$D4$
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	1	1	0	0
4	0	0	1	0
5	1	0	1	0
6	0	1	1	0
7	1	1	1	0
8	0	0	0	1
9	1	0	0	1
10	0	1	0	1
11	1	1	0	1
12	0	0	1	1
13	1	0	1	1
14	0	1	1	1
15	1	1	1	1

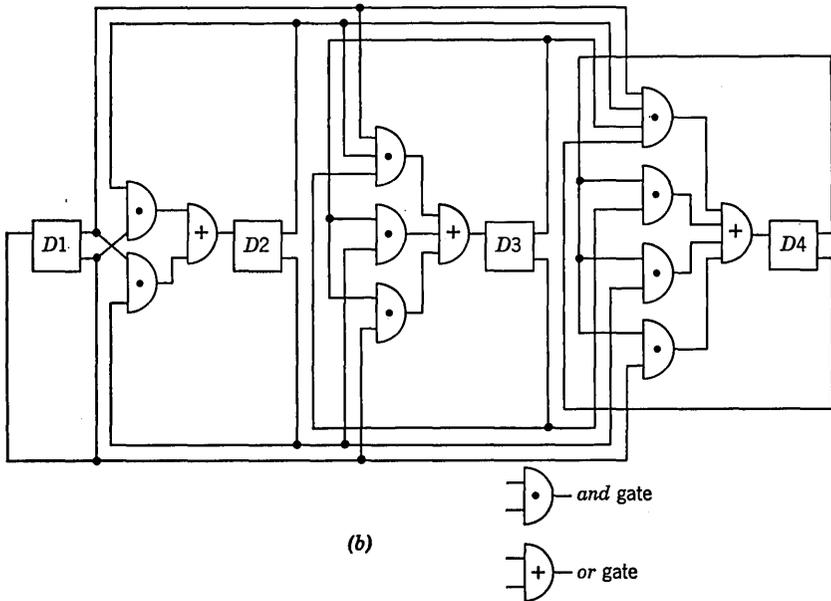


FIG. 2. Scale of 16 counter. (a) Truth table, (b) block diagram.

An  $M$  stage counter that takes all succeeding values of  $N$  from 0 to  $2^M - 1$  and then repeats is called a *scale of  $2^M$  binary counter*. As a truth table shows, the state of  $F_m$  at digit time  $n + 1$  in terms of the flip-flop states at digit time  $n$  is

$$F_{m+1} = [F_1 \cdot F_2 \cdots F(m-1)] \overline{F_m} + \overline{[F_1 \cdot F_2 \cdots F(m-1)]} F_m.$$

If delay flip-flops are substituted for these unspecified flip-flops, the input to the  $m$ th flip-flop is

$$I_m = [D_1 \cdot D_2 \cdots D(m-1)] \overline{D_m} + \overline{[D_1 \cdot D_2 \cdots D(m-1)]} D_m$$

as may be easily seen since  $D_{m+1} = I_m$ .

EXAMPLE. For a scale of 16 counter consisting of  $D_1, D_2, D_3, D_4$ , the inputs are

$$I_1 = \overline{D_1}$$

$$I_2 = D_1 \overline{D_2} + \overline{D_1} D_2$$

$$\begin{aligned} I_3 &= [D_1 \cdot D_2] \overline{D_3} + \overline{[D_1 \cdot D_2]} D_3 \\ &= D_1 D_2 \overline{D_3} + \overline{D_1} D_3 + \overline{D_2} D_3 \end{aligned}$$

$$\begin{aligned} I_4 &= [D_1 \cdot D_2 \cdot D_3] \overline{D_4} + \overline{[D_1 \cdot D_2 \cdot D_3]} D_4 \\ &= D_1 D_2 D_3 \overline{D_4} + \overline{D_1} D_4 + \overline{D_2} D_4 + \overline{D_3} D_4 \end{aligned}$$

These results can be verified by the truth table in Fig. 2. A block diagram of the logic appears in the same figure.

### Shift

The state of the  $m$ th stage,  $F_m$ , of a shifting register at digit time  $n+1$  in terms of the stage  $F(m-1)$  at digit time  $n$  is given by

$$F_{m+1} = F(m-1).$$

If the shifting register consists of set-reset flip-flops  $Q_m$ , then

$$S_m = Q(m-1)$$

$$R_m = \overline{Q(m-1)}.$$

It follows from eq. (3) that

$$\begin{aligned} Q_{m+1} + [\overline{Q(m-1)}] \cdot [Q(m-1)] &= Q(m-1) \overline{Q_m} + \overline{[\overline{Q(m-1)}]} Q_m \\ &\quad + [\overline{Q(m-1)}] \cdot [Q(m-1)] \end{aligned}$$

or

$$\begin{aligned} Q_{m+1} &= Q(m-1)[\overline{Q_m} + Q_m] \\ &= Q(m-1). \end{aligned}$$

This shows that the set-reset flip-flops with the indicated inputs act as a shifting register. See Fig. 3a.

The shifting register can be constructed with a single magnetic core per stage. In this case the core circuit has the same logical description as the delay flip-flop described in Sect. 2, but it lacks the complementary output. A typical logic equation is (see Fig. 3b, c),

$$I_m = D(m-1).$$

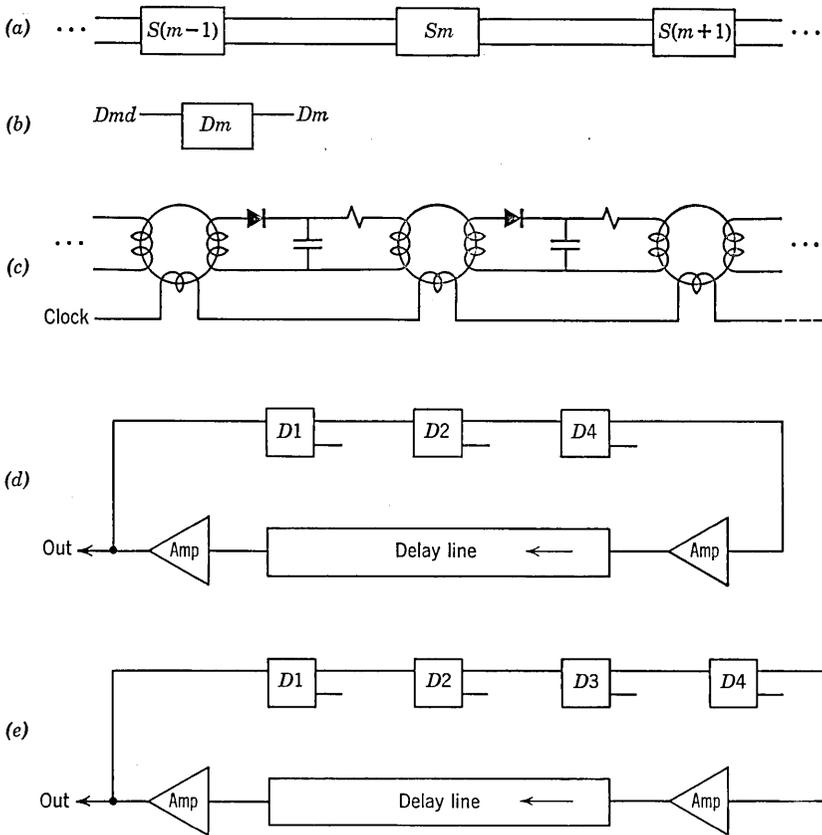


FIG. 3. Shift registers. (a) Set-reset flip-flops, (b) magnetic core, block diagram, (c) magnetic core, single core per stage, (d) and (e) variable length shift registers using delay lines.

In serial computers a multibit delay line closed on itself through flip-flops may be used as a register with the bits available sequentially. It may be necessary to left shift or right shift the contents of this register with respect to the contents of other similar registers. A left or right shift of one bit per circulation time may be accomplished by increasing or decreasing the register length one bit. Say the register consists of a delay line and three flip-flops, as in Fig. 3d. An additional flip-flop introduced, as in Fig. 3e, will delay the contents of the delay line one bit each circulation time. Conversely, removing a flip-flop, as in Fig. 3f, will advance the contents of the delay line one bit each circulation time.

The delay line length may be varied under the influence of two control flip-flops, say  $D10$  and  $D11$ . If both are in the 0 state, the register is to

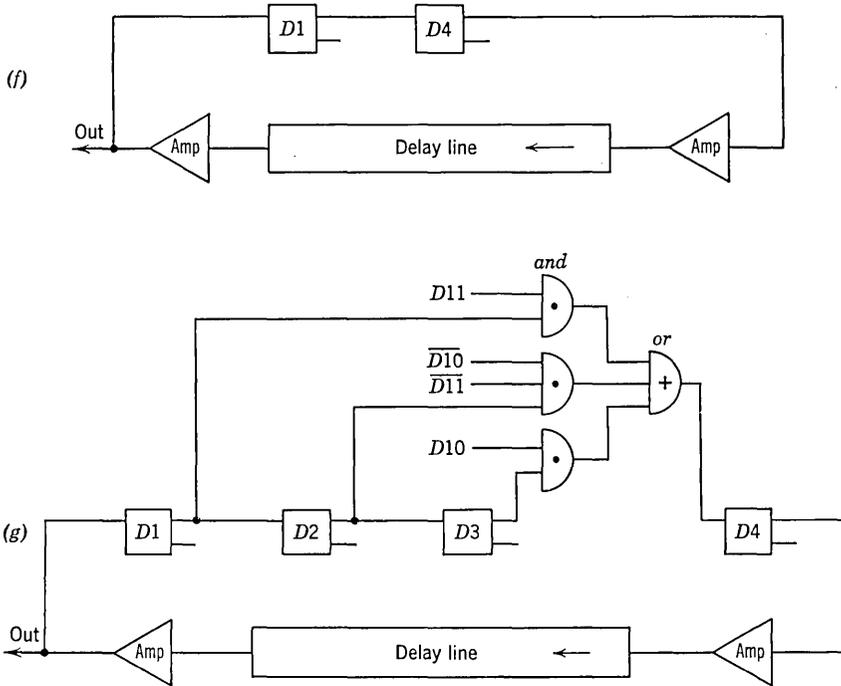


FIG. 3 (continued). Shift Registers. (f) and (g) variable length shift registers using delay lines.

circulate normally. If  $D10$  is 1 the register is to be lengthened one bit. If  $D11$  is 1 the register is to be shortened one bit.  $D10$  and  $D11$  are not allowed to be 1 simultaneously. For the variable length delay line, then

$$I_2 = D1$$

$$I_3 = D2$$

$$I_4 = D10 D3 + \overline{D10} \overline{D11} D2 + D11 D1.$$

### Readin

A new word may be read into a register either serially or in parallel.

In *serial* readin the register may be a static register so that each new bit must be addressed to a flip-flop by external control circuitry, or the register may be a shift register so that all the bits are read successively into the same flip-flop of the register and are distributed by the normal shift operation.

In *parallel* readin, all the bits of the new word are simultaneously placed

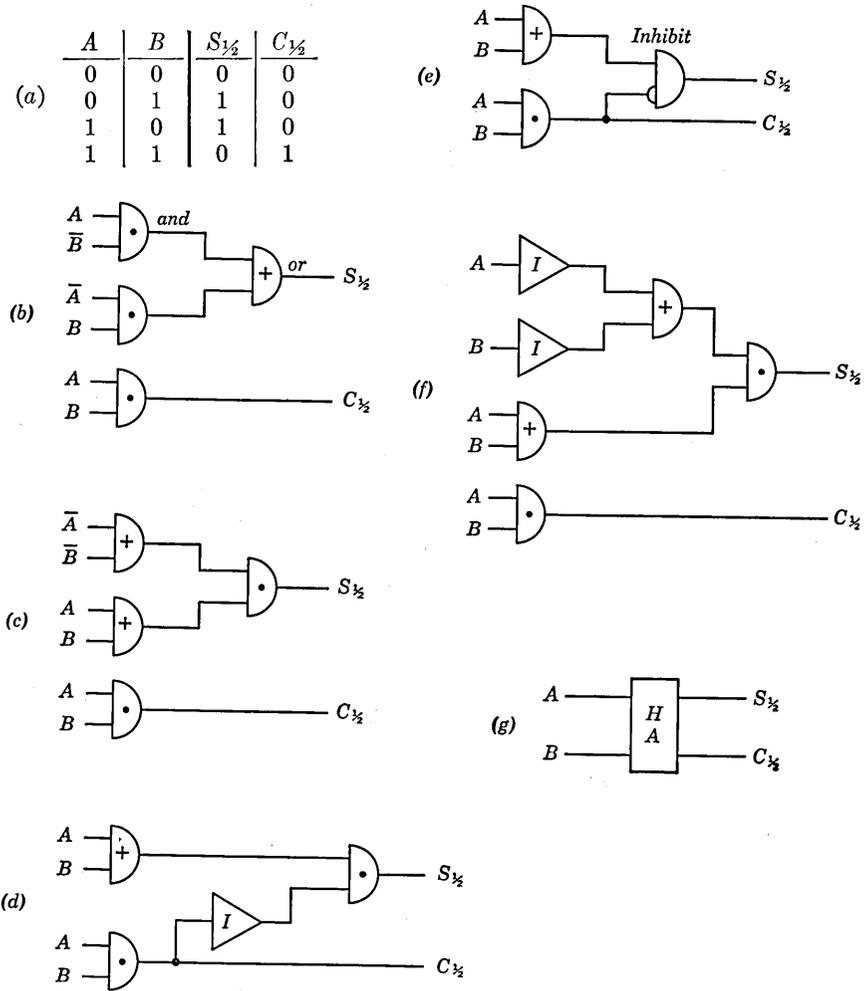


FIG. 4. Half adder. (a) Truth table, (b), (c), (d), (e), (f) block diagrams, (g) symbols.

in the correct flip-flops. With set-reset flip-flops in this application it is sometimes economical first to set all flip-flops in the register to 0, and then supply 1's inputs only to the flip-flops that are to be set to 1.

**Add**

**Half Adder.** A half adder is a device that produces two outputs, one of which is the sum modulo 2 of its two input signals, and the other is a "carry" that is 1 only if both input signals are 1's. Figure 4a is a truth table for the half adder.

Let  $A$  and  $B$  be the half adder inputs. The outputs, named  $S_{\frac{1}{2}}$  and  $C_{\frac{1}{2}}$  (for sum and carry) are

$$\begin{aligned} S_{\frac{1}{2}} &= A\bar{B} + \bar{A}B, \\ C_{\frac{1}{2}} &= AB. \end{aligned}$$

To illustrate some of the many embodiments of these equations, first assume that the complements  $\bar{A}$  and  $\bar{B}$  are available. The above equations may then be translated directly into hardware, as in Fig. 4b.

The sum equation may also be written as

$$S_{\frac{1}{2}} = (A + B)(\bar{A} + \bar{B})$$

and mechanized as in Fig. 4c.

If  $\bar{A}$  and  $\bar{B}$  are not available, it may be better to write the sum equation in another form that will save equipment. For example,

$$\begin{aligned} S_{\frac{1}{2}} &= (A + B)(\overline{AB}), \\ &= (A + B)\bar{C}_{\frac{1}{2}}. \end{aligned}$$

This statement requires only one inverter as in Fig. 4d, or one inhibiting gate as in Fig. 4e. If circuit design does not permit multiple use of the output of an inverter or gate, then one way of mechanizing the half adder with only normal outputs is shown in Fig. 4f.

A half adder may be indicated schematically as in Fig. 4g.

**Full Adder.** A full adder has three inputs, two of which are an augend bit  $A$ , and an addend bit  $B$ . The third input is the carry  $C$  from the previous addition. The modulo 2 sum of the three adder inputs is called the sum  $S$ . There is a carry output  $C_1$  if at least two of the inputs are 1's.

Of the many ways to implement a full adder, hereafter called an *adder*, only three are illustrated here. The adder truth table is Fig. 5a. The logic equations are

$$\begin{aligned} S &= \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC, \\ C_1 &= AB\bar{C} + A\bar{B}C + \bar{A}BC + ABC \\ &= AB + BC + CA. \end{aligned}$$

*Adder 1.* The sum equation may also be written

$$\begin{aligned} S &= (A\bar{B} + \bar{A}B)\bar{C} + (\bar{A}\bar{B} + AB)C \\ &= (A\bar{B} + \bar{A}B)\bar{C} + (\overline{A\bar{B} + \bar{A}B})C, \end{aligned}$$

and this shows that a full adder may be made with two half adders, as in Fig. 5b. In this figure, the box containing a  $C$  represents a delay flip-flop, such as might be used in a serial adder to store the carry for one digit time.

*Adder 2.* An adder can also be produced by direct embodiment of the first equations for  $S$  and  $C_1$ , as in Fig. 5c.

A	B	C	S	C <sub>+1</sub>
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(a)

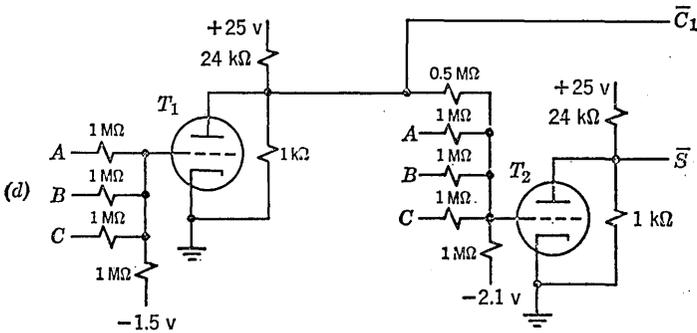
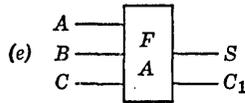
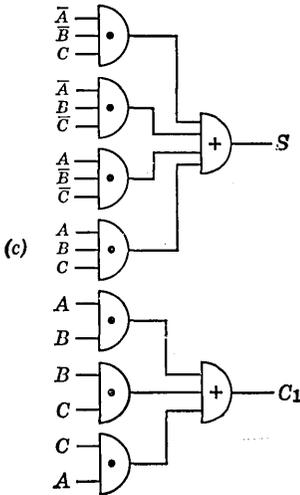
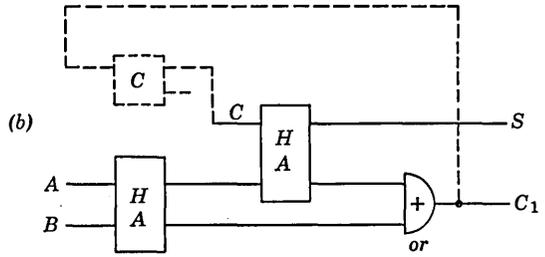


FIG. 5. Adder. (a) Truth table, (b), (c) block diagrams, (d) Kirchhoff adder, (e) symbols.

*Adder 3.* Still another form of adder, shown in Fig. 5*d*, is called a Kirchhoff adder, for it employs Kirchhoff's law. For this adder the logic equations are

$$S = (A + B + C)\bar{C}_1 + ABCC_1,$$

$$C_1 = AB + BC + CA.$$

$A, B, C$  are voltage sources of 1 volt if they represent 1's, and 0 volts if they are 0's.  $T1$  and  $T2$  are ideal triodes such that grids negative with respect to the cathodes result in zero plate current, and grids positive with respect to the cathodes reduce the cathode-plate impedances to 0. Then, for the circuit shown,  $T1$  conducts if and only if the logical sum  $AB + BC + CA$  is 1; i.e., the current in  $T1$  represents  $C_1$ , and the plate voltage of  $T1$  represents  $\bar{C}_1$ .  $\bar{C}_1, A, B$  and  $C$  can then be used in another current adder to produce  $S$  at the grid of  $T2$  and  $\bar{S}$  at its plate.

A full adder (*FA*) may be indicated schematically as in Fig. 5*e*.

**Parallel Adder.** In the parallel adder all the addend and augend bits are available simultaneously. The parallel adder for two  $n$ -bit numbers is not necessarily  $n$  times faster than a serial adder, for the addition time of a parallel adder is determined by the time needed to propagate a carry from the least significant bit to the most significant bit, or the time required to set up all carries before actual addition.

In parallel addition the sum may be placed in a sum register separate from the addend and augend registers, or the sum may replace the augend. In the latter case the register holding the sum is called an *accumulator*.

**EXAMPLE 1.** Use of a sum register and carry propagation are illustrated in Fig. 6*a*. A typical stage is shown, where  $A$  and  $B$  are flip-flops in the addend and augend registers.  $S$  is the sum digit resulting at this stage from the addition of the augend bit, the addend bit, and the carry from the previous stage. Separate lines  $M$  and  $N$  are used to indicate carry ( $C$ ) or no carry ( $\bar{C}$ ) between stages. Two lines are needed so that whether or not there is a carry, there is a signal to the next stage. Addition is initiated by a pulse in the no carry line of the least significant stage. Since there must be a carry or no carry from this stage, line  $M$  or  $N$  to the next stage will be energized. The process progresses from stage to stage until completion is indicated by a signal on the  $M$  or  $N$  line from the most significant stage. With  $M_1$  and  $N_1$  for the lines out of a stage, the logic equations for that stage are

$$\begin{aligned} S &= \bar{A}\bar{B}M + \bar{A}BN + A\bar{B}N + ABM \\ &= \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC, \end{aligned}$$

$$M_1 = ABN + BM + AM (= AB\bar{C} + BC + AC),$$

$$N_1 = \bar{A}\bar{B}M + \bar{B}N + \bar{A}N (= \bar{A}\bar{B}C + \bar{B}\bar{C} + \bar{A}\bar{C}).$$

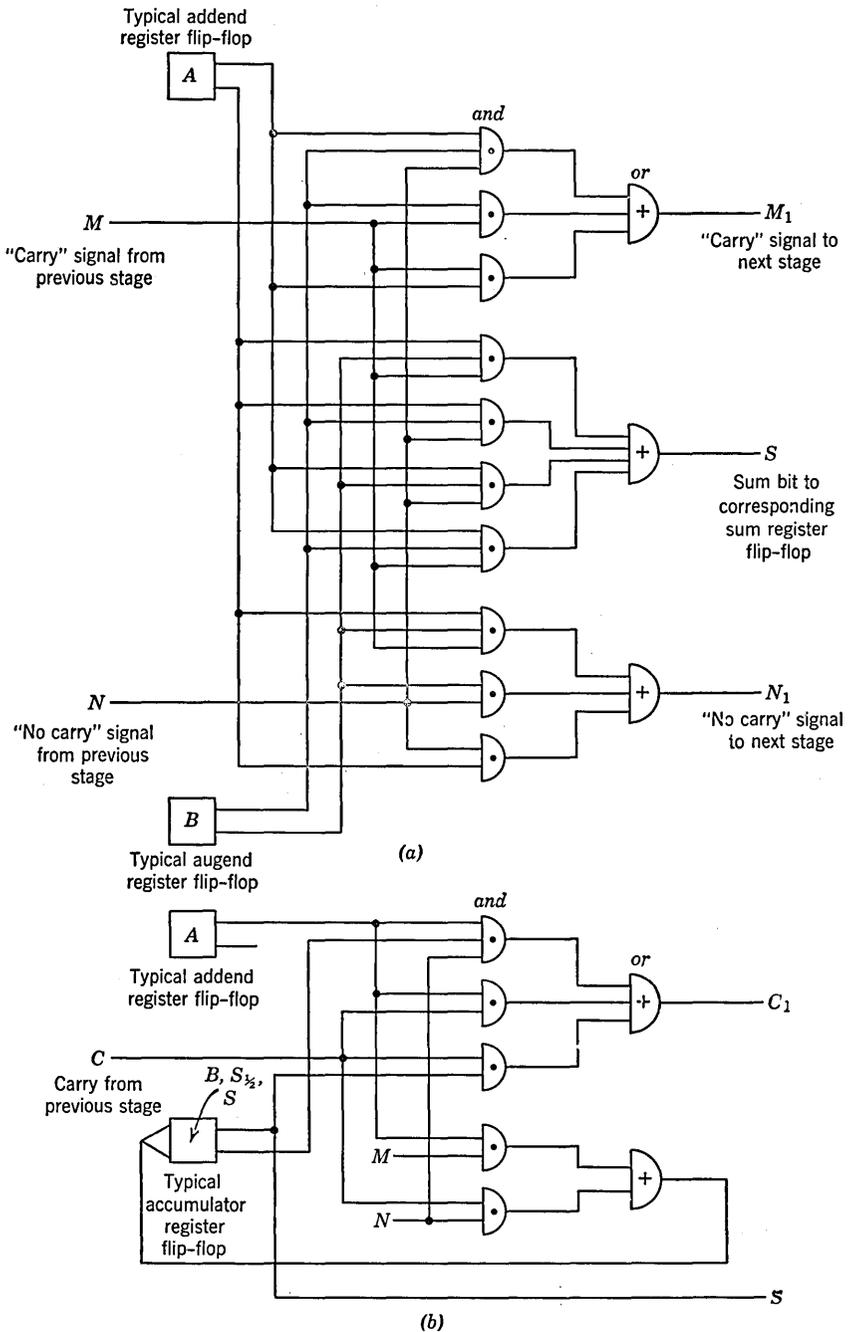


FIG. 6. Parallel addition. (a) Using sum register and carry propagation, (b) using accumulator and half sums.

EXAMPLE 2. An accumulator and a different carry technique are illustrated in the next example. See Fig. 6b. Here the addend and augend bits are combined in one digit time to produce half sums in the accumulator, and in the next digit time the half sums are altered in accordance with the carries from each stage.

For any stage

$$\begin{aligned} S_{1/2} &= A\bar{B} + \bar{A}B, \\ S &= C\bar{S}_{1/2} + \bar{C}S_{1/2}, \\ C_1 &= AB + BC + CA. \end{aligned}$$

Since  $B$  is not available at the time the carries must be formed, having been replaced by  $S_{1/2}$ , it is necessary to find an expression for  $C_1$  not involving  $B$ . By multiplying the equation for  $S_{1/2}$  by  $\bar{A}$ , it is seen that

$$\bar{A}S_{1/2} = \bar{A}\bar{B}.$$

From the same equation

$$\bar{S}_{1/2} = \bar{A}\bar{B} + AB$$

and

$$A\bar{S}_{1/2} = AB.$$

Further  $C_1$  may be written

$$\begin{aligned} C_1 &= AB + (\bar{A}\bar{B})C + CA \\ &= A\bar{S}_{1/2} + \bar{A}S_{1/2}C + CA \\ &= A\bar{S}_{1/2} + S_{1/2}C + CA. \end{aligned}$$

With a trigger flip-flop to mechanize this, in the first digit time

$$\begin{aligned} U_{+1} = S_{1/2} &= J\bar{B} + \bar{K}B \\ &= A\bar{B} + \bar{A}B \end{aligned}$$

so

$$J = K = A.$$

In the second digit time

$$\begin{aligned} U_{+1} = S &= J\bar{S}_{1/2} + \bar{K}S_{1/2} \\ &= C\bar{S}_{1/2} + \bar{C}S_{1/2} \end{aligned}$$

so

$$J = K = C.$$

With  $M$  and  $N$  to denote first and second digit times, it follows that

$$J = K = MA + NC$$

and

$$C_1 = A\bar{U}N + UC + CA.$$

### Subtract

**Half Subtractor.** The half subtractor accepts a minuend bit and a subtrahend bit, and produces a half difference and a half borrow. Letting  $A$ ,  $B$ ,  $S_{\frac{1}{2}}$  and  $C_{\frac{1}{2}}$  represent these quantities in order, the logic equations of the half subtractor are

$$S_{\frac{1}{2}} = A\bar{B} + \bar{A}B,$$

$$C_{\frac{1}{2}} = \bar{A}B.$$

The corresponding truth table is Fig. 7. No examples of implementation are given because of similarity to the half adder.

$A$	$B$	$S_{\frac{1}{2}}$	$C_{\frac{1}{2}}$
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

FIG. 7. Half subtractor truth table.

**Serial Subtractor.** The three inputs of a serial subtractor are a minuend bit  $A$ , a subtrahend bit  $B$ , and a borrow bit  $C$ . The outputs are a difference bit  $S$ , and a borrow bit  $C_1$ . The logic equations are

$$S = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC,$$

$$C_1 = \bar{A}B + BC + C\bar{A}.$$

The truth table appears in Fig. 8a. The difference equation is identical with the sum equation of the adder. The borrow equation is like the adder carry equation except that  $\bar{A}$  replaces  $A$ . One embodiment of a subtractor is shown in Fig. 8b, and a conventional representation in Fig. 8c.

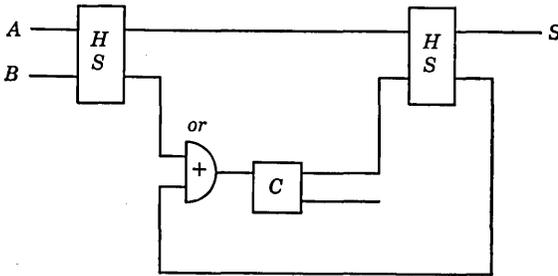
**Subtraction Using Complements.** The similarity of the logic equations for addition and subtraction suggests that subtraction can be accomplished by *adding* the 2's complement of the subtrahend to the minuend. This may be done with an adder by interchanging  $B$  and  $\bar{B}$  whenever they occur in the logic equations for sum and carry, provided the carry flip-flop is initially 1. This is illustrated in subtracting 0.0110000 from 0.1001001. Normally this would be

$$\begin{array}{r} 0.1001001 \quad A \\ -0.0110000 \quad -B \\ \hline 0.0011001 \quad S \end{array}$$

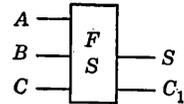
Replacing 0's by 1's and vice versa in  $B$ , and putting in an initial carry

<i>A</i>	<i>B</i>	<i>C</i>	<i>S</i>	<i>C</i> <sub>1</sub>
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

(a)



(b)



(c)

FIG. 8. Serial subtractor. (a) Truth table, (b) block diagram, (c) symbol.

it is found that

$$\begin{array}{r}
 0.1001001 \quad A \\
 +0.1001111 \quad \text{1's complement of } B \\
 + \quad \quad \quad 1 \quad \text{Initial carry setting} \\
 \hline
 1.0011001
 \end{array}$$

If the overflow to the left of the binary point is ignored, the result is *A* minus *B*.

A subtrahend larger than the minuend results in a negative difference. In this case the output of the subtractor is the 2's complement of the result. If the computer represents negative numbers as 2's complements, this is fine. If the machine stores numbers as magnitude and sign, the result,  $2^n - N$ , must be complemented. The magnitude may be found by subtracting the result from  $2^n$  since  $N = 2^n - (2^n - N)$ , or by subtracting the result from 0 and discarding the overflow since  $N \equiv 0 - (2^n - N) \pmod{2^n}$ , or by generating the 1's complement and adding 1 in the least significant place.

**Parallel Subtractor.** The techniques of logical design of a parallel subtractor are so like those for a parallel adder that no examples are given here.

**Zero Representation.** A machine that stores numbers as magnitude and sign may have two representations for zero, namely  $+0$  and  $-0$ . In such a machine if it is desired to always store a 0 as  $+0$ , it may be necessary to provide special circuits to detect a 0 result.

### Add-Subtract

Since the logical equations for addition and subtraction are so alike, it is frequently convenient to use the same equipment for both operations. Only one example is given here. Letting  $M$  be the signal to add,  $N$  the signal to subtract, and maintaining  $\overline{M}\overline{N} = 0$ ,

$$\begin{aligned} S &= M(\overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC) \\ &\quad + N(\overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC) \\ &= \overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC, \\ C_1 &= M(AB + BC + CA) + N(\overline{A}B + BC + C\overline{A}) \\ &= BC + MAB + MCA + N\overline{A}B + NA\overline{B}. \end{aligned}$$

### Multiply

**Binary Multiplication.** Binary multiplication is very easy since the multiplication table is just a two by two array, as shown in Fig. 9a. Figure 9b is an example of binary multiplication. In a digital computer it is usually more convenient to determine each partial product in succession, as in Fig. 9c.

If numbers in a computer are represented as magnitude and sign, multiplication is accomplished by multiplying the magnitudes and affixing a sign to the result determined from the signs of the multiplicand and multiplier.

If negative numbers are represented by 2's complements, the product of signed numbers could be found by a subroutine that would determine the signs and magnitudes of the operands, perform the multiplication as indicated above, and then, if the result were negative, cast the result into 2's complement form. It seems preferable, however, to devise a scheme for multiplying such numbers without a special subroutine. Two techniques are presented here.

**Shaw Method.** In the Shaw method,  $x$  and  $y$  are two signed numbers of magnitude less than 1.  $X$  and  $Y$  are their binary representations, being equal to the numbers when those are positive, and being their 2's complements when the numbers are negative. If a bit of  $Y$  is 1,  $X$  (less its portion to the left of the binary point) is added to the partial product. If a

bit of  $Y$  is 0, the bit of  $X$  to the left of the binary point is added to the partial product. This process ends just before the bit of  $Y$  to the left of

	Multiplicand		
	Digit		
	0	0	1
Multiplier	0	0	0
Digit	1	0	1

(a)

Multiplicand	0.11101	Multiplier
Multiplier	0.11110	Digit
	00000	0
	11101	1
	11101	1
	11101	1
	11101	1
Product	0.1101100110	

(b)

Multiplicand	0.11101	Partial Product	Multiplier Digit
Multiplier	0.11110		
	00000	✓	
	00000		0
	00000	✓	
	11101		1
	111010	✓	
	11101		1
	10101110	✓	
	11101		1
	110010110	✓	
	11101		1
Product	0.1101100110		

(c)

FIG. 9. Binary multiplication. (a) Multiplication table, (b) example of manual operation, (c) example of machine operation.

the binary point is examined. Next, if  $x$  is negative  $1 + 2^{-n}$  is added to the partial product. Finally, if  $y$  is negative, the 2's complement of  $X$  is added to the partial product.

EXAMPLES.		<i>Multiplication.</i>	
0.1001	$x = \frac{9}{16}$	1.0111	$x = -\frac{9}{16}$
0.1011	$y = \frac{11}{16}$	0.1011	$y = \frac{11}{16}$
<hr/> 01001		<hr/> 00111	
01001		00111	
00000		10000	
01001		00111	
<hr/> 001100011		<hr/> 010001101	
00000	sign $x$ correction	10001	sign $x$ correction
<hr/> 001100011		<hr/> 110011101	
00000	sign $y$ correction	00000	sign $y$ correction
<hr/> 0.01100011		<hr/> 1.10011101	
0.1001	$x = \frac{9}{16}$	1.0111	$x = -\frac{9}{16}$
1.0101	$y = -\frac{11}{16}$	1.0101	$y = -\frac{11}{16}$
<hr/> 01001		<hr/> 00111	
00000		10000	
01001		00111	
00000		10000	
<hr/> 000101101		<hr/> 011000011	
00000	sign $x$ correction	10001	sign $x$ correction
<hr/> 000101101		<hr/> 111010011	
10111	sign $y$ correction	01001	sign $y$ correction
<hr/> 1.10011101		<hr/> (1)0.01100011	

overflow ↗

**Booth Method.** In the Booth method,  $Y_m$  is the  $m$ th most significant bit of an  $n$  bit multiplier representation.  $Y_{n+1}$  is zero. Starting with  $m = n$ ,  $Y_m$  and  $Y_{m+1}$  are compared:

1. If  $Y_m = Y_{m+1}$  add 0.
2. If  $Y_m = 1, Y_{m+1} = 0$  add the 2's complement of  $X$  to the partial product.
3. If  $Y_m = 0, Y_{m+1} = 1$  add  $X$  to the partial product.

EXAMPLE.

1.0111	$x = -\frac{9}{16}$
0.1011(0)	$y = \frac{11}{16}$
<hr/> 000001001	$Y_4 = 1 \quad Y_5 = 0$
00000000	$Y_3 = 1 \quad Y_4 = 1$
11101111	$Y_2 = 0 \quad Y_3 = 1$
001001	$Y_1 = 1 \quad Y_2 = 0$
10111	$Y_0 = 0 \quad Y_1 = 1$
<hr/> (1)1.10011101	

overflow ↗

**Serial Multiplication.** Machine processes in multiplication are most easily explained in terms of positive numbers. In serial multiplication the digits of the multiplicand and multiplier are each assumed available in succession from shifting registers or delay lines. Another register or delay line, here called the accumulator, holds the partial products. See Fig. 10. In this example, the multiplicand and intermediate sums are assumed to be in delay lines and the multiplier in a shifting register. The

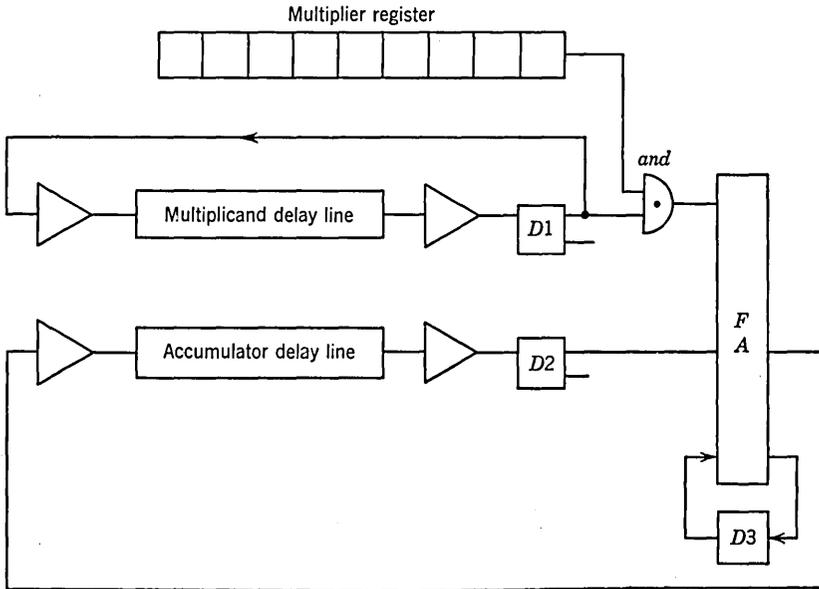


FIG. 10. Serial multiplication.

multiplicand delay line is one bit longer than the accumulator delay line. As a result, the multiplicand precesses, i.e., is shifted left, one digit with respect to the partial product each time the accumulator register circulates. This permits successive powers of two times the multiplicand to be added to the partial product in successive circulations. The multiplier register shifts right just once at the end of each circulation time to bring a new multiplier bit into position. According as the multiplier digit in the last flip-flop of the multiplier register is one or zero the multiplicand is added or not added to the partial product. *Serial-parallel* multiplication is multiplication using a parallel adder or parallel accumulator for adding.

**Parallel Multiplication.** Multiplication of two binary numbers can be achieved in a single digit time by *parallel* multiplication, in which each bit of the product is expressed directly in terms of all the bits of the multi-

plier and the multiplicand. It can be seen that the logical expressions for the product bits rapidly become complicated as the number of bits of the operands are increased. The amount of hardware necessary to generate the product bits makes parallel multiplication impractical for operands more than four digits long. To illustrate this point, the expression for a single bit of the product resulting from the multiplication of two three-bit numbers is indicated below. The multiplicand is  $a_2a_1a_0$  ( $= 2^2a_2 + 2^1a_1 + 2^0a_0$ ) and the multiplier is  $b_2b_1b_0$ . The product is  $c_5c_4c_3c_2c_1c_0$ . The logic equation for  $c_3$  is

$$c_3 = \bar{a}_0a_1\bar{a}_2b_2 + a_1\bar{a}_2\bar{b}_1b_2 + a_0a_1a_2\bar{b}_0b_2 + \bar{a}_0a_1\bar{b}_1b_2 + a_2\bar{b}_0b_1\bar{b}_2 + \bar{a}_1a_2b_1\bar{b}_2 \\ + \bar{a}_0a_2b_0b_1b_2 + \bar{a}_1a_2\bar{b}_0b_1 + a_0a_1\bar{a}_2b_0b_1\bar{b}_2 + a_0\bar{a}_1a_2b_0\bar{b}_1b_2.$$

**Precision.** In the explanations of multiplication given above, all the bits of the product have been retained, so that the product may have as many bits as the multiplicand and multiplier together. In designing a digital computer it is often desirable to allow no more digits in the product than in the largest possible multiplicand or multiplier. In this case, instead of shifting the multiplicand left one place for each successive partial product, the partial products are shifted right one place for each successive partial product. This results in the loss of the least significant digits of the product. To illustrate this point the example of Fig. 9c is repeated below.

Multiplicand	0.11101	
Multiplier	0.11110	
$0 \times$ multiplicand	00000	
Partial product <i>A</i>	00000	
Right shift	00000	
$1 \times$ multiplicand	11101	
Partial product <i>B</i>	11101	
Right shift	01110	
$1 \times$ multiplicand	11101	
Partial product <i>C</i>	101011	
Right shift	10101	
$1 \times$ multiplicand	11101	
Partial product <i>D</i>	110010	
Right shift	11001	
$1 \times$ multiplicand	11101	
Partial product <i>E</i>	110110	
Right shift	0.11011	Product

**Roundoff Correction.** Now the product obtained by this means is less than the actual product by 0.0000000110. In another example it might have been smaller than the full product by almost 0.00001. The

difference between the product obtained by this means and the full product is called the truncation error. The truncation error is always in the same direction. To eliminate this bias it is the usual practice to add a correction term to the product. The two most common roundoff corrections are (1) always replace the last digit retained in the product by 1, and (2) add a 1 to the digit place beyond the last to be retained, allow all carries to occur, and then drop all digits beyond the last to be retained. In the second roundoff procedure it may be more convenient to add the rounding 1 to the product register before the multiplicand can first be added in.

Both of these methods result in essentially unbiased products, but the second results in a smaller variance.

### Divide

In a binary digital computer division may be done in at least four ways. They are described as though divisor and dividend are both positive.

**Iteration.** If  $a/b$  is to be found,  $1/b$  can be determined by an iteration scheme involving only addition, subtraction and multiplication, and then  $a/b$  found by multiplying by  $1/b$ . One iteration formula for  $1/b$  is

$$u_{n+1} = u_n(2 - u_nb)$$

where  $u_n$  approaches  $1/b$  as  $n$  approaches  $\infty$ .

**Restoring Division.** In this the divisor is subtracted from the remainder; if the result is positive, a 1 is placed in the quotient; if the result is negative, a 0 is placed in the quotient and the remainder is restored to its previous value by adding the divisor to it.

**Trial Division.** Special circuitry is provided to determine if the divisor exceeds the remainder without actually performing a subtraction. If the divisor is greater, no subtraction is performed and a 0 is placed in the quotient. If the divisor is not greater than the remainder, the divisor is subtracted from the remainder and a 1 is placed in the quotient.

**Nonrestoring Division.** In this method the divisor is always subtracted from the remainder if the remainder is positive, or added to the remainder if it is negative. If the new remainder is positive, a 1 is placed in the quotient. If the new remainder is negative, a 0 is placed in the quotient.

Division may be performed serially or serial-parallel, depending on the equipment available.

The same roundoff procedures may be used for the quotient as for the product, although to use the second method one more quotient digit must be found.

**Fixed Point Division.** If the dividend exceeds the divisor in a fixed point machine the methods given above may result in very erroneous quotients. It is common in fixed point machines to provide special circuitry to detect this situation. It is also possible to mechanize the division

process so that the largest number the quotient register can hold will result if the true quotient is beyond the range of that register.

**Magnitude and Sign.** If numbers are stored in the computer as magnitude and sign, the magnitude of the quotient is determined from the magnitudes of the divisor and dividend, and the sign of the quotient determined from their signs.

**Complements.** If negative numbers appear in the computer as 2's complements, the quotient could be found through a subroutine involving determination of the magnitudes of the divisor and dividend. However, it is more convenient, for numbers of this kind, to have a division process that does not depend upon the signs of the dividend and divisor. One such scheme is described below.

**EXAMPLE.** Compare the sign of the remainder (the dividend is considered the zeroth remainder) with the sign of the divisor. Shift the remainder left one place; this results in loss of the sign digit of the remainder, but this does no harm. If the divisor and remainder had like signs, perform a subtraction. If the signs were unlike, perform an addition. Each time an addition is performed the quotient digit is made 1, each time a subtraction is done the quotient digit is made 0. The quotient determined by this scheme is not correct. To it a 1 in the sign position and a rounding 1 in the final position must be added. This process is illustrated below.

1.1001	$a(=r_0)$	$-\frac{7}{16}$			
0.1101	$b$	$\frac{13}{16}$			
1.0010	$2r_0$				
0.1101	$+y$				
1.1111	$r_1$				
1.1110	$2r_1$				
0.1101	$+y$				
0.1011	$r_2$				
1.0110	$2r_2$				
0.1101	$-y$				
0.1001	$r_3$				
1.0010	$2r_3$				
0.1101	$-y$				
0.0101	$r_4$				
		0.	0	1	1
		1.	0	0	0
		1.	0	1	1

$1 + 2^{-4}$  (correction)  
 $a/b = -\frac{9}{16}$

### Square Root

Of the many ways to find the square root of a number only three are described here. These are by (1) iteration, (2) subtraction of successive odd numbers, and (3) square root process.

**Iteration.**  $\sqrt{a}$  can be found from the iteration formula

$$x_{n+1} = \frac{1}{2} \left( x_n + \frac{a}{x_n} \right).$$

where  $x_n$  is the  $n$ th approximation of  $\sqrt{a}$ .

**Subtraction of Successive Odd Numbers.** There is a theorem in the theory of numbers that the sum of the first  $n$  odd numbers is  $n^2$ . Letting  $a$  be  $n^2$ , it can be seen that  $\sqrt{a}$  can be found by determining how many times successive odd integers can be subtracted from  $a$ .

**Square Root Process.** Square root can be found by a process akin to that learned in high school. Here it is modified so that restoring is not necessary. Let the positive number whose root is desired be called the zeroth remainder,  $r_0$ . The first divisor,  $p_1$ , is 0.01. If a remainder  $r_k$  is positive, the divisor  $p_{k+1}$  is subtracted from it. If a remainder  $r_k$  is negative,  $p_{k+1}$  is added to it. If  $r_{k+1}$  is positive, the triplet  $1 \cdot 2^{-(k+1)}$ ,  $0 \cdot 2^{-(k+2)}$ ,  $1 \cdot 2^{-(k+3)}$  is injected into  $p_{k+1}$  in place of the digits in the corresponding positions to obtain  $p_{k+2}$ . If  $r_{k+1}$  is negative, instead, the triplet  $0 \cdot 2^{-(k+1)}$ ,  $1 \cdot 2^{-(k+2)}$ ,  $1 \cdot 2^{-(k+3)}$  is injected.  $p_k$  differs from  $\sqrt{a}$  by less than  $3 \cdot 2^{-(k+2)}$ . An example is given below.

EXAMPLE.

$p_1$	0.01	0.100101100001	$a(=r_0)$
		.01	
$p_2$	0.101	0.0101	$r_1$
		101	
$p_3$	0.1101	0.000001	$r_2$
		1101	
$p_4$	0.11011	1.11010010	$r_3$
		11011	
$p_5$	0.110111	1.1110110100	$r_4$
		110011	
$p_6$	0.1100111	1.111110011101	$r_5$
		1100011	
$p_7$	0.11000101	0.000000000000	$r_6$

### Number System Conversion

A binary digital computer may have decimal inputs and outputs or Gray code inputs and outputs. The binary digital computer has to perform the conversions from code to code.

**Decimal to Binary.** A binary machine can convert a binary coded decimal integer to a binary number as follows. Treating each binary coded decimal digit as a four-bit binary number, first multiply the most significant digit by ten (=1010). To this product add the next decimal

digit and again multiply by ten. Continue adding decimal digits and multiplying by 10, until the last decimal digit is added in.

**Binary to Decimal.** To convert a binary integer to a binary coded decimal number it is necessary to divide successively by ten ( $=1010$ ), the remainders being the binary coded decimal digits in order—the units digit, the tens digit, etc.

EXAMPLE. Convert the binary number 11011010110 to a decimal number.

$$\begin{array}{r}
 \phantom{1010} \overline{10101111} \\
 1010 \overline{) 11011010110} \\
 \underline{1010} \phantom{000000000} \\
 1110 \phantom{000000000} \\
 \underline{1010} \phantom{000000000} \\
 10010 \phantom{000000000} \\
 \underline{1010} \phantom{000000000} \\
 10001 \phantom{000000000} \\
 \underline{1010} \phantom{000000000} \\
 1111 \phantom{000000000} \\
 \underline{1010} \phantom{000000000} \\
 1010 \phantom{000000000} \\
 \underline{1010} \phantom{000000000} \\
 0 \phantom{000000000} = 0
 \end{array}$$
  

$$\begin{array}{r}
 \phantom{1010} \overline{10001} \\
 1010 \overline{) 10101111} \\
 \underline{1010} \phantom{00000000} \\
 0001111 \phantom{00000000} \\
 \underline{1010} \phantom{00000000} \\
 0101 \phantom{00000000} = 5
 \end{array}$$
  

$$\begin{array}{r}
 \phantom{1010} \overline{01} \\
 1010 \overline{) 10001} \\
 \underline{1010} \phantom{00000000} \\
 0111 \phantom{00000000} = 7
 \end{array}$$
  

$$\begin{array}{r}
 \phantom{1010} \overline{0} \\
 1010 \overline{) 01} \\
 \underline{0000} \phantom{00000000} \\
 0001 \phantom{00000000} = 1 \\
 = 1750
 \end{array}$$

**Gray Code to Binary.** The binary equivalent of a Gray code number can be obtained by a rule. The most significant binary digit is the same

as the corresponding Gray code digit. Thereafter, the binary digit changes if the Gray code digit is a 1 and remains the same if the Gray code digit is a 0.

EXAMPLE. Convert the Gray code 110100110 to a binary number.

<i>Gray Code</i>		<i>Binary</i>
1	Same	→ 1
1	Change	→ 0
0	No change	→ 0
1	Change	→ 1
0	No change	→ 1
0	No change	→ 1
1	Change	→ 0
1	Change	→ 1
0	No change	→ 1

Hence the binary number is 100111011.

**Binary to Gray Code.** This conversion can be accomplished as follows: The most significant Gray code digit is the same as the corresponding binary digit. Thereafter the Gray code digit is 1 if the binary digit changes and 0 if it does not.

EXAMPLE. Convert the binary number 0110100 to Gray code.

<i>Binary</i>		<i>Gray</i>
0	Same	→ 0
1	Change	→ 1
1	No change	→ 0
0	Change	→ 1
1	Change	→ 1
0	Change	→ 1
0	No change	→ 0

Gray code = 0101110.

#### 4. DECIMAL OPERATIONS

##### Decimal Codes

There are very many possible decimal codes. Choice of a particular code will generally depend on system considerations.

There are weighted decimal codes in which each position has a numerical value associated with it. One such code would use 10 binary digits for the representation of a decimal digit, only one of the binary digits being permitted to be a 1. A more common code, known as "binary coded decimal" uses four binary digits for each decimal digit and assigns these binary digits the weights  $2^3$ ,  $2^2$ ,  $2^1$ , and  $2^0$ . Two other common weighted decimal codes assign the values 1, 2, 4, 7 or 1, 2, 2, 4 to the binary digits.

In addition to the weighted codes there are others with special advantages. One such code is the "excess 3" code, which can be obtained by adding 3 to the binary coded decimal representations of the decimal digits. One of the advantages of this code is that 10's complements can be obtained by substituting 0's for 1's and 1's for 0's.

Another code is the 2 out of 5 code, in which the representation of each decimal digit has exactly two 1's. An advantage of this code is that errors resulting in a different number of 1's are easily detected.

### Count

The way in which counting is done depends upon the code in use.

**One Out of Ten Code.** In a 1 out of ten code, counting can be done simply by shifting. Here a supplementary circuit must be supplied to shift the next higher decade by 1 when all the lower decades are changing from 9 to 0.

EXAMPLE. Representation of counting in 1 out of 10 code.

<i>N</i>	<i>Units</i>	<i>Tens</i>	<i>Hundreds</i>
	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9
795	0 0 0 0 0 1 0 0 0 0	0 0 0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 1 0 0
796	0 0 0 0 0 0 1 0 0 0	0 0 0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 1 0 0
797	0 0 0 0 0 0 0 1 0 0	0 0 0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 1 0 0
798	0 0 0 0 0 0 0 0 1 0	0 0 0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 1 0 0
799	0 0 0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 1 0 0
800	1 0 0 0 0 0 0 0 0 0	1 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 1 0

**Binary Coded Decimal.** In other codes counting may be more complicated. For example, in binary coded decimal each counter stage may be designed as a scale of 16 counter modified so that it will count only from 0 to 9 and then reset to zero.

EXAMPLE. To illustrate this, the scale of 16 counter of Sect. 3 is modified:

<i>N</i>	<i>D1</i>	<i>D2</i>	<i>D3</i>	<i>D4</i>	
0	0	0	0	0	$I1 = \overline{D1}$
1	1	0	0	0	
2	0	1	0	0	$I2 = D1 \overline{D2} \overline{D4} + \overline{D1} D2$
3	1	1	0	0	
4	0	0	1	0	$I3 = D1 D2 \overline{D3} + \overline{D1} D3 + \overline{D2} D3$
5	1	0	1	0	
6	0	1	1	0	$I4 = D1 D2 D3 + \overline{D1} D4$
7	1	1	1	0	
8	0	0	0	1	
9	1	0	0	1	

**Shift**

In a binary computer a shift of one bit position corresponds to multiplication by the radix 2. In the decimal computer, multiplication by the radix (10) requires that the contents of the four or more binary digit positions used to represent a decimal digit be shifted to the corresponding position for the next higher order digit. A like rule covers division by the radix.

**Add**

Decimal addition, too, depends upon the code employed.

Addition may be done by counting. In this scheme as the digits of one operand are counted down to zero the corresponding digits of the other operand are increased. Special provision must be made for carries between orders of digits.

Design of the computer may be such that if two decimal digits are to be *added* the digits of their binary representations are available serially. It may be possible to take advantage of this. For example, if the "excess 3" code is used, four bit digits from each of the operands may be added together as though they were binary digits, and then a simple correction may be performed to obtain the right answer; if there is no carry to the fifth digit, the correction consists of subtracting 3 from the sum; if there is a carry to the fifth place, the correction consists of adding 3 to sum. The addition or subtraction of 3 is accomplished by special circuitry. The above-mentioned carry is also a true carry to the next decimal digit.

If the four or more bits of a decimal digit must be added in parallel, the same techniques for designing a parallel binary adder may be employed to construct an adder capable of summing two decimal digits at a time.

**Subtract**

Decimal subtraction can be performed by true subtraction or by adding the complement of the subtrahend. In the latter case either 9 or 10's complement may be used. If a 9's complement is used, an "end-around carry" is needed to correct the difference; i.e., any overflow from the most significant digit must be added in again at the least significant position.

The discussion of codes and methods of decimal addition apply to decimal subtraction as well.

**Multiply**

There are many ways to perform decimal multiplication. Some are listed here.

**Russian Peasant Method.** In the Russian Peasant method the multiplicand is written at the top of one column, and at the top of a second column the multiplier is written. The multiplicand is repeatedly doubled and the multiplier repeatedly halved, and the successive results are written in columns one and two. In column three the remainders are written, and in column four are written those entries in column 1 preceding remainders of 1. The sum of column four is the desired product.

EXAMPLE.

$57 \times 38$			
114	19	0	
228	9	1	114
456	4	1	228
912	2	0	
1824	1	0	
	0	1	1824
			<u>2166</u>
			Product

**Adding and Shifting Method.** Decimal multiplication can also be performed by adding the multiplicand into the accumulator as many times as indicated by the least significant digit of the multiplier, then shifting the multiplicand one decimal place to the left and adding the multiplicand into the accumulator as many times as indicated by the second multiplier digit, and so on until the multiplier is exhausted.

EXAMPLE.

5792
$\times 352$
<u>5792</u>
5792
5792
5792
5792
5792
5792
5792
5792
5792
5792
<u>5792</u>
2038784

**Stored Multiples.** Another way of performing decimal multiplication is first to generate and store 1, 2, 3, . . . , 8, 9 times the multiplicand. Then the multiplier digits are used to select from these multiples the partial products to be accumulated.

EXAMPLE.

		Multiplicand 5792	
1 × 5792 =	5792		5792
2 × 5792 =	11584		×352
3 × 5792 =	17376	3	<hr style="width: 100%; border: 0.5px solid black;"/>
4 × 5792 =	23168	5	17376
5 × 5792 =	28960	2	28960
6 × 5792 =	34752		<hr style="width: 100%; border: 0.5px solid black;"/>
7 × 5792 =	40544		11584
8 × 5792 =	46336		<hr style="width: 100%; border: 0.5px solid black;"/>
9 × 5792 =	52128		2038784

**Multiplication Table.** In another method of performing decimal multiplication, the multiples of the multiplicand may be generated as needed, digit by digit, by building a decimal multiplication table into the computer. The multiplication table must produce a units digit and a tens digit for the product of two decimal digits.

The multiplication table may be constructed in terms of the decimal code used throughout the machine. It may also be constructed in a more convenient code with provision for translating the operands to this code and the results back to the normal machine code.

**Roundoff.** If a machine is equipped for decimal multiplication and the results are to be rounded, the usual rounding technique is to add 5 in the place just beyond the last place to be retained, permit all carries to propagate, and then drop all places to be discarded.

## Divide

**Restoring Division.** In a decimal machine division is usually performed by successive subtractions of the divisor until the remainder goes negative. If restoring division is being used, the divisor is added in once to give a positive remainder. The quotient digit is the number of subtractions performed less one. The remainder is then left shifted one decimal place and the subtraction process continued to obtain the next quotient digit.

**Nonrestoring Division.** If, on the other hand, nonrestoring division is being used, the divisor is not added back in when the remainder goes negative. Instead, the remainder is left shifted one decimal place and the divisor added in repeatedly until the remainder becomes positive. Again the remainder is left shifted one decimal place and subtraction begun, and so forth. If, in any decimal position nine additions or subtractions do not affect the sign of the remainder, the remainder is left shifted one decimal place and the additions or subtractions continued unchanged.

The number of subtractions or additions at each decimal place is recorded and forms a pseudo-quotient from which the quotient may be obtained. Subtractions result in a positive pseudo-quotient digit, and additions a negative pseudo-quotient digit. The true quotient is obtained from the pseudo quotient by adding and subtracting the quotient digits, starting at the least significant end.

EXAMPLE. Division methods.

<i>Restoring</i>	<i>Nonrestoring</i>
$  \begin{array}{r}  147 \quad \overline{) 3675441} \\  \underline{-441} \quad 3 - 1 = 2 \\  9926 \\  \underline{+147} \\  00735 \\  \underline{882} \quad 6 - 1 = 5 \\  99853 \\  \underline{+147} \\  000004 \\  \underline{-147} \quad 1 - 1 = 0 \\  999857 \\  \underline{+147} \\  0000044 \\  \underline{-147} \quad 1 - 1 = 0 \\  9999897 \\  \underline{+147} \\  00000441 \\  \underline{-588} \quad 4 - 1 = 3 \\  99999853 \\  \underline{+147} \\  00000000 \\  \hline  \text{Quotient} \quad 25003  \end{array}  $	$  \begin{array}{r}  147 \quad \overline{) 3675441} \\  \underline{-441} \quad 3 \\  99265 \\  \underline{+735} \quad \bar{5} \\  000004 \\  \underline{-147} \quad 1 \\  9998574 \\  \underline{+1323} \quad \bar{9} \\  99998971 \\  \underline{+1029} \quad \bar{7} \\  00000000 \\  \hline  \text{Pseudo-quotient} \quad 3 \quad \bar{5} \quad 1 \quad \bar{9} \quad \bar{7} \\  \phantom{\text{Pseudo-quotient}} \quad +3 \quad 0 \quad 0 \quad 0 \quad 0 \\  \phantom{\text{Pseudo-quotient}} \phantom{+3} \quad -5 \quad 0 \quad 0 \quad 0 \\  \phantom{\text{Pseudo-quotient}} \phantom{+3} \phantom{-5} \quad +1 \quad 0 \quad 0 \\  \phantom{\text{Pseudo-quotient}} \phantom{+3} \phantom{-5} \phantom{+1} \quad -9 \quad 0 \\  \phantom{\text{Pseudo-quotient}} \phantom{+3} \phantom{-5} \phantom{+1} \phantom{-9} \quad -7 \\  \hline  \text{Quotient} \quad 2 \quad 5 \quad 0 \quad 0 \quad 3  \end{array}  $

### Square Root

Restoring decimal square root is related to the manual square root process in the same way as restoring division (above) is related to the manual process of division.

## 5. SPECIAL OPERATIONS

A number of other operations may be performed with the arithmetic element of a digital computer. In some cases these operations could be

performed by means of a sequence of operations like addition, subtraction, division, and square root. In many instances, however, it costs very little equipment to mechanize these operations so that they are carried out by a single command. In many other instances the cost of these special operations is high but the convenience of having the special operation has outweighed cost. Some of these special operations are described here.

**Extract.** A variety of extract operations permit the programmer to pull out of any word only those digits which he wishes to retain. In one extract command, one operand, known as the extractor, indicates which digits are to be retained in the second operand. Let  $A_n$  be one digit of the extractor,  $B_n$  the corresponding digit of the second operand, and  $D_n$  the corresponding digit of the result. Then  $D_n = A_n \cdot B_n$ .

EXAMPLE.

$$\begin{array}{r} 000111100 \quad A \\ \underline{101101011} \quad B \\ 000101000 \quad D \end{array}$$

**Logical Transfer.** In another variety of extract command, known as a "logical transfer," there are three operands:  $A$  and  $B$  defined above, and a third operand  $C$ . Each digit  $D_n$  of the result can be expressed as  $D_n = A_n \cdot B_n + C_n$ . (Here  $+$  indicates logical sum.)

EXAMPLE.

$$\begin{array}{r} 000111100 \quad A \\ 101101000 \quad B \\ \underline{110001001} \quad C \\ 110101001 \quad D \end{array}$$

**Comparison.** Comparison commands are used for making decisions. A comparison command can be phrased in the form of a question that can be answered yes or no; for example, "Is  $a \geq b$ ?" If the answer is yes, the computer follows one routine. Otherwise the computer pursues an alternate routine. Some typical comparison commands are:

$$\text{Is } a \geq b ?$$

$$\text{Is } a > b ?$$

$$\text{Is } a = b ?$$

$$\text{Is } |a| \geq |b| ?$$

The above comparisons can be determined by performing an arithmetic operation. They can also be accomplished without an arithmetic operation if special circuitry is provided.

A compare command can also include an extract command, as, for example, "Is the logical product of the digits of the result of an extraction 1?"

**Checking.** One means for checking arithmetic, the 2 out of 5 code, was mentioned in Sect. 4. Another scheme, adaptable to machines of any radix, is modulus checking. It is familiar to most everyone in the decimal system as "casting out 9's." More formally, this is a modulo 9 check.

Let  $a$ ,  $b$ , and  $c$  be the sums modulo 9 of the digits of the decimal numbers  $A$ ,  $B$ , and  $C$ , and let  $\odot$  indicate addition, subtraction, or multiplication. Then if

$$A \odot B = C, \text{ it follows that } a \odot b = c \pmod{9}.$$

In binary computers another modulo check is useful. Let  $A$ ,  $B$ , and  $C$  be binary numbers, and  $a$ ,  $b$ , and  $c$  the sums of the bits taken with alternating signs. Then for three binary numbers where  $A \odot B = C$ , it follows that  $a \odot b = c \pmod{2}$ .

**Fixed Point Operation.** Thus far in this chapter attention has been restricted to fixed point digital computers in which a number has a fixed number of digits and a fixed decimal or binary point. Such machines have a comparatively limited range of numbers, and consequently the programmer is burdened with the need to watch scale factors closely or all significance in the result may be lost. Consider, for example, a digital computer having words consisting of up to 31 binary digits and a sign. The computer could then hold 0 or any positive or negative integer up to  $2^{31} - 1$ . If six more binary digits are added to each word, the computer can hold 0 or any positive or negative integer up to  $2^{37} - 1$ . If, instead, these six extra digits are used to indicate the sign and five binary digits of the power of two by which the original 31-bit number should be multiplied, the computer can specify 0 and any positive or negative number with magnitude between 1 and  $2^{62} - 2^{31}$  to 31 significant bits.

**Floating Point Operation.** A machine in which numbers are indicated by a number times a variable power of the radix is known as a floating point machine. In a floating point machine multiplication is accomplished by taking the product of the numbers and the sum of the exponents. Division is less trouble than in a fixed point machine because the relative magnitudes of the numbers are less important. Addition and subtraction, however, are difficult, for the number with the smaller exponent must be altered to have the same exponent as the other. If the number part of the result does not lie in bounds, it must be shifted and the exponent altered.

EXAMPLE 1. *Floating point addition.*

$$\begin{array}{r} 20,370,000. \times 10^{-9} + 59,250,000. \times 10^{-13} \\ 20,370,000. \times 10^{-9} \\ +00,005,925. \times 10^{-9} \\ \hline 20,375,925. \times 10^{-9} \end{array}$$

EXAMPLE 2. *Floating point addition.*

$$\begin{array}{r} 20,370,000 \times 10^{-9} + 89,000,000 \times 10^{-9} \\ 20,370,000 \times 10^{-9} \\ 89,000,000 \times 10^{-9} \\ \hline 109,370,000 \times 10^{-9} = 10,937,000 \times 10^{-8} \end{array}$$

## 6. CONTROL ELEMENTS

The design of a computer control unit is best illustrated by the actual design of such a unit. This design would indicate the many tricks that are available to the designer and the many choices he must make.

This section will be restricted to a general discussion of the functions of a control unit, and the design of a "toy" computer.

**Function of a Control Unit.** The function of the control unit is the interpretation of commands. When a command is received in the control unit, that unit must provide or gate the signals that will cause the computer to perform that command. The actions of the control unit may be divided into gating, timing, and counting.

1. First, the control unit must open those gates that will cause the command to be carried out. In the adder-subtractor of Sect. 3, for example, the signals  $M$  and  $N$  would be outputs of the control unit.  $M$ , being the signal to add, would cause the carry to be formed in the manner necessary for addition.  $N$ , the signal for subtraction, would cause the carry to be formed as needed for subtraction. This example is typical of computer design.

2. The second function of the control unit, timing, has not been mentioned at all in earlier sections in order to simplify the explanations there. Actually the gating signals from the control unit must be functions of time. Consider a machine with multiple address commands. A single command might well be "Multiply the number in storage location  $a$  by that in storage location  $b$ . Put the result in storage location  $c$  and pick up the next command from storage location  $n$ ." In sequence, the computer must transfer the contents of  $a$  to the arithmetic unit, transfer the contents of  $b$  to the arithmetic unit, form the product, round it, transfer the

result to  $c$ , and pick up a new command from  $n$ . Each of these operations may take a fixed length of time, in which case the control unit can use a digit counter to determine when the operation is completed. On the other hand, the time required for an operation may be a function of the operands, as in an asynchronous computer, and in this case the arithmetic unit may signal to the control unit the completion of the operation.

3. The third function of the control unit is counting. A command may say, for example, "Shift the contents of the  $A$  register left five places," or in a relative address machine a command may state, "Wait 11 words before picking up the next command." In these instances and others, the control unit must count the number of times an operation occurs to find when that operation is completed.

**External Control.** Control of the digital computer may be external to the computer, as in a punched card sorter, a desk calculator or an abacus, where the human operator may enter every operand and determine every command. A less elementary form of control is wired control, which has appeared on a number of punched card machines and a few electronic calculators. A machine with wired control has a plugboard that can be wired to make the machine perform sequences of orders. A more sophisticated form of control is internal control, in which the computer performs long sequences of orders stored in its memory unit or stored on external media under the control of the computer.

**Programmed Control.** Programmed control comprises all forms of control in which the computer performs long sequences of varying commands without intervention from a human operator; that is, a machine for which the sequence of commands has been predetermined by the designer or by a programmer is said to have programmed control. In such a machine, once the start button is pressed, the control unit will cause the computer to perform commands according to the instructions within the machine or accessible to the machine. The sequence of commands, called a program, may be very simple as would be required to obtain the sum  $\sum a_i b_i$ ; or it might be very complicated, as would be the case in solving sets of partial differential equations, with mixed boundary conditions.

Digital computers have no intelligence; they can only perform operations that are spelled out in detail in terms of the limited variety of commands that the computer has. This means that finding the solutions of an equation like  $x = \tan x$ , which is fairly simple for a desk calculator operator to do if supplied with tangent tables, may involve a long sequence of commands in a computer having addition, subtraction, multiplication, and comparison as its arithmetic functions. See Chap. 2, Programming and Coding.

## Design of a Control Unit

The functions of a control unit will become clearer in the following example. Design of a very simple computer will be used just to indicate the techniques used in designing a control unit, and to illustrate the functions of a control unit:

**Description of a Hypothetical Computer.** Numbers in this fixed point serial binary computer are less than one in magnitude, and negative numbers are represented by 2's complements. Numbers are transferred least significant digit first; the sign, which is represented by the one digit to the left of the binary point, comes last.

The computer memory consists of a flip-flop shifting register  $A$ , and a magnetic drum. The drum has four bands for commands and four bands for numbers.

The commands the computer can perform are:

1. Tab Transfer contents of address  $a$  to address  $b$ .
2. Aab Add contents of address  $a$  to contents of  $A$ , place result in address  $b$ , and replace contents of  $A$  by contents of address  $a$ .
3. Buv If the contents of  $A$  are  $> 0$ , pick up the next command in the present command band; otherwise pick up the next command in command band  $u$ , waiting  $v$  word times ( $v = 1, 2, 3$ ).
4. S Stop. Carry out no more orders.

Since there are only four different commands, the states of two flip-flops considered together suffice to specify the type of command. These two flip-flops in the control unit will be called  $X1$  and  $X2$ .

Since there are only four sources of numbers that must be specified as  $a$  addresses, two flip-flops,  $X3$  and  $X4$ , serve this function. Similarly  $X5$  and  $X6$  indicate the  $b$  address.

Addresses  $u$  and  $v$ , also, each require two flip-flops, but  $X3$  through  $X6$  shall be time shared for this purpose. Two additional flip-flops,  $X7$  and  $X8$ , however, are required to control from which band the current command is read.

Furthermore, since all commands are effected in one word time (other than S), another register,  $P1$  through  $P6$ , is provided to hold the succeeding command for parallel transfer into the command register  $X1$  through  $X6$ .

In addition, a *relative address scheme* will be used for orders, the next order in sequence in the current command channel always being the next picked up, except in Buv orders, where a following command may be chosen from a selected channel.

Each number will consist of 5 bits and a sign, so that numbers and

commands have the same number of digits. This is convenient but not really required. The  $A$  register, then, consists of  $A1$  through  $A6$ .

A flip-flop  $C1$  is needed to hold carries in addition. The reading amplifiers for the command channels are named  $R1$  through  $R4$ , the reading

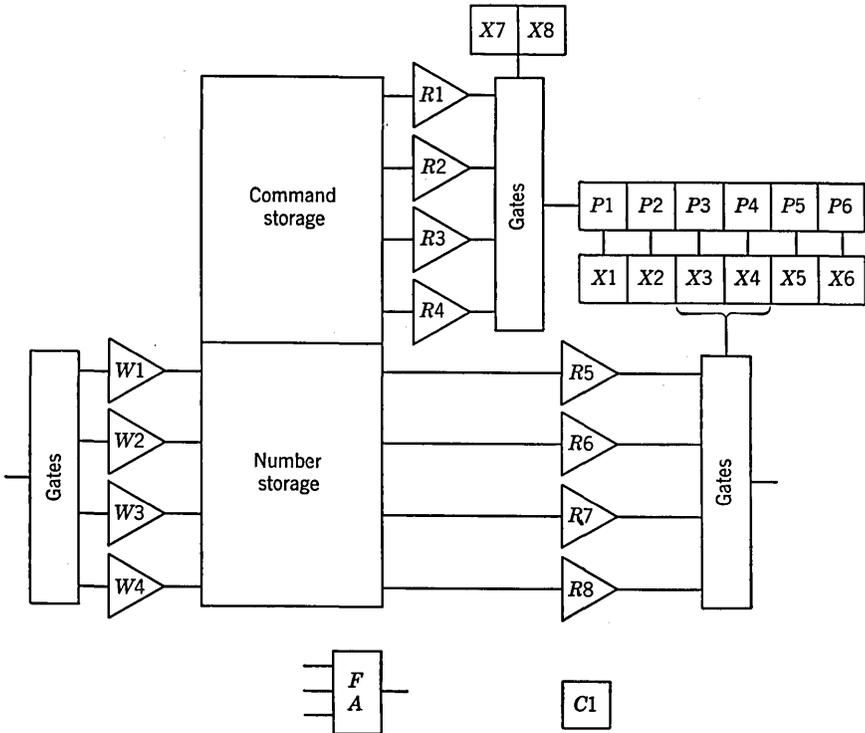


FIG. 11. Outline block diagram of computer.

amplifiers for the number channels  $R5$  through  $R8$ , and the writing amplifiers for the number channels  $W1$  through  $W4$ . The read and write heads are separate.

A partial block diagram of this computer may now be drawn. See Fig. 11. All flip-flops are set-reset flip-flops. (See Sect. 1.)

**Logical Equations.** Next, the logical equations for this computer can be written. To do this symbols must be introduced for digit times. Each word will consist of six information bits and a space bit. The digit time during which the least significant bit of a number is available at the output of a reading amplifier is called digit time one and is represented by  $T1$ . The succeeding digit times are  $T2$ ,  $T3$ ,  $T4$ ,  $T5$ ,  $T6$ , and  $T7$ .

The  $A$  register shifts at all digit times other than  $T7$ , so it follows from Sect. 3 that

$$\begin{aligned} S(A2) &= A1 \overline{T7}, & S(A4) &= A3 \overline{T7}, \\ R(A2) &= \overline{A1} \overline{T7}; & R(A4) &= \overline{A3} \overline{T7}; \\ S(A3) &= A2 \overline{T7}, & S(A5) &= A4 \overline{T7}, \\ R(A3) &= \overline{A2} \overline{T7}; & R(A5) &= \overline{A4} \overline{T7}; \\ S(A6) &= A5 T7, \\ R(A6) &= \overline{A5} \overline{T7}. \end{aligned}$$

**Inputs to  $A1$ .**

The inputs to  $A1$  depend upon the command being performed. The four different kinds of commands are indicated thus

Tab	$\overline{X1} \overline{X2}$
Aab	$\overline{X1} X2$
Buv	$X1 \overline{X2}$
S	$X1 X2$

The input to  $A1$  may now be written, for in the command Add the contents of register  $A$  are to be the contents of address  $a$

$$\begin{aligned} S(A1) &= \overline{X1} X2 [\overline{X3} \overline{X4} R5 + \overline{X3} X4 R6 + X3 \overline{X4} R7 + X3 X4 R8] \overline{T7}, \\ R(A1) &= \overline{X1} X2 [\overline{X3} \overline{X4} \overline{R5} + \overline{X3} X4 \overline{R6} + X3 \overline{X4} \overline{R7} + X3 X4 \overline{R8}] \overline{T7}, \end{aligned}$$

whereas in all other commands the contents of  $A$  are to be unchanged

$$\begin{aligned} S(A1) &= (\overline{X1} X2) A6 \overline{T7}, \\ R(A1) &= (\overline{X1} X2) \overline{A6} \overline{T7}. \end{aligned}$$

The quantity in the brackets in the first equation for  $S(A1)$  will be used in a number of places. In order to save gating equipment an amplifier named  $Y1$  is introduced. The input to this amplifier is called  $L(Y1)$ , and it has the outputs  $Y1$  and  $\overline{Y1}$ . Then

$$L(Y1) = \overline{X3} \overline{X4} R5 + \overline{X3} X4 R6 + X3 \overline{X4} R7 + X3 X4 R8.$$

The equations for flip-flop  $A1$  now become

$$\begin{aligned} S(A1) &= \overline{X1} X2 Y1 \overline{T7} + X1 A6 \overline{T7} + \overline{X2} A6 \overline{T7}, \\ R(A1) &= \overline{X1} X2 \overline{Y1} \overline{T7} + X1 \overline{A6} \overline{T7} + \overline{X2} \overline{A6} \overline{T7}. \end{aligned}$$

The sum in an Add order is used as the input to an amplifier  $Y2$ . It is

$$L(Y2) = A6 \overline{Y1} \overline{C1} + \overline{A6} Y1 \overline{C1} + \overline{A6} \overline{Y1} C1 + A6 Y1 C1.$$

**Write Amplifiers.** The storage drum write amplifiers have two inputs. The input named  $M$  causes 1's to be written. The input named  $N$  causes 0's to be written. If there is no input, nothing is written. In a Transfer order the quantity to be written is  $Y1$ . In an Add order it is  $Y2$ .

$\overline{X1}$  is required to write 1's or 0's.  $\overline{X2}$  indicates  $Y1$  is to be written.  $X2$  indicates  $Y2$  is to be written.  $X5$  and  $X6$  specify the particular write amplifier. The inputs to the write amplifiers are then

$$M(W1) = \overline{X1} \overline{X2} Y1 \overline{X5} \overline{X6} + \overline{X1} X2 Y2 \overline{X5} \overline{X6},$$

$$N(W1) = \overline{X1} \overline{X2} \overline{Y1} \overline{X5} \overline{X6} + \overline{X1} X2 Y2 \overline{X5} \overline{X6};$$

$$M(W2) = \overline{X1} \overline{X2} Y1 \overline{X5} X6 + \overline{X1} X2 Y2 \overline{X5} X6,$$

$$N(W2) = \overline{X1} \overline{X2} \overline{Y1} \overline{X5} X6 + \overline{X1} X2 Y2 \overline{X5} X6;$$

$$M(W3) = \overline{X1} \overline{X2} Y1 X5 \overline{X6} + \overline{X1} X2 Y2 X5 \overline{X6},$$

$$N(W3) = \overline{X1} \overline{X2} \overline{Y1} X5 \overline{X6} + \overline{X1} X2 Y2 X5 \overline{X6};$$

$$M(W4) = \overline{X1} \overline{X2} Y1 X5 X6 + \overline{X1} X2 Y2 X5 X6,$$

$$N(W4) = \overline{X1} \overline{X2} \overline{Y1} X5 X6 + \overline{X1} X2 Y2 X5 X6.$$

Flip-flop  $C1$  is used only for storing carries in the command Add. Its inputs in that command are

$$S(C1) = Y1 A6,$$

$$R(C1) = \overline{Y1} \overline{A6}.$$

In an Add command, the carry must be initially zero, and this is accomplished by always using  $T7$  to set  $C1$  to zero. Then

$$S(C1) = Y1 A6 \overline{T7},$$

$$R(C1) = \overline{Y1} \overline{A6} + T7.$$

Observe that it has not been necessary to restrict inputs to  $C1$  to just the command Aab because the output of  $C1$  affects the action of the computer only in the command Aab.

**Command Register.** The flip-flops  $X1$  through  $X6$  of the command register receive new commands in parallel from the preparatory register at  $T7$  unless the command in the command register is S or a branch command that causes a command channel change with wait time not yet exhausted. A stop order is  $X1 X2$ , a branch command causing a channel change and with wait time not yet exhausted is indicated by  $T7 A1 X1 \overline{X2} [X5 + X6]$ . Letting  $L(Y3)$  be the signal to permit a transfer from the preparatory register to the command register, it follows

$$\begin{aligned} L(Y3) &= \overline{(\overline{X1} \overline{X2})(T7 A1 X1 \overline{X2} [X5 + X6])} T7 \\ &= \overline{X1} T7 + \overline{X2} \overline{A1} T7 + \overline{X2} \overline{X5} \overline{X6} T7. \end{aligned}$$

If there is a branch command with wait time not yet exhausted  $X_5$ ,  $X_6$  must count down. What happens to  $X_5$  and  $X_6$  during  $S$  is not important, so  $\overline{Y_3}$  can be used as the signal for  $X_5$ ,  $X_6$  to count down.

$$\begin{aligned} X_1S &= Y_3 P_1, & X_3S &= Y_3 P_3, & X_5S &= Y_3 P_5, \\ X_1R &= Y_3 \overline{P_1}; & X_3R &= Y_3 \overline{P_3}; & X_5R &= Y_3 \overline{P_5} + \overline{Y_3} \overline{X_6} T_7; \\ X_2S &= Y_3 P_2, & X_4S &= Y_3 P_4, & X_6S &= Y_3 P_6 + \overline{Y_3} \overline{X_6} T_7, \\ X_2R &= Y_3 \overline{P_2}; & X_4R &= Y_3 \overline{P_4}; & X_6R &= Y_3 \overline{P_6} + \overline{Y_3} X_6 T_7. \end{aligned}$$

The *preparatory register*,  $P_1$  through  $P_6$ , is a shifting register with  $P_2$  through  $P_6$  having as inputs the outputs of the next lower numbered flip-flops. The input to  $P_1$  is the order read head selected by  $X_7$  and  $X_8$ .

$$\begin{aligned} S(P_1) &= \overline{X_7} \overline{X_8} R_1 + \overline{X_7} X_8 R_2 + X_7 \overline{X_8} R_3 + X_7 X_8 R_4, \\ R(P_1) &= \overline{X_7} \overline{X_8} \overline{R_1} + \overline{X_7} X_8 \overline{R_2} + X_7 \overline{X_8} \overline{R_3} + X_7 X_8 \overline{R_4}; \end{aligned}$$

$$S(P_2) = P_1,$$

$$R(P_2) = \overline{P_1};$$

$$S(P_3) = P_2,$$

$$S(P_5) = P_4,$$

$$R(P_3) = \overline{P_2};$$

$$R(P_5) = \overline{P_4};$$

$$S(P_4) = P_3,$$

$$S(P_6) = P_5,$$

$$R(P_4) = \overline{P_3};$$

$$R(P_6) = \overline{P_5}.$$

Flip-flops  $X_7$  and  $X_8$  are set from  $X_3$  and  $X_4$  in a branch command if the sign of the contents of  $A$  is negative. This decision can be made at  $T_7$  by inspecting  $A_1$  at that time.

$$S(X_7) = X_1 \overline{X_2} X_3 T_7 A_1,$$

$$S(X_8) = X_1 \overline{X_2} X_4 T_7 A_1,$$

$$R(X_7) = X_1 \overline{X_2} \overline{X_3} T_7 A_1;$$

$$R(X_8) = X_1 \overline{X_2} \overline{X_4} T_7 A_1.$$

Throughout this design it has been necessary to use only one digit timing pulse, namely  $T_7$ . Consequently a digit time counter is not needed. Instead,  $T_7$  may be derived directly from a track on the drum that has one pulse per word.

**Summary.** The equations describing the logic of a simple digital computer have now been written. This particular computer is useless because means have not been provided for starting it, nor any connections to the external world for input or output of data. Even though the computer is useless, the description is not, for it shows how a control element may be designed and functions. It also demonstrates the fact that the design of the arithmetic element may be intertwined with that of the control element.

REFERENCES

1. R. F. Shaw, Arithmetic operations in a binary computer, *Rev. Sci. Instr.*, **21**, 687 (1950).
2. A. D. Booth, A signed binary multiplication technique, *Quart. J. Mech. and Appl. Math.*, **4**, Pt. 2, 236 (1951).
3. R. K. Richards, *Arithmetic Operations in Digital Computers*, Van Nostrand, Princeton, N. J., 1955.
4. E. E. Bolles and H. L. Engel, Control elements in the computer, *Con. Engr.*, **3**, 93-98 (1956).
5. M. Phister, Jr., *Logical Design of Digital Computers*, Wiley, New York, 1958.

## Storage\*

*David R. Brown and Jack I. Raffel*

1. Basic Concepts	19-01
2. Magnetic Drum Storage	19-04
3. Magnetic Core Storage	19-13
4. Other Storage Techniques	19-29
References	19-33

### 1. BASIC CONCEPTS

**The Storage Unit.** The storage unit of a general purpose digital computer stores both the instructions which control machine operation and the data which are to be processed. The principal information transfer paths between the storage unit and other parts of a typical general purpose computer are shown in Fig. 1. Instructions may be transferred from the storage unit to the control, and storage addresses transferred from the control to the storage unit. Numbers or instructions may be transferred between the storage unit and the arithmetic unit and between the storage unit and the input and output equipment.

**Principal Parts of the Storage Unit.** The principal parts of a typical storage unit are: the address register, selection circuits, storage medium, storage (or buffer) register, and storage control. The information transfer paths between these parts are shown in Fig. 2.

\* Many of the examples and circuits in this chapter are the result of research programs supported jointly by the Army, the Navy and the Air Force, under contract with the Massachusetts Institute of Technology.

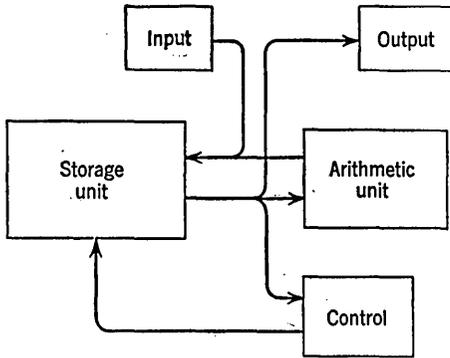


FIG. 1. Principal information paths between storage and other units of the computer.

Sometimes more than one storage register may be used. The storage register is used as a buffer between the storage medium and the other elements of the computer such as the arithmetic registers or the input and output units. In some computers this register is time-shared and utilized as one of the arithmetic registers.

#### Principle of Operation.

Words are read from or stored in a typical storage unit in the following sequence.

1. The address of the word to be read or written is transferred to the address register and, if the word is to be written, the word is transferred to the storage register.

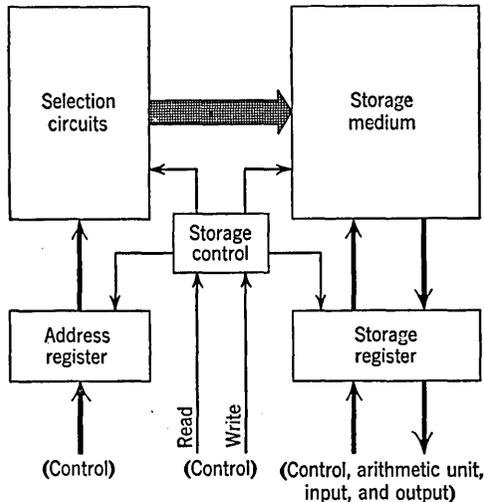


FIG. 2. Simplified block diagram of a storage unit. Connecting line width indicates roughly the number of parallel conductors.

2. The selection circuits decode the address and select the storage location involved.

3. Storage control is informed, by the central control, when to begin the reading or writing process.

4. If reading, the word is then transferred from the storage medium to the storage register.

5. If writing, the word is transferred from the storage register to the storage medium.

**Characteristics.** The important characteristics of a storage unit are: capacity, speed, reliability, cost, power requirement, and physical size. The emphasis placed on each factor varies with the particular application. The first three characteristics require definition.

**Capacity.** Storage capacity is generally measured by the equivalent number of bits or in terms of the number of words and digits of a stated base; e.g., 4096 32-bit words or 131,072 bits, or 4096 8-decimal-digit words.

**Speed.** Measurement of the speed of a particular storage unit is least ambiguous when accompanied by a statement of the type of access used. Two classes of access are recognized.

1. A *random access* storage unit is one in which the time required to read or write a word is independent of the location. The access time is the time required from the instant the address has been transferred to the address register until the word is available in the storage register, or until the word has been transferred from the storage register to the storage medium. The minimum time from the beginning of one access to the beginning of the next is often important and is sometimes called cycle time; it is generally longer than the access time. *Example.* Magnetic core storage unit.

2. A *sequential access* storage unit is one in which the access time is dependent on the location. Maximum, minimum, and average access times are often stated, also the minimum time from the beginning of one access to the beginning of the next. *Example.* Units which use cyclical mechanical positioning (drums) or pulse regeneration (delay lines).

**Reliability.** Three important measures of reliability are:

1. Mean time between malfunctions during scheduled operation.
2. Percentage of scheduled operating time lost because of malfunctions.
3. The number of hours per week of scheduled maintenance required.

The most common checking technique for detecting storage malfunctions is the use of a word parity check which requires an additional bit per word and the equipment needed to calculate and check parity. Redundance in other forms is sometimes used to increase storage reliability. Critical parts, or an entire storage unit, can be duplicated. Self-checking codes can be used, transfers can be repeated, or words can be stored in more than one location.

**Storage Hierarchies.** A hierarchy of different storage units within a single machine may be used. The advantages of each type for handling

different parts of the overall machine storage function may then be realized. For instance, a relatively small, high-speed, random access storage unit is commonly supplemented by a large-capacity, relatively slow, sequential access unit. Auxiliary storage, in the form of punched paper tape, magnetic tape, or punched cards, is also used extensively. (See Chaps. 5 and 20.)

The trend in recent years has been toward greater use of magnetic core storage, supplemented by magnetic drums, further supplemented by magnetic tapes.

**Storage Techniques.** Storage techniques and the resulting characteristics of storage units have had a dominating influence on the design of digital computers and data processing systems. For a history of the development of storage techniques see Ref. 3.

1. Storage capacity is a fundamental characteristic of the system, establishing the size of problem which can be undertaken or the amount of information which can be processed.

2. The access time of the storage unit often determines the speed of operation of the system.

Magnetic drum storage and magnetic core storage, the two most important types used for computer storage, will be discussed in some detail in this chapter; other types will be described only briefly.

## 2. MAGNETIC DRUM STORAGE

**Principle of Operation.** A cylindrical drum, rotating with constant angular velocity, may have information written on, or read from, its magnetic surface by means of magnetic heads. Figure 3 is a photograph of a 4-inch diameter magnetic drum. Usually, many tracks of information are spaced along the axis of the drum, each track having a single head used for both reading and writing. Information can be read only at the times when that place where the information was written is passing under a read-write head. This means that time (the angular position of the drum) becomes one of the selection coordinates.

**Block Diagram.** A simplified block diagram of a parallel magnetic drum storage unit is shown in Fig. 4. Many drum systems record words in a serial manner; however, the principles of operation are the same for both serial and parallel drum storage systems. In this example, the 16 bits (binary digits) of a word, plus one parity bit, are read or written in parallel on any one of 12 bands, each band having 17 tracks with read-write heads. Words are stored at 2048 angular positions, or slots, giving a total capacity of 24,576 words. Fifteen bits are required to address all the storage locations, 4 for band selection and 11 for angular position selection. The reading or writing cycle is as follows.

1. The 4 flip-flops of the address register which are used for band selection provide the input for a band selection decoder which selects one of the 12 bands in which reading or writing is to take place.
2. The storage control is informed, by central control, whether to read or write.

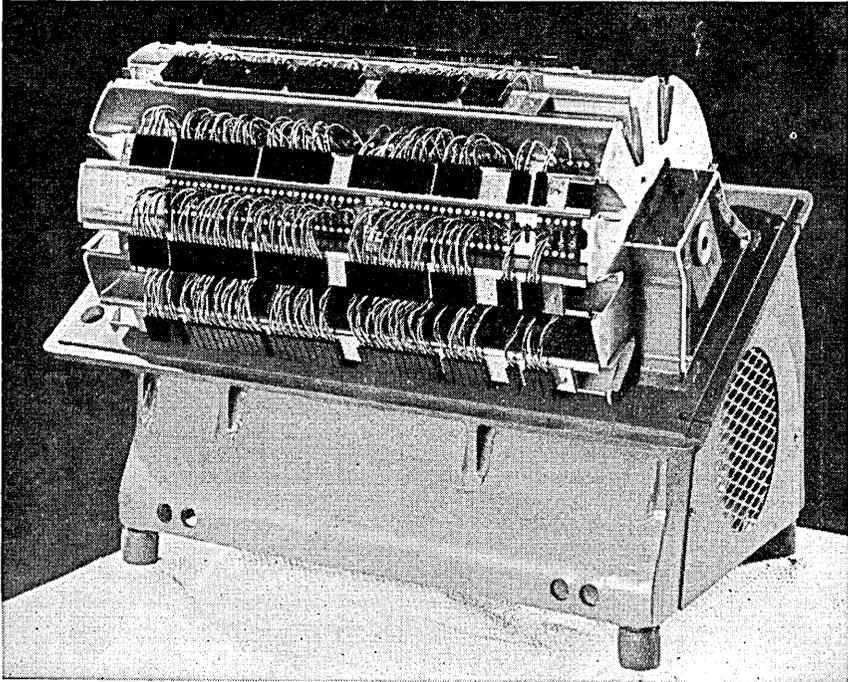


FIG. 3. IBM Type 650 magnetic drum. The drum is 16 in. long and 4 in. in diameter.

3. The 11 flip-flops of the address register which are used for angular position selection are compared, in a coincidence detector, with an angular position counter which is indexed by pulses derived from a separate timing track permanently recorded on the drum.

4. Upon coincidence, the coincidence detector sends a pulse to storage control which, in turn, causes the reading or writing to take place in the selected band and sends an operation-complete pulse to the central control.

**Access Time and Interlace.** The average access time is one-half a drum revolution time, and the maximum access time is a complete revolution time. The time between two adjacent angular positions, or slots, however, is much shorter.

The drum described (see Fig. 4) is an auxiliary storage unit designed by Engineering Research Associates for the Massachusetts Institute of Technology's Whirlwind I computer. A complete drum revolution requires 16 milliseconds, and adjacent slots are 8 microseconds apart. The central computer can process each word in slightly less than 32 microseconds. Therefore, the circuits comparing the angular position counter

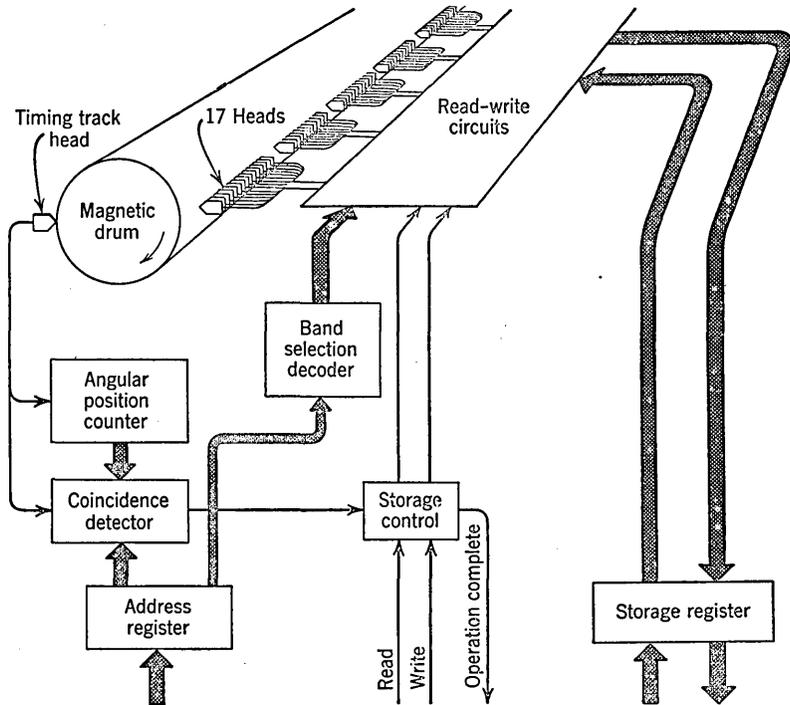


FIG. 4. Simplified block diagram of a magnetic drum storage unit. Width of connecting lines indicates roughly the number of parallel conductors.

with the address register are arranged to provide an interlace of four by having consecutive addresses refer to every fourth slot on the drum. In this case then, consecutive addresses appear every 32 microseconds. Since the computer frequently refers to consecutive addresses (e.g., block transfers), the effective access time is made much less than the average access time.

**Forced Coding.** In addition to interlace, a technique known as forced or minimum latency coding can reduce the effective access time. Here the program is coded and storage locations are allocated so that the desired storage location will become available just after the address is transferred to the storage unit by central control.

**Checking.** In the example, a parity bit is determined for each word just before it is stored, and a parity check is performed whenever a word is read from the drum. Other checking schemes may be used, as discussed previously.

TABLE 1. SOME OPERATING DRUM SYSTEMS<sup>a</sup>

	IBM 650	ERA 1110
Diameter, inches	3.98	22
Length, inches	16	29
Nominal rpm	12620	1190
Capacity, kilobits	124	2552
Maximum access time, milliseconds	4.8	51
Scan rate, kilobits/second	126.2	110
Circular packing density, bits/inch	48	80
Bits/track	600	5500
Number of tracks	207	464
Recording method	RZ	RZ
Same head read write	Yes	Yes
Head spacing, inches	0.0005-0.001	0.002
Horsepower drive	$\frac{1}{2}$	$1\frac{1}{2}$

<sup>a</sup> Bits is used as an abbreviation for binary digits.

**Drum Construction.** The dimensions and angular velocities of drums vary over a wide range as indicated in Table 1 (Ref. 6). Sometimes a disk, rather than a drum, is used, with a magnetic coating on one or both sides of the disk. Normally in drum systems, the heads do not make contact with the drum surface; wear would be too severe. The most common technique is to mount the head rigidly to the same assembly which holds the drum bearings, although air-floated heads and some other techniques have been used. When the heads are rigidly mounted, a spacing of 0.001 to 0.002 inch is common. The specification on drum runout (total indicator reading) may be less than 0.0002 inch; precision machining is necessary. Special attention must be given to prevent strains which might be caused by centrifugal force, thermal stress, etc.

**Drum Coatings.** Characteristics of some iron oxide and metal drum coatings are listed in Table 2.

**Heads.** Typical read-write heads are shown in Fig. 5*a, b*. The air gap, sometimes filled with a nonmagnetic spacer, plays an essential role in both reading and writing. The head is mounted close to the drum surface, as indicated in Fig. 5*c*, so that the fringing field in the vicinity of the air gap will be sufficient to magnetize the drum surface during writing and so that the remanent magnetization in the drum coating will generate a voltage in the head winding during reading. The width of the air gap is usually less than 0.001 inch. The head itself may be made up of thin laminations of a metal like Permalloy, or a high-permeability, low-loss

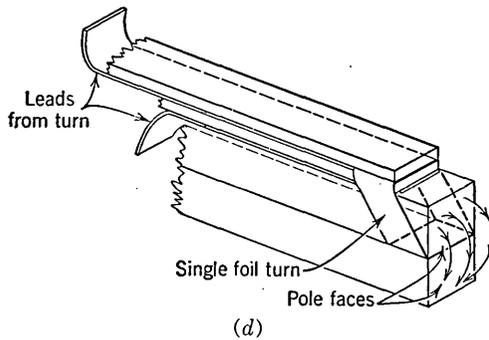
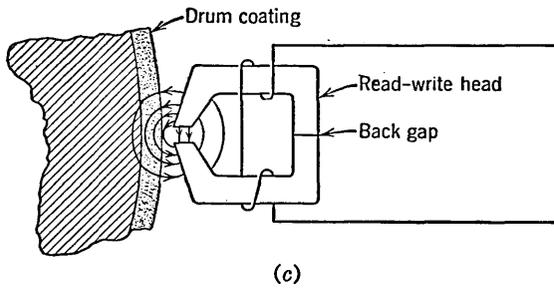
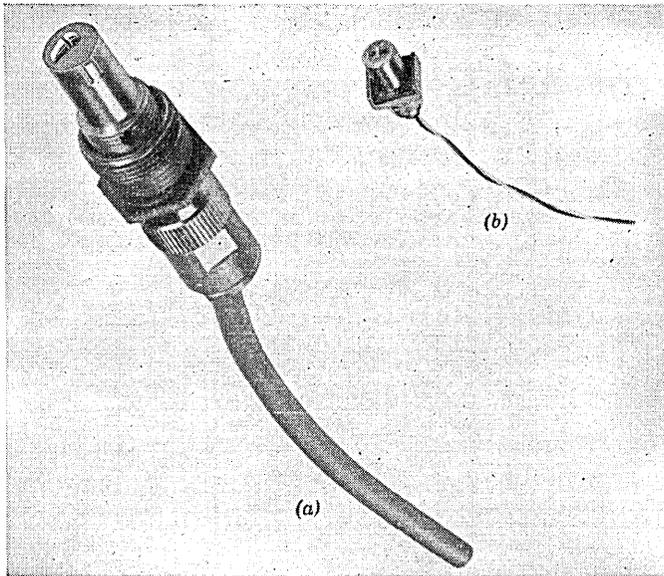


FIG. 5. Drum heads: (a) ERA type 202,  $\frac{3}{8}$ -in. outer diameter; (b) ERA type 205,  $\frac{1}{4}$ -in. outer diameter; (c) sketch of conventional read-write head; (d) single turn head.

TABLE 2. CHARACTERISTICS OF SOME DRUM COATINGS

Material	Minnesota Mining & Manufacturing Co.	Brush Electronics Co.
	Oxide	80% Co, 20% Ni
Manufacturer's type	RD3010, R1519	
How applied to drum	Sprayed	Electroplated
Thickness, inches	0.001-0.002	0.0005
Coercivity, oersteds	200, 250	220-240
Retentivity, gauss	450	12,000

ferrite. It is usually made in two pieces, the windings being set in place before final assembly of the head. Other types of heads, e.g., the single turn head, have been utilized.

The single turn head (Ref. 7), shown in Fig. 5*d*, uses a yoke formed by two rectangular blocks separated only by a single foil turn placed between them at the extremes (flush with the pole faces). This construction forces all the flux produced to pass outside the head and eliminates the additional wasted flux which crosses the gap in the conventional head. This type of head provides higher efficiency, compactness, ease of construction, and simplified shielding and gap positioning.

**Writing Techniques.** The writing techniques discussed in the following paragraphs are the most common. However, numerous other techniques for writing have been proposed, some using a modulated carrier.

**Return-to-Zero.** The return-to-zero (RZ) technique is illustrated in Fig. 6*a*, where the writing current or longitudinal flux pattern is shown. For one kind of binary digit, the flux changes from zero to an extreme value and returns to zero; for the other kind the flux changes in the opposite sense. Ordinarily the extreme values are positive and negative saturation. The writing pulses may be very short compared with the time between slots; the pulse duration used will depend upon the head design, drum speed, drum coating, etc. The drum surface must be initially erased by an a-c erase current, but properly timed writing currents, during normal operation, automatically erase, or write over, information previously stored.

**Return-to-Bias.** A return-to-bias (RB) technique, shown in Fig. 6*b*, is similar to the RZ technique except that the reference level is one of the saturation states instead of zero flux. The flux changes from one extreme value to the other and returns for one kind of binary digit, and does not change for the other kind. A d-c erase current is used initially, instead of an a-c erase current.

**Non-Return-to-Zero.** A non-return-to-zero (NRZ) technique is shown in Fig. 6*c*. Here one extreme value of flux corresponds to 1 and

the other to 0. A flux change occurs only when a digit differs from the digit immediately preceding. The initial state of magnetization of the drum surface is not important; writing automatically erases information previously stored. This technique permits higher linear cell densities than can be obtained with the RZ or RB techniques. This is because

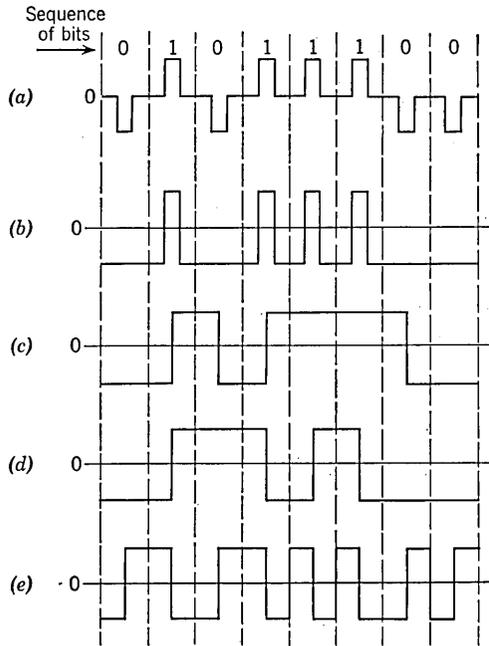


Fig. 6. Write current or flux pattern for a sequence of bits for (a) three-level RZ, (b) RB, (c) two-level NRZ, (d) NRZI, and (e) Manchester.

the shortest region of unidirectional magnetization is about twice as long in the case of the NRZ technique, for the same linear cell density. The NRZ technique can be used effectively only in a serial storage unit, whereas the RZ and RB techniques can be used in serial or parallel storage units. Also, the NRZ technique has the disadvantage of requiring writing current continuously, when writing a sequence of bits, with the possibility of a large d-c component.

**Non-Return-to-Zero, Invert.** A variation of the NRZ technique, known as NRZI, is illustrated in Fig. 6d. The flux changes from one extreme to the other for each 1, and does not change for a 0.

**Manchester.** Another variation of the NRZ technique, known as the phase reversal or Manchester technique, is illustrated in Fig. 6e. It eliminates the d-c component of writing current. The flux changes be-

tween the extreme values for each digit. A 1 has one extreme value followed by the other, and a 0 has the opposite sequence.

**Writing Circuits.** Circuits for generating the writing current vary widely, and are closely related to the selection circuits used. Circuits employ relays, thyratrons, hard tubes, magnetic cores and crystal diodes, and transistors. The writing circuits used for the Whirlwind I auxiliary drum are indicated in Fig. 7; a 2-microsecond, 0.5-ampere pulse is applied to the 30-turn winding on the head. The circuit is designed to permit the maximum possible writing rate, every 8 microseconds. The RZ technique is used.

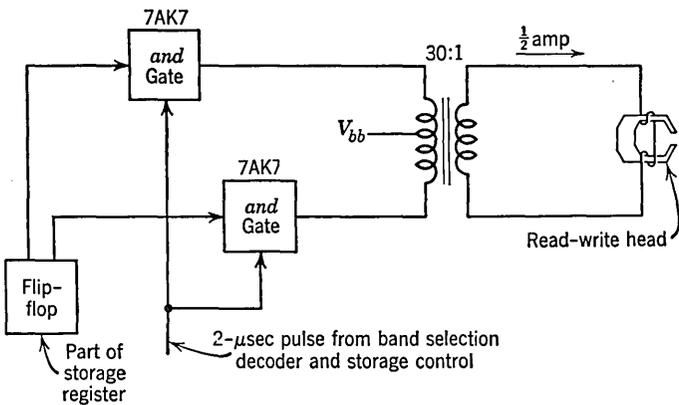


FIG. 7. A writing circuit for a magnetic drum using the three-level RZ technique.

**Reading Circuits.** The circuits used for reading are also extremely varied. The signal at the head terminals may be a few millivolts, or a few volts. The operations performed on the read signal may be complex, depending very much on the writing technique being used. With the RZ or RB techniques, the signal is usually amplified, differentiated, and then strobed by a delayed timing pulse (supplied by a storage control). When an NRZ technique is used, some logical manipulations are necessary to restore the information to standard form. With very high cell densities on large drums, strobing with timing pulses from a reference track may not be practical, and self-strobing schemes must be used.

**Effect of Loss of Power.** A magnetic drum storage unit can be designed so that the information stored will not be lost in case the power being supplied to the system is interrupted. The design must ensure that the sequence in which different voltages are lost, or turned on, does not result in spurious writing currents.

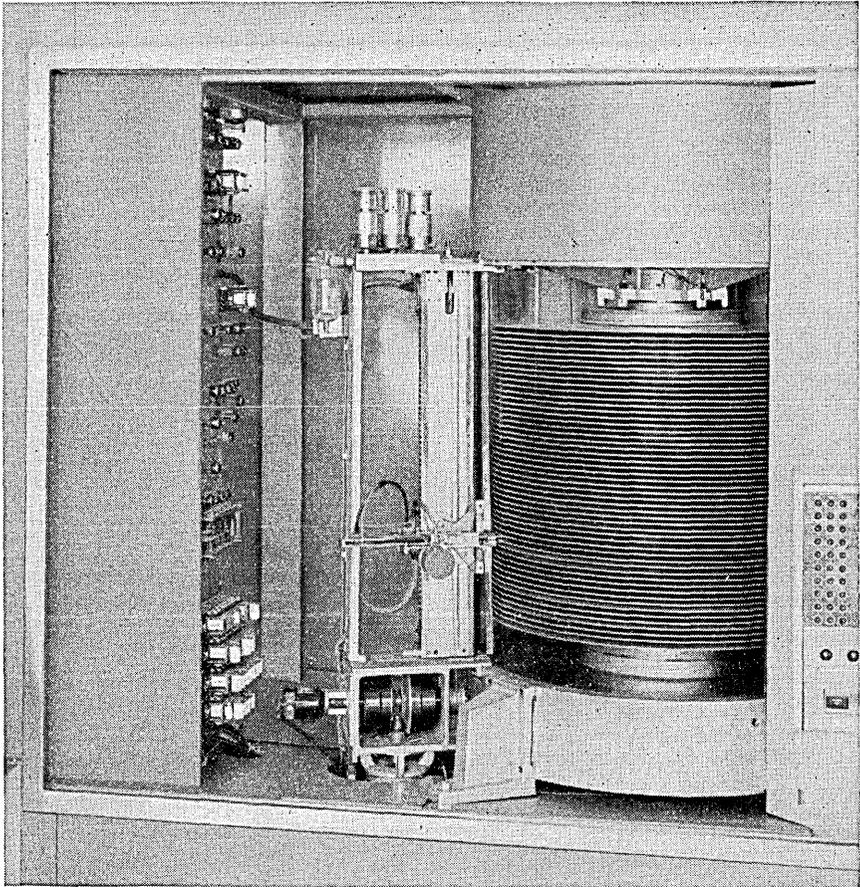


FIG. 8. IBM magnetic disk file. Disk diameter is 24 in.

**Reliability.** The reliability of a carefully designed magnetic drum storage unit can be good, and is limited by the circuitry rather than the drum itself. Adequate margins in the reading circuits are essential to accommodate variations from head to head, modulation of the read signal caused by drum runout, and variations caused by temperature, vibration, etc. Drums have been operated continuously for several years without mechanical failure, or signs of wear. Malfunction of the circuits associated with the drum can be expected, and routine preventive maintenance is advisable. Intermittent failures, especially those common to electron tubes, are most troublesome. (See the section on circuit design for a discussion of circuit reliability.)

The Whirlwind I auxiliary drum unit has a storage capacity of 393,216

bits on a drum 8.5 inches in diameter and 15 inches long. The drum is in a temperature-controlled enclosure which protects it from dust and dirt. The floor area required is approximately 8 square feet, with another 16 square feet for the associated circuits, which include 780 electron tubes. Total power consumption is approximately 4000 watts. The average time between malfunctions during scheduled operating periods is 350 hours, and 3 hours of maintenance time are required per week.

**Magnetic Disk Storage.** An experimental storage unit of unusual design, the IBM type 305 magnetic disk storage unit (Ref. 11) is shown in Fig. 8. Fifty disks, 24 inches in diameter, rotate at 1200 revolutions per minute. The disks are coated with an iron oxide dispersion on both sides; the heads are air-floated. The storage capacity is 5 million characters, with 7 binary cells plus a space per character. The NRZI technique is used. The reading circuits are self-strobing. Multiple independent access can be provided. Maximum access time for one pair of pneumatically positioned heads is 0.7 second.

### 3. MAGNETIC CORE STORAGE

**Principle of Operation.** In a magnetic core storage unit, each bit is stored in the magnetic field of a small, ring-shaped magnetic core. The most important storage techniques make use of coincident current, in which two aspects of the core's rectangular flux current characteristic are utilized: (1) flux remanence for storage and (2) extreme nonlinearity for selection (Ref. 12).

**Two-Coordinate-Read, Three-Coordinate-Write System.** Figure 9 shows the flux current loop for a typical core. The remanent flux points are arbitrarily designated as 0 and 1. The loop is sufficiently nonlinear so that the application of a current  $I_m/2$  switches negligible flux, whereas the full current  $I_m$  is capable of switching the core to the opposite remanent state. Figure 10 illustrates how this nonlinearity may be used to select one core out of many by the coincidence of two half-currents in a two-coordinate scheme. The extension to three coordinates may be accomplished by juxtaposing planes like those schematically illustrated in Fig. 10 and connecting the respective  $x$  and  $y$  selection lines in series to obtain a three-dimensional array, as indicated in Fig. 11. Each plane in the array can then be threaded with its own  $z$  winding linking all the cores in that plane. This third coordinate can then be used for selective excitation during the write process (Ref. 13).

**Read.** The application of a half-current to the coordinate  $x_i$  (Fig. 11) results in the half excitation of a selection "plane" through the array. The same is true for the coordinate  $y_i$  and the result is full-current excitation of the line of cores along the  $z$  axis at the intersection of the two

selection "planes." A parallel type storage unit might well resemble Fig. 11, and the selected line of cores represent the selected storage loca-

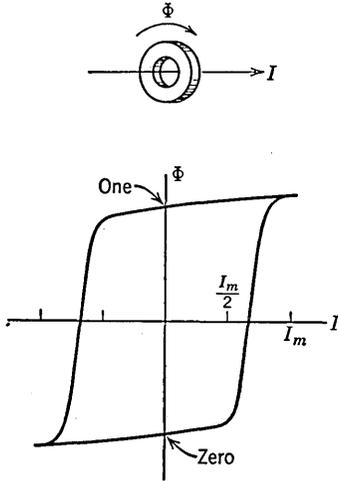


FIG. 9. Flux current characteristic of a typical core.

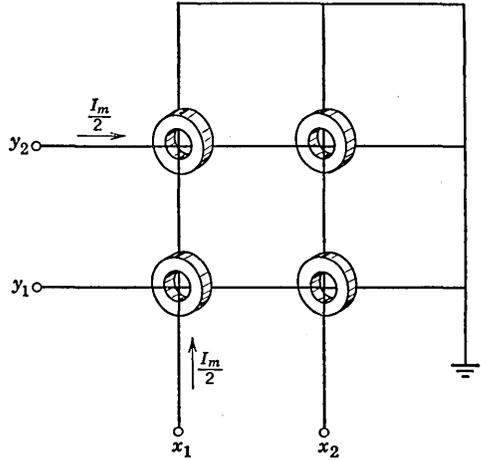


FIG. 10. A two-coordinate coincident current array.

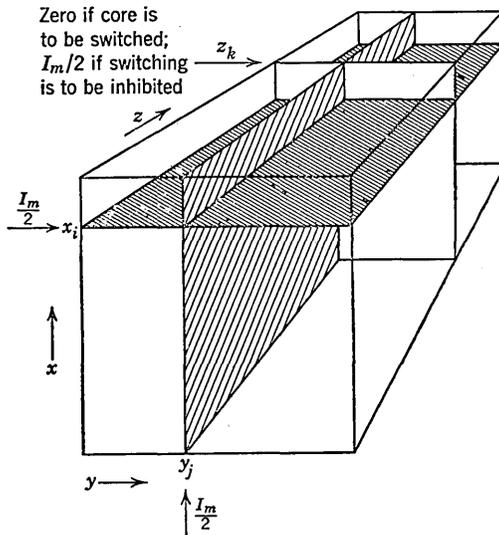


FIG. 11. Selection in a three-coordinate array.

tion. Each plane, or place, has its own read winding threaded through every core to bring out the signal representing the stored digit. If a core

holds a 1, a voltage pulse will occur when the core switches. If it holds a 0, there is negligible output. This part of the read operation is destructive, and the word must be rewritten. Note that no excitation is required for any of the  $z$  selection lines during the read process.

**Write.** For the rewrite (or write) part of the cycle, the selection technique is basically the same. The half-currents on the  $x$  and  $y$  selection planes, in the write polarity, would result in the writing of 1's into all the cores of the selected storage location, except that this writing is controllable in each place by the use of the  $z$  winding on which may be applied a half-current of a polarity opposite to the write currents in the  $x$  plane and  $y$  plane. The presence of this inhibiting current in any  $z$  plane, or place, during the write operation leaves a 0; absence of the inhibiting current, leaves a 1.

**Multicoordinate Selection Principles.** The system described in the preceding paragraphs employs a two-coordinate read and a three-coordinate write. This system is represented in tabular form in Table 3a, where  $S$  and  $U$  denote selecting and unselecting coordinate excitations. The ratio,  $R$ , of the excitation of the selected core to the absolute value of the maximum excitation of an unselected core, is 2 : 1.

TABLE 3. SELECTION SYSTEM

	Excitation	$X$	$Y$	$Z$
(a) 2-to-1 Read, 2-to-1 Write				
Read	$S$	$\frac{1}{2}$	$\frac{1}{2}$	0
	$U$	0	0	0
Write	$S$	$-\frac{1}{2}$	$-\frac{1}{2}$	0
	$U$	0	0	$\frac{1}{2}$
(b) 3-to-1 Read, 2-to-1 Write				
Read	$S$	$\frac{2}{3}$	$\frac{1}{3}$	0
	$U$	0	$-\frac{1}{3}$	0
Write	$S$	$-\frac{1}{2}$	$-\frac{1}{2}$	0
	$U$	0	0	$\frac{1}{2}$

In general, in an  $m$  coordinate system (Ref. 14), each core would be at the unique intersection of  $m$  wires, each wire from a different coordinate. To determine the maximum selection ratio,  $R_{\max}$ , that can be obtained for any  $m$ , assume a difference excitation of one unit between selected and unselected lines in each coordinate. The selected core is selected in all  $m$  coordinates. All others are unselected in from 1 to  $m$  coordinates. If  $t$  is the maximum positive excitation which results from unselecting in only one coordinate, then to obtain  $R_{\max}$ , make

$$t - (m - 1) = -t.$$

That is, unselecting the remaining  $m - 1$  coordinates should leave an equal but opposite excitation. Since the selected core receives an excitation  $t + 1$ ,

$$R_{\max} = \frac{t + 1}{t}.$$

Substituting for  $t$ ,

$$R_{\max} = \frac{m + 1}{m - 1}.$$

This means that for a two-coordinate read cycle,  $R_{\max}$  is 3 : 1, an improvement over the system of Table 3a, which has an  $R$  of 2 : 1. The selection system for the improved selection ratio is shown in Table 3b. The write, being three-coordinate, has  $R_{\max} = 2 : 1$ , so no overall improvement can be made.

**Redundant Coordinates.** In the systems just described, the selection of one line in each coordinate is sufficient to select the cell at an intersection. The addition of redundant coordinates is possible by the introduction of selection planes having different slopes. Each line of an added coordinate intersects a line of one of the original coordinates at only one cell. The excitation of a line in each of these redundant coordinates subjects the selected element to an additional unit excitation without adding any excitation to any other elements previously selected. Using  $P$  redundant coordinates improves the selection ratio to

$$R_{\max} = \frac{m + 1 + 2P}{m - 1}.$$

At the same time the number of input drivers and wires per core go up in proportion to the number of redundant coordinates (Ref. 15).

**Core Switches.** Multicoordinate core switches are commonly used for switching the high-current pulses needed to drive selection lines of core memories. The symmetrical read-write excitations required are well suited to the alternating flux output from switch cores (see Chap. 15).

**Magnetic Core Storage.** A block diagram of a parallel magnetic core storage for 4096, 16-bit words (65,536 bits), is shown in Fig. 12. This unit has a cycle time of 5 microseconds and an access time of 2 microseconds. A seventeenth plane is used to store parity.

The core array measures 10 inches by 10 inches by 20 inches; a floor area of 29 square feet is required for the entire unit, mainly for the circuits, which include 652 electron tubes. Total power consumption is 8800 watts. The average time between malfunctions during scheduled operating periods is 623 hours, and 4 hours of maintenance time are required per week.

The operation is as follows:

1. The read-rewrite cycle begins when the address is transferred to the address register and storage control has been informed, say, that a

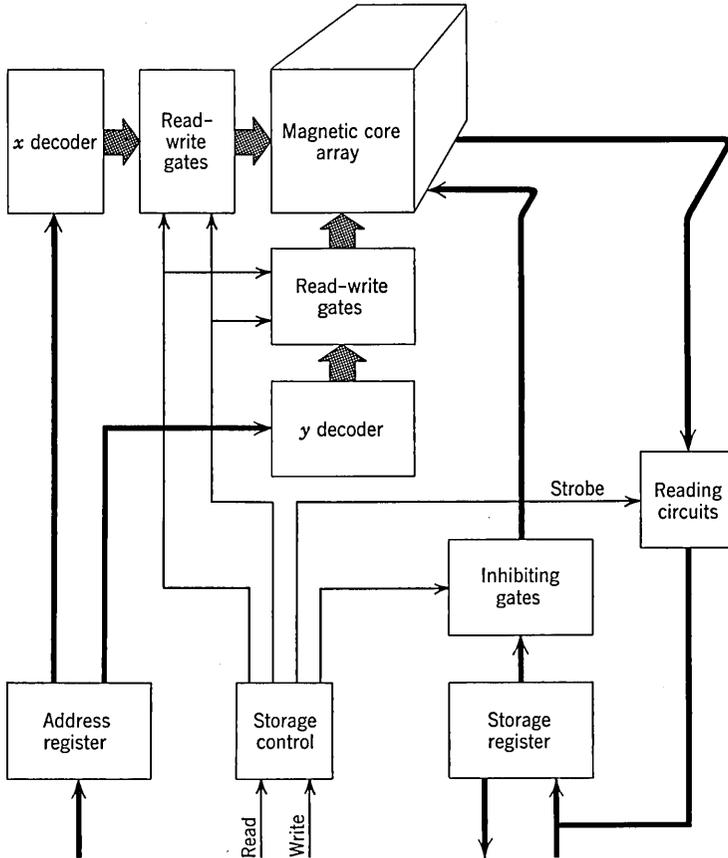


FIG. 12. Simplified block diagram of a magnetic core storage unit. Width of connecting lines indicates roughly the number of parallel conductors.

read operation is to take place. Six of the flip-flops of the address register control the  $x$  decoder and the other six control the  $y$  decoder.

2. The decoders translate from the base 2 to the base 64 and set the read-write gates.

3. After the decoders have had time to set the read-write gates, storage control pulses the read-write gates, and causes the selected  $x$  and  $y$  lines to be pulsed in the read direction.

4. At the proper instant, storage control then sends a timing pulse to all 17 reading circuits, and samples the output voltage of each plane.

5. For each plane in which a 1 had been stored, the timing pulse will be gated to set the corresponding place in the storage register. The word is thus transferred from the selected storage location to the storage register.

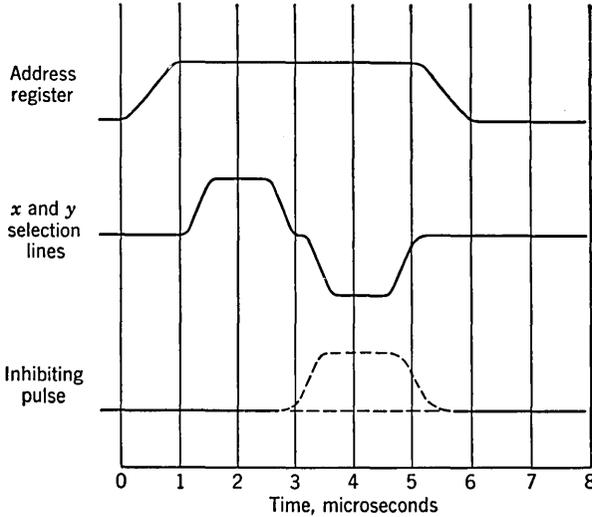


FIG. 13. Timing diagram.

6. If the information is to be retained, the next step must be to rewrite the word in the same storage location. After the inhibiting gates have been set, storage control pulses the read-write gates again, and the inhibiting gates, and causes the  $x$  and  $y$  lines to be pulsed in the write direction. However, the inhibiting pulse, which pulses all  $z$  planes in which 0's are to be rewritten, overlaps the write pulses in the  $x$  and  $y$  lines. The timing is illustrated in Fig. 13.

If a write operation had been called for, the word to be stored would have been transferred to the storage register at the beginning of the cycle. The cycle is then the same except that no timing pulse is sent to the reading circuits.

**Selection Circuits.** The read-write and address selection circuits of a magnetic core storage unit are illustrated in Fig. 14 (Ref. 16). When selected, one of the 64 output lines of the crystal diode matrix is pulled down to a voltage level which cuts off the first 5965 section, the resulting rise in plate voltage is coupled through the cathode follower and raises both grids of the 5998 array driver. No current flows, however, because the cathodes of the 5998 are normally held at a high voltage level by the read and write gate generators. At the appropriate time a start-read

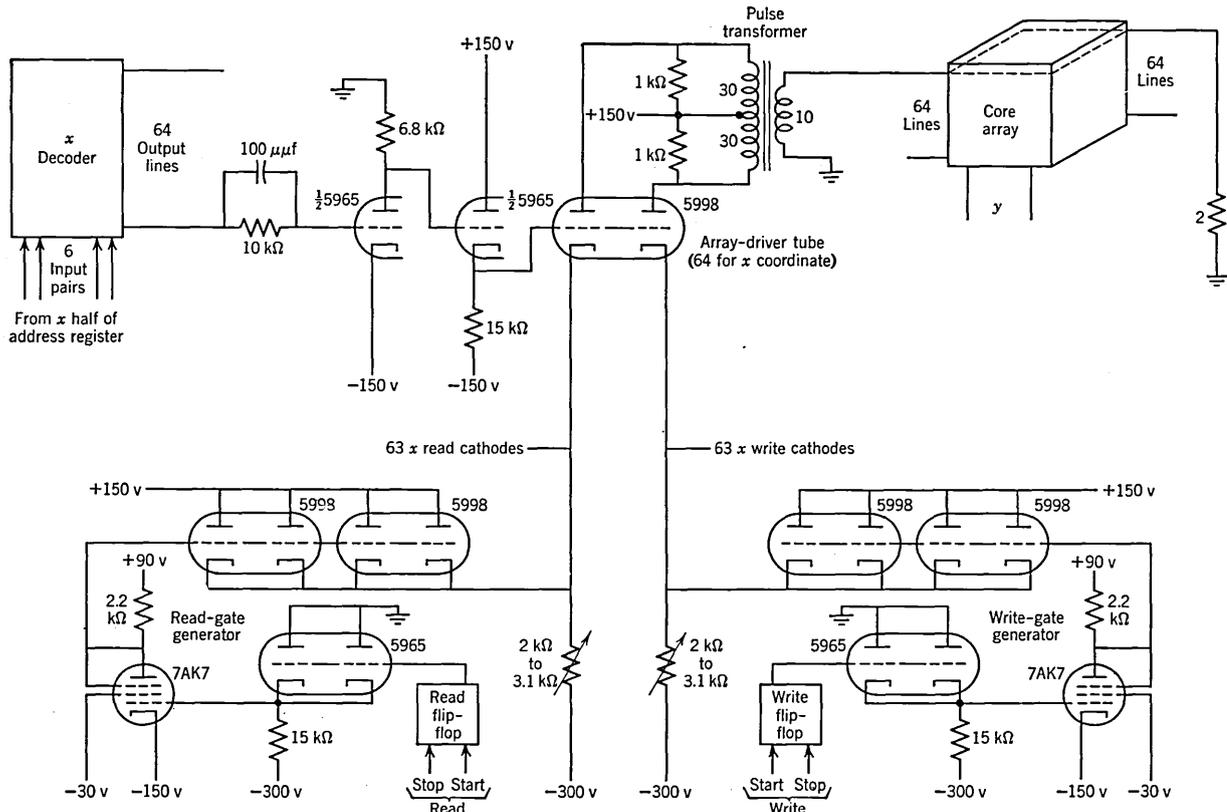


FIG. 14. Read-write and address-selection circuits of MIT magnetic core storage unit.



strobe pulse on its control grid. The output of this gate tube, a 0.1-microsecond pulse for a 1 and nothing for a 0, goes to the set input of the corresponding flip-flop in the storage register. Figure 16 shows waveforms at the input to the reading amplifier and at the suppressor grid of the gate tube.

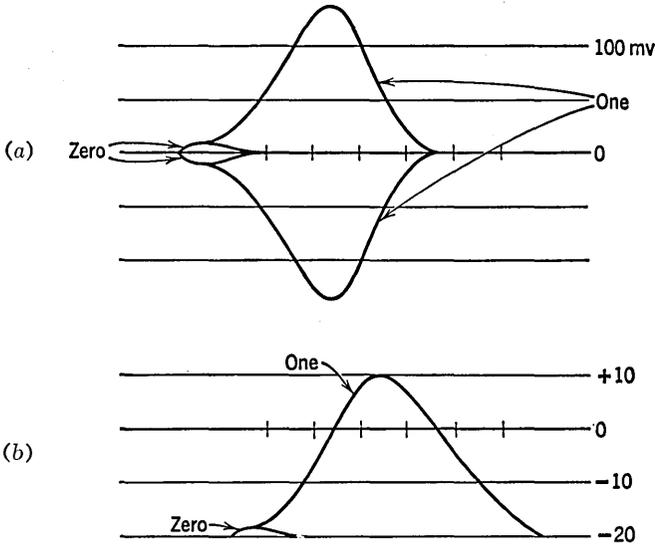


FIG. 16. Voltages at (a) input to reading amplifier and (b) suppressor grid of the gate tube.

**Parity Check.** An additional plane is included in the array described, to provide for a parity check. When a word is to be stored, a parity check bit is determined while the word is in the storage register, and the seventeenth flip-flop of the storage register is set correspondingly. When a word is read, the parity is checked immediately after the word has been transferred from the storage location to the storage register.

**Fabrication Details.** Figure 17 shows the wiring details of a 16-by-16 plane. The cores are spaced on 0.1-inch centers. Four wires pass through each core, an  $x$  wire, a  $y$  wire, a  $z$  wire, and a read wire. When the planes are stacked in an array as illustrated in Fig. 18, the corresponding  $x$  and  $y$  terminals of all the planes are connected by means of short jumpers so that two terminals at opposite ends of the array are obtained for each of the  $x$  selection lines and the  $y$  selection lines (Ref. 17).

**Core Design.** Very satisfactory ferrite cores have been developed (Ref. 18) having diameters as small as can be handled, tested, and wired into arrays conveniently. A high squareness ratio, as defined in Fig. 19,

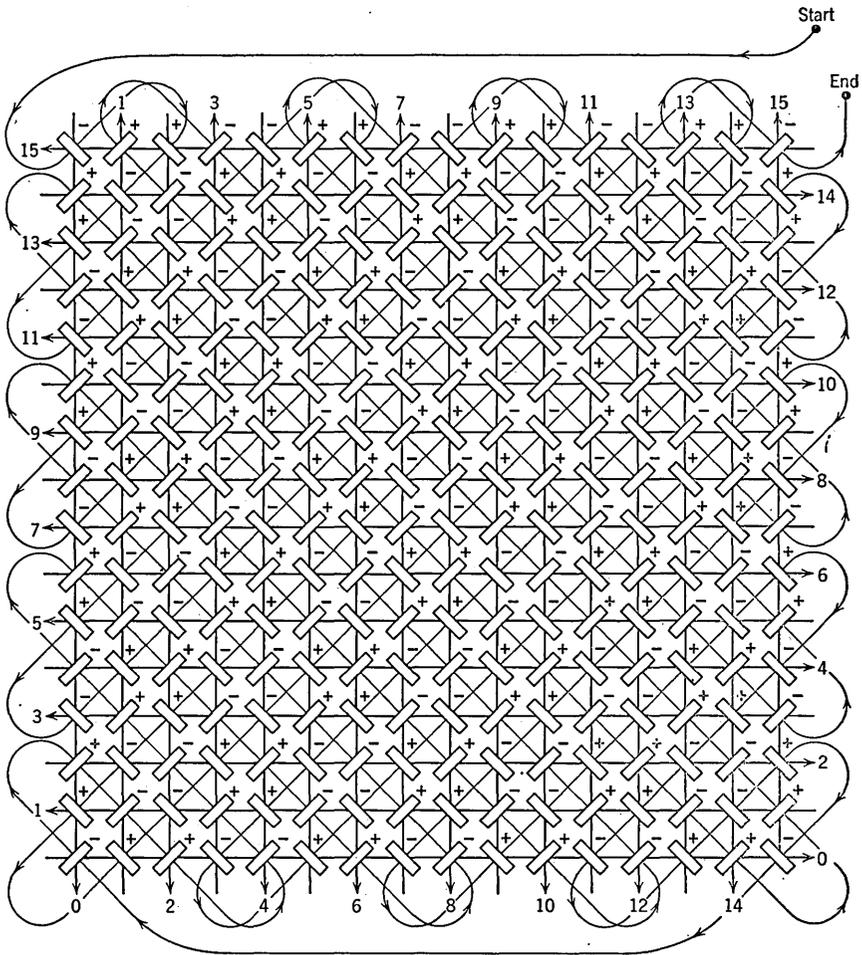


FIG. 17. Sixteen by sixteen plane wiring details.

is essential. The dimensions and characteristics of typical commercial cores are listed in Table 4. The following general statements apply.

1. Cores with higher coercive forces have shorter switching times in coincident current storage units. This is because switching time is inversely proportional to the absolute difference between the applied field and a threshold field which is very nearly the same as the coercive force.

2. For a given core material, the current required for operation of an array is directly proportional to the mean diameter of the core.

3. The energy required to switch a typical core is of the order of one erg.

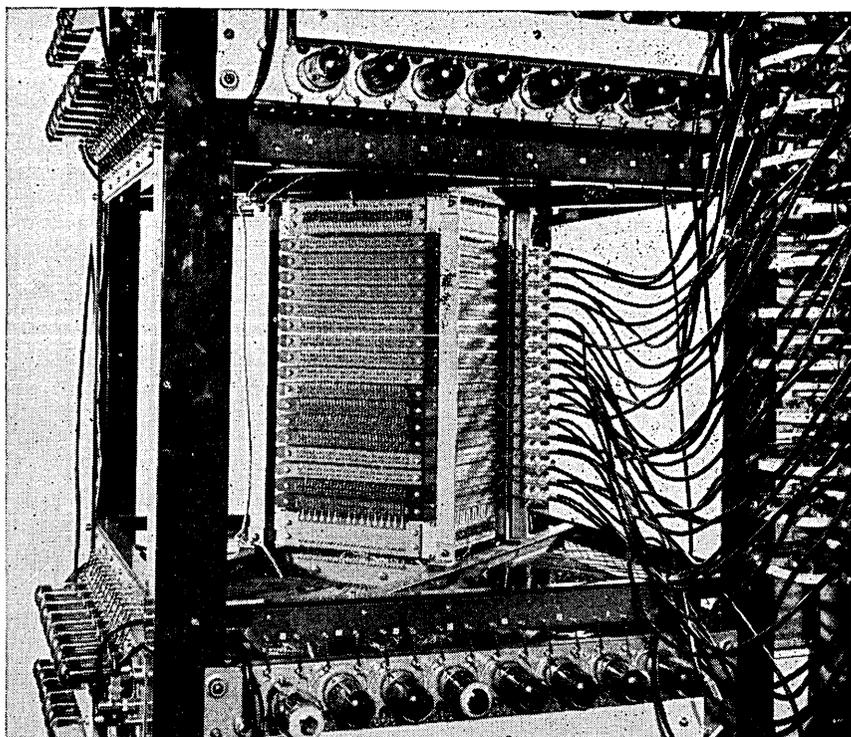


FIG. 18. A 4096-word magnetic core array. Plane dimensions are 9 by 9 in.

**Detection Problem.** The sensing problem in a magnetic core array is primarily due to undesired outputs from half-selected cores on the single  $x$  and single  $y$  lines which are excited during the readout time (Ref. 19). If all cores had equal half-select outputs, a sense winding which linked halves of each row and each column in opposite directions,

TABLE 4. CHARACTERISTICS OF SOME FERRITE CORES FOR MAGNETIC CORE STORAGE UNITS

Type	General Ceramics Corporation	
	Ferramic S1 F-394	Ferramic S3 F-394
Saturation flux density, gauss	1780	2000
Retentivity, gauss	1590	1920
Coercivity, oersteds	1.5	0.65
Curie temperature, °C	300	300
Dimensions, inches	0.080 O.D., 0.050 I.D., 0.025 thick	
Squareness ratio, $R_s$	0.8	0.95
Optimum current, $I_m$ , amperes	0.82	0.35
Switching time, microseconds	1	5

would give a net half-select output of approximately zero for excitations on any row and column. (One core on the row and one on the column remain uncanceled.)

For the canceling sense winding of Fig. 17, + and - denote positive or negative core coupling to the sense wire. It is obvious that any "checkerboard" winding will fulfill the core cancellation requirements.

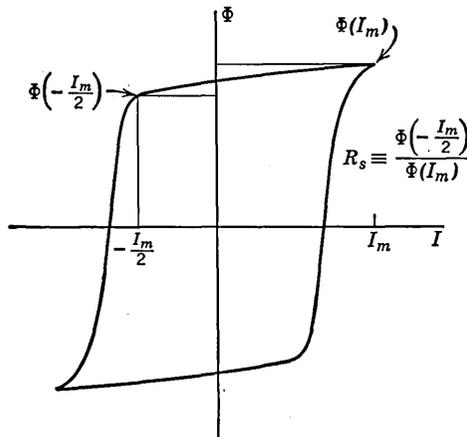


Fig. 19. Definition of the squareness ratio,  $R_s$ .

In practice, the half-select signal from a core is a function of both the information it holds (1 or 0), and the previous half-select pulse which it has received. It is possible to have all the largest half-select outputs located on the positive half of the sense winding and all of the smallest on the negative half, or vice versa. The result then is that the core output voltage for a sense winding linking an  $n$ -by- $n$  array will be:

$$V_t = \pm[V_s - 2V_{hs} \pm (n - 2)V_\delta],$$

where  $V_s$  is the magnitude of the voltage from the selected core,  $V_{hs}$  is the magnitude of the voltage from a half-excited core,  $V_\delta$  is the uncanceled voltage from a pair of half-excited cores of opposite polarity (sometimes called delta noise). In the worst case, then, 1's may be decreased and 0's increased by delta noise.

Since the noise component varies linearly with  $n$ , it establishes an upper limit on the size of a single sense winding. This factor also accounts for the emphasis on uniformity and the care exerted in core testing. Single sense windings are used for  $n$  as large as 64, and, by the use of time-consuming tricks, can probably be extended to 128.

Any checkerboard winding (Ref. 20) will also be free of air coupling

to the  $x$  and  $y$  lines if care is taken to keep the air-flux areas of both polarities equal. The canceling areas of the winding of Fig. 18 are

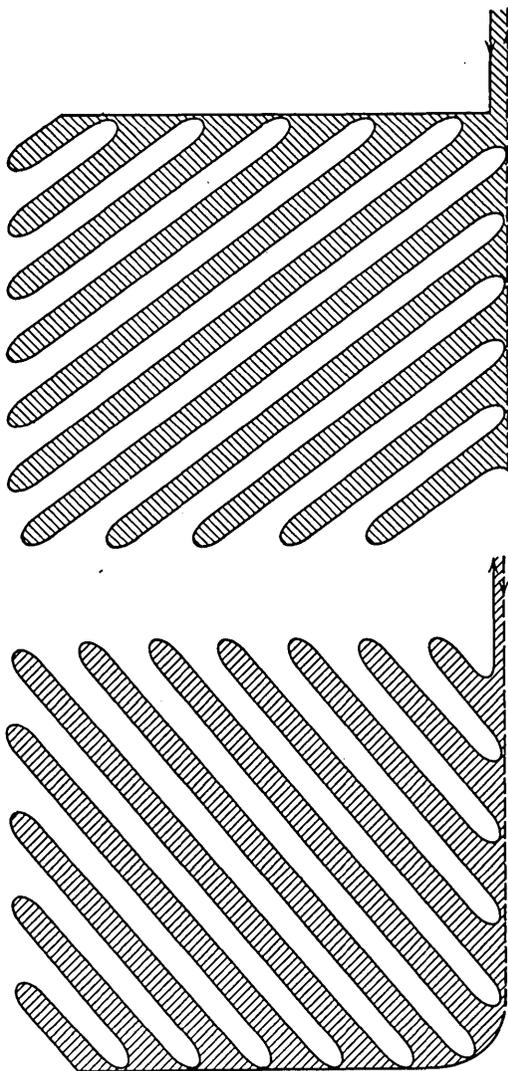


FIG. 20. Canceling air flux components of sense winding.

denoted by the diagonal dark and light areas of Fig. 20. This complicated looking winding has sometimes been used instead of simple rectangular sections because it eases insertion of the sense wire, after  $x$ ,  $y$ ,

and  $z$  are in place; by allowing it to enter perpendicular to the hole surface.

**Ambient Temperature.** Although the power dissipated by the cores in the array of a magnetic core storage unit is negligible compared with the power required by the associated circuitry, some attention must be given to the maintenance of a satisfactory ambient temperature for the array. The coercive force and other characteristics of the cores depend upon temperature, and any given system will have certain minimum and maximum limits for the ambient temperature. Limits of the order of  $\pm 10^\circ\text{F}$  are typical.

**Current Regulation.** Also, care must be taken to ensure that well regulated array-driving currents, with clean waveshapes, are obtained. Regulation of the driving current in a coincident current storage unit should keep the driving current within less than  $\pm 5$  per cent of the nominal amplitude.

**Core Stability.** The characteristics of ferrite storage cores are extremely stable. Ferrite cores will withstand considerable shock and vibration, and their characteristics will not be changed permanently by temperature cycling, provided the temperature does not rise high enough to change the structure of the ferrite. A carefully constructed array will not malfunction and will last indefinitely. However, a magnetic core storage unit as a whole can be expected to malfunction and require maintenance. The reliability of the storage unit will depend on the reliability of the associated circuitry.

**Typical Storage Units.** Characteristics of several magnetic core storage units are listed in Table 5 (Ref. 21). Core storage units employing only transistors in associated circuitry have been designed. The detection circuits for such units are ideally suited to transistors. The high-current switches required to drive lines of storage cores which appear as large inductive loads are a more serious problem. The combination of high current, fast rise time, and high peak inverse voltage represents a significant challenge which is beginning to be met successfully by transistor manufacturers.

### Other Discrete Magnetic Storage Techniques

A number of different techniques which make use of the nonlinearity and remanence of discrete magnetic cells for storage units have been developed.

**Nondestructive Readout.** Nondestructive readout of cores is possible using a number of different systems. None of these has been used in any large-scale storage but nondestructive readout techniques can be applied to smaller storage units.

TABLE 5. SOME OPERATING MAGNETIC CORE STORAGE UNITS

	M.I.T. Lincoln Laboratory	Sperry-Rand Corp. UNIVAC Division	International Telemeter Corp.	IBM IBM 705
Type	TX-2	ERA 1103A	Johnniac	
Storage capacity, bits	2,490,368	147,456	163,840	280,000
Read-write cycle time, microseconds	6	12	15	9
Number of words	65,536	4,096	4,096	8,000
Word length, bits	38	36	40	35
Tube count	604 + 1406 transistors	469	1,200	711
Array structure and circuits	Thirty-eight 256 by 256 planes. Tube-driven core switches on each co- ordinate. Sense wind- ing in four sections.	Thirty-six 64 by 64 planes. Tube-driven diode transformer matrices drive both <i>x</i> and <i>y</i> lines.	Forty 32 by 128 planes. Separate 128 position core switches for <i>y</i> in each plane. 32 tube- driven <i>x</i> lines in series. No <i>z</i> drivers.	Thirty-five 100 by 80 planes. Core switches 8 by 10 and 10 by 10 drive <i>x</i> and <i>y</i> lines respectively.

One system (Ref. 22) uses a transverse field excitation obtained by running current through the wraps of tape cores or by using an auxiliary core at right angles to the main core. It is possible to obtain small reversible rotational outputs which will be of opposite polarity for 1's and for 0's.

Another system (Ref. 23) uses distinct r-f excitations on  $x$  and  $y$  coordinates. The core nonlinearity produces sum and difference frequencies due to the second-degree term in the power series expansion for the hysteresis loop at remanence. The coefficient of this term is of opposite sign at either remanent state so that there is 180 degree phase shift in the beat frequency output for the two remanent states. The 1 and 0 outputs are then identified by tuning to the beat frequency and detecting the phase.

**Thin Magnetic Films.** Thin magnetic films of Permalloy deposited by vacuum evaporation or by electrochemical deposition can be used as storage cells in a fashion very similar to magnetic toroids (Ref. 24). These films, when put down in a strong magnetic field, exhibit an easy axis of magnetization in the direction of the field. The hysteresis loop of such materials is extremely square for circular spots of about  $\frac{1}{8}$  inch diameter and about 1000 Å thick.

The distinguishing property of these films is that magnetization reversal can take place by rotation, an inherently fast process, rather than by the domain wall motion which is present in bulk material and relatively slow (Ref. 25). The rotational switching is enhanced by the presence of a field perpendicular to the easy axis. The important characteristics of these film storage cells are:

1. Rotational switching allows fast switching at relatively low fields.
2. The amount of material, and hence flux switched, is extremely small, about three orders of magnitude less than small toroids.
3. Many cells can be fabricated at a single time and need not be handled individually.
4. Power dissipation at very high speeds is no great problem because of the small amount of material and large surface-to-volume ratio.
5. Flux return paths are through air; films must be shielded from outside fields, such as the earth's.
6. Input and output wires are placed in proximity to spots and do not thread holes.

The principal problems are in obtaining spot-to-spot uniformity and in eliminating the formation of domain walls below the rotational threshold.

Many of the multidimensional selection principles used in core storage are directly applicable to films, with many additional variations possible because of the strong effect of transverse fields on mode of operation.

Demagnetizing factors which tend to cause loop squareness to deteriorate increase with thickness and decrease with spot diameter. Since the total signal out is proportional to the product of thickness and diameter, packing density and output signal may be traded for one another. Most work has been done with spots  $\frac{1}{8}$  inch in diameter in the range from 500 A to 3000 A. These films, when switched in 0.1 microsecond, give outputs of 1 to 10 millivolts.

Figure 21 shows hysteresis loops for films in the easy and transverse directions.

**Ferrite Plates.** The use of sheets of ferrite prepared with holes eliminates the handling and fabrication of individual cores (Ref. 26). These units operate the same as magnetic toroids. The main disadvantages are the difficulty of obtaining perfect plates and the slight degeneration in performance due to interference between adjacent holes.

**Multiperture Cores.** Cores having more than one hole have been used experimentally to provide higher-speed storage (Ref. 27). Here, by separating selection and storage functions, it has been possible to obtain extremely fast switching at the cost of higher current drive. Very high back voltages and core dissipation at such high-speed operation are an important drawback to such systems.

#### 4. OTHER STORAGE TECHNIQUES

##### Acoustic Delay Line Storage

Acoustic delay line storage was used in many of the systems which followed the ENIAC, notably in the EDSAC, SEAC, RAYDAC, and the first UNIVAC's. A simplified block diagram of an acoustic delay line storage unit is shown in Fig. 22. Information is stored as a sequence of bits, pulses representing 1's and the absence of pulses representing 0's (Ref. 28). The driver causes each pulse to apply a fractional microsecond burst of r-f power to a transducer at the transmitting end of the

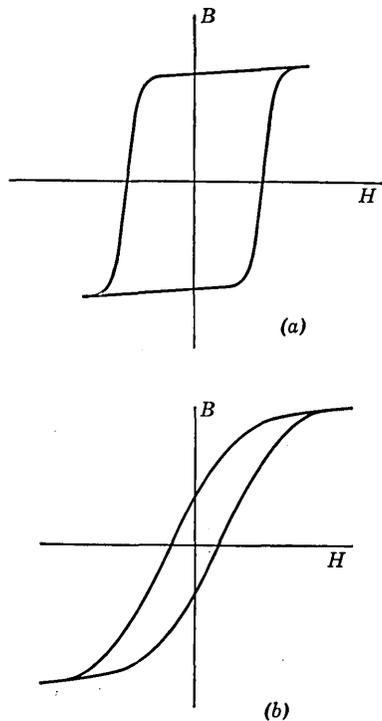


FIG. 21.  $B$ - $H$  characteristics for typical Permalloy films in (a) easy direction and (b) transverse direction.

line. After a time equal to the total delay of the line, the acoustic pulse strikes the receiving transducer and produces a voltage which can be detected and amplified. The amplified pulse is then gated and reshaped by a timing pulse and reentered into the delay line. Stored information is thus circulated indefinitely, and can be read at times when it is being reentered. To write, the clear gate is closed, and the new sequence of pulses is sent to the driver by way of the write gate. Many words are usually stored in a single line. In the UNIVAC, ten 91-pulse words (including a 7-pulse space between words) are circulated in a mercury

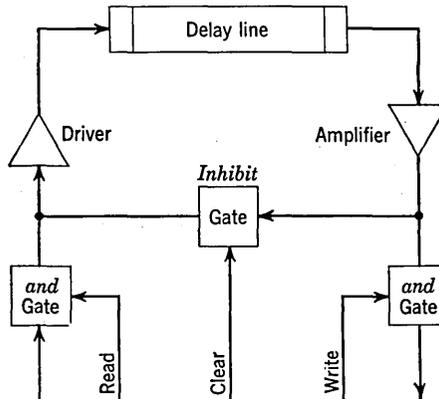


Fig. 22. Simplified diagram of a delay line storage unit.

tank with a total delay time of 404 microseconds. A tank actually accommodates 18 channels, each with its own circulating amplifier and driver. Addressing the storage unit requires both channel and time selection.

**Mercury Delay Lines.** Mercury delay lines have been most common and have the following characteristics.

1. Quartz crystal transducers are used, resonant at the radio frequency (5 to 30 megacycles) of the driver.
2. The delay in mercury is 17.5 microseconds per inch, and several pulses can be stored per microsecond.
3. The velocity is dependent on temperature, and necessitates precise temperature control.
4. The amount of circuitry per channel is relatively large, considering the number of bits which can be stored.
5. Larger channels, which improve the ratio of stored bits to circuitry, have the disadvantage of longer access time.

Improvements in magnetic core storage have made acoustic delay line storage less desirable for general use.

**Other Delay Lines.** Delay line storage units using magnetostrictive or piezoelectric delay lines have had limited application. In the case of magnetostrictive delay lines, pulses can be picked up or inserted at any point along a magnetic rod. Such features lend themselves to special applications.

### **Electrostatic, Ferroelectric, and Cryogenic Film Storage**

**Storage Tubes.** Before magnetic core storage was developed, electrostatic storage provided the shortest access times obtainable with a storage capacity of more than a few hundred words. Of the several types used, the type attributed to F. C. Williams has been most common (Ref. 31). Bits are stored on the phosphor screen of a cathode ray tube of conventional design, and the electron beam is used to write a bit at any one of, say, 1024 locations on the screen of a single tube. Reading is accomplished by returning the beam to the same location and observing the signal produced at a metal backplate which covers the outside face of the cathode ray tube screen. Access times of the order of 10 microseconds can be obtained.

In a typical computer (Ref. 32), a parallel storage unit employs 72 cathode ray tubes, two in each place of a 36-bit word. Storing 1024 bits in each tube gives a storage capacity of 2048 words. The operation is as follows.

1. All the electron beams are deflected simultaneously by  $x$  and  $y$  decoders controlled by the address register.

2. After the beam deflection voltages have stabilized, the beam is unblanked for 1 microsecond, and the initial polarity of the read signal determined.

3. If it is positive, indicating that a dash (or 1) had been stored, the beam is turned on again, this time for nearly 2 microseconds, and deflected slightly to rewrite a dash.

4. If the initial polarity of the read signal is negative, indicating that a dot (or 0) had been stored, no further unblanking is required during the cycle; the dot is effectively rewritten during the read operation.

5. The stored information also requires periodic regeneration, even at those locations to which no reference is made. Three regeneration cycles are required for each instruction executed. Two storage locations can be regenerated during each 12-microsecond regeneration cycle. Regeneration is interleaved with other operations; regeneration requirements are maintained by a separate regeneration control unit. Storage locations are scattered so that, by maintaining a read-around ratio greater than 400, the programmer is not concerned with read-around ratio. (This ratio is the number of references to adjacent storage locations required to change the storage location from a dot to a dash.)

Many other electrostatic storage systems have been used (Ref. 33). All have required special tubes, with the possible exception of Williams' technique.

Electrostatic storage is marginal in operation, requiring excessive maintenance and subject to frequent malfunction. By 1955, magnetic core storage had replaced electrostatic storage in new designs.

**Capacitor-Diode Storage.** Information can be stored in capacitors. With high back resistance diodes used for switching, regeneration require-

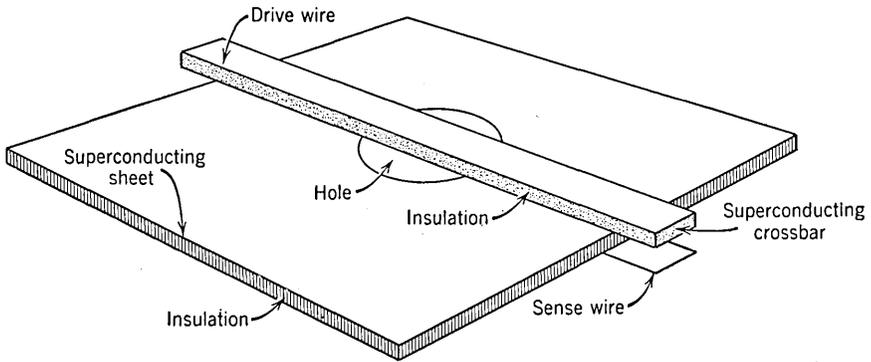


FIG. 23. Superconducting storage cell.

ments may not be severe (Ref. 36). An experimental storage unit at the National Bureau of Standards employs one capacitor and two diodes per bit and has a cycle time of 3 microseconds.

**Ferroelectric Storage.** Ferroelectric materials such as barium titanate can have square hysteresis loops. Such materials have been used in experimental storage units, whose operation is the dual of magnetic core storage (Ref. 37). The principal disadvantages of this storage medium are the high energy required to reverse the remanent charge, the disastrous heating which occurs under continuous operation, and an instability which results in deterioration of characteristics with time. However, the development of new materials like triglycine sulphate has greatly improved future prospects.

**Cryogenic Films.** Single storage cells have been constructed that use superconducting thin films of lead at liquid helium temperature (Ref. 38). These operate on the principle that flux lines cannot cut the ring of zero resistance material that surrounds them. The application of magnetic fields above a certain threshold level forces the material into the normal state, and allows flux lines to penetrate and become trapped when the material goes superconducting again. Figure 23 shows the orientation of a drive wire, superconducting cell, and sense wire. The cell consists of a circular hole in a superconducting sheet having a super-

conducting crossbar. Binary information is denoted by the direction of circulation of flux lines around the crossbar and through the hole areas on either side. The principal advantages of this storage medium are the sharp switching threshold, low drive fields, the near perfect shielding between drive and sense circuits provided by the superconducting sheet, and the possibility of very high switching speed. The principal disadvantages are difficulties in fabrication, which requires successive evaporations of lead and insulating layers, and the problems associated with a liquid helium environment.

**The Twistor.** Experiments (Ref. 39) have been performed on magnetic wire of various Ni-Fe compositions in which helical flux patterns are used for information storage. The application of tension to the ends of the wire introduces a helical strain anisotropy and by using the magnetic wire itself for the sense winding the circular flux component is detected giving rather large signals even for very thin wire. Many bits can be stored along a continuous piece of wire. Principal problems are in maintaining uniform tension, in obtaining high bit densities without interaction, and in developing thin enough wires to provide a sufficiently low Sw.

**The Parametron.** Information storage is possible with two phases of oscillation of a resonant circuit (Ref. 40). If the  $L$  or  $C$  of a resonant circuit is modulated at a pump frequency  $\omega_p$  which is twice its resonant frequency, a sustained oscillation at resonance will occur if it is excited above its threshold level. This oscillation will be locked to the pump frequency, but it can be of either phase. Work is being done with both the voltage-variable capacity of a  $p$ - $n$  junction and the current-controlled inductance of nonlinear magnetic materials.

## REFERENCES

1. R. K. Richards, *Arithmetic Operations in Digital Computers*, Van Nostrand, Princeton, N. J., 1955.
2. R. K. Richards, *Digital Computer Components and Circuits*, Van Nostrand, Princeton, N. J., 1958.
3. J. P. Eckert, Jr., A survey of digital computer memory systems, *Proc. I.R.E.*, **41** (10), 1391-1406 (1953).
4. A. A. Cohen, Magnetic drum storage for digital information processing systems, *Mathematical Tables and Other Aids to Computation*, **IV**, 31-39 (Jan. 1950).
5. B. F. C. Cooper, A magnetic drum digital storage system, *Proc. I.R.E.* (Australia), **14**, 169-177 (1953).
6. M. H. Weik, A survey of magnetic drum memory systems for electronic computers, *Ballistic Research Laboratories Rept. 819*, Aberdeen Proving Ground, Md., August 1954.
7. D. F. Brower, A "one-turn" magnetic reading and recording head for computer use, *I.R.E. Convention Record*, Pt. 4, *Computers, Information Theory, Automatic Control*, pp. 95-100, 1955.

8. E. S. Hughes, Jr., The IBM magnetic drum calculator type 650, engineering and design considerations, *Proc. Western Computer Conference*, pp. 140-154, Los Angeles, Calif., Feb. 11, 12, 1954.
9. H. W. Fuller, P. A. Husman, and R. C. Kelner, Techniques for increasing storage density of magnetic drum systems, *Proc. Eastern Joint Computer Conference*, pp. 16-21, Philadelphia, Pa., Dec. 1954.
10. M. K. Taylor, A small high-speed magnetic drum, *Computers and Automation*, **4**, 18-19 (1955).
11. T. Noyes and W. E. Dickinson, Engineering design of a magnetic-disk, random-access memory, *Proc. Western Joint Computer Conference*, 42-44, San Francisco, Calif., Feb. 7-9, 1956.
12. J. W. Forrester, Digital information storage in three dimensions using magnetic cores, *J. Appl. Phys.*, **22**, 44-48 (1951).
13. W. N. Papian, A coincident-current magnetic memory cell for the storage of digital information, *Proc. I.R.E.*, **40**, 475-478 (1952).
14. R. R. Everett, Selection systems for magnetic core storage, *Engineering Note E-413*, p. 13, Project Whirlwind, Servomechanisms Laboratory, M.I.T., Cambridge, Mass., Aug. 1951.
15. R. C. Minnick and R. L. Ashenhurst, Multiple-coincidence magnetic storage systems, *J. Appl. Phys.*, **26**, 575-579 (1955).
16. W. N. Papian, New ferrite-core memory uses pulse transformers, *Electronics*, **28**, 194-197 (March 1955).
17. E. A. Guditz and L. B. Smith, Vacuum and vibration speed assembly of core memory planes, *Electronics*, **29** [2], 214-228 (1956).
18. D. R. Brown and E. Albers-Schoenberg, Ferrites speed digital computers, *Electronics*, **26** [4], 146-149 (1953).
19. J. R. Freeman, Pulse response of ferrite memory cores, *Proc. WESCON Computer Sessions*, 50-61, Los Angeles, Calif., Aug. 25, 1954.
20. J. I. Raffel, Sensing winding geometry and information patterns, *M.I.T. Lincoln Laboratory Rept. 6M-2919*, Lexington, Mass., July 1954.
21. M. H. Weik, A second survey of domestic electronic digital computing systems, *Ballistic Research Laboratories, Rept. 1010*, Aberdeen Proving Ground, Md., June 1957.
22. D. A. Buck and W. I. Frank, Non-destructive sensing of magnetic cores, *Trans. Am. Inst. Elec. Engrs.*, **72**, Pt. 1 (Comm. and Elec.), 822-830 (1953).
23. B. Widrow, Radio-frequency non-destructive readout for magnetic core memories, *Trans. I.R.E., Prof. Group on Electronic Computers*, **EC-3** [4], 12-15, Dec. 1954.
24. A. V. Pohm and S. M. Rubens, A compact coincident-current memory, *Proc. Eastern Joint Computer Conference*, pp. 120-124, Dec. 1956.
25. D. O. Smith, Magnetization reversal and thin films, static and dynamic behavior of thin Permalloy films, *J. Appl. Phys.*, **29**, 264-273 (1958).
26. J. A. Rajchman, Ferrite apertured plate for random-access memory, *Proc. Eastern Joint Computer Conference*, pp. 107-115, Dec. 1956.
27. W. W. Lawrence, Jr., Recent developments in very-high-speed magnetic storage techniques, *Proc. Eastern Joint Computer Conference*, pp. 101-104, Dec. 1956.
28. I. L. Auerbach, J. P. Eckert, R. F. Shaw, and C. B. Sheppard, Mercury delay line memory using a pulse rate of several megacycles, *Proc. I.R.E.*, **37**, 855-861 (1949).
29. E. A. Newman, D. O. Clayden, and M. A. Wright, Mercury-delay-line storage system of ACE pilot model electronic computer, *Proc. Inst. Elec. Engrs.*, **100**, Pt. 2 (76), 445-452 (1953).

30. H. N. Beveridge and W. W. Keith, Piezoelectric transducers for ultrasonic delay lines, *Proc. I.R.E.*, **40**, 828-835 (1952).
31. Symposium of papers on digital computers, *Proc. Inst. Elec. Engrs.*, **100**, Pt. 2 (77), 487-539 (1953).
32. S. H. Dodd, H. Klemperer, and P. Youtz, Electrostatic storage tube, *Elec. Eng.*, **69**, 990-995 (1950).
33. M. Knoll and B. Kazan, *Storage Tubes and Their Basic Principles*, Wiley, New York, 1952.
34. J. A. Rajchman, The selective electrostatic storage tube, *RCA Rev.*, **12**, 53-97 (1951).
35. J. P. Eckert, Jr., H. Lukoff, and G. Smoliar, A dynamically regenerated electrostatic memory system, *Proc. I.R.E.*, **33**, 498-510 (1950).
36. A. W. Holt, An experimental rapid access memory using diodes and capacitors, *Proc. Assoc. for Computing Machinery*, pp. 133-141, Sept. 8-10, 1952, Toronto, Canada.
37. D. A. Buck, Ferroelectrics for digital information storage and switching, *M.I.T. Digital Computer Laboratory Rept. R-212*, Cambridge, Mass., June 5, 1952.
38. J. W. Crowe, Trapped-flux superconducting memory, *IBM J. Research and Development*, **1** (4), 295-303 (1957).
39. A. H. Bobeck, A new storage element suitable for large-sized memory arrays: The twistor, *Bell System Tech. J.*, **36**, 1319-1340 (1957).
40. J. von Neumann, Non-linear capacitance or inductance switching, amplifying, and memory organs, U.S. Patent Office, Patent No. 2,815,488, Dec. 3, 1957.



# Input-Output Equipment for Digital Computers

*J. K. Brigden*

1. The Input-Output System	20-01
2. Printed Page	20-06
3. Perforated Tape	20-19
4. Punched Card Machines	20-30
5. Magnetic Tape	20-33
6. Analog-Digital Conversion Techniques	20-44
References	20-66

## I. THE INPUT-OUTPUT SYSTEM

Input-output equipment is a term generally applied to the components of a computing system which are used for the transfer of data to and from the computer and for manipulation of data external to the computer. The requirements for input-output equipment vary considerably depending on the specific application for which the computing system is intended. There are, however, three major areas of application which may be considered: (1) business, (2) scientific, and (3) automatic control.

**Business Applications.** Business computers usually require the most extensive input-output systems. Large amounts of data are frequently associated with a business problem and very often comparatively little computation is required. Thus a large percentage of computer time will be spent conducting input-output operations unless care is taken to provide an adequate input-output system. The following simplified model will serve to illustrate the problem of selecting an input-output system with the appropriate capacity. Let  $C$  be the average cost of the com-

puting system per problem (input-output equipment included); then:

$$C = (T_c + T_r)(C_c + C_r),$$

where  $T_c$  = average time per problem required for computation (determined by choice of computer and the class of problems to be solved),

$T_r$  = average time per problem required to conduct input-output operations (determined by choice of input-output equipment and the class of problems to be solved),

$C_c$  = cost of computer per hour,

$C_r$  = cost of input-output equipment per hour.

To match the input-output equipment to the computer assume that  $C_c$  has been fixed in advance. Further assume that  $C_r$  is proportional to the speed of the input-output equipment, i.e.,

$$C_r = k/T_r.$$

From this one obtains:

$$C = T_c C_c + k + \frac{k C_c}{C_r} + T_c C_r.$$

The value of  $C_r$  for which the cost per problem,  $C$ , is minimized in the above expression is given by:

$$C_r = \left( \frac{k C_c}{T_c} \right)^{1/2}.$$

Since, for a fixed  $C_r$ ,  $k$  is proportional to  $T_r$ ,  $k$  is, to some extent, a measure of the average amount of data which must be handled per problem during input-output operations. For a business problem where this amount is large and the time required for computation,  $T_c$ , is small, it is likely that most optimum use of the computer system will be obtained by using a relatively costly or elaborate input-output system.

**Scientific Applications.** Scientific problems, on the other hand, generally require considerable computation on small amounts of data. From our simple formulation above, one would expect that, for optimum use of the overall system, the input-output system should be relatively small. Considerable programming effort accompanies scientific problems, however, and in utilizing a computer for machine programming, one is again faced with a class of problems whose input-output requirements are similar to those found associated with business problems. In addition to this, scientific problems involving data reduction also have these same characteristics of large amounts of data and relatively short computation times. For these reasons information regarding data reduc-

tion requirements and the programming techniques which are to be employed should be available prior to establishing the input-output system for the scientific computer.

Some scientific computers have also been used in conjunction with analog computers to increase the overall computational capability of both. (See Chap. 30.) In these instances analog-digital conversion units are required as part of the input-output system.

**Automatic Control Applications.** Problems associated with digital control systems very often require large amounts of data to be handled by the computer. Although these data are occasionally in digital form, more often than not the requirement exists for conversion both from and to analog signals. Input from the operator is accomplished much in the same manner as with computers used in other applications. An additional problem associated with the automatic controller input-output system is that the timing of the input and output signals is frequently critical. (See Vol. 3, Chap. 8.)

**General Requirements for an Input-Output System.** In any of the three applications above, the one necessary requirement for an input-output system is modification of data from one form to another so it may be accepted by the computer during input operations or presented in usable form during output operations. A second requirement is that the computing system be utilized in a reasonably efficient manner. The most critical problem arises when large amounts of data must be handled at rapid rates. A number of approaches have been taken in meeting this problem, such as increasing the speed and number of input-output devices which are operated with the computer. Although considerable success has been achieved along these lines, the problem is not generally solved by this method alone.

**Buffer Storage.** Another approach has been to install intermediate or buffer storage between the computer and input-output devices. Utilization of this approach provides for more economic use of the computing system in two ways: (1) It allows the programmer greater freedom in executing input-output operations and in many cases permits him to conduct these operations in a more expeditious fashion. (2) Both computation and input-output operations may be carried on for the most part simultaneously.

The latter advantage allows the use of input-output equipment which is only as fast as the *average* rate at which the computer puts data out or takes data in. However, if equipment with only the minimum speed is used, it becomes necessary to provide a buffer storage of considerable capacity if something is to be gained from the first advantage. A compromise is necessary among (1) the cost of input-output equipment,

(2) cost of the buffer storage and its associated control circuitry, (3) cost of computer time which is spent conducting input-output operations, and (4) cost of constraining the programmer in executing these operations. The optimum values of all these factors is difficult if not impossible to calculate, but with sufficient judgment and experience it is possible to arrive at a configuration which is close enough to optimum for all practical purposes.

A slightly different method of accomplishing buffer action is to utilize the main computer storage as the buffer storage also. This is accomplished by establishing appropriate computer control circuitry so that the computer may carry on computation at the same time it is reading data into and out of its main storage. Although this approach does not impose the requirement for an additional independent storage, it does necessitate a considerable amount of control circuitry which is rarely warranted except under most unusual input-output requirements.

Another approach has been to utilize comparatively high-speed input-output equipment, usually magnetic tape units, for transfer of data to and from the computer and to make use of equipment external to the computer for preparing and reading the magnetic tape. This, in effect, is another method of accomplishing buffer action since the computer is free to proceed with additional computation while the magnetic tape is being processed. Here, one can also note the distinction between on-line and off-line equipment.

**On-Line Equipment.** Equipment that is used for transfer of data to and from the computer and that is directly connected to it is referred to as *on-line* equipment. When large amounts of data are involved and comparatively little computation is required, on-line equipment should be of the high-speed variety. On the other hand, if only a small portion of computer time is spent conducting input-output operations, it becomes more efficient to utilize slower equipment such as keyboards, typewriters, and printers, and thereby to avoid the expense of extensive off-line equipment. Thus nearly all the different input-output devices are applicable to on-line use, and the selection of any particular device depends on the intended use of the computing system.

**Off-Line Equipment.** Equipment which is used for manipulation of data external to the computer is referred to as *off-line* equipment. Obviously all the equipment used for on-line purposes can also be used off-line. In fact, a more flexible computing system is obtained by allowing the equipment to be used in either capacity. Aside from the readers and recorders found associated with off-line equipment one also finds various control units which are used to couple the readers and recorders

together. These control units are occasionally extremely complex and may conduct simple arithmetic operations, change format, and buffer incoming or outgoing data. Equipment which is employed solely in off-line applications is frequently referred to as *auxiliary equipment*.

**External Storage Requirements.** External storage devices which are not under computer control are an important feature of many computing systems. Although not directly related to input-output operations in the sense of communicating with personnel or equipment outside the computing system, external storage does involve communication with the computer and to some extent manipulation of data external to the computer. Combination of the input-output and external storage functions may result in more optimum use of the equipment available. Requirements for external storage should therefore be considered in establishing the input-output system.

**Recording Media.** Recording media are used extensively in practically all input-output systems and are, therefore, an important design consideration. Since the choice of a particular recording medium dictates to a large extent which type of equipment will be used, one should, of course, consider not only the characteristics of the medium but also the general characteristics of the equipment associated with that medium. Some of the more important items to consider are listed:

1. Compatibility with the computer and other input-output equipment.
2. Form and quality of displayed data or compatibility with the user.
3. Ease of handling and accessibility of data.
4. Erasability.
5. Permanence of record.
6. Durability of recording medium.
7. Storage density or volume of bulk storage for a given amount of data.
8. Costs associated with bulk storage of data.
9. Size, weight, and power consumption of equipment.
10. Speed of reading and recording.
11. Reliability and maintainability of equipment, reliability of data obtained.
12. Installation and operating costs of equipment, cost of recording material.

The printed page, punched tape, punched cards, and magnetic tape have, in the past, been the most popular recording media. Other media available which hold some promise employ processes such as photography, electrophotography, electrography, and magnetography.

## 2. PRINTED PAGE

### Page Reading Equipment

The printed page provides data in one of the more convenient forms for the user and therefore is as important as other digital computer recording media. The major disadvantage associated with this medium is that considerable difficulty has been experienced in designing machines which will read printed data. The basic problem here is one of character recognition. Some success along these lines has been achieved, and commercial print reading machines are available. Of the various approaches taken to solve the problem, three seem most promising: (1) use of a light source and either photoelectric or kinescope sensing, (2) use of magnetic ink and flux sensing, and (3) use of conductive ink and current sensing.

All the machines developed so far use matrix printing or a special type font to simplify recognition.

### Keyboards

The most practical means of reading data from the printed page to machines is by using an operator and keyboard. Most frequently, keyboards are operated off-line and used to enter data on some other recording medium such as punched tape or punched cards. With appropriate buffer storage, however, it may be more practical to avoid the intermediate step by connecting keyboards directly to the computer and operating them on-line. Typewriters, calculators, and keypunches are examples of equipment that have keyboards directly associated with them. Separate units are also available and usually cost under \$1000.

### Mechanical Printers

Two general categories of page printing equipment have been developed. Those which make use of mechanical motion of, say, a hammer or type bar to transfer ink to paper are referred to as *mechanical printers*. The second category makes use of other means for marking the paper and is discussed under the heading of *nonmechanical printers* in the next section. In selecting the appropriate printing device various general requirements and characteristics should be considered. A number of these are listed below.

1. Mechanism for recording
  - a. Mechanical
  - b. Nonmechanical

2. Mechanism for character selection
  - a. Electromechanical
  - b. Electronic
3. General characteristics of the machine
  - a. Size, weight, and power consumption
  - b. Reliability and maintainability of machine, reliability of data obtained
  - c. Installation and operating costs
4. Recording medium (if other than ordinary paper stock is required)
  - a. Durability of medium
  - b. Permanence of record
  - c. Cost of medium
5. Input
  - a. Serial
  - b. Parallel
6. Output
  - a. Serial—single action
  - b. Parallel, e.g., line-a-time
7. Line feed
  - a. Intermittent or continuous
  - b. Pin fed, friction fed, or both
8. Lines per inch
9. Line size
  - a. Columns per line
  - b. Columns per inch
10. Form and quality of printed data
  - a. Character size
  - b. Type font or matrix printing
  - c. Registration of characters
11. Speed of printing
  - a. Characters per second
  - b. Lines per second
12. Characters per column
  - a. Numerical
  - b. Alphabetical
  - c. Alphanumerical
13. Number of copies obtained

Table 1 lists values for some of the above characteristics which are typical of the various types of mechanical printers.

**Typewriters.** Standard electric typewriters may be used as output printers. These devices are relatively slow compared with other printing machines because they are *single action* in nature and only one character

TABLE 1. TYPICAL CHARACTERISTICS OF MECHANICAL PRINTERS

	Type-writers	Line-a-Time	On-the-Fly	Matrix
Number of copies	7	7	4	4-7
Characters per second	10-20	—	—	—
Lines per second	—	1-3	7.5-20	15-350
Characters per column	10-86	10-47	10-64	47
Columns per inch	12, 16	6-10	6-10	8-10
Columns per line	110-300	5-120	20-130	5-120
Lines per inch	6	6-8	4-8	5-6
Cost, dollars	\$1000-\$3000	\$300-\$50,000	\$50,000-\$165,000	\$100,000-\$200,000
Typical inputs	Keyboard, punched tape	Punched cards	Magnetic tape, digital computer	Magnetic tape, digital computer, punched cards

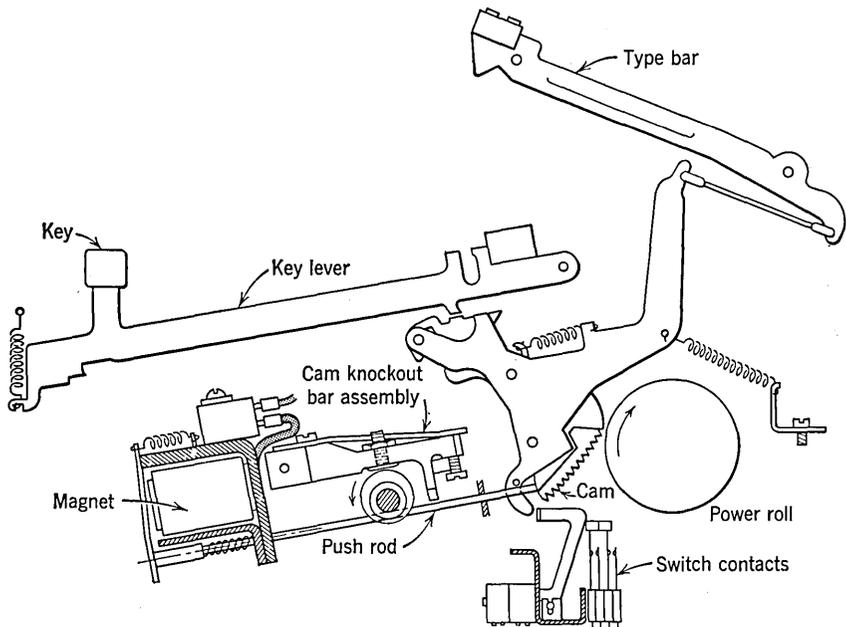


FIG. 1. IBM electric typewriter action.

may be printed at a time. Speeds up to twenty characters per second are obtained on some machines, but most of them cannot exceed ten or twelve. Typewriters may also be adapted for use as keyboard input devices. Figure 1 illustrates a method used by IBM for mechanizing

the input and output functions in their typewriter. Only the equipment associated with one type bar has been shown. Pressing the key a very short distance actuates a linkage, which causes a cam to engage the rubber-covered power roll. The motor-driven power roll flips the cam upward until it disengages the roll. This action causes the character to be printed and opens or closes the switch contacts. These contacts may

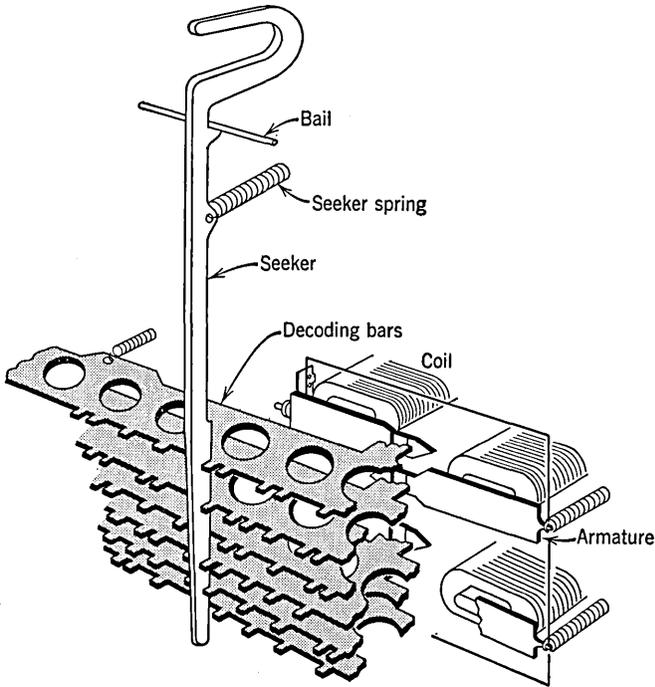


FIG. 2. Mechanical decoder.

be used to generate signals for an external device. Energization of the magnet by an external signal actuates the push rod and thus causes the cam to engage the power roll just as if the key were depressed. The mechanism therefore operates as described above, and data from an external source may be printed out in this fashion.

The equipment described above requires a separate signal line for each character which is to be printed from an external source. To avoid this, many typewriters are equipped with a mechanical decoder such as the one illustrated in Fig. 2. This device accepts a binary-coded number which is used to position decoding bars. Projections on the bars are so arranged that only one of a set of seekers is opposite a slot in each bar and can be pulled in by its spring. A bail moving vertically downward

causes this seeker also to move downward and engage its key and thus actuate the printing mechanism for the appropriate character.

Some typewriters also have provision for incorporating perforated tape punches and readers. The most popular of these is the Flexowriter manufactured by Commercial Controls Corporation. Its basic unit consists of a heavy-duty, motor-driven typewriter capable of printing at a rate of about 9.5 characters per second. This machine receives data through its keyboard or, with appropriate accessories, from a computer or other digital device, or from a punched tape reader which may be mounted directly on the typewriting unit. Its output is in the form of the printed page and, with appropriate accessories, may also include punched tape and signals to a computer or other digital device. The cost of this unit ranges from \$1800 to \$3100 depending on the accessories purchased. Standard electric typewriters modified for computer applications cost around \$1200.

**Line-a-Time Printers.** Machines which print an entire line at one time are referred to as line-a-time printers. Their speeds are in general higher than those of typewriters because the printing operation for each character position on a line is carried out simultaneously rather than individually as is the case with single action machines. There are two basic types of line-a-time printers, (1) type bar printers and (2) type wheel printers. The first of these employs a separate type bar for each character position or column on the line. The type bars each have a complete set of pallets. Printing is accomplished by first positioning the type bars so that the appropriate pallet in each column is directly opposite the paper and then by striking all the pallets simultaneously with a set of hammers. This drives the pallets against the paper, which is held on a platen roll. These machines cost anywhere from \$300 to \$50,000 or more depending on the number of columns available and the control circuitry required. Printing speeds are about  $1\frac{1}{2}$  lines per second. For greater speeds, use of type wheels in lieu of the bars allows more rapid positioning. Machines of this type cost about the same as type bar machines and can print about twice as fast. Line-a-time printers are often an integral part of desk calculators or tabulating machines and occasionally these machines are used instead of separate printers as output devices for digital computers.

**On-the-Fly Printers.** Printing speeds are limited with line-a-time machines because each printing operation requires acceleration of relatively heavy type bars or type wheels. This difficulty is overcome in the on-the-fly printers by allowing type wheels to rotate continuously. Printing is accomplished by actuating hammers when the appropriate character is opposite the paper. The movement of the hammers drives

the paper against the type wheel, and the character is thus printed. There are two basic types of on-the-fly printers, (1) multiwheel and (2) single wheel. Figure 3 illustrates a possible mechanization of the multiwheel on-the-fly printer. At an appropriate time data are inserted into the binary counters and the circuit waits for a pulse from the upper pickoff. Upon receipt of this pulse the *and* gate is enabled and the binary counters begin to count pulses from the lower pickoff. As each

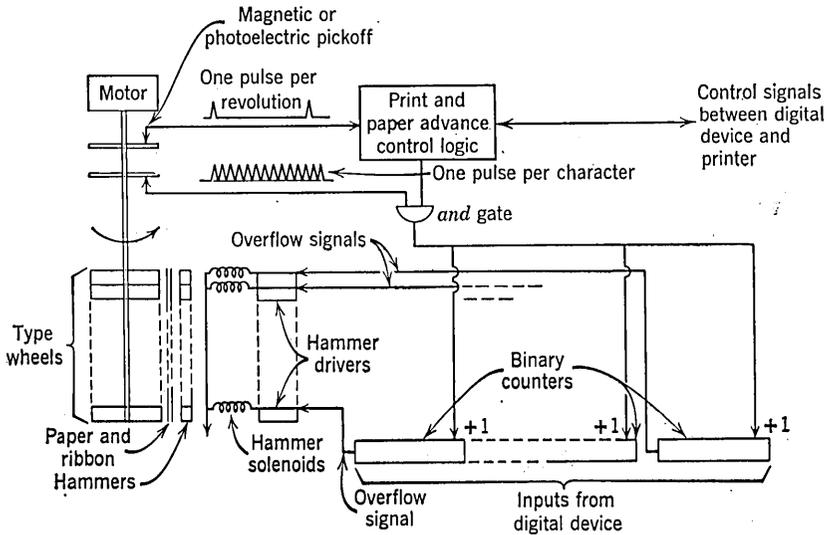


FIG. 3. A mechanization of the multiwheel on-the-fly printer.

register overflows, the corresponding hammer solenoid is actuated and the character opposite the hammer at that particular time is printed. Upon receipt of the second pulse from the upper pickoff, the paper is advanced and new data are inserted into the binary registers in preparation for printing the next line.

Figure 4 illustrates the single wheel printer and shows the relationship between the paper, print wheel, and hammers. Actuation of the hammers may be controlled in a fashion somewhat similar to that used for the multiwheel printer. For the circuit shown in Fig. 3, however, the counters would have to be started at different times. This would allow for differences in timing that are dependent upon the particular position of each hammer along the line. Potter Instrument Company is the only manufacturer of printers of this type.

In general, the speeds of on-the-fly printers vary according to the number of different characters which may be printed in a column.

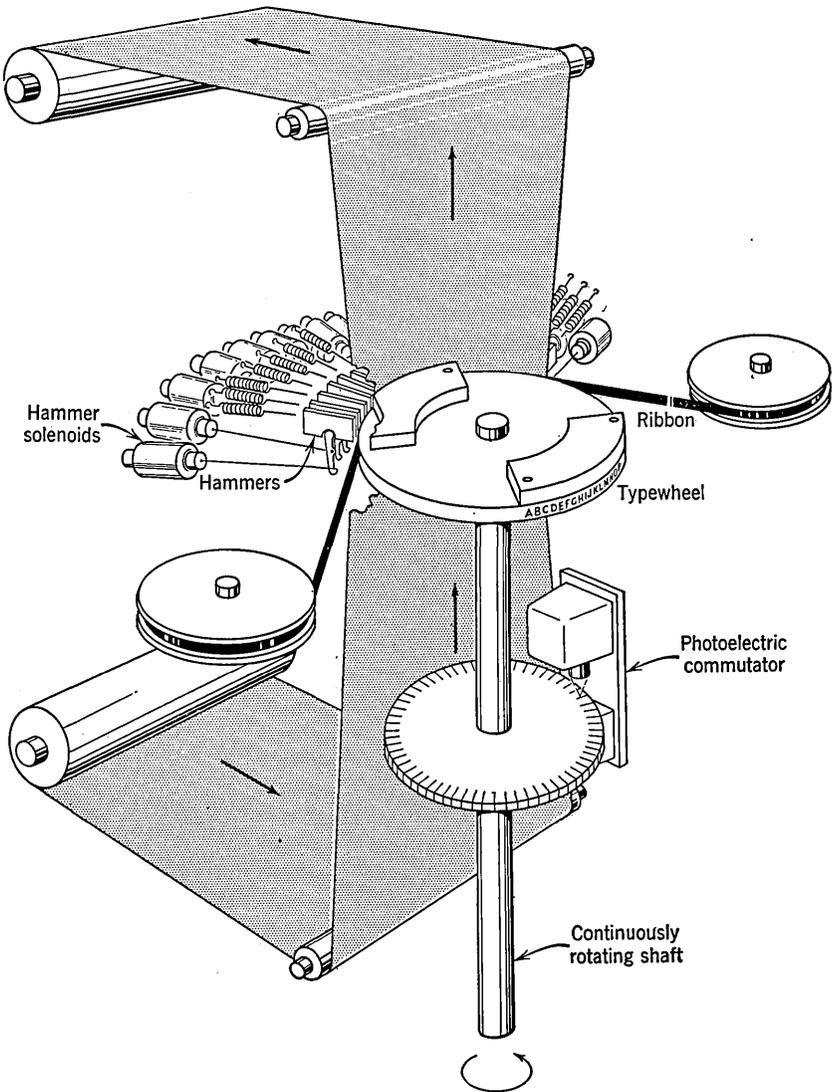


FIG. 4. Potter Instrument Company single wheel on-the-fly printer.

Numeric machines with 10 characters per column are capable of speeds between 15 and 20 lines per second whereas alphanumeric printers run at a slower rate somewhere between 7.5 and 15 lines per second. The number of columns per line for this type machine varies between 20 and 130. It can be seen that considerable electronic circuitry is required

to control and operate these machines when numerous columns are to be printed, and the cost is accordingly quite high compared with line-a-time printers. Typically these machines cost anywhere between \$50,000 and \$165,000.

**Matrix Printers.** Rather than using type font for printing characters, matrix printers record a number of dots arranged in a  $5 \times 3$  or  $7 \times 5$  matrix. The dots to be printed are selected so as to form an image of the desired character. Although numbers formed using a  $5 \times 3$  matrix are readable, the  $7 \times 5$  matrix yields a much clearer representation for alphanumeric characters. The dots themselves are printed by using small hammers or wires which strike the paper in order to transfer ink to it.

Matrix printers are of two types: (1) the simultaneous matrix printer and (2) the scanning matrix printer. The first uses a complete matrix of wires for each column on the line. The second type uses a row of wires which are actuated several times in order to print a single line of characters. Intrinsically printers that employ the matrix technique are capable of speeds which are high compared with other types of mechanical printers. This is because no requirement exists for moving type font in and out of position. In practice, however, speeds for matrix printers rarely exceed those for on-the-fly machines, and their capabilities are about the same. A notable exception to this is an Eastman Kodak address label printer which produces five columns of data at a rate of 350 lines per second.

Since it is necessary to control somewhere between 5 and 35 hammers or wires per column, the electronic circuitry associated with matrix printers is extensive if numerous columns are to be printed; the cost is correspondingly high. The quality of matrix printing tends to be somewhat inferior to conventional printing, but registration of the characters is generally better than that obtained in using an on-the-fly machine.

### **Nonmechanical Printers**

The term *nonmechanical printer* refers to the class of printers which use means other than mechanical for selecting and transferring characters to be printed. By avoiding the use of mechanical equipment such as hammers and type wheels, this class of equipment provides the fastest means of producing printed data. To accomplish the printing operation various processes may be used. Photography, electrophotography, magnetography, and electrography are discussed below. Since two of these methods frequently employ cathode ray tube display devices, a section is also included on this equipment which may also be classed as input-output equipment in its own right. In general, attention is focused

toward the printing of numeric or alphanumeric characters. All the methods, however, lend themselves to the printing of binary information, and extremely high recording speeds are possible. In addition, reliable techniques have been developed for reading binary information printed in this fashion.

**Cathode Ray Tube Display Devices.** Conventional cathode ray tubes may be used conveniently for the display of quantities in plotted form. Using digital to analog converters, 0.1% accuracies are obtainable on each axis, and the beam may be directed by a 20-bit number to any one of the spots in a  $1024 \times 1024$  matrix. Alphanumeric characters may be displayed in matrix form by selecting the appropriate spots, but this procedure requires either considerable electronic circuitry or extensive programming.

A more practical method for displaying numeric or alphanumeric characters is to employ a shaped beam tube. Typical of these tubes is the Characteron manufactured by the Stromberg-Carlson Division of General Dynamics Corporation. It is illustrated in Fig. 5. The tube uses an electron gun which generates a diffused beam. Deflection plates are employed to direct the beam toward any one of 64 holes in an aperture plate. This aperture plate shapes the beam, and the desired character is formed. After leaving the aperture, the beam is redirected toward the desired position on the viewing screen. The character may be written at any point on a  $1024 \times 1024$  matrix. One 6-bit number is required for character selection, and up to 20 bits can be used for position selection. Between writing of successive characters a blanking signal is required. An additional feature of this tube is that it may also be used as a conventional cathode ray tube by directing the beam through a round hole in the aperture plate rather than through a shaped character. At present writing speeds on the order of 15,000 to 30,000 characters per second can be obtained, but from 100,000 to 200,000 characters per second should be possible in the future. For use as a display device, tubes of this type have screens up to 19 inches in diameter.

Another example of a shaped beam tube is the Typotron manufactured by Hughes Aircraft Company. It is similar in many respects to the tube previously discussed, but it has an additional feature in that characters are retained on the face of the tube until an erase signal is applied. (See Ref. 1.)

**Photography.** One of the more obvious means of recording data displayed on a cathode ray tube is to employ photographic techniques. This approach is particularly desirable if large amounts of data must be filed since extremely high recording densities are obtainable. The procedures employed are fairly standard and so the discussion here is

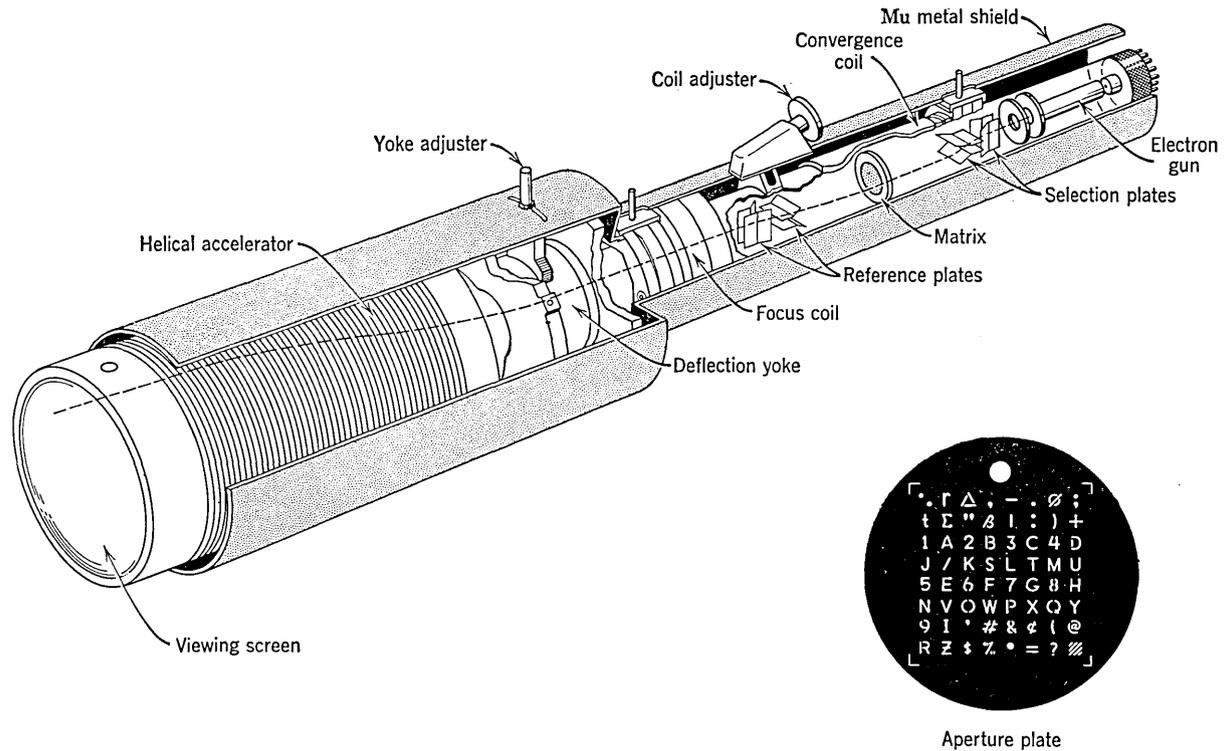


FIG 5. Charactron shaped beam tube. The aperture plate matrix has 64 characters and symbols. (Courtesy of Stromberg-Carlson, a division of General Dynamics.)

somewhat limited. The chief disadvantage associated with this technique is that the wet developing process is not well suited to printer applications. It is both expensive and cumbersome when employed in a printing device.

**Electrophotography.** The term electrophotography refers to recording techniques which involve the capture of light images on an electrically charged surface and the subsequent development of these images. Electrophotographic processes are similar to conventional photographic processes in the sense that a light source is used to create latent images in some recording medium. The two processes differ, however, in the nature of the latent images that are stored and therefore in the methods

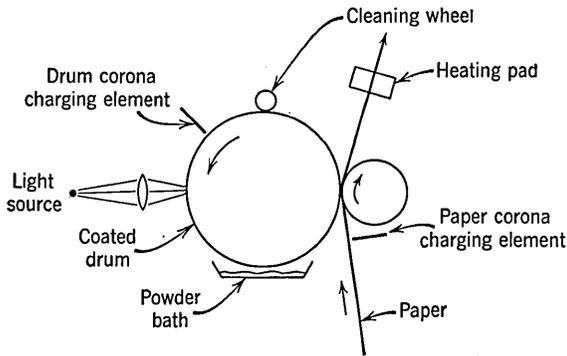


FIG. 6. Xerography.

required for development for these images. For printer applications, electrophotography appears superior to conventional photography because the methods required for development do not involve wet processing. Figure 6 illustrates an electrophotographic process developed by the Haloid Company under the name of Xerox. Xerography interpreted literally means dry printing. As seen in Fig. 6, a drum coated with a thin film of photoconductive material is charged just prior to being exposed to a light source. Area upon which the light falls becomes conductive and loses its charge through the drum. The surface of the drum is then brought in contact with a developing powder of opposite charge. The particles of powder cling to the drum until it contacts a roll of paper which is charged so that the particles are transferred to it. The powder images are then fixed on the paper by heating. The Stromberg-Carlson Division of General Dynamics Corporation has marketed a printer that employs this process and the Charactron display tube. The

unit costs about \$150,000 and is capable of printing eighty-three, 120 column lines per second.

Other xerographic machines have been developed which may be used for the duplication of data which have already been printed. In these cases reflected light from the master copy is used in lieu of the cathode ray tube display device. Another example of electrophotographic processes has been developed by RCA under the name of Electrofax. It is

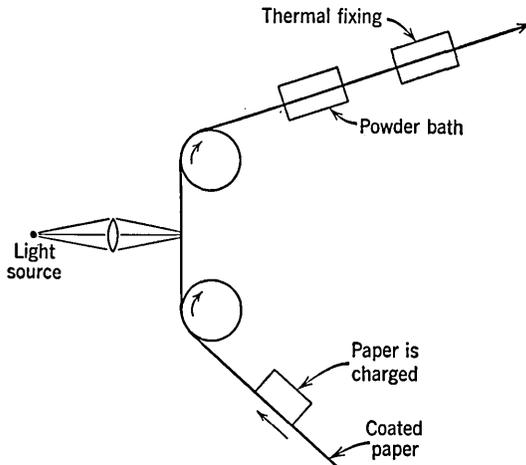


FIG. 7. Electrofax.

similar to the one previously discussed except that the electrostatic image is produced directly on the paper. Figure 7 illustrates this process and shows the various operations which are required.

**Magnetography.** A printing process that has been developed by the General Electric Company involves the use of magnetic materials. Latent magnetic images are recorded on a steel tape that is passed through a bath of ferromagnetic powder. The small particles of powder cling to the magnetized portions of the steel tape until the tape is pressed against a strip of warm wax paper. At this point the powder is carried away by the paper and secured to it by cooling. The tape is then demagnetized and new images are recorded on it. Figure 8 shows a representation of this process. To record the magnetic images on the tape, a drum is provided which has core pieces mounted upon it, shaped in the form of characters. A special electromagnet or pulsing shoe is mounted directly opposite the drum on the other side of the steel tape. Pulsing this shoe causes a magnetic image of the particular core piece opposite the shoe at that time to be formed in the steel tape. The relative speeds

of the steel tape and drum must be properly adjusted so that all the different characters on the drum pass each line on tape within range of the pulsing shoe. Timing circuitry is required in order to select the desired character and to position it correctly on the steel tape. Speeds up to 200 lines per second have been reported with this technique.

**Electrography.** This term generally refers to those recording techniques which make use of an electric spark for recording images of characters on a medium. To accomplish this a row of wires is usually placed across the width of the paper which is moved continuously past

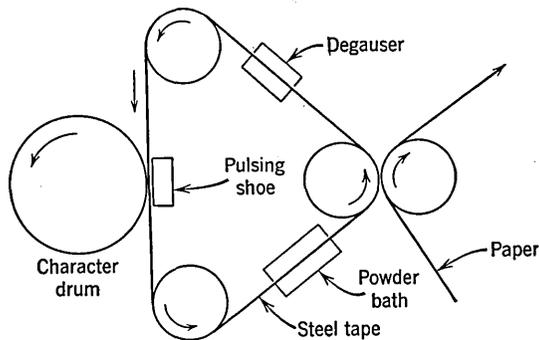


FIG. 8. Magnetography.

the recording head. The characters are printed in matrix form by using the scanning matrix technique. One type of electrographic printing employs Teledeltos paper manufactured by Western Union. This paper is a multilayer material used in facsimile work and contains beneath the light gray reading surface a pitch black layer which is exposed during the sparking process. Atomic Instrument Company has developed a printer that uses this paper. Speeds up to 600 lines per second have been obtained.

Another form of electrographic recording has been developed by the Burroughs Corporation (see Ref. 2). Their technique involves the use of paper coated with a thin layer of thermoplastic material of high resistivity. The coating maintains a latent electrostatic image of the character after it has passed through the recording head. The image is made visible by inking with a powder which clings to the charged portion of the medium. The powder is fixed by heating. Speeds in the neighborhood of 5000 characters per second have been reported using only one recording head.

3. PERFORATED TAPE

Types of Perforated Tape

Paper is the most frequently used material for punched tape. Mylar plastic bonded to metal foil or paper has also found application. Perforated tape is available in several widths depending on the number

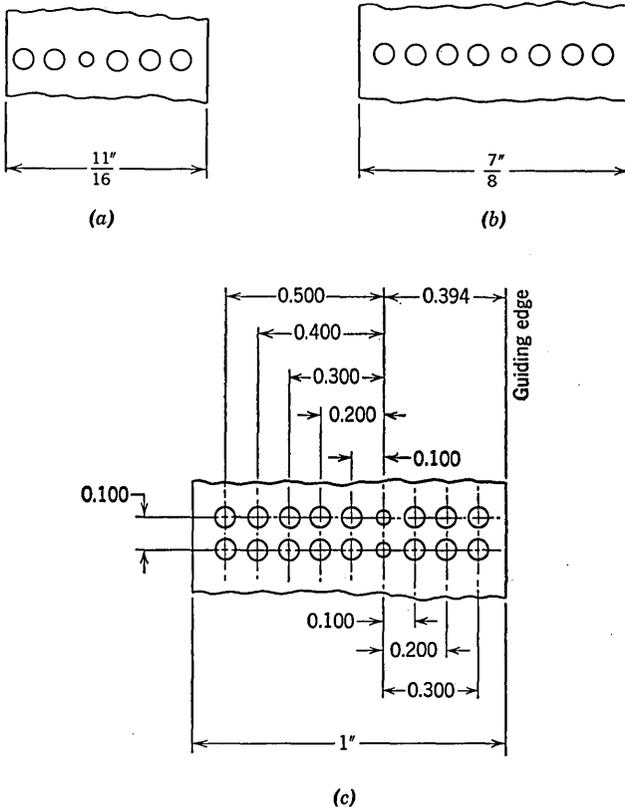


FIG. 9. Standard punched tape: (a) 5-level tape, (b) 6- and 7-level tape, (c) 8-level tape.

of channels used. The number of *channels*, or *levels*, is the number of information bits per character. The term character refers to one row of holes across the width of the tape. Five-, six-, seven-, and eight-level tapes are common. For these tapes an additional smaller hole is perforated in each character so that the tape may be conveniently pulled over a reading or recording head. This hole is called a *feed* or *sprocket* hole. Figure 9 depicts various types of punched tapes and shows the

position of holes. Six-level tape has the same measurements as seven-level tape but uses only six of the seven hole positions. Table 2 summarizes these data in tabular form. Although Fig. 9 indicates that the sprocket holes are in-line with the information holes, some commercially available punches produce sprocket holes a small distance (0.013 inch) ahead of the information holes.

TABLE 2. TYPICAL CHARACTERISTICS OF PERFORATED TAPE

Tape width (inches)	
5-level	$\frac{11}{16}$
6-level	$\frac{7}{8}$
7-level	$\frac{7}{8}$
8-level	1
Tape thickness (mils)	3-8
Distance between holes, center to center (inches)	0.100
Distance between sprocket hole centers and guiding edge of punched tape (inches)	0.394
Diameter of information holes (inches)	0.072
Diameter of sprocket holes (inches)	0.046
Characters or codes per inch	10
Approximate diameter of punched tape rolls (inches)	
1000 ft	9
250 ft	4
Approximate volume of 1000 pieces of loosely packed chad (waste) (cubic inches)	0.02

Table 3 summarizes the characteristics of typical perforated tape equipment.

TABLE 3. CHARACTERISTICS OF TYPICAL PERFORATED TAPE EQUIPMENT

		Speeds, Characters/ Sec	Weight, lb	Size, cu ft	Power, watts	Cost, dollars
Punches	Mechanical					
	Motor	20-250	25	1	150	1000
Mechanical readers	Motor	20-60	25	1	150	1000
	Solenoid	20-40	2	0.05	30	800
Photoelectric readers	Motors	150-750	125	3	350	4000

### Tape Punches

**General Characteristics.** Commercially available tape punches have maximum speeds which range from 20 to about 250 characters per second. A large majority of them, however, cannot operate faster than 60 characters per second. Punches which fall into this category are generally priced under \$1000 whereas a very high-speed punch will cost as much

as \$10,000. Tape punches derive mechanical power from an electrical motor and conduct three basic operations: (1) punching, (2) tape feed, and (3) synchronization or timing control. Some punches are capable of accepting any common width of tape with only minor adjustments while others have been designed to handle only one particular width.

**Punching Mechanisms.** Various means are available to accomplish the basic operation of punching; three illustrative examples of mechanisms which accomplish this function are described below.

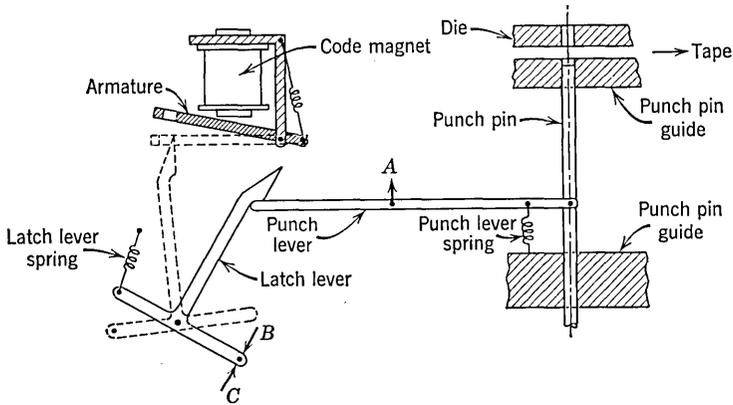


Fig. 10. Punching action of the Commercial Controls 20-character-per-second punch.

Figure 10 shows a simplified version of the approach taken by Commercial Controls Corporation for their 20-character-per-second punch. Only that equipment necessary to punch one hole is shown. With the latch lever in the latched position indicated by the solid lines, a force is applied at A which drives the punch pin up into the die, thus perforating the tape. In the unlatched position indicated by the dotted lines, a force at A will cause only the punch lever to move and not the punch pin. This is due to the restraining force applied by the punch lever spring. During an appropriate time of the punch cycle positive latching is insured by applying a force to all the latched latch levers at point B. Upon completion of the punch cycle, a force is applied to all latched levers at point C. This action returns the latch levers to the position indicated by the dotted lines in preparation for punching the next character.

Figure 11 shows a simplified version of the approach taken by Teletype Corporation for their 60-character-per-second punch. A vertical reciprocating type motion is imparted to the long toggle arm at point A. When the armature and blocking pawl are in the position indicated

by the solid lines, the punch pin is driven through the tape on each downward stroke. When the armature is in the position indicated by the dotted lines, the blocking pawl engages the long toggle arm on its downward stroke and causes the knee to buckle in the direction indicated by the arrow at *B*. This action prevents appreciable movement of the drag link and thus inhibits the punching of a hole. The long toggle arm

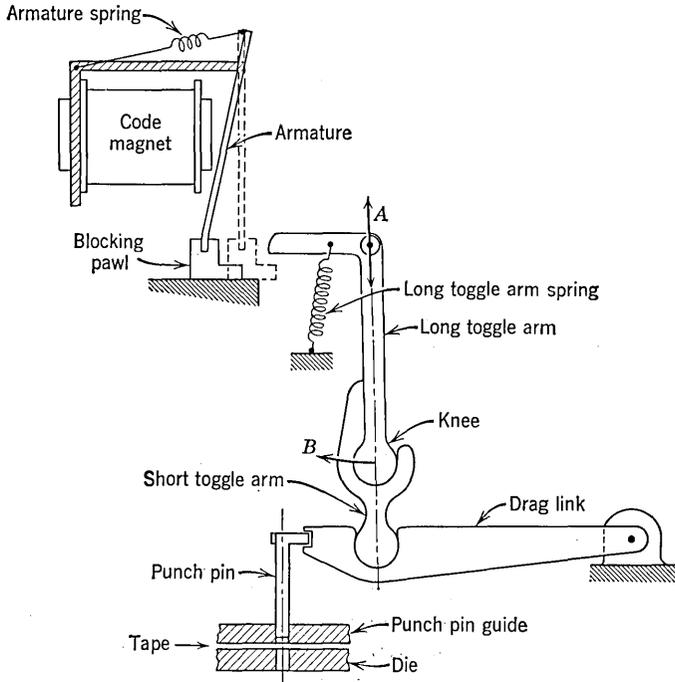


FIG. 11. Punching action of the Teletype Corporation's 60-character-per-second punch.

spring insures that the knee will not buckle when a hole is to be punched. The force applied by this spring is overcome if the blocking pawl has been moved to intercept the downward movement of the long toggle arm.

Figure 12 shows a simplified version of an approach taken by Soroban Engineering, Inc., in one of their 240-character-per-second punches. With the interposer in the position indicated by the solid lines, a downward motion of the punch bail drives the punch pin into the die and thus perforates the tape. On the upward stroke of the punch bail, the punch pin is withdrawn from the die. If the interposer is in the position indicated

by the dotted lines, the punch bail slides along the punch pin, but does not drive it through the tape. The same punch bail is used to drive all the punch pins. At the end of the cycle another bail applies a force to all the interposed interposers at point A, and thus returns them to the position indicated by the dotted lines in preparation for

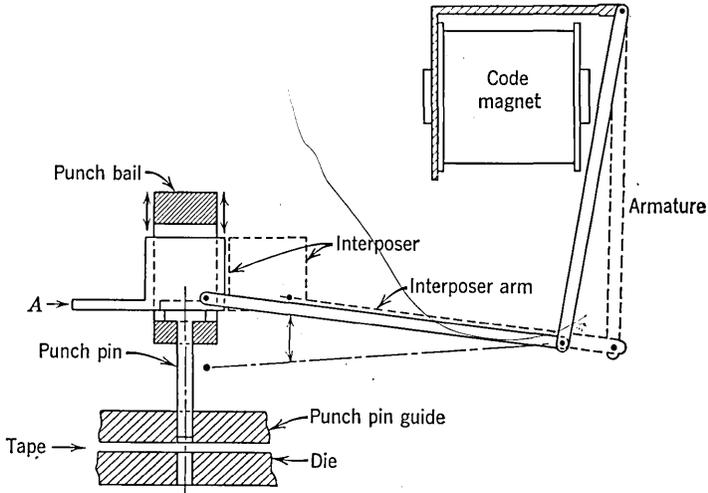


FIG. 12. Punching action of a Soroban 240-character-per-second punch.

punching the next character. Considerable effort is required in the design of very high-speed punches to reduce accelerations and impact loads. The motion of the punch bail in this case is fixed by a special cam in order to reduce impact on the interposers. The elimination of springs is also important since they can cause undesirable mechanical resonances.

**Tape Feed Mechanisms.** Various means are available for advancing the tape. A typical method is shown in Fig. 13. In this diagram, the ratchet wheel and sprocket wheel form an integral unit. The feed pawl arm moves under the influence of the feed rod and causes the feed pawl to engage the ratchet. The ratchet rotates in the direction shown. The pins on the sprocket wheel engage the tape feed holes previously punched by the punching mechanism, and the tape is therefore advanced each time the feed rod is actuated. Actuation of the feed rod may be accomplished by a number of means. Frequently, feed magnets are used to control the motion of various mechanical parts which in turn actuate the mechanism. The detent in Fig. 13 insures positive indexing of the ratchet-sprocket combination.

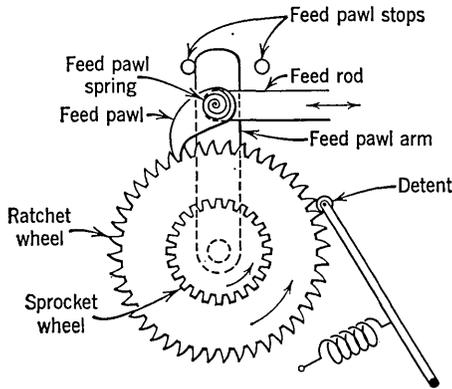


FIG. 13. Typical tape feed mechanism.

**Synchronization and Timing.** Requirements for synchronization fall into two general categories: (1) synchronization of input data with the punch and (2) synchronization of the internal elements of the punch with each other. In the first category, some means is required to ensure that the code magnets are energized during the appropriate portion of the punching cycle. A common method of accomplishing this is to mount electric contacts or reluctance pickups on the drive shaft in order to generate timing signals which can be used by the input circuitry. A simplified block diagram of such an approach is illustrated in Fig. 14. It has been assumed that the punch generates two pulses each cycle, a *ready signal* indicating that the code magnets should be energized by the appropriate word, and a *complete signal* indicating that the code magnets

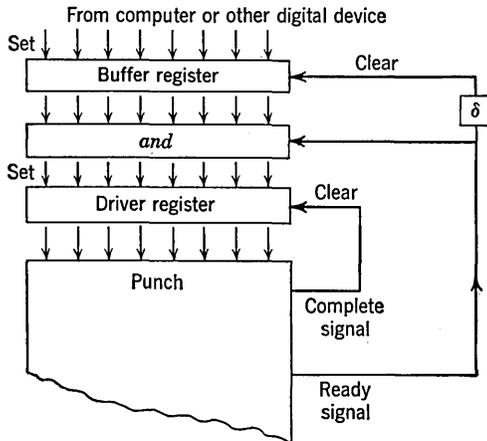


FIG. 14. Block diagram of synchronizing system.

should be deenergized. A buffer register has been provided for loading the driver register.

Note that in the configuration shown, the time between insertions of words into the buffer register should be at least as great as the time required for one complete punch cycle and that no word should be inserted during the small interval of time required to load the driver register and clear the buffer register. Additional buffer storage is required if it is desirable to accept data at higher instantaneous rates. The average rate at which data are transferred to the punch input circuitry over a long period of time of course cannot exceed the punching speed regardless of the buffer size. Sufficient buffer action is obtained from the two registers shown, however, to gain the following advantages:

1. The computer or digital source of information has been left free between word insertions to accomplish other operations while punching takes place.
2. The punch can record at near its maximum rate and computer operation is inhibited only a small portion of the time. Without proper buffering or timing control it may be impossible to utilize the punch in an efficient manner. This is due to timing uncertainties associated with the time between insertion of data and use of them by the punch.

In addition to proper energization of the code magnets, some punches also require external logic to control the actuation of a feed magnet. This may be accomplished by using timing signals from the punch and a signal generated by the data source to indicate that an output operation is being carried out.

Synchronization of the internal elements of the punch is accomplished for the most part by mechanical means through the use of linkages, cammed surfaces, etc. Electric contacts, magnets, and electrically operated clutches are also used in some punches to achieve the necessary control. Some of the control requirements are to insure that:

1. The tape is not advanced before the punch pins have been sufficiently withdrawn from the tape.
2. The punching operation does not commence too soon in relation to the normal time that the code magnets are energized.
3. The tape advance has been sufficiently completed before actual punching occurs so as to prevent tearing of the tape.
4. The various operations are carried out in the appropriate order and with a minimum of delay.

### **Tape Readers**

Commercially available tape readers fall into two general categories: (1) mechanical readers and (2) photoelectric readers, depending on the

method of sensing holes. With photoreaders, sensing is accomplished by using a light source and photoelectric cells. This allows much higher reading speeds than are possible with mechanical readers. A second difference between the two types of readers is the method for transporting the tape. Photoreaders have a more elaborate tape transport system in order to utilize effectively their intrinsic capability of high-speed sensing.

**Mechanical Tape Readers.** Commercially available mechanical readers generally cost less than \$1000, and have maximum speeds which range from 20 to 60 characters per second. They employ either a solenoid or motor to obtain the necessary mechanical power. Readers that use motor drive have some form of latching mechanism or clutch in order to control the application of power to the reader. The latching mechanism or clutch is usually actuated by an electromagnet. In general, the motor-driven readers are more appropriate than solenoid-actuated readers in applications which require continuous reading and higher speeds.

Readers perform three basic operations: (1) sensing, (2) tape advance, and (3) synchronization or timing control. Tape advance may be accomplished by a number of means. The ratchet-sprocket combination described in the previous section for tape punches is a typical method. Synchronization of the internal elements of the reader is achieved by appropriate mechanical design rather than through the use of electric control circuitry. Synchronization of the reader with external equipment may be accomplished by transmitting a signal to the tape reader each time a character is to be sensed. Higher reading speeds are usually obtainable, however, if the tape reader is allowed to run at its own speed. In this case the external equipment is synchronized to the reader and employs pulses which are generated by the reader each time a character is sensed. In addition, start and stop signals are required from the external equipment in order to initiate or discontinue the reading operation.

**Mechanical Sensing Mechanisms.** Mechanical readers generally sense holes by driving a pin toward the punched tape. Electric contacts are closed if the pin passes through a hole. A typical mechanization of the sensing operation is shown in Fig. 15. It is a simplified version of one employed by the Teletype Corporation. The main bail is common to all the sensing pins and moves with a vertical reciprocating type motion. On its upward stroke it allows the sensing pin to move up under the force of the sensing pin spring. If there is a hole in the tape, the sensing pin continues its upward motion, passing into the hole. The switch bar, which follows the sensing pin, rocks from the spacing contact to the marking contact. If there is no hole in the tape, the

upward motion of the sensing pin is halted by the tape, and the switch bar remains on the spacing contact.

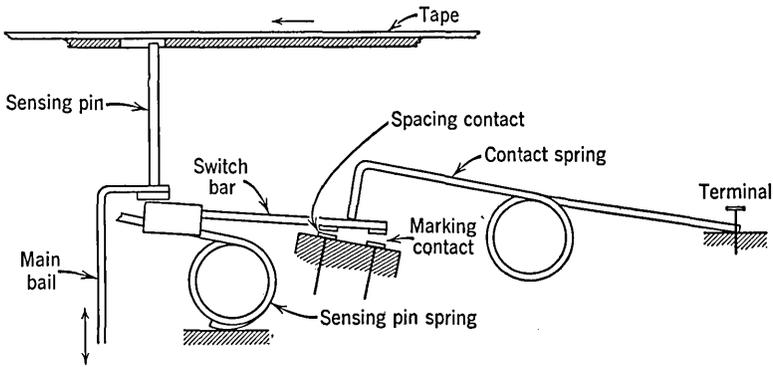


Fig. 15. A mechanical sensing mechanism for paper tape.

**Photoelectric Tape Readers.** Typical reading speeds for photoelectric tape readers range from 150 to 750 characters per second. Higher speeds are obtainable, however, and up to 2000 characters per second have been successfully read. The cost of these units ranges from \$3000 to \$5000. Other data concerning them have been provided in Table 4.

TABLE 4. TYPICAL CHARACTERISTICS OF COMMERCIAL HIGH-SPEED PHOTOELECTRIC TAPE READERS (TAPE TRANSPORT INCLUDED)

Number of channels	5, 6, 7, or 8
Tape speed (inches per second)	15~75
Reading speed (characters per second)	150~750 (2000 characters per second possible with some very high-speed photoreaders)
Start or stop time	Less than 5 milliseconds
Stopping distance	On the stop character or on the character following the stop character
Tape capacity of transport (feet)	1000
Size	2 feet of standard relay rack
Weight (pounds)	75-150
Power requirements (watts)	100-450
Cost (dollars)	3000-5000

Photoelectric tape readers have three major components: (1) the reading head, (2) the tape drive, and (3) the reel assembly. The last two items make up the tape transport system and are similar in many respects to the tape transports associated with magnetic tape recorders.

Some photoelectric readers are not supplied with a reel assembly, and discussion of this unit is deferred to the section on magnetic tape equipment.

**Tape Drivers for Photoelectric Readers.** These units generally employ a capstan friction drive system to pull the paper past the reading head. Use of friction drive eliminates much of the tape wear experienced with mechanical readers. A means of stopping the tape is also required. Short starting and stopping times are desirable and many photoreaders

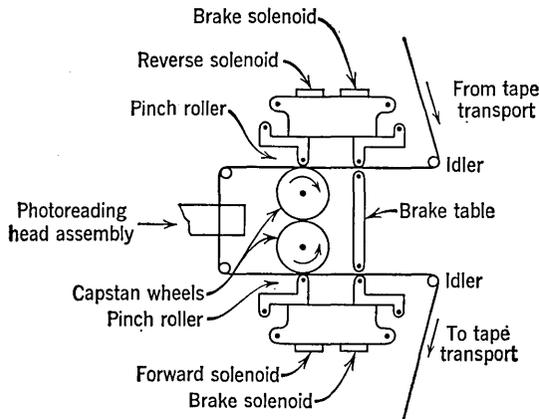


FIG. 16. Potter tape drive assembly.

are capable of stopping and starting without missing a character. This capability eliminates the requirement for leaving blank tape between blocks of data during the recording operation. Two examples of tape drive assemblies are discussed below.

The first, pictured in Fig. 16, is one employed by Potter Instrument Company. In the forward mode, the forward solenoid pinches the tape to the lower capstan. In the reverse mode, the reverse solenoid pinches the tape to the upper capstan. The lower capstan is friction driven by the upper capstan, which in turn is driven by a motor. To stop the tape, the brake solenoids operate together to pinch the tape between their pinch rollers and the brake table.

Figure 17 illustrates a differential gear arrangement used by Ferranti Ltd., for driving punched tape in their high-speed reader. The input to the differential gear is driven by a continuously running motor. One output shaft of the differential gear is used as a combined tape drive and brake. The other output shaft is used as a clutch drum. The tape drive and brake drum is started or stopped by the application of electromechanically operated brakes to one or the other of the output

shafts. The brake coils are driven by a flip-flop so that at any particular time one and only one brake is actuated.

**Photoreading Heads.** Reading head assemblies for photoelectric tape readers consist of a light source, a bank of photoelectric cells, and the equipment necessary to focus the light through the perforated holes onto the photo cells. One cell is used for each information channel, and

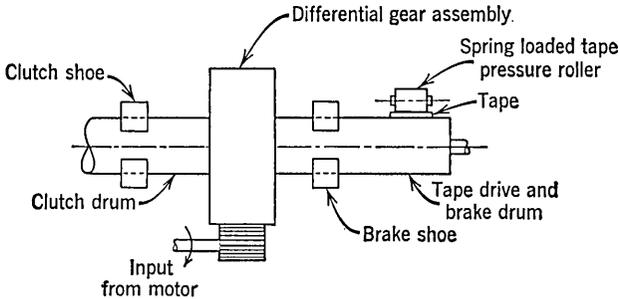


FIG. 17. Ferranti tape drive assembly.

one is used for the sprocket channel. The signal obtained from the sprocket channel is used for control purposes. Both photoconductive and photovoltaic devices have been successfully employed. Some readers have been provided with a mechanical light chopper. Modulation of the light source in this fashion allows a-c amplification and eliminates drift problems due to temperature variations of the photocells.

**Auxiliary Equipment**

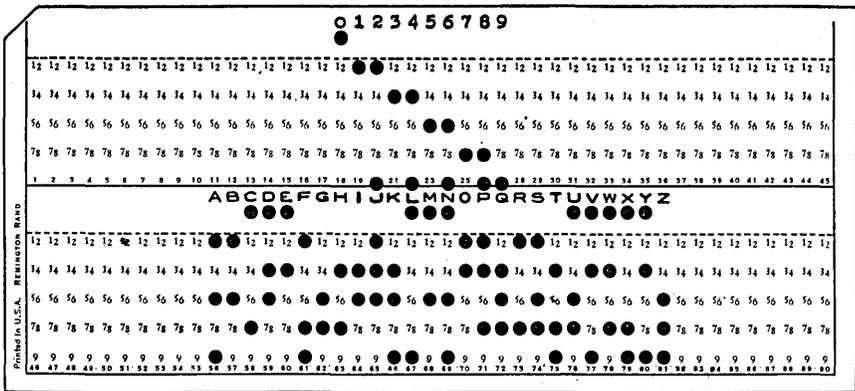
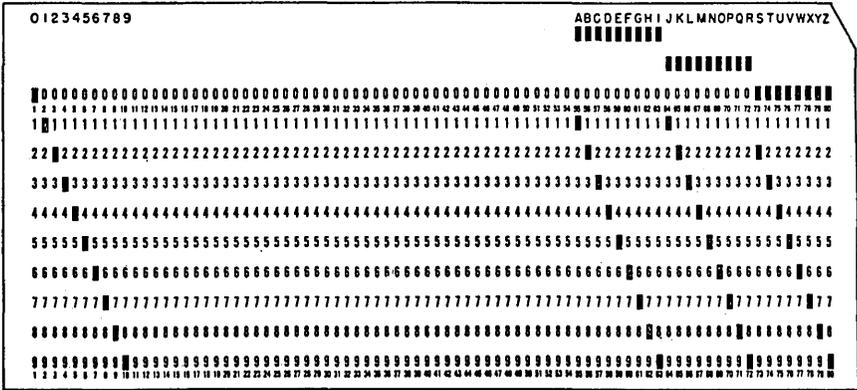
Auxiliary equipment associated with punched tape includes the following items:

1. *Key punches.* Consists of a keyboard, tape punch, and control circuitry necessary to prepare tape manually. Typewriters are frequently used in this application. Occasionally a mechanical reader is also provided so that the data being punched may be compared with another tape for verification purposes.
2. *Comparators.* Consists of two readers (sometimes one dual mechanical reader) and the circuitry necessary for comparison of two tapes.
3. *Reproducers.* Consists of reader, punch, and circuitry necessary for reproduction of tapes.
4. *Converters.* Consists of a reader or punch and equipment associated with some other recording medium. Used to transfer data to and from other recording media.

4. PUNCHED CARD MACHINES

Punched Cards

Of the several types of punched cards which are available for various machine accounting systems, two have found the widest use in digital computer applications. The first of these is manufactured by IBM and



Thickness, 0.0070" to 0.0067"; width, 3 1/4"; length, 7 3/8".

FIG. 18. Standard punched cards: (a) IBM, (b) Remington Rand.

the other by Remington Rand. They are the same size and both cost approximately \$1.50 per thousand. They differ however in the format used for punching data. The IBM card has 80 columns with 12 positions for rectangular holes in each column, while the Remington Rand card uses 90 columns each with 6 positions for round holes. Figure 18 shows

these cards with their measurements and illustrates the standard code used for recording alphanumeric data on each of the cards. Of course other codes are also possible and frequently used particularly when binary data must be recorded. (See Chap. 5.)

### **Punched Card Readers**

Card readers may be used in both on-line and off-line applications. When used for off-line purposes, they are generally associated with other equipment such as that discussed in the paragraph on auxiliary punched card equipment. There are a number of different techniques for reading cards. The most common of these involves the use of mechanical brushes which sense the holes. Electric contact is usually made through the hole or less frequently in a fashion similar to that described for punched tape readers. Other reading techniques involve photoelectric or electrostatic sensing. These machines are capable of speeds which range from 100 to 1000 cards per minute. They cost in the neighborhood of \$40,000.

### **Card Punches**

Card punches, like card readers, may be found in both on-line and off-line applications. Their speeds are slower than readers, however, and range between 100 and 200 cards per minute. When used for off-line purposes, they are generally associated with other equipment such as that discussed in the next section. The price of these units is in the neighborhood of \$30,000. Additional information on card punches and card readers may be found in Chap. 5.

### **Auxiliary Equipment**

Aside from readers and punches there are numerous other types of punched card equipment which have been developed for machine accounting systems prior to the advent of the large-scale digital computer. In addition, conversion equipment has been developed for transferring data between punched cards and other recording media. All these units make use of either a card reader, a card punch, or both, but they have additional equipment associated with them for accomplishing various other functions. Since these functions may be found useful in off-line digital computer applications, a number of them are discussed below.

**Key Punches.** These machines have a keyboard, a card punch, and the control equipment necessary for manual preparation of cards. Some have a small mechanical storage which may be set and used to punch any data which are identical on all the cards being prepared. Other key punches are capable of printing the data on the card or *interpreting* while the punching operation takes place. Operator speeds for these

machines of course varies widely; a good operator, however, should be able to punch roughly 125 eighty-column cards per hour.

**Verifiers.** These machines are used to *verify* the correctness of holes punched in cards which have been prepared by an operator. There are various methods whereby this is accomplished, but in general it amounts to another keying operation in which the second version is compared with the first. Errors are indicated by locking the keyboard or by other appropriate means. Verifiers are frequently combined with a key punch to form a single unit capable of both functions.

**Reproducing Punches.** These machines have a card reader, a card punch, and the control equipment necessary for duplicating punched cards. Some have additional readers and capabilities for collating, comparison, addition, and subtraction. Most of them are capable of reproducing cards at a rate of at least 100 per minute.

**Tabulators.** Tabulating machines generally include several card readers and a line-a-time printer. (See Sect. 2, Mechanical Printers.) Data may be read from the cards and printed out for visual study. In addition these machines also have the capability of following instructions on the cards and adding or subtracting numbers as required by these instructions. Results of the computation and information in alphabetical form can be printed out. The usual speed for these machines is 150 cards per minute.

**Summary Punches.** Results and other data obtained by tabulators may be punched onto cards with summary punches. These machines are not capable of reading cards as is the case with reproducing punches.

**Sorters.** Card sorters have a card reader and the appropriate equipment for separating cards according to the data on them. Sorting of the cards usually requires one pass per column. Speeds up to 1000 cards per minute are possible on some card sorters.

**Collators.** These machines employ two separate magazines for the cards, and are capable of accomplishing more complicated rearrangements of cards than is possible with sorters. Sets of cards are read and then merged or separated depending on how the data compare. Collators also have the capability of checking the sequence of cards in the magazine. They are capable of speeds of about 250 cards per minute.

**Interpreters.** Punched data may be read by these machines and printed on the same or different cards at a rate of approximately 100 cards per minute.

**Calculating Punches.** These machines are basically small digital computers. They read numerical data and instructions from punched cards, perform various arithmetic operations, and punch out the results.

**Converters.** A large variety of equipment has been developed for transferring data back and forth between punch cards and other recording media such as punched tape and magnetic tape. Equipment is also available for transferring data between IBM and Remington Rand cards. Digital to analog converters have been incorporated with punched card readers for obtaining signals which may be used with plotting devices.

## 5. MAGNETIC TAPE

### Magnetic Tape Description

Magnetic tape employed in computer applications generally consists of a cellulose acetate or polyester base of about 1.5 mils thickness and a magnetic coating which may be from 0.3 to 0.7 mil in thickness. The magnetic coating is made up of finely divided iron oxide particles embedded in a binder such as vinyl resin. The particles themselves are less than 0.7 micron in length and are generally about one-quarter of that in thickness. The coatings are roughly 75% iron oxide by weight and 50% by volume. Of the two bases, cellulose acetate and polyester, the latter appears to have better properties for magnetic tape. The polyester, or Mylar (DuPont trade name) as it is usually called, is more immune to changes in physical dimensions due to environmental conditions. Magnetic tape has also been made with a nonmagnetic metal base plated with a magnetic material. Only a few of the commercially available tape readers, however, have been specifically designed to use this type of tape. Rolls of plastic tape are available in lengths up to 5000 feet for the 1.5-mil base tapes and 7200 feet for the 1.0-mil tapes. Common widths vary between  $\frac{1}{4}$  and 1 inch. A major problem associated with magnetic tape recording is concerned with blemishes and imperfections in the tape that hinder proper recording. The problem has been severe enough that, in some instances, imperfections have been marked and equipment provided to avoid recording over the imperfect areas. Various other characteristics which are representative of commercially available plastic base tapes have been listed in Table 5.

### Magnetic Heads

Small electromagnets or heads, as they are called, are used to record, read, and erase data on magnetic tape. (See also Chap. 19.) The heads are grouped together to form head stacks. These stacks may contain anywhere from 15 to 30 heads per inch; thus, for example, from 7 to 15 tracks can be recorded on  $\frac{1}{2}$ -inch tape. Table 6 provides data for a typical magnetic head stack. However, since the design of heads varies considerably throughout the industry and differs depending on whether the

TABLE 5. CHARACTERISTICS OF TYPICAL PLASTIC BASE MAGNETIC TAPE

	Polyester		Cellulose Acetate
Typical thicknesses (mils)			
Base	1.0	1.5	1.5
Coating	0.35	0.6	0.6
Total	1.35	2.1	2.1
Tensile strength (pounds per $\frac{1}{4}$ -in. tape)			
70°F 50% relative humidity	7.3	12.0	5.6
70°F 90% relative humidity	7.3	12.0	4.2
140°F 50% relative humidity	6.0	10.0	4.4
Yield strength (pounds per $\frac{1}{4}$ -in. tape 5% stretch)			
70°F 50% relative humidity	4.0	5.7	4.7
70°F 90% relative humidity	4.0	5.7	3.0
Coefficient of expansion			
Humidity (per % relative humidity)	$1 \times 10^{-5}$	$1 \times 10^{-5}$	$15 \times 10^{-5}$
Temperature (per °F)	$2 \times 10^{-5}$	$2 \times 10^{-5}$	$3 \times 10^{-5}$
Approximate cost (\$ per inch <sup>2</sup> ; the price per inch <sup>2</sup> is greater for long or wide tapes)	0.0018	0.0022	0.0015
Common lengths (feet)	1200-7200	1200-5000	1200-5000
Coefficient of friction	0.2		
Safe temperature limits (°F)	-40 to 160		
Standard widths (inches)	$\frac{1}{4}$ , $\frac{1}{2}$ , $\frac{5}{8}$ , $\frac{3}{4}$ , 1 (+0.000 in., -0.004 in.)		
Reel size (inches)			
2500 ft.	10 $\frac{1}{2}$		
5000 ft	14		
Coercive force (oersteds)	250		
Remanence (maxwells)	0.7		
Retentivity (gausses)	800		
Approximate number of errors per 2500-ft roll with 100 bits per inch at 60 in. per second with two 50- mil heads per $\frac{1}{4}$ -in. tape	2		

TABLE 6. CHARACTERISTICS OF TYPICAL MAGNETIC HEAD STACK

Gap scatter (inches)	$\pm 0.00005$
Gap azimuth (degrees)	$\pm 0.02$
Gap width (mils)	0.3
Track width (mils)	50
Track spacing (mils)	70
Inductance (millihenries)	10
Resistance (ohms)	15

head is used for reading or recording, wide variations are to be expected from some of the data given.

**Head Design.** Some of the considerations which go into the design of magnetic heads are as follows:

1. The poles of the magnet are arranged in the direction shown in Fig. 19 since higher bit densities can be achieved with the axis of the magnet in this direction.

2. To allow the use of low writing currents and to obtain higher reading voltages the reluctance of the magnetic path should be low. This dictates a short path and use of a magnetic material with high permeability.

3. Low writing currents and high reading voltages may also be obtained by increasing the number of turns. A compromise is necessary in this area, however, since both inductance and mechanical size are increased as the number of turns are increased.

4. The head construction should take into account the frequency response required for digital recording.

5. The magnetic material should have a high resistivity so as to limit losses due to eddy currents.

6. Track width should be small for high bit densities, but a compromise must be established since the output voltage is decreased as the track width is made smaller.

7. Appropriate consideration must be given to the reduction of cross talk between heads in the same headstack.

8. Mechanical design is one of the more critical factors. The tolerances are small, and the design must be extremely compact. In addition the metal surface over which the tape moves must be prepared carefully.

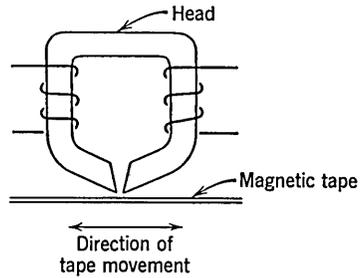


FIG. 19. Magnetic head.

### Methods of Reading and Recording

Various methods exist for recording digital data on a magnetic medium and for extracting these data from the output voltage obtained during a reading operation. A number of these are discussed below. Some of the various characteristics which should be considered in selecting any particular method are as follows: (1) requirements for erasing, (2) requirements for a synchronization track, (3) recording density obtainable, (4) amount of circuitry required, and (5) reliability.

The writing currents for various recording techniques are summarized at the end of this section in Fig. 23.

**Polarized Dipole Method.** For this method, the direction of magnetization of small dipoles is used to differentiate between 0's and 1's. The tape is magnetized by current pulses which may be positive or

negative depending on whether a 1 or a 0 is to be recorded. Figure 20a illustrates the waveform of the writing current and Fig. 20b indicates roughly what the waveform of the reading or output voltage will be if the bits are not packed too closely together. Note that the current in the writing head goes to 0 after each bit is recorded. This is a property of a class of recording methods known as *return to zero* or RZ methods.

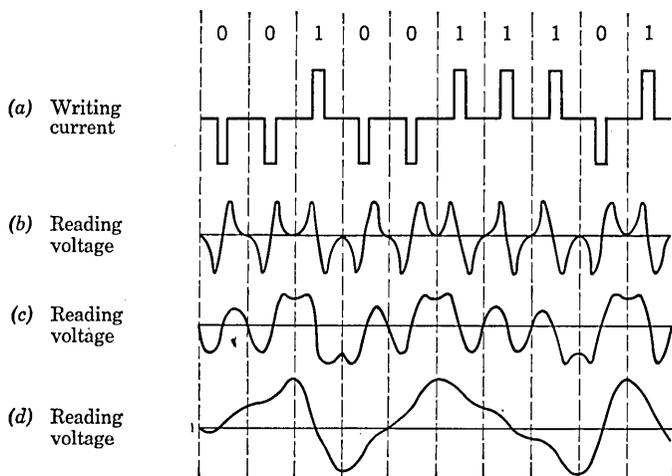


Fig 20. Polarized dipole method. Reading voltages are for different recording densities.

Of the various RZ methods, the polarized dipole method is the most commonly used. Conventional pulse techniques can be used to extract data from the waveform shown in Fig. 20b, and numerous methods are possible. Note that a synchronizing signal is not required to complete the extraction process. Three levels of magnetization are involved, and complete erasure is necessary before recording.

*High-Density Recording.* With increased packing density certain refinements in extraction techniques become more important. Figure 20c and Fig. 20d give some indication of the appearance of the output voltage waveforms which are obtained if the bits are packed closer together. As packing density increases, a sequence of identical bits has a tendency to become one large dipole rather than a series of individual dipoles. For this reason, the magnitude of flux change and, therefore, the magnitude of the output voltage are greatest at points between dissimilar bits and least in the middle of long sequences of identical bits. A common property of all the output waveforms shown is that cells which contain 1's generate a signal which is more positive in the first

half of the cell than in the second half. The opposite is true for cells containing a 0. This fact is frequently used to advantage in extracting data from the output signal. By delaying the output signal and subtracting it from itself, a difference signal is obtained which is positive or negative depending on whether a 1 or a 0 has been stored. This difference signal may be amplified, clipped, and then sampled at the appropriate time to obtain the digital data. This procedure requires a synchronizing pulse to insure that sampling takes place at the proper time. A time delay which corresponds to one-half a cell will usually be appropriate. Note that as the delay decreases the magnitude of the difference signal also decreases and a corresponding increase in amplification is required. As the delay approaches zero, this procedure amounts to taking the derivative of the output signal, and difficulties due to noise may be encountered.

It was previously noted that the magnitude of the output voltage would become small in the middle of long sequences of identical bits. In some cases, therefore, it will be impossible to determine whether the difference signal indicates a 1 or a 0. Since this situation should arise only in a sequence of identical bits, the difficulty may be avoided by providing logical circuitry which changes any indeterminate bit to correspond to its predecessor. Recording densities up to 900 bits per inch have been achieved using the polarized dipole method, but in practice densities greater than 200 bits per inch are rarely used.

**Dipole Method.** When dipoles are recorded on the tape to represent 1's and nothing is recorded for 0's, the recording method may be referred to as the *dipole method*. In its simplest form, this method involves the erasure of the tape prior to use. Erasing may be accomplished by using an alternating current to obtain an unmagnetized surface or by using a direct current which will saturate the medium in a direction opposite to that which will be employed to record the dipole. Figure 23b illustrates what the writing current will be in this case. Note that the current returns to zero after each pulse; hence this is an RZ method. A variation of this method which eliminates the need for erasure prior to use is to maintain a d-c bias in the head at all times during the recording operation. The direction of current is switched momentarily when it is desired to record a dipole representing a 1. This variation is referred to as the *modified dipole method* in Fig. 23c, which shows the writing current used for this approach.

The voltage waveform obtained by using the dipole method is similar to that shown for the polarized dipole method in Fig. 20b, except that no signal is obtained in cells which contain a 0. To detect 0's, therefore, a synchronizing signal is required.

**Non-Return to Zero Method.** The term *non-return to zero*, or NRZ, refers to a characteristic of several recording methods in which the direction of the current is switched in an appropriate fashion to record information, but the magnitude of the current does not remain at zero for an appreciable amount of time. In general, NRZ methods require a synchronizing signal and have heavier recording head duty cycles than

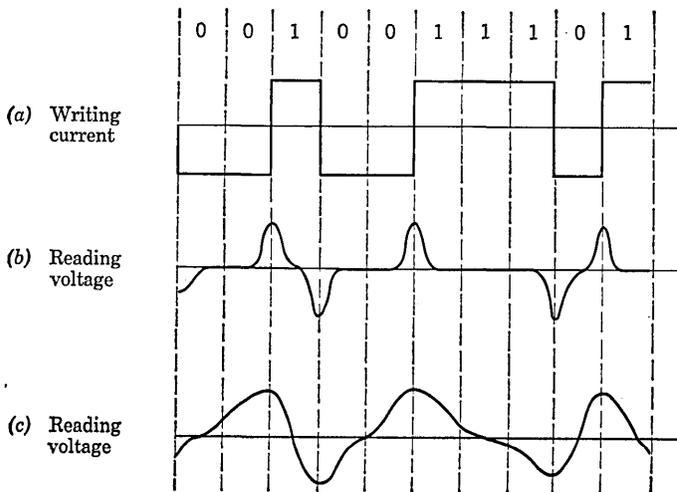


FIG. 21. NRZ method of recording.

the RZ methods. Advantages of the NRZ methods over RZ methods are: (1) the erasure of previous information stored on the tape is more complete and may be conducted during the reading operation, and (2) increased bit densities may be obtained while still using conventional pulse techniques.

The most common NRZ method makes use of the direction of magnetization to determine whether 1's or 0's are recorded. Although the term NRZ refers to a general class of recording techniques, it may also be used in reference to only this particular method. The writing current for this method is depicted in Fig. 21a. The output voltage waveform has been illustrated in Fig. 21b. Conventional pulse techniques may be used to extract the data from this signal, and since the flux changes at a maximum of only once per cell, bit density may be approximately twice that for the polarized dipole method using conventional circuits. Figure 21c illustrates the appearance of the output voltage waveform when the bits are packed closer together. Data extraction in this case may be accomplished in a fashion similar to that discussed for high density

extraction in the polarized dipole method. Note the similarities between the waveforms for both methods at high bit densities. Although high bit densities may be achieved with the NRZ method, bit densities greater than 400 per inch are rarely used in practice.

**Change-on-One or NRZI Method.** A variation of the NRZ method of recording is to change the direction of the writing current each time a 1 is to be recorded. This approach is known as the NRZI method where the I represents the word *invert*. Figure 23e illustrates the writing current for this method. Techniques for data extraction are similar to

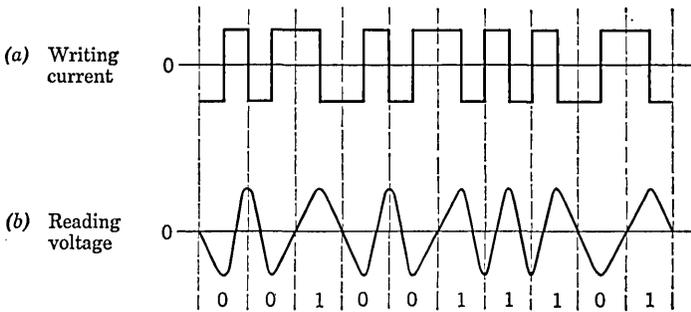


Fig. 22. Manchester recording method.

those discussed previously. If a number of bits are to be recorded simultaneously across the width of the tape, use of this method may be advantageous since a code may be easily devised in such a way that there will be at least one 1 at every character position along the length of the tape. This procedure eliminates the need for a separate timing track.

**Phase Modulation or Manchester Method.** To avoid difficulties experienced when long sequences of identical bits are recorded by the methods discussed previously, the phase modulation method may be employed. With this system a 1 is recorded by sending current through the head in the positive direction for the first half of the digit interval and in the negative direction for the second half. Zeros are recorded in the same manner except that the phase is shifted 180 degrees. Figure 22a illustrates the writing current for this method. Figure 22b gives some indication of the output voltage waveform. Note that the voltage in the center of each digit interval is positive for 1's and negative for 0's. This signal may be amplified, clipped, and then sampled at the appropriate time to obtain the digital data. There are also other means of extracting data from this signal which are similar to those already discussed for previous recording methods. The approach discussed here

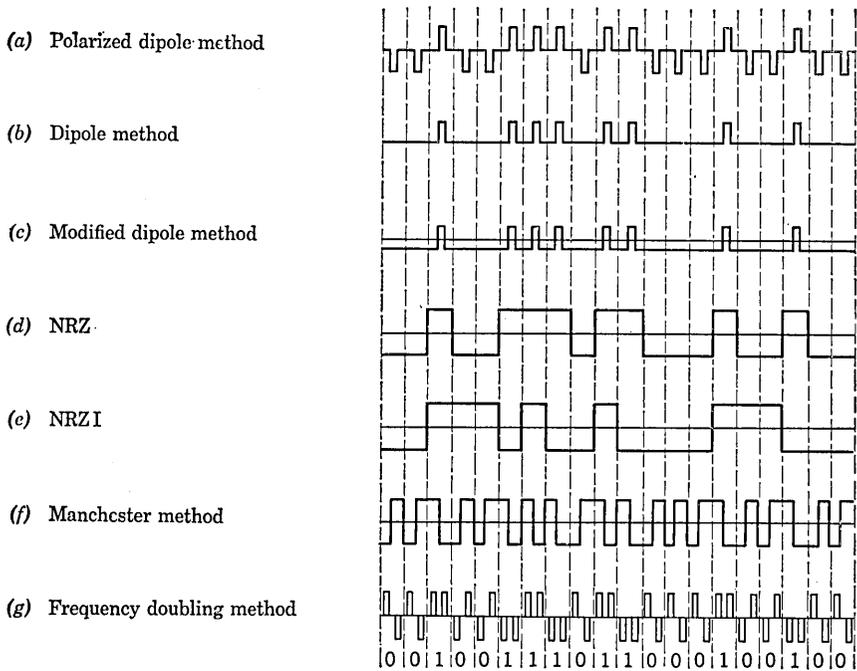


FIG. 23. Writing currents for various recording techniques.

is an NRZ method; a similar technique has been developed which involves phase modulation using RZ recording.

**Frequency Doubling Method.** Figure 23g illustrates the writing current for this method of recording. At high bit densities, where this method is used, the frequency associated with a 0 will be twice that associated with a 1. Through use of appropriate circuitry, data have been extracted from signals for bit densities as high as 1100 per inch.

### Tape Transports

The term tape transport refers to the equipment used to hold the tape and move it past the head. Transports consist of two major units: (1) the tape drive and (2) the tape reel assembly. The drive unit is required for starting and stopping the tape, moving it past the head, and for controlling tape speed. Except for the considerable emphasis on short starting and stopping times mechanical configurations for magnetic tape drives are somewhat similar to those used in photoelectric tape units; the reader is therefore referred to that section for additional information. Magnetic tape reel units are also similar to their counterpart in photoelectric readers; they supply and take up tape as required

by the drive unit. Since it is required to accelerate the tape rapidly during start and stop operations, reel units are designed to maintain slack tape between the reels and tape drive assembly. This allows the reels to accelerate more slowly than the tape and is desirable because of the relatively high inertia of the loaded reels and motors which drive them. The speeds of these motors are controlled by sensing the amount of tape which has been left slack. A number of different sensing techniques have been employed. Use of the follower arms is most common;

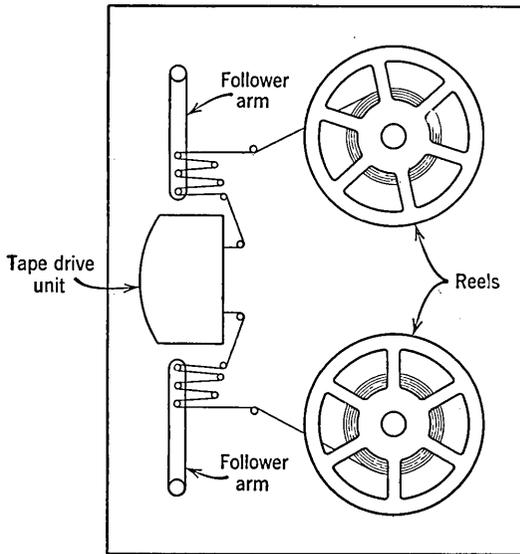


FIG. 24. Follower arm tape transport system.

other methods include pneumatic control, photoelectric sensing, and weight measurement (Ref. 3).

Figure 24 depicts the front panel of a transport system that uses follower arms. The arms are spring loaded and move back and forth maintaining tape tension at about  $1\frac{1}{2}$  ounces. The positions of the follower arms are sensed, and the signals generated are used to control the speed of the reel motors. When sudden changes in tape speed take place, the movement of the arms acts as a buffer, and they either let out or take up slack until the reel motors catch up. Table 7 lists various characteristics of typical commercially available magnetic tape transport systems which employ follower arms.

Figure 25 shows a transport system that employs pneumatic control. The tape is drawn into columns by a vacuum, and pressure-sensitive

TABLE 7. TYPICAL CHARACTERISTICS OF MAGNETIC TAPE TRANSPORTS

Typical tape widths (inches)	$\frac{1}{2}$ , $\frac{5}{8}$ , $\frac{3}{4}$ , 1
Number of tracks	6-20
Typical tape speeds (inches/second) (rewind speeds may be higher)	15, 30, 60, 75, 100, 150
Stop and start time (milliseconds)	5.0-1.5
Tape capacity	2500 ft on 10 $\frac{1}{2}$ -in. reels
Size	2 ft of standard relay rack; depth 1 ft
Weight (pounds)	150
Power (watts)	500
Cost (dollars)	3000-20,000

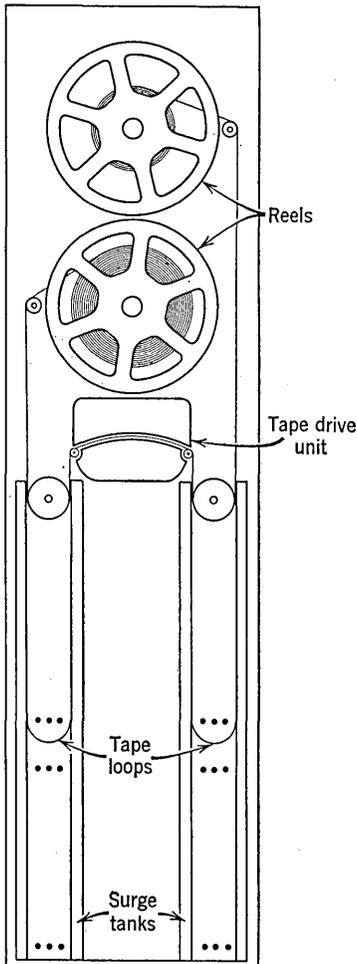


Fig. 25. Pneumatically controlled tape transport system.

switches determine whether the slack tape supply is above or below a predetermined level. These switches are used to control the reel motors and thus regulate the supply of tape. Transports which use pneumatic control are generally larger and heavier than the follower arm variety, but they often have the capability of higher speeds with shorter starting and stopping times.

### Auxiliary Equipment

Auxiliary equipment involving magnetic tape units generally falls into the category of conversion equipment for transferring data to or from the printed page, punched tape, and punched cards. Since the reading and recording speed for magnetic tape units is considerably higher than for the other media, buffer storage is frequently desirable, and the units may be both complex and expensive. There are also commercially available units for direct keyboard input to magnetic tape. These units do not employ a buffer storage, but instead move the tape intermittently once for each character (Ref. 4).

### Magnetic Cards

A recent development in the area of magnetic recording equipment has

been the Magnacard system produced by the Magnavox Company. This equipment makes use of a magnetic card which consists of a Mylar base (0.005 in.) with a ferric oxide coating (0.0007 in.) protected by a Mylar overlay (0.0005 in.). The card measures 1 in. by 3 in., has 17 tracks arranged along the length of the card, and has a capacity of 1000 decimal digits. These cards are arranged in stacks and are transferred to rotating vacuum drums which move the cards to the desired locations. The machines are capable of transferring cards between feed-stack stations, past reading or recording heads, or between drums. Four basic units have been made available.

1. *The Transcriber* handles input-output operations. It consists of two vacuum drums, one feed station, one stack station, a reading head, a recording head, and a drum-to-drum transfer station. Cards are taken from the feed station, passed under the reading and recording heads, and subsequently transferred to the stack station. Data may be recorded on the cards or read from the cards; it is also possible to verify what has been recorded. The cards are handled at a rate of 100 cards per second.

2. *The Collator* accomplishes all card rearrangement functions, including sorting, merging, and selection. It consists of four vacuum drums, four double drum-to-drum transfer stations, five reversible feed-stack stations, two reading heads, one recording head, and two hold stations. The unit is capable of executing its operations under control of data read from the cards. The speed of the various operations which are conducted on this machine depends on the number of passes which are required. A typical decimal sorting operation requires four passes, giving an effective speed of 1500 cards per minute for each digit.

3. *The File Block* is a large capacity magnetic card filing device for use with the collator. It consists of two rectangular arrays of 51 magnetic card trays; each tray contains up to 3000 cards. The unit measures 28 in. by 24 in. by 31 in. In operation the file block is attached to a collator. All positioning and selection of trays is under program control. The tray array within the file block cabinet is positioned horizontally and vertically to the desired tray, which is then automatically pulled into the master feeding station of the collator. Processing of cards in the tray commences as described for the collator. After processing, the tray is automatically reinserted into the file, and positioning of the next tray begins. Positioning, tray extraction, and reinsertion operations require about five seconds for each tray.

4. *The Interrogation File* is a rapid access magnetic card file. It consists of a set of 40 trays, each holding 3000 magnetic cards, and a vacuum drum array which includes four drums, two reversible feed-stack stations, a reading head, a writing head, one hold station, three drum-to-drum

transfer stations, and one special file access station. The trays are arranged with one side open to the drum array to permit access to a segment of 100 cards at a time. All positioning, selection, and processing are under program control. In operation, the file is positioned vertically to a desired tray, and horizontally to a desired index position within the tray. The set of 100 cards is located and extracted, and cards are fed continuously into the vacuum drum array. When a match is found, the feed is stopped and the desired card circulates on a drum for processing. Information can be read, recorded, and verified. When this processing is completed, feed is again started and transfer is reinstated from drum to drum; cards are finally stacked in the original order in the file. The interrogation file also can be used for merging new cards into the file, extraction, and other typical card-handling functions.

## 6. ANALOG-DIGITAL CONVERSION TECHNIQUES

### General Considerations

**Definitions.** An analog to digital (A/D) converter is a device which converts analog quantities or measurements into digital numbers. It may be part of some measuring equipment or a separate unit, and is often referred to as a *coder*. Digital to analog (D/A) converters or *decoders* perform the opposite function. The term *analog-digital conversion* refers to both processes and does not specify the direction in which the transformation takes place.

**Areas of Application.** There are three major classes of systems which require analog-digital conversion equipment as shown in Table 8. In certain of these applications, digital computers play an important role, and the conversion equipment forms a part of the computer's input-output system. In the area of data handling, conversion equipment may be used on-line, but it is more often found associated with off-line equipment. In the other categories, conversion equipment associated with digital computers is usually operated on-line.

**Characteristics of Conversion Equipment.** In evaluating the features and performance of conversion equipment the following characteristics are normally considered.

1. *Form of Analog Data.* This characteristic refers to the type of analog data found associated with the analog-digital conversion equipment. Time, frequency, voltage, and shaft position are the most common quantities, and other forms of analog data are most frequently obtained or derived from these.

2. *Range of Analog Data.* The term range refers to the limits in magnitude between which the analog signal may vary.

3. *Digital Code.* Decimal, straight binary, cyclic binary, and the 8421 code are the most frequently used. The decimal code requires ten lines for each digit, but is the most desirable when data are to be

TABLE 8. APPLICATIONS OF ANALOG-DIGITAL CONVERSION EQUIPMENT

System Application	Inputs	Outputs	Conversion Required	Associated with Digital Computers	Examples
Data handling systems					
Data transmission	A, D	A, D	A/D, D/A	Rarely	Telemetry, PCM
Data logging	A	D	A/D	Occasionally	Process data logging
Data reduction	A	D A display	A/D, D/A	Frequently	Engine tests, wind tunnel, aircraft and missile tests
Digital control systems					
Programmed control	D	A	D/A	Occasionally	Machine tool control, aircraft and missile checkout
Feedback control	A, D	A	A/D, D/A	Frequently	Aircraft flight, missile guidance, chemical processes
Computing systems					
Combined analog-digital computation	A, D	A, D	A/D, D/A	Invariably	Computing installations (see Chap. 30)
Analog computer devices	A, D	A	A/D, D/A	Never	Function generators, multipliers (see Chap. 23)

displayed. The 8421 code is next most desirable for data display, and the straight binary code is the most suitable one for use as a digital computer input. Table 9 provides a comparison of these codes for numbers up to fifteen. For additional information concerning codes see Chap. 12.

4. *Number of Bits.* The precision of conversion equipment is largely determined by the number of bits (binary digits) generated or used.

5. *Quantization Error.* With A/D converters the most precise measurement which can be made corresponds to the least significant bit. Error in measurement due to this limited precision is frequently referred to as quantization error.

6. *Accuracy.* This term refers to the correctness of the measurements rather than to their precision. Since some conversion equipment has a

TABLE 9. COMMON DIGITAL CODES EMPLOYED WITH ANALOG-DIGITAL CONVERSION EQUIPMENT

Decimal	Straight Binary	Cyclic Binary	8421 Code
0	0000	0000	0000 0000
1	0001	0001	0000 0001
2	0010	0011	0000 0010
3	0011	0010	0000 0011
4	0100	0110	0000 0100
5	0101	0111	0000 0101
6	0110	0101	0000 0110
7	0111	0100	0000 0111
8	1000	1100	0000 1000
9	1001	1101	0000 1001
10	1010	1111	0001 0000
11	1011	1110	0001 0001
12	1100	1010	0001 0010
13	1101	1011	0001 0011
14	1110	1001	0001 0100
15	1111	1000	0001 0101

tendency to drift, it is often important to speak of accuracy over a given period of time without adjustment or calibration.

7. *Holding Characteristics.* Frequently it is necessary to hold the input signal to an A/D converter constant during the time conversion takes place. Requirements for *holding* and the manner in which it may be accomplished are referred to as the holding characteristics of a coder. In the case of a decoder, it may be necessary to store the digital data or to insure that digital data are transferred to the decoder at periodic intervals; this is in order to properly *hold* or maintain the analog output signal. In its simplest form a holding device for a decoder amounts to a binary register and converts the impulse outputs of the computer to step functions.

8. *Number of Channels.* Depending on the nature of the conversion equipment, it may be possible to time share various portions of the converter with a number of inputs or outputs. This process is referred to as *multiplexing*. Certain types of conversion equipment lend themselves to a multiplexing operation better than others.

9. *Conversion Time.* The time required to complete one decoding or coding process is referred to as conversion time.

10. *Conversion Rate.* This term refers to the number of conversions which may be accomplished during a unit interval of time. The conversion rate of some coders varies according to certain characteristics of the input analog signal.

11. *Total Value and Incremental Methods.* Certain types of analog-

digital conversion equipment obtain a complete new value at each conversion whereas other types change their output only in small incremental steps.

12. *Feedback and Nonfeedback Methods.* Some types of analog-digital conversion equipment conduct the actual conversion on the output signal rather than on the input. The data obtained in this fashion are compared with the input, and an error signal is developed. This error signal is used to adjust the output and thus reduce itself to zero. Nonfeedback methods are straightforward and do not involve comparison.

13. *Direct and Indirect Conversion.* Indirect conversion is used here to imply that the analog data are converted to another analog form before use. Such conversion is frequently employed when it is necessary to obtain or measure analog quantities other than time, frequency, voltage, or shaft position. In this case, however, the analog-to-analog equipment is often completely dissociated from the analog-digital conversion equipment, and thus is not discussed in this chapter. In some cases where indirect methods are employed, the analog-to-analog conversion involves the quantities listed above and is directly associated with the analog-digital conversion equipment. Examples of these particular indirect conversion methods are given later.

Requirements on conversion equipment from the point of view of the characteristics listed above are determined to a considerable extent by the analog portion of the system and are therefore not discussed here. For a more overall treatment of systems which employ both analog and digital quantities see Chaps. 28, 29, and 30 and Vol. I, Chap. 26.

The remaining portions of this section include various examples of analog-digital conversion techniques. No attempt has been made to list all methods which have been developed, but those given should serve to illustrate the most common techniques presently employed. Since the primary concern here is with input-output equipment for digital computers, the examples given are almost all straight binary code devices.

### **Analog to Digital Conversion**

**Time Interval.** An example of a common method for converting the duration of time between two pulses to a binary number is illustrated in Fig. 26. With the flip-flop and binary counter initially set to 0, the first timing pulse will set the flip-flop to the 1 position and start the counting cycle. The second pulse will reset the flip-flop and stop the counting cycle. Thus an appropriate binary number will be left in the counter. After the number is sampled by the computer, the counter is reset to 0 in preparation for taking the next measurement. Elimination of the flip-flop makes it possible to measure the time duration of the received

pulses rather than the time interval between pulses. Increased precision of measurement may be obtained by increasing the frequency of the oscillator. Above 10 megacycles, however, counting is somewhat impractical and vernier techniques, which do not require high counting rates, are resorted to for more precise measurements (Ref. 5).

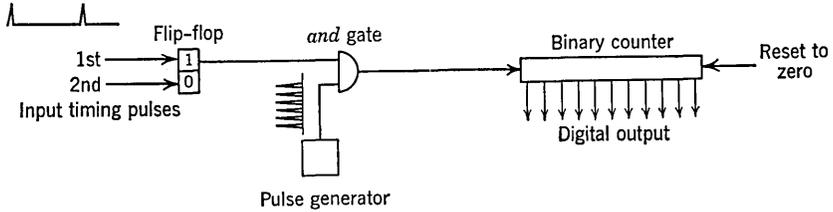


FIG. 26. A/D time conversion.

An indirect method of converting time can be developed with a ramp generator. The voltage obtained from the generator may be converted to a digital number as described later.

**Frequency.** A method for obtaining a digital representation of frequency is illustrated in Fig. 27. An oscillator with a frequency lower than that being measured is used to establish a fixed time interval

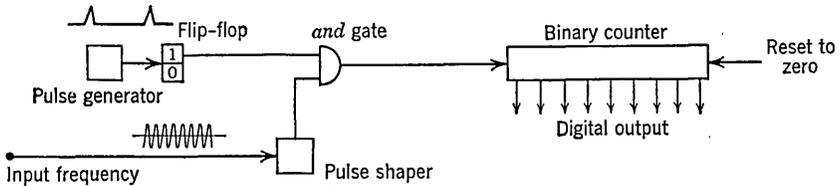


FIG. 27. A/D frequency conversion.

between two pulses. These pulses gate the input signal to be measured into a binary counter for this fixed amount of time. The number obtained in the counter will be a measure of the frequency.

An indirect method of converting frequency is to establish a time interval by counting a fixed number of input cycles. This time interval may then be converted to a digital number as previously described.

**Voltage.** Many voltage-to-digital converters make use of feedback techniques discussed later. There are, however, a number of straight-forward techniques.

**Cathode Ray Tube.** A diagram of a cathode ray tube which may be used for voltage A/D conversion is shown in Fig. 28 (Ref. 6). The voltage to be converted is applied to the vertical deflection plates. When a readout is desired, a sweep voltage is applied to the horizontal deflec-

tion plates. As the beam sweeps across the face of the tube, it will strike the output plate only if it is directed at an opening in the aperture plate. Brief study of the aperture plate depicted in Fig. 28 will show that if the vertical deflection remains constant during a sweep, the signal

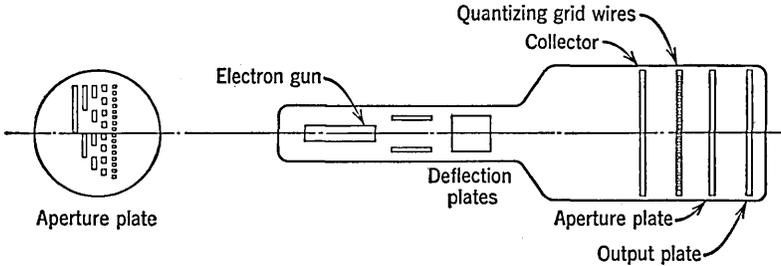


FIG. 28. Cathode ray tube for A/D voltage conversion.

which appears on the output plate will be a serial straight binary representation of the input voltage. The quantizing grid wires and collector are used to generate a signal which is fed back to the vertical deflection plates in order to insure that the vertical deflection of the beam remains constant and in the center of quantum level during the time it is swept across the aperture plate. Since this circuit is made inoperative after completing a sweep, the input voltage is free to move the electron beam to any other quantum level before each new measurement is made. This conversion method is particularly suited to applications where short conversion times on the order of 10 microseconds are required. The cyclic binary code has also been used with this method (Ref. 7).

**Ramp Method.** The ramp method is an indirect voltage-to-digital conversion technique in that voltage is first converted to a time interval and then to digital form. A simplified block diagram for such a scheme is illustrated in Fig. 29. The upper portion of the diagram is identical with Fig. 26 and has been discussed previously as a method for converting from time to digital. In the lower portion of the diagram there is equipment for generating a ramp function which is compared with the input analog signal to be converted. Upon detection of coincidence between the input signal and ramp function, the coincidence detector emits a pulse which is used to stop the counting cycle. The binary number left in the counter thus represents the magnitude of the input voltage at the time the counting cycle was completed. After the computer samples the binary register, both the register and the ramp generator are reset to 0 in preparation for taking the next measurement.

A variation of this approach is to generate a ramp type function which

consists of a series of small step functions rather than a smooth continuous slope. The rate at which the function increases is controlled by the oscillator, and one incremental step is made in the ramp each time

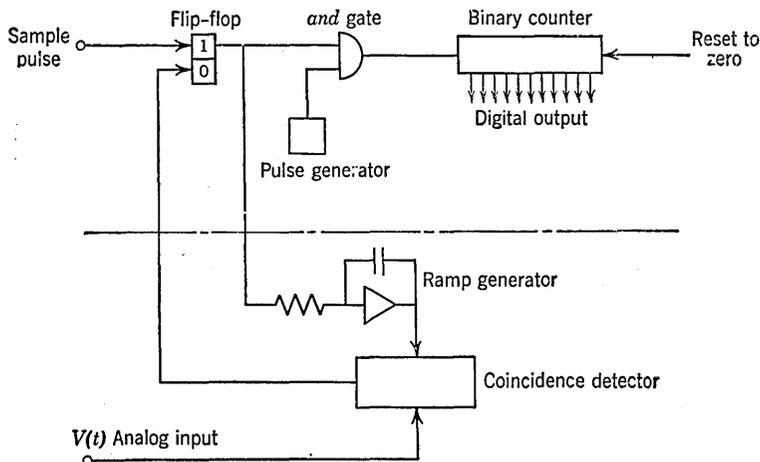


FIG. 29. Ramp method for A/D voltage conversion.

the counter is pulsed. In this fashion the requirement for correlation between the slope of the ramp and the oscillator frequency may be eliminated. This technique is almost logically equivalent to the continuous balance feedback method discussed later.

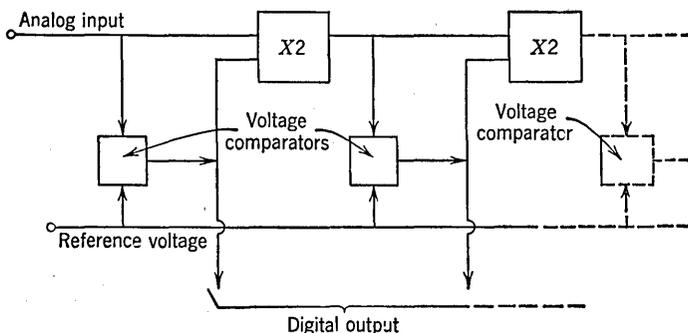


FIG. 30. Cascaded stages voltage-to-digital conversion.

**Cascaded Stages.** Figure 30 illustrates a method for converting voltage to a digital number by successive comparisons and subtractions (Ref. 8). At each stage the input voltage is compared with the reference voltage and then on basis of the comparison either a 0 or a 1 is read out.

If a 1 is read out, the reference voltage is subtracted from the input voltage; the result is multiplied by two and fed to the next stage. If a 0 is read; only the multiplication by two takes place.

**Shaft Position.** Three major classes of conversion methods are associated with A/D shaft position converters: (1) coded pattern methods, (2) incremental pattern methods, and (3) indirect conversion methods involving voltage or time. Although the discussion here is limited to the shaft position type converter, devices which employ techniques similar to those discussed have been developed for other types of mechanical motion such as translation.

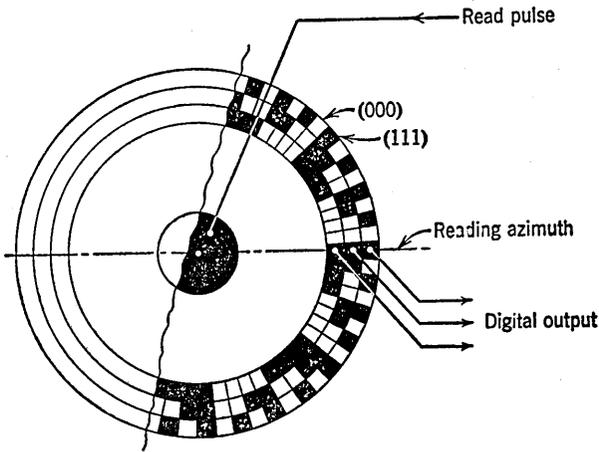


FIG. 31. Straight binary code wheel.

**Coded Pattern Methods.** The most common shaft position to digital converters make use of a code wheel and direct reading techniques. Reading is accomplished by using either mechanical brushes or photoelectric cells. The code wheel is designed to give a digital output, which is a function of the angular position of the shaft. The most common function, of course, is a linear representation of the angular position; others available include sine or cosine of the angular position. The digital output may be in one of several codes; straight binary, binary coded decimal, and cyclic binary are common. Figure 31 illustrates a straight binary coded wheel. Only the three least significant bits are shown. The black areas represent interconnected conducting material and the small dots represent brushes. The read pulse will appear on any of the output lines that are resting on the conducting material. Photoelectric cells and a light source can be used with an aperture in a somewhat similar arrangement.

**Ambiguities.** A problem associated with coded pattern devices is the ambiguous reading which may result when two or more bits are required to change at the same time. For example, if the wheel in Fig. 31 were reading between  $(000)_2$  and  $(111)_2$ , any other number between these two would be obtainable by offsetting the brushes or conducting material only a small amount. There are several methods of preventing this ambiguous representation of binary numbers, and these are discussed below.

**Cyclic Binary Codes.** Use of the cyclic binary code is one popular method. Such a wheel has been illustrated in Fig. 32. This figure, or a study of Table 9, will show that no more than one bit changes at any given time. Thus it is ensured that the digital number obtained is never off from the correct value by more than one count. Logical circuitry for converting from cyclic binary to straight binary code is also shown in Fig. 32. Rules for this code conversion are given in Chapter 18, Sect. 3.

*Dual Brush Method.* Figure 33 illustrates another antiambiguity scheme which alleviates this difficulty by allowing only one of two numbers to be read from the wheel as it turns from one number to another. Its operation may be understood by realizing that so long as the least significant brush is on the black or conducting material an improper reading will not result if the remaining brushes are displaced from the reading azimuth by a small angle in a clockwise direction. Likewise, so long as the least significant brush is on the white or nonconducting material, an improper reading will not result if the remaining brushes are displaced from the reading azimuth by a small angle in a counterclockwise direction. A study of the logic will show that this in effect is what is accomplished. By doing this it can be seen that:

1. Switching for all bits takes place simultaneously, and therefore ambiguous results are not obtained.
2. The exact position of the wheel at which switching takes place is determined solely by the outside, least significant track.
3. Misalignment of the least significant brush results in only a constant bias error.
4. Misalignment of the other brushes by a small amount does not affect what is read.
5. Except for the least significant track, the accuracies of the tracks are not critical.
6. Except for the least significant track, all switching takes place after the brush has made contact.
7. The track which is critical is on the outside where more accuracy is possible.

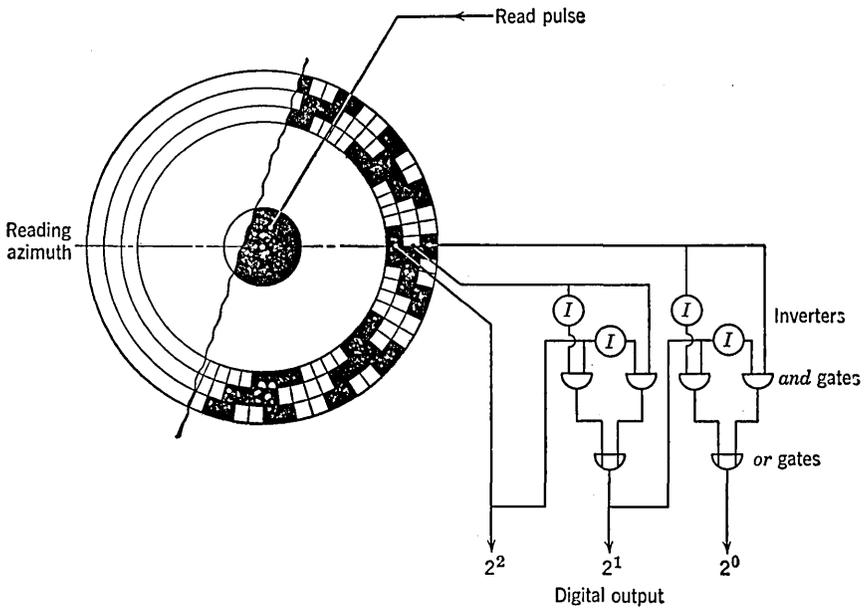


FIG. 32. Cyclic binary code wheel.

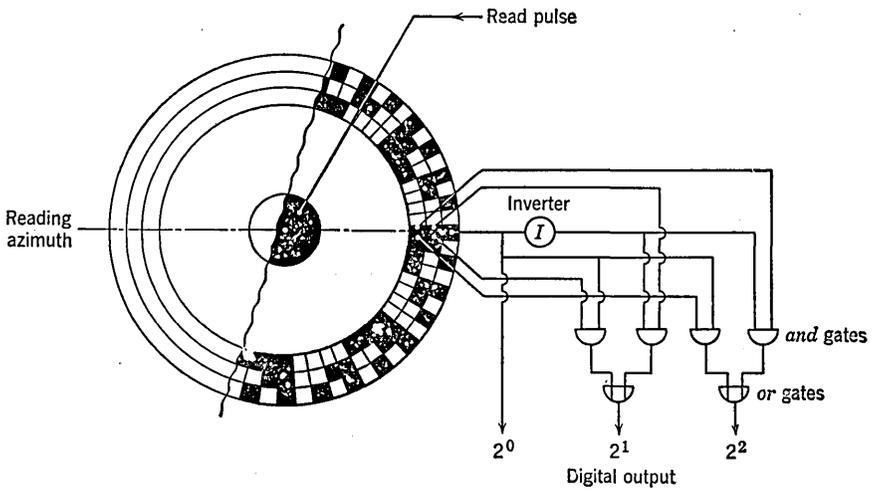


FIG. 33. Dual brush method.

*V Brush Method.* Figure 34 illustrates a variation of the previous scheme and is referred to as the V brush method. Here the particular brushes which are used are not determined solely by the least significant brush. Instead the particular brush used is dependent on what is read from the next least significant bit. However, the operation may be under-

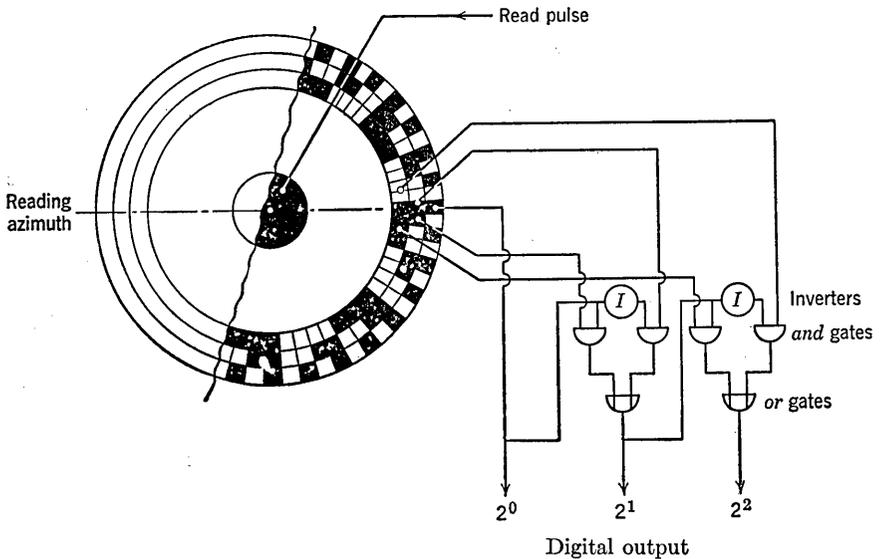


FIG. 34. V brush method.

stood in the same way as the previous scheme, and the list of advantages all apply; an additional advantage is that the placing of the brushes in the more significant positions is even less critical than before. This is desirable of course, since the circumference of the tracks gets smaller as they approach the center of the wheel. Note that the logic required for brush selection is similar to that shown for cyclic to binary conversion in Fig. 32.

A disadvantage of the above antiambiguity schemes is that external logical circuitry is required. This is not acute if a number of converters are to be used, since this logical circuitry may be time shared and little cost will result from an overall point of view. Figure 35 illustrates a converter which, for the most part, avoids the necessity for external logic by accomplishing most of the required logical functions on the wheel itself. The elimination of external logic is paid for by increasing the number of brushes and complexity of the wheel. This method is referred to as the *self-switching V brush method*.

**Multispeed Coders.** In order to increase the precision of measurement without incurring problems associated with providing numerous tracks on a single wheel, many code wheel converters employ a second wheel which is coupled to the first through a reduction gear train. These units are referred to as multispeed coders. Although considerable care must be exercised in coupling the input shaft to the converter, no serious difficulty arises in the design of the gear train between the two wheels.

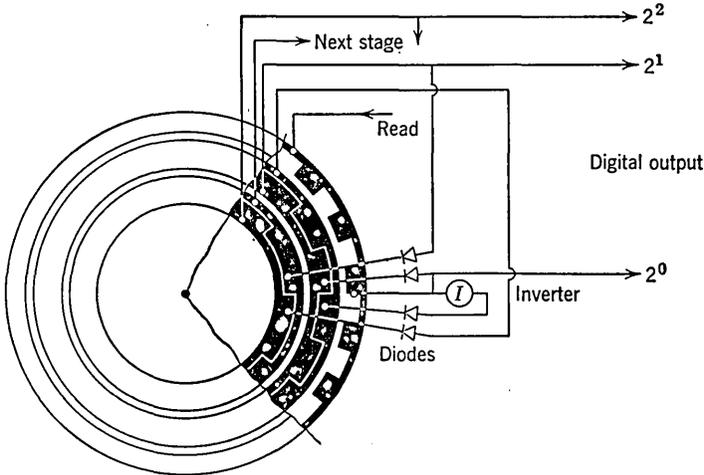


FIG. 35. Self-switching V brush method.

This is because various antiambiguity schemes may be employed to eliminate the effects of backlash etc. The V brush method appears to be the most practical approach. For converters which employ the cyclic code, it is possible to eliminate improper reading by providing redundant tracks on the two wheels and logical circuitry which makes the necessary corrections.

**Typical Code Wheel Converters.** Table 10 provides data on typical commercially available code wheel converters. Some remarks are in order concerning what has been presented. First it will be noted that photoelectric code wheels normally employ the cyclic binary code. This is because the other antiambiguity methods do not lend themselves particularly well to an optical mechanization. Secondly it may be inferred from the figures for the number of bits and counts per revolution that the photoelectric code wheel converters employ a single wheel whereas brush type devices are often multispeed converters. The readout rate for photoelectric coders is usually limited by the flash lamp. Continuous

TABLE 10. SOME TYPICAL CHARACTERISTICS OF COMMERCIALY AVAILABLE CODE WHEEL CONVERTERS

Digital code	Mechanical Brush		Photoelectric
	Heavy Duty	Light Duty	
	Binary, binary coded decimal, decimal	Binary, binary coded decimal, cyclic binary	Cyclic binary
Readout	Linear	Linear	Linear, sine, cosine
Number of bits	10-15	7-19	13-18
Counts per revolution	10-64	128-256	$2^{13}$ - $2^{18}$
Maximum readout rate (per second)	2	$10^6$	100(w/flash lamp)
Maximum rotation speed (rpm)	1800-12,000	200-1500	50-1200
Operating torque (inch-ounce)	0.1-0.8	0.2-0.8	0.1-0.2
Contact rating (amperes)	$\frac{1}{4}$ -1	0.002-0.020	—
Housing length (inches)	3-6	1-5	3-7
Housing diameter (inches)	2-3	$1\frac{1}{2}$ -3	4-15
Weight (pounds)	2	$\frac{1}{4}$ -2	1-4
Cost	\$200-700	\$200-700	\$4500

lighting may be used, but this complicates the sensing circuitry since the light-dark adjustments are more critical. The readout rate for heavy-duty coders is shown considerably lower than that for the rest because frequently the brushes are lifted on these devices while the wheel is in motion, and time is required to move them back into position. For this reason, however, much higher operating speeds are possible. The speeds shown for the other devices are given on the assumption that they are read on the fly. However, the figure of 1500 rpm would be applicable for only short periods of time, and 200 rpm would be closer to the upper limit for continuous operation without undue brush wear. In this regard it should be remembered that if it is desirable to gear up to the converter in order to allow the full digital range to correspond with one revolution of the input shaft, the maximum speed of the input shaft must be even lower.

**Incremental Pattern Methods.** Converters in this class of position to digital devices generate a signal each time the position is changed by an incremental amount. A digital representation of the position is obtained by summing these incremental signals. An example of such a device is illustrated in Fig. 36. Photoelectric cells sense the movement of the slotted disk, and the interpreting circuit determines whether the number in the counter should be increased or decreased by one increment. Note that once an error occurs with this configuration, the readout will remain incorrect until another compensating error is made. Some incremental pattern devices limit this possibility by providing error correcting

circuitry. Although photoelectric cells have been shown in the diagram, other means are also available for sensing movement. Magnetic recording techniques and commutator-brush arrangements have been frequently utilized in this connection.

Note that incremental pattern devices may also be used for measuring speed. The frequency of the incremental signals may be obtained in digital form by a method previously discussed and thus will provide appropriate digital rate data.

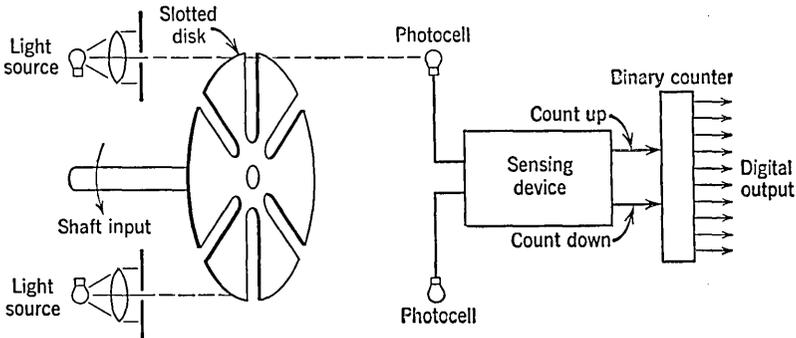


FIG. 36. Incremental pattern device.

**Indirect Methods.** A typical indirect method for shaft to digital conversion involves an intermediate conversion of shaft position to voltage and the subsequent conversion of the voltage signal to digital form by means previously discussed. Perhaps the simplest method of effecting the intermediate conversion is through the use of a potentiometer mounted directly to the shaft which is to be measured.

Another common indirect conversion method involves an intermediate conversion of shaft position to time in the form of phase angle between two a-c voltages. Use of synchros is one popular means of obtaining the required phase shifted signals.

**Digital to Analog Conversion**

**Time Interval.** One of the most common approaches taken for generating a time interval which corresponds to the magnitude of a binary number is illustrated in Fig. 37. A start pulse that is received at the beginning of each cycle forms the first output pulse and is used to gate the complement of digital word to be converted into the binary counter. In addition, it sets the flip-flop to the 1 position and thereby enables the *and* gate. The resultant train of pulses which pass through the *and* gate increases the stored number by one count for each pulse

transmitted. A stop pulse is generated by the counter when it overflows. This stop pulse forms the second output pulse and is used to reset the flip-flop to 0 and thereby disables the *and* gate. The conversion unit then waits idle until the next start pulse is received. The signal at  $x$  can also be used as an analog output since the duration of the pulse which appears at this point corresponds to the binary number being converted.

The precision obtained with this method is limited by the counting rate. Rather than using frequencies above 10 megacycles for greater precision, it is usually better to employ the least significant bits to gate short delay lines in or out of the output line.

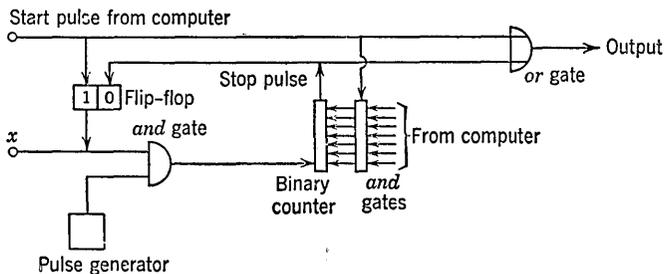


FIG. 37. D/A time conversion.

**Voltage.** Two general D/A methods for obtaining voltage are considered here: (1) resistance network methods and (2) RC network methods. The first of these provides the greater accuracy while the second approach will usually lead to a configuration which requires the least amount of circuitry.

**Resistance Network Methods.** This general category of methods involves the gating of constant voltage or constant current sources into appropriate points of a resistance network or the modification of the values of resistances in a network so as to obtain an output voltage which is proportional to the number being converted. Two examples are illustrated. Figure 38 shows a *current summation* method that makes use of flip-flops which deliver zero current when in the *zero* state and some fixed current,  $I$ , when in the *one* state. Similar circuits are easily developed for constant voltage sources, but in general use of current sources entails fewer circuit design problems when electronic switching is required. If requirements on conversion rate are not severe, relays or other electromechanical switching devices can be used to good advantage. Figure 39 illustrates an appropriate circuit in this case. Study of the diagram will show that it functions as a *digital potentiometer*. Numerous other network configurations are possible, and those given here should

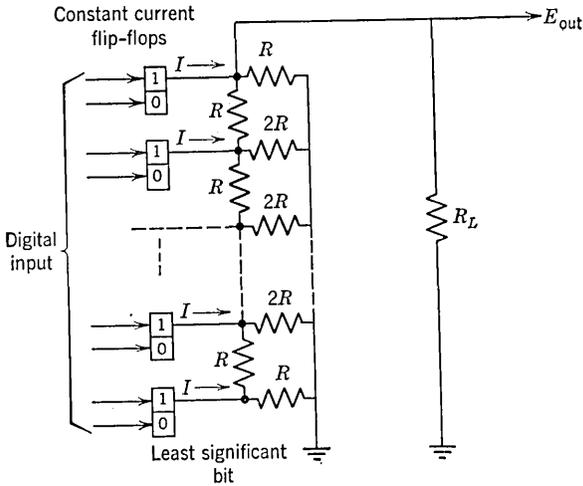


FIG. 38. Current summation digital-to-voltage conversion.

$$E_{out} = \frac{IR}{3} \left[ \frac{R_L}{R_L + \frac{3R}{2}} \right] \frac{N}{2^{n-2}}$$

where  $n$  = number of bits

$N$  = magnitude of the binary number  $0 \leq N \leq 2^n - 1$

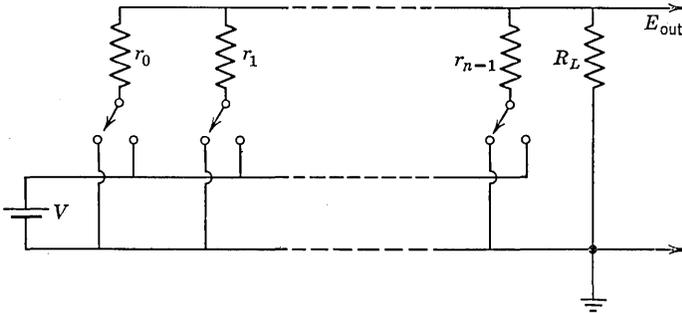


FIG. 39. Digital potentiometer-digital-to-voltage conversion.

$$r_k = r_0 2^{-k}, \quad k = 0, 1, 2, \dots, n - 1$$

$$E_{out} = \left[ \frac{V}{(2^n - 1) + \frac{r_0}{R_L}} \right] N$$

where  $n$  = number of bits

$N$  = magnitude of the binary number  $0 \leq N \leq 2^n - 1$

All switches are shown in the 0 position.

only serve to illustrate a general approach. Often the output of these networks is buffered by using a d-c operational amplifier.

**RC Network Methods.** Figure 40 shows a simple *RC* network which can be used for D/A voltage conversion. The flip-flop shown delivers either zero or some fixed value of current depending on its state and is driven by a pulse train which corresponds to the binary number being converted. Numerous methods are available for generating an appropriate sequence of pulses, and the particular method chosen usually depends to a large extent upon holding requirements.

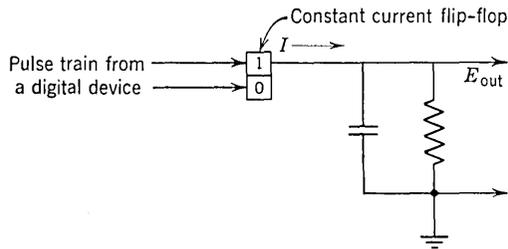


Fig. 40. *RC* network digital-to-voltage conversion.

**Shaft Position.** There are three general classifications of methods for converting a digital number to a shaft position: (1) use of quantized motor devices, (2) feedback methods such as those discussed in the next section, and (3) indirect methods. A typical example of the third category has been provided in the discussion of plotters where a digital number is first converted to a voltage and then, through the use of standard servo techniques, into position.

In the first category, a train of pulses is usually generated which corresponds to the desired incremental change in position. This pulse train can be fed to any one of a number of devices such as a stepping motor or rotary solenoid.

### Feedback or Comparison Techniques

**General Method.** All the various analog-digital conversion techniques which have been described thus far have been straightforward in the sense that signals were transformed *into* the desired type of quantity. A second class of techniques involves feedback methods in which actual conversion takes place in the direction opposite to that required. Figure 41 illustrates block diagrams for two such methods, Fig. 41a for A/D conversion and Fig. 41b for D/A conversion. In both these cases a wide latitude of possible circuits is available and nearly all the techniques

discussed thus far could be incorporated in one of these two schemes. A few of the more practical configurations are discussed below.

**D/A Shaft Conversion.** A method for converting a digital number to a shaft position is shown in Fig. 42. Here a number which has been

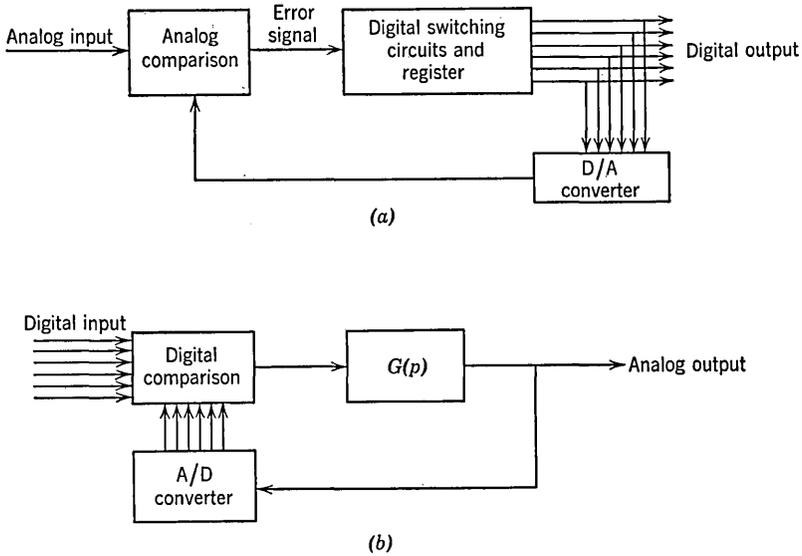


Fig. 41. General feedback method: (a) for A/D conversion, (b) for D/A conversion.

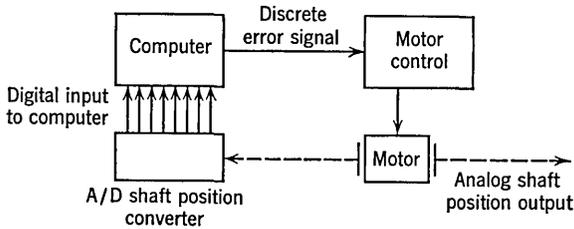


Fig. 42. D/A shaft conversion using feedback.

generated during computation by the computer represents the required shaft position. The computer samples the number in the shaft position converter and then, on the basis of a comparison of the two digital numbers, issues a discrete error signal that drives the motor toward the correct shaft position. This procedure would be used in situations where very accurate positioning was required.

**A/D Voltage Conversion.** The two methods illustrated in Fig. 43 and Fig. 44 employ a feedback technique which is referred to as the *continuous balance method*. In Fig. 43 the digital-to-voltage portion

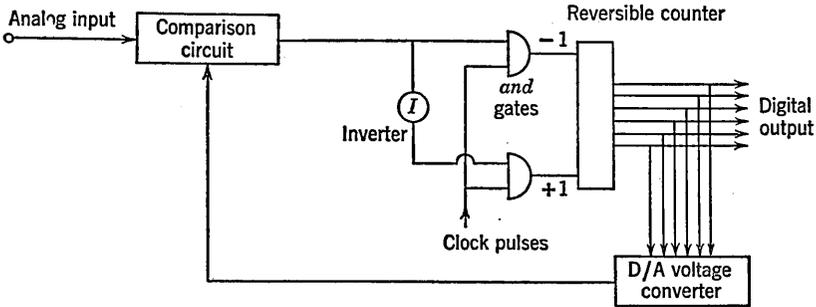


Fig. 43. Continuous balance voltage-to-digital conversion.

of the circuit has not been specified, but any method that is sufficiently accurate and fast might be used. The output of the D/A converter is compared with the analog input signal. The resultant error signal

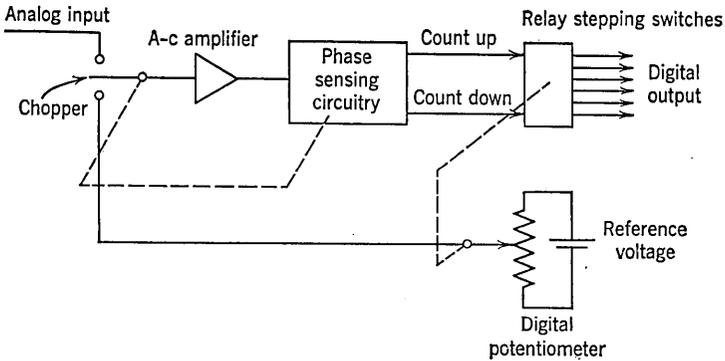


Fig. 44. Electromechanical continuous balance voltage-to-digital conversion.

is clamped and used to control the direction in which the counter is stepped. Figure 44 illustrates an electromechanical converter which is typical of those used in digital voltmeters. A study of the diagram will show that it is logically equivalent to the one previously discussed.

If either of the two methods discussed above have  $n$  bits in the output, then  $2^n - 1$  steps are required to cover the entire range. For this reason another feedback method is more appropriate when speed is important and  $n$  must be large.

**Successive Approximation Method.** This second method is referred to as the successive approximation method and although it makes a complete new measurement each time digital data is sampled, it requires only  $n$  steps to complete the measurement. Figure 45 illustrates a simplified version of such a converter. The diagram is the same as that shown in Fig. 45 except that the binary register is adjusted in a different fashion. At the beginning of the sampling interval each flip-flop is set to zero by a clock pulse. The numbered circles represent consecutive

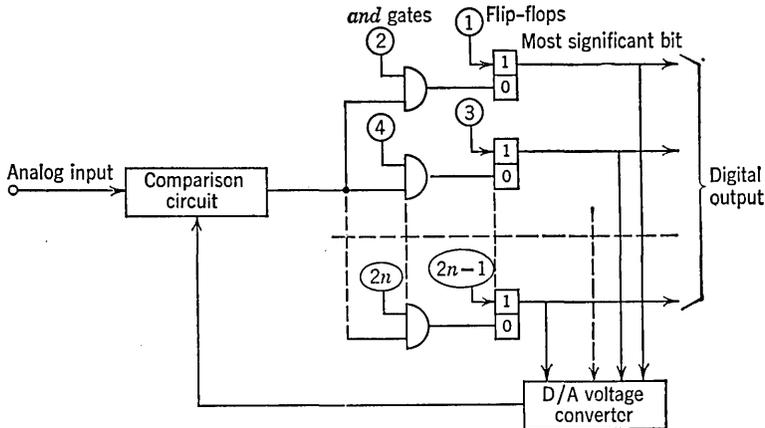


FIG. 45. Successive approximation voltage-to-digital conversion.

clock pulses following the original one and may be generated by a computer or an oscillator and a ring counter. The operation is as follows. The most significant digit is switched on at time 1. If this is larger than the input voltage, it is switched off; if smaller, it remains on. The same procedure is repeated at times 2, 3, ...,  $2n - 1, 2n$ . It should be noted that the circuit requirements may be simplified by allowing pulses 2 and 3, 4 and 5, etc., to occur simultaneously. This converter lends itself well to time-sharing techniques and only a minimum of control circuitry is required to switch from one analog input to another.

**Plotters**

**Requirements.** Frequently the results obtained from digital computer runs are so voluminous that considerable effort is required on the part of the user to make practical use of the data. Editing by programming is a partial solution to this problem, but often it is difficult, if not impossible, to program in the required "judgment." Another solution to the rapid assimilation of data by the user is through the use

of display devices such as plotters. This approach provides a means whereby the user may intelligently and rapidly accomplish his own editing. The discussion which follows concerns the use of plotters as digital computer output devices and gives some indication of the D/A equipment normally associated with them in this application.

**Continuous and Discrete Plots.** Both continuous and discrete plotting are often possible with commercial plotters. Choice of the most appropriate mode is to a large extent determined by the number of points to be plotted and the distance between them. In general, the points are close together when there is a great number to plot. As the

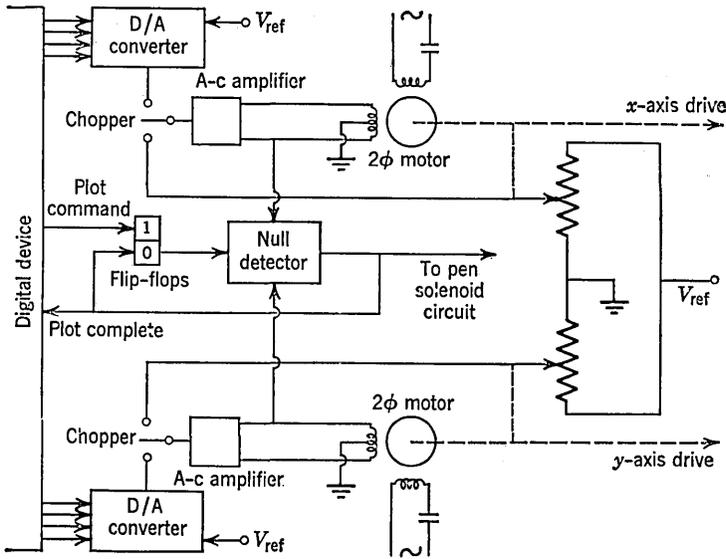


FIG. 46. Circuitry for discrete plotting.

number of points increases, the importance of plotting speed also tends to increase. This is compatible with plotter characteristics since speed of plotting increases as the distance between points decreases. Discrete plotting is the most desirable in the sense that it presents a more intrinsic representation of digital data. This is particularly true when the points are far apart since the trace between points provided by a continuous plot has little significance other than the response of the plotter to a unit step function. On the other hand, if speed of plotting is important, continuous plotting is more appropriate since there is no requirement to actuate the pen each time a point is plotted and data may therefore be accepted at higher rates. The trace obtained between points in this

case is usually insignificant since the points are likely to be close together.

**Plotter Operation.** A simplified block diagram of plotter used to plot discrete points is illustrated in Fig. 46. By removing the null detector and flip-flop, the plotter circuit would be suitable for continuous plotting. The digital device represented on the left may be a computer or some item of off-line equipment. The D/A converters are generally of the resistance network variety discussed previously. All-electronic converters are required for high-speed plotting whereas electromechanical converters are satisfactory for slow speed plotting.

TABLE 11. TYPICAL CHARACTERISTICS OF COMMERCIAL PLOTTERS

	Table or Rack Mounted	Console
Plotting board size (inches)	$8\frac{1}{2} \times 11-11 \times 16\frac{1}{2}$	$30 \times 30$
Slewing speed	20 in./sec	20 in./sec
Maximum plotting speeds	5-12 in./sec	9 in./sec
Static accuracy, including point plotting	.075-0.25% full scale	0.05% full scale
Dynamic accuracy, at normal plotting speeds	0.1-0.5% full scale	0.10% full scale
Maximum plotting rate	1-10 points/sec	1-10 points/sec
Pen or arm acceleration (maximum)	100-750 in./sec <sup>2</sup>	100-350 in./sec <sup>2</sup>
Sensitivity	0.5 mv/in.-10 volts/in.	50 mv/in.-10 volts/in.
Paper hold down	Mech. or vacuum-mech.	Vacuum-mech.
Size (inches)	$24 \times 19 \times 10$	$50 \times 45 \times 40$
Weight	25-75 lb	500-1000 lb
Power requirements	60-300 watts	500-2000 watts
Cost	\$2000	\$10,000

The outputs of the D/A converters are compared with voltages which represent the position of the pen. If the D/A converters are of the digital potentiometer variety, these comparison circuits in effect form bridge type null detectors. The magnitude of the a-c voltages generated by the choppers represents the error, and the phase of these voltages represents the direction or sign of the error. After the a-c voltages have been amplified, they are fed to phase sensitive servo motors which drive the pen toward the correct position. Some commercially available plotters have tachometers mounted on the drive shafts. The signals generated by these tachometers are fed back to the amplifiers for damping purposes. Others make use of RC networks to obtain damping. Just after the digital words have been inserted into the D/A converters, the digital device shown on the left sends a *plot command* pulse. This pulse sets the flip-flop and thereby enables the null detector. When the pen

has reached the correct position on both axes the null detector generates a *plot complete* signal which causes the pen to plot a point and signals the digital device that the plotter is ready to accept the next set of data. In addition, this signal resets the flip-flop and thus, in effect, turns itself off in preparation for plotting the next point. An a-c amplifier and chopper have been shown in the diagram since this mechanization is characteristic of commercially available plotters. When D/A converters are of the digital potentiometer type, the choppers may be eliminated by using an a-c voltage source in lieu of the d-c source shown. Many variations of the scheme shown here are possible, and a number of them are used in commercial equipment. Table 11 lists various characteristics of commercially available plotters together with representative data. The discussion here has been limited to the *x-y* type plotter; strip charts and printing devices may also be used for plotting digital data.

## REFERENCES

1. H. M. Smith, The typotron, a novel character display storage tube, *I.R.E. Convention Record*, Pt. 4, 129-134, March 21-24, 1955.
2. H. Epstein and F. Innes, The electrographic recording technique, *I.R.E. Convention Record*, Pt. 4, 135-138, March 21-24, 1955.
3. S. Baybick and R. E. Montijo, Jr., An RCA high performance tape transport system, *Proc. Western Jt. Comp. Conf.*, 52-56, Feb. 26-28, 1957.
4. Louis D. Wilson and Saul Meyer, The model II unityper, *Trans. I.R.E. PGEC*, EC-2, No. 4, 19-27, Dec. 1953.
5. D. W. Burbeck and W. E. Frady, Precision automatic time measurement equipment, Hughes Aircraft Co., Culver City, presented at I.R.E. National Convention, New York, March 5, 1952.
6. R. W. Sears, Electron beam deflection tube for pulse code modulation, *Bell System Tech. J.*, 27 (1), 44-57 (1958).
7. H. J. Gray, Jr., P. V. Levonian, and M. Rubinoff, An analog-to-digital converter for serial computing machines, *Proc. I.R.E.*, 41, 1462-1465, Oct. 1953.
8. W. S. Shockency, Multi-channel analog-digital conversion system for d.c. voltages, *Proc. Western Comp. Conf., AIEE*, 113-117, Apr. 1954.
9. Taylor C. Fletcher and Norman C. Walker, Analog measurement and conversion to digits, *ISA J.*, 2, 345, Sept. 1955.
10. Ivan Flores, Reflected number systems, *Trans. I.R.E. PGEC*, EC-5 (2), 79-82, June 1956.
11. R. Hunt Brown, *Office Automation*, Automation Consultants, Inc., New York.
12. W. F. Bauer, Input-output equipment, Chap. 10, *Automation in Business and Industry*, E. M. Grabbe, Editor, Wiley, New York, 1957.
13. Bernard S. Benson and George G. Bower, Analog-to-digital conversion units, Chap. 9, *Automation in Business and Industry*, E. M. Grabbe, Editor, Wiley, New York, 1957.
14. J. M. Carroll, Trends in computer input-output devices, *Electronics*, 29, 142-149, Sept. 1956.

15. M. Phister, *Logical Design of Digital Computers*, Chap. 8, Wiley, New York, 1958.
16. J. A. Brustman, K. L. Chien, D. Flechtner, Input and output devices of the RCA BIZMAC system, *I.R.E. Convention Record*, Pt. 4, 88-93, March 19-22, 1956.
17. James W. Forgie, The Lincoln TX-2 input-output system, *Proc. Jt. Comp. Conf.*, 156-160, Feb. 26-28, 1957, Los Angeles, Calif.
18. Review of input and output equipment used in computing systems, *Jt. AIEE-IRE-ACM Comp. Conf.*, Dec. 10-12, 1952, published March 1953.
19. D. T. Olmsted, Charactron shaped beam tube, *Military Automation*, 138-140, May-June 1957.
20. E. M. DiGiulio, Burroughs G-101 high speed (matrix) printer, *I.R.E. Convention Record*, Pt. 4, 94-100, March 19-22, 1956.
21. C. B. Tompkins, J. H. Wakelin, W. W. Stifter, Jr., *High Speed Computing Devices*, 146-181, McGraw-Hill, New York, 1950.
22. D. F. Brower, A "one turn" magnetic reading and recording head for computer use, *I.R.E. Convention Record*, Pt. 4, 95-100, March 21-24, 1955.
23. R. K. Richards, *Digital Computer Components and Circuits*, Chaps. 7 and 11, Van Nostrand, Princeton, N. J., 1957.
24. M. V. Wilkes, *Automatic Digital Computers*, pp. 168-188, Wiley, New York, 1956.
25. S. Lubkin, An improved reading system of magnetically recorded digital data, *Trans. of the I.R.E. PGEC*, EC-3 (3), Sept. 1954.
26. A. Dean Glick, High-speed digital-to-analog conversion by integration of a variable-rate pulse train, *Proc. Western Jt. Comp. Conf.*, 128-133, Feb. 26-28, 1957, Los Angeles, Calif.
27. A. K. Susskind, *Notes on Analog-Digital Conversion Techniques*, Technology Press of M.I.T., and Wiley, New York, 1957.
28. M. Rubinoff and R. H. Beter, Input and output equipment, *Control Eng.*, **3**, 115-123 (1956).
29. P. Partos, Industrial data-reduction and analogue-digital conversion equipment, *J. Brit. Inst. Radio Eng.*, **16**, 651-678, Dec. 1956.
30. K. R. Eldredge, F. J. Kamphoefner, and P. H. Wendt, Automatic input for business data-processing systems, *Proc. Eastern Jt. Comp. Conf.*, December 10-12, 1956, New York, 69-72 (1957).
31. D. N. MacDonald, Datafile—a new tool for extensive file storage, *Proc. Eastern Jt. Comp. Conf.*, Dec. 10-12, 1956, New York, 124-127 (1957).
32. H. Epstein and P. Kinter, The Burroughs electrographic printer-plotter for ordnance computing, *Proc. Eastern Jt. Comp. Conf.*, Dec. 10-12, 1956, New York, 73-80 (1957).
33. A. K. Susskind, Characteristics and appraisal of analog-digital conversion techniques, *Proc. Comp. in Control Conf.*, October 16-18, 1957, Atlantic City, AIEE, T-101, 26-29, May 1958.
34. D. H. Shepard, P. F. Bargh, C. C. Heasley, Jr., A reliable character sensing system for documents prepared on conventional business devices, *I.R.E. 1957 Wescon Convention Record*, Pt. 4, 111-120.
35. F. Bauer, The Burroughs 220 high speed printer system, *Proc. Western Jt. Comp. Conf.*, March 3-5, 1959, San Francisco.
36. R. M. Hayes and J. Wiener, Magnacard—a new concept in data handling, *I.R.E. 1957 Wescon Convention Record*, Pt. 4, 205-210.

37. A. M. Nelson, H. M. Stern and L. R. Wilson, Magnacard—mechanical handling techniques, *I.R.E. 1957 Wescon Convention Record*, Pt. 4, 210-214.
38. J. Burkig and L. E. Justice, Magnacard—magnetic recording studies, *I.R.E. 1957 Wescon Convention Record*, Pt. 4, 214-218.
39. A. M. Angel, A very high speed punched paper tape reader, *I.R.E. 1957 Wescon Convention Record*, Pt. 4, 218-227.
40. K. R. Eldredge, Magnetic reader speeds travelers check processing, *Control Eng.*, **5**, 79-84, July 1958.
41. E. J. Kompass, What about digital transducers?, *Control Engr.*, **5**, 94-100, July 1958.
42. E. G. Wildanger, Tape recording systems for computers, *Automatic Control*, **7**, 36-42, Dec. 1957.
43. W. Gersch, Recording media, techniques and devices, Pt. II, *Automatic Control*, **7**, 11-17, Jan. 1958.

# DESIGN AND APPLICATION OF ANALOG COMPUTERS

## E. DESIGN AND APPLICATION OF ANALOG COMPUTERS

W. J. Karplus, Editor

21. *Analog Computation in Engineering, by W. J. Karplus and W. Kindle*
22. *Linear Electronic Computer Elements, by I. Pfeffer*
23. *Nonlinear Electronic Computer Elements, by G. A. Bekey*
24. *Analogs and Duals of Physical Systems, by R. C. Mackey*
25. *Solution of Field Problems, by W. J. Karplus*
26. *Noise and Statistical Techniques, by H. Low*
27. *Mechanical Computer Elements, by W. J. Karplus*
28. *Digital Techniques in Analog Computation, by C. T. Leondes*



## Analog Computation in Engineering

*Walter J. Karplus and William Kindle*

1. Definition of Analog Computation	21-01
2. Classification of Analog Computers	21-02
3. Requirements of Analog Computers	21-05
4. General Steps in the Solution of Engineering Problems	21-06
5. Areas of Application of Analog Computers	21-09
6. Symbols and Diagram Notation	21-11
References	21-11

### I. DEFINITION OF ANALOG COMPUTATION

**Introduction.** Stimulated by the demands of modern technology, engineers have undertaken the design of systems of ever increasing complexity. Prior to the advent of automatic computing equipment, engineering design and product testing were separated at a point determined primarily by the ability of the engineer to solve the design problems manually, or by using such restricted aids as the slide rule or desk calculator. As systems became more and more complex and as equipment became extremely expensive, and sometimes dangerous, to build and test, means had to be found to expand greatly the capabilities of the design engineer. Analogs and digital computers were developed to help fill this need.

**Definition.** The *analog computer* is an engineering tool used in the laboratory to study physical systems which are too complicated to analyze with pencil and paper and with manual computational aids, and for which the "cut-and-try" process of design and test is prohibitively

## 21-02 DESIGN AND APPLICATION OF ANALOG COMPUTERS

time-consuming and expensive. The principal distinctive feature of such computers is that the data involved in the computation are carried in *continuous form*, for example as continuously varying voltages or shaft rotations. The sensing and display of these quantities is likewise continuous. Thus, the precision, or number of significant figures available, is determined entirely by the quality of the computer components and the output equipment. In most analog computers, each step or operation of computation is performed by a separate unit, and all units operate simultaneously. This type of computation is termed *parallel operation*, and makes the solution available almost immediately. This feature is particularly important in engineering design problems, for it permits the engineer to adjust and vary any of the design parameters and observe at once the effect of these variations upon the response of the system. The direct insight into the operation of the system, gained in this manner, constitutes an important advantage of analog computers.

**Analog and Digital Computers.** By contrast, the other major type of automatic computer, the digital computer, handles data in discrete steps. Arithmetical operations are performed consecutively in a pre-determined sequence, termed *serial operation*. The precision of a digital computer is limited only by the number of significant figures carried in the solution. This is determined by the size of the computer, and can be increased as desired by expanding the installation. The major differences between analog and digital equipment are summarized in Table 1.

TABLE 1. MAJOR DIFFERENCES BETWEEN ANALOG AND DIGITAL COMPUTERS

Analog Computers	Digital Computers
Data in continuous form	Data in discrete form
Operations performed simultaneously (parallel)	Operations performed sequentially (serial)
Precision and accuracy limited by quality of components	Precision limited by size of installation
Relatively inexpensive for accuracies within 1 per cent, relatively expensive for higher accuracies	Basic cost relatively high regardless of accuracy

### 2. CLASSIFICATION OF ANALOG COMPUTERS

Analog computers may be classified on the basis of their application, their basic principles of operation, and the types of physical variables within the computer which constitute the continuous data.

**Special Purpose and General Purpose.** *Special purpose* analog computers are designed to perform specific and frequently highly specialized operations within a larger system. Their design characteristics vary widely, depending upon their application. *General purpose* analog

computers include all analog devices which are designed for the solution of a general class of problems and which may be applied, as desired by the engineer, in the design and analysis of physical systems. Such computers are generally not a part of any specific system but are maintained separately as a permanent laboratory installation. See Fig. 1.

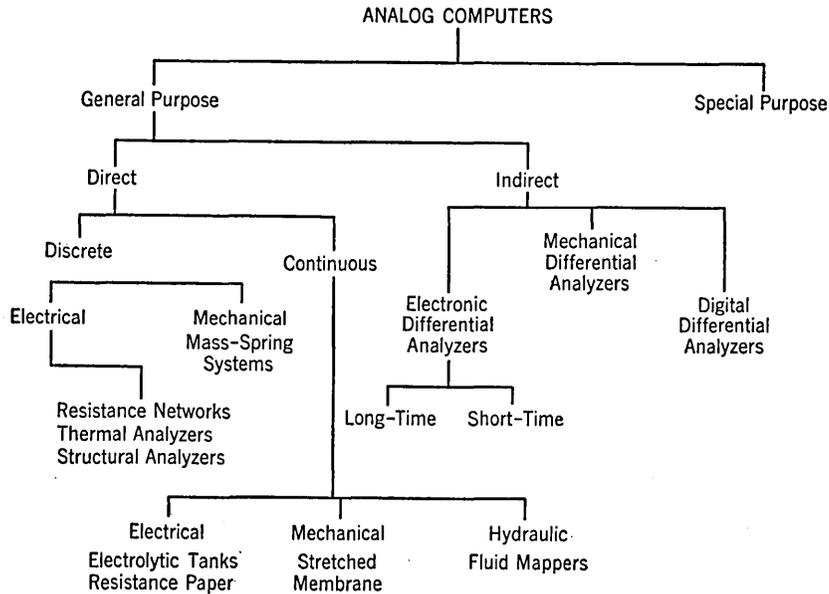


FIG. 1. Classification of analog computers.

**Direct and Indirect.** General purpose analog computers may be divided into two broad categories: direct and indirect. *Direct* computers, also known as direct simulators, establish a physical analogy between the simulator and the prototype system under study. Such an analogy is recognized by comparing the characteristic equations, usually ordinary or partial differential equations, describing the transient or static behavior of the two systems. If these equations are similar in form, an analogy exists, and the response of the prototype system to an excitation may be determined by subjecting the analog system to a similar excitation and observing its response. Such analog systems are constructed by simulating every element in the prototype by an element having similar properties, that is, by an element whose excitation and response are related in a similar manner. These analogous elements are interconnected so that the topological properties of the original system are

## 21-04 DESIGN AND APPLICATION OF ANALOG COMPUTERS

conserved. The concept of *duality* may then be employed to obtain analog systems with inverse structures.

Direct analogs may be either of the *discrete* or *continuous* variety. *Discrete* analogs employ lumped physical elements such as resistors and capacitors, and the behavior of the system is defined only for the node points of the circuit. Analog models of this type are useful for the analysis of systems comprised of lumped mechanical elements governed by ordinary differential equations, and for the solution of the finite difference approximations of the partial differential equations governing continuous fields. *Continuous* analog systems are used to simulate distributed field problems, containing time and space variables, so that every point in the analog corresponds to a specific point in the prototype. Such simulations are employed in the solution of problems governed by partial differential equations.

*Indirect* analog computers are employed in engineering to perform the mathematical operations necessary to solve the equations governing the system. Such devices employ only one dependent variable, for instance, voltage, to represent all the dependent variables of the prototype. For example, at one point in a computer voltage may be analogous to displacement, whereas at another location voltage may represent velocity. Such a computer is essentially an equation solver, and the manner of interconnecting its components generally has no direct relation to the topology of the system being simulated. *Indirect analog computers are by far the most important and widely used analog computers for automation and control problems.*

**Other Classifications.** Both the direct and indirect analog computers may be subdivided according to the physical area to which they belong. Thus, *electrical* analogs employ voltage or current as dependent variables, while in *mechanical* analog computers linear or rotational shaft displacements compose the basic data. The indirect electrical analogs are further subdivided into the *long-time* and *short-time* classes. The former are designed so that solutions are obtained in from 10 seconds to several minutes, and the output displayed on a strip chart or servo-driven recorder. Short time computers, also known as *repetitive* computers, have solution times of several milliseconds. The solution of the problem is then repeated periodically and displayed on a cathode ray oscilloscope. A recently developed class of long-time, indirect analog computers, known as *digital differential analyzers*, solve the system of equations comprising the problem in essentially the same mathematical fashion as other indirect analog computers, but perform the individual operations digitally, with pulses and a magnetic drum memory.

### 3. REQUIREMENTS OF ANALOG COMPUTERS

**Input Requirements.** The analog computer must be able to accept all data necessary to simulate the excitation and characteristics of the prototype system under study. These include:

a. The *complete system of equations*, differential and algebraic, as well as necessary graphical or tabular data which together specify the system. The complexity of this analog model is governed by the nature of the physical system under study and how accurately its characteristics are known, as well as by the specific use to which the solution is to be put.

b. The *parameters* of the system. These may remain permanently fixed, or they may be a function either of time or of the dependent variables of the problem.

c. *Initial conditions*, denoting the magnitudes of all dependent variables and their derivatives at the commencement of the computation.

d. All *external excitations* or stimuli imposed upon the system. These may be of constant magnitudes, varying with time in a prescribed manner, or may be of a random character.

**Output Requirements.** The computer must furnish the operator all data required for the accomplishment of his objective. The requirements placed upon the analog computer, therefore, tend to vary widely, depending upon their specific area of utilization, and generally include the following:

a. A *continuous display* of the variation of the dependent variables and their derivatives as a function of time or some other independent variable. Frequently these data are required only for a limited number of locations in the overall system.

b. A *display* demonstrating the effect of varying a number of the system parameters. Particularly in design problems, the experimental determination of an optimum solution demands the systematic investigation of the behavior of numerous alternative designs.

c. *Permanent records* of the system response. These may be strip chart records, servo-driven potentiometer records, photographs of cathode ray oscillograph displays, tables of measured d-c or a-c voltages, or merely indications of whether the system is stable or unstable under specified conditions.

**Flexibility.** To be a truly general purpose engineering tool, the adaptation of the computer to the treatment of any of a large class of problems must require no major modifications or alterations. In the indirect analog computer, units performing specific computational functions are arranged in such a manner that they may readily be interconnected and utilized

## 21-06 DESIGN AND APPLICATION OF ANALOG COMPUTERS

as required by the specific application. The devices for the generation of excitations, establishment of initial conditions, and displaying the outputs of interest are likewise arranged so as to permit their connection to any point of the simulation system. The excitation instruments and elements determining the magnitude of the system parameters must be adjustable over a wide range.

**Convenience.** A major portion of the time spent in the computer treatment of engineering problems is devoted to the setting up and checking out of the computer system prior to the obtaining of any useful data. To minimize this effort, the interconnecting and adjusting of the computer units and subunits should be as simple and convenient as possible. To this end, all internal connection points are generally brought out to a centrally located patch bay, and plug-in type patch cords are employed to interconnect the terminals as required. These patch boards are usually removable, so that they may be programmed, checked, and stored separately. This avoids the tying up of the computer installation by a single problem and greatly extends the usefulness of the system.

### 4. GENERAL STEPS IN THE SOLUTION OF ENGINEERING PROBLEMS

**Formulation.** A clear and complete description of the problem must be provided. The problem statement must include an adequate specification of the physical system, its excitations, and the data which would constitute an acceptable solution. This step usually requires the cooperation of the design engineer and the computer specialist, and frequently poses a serious communication problem. The design engineer should have at least a general understanding of the principles of operation of the computer, and the computer operator should possess some insight into the characteristics of the physical system under study.

**Mathematical Modeling.** The problem statement is then translated into mathematical language. This process almost invariably requires approximations or idealizations of system behavior and strongly affects the final accuracy and value of the solution. Considerable experience and analytical skill are required for this purpose.

**Rearrangement.** The mathematical model is then rearranged to make it suitable for computer solution. This is determined to some extent by the idiosyncrasies of available computer equipment.

**Block Diagram.** The computer units required to perform the specified mathematical operations, to supply the necessary excitations, and to display the desired outputs, are summarized in a block diagram. From such a block diagram it may readily be determined whether or not sufficient computer equipment is available, and how the various units are to be interconnected.

**Scale Factoring.** Scale factors relating each dependent and independent variable of the prototype system with a corresponding variable within the computer are selected. The characteristics of the computer place definite upper and lower limits upon the excursions permitted the machine variables. Since the transient behavior of the dependent variables may be initially unknown and may actually comprise the solution of the problem, scale factoring frequently involves the making of judicious guesses, which are revised and improved subsequently. A detailed machine diagram is then drawn, showing the detailed patch board interconnections of all computer units, as well as the dial settings of all parameter units, excitations, etc.

**Problem Setup.** The computer is programmed as indicated by the detailed machine diagram. Since errors in patch board connections are frequently very difficult to pinpoint after all wiring has been completed, it is vital to employ great care and a systematic approach in making the plug board connections.

**Check of Problem Setup.**

Where possible, check runs are made for excitations or initial conditions for which the response is known either approximately or exactly. In this way errors may be recognized and eliminated by trouble shooting.

**Problem Solution.** The data comprising the solution of the problem are obtained and recorded. Most frequently, the final records are families of curves illustrating the system behavior under various conditions.

**Application of Computer Results.** The computer solutions are translated back into the language of the original problem, and the results are utilized as required.

**EXAMPLE.** As a simple illustration of the steps involved in the analog computer treatment of a system, consider the problem of predicting the behavior of the electric circuit shown in Fig. 2 with initial conditions as shown.

a. The objective of this problem is to determine what capacitance,  $C$ , is required to make the frequency of oscillation equal to 100 radians per second.

b. The dynamic equation for the circuit of Fig. 2 is

$$(1) \quad L \frac{di}{dt} + Ri + \frac{1}{C} \int_0^t i dt = 0,$$

at  $t = 0, i = 1$ .

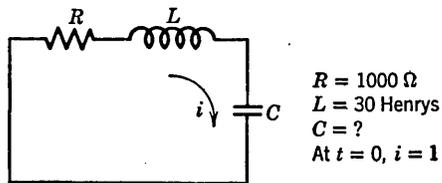


FIG. 2. Simple electric system.

## 21-08 DESIGN AND APPLICATION OF ANALOG COMPUTERS

c. This may be rearranged for computing purposes as

$$(2) \quad \frac{dq^2}{dt^2} = -\frac{R}{L} \frac{dq}{dt} - \frac{1}{LC} q,$$

where 
$$q = \int_0^t i dt.$$

d. A block diagram for the computer solution of this simple equation is shown in Fig. 3.

e. If servo-driven recorders are to be used, the frequency of operation should not exceed 1 radian per second. It is, therefore, necessary to

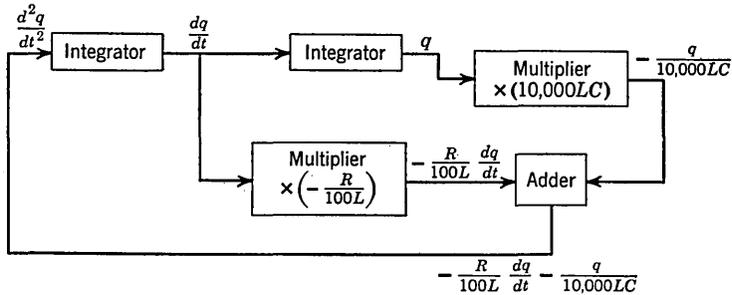


FIG. 3. Block diagram for computer solution.

define a computer time variable  $\tau$  which is related to the time variable in the prototype system according to

$$(3) \quad t = \frac{\tau}{100}.$$

Equation (2) then becomes

$$(4) \quad \frac{d^2q}{d\tau^2} = -\frac{R}{100L} \frac{dq}{d\tau} - \frac{1}{10,000LC} q.$$

f. A detailed wiring diagram as shown in Fig. 4 is now drawn. A potentiometer is employed to permit adjustment of the parameter  $C$ . A complete discussion of this procedure is presented in Chap. 22.

g. Computer runs are now made using various settings of the potentiometer. This process is continued until a potentiometer setting has been obtained for which the recorded frequency is 1 radian per second, corresponding to 100 radians per second in the original system.

h. The setting of this potentiometer is then translated into capacitance

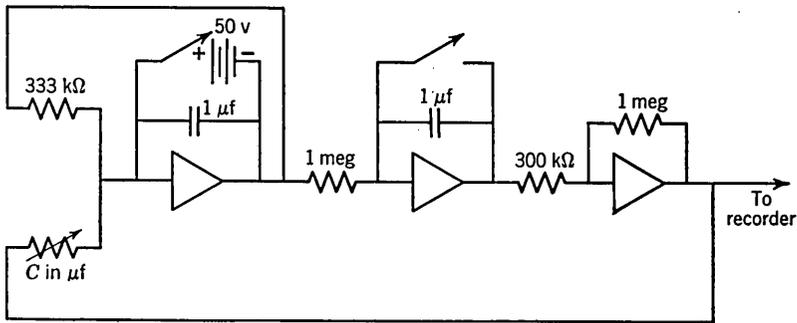


FIG. 4. Circuit diagram for computer solution.

values in the original system. According to

- (5) Resistance setting in megohms =  $C$  in  $\mu$ farads,  
which is the solution of the problem.

## 5. AREAS OF APPLICATION OF ANALOG COMPUTERS

**Indirect Computers.** Indirect computers have found wide application in many areas of engineering analysis and design. Some of the more important of these are listed below. Since many similar types of problems arise in engineering fields, no clear-cut line can be drawn between the type of problem and the associated engineering field.

### *Engineering Fields*

1. Guided missiles
2. Aircraft
3. Automotive design
4. Submarines
5. Nuclear reactors
6. Chemical unit and process operation
7. Internal combustion and turbine engines
8. Electrical, hydraulic, and pneumatic control devices
9. Electronic, electromagnetic, electromechanical instruments
10. General process control
11. Information theory
12. Operations research and linear programming

### *Types of Problems*

1. Dynamics of rigid bodies
2. Structural dynamics
3. Dynamics of bodies of varying characteristics
4. Aerodynamic and hydrodynamic stability
5. Stability and accuracy of automatic control systems
6. Heat transfer
7. Dynamics of compressible and incompressible fluids
8. Nuclear kinetics
9. Chemical kinetics and statics
10. Nonlinear electronics
11. Signal processing
12. Trajectory computations

## 21-10 DESIGN AND APPLICATION OF ANALOG COMPUTERS

**Direct Computers.** Direct computers are used primarily in the simulation of field problems, and some of their areas of application have included:

1. Determination of stability and load characteristics of electric power systems.
2. Temperature distribution in irregular bodies.
3. Stress distribution in complex structures under static and transient conditions.
4. Pressure and fluid distribution in oil reservoirs.
5. Electrostatic fields in capacitors and vacuum tubes.
6. Neutron diffusion in nuclear reactors.
7. Magnetic field patterns in the vicinity of antennas, in wave guides, and in resonators.
8. Pressure distribution in gas pipelines.
9. Trajectory of charged particles in electromagnetic fields.
10. Diffusion of air pollutants in the atmosphere.

TABLE 2. SYMBOLS USED IN CHAPTERS ON ANALOG COMPUTERS

Symbol	Meaning
$A$	Amplifier gain constant
$B$	Friction constant; time scale factor
$C$	Capacity
$e, e(t)$	Time-varying voltage
$E$	Constant voltage
$E(s)$	Laplace transform of $e(t)$
$f$	Function; force; frequency; subscript indicating feedback
$h$	Weighting function, unit impulse response
$i$	Current; subscript for input
$j$	Imaginary
$k$	Constant; subscript for summation
$K$	Constant, spring constant
$L$	Inductance
$M$	Mass
$o$	Subscript for output
$q$	Charge
$R$	Resistance
$s$	Laplace transform (complex frequency variable)
$t$	Time variable
$T$	Time interval; integration time
$u, v, w$	Variables
$v$	Velocity
$x, y, z$	Variables
$Z$	Impedance
$\delta$	Unit impulse
$\theta$	Angle
$\tau$	Time constant, delay time
$\omega$	Frequency (radians per second)

## 6. SYMBOLS AND DIAGRAM NOTATION

**Symbols.** An effort has been made to standardize symbols used in the chapters on analog computers. Lower-case letters are used for variables. Laplace transforms are capitals, always indicated as a function of  $s$ . Capitals are used for circuit constants. Other constants may be lower case or capital.

Those symbols appearing frequently are listed in Table 2. In a few cases, deviations from the normal usage are also listed in the table. Other symbols used are defined in the sections in which they are used.

**Diagram Notation.** Notation for analog computing elements is given in the following chapters:

Linear computing elements	Chap. 22, Sect. 1
Nonlinear computing elements	Chap. 23, Sect. 1
Mechanical computing elements	Chap. 27, Sect. 2
Analogs and duals	Chap. 24, Sect. 2
Digital differential analyzer	Chap. 28, Sect. 2

## REFERENCES

1. G. A. Korn, and T. N. Korn, *Electronic Analog Computers*, McGraw-Hill, New York, 1956.
2. H. M. Paynter, *A Palimpsest on the Electronic Analog Art*, G. A. Philbrick Researchers, Boston, Mass., 1955.
3. W. J. Karplus, *Analog Simulation: Solution of Field Problems*, McGraw-Hill, New York, 1958.
4. W. W. Soroka, *Analog Methods in Computation and Simulation*, McGraw-Hill, New York, 1954.



## Linear Electronic Computer Elements

*Irwin Pfeffer*

1. Introduction and Computer Diagram Notation	22-01
2. Passive Computer Elements	22-04
3. Direct-Current Operational Amplifiers with Feedback	22-08
4. Scale Factors	22-10
5. Typical Problem Setup	22-12
6. Representation of Complex Transfer Functions	22-13
7. Operational Amplifier Design	22-16
8. Errors in Linear Computer Elements	22-33
References	22-37

### I. INTRODUCTION AND COMPUTER DIAGRAM NOTATION

One of the most important applications of electronic analog computers is *the solution of systems of ordinary linear differential equations with constant coefficients*. The method employed in the solution is a fundamental one and is readily extended to the solution of systems of ordinary nonlinear differential equations.

Consider an ordinary linear differential equation with constant coefficients.

$$(1) \quad \sum_{k=0}^n a_k \frac{d^k y}{dt^k} = f(t).$$

**General Solution.** This may be rewritten to solve for the highest order derivative.

$$(2) \quad \frac{d^n y}{dt^n} = \frac{f(t)}{a_n} - \sum_{k=0}^{n-1} \frac{a_k}{a_n} \cdot \frac{d^k y}{dt^k}.$$

## 22-02 DESIGN AND APPLICATION OF ANALOG COMPUTERS

The general method of solution utilizes a procedure of repeated integration and is based upon the form of eq. (2). A block diagram of the computer configuration is shown in Fig. 1. The solution has the following steps:

a. The analog computer independent variable is real time. Assume that a continuously varying electric voltage, proportional in amplitude

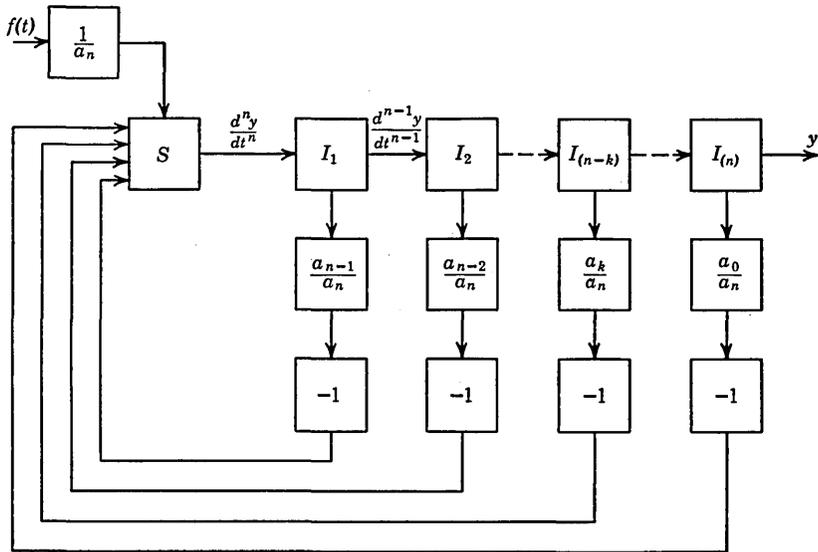


FIG. 1. Block diagram of computer configuration for solution of general ordinary linear differential equation.

at every instant to the highest order derivative,  $d^n y/dt^n$ , is generated at the output of element  $S$  in Fig. 1.

b. This output is connected to the input of integrating element  $I_1$  which generates an output voltage proportional to the time integral of its input, i.e., proportional to  $d^{n-1}y/dt^{n-1}$ .

c. This output is connected to  $I_2$ , etc., forming a cascade of  $n$  integrating elements, each generating a voltage representing one of the derivatives of  $y$ .

d. Each output voltage is multiplied by an appropriate constant,  $a_k/a_n$ , producing a voltage  $(a_k/a_n) \cdot (d^k y/dt^k)$  at the output of the multiplying element.

e. These voltages are passed through devices which invert the sign, i.e., reverse the polarity, and are introduced into the summing element  $S$ .

f. A voltage representing  $f(t)/a_n$  is also introduced into  $S$ .

g. Equation (2) indicates that the output of  $S$  is indeed the quantity  $d^n y/dt^n$ , originally assumed to be available.

The computer configuration is a closed loop system, operating in real time in a manner similar to any closed loop servo system, generating continuously the quantity  $y$  and its derivatives. The integrations, multiplications, sign inversions, and summation all occur simultaneously, and the quantities  $d^n y/dt^n$ ,  $(a_k/a_n) \cdot (d^k y/dt^k)$ ,  $y$ , etc., exist simultaneously as continuous voltage waveforms at the outputs of the various computing elements within the computer.

Equation (1) can be written alternatively in the form

$$(3) \quad y = \frac{f(t)}{a_0} - \sum_{k=1}^n \frac{a_k}{a_0} \frac{d^k y}{dt^k}.$$

A method of solution utilizing a procedure of repeated differentiation based on eq. (3) is equally acceptable mathematically as the method actually described. In practice, however, this method is almost never used since the process of differentiation tends to accentuate the noise present in all electrical apparatus. A solution based upon differentiation therefore would tend to be noisy and inaccurate; worse yet, the noise might saturate some element in the computer, resulting in a grossly incorrect solution.

**Computing Elements.** The following linear computing elements are required for the solution of systems of linear ordinary differential equations with constant coefficients:

1. An element capable of *multiplying* a voltage by a positive constant coefficient.
2. An element capable of *inverting* the sign of a voltage.
3. An element capable of *generating the sum* of two or more voltages.
4. An element capable of *generating the time integral* of a voltage.

In addition, a device for *generating an arbitrary function* of time is required if nonhomogeneous differential equations are to be solved. Strictly speaking, such a device is not a linear element, and its treatment is deferred to Chap. 23.

**Computer Diagram Notation.** Two basic notations for drawing computer diagrams are in widespread use. The first has evolved for use with computers featuring amplifiers with fixed internal resistors and capacitors, and hence having fixed gains. The second has evolved for use with computers featuring plug-in components. Figure 2 shows the correspondence between these notations. It is felt that the first notation enjoys the advantage of brevity. It will be used in this manual wherever applicable.

## 22-04 DESIGN AND APPLICATION OF ANALOG COMPUTERS

**Symbols.** The symbols used throughout this and other chapters are defined in Chap. 21, Sect. 6.

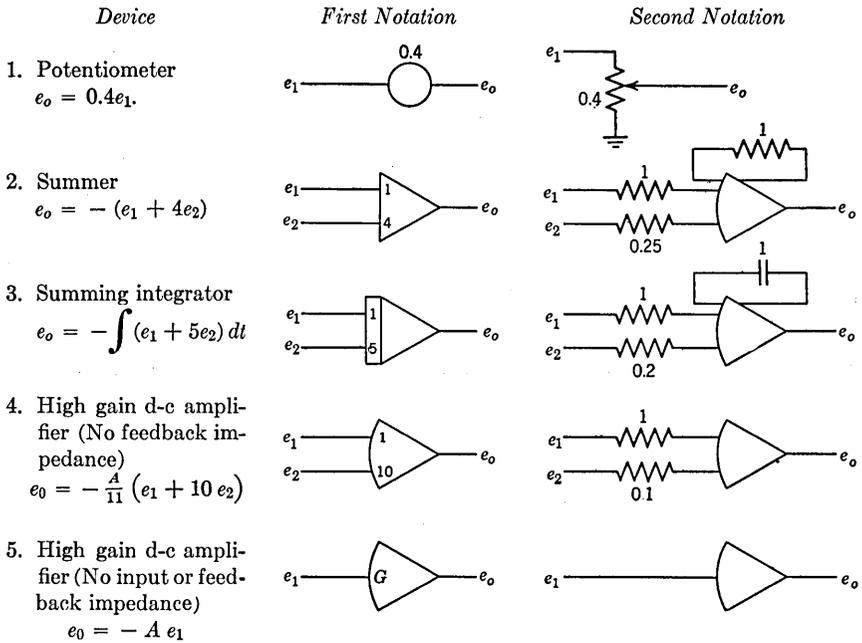


FIG. 2. Two notations for drawing computer diagrams. Values of resistors and capacitors given in megohms and microfarads.

### 2. PASSIVE COMPUTER ELEMENTS

**Multiplication of a Voltage by a Constant Coefficient.** The simplest and most satisfactory means of multiplying a voltage by a constant,  $a$ , uses a simple resistive voltage divider, or potentiometer as shown in Fig. 3.

In the absence of loading

$$(4) \quad e_o = ae_i \quad (0 \leq a \leq 1),$$

where  $a$  is the mechanical rotation of the arm, i.e., dial reading, provided the potentiometer is perfectly linear. If the potentiometer is not linear, or if it is loaded, the arm is set to provide the proper electrical output regardless of the dial reading. This is usually done by applying a known voltage,  $e_i$ , and adjusting the arm for the desired voltage  $e_o$  as read by an accurate voltmeter with the load resistance connected across the output terminals. For a potentiometer set in this manner linearity is of little importance. The only important electrical characteristics are reso-

lution, freedom from noise, and stability with temperature and age. For a potentiometer set by means of a dial, linearity is indeed an important factor, since any departure from linearity results in an error in the electrical setting. When setting by means of a dial, the following load correction

$$(5) \quad \epsilon = \frac{a^2(1 - a)(R_P/R_L)}{1 + a(1 - a)(R_P/R_L)}$$

must be added to the desired electrical setting  $a$ , and the dial actually set to  $(a + \epsilon)$ . Figure 4 shows a normalized plot of the loading correction  $\epsilon$  for the case  $R_P/R_L \ll 1$ .

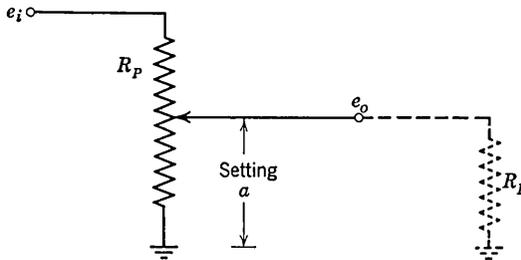


FIG. 3. Multiplication of a voltage by a constant coefficient using a potentiometer.

Multiturn, helical, wire-wound potentiometers are usually used for coefficient setting. Values of  $R_P$  commonly range from 10,000 ohms to 100,000 ohms. The lower limit is determined by power requirements, the upper limit by considerations of loading and the difficulty in manufacture of high-resistance elements.

**Passive Summing Network.** The passive summing network of Fig. 5 has an output voltage, in the absence of loading, given by

$$(6) \quad e_o = \frac{1}{\frac{1}{R_0} + \frac{1}{R_1} + \frac{1}{R_2} + \dots} \left( \frac{e_1}{R_1} + \frac{e_2}{R_2} + \dots \right)$$

$$= \frac{1}{R_0 + \sum_{k=1}^n \frac{1}{R_k}} \sum_{k=1}^n \frac{e_k}{R_k}$$

This network performs the operation of multiplication by a constant, as well as the operation of summation. There exists a serious drawback, however, in that the gain with loading depends appreciably on the load

## 22-06 DESIGN AND APPLICATION OF ANALOG COMPUTERS

resistance  $R_L$ . This dependence can be made small by use of a very small output resistor  $R_0$ , but this results in severe attenuation of the output.

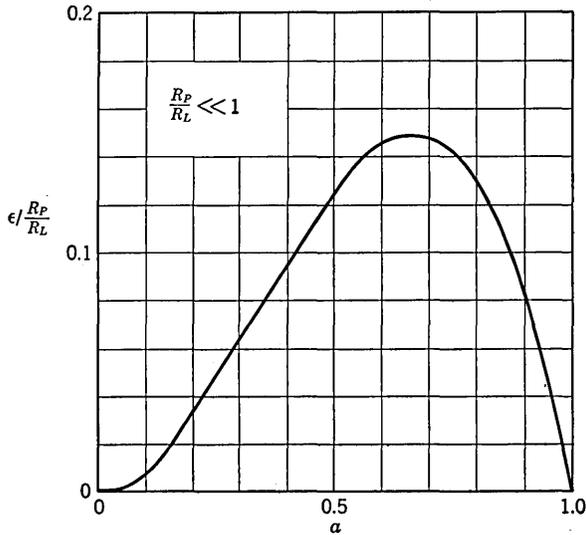


FIG. 4. Loading correction curve for lightly loaded potentiometer.

An output buffer amplifier with high input impedance might be used to restore the gain, but the gain of this amplifier would have to be very

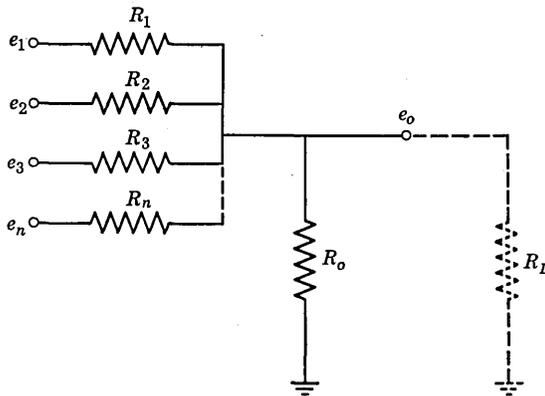


FIG. 5. Passive summing network.

stable and very accurately known. A much more basic and satisfactory method of using amplifiers is presented in Sect. 3.

**Passive Integrating Network.** The passive  $RC$  integrating network of Fig. 6 has an output voltage, in the absence of loading, given by the differential equation

$$(7) \quad \frac{de_o}{dt} = \frac{1}{RC} (e_i - e_o).$$

In operational, or Laplace, notation, eq. (7) may be rewritten as

$$(8) \quad E_o(s) = \frac{1}{RCs + 1} E_i(s),$$

or

$$(8a) \quad E_o(s) = \frac{1}{RC} \frac{1}{s} \frac{\tau_I s}{\tau_I s + 1} E_i(s)$$

where  $\tau_I$  equals  $RC$ ,  $s$  is the complex frequency variable, and  $E_o(s)$  and  $E_i(s)$  are the Laplace transforms of the output and input voltages respectively. The network of Fig. 6 can be made to approach a true integrator

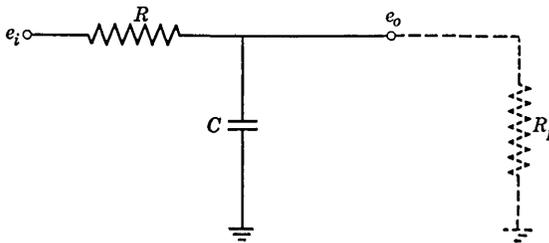


FIG. 6. Passive integrating network.

as the time constant  $\tau_I$  appearing in the expression  $\tau_I s / (\tau_I s + 1)$  is made very large. Several formidable obstacles exist to the widespread use of this network as an integrator:

1. In order to obtain acceptably large time constants, very large, costly capacitors are required.
2. The integrator time constant  $\tau_I$  and the integrator gain  $1/RC$  are reciprocally related in this network, so that large time constants necessarily result in low gain.
3. The integrator time constant is strongly dependent on the load resistance  $R_L$ .

The practical use of passive integrating networks is restricted to those cases where a reasonably small integrator time constant is satisfactory, i.e., those cases involving high-frequency inputs and/or short computing times.

## 3. DIRECT-CURRENT OPERATIONAL AMPLIFIERS WITH FEEDBACK

**General Operational Amplifier with Feedback.** The high gain d-c operational amplifier with feedback is truly the heart of the electronic analog computer. This device is capable of performing the basic linear operations of multiplication by a constant, sign inversion, summation, and integration, as well as other more complex operations. Figure 7 shows a block diagram of this device.

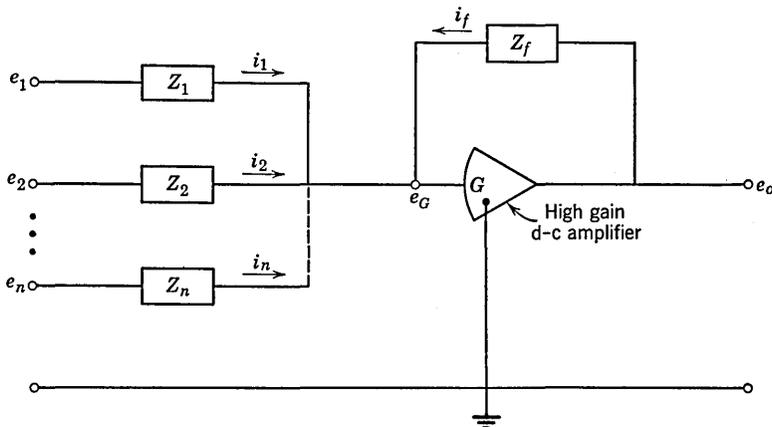


FIG. 7. Block diagram of d-c operational amplifier with feedback.

$Z_f$  is a series feedback impedance with current  $i_f$ ,  $Z_1, Z_2, \dots, Z_n$  are series input impedances with currents  $i_1, i_2, \dots, i_n$ , and  $e_G$  is the grid voltage. The gain of the amplifier must be negative, i.e., the amplifier must provide a polarity reversal. Calling this gain  $-A$ , eqs. (9-12) describe the circuit operation if no grid current is drawn.

$$(9) \quad e_o = -Ae_G,$$

$$(10) \quad i_f + \sum_{k=1}^n i_k = 0,$$

$$(11) \quad i_f = \frac{e_o - e_G}{Z_f},$$

$$(12) \quad i_k = \frac{e_k - e_G}{Z_k}, \quad k = 1, 2, \dots, n.$$

Solution of this set of equations for the output voltage gives

$$(13) \quad e_o = - \frac{1}{1 + (1 + K)/A} \left[ e_1 \frac{Z_f}{Z_1} + e_2 \frac{Z_f}{Z_2} + e_3 \frac{Z_f}{Z_3} + \dots \right]$$

$$= - \frac{Z_f \sum_{k=1}^n \frac{e_k}{Z_k}}{1 + (1 + K)/A},$$

where

$$K = \frac{Z_f}{Z_1} + \frac{Z_f}{Z_2} + \frac{Z_f}{Z_3} + \dots = Z_f \sum_{k=1}^n \frac{1}{Z_k}.$$

The amplifier gain  $A$  is made very high, in the order of many thousands or even millions. For high amplifier gain the denominator of eq. (13) approaches unity, so that the output voltage is very nearly given by

$$(14) \quad e_o \cong - \left[ e_1 \frac{Z_f}{Z_1} + e_2 \frac{Z_f}{Z_2} + e_3 \frac{Z_f}{Z_3} + \dots \right]$$

$$\cong - Z_f \sum_{k=1}^n \frac{e_k}{Z_k}.$$

Note that the grid voltage  $e_G$  must be very nearly zero since in eq. (9)  $e_o$  is finite. The grid is said to be at *virtual ground* potential.

**Sign Inverters and Summers.** If all the impedances  $Z_f, Z_1, Z_2, \dots, Z_n$  are of the same kind, e.g., pure resistances, the output voltage is the negative sum of the input voltages, each multiplied by a constant which is the ratio of the feedback resistance to the particular input resistance. For the case of a single input with the feedback and input resistances made equal, the output is simply the negative of the input and the amplifier is spoken of as a sign inverter. For the multiple input case the amplifier is spoken of as a summing amplifier or summer. Gains greater than unity are achieved by choosing an individual input resistor smaller than the feedback resistor. In some commercially available computers, the feedback and input resistors are selected and plugged in by the operator. In others the feedback and several input resistors are permanently connected with gains of 1, 4, 5, and 10 commonly provided. Values of resistors for this purpose usually range from 100,000 ohms to 1 megohm.

**Integrators and Summing Integrators.** A nearly perfect integrator is obtained by using a capacitor  $C$  as the feedback impedance and a resistor  $R$  as the input impedance. The output voltage, in operational notation, is, from eq. (14)

$$(15) \quad E_o(s) = - \frac{1}{RCs} E_i(s).$$

## 22-10 DESIGN AND APPLICATION OF ANALOG COMPUTERS

The time constant of this integrator ideally is infinite, as may be seen by comparing eqs. (15) and (8a). The gain is finite and equal to  $-1/RC$ . Values of feedback capacitance from 0.1 to 1.0 microfarad are commonly used with input resistances from 100,000 ohms to 1 megohm. By use of multiple input resistors, the sums of integrals may be generated with a single amplifier. Such an amplifier is sometimes spoken of as a summing integrator. In like manner to the summing amplifier, integrator gains of 1, 4, 5, and 10  $\text{second}^{-1}$  are commonly provided by proper selection of input resistors and feedback capacitor.

**Initial Conditions.** In general,  $n$  initial conditions are required to specify the solution of a homogeneous  $n^{\text{th}}$  order differential equation. In the electronic analog computer, these initial conditions are applied as initial voltages at the output of the  $n$  integrators shown in Fig. 1. An initial voltage at the output of an integrator is produced by charging the feedback capacitor to the desired level prior to the start of computation. This usually is accomplished by using an appropriate voltage source together with an auxiliary charging network. The charging network, consisting of a separate input resistor and a feedback resistor in shunt with the feedback capacitor, is removed at the start of computation.

### 4. SCALE FACTORS

**Dependent Variables.** In electronic analog computation, the magnitude of each dependent variable of the problem is represented by a voltage at the output of a computing element. The constant relating the magnitude of the voltage to one unit of the variable is called the amplitude scale factor. Every element in the computer has a scale factor associated with it. Considerable care must be exercised in the selection of these scale factors to assure proper operation of the computer.

**Considerations in Choice of Amplitude Scale Factors.** The maximum available output voltage from an operational amplifier is limited by practical considerations to approximately one hundred volts. Before scale factors are assigned, the maximum magnitudes of all variables in the problem must be roughly ascertained by some means or other. The scale factor for each variable is then chosen so that the maximum voltage expected is a sizable fraction of the amplifier limiting level. If the scale factor is thus chosen, the amplifier will never be driven into limiting; yet the output voltage will be large compared with the spurious noise or drift voltages always present in electronic equipment. It frequently happens, particularly when dealing with new or novel problems, that initial scale factor assignments are greatly in error. This results either in overloading of one or more amplifiers or else in noisy, inaccurate

solutions. The scale factors must be readjusted by trial and error when this situation occurs.

When convenient, it is usually desirable to choose simple rational scale factors, such as 0.01, 0.4, 5.0, or 200. This facilitates both problem checking and the recording of solutions. *Example.* If a variable representing displacement is known to have a maximum value in the range of 0.2 to 0.4 feet, a scale factor of 200 volts per foot is appropriate.

**Independent Variable.** In electronic analog computation the independent variable is represented by computer time. The constant relating computer time in seconds to one unit of the independent variable is called the *time scale factor*. Thus

$$(16) \quad t_c = Bx,$$

where  $t_c$  = computer time in seconds,

$x$  = independent variable in appropriate units,

$B$  = time scale factor.

It frequently occurs, particularly in the study of dynamic systems, that the problem independent variable is time. In this case, the time scale factor relates computer time to problem time. A value of  $B$  greater than unity indicates that the computer operates more slowly than the actual system.

**Considerations in Choice of Time Scale Factors.** In a linear problem, choice of time scale factor is influenced by the following considerations:

1. Integrator errors are increased by long computer runs.
2. Long computer runs are associated with low, inaccurate potentiometer settings.
3. Short runs are associated with high amplifier gains and frequently require cascaded amplifiers to provide sufficient gain.
4. The high frequencies associated with short runs cause phase shift in amplifiers, and thus produce errors.
5. The dynamics of recording devices must be considered so that recorder response characteristics do not affect the recording.

Integrating errors in modern drift-stabilized computers are usually sufficiently low so that runs of many minutes duration may be tolerated. Much more serious limitations apply to high-speed operation. Cumulative phase shift due to cascading of amplifiers generally imposes a maximum frequency limitation of about 10 cycles per second on the computer. In some cases the errors resulting from phase shift may even become significant at frequencies below 10 cycles per second. If a servo-driven recorder is used, rather than a galvanometer type recorder or an

## 22-12 DESIGN AND APPLICATION OF ANALOG COMPUTERS

oscilloscope, the maximum permissible frequencies usually are limited to 1 cycle per second or less, owing to the inability of this device to follow large amplitude excursions at high frequencies.

In most computer usage, normal operating frequencies range from about 0.02 to 3.0 cycles per second; solution times range from 2 or 3 seconds to 5 minutes.

### 5. TYPICAL PROBLEM SETUP

**Problem Statement.** An understanding of the use of linear computing elements and of the manner in which scale factors are determined is best obtained by following through the steps required for the setup of a typical linear problem on the computer. These steps will be carried out for the differential equation

$$(17) \quad \frac{d^2y}{dt^2} + a_1 \frac{dy}{dt} + a_0y = a_0u(t),$$

which relates the response  $y(t)$  of a second order system to a unit step forcing function  $u(t)$ , applied at  $t = 0$ . For definiteness assume  $a_0 = +1600$ ,  $a_1 = +8$ , and all initial conditions are zero.

**Scale Factor.** To determine a proper scale factor for this problem, it is first noted that the undamped natural frequency of the system is

$$\omega_n = \sqrt{a_0} = 40 \text{ rad/sec.}$$

If it is desired to display the solution on a servo-driven recorder, it is necessary to effect a time scale change which will reduce the natural frequency on the computer to a much lower value, in the order of 1 to 5 radians per second. A time scale factor,  $B = 20$ , will be chosen for this purpose. From eq. (16) it is noted that

$$(18) \quad \begin{aligned} \frac{d}{dt} &= B \frac{d}{dt_c}, \\ \frac{d^2}{dt^2} &= B^2 \frac{d^2}{dt_c^2}. \end{aligned}$$

With these relations, eq. (17) may be rewritten for the computer in the form

$$(19) \quad \frac{d^2y}{dt_c^2} = -\frac{a_1}{B} \frac{dy}{dt_c} - \frac{a_0}{B^2} y + \frac{a_0}{B^2} u\left(\frac{t_c}{B}\right).$$

**Problem Setup.** Examination of eq. (17) shows that the steady-state value of  $y$  is unity. It is easily shown, however, that for very small (positive)  $a_1$ , the maximum value of  $y$  can approach 2. It can also be

shown that the maximum value of the derivative  $dy/dt_c$  cannot exceed a value equal to the product of the computer natural frequency and the peak amplitude of the sinusoidal component of  $y$ . In this example, therefore,  $dy/dt_c$  is limited to a value of 2 per second. Similar reasoning indicates that  $d^2y/dt_c^2$  will not exceed a value of 4 per second<sup>2</sup>. Hence

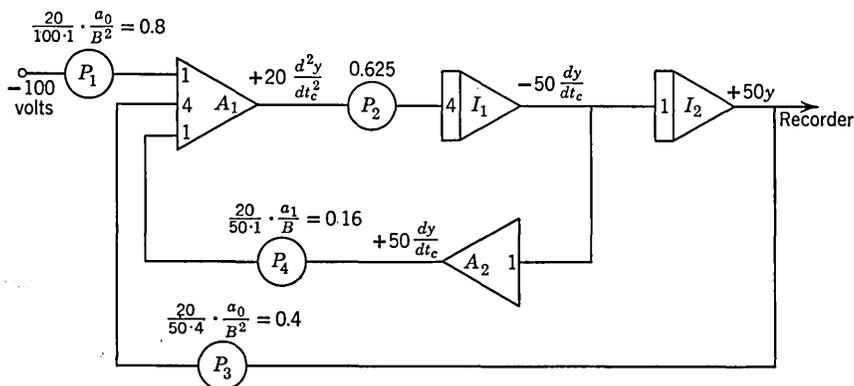


FIG. 8. Computer schematic diagram for solution of second order differential equation.

amplitude scale factors  $50y$ ,  $50dy/dt_c$ , and  $20d^2y/dt_c^2$  are reasonable for this problem. The computer schematic diagram is shown in Fig. 8. It is not necessary to use a switch in series with potentiometer  $P_1$  to apply the step-forcing function,  $u$ , since the very act of starting the computation applies this function automatically.

## 6. REPRESENTATION OF COMPLEX TRANSFER FUNCTIONS

**Definition and Methods.** The concept of the complex transfer function is extremely valuable in the study of dynamic systems on the analog computer. The *linear complex transfer function* is defined as a mathematical operator which relates the output function (response) to the input function (stimulus) for a particular linear device.

$$(20) \quad G(s) = \frac{E_o(s)}{E_i(s)},$$

where  $G(s)$  = linear complex transfer function,

$E_i(s)$  = Laplace transform of input,

$E_o(s)$  = Laplace transform of output.

If the device is such that the relation between its output and input can be described by a linear differential equation with real constant

## 22-14 DESIGN AND APPLICATION OF ANALOG COMPUTERS

coefficients,

$$(21) \quad \sum_{k=0}^m b_k \frac{d^k e_i}{dt^k} = \sum_{k=0}^n a_k \frac{d^k e_o}{dt^k},$$

then the complex transfer function  $G(s)$  may be represented as a quotient of two polynomials in  $s$ .

$$(22) \quad G(s) = \frac{\sum_{k=0}^m b_k s^k}{\sum_{k=0}^n a_k s^k} = \frac{P(s)}{Q(s)}.$$

Transfer functions of the type of eq. (22) include most cases of interest to the engineer. They may be represented on an analog computer, with certain restrictions, by any of three general methods: (1) *direct analog* method using passive networks; (2) *operational amplifier* method using complex input and feedback networks; (3) *differential analyzer* method which solves eq. (21) by use of potentiometers, summers, integrators, and (when necessary) differentiators.

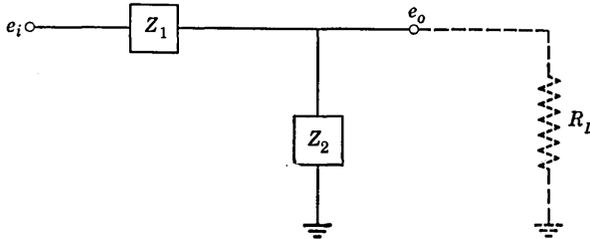


FIG. 9. Typical passive network circuit for realizing simple transfer functions. With no loading:

$$\frac{E_o(s)}{E_i(s)} = \frac{Z_2(s)}{Z_1(s) + Z_2(s)}.$$

**Direct Analog Method.** This method consists of using a passive network whose input voltage and open-circuit output voltage are related by the desired transfer function. A typical network of this type is shown in Fig. 9. The impedances  $Z_1$  and  $Z_2$  usually consist of series and/or parallel combinations of resistance and capacitance.

Ladder networks consisting of many sections of the type of Fig. 9 are sometimes useful. So too are bridged T and parallel T networks.

Passive networks of this type are simple but somewhat restricted in application. For *example*, if only resistance and capacitance are used,

the transfer function poles are restricted to the negative real axis. Also, output loading affects both the gain and transfer function.

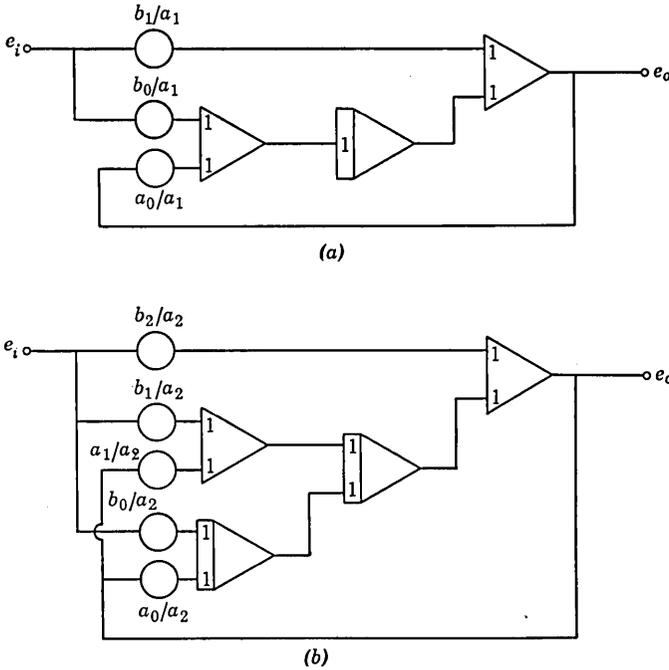


FIG. 10. Generation of first and second order transfer functions by differential analyzer method.

(a) 
$$\frac{E_o(s)}{E_i(s)} = -\frac{b_1s + b_o}{a_1s + a_o}$$

(b) 
$$\frac{E_o(s)}{E_i(s)} = -\frac{b_2s^2 + b_1s + b_o}{a_2s^2 + a_1s + a_o}$$

**Operational Amplifier Method.** The operational amplifier method is based upon eq. (14), rewritten here for the case of a single amplifier input,  $E_i(s)$ .

(23) 
$$\frac{E_o(s)}{E_i(s)} = -\frac{Z_f(s)}{Z_i(s)}$$

Suitable selection of input and feedback impedances allows a large class of transfer functions to be realized. Moreover, the input and feedback impedances are not restricted to the usual two-terminal or series type impedance, but may include linear four-terminal networks as well.

## 22-16 DESIGN AND APPLICATION OF ANALOG COMPUTERS

Equation (23) is valid for four-terminal networks provided that the impedance functions used are the short-circuit transfer impedances of the networks, that is, the ratio of input voltage to short-circuit output current. The proof of this follows immediately from the fact that the input grid is at virtual ground potential and draws no current. Table 1 lists the short-circuit transfer impedances of a number of networks suitable for use with operational amplifiers.

**Differential Analyzer Method.** Figure 10 shows how a general first order and second order transfer function is realized using potentiometers, amplifiers, and integrators to solve eq. (21).

Higher order transfer functions may be realized by logical extension of the schematics shown or by decomposition of the function into products of first and second order transfer functions. If the order  $m$  of the numerator exceeds the order  $n$  of the denominator,  $(m - n)$  differentiators are required. This situation rarely arises in practical problems.

### 7. OPERATIONAL AMPLIFIER DESIGN

**Characteristics.** Operational amplifiers used in electronic analog computers must possess a number of characteristics in order to insure satisfactory performance. Some of the most important of these characteristics are the following:

1. High gain at frequencies from dc to several kilocycles.
2. Linear output range from +100 volts to -100 volts (bipolar output).
3. High input impedance.
4. Low output impedance.
5. Stable operation when connected to a variety of feedback and load impedances.
6. Low closed loop phase shift at frequencies from dc to several kilocycles.
7. Low noise output.
8. Low output drift.
9. Low grid current.

The high gain characteristic usually requires that the amplifier be a multistage device. The d-c gain requirement necessitates use of a d-c amplifier. The linear, bipolar output range requirement necessitates the use of both positive and negative power supplies of sufficient voltage, and also influences the design of the amplifier output stage. High input impedance is usually obtained by placing the signal directly on the first stage grid. Low output impedance results from proper design of the output stage and the strong effect of large negative feedback. Stability, good frequency response (to several kilocycles), and low noise output

(to some extent) are achieved by careful attention to the design of inter-stage coupling networks and compensation networks.

The achievement of low output drift and low grid current are perhaps the most difficult goals of amplifier design and will be considered at greater length.

**Drift in D-C Amplifiers.** The output voltage of a d-c amplifier is sensitive to variations in plate, bias, and filament supply voltages, vacuum tube characteristics, and circuit resistance values. The output voltage thus contains, in addition to its correct value a spurious component which depends upon the aforementioned factors and tends to change slowly with time. This spurious component is usually called drift, and its presence gives rise to computer errors. These errors can be serious, particularly when they are integrated. Great care is given in amplifier

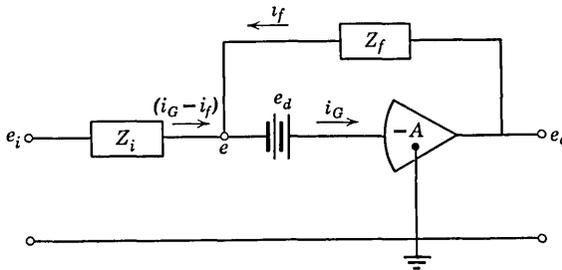


FIG. 11. Equivalent circuit for analysis of the effect of drift and grid current.

design to selection of high quality components and tubes, choice of operating conditions, and stabilization of supply voltages, in order to minimize drift. In addition a manual drift control is usually provided, which varies the cathode, grid or screen potential in the first or second stage, so that the amplifier may, from time to time, be adjusted to zero output for zero input. Satisfactory performance is sometimes achieved by such means, but for highest accuracy some additional drift reduction device must be provided.

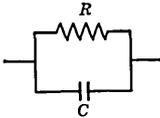
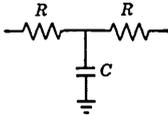
**Representation and Analysis of Drift.** For purposes of analysis, it is convenient to replace those actual factors causing drift by a single, small, equivalent voltage at the first stage grid. The magnitude  $e_d$  of this voltage is just that which would produce the observed output drift voltage in the absence of the actual factors. These factors may act in any stage, of course, but those acting within the first stage will in general have the most effect on output drift.

In Fig. 11,  $e_d$  is the equivalent drift voltage,  $i_G$  the first stage grid current,  $Z_f$  the feedback impedance, and  $Z_i$  an input impedance. For this

TABLE 1. SHORT-CIRCUIT TRANSFER IMPEDANCES FOR OPERATIONAL AMPLIFIERS<sup>a</sup>

This table is used to find networks whose impedance  $Z_0(s)$ ,  $Z_1(s)$ ,  $Z_2(s)$ ,  $\dots$ , match the desired equation for operational amplifiers:

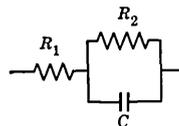
$$X_0 = -\frac{Z_0(s)}{Z_1(s)} X_1 \quad \text{or} \quad X_0 = -Z_0(s) \left[ \frac{X_1}{Z_1(s)} + \frac{X_2}{Z_2(s)} + \dots \right].$$

Transfer Impedance	Network	Relations	Inverse Relations
$A$		$A = R$	$R = A$
$\frac{A}{1 + sT}$		$A = R$ $T = RC$	$R = A$ $C = \frac{T}{A}$
$A(1 + sT)$		$A = 2R$ $T = \frac{RC}{2}$	$R = \frac{A}{2}$ $C = \frac{4T}{A}$

<sup>a</sup> Reproduced by permission of the McGraw-Hill Publishing Company from F. R. Bradley and R. McCoy, Driftless d-c amplifier, *Electronics*, April 1952. This table was developed by S. Godet of the Reeves Instrument Corporation, New York City.

$$A \left( \frac{1 + sT'}{1 + sT} \right)$$

$$\theta < 1$$



$$A = R_1 + R_2$$

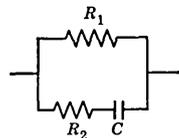
$$T = R_2 C$$

$$\theta = \frac{R_1}{R_1 + R_2}$$

$$R_1 = A\theta$$

$$R_2 = A(1 - \theta)$$

$$C = \frac{T}{A(1 - \theta)}$$



$$A = R_1$$

$$T = (R_1 + R_2)C$$

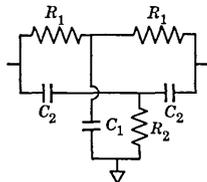
$$\theta = \frac{R_2}{R_1 + R_2}$$

$$R_1 = A$$

$$R_2 = \frac{A\theta}{1 - \theta}$$

$$C = \frac{T(1 - \theta)}{A}$$

$$A \left( \frac{1 + sT_1}{1 + s^2 T_1 T_2} \right)$$



$$A = 2R_1$$

$$T_1 = \frac{R_1 C_1}{2} = 2R_2 C_2$$

$$T_2 = R_1 C_2$$

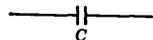
$$R_1 = \frac{A}{2}$$

$$R_2 = \frac{A T_1}{4 T_2}$$

$$C_1 = \frac{4 T_1}{A}$$

$$C_2 = \frac{2 T_2}{A}$$

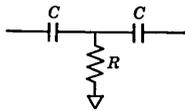
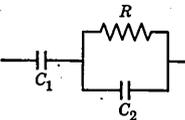
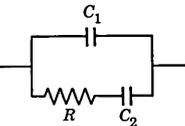
$$\frac{1}{sB}$$



$$B = C$$

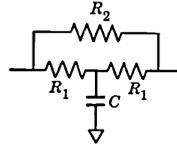
$$C = B$$

TABLE 1. SHORT-CIRCUIT TRANSFER IMPEDANCES FOR OPERATIONAL AMPLIFIERS—Continued

Transfer Impedance	Network	Relations	Inverse Relations
$\frac{1}{sB}(1 + sT)$		$B = C$ $T = RC$	$R = \frac{T}{B}$ $C = B$
$\frac{1}{sB} \left( \frac{1 + sT}{sT} \right)$		$B = \frac{C}{2}$ $T = 2RC$	$R = \frac{T}{4B}$ $C = 2B$
$\frac{1}{sB} \left( \frac{1 + sT}{1 + s\theta T} \right)$ $\theta < 1$		$B = C_1$ $T = R(C_1 + C_2)$ $\theta = \frac{C_2}{C_1 + C_2}$	$R = \frac{T(1 - \theta)}{B}$ $C_1 = B$ $C_2 = \frac{B\theta}{1 - \theta}$
		$B = C_1 + C_2$ $T = RC_2$ $\theta = \frac{C_1}{C_1 + C_2}$	$R = \frac{T}{B(1 - \theta)}$ $C_1 = B\theta$ $C_2 = B(1 - \theta)$

$$A \left( \frac{1 + sT}{1 + s\theta T} \right)$$

$$\theta < 1$$



$$A = \frac{2R_1 R_2}{2R_1 + R_2}$$

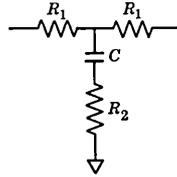
$$T = \frac{R_1 C}{2}$$

$$\theta = \frac{2R_1}{2R_1 + R_2}$$

$$R_1 = \frac{A}{2(1 - \theta)}$$

$$R_2 = \frac{A}{\theta}$$

$$C = \frac{4T(1 - \theta)}{A}$$



$$A = 2R_1$$

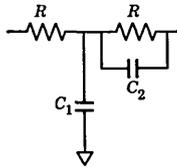
$$T = \left( R_2 + \frac{R_1}{2} \right) C$$

$$\theta = \frac{2R_2}{2R_2 + R_1}$$

$$R_1 = \frac{A}{2}$$

$$R_2 = \frac{A\theta}{4(1 - \theta)}$$

$$C = \frac{4T(1 - \theta)}{A}$$



$$A = 2R$$

$$T = \frac{R}{2} (C_1 + C_2)$$

$$\theta = \frac{2C_2}{C_1 + C_2}$$

$$R = \frac{A}{2}$$

$$C_1 = \frac{2T(2 - \theta)}{A}$$

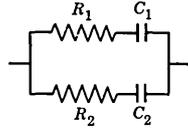
$$C_2 = \frac{2T\theta}{A}$$

TABLE 1. SHORT-CIRCUIT TRANSFER IMPEDANCES FOR OPERATIONAL AMPLIFIERS—Continued

Transfer Impedance	Network	Relations	Inverse Relations
$\frac{1}{sB} \left( \frac{1 + s\theta T}{1 + sT} \right)$ $\theta < 1$		$B = C_2$ $T = RC_1 \left( \frac{2C_2 + C_1}{C_2} \right)$ $\theta = \frac{2C_2}{2C_2 + C_1}$	$R = \frac{T\theta^2}{4B(1 - \theta)}$ $C_1 = \frac{2B(1 - \theta)}{\theta}$ $C_2 = B$
		$B = \frac{C_1^2}{2C_1 + C_2}$ $T = RC_2$ $\theta = \frac{2C_1}{2C_1 + C_2}$	$R = \frac{T\theta^2}{4B(1 - \theta)}$ $C_1 = \frac{2B}{\theta}$ $C_2 = \frac{4B(1 - \theta)}{\theta^2}$
		$B = \left( \frac{R_1}{R_1 + R_2} \right) C$ $T = R_2 C$ $\theta = \frac{2R_1}{R_1 + R_2}$	$R_1 = \frac{T\theta^2}{2B(2 - \theta)}$ $R_2 = \frac{T\theta}{2B}$ $C = \frac{2B}{\theta}$

$$\frac{1}{sB} \left[ \frac{(1 + sT_1)(1 + sT_3)}{1 + sT_2} \right]$$

$$T_1 < T_2 < T_3$$



$$B = C_1 + C_2$$

$$T_1 = R_1C_1$$

$$T_2 = (R_1 + R_2) \left( \frac{C_1C_2}{C_1 + C_2} \right)$$

$$T_3 = R_2C_2$$

$$R_1 = \frac{T_1(T_3 - T_1)}{B(T_2 - T_1)}$$

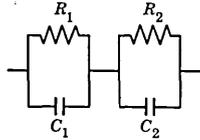
$$R_2 = \frac{T_3(T_3 - T_1)}{B(T_3 - T_2)}$$

$$C_1 = \frac{B(T_2 - T_1)}{T_3 - T_1}$$

$$C_2 = \frac{B(T_3 - T_2)}{T_3 - T_1}$$

$$A \left[ \frac{1 + sT_2}{(1 + sT_1)(1 + sT_3)} \right]$$

$$T_1 < T_2 < T_3$$



$$A = R_1 + R_2$$

$$T_1 = R_1C_1$$

$$T_2 = \left( \frac{R_1R_2}{R_1 + R_2} \right) (C_1 + C_2)$$

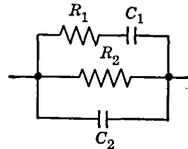
$$T_3 = R_2C_2$$

$$R_1 = \frac{A(T_2 - T_1)}{T_3 - T_1}$$

$$R_2 = \frac{A(T_3 - T_2)}{T_3 - T_1}$$

$$C_1 = \frac{T_1(T_3 - T_1)}{A(T_2 - T_1)}$$

$$C_2 = \frac{T_3(T_3 - T_1)}{A(T_3 - T_2)}$$



$$A = R_2$$

$$T_2 = R_1C_1$$

$$T_1T_3 = R_1R_2C_1C_2$$

$$T_1 + T_3 = R_1C_1 + R_2C_2 + R_2C_1$$

$$R_1 = \frac{AT_2^2}{(T_3 - T_2)(T_2 - T_1)}$$

$$R_2 = A$$

$$C_1 = \frac{(T_3 - T_2)(T_2 - T_1)}{AT_2}$$

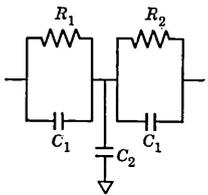
$$C_2 = \frac{T_1T_3}{AT_2}$$

TABLE 1. SHORT-CIRCUIT TRANSFER IMPEDANCES FOR OPERATIONAL AMPLIFIERS—Continued

Transfer Impedance	Network	Relations	Inverse Relations
$A \left[ \frac{1 + sT_2}{(1 + sT_1)(1 + sT_3)} \right]$ $T_1 < T_2 < T_3$		$A = R_1 + R_2$ $T_2 = \left( \frac{R_1 R_2}{R_1 + R_2} \right) C_2$ $T_1 T_3 = R_1 R_2 C_1 C_2$ $T_1 + T_3 = R_1 C_1 + R_2 C_2 + R_2 C_1$	$R_1 = \frac{AT_2^2}{T_1 T_2 + T_2 T_3 - T_1 T_3}$ $R_2 = \frac{A(T_3 - T_2)(T_2 - T_1)}{T_1 T_2 + T_2 T_3 - T_1 T_3}$ $C_1 = \frac{T_1 T_3}{AT_2}$ $C_2 = \frac{(T_1 T_2 + T_2 T_3 - T_1 T_3)^2}{AT_2(T_3 - T_2)(T_2 - T_1)}$
		$A = R_1$ $T_2 = R_2(C_1 + C_2)$ $T_1 T_3 = R_1 R_2 C_1 C_2$ $T_1 + T_3 = R_1 C_1 + R_2 C_2 + R_2 C_1$	$R_1 = A$ $R_2 = \frac{A(T_3 - T_2)(T_2 - T_1)}{(T_1 + T_3 - T_2)^2}$ $C_1 = \frac{T_1 + T_3 - T_2}{A}$ $C_2 = \frac{T_1 T_3(T_1 + T_3 - T_2)}{A(T_3 - T_2)(T_2 - T_1)}$
$A \left[ \frac{1 + sT_2}{(1 + sT_1)(1 + sT_3)} \right]$ $T_2 \leq T_1 \leq T_3$		$A = 2R_1 + \frac{R_1^2}{R_2}$ $T_1 = R_1 C_1$ $T_2 = \left( \frac{R_1 R_2}{R_1 + 2R_2} \right) (C_1 + C_2)$ $T_3 = R_1 C_2$	$R_1 = \frac{AT_2}{(T_1 + T_3)}$ $R_2 = \frac{AT_2^2}{(T_1 + T_3)(T_1 + T_3 - 2T_2)}$ $C_1 = \frac{T_1(T_1 + T_3)}{AT_2}$ $C_2 = \frac{T_3(T_1 + T_3)}{AT_2}$

$$A \left[ \frac{1 + sT_2}{(1 + sT_1)(1 + sT_3)} \right]$$

$$T_1 \leq T_3 \leq T_2$$



$$T_1 = R_1 C_1$$

$$T_2 = \frac{R_1 R_2}{R_1 + R_2} (2C_1 + C_2)$$

$$T_3 = R_2 C_1$$

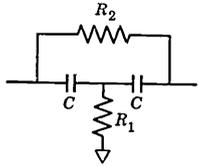
$$R_1 = \frac{1}{(T_1 + T_3)}$$

$$R_2 = \frac{A T_3}{T_1 + T_3}$$

$$C_1 = \frac{T_1 + T_3}{A}$$

$$C_2 = \frac{T_1 + T_3}{A} \left( \frac{T_2}{T_3} + \frac{T_2}{T_1} - 2 \right)$$

$$A \left( \frac{1 + sT_1}{1 + sT_1 + s^2 T_1 T_2} \right)$$



$$A = R_2$$

$$T_1 = 2R_1 C$$

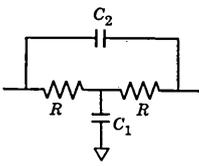
$$T_2 = \frac{R_2 C}{2}$$

$$R_1 = \frac{A T_1}{4 T_2}$$

$$R_2 = A$$

$$C = \frac{2 T_2}{A}$$

$$A \left( \frac{1 + sT_2}{1 + sT_1 + s^2 T_1 T_2} \right)$$



$$A = 2R$$

$$T_1 = 2RC_2$$

$$T_2 = \frac{RC_1}{2}$$

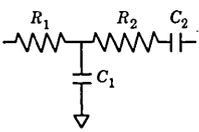
$$R = \frac{A}{2}$$

$$C_1 = \frac{4T_2}{A}$$

$$C_2 = \frac{T_1}{A}$$

$$\frac{1}{sB} (1 + sT_1)(1 + sT_2)$$

$$T_1 \neq T_2$$



$$B = C_2$$

$$T_1 T_2 = R_1 R_2 C_1 C_2$$

$$T_1 + T_2 = R_1 C_1 + R_2 C_2 + R_1 C_2$$

$$R_1 = \frac{(\sqrt{T_1} - \sqrt{T_2})^2}{B}$$

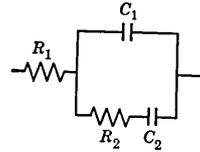
$$R_2 = \frac{\sqrt{T_1 T_2}}{B}$$

$$C_1 = \frac{B \sqrt{T_1 T_2}}{(\sqrt{T_1} - \sqrt{T_2})^2}$$

$$C_2 = B$$

TABLE 1. SHORT-CIRCUIT TRANSFER IMPEDANCES FOR OPERATIONAL AMPLIFIERS—Continued

Transfer Impedance	Network	Relations	Inverse Relations
$\frac{1}{sB} \left[ \frac{(1 + sT_1)(1 + sT_2)}{s\sqrt{T_1T_2}} \right]$ <p style="text-align: center;"><math>T_1 \neq T_2</math></p>		$B = C_2$ $T_1T_2 = R_1R_2C_1C_2$ $T_1 + T_2 = R_1C_1 + R_2C_2 + R_1C_2$	$R_1 = \frac{(\sqrt{T_1} - \sqrt{T_2})^2}{B}$ $R_2 = \frac{\sqrt{T_1T_2}}{B}$ $C_1 = \frac{B\sqrt{T_1T_2}}{(\sqrt{T_1} - \sqrt{T_2})^2}$ $C_2 = B$
$\frac{1}{sB} \left[ \frac{(1 + sT_1)(1 + sT_2)}{s^2T_1T_2} \right]$ <p style="text-align: center;"><math>T_1 &lt; T_2</math></p>		$B = \frac{C_1C_2}{C_1 + 2C_2}$ $T_1 = RC_1$ $T_2 = R(C_1 + 2C_2)$	$R = \frac{T_1(T_2 - T_1)}{2BT_2}$ $C_1 = \frac{2BT_2}{T_2 - T_1}$ $C_2 = \frac{BT_2}{T_1}$
$\frac{1}{sB} \left[ \frac{(1 + sT_1)(1 + sT_3)}{1 + sT_2} \right]$ <p style="text-align: center;"><math>T_1 &lt; T_2 &lt; T_3</math></p>		$B = C_1$ $T_2 = (R_1 + R_2)C_2$ $T_1T_3 = R_1R_2C_1C_2$ $T_1 + T_3 = R_1C_1 + R_2C_2 + R_1C_2$	$R_1 = \frac{T_1 + T_3 - T_2}{B}$ $R_2 = \frac{T_1T_3(T_1 + T_3 - T_2)}{B(T_3 - T_2)(T_2 - T_1)}$ $C_1 = B$ $C_2 = \frac{B(T_3 - T_2)(T_2 - T_1)}{(T_1 + T_3 - T_2)^2}$



$$B = C_1 + C_2$$

$$T_2 = R_2 \left( \frac{C_1 C_2}{C_1 + C_2} \right)$$

$$T_1 T_3 = R_1 R_2 C_1 C_2$$

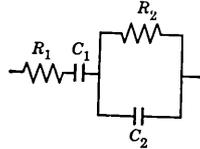
$$T_1 + T_3 = R_1 C_1 + R_2 C_2 + R_1 C_2$$

$$R_1 = \frac{T_1 T_3}{B T_2}$$

$$R_2 = \frac{(T_1 T_2 + T_2 T_3 - T_1 T_3)^2}{B T_2 (T_3 - T_2)(T_2 - T_1)}$$

$$C_1 = \frac{B T_2^2}{T_1 T_2 + T_2 T_3 - T_1 T_3}$$

$$C_2 = \frac{B(T_3 - T_2)(T_2 - T_1)}{T_1 T_2 + T_2 T_3 - T_1 T_3}$$



$$B = C_1$$

$$T_2 = R_2 C_2$$

$$T_1 T_3 = R_1 R_2 C_1 C_2$$

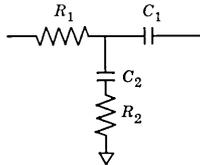
$$T_1 + T_3 = R_1 C_1 + R_2 C_2 + R_2 C_1$$

$$R_1 = \frac{T_1 T_3}{B T_2}$$

$$R_2 = \frac{(T_3 - T_2)(T_2 - T_1)}{B T_2}$$

$$C_1 = B$$

$$C_2 = \frac{B T_2^2}{(T_3 - T_2)(T_2 - T_1)}$$



$$B = C_1$$

$$T_2 = R_2 C_2$$

$$T_1 T_3 = R_1 R_2 C_1 C_2$$

$$T_1 + T_3 = R_1 C_1 + R_2 C_2 + R_1 C_2$$

$$R_1 = \frac{T_1 T_3}{B T_2}$$

$$R_2 = \frac{T_1 T_2 T_3}{B(T_3 - T_2)(T_2 - T_1)}$$

$$C_1 = B$$

$$C_2 = \frac{B(T_3 - T_2)(T_2 - T_1)}{T_1 T_3}$$

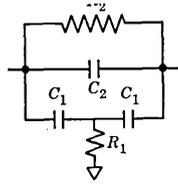
TABLE 1. SHORT-CIRCUIT TRANSFER IMPEDANCES FOR OPERATIONAL AMPLIFIERS—Continued

Transfer Impedance	Network	Relations	Inverse Relations
$A \left[ \frac{1 + sT_3}{1 + sT_1 + s^2T_1T_2} \right]$ $T_2 > \frac{T_1}{4} \text{ (complex roots)}$ $T_3 > T_2$		$A = \frac{2R_1R_2}{2R_1 + R_2}$ $T_1 = \frac{R_1(R_1C_1 + 2R_2C_2)}{2R_1 + R_2}$ $T_2 = \frac{R_1R_2C_1C_2}{R_1C_1 + 2R_2C_2}$ $T_3 = \frac{R_1C_1}{2}$	$R_1 = \frac{AT_3^2}{2[T_3^2 - T_1(T_3 - T_2)]}$ $R_2 = \frac{AT_3^2}{T_1(T_3 - T_2)}$ $C_1 = \frac{4[T_3^2 - T_1(T_3 - T_2)]}{AT_3}$ $C_2 = \frac{T_1T_2}{AT_3}$
		$A = 2R_1$ $T_1 = R_2C_1 + 2R_1C_2$ $T_2 = \frac{R_1(R_1 + 2R_2)C_1C_2}{R_2C_1 + 2R_1C_2}$ $T_3 = \left( R_2 + \frac{R_1}{2} \right) C_1$	$R_1 = \frac{A}{2}$ $R_2 = \frac{AT_1(T_3 - T_2)}{4[T_3^2 - T_1(T_3 - T_2)]}$ $C_1 = \frac{4[T_3^2 - T_1(T_3 - T_2)]}{AT_3}$ $C_2 = \frac{T_1T_2}{AT_3}$
		$A = 2R$ $T_1 = R(C_2 + 2C_3)$ $T_2 = \frac{RC_3(C_1 + C_2)}{C_2 + 2C_3}$ $T_3 = \frac{R}{2} (C_1 + C_2)$	$R = \frac{A}{2}$ $C_1 = \frac{2[2T_3^2 - T_1(T_3 - T_2)]}{AT_3}$ $C_2 = \frac{2T_1(T_3 - T_2)}{AT_3}$ $C_3 = \frac{T_1T_2}{AT_3}$

$$A [1 + sT_1 + s^2T_1T_2]$$

$$T_2 > \frac{T_1}{4} \text{ (complex roots)}$$

$$T_3 < T_1$$



$$T_1 = 2R_1C_1 + R_2C_2$$

$$T_2 = \frac{R_1R_2C_1(C_1 + 2C_2)}{2R_1C_1 + R_2C_2}$$

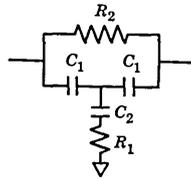
$$T_3 = 2R_1C_1$$

$$R_1 = \frac{A^2 T_3}{4[T_1T_2 - T_3(T_1 - T_3)]}$$

$$R_2 = A$$

$$C_1 = \frac{2[T_1T_2 - T_3(T_1 - T_3)]}{AT_3}$$

$$C_2 = \frac{(T_1 - T_3)}{A}$$



$$A = R_2$$

$$T_1 = \frac{C_1(2R_1C_2 + R_2C_1)}{2C_1 + C_2}$$

$$T_2 = \frac{R_1R_2C_1C_2}{2R_1C_2 + R_2C_1}$$

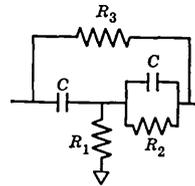
$$T_3 = \frac{2R_1C_1C_2}{2C_1 + C_2}$$

$$R_1 = \frac{AT_3^2}{4[T_1T_2 - T_3(T_1 - T_3)]}$$

$$R_2 = A$$

$$C_1 = \frac{2T_1T_2}{AT_3}$$

$$C_2 = \frac{4T_1T_2[T_1T_2 - T_3(T_1 - T_3)]}{AT_3^2(T_1 - T_3)}$$



$$A = R_3$$

$$T_1 = \frac{R_1(2R_2 + R_3)C}{R_1 + R_2}$$

$$T_2 = \frac{R_2R_3C}{2R_2 + R_3}$$

$$T_3 = \frac{2R_1R_2C}{R_1 + R_2}$$

$$R_1 = \frac{AT_3^2}{2[2T_1T_2 - T_3(T_1 - T_3)]}$$

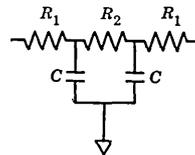
$$R_2 = \frac{AT_3}{2(T_1 - T_3)}$$

$$R_3 = A$$

$$C = \frac{2T_1T_2}{AT_3}$$

$$A(1 + sT_1)(1 + sT_2)$$

$$T_1 < T_2$$



$$A = 2R_1 + R_2$$

$$T_1 = \left( \frac{R_1R_2}{2R_1 + R_2} \right) C$$

$$T_2 = R_1C$$

$$R_1 = A \left( \frac{T_2 - T_1}{2T_2} \right)$$

$$R_2 = A \frac{T_1}{T_2}$$

$$C = \frac{2T_2^2}{A(T_2 - T_1)}$$

## 22-30 DESIGN AND APPLICATION OF ANALOG COMPUTERS

arrangement

$$(24) \quad e_o = -A(e_d + e),$$

$$(25) \quad i_f Z_f = e_o - e,$$

$$(26) \quad e = e_i - Z_i(i_G - i_f).$$

For large  $A$ , solution of the above equations yields for the output voltage with feedback:

$$(27) \quad e_o = -Ke_i - (1 + K)e_d + i_f Z_f,$$

where  $K = Z_f/Z_i$ .

If for the moment the effect of grid current is neglected, the output voltage *without* feedback is merely

$$(28) \quad e_o = -Ae_i - Ae_d.$$

Although feedback reduces the output due to drift in the ratio of  $A/(1 + K)$ , the output due to the signal  $e_i$  is reduced by an even larger ratio  $A/K$ . Therefore, feedback, despite its many other virtues, actually increases the fractional error due to drift in the ratio of  $(1 + K)/K$ .

Besides the drift, there is in eq. (27) a spurious component of output voltage due to first stage grid current, the magnitude of the voltage being that produced by the flow of all the grid current through the feedback impedance. In practice, there is no simple method of determining whether a given spurious output voltage is caused by drift or by grid current, but the distinction is nevertheless important, particularly in the consideration of drift-corrected amplifiers.

**Drift-Corrected Amplifiers.** The effect of drift can be reduced by adding to the main d-c amplifier an auxiliary drift-free amplifier which amplifies the input signal but not the drift voltage  $e_d$ . The circuit used is shown in Fig. 12, and a practical method of providing the auxiliary drift-free amplification is presented in the following section. If the gain of the auxiliary amplifier is  $A'$ , the following equations describe the operation of the circuit of Fig. 12.

$$(29) \quad e_o = -Ae_d - A(1 + A')e,$$

$$(30) \quad i_f Z_f = e_o - e,$$

$$(31) \quad e = e_i - Z_i(i_G - i_f).$$

It is seen that eqs. (29–31) are identical to eqs. (24–26) except for the additional amplification  $-AA'$  provided for the signal  $e$ . Solution of

eqs. (29-31) for large  $A$  results in

$$(32) \quad e_o = -Ke_i - \frac{(1 + K)}{(1 + A')} e_d + i_g Z_f,$$

where  $K = Z_f/Z_i$ .

Comparison of eqs. (27) and (32) shows that the addition of the auxiliary amplifier reduces the output voltage due to drift by a factor  $(1 + A')$ . Values of  $A'$  of the order of  $10^3$  are not difficult to achieve, so that the reduction in drift is indeed dramatic. *Example.* Modern,

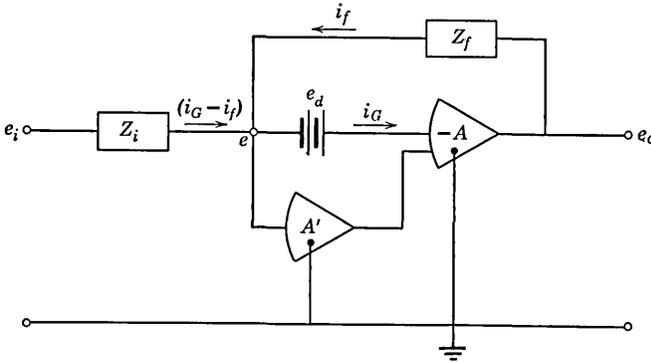


FIG. 12. Use of auxiliary amplifier for drift correction.

high quality, operational amplifiers utilizing this method of drift correction may have output drift levels as low as 10 to 100 microvolts over very long periods of time.

Comparison of eqs. (27) and (32) once again shows that the spurious output due to grid current is in no way lessened by addition of the auxiliary amplifier. The only practical remedy for input grid current troubles is the selection of a tube with sufficiently small grid current. Fortunately some currently available triodes have grid currents of the order of  $10^{-10}$  to  $10^{-11}$  ampere. After initial balancing, the output due to grid current can be maintained as low as 10 to 20 microvolts if a 1-megohm feedback resistor is used.

**Chopper Amplifiers.** The usual method of providing auxiliary drift-free amplification is to use the voltage  $e$  at the junction of the input and feedback resistors to modulate a carrier signal. The auxiliary amplifier is an a-c amplifier, inherently drift free, which amplifies this carrier signal. The output of the auxiliary amplifier is demodulated and applied to the first stage of the main amplifier along with the direct path voltage  $e$ . Addition of these two voltages is usually accomplished

22-32 DESIGN AND APPLICATION OF ANALOG COMPUTERS

through the use of differential amplification in the first stage of the main amplifier. Modulation and demodulation are usually achieved by

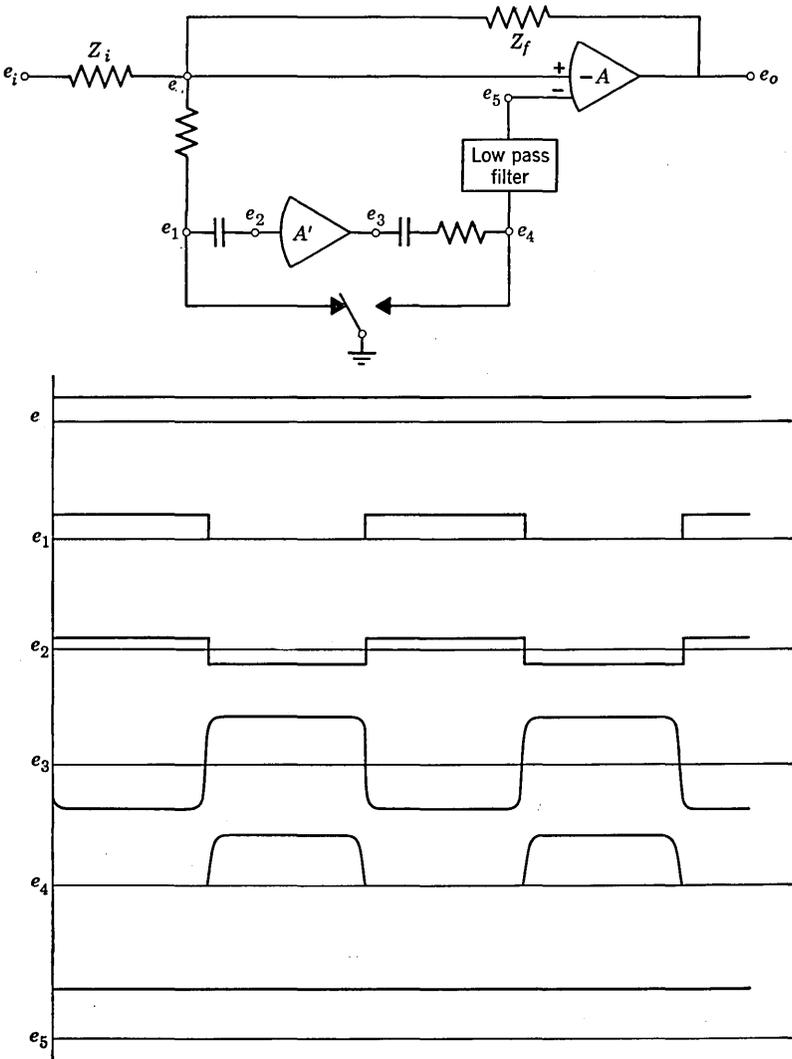


FIG. 13. Operation of chopper amplifier circuit.

the action of a single vibrating relay or chopper, driven by an alternating current of suitable frequency.

Figure 13 illustrates the operation of this circuit, commonly referred to as a chopper amplifier. The voltage  $e$  is chopped, amplified, rectified,

filtered, and applied to the main amplifier. The output filter has a very low passband compared with the chopper frequency, so that the combination of chopper contacts, a-c amplifier, and filter form, effectively, an auxiliary driftless d-c amplifier having a bandwidth from zero to perhaps 1 cycle per second. This small bandwidth is the principal reason why the chopper amplifier cannot be used by itself as the main amplifier. The benefits of the increased gain from the chopper amplifier are, of course, only realized over this very narrow band, but this is sufficient for the suppression of drift in the main amplifier.

The combination of the main amplifier and the chopper amplifier is usually spoken of as either a chopper stabilized d-c amplifier or a drift corrected d-c amplifier. Development of the chopper stabilized d-c amplifier was one of the most important events in the development of present day electronic analog computers.

The schematic of a typical amplifier employing chopper stabilization is shown in Fig. 14.

## 8. ERRORS IN LINEAR COMPUTER ELEMENTS

**Error Sources.** The most important errors in the solution of ordinary linear, constant coefficient differential equations on an electronic analog computer arise from the following sources: (1) potentiometer errors, (2) amplifier gain errors, (3) loading errors, (4) amplifier time constant effects (limited bandwidth effects), (5) integrator time constant effects, (6) amplifier drift, (7) amplifier noise, and (8) recording errors.

**Potentiometer Errors.** For potentiometers set by means of an external voltmeter, the principal errors are due to voltmeter inaccuracy, potentiometer resolution, and potentiometer stability. For modern multi-turn potentiometers and precision voltmeters, potentiometer errors as low as 0.01 to 0.02 per cent may be achieved.

**Amplifier Gain Errors.** The closed loop gain of a summer or integrator is a function only of the ratio of the feedback and input impedances provided the open loop amplifier gain is sufficiently high. By use of precision resistors and capacitors enclosed in temperature controlled ovens, this ratio can be made accurate to about 0.01 to 0.02 per cent. The error in closed loop gain due to finite open loop amplifier gain is considerably less than 0.01 per cent, under normal operating conditions, and is therefore negligible.

**Loading Errors.** Output loading errors, so serious in passive computing networks, are virtually eliminated by the action of high amplifier gain in a feedback loop. The closed loop amplifier output impedance is of the order of 0.1 ohm or less. The load impedance seen is almost never

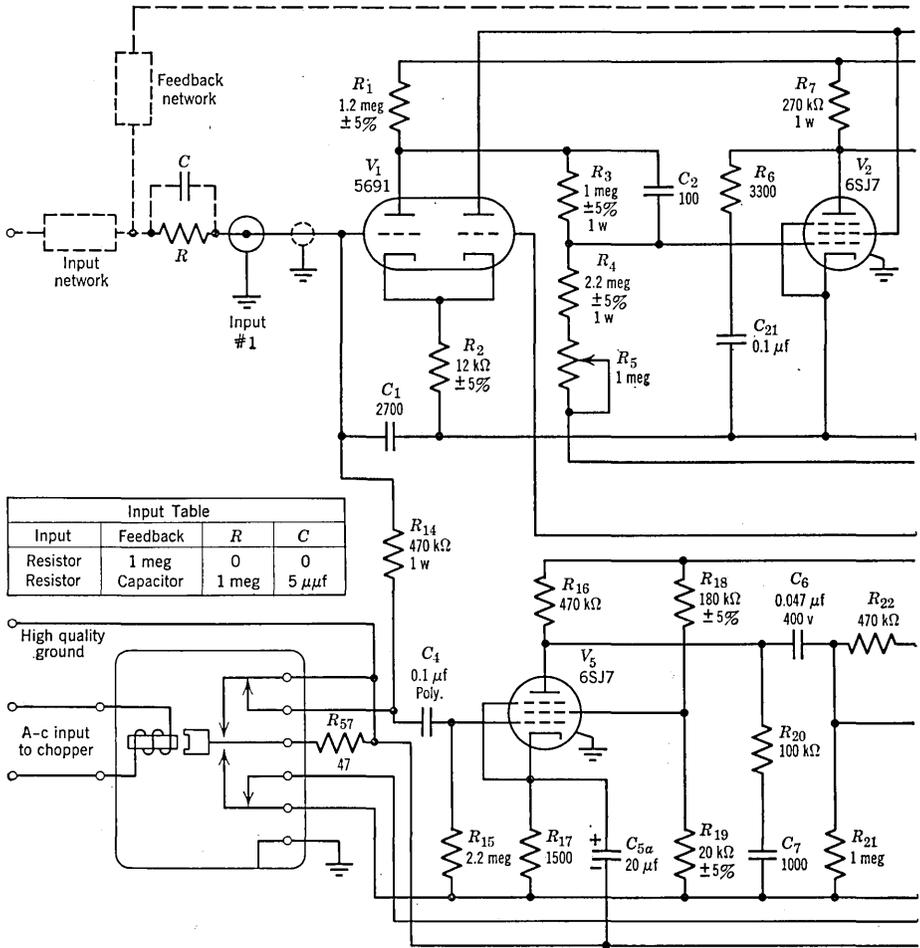
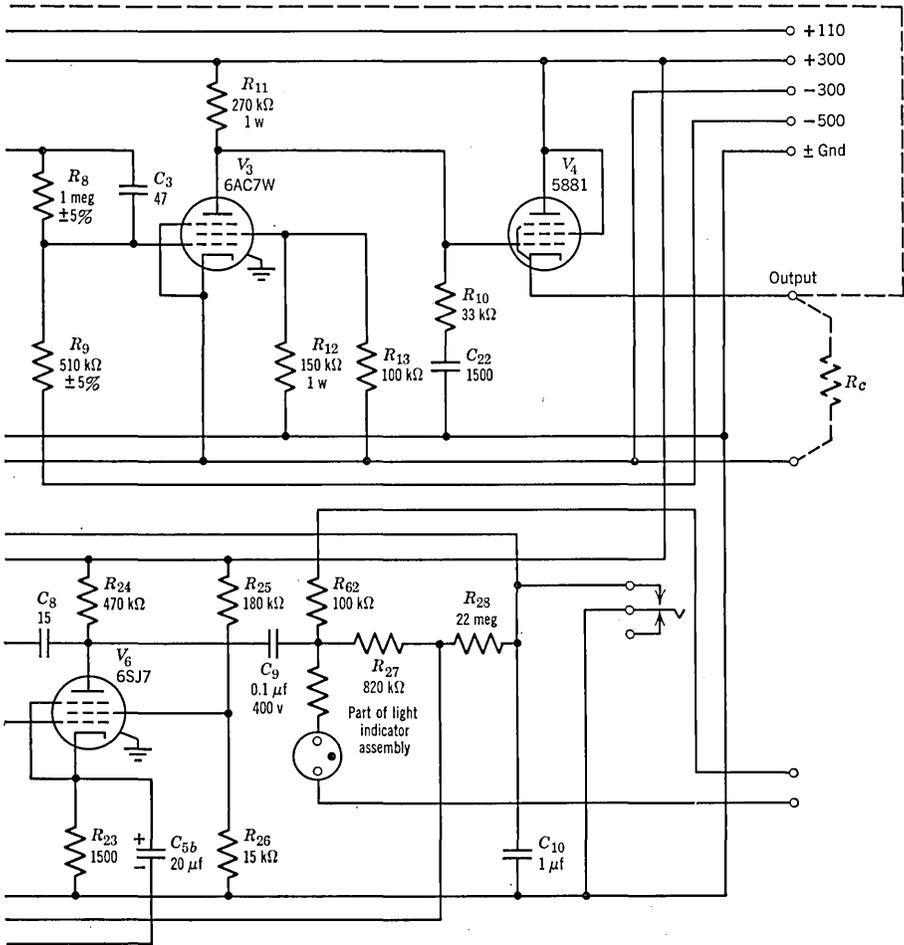


FIG. 14. A typical operational amplifier schematic diagram.



(Courtesy of Electronic Associates Inc., Long Branch, N. J.)

## 22-36 DESIGN AND APPLICATION OF ANALOG COMPUTERS

less than about 5000 ohms. The resulting gain error due to loading is considerably less than 0.01 per cent and is therefore negligible.

**Amplifier Time Constant Effects.** Actual amplifiers used in high quality analog computers have a closed loop bandwidth extending from dc to about 10 kilocycles. The effect of limited bandwidth may be represented approximately by including a small, low pass, time constant,  $\tau_A$ , in the amplifier transfer function

$$(33) \quad \frac{E_o(s)}{E_i(s)} = - \frac{Z_f(s)}{Z_i(s)} \cdot \frac{1}{1 + \tau_A s},$$

where  $\tau_A$  is of the order of 10 to 30 microseconds. The gain and phase errors contributed by this effect are generally significant only at reasonably high frequencies, above about 10 cycles per second.

**Integrator Time Constant Effects.** The time constant of an integrator consisting of an operational amplifier with capacitive feedback, is non-infinite because of two practical effects, capacitor leakage and noninfinite amplifier gain. The actual integrator transfer function is

$$(34) \quad Z(s) = - \frac{1}{RC} \frac{1}{s} \frac{\tau_I s}{\tau_I s + 1},$$

where  $1/RC$  is the integrator gain, and  $\tau_I$ , the integrator time constant, is given by

$$(35) \quad \tau_I = \frac{RC}{\frac{1}{A} + (R/R_l)}$$

$R_l$ , the capacitor shunt leakage, is of the order of  $10^{11}$  ohms for a high quality polystyrene capacitor of 1 microfarad. For a 1-megohm input resistance,  $R_l/R$  is of the order of  $10^5$ . Since the gain of a chopper stabilized amplifier at low frequencies is in the order of  $10^7$ , leakage is the dominant effect and  $\tau_I$  is of the order of  $10^5$  seconds. The computation errors due to  $\tau_I$  are quite negligible.

**Amplifier Drift.** As indicated in Sect. 7, the output drift voltage of a chopper stabilized amplifier, referred to an amplifier gain of unity, is of the order of 100 microvolts or less. The computational error resulting from this drift is generally quite negligible, except for very long computer runs involving open ended integrations.

**Amplifier Noise.** Random noise at the output of an operational amplifier exists at all frequencies passed by the amplifier. The rms value of this noise may be of the order of 10 millivolts or even greater. However, that portion of the noise within the usual computation frequency bandwidth is much less, perhaps of the order of 1 or 2 millivolts

rms. By proper scaling of a problem, it is almost always possible to minimize the effect of noise on computer accuracy.

**Recording Errors.** The accuracy of solutions of linear differential equations obtained with an electronic analog computer is frequently limited by the characteristics of the output or recording device. The most commonly used recording devices are the galvanometer type and the servo-driven type. Assuming that the time scale of the problem is properly chosen so that the dynamic characteristics of the recorder do not adversely affect the solution, static recording errors and reading errors will nevertheless deteriorate the accuracy in the order of 2 to 5 per cent for the galvanometer type recorder, and 0.1 to 0.5 per cent for the servo-driven recorder.

**Overall Accuracy.** Exclusive of recording error, the most important errors occurring in the solution of linear systems on a properly scaled analog computer result from potentiometer and amplifier gain errors. The error in each individual linear operation is of the order of 0.01 to 0.02 per cent. The overall error will grow in some manner with the size of the problem and the number of operations required. For many linear systems, the overall accuracy of solution is of the order of 0.1 to 1.0 per cent, which is compatible with the accuracy of the recording devices used, and is sufficient for most engineering purposes.

## REFERENCES

1. C. L. Johnson, *Analog Computer Techniques*, McGraw-Hill, New York, 1956.
2. G. A. Korn, and T. M. Korn, *Electronic Analog Computers*, 2nd edition, McGraw-Hill, New York, 1956.
3. C. A. A. Wass, *Introduction to Electronic Analogue Computers*, McGraw-Hill, New York, 1955.



## Nonlinear Electronic Computer Elements

*George A. Bekey*

1. Function Multipliers	23-01
2. Function Generators	23-14
3. Switching Devices	23-22
4. Trigonometric Devices	23-31
5. Time Delay Simulators	23-34
References	23-39

### I. FUNCTION MULTIPLIERS

**Introduction.** Chapter 22 describes linear operations such as addition, multiplication by a constant, and integration with respect to time. Nonlinear operations cannot be performed with amplifiers and potentiometers alone, but require specially constructed computer elements.

Many devices have been developed for performing multiplication of variable quantities in electronic analog computers. The most important of these in practical use are the electronic time division multiplier, the "quarter squares" multiplier, and the electromechanical servo multiplier. These are treated in detail in this section. Other types of multipliers are surveyed more briefly. A general list of symbols is given in Chap. 21, Sect. 6.

**Multiplication of Positive and Negative Numbers.** In general an electronic multiplier will have two input voltages  $x$  and  $y$ . The output will be a voltage proportional to the product.

$$(1) \quad z = kxy,$$

where  $k$  is a positive or negative constant.

## 23.02 DESIGN AND APPLICATION OF ANALOG COMPUTERS

*Four-quadrant multipliers* are devices that accept plus or minus for both  $x$  and  $y$  and give a product with the correct sign.

*Two-quadrant multipliers* permit one variable to change sign but not the other.

*One-quadrant multipliers* can accept one sign only for either  $x$  or  $y$ .

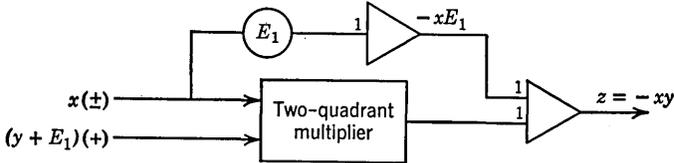


FIG. 1. Four-quadrant multiplier,  $E_1 \pm y > 0$ .

A two-quadrant multiplier may be made to behave as a four-quadrant device by adding a constant voltage to the variable that is not permitted to change sign.

EXAMPLE. Multiply  $\pm x$  and  $\pm y$  by using a two-quadrant multiplier. Let  $x$  be the variable that can assume plus and minus values for input. Choose  $E_1$  so that  $E_1 \pm y$  is always positive. Then the product

$$(2) \quad (x)(E_1 + y) = xE_1 + xy$$

will be formed as shown in Fig. 1, and the desired product is

$$(3) \quad z = xy = x(E_1 + y) - xE_1.$$

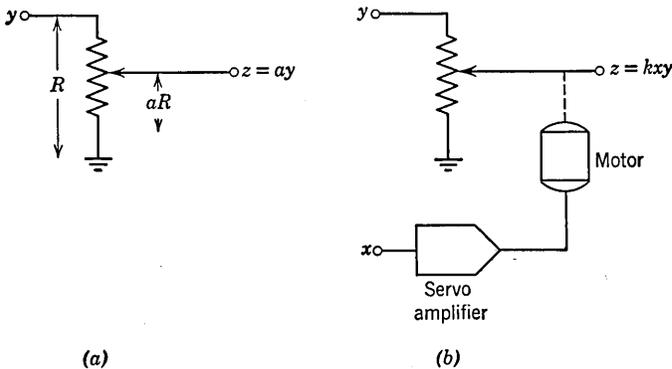


FIG. 2. Comparison of linear and servo-driven potentiometers.

### Electromechanical Servo Multipliers

Potentiometers are used to multiply by constants, since the voltage at the potentiometer arm represents a constant fraction of the total voltage applied across the potentiometer (see Fig. 2a). If the position

of the potentiometer arm is made proportional to another variable, the voltage appearing at the wiper will be proportional to the product of the voltage across the potentiometer and the wiper position, or

$$(4) \quad z = kxy,$$

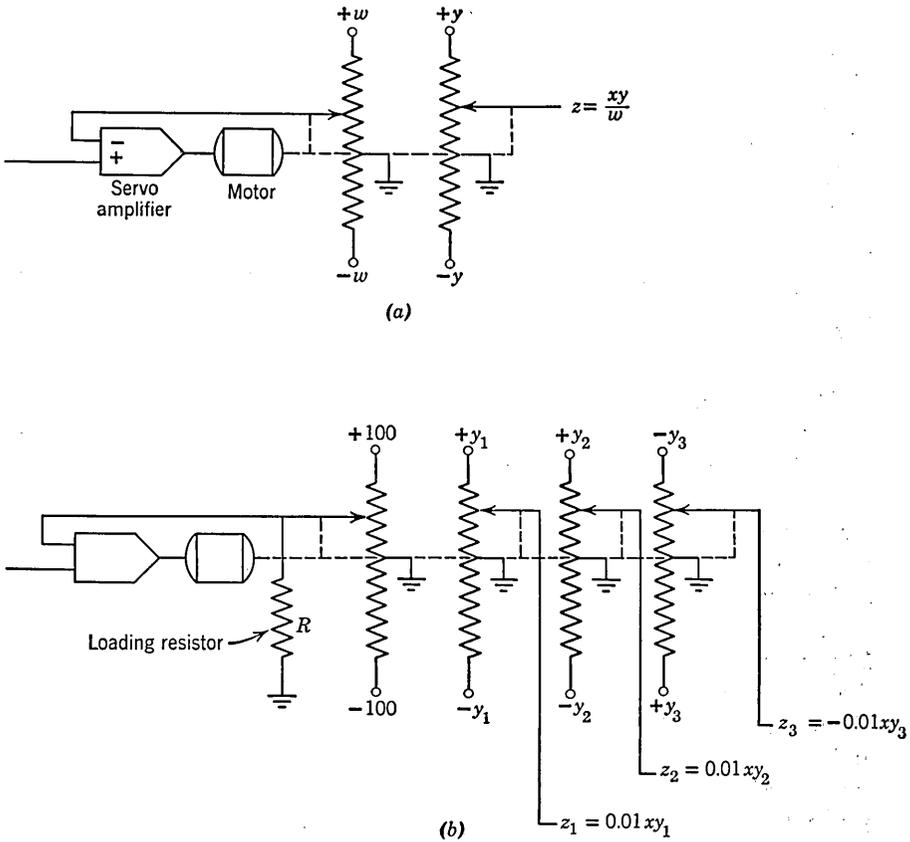


FIG. 3. (a) Basic servo multiplier. (b) Servo multiplier with multiple outputs.

where  $x$  is the wiper position and  $y$  is the voltage applied across the potentiometer, as illustrated in Fig. 2b. A practical servo multiplier is shown in schematic form in Fig. 3a in which  $x$  is the input voltage,  $\pm w$  is the constant voltage on the follow-up potentiometer, and  $\pm y$  is the voltage on the multiplying potentiometer. The follow-up and multiplying potentiometers are ganged together mechanically so that their positions always correspond. Then,

$$(5) \quad \frac{x}{w} = \frac{z}{y}, \quad z = \frac{1}{w} xy = kxy \quad \left( \frac{1}{w} = k = \text{constant} \right).$$

## 23-04 DESIGN AND APPLICATION OF ANALOG COMPUTERS

It is customary to set  $w = 100$  so that  $k = 1/w = 0.01$ . A practical servo multiplier is shown in Fig. 3b. Note the following:

(a) More than one product can be obtained with a single servo amplifier by mounting additional potentiometers on the shaft, thus obtaining a series of products  $kxy_1, kxy_2, kxy_3$ , etc.

(b) Four-quadrant multiplication is obtained by applying positive and negative voltages across the multiplying potentiometers, as illustrated in Fig. 3. In this case the grounded center taps are employed.

(c) Sign reversal of the output is obtained by reversing the polarities of the voltages applied to the multiplying potentiometers.

(d) Increased accuracy can be obtained in two-quadrant multiplication by removing the center taps and grounding one end of all the potentiometers. In this case the  $x$  input is restricted to one polarity and the greater accuracy results from increased resolution on the multiplying potentiometers.

(e) An indicating dial can be mounted on the servo shaft to provide a continuous measure of the input voltage.

**Limitations.** Owing to the presence of mechanical elements (motor and gear trains) the frequency response of a servo multiplier is severely limited. With 60 cps servos this response seldom exceeds a few cycles per second for full-amplitude input signals, and may reach 10 or 15 cps for low-level input signals. Considerable improvement in frequency response is achieved by the use of 400 cycle servo motors, since at a higher frequency less rotor material is required and thus the inertia of the motor is decreased. Servo multiplier response is often specified for 10 per cent amplitude signals. However, a low amplitude signal reduces the resolution available and consequently reduces the accuracy of multiplication. Overall accuracy can be improved by the use of highly stable and accurate reference voltages, larger diameter or multiturn potentiometers of high linearity, low-inertia servos, and by attention to loading errors (see below). Accuracies better than 0.05 per cent of full scale have been obtained with these devices.

**Loading.** Errors due to loading of potentiometers can be compensated by loading the feedback potentiometer with a resistance equal in value to the load on the multiplying potentiometers, as illustrated by the resistor  $R$  in Fig. 3b. It should be noted that:

(a) For proper loading compensation, all the multiplying potentiometers must connect to equal resistances, such as unity gain inputs of amplifiers.

(b) Since this method is one of compensation and not correction, the servo position (and thus the indicating dial) is no longer a correct indication of the input voltage.

(c) A multiplying potentiometer itself presents a variable load which depends on the servo position, and should therefore always be fed from a low-output impedance device, such as an amplifier, rather than directly from another potentiometer.

**Time Division Electronic Multipliers**

The basic principle of this type of multiplier consists of determining the average value of a voltage which consists of a series of modulated

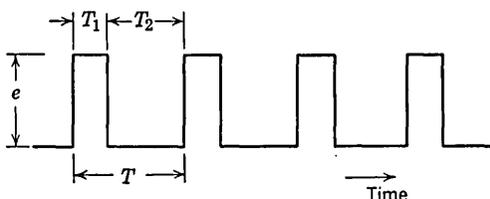


Fig. 4. Output pulse train in a time division multiplier.

rectangular pulses, such as shown in Fig. 4. With reference to this figure, the average value of the voltage is

$$(6) \quad e_{av} = \frac{eT_1}{T},$$

where  $e$  is the height of the pulse,  $T_1$  is its duration, and  $T$  is the period. Thus, if one of the input variables is made proportional to the pulse height, and the other to the ratio of pulse duration to cycle time, the average amplitude of the rectangular wave will be proportional to the product  $xy$ , since

$$(7) \quad x = k_1e, \quad y = k_2 \frac{T}{T_1}.$$

Hence 
$$z = kxy, \quad k = \frac{1}{k_1k_2}.$$

The words “time division” are due to the fact that one of the variables is used to control the ratio of on time to on-plus-off time during a cycle. In England the device is known as a “variable mark space multiplier” for the same reason (Ref. 4).

**Basic Description.** A practical time division multiplier is shown in block diagram form in Fig. 5a and the corresponding waveshapes in Fig. 5b. The operation of the device may be summarized as follows:

(a) The bistable multivibrator generates a series of gating pulses which control two electronic switches.

23-06 DESIGN AND APPLICATION OF ANALOG COMPUTERS

(b) When the switches are closed, the input to the integrator is  $(E_1 - E_R + y)$ .  $E_1$  and  $E_R$  are reference voltages. Scale factors are selected in such a manner that a linearly decreasing integrator output is

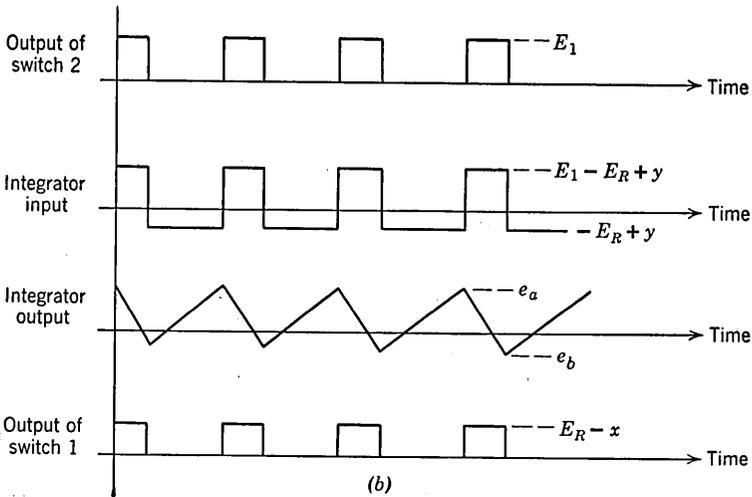
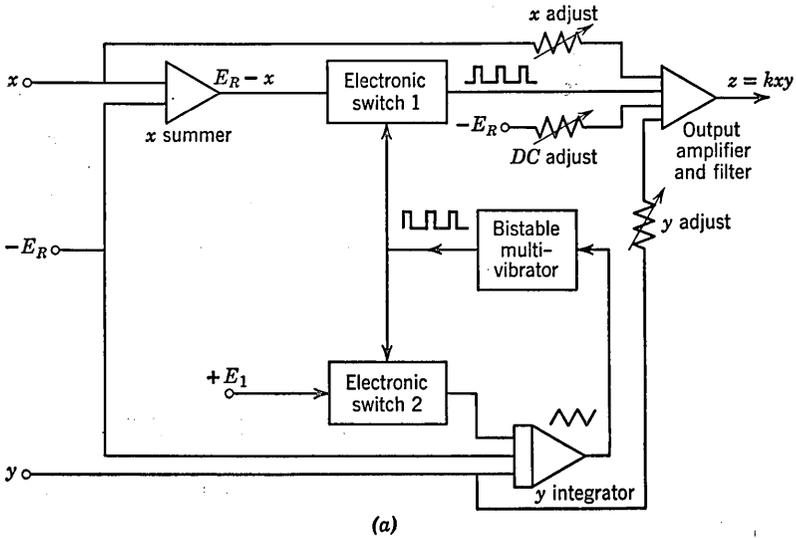


FIG. 5. (a) Block diagram of electronic time division multiplier. (b) Typical waveforms in a time division multiplier.

produced, with a slope proportional to  $-(E_1 - E_R + y)$ . When the integrator output reaches the switching level  $e_b$  of the multivibrator, the switches open (see Fig. 5b).

(c) With the switch open, the integrator input is  $(-\bar{E}_R + y)$ , and the output has a positive slope proportional to  $(E_R - y)$  which continues to rise until it reaches the multivibrator switching level  $e_a$ .

(d) The output of switch 1 is zero when the switch is open, and is proportional to  $(E_R - x)$  when the switch is closed, thus creating a series of output pulses as shown in Figs. 4 and 5b.

(e) The duration of the pulses is determined from their slope and amplitude, as follows (see Fig. 5b):

$$(8) \quad T_1 = \frac{k(e_a - e_b)}{E_1 - E_R + y},$$

$$(9) \quad T_2 = \frac{k(e_a - e_b)}{E_R - y},$$

where  $k$  is a constant of proportionality. The average amplitude of the output is therefore  $(e = E_R - x)$

$$(10) \quad E_{av} = \frac{eT_1}{T_1 + T_2} = \frac{1}{E_1} (E_R - x)(E_R - y).$$

Thus,

$$(11) \quad E_{av} = \frac{1}{E_1} (E_R^2 - E_Rx - E_Ry + xy).$$

In practice, the terms  $E_R^2$ ,  $E_Rx$ , and  $E_Ry$  are usually removed before filtering of the pulse train. As indicated in Fig. 5, the output amplifier and filter adds the correcting factors and filters the output to produce a voltage proportional to the product  $xy$ .

(f) The operation of the circuit as described above depends on maintaining the inputs  $x$  and  $y$  essentially constant over several cycles of operation. Thus, the repetition rate of the multivibrator must be considerably higher than the highest frequency of interest. Practical rates are of the order of 1 to 50 kilocycles.

**Accuracy.** The accuracy of a multiplier of the type shown in Fig. 5 is of the order of 0.1 to 1.0 per cent of full scale. Higher accuracies, of the order of 0.01 per cent are possible with chopper-stabilized amplifiers used for the summing, integrating, and output filtering functions; with higher repetition rates; and with the use of stabilized electronic switches. Electronic multiplier accuracies are quoted as percentage of full scale since they generally refer to a specific error in volts. Thus a multiplier with a quoted accuracy of 0.02 per cent can have a possible error of 0.04 volt. This voltage represents the error of 0.02 per cent with 100-volt inputs, but may represent a much greater percentage error if one or both inputs are near zero. Some commercial multipliers have a band-

width selector, which allows the multiplier to obtain maximum accuracy at a sacrifice in frequency response or vice versa. The bandwidth is thus dependent on accuracy and repetition rate. Practical values range from 50 cps to 2000 cps. During operation, the repetition rate of a multiplier of the type shown in Fig. 5 may vary over a ratio of 2 to 1. High-accuracy multipliers have also been built using a constant repetition rate. An excellent discussion of problems involved in multiplier design is given in Ref. 2, pages 275-280, and Refs. 5 and 6.

### Quarter Squares Multipliers

This type of multiplier is based on the relationship

$$(12) \quad z = xy = \frac{1}{4} [(x + y)^2 - (x - y)^2].$$

The main differences in the methods of mechanizing this equation come about from differences in the squaring circuits used.

**Triode Squaring Circuits.** Early models of quarter squares multipliers were based on the fact that the plate current of a triode is approximately proportional to the square of the plate voltage over a limited range of operation, i.e.,

$$(13) \quad i_p = ke_p^2.$$

A number of repetitive analog computer installations use multipliers based on this principle.

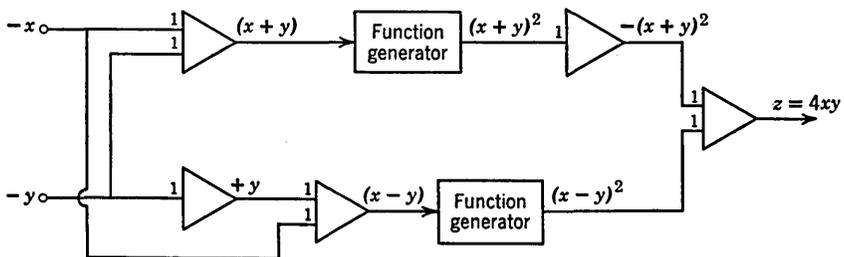


FIG. 6. Block diagram of quarter squares multiplier.

**Biased Diode Squaring Circuits.** Biased diode type function generators (see Sect. 2) can be used to perform accurate squaring of the input voltages, and thus, in conjunction with computer amplifiers, can be used to form a quarter squares multiplier as shown in Fig. 6 (Ref. 7).

#### Features.

(a) *Frequency Response.* Can be excellent, since it is not limited by mechanical elements as is the servo multiplier.

(b) *Accuracy.* Can be of the order of 0.1 per cent of full scale. The error can be quite large when the values of the variables are approximately equal and thus make the difference  $(x - y)$  small. No difficulty is experienced near zero for one of the variables, as is the case for certain models of the time division multiplier (see above).

(c) *Modifications.* The quarter squares multiplier is sometimes modified to obtain functions which are easier to set up on function generators than the square. One type of multiplier makes use of the functions  $U_1 + U_1^2/a$  and  $U_2 - U_2^2/a$ , where

$$(14) \quad U_1 = (x + y), \quad U_2 = (x - y), \quad a = \text{constant}$$

thus solving the relation:

$$(15) \quad z = xy = \frac{a}{4} \left\{ \left[ (x + y) + \frac{(x + y)^2}{a} \right] + \left[ (x - y) - \frac{(x - y)^2}{a} \right] - 2x \right\}.$$

**Other Types of Multipliers**

The three types of multipliers discussed above are those in most common use in large analog computer installations in the United States. A number of other devices are briefly mentioned below. More extended discussions and further references can be found by consulting the references at the end of the chapter (in particular, Refs. 2 and 8).

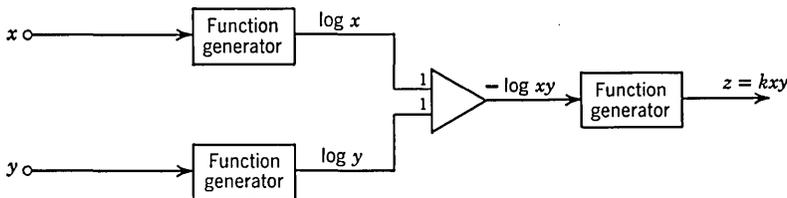


FIG. 7. Block diagram of logarithmic multiplier.

**Logarithmic Multiplier.** Function generators can be used for multiplication by using the rule for the logarithm of a product,

$$(16) \quad \log xy = \log x + \log y,$$

and therefore

$$(17) \quad z = xy = \text{antilog} (\log x + \log y).$$

as indicated in block diagram form in Fig. 7.

**Dynamometer Multiplier.** Based on the principle of operation of a dynamometer type measuring instrument, in which the torque  $T$  acting on the needle is proportional to the product of the two coil currents  $I_1$

## 23-10 DESIGN AND APPLICATION OF ANALOG COMPUTERS

and  $I_2$  if the angle of displacement is small, i.e.,

$$(18) \quad T = kI_1I_2.$$

A feedback circuit which detects shaft motion creates an equal and opposite torque to keep the angular displacement small. Overall accuracy, about 1 per cent.

**Crossed Fields Multiplier.** This device operates on the crossed fields in a cathode ray tube, and is illustrated in Fig. 8. A voltage proportional to  $x$ , applied to the horizontal deflection plates, gives the cathode ray beam a horizontal component of velocity. In the presence of a concentric magnetic field proportional to the voltage  $y$ , the cross product

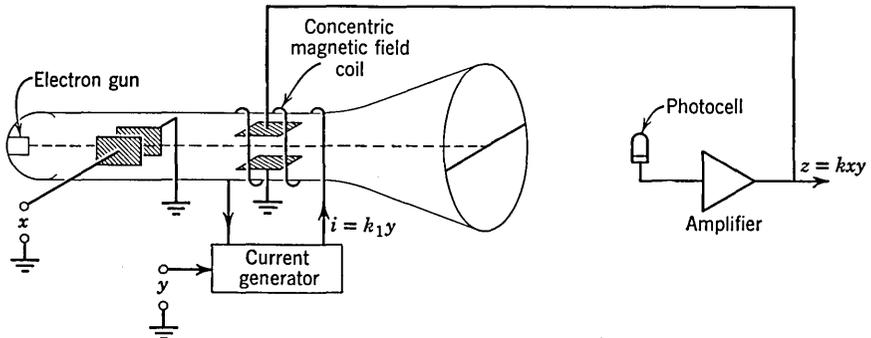


FIG. 8. Crossed fields multiplier.

of horizontal velocity and magnetic field deflects the beam in a vertical direction. This vertical deflection is detected by an error amplifier which applies a voltage to the vertical plates which keeps the beam horizontal. The governing equation is, in vector form:

$$(19) \quad \mathbf{f} = \mathbf{B} \times \mathbf{v}$$

which reduces to  $z = ei = kxy$ . Accuracy of the order of 0.5 per cent and frequency response of 3000 cycles per second have been obtained with this multiplier (Ref. 9).

**AM-FM Multiplier.** In this scheme one of the variables controls the carrier amplitude into an FM discriminator, while the other variable affects the carrier frequency through a frequency modulator as indicated in block diagram form in Fig. 9. Accuracy of 0.5 per cent has been obtained with a frequency response of several kilocycles (Ref. 10).

**Step Relay Multiplier.** This device is basically a variable gain amplifier, thus obtaining a product of input voltage and gain. The variable gain is achieved by using relay-controlled precision attenuator networks actuated by a binary counter circuit. While the cost of this type of

multiplier is quite high, accuracy of 0.02 per cent has been reported (Ref. 11).

**Probability Multiplier.** The operation of this device is based on a theorem in probability theory which states that the probability of simultaneous occurrence of two events which have a random distribution in time is equal to the product of their separate probabilities. This principle can be mechanized by generating two trains of pulses the durations

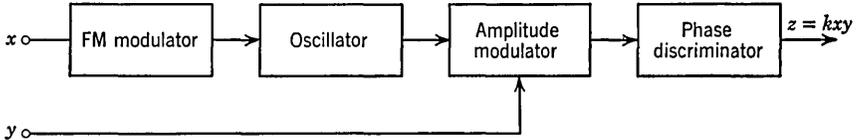


FIG. 9. Block diagram of AM-FM multiplier.

of which are proportional to the variables to be multiplied. If there is no integer relation between the two frequencies, the product can be obtained by detecting the coincidence of the pulse trains (Ref. 12).

**Use of Multipliers**

**Scaling.** Most commercial multipliers introduce a scale factor of 1/100 into the computation of a product, so that, if  $z = xy$

$$(20) \quad k_z = \frac{k_x k_y}{100},$$

where  $k_z$ ,  $k_x$ , and  $k_y$  are the scale factors of the variables  $z$ ,  $x$ , and  $y$  respectively. Consequently, care must be taken when the output of a multiplier is required at a high scale factor. Amplification of multiplier outputs is almost always undesirable, since noise levels are generally higher at the output of the multiplier than elsewhere in the computer.

**Powers.** Any multiplier can be used to obtain powers of the input variables by successive multiplication. A single servo multiplier can be used to obtain higher powers by applying the output of a multiplication across another multiplying potentiometer, while loading and scale factors are carefully observed (see Fig. 10). It should be noted that powers of the independent variable time can be obtained simply by successive integration, since

$$(21) \quad 2 \int t dt = t^2,$$

$$3 \int t^2 dt = t^3,$$

and so forth.

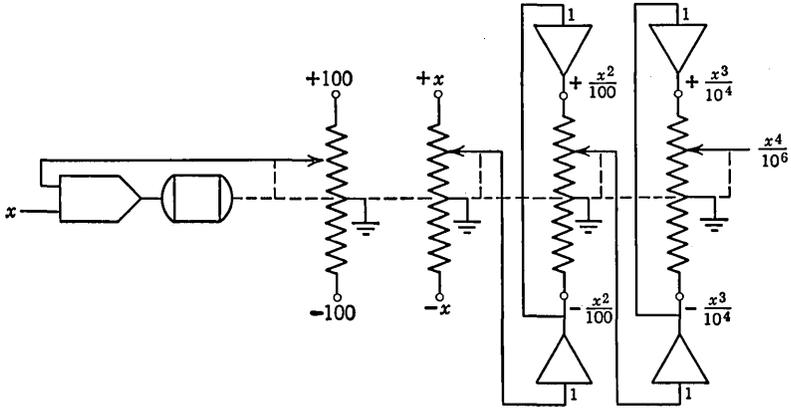


FIG. 10. Higher powers using a servo multiplier.

**Division.** Division can be performed by using multipliers in the feedback circuit of a high-gain operational amplifier, and thus solve an implicit equation of the form

$$(22) \quad x + zy = 0,$$

so that

$$(23) \quad z = -\frac{x}{y}.$$

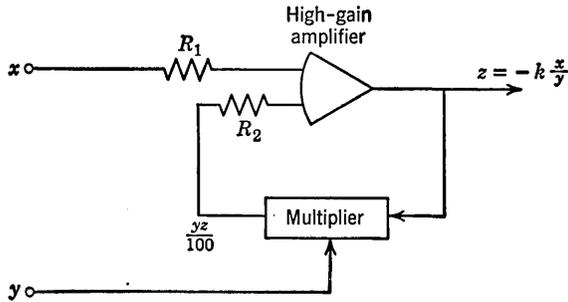


FIG. 11. Division by inverse multiplication.

Such a circuit for division is shown in Fig. 11. Similar circuits can be used for extracting square roots, cube roots, and for other operations. The following should be noted in connection with Fig. 11:

(a) Operation is limited to two quadrants. This can be noted by observing that negative feedback is required for stability in the circuit of Fig. 11. If the multiplier introduces a sign change into the compu-

tation, the situation is as follows. Assume that  $x$  is positive and  $y$  is negative. Then the output of the high-gain amplifier  $z$  will be negative and the output of the multiplier,  $yz$ , will be negative thus resulting in a feedback signal, which will cancel the original input. If  $y$  were positive the multiplier output would also be positive and so would give positive feedback and instability. Thus for this situation the  $x$  input to the divider can be either positive or negative, but the  $y$  input must be negative. If the multiplier introduces no sign change, the  $y$  input is restricted to positive values for stability.

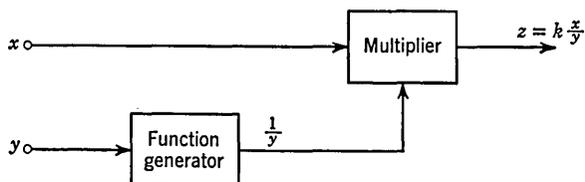


FIG. 12. Division by reciprocal multiplication.

(b) Care must be taken with a division circuit if the denominator approaches zero.

(c) Because of the high gain of the system, noise is considerably amplified at the output of a division circuit. Operation can be improved by filtering out high frequencies with a small condenser across the amplifier, if an electronic multiplier is utilized in the feedback. Servo multipliers have sufficient mechanical inertia to enable them to act as a low-pass filter.

Division is also possible by using a function generator to obtain the reciprocal and then multiplying, as shown in Fig. 12.

A servo multiplier can be used for division directly by applying a variable voltage to the feedback potentiometer. However, since the gain of the feedback loop depends on the voltage applied to the feedback potentiometer, the servo divider is unsatisfactory unless some form of automatic gain control (AGC) is used in the feedback loop.

**Integration with Respect to a Dependent Variable.** Multipliers are often used for this operation, which may be performed on an electronic analog computer by the relation

$$(24) \quad \int y \, dx = \int \left( y \frac{dx}{dt} \right) dt,$$

since the electronic integrator is capable only of integration with respect to time.

## 2. FUNCTION GENERATORS

Many nonlinear problems require the use of information which is available in graphical form, such as the results of experimental work, certain nonlinear parameters, or arbitrary functions of the dependent variables. These functions are usually introduced into the computer by one of the following techniques: (a) curve follower devices, (b) biased diode function generators, (c) servo-driven tapped potentiometers, (d) photoformers.

### Curve Follower Devices

**Manual.** An arbitrary function can be introduced into the computer by hand-following a drawn curve. The abscissa is servo-driven in accordance with a particular computer variable and a sight is positioned on the curve by the operator using a hand crank.

**Advantages.** (a) Simplicity, (b) direct utilization of graphical information.

**Disadvantages.** (a) Slow speed, (b) limited accuracy, (c) human tendency to overcompensate for errors.

**Photoelectric.** The above process can be mechanized by using an automatic curve follower device which tracks a black line on white paper (or the edge between a black and white boundary). Its *main disadvantages* are that it is complex and difficult to adjust, and it cannot return to the curve if accidentally displaced.

**Potentiometer Type.** If the function to be generated is represented by a suitably shaped wire which makes contact at one point with a linear potentiometer, as illustrated in Fig. 13, the voltage obtained is proportional to the ordinate of the curve. In Fig. 13 the wire curve is wrapped on the surface of a rotating cylinder, and thus simplifies contact problems and makes repetitive operation possible. The wire curve is usually prepared by cementing the wire directly over a drawn curve. Its *main disadvantage* is that a great deal of care and adjustment is required for proper operation. Obtainable accuracy depends on the linearity of the potentiometer and the resolution obtainable on a given size of input curve.

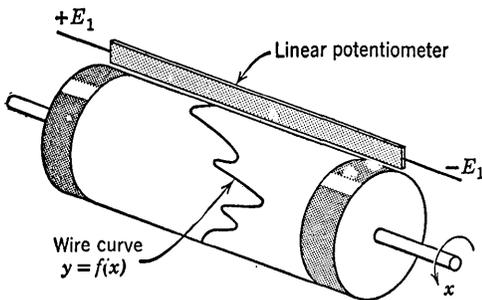


FIG. 13. Drum type curve follower function generator.

**Conducting Ink Curves.** A variation of the device above makes use of a curve drawn on paper with conducting ink and excited with a high-frequency voltage, thus enabling a pickup coil to position itself above the curve. The voltage signal from the pickup head is demodulated and used as an input to the servo amplifier which drives the motor to position it.

*Advantages.* (a) Makes use of graphical information directly; (b) can find the line if displaced.

*Disadvantages.* (a) Limited speed (bandwidth) on account of mechanical elements; (b) accuracy limited by thickness of conducting line.

*Note.* Most curve follower type function generating devices can also be used as output recorders if the follower heads are replaced with an appropriate pen or stylus.

### Biased Diode Function Generators

By means of diodes it is possible to approximate a desired curve by a series of straight line segments, thus producing a function generator without the mechanical elements of the curve follower devices discussed above.

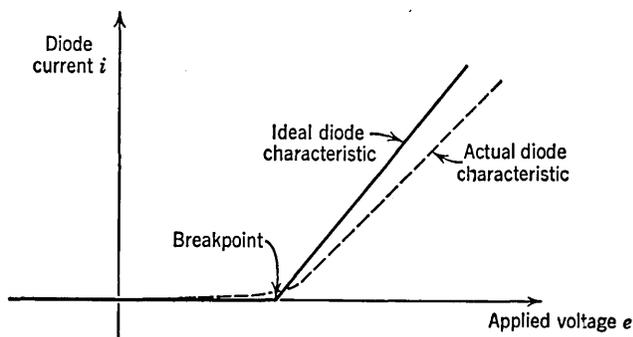


Fig. 14. Typical thermionic diode characteristics.

**Basic Theory.** Figure 14 is a representation of the switching characteristic of a thermionic diode. An ideal diode can be considered to be an on-off switch. By suitable choice of bias voltages and circuit resistors, the location of the breakpoint and the slope of the conduction line can be varied. If a number of diodes are biased in such a way as to begin conduction in succession, an arbitrary curve can be approximated by a series of straight line segments (see Ref. 13).

**Practical Circuits.** Two stages of a typical variable-breakpoint diode function generator are shown in Fig. 15. The portion of the circuit



### Tapped Potentiometer Function Generators

A potentiometer will perform linear interpolation between two values applied as voltages across it, if the potentiometer arm is connected to an infinite impedance. Based on this principle it is possible to construct a function generator from a potentiometer with a large number of taps which are supplied with appropriate voltages. If the wiper is driven in accordance with an input variable  $x$ , the wiper voltage  $y$  will result from interpolation between each pair of tap voltages.

**Construction of a Practical Device.** Multitap potentiometers are often included with a servo multiplier (see Sect. 1) and a wiper displacement proportional to the input variable is thus obtained. Figure 16a illustrates the simulation of a particular function, and Fig. 16b shows a practical circuit for a potentiometer padder system. In Fig. 16, the following should be noted:

(a) Each padding potentiometer can be connected to the plus or the minus reference voltage  $E_R$ , it can be left open, or grounded (see Fig. 16b).

(b) Isolation amplifiers are provided to prevent interaction between tap voltages during setup. Unless such isolation is provided, the taps will have to be set several times until the voltages converge to the desired values.

(c) The taps are set in sequence in order to apply the correct voltage to the next tap to be adjusted.

(d) Fixed resistor networks can be used to pad the tapped potentiometer for a particular function. These networks can be stored and re-used.

(e) On account of loading effects the interpolation between tap voltages is not linear, and the curve is approximated by a series of curved line segments, as shown in Fig. 16a.

#### Features.

*Advantages.* (a) Can be combined with a servo multiplier. (b) Adjustment networks can be stored and re-used. (c) Accuracy of the order of 0.1 per cent can be obtained.

*Disadvantages.* (a) Output has slope discontinuities. (b) Frequency response is limited because of mechanical elements.

**Nonlinear Potentiometers.** A potentiometer can be used to generate many functions directly, without the necessity of padding a series of taps, if the winding is nonlinear and corresponds to the function being simulated. Such nonlinear potentiometers have been wound for trigonometric, exponential, and logarithmic functions (see Sect. 4).

## 23-18 DESIGN AND APPLICATION OF ANALOG COMPUTERS

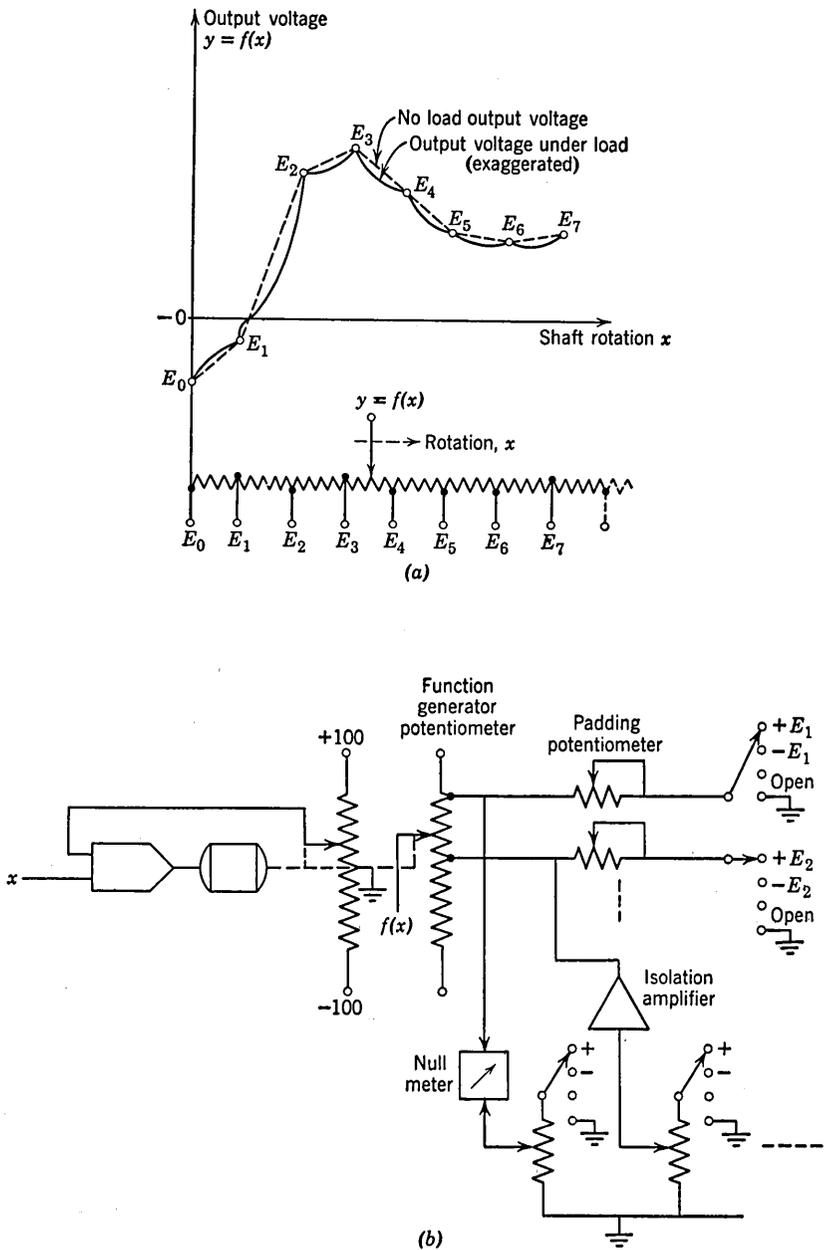


FIG. 16. (a) Simulation of a function with a padded potentiometer. (b) Simplified schematic of potentiometer padder system. Two padders are shown; others may be added.

**The Photoformer**

In this device the spot on the face of a cathode ray tube is made to follow an opaque mask which represents the function being generated, and thus a function generator of excellent frequency response which is particularly suitable for repetitive analog computers is obtained.

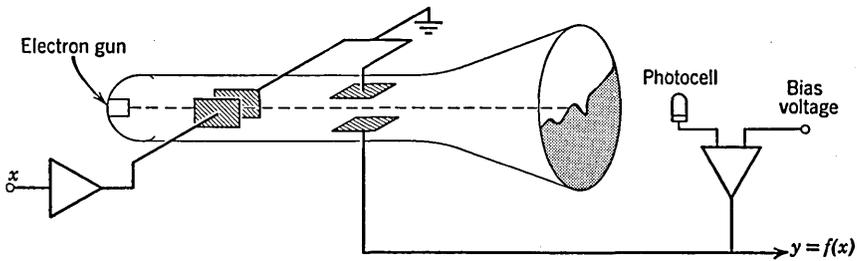


FIG. 17. Block diagram of photoformer.

**Construction Features.** A typical photoformer circuit is indicated in block diagram form in Fig. 17, and its operation can be summarized as follows:

- (a) A mask is prepared as a plot of the function  $f(x)$  desired, and the lower area of the mask below the curve made opaque.
- (b) The input voltage  $x$  is applied to the horizontal deflection plates and thus produces a horizontal beam displacement which is proportional to  $x$ .
- (c) The vertical position of the beam is made to follow the edge of the beam by the combination of photocell detector and biased amplifier which drive the vertical deflection plates of the tube.

**Features.**

*Advantages.* (a) Excellent frequency response; (b) masks can be stored and re-used.

*Disadvantages.* (a) Accuracy limited by size of face on cathode ray tube and spot size necessary for satisfactory operation; (b) distortion of the function may result from nonlinearities in the tube deflection circuits and the fact that horizontal and vertical deflection plates may not be exactly perpendicular.

**Function Generator Applications**

**Scaling.** The following points should be noted when function generators are used to include arbitrary functions in an analog problem:

- (a) Maximum accuracy is obtained with as wide a range of voltages as possible.

## 23-20 DESIGN AND APPLICATION OF ANALOG COMPUTERS

(b) Wide voltage ranges may exceed the slope limitations of certain function generators, in which case vertical scales need to be reduced and horizontal scales increased.

(c) If the range of variation of dependent or independent variable is limited, it is frequently possible to bias and amplify this variable in such a way as to make possible utilization of the full range of the function generator.

**Uses.** Function generators are among the most useful and versatile components available with an analog computer. Some of the uses are:

(a) Introduction of empirical data into the computer.

(b) Generation of analytic functions which may be difficult to obtain with other components, such as certain trigonometric functions or combinations of transcendental functions. Function generators can be used to obtain higher powers of variables.

(c) Where multipliers are in short supply, quarter squares multipliers can be synthesized at the patchboard by using function generators and amplifiers.

(d) Machine variables can be transformed to logarithms for easier computation in certain problems.

(e) Division can be facilitated by obtaining the reciprocal of a quantity by using a function generator and then multiplying.

### Generation of Functions of Two Variables

In many physical problems it is necessary to consider arbitrary functions which depend on more than one variable, such as  $f(x,y)$ , where both  $x$  and  $y$  are functions of the independent variable. If the function is analytic, for *example*,

$$(25) \quad f(x,y) = \sin(x + y),$$

it is possible simply to add  $x$  and  $y$  and obtain the sine of the sum by using a function generator or servo resolver (see Sect. 4). The following sections deal with several techniques of obtaining arbitrary functions of two variables, when the desired function cannot be obtained as a combination of functions of a single variable. For an excellent discussion of function generation see Ref. 14.

#### Interpolation Techniques.

(a) *Tapped potentiometers.* If the taps on the potentiometers discussed above are supplied with voltages proportional to  $f(x,y_1)$ ,  $f(x,y_2)$ , etc., where the  $y_k$  ( $k = 1, 2, \dots$ ) are chosen so as to coincide with the location of the taps on the potentiometer, and if the wiper displacement is made proportional to  $y$ , the output voltage will be the required function of two variables,  $f(x,y)$ . The accuracy of the method is dependent on

the accuracies of the voltages supplied to the taps and on the number of taps (see Fig. 18). The functions  $f(x, y_k)$  may be the outputs of other function generators.

(b) *Resistive materials.* Two-dimensional field effects can be represented conveniently by using an electrolytic tank or resistive sheet analogy. A servo-positioned probe is used to pick up an electrical signal in the field, which will be a function of its vertical and horizontal position. A more general function of two variables can be represented by a

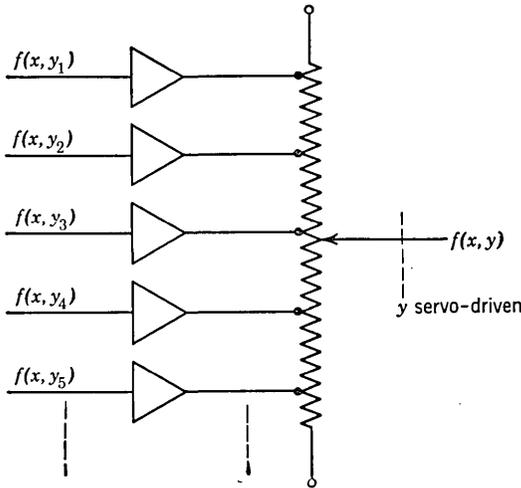


FIG. 18. Generation of function of two variables using a tapped potentiometer.

resistive sheet if lines corresponding to constant value of the function are drawn on the sheet with a conductive material and excited by a voltage proportional to the value of the function (see Fig. 19). The pickup probe is then positioned in the  $x$  and  $y$  directions by a servo system (a standard  $x$ - $y$  plotter may be used), and the probe will pick up a voltage proportional to the function of  $x$  and  $y$  being considered. The linearity of interpolation is dependent on the linearity of the resistive material between the lines and on the input impedance of the probe circuit.

**Variable Reference Diode Function Generators.** Biased diode function generators can be made useful for the generation of functions of two variables. For certain classes of functions a good representation can be obtained by replacing the reference voltage of the function generator with a variable voltage  $y$ . Then an output is obtained for each value of  $y$  and the breakpoints are proportional to this value; a repre-

## 23-22 DESIGN AND APPLICATION OF ANALOG COMPUTERS

sentation, as shown in Fig. 20, is thus obtained. A more general function of two variables can be obtained if the bias voltages of the individual diode circuits are made functions of  $y$ , as discussed in Ref. 15.

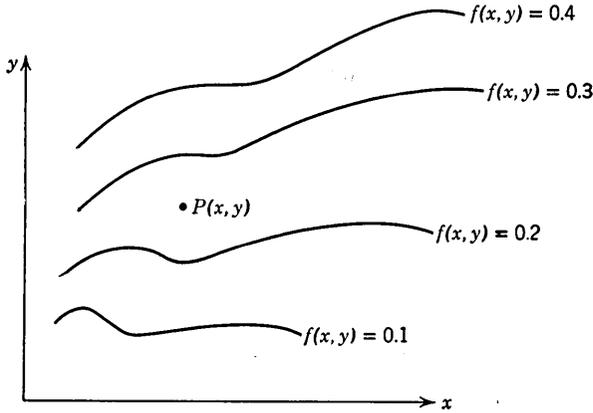


FIG. 19. Resistive sheet representation of a function of two variables.

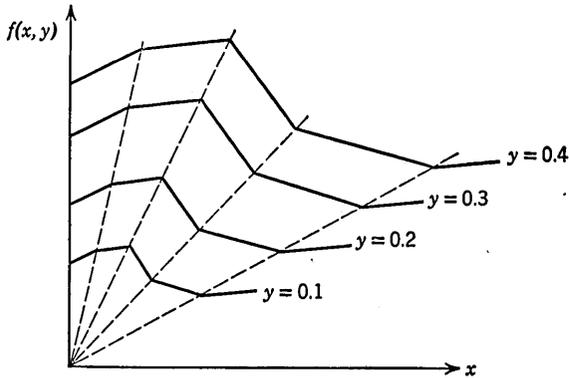


FIG. 20. Generation of function of two variables with variable reference diode function generator.

**Variable Density Film.** A number of techniques which utilize film of variable optical density are being investigated. Opacity variations of 200 to 1 ratio can be detected. The film density at any one point on a sheet of film can be a function of the abscissa and ordinate at that point, and thus represent a function of two variables (Ref. 16).

### 3. SWITCHING DEVICES

In the study of physical phenomena it is often necessary to include the effects of quantities which are defined differently in different regions

and thus show discontinuities in slope. Among these are such nonlinear phenomena as limiting, backlash, and coulomb friction. This type of phenomenon is usually represented on an analog computer by means of diodes or polarized relays, both of which approximate ideal on-off switches in their action (Ref. 17).

### General Features

Since relays and diodes are used to represent functions with slope discontinuities, they should have as sharp a switching action as possible. The features of each can be summarized as follows:

#### Relays.

*Advantages.* (a) Switching action is true on-off action, (b) three stable states are possible with a polarized relay.

*Disadvantages.* (a) Finite closing time introduces an error into many problems, (b) high sensitivity is required for fast action, (c) special amplifiers may be required for driving polarized relays.

#### Diodes.

*Advantages.* (a) Low cost and ready availability, (b) simplicity of application, (c) no mechanical motion in switching action.

*Disadvantages.* (a) Imperfect switching action, (b) finite back resistance, (c) nonzero forward resistance.

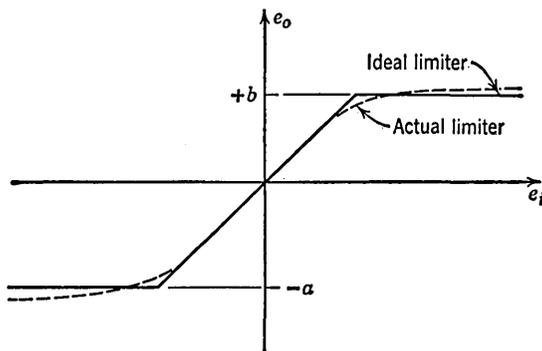


FIG. 21. Diode limiter characteristic.

**Limiting Operations.** In many mechanical systems the motion is limited by physical constraints. A typical limiting operation is shown in Fig. 21, where the dotted line represents the limiting obtained with diodes. The lack of sharpness in clipping is due to diode contact potential, as may be seen in a representation of diode characteristics, such as Fig. 14.

**Computer Representation.** The limiting operation can be stated

## 23-24 DESIGN AND APPLICATION OF ANALOG COMPUTERS

mathematically as:

$$(26) \quad \begin{aligned} e_o &= -a, & e_i &< -a; \\ e_o &= e_i, & -a &< e_i < b; \\ e_o &= b, & e_i &> b. \end{aligned}$$

This circuit, shown in schematic form in Fig. 22, operates as follows:

(a) If  $-a < e_i < b$ , neither diode conducts, and  $e_o = e_i$ .

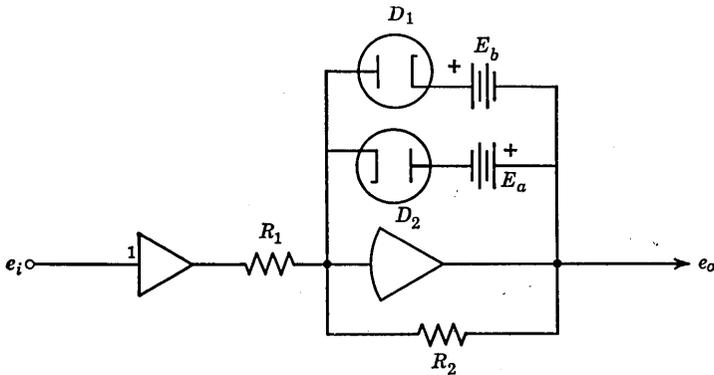


FIG. 22. Shunt output limiter circuit.

(b) If  $e_i < -a$ , diode  $D_1$  conducts and the output voltage is limited by the battery or floating d-c supply in series with  $D_1$ .

(c) If  $e_i > b$ , diode  $D_2$  conducts and the output is limited by the voltage in series with  $D_2$ .

The limiting action is due to the conducting diode and voltage source acting as a low-impedance feedback path for the operational amplifier; this reduces the gain until the output voltage is reduced to the bias level.

**Generalized Limiting Circuit.** Insertion of resistances  $R_1$ ,  $R_2$ ,  $R_3$ , and  $R_4$  into the circuit of Fig. 22 to give that of Fig. 23, makes it possible to vary the slopes of all three line segments of the output and thus give the response of Fig. 24.

*Input Shunt Limiter Circuit.* Both the circuits of Fig. 22 and Fig. 23 apply limiting to the output of an amplifier. It is possible to limit the input of an amplifier with a pair of diodes to obtain similar results, but generally this also reduces sharpness in clipping action.

*Idealized Limiter.* The use of a more complex circuit which uses four diodes and three amplifiers produces a considerable increase in the sharpness of the clipping action (see Fig. 25), since the high-gain amplifier makes operation of the circuit independent of diode characteristics.

**Limiting Integrator Outputs.** The limiting circuits discussed above cannot be applied directly to the output of an integrator without caution, since such limiting would result in an incorrect value of the derivative

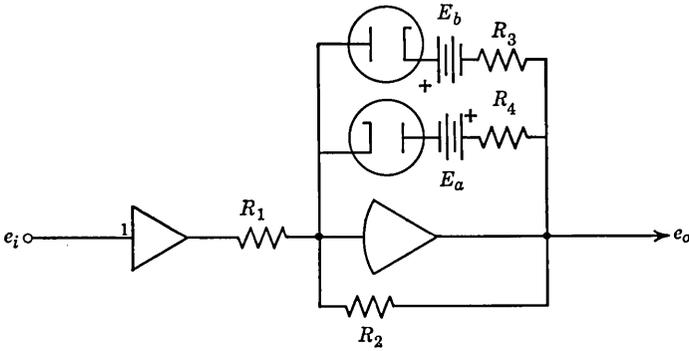


FIG. 23. Generalized shunt output limiting circuit.

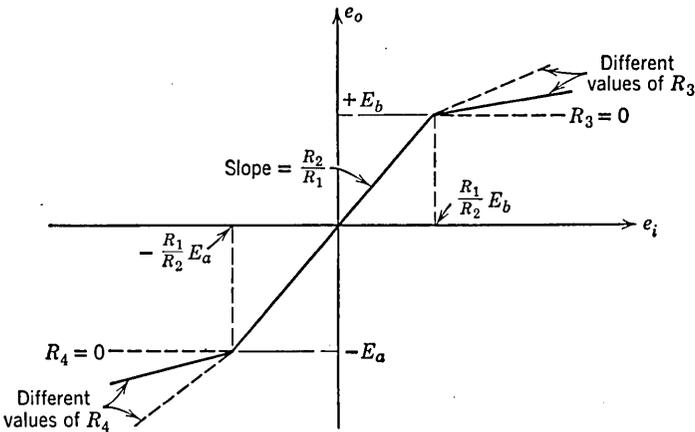


FIG. 24. Response of limiting circuit of Fig. 23.

and consequent wrong results. In general, the differential equation governing the integrator in question changes during the limiting period and the circuit needs to be adjusted accordingly (see Ref. 1, page 121).

**Simulation by Switching Devices**

**Dead Zone Simulation.** The simulation of a dead zone, i.e., no driving action until the input exceeds certain limits, is performed by using the circuits of Fig. 26. It should be noted that this circuit resembles

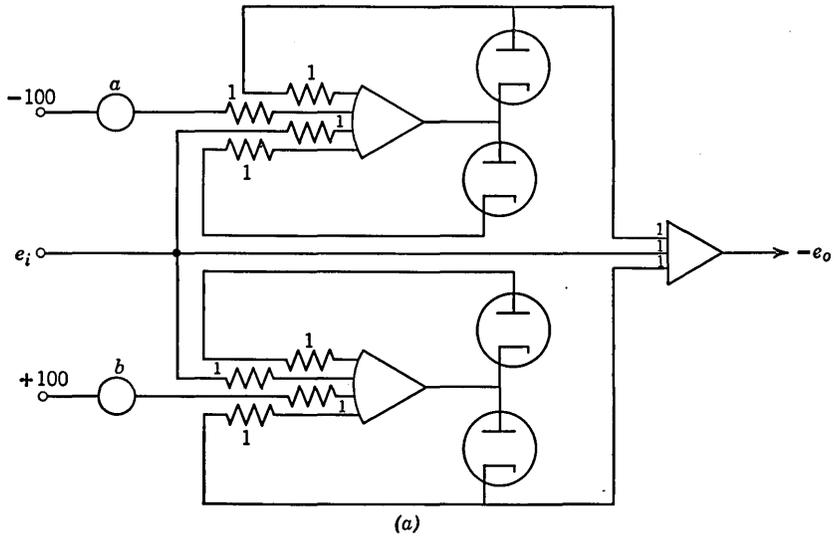


FIG. 25. Idealized diode limiter circuit and its response.

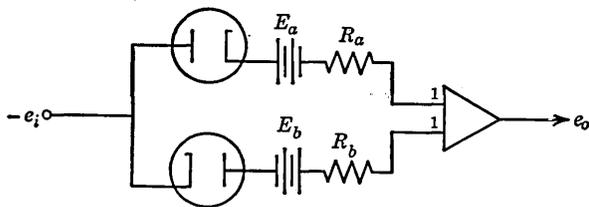


FIG. 26. Diode circuit for dead zone simulation.

very closely those used for limiting operations, but that the diodes are placed in series with the amplifier and thus the effect shown in Fig. 27 is produced.

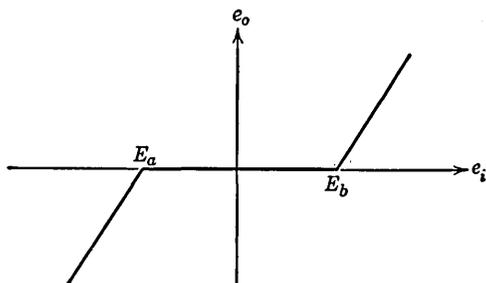


FIG. 27. Response of dead zone circuit of Fig. 25.

**Coulomb Friction Simulation.** "Dry," "solid," or coulomb friction is simulated by a variation of the limiter circuits above which omit the feedback resistor, and thus make the output of the amplifier very large for even small variations from zero at the input. The output is then limited as above and gives the response shown in Fig. 28.

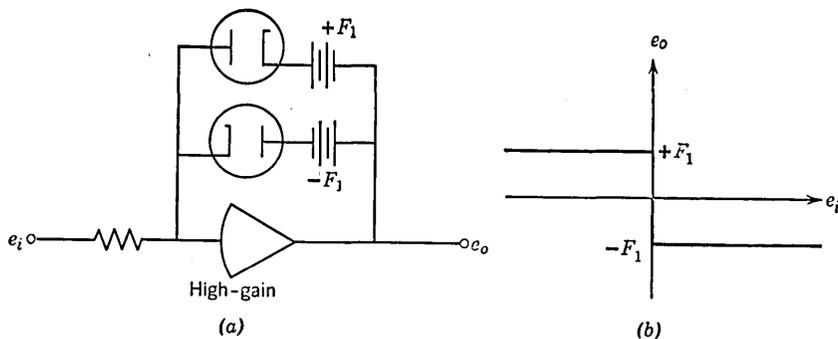


FIG. 28. Simulation of coulomb friction.

**Absolute Value of a Quantity.** A circuit commonly used to obtain the absolute value is shown with its response in Fig. 29. This circuit is used for the simulation of a full-wave rectifier.

**Simulation of Backlash.** Backlash or lack of driving action occurs in gear reversal and other reversing physical phenomena. Two diode circuits for simulation of backlash are shown with their response in Fig. 30. Two alternative circuits are shown to indicate that none of the

23-28 DESIGN AND APPLICATION OF ANALOG COMPUTERS

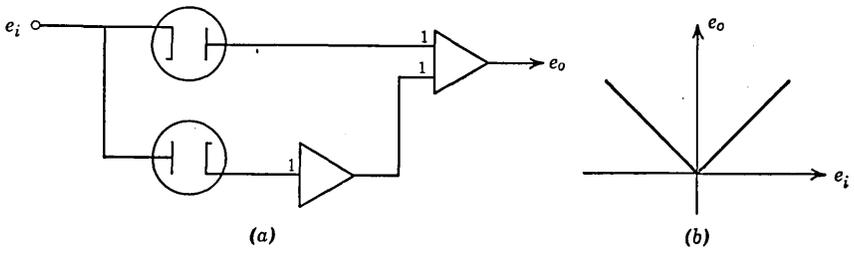


FIG. 29. Absolute value circuit and its response.

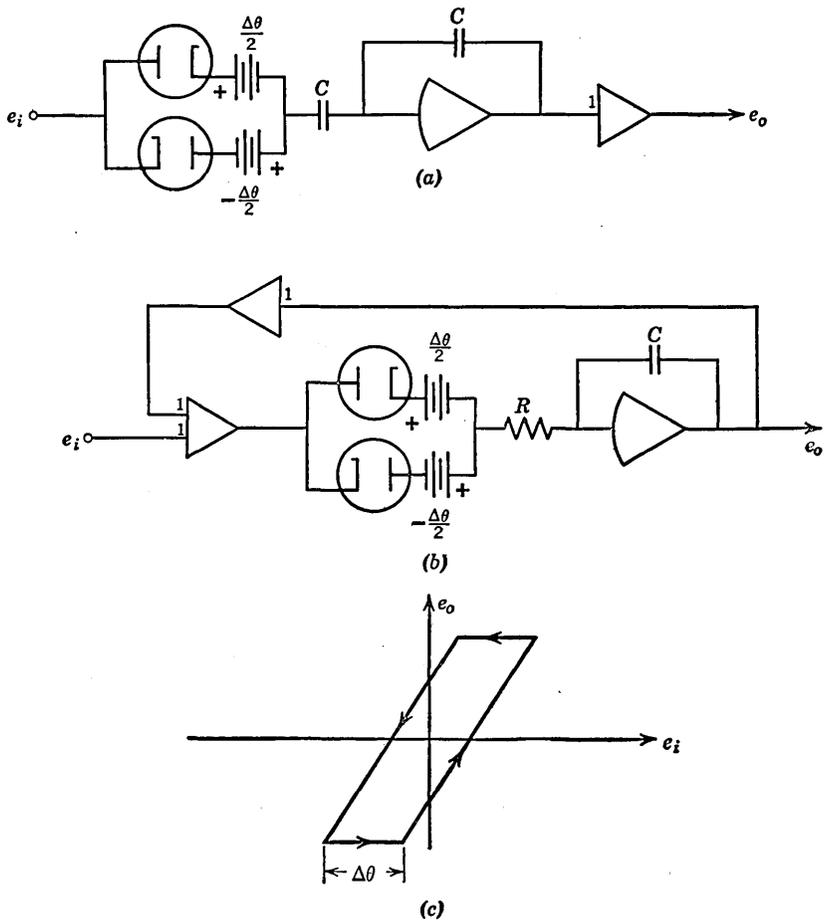


FIG. 30. Diode circuits for backlash simulation and their response.

circuits discussed above is unique, but that other schemes may be used to accomplish the desired nonlinear operations.

**Use of Polarized Relays**

All the circuits described above can be constructed with relays instead of diodes to perform the limiting operations, if relay-driving amplifiers are available. If special relay amplifiers are not available, it is desirable

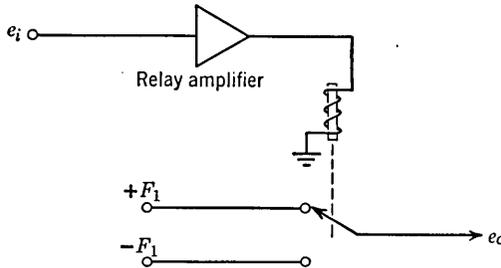


Fig. 31. Dry friction simulator with a polarized relay.

to use operational amplifiers without feedback resistors to obtain the necessary high gain and sensitivity. An open loop amplifier, however, will saturate with even a very small input, and unless the chopper stabilization loop can be disabled, the amplifier's recovery from saturation will be too slow for satisfactory operation. The alternative is to limit the output of the amplifier with diodes as discussed above. This technique offers certain advantages due to the sharp switching characteristics of the relay, but it does not replace diodes. Some types of discontinuities, such as changing from one equation to another during a computation, are best performed with relays.

**Relay Simulation of Coulomb Friction.** This circuit is shown in Fig. 31. Similar circuits can be constructed for the other operations above.

**Generation of Square and Sawtooth Waves**

Switching devices can be used conveniently in the generation of square or sawtooth driving functions to be used in the computer. The use of a relay to produce a low-frequency square wave is shown in Fig. 32. The voltages  $E_1$  and  $-E_1$  are supplied from the computer reference voltage supply. The switch is closed at  $t = 0$ . Prior to closing the switch, the voltage  $e_a$  equals minus the input voltage  $E_1$ , and the polarized relay remains closed in the positive direction as indicated. When the switch is opened, the integrator will begin integrating the voltage  $E_1$

## 23-30 DESIGN AND APPLICATION OF ANALOG COMPUTERS

until the output of the summer becomes zero, when the relay is thrown in the opposite direction. The frequency of the square wave obtained is  $(1/4RC)$ , where  $RC$  is the time constant of the integrator.

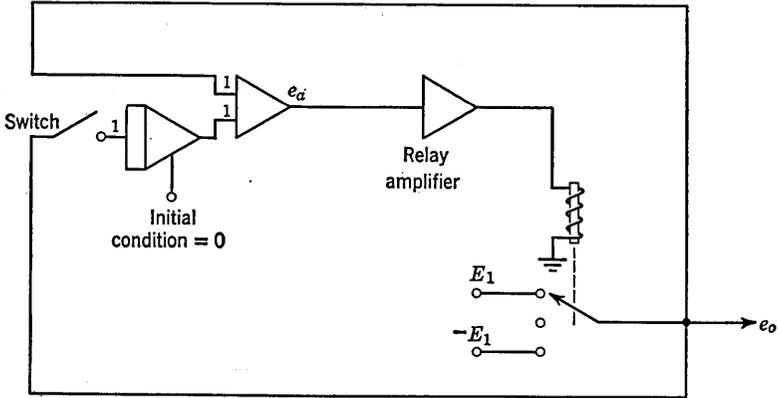


Fig. 32. Low-frequency square wave generator.

**Diode Circuit for Square Wave.** The circuit of Fig. 32 is adequate only for frequencies of the order of a few cycles per second owing to the finite closing time of the relays. Higher frequency square waves can

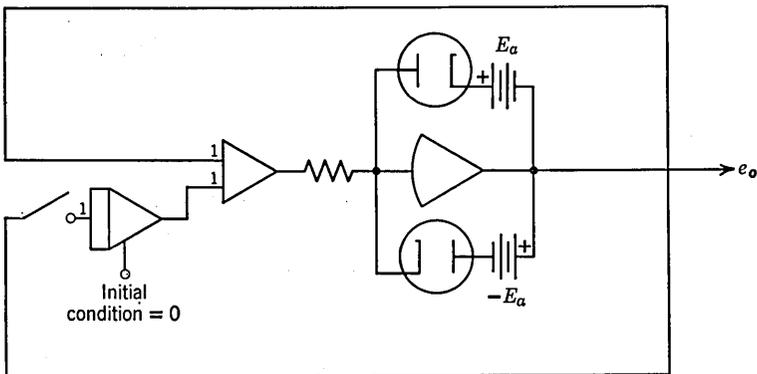


Fig. 33. Diode circuit for square wave generation.

be produced by using diode shunt limiter circuits to replace the relay, as shown in Fig. 33.

**Sawtooth Wave.** A sawtooth wave can be obtained by passing the summer voltage  $e_a$  in either Fig. 32 or 33 through a full-wave rectifier (Fig. 29). The repetition rate of the sawtooth is  $1/(2RC)$ .

#### 4. TRIGONOMETRIC DEVICES

Many physical problems require the use of trigonometric functions to express angular variations or to perform coordinate transformations. Several techniques for obtaining these circular functions are described in this section.

**Trigonometric Functions of Time.** The solution of the differential equation

$$(27) \quad \frac{d^2x}{dt^2} + \omega^2x = 0$$

yields sine and cosine functions of time with adjustable frequency. The long time stability of the sine wave obtained from the solution of this equation on the computer is a function of amplifier phase shift. At very low frequencies the oscillations may tend to decrease in amplitude, whereas at higher frequencies they may tend to diverge. Stable sine and cosine functions can still be obtained by including a damping term in eq. (27) which corrects for the phase shift by adding in a positive or negative damping term.

**Series Expansions.** Very accurate approximations to the trigonometric functions can be obtained by using the first few terms of the series expansions for these functions, if their total angular excursion is not too large. Multipliers and function generators can be used to obtain the necessary powers of the argument (see Sects. 1 and 2).

**Function Generator Representation.** Good accuracies can be obtained by setting up the trigonometric function on a function generator, if the total angular variation is small.

**Servo-Driven Potentiometers.** These represent the most frequently used method of obtaining trigonometric functions.

*Linear Potentiometers.* Combinations of linear potentiometers and resistors can be used for approximating the sine function over certain restricted ranges of its argument. Such potentiometer circuits are described in Ref. 2, page 425.

*Nonlinear Servo-Driven Potentiometers.* Tapered potentiometers are frequently used in conjunction with servo multipliers (see Sect. 1) to perform polar to rectangular transformations. In schematic outline, a combination of four such tapered potentiometers would appear as in Fig. 34, where two brushes separated by  $90^\circ$  are used to give the sine and cosine respectively. The two outputs then correspond to the results of the transformation, since

$$(28) \quad \begin{aligned} x &= R \cos \theta, \\ y &= R \sin \theta. \end{aligned}$$

## 23-32 DESIGN AND APPLICATION OF ANALOG COMPUTERS

These potentiometers can be mounted on the same shaft with linear potentiometers in a servo multiplier, and the device can perform multiplication as well as resolution. Transformation from rectangular to polar coordinates is considered below.

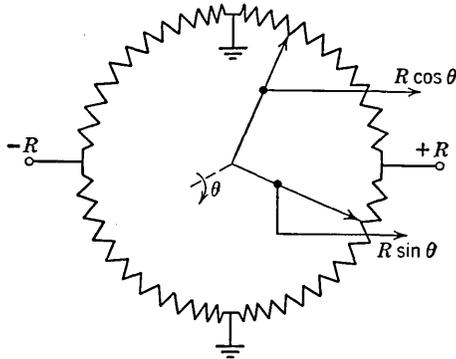


FIG. 34. Tapered sine-cosine potentiometer.

**A-C Resolvers.** Induction resolvers may be used to obtain the polar to rectangular transformations if the input a-c voltages are modulated by the proper d-c values from the computer and the outputs are demodulated to obtain the quantities  $x$  and  $y$ . The modulating and demodulating circuits are subject to drift and are difficult to construct for high accuracy, so that induction resolvers are generally used only with a-c computers on flight tables or other special applications.

**Sines and Cosines from Implicit Computation.** If the derivative of the argument  $\theta$  with respect to time is available in the problem, it is possible to obtain sines and cosines by using amplifiers and multipliers only as shown in Fig. 35. This circuit gives satisfactory answers in many applications, but must be used with caution since it is possible to lose the uniqueness of  $\theta$  in the problem if the computation is long.

**Rectangular to Polar Transformations.** The equations governing this transformation are

$$(29) \quad \begin{aligned} R &= \sqrt{x^2 + y^2}, \\ \theta &= \arctan (y/x). \end{aligned}$$

Although it is possible to solve these equations directly, it is usually more convenient to use servo-driven resolvers to solve for  $R$  and  $\theta$  by implicit computation by using the equations

$$(30) \quad \begin{aligned} -y \sin \theta + x \cos \theta &= 0, \\ x \cos \theta + y \sin \theta &= R, \end{aligned}$$

which are equivalent to the transformation eq. (29). If resolver potentiometers are used to obtain the quantities  $x \sin \theta$ ,  $x \cos \theta$ ,  $y \sin \theta$ , and  $y \cos \theta$ , these can be combined in accordance with the equations above.

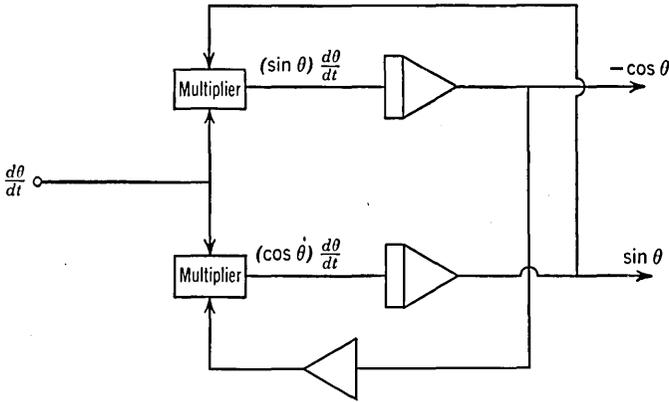


FIG. 35. Sines and cosines from implicit computation.

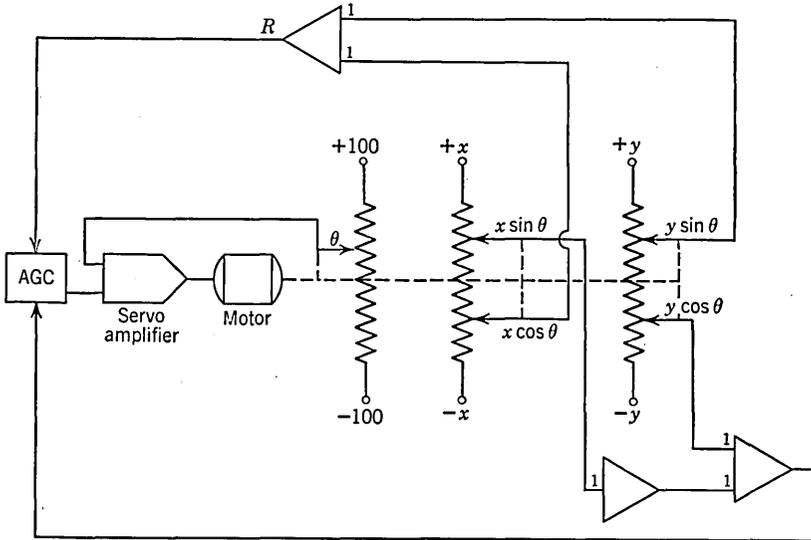


FIG. 36. Servo resolver connections for rectangular to polar transformation.

The output of the second equation provides the vector length  $R$ . The output of the first equation should be zero, and is therefore applied as an input to the servo amplifier which positions the angle  $\theta$  to reduce it to zero. For satisfactory performance such "inverse resolvers" require

## 23-34 DESIGN AND APPLICATION OF ANALOG COMPUTERS

servo amplifiers with automatic gain control (AGC). The schematic of this transformation is shown in Fig. 36.

### 5. TIME DELAY SIMULATORS

Time delays are encountered in the solution of two types of problems on analog computers. The first involves "transportation lags" such as are those found in flow of material in pipes, signal transmission in hydraulic or pneumatic lines, or the motion of neutrons in nuclear reactors. The need for time delay units also arises in the evaluation of correlation functions and convolution integrals. In this section several methods of introducing time delays into a computation are discussed.

While time delay is properly a linear operation, it is included in this chapter because it is often simulated with specially constructed devices.

#### True Time Delays

Simulation of a true time delay of the form

$$(31) \quad f(t - \tau)$$

necessarily involves use of some device to store information during the delay period.

**Two-Pen Recorders.** Probably the oldest technique of introducing a function of the form of eq. (31) into a computer was used with a mechanical differential analyzer, and in modified form is still used in some cases. A two-pen recorder is a device used for plotting two functions of the same independent variable on the same sheet of paper with fixed displacement between their vertical axes. If the displacement between the two pens is made equal to the delay time, and the second pen is replaced by a sight, the curve can be manually tracked, thus generating the function  $f(t - \tau)$ . An automatic curve follower device can also be used, but this may require generation and plotting of  $f(t)$  beforehand.

*Disadvantages.* (a) A very limited range of delay times is available due to the spacing of the recorder heads. (b) Speed of response is severely limited by human tracking and mechanical components.

**Magnetic Tape.** A complex and expensive but very effective way of introducing a time delay into a computation is by means of a magnetic tape recorder with two recording heads. One head is used for recording the information and the second for reading it back to the computer after a suitable delay. The delay is introduced either by an adjustment of movable heads, or by an adjustment of pulleys which regulate the length of tape between the two heads. Tape speed, which directly affects the delay time, is generally fixed by the desired frequency response of the unit.

**Digital Storage Techniques.** In large installations where analog to digital converters enable the computer to be interconnected with a digital computer, it is possible to take advantage of the digital storage for introduction of a time delay. This may take the form of recording the function  $f(t)$  on a magnetic drum channel in digitized form and then reading it back after a suitable delay through a digital to analog converter. It may also be possible to utilize the buffer storage present in the analog-digital-analog converter unit itself. The frequency response available in this type of system is related to the sampling rate of the converter, and needs to be carefully considered.

### Approximations to the Time Delay

All the devices discussed above utilize a form of storage and retrieval of information to achieve a time delay. It is also possible to simulate an approximation to the time delay by noting that the Laplace transform of eq. (31) is

$$(32) \quad \mathcal{L} [f(t - \tau)] = F(s)e^{-s\tau},$$

where  $F(s)$  is the Laplace transform of  $f(t)$ .

The problem of obtaining a time delay is then reduced to that of approximating the function  $e^{-\tau s}$  in such a way as to allow convenient variation of the delay time over a wide range while maintaining a constant amplitude characteristic.

**Power Series Approximation.** It is possible to expand the function  $e^{-\tau s}$  by its Taylor series expansion

$$(33) \quad e^{-\tau s} = 1 - \tau s + \frac{(\tau s)^2}{2!} - \frac{(\tau s)^3}{3!} + \dots$$

Unfortunately, this series requires a series of differentiations for its simulation on the computer. Furthermore, the rate of convergence of the series is slow for large values of the argument  $\tau s$  and therefore it is not suitable for high frequencies or long values of delay. For short delays and limited frequency response, it is possible to obtain satisfactory response from a circuit which simulates the first few terms of the series eq. (33), especially if the problem includes a first order lag in series with the delay, and thus filters out noise from the differentiation process. In many physical problems, such as a pipe transport lag, the transfer functions are of the form

$$(34) \quad KG(s) = \frac{Ke^{-\tau_1 s}}{\tau_2 s + 1},$$

and thus include a lag term with a delay circuit.

## 23-36 DESIGN AND APPLICATION OF ANALOG COMPUTERS

**Multiple Lag Approximation.** The exponential function can be written as the limit

$$(35) \quad e^{-\tau s} = \lim_{n \rightarrow \infty} \left( \frac{1}{1 + \tau s/n} \right)^n.$$

If a finite value of  $n$  is used, the approximation consists of a pole of order  $n$  located on the negative real axis of the  $s$  plane at  $-n/\tau$ . For small values of  $n$  the approximation is not very good (Ref. 18). In one computer study the time delay was represented by 80 cascaded first order lags, i.e.,  $n = 80$  in the above equation (Ref. 19).

**The Padé Approximation.** A much more satisfactory method of approximating  $e^{-\tau s}$  is by means of a rational algebraic function in which both numerator and denominator are polynomials in  $s$ . This rational function is known as the Padé approximation (Ref. 20) with numerator of degree  $n$  and denominator of degree  $m$ :

$$(36) \quad e^{-\tau s} \cong \frac{P_n(s)}{Q_m(s)}.$$

The coefficients of the polynomials are selected in such a way that the Maclaurin expansion of the approximation agrees with the expansion of  $e^{-\tau s}$  (eq. 33) for the largest number of terms possible.

**EXAMPLE.** Approximate  $e^{-\tau s}$  by a ratio of a third to a second degree polynomial. There are six coefficients to be determined:

$$(37) \quad e^{-\tau s} \cong \frac{1 + a_1s + a_2s^2 + a_3s^3}{b_0 + b_1s + b_2s^2}.$$

If the coefficients are chosen so that the first six terms of the Maclaurin series are correct, one obtains

$$(38) \quad e^{-\tau s} \cong \frac{1 - \frac{3}{4}\tau s + \frac{3}{20}(\tau s)^2 - \frac{1}{60}(\tau s)^3}{1 + \frac{2}{5}\tau s + \frac{1}{20}(\tau s)^2}.$$

The Maclaurin series expansion of eq. (38) is

$$(39) \quad 1 - \tau s + \frac{1}{2}(\tau s)^2 - \frac{1}{6}(\tau s)^3 + \frac{1}{24}(\tau s)^4 - \frac{1}{120}(\tau s)^5 + \frac{1}{800}(\tau s)^6 - \dots.$$

Since the expansion of  $e^{-\tau s}$  is

$$(40) \quad e^{-\tau s} = 1 - \tau s + \frac{1}{2}(\tau s)^2 - \frac{1}{6}(\tau s)^3 + \frac{1}{24}(\tau s)^4 - \frac{1}{120}(\tau s)^5 + \frac{1}{720}(\tau s)^6 - \dots.$$

Note that the two series differ only in the seventh term.

The coefficients of the Padé approximation for various degrees of polynomials in the numerator and denominator have been tabulated (Refs. 18, 20). Commercially available time delay units have made use of approximations where  $n = m = 2$ , and also  $n = m = 4$ . Thus,

the approximation for  $n = m = 2$  is

$$(41) \quad e^{-\tau s} \cong \frac{12 - 6(\tau s) + (\tau s)^2}{12 + 6(\tau s) + (\tau s)^2}$$

There are two ways of mechanizing the transfer function of eq. (41). Figure 37 shows a circuit for the second order approximation which can

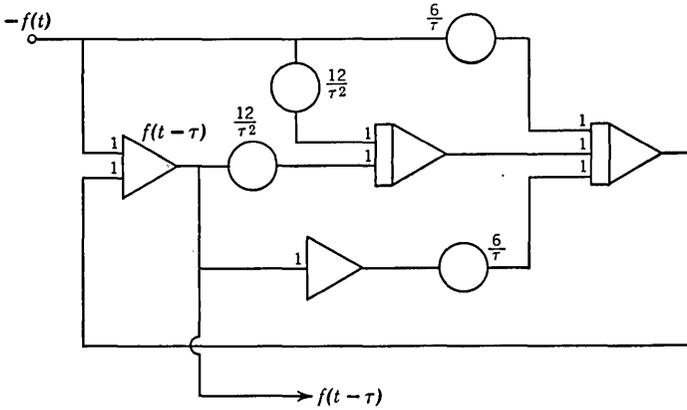


FIG. 37. Circuit for the generation of the second order Padé approximation.

be set up on the computer with amplifiers and potentiometers. Figure 38 is a representation of eq. (41) with amplifiers with complex input and

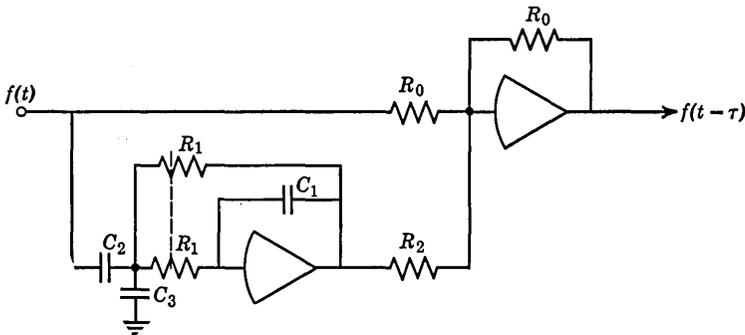


FIG. 38. Two-amplifier circuit for the second order Padé approximation.

feedback networks, which thus reduce to two the total number of amplifiers required.

**Limitations and Features of the Approximation.** Time delay in the circuits for the Padé approximation means a linear variation of phase

## 23-38 DESIGN AND APPLICATION OF ANALOG COMPUTERS

shift with frequency. The phase shift limit of the second order approximation ( $n = m = 2$ ) is approximately  $100^\circ$ , and its frequency limit is too low for all but crude approximations. The phase shift limit for linearity in the fourth order approximation ( $n = m = 4$ ) is approximately  $400^\circ$ , thus giving a four-to-one improvement in frequency response. A practical circuit for the fourth order approximation, which

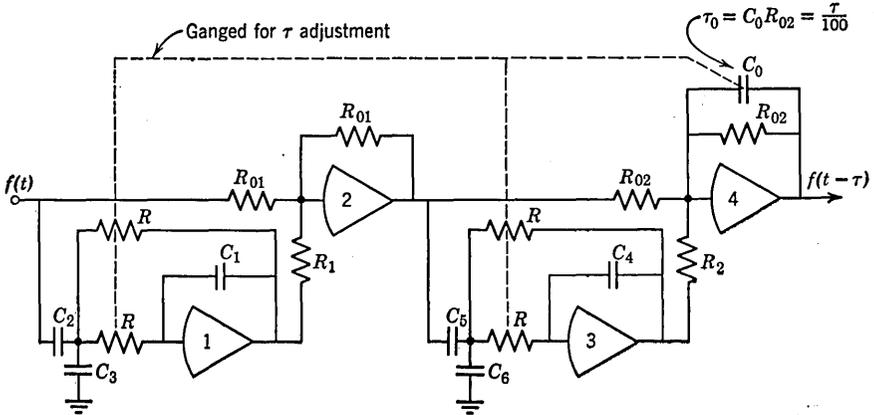


FIG. 39. Practical circuit for fourth order approximation to time delay.

includes a small lag term for filtering the output, is shown in Fig. 39 (Ref. 21). Regarding this circuit, the following can be noted:

(a) Frequency response can be improved by cascading several units of the type shown in Fig. 39. A comparison in approximate frequency response to a step function input is shown in Fig. 40.

(b) If the resistors  $R$  connected with dotted lines in Fig. 39 are ganged potentiometers, it is possible to make the delay itself a function of time by driving these potentiometers with an appropriate servo. Over a limited range, the delay can be varied as function of time without distortion in this circuit.

(c) Output amplitude is independent of delay time.

(d) The amplitude-frequency characteristic is flat over the range of frequencies of interest, up to the breakpoint determined by the time constant of the output filter, i.e.,  $1/(R_0 C_0)$  rad per second. This time constant is sometimes chosen as  $\frac{1}{100}$  of the delay time.

(e) In certain closed loop applications the so-called diagonal approximation (degree of numerator and denominator equal) will result in instability. In this case, an approximation by a higher degree polynomial

in the denominator must be used to restrict the frequency range of the unit.

(f) It should be noted that even the rather sophisticated circuit discussed here is only an approximation, and it is still quite limited in fre-

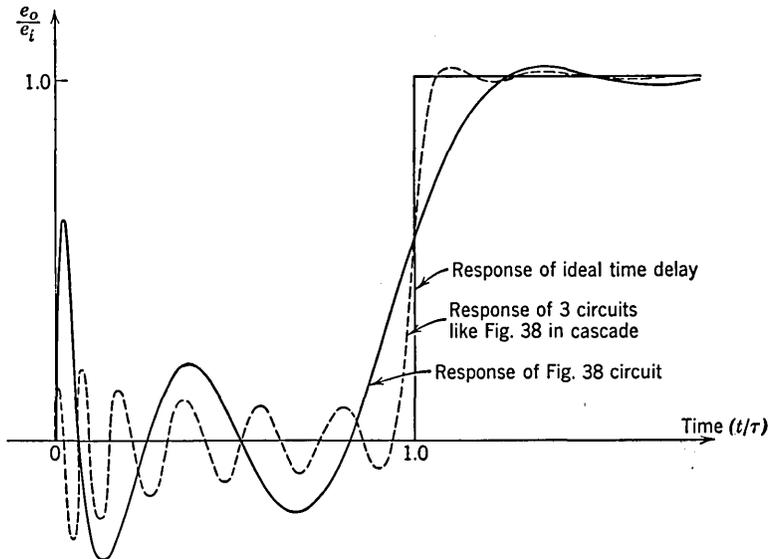


FIG. 40. Step function response of circuit for fourth order Padé approximation (circuit of Fig. 38).

quency response for large values of delay, since its frequency limit is approximately equal to the inverse of the delay time selected. External storage type delay simulators, such as a magnetic tape recorder, can offer advantages in frequency response, but these may be offset by the cost and complexity of the interconnecting equipment with the computer.

## REFERENCES

1. C. L. Johnson, *Analog Computer Techniques*, McGraw-Hill, New York, 1956.
2. G. A. Korn and T. M. Korn, *Electronic Analog Computers*, 2nd edition, McGraw-Hill, New York, 1956.
3. W. W. Soroka, *Analog Methods in Computation and Simulation*, McGraw-Hill, New York, 1954.
4. C. A. Wass, *Introduction to Electronic Analogue Computers*, McGraw-Hill, New York, 1955.
5. E. A. Goldberg, A high-accuracy time-division multiplier, *RCA Rev.*, **23**, 265 (1952).

## 23-40 DESIGN AND APPLICATION OF ANALOG COMPUTERS

6. C. D. Morrill and R. V. Baum, A stabilized electronic multiplier, *Trans. I.R.E., PGECE*, December 1952.
7. S. Giser, An all-electronic high-speed multiplier, *M.I.T. Instrumentation Laboratory*, Rept. R-67, November 1953.
8. C. M. Edwards, Survey of analog multiplication schemes, *J. Assoc. Computing Machinery*, **1**, 27-35 (1954).
9. A. B. MacNee, A crossed-fields electron beam multiplier, *Proc. I.R.E.*, **37**, 1315 (1949).
10. W. A. McCool, An AM-FM electronic analog multiplier, *Proc. I.R.E.*, **41**, 1470-1477 (1953).
11. E. A. Goldberg, Step multiplier in a guided missile computer, *Electronics*, **24**, 121-124 (August 1951).
12. W. R. Thomas and M. Squires, Electronic analog methods of multiplication, *RAE Tech. Note GW-53*, October 1949.
13. H. Hamer, Optimum linear-segment function generation, *Trans. Am. Inst. Elec. Engrs.*, Paper No. 56-696, 1956.
14. P. E. Stanley, The generation of functions for analog computing, *Am. Inst. Elec. Engrs., Conference Paper*, No. CP 57-661, 1957.
15. H. R. Meissinger, An electronic circuit for the generation of functions of several variables, *I.R.E. Convention Record*, Pt. IV, 1955.
16. R. A. Sinker, A photo-electric function generator, *Electronics*, **9**, 178 (October 1956).
17. C. D. Morrill and R. V. Baum, Diode limiters simulate mechanical phenomena, *Electronics*, **25**, 122-126 (November 1952).
18. J. G. Truxal, *Automatic Feedback Control System Synthesis*, Sect. 9.8 (pp. 546-553), McGraw-Hill, New York, 1955.
19. K. L. Chien, J. B. Reswick, and J. A. Hrones, On the automatic control of generalized passive systems, *Trans. Am. Soc. Mech. Engrs.*, **74**, 175-183 (1952).
20. R. D. Teasdale, Time domain approximation by use of Padé approximants, *I.R.E. Convention Record*, Pt. V, pp. 89-94, 1953.
21. C. H. Single and G. S. Stubbs, Transport delay simulation circuits, *Westinghouse Atomic Power Division Tech. Rept. WAPD-T-38*, May 1952.

## Analogs and Duals of Physical Systems

*Richard Mackey*

1. Electric Analogy of Dynamic System	24-01
2. General Terminology	24-03
3. Analysis of General Systems	24-03
4. Energy Considerations	24-07
5. Duality	24-08
6. Construction of Duals	24-09
7. Across and Through Variables in Physical Systems	24-12
References	24-12

### I. ELECTRIC ANALOGY OF DYNAMIC SYSTEM

**Introduction.** True analogies are not just a superficial correspondence of a few parameters; consequently in many cases their significance is not fully appreciated. The validity of an analogy depends solely on the mathematical proof of the correspondence between the two systems. The existence of a rigorous analogy implies an exact similarity in the form of the mathematical equations which describe the behavior of the two systems under consideration. (See Chap. 21, Sect. 6, for table of symbols and a brief discussion of usage.)

**Basic Mechanical Elements.** Consider the simple damped oscillator shown in Fig. 1. The free element behavior of the mechanical com-

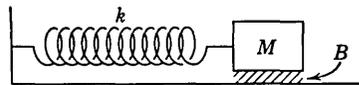
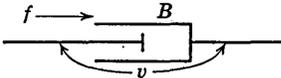
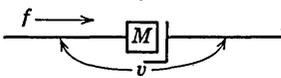
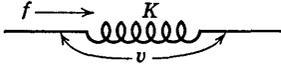


FIG. 1. Basic mechanical circuit.

## 24-02 DESIGN AND APPLICATION OF ANALOG COMPUTERS

ponents is given by:

<i>Friction</i>		$f = Bv, \quad v = \frac{1}{B}f$
<i>Mass</i>		$f = M \frac{dv}{dt}, \quad v = \frac{1}{M} \int f dt$
<i>Spring</i>		$f = k \int v dt, \quad v = \frac{1}{k} \frac{df}{dt}$

Free element response of mechanical components:  $f$  = force,  $v$  = velocity,  $B$  = dashpot constant (represents any form of viscous friction),  $M$  = mass,  $K$  = spring constant.

**Analysis.** This mechanical circuit may be solved by either the path or junction (loop or node) principle. A *loop* is defined as any closed path in a circuit; a *node* or junction is the point of intersection of two or more distinct paths through the network. The method used will in general depend upon the particular structure (circuit) at hand.

Each of the elements of Fig. 1 has the same velocity difference with respect to the reference frame and therefore the system may be said to have one independent velocity node (junction). This circuit is most easily solved by the junction method since there are fewer junctions than paths. Writing the summation of forces at the velocity node equals zero gives:

$$(1) \quad \sum f = 0 = M \frac{dv}{dt} + Bv + k \int v dt.$$

Because mechanical circuits may be drawn in any number of pictorial ways, the actual circuit connections may be hidden and thus cause confusion during analysis. It is of advantage to develop a technique for reducing such pictorial connection diagrams to standard schematic diagrams. This schematic system will be worked out along the lines of electric circuit principles mainly because of the systematic formulation established for the solution of electric networks.

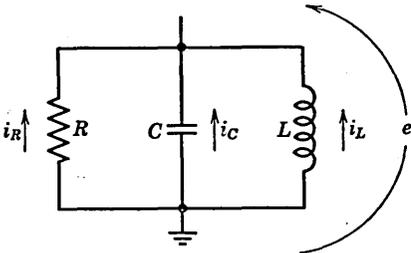
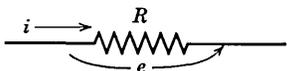
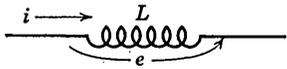
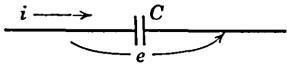


FIG. 2. Basic electric circuit.

### Basic Electrical Elements.

Consider the three-element electric circuit given below which represents the general oscillator. The basic electric circuit is shown in Fig. 2. The

free element behavior of the component parts is given by

Resistance		$e = iR,$	$i = \frac{1}{R} e$
Inductance		$e = L \frac{di}{dt},$	$i = \frac{1}{L} \int e dt$
Capacitance		$e = \frac{1}{C} \int i dt,$	$i = C \frac{de}{dt}$

Free element response of electrical components:  $e =$  voltage,  $i =$  current,  $R =$  resistance,  $L =$  inductance,  $C =$  capacitance.

**Analysis.** Since all these elements have the same voltage difference across them, a simple junction equation is suggested.

$$(2) \quad \sum i = 0 = C \frac{de}{dt} + \frac{1}{R} e + \frac{1}{L} \int e dt.$$

Equations (1) and (2) have the same general structure but different symbols for the element constants and variables. Note that the across and through variables share the *same* role in the two systems (dependent variable or excitation)—this is the basis for the structure analogy. An *across variable* requires two points for its specification and is a measure of the difference of magnitude of a variable between these points such as voltage, velocity, and temperature. The *through variable* requires only one quantity for its specification and is a measure of variables that have the same magnitude at all points along the element, such as current, force, and heat flux.

## 2. GENERAL TERMINOLOGY

It is convenient to reduce all circuits, whether electric, magnetic, thermal, mechanical, or other, to a schematic diagram using the electrical symbols to represent the various components. Thus Table 1 gives the relations of across variables to the through variables for various types of elements and the corresponding symbol representation.  $\alpha$ ,  $\beta$ , and  $\gamma$  are parameters. Relations (A), (B), (C) may be solved explicitly for the through variable resulting in (A'), (B'), (C').

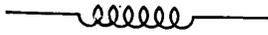
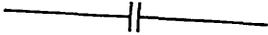
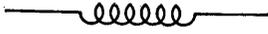
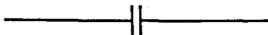
## 3. ANALYSIS OF GENERAL SYSTEMS

**Schematic Diagrams.** The schematic diagram is developed with the following definitions:

*Series.* Elements are said to be in series if the same through variable passes through each element.

## 24-04 DESIGN AND APPLICATION OF ANALOG COMPUTERS

TABLE 1. ELEMENT REPRESENTATION FOR ACROSS-THROUGH RELATIONS

Across-Through Variable Relation	Symbol
(A) Across = $\alpha$ (through)	
(B) Across = $\beta \frac{d}{dt}$ (through)	
(C) Across = $\gamma \int$ (through) $dt$	
(A') Through = $\frac{1}{\alpha}$ (across)	
(B') Through = $\frac{1}{\beta} \int$ (across) $dt$	
(C') Through = $\frac{1}{\gamma} \frac{d}{dt}$ (across)	

*Parallel.* Elements are said to be in parallel if the same across variable exists across each element.

Now Kirchoff's laws may be expressed as:

*Path principle.*  $\sum$  across variable around closed path = 0.

*Junction principle.*  $\sum$  through variable at junction = 0.

The following relationships also exist between the across and through variables:

$$\frac{\text{Across variable}}{\text{Through variable}} = \text{Circuit parameter (structure).}$$

EXAMPLE. Electrical resistance,  $e/i = R$ .

$$(\text{Across variable}) \times (\text{Through variable}) = \text{Power.}$$

EXAMPLE. Electrical power,  $ei = \text{watts}$ .

**Identifying Elements.** The ideas behind the equations representing the characteristic behavior of the schematic symbols may be expanded to provide a test (conceptual or experimental) for the element behavior. Suppose an unknown component from some system is encountered and the schematic circuit diagram is desired. The across variable response to a known through stimulus immediately determines which symbol to use according to the relations (A), (B), (C). Thus if the unknown system component is tested with a step function of through variable and the result is a unit impulse function, by relation (B) that component must be represented by the schematic element . These tests may be presented in a tabular form shown in Table 2.

The table is entered by picking a function of one of the forms in the row under Test Variable. The element response is one of the three

functions in the column headed by the chosen test variable. When the response has been determined, move along that row to the last column to find the schematic symbol to represent the unknown element. If the testing function was a through variable, move left to the first column to find the schematic symbol; if the testing function was an across variable, move right to the last column for the schematic symbol.

TABLE 2. TEST TABLE FOR COMPONENT BEHAVIOR

Through Test Symbol	Test Variable						Across Test Symbol
	$t^2$	$t$					
	$t^3$	$t^2$					
	$t^2$	$t$ <td></td> <td></td> <td></td> <td></td> <td></td>					
	$t$ <td></td> <td></td> <td></td> <td></td> <td></td> <td></td>						

If these principles are applied to the mechanical circuit of Fig. 1, the schematic diagram of Fig. 3 may be arrived at immediately by noting that the across variable is velocity and the through variable is force.

**Junction Technique.** Consider the more complex system shown in Fig. 4. Mark all points with velocity different from reference frame. (These are velocity nodes,  $v_1, v_2, v_3$ ).

It is important to realize in any method of analysis that the mass is

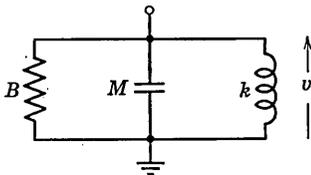


FIG. 3. Symbolic representation of circuit of Fig. 1.

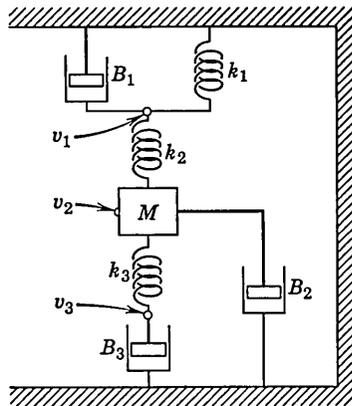


FIG. 4. Complex mechanical circuit.

a two-terminal element with one terminal connected to the inertial reference (usually ground or frame). Thus before either the path or junction technique is applied sketch in the reference terminals for all the masses.

## 24-06 DESIGN AND APPLICATION OF ANALOG COMPUTERS

Mark these nodes above a reference plane and connect the corresponding schematic symbols between the various nodes and reference. The resulting circuit is the desired schematic shown in Fig. 5.

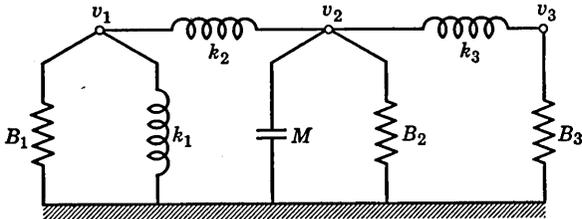


FIG. 5. Schematic of circuit of Fig. 4.

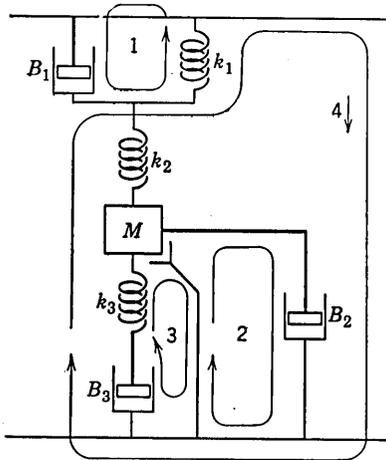


FIG. 6. Paths traced through circuit of Fig. 4.

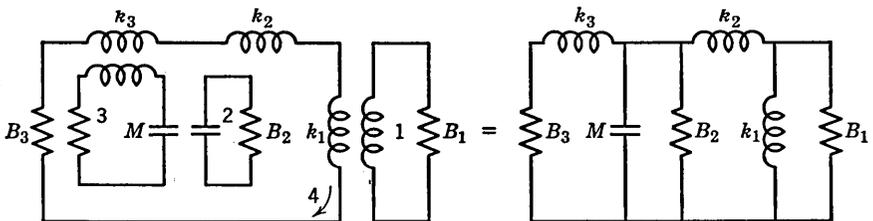


FIG. 7. Individual loops for circuit of Fig. 6.

**Path Technique.** Start anywhere, trace through the pictorial of Fig. 5 in such a way as to end up at the starting point (from a closed loop), and replace each pictorial element by its schematic symbol. Continue

making loops until every element has been traced through, as shown in Fig. 6. The individual loops are shown symbolically in Fig. 7.

*Note.* The resulting schematic is the same regardless of the technique used. The choice of which technique to use will depend on the relative number of junctions and paths.

4. ENERGY CONSIDERATIONS

The power delivered to a system must equal the rate of kinetic energy storage plus the rate of potential energy storage plus the power loss due to dissipation. The rate at which work is done or power delivered to a system (considered as a driving point impedance) is given by eq. (3).

$$(3) \quad P = \frac{d}{dt} (T + V) + D,$$

where  $P$  = power supplied = (across) (through),  
 $T$  = kinetic energy,  
 $V$  = potential energy,  
 $D$  = dissipation.

Continuing the mechanical-electrical comparison.

*Mechanical System.*

$$(4) \quad P = fv, \quad T = \frac{1}{2} Mv^2, \quad V = \frac{1}{2} kx^2, \quad D = Bv^2.$$

$$P = M \frac{dv}{dt} v + k \frac{dx}{dt} x + Bv^2 = fv.$$

Dividing through by  $v$  gives

$$(5) \quad f = M \frac{dv}{dt} + Bv + k \int v dt.$$

*Electrical System.*

$$(6) \quad P = ei, \quad T = \frac{1}{2} Li^2, \quad V = \frac{1}{2} Ce^2 = \frac{1}{2} q^2/C, \quad D = Ri^2.$$

$$P = L \frac{di}{dt} i + \frac{q}{C} \frac{dq}{dt} + Ri^2 = ei.$$

Dividing through by  $i$  gives

$$(7) \quad e = L \frac{di}{dt} + Ri + \frac{1}{C} \int i dt.$$

Expressions (5) and (7) meet all the requirements of analogous equations and hence must represent analogous systems.

The choice of electrical element to store kinetic energy and the one to store potential energy is somewhat arbitrary. An analogy exists between

## 24-08 DESIGN AND APPLICATION OF ANALOG COMPUTERS

two elements or systems if the form of their mathematical equations is the same. Thus the energy stored in a mass by virtue of its motion could equally well be considered analogous to the stored energy of a capacitor or an inductor, that is,

$$(8) \quad \frac{1}{2} C e^2 \approx \frac{1}{2} M v^2 \approx \frac{1}{2} L i^2.$$

However, closer observation of these relations seems to favor the choice of the inductor for kinetic energy storage and the capacitor for potential energy storage. Rewriting the above eq. (8) with the relations  $e = q/C$

and  $q = \int i dt$  gives

$$(9) \quad \frac{1}{2} \frac{1}{C} \left( \int i dt \right)^2 \approx \frac{1}{2} M \left[ \frac{dx}{dt} \right]^2 \approx \frac{1}{2} L \left[ \frac{dq}{dt} \right]^2.$$

Likewise for the stored potential energy of a spring,

$$(10) \quad \frac{1}{2} \frac{1}{C} \left( \int i dt \right)^2 \approx \frac{1}{2} k \left( \int v dt \right)^2 \approx \frac{1}{2} L \left[ \frac{dq}{dt} \right]^2.$$

Thus the similarity of energy terms in eqs. (9) and (10) leads to the mass-inductor, spring-capacitor analogy. Physically this is also reasonable since kinetic energy is energy of motion. Energy can be stored in the field of an inductor only when there are charges in motion. Potential energy is stored in the spring by displacement of an elastic medium and likewise for the dielectric of a capacitor. This similarity is thus termed the *conventional energy analogy* but is not unique.

### 5. DUALITY

Since eqs. (1) and (5) are identical and express  $\Sigma f = 0$  for the mechanical system, eqs. (2) and (7) must bear some relation to each other.

They must have the same solution since each represents the same physical phenomena and each has form  $\Sigma f = 0$ , where in (2) the symbol  $i$  corresponds to force and in (7)  $e$  corresponds to force. Equation (7) represents an application of Kirchoff's voltage law and must therefore apply to an end-to-end or loop connection of the elements, as shown in Fig. 8. Thus there are two electric circuits which behave like the given mechanical circuit. This relationship is illustrated in Fig. 9. The two electric circuits are termed "duals."

Thus the conventional energy approach suggests another set of corresponding quantities. A comparison of the coefficients of eqs. (5) and

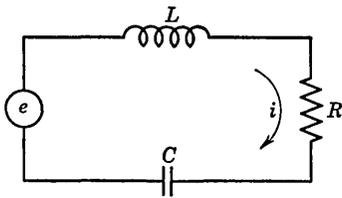


FIG. 8. Second analog of mechanical circuit of Fig. 1.

(7) shows that mass, mechanical resistance, and compliance in the mechanical system are analogous to inductance, electrical resistance, and

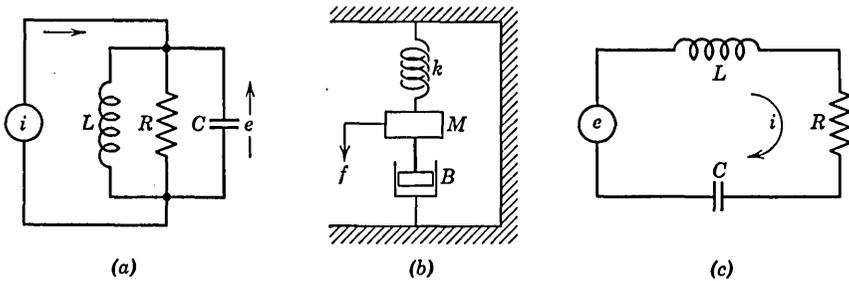


FIG. 9. (a) Structural analog, preservation of through and across variables. (b) Given mechanical circuit. (c) Energy analog, preservation of “conventional” kinetic and potential energy terms.

capacitance respectively. Also it may be seen that force corresponds to voltage and velocity to current (Table 3).

TABLE 3. CORRESPONDENCES: ANALOGOUS QUANTITIES IN MECHANICAL AND ELECTRICAL SYSTEMS

Electrical (Structure Basis)	Mechanical (Rectilinear System)	Electrical (Energy Basis)
Current, $i$	Force, $f$	Voltage, $e$
Voltage, $e$	Velocity, $v$	Current, $i$
Flux, $f$	Displacement, $x$	Charge, $q$
Capacitance, $C$	Mass, $M$	Inductance, $L$
Conductance, $1/R$	Resistance, $B$	Resistance, $R$
Inductance, $L$	Compliance, $1/k$	Capacitance, $C$
Through	Through	Across
Across	Across	Through

Just as two electric circuits were found to represent a given mechanical circuit, two mechanical circuits can conceptually be found to represent a given electric circuit. These mechanical circuits will also be duals of one another. This correspondence is represented in Fig. 10.

### 6. CONSTRUCTION OF DUALS

Given any planar schematic network, there are two methods of finding its dual (duality fails if the original network is nonplanar). One method may be considered the junction method and the other the path method.

*Junction Method.* Consider the schematic network shown in Fig. 11. The procedure is:

24-10 DESIGN AND APPLICATION OF ANALOG COMPUTERS

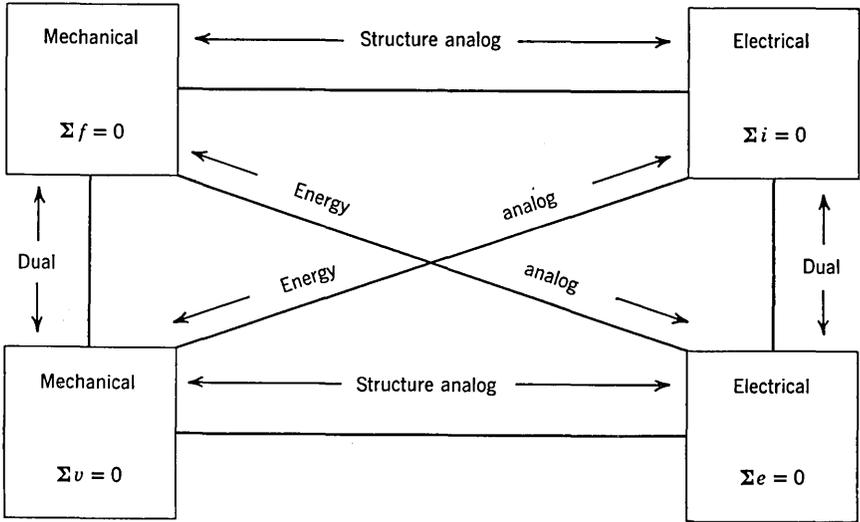


FIG. 10. Relation of duals and analogs.

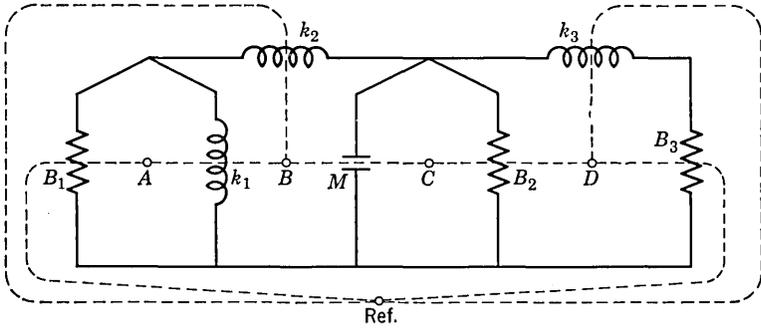


FIG. 11. Original circuit; center of loops marked.

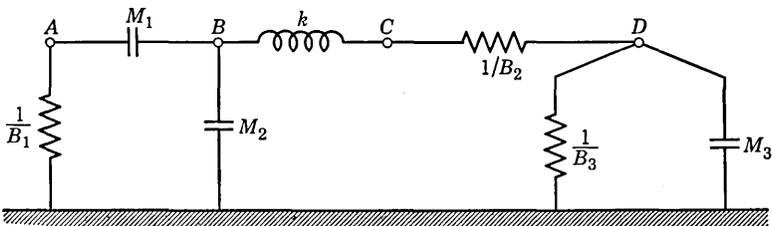


FIG. 12. Dual of circuit of Fig. 11.

1. Mark the center of each loop or "windowpane" with a node and place a reference node outside the circuit.
2. Place these nodes above a reference plane.
3. Connect these nodes in the original circuit by crossing elements.
4. Place the dual of each element between corresponding nodes in the dual circuit.

The result of this procedure is shown in Fig. 12.

*Path Method.* The path method will be applied to the dual circuit just derived so as to return to the original circuit (often a convenient check on the correctness of the derived dual). The procedure is:

1. Mark all nodes in the network and encircle each node, as shown in Fig. 13.

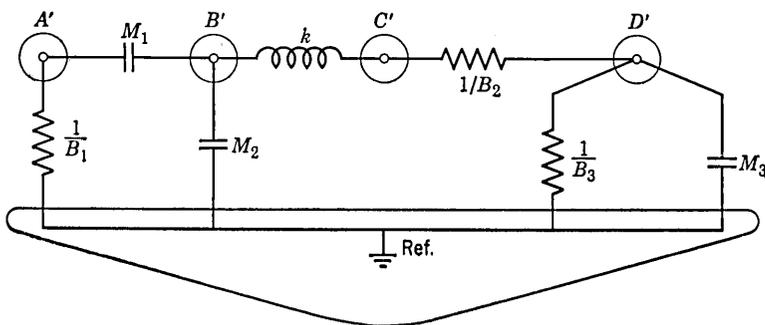


FIG. 13. Circuit of Fig. 12, nodes encircled.

2. Connect all elements crossing the circle at each node except the reference node in an end-to-end series manner replacing each element by its dual, as shown in Fig. 14.

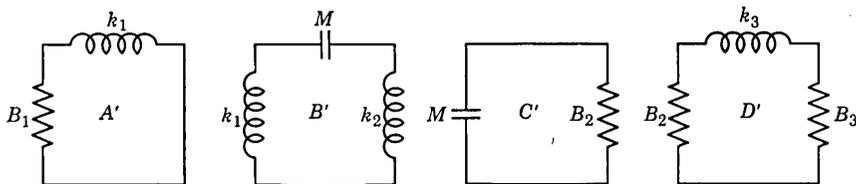


FIG. 14. Elements of Fig. 13 replaced by duals.

3. Now make this connection for the reference node (Ref.) and join the other loops to it, as shown in Fig. 15.

Note that one loop is redundant; this comes about because the circuit may be solved by  $N - 1$  node equations. A separate node equation for the reference node is therefore not written.

## 24-12 DESIGN AND APPLICATION OF ANALOG COMPUTERS

If the derived circuit is redrawn, the original circuit, shown in Fig. 16, results.

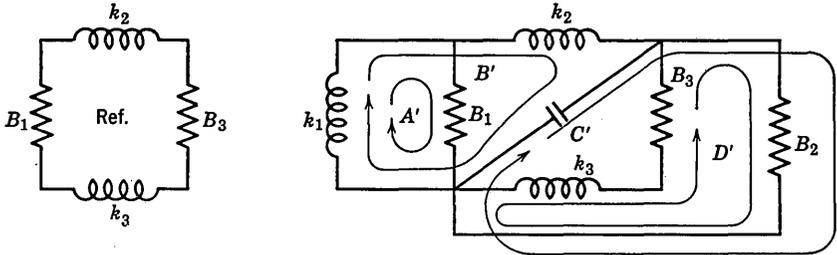


FIG. 15. Loops joined in Fig. 14.

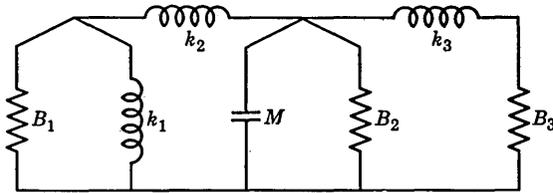


FIG. 16. Redrawn Fig. 15, identical with Fig. 11.

## 7. ACROSS AND THROUGH VARIABLES IN PHYSICAL SYSTEMS

The pertinent variables of several physical systems are summarized in Table 4.

TABLE 4. ACROSS AND THROUGH VARIABLES IN PHYSICAL SYSTEM

System	Across	Through
Electrical	Voltage	Current
Mechanical (rectilinear)	Velocity	Force
Mechanical (rotation)	Angular velocity	Torque
Acoustical	Pressure	Volume current
Thermal	Temperature	Heat flow
Fluid	Pressure	Flow
Magnetic	Magnetomotive force	Flux
Chemical	Chemical potential	Speed of reaction (molecules)

## REFERENCES

1. H. F. Olson, *Dynamical Analogies*, D. Van Nostrand, Princeton, N. J., 1943.
2. W. C. Johnson, *Mathematical and Physical Principles of Engineering Analysis*, McGraw-Hill, New York, 1944.

3. L. A. Pipes, *Applied Mathematics for Engineers and Physicists*, McGraw-Hill, New York, 1946.
4. M. F. Gardner and J. L. Barnes, *Transients in Linear Systems*, Wiley, New York, 1942.
5. W. Thompson, *Mechanical Vibrations*, Prentice-Hall, Englewood Cliffs, N. J., 1948.
6. B. Gehlshoj, *Electromechanical and Electroacoustical Analogies*, Denmark, 1947.
7. M.I.T. Electrical Engineering Staff, *Electric Circuits*, Wiley, New York, 1943.



## Solution of Field Problems

*Walter J. Karplus*

1. Formulation of Engineering Problems as Partial Differential Equations	25-01
2. Continuous Type Electric Analogs	25-05
3. Discrete Element Type Electric Analogs	25-11
4. Nonelectric Field Analogs	25-22
References	25-23

### 1. FORMULATION OF ENGINEERING PROBLEMS AS PARTIAL DIFFERENTIAL EQUATIONS

**Introduction.** Field problems arise in physical systems in which the system properties or parameters are distributed in a continuous fashion throughout the system. The independent variables of the problem therefore include one or more of the space coordinates, as well as the time variable in transient problems. The complete solution of such a problem then provides values for the dependent variables *at all points* within a region of interest, i.e., the field.

Analogs represent a powerful tool for dealing with such problems because the field problems characteristic of the many diverse areas of physics and engineering may be described compactly by a relatively small number of partial differential equations. General analog techniques developed to solve these equations may then be applied as desired to the specific problem of interest. A more detailed discussion of this subject is presented by Karplus (Ref. 1), and a portion of this chapter is

## 25-02 DESIGN AND APPLICATION OF ANALOG COMPUTERS

abstracted from this reference with permission of the publishers. (See Chap. 21, Sect. 6, for a brief table of symbols.)

**Dependent Variables.** Just as in lumped systems (see Chap. 24), it is convenient in treating field problems to identify *across* and *through* variables. The *across* variable, generally termed the *potential function*  $\phi$ , expresses the value of a dependent variable *with respect to some reference point*. It is therefore really the difference in potential between (or across) two points. The *through* variable, usually called the *stream function*  $j$ , is proportional to the gradient of the potential function (first partial derivative of  $\phi$  with respect to a space coordinate) and describes the flow through a region of the field. In Cartesian coordinates

$$(1) \quad j_x = -\frac{\partial\phi}{\partial x}, \quad j_y = -\frac{\partial\phi}{\partial y}, \quad j_z = -\frac{\partial\phi}{\partial z},$$

where  $j_x$ ,  $j_y$ , and  $j_z$  represent the flow rate in the  $x$ ,  $y$ , and  $z$  directions respectively.

In presenting the solution of a field problem all points having the same potential  $\phi$  are connected to form *equipotential lines*; all points for which the stream function  $j$  is the same are joined to form *streamlines*. The complete solution of a field problem is then usually a grid of orthogonal equipotential and streamlines.

**Coordinate Systems.** It is advisable to formulate a field problem in the coordinate system which will make the resulting mathematical expressions take as simple a form as possible. The selection of the coordinate system is therefore governed to a large extent by the shape and symmetry of the field boundaries. Where these boundaries are rectangular, *Cartesian* coordinates are indicated; where there exists symmetry about a straight line, *cylindrical* coordinates become appropriate; and where there is symmetry about a point, *spherical* coordinates are preferable. These basic coordinate systems are defined in Fig. 1. Other coordinate systems may be developed as needed. In two-dimensional problems the technique of *conformal mapping* (Vol. 1, Chap. 10) is very useful. Parabolic and hyperbolic coordinates, and phase plane plots are also of occasional advantage.

**The Laplacian.** In order to make the partial differential equations of physics as general as possible it is desirable to make them independent of specific coordinate systems. To this end it is convenient to define the *Laplacian operator*  $\nabla^2$  as a shorthand notation. Expressions for  $\nabla^2$  in the principal coordinate system are included in Fig. 1.

In problems of elasticity the *biharmonic operator*  $\nabla^4\phi$  represents

$$\frac{\partial^4\phi}{\partial x^4} + 2\frac{\partial^4\phi}{\partial x^2\partial y^2} + \frac{\partial^4\phi}{\partial y^4}$$

in Cartesian coordinates.

Coordinate System	To Convert to			$\nabla^2\phi$
	Cartesian	Cylindrical	Spherical	
Cartesian $x, y, z$	$x = x$ $y = y$ $z = z$	$r = \sqrt{x^2 + y^2}$ $\theta = \cos^{-1}(x/\sqrt{x^2 + y^2})$ $z = z$	$r_s = \sqrt{x^2 + y^2 + z^2}$ $\alpha = \cos^{-1}(x/\sqrt{x^2 + y^2})$ $\beta = \cos^{-1}(z/\sqrt{x^2 + y^2 + z^2})$	$\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2}$
Cylindrical $r, \theta, z$	$x = r \cos \theta$ $y = r \sin \theta$ $z = z$	$r = r$ $\theta = \theta$ $z = z$	$r_s = \sqrt{r^2 + z^2}$ $\alpha = \theta$ $\beta = \cos^{-1}(z/\sqrt{r^2 + z^2})$	$\frac{1}{r} \frac{\partial}{\partial r} \left( r \frac{\partial \phi}{\partial r} \right) + \frac{1}{r^2} \frac{\partial^2 \phi}{\partial \theta^2} + \frac{\partial^2 \phi}{\partial z^2}$
Spherical $r_s, \alpha, \beta$	$x = r_s \sin \beta \cos \alpha$ $y = r_s \sin \beta \sin \alpha$ $z = r_s \cos \beta$	$r = r_s \sin \beta$ $\theta = \alpha$ $z = r_s \cos \beta$	$r_s = r_s$ $\alpha = \alpha$ $\beta = \beta$	$\frac{1}{r^2} \frac{\partial}{\partial r} \left( r^2 \frac{\partial \phi}{\partial r} \right) + \frac{1}{r^2 \sin^2 \beta} \frac{\partial^2 \phi}{\partial \alpha^2} + \frac{1}{r^2 \sin \beta} \frac{\partial}{\partial \beta} \left( \sin \beta \frac{\partial \phi}{\partial \beta} \right)$

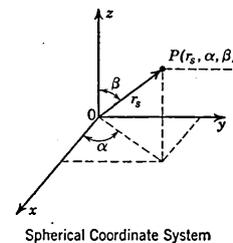
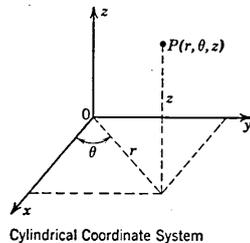
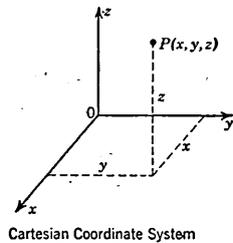


Fig. 1. Basic coordinate systems.

## 25-04 DESIGN AND APPLICATION OF ANALOG COMPUTERS

### Important Equations

The fundamental partial differential equations of physics and engineering and examples of the more important areas of their application are summarized below.

#### Laplace's Equation. $\nabla^2\phi = 0$ .

- Gravitation:* gravitational potential in free space
- Magnetics:* magnetic potential in free space
- Electrostatics:* electrostatic potential in an ideal dielectric
- Electrodynamics:* voltage distribution in a nonreactive field
- Heat transfer:* steady-state temperature distribution
- Fluid mechanics:* velocity potential of an incompressible fluid in a porous medium, velocity potential of homogeneous liquid moving irrotationally
- Statics:* steady-state deflection of mass spring systems

#### Diffusion Equation. $\nabla^2\phi = K(\partial\phi/\partial t)$ .

- Electrodynamics:* voltage distribution in a field having distributed resistance and distributed inductance or capacitance
- Heat transfer:* transient temperature distribution
- Particle diffusion:* concentration of particles of one fluid, diffusion into space occupied by another fluid
- Fluid mechanics:* velocity potential of compressible fluid in a porous medium
- Electromagnetics:* skin effect equation
- Optics:* diffusion of light by a scattering material
- Soil compaction:* hydrostatic pressure in consolidating soils

#### Wave Equation. $\nabla^2\phi = K(\partial^2\phi/\partial t^2)$ .

- Electrodynamics:* voltage distribution in fields containing distributed inductance and capacitance (ideal transmission lines)
- Dynamics:* vibration of system with negligible mass
- Fluid mechanics:* propagation of vibration in compressible media
- Electromagnetics:* propagation of electromagnetic energy in field with negligible conductivity

#### Poisson's Equation. $\nabla^2\phi = f(x,y,z)$ and the related equation $\nabla^2\phi = f(\phi)$ .

- Heat transfer:* temperature distributions in solids in which nuclear, electrical, or chemical energy is being converted to heat
- Electron optics:* motion of electron in space charge regions
- Shear analysis:* shear distribution in torsional problem

#### Biharmonic Equation. $\nabla^4\phi = 0$ , $\nabla^4\phi = K$ , $\nabla^4\phi = K(\partial^2\phi/\partial t^2)$ .

- Elasticity:* stress distribution in solid structure

## 2. CONTINUOUS TYPE ELECTRIC ANALOGS

**Basis for the Analogy.** Continuous type analogs are based upon the recognition that the current and voltage distribution in an electrical conductor having negligible reactance is governed by Laplace's equation. They may therefore be employed to simulate any field governed by Laplace's equation. Occasionally their application may be extended to fields containing distributed internal sources and sinks and governed by Poisson's equation as well. If the conductive material simulating the field is a solid, only the potential field at the external surfaces of the analog system can be studied. These are therefore generally limited to field problems which can be formulated in two space dimensions. Liquid analog systems on the other hand are suitable for three-dimensional problems as well as two-dimensional ones.

### Conducting Sheet Analogs

As the name implies, conducting sheet analogs consist of a thin layer of electrically conductive material. Kirchoff introduced this technique in 1845 by describing the simulation of an electric field by means of a thin copper disk. The application of conducting sheet analogs was hampered for many years by the unavailability of sufficiently uniform and isotropic sheets with resistivities of the right order of magnitude. Great strides have been made in this direction since World War II, however. The solution of problems governed by Laplace's equation proceeds in three steps:

1. A conductive sheet having the same geometrical shape (or a conformal transformation thereof) as the field under study is devised.

2. The boundary conditions of the original field are simulated in the analog systems by the application of voltage and/or current sources at the boundaries of the sheet. In the case of equipotential boundaries, a highly conductive material is placed in contact with the sheet along the entire boundary line, and a constant voltage source is connected to this conductor.

3. By means of suitable sensing equipment the voltage distribution on the surface of the conductive sheet is measured and recorded. The voltages measured in this manner are then directly proportional to the potentials existing at corresponding points of the original field.

4. If desired, the streamlines of the original field may be plotted by changing all streamline boundaries to equipotential boundaries and vice versa, and repeating the above procedure.

A typical conducting sheet analog system is illustrated in Fig. 2.

**Resistance Paper.** The impetus for the recent popularization and development of the conducting sheet technique was provided by the

## 25-06 DESIGN AND APPLICATION OF ANALOG COMPUTERS

introduction of so-called *resistance paper*. First described by Hotchkiss (Ref. 2) in 1948, resistance paper, manufactured under the trade name Teledeltos, was designed for use as electrosensitive recording paper in telegraphic instruments. It is formed by adding carbon black to the paper pulp in the pulp beating stage of the manufacturing process. The

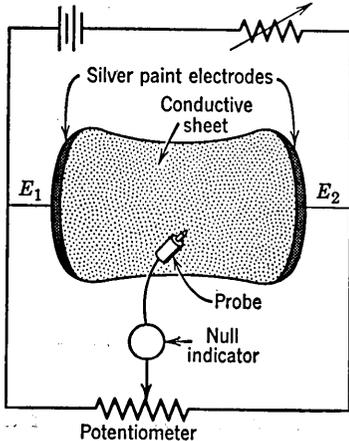


FIG. 2. Conductive sheet analog system.

electrical behavior of such paper is usually specified as "ohms per square," referring to the resistance between opposite sides of a square of paper (the size of the square is irrelevant).

Boundaries at which the voltage is constant are painted directly on the paper with *silver paint*. Occasionally a copper wire is pasted on top of the silver strip to assure equipotential conditions. Boundaries of varying potential are readily obtained by connecting discretely spaced points along the boundary to voltage sources of suitable magnitudes. Alternating or direct current may be used to excite the system. The paper may be

cut as desired to form streamline boundaries.

The *field plotting* apparatus may be very simple. A blunt metal probe, possibly a roller, is guided along the paper in such a fashion that the galvanometer reads zero at all times for a given potentiometer setting. The line drawn in this manner is an equipotential line.

*Accuracy.* This type of apparatus has a limit of accuracy of about 2 per cent and is probably the simplest and cheapest of all the analog methods.

**Other Types of Conducting Sheet Analogs.** From time to time other types of conducting sheet analogs have been used to advantage. The more important of these include: metal sheets, metalized paper, conductive rubbers (rubbers with carbon black dispersions), cloth impregnated with graphite, and untreated paper. The suitability and relative merits of these materials for use as conducting sheet analogs are summarized in Table 1.

### Conducting Liquid Analogs

Water containing small amounts of dissolved salts is an electrical conductor, and may be used as a continuous type conducting analog. Until the development of Teledeltos paper, conductive liquid analogs, generally termed electrolytic tanks, were by far the most widely used

TABLE 1. COMPARISON OF CONDUCTIVE SHEET ANALOGS

Type	Source	Approx. Resistivity, $\Omega/\text{sq.}$	Approx. Uniformity, % Fluctuations	Approx. Isotropy $\frac{\text{R. longitudinal}}{\text{R. transverse}}$	Advantages	Disadvantages
Teledeltos paper	Western Union	Type L: 2000 Type H: 20,000	10%	0.9	Cheap, easy to cut, convenient to use	Easily damaged by probes; absorbs moisture (hence sensitive to temperature changes); easily damaged by bending
Untreated paper	Ordinary drafting paper (non-fibrous)	$5 \times 10^9$	2%	Good	Cheap, convenient, electrodes may be drawn with pencil	Very sensitive to humidity; requires very high impedance sensing circuit
Conductive rubbers (Uskon)	U. S. Rubber Co.	400-2000	5%	0.9	May be "layered" to effect local change in resistivity, layers adhere to each other	Sensitive to mechanical stress
Conductive fabrics	Pacific Mills	1000	Excellent	0.95	May be layered	Sensitive to mechanical stress
Conductive plastics (Markites)		100-1000	Poor	Poor	Very hard, not easily damaged, not affected by moisture	Sensitive to mechanical stress; hard to cut and shape
Metal sheets		Very low	Poor	Good	Readily available	Electrodes must be highly conductive
Homemade sprays		1K-300K	5%	Good	Hard surface not easily damaged	Requires special apparatus for preparation of sheets; difficult to work with
Graphite and wax models		Low	Poor	Poor	Suitable for studies of 3-dimensional fields	

## 25-08 DESIGN AND APPLICATION OF ANALOG COMPUTERS

electric analog systems for the solution of Laplace's equation. The solution of field problems by this method proceeds as follows:

1. A large container open at the top is filled with a suitable electrolyte.
2. A scale model of the boundary configuration of the field under study, or a conformal transformation thereof, is immersed in the tank. Boundaries which are equipotential surfaces are made of metal, while an insulating material is employed for streamline boundaries.

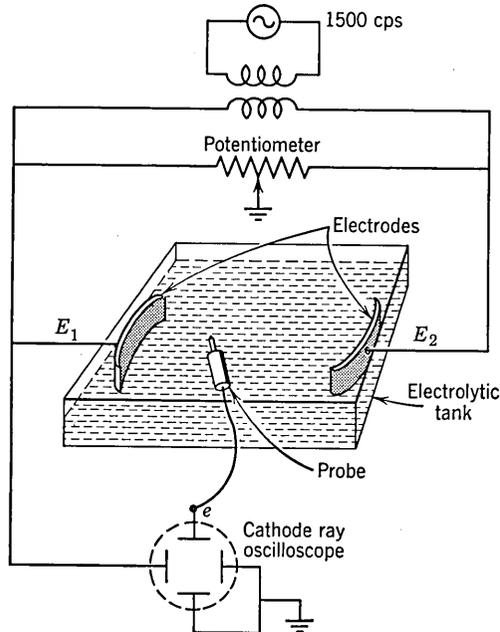


FIG. 3. Conductive liquid analog system.

3. A-c voltage sources of appropriate magnitudes are then applied to all equipotential boundaries.

4. The voltage distribution along the surface of the liquid is measured and recorded by means of suitable sensing equipment. In three-dimensional field problems the sensing probe is inserted into the liquid, while it is kept at the liquid-air interface in two-dimensional problems.

A typical electrolytic tank system is illustrated in Fig. 3.

**Electrolyte.** The conductive liquid must possess the following properties: (1) no electrical reactance; (2) uniform, linear resistivity; (3) resistivity large compared with resistivity of electrodes, small compared with input resistance of sensing device; (4) small surface tension to obviate errors due to meniscus; (5) inert chemically, free of films.

Often ordinary tap water is adequate. Small amounts of copper

sulfate, sulfuric acid, and ethyl alcohol are frequently used additives.

**Excitation.** At low a-c frequencies most electrolytes exhibit non-linear resistive and reactive properties at the electrode-electrolyte interface. These phenomena tend to become negligible at operating frequencies above approximately 1500 cps. Sinusoidal voltage generators are most frequently employed, although square waves have been used to advantage occasionally.

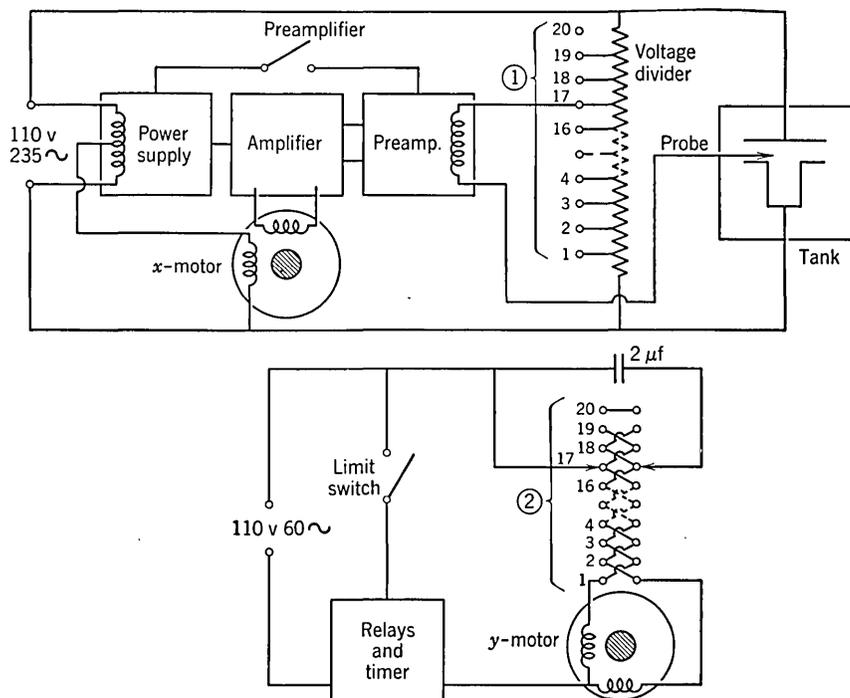


FIG. 4. Automatic field plotter (Ref. 3).

**Boundaries.** Boundaries at which the voltage is constant may be graphited silver or aluminum painted surfaces, or tin or aluminum foil may be used to line the container at the required locations.

**Field Plotters.** Single *probes*, connected to a potentiometer, are employed to sense the field potential. These are generally thin, platinum wires or needles. To detect streamlines, two and four probe arrays are employed.

To achieve high accuracies, precision *field plotting* apparatus is required. *Automatic field plotters* for sensing and recording equipotential lines have been described by Green (Ref. 3) and others. A typical automatic plotting device is shown in Fig. 4.

## 25-10 DESIGN AND APPLICATION OF ANALOG COMPUTERS

In this type of plotter the probe is made to move with a constant velocity in the  $x$  direction. Any difference between the voltage at the probe and an arbitrarily set reference voltage actuates a servo system, which alters the position of the probe along the  $y$  coordinate in such a direction that this discrepancy is eliminated. The probe is therefore kept on the desired equipotential line. When the probe reaches a boundary, a relay causes the probe to reverse its direction of travel and changes the reference voltage at the same time, so that the next desired equipotential line is plotted. In this way any number of preset equipotentials may be traced. A pantograph is employed to obtain a graphic record of the position of the probe within the tank.

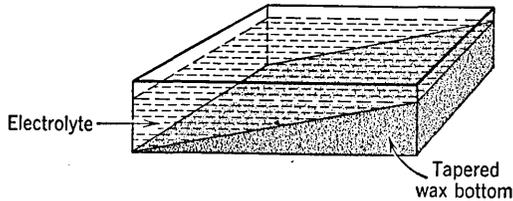


Fig. 5. Electrolytic tank for system with radial symmetry.

**Cylindrical Coordinates.** Electrolytic tanks are well suited for the study of problems in which there exists symmetry about a straight line. A typical wedge of the cylindrical system may readily be simulated by tilting a shallow electrolytic tank as shown in Fig. 5. Since the depth of

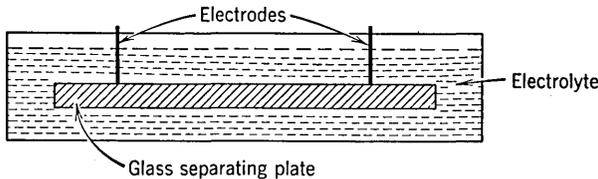


Fig. 6. Electrolytic tank for the simulation of infinite fields.

the electrolyte is everywhere proportional to the distance  $r$  from the cylinder axis, Laplace's equation  $\nabla^2\phi = 0$ , is satisfied.

**Infinite Fields.** Fields which extend to infinity in all directions may be simulated by a double layer electrolytic tank. In this method, according to Boothroyd (Ref. 4), a thin circular sheet of insulating material is mounted as shown in Fig. 6, and separates the electrolyte into two circular conductive sheets, joined at their periphery. The top sheet behaves as though it were part of an infinite plane.

**Accuracy.** Accuracies of the order of 5 per cent are readily achieved with inexpensive and simple instrumentation. Accuracies as high as 0.1 per cent have been reported by Einstein (Ref. 11), Sanders (Ref. 12), and others by means of extremely careful and precise experimental techniques.

### 3. DISCRETE ELEMENT TYPE ELECTRIC ANALOGS

**Basis for the Analogy.** Discrete element type analogs are based upon the recognition of the formal similarity between Kirchhoff's law equations for networks of electric elements and the equations resulting from the finite difference expansion of the partial differential equations describing field problems. The analog method of solution proceeds as follows:

1. The partial differential equations governing the field under study are expanded in finite differences. The grid spacing (mesh size) is chosen so that the expected truncation errors resulting from this process are sufficiently small in all regions of the field.

2. An electric network is constructed in which (a) each node in the finite difference grid is represented by a junction of two or more electric elements and (b) the node equation (according to Kirchhoff's current law) for each node of the network is similar in form, term by term, to the finite difference equation at the corresponding location in the original field.

3. By means of suitable voltage and current sources the electric network is excited at its boundaries as well as possibly at internal node points as specified by the finite difference equation.

4. The voltages appearing at the nodes of the network are measured and recorded. These are then proportional to the potential at the corresponding points in the original field. If desired, the family of equipotential lines may be sketched by interpolating between points of known potential.

#### **Finite Difference Expansions of Partial Differential Equations.**

The mathematical techniques whereby partial differential equations may be transformed into systems of algebraic equations is discussed in Vol. 1, Chaps. 4 and 14. Numerous expansions for first and second derivatives are available. For analog simulation purposes however, the so-called second central difference type of expansion is most useful in approximating the second derivative. Frequently only the space derivatives are expanded in finite differences whereas the time derivatives are left in continuous form.

Table 2 is a summary of the most useful finite difference expansions for the terms of the basic partial differential equations in Cartesian

TABLE 2. FINITE DIFFERENCE EXPANSIONS IN CARTESIAN COORDINATES

	Location of Node Points	
	Node No.	Coordinate
$\frac{\partial^2 \phi}{\partial x^2} = \frac{1}{\Delta x^2} [\phi_1 + \phi_2 - 2\phi_0]$	0	$t_0, x_0, y_0, z_0$
	1	$t_0, x_0 + \Delta x, y_0, z_0$
$\frac{\partial^2 \phi}{\partial y^2} = \frac{1}{\Delta y^2} [\phi_3 + \phi_4 - 2\phi_0]$	2	$t_0, x_0 - \Delta x, y_0, z_0$
	3	$t_0, x_0, y_0 + \Delta y, z_0$
$\frac{\partial^2 \phi}{\partial z^2} = \frac{1}{\Delta z^2} [\phi_5 + \phi_6 - 2\phi_0]$	4	$t_0, x_0, y_0 - \Delta y, z_0$
	5	$t_0, x_0, y_0, z_0 + \Delta z$
$\frac{\partial \phi}{\partial x} = \frac{\phi_1 - \phi_0}{\Delta x}$ , forward difference	6	$t_0, x_0, y_0, z_0 - \Delta z$
$\frac{\phi_0 - \phi_2}{\Delta x}$ , backward difference	7	$t_0 + \Delta t, x_0, y_0, z_0$
$\frac{\phi_1 - \phi_2}{2\Delta x}$ , average difference	8	$t_0 - \Delta t, x_0, y_0, z_0$
	9	$t_0, x_0 + 2\Delta x, y_0, z_0$
	10	$t_0, x_0 - 2\Delta x, y_0, z_0$
$\frac{\partial \phi}{\partial y} = \frac{\phi_3 - \phi_0}{\Delta y}, \frac{\phi_0 - \phi_4}{\Delta y}, \frac{\phi_3 - \phi_4}{2\Delta y}$	11	$t_0, x_0, y_0 + 2\Delta y, z_0$
	12	$t_0, x_0, y_0 - 2\Delta y, z_0$
$\frac{\partial \phi}{\partial z} = \frac{\phi_5 - \phi_0}{\Delta z}, \frac{\phi_0 - \phi_6}{\Delta z}, \frac{\phi_5 - \phi_6}{2\Delta z}$	13	$t_0, x_0, y_0, z_0 + 2\Delta z$
	14	$t_0, x_0, y_0, z_0 - 2\Delta z$
$\frac{\partial^4 \phi}{\partial x^4} = \frac{1}{\Delta x^4} [\phi_9 + \phi_{10} - 4\phi_2 - 4\phi_1 + 6\phi_0]$	15	$t_0, x_0 + \Delta x, y_0 + \Delta y, z_0$
	16	$t_0, x_0 - \Delta x, y_0 - \Delta y, z_0$

$$\frac{\partial^4 \phi}{\partial y^4} = \frac{1}{\Delta y^4} [\phi_{11} + \phi_{12} - 4\phi_3 - 4\phi_4 + 6\phi_0]$$

$$\frac{\partial^4 \phi}{\partial z^4} = \frac{1}{\Delta z^4} [\phi_{13} + \phi_{14} - 4\phi_5 - 4\phi_6 + 6\phi_0]$$

$$\frac{\partial^4 \phi}{\partial x^2 \partial y^2} = \frac{1}{\Delta x^2 \Delta y^2} [\phi_{15} + \phi_{16} + \phi_{17} + \phi_{18} - 2\phi_1 - 2\phi_2 - 2\phi_3 - 2\phi_4 + 4\phi_0]$$

$$\frac{\partial^4 \phi}{\partial x^2 \partial z^2} = \frac{1}{\Delta x^2 \Delta z^2} [\phi_{19} + \phi_{20} + \phi_{21} + \phi_{22} - 2\phi_1 - 2\phi_2 - 2\phi_5 - 2\phi_6 + 4\phi_0]$$

$$\frac{\partial^4 \phi}{\partial y^2 \partial z^2} = \frac{1}{\Delta y^2 \Delta z^2} [\phi_{23} + \phi_{24} + \phi_{25} + \phi_{26} - 2\phi_3 - 2\phi_4 - 2\phi_5 - 2\phi_6 + 4\phi_0]$$

$$\frac{\partial^2 \phi}{\partial t^2} = \frac{1}{\Delta t^2} [\phi_7 + \phi_8 - 2\phi_0]$$

$$\frac{\partial \phi}{\partial t} = \frac{\phi_7 - \phi_0}{\Delta t}, \text{ forward difference}$$

$$\frac{\phi_0 - \phi_8}{\Delta t}, \text{ backward difference}$$

$$\frac{\phi_7 - \phi_8}{2\Delta t}, \text{ average difference}$$

- 17  $t_0, x_0 - \Delta x, y_0 + \Delta y, z_0$
- 18  $t_0, x_0 + \Delta x, y_0 - \Delta y, z_0$
- 19  $t_0, x_0 + \Delta x, y_0, z_0 + \Delta z$
- 20  $t_0, x_0 - \Delta x, y_0, z_0 + \Delta z$
- 21  $t_0, x_0 - \Delta x, y_0, z_0 - \Delta z$
- 22  $t_0, x_0 + \Delta x, y_0, z_0 - \Delta z$
- 23  $t_0, x_0, y_0 + \Delta y, z_0 + \Delta z$
- 24  $t_0, x_0, y_0 - \Delta y, z_0 + \Delta z$
- 25  $t_0, x_0, y_0 - \Delta y, z_0 - \Delta z$
- 26  $t_0, x_0, y_0 + \Delta y, z_0 - \Delta z$

TABLE 3. FINITE DIFFERENCE EXPANSIONS IN CYLINDRICAL COORDINATES

$$\frac{1}{r} \left[ \frac{\partial \phi}{\partial r} \left( r \frac{\partial \phi}{\partial r} \right) \right] = \frac{1}{r_0} \left[ \frac{(2r_0 + \Delta r)(\phi_1 - \phi_0)}{2\Delta r^2} + \frac{(2r_0 - \Delta r)(\phi_2 - \phi_0)}{2\Delta r^2} \right]$$

$$\frac{\partial^2 \phi}{\partial z^2} = \frac{\phi_5 - \phi_0}{\Delta z^2} + \frac{\phi_6 - \phi_0}{\Delta z^2}$$

$$\frac{1}{r^2} \frac{\partial^2 \phi}{\partial \theta^2} = \frac{\phi_3 - \phi_0}{r_0^2 \Delta \theta^2} + \frac{\phi_4 - \phi_0}{r_0^2 \Delta \theta^2}$$

Location of Node Points	
Node No.	Coordinate
0	$r_0, \theta_0, z_0$
1	$r_0 + \Delta r, \theta_0, z_0$
2	$r_0 - \Delta r, \theta_0, z_0$
3	$r_0, \theta_0 + \Delta \theta, z_0$
4	$r_0, \theta_0 - \Delta \theta, z_0$
5	$r_0, \theta_0, z_0 + \Delta z$
6	$r_0, \theta_0, z_0 - \Delta z$

TABLE 4. FINITE DIFFERENCE EXPANSIONS IN SPHERICAL COORDINATES

$$\frac{1}{r_s^2} \frac{\partial}{\partial r_s} \left( r_s^2 \frac{\partial \phi}{\partial r_s} \right) = \frac{1}{r_{s0}^2} \left[ \frac{\left( r_{s0} + \frac{\Delta r_s}{2} \right)^2 (\phi_1 - \phi_0)}{\Delta r_s^2} + \frac{\left( r_{s0} - \frac{\Delta r_s}{2} \right)^2 (\phi_2 - \phi_0)}{\Delta r_s^2} \right]$$

$$\frac{1}{r_s^2 \sin^2 \beta} \frac{\partial^2 \phi}{\partial \alpha^2} = \frac{1}{r_{s0}^2} \left[ \frac{(\phi_3 - \phi_0)}{\Delta \alpha^2 \sin^2 \beta_0} + \frac{\phi_4 - \phi_0}{\Delta \alpha^2 \sin^2 \beta_0} \right]$$

$$\frac{1}{r_{s0}^2 \sin \beta} \frac{\partial}{\partial \beta} \left( \sin \beta \frac{\partial \phi}{\partial \beta} \right) = \frac{1}{r_{s0}^2} \left[ \frac{\sin \left( \beta_0 + \frac{\Delta \beta}{2} \right) (\phi_5 - \phi_0)}{\Delta \beta^2 \sin \beta_0} + \frac{\sin \left( \beta_0 - \frac{\Delta \beta}{2} \right) (\phi_6 - \phi_0)}{\Delta \beta^2 \sin \beta_0} \right]$$

Location of Node Points	
Node No.	Coordinate
0	$r_{s0}, \alpha_0, \beta_0$
1	$r_{s0} + \Delta r_s, \alpha_0, \beta_0$
2	$r_{s0} - \Delta r_s, \alpha_0, \beta_0$
3	$r_{s0}, \alpha_0 + \Delta \alpha, \beta_0$
4	$r_{s0}, \alpha_0 - \Delta \alpha, \beta_0$
5	$r_{s0}, \alpha_0, \beta_0 + \Delta \beta$
6	$r_{s0}, \alpha_0, \beta_0 - \Delta \beta$

coordinates. Table 3 lists the expansions for cylindrical coordinates, and Table 4 summarizes the pertinent approximations in spherical coordinates. Finite difference expansions for complete equations are constructed by

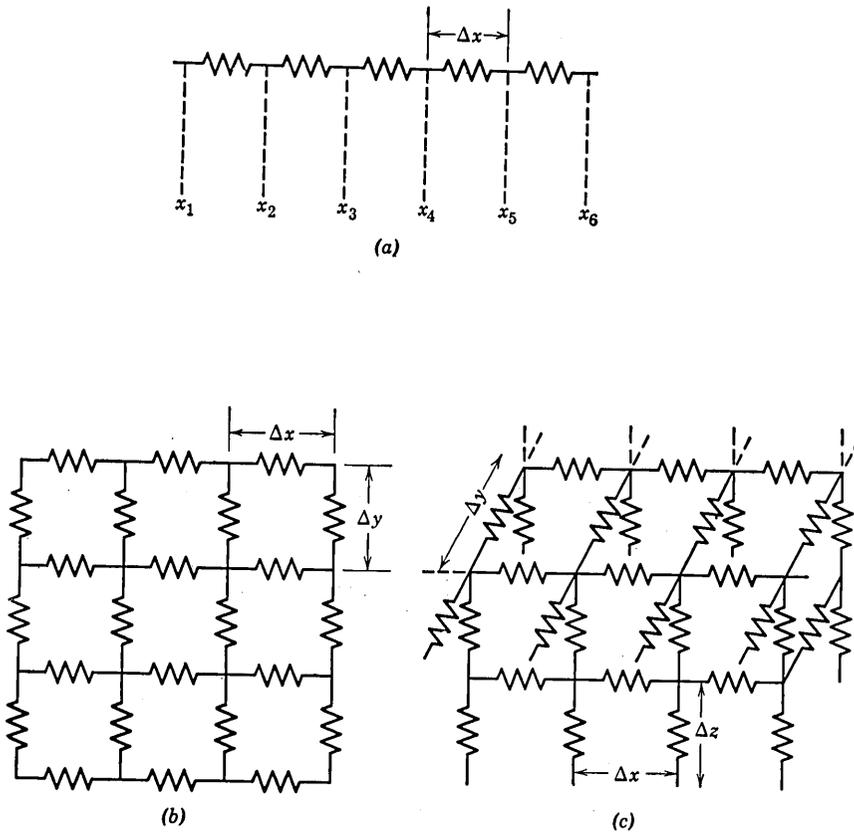


FIG. 7. Resistance networks: (a) one dimension, (b) two dimensions, (c) three dimensions.

combining the expressions for the individual terms making up the equations.

**Resistance Networks.** These network analogs are in general rectangular arrays of electrical resistors. Portions of one-, two-, and three-dimensional resistance networks are shown in Fig. 7. Typical junctions in these networks and the corresponding node equations are presented in Fig. 8.  $V_0, V_1, V_2, \dots$  are voltages at points 0, 1, 2,  $\dots$ . A formal similarity is evidenced between the above node equations and the finite difference equations from Tables 2, 3, and 4, when written as:

25-16 DESIGN AND APPLICATION OF ANALOG COMPUTERS

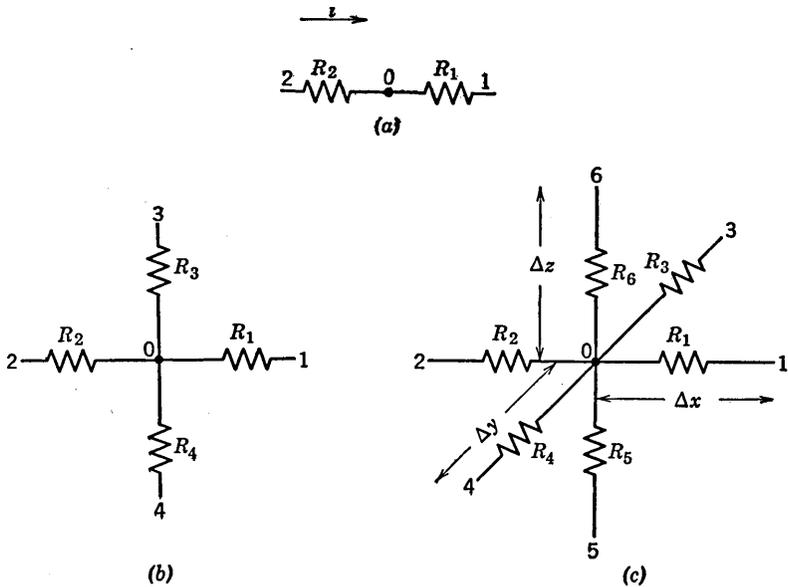


FIG. 8. Typical functions and node equations:

(a) One dimension,

$$\frac{1}{R_1} (V_1 - V_0) + \frac{1}{R_2} (V_2 - V_0) = 0.$$

(b) Two dimensions,

$$\frac{1}{R_1} (V_1 - V_0) + \frac{1}{R_2} (V_2 - V_0) + \frac{1}{R_3} (V_3 - V_0) + \frac{1}{R_4} (V_4 - V_0) = 0.$$

(c) Three dimensions,

$$\begin{aligned} \frac{1}{R_1} (V_1 - V_0) + \frac{1}{R_2} (V_2 - V_0) + \frac{1}{R_3} (V_3 - V_0) + \frac{1}{R_4} (V_4 - V_0) \\ + \frac{1}{R_5} (V_5 - V_0) + \frac{1}{R_6} (V_6 - V_0) = 0. \end{aligned}$$

*Cartesian coordinates* ( $x, y, z$ ):

$$(2) \quad \nabla^2 \phi = \frac{\phi_1 - \phi_0}{\Delta x^2} + \frac{\phi_2 - \phi_0}{\Delta x^2} + \frac{\phi_3 - \phi_0}{\Delta y^2} + \frac{\phi_4 - \phi_0}{\Delta y^2} + \frac{\phi_5 - \phi_0}{\Delta z^2} + \frac{\phi_6 - \phi_0}{\Delta z^2}.$$

*Cylindrical coordinates* ( $r, \theta, z$ ):

$$(3) \quad \nabla^2 \phi = \frac{2r_0 + \Delta r}{2r_0 \Delta r^2} (\phi_1 - \phi_0) + \frac{2r_0 - \Delta r}{2r_0 \Delta r^2} (\phi_2 - \phi_0) \\ + \frac{\phi_3 - \phi_0}{r_0^2 \Delta \theta^2} + \frac{\phi_4 - \phi_0}{r_0^2 \Delta \theta^2} + \frac{\phi_5 - \phi_0}{\Delta z^2} + \frac{\phi_6 - \phi_0}{\Delta z^2}.$$

Spherical coordinates  $(r_s, \alpha, \beta)$ , radial symmetry  $(r_s)$ :

$$(4) \quad \nabla^2\phi = \frac{[r_{s0} + (\Delta r_s/2)]}{r_{s0}^2 \Delta r_s^2} (\phi_1 - \phi_0) + \frac{[r_{s0} - (\Delta r_s/2)]}{r_{s0}^2 \Delta r_s^2} (\phi_2 - \phi_0).$$

A resistance network may, therefore, be employed to simulate the potential distribution in fields governed by *Laplace's equation*. The relative magnitudes of the resistors are determined by comparing the finite difference equation for the particular node under consideration with the corresponding electrical node equation. For example, in the case of a node located at radius  $r_0$ , in a field in polar coordinates, the resistors comprising the two-dimensional net are found by eq. (3) to be

$$(5) \quad \frac{1}{R_1} = \frac{2r_0 + \Delta r}{R 2r_0 \Delta r^2}, \quad \frac{1}{R_2} = \frac{2r_0 - \Delta r}{R 2r_0 \Delta r^2}, \quad \frac{1}{R_3} = \frac{1}{R_4} = \frac{1}{R r_0^2 \Delta \theta^2},$$

where  $R$  is a scale factor introduced to obtain resistors of convenient magnitudes. The magnitudes of resistors along streamline boundaries and in the vicinity of irregular boundaries must be appropriately modified. Similarly, if the properties of the field to be simulated vary with position, or if different mesh spacings are employed in different portions of the field, the relative magnitudes of the network resistors are changed accordingly. Detailed discussions of this technique are presented in Refs. 1 and 5.

**Resistance Networks with Internal Excitation.** Fields with internal distributed excitations governed by Poisson's equation and equations of the type

$$(6) \quad \nabla^2\phi = f(x, y, z, t, \phi)$$

are simulated by introducing currents of the appropriate magnitude at each network node. The equations for typical network junctions under these conditions are given in Fig. 9. The currents are frequently applied to the network nodes by voltage sources coupled to the node by variable resistors. If the value of these feed-in resistors is large compared with that of the network resistors, the precise adjustment of their resistance magnitudes is relatively simple in most cases. Where the applied currents are specified to be a function of the node voltages or their derivatives, the feed-in resistors must be adjusted and readjusted by an iterative procedure. Electronic computer units may be used to advantage for this purpose.

**Applications of Resistance Networks.** Resistance networks have been applied to the solution of the *diffusion equation* by Liebmann (Ref. 6). The method employs a "backward difference" approximation of the time derivative, and its underlying principle is illustrated in

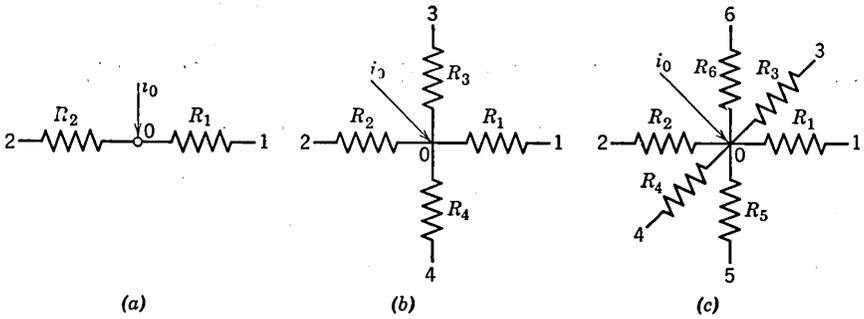


FIG. 9. Typical functions of resistance networks with current applied to each node.

(a) One dimension,

$$\frac{1}{R_1} (V_1 - V_0) + \frac{1}{R_2} (V_2 - V_0) = -i_0.$$

(b) Two dimensions,

$$\frac{1}{R_1} (V_1 - V_0) + \frac{1}{R_2} (V_2 - V_0) + \frac{1}{R_3} (V_3 - V_0) + \frac{1}{R_4} (V_4 - V_0) = -i_0.$$

(c) Three dimensions,

$$\frac{1}{R_1} (V_1 - V_0) + \frac{1}{R_2} (V_2 - V_0) + \frac{1}{R_3} (V_3 - V_0) + \frac{1}{R_4} (V_4 - V_0) + \frac{1}{R_5} (V_5 - V_0) + \frac{1}{R_6} (V_6 - V_0) = -i_0.$$

Fig. 10. The resistances  $R$  form the network analog of the space derivatives. At each node, a current is fed through the feed-in resistor  $R_f$ , whose magnitude is proportional to  $R\Delta t/\Delta x^2$ , in the case of the problems in one space dimension. It can be shown that if the feed-in voltages  $V_n$  represent the potentials existing in the network at time  $n\Delta t$ , the voltages at the network nodes are proportional to the potential distribution at

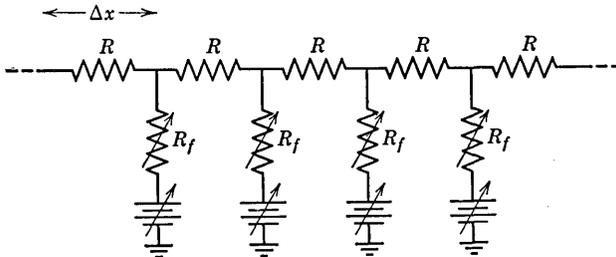


FIG. 10. Resistance network for iterative solution of the diffusion equation.

time  $(n + 1)\Delta t$ . The transient potentials are then obtained by repeating these measurements over as many time increments as required, each time resetting the feed-in voltages.

Liebmann (Ref. 7) is also responsible for a resistance network method for solving the *biharmonic equation* in two space dimensions. Two rectangular resistance networks interconnected by feed-in resistors are employed.

The *wave equation* under steady-state conditions becomes

$$(7) \quad \nabla^2\phi = -K\omega^2\phi.$$

This equation is similar in form to eq. (6) and may be treated by the resistance analog method by introducing a current proportional to  $K\omega^2\phi$  at each node.

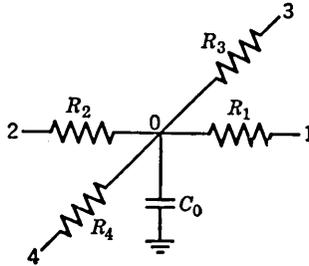


FIG. 11. Typical nodes of  $R$ - $C$  network for simulation of heat conduction in a two-dimensional system,

$$\frac{1}{R_1}(V_1 - V_0) + \frac{1}{R_2}(V_2 - V_0) + \frac{1}{R_3}(V_3 - V_0) + \frac{1}{R_4}(V_4 - V_0) = C_0 \frac{dV_0}{dt}.$$

Comparatively high accuracies are readily obtained with resistance networks. If the manufacturer's tolerance of the individual components is  $\pm 1$  per cent, accuracies within 0.1 per cent are attainable. Truncation errors due to the approximation of a continuous field by a discrete analog may be minimized by refining the network as required.

**Resistance-Capacitance Networks.** Resistance networks modified by connecting a capacitor from each network node to ground are used to simulate fields governed by the diffusion equation. Their principal application is in the area of heat transfer, where thermal analyzers have assumed an important role. Network models of this type are also important in petroleum reservoir engineering in studying unsteady state fluid flow through porous media. A typical node of an  $R$ - $C$  network is shown in Fig. 11.

In selecting resistor and capacitor values, the solution time is first

## 25-20 DESIGN AND APPLICATION OF ANALOG COMPUTERS

“scaled” to have a convenient magnitude. The element values are then determined by establishing a term-by-term correspondence between the finite difference equation and the node equation of the electric network.

The basic  $R$ - $C$  network technique has been extended by Paschkis (Ref. 8) and others to the treatment of fields with time-varying parameters and nonlinear parameters, fields in which latent heats of fusion play

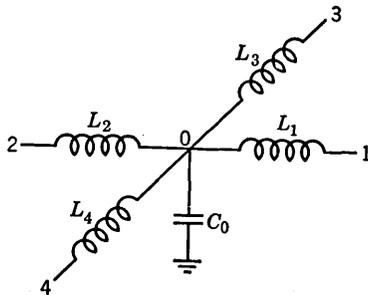


Fig. 12. Typical node of network analog for wave equations,

$$\frac{1}{L_1} (V_1 - V_0) + \frac{1}{L_2} (V_2 - V_0) + \frac{1}{L_3} (V_3 - V_0) + \frac{1}{L_4} (V_4 - V_0) = C_0 \frac{d^2 V_0}{dt^2}.$$

a part, and a multitude of combinations of initial and boundary conditions.

**Inductance-Capacitance Network.** Networks containing only reactors are occasionally employed for the simulation of fields governed by the wave equation. The structure of the network is similar to that of the  $R$ - $C$  systems described above, with inductors replacing the resistance elements. A typical node of such a network is illustrated in Fig. 12.

At any given frequency an inductor has a positive reactance of constant magnitude, while a capacitor has a negative reactance of constant magnitude. This property has led to the application of  $L$ - $C$  networks to problems generally treated by means of resistance networks, but in which negative resistors are required.

**$R$ - $L$ - $C$  Networks.** Networks including large numbers resistors, inductors, capacitors, as well as transformers, all of which may be interconnected as desired, constitute a powerful tool for the solution of numerous problems occurring particularly in the areas of power utilization and distribution, and the stress and vibration analysis of elastic structures. So-called a-c network analyzers for the simulation of electric power lines, generators, loads, and transformer stations play an important part in the planning of electric power systems. A partial list of problems handled by this method includes:

*Load division problems:* circuit loadings, desirable locations of new generators, effect of increased system loadings, effects of system reactances, ratings of transformers.

*Short-circuit problems:* effect of three-phase and unbalanced faults, evaluation of protective measures.

*Stability problems:* steady-state and transient limits, remedial measures.

Such networks are, of course, also useful in solving problems governed by any of the basic field equations of engineering and physics.

McCann (Ref. 9) has developed a large scale, highly flexible network analyzer for the solution of static and transient beam problems. The network permits the study of one-, two-, and three-dimensional problems with a wide variety of end conditions. A typical application of this analog simulator is illustrated in Fig. 13.

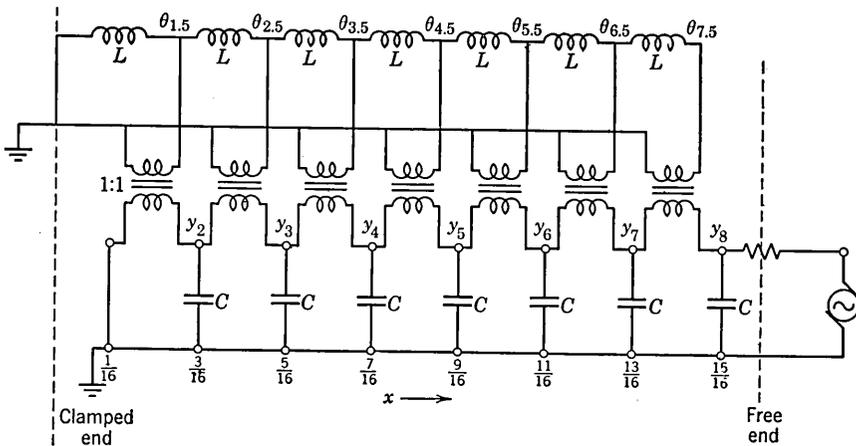


FIG. 13. Circuit for steady-state vibration of a uniform cantilever beam with eight cells:  $\theta$ , slope;  $y$ , vertical deflection;  $L$ , inductance;  $C$ , capacitance (Ref. 9).

**Use of Electronic Analog Computers.** Howe (Ref. 1) and Rogers (Ref. 5) have described techniques for employing d-c analog computers at each node of an analog network to perform the additions and integrations specified by the finite difference equation for that node. Complex-element transfer functions (see Chap. 22) may be used to advantage to obtain the required response with a minimum of equipment. Fields governed by the wave equation, equations containing first space derivatives, and fields with certain nonlinearities, are successfully treated by this method.

## 25-22 DESIGN AND APPLICATION OF ANALOG COMPUTERS

In general, however, the d-c analog computer finds its most important application in the solution of field problems as an adjunct to passive network analogs. The generation of time functions comprising the excitation of network models is readily accomplished with this equipment. The application of currents to the internal network nodes is likewise facilitated.

### 4. NONELECTRIC FIELD ANALOGS

Numerous mechanical, thermal, chemical, and hydraulic systems satisfying *Laplace's equations* have been employed from time to time to locate the equipotential and streamlines in two- and three-dimensional fields. For the most part, these methods are qualitative at best and have in recent years been replaced to a large extent by the electric analog models. Some of the more familiar of these systems are briefly discussed below and are considered in more detail in Refs. 1, 5, and 10.

**Stretched Membranes.** If a very thin film of negligible weight is placed under uniform tension, its vertical displacement, perpendicular to the plane of the membrane, is determined by Laplace's equation. A model of the field boundary is constructed so that its height is proportional to the boundary potentials of the field under study, and a thin membrane is then stretched over this boundary structure. The membrane then adjusts itself so that its height at interior points of the model is proportional to the potential field at the corresponding points in the original system. Commonly used membranes include soap films and thin sheets of rubber. Considerable skill is required in the application of this method to complex regions and fields with internal excitations. Models of this type have been used widely in the area of electron optics.

**Fluid Mappers.** The pressure in an inviscid incompressible fluid obeys Laplace's equation. This condition may also be approached by causing a viscous, incompressible fluid such as glycerin to flow between two closely spaced parallel walls. Dyes such as potassium permanganate may be introduced into the flow stream at discretely spaced points and a visual display of the streamline pattern obtained. Regions of varying characteristics are readily studied by this method.

**Blotter Type Electrolytic Models.** The motion of ions in an electrolyte under the influence of externally applied electric potentials is governed by Laplace's equation. Ionized salts solution may therefore be injected into an electrolytic system at points corresponding to sources or sinks in the original system and their progress through the system noted. The motion of the ions can be made visible by employing electrolyte-ion solution combinations which change color on contact. For *example*, a piece of blotting paper may be soaked in a colorless

electrolyte containing phenolphthalein, which turns red in the presence of  $\text{OH}^-$  ions. A weak solution of sodium hydroxide is then used as the injected fluid, and a permanent record of the flow pattern may be obtained by photographing the paper at regular time intervals. This method has been used successfully to study the shapes of injection fluid fronts in petroleum engineering problems.

**Electro-Optical Method.** A liquid having a large Kerr constant, such as a colloidal suspension of bentonite, is placed between two parallel pieces of polaroid, and is subjected to an electric field determined by the boundary conditions of the field to be simulated. The applied voltage (approx. 60 cps ac) effects a change in the polarization characteristics of the liquid so that, if light is beamed through the liquid and the polaroid boundaries, a distinctive pattern of dark lines is observed. These are then related to the potential field within the liquid.

**Guebard's Rings.** A polished sheet of copper is placed in an electrolytic tank, parallel to the air-electrolyte interface. Solutions of copper and lead acetates are used as the conductive liquid, and deposits of lead peroxide form on the surface of the copper sheet when the system is excited electrically. Lines of constant color of this deposit correspond to the equipotential lines of the field.

**Lichtenberger Figures.** A high transient voltage is applied to a piece of photographic paper so that a discharge occurs across its surface. Upon developing, a pattern corresponding to the streamlines of the field are observed on the paper.

## REFERENCES

1. W. J. Karplus, *Analog Simulation: Solution of Field Problems*, McGraw-Hill, New York, 1958.
2. G. Hotchkiss, Electrosensitive recording paper for facsimile telegraphic apparatus and graphic chart instruments, *Western Union Tech. Rev.*, **2**, 176-187 (1948).
3. P. C. Green, Automatic plotting of electrostatic fields, *Rev. Sci. Instr.*, **19**, 646-653 (1948).
4. A. R. Boothroyd, E. C. Cherry, and R. Makar, An electrolytic tank for the measurement of steady state response, transient response, and allied properties of networks, *Proc. Inst. Elec. Engrs.*, **96**, Pt. I, 163-177 (1949).
5. W. W. Soroka, *Analog Methods in Computation and Simulation*, McGraw-Hill, New York, 1954.
6. G. Liebmann, A new electrical analog method for the solution of transient heat conduction problems, *Trans. Am. Soc. Mech. Engrs.*, **73**, 655-665 (1956).
7. G. Liebmann, Solution of plane stress problems by an electric analogue method, *Brit. J. Appl. Phys.*, **6**, 145-157 (1955).
8. V. Paschkis and H. D. Baker, A method for determining unsteady-state heat

## 25-24 DESIGN AND APPLICATION OF ANALOG COMPUTERS

transfer by means of an electrical analogy, *Trans. Am. Soc. Mech. Engrs.*, **64**, 105-112 (1942).

9. G. D. McCann and R. H. McNeal, Beam vibration analysis with electric analog computer, *J. Appl. Mechanics*, **17**, 13-26 (1950).

10. P. E. Green, Methods of plotting electrostatic fields, *Eng. School Bull.* 45, Dept. of Eng. Research, N. Carolina State College, Raleigh, 1949.

11. P. A. Einstein, Factors limiting the accuracy of electrolytic plotting tanks, *Brit. J. Appl. Phys.*, **2**, 49-55 (1951).

12. K. F. Sanders and J. G. Yates, The accurate mapping of electric fields in an electrolytic tank, *Proc. Inst. Elec. Engrs.*, **100**, Pt. II, 167-183 (1953).

## Noise and Statistical Techniques

*Henry Low*

1. Introduction and Definition	26-01
2. Random Variable Concepts	26-02
3. Treatment of Linear Systems	26-06
4. Treatment of Nonlinear Systems	26-09
5. Noise Generators	26-12
References	26-20

### I. INTRODUCTION AND DEFINITION

Noise theory has become an essential element in communications and control systems. In such systems noise is usually the predominant factor limiting range, sensitivity, and reliability of the equipment, consistent with economic and size considerations. On the other hand, the development of efficient testing procedures based on the use of a noise wave as input to a system has re-evaluated the concept that noise is inherently evil (Refs. 1 and 2).

*Noise*, from the point of view of this chapter, is perhaps best defined as phenomena which may not be predicted with complete certainty. These random phenomena include thermal noise arising from Brownian motion of electrons in kinetic equilibrium with the molecules of a conductor, static from electrical disturbances in the atmosphere, contact noise associated with fluctuating conductivity, random mechanical vibrations such as encountered in electron tube elements and aircraft landing gears, atmospheric air turbulence and random fluctuations in manufacturing tolerances.

## 26-02 DESIGN AND APPLICATION OF ANALOG COMPUTERS

In many branches of engineering and science there occur problems whose solutions may be expressed only in statistical terms. A notable example is the frequent occurrence of such problems in modern aerial guidance and control systems. These systems are seldom designed to perform a single task which may be completely specified beforehand; rather, they are designed to perform a task selected at random from a complete repertory of possible tasks. Analysis of the behavior of systems whose forcing functions are a combination of predictable and random functions, is often complicated by the fact that these systems are non-linear, time-varying, or both. With proper random function generators and appropriate computing techniques, the analog computer approach is a powerful means for studying the effects of random disturbances on complicated systems. (See Chap. 21, Sect. 6, for a brief table of symbols.)

### 2. RANDOM VARIABLE CONCEPTS

**Probability Theory.** An understanding of the basic concepts of probability theory is essential to the analysis of statistical problems on an analog computer. (See Vol I, Chap. 12.) Only a rudimentary explanation of statistical terms applicable to this chapter will be given here. For more information on this subject see Ref. 3.

Consider the quantitative description of the voltage output  $x(t)$  of a noise generator most commonly used with analog computers and described in Sect. 5. The output of the noise generator will fluctuate in a random fashion around some average value. It is impossible to predict what the value of the noise voltage  $x(t)$  will be at any given time; however, the problem may be clearly formulated in terms of probability theory.

**Ensemble and Probability Functions.** The outputs  $x_n(t)$  of a large number of identical noise generators (an *ensemble* of noise generators) are recorded at a definite time  $t$ . If the number of voltages falling in a voltage interval  $\Delta x$  is counted, a *probability density function* of  $x(t)$  is approximated. This probability density function  $p(x, t)$  is defined by

$$(1) \quad p(x, t) = \lim_{n \rightarrow \infty, \Delta x \rightarrow 0} \frac{(\text{Number of voltages in range } \Delta x) / \Delta x}{\text{Total number of noise generators} = n},$$

The probability that a particular measured voltage lies in an infinitesimal interval  $dx$  is given by  $p(x, t) dx$  and the probability that the measured voltage lies in a range  $x_a$  to  $x_b$  is given by

$$(2) \quad Pr(x_a < x < x_b) = \int_{x_a}^{x_b} p(x, t) dx.$$

The *probability distribution function*  $P(x, t)$  is defined as the probability that the measured voltage is less than some specified voltage  $x$ . It is given by

$$(3) \quad P(x, t) = \int_{-\infty}^x p(x, t) dx.$$

The average or mean value of the ensemble of voltages  $x_n(t)$  may be found from

$$(4) \quad \overline{\widetilde{x}(t)} = \int_{-\infty}^{+\infty} xp(x, t) dx,$$

and will in general depend on the time  $t$ .  $\overline{\widetilde{x}(t)}$  can also be determined by averaging at time  $t$  the noise outputs  $x_1(t), x_2(t), \dots, x_n(t)$  of the ensemble of noise generators; hence,  $\overline{\widetilde{x}(t)}$  is termed the *ensemble average*.

**Stationary and Ergodic Processes.** If the above-mentioned noise generators do not change their characteristics with time, each will generate a *stationary* output; i.e., the noise outputs will have statistics which do not change with time. In dealing with stationary random processes it is usually assumed that time averages are equivalent to ensemble averages. This is the so-called ergodic hypothesis of statistical mechanics. Since the nature of the underlying mechanism does not change with time, it is expected that a large number of observations made on a single noise generator at randomly chosen times will have the same statistical properties as the same number of observations made on randomly chosen noise generators at the same time.

The noise output from a noise generator is said to belong to an *ergodic* process if its *time average*

$$(5) \quad \overline{x(t)} = \lim_{T \rightarrow \infty} \frac{1}{2T} \int_{-T}^T x(t) dt,$$

is equal to the ensemble average  $\overline{\widetilde{x}(t)}$ . In this case the output record for a long period of time from a single noise generator is sufficient to determine  $p(x)$ , which now will be independent of time. The ensemble and time averages of any function of  $x(t)$  are given, respectively, by

$$(6) \quad \overline{\widetilde{f[x(t)]}} = \int_{-\infty}^{\infty} f[x(t)] p(x, t) dx,$$

and

$$(7) \quad \overline{f[x(t)]} = \lim_{T \rightarrow \infty} \frac{1}{2T} \int_{-T}^T f[x(t)] dt,$$

where  $f[x(t)]$  is any function of  $x(t)$  such as  $x, x^2, (x - \bar{x})^2$ , etc.

**Variance and Standard Deviation.** Time and ensemble averages are very useful in describing a random process. Besides physical interpreta-

## 26-04 DESIGN AND APPLICATION OF ANALOG COMPUTERS

tions, such as the d-c value of the output of the noise generator given by  $\overline{x(t)}$  and the power output given by  $\overline{(x - \bar{x})^2}$ , the averages are measures of the shape of the probability density function.  $\overline{(x - \bar{x})^2}$  is termed the *variance* and is a measure of the spread of the probability density function. The positive square root of the variance is termed the *standard deviation*  $\sigma$ .

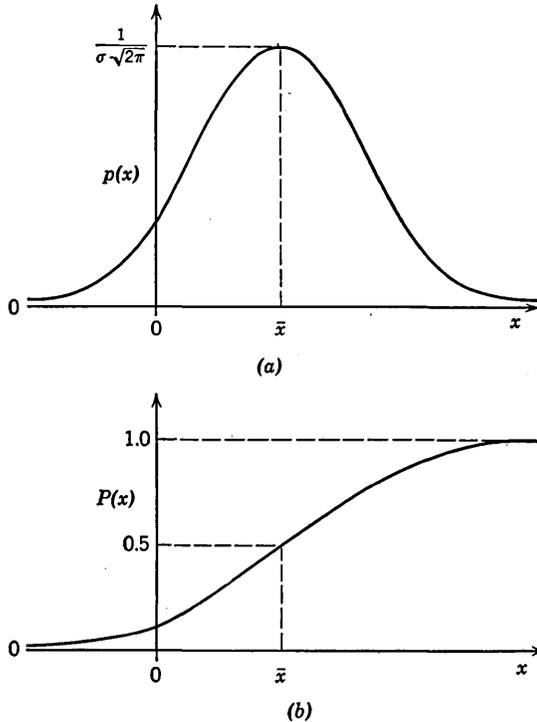


FIG. 1. (a) Probability density function for Gaussian distribution. (b) Probability distribution function for Gaussian case.

**Gaussian Distribution.** A probability density function which occurs very frequently in physical problems is the *normal* or *Gaussian* function defined by

$$(8) \quad p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp \left[ -\frac{(x - \bar{x})^2}{2\sigma^2} \right],$$

and is illustrated in Fig. 1a. The associated probability distribution function for the Gaussian case

$$(9) \quad P(x) = \frac{1}{2} \left[ 1 + \operatorname{erf} \left( \frac{x - \bar{x}}{\sqrt{2}\sigma} \right) \right],$$

where the *error function* (erf) is defined as

$$\operatorname{erf} z = \frac{2}{\sqrt{\pi}} \int_0^z \exp[-\lambda^2] d\lambda$$

and is illustrated in Fig. 1b.

**Autocorrelation and Spectral Density.** The average value of the product of a function of time with the function displaced  $\tau$  seconds is termed the *autocorrelation function*. This function is important because it is the link between the direct or time description and the frequency component description of a random variable. For an ergodic process the autocorrelation function is given by

$$(10) \quad \phi(\tau) = \overline{x(t)x(t+\tau)} = \lim_{T \rightarrow \infty} \frac{1}{2T} \int_{-T}^T x(t)x(t+\tau) dt, \quad T \rightarrow \infty.$$

The majority of statistical problems studied on an analog computer concern themselves with the effect of a random variable on a frequency filter. The description of the spectral structure of a random function is a very useful and important concept; this description is the spectral density  $\Phi(\omega)$ . (See Vol. I, Chaps. 17 and 24.) In this treatment  $W(f)$  will be used as the spectral density, since most measurements are made in cycles per second rather than in radial frequency. To convert  $W(f)$  to  $\Phi(\omega)$  replace  $f$  by  $\omega/2\pi$ .

For a stationary random voltage  $x(t)$  impressed across a 1-ohm resistor,  $W(f) df$  is the average power dissipated in the resistance in the frequency interval  $(f, f + df)$ . In this case,  $W(f)$  has the dimensions of (volt)<sup>2</sup>/cps. The principal usefulness of the spectral density lies in the fact that if  $Y(2\pi jf)$  is the transfer function of a linear time-invariant system and  $W_i(f)$  is the spectral density of the input, then the output spectral density  $W_o(f)$  is

$$(11) \quad W_o(f) = |Y(2\pi jf)|^2 W_i(f).$$

The correlation function and power spectral density are Fourier transforms of each other:

$$(12) \quad \phi(\tau) = \int_{-\infty}^{+\infty} W(f) e^{i2\pi f\tau} df$$

$$(13) \quad W(f) = \int_{-\infty}^{+\infty} \phi(\tau) e^{-i2\pi f\tau} d\tau.$$

**White Noise.** The concept of white noise is a useful mathematical tool and arises in connection with such physical phenomena as shot noise and thermal noise. White noise has a Gaussian probability distribution and a spectral density  $N$  that spreads uniformly over all frequencies.

## 26-06 DESIGN AND APPLICATION OF ANALOG COMPUTERS

Noise is considered white with respect to any given system if the bandwidth of the system is small compared with the noise spectrum.

### 3. TREATMENT OF LINEAR SYSTEMS

*Note.* Portions of the material used in Sects. 3 and 4 were taken from an article by R. R. Bennett (Ref. 4).

**Definitions.** A system is said to be linear provided that the principle of superposition can be applied; i.e., for every pair of inputs  $[x_1(t), x_2(t)]$  and corresponding outputs  $[y_1(t), y_2(t)]$ , the input  $ax_1(t) + bx_2(t)$  produces an output  $ay_1(t) + by_2(t)$ , where  $a$  and  $b$  are arbitrary constants or time-varying coefficients. Time-varying systems are thereby included. Any linear system may be completely described by a *unit-impulse-response* or *weighting function*  $h(t_2, t_1)$  which is defined as follows:

$h(t_2, t_1)$  is the system output measured at time  $t_2$  in response to a unit impulse  $\delta(t - t_1)$  applied at time  $t_1$ , where  $t_2 \geq t_1$ .

The *fundamental characteristic* of constant coefficient linear systems is that their weighting functions depend solely upon the difference between the instant of observation  $t_2$  and the time  $t_1$  of application of the unit impulse.

If the input to the linear system is  $x(t)$  the output  $y(t)$  at a particular time  $t_2$  is by superposition

$$(14) \quad y(t_2) = \int_{-\infty}^{t_2} x(t)h(t_2, t_1) dt_1.$$

If the input to the linear system is white noise of spectral density  $N$ , where  $N$  is measured over real frequencies only, the mean square ensemble output at time  $t_2$  is given by

$$(15) \quad \overline{y^2(t_2)} = \frac{N}{2} \int_{-\infty}^{t_2} h^2(t_2, t_1) dt_1.$$

The mean square output is usually the most significant value of interest.

For stationary noise of spectral density  $W(f)$ , the *mean square ensemble output* is given by

$$(16) \quad \overline{y^2(t_2)} = \int_0^{\infty} W(f) |H(t_2, s)|_{s=j2\pi f}^2 df,$$

where  $H(t_2, s)$  is the Laplace transform of the weighting function. Equations (14) and (15) show that the weighting function  $h(t_2, t_1)$  is needed if the output to a system for an arbitrary input is desired. The role of the analog computer is that of determining the weighting function, since this may be difficult by analytical methods.

If a unit impulse is introduced at time  $t_1$ , a simulated system gives  $h(t_2, t_1)$  as a function of  $t_2$ . Since the variable of integration in eqs. (14) and (15) is  $t_1$ , to evaluate  $\overline{y(t)^2}$  for a particular value of  $t_2$  it would appear necessary to take a number of unit impulse responses for various values of  $t_1$ , cross-plot the results, and integrate eqs. (14) and (15) numerically. Laning and Battin (Refs. 5, 6) have shown that by replacing the system under study by another that is closely related to it,

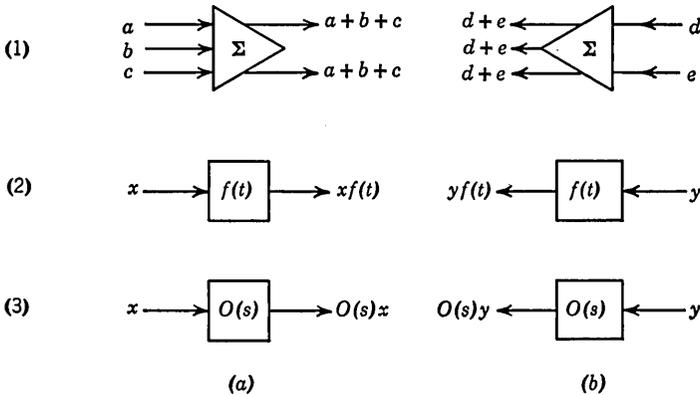


FIG. 2. Construction of adjoint system used to obtain weighting function: (a) building blocks of original system; (b) adjoint system. The three types of operations are (1) addition, (2) multiplication by a function  $f(t)$ , and (3) application of a linear constant coefficient operator,  $O(s)$ .

called the *adjoint system*, the desired weighting function may be computed directly in terms of the variable  $\tau = t_2 - t_1$ . The adjoint technique can often effect a considerable saving in computing time.

**Adjoint Method.** Any linear system may be instrumented by a combination of three types of operations. In terms of analog computer block diagrams these three operations are shown in Fig. 2a. They are summation, multiplication, and linear constant coefficient operations. By using the same topological system block diagram, but with the direction of flow through each block reversed as shown in Fig. 2b, the adjoint system is evolved. Note that new variables appear in the adjoint.

The input to the original system becomes the output of the adjoint system, and vice versa. The *adjoint system* has the property that its response to a unit impulse, applied at time  $t_2$ , is the desired weighting function. The difficulty of having time run backwards is circumvented by making the mathematical change of variable  $\tau = t_2 - t_1$ , where  $\tau$  is now the independent variable for the computer. Since the change of variable has shifted the origin of time as well as reversed the direction of time, the unit impulse input is applied at time  $t_1 = t_2$  or  $\tau = 0$ , which

## 26-08 DESIGN AND APPLICATION OF ANALOG COMPUTERS

corresponds to the instant of starting the simulator test. All time-varying parameters in the system will be generated as if the independent variable were decreasing from  $t_2$  to  $-\infty$  rather than increasing from  $-\infty$  to  $t_2$ .

The adjoint technique is also valid for nonstationary inputs to the linear system, provided that a suitable time-varying shaping filter can be found.

If physical hardware is employed in the simulation the use of the adjoint technique is generally not convenient. The presence of time-varying elements within the hardware may preclude reversal of the system to form the adjoint.

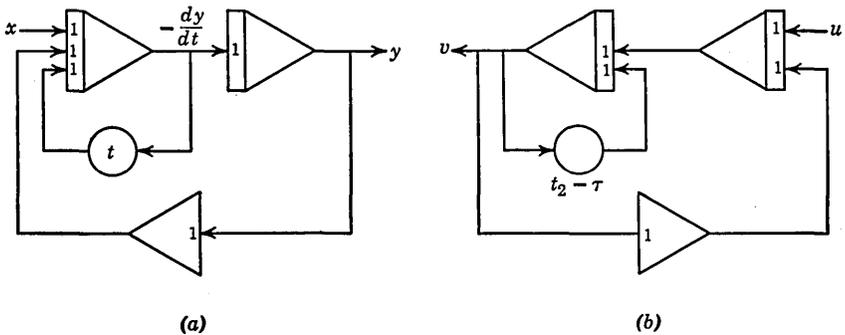


FIG. 3. (a) Schematic diagram for solution of eq. (17). (b) Schematic diagram for solution of adjoint of eq. (18).

$$(a) \frac{d^2y}{dt^2} + t \frac{dy}{dt} + y = x \quad (b) \frac{d^2v}{dt_1^2} + \frac{d}{dt} [(t_2 - \tau)v] + v = u.$$

EXAMPLE. Consider the system described by the following second order differential equation:

$$(17) \quad \frac{d^2y}{dt^2} + t \frac{dy}{dt} + y = x.$$

This system may be simulated by using only integrators, summers, and time-varying scale factor potentiometers, as shown in Fig. 3a. The adjoint system is obtained from this simulation by interchanging all inputs and outputs of each of the computing elements and replacing the variable  $t$  by  $t_2 - \tau$ . The schematic diagram for the adjoint system is shown in Fig. 3b.

The differential equation instrumented with the schematic of Fig. 3b is found to be

$$(18) \quad \frac{d^2v}{dt_1^2} + \frac{d}{dt_1} [(t_2 - \tau)v] + v = u.$$

Equation (18) is the adjoint of the equation of the original system with  $t$  replaced by  $t_2 - \tau$ . The collection of responses of the original system at time  $t_2$  as a result of unit impulses applied at all times prior to time  $t_2$  may be generated by a single computer run by recording the response of the adjoint system when this system is excited by a unit impulse  $\delta(\tau)$  at time  $\tau = 0$ . The unit impulse input is accomplished by placing an initial condition on an integrator.

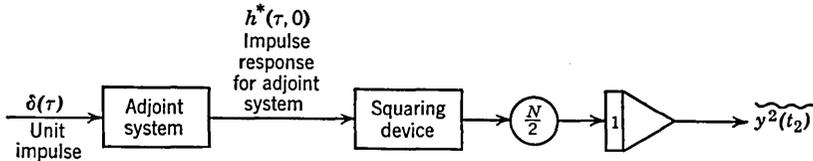


FIG. 4. Block diagram for obtaining mean square ensemble output with adjoint system.

If the input to the system is white noise of spectral density  $N$ , the relation between the mean square ensemble output of the original system and the weighting function of the modified system,  $h^*$ , is obtained from eq. (15):

$$(19) \quad \widetilde{y^2(t_2)} = \frac{N}{2} \int_{-\infty}^{t_2} h^2(t_2, t_1) dt_1 = \frac{N}{2} \int_0^{\infty} h^{*2}(\tau, 0) d\tau.$$

The mean square ensemble output may be obtained by a single computer run as indicated in Fig. 4. The quantity  $\widetilde{y^2(t)}$  is the steady-state value of the integrator output.

#### 4. TREATMENT OF NONLINEAR SYSTEMS

For nonlinear systems, when attempting to find the response to statistical inputs, in general it is necessary to generate the appropriate inputs and apply these to the analog computer. The measurements taken from the analog computer will then exhibit a statistical character. Section 5 describes random function generators for analog computers.

If a complete time history of the variables of interest is desired, the usual recording methods common to analog computer operation are applicable. If, however, the values of the variables at a specific instant are desired, it is expeditious to record only those values. It is economical to provide some automatic means for repeating solutions on the computer (Ref. 7); the only difference between successive solutions is that a different sample of noise or random input is applied each time. Mathematically, the form of the equations is the same each time; the forcing

## 26-10 DESIGN AND APPLICATION OF ANALOG COMPUTERS

TABLE 1. STATISTICAL PROBLEMS AMENABLE TO ANALOG COMPUTER STUDY

Problem Field	Random Variable	References	
1. Radar	Angular scintillation due to multiple reflections from targets (target glint)	8 to 12	
	Amplitude scintillation due to atmospheric disturbances (fading)	8 to 12	
	Receiver noise	11, 13, 14	
	Quantizing errors in analog-to-digital signal conversion		
2. Missile and aircraft systems	Launching errors	8, 9, 12	
	Target maneuver	8, 9, 12, 15	
	Thrust vector variations		
	Wind disturbances	16, 17	
	Instrument errors		
	Human response time	16, 18, 19	
3. Servo system	Probability of hit	9, 58	
	Input	2, 4, 5, 6, 20 to 25	
	Gear noise	26	
	Potentiometer noise	26	
	System gain	27	
4. Aerodynamics	Atmospheric conditions		
	Winds	16, 17	
	Turbulence		
	Air density		
	Temperature		
	Runway roughness (land and sea)	28	
5. Operations research	Arrival time	29, 31	
	Economics		
	Service time		
	Inventory	Mean time in queue	29
	Traffic	Manufacturing irregularities	29
	Cargo handling	Human response time	
6. Data reduction	Economic fluctuations	30	
	Data	32, 33, 34	
7. Filter analysis and synthesis	Input	11, 14, 35, 36	
8. Noise analysis, generation, and measurement	Noise	7, 10, 14, 21, 22, 23, 25, 37 to 55, 59	

functions, initial conditions, and certain parameters vary randomly from solution to solution. The statistical measure now consists of an ensemble measure over the possible solutions of the same transient problem. It is important that the significance which can be attached to a set of such measurements be fully understood (Ref. 4).

**Random Variables.** The random inputs to the analog computer correspond to the various quantities of the problem of interest which are statistical in nature. These statistical quantities may be of various types. Table 1 indicates a number of problems that can be studied on an analog computer and the random variables that are usually associated with that particular problem. A selected list of references is given. The indicated references show specific applications of analog computers to statistical problems.

**EXAMPLE.** Consider the simulation of stationary, normally distributed radar scintillation noise  $\mu$  with a zero mean and a spectral density given by

$$(20) \quad W(f) = \frac{W_{f=0}}{(f/f_0)^2 + 1} = \frac{3000}{(f/2)^2 + 1} \text{ ft}^2/\text{cps}.$$

The standard deviation or root-mean-square value of the radar noise is obtained from

$$(21) \quad \sigma_\mu = \sqrt{\int_0^\infty W(f) df} = \sqrt{\frac{\pi W_{f=0} f_0}{2}} = 97.08 \text{ ft}.$$

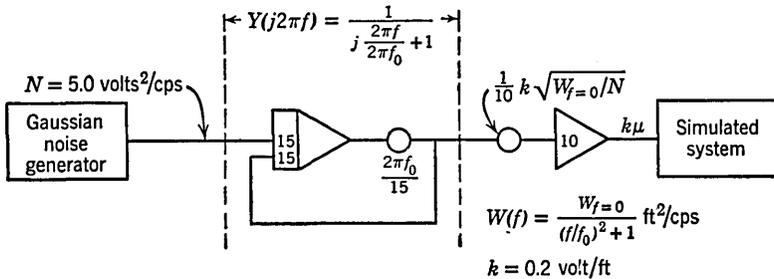


FIG. 5. Schematic of Radar noise simulation.

Assume that the analog computer Gaussian noise generator has an output spectral density  $N$ , which is uniform from zero to 50 cps with a value of 5.0 volts<sup>2</sup>/cps. The noise voltage has a zero mean and an rms voltage in the order of 18.0 volts.

Since the noise generator has a constant spectral density over a frequency range much greater than the cutoff frequency  $f_0$  of the noise to be simulated, it can be considered as a source of white noise.

By using the relationship of eq. (10) with  $W_i = N = 5.0$  volts<sup>2</sup>/cps and a computer scale factor  $k = 0.2$  volt/foot, the radar noise may be simulated as indicated in Fig. 5.

## 26-12 DESIGN AND APPLICATION OF ANALOG COMPUTERS

### 5. NOISE GENERATORS

**Desirable Characteristics.** An analog computer noise generator should possess the following characteristics:

1. A stationary output voltage with the desired probability distribution.

2. An output voltage of adequate root-mean-square value in order to reduce further amplification requirements.

3. A stable power spectral density that is uniform from zero frequency to a frequency that is sufficiently large when compared to the noise spectrum to be simulated.

4. A stable zero mean. A mean other than zero is obtained by adding the required d-c voltage.

5. An averaging type output monitoring circuit with a large time constant.

**Noise Sources.** The two most commonly used primary noise sources for analog computer noise generators are (a) a radioactive emitter with an associated Geiger-Mueller tube and (b) a grid-controlled gas-discharge tube.

Random numbers (Refs. 52, 53), current-carrying resistors (Refs. 1, 50), temperature-saturated diodes (Ref. 1), and photoelectron multiplier tubes (Ref. 1) have also been used as primary noise sources.

**Radioactive Source.** Gamma and beta radiations emitted by a radioactive source and triggering a self-quenching Geiger-Mueller tube will result in random voltage pulses with a *Poisson probability distribution function* given by

$$(22) \quad P(n) = \frac{(\lambda T)^n \exp(-\lambda T)}{n!}.$$

That is, in any fixed time of observation  $T$ , the number of voltage pulses  $n$  obtained in that interval would have the probability distribution  $P(n)$ , where  $\lambda$  is the average rate of occurrence of the pulses in a unit time interval.

The GM tube is a gas-filled diode operated in the region of unstable corona discharge. The condition for triggering a discharge in the GM tube is that at least one low-energy electron be produced within the GM tube by the radioactive rays. A voltage pulse is emitted by the formation of an ion sheath which first forms at the anode and then moves to the cathode. The deionization time of the GM tube is the fundamental barrier limiting the highest noise frequency available from a tube (Ref. 56). The Anton 315 S, Victoreen 1 B 85 and Raytheon CK 1019 GM tubes have been used for noise generator applications.

**Gas-Discharge Tube.** The noise in a grid-controlled gas-discharge tube arises from fluctuations in the dense layer of positive ions near the cathode. The noise has a normal probability density distribution and the frequency beyond which the noise energy begins to fall off varies inversely with the atomic weight of the gas. Resonance peaks associated with the natural oscillation frequencies of the ions may be suppressed by application of a transverse magnetic field from a permanent magnet enclosing the tube (Ref. 1). Most gas-discharge tubes exhibit an inherent instability with distinct jumps in noise level occurring at random times.

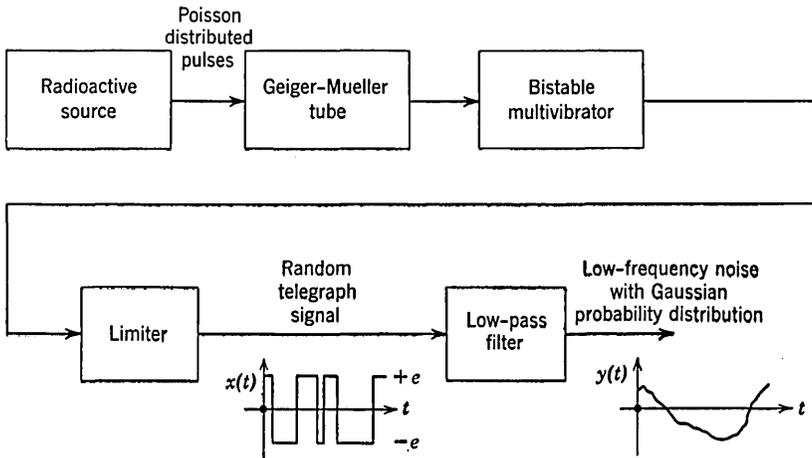


FIG. 6. Block diagram of Geiger-Mueller type generator.

These jumps have been attributed to (a) variations in heater and plate supply voltages, (b) variations in tube envelope temperatures, and (c) variations in cathode emission. Since these variations are at low frequencies, noise generators using gas tubes resort to a regulator circuit to compensate for them (Ref. 39). The 6D4 argon-filled triode and the 2D21 xenon-filled tetrode are extensively used as hot cathode arc noise sources.

**Geiger-Mueller Type Noise Generators.** The basic principle of this method is the utilization of a radioactive material and a Geiger-Mueller tube to trigger a bistable multivibrator circuit (Refs. 41, 43, 45, 51). A block diagram of a GM type noise generator is shown in Fig. 6. The output of the multivibrator is clipped at equal positive and negative voltages. The result is a *random telegraph signal* which is defined as a signal that has two possible levels of  $+e$  and  $-e$  volts, the crossovers between these levels being Poisson distributed in time.

## 26-14 DESIGN AND APPLICATION OF ANALOG COMPUTERS

The spectral density  $W(f)$  of the random telegraph signal can be shown to be

$$(23) \quad W(f) = \frac{e^2}{\lambda} \frac{1}{1 + \left(\frac{2\pi f}{2\lambda}\right)^2},$$

where  $e$  is the magnitude of the positive or negative excursion from zero of the random telegraph signal and  $\lambda$  is the average rate of occurrence of the Poisson distributed pulses. If the random telegraph signal is passed through a suitable low-pass filter, the filter output will have an amplitude probability density distribution which is approximately Gaussian. The power spectral density of the output will be essentially uniform from zero frequency to the filter cutoff frequency, provided the input power spectrum  $W(f)$  of the random telegraph signal is uniform over this frequency interval.

**Gas Tube Noise Generator.** Noise generators using a gas-discharge tube may be classified into two types: (a) sampling type noise generator, and (b) demodulator type noise generator. Both types employ a frequency shifting technique in order to obtain low-frequency noise from the wide band noise source. This technique produces a noise output of a magnitude which is comparable in magnitude to that of the input energy, rather than being a small fraction thereof. Owing to the limitations of the gas tube noise source at low frequencies, the frequency shifting technique also insures a uniform spectrum at ultra-low frequencies.

**Sampling Type Noise Generator.** A block diagram showing the basic principle of the sampling type noise generator is shown in Fig. 7 (Refs. 14, 47, 55). The wide band output from the gas tube is first passed through an amplifier to equalize the noise tube characteristics. The output of this amplifier is passed through an automatic gain control (AGC) regulating circuit to compensate for the low-frequency spectrum variations from the gas tube. The noise voltage from the AGC circuit is fed to a sample and hold circuit. The requirements for the sample and hold circuit are that the sampling occur in a linear manner during a very short portion of the sampling period. The sampling may be performed by a rotating commutator or an electronic gate. The holding or storing part of the circuit is usually a condenser.

The output from the sample and hold device will be a stepped voltage wave  $e(t)$ . Each step being a fixed duration  $T$  and equal to the duration of every other step (see Fig. 7). The amplitude probability density distribution will be the same as the amplitude distribution of the noise source, namely Gaussian. If the bandwidth of the primary noise source

is large when compared to the sampling frequency  $1/T$ , successive samples of the primary noise voltage may be considered to be uncorrelated. In this case the spectral density  $W(f)$  of the stepped voltage wave in Fig. 7 will be

$$(24) \quad W(f) = \sigma^2 T \left[ \frac{\sin \pi f T}{\pi f T} \right]^2,$$

where  $\sigma^2$  is the mean square value of the sampled noise and  $T$  is the sampling period. The spectral density is essentially uniform from zero frequency to some higher frequency determined by the sampling rate.

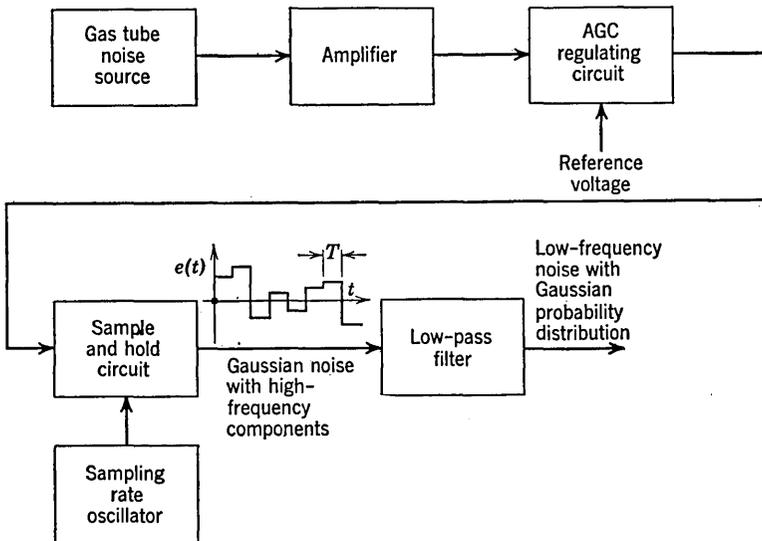


FIG. 7. Block diagram of sampling type noise generator.

**Demodulator Type Noise Generator.** A block diagram showing the basic principle of this type of low-frequency noise generator is shown in Fig. 8 (Refs. 21, 38, 39, 42, 48). A description of a widely used noise generator employing the principles used in Fig. 8 follows (Ref. 39):

The primary source of noise is a type 5727 or 2D21 thyratron connected in a diode configuration. The noise supplied to the regulating circuit extends from about 30 cps to 3 kc. This bandwidth is sufficiently wide to permit adequate averaging of the noise level and yet allows for a reasonably short time constant for the regulating action.

Figure 9 shows a block diagram of the AGC regulator circuit. The noise is passed through a variable gain pentode, amplified, and then half-wave rectified. The rectified noise is compared with a reference

## 26-16 DESIGN AND APPLICATION OF ANALOG COMPUTERS

voltage and the difference is averaged by the integrator circuit. If the average noise amplitude is greater than the reference voltage the gain of the variable gain pentode is reduced, and if the noise is less, the gain

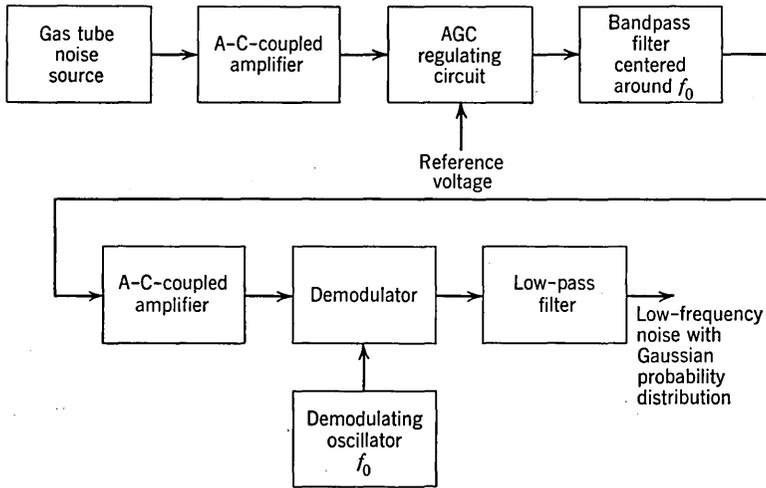


Fig. 8. Block diagram of demodulator type noise generator.

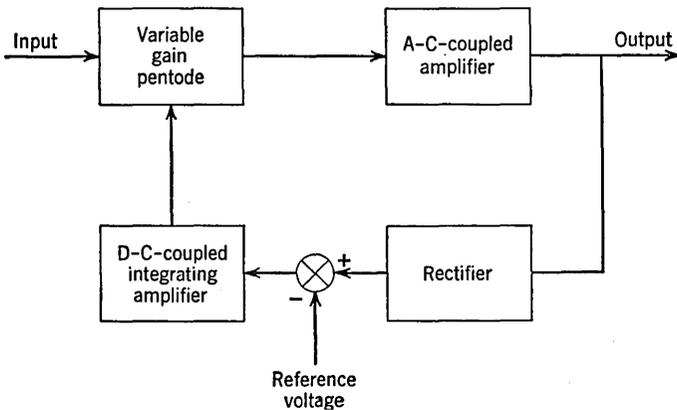


Fig. 9. Block diagram of regulator circuit.

is raised. Thus, the output of the regulator is noise whose average amplitude is constant and whose spectrum extends from 30 cps to 3 kc. A portion of this spectrum is selected by a bandpass filter whose transfer gain is given in Fig. 10a. The output of the filter is centered at 400 cps and has a bandwidth of approximately 100 cps.

After amplification the noise is demodulated by the use of a 400-cps electromechanical chopper. The chopper alternately multiplies the noise voltage by plus one and minus one. This multiplication results in frequency components consisting of sums and differences of the noise frequencies and the chopper frequency and its harmonics. Thus, there results a low-frequency noise and noise centered at 800 cps, 1200 cps, 1600 cps,

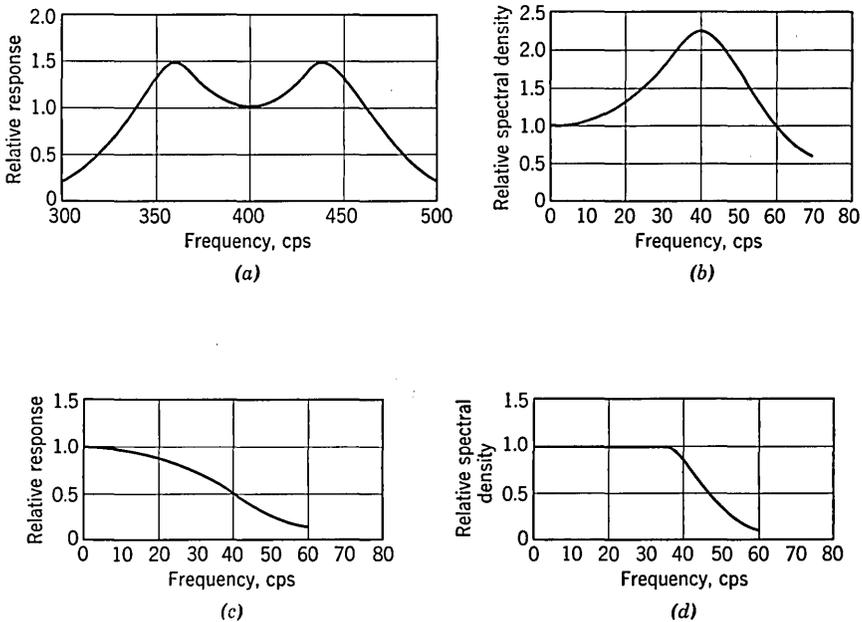


FIG. 10. Development of uniform spectrum: (a) gain of 400-cps filter; (b) low-frequency portion of noise spectrum before modification by  $RC$  filter; (c) gain of  $RC$  filter which follows chopper; (d) net output spectrum.

etc. The low-pass filter which follows the chopper effectively eliminates the high-frequency components. If there were no low-pass filter, the spectrum of the low-frequency portion of the output would be as shown in Fig. 10b. The low-pass  $RC$  filter, whose gain is shown in Fig. 10c, modifies the spectrum of Fig. 10b to give the output spectrum shown in Fig. 10d.

The filtering and demodulating process described above results in a noise generator with the following characteristics:

(a) An output spectrum that is within 0.1 db of uniform from zero to 35 cps.

(b) The spectral density has a value of approximately 4.0 volts<sup>2</sup>/cps.

## 26-18 DESIGN AND APPLICATION OF ANALOG COMPUTERS

(c) The output voltage has a probability density distribution which does not deviate from the Gaussian distribution by more than 1 per cent.

(d) The average output is less than 50 millivolts, with 95 per cent certainty.

Figure 11 shows a typical recorded sample of the noise generator output.

**Noise with Other Characteristics.** Beginning with a noise generator whose output spectrum is uniform over a sufficient range, it is a simple matter to alter the spectrum shape in a known manner by passing the noise through a filter composed of passive elements or feedback amplifiers. One may generate special types of nonstationary noise by utilizing time-varying filters. Noise having certain distributions other than Gaussian may be generated from Gaussian noise by combinations of linear and nonlinear filtering. For example, narrow band filtering and subsequent envelope detection of Gaussian noise results in an output having a Rayleigh distribution; phase-detecting narrow band Gaussian noise with reference to a fixed-phase carrier frequency and subsequent filtering will result in an output with a uniform probability density distribution (Ref. 40).

**Measuring Noise Generator Characteristics.** The most demanding aspect of low-frequency noise measurement is the length of time necessary to establish accurate estimates of the statistical properties of the noise. Bennett and Fulton (Ref. 38) have derived the measurement time required in order to determine the characteristics of low-frequency noise generators.

*Determination of Probability Distribution.* The easiest probability distribution to determine experimentally is the probability distribution function  $P(x)$  of eq. (3). All that is necessary is to measure the time the voltage from the noise generator is below a level  $x_1$  and to compare it to the total observation time. In other words,

$$(25) \quad P(x_1) = \frac{T_1}{T_T},$$

where  $T_1$  and  $T_T$  are the times below the level  $x_1$ , and the total observation time, respectively. By varying the comparison voltage  $x_1$ , the cumulative distribution can be obtained for all  $x$ .

Special cumulative distribution analyzers have been built (Ref. 41) and commercial instruments for this purpose are readily available (Ref. 57).

*Determination of Mean Value.* The mean value of a noise voltage may be found by integrating the voltage for a sufficiently long time. For a

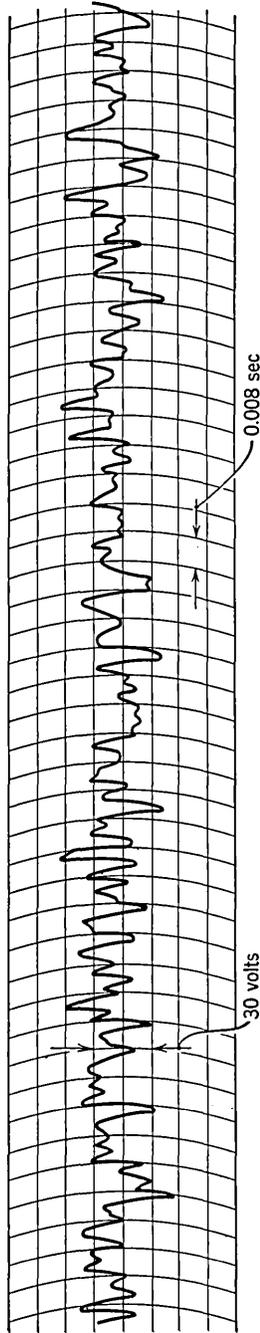


Fig. 11. Low-frequency noise generator output.

## 26-20 DESIGN AND APPLICATION OF ANALOG COMPUTERS

normally distributed noise voltage with a uniform spectral density of  $N$  volts<sup>2</sup>/cps and a theoretical mean value of  $\bar{x}$  volts, the measured mean voltage  $e$  will be within 95 per cent of the theoretical mean as dictated by

$$(26) \quad e = \bar{x} \pm \sqrt{2N/T} \text{ volts,}$$

where  $T$  is the integration time in seconds. The only way to improve the accuracy of the measurement of the mean is to integrate over a longer period of time.

*Determination of Mean Square Value.* The mean square value  $\overline{x^2}$  of a low-frequency noise voltage may be obtained on an analog computer by squaring the noise voltage and integrating the square over a period of time  $T$ . The length of time  $T$  required to determine, with 95 per cent certainty, the mean square value to within a percentage accuracy  $P$ , for normally distributed noise with zero mean having a spectral density given by eq. (20), is approximately

$$(27) \quad T = \frac{4 \times 10^4}{\pi f_0 P^2} \text{ seconds.}$$

*Determination of Spectral Density.* There is a variety of methods used for determining the spectral density of a noise voltage (Refs. 33, 34, 41). In the case of an analog computer noise generator whose spectrum is known to be essentially uniform in the regions of interest, the noise is usually passed through a selective filter composed of analog computer amplifiers, and the transmitted power is then measured. Repeating this process at a number of different frequencies will give a measure of the spectrum shape. The time of observation required for selective filters of this type is inversely proportional to the bandwidth of the filter. In order to obtain finer resolution of the spectral density, longer measurement times are required.

## REFERENCES

1. W. R. Bennett, Equipment for generating noise, *Electronics*, **29** (4), 134-135 (1956).
2. M. V. Mathews, A method for evaluating non-linear servos, *Trans. Am. Inst. Elec. Engrs.*, **74**, Pt. 2, 188 (May 1955).
3. W. R. Bennett, Methods of solving noise problems, *Proc. I.R.E.*, **44**, 609-638 (1956).
4. W. R. Bennett, Analogue computing applied to noise studies, *Proc. I.R.E.*, **41**, 1509-1513 (1953).
5. J. H. Laning, Jr., and R. H. Battin, On an application of the use of analogue computers to methods of statistical analysis, *Project Cyclone Symposium II*, Reeves Instrument Corp., New York, 1952.

6. J. H. Laning, Jr., and R. H. Battin, *Random Processes in Automatic Control*, McGraw-Hill, New York, 1956.

7. R. R. Bennett and A. S. Fulton, Automatic REAC operation for statistical studies, *Project Cyclone Symposium II on Simulation and Computing Techniques*, Pt. 2, p. 129. Reeves Instrument Corp., New York, 1956.

8. R. R. Bennett, R. R. Favreau, and I. Pfeffer, The adjoint computing method applied to guided missile system design, *Project Typhoon Symposium III on Simulation and Computing Techniques*, Univ. of Pennsylvania, Philadelphia, Pa., October 1953, Confidential.

9. R. R. Bennett and W. E. Mathews, Analytical determination of miss distance for linear homing navigation systems, *Secret Technical Memorandum No. 260*, Hughes Aircraft Company, Los Angeles, Calif., March 31, 1952.

10. Columbia University, Beam splitting and search radar simulation, Confidential Rept. F/A-VIII, ERL Columbia University, New York, November 30, 1954.

11. R. H. Delano and I. Pfeffer, The effect of AGC on radar tracking noise, *Proc. I.R.E.*, **44**, 801-810 (1956).

12. W. W. Seifert, *A Study of Noise in Missile Control Systems*, Confidential Sc. D. Thesis, Department of Electrical Engineering, Massachusetts Institute of Technology, Cambridge, Mass., 1951.

13. J. A. Aseltine, *The Use of Electronic Analog Computers in the Solution of Certain Radar Noise Problems*, Presented at the I.R.E. West Coast Convention, San Francisco, Calif., August 1955.

14. D. F. Winter *et al.*, Radar tracking data and filter studies, Confidential NAVORD Rept. 1414, Washington University School of Engineering, Division of Sponsored Research, St. Louis, June 30, 1952.

15. H. Ehlers and E. Vogel, Simulation of noise in missile homing problems, *Project Cyclone Symposium I on REAC Techniques*, Reeves Instrument Corp., New York, March 1951.

16. Goodyear Aircraft Corporation, Application of Geda to human dynamics studies, *Rept. GER-6608*, Goodyear Aircraft Corp., Akron, O., March 4, 1955.

17. N. W. Trembath, Wind gust simulation, *Internal Memo GM 453-268*, Ramo-Wooldridge Corp., Los Angeles, Calif., May 9, 1957.

18. J. R. Clark, A. B. Fontaine, and C. E. Warren, The generation of a continuous random signal for use in human tracking studies, *Research Bull. 53-40*, Human Resources Research Center, Air Research and Development Command, Lacklin Air Force Base, Tex., October 1953.

19. C. E. Warren, P. M. Pitts, and J. R. Clark, An electronic apparatus for the study of the human operator in a one-dimensional, closed-loop continuous pursuit task, *Trans. Am. Inst. Elec. Engrs.*, **71**, 19-23 (1952).

20. J. A. Aseltine and R. R. Favreau, Weighting functions for time varying feedback systems, *Proc. I.R.E.*, **42**, 1559-1564 (1954).

21. R. C. Booton, Jr., Nonlinear control systems with statistical inputs, *Rept. No. 61*, Dynamic Analysis and Control Laboratory, Massachusetts Institute of Technology, Cambridge, Mass., March 1, 1952.

22. R. C. Booton, Jr., M. V. Mathews, and W. W. Seifert, Nonlinear servomechanisms with random inputs, *Rept. No. 70*, Dynamic Analysis and Control Laboratory, Massachusetts Institute of Technology, Cambridge, Mass., August 1953.

23. T. B. Garber, *RMS Errors in Systems with Constant and Time Varying Coefficients*, M.S. Thesis, Department of Aeronautical Engineering, Massachusetts Institute of Technology, Cambridge, Mass., 1952.

## 26-22 DESIGN AND APPLICATION OF ANALOG COMPUTERS

24. Goodyear Aircraft Corporation, Geda simulation study of a relay servo-mechanism, *Rept. GER-4799*, Goodyear Aircraft Corp., Akron, O., May 9, 1952.
25. E. C. Hall, Analog simulation for noise analysis in control systems, *Rept. R-74*, Instrumentation Laboratory, Massachusetts Institute of Technology, Cambridge, Mass., May 1954.
26. Goodyear Aircraft Corporation, Geda simulation study of a carrier-type instrument servomechanism, *Rept. GER-5779*, Goodyear Aircraft Corporation, Akron, O., April 21, 1954.
27. A. Rosenbloom, *Analysis of Randomly Time-Varying Linear Systems*, Ph.D. Dissertation, University of California at Los Angeles, Calif., July 1954.
28. H. Press and J. W. Tukey, Power spectral methods of analysis and their application to problems in airplane dynamics, *AGARD Flight Test Manual*, Pt. IVC, pp. 1-41, June 1956, Bell Telephone System, Monograph 2606, New York.
29. Goodyear Aircraft Corporation, Two applications of Geda computers to statistical problems of operations research, *Rept. GER-6729*, Goodyear Aircraft Corp., Akron, O., June 4, 1955.
30. O. J. M. Smith, Economic analogs, *Proc. I.R.E.*, **41**, 1514-1519 (1953).
31. D. L. Trautman *et al.*, Analysis and simulation of vehicular traffic flow, *Research Rept. No. 20*, Institute of Transportation and Traffic Engineering, University of California, Los Angeles, Calif., December 1954.
32. R. A. Johnson and D. Middleton, *Measurement of Correlation Functions of Modulated Carriers and Noise Following a Non-Linear Device*, Paper presented at Symposium on Applications of Communication Theory, Institute of Electrical Engineers, London, September 1952.
33. F. B. Smith, Analog equipment for processing randomly fluctuating data, *Aeronaut. Eng. Rev.*, **14**, 113-119 (1955).
34. H. W. Smith *et al.*, A power spectrum computer, *Rept. No. 86*, Electrical Engineering Research Laboratories, University of California, Berkeley, Calif., November 15, 1956.
35. R. R. Favreau, H. Low, and I. Pfeffer, Evaluation of complex statistical functions by an analog computer, *I.R.E. Convention Record*, **4**, Pt. 4, 31, March 1956.
36. A. Rosenbloom, Analysis of an optimum filter with derivative constraints, *Memo GM 41-5*, Ramo-Wooldridge Corp., Los Angeles, Calif., May 8, 1956.
37. G. W. Anderson and J. E. Murrin, *Autocorrelator for Radioactive Sample Noise Generator*, Paper given at the I.R.E. WESCON Conference, August, 1956, Los Angeles, Calif.
38. R. R. Bennett and A. S. Fulton, The generation and measurement of low-frequency random noise, *J. Appl. Phys.*, **22**, 1187-1191 (1951).
39. R. R. Bennett, D. E. Beecher, and H. Low, Electronically stabilized noise generation, *Electronics*, **27**, 163-165 (July 1954).
40. R. Bernstein, H. Bickel, and E. Brookner, A generator of uniformly distributed random noise, *I.R.E. Convention Record*, Pt. 10, 97, 1954.
41. H. Dern *et al.*, Noise studies, *Tech. Rept. T-1/122*, Electronic Research Laboratories, Columbia University, New York, December 31, 1955.
42. R. R. Favreau, Noise generators, *Rept. PCC 19*, Electronic Associates Princeton Computation Center, Princeton, N. J., 1955.
43. J. W. Follin, Jr., G. F. Emch, and F. H. Walters, Modifications and additions to the REAC, *Project Cyclone Symposium II on Simulation and Computing Techniques*, p. 173, Reeves Instrument Corp., New York, April-May 1952.

44. H. H. Goode, W. A. Wheatley, and G. G. den Broeder, The generation of a vector with an  $N$ -dimensional normal distribution by means of analog equipment, *Project Cyclone Symposium I on REAC Techniques*, p. 75, Reeves Instrument Corp., New York, March 1951.
45. Goodyear Aircraft Corporation, Random noise generator for simulation studies, *Rept. GER-6436*, Goodyear Aircraft Corp., Akron, O., December 13, 1954.
46. C. E. Hendrix and R. B. Purcell, A low-frequency random noise generator, *T. M. 1788*, U. S. Naval Ordnance Test Station, Inyokern, China Lake, Calif., February 1954.
47. M. A. Karpeles, Noise Compressor, U. S. Patent 2,649,386, May 19, 1953.
48. M. V. Mathews, Noise generator evaluation study, *Rept. DIC 6387, M-4.40-2*, Dynamic Analysis and Control Laboratory, Massachusetts Institute of Technology, Cambridge, Mass., December 19, 1951.
49. M. V. Mathews, White noise tape specifications, *Rept. DIC 6978, M-4.41-1*, Dynamic Analysis and Control Laboratory, Massachusetts Institute of Technology, Cambridge, Mass., April 1, 1953.
50. H. Meister, A thermal noise generator for low-frequency test, *Tech. Mitt. Schweiz. Telegr.-Teleph. Verw.*, **28**, 320-324 (1950).
51. K. A. Otto, *Generation of Low-Frequency Random Noise*, Master's Thesis, U.S.A.F. Institute of Technology, Wright-Patterson Air Force Base, O., March 1955.
52. S. Sherman and E. Lakatos, Noise generation for analog simulation, *Project Typhoon, Symposium III on Simulation and Computer Techniques*, Pt. I, p. 155, University of Pennsylvania, Philadelphia, Pa., October 1953.
53. L. B. Wadel, C. C. Calvin, and J. W. Atkins, Jr., Repeatable generation of noise with a masked cathode-ray tube, *Proc. Natl. Simulation Conf.*, p. 34.1, Dallas, Tex., January 1956.
54. C. A. A. Wass, *Introduction to Electronic Analogue Computers*, p. 177, McGraw-Hill, New York, 1956.
55. D. F. Winter, A Gaussian noise generator for frequencies down to 0.001 cycles per second, *I.R.E. Convention Record*, Pt. 4, p. 23, 1954.
56. H. Friedman, Geiger counter tubes, *Proc. I.R.E.*, **37**, 791 (1949).
57. Berkeley Division of Beckman Instruments, *Instruction Manual for Universal Counter and Timer, Model 7360-15*, Richmond, Calif., March 6, 1957.
58. T. B. Van Horne, An analog method for the solution of probability of hit and related statistical problems, *I.R.E. Trans. Electronic Computers*, **6**, 170 (September 1957).
59. M. R. Bates, D. H. Bock, and F. D. Powell, Analog computer applications in predictor design, *I.R.E. Trans. Electronic Computers*, **6**, 143 (September 1957).



## Mechanical Computer Elements

*Walter J. Karplus*

1. Introduction	27-01
2. Basic Operations	27-02
3. Function Generation	27-05
4. Solution of Equations	27-09
5. Scale Factors	27-14
References	27-15

### I. INTRODUCTION

Historically, mechanical computing elements represent the oldest form of automatic computation. In this method the dependent and independent variables of the problem are represented by either linear or angular displacements. Elements employing linear displacements or combinations of angular and linear displacements in the computing process include such well-known devices as linkage mechanisms, contour cams and gears, and levers. Computing elements of this type, described in some detail by Svoboda (Ref. 1) and Soroka (Ref. 2), are used almost exclusively in special purpose applications, while virtually all general purpose computation by mechanical means is carried out with angular displacement mechanisms. This discussion will, therefore, be limited to the later type of elements. *Note.* Most of the examples used in this treatment were contributed by Professor W. C. Hurty of the University of California in Los Angeles. A brief table of symbols is given in Chap. 21, Sect. 6.

## 27-02 DESIGN AND APPLICATION OF ANALOG COMPUTERS

**Comparison of Mechanical and Electronic Elements.** Mechanical computing elements are precision-machined devices employing shafts, gears, and disks. The accuracy of such elements is limited by the following:

Machining errors, resulting from machine shop tolerances.

Slippage errors, where frictional contact forces are employed to transmit displacements.

Theoretical errors, resulting from approximations made in mechanizing the mathematical operation.

Since all these errors may be minimized by careful design and fabrication, accuracies of the order of 0.01–0.1% are readily obtained in mechanical computers. Compared with electronic computing elements (see Chapters 22 and 23), mechanical general purpose computing elements have the following advantages and disadvantages:

<i>Advantages</i>	<i>Disadvantages</i>
Greater accuracy	Require more space
More rugged	Greater weight
Simplicity of design	Generally more expensive
More dependable	More difficult to “program”
Need not work in real time	Less flexible in application
Solution may be interrupted and continued as desired	Require more time for solution
Not greatly influenced by power line fluctuations	

## 2. BASIC OPERATIONS

### **Multiplication by a Constant**

This is accomplished simply by stepping up or stepping down as the case may be, the angular rotation of the shaft by means of a pair of gears or a gear train. Since the direction of rotation is identified with the algebraic sign of the variable, coupling two shafts with identical gears would represent a multiplication by minus one, or a simple change in sign.

### **Addition and Subtraction**

A differential gearing arrangement is used to obtain an angular displacement which is the sum of two separate angular displacements. Familiar differential arrangements include the bevel gear differential and the spur gear differential. In such devices the rotation of the output shaft is equal to one-half the algebraic sum of the rotation of the two

input shafts. Subtraction is accomplished by merely employing a one-to-one gear coupling to change the sign of one of the variables prior to applying it to the adder.

### Integration

A mechanical integrator relates three quantities,  $u$ ,  $v$ , and  $w$ , in the following way

$$w = \int u \, dv,$$

where all three functions are generally continuous and have continuous derivatives. Discontinuities in  $u$  and  $v$  are permissible provided that they are finite in number and do not occur at the same instant in the computing process.

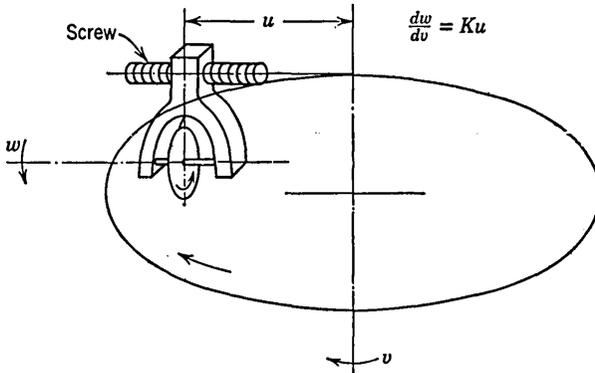


FIG. 1. Kelvin disk integrator.

**Kelvin Disk Integrator.** The Kelvin disk integrator, shown in Fig. 1, is the most commonly used general purpose integrator. The angular displacement of the turntable is proportional to  $v$ , that of the integrator wheel to  $w$ , and that of the lead screw to  $u$ . Accordingly, the distance from the center of the turntable to the plane of the integrator wheel is proportional to  $u$ . If it is assumed that there is no slippage of the integrator wheel on the surface of the turntable, a small displacement of the turntable,  $\Delta v$ , will produce a small rotation of the wheel  $\Delta w$ , according to

$$\Delta w = Ku \, \Delta v,$$

where  $K$  is a constant. This may be generalized as

$$w = \int_{v_1}^{v_2} u \, dv.$$

**Initial Values.** It is seen that in so far as initial values are concerned, the initial setting of  $u$  is the only one which has significance in determining the subsequent output of the integrator. The lead screw of the integrator must, therefore, be set to correspond to the appropriate initial value before the commencement of each computation.

**Torque Amplifiers.** The accuracy of an integrator is limited by the extent to which the integrator wheel is capable of following the rotation of the turntable. Since this drive is accomplished by frictional forces, care must be taken to assure that there is no slippage at the point of contact. The mechanical load or torque upon the integrator wheel must, therefore, be minimized. The first successful torque amplifiers for this purpose were developed in 1925 by Bush and consisted of a relay servomechanism activated by means of mercury drop contacts between two disks. Subsequent torque amplifiers having gains in excess of 10,000 are described by Soroka (Ref. 2) and include the following:

1. *Two-stage capstan type*, a two-stage amplifier consisting of two stepped drums driven in opposite directions by a powerful electric motor.
2. *Polarized light servo*, in which a beam of light is passed through a polaroid disk mounted on the integrator wheel as well as through a polaroid disk mounted on a follow-up disk, and a servomechanism is employed to keep the light falling on a photocell at a constant value.
3. *Capacitance type*, in which the integrating wheel essentially acts as the plate of a condenser and the variations in capacity resulting from the rotation of the wheel are sensed and amplified.

### Input and Output Tables

Units, called input and output tables, are devices from which plotted functions may be applied as angular displacements to the system, or to which functions present in the system as angular displacements may be applied and translated into plotted curves. In either case, for Cartesian coordinates, they consist of a carriage carrying either a sighting target or a pen that is made to move in a horizontal as well as a vertical direction by means of two screws, which may be rotated by connecting them to appropriate shafts in the computing system. If the argument of the function is fed into the input table and is made to control one of the coordinates, say the abscissa, then the operator can, by turning a crank, adjust the other screw, so that the sighting target will always follow the plotted curve. The crank rotation, then, will be proportional to the function and may be fed back into the system. If both screws are driven by machine variables and a pen is attached to the carriage, the graphical relationship between the two machine variables will be plotted automatically by the pen. Input and output tables may readily be adapted for use with polar coordinates by making one screw rotate

the table while the other screw is used to adjust the radial position of the carriage.

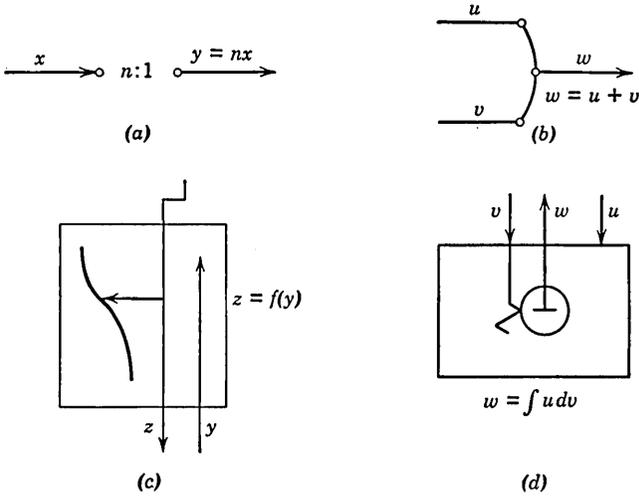


FIG. 2. Schematic representation of basic operations: (a) gear train, multiplies by constant factor  $n$ ,  $y = nx$ ; (b) adder; (c) function table; (d) integrator.

**Schematic Representation of Basic Operations**

Diagrams of the basic operations of multiplication by a constant, addition, function unit (input and output tables), and integration are shown in Fig. 2.

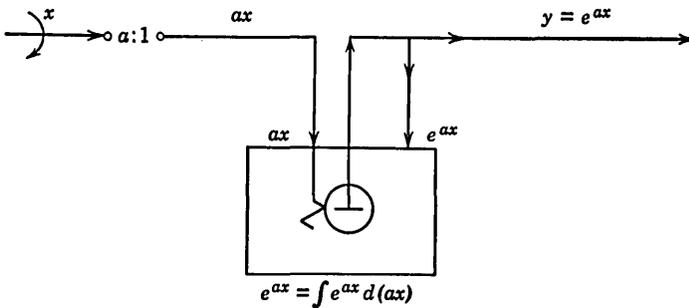


FIG. 3. Generation of exponential,  $e^{ax}$ . Governing equation:  $dy/dx = ae^{ax}$ .

**3. FUNCTION GENERATION**

A wide variety of algebraic and transcendental functions may be generated mechanically by suitable combination and utilization of the basic operations of constant multiplication, addition, and integration. Sche-

## 27-06 DESIGN AND APPLICATION OF ANALOG COMPUTERS

matic diagrams of mechanical circuits suitable for the generation of some of the more important analytic functions, without the aid of input tables, are shown in Figs. 3 through 9, together with the basic equation from which these generators were constructed. The governing equations are given in Table 1. The trick in each case is to recognize or derive a

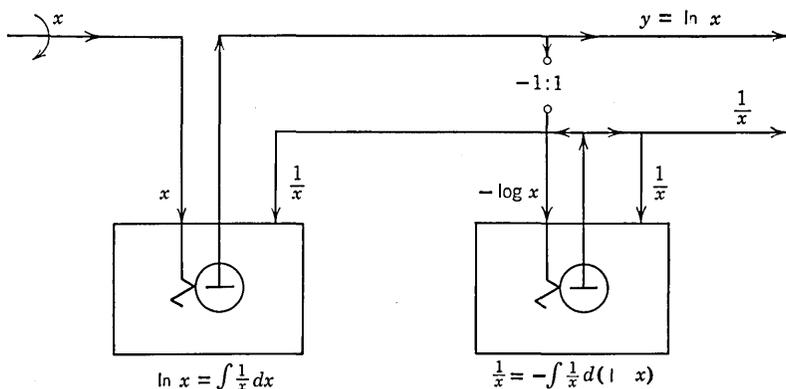


FIG. 4. Generation of logarithm and reciprocal. Governing equation:

$$\frac{d^2y}{dx^2} + \left(\frac{dy}{dx}\right)^2 = 0.$$

functional relationship which permits the realization of the desired function with the basic elements available.

TABLE 1. ANALYTIC FUNCTIONS THAT MAY BE GENERATED BY MECHANICAL CIRCUITS

Function	Governing Equation	Solution
Exponential, $e^{ax}$	$\frac{dy}{dx} = ae^{ax}$	Fig. 3
Reciprocal, $\frac{1}{x}$	$\frac{dy}{dx} = \frac{-\log x}{x}$	Fig. 4
Logarithm, $\log x$	$\frac{d^2y}{dx^2} = -\left(\frac{dy}{dx}\right)^2$	Fig. 4
Square, $x^2$	$\frac{dy}{dx} = 2x$	Fig. 5
Product, $xy$	$xy = \int y dx + \int x dy$	Fig. 6
Sine $\omega x$ Cosine $\omega x$	$\frac{d^2y}{dx^2} + \omega^2 y = 0$	Fig. 7
Tangent $x$	$\frac{dy}{dx} = 1 + y^2$	Fig. 8
Power, $x^n$	$\frac{dy}{d(n \log x)} = y$	Fig. 9

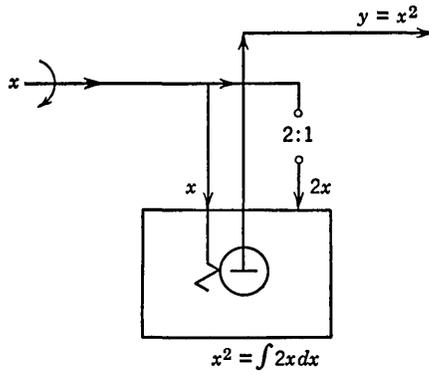


FIG. 5. Generation of square. Governing equation:  $dy/dx = 2x$ .

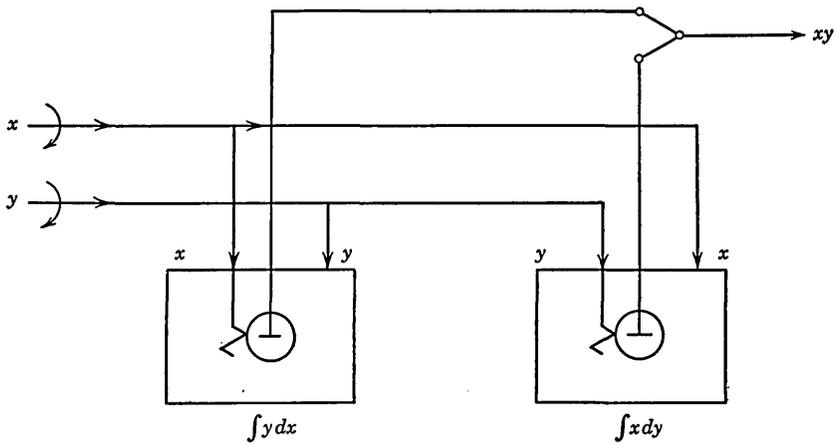


FIG. 6. Generation of product. Governing equation:  $xy = \int y dx + \int x dy$ .

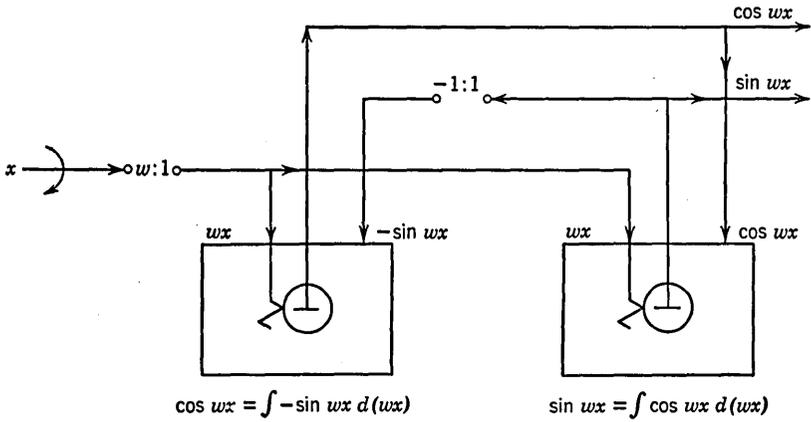


FIG. 7. Generation of sine  $w x$  and cosine  $w x$ . Governing equation:  
 $(d^2y/dx^2) + \omega^2 y = 0$ .

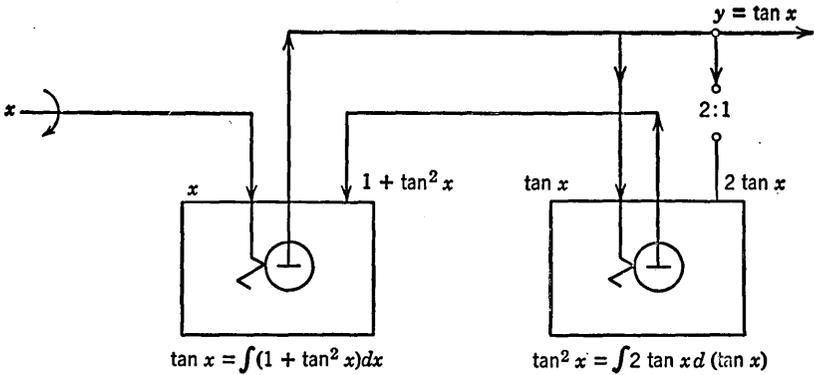


FIG. 8. Generation of tangent  $x$ . Governing equation:  $dy/dx = 1 + y^2$ .

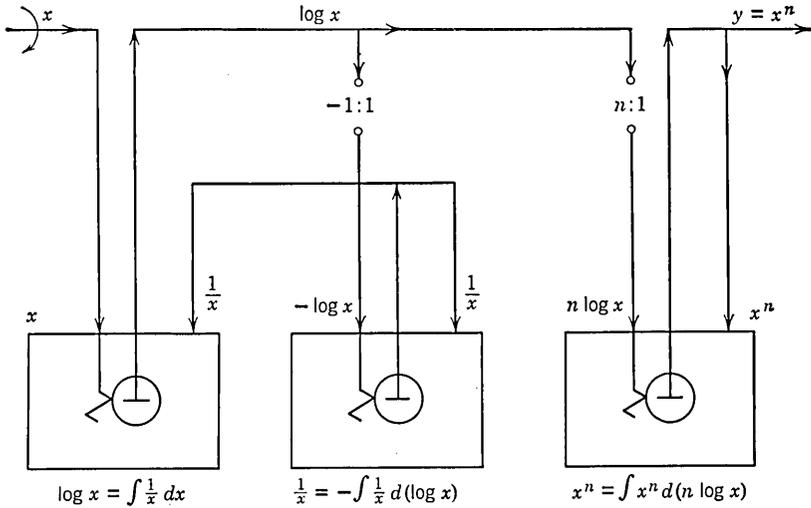


Fig. 9. Variable raised to power  $n$ . Governing equation:  $dy/d(n \log x) = y$ .

EXAMPLE. Suppose it is desired to generate  $y = \tan x$ . Then

$$dy/dx = \sec^2 x = 1 + \tan^2 x = 1 + y^2.$$

The generation of the desired function is then equivalent to the solution of the equation

$$dy/dx = 1 + y^2,$$

so that after integration

$$y = \int (1 + y^2) dx.$$

This in terms of  $x$  is

$$\tan x = \int (1 + \tan^2 x) dx.$$

If it is recognized that

$$\tan^2 x = \int 2 \tan x d(\tan x),$$

the appropriate diagram is readily constructed as shown in Fig. 8.

#### 4. SOLUTION OF EQUATIONS

**Ordinary Second Order Differential Equation.** Consider the equation

$$(1) \quad \frac{d^2y}{dt^2} + 2r \frac{dy}{dt} + k^2y = 0.$$

## 27-10 DESIGN AND APPLICATION OF ANALOG COMPUTERS

Rearrange to express the highest order term in terms of lower order.

$$(2) \quad \frac{d^2y}{dt^2} = -2r \frac{dy}{dt} - k^2y.$$

Integrate

$$(3) \quad \frac{dy}{dt} = -2ry - k^2 \int y dt.$$

A second integration gives

$$(4) \quad y = \int \frac{dy}{dt} dt.$$

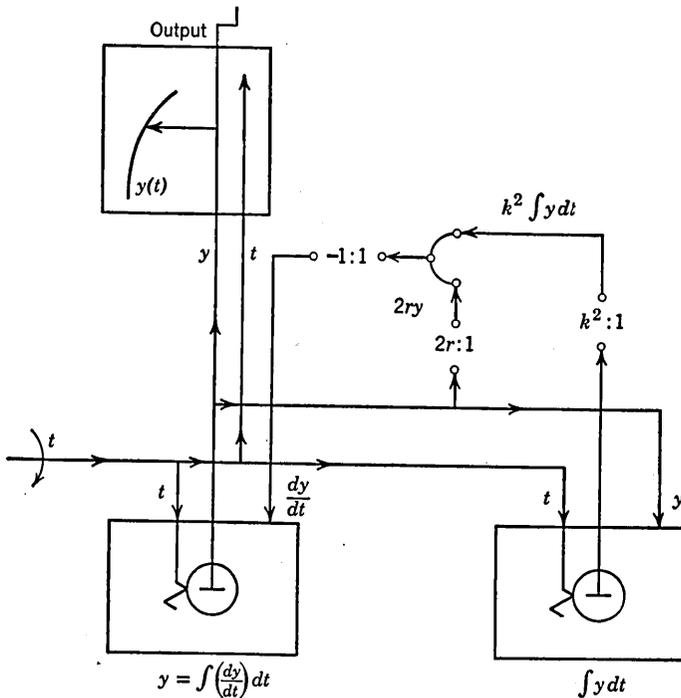


FIG. 10. Schematic diagram for solution of

$$\frac{d^2y}{dt^2} + 2r \frac{dy}{dt} + k^2y = 0.$$

Two integrators are required, the first to give  $\int y dt$  (eq. 3) and the second to give  $\int (dy/dt) dt$  (eq. 4).  $y$ , which is generated by the second integration, is fed back as an input to the first and also to the other term in eq. (3).

The schematic arrangement for the mechanization of eqs. (3) and (4) is shown in Fig. 10.

The constant coefficients  $2r$  and  $k^2$  are represented by gear trains with corresponding ratios. An adder is required to give the sum of the two terms on the right side of eq. (3). The initial values of  $y$  and  $dy/dt$  must be known to proceed with the solution. Note that both these quantities are instrumental as settings of the leadscrews of integrators.

**Nonlinear Differential Equation.** Consider the equation

$$(5) \quad \frac{d^2y}{dx^2} + y \frac{dy}{dx} + f(y) = \cos x.$$

Rearrange to express the highest order term in terms of lower order.

$$(6) \quad \frac{d^2y}{dx^2} = \cos x - y \frac{dy}{dx} - f(y).$$

Integrate

$$(7) \quad \frac{dy}{dx} = \sin x - \int y dy - \int f(y) dx.$$

A second integration gives

$$(8) \quad y = \int \frac{dy}{dx} dx.$$

It is also necessary to generate  $\cos x$ .

$$(9) \quad \sin x = \int \cos x dx; \quad \cos x = \int -\sin x dx.$$

Five integrators are required, two in eq. (7), one in eq. (8), two in eq. (9). Two adders eq. (7) and one input table, to supply the arbitrary function  $f(y)$ , are also necessary.

The schematic arrangement is shown in Fig. 11. The initial values of  $x$ ,  $y$ , and  $dy/dx$  are needed to proceed with the solution.

Note that one of the integrations is with respect to a dependent variable. When this occurs it usually means that the turntable (disk) of one integrator is being driven by the output of another, a condition referred to as "cascading."

**Simultaneous Differential Equations.** Consider the two-loop network in Fig. 12, with voltage  $e$  any arbitrary function of time and with the nonlinear inductance  $L_2$  a function of the current  $i_2$ .

The pertinent differential equations are:

$$(10) \quad L_1 \frac{di_1}{dt} + (R_1 + R_2)i_1 - R_2i_2 = e(t),$$

$$(11) \quad L_2(i_2) \frac{di_2}{dt} + R_2i_2 + \frac{1}{C} \int i_2 dt - R_2i_1 = 0.$$

27-12 DESIGN AND APPLICATION OF ANALOG COMPUTERS

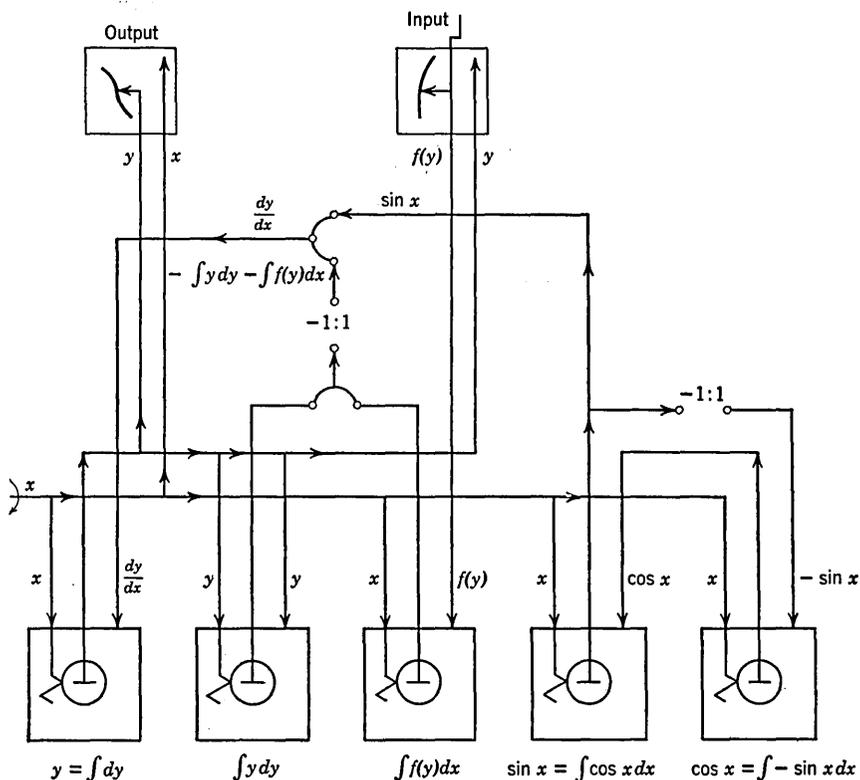


FIG. 11. Schematic diagram for solution of

$$\frac{d^2y}{dx^2} + y \frac{dy}{dx} + f(y) = \cos x.$$

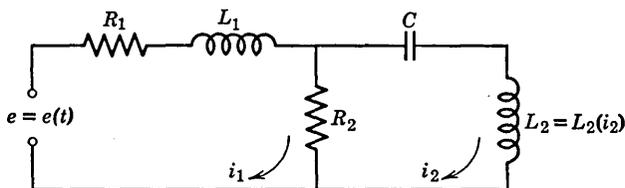


FIG. 12. Two-loop network. Equations:

$$L_1 \frac{di_1}{dt} + (R_1 + R_2)i_1 - R_2i_2 = e(t),$$

$$L_2(i_2) \frac{di_2}{dt} + R_2i_2 + \frac{1}{C} \int i_2 dt - R_2i_1 = 0.$$

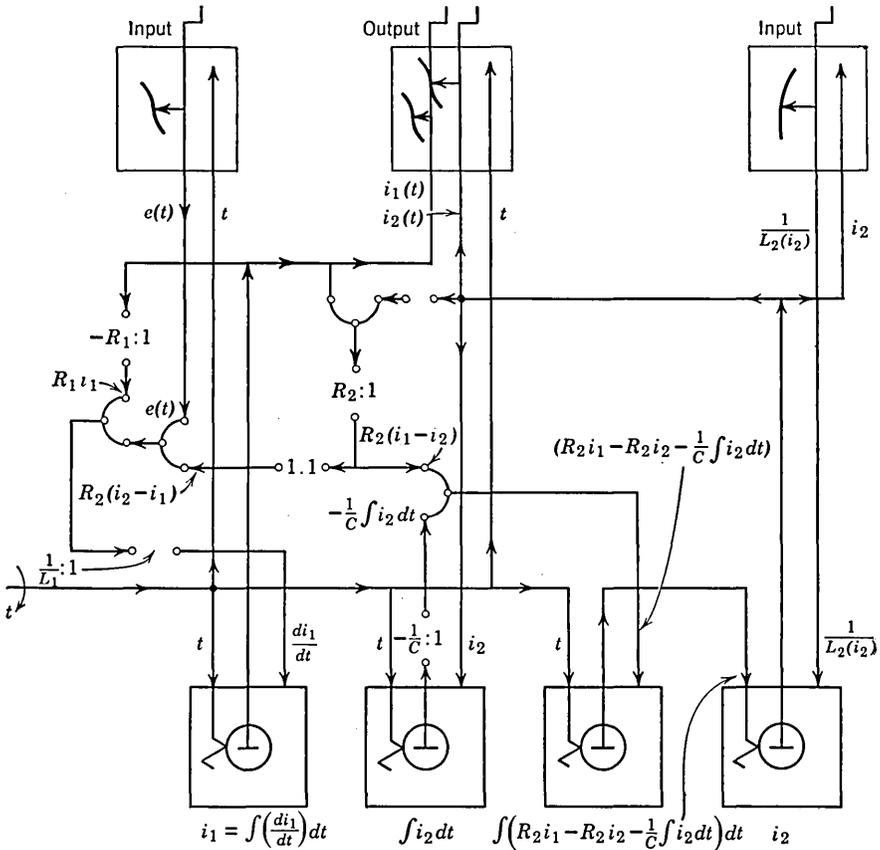


FIG. 13. Schematic diagram for two-loop network problem of Fig. 12.

Rearrange each equation to express the highest order term as a function of lower order terms.

$$(12) \quad \frac{di_1}{dt} = \frac{1}{L_1} \left[ e(t) + R_2 i_2 - (R_1 + R_2) i_1 \right],$$

$$(13) \quad \frac{di_2}{dt} = \frac{1}{L_2(i_2)} \left[ R_2 i_1 - R_2 i_2 - \frac{1}{C} \int i_2 dt \right].$$

Equation (12) may be integrated to give

$$(14) \quad i_1 = \int \frac{di_1}{dt} dt.$$

But in eq. (13), the expression for  $di_2/dt$  is the product of two variables.

## 27-14 DESIGN AND APPLICATION OF ANALOG COMPUTERS

This may be treated as the integration of a product as follows

$$(15) \quad i_2 = \int \frac{1}{L_2(i_2)} d \left[ \int \left( R_2 i_1 - R_2 i_2 - \frac{1}{C} \int i_2 dt \right) dt \right].$$

A total of four integrators, four adders, and two input tables, one for the function  $e(t)$  and the other for the function  $1/L_2(i_2)$ , is required. Five gear trains are needed for constant coefficients. The initial values of  $di_1/dt$ ,  $i_1$ ,  $i_2$ , and  $\int i_2 dt$  must be supplied as well. The schematic arrangement is shown in Fig. 13.

### 5. SCALE FACTORS

**Definition.** In mechanical computation the magnitude of each variable of the problem is simulated by the angular displacement of a shaft. Some number of revolutions of that shaft is therefore proportional to one unit of magnitude of the corresponding variable. The constant relating the number of shaft revolutions to one unit of the variable represented by this shaft is called the *scale factor*. Every shaft in a mechanical computing system has a scale factor associated with it. Considerable care must be exercised in the selection of these scale factors to assure proper operation of the computer system.

**Consideration in Choice of Scale Factors.** Before an intelligent assignment of scale factors can be made, the maximum and sometimes the minimum values of all variables and their derivatives must be approximated. Frequently the initial approximation of these values may be excessively erroneous and may result in poor performance of the computer system. The scale factors must then be readjusted by trial and error.

*Limitations on Dependent Variable Scale Factors.*

1. The excursion of the integrator wheel is limited by the number of threads on the lead screw.
2. The excursion of the carriage of input and output tables is limited by the number of threads on its screw.
3. The excursion of the shaft as a result of the computing process should be large compared to "noise" deflections due to vibration, backlash, etc.

*Limitation on Independent (Time) Variable Scale Factor.*

1. The maximum speed of shafts is limited by: characteristics of bearings, sensitivity of follow-up servos in torque amplifiers, capability of driving motors, characteristics of output tables (pens, paper, etc.), capabilities of human operators of input tables.
2. The time scale factor should not be so small that an inconveniently

long time is consumed in obtaining a solution over a specified region of interest.

The time scale factor should be selected so that a plot of the entire region of interest is displayed on the output table to a convenient scale.

*Other Considerations.*

1. The scale factors at the two inputs of an adder must be equal.
2. The output scale factor of an adder is determined by the input scale factors.
3. The scale factor at the output of an integrator is equal to the product of the two input scale factors divided by a constant (usually 16 or 32).

### REFERENCES

1. A. Svoboda, *Computing Mechanisms and Linkages*, MIT Radiation Laboratory Series, **27**, McGraw-Hill, New York, 1948.
2. W. W. Soroka, *Analog Methods in Computation and Simulation*, McGraw-Hill, New York, 1954.



## Digital Techniques in Analog Computation

*Cornelius T. Leondes*

1. Introduction	28-01
2. Digital Differential Analyzer	28-02
3. Digital Operational Computers	28-11
4. Auxiliary Digital Computing Techniques	28-15
5. Auxiliary Digital Control Techniques	28-17
References	28-18

### I. INTRODUCTION

Digital techniques may be incorporated in analog computers in certain instances with resultant extensions in their capabilities. There are several aspects to this approach.

*a.* In the basic analog computer operational units, discrete variable or digital number representation may be employed rather than the usual continuous variable representation.

*b.* Data entering an analog computer may be in digital number form. Such instances occur, for example, in analog computers employed in traffic control systems. Modified analog operational units such as multipliers, may accept a discrete variable for one input and a continuous variable for the other to produce the product of the two variables in analog form. This provides equipment economies and better overall accuracy when compared with a procedure which employs a digital to analog converter and then supplies the two variables to a conventional analog multiplier.

*c.* The generation of arbitrary functions of machine variables is often

## 28-02 DESIGN AND APPLICATION OF ANALOG COMPUTERS

a troublesome process in analog computers, particularly when functions of several variables are to be generated. In this instance digital techniques can often be introduced into the analog computer to advantage.

*d.* In addition, digital techniques can often be advantageously introduced into analog computers for the purpose of automatically controlling large analog computing installations. Complete problem setups can be stored on punched tapes or cards and patch panels so that by feeding the punched tapes or cards into the digital tape or card reader complete problems can be automatically set up on the analog computer. Time can be saved, errors can be more readily detected, more reliable operation can be achieved, and more efficient use can be made of the analog computer.

**Definitions.** The mechanization of analog computer operational units can be based upon the employment of discrete variable representation rather than the usual continuous variable representation. The discrete representation employed can be in the form of digital numbers or variable pulse repetition rates. When digital number representation is employed the associated computer mechanization will be referred to as a *digital differential analyzer* or DDA. When variable pulse repetition rate representation is employed the associated computer mechanization will be referred to as a *digital operational computer*.

### 2. DIGITAL DIFFERENTIAL ANALYZER

**Method of Integration.** A mechanical integrator establishes the relationship of eq. (1) between the  $x$ ,  $y$ , and  $z$  shafts (see Chap. 27).

$$(1) \quad dz = ky \, dx.$$

These shaft positions represent directly and on a continuous basis the variables  $x$ ,  $y$ , and  $z$  and it follows that

$$(2) \quad z = k \int y \, dx.$$

If one considers finite increments and a discrete variable representation for the variables involved then, corresponding to eq. (1) is the equation

$$(3) \quad \Delta z = ky \, \Delta x,$$

and if the increments are taken to be very small, the integral of eq. (2) may be approximated to a very high degree by a device which mechanizes the equation

$$(4) \quad z_n = \sum_{i=0}^n y_i \, \Delta x_i.$$

This is the approximation to the integral when rectangular integration

as implied by eq. (3) is employed. A better approximation can be provided by more sophisticated integration techniques such as the trapezoidal integration equation

$$(5) \quad \Delta z = (y + \Delta y/2) \Delta x.$$

Basically, a DDA actuates eq. (4) for variables represented in a discrete manner, and solves differential equations in a manner analogous to that outlined for the mechanical differential analyzer.

**Operational Integrators in a DDA.** Each operational integrator unit in a DDA accepts as inputs  $\Delta x$  and  $\Delta y$ , and provides a  $\Delta z$  output. The integrator must contain at the start of the problem the initial value of  $y$ , and this value of  $y$  is continually kept up to date by the accumulation of all subsequent  $\Delta y$ 's supplied to the integrator. The  $\Delta z$  outputs may serve as inputs to other integrators in the computer and are also accumulated to provide a continual and up-to-date indication of the value for the variable  $z$ . All the numbers  $x$ ,  $\Delta x$ ,  $y$ ,  $\Delta y$ ,  $z$ , and  $\Delta z$  are represented digitally in the computer on a binary, decimal, or other basis depending on such considerations as ease of circuit mechanization, programming convenience, etc.

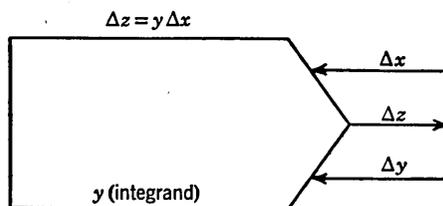


FIG. 1. Symbol for integrating unit of DDA.

**Integrator Schematic.** Schematically an integrating unit of a DDA may be represented externally as shown in Fig. 1. Internally an integrating unit of a DDA may be represented as shown in Fig. 2. The  $y$  *integrand register* contains the value for  $y$  at the start of the problem and accumulates the incoming values of  $\Delta y$  in order to keep the value of  $y$  up to date. The contents of the  $y$  register are multiplied by  $\Delta x$  and added periodically to the  $R$  *accumulator register*. The overflow from the  $R$  register is  $\Delta z$ . The  $R$  register contains the digits of lowest significance of the variable  $z$  with the digits of highest significance held in the register which is supplied by  $\Delta z$ . There may be multiple  $\Delta$  inputs to the integrand register.

**Solution of Differential Equation.** The integrators of a DDA may be interconnected to solve differential equations in much the same manner

## 28-04 DESIGN AND APPLICATION OF ANALOG COMPUTERS

as integrators of d-c analog computers or mechanical integrators of electromechanical analog computers. Thus in this procedure, the highest order derivative is isolated in terms of all other variables. *Its existence as an integrand is assumed, and it is integrated repeatedly to provide all lower order derivatives.* The feedback path is closed to the highest order derivative by combining all lower order terms according to the conditions of the differential equations.

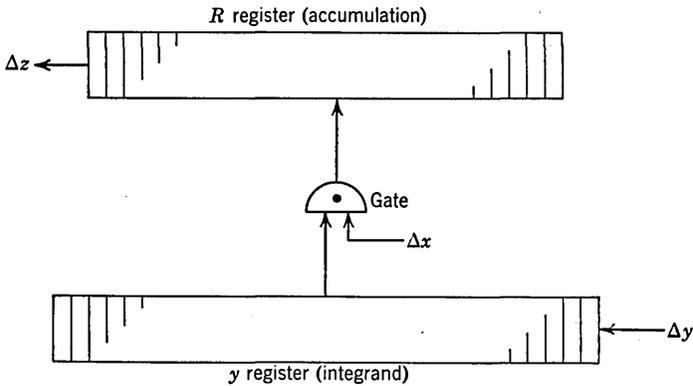


FIG. 2. Internal configuration of an integrating unit of a DDA.

EXAMPLE. Consider the differential equation

$$(6) \quad \frac{d^2y}{dx^2} - \frac{dy}{dx} - 2y = x$$

according to the above procedure and rewrite eq. (6) as

$$(7) \quad \frac{d^2y}{dx^2} = \frac{dy}{dx} + 2y + x.$$

The solution to this equation is shown in Fig. 3 and integrator 1 has the output  $d(dy/dx)$ . This output serves as the input to the integrand register of integrator 2. The output of integrator 2 then is summed with the other inputs to the integrand register of integrator 1 to generate the integrand  $d^2y/dx^2$ . The output of integrator 2 also goes to the integrand register of integrator 3 where the variable  $y$  is accumulated and is thus always available. The independent variable  $x$  may be generated as part of some other set of equations being solved in the computer, or it may be generated at a regular computer-timed rate.

**Scaling Problem.** In placing a problem on the DDA, variables must be scaled to fit the numerical range of the machine. Once the scaling relations are established it is possible to determine the initial value of

every integrand and to express these integrands in terms of machine values. Some of the points to be considered in scaling a problem for a DDA are analogous to those to be considered in an analog computer

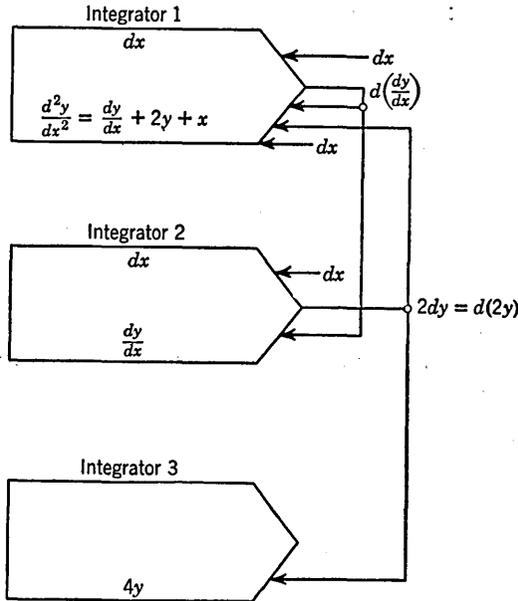


FIG. 3. Computer solution for the equation:

$$\frac{d^2y}{dx^2} - \frac{dy}{dx} - 2y = x.$$

which employs continuous variable representation. The important factors in scaling a problem for the DDA are:

1. All quantities are represented as integers.
2. The output relationship for an integrator is  $dz = y dx$ .
3. Scaling is done in terms of powers of the radix  $r$  of the DDA number representation. Thus for binary number representation  $r = 2$ , and the scaling is done in terms of powers of two.
4. Associated with each variable  $u$  there exists an *integer scale factor*  $S_u$  having the significance that the number  $r^{S_u}$  represents one unit of the quantity to the machine.
5. Associated with each integrand there exists a quantity  $m$  having the property that  $r^m >$  maximum absolute value of the integrand, where  $m$  is the smallest integer satisfying this property.
6. In general, one of two incompatible criteria, *precision* or *time*, is used to fix the scaling. Some particular variable may be required to be

## 28-06 DESIGN AND APPLICATION OF ANALOG COMPUTERS

of a certain precision, and thus fix its scale and establish all others. Alternatively, the length of computing time for a solution may be specified, and thus fix the scale on the independent, or driving variable of the problem. *An increase of one in the scale of the independent variable causes the computer to execute  $r$  times as many operations ( $r$  is the radix of number representation), each of which is of fixed time duration.* Hence, increased accuracy is obtained at the expense of increased computing time, and decreased computing time is obtained at the expense of decreased accuracy.

7. Once a choice has been made eq. (12) below may be applied to define the length of each integrator.

8. In the final analysis, the scaling of a problem is completely determined by the integrator lengths. A correctly scaled problem may be stepped up in accuracy or in speed of computation by readjusting all integrator lengths by the same amount.

**Scaling Relation for an Integrator.** The  $\Delta z$  output of an integrator represents the spillover from the least significant digits which are contained in the  $R$  register. Thus if the  $R$  and  $Y$  (integrand) registers are both  $N$  digits long, where the digits are expressed in the radix  $r$ , then it follows that

$$(8) \quad \Delta z = \frac{1}{r^N} y \Delta x$$

for an integrator. Now letting  $S_z$  equal the scale factor of the  $\Delta z$  output,  $S_y$  equal scale factor of  $\Delta y$  input, and  $S_x$  equal the scale factor of  $\Delta x$  input, the relationship realized will be

$$(9) \quad r^{S_z} \Delta z = \frac{1}{r^N} r^{S_y} y r^{S_x} \Delta x,$$

or

$$(10) \quad \Delta z = r^{S_y + S_x - N - S_z} y \Delta x.$$

To meet the condition that  $\Delta z = y \Delta x$  requires that

$$(11) \quad r^{S_y + S_x - N - S_z} = 1,$$

or

$$(12) \quad S_y + S_x - N - S_z = 0.$$

Equation (12) is a basic scaling relation and establishes a relationship among the number of digits in the  $R$  and  $Y$  registers, the scale factors on the variables of integration, and the scale factor on the output variable.

The  $Y$  register must be capable of holding the integrand  $y$  of the integrator at all times during the computation. Thus for each unit of

the integrand  $y$ , the  $Y$  register will have to hold the number  $r^{S_y}$ . Moreover, the integrand may be almost as large as  $r^m$  units at some time during the computation. Therefore, the total capacity of the  $Y$  register of any integrator must be capable of holding a number as large as  $r^{m+S_y}$  or  $r^{m+S_y}$ . Hence, since the  $Y$  register is  $N$  digits long, it follows that

$$(13) \quad N \geq m + S_y,$$

which says that the number of digits required for an integrator is equal to the scale factor of the integrand plus the quantity  $m$ . If  $m$  is not correctly known, or estimated, the capacity of the  $Y$  register may be exceeded. In this case the problem will be automatically halted (by an overflow signal) and must then be rescaled.

Equations (12) and (13) may be combined to yield a very useful, though not independent equation as follows:

From eq. (12)

$$(14) \quad S_y = S_z + N - S_x.$$

From eq. (13)

$$(15) \quad S_y \leq N - m.$$

Therefore,

$$(16) \quad S_z + N - S_x \leq N - m,$$

or

$$(17) \quad S_z - S_x \leq -m,$$

or

$$(18) \quad S_x - S_z \geq m.$$

**Scaling Relations between Integrators.** Equations (12), (13), and (18) define the relations necessary for the scaling of any single integrator, but they say nothing about scaling relations between integrators. These are simple and straightforward. To achieve compatible operation all integrators must accept a variable at a common scale factor. Violating this rule results in multiplication by powers of the radix, which may be useful at times.

Any set of scale factors satisfying eqs. (11) and (12) and meeting the compatibility requirement may be successfully used. The following procedure leads to a simple solution of the scaling problem.

Draw a schematic of the integrator interconnection as was done, for example, in Fig. 3. For each integrator enter scale factors and integrator

## 28-08 DESIGN AND APPLICATION OF ANALOG COMPUTERS

length as indicated in Fig. 4, with  $N_i$  the length of accumulator and integrand registers of the  $i^{\text{th}}$  integrator.

Now, let  $S_1$  be the output ( $dz$ ) scale factor for integrator 1. Enter  $S_1$  as the scale factor for the accumulator or integrand inputs for each integrator where the output of integrator 1 is used as an input. Let  $S_2$  be the output scale of integrator 2 and repeat the operation. Proceed in this manner until all scales are established. Note that if more than one integrator output is used as the integrand input to an integrator,

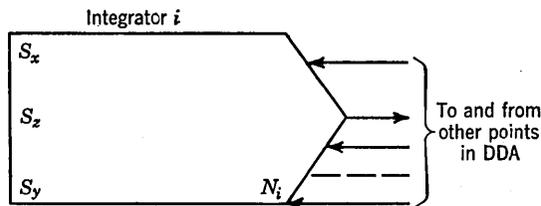


FIG. 4. Integrator marked for scaling purposes.

all such outputs must have a common scale. This process assures compatibility of all scale factors.

For each integrator write eq. (18). This yields a set of simultaneous inequalities of the form  $S_i - S_j > m_k$  which must be satisfied.

**Magnetic Drum Registers.** Thus far in the description, the registers have been depicted as being physically laid out as such. Actually, in the case of binary addition or subtraction, for example, the steps can be performed one column at a time, and it is therefore not necessary to have entire registers present in the electronic circuits of the DDA at one time. Thus in the DDA the integrator registers need be nothing more than segments on a magnetic drum (see Chap. 19) or bits in an acoustic or lumped parameter electrical delay line. The integrators can be processed either serially or in parallel.

For serial processing, numbers or areas from these registers are read one column at a time into the adding circuits and the altered numbers are replaced on the drum or in the delay lines. The integrators are processed serially or in parallel once per revolution of the drum or in each circulation through the delay line.

**Mechanization of a Serial DDA.** Consideration will now be given to some of the mechanization details for a DDA which processes the integrators serially. Mechanization details for an all parallel DDA follow in like manner except that the integrators are processed in parallel. For the serial DDA one track of the magnetic drum storage must be reserved to contain serially the integrand or  $Y$  registers for each of the

integrators. Similarly, one track must be reserved to contain serially the  $R$  registers of each integrator. There is also one "hookup track," labeled the  $L$  track, which contains serial registers, one for each integrator, and provides the necessary information to interconnect the outputs and inputs of each integrator. The  $L$  track operates in conjunction with an additional track referred to as the  $\Delta Z$  track to supply the  $\Delta y$  and  $\Delta x$  inputs to each of the integrators. Thus as each  $\Delta z$  is generated in the DDA it is deposited on the  $\Delta Z$  track in a space reserved

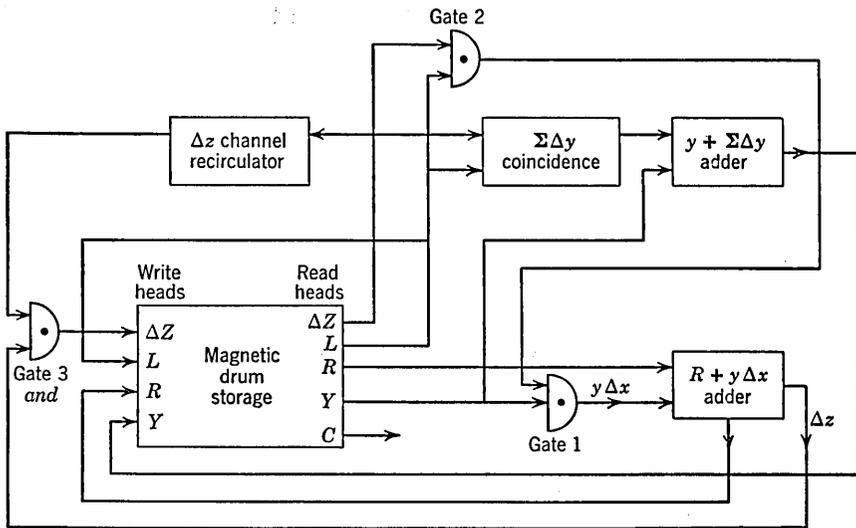


FIG. 5. DDA schematic.

for the  $\Delta z$  output of the particular integrator. Hence as indicated in Fig. 3, this  $\Delta z$  output can serve as a  $\Delta y$  input for appropriate integrators in the problem under consideration. Thus typically as in Fig. 3, when integrator 1 is processed, it can generate a  $\Delta z$  output. This  $\Delta z$  output is placed in a space reserved in the  $\Delta Z$  track for integrator 1. As a result of the equation involved, during the processing of integrator 2 its  $\Delta y$  input is obtained from the  $\Delta z$  output of integrator 1. The  $L$ , or problem hookup, track is coded so that it inspects the  $\Delta Z$  track only in the space at which this  $\Delta z$  is placed, and upon detecting a  $\Delta z$  in this space sends it to the  $y$  register of integrator 2 as a  $\Delta y$  input. This process is then readily generalized for more complicated problem hookups.

**DDA Processing Schematic.** Schematically the DDA may be depicted as shown in Fig. 5. In the box labeled storage, on the right-hand side, are depicted the read head outputs (see Chap. 19) of the

## 28-10 DESIGN AND APPLICATION OF ANALOG COMPUTERS

various DDA tracks. The lowest track labeled  $C$  is the clock pulse track. The function of this track is to supply pulses for timing and pulse reshaping purposes. The output of the  $Y$  track, which contains the integrand register contents in serial for the various integrators of the DDA, goes two places. (a) First it goes to block 1, which combines it with  $\Delta x$  in order to produce  $y \Delta x$ , which in turn is added to the  $R$  register, as pointed out earlier. (b) It also goes to the  $y + \Sigma \Delta y$  adder, where it is combined with the sum of the incremental inputs from the other integrator outputs which are to supply the integrand currently being processed. Thus in Fig. 3, for example, the integrand of integrator 1 picks up incremental inputs from three sources. The sum of incremental inputs from such sources is represented as  $\Sigma \Delta y$ . The output of the  $y + \Sigma \Delta y$  adder then goes back to the  $Y$  track of the drum to bring the contents of the  $Y$  integrand register up to date.

The output of the  $R$  track goes directly to the  $R + y \Delta x$  adder. This adder produces two results, an overflow pulse which is placed in an appropriate place on the  $\Delta Z$  track and an updated value for  $R$  which is placed in the  $R$  track and which thus replaces the previous value of  $R$  for the integrator being processed.

The output of the  $L$  track goes to three places. (a) It goes first to a  $\Sigma \Delta y$  coincidence block. The function of this block is to examine the  $L$  track which has been coded at the outset of the problem according to the problem being solved. Pulses placed in the  $L$  track for each integrator register cause the  $\Delta Z$  track to be examined at the spaces reserved for the appropriate  $\Delta z$  outputs through the action of the coincidence block. Thus the increment in  $y$  is determined by computing  $\Sigma \Delta y$  in the coincidence circuit. The output of the coincidence circuit then goes to the  $y + \Sigma \Delta y$  adder to form the updated value for  $y$ . (b) The  $L$  track also goes to block 2, which is a coincidence circuit for producing  $\Delta x$ . That is, the  $L$  track is also coded at the outset of the problem to pick up the required  $\Delta x$  for each integrator of the problem. (c) The  $L$  track also goes back to the input of the  $L$  track to be replaced without change since the coding stays fixed throughout the problem.

The  $\Delta Z$  track also goes three places. (a) It goes to the  $\Sigma \Delta y$  coincidence block and (b) to block 2 to combine with the  $L$  track at both of these blocks with the results described above. (c) It also goes back to the input of the  $\Delta Z$  track after combining at block 3 with the  $\Delta z$  output of the integrator just processed.

**Handling Positive and Negative Increments.** There are several alternatives for transferring positive and negative information in the DDA. The first of these involves a two-level or *binary transfer* scheme. In this scheme positive incremental rates are represented by a predominance of

1's transmitted, negative incremental rates are represented by a predominance of zeros transmitted, and zero incremental rates are represented by alternate 1's and 0's. It is readily possible to mechanize a DDA using this information transferral scheme (Ref. 1).

A second alternative involves the use of a three-level or *ternary transfer* scheme. Here information is transmitted directly as +1, 0, or -1. This scheme is more accurate than the binary transfer scheme in that it has less inherent roundoff error, but requires more equipment to realize (Ref. 2).

**Selection of Method of Mechanization.** It is possible to mechanize DDA's by using rectangular, trapezoidal, or other integration techniques. The more sophisticated technique will, in general, produce a more accurate computation, but it will, in general, also require more equipment to mechanize.

The choice of integration technique and information transfer scheme for a DDA intended to solve specific classes of problems is based on such considerations as accuracy requirements consistent with reasonable equipment demands. For a DDA intended for general problem usage where accuracy and frequency requirements may be specified to cover a broad spectrum, it is often felt essential to consider the construction of the DDA to employ ternary transfer and trapezoidal integration. However, special purpose DDA's intended for problems with modest requirements may make it possible to employ a simpler DDA configuration.

The integration technique and information transfer schemes required of a DDA to handle any particular class of problems may be determined to a good degree of approximation by analytical means (Refs. 3, 4, and 5) whose results can be verified by suitable simulation studies (Ref. 6).

### 3. DIGITAL OPERATIONAL COMPUTERS

**Approach.** An analog computer can employ discrete variable representation for information transfer through the use of a variable pulse repetition frequency. Thus to establish the feasibility of such a mechanization it is merely necessary to establish the configuration of suitable operational units such as multipliers, dividers, and integrators. Once these units have been described, it is then a straightforward step to realize that such an approach can be suitably employed to solve differential equations by properly interconnecting the units as was done, for example, to solve the problem shown in Fig. 3. For more details of operational digital techniques see Chap. 29.

**Multiplier.** For configuration of a multiplier for a system employing this data representation technique may be established by considering the

## 28-12 DESIGN AND APPLICATION OF ANALOG COMPUTERS

simplified situation depicted in Fig. 6. The blocks labeled 1 through 4 are the stages of a binary counter. Thus if a chain of pulses whose rate is proportional to the number  $x$  is supplied at the left hand of the counter, the *and* gate (see Chap. 14) on the first line of the multiplier receives from the first stage of the binary counter a pulse rate of  $x/2$ , and so on for the succeeding stages of the binary counter. The contents of the  $y$  register then determine which of these sequences of

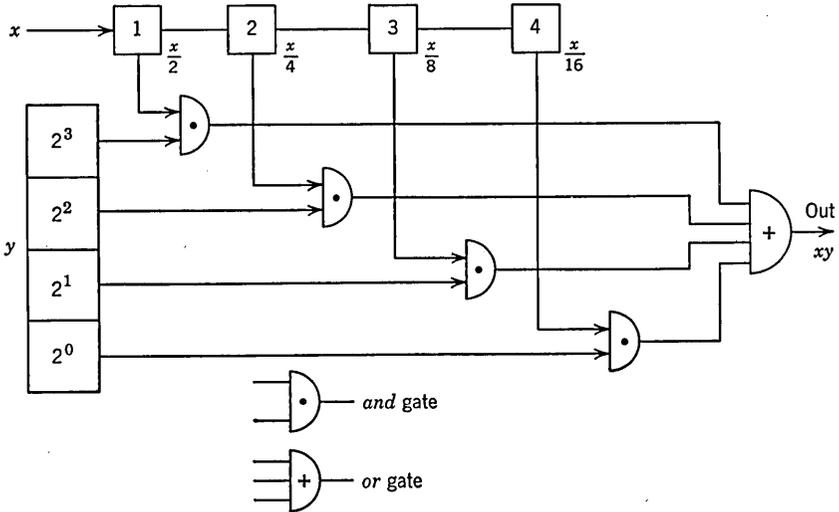


FIG. 6. A digital operational multiplier.

pulses are passed on to be added through the *or* gate (see Chap. 14) shown at the right of the multiplier. The output of the *or* gate then is the desired product. Thus if  $y$  is zero and the contents of the  $y$  register consists of all zeros, all the *and* gates will prohibit the scaled down  $x$  pulse rates from passing through to the *or* gate. Thus the output product will be zero as it should. Similarly, if  $y$  is at its maximum value, the maximum number of pulses will be passed on to the output of the multiplier, and the output pulse rate will again be proportional to the product  $xy$ .

The pulses from the various *and* gates must be separate from one another, otherwise when these pulses are combined at the output of the *or* gate any overlap would result in a loss of information. This overlap is avoided in a straightforward manner by taking the pulses out of the binary counter into the *and* gate at the noncarry time of the respective stages of the counter. Thus when the pulses are combined at the output

of the *or* gates, they are separate and distinct, and the resultant pulse train output is indeed proportional to the product  $xy$ .

**Divider.** Now that a multiplier configuration has been established a mechanization for a divider follows readily. Thus in Fig. 7 the three multipliers combined with a forward backward counter, i.e., a counter which counts up or down depending upon the inputs to  $F$  (forward) or  $B$  (backward), result in a circuit which produces the quotient in the forward backward counter. The schematic shown in Fig. 7 has one feedback loop involving  $z$ . The  $P$  input is a pulse rate, and the numbers  $x$  and  $y$  are held in the indicated registers. The forward backward

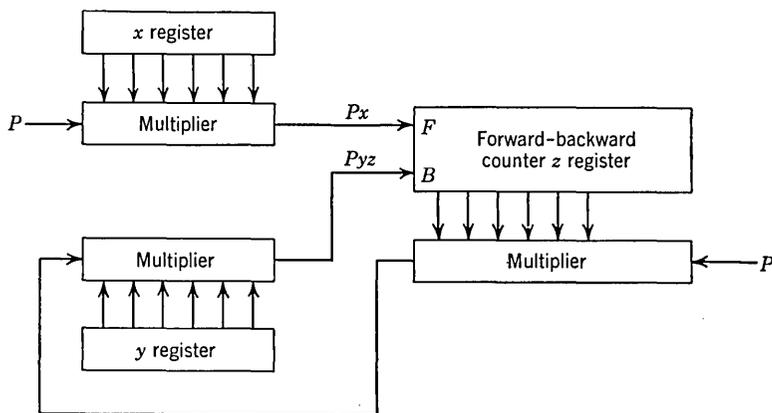


FIG. 7. A digital operational divider.

counter will count forward or backward, adjusting  $z$  until the counter input  $Px - Pyz$  is equal to zero. At that time the quantity  $z$  held in the forward backward counter is related to  $x$  and  $y$  by

$$(19) \quad Px - Pyz = 0,$$

or

$$(20) \quad z = \frac{x}{y},$$

and  $z$  is the desired quotient.

**Divider Time Constant.** A transient analysis may be made of this divider. Thus if there are  $n$  stages in the  $X$ ,  $Y$ , and  $Z$  registers, from an examination of Fig. 7 there follows

$$(21) \quad Z(t) = \int_0^t \frac{XP}{2^n} dt - \int_0^t \frac{YPZ(t)}{2^n} dt.$$

## 28-14 DESIGN AND APPLICATION OF ANALOG COMPUTERS

By differentiating this there follows

$$(22) \quad \frac{dZ(t)}{dt} + \frac{YP}{2^n} Z(t) - \frac{XP}{2^n} = 0.$$

Solving this equation for the initial conditions at  $t = 0$ ,  $Z(t) = Z(0)$  gives

$$(23) \quad Z(t) = \frac{X}{Y} \{1 - [1 - Z(0)] e^{-(YP/2^n)t}\}.$$

Thus the equivalent time constant of an operational divider is given as  $2^n/YP$ . The equivalent time constant can be reduced by increasing the pulse repetition rate  $P$  or by decreasing  $n$ , the number of stages.

**Integration.** The process of integration by this representation technique may be mechanized by simply accumulating all incoming pulses in a forward backward counter such as that used in Fig. 7. As a positive pulse rate is supplied to the  $F$  input of a forward backward counter, its contents increase. Negative pulse rates are supplied to the  $B$  input and the counter's contents decrease.

### Reasons for Using Discrete Variable Representation

There are several reasons for considering the use of this approach in computing.

1. A serial DDA where only one set of adders and other computing functions are required may lead to a simpler computing equipment when the dynamic and accuracy demands of the equations being solved are modest.

2. In real time simulation or control where high accuracy requirements must be met and/or where high dynamic requirements exist, the best solution to the computing problem may well be provided by the use of all parallel DDA elements either throughout the computing system or else at critical positions in the system. For example, an all parallel DDA employing registers 30 binary digits in length, a basic pulse repetition frequency of one megacycle, trapezoidal integration, and ternary transfer provides an accurate computing facility with good dynamic properties as evidenced by the fact that its precision is one part in better than 1,000,000,000, and its integration interval is 30 microseconds, i.e., it provides better than 30,000 quadratures a second.

**Choice of Computing Method.** Thus, for example, as one approach to choosing between the computing methods, a comparison between the true characteristic frequencies of a linearized set of system equations may serve to provide sufficient data to select between the methods, particularly so if one of them shows up very badly in the comparison. Non-linear systems can often be linearized in the Liapounoff sense (Ref. 10) for such analyses to a very good degree of approximation over a consid-

erable region. Consider for example, the very good degrees of approximation afforded by linearizing the generally nonlinear equations of motion of aircraft over certain ranges of the dependent variables. Techniques for determining the difference between the true characteristic frequencies of a system and those computed by an analog computer employing continuous variable representation have been presented in the literature (Refs. 8 and 9). Similar techniques for analog computers employing discrete variable representation have also been presented (Ref. 3). If the system equations are nonlinear to the extent that this approach is not held to be valid or if complete verification is desired before embarking on an extensive computer design and construction program, full scale system simulation must ultimately be employed (Ref. 6).

**Summary.** The reasons for using discrete variable representation in an analog computer include possible equipment reductions or/and better accuracy and dynamic performance.

#### 4. AUXILIARY DIGITAL COMPUTING TECHNIQUES

It is often possible and desirable to incorporate auxiliary digital computing elements in an analog computer. Thus function generation may be conveniently carried out in this manner particularly when functions of several variables are involved. Also in certain control systems it is often necessary or desirable to be able to multiply an analog variable directly by a digitally represented variable to produce a product in analog form. By avoiding the conversion of the digital variable to an analog variable through the use of a digital to analog converter and then applying the result to a conventional analog multiplier an overall system which is simpler and generally more accurate results.

**Digital Function Generation in Analog Computers** (Ref. 11). It is possible to generate functions of analog variables by converting the analog variable to a digital number through an analog to digital converter. The digital number then may go to a small auxiliary magnetic drum and associated circuitry which is programmed to generate the desired function. Arbitrary functions of several variables can also be generated in this same manner. If it is then desired to multiply the desired function directly by another variable in the computer as is so often the case, it is not necessary to convert the digitally evaluated function back to analog form. Instead, the quantity can be fed directly to the digital analog multiplier described below.

Function generation can also be accomplished by means of a digital function table as shown in Fig. 8. Thus values of  $x$  over the expected range of  $x$ , in suitable increments are stored in parallel in the  $x$  space

## 28-16 DESIGN AND APPLICATION OF ANALOG COMPUTERS

of the drum. This space is examined by the read heads and converted to an analog quantity by the digital to analog converter shown in the figure. This converted quantity is compared with the variable  $x$  for which an arbitrary function is desired. When coincidence occurs the read heads associated with the function digits table, stored on the drum and shown as  $f(x)$ , are gated, and the digits passing under them are read

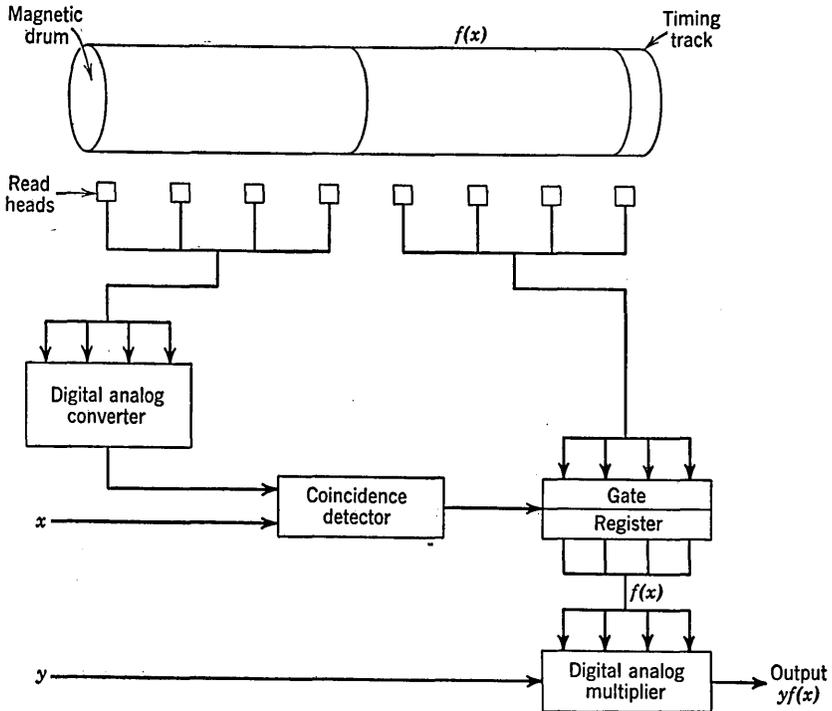


FIG. 8. Digital function table.

through the gate shown into the register. The register contents then may go to a digital analog multiplier, as shown, to generate the quantity  $yf(x)$ . Although only one arbitrary function was shown generated in Fig. 8, it is evident that this same technique can be extended to the instance where several different arbitrary functions of any arbitrary variable can likewise be generated.

**Digital Analog Multiplier** (Ref. 11). It was shown in Chap. 22 that for an operational amplifier the output voltage is equal to the input voltage multiplied by the ratio of the output impedance to the input impedance. Thus if this impedance ratio can be directly related to one

variable and the input voltage is another variable, the output voltage is proportional to the product of the two variables. This concept then leads to the simple digital analog multiplier shown in Fig. 9, wherein

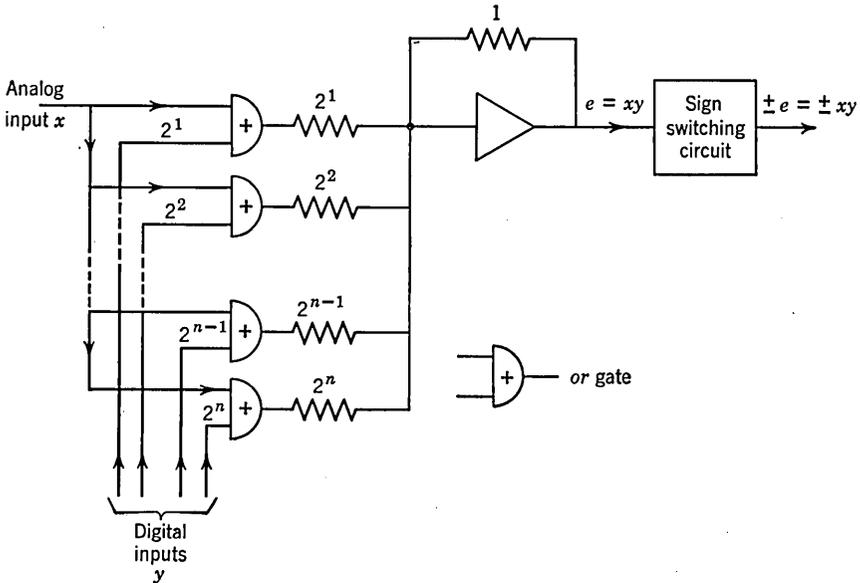


FIG. 9. Digital analog multiplier.

the parallel conductances of the input conductance are switched into or out of the circuit according to the variable. The sign digit of the digital variable can be used for switching purposes at the output in order to produce a four-quadrant multiplier.

**5. AUXILIARY DIGITAL CONTROL TECHNIQUES** (Refs. 12 and 13)

Digital techniques can also be advantageously employed for control purposes in an analog computer. These techniques are of particular advantage in large scale installations. Thus entire problems can be placed on punched tapes or cards operating in conjunction with patch panels. Thus the tapes or cards can control the settings of coefficient potentiometers, condensers, initial conditions, etc., through the use of associated suitable switching circuitry operating in conjunction with servomechanisms coupled to the proper points at the proper time in the computer through suitable clutching arrangements. The patch panel would contain the information to interconnect the proper amplifiers, potentiometers, etc.

## 28-18 DESIGN AND APPLICATION OF ANALOG COMPUTERS

*Advantages.* Problems can be readily checked since at any desired time the interconnections and all settings can also be arranged to be printed out in some suitable form.

Problems can be rapidly placed on the machine and removed if need be temporarily with the full realization that the problems can again be rapidly placed on the computer.

More efficient use can be made of extensive facilities.

### REFERENCES

1. I. Reed, Fundamental concepts of the DDA, *Mathematical Tables and Other Aids to Computation*, **6**, 41-49 (1952).
2. M. Poalalevsky, The design of the Bendix DDA, *Proc. I.R.E.*, **41**, 1352-1356 (1953).
3. H. J. Gray, Numerical methods in digital real time simulation, *Quart. Appl. Math.*, **12**, 133-140 (1954).
4. K. Miller and F. J. Murray, A mathematical basis for an error analysis of differential analyzers, *J. Math. and Phys.*, **32**, 136-163 (1953).
5. H. Rademacher, On the accumulation of errors in integration, *Ann. Computation Lab. Harvard Univ.*, **16** (1948).
6. F. Wright Jr., *Dynamic System Studies*. Pt. 13. *Error Studies*, WADC Tech. Rept. 54-250, September 1956.
7. L. P. Meissner, Real-time digital differential analyzer (DART), *Proc. 1954 West. Jt. Comp. Conf.*, 134-139, Los Angeles, Calif., February 1954.
8. A. B. MacNee, Some limitations on accuracy of electronic differential analyzers, *Proc. I.R.E.*, **40**, 303-308 (1952).
9. E. Goldberg and G. Brown, An electronic simultaneous equation solver, *J. Appl. Phys.*, **19**, 339-345 (1948).
10. N. Minorsky, *Introduction to Nonlinear Mechanics*, Edwards Bros., Ann Arbor, Mich., 1947.
11. Digital techniques in analog computers, *Proc. West. Jt. Comp. Conf.*, 60-63, San Francisco, Calif., February 1956.
12. An automatically controlled large scale Geda computing and simulation installation, GER-7808, Goodyear Aircraft, Akron, O., August 1956.
13. *Operating Manual EASE 1200 Series Analog Computer*, Beckman Instruments, Palo Alto, Calif., 1957.
14. J. M. Mitchell and S. Ruhman, The THRICE—a high speed incremental computer, 1958 I.R.E. Convention Record, Pt. 4, 206-216.

# UNUSUAL COMPUTER SYSTEMS

## F. UNUSUAL COMPUTER SYSTEMS

29. *Operational Digital Techniques, by Bernard M. Gordon and John F. La Fontaine*
30. *Combined Analog-Digital Computing Systems, by George P. West*
31. *Simple Turing Type Computers, by Joseph O. Campeau*



## Operational Digital Techniques

*Bernard M. Gordon and John F. La Fontaine*

1. Introduction	29-01
2. Basic Devices	29-05
3. Applications of Operational Digital Techniques	29-14
4. Incremental Computation	29-17
References	29-29

### 1. INTRODUCTION

Operational digital techniques describe an approach to computation that attempts to combine the advantages of both the analog and the programmed digital methods of computation. These combined techniques effect what is called a *hybrid* system. They do not always provide the best solution to a computation problem, and their useful application is generally economically possible only when especially developed novel components are used.

**Characteristics of Hybrid Systems.** Essentially, in the hybrid system, digital techniques are used in functional units that are laid out in operational form. Objectives of the hybrid system are:

1. Greater precision that can be attained with analog techniques and at lower cost.
2. Greater control and speed than is possible with programmed digital units.
3. Less complexity than is involved when programmed digital units alone are used.

4. The ability to accept input data in the analog form, in which they usually occur. These data are translated into the digital language of human beings and with as little recourse as possible to accuracy limiting analog components.

A consideration of operational analog systems and of programmed digital systems indicates the advantages and disadvantages of each type.

**Analog Systems.** In analog systems, the variables to be represented or controlled are physically characterized by parameters which may vary over a limited range. A varying voltage, for example, may be proportional to, and thus represent, a varying parameter. Other examples are: signal frequencies, mechanical shaft positions, and light intensities.

### *Advantages*

1. The analog system is essentially a visual working model of the parameters being controlled or measured. The design of a given system is reduced to the interconnection of components in a straightforward "operational" manner. Functional units (gears, potentiometers, and synchros), each performing a specific function, are arranged so that the overall transfer function has the desired properties. *Example.* An analog system for multiplying an input quantity by three would consist of gears arranged so that three turns of an output shaft occurred for every single turn of the input shaft.

2. The real time characteristic of analog operational techniques is also an advantage. By *real time* is meant simply that the analog process occurs during the same time as does the variation of the parameters it represents. This continuous, as it may be called, nature of analog operational techniques makes them especially applicable for closed loop systems. Particularly when the control function is complex, the operational system may respond faster than is sometimes possible with the periodic sampling operation of programmed digital systems.

### *Disadvantages*

1. In terms of accuracy and ultimate reliability, the analog system is limited by the precision of the components composing it.

2. Economics often impose restrictions on the precision of an analog system. There is a practical consideration here, namely how much time and money should be put into developing ultra-precision components for a particular analog system?

3. In practice, the accuracies of practical complex analog systems are limited to one part in a thousand.

4. Some operations such as multiplication may be slow.

*Summary.* The analog approach has the advantages of high speed and real time operation. It is limited in precision, however; and it is also limited in its ability to perform those operations that are not readily representable in analog form, namely certain nonlinear mathematical functions.

**Digital Systems.** The programmed digital technique is quite different from the operational analog technique. Data are not processed in a form analogous to the input information, but rather in the discrete form of a series or set of characters, i.e., information in language form.

#### *Advantages*

1. The components in a digital system can vary and drift and be off tolerance considerably before they cease being dependable functional units. (See Chap. 12.) Such extreme component tolerances are never possible in the analog system.

2. Any degree of accuracy can be obtained and maintained if a sufficient number of characters are used in the language code of the system. In straight binary code, for example, accuracies on the order of one part in 1000 can be obtained if 10 decision elements are used, or one part in 2000 if 11 decision elements are used.

3. The digital computer is flexible enough to perform any type of mathematical operation.

#### *Disadvantages*

1. Since the processing is carried out by means of an ordered program, the digital machine must be complex and must contain an internal storage to provide storage of data and commands. The computer and storage are both large and costly.

2. In the digital computer the input quantity must be periodically sampled in order for the machine to determine if a change has taken place. *Example.* Consider the same example given above, that is, multiplying an input quantity by three. For the digital machine to execute this simple function it is necessary for the input data to be transferred to the internal storage. The data are then transferred to the arithmetic element where each digit is multiplied by three in accordance with the program previously inserted in the machine, and then the result of this multiplication is returned to the storage and finally transferred to the output.

3. These operations—sampling, transferring to storage, operating as programmed, transferring to storage, generating control data, etc.—take time. Consequently, in a digital computer the rate of sampling must be limited, and thus information sacrificed, or the computer must operate

at extremely high speeds; or a combination of these adjustments must exist.

*Summary.* The digital computer techniques have the advantages of reliability and versatility; these advantages are offset by complexity and expensiveness. But if the problem to be solved or the measurement required demands that minimum time elapse between receipt of the input information and the computed response, much of the utility of the digital type of instrumentation is negated.

**Hybrid or Operational Digital Systems.** The purpose of the hybrid system, therefore, is to combine the advantages noted above for each of the two types of conventional computer, while at the same time obviating the disadvantages.

It is often assumed that because the physical world is a continuously moving analog system, only analog techniques are applicable to it. A closer look at nature, however, reveals that there are many physical phenomena that are basically discrete. To name a few, the breakdown of nuclear energy, the passing of time intervals, and the occurrence of resonant cyclic vibrations occur incrementally.

**EXAMPLE.** The application of operational digital techniques to the doppler phenomenon as the basis for the analysis of motion is a good example. It is known that when energy of a specific frequency is reflected from a moving object, the returned energy undergoes an apparent frequency shift, the doppler shift. This change, or doppler frequency, is proportional to the velocity of the object under observation, and therefore it can be used as a source of data in velocity measuring, tracking, and navigational systems. In analog systems this doppler information is converted to shaft rotation rate or proportional voltage. A closer investigation of the doppler relation indicates that each cycle of the received data can indicate a definite distance traveled by the object causing the frequency shift. The doppler relation is of the form

$$f_d = k \frac{v}{c} f = k \lambda v,$$

where  $f_d$  = doppler frequency,  
 $v$  = velocity to be measured,  
 $f$  = transmitted frequency,  
 $c$  = propagation velocity,  
 $k$  = a constant,  
 $\lambda$  = transmitted wavelength.

Since  $f_d$  (cycles per second) is proportional to  $v$  (distance per second), by removing time from the relationship, each cycle of the received data represents a definite distance traveled. To integrate, it is necessary only

to count the number of cycles received. No conversions are required. Limitations on accuracy are (1) the wavelength of transmission and (2) the number of digits that can be held in the counting device.

Note the following:

1. The doppler information is an analog of the moving object, hence, the system receiving and using this information is in a sense an analog system.

2. The measurement is made in real time.

3. A discrete meaning is applied to each cycle of the received energy, hence, the information is digitally characterized.

4. The combination is a hybrid system.

In this hybrid system:

- a. The measurement is operational, that is, it is accomplished in real time; the received data are proportional to the parameter being measured.

- b. As a digital system, the received data are in language form, and the degrees of accuracy realizable with the digital technique are therefore possible. A characteristic name for such a technique, therefore, is *operational digital*.

## 2. BASIC DEVICES

Data representation within operational digital systems takes the form of both unitary-weighted pulse trains and binary codes. Pulses which form a unitary-weighted pulse train each have the same numeric weight. Thus, 432 units are represented by a train of 432 pulses; 654 units are represented by a train of 654 pulses, etc.

### Binary Rate Multiplier

#### Characteristics.

1. This unit accepts a unitary pulse train as one input and a numeric, that is, binary, code as the other input.

2. The output is a pulse train containing a number of pulses equal to the product of the number of input pulses (the multiplicand) and the numeric code (the multiplier).

3. Each pulse of the output train has exactly the same unitary weight as those of the input train.

**Operation.** A binary rate multiplier consists of binary scalars and diode gates. The scalar is a device that yields an output for every two inputs and is indicated by  $C_2$  in figures. Thus, if several scalars are arranged in a series fashioned as in Fig. 1 and a pulse train applied to the input of the series:

1. The first scalar will yield an output after two inputs.

2. The second scaler will yield an output after two inputs from the first scaler, and therefore after four inputs to the series of scalars.

3. The third scaler will, similarly, yield an output after two inputs from the second scaler, which means it will yield an output after eight inputs to the series of scalars.

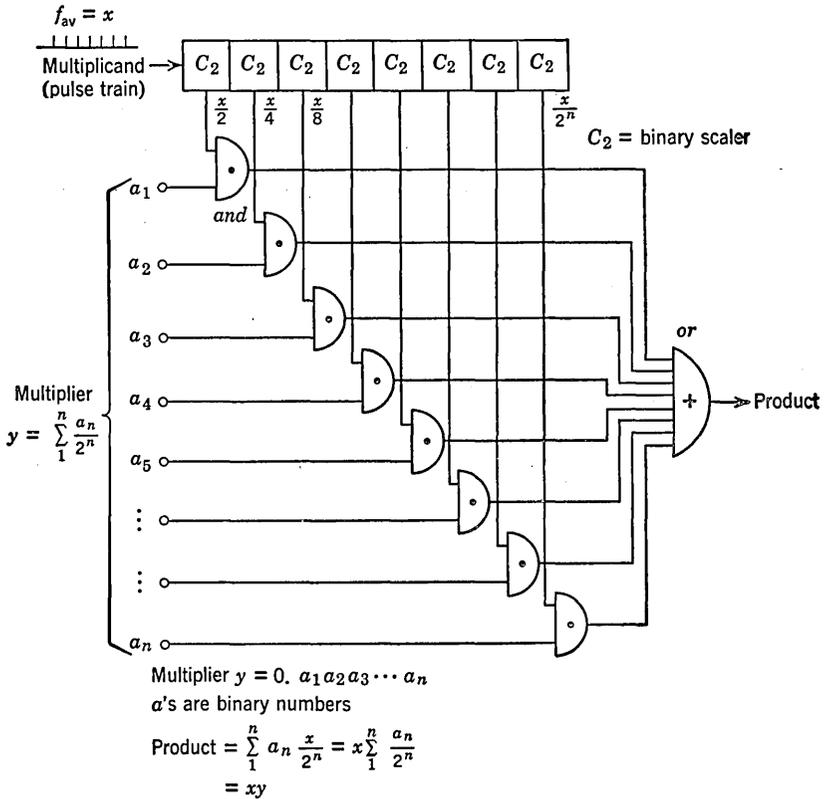


Fig. 1. Binary rate multiplier block diagram.

4. The scaler outputs, therefore, during the interval in which input pulses arrive are:  $x/2, x/4, x/8, \dots, x/2^n$ , when  $n =$  number of scalars.

5. This means that one-half of the total number of input pulses, during a certain interval, will occur at the output of the first scaler, one-quarter at the output of the second scaler, one-eighth at the output of the third, etc. Since the sum of all these fractions approaches one, as  $n$  becomes large, it is evident that the total number of scaler outputs will tend to equal the number of input pulses.

The binary-weighted scaler outputs are applied to gates that are opened or closed according to the dictates of the binary-coded multiplier. When a gate is opened, the output pulses of the associated scaler will be passed. The outputs of all gates are combined through a buffer to yield a train of pulses that are the product desired. The multiplier must be scaled to be a fraction less than unity.

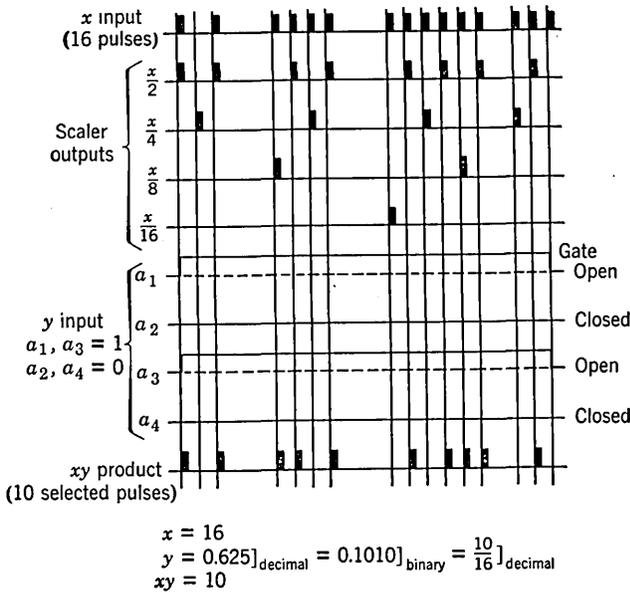


FIG. 2. Binary rate multiplier timing diagram.

**EXAMPLE 1.** Consider an input pulse train of 1000 pulses. If it is desired to multiply this input by zero, the binary numeric code for zero, 0.0000  $\dots$  0, sets all the gates closed, and thus none of the input pulses appears at the output, and the product is zero. If it were desired to multiply the input train by one, the binary code for one, 0.1111  $\dots$  1, would open all the gates, and thus all the scaler outputs would be passed. For a more sophisticated example, assume that it were desired to multiply the 1000 pulses by 0.6250, the binary code for which would be 0.1010  $\dots$  0 (i.e.,  $\frac{1}{2} + \frac{1}{8}$ ). In this case, the gate associated with the first scaler would be opened, and the one associated with the third scaler would be opened. Since half of the pulses, or 500, would be passed by the first scaler, an one-eighth of the pulses, or 125, passed by the third scaler, the output train would consist of 625 pulses, which is the correct product for such multiplication.

**EXAMPLE 2.** Figure 2 illustrates the input, gating, and output involved in multiplying  $x = 16$  by  $y \times 0.625$ .

**Accuracy.** According to the number of scalers and gates used, namely the number of digits used in the binary code multiplier, any degree of accuracy can be achieved. For example, for an accuracy of one part in 2000, 11 gates would be required.

**Timing.** In order that the output device receives pulses one at a time, which it must do if a unitary pulse train composed of the correct number of pulses is to be obtained, no two scaler outputs can occur at the same time. This prevention is accomplished by interconnecting the scalers so that the carry from one stage to another is out of phase with the output pulses.

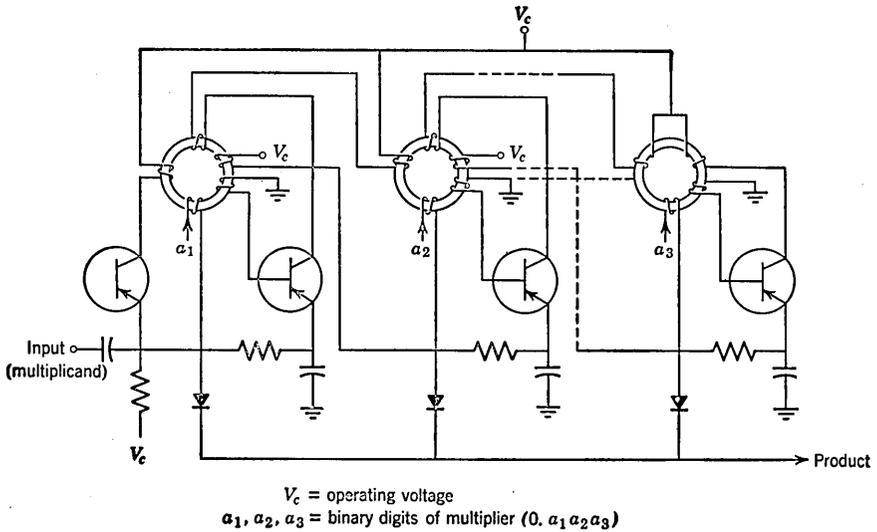


FIG. 3. Magnetic core transistor multiplier block diagram.

**Magnetic Core Transistor Multiplier.** Another operational digital device is the magnetic transistor multiplier. This unit consists of one magnetic square loop core and one current source per binary stage. Each stage operates as a combination gate, flip-flop, and blocking oscillator.

**Operation.** The operation of the unit is as follows. (See Fig. 3.)

1. Each collector drives current around both its related core and the succeeding core; each core then has two collector windings from successive stages. These windings are in opposing directions.

2. As the successive stages are excited, the flux is caused to vary from the upper remanence point (arbitrarily called 1 state) to the lower remanence point (zero state).

3. During a flux change, the polarity of a pulse developed in the lower winding of a given core will depend on the initial condition of that core.

Consider any particular stage. When a positive pulse is developed in the second winding, it is applied to the emitter of the next transistor. Whether or not that transistor is excited depends on the polarity of the pulse simultaneously developed in its base winding. If that polarity is positive, the emitter-to-base voltage does not become positive, and no transistor action ensues. If the base pulse is negative, the combination transistor-magnetic circuit is caused to regenerate much in the manner of a blocking oscillator. Thus, each stage is caused to read out once for every two pulses received and a binary count is achieved. *Note.* This alternate reinforcement and cancellation action ensures that pulses never exist simultaneously on two or more output lines.

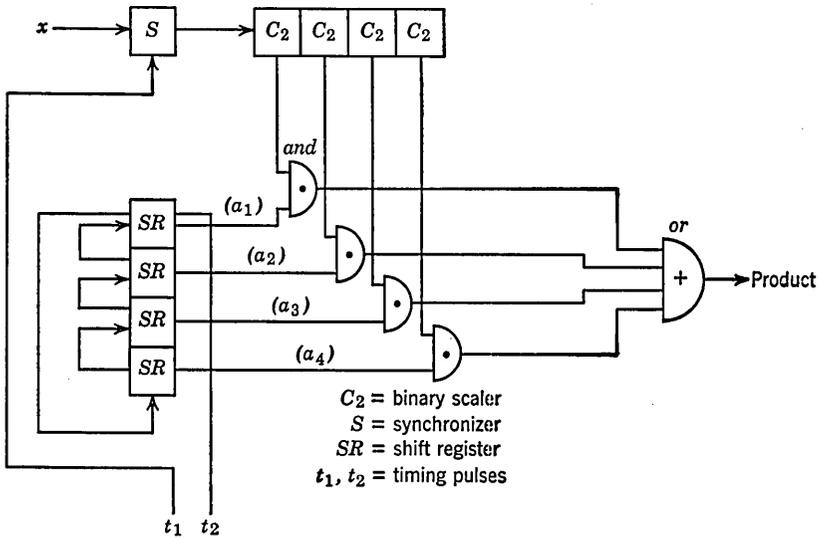


FIG. 4. Dynamic rate multiplier block diagram.

### Dynamic Rate Multiplier

**Description.** In the basic rate multiplier, as discussed above, the input pulses occur asynchronously, and consequently the gate signal inputs of the binary coded multiplier had to be present at all times. Thus, storage in static storage elements was necessary. An arrangement known as a dynamic rate multiplier (see Fig. 4) permits the binary multiplier coefficient to be stored in a circulating shift register, and once every major shifting cycle, the multiplier code is in the proper position relative to the gates. If the incoming pulses are synchronized to the shift pulses, proper

gating action takes place. The advantage of this arrangement is that static storage is obviated and full use is made of magnetic switching elements basically suited for dynamic operation.

**Pulse Rates.** In the dynamic rate multiplier, the rate of pulse occurrence determines the multiplication required for a given accuracy. For example, if 0.1 per cent accuracy is desired, time intervals for occurrence of at least 1000 pulses must be allowed. Such pulse rates are easily attained with present magnetic circuit techniques.

**Applications.** The basic rate multiplier and its variations may be used to perform a series of rate multiplications in which the output of one multiplier is fed directly into another without need for intermediate storage. The units may also be incorporated into feedback systems to provide operational units whose transfer characteristics can be analyzed in terms of equivalent time constants in a manner analogous to proportional servomechanisms.

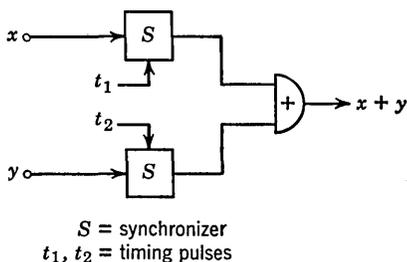


FIG. 5. Adder block diagram.

### Adder

The addition of unitary-weighted pulse trains is simply performed, as indicated in Fig. 5. The pulse trains  $x$  and  $y$  are synchronized at pulse times  $t_1$  and  $t_2$ , respectively, so that the pulses of each train occur out of phase with the pulses of the other train. The two out-of-phase scaler output trains are then combined, and the output pulse train is the sum of the two input trains. A simple adder could be designed using only two saturable core elements and two diodes.

### Subtractor

The next diagram shows a device used to subtract one pulse train input  $x$  from another. Pulse train  $y$  is to be subtracted from pulse train  $x$ . A reversible binary counter, a forward-backward counter, is used, along with two binary scalars. The  $x$  pulse train is applied to the forward input of the counter and the  $y$  to the backward input. Thus, the count

in the counter is, at all times, the difference between the two pulse trains. The difference  $x - y$  in the counter register may be used as the multipliers input for a binary rate multiplier (BM) to produce the product  $f_0(x - y)$  as shown in Fig. 6.

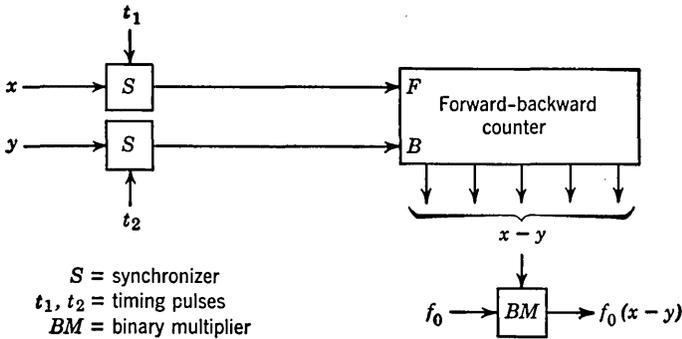
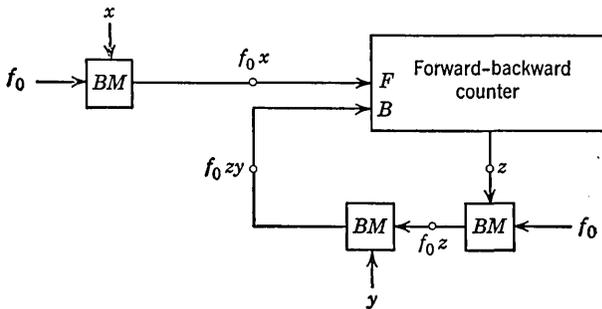


Fig. 6. Subtractor block diagram.

**Divider**

With the basic devices of the binary rate multiplier and the reversible counter, an operational digital divider can be devised. The divider shown in Fig. 7 consists of three binary rate multipliers and one reversible



At equilibrium:  $f_0 x = f_0 yz, x = yz, z = \frac{x}{y}$   
 $BM$  = binary multiplier

Fig. 7. Divider block diagram.

counter, connected as shown. Assume that it were desired to divide binary code  $x$  by binary code  $y$ . The  $x$  code is applied to a binary rate multiplier to generate a pulse train proportional to  $x$ . The  $y$  code is applied to a binary rate multiplier. The  $x$  pulse train is applied to the

forward input of the counter and the  $y$  pulse train to the backward input. In the counter  $z$  is the desired quotient, i.e.,  $z = x/y$ . The count  $z$  is applied to another binary rate multiplier to produce a pulse train proportional to  $z$ . This pulse train is multiplied by the  $y$  code input mentioned above, and the output of the binary rate multiplier is a pulse train proportional to the product of  $yz$ . This output is applied to the backward input of the counter.

Since  $z = x/y$  can be written  $x = yz$ , due to the feedback loop, the counter will tend to reach equilibrium with as many pulses applied to the forward input as applied to the backward input. At equilibrium the contents in the counter  $z$  will therefore be the desired quotient.

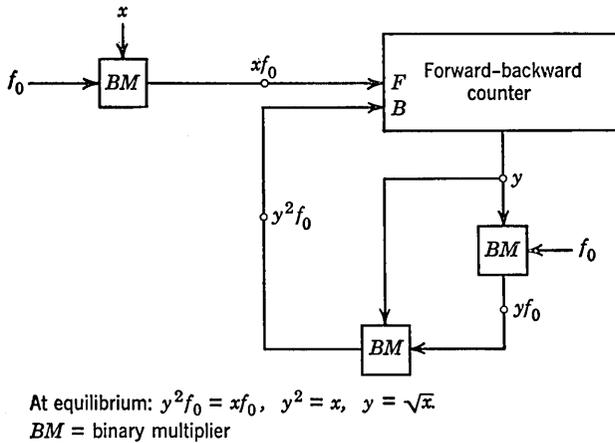


FIG. 8. Square root unit block diagram.

### Square Root

The ease with which algebraic and exponential relationships may be accommodated might best be illustrated by implementing a device to extract square roots with exactly the same circuit elements as just described for an operational digital divider. Refer to Fig. 8. Whereas in the divider, the output of the counter is multiplied by a rate  $f_0$ , and that product is multiplied again by an external function  $y$ , in the case of the square rooter, the output of the counter will be designated  $y$ , and applied simultaneously to the two subsequent binary rate multipliers. The output of the first multiplier is  $yf_0$ . The second rate multiplier is that product multiplied by  $y$ , or  $y^2f_0$ . When this feedback is applied to the backward count, while  $xf_0$  is applied to the forward count, equilibrium will be achieved in the counter when

$$xf_0 = y^2f_0 \quad \text{or} \quad y = \sqrt{x}$$

A comparison of Fig. 8 with Fig. 7 illustrating the divider will show that the circuit configuration is identical with the exception of one lead, used to set  $z = y$ .

**Variations.** There are obvious variations to this setup. If, for example, a unitary-weighted pulse train were to be divided by a binary code, only two binary rate multipliers would be required, for the pulse train could be applied directly to one of the inputs to the counter. If the problem involved two unitary-weighted pulse trains, only one binary rate multiplier would be required. In such a case, one pulse train would be applied directly to the forward input to the counter and the  $y$  pulse train would be multiplied by the count in the counter  $z$ , and the product applied to the backward input to the counter.

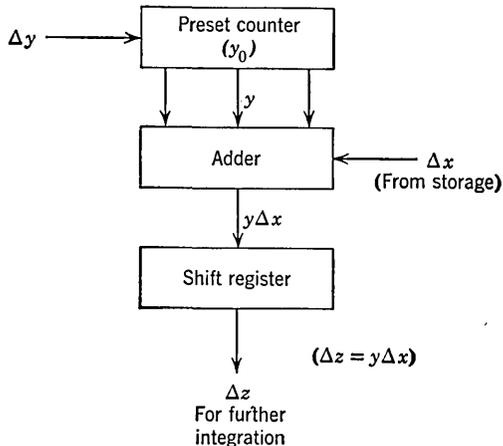


FIG. 9. Digital differential analyzer block diagram.

### Differential Analyzer

A description of the basic tools of operational digital techniques in hybrid operations would not be complete without reference to a familiar device, the digital differential analyzer (DDA). The DDA can be likened in many ways to an operational amplifier of the type commonly employed in analog computers, in that it can sum, integrate, and multiply. However, an analog operational amplifier can integrate only with respect to time. In the case of the DDA, the operational digital technique can be used to integrate some variable with respect to any other variable, clearly an important advantage in control or simulation applications.

A typical DDA is all digital in operation, i.e., all information flow between integrators is in the form of pulse trains. In integrating  $y$  with respect to  $x$  as shown in Fig. 9, the procedure is as follows. A counter

has the initial value of  $y$  preset into it. It is stepped by a unitary pulse train of  $\Delta y$  values. Associated with the counter is a serial adder which has a pulse input of  $\Delta x$  values. At the occurrence of a  $\Delta x$  pulse, the contents of the  $y$  counter is added to a second  $z$  counter, which puts out overflow pulses, when filled, having the value  $\Delta z$ . The rate of the  $\Delta z$  pulses is proportional to  $y$  and the rate of  $\Delta x$  pulses, such that  $\Delta z = y\Delta x$ .

If the  $\Delta z$  pulse train is counted in another similar integrator, the result is a step-by-step rectangular area integration of  $y$  with respect to  $x$ .

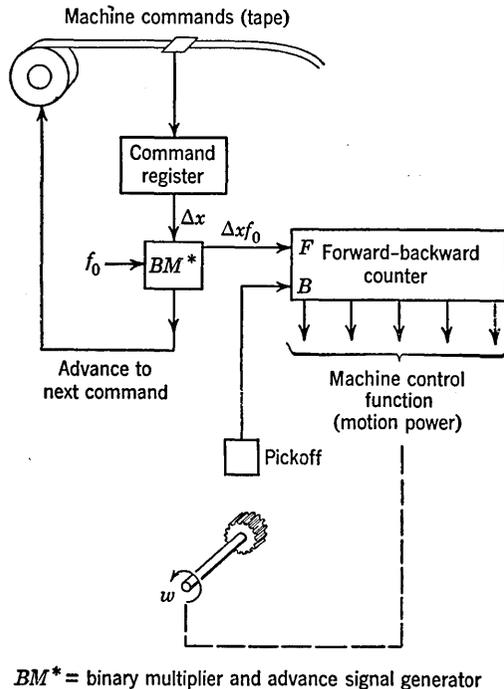


FIG. 10. Machine tool control diagram.

### 3. APPLICATIONS OF OPERATIONAL DIGITAL TECHNIQUES

#### Machine Tool Control

A practical application for use of the operational digital devices can be derived from one of the variations described above. In this application a machine tool, such as a milling machine or lathe, is to be automatically controlled by means of programmed commands from a tape mechanism. The device involves two unitary-weighted pulse trains, requiring but one rate multiplier and one forward-backward counter as shown in Fig. 10. The programmed commands  $\Delta x$  from the tape mech-

anism are fed to a binary rate multiplier via a command register. A fixed rate input,  $f_0$ , is applied to the other input of the rate multiplier. The commands from the tape mechanism modify the fixed pulse rate so that the output of the multiplier is a train of pulses,  $\Delta x f_0$ , proportional in number to the command increment. This pulse train is applied to the forward input  $F$  of the forward-backward counter in such a manner that the output of the counter is the machine control information, namely, the output provides the forward motion power for the machine.

As the machine moves in the direction indicated by this information, a pickoff transducer generates incremental motion pulses, which are fed to the backward input  $B$  of the forward-backward counter as the second unitary-weighted pulse train. Thus, the count in the reversible counter is always the difference between the machine commands and the actual position of the controlled element. The entire device is a closed loop positioning servo. It functions as a smoothing and rate control mechanism. The last pulse of a pulse group from the binary rate multiplier is used to advance the tape mechanisms so that the next command is then read into the command register.

### Information Averaging

In some industrial process control situations, (1) it is necessary to sample periodically and find the derivative for the rate of progress of the operation, and (2) it may be desirable also to integrate the rate information over a period of time, so as to yield an optimum approximation or prediction.

**Problem.** Figure 11a is a block diagram of a two-loop system where averaged output data  $x(t)$  are compared periodically with the sampled input data  $x(t_n)$ . The difference between the sampled input and the averaged output is applied to one integrator, is then combined with the difference information and, finally, is applied to a second integrator whose output is the averaged process state information. The difference values applied to both integrators are weighted by factors that indicate the importance that is to be given to the information involved and are appropriately chosen for the specific system. If it is recognized that a temperature-measuring device, for example, is not absolutely accurate, its full output will not be used to modify the system. Instead, its output will be weighted and only a fraction of it applied to the computing section. Thus, the averaged data integrator (the output integrator) not only gets process state corrections at the time of the sampling measurement but, of interest to the present discussion, continuously accumulates rate data and keeps up to date by extrapolating that rate information.

**Rate Aiding.** The problem exists of how to store accurately and utilize the rate information. As compared to analog or programmed

digital approaches, the operational digital technique for "rate-aiding" (as this averaging system is called) is extremely simple and requires only a single rate multiplier as shown in Fig. 11b. Assume that an integrated rate  $x$  is stored in a digital storage register. Physically, this might be a magnetic shift register or counter. Let the maximum possible process rate be represented by a continuous pulse train of rate  $f_{\max}$ . Then, as shown in the lower portion of the figure, the entire rate integrator consists of a single binary multiplier. As the code of the instantaneous rate is applied to the binary multiplier to modify the maximum

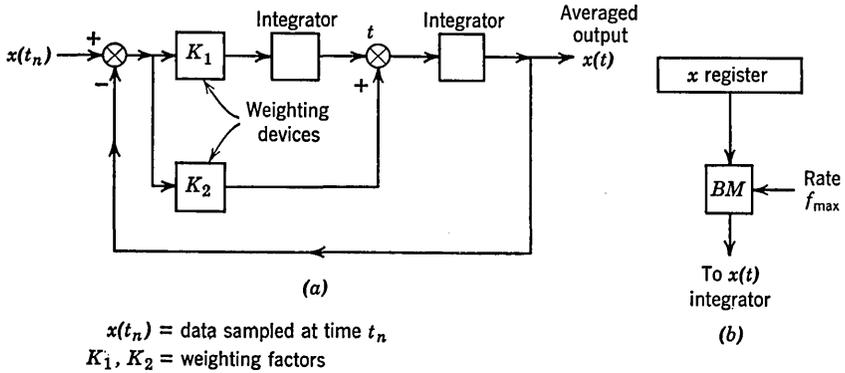


FIG. 11. Information-averaging system diagram.

rate  $f_{\max}$ , a new pulse train whose rate is proportional to the process rate is generated. The pulse outputs of this binary rate multiplier may now be applied to the extrapolating integrator so as to predict  $x(t)$  continuously.

**EXAMPLE.** This action may be better understood by considering a numerical example. Suppose that in a certain chemical process a temperature builds up toward a final critical value. The nature of the process is such that only periodic measurements of the temperature may be obtained and, indeed, these measurements are less accurate than are required. A rate-aided system is to be employed to average these sampled data and to provide continuous temperature information. For purposes of numerical example only, let the maximum rate of temperature change be  $1000^\circ\text{C}$  per minute. As the periodic measurements are carried out, the rate integrator will average itself to the proper instantaneous rate. It is required that this rate information be fed into the temperature integrator to provide, on a continuous basis, the instantaneous temperature of the reaction.

Let  $f_{\max}$  be equal to 1000 pulses per minute. Then, if the temperature

reaction is changing at its maximum possible rate, the stored binary code for 1000 will be .11111, and all the 1000 pulses per minute will be gated out of the binary rate multiplier to increase the averaged and predicted temperature at the proper rate.

Should the rate of reaction change so that the temperature is changing at half the rate of 500°C per minute, the binary rate code would then be .01111, and 500 ppm will be fed into the temperature-averaging integrator. For any other rate of change of temperature a proportional pulse rate will be gated through the binary multiplier, and the numerical value of the temperature will be continuously predicted and extrapolated, as desired.

**Other Applications.** This same technique has proved useful for a variety of position and velocity measurements, level and flow measurements, and other processes where periodically obtained data samples must be averaged to obtain more accurate information and to predict the state of the variables at the time when, for one reason or another, no actual measurements can be made.

#### 4. INCREMENTAL COMPUTATION

**Definition and Application.** The computer to be described is a hybrid computer. It is composed essentially of digital computer circuitry, and therefore achieves the high accuracy of a digital computer, and it operates in nearly real time, by means of incremental computation techniques, and therefore possesses one of the salient characteristics of the analog computer.

The industries which potentially could derive the most benefit from application of hybrid system techniques are the process industries. These industries are characterized by having large numbers of parameters which must be monitored for proper control of the process and, further, by the existence of relatively large inertias on all the parameters being monitored.

#### **Automatic Data Acquisition and Process Control**

**Requirements for Data Acquisition.** The typical organization of an electronic data acquisition system makes use of a rather sophisticated device, namely, an analog-to-digital converter, and multiplexes this converter to the various inputs. Since it is generally required that the digital information prepared by the analog-to-digital converter be in engineering units compatible with the parameter being measured, various normalization circuits are customarily interposed between the multiplexer and the converter and are programmed in accordance with the type of transducer being monitored for a variety of scale factors, linearization

constants, zero offset (or zero suppression) quantities, and square root. By whatever means the variations in normalization are achieved, the means to variability necessarily imply a device which is tantamount to storage.

**Storage Requirements.** The normal operation of an analog-to-digital converter involves a series of successive approximations toward a final measurement value and is performed much in the manner of the optimum program for weighing an object of completely unknown weight on a chemical balance. A distinguishing feature of the process industry systems is the fact that the parameters being monitored have large inertias. Therefore, a measurement of such a parameter is *not* in the category of the "unknown" weight, inasmuch as the measurement produced on an  $n$ th cycle of operation will be either equal to or have trivial deviation from the measurement on the  $n - 1$  cycle. Such systems must contain some forms of storage, such as mechanical inertias and relays. It is reasonable to consider a system organization in which a single storage unit is used for all the required normalization constants, alarm limit values, *and* the value of measurement on the  $n - 1$  cycle. Such a storage can be extremely simple in its organization inasmuch as the program sequence follows a well-ordered selection arrangement. Any storage device which presents information in a sequential fashion may be utilized without any requirement for complex storage selection circuitry or dead times imposed while waiting for a desired selection.

**Analog-to-Digital Converters for Incremental Systems.** The analog-to-digital converter is now called upon to make only one decision per measurement, namely, the determination of whether the parameter being measured is more or less than its last value. The output of the comparator of the converter, which was previously sensed on each of a succession of decision cycles, is now used to activate a simple incremental adder for the purpose of updating the stored measurement value. A system organized on this basis is inherently faster than its conventional counterpart and in many applications may result in significant economies.

**Control Applications.** However, the present intent is not to show a better way of doing a conventional operation but, rather, to explore the possibilities of this approach to the construction of systems of much higher ability. The fundamental intent of any data monitoring system is the evolution of control information. Inasmuch as many process control variables are of small relative magnitude, repeated output of data on such channels is often extraneous. The important characteristics of a true information-gathering system are:

1. The ability to distinguish information from redundancy.
2. The inclusion of sufficient computing ability to interrelate monitored

quantities in whatever manners are necessary to provide controlled information in a form suitable for immediate control action by the system operator or by the system itself.

Many of the mathematical operations required in process control computations are surprisingly complex. But the same remarks which applied to the use of old measurement data in the simplification of a new measurement have equal significance to the solution of these equations. The precise solution of the control equations by conventional computing means is exactly analogous to conventional analog-to-digital converter operation. Where nothing is known about the values of the various operands until they appear on the scene from storage, complex computing circuitry must be employed and the evolution of the answer must await the many cycles required in their operation. In contrast, the process control situations frequently involve the repeated call for solution of equations where both the operands *and* the answer are predictable within close limits. The job of the incremental computer, therefore, is to update a last solution rather than to start fresh on the effectively blind solution of a "new" problem.

### Machine Organization

**Block Diagram.** The computational unit shown in Fig. 12 comprises three adders, three incremental adders ( $\pm 1$ ), and three complementers. This apparatus, when connected together, performs the arithmetic given in eqs. (1), (2), and (3) of Table 1. On each storage cycle it receives operands  $S$ ,  $R$ ,  $V$ , and  $U$  from storage, operates on them in the manner indicated, and returns new values of  $R$ ,  $V$ , and  $U$  to the corresponding storage positions. The quantities  $S$  are used as scaling factors and are never modified except by change of the computational program. The complete equations for the incremental computer are given in Table 1 and are grouped as follows:

1-3	Basic
4-12	Addition and subtraction
13-25	Multiplication
26-32	Division
33-45	Square root
46-59	Integration and differentiation
60-64	Logarithm and exponential

**Control.** Control of the basic equations to effect the desired operations of addition, multiplication, square root, etc., is done by properly combining (or disconnecting) the sensing lines from incremental storage, and by the choice of which one relates to the dependent variable. In

most cases the dependent variable is not actually generated by the main storage cycle but, rather, information is placed in incremental storage which *may* be used to update the "result" in a separate storage cycle. In most cases the output increment of one program step will be applied directly as an input to a subsequent operation, and only the final solution will be generated for readout.

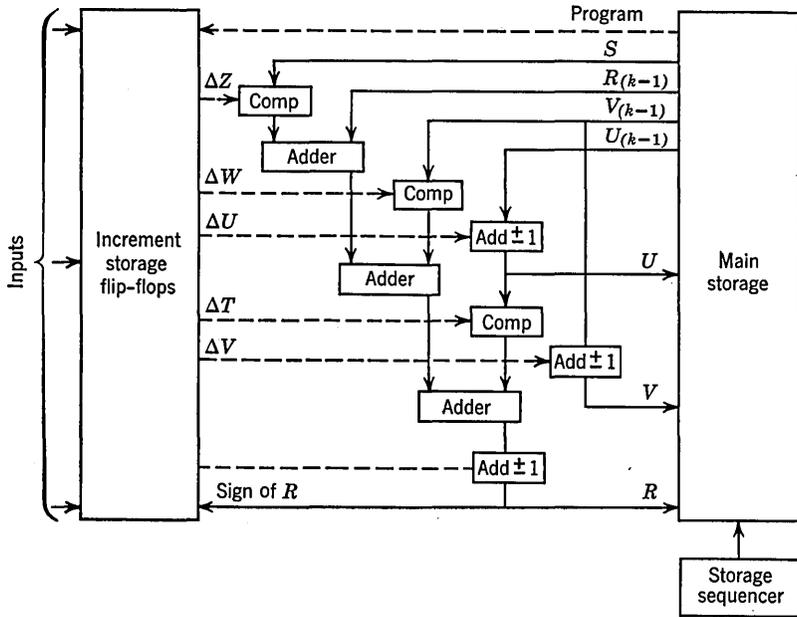


FIG. 12. Incremental computer block diagram.

Thus, the increment storage is the means for communicating information between parts of a problem solution, the mechanism for causing the basic machine algorithm to do different sorts of arithmetic, and the channel for input information.

**Program.** The program which accomplishes all this is stored in the main storage and is presented to the increment storage at each step of the storage cycle. Since the program command must be known in advance of the operands, the first bits read out from the four storages at the beginning of a cycle may be used for the program to control that cycle. The main storage may be a drum or an array of magnetic cores.

Assume eight binary bits as sufficient for command; then the first two pulse positions on the  $R$ ,  $S$ ,  $V$ , and  $U$  output lines will contain this information. A word length of thirty bits, plus one for sign, is probably

TABLE 1. EQUATIONS FOR THE INCREMENTAL COMPUTER OF FIG. 12

I Basic Machine Equations

- (1)  $U_k = U_{k-1} + \Delta U_k$
- (2)  $V_k = V_{k-1} + \Delta V_k$
- (3)  $R_k = R_{k-1} + U_k \Delta T_k + V_{k-1} \Delta W_k + \Delta X_k - S \Delta Z_k$

II Addition and Subtraction

A. Desired Output

(4) 
$$Z = \frac{X + U_0 T + V_0 W}{S}$$

B. Initial Conditions

- (5)  $\Delta U_k = \Delta V_k = R_0 = 0$
- (6)  $SZ_0 = X_0 + U_0 T_0 + V_0 W_0$
- (7)  $\Delta Z_{k+1} = \begin{cases} +1 & \text{for } R_k \text{ positive} \\ -1 & \text{for } R_k \text{ negative} \end{cases}$  reversed for  $S$  negative

C. Proof

- (8)  $R_k = R_{k-1} + U_0 \Delta T_k + V_0 \Delta W_k + \Delta X_k - S \Delta Z_k$
- (9)  $\sum_{k=0}^{k=N} (R_k - R_{k-1}) = \sum_{k=0}^{k=N} U_0 \Delta T_k + \sum_{k=0}^{k=N} V_0 \Delta W_k + \sum \Delta X_k - \sum_{k=0}^{k=N} S \Delta Z_k$
- (10)  $R_N - \cancel{R_0} = U_0 T_N - \cancel{U_0 T_0} + V_0 W_N - \cancel{V_0 W_0} + X_N - \cancel{X_0} - SZ_N + \cancel{SZ_0}$
- (11)  $SZ_N + R_N = U_0 T_N + V_0 W_N + X_N$
- (12)  $Z_N + \frac{R_N}{S} = \frac{X + U_0 T_N + V_0 W_N}{S}$

III Multiplication

A. Desired Output

(13) 
$$Z = \frac{UV + X}{S}$$

B. Initial Conditions

- (14)  $\Delta T_k = \Delta V_k$
- (15)  $\Delta W_k = \Delta U_k$
- (16)  $R_0 = 0$
- (17)  $SZ_0 = U_0 V_0 + X_0$
- (18)  $\Delta Z_{k+1} = \begin{cases} +1 & \text{for } R_k \text{ positive} \\ -1 & \text{for } R_k \text{ negative} \end{cases}$  reversed for  $S$  negative

C. Proof

- (19)  $R_k = R_{k-1} + U_k \Delta V_k + V_{k-1} \Delta U_k + \Delta X_k - S \Delta Z_k$

TABLE 1. EQUATIONS FOR THE INCREMENTAL COMPUTER OF FIG. 12  
(Continued)

$$\begin{aligned}
 (20) \quad \Delta(U_k V_k) &= U_k V_k - U_{k-1} V_{k-1} \\
 &= (U_{k-1} + \Delta U_k)(V_{k-1} + \Delta V_k) - U_{k-1} V_{k-1} \\
 &= \cancel{U_{k-1} V_{k-1}} + U_{k-1} \Delta V_k + V_{k-1} \Delta U_k + \Delta U_k \Delta V_k - \cancel{U_{k-1} V_{k-1}} \\
 &= V_{k-1} \Delta U_k + \Delta V_k (U_{k-1} + \Delta U_k) \\
 &= V_{k-1} \Delta U_k + U_k \Delta V_k
 \end{aligned}$$

$$(21) \quad R_k - R_{k-1} = \Delta(U_k V_k) + \Delta X_k - S \Delta Z_k$$

$$(22) \quad \sum_{k=1}^{k=N} (R_k - R_{k-1}) = \sum_{k=0}^{k=N} \Delta(U_k V_k) + \sum_{k=0}^{k=N} \Delta X_k - \sum_{k=0}^{k=N} S \Delta Z_k$$

$$(23) \quad R_N - \cancel{R_0} = U_N V_N - \cancel{U_0 V_0} + X_N - \cancel{X_0} - S Z_N + \cancel{S Z_0}$$

$$(24) \quad S Z_N + R_N = U_N V_N + X_N$$

$$(25) \quad Z_N + \frac{R_N}{S} = \frac{U_N V_N + X_N}{S}$$

#### IV Division

##### A. Desired Output

$$(26) \quad U = \frac{SZ - X}{V}$$

##### B. Initial Conditions

$$(27) \quad \Delta T_k = \Delta V_k$$

$$(28) \quad \Delta W_k = \Delta U_k$$

$$(29) \quad R_0 = 0$$

$$(30) \quad U_0 V_0 = S Z_0 - X_0$$

$$(31) \quad \Delta U_{k+1} = \begin{cases} +1 & \text{for } R_k \text{ positive and } V_k \text{ negative} \\ +1 & \text{for } R_k \text{ negative and } V_k \text{ positive} \\ -1 & \text{for } R_k \text{ positive and } V_k \text{ positive} \\ -1 & \text{for } R_k \text{ negative and } V_k \text{ negative} \end{cases}$$

##### C. Proof

$$(32) \quad \text{Solving eq. (24) for } U_N,$$

$$U_N = \frac{S Z_N + R_N - X_N}{V_N}$$

#### V Square Root

##### A. Desired Output

$$(33) \quad V = \sqrt{SZ - X}$$

##### B. Initial Conditions

$$(34) \quad V_0 = U_0$$

$$(35) \quad \Delta T_k = \Delta W_k = \Delta V_k = \Delta U_k$$

TABLE 1. EQUATIONS FOR THE INCREMENTAL COMPUTER OF FIG. 12  
(Continued)

$$(36) \quad R_0 = 0$$

$$(37) \quad U_0^2 = SZ_0 - X_0$$

$$(38) \quad \Delta U_{k+1} = \begin{cases} +1 & \text{for } R_k \text{ negative} \\ -1 & \text{for } R_k \text{ positive} \end{cases}$$

C. Proof

$$(39) \quad R_k = R_{k-1} + U_k \Delta U_k + U_{k-1} \Delta U_k + \Delta X_k - S \Delta Z_k$$

$$(40) \quad \begin{aligned} \Delta U_k^2 &= U_k^2 - U_{k-1}^2 \\ &= (U_{k-1} + \Delta U_k)^2 - U_{k-1}^2 \\ &= \cancel{U_{k-1}^2} + \Delta U_k U_{k-1} + \Delta U_k U_{k-1} + \Delta U_k \Delta U_k - \cancel{U_{k-1}^2} \\ &= U_{k-1} \Delta U_k + \Delta U_k (U_{k-1} + \Delta U_k) \\ &= U_{k-1} \Delta U_k + U_k \Delta U_k \end{aligned}$$

$$(41) \quad R_k - R_{k-1} = \Delta(U_k^2) + \Delta X_k - S \Delta Z_k$$

$$(42) \quad \sum_{k=0}^{k=N} (R_k - R_{k-1}) = \sum_{k=0}^{k=N} \Delta(U_k^2) + \sum_{k=0}^{k=N} \Delta X_k - S \sum_{k=0}^{k=N} \Delta Z_k$$

$$(43) \quad R_N - R_0 = U_N^2 - U_0^2 + X_N - X_0 - SZ_N + SZ_0$$

$$(44) \quad R_N = U_N^2 + X_N - SZ_N$$

$$(45) \quad U_N = \sqrt{SZ_N - X_N + R_N}$$

VI Integration

A. Desired Output

$$(46) \quad Z = \frac{2}{S} \int U dT$$

$$(47) \quad Z_N = \frac{2}{S} \sum_{k=0}^{k=N} \frac{1}{2} (U_k + U_{k-1}) \Delta T_k \cong \frac{2}{S} \int U dT$$

B. Initial Conditions

$$(48) \quad V_0 = U_0$$

$$(49) \quad \Delta V_k = \Delta U_k$$

$$(50) \quad \Delta W_k = \Delta T_k$$

$$(51) \quad R_0 = X_0 = \Delta X_k = 0$$

$$(52) \quad Z_0 = \text{desired value at } T = T_0$$

$$(53) \quad \Delta Z_{k+1} = \begin{cases} +1 & \text{for } R_k \text{ positive} \\ -1 & \text{for } R_k \text{ negative} \end{cases}$$

C. Proof

$$(54) \quad R_k = R_{k-1} + U_k \Delta T_k + U_{k-1} \Delta T_k - S \Delta Z_k$$

TABLE 1. EQUATIONS FOR THE INCREMENTAL COMPUTER OF FIG. 12  
(Continued)

$$(55) \quad \sum_{k=1}^N (R_k - R_{k-1}) = \sum_{k=1}^N (U_k - U_{k-1})\Delta T_k - S \sum_{k=1}^N \Delta Z_k$$

$$(56) \quad R_N - R_0 = 2 \sum_{k=1}^N \frac{1}{2}(U_k + U_{k-1})\Delta T_k - SZ_N + SZ_0$$

$$(57) \quad Z_N + \frac{R_N}{S} = \frac{2}{S} \sum_{k=1}^N \frac{1}{2}(U_k + U_{k-1})\Delta T_k + Z_0 \\ \cong \frac{2}{S} \int U dT + Z_0$$

VII *Differentiation*

$$(58) \quad U \cong \frac{S}{2} \frac{dZ}{dt} \text{ (substituting } \Delta U_{k+1} \text{ for } \Delta Z_{k+1} \text{ in eq. 53)}$$

VII *Reciprocal Integration*

$$(59) \quad T \cong \frac{S}{2} \int \frac{dZ}{U} \text{ (substituting } \Delta T_{k+1} \text{ for } \Delta Z_{k+1} \text{ in eq. 53)}$$

VIII *Natural Logarithms*

A. Desired Output

$$(60) \quad \frac{2}{S} T = \ln Z$$

B. Initial Conditions

$$(61) \quad Z = U$$

$$(62) \quad \Delta Z = \Delta U$$

$$(63) \quad R_0 = X_0 = \Delta X_k = 0$$

$$(64) \quad \Delta T_{k+1} = \begin{cases} +1 & \text{for } R_k \text{ positive} \\ -1 & \text{for } R_k \text{ negative} \end{cases}$$

C. Proof

Same as VII, since:

$$(65) \quad T = \frac{S}{2} \int \frac{dZ}{Z} = \frac{S}{2} \ln Z, \text{ whence}$$

$$(66) \quad \frac{2}{S} T = \ln Z$$

IX *Exponential*

$$(67) \quad Z = e^{2T/S} \text{ (substituting } \Delta Z_{k+1} \text{ for } \Delta T_{k+1} \text{ in eq. 64)}$$

sufficient for any application. The prototype will, therefore, be assumed to have a minor cycle length of thirty-three pulse times, and have a major cycle (complete storage cycle) which contains thirty-three ( $1092 + 33$ ) possible program steps. This seems sufficient, inasmuch as considerable arithmetic may be accomplished in each step. For example, a complete summation of the displacements in the thirty monitored lines requires only fifteen cycles.

**Error Term.** In all the machine operations the quantity  $R$  represents an error term which at any instant in a problem solution is either at its minimum possible value or is being reduced to that value at the maximum possible rate. The entire intent of the machine algorithm is to reduce  $R$  on each cycle of computation. The sign of the error may therefore be used as an input to the incremental storage such that, by appropriate action of the program, the next solution for  $R$  on that program step will either reverse its sign or reduce its magnitude. The incremental storage flip-flop which, by program control, receives its information from the sign of  $R$ , is the dependent variable of a desired operation.

## Operations

**I. Addition and Subtraction.** The computing unit is capable of continuously updating the sum of three quantities, according to eq. (4). The quantities  $X$ ,  $T$ , and  $W$  are presented to the arithmetic unit in the form of increments, and may be entirely derived from flip-flops which receive their information externally or come from a combination of such information with increments obtained from other computing processes. This mixed mode will become clearer in its operation as the description proceeds.

No explicit storage for the quantity  $Z$  exists. The sequencing described does not result in the production of an explicit quantity  $Z$  unless that is required by the program. Instead, the operation produces as an output information for updating a quantity  $Z$  by units of  $\pm 1$ . These units may be used to modify either  $U$  or  $V$  in a subsequent cycle of operation to accumulate a quantity equal to  $Z$ , if that is desired. But if, for example,  $Z$  is to be further added to other quantities, and is not required as a result in its own right, then  $\Delta Z$  will be sensed in a subsequent solution of eq. (4) as being a quantity  $\Delta X$ ,  $\Delta T$ , or  $\Delta W$ .

The quantity  $S$  is used in all the mathematical operations as a scaling factor. In addition and subtraction the quantities  $U$  and  $V$  are also available for scaling. However, the choice of  $S$ ,  $U$ , and  $V$  must be made in accordance with the requirements of eq. (6), which represents an initial solution of the problem at the start of any computational run. The computation proceeds as follows:

a. Equations (5), (6), and (7) are the microprogramming for the initial solution. Equation (5) says, simply, that the quantities  $U$  and  $V$  remain invariant and that the initial value of  $R$  in the storage is zero. Equation (6) states certain necessary balances of the numbers in the storage at the start of the program sequence.

b. Substituting the conditions of eq. (5) in eq. (3), leaves the modified arithmetic operation of eq. (8).

c. Summing for  $N$  cycles eq. (9) leads to eq. (10). Subtracting out eq. (6), and the fact that  $R$  is equal to zero (eq. 5) gives us eq. (11).

d. Equation (11) rewritten in eq. (12), is identical to eq. (4), except for the error term.

e. It can be shown that the solution is at all times within  $\pm 1$  of its correct value or is approaching that value at the maximum possible rate.

**2. Multiplication.** Multiplication is accomplished as follows. As before, the quantities which are independent variables ( $U$ ,  $V$ , and  $X$ ) are presented to the arithmetic unit in the form of increments. Unlike the previous example, two of the variables,  $U$  and  $V$ , are updated as a part of the operation, and could be called from the storage. The single operation thus provides a means for updating information for a product quantity  $Z$ . The sequence is as follows.

a. Equations (14) through (18) are the microprogramming equations which adapt the machine algorithm for multiplication. Equations (16) through (18) resemble start conditions for the addition process, while eqs. (14) and (15) merely say that two increment storage flip-flops serve as the controlling devices for four operations.

b. Substituting the initial conditions into eq. (3), yields eq. (19).

c. This becomes eq. (21) following the arithmetic of eq. (20).

d. Summing over  $N$  cycles and subtracting eqs. (16) and (17) leaves eq. (24).

e. The rewritten form in eq. (25) is equivalent to eq. (13), except for the error function, and can be shown to be either equal to  $\pm 1$  of the correct value or be moving toward that value at the maximum machine rate.

**3. Division.** The operation of division as expressed by eq. (26) is the same as that for multiplication except for the choice of dependent variable. With the exception of the rules given in eq. (31), all the microprogramming for the two operations is identical.

The solution of the machine equation for division follows the same lines as in multiplication to eq. (24). Solving eq. (24) for  $U_N$  yields the result, eq. (32). This is identical with eq. (26) except for the error term. It can be shown that the solution is within  $\pm 1$  increment of the correct value or is approaching the correct value at the maximum machine rate.

Division is the first of the machine operations thus far discussed which directly accumulates its result in the main storage. This property is true only for operations which use the quantity  $U$  or  $V$  as the dependent variable.

**4. Square Root.** Square root, according to the relationship of eq. (33), is performed by the computer by imposing the restrictions of eqs. (34) through (38). Note that while all six increment control lines are active, four of them are derived from a single increment storage flip-flop. The procedure is:

- a. Substituting eqs. (34) and (35) into eq. (19) yields eq. (39).
- b. Equation (39), following the arithmetic of eq. (40), becomes eq. (41).
- c. Summing over  $N$  cycles and subtracting the initial conditions of eqs. (36) and (37) give eq. (44).
- d. Rewritten as eq. (45) it may be seen that this is identical with eq. (33) except for the error term.
- e. It may be shown that  $U_N$  is within  $\pm 1$  increment of the correct solution or is approaching that value at the maximum machine rate.

Because of the choice of  $U$  as dependent variable, the quotient is accumulated directly in the main storage.

**5. Integration and Differentiation.** None of the arithmetic processes discussed in the preceding sections has involved approximations. The results in all cases have been exact and, presuming reasonable rates of change of the input variables, are at all times within  $\pm 1$  of true value.

The subject operations and those which follow are limited in their accuracy by certain approximations which have been required. The accuracy of the approximation depends on the behavior of the variables. In most cases the errors will be quite small. The procedure is:

- a. Integration according to the relationship shown in eq. (46) is performed in the arithmetic unit by the trapezoidal approximation shown in eq. (47).
- b. The microprogramming for this operation is given in eqs. (48) through (53).
- c. Substituting the initial conditions into eq. (3) yields eq. (54).
- d. The usual procedure of summing over  $N$  cycles and subtracting out initial conditions results in eq. (56).
- e. As rewritten in eq. (57), the result is equivalent to eq. (46), except for the error term and the approximation error of eq. (47).

Given the ability to integrate, it follows that the arithmetic organ can also differentiate or perform reciprocal integrand integration as shown in eqs. (58) and (59). These, of course, have the same limitations on accuracy as noted for the basic integration and operation.

These operations are handled identically with integration except for the choice of independent variables, as noted.

**6. Logarithm and Exponential.** With the aforementioned cautions on accuracy, eq. (59) may be modified according to the conditions of eqs. (60) and (61) to effect natural logarithms according to eq. (62). Further manipulation leads to eq. (64) for exponential.

### Practical Considerations

**Transient Solutions.** All the foregoing operations function to update their solutions on each cycle of the machine storage. At the beginning of machine operations these corrections are made from a trial solution stored in the machine storage. This solution may or may not be a good approximation, but in any case serves to relate the scaling factors used in the problem solution properly.

In a continuously operating system, where previous values of variables are not erased from the storage, there will never be a starting transient during which the "answers" differ appreciably from their true values. Readout may be initiated at any time and the solutions read out will represent the results of computation which has been performed on the very latest input data.

Assume that it were necessary to clear the counters once every 24 hours. If this method of operation were used with an incremental computing system, there would be a short period of time at the beginning of each day's run (probably in the order of seconds) during which the machine's solutions would not be valid. However, it is probable that the clear operation would no longer be required once the computational ability is built into the system.

**Incremental System Benefits.** The addition of incremental computing ability to a data gathering facility can provide a number of very important operating advantages.

1. System cost should be very much less than that originally intended, presuming that at least part of the cost of a medium-sized drum computer must be added to the basic data gathering facility.

2. System reliability should be greatly improved because no electro-mechanical devices are involved in the computation itself, and only the desired end results need be communicated to the operator.

3. System versatility offers numerous features not heretofore possible. One may, for example, have rate alarming at no additional cost other than the cost of the time for programming.

4. Typeout of any or all of the final results is available at any time during the day, either on demand readout or by automatic initiation at specified intervals.

5. The fact that most system inputs are analog in nature and that analog-to-digital conversion is required is no detriment whatsoever to the application of incremental techniques. Indeed, the analog-to-digital conversion process can be made very much easier by combining its operation with the computing unit. Analog-to-digital conversion as performed in present systems makes no use whatever of previous input values and must, therefore, make all decisions required for the digital quantity over again on each cycle of operation. But when previous values are available from a storage, the last updated digital value may be read out into a digital-to-analog converter for a present status comparison. The results of this comparison, in the form of an increment ( $\pm 1$ ) may then be used for further updating the digital quantity in the main storage.

### REFERENCES

1. B. M. Gordon, Adapting digital techniques for automatic control. I, *Elec. Mfg.*, 136-144, Nov. 1954.
2. B. M. Gordon, Adapting digital techniques for automatic control. II, *Elec. Mfg.*, 120-126, Dec. 1954.
3. C. T. Leondes and M. Nothman, Real time hybrid computers for control systems, *Proc. Am. Inst. Elec. Engrs. Conference on Computers in Control Systems*, Atlantic City, N. J., Oct. 16-18, 1957.
4. R. Leger and J. Greenstein, Simulate digitally, or by combining analog and digital computing facilities, *Control Eng.*, **3**, 145-153 (1956).
5. J. H. McLeod and R. M. Leger, Combined analog and digital systems—Why, when, and how, *Instr. and Automation*, **30**, 1126-1130 (1957).
6. B. K. Smith, Practical information theory aspects of high-speed data handling, *Trans. I.R.E., Prof. Group on Information Theory*, March 1958.
7. B. Gordon and R. Niccola, Special purpose digital data processing computers, *Proc. Assoc. Computing Machinery*, 33-45, Pittsburgh, May 2-3, 1952.



## Combined Analog-Digital Computing Systems

*George P. West*

1. Description and Applications	30-01
2. System Components	30-02
3. Control and Timing	30-08
4. Modes of Operation	30-13
References	30-15

### 1. DESCRIPTION AND APPLICATIONS

A *combined system* consists of analog computation elements and a digital computer interconnected by transducers and simultaneously employed in computation. A typical combined system comprises:

1. A general purpose digital computer of moderate to high computation speed.

2. Electronic analog computing equipment of the low frequency, or nonrecursive type, in amounts which vary from a few elements to several racks of equipment.

3. The transducers are usually electronic analog-to-digital and digital-to-analog converters.

A typical system is shown in block diagram form in Fig. 1.

*Note.* Throughout this chapter the term bit will be used as an abbreviation for binary digit.

**System Applications.** Combined systems are employed to solve systems of differential equations which possess one or more of the following characteristics:

1. Exceed the capacity of the largest digital computers in that they require an excessive period of computation to produce results.

2. Exceed the capacity of the analog computation facility usually by requiring more nonlinear elements than are available.

3. Require greater accuracy than is obtainable by analog techniques.

**EXAMPLE.** A system of ordinary differential equations of moderate order and containing many nonlinear coefficients will quite often require more nonlinear elements than are available in a large analog computer. If the system response contains both high- and low-frequency effects

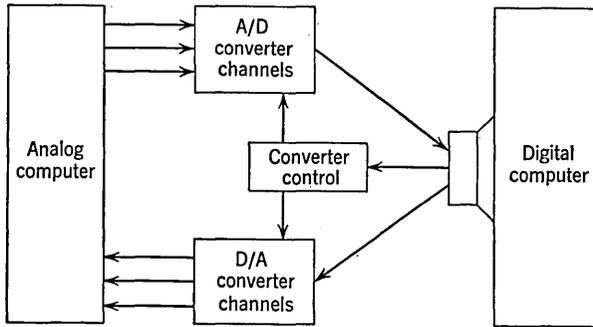


FIG. 1. Block diagram of a combined analog-digital computing facility.

which are of interest, the problem may exceed the capacity of a high-speed digital computer. For the digital computer must employ a computation interval small enough to reproduce faithfully the high-frequency effects and at the same time carry the computation forward over many of these small intervals until the low-frequency effects can be observed.

If additional computing capacity is sought by attempting to employ two digital computers simultaneously, the problems of coordinating the actions of the two computers, so that they do not interfere with one another while attempting to communicate results, is a difficult one. Furthermore, programming the computation to keep both machines occupied is awkward. Although the combined analog-digital approach requires a radical change in the number representation for communication between the two computers, problems of mutual interference and of programming for effective utilization of both machines are much simpler.

**Problem Types.** Table 1 contrasts problem types successfully employing analog, digital, and combined techniques for their solution.

## 2. SYSTEM COMPONENTS

**Analog Computer.** The low-frequency electronic analog computer performs all computations simultaneously so that, within the restrictions

TABLE 1. COMPARISON OF PROBLEM TYPES FOR ANALOG, DIGITAL, AND COMBINED COMPUTERS

Analog	Digital	Combined
Linear and nonlinear ordinary differential equations of moderate order with simple nonlinear terms and with both high- and low-frequency effects in response	Linear and nonlinear differential equations of moderate order with complex nonlinear coefficients and with either high- or low-frequency response characteristic	Linear and nonlinear differential equations of high order with complex nonlinear coefficients and with both high- and low-frequency effects in system response
Continuous control system studies requiring about 0.1% accuracy in solution	Continuous or discrete control system studies of moderate complexity requiring better than 0.1% accuracy in solution	Highly complex control systems containing discrete as well as continuous elements and requiring accuracy of better than 0.1% in solution
Computations requiring graphical input data and producing graphical results	Computations requiring tabular input data and producing tabular or graphical results	Computations requiring graphical and tabular input data and producing graphical and tabular results

imposed by the number of computing elements available in the computer, problem complexity does not restrict solution time. These analog computers can furnish solutions to about 0.1 per cent accuracy provided the response does not exceed about 20 cycles per second. Problem changes are easy to effect by plugboard changes or by resetting potentiometers. Input and output of graphical data is easy and straightforward. Output graphical data must be manually identified and scaled.

**Digital Computer.** The high-speed digital computers perform computations sequentially so that computation time depends on the complexity of the problem. The computation time determines the highest sampling frequency possible and, therefore, the highest frequency of the system response which can be accommodated. A high-speed digital computer may well require 25 milliseconds to compute the sine or cosine of six angles so that a coordinate transformation exceeds the computer capacity if 20-cycle variations in position occur. Digital computers handle logical decisions very effectively. They accept tabular data for input and produce tabular output as well as graphical output with identification and scales. Graphical output requires special plotting symbols on the printer as well as the usual alphanumeric characters.

### Conversion Equipment

**Mechanical Analog-Digital Converters.** There are many types of analog-digital conversion equipment in use (Refs. 1, 2). The mechanical converters are usually shaft position converters in which a servo positions a code wheel to establish the required digital code, or relay switching networks which switch increments of voltage until the input voltage is matched. Mechanical converters are normally too slow for use in a combined system.

**Electronic Analog-Digital Converters.** The converters most used in combined systems employ vacuum tube or transistor switching circuits to switch increments of current into a ladder network which sums the voltage drops and adds to or subtracts current increments until the output of the ladder network equals the input voltage (Refs. 3, 4). These converters are of two types; the free running or servo type, and the sampling type. Another type of electronic analog converter which has not been employed in combined systems is the pulse width system.

1. *Free Running.* In this type, the current switches are controlled by the bits in an accumulating register. A least count is added to or subtracted from the register, depending on whether the ladder output is greater or less than the input, at equal intervals of time, until the register counts to a value corresponding to the input voltage. The accumulator then oscillates within plus or minus one least count of the input value. The maximum voltage change that the converter can follow is the voltage change produced by the least significant digit divided by the clock period between successive additions. The dynamic response can be improved by sensing whether the comparator is out of balance by more than one voltage increment and when it is causing the bit to add in the next to least significant position. In this way, the converter follows twice as rapid a voltage change but to one-half accuracy.

2. *Sampling.* Another type of electronic converter starts from zero and switches successively from the most significant bit to the least, each time comparing the input to the ladder output. In this way, the converter establishes a number which was equal to the input voltage at some time during the sampling interval. This value is held until the converter is commanded to sample again. By making the decision interval short, the input changes during sampling are made small.

A variation of this converter employs a sample and hold circuit to clamp the input so that it is constant during conversion. The problems inherent in making a fast sample and hold circuit are nearly as great as in producing a fast converter; however, sample and hold devices can improve the accuracy of this type of converter by at least a factor of 10.

3. *Pulse Width.* The pulse width converter first transforms the analog voltage into a pulse with duration proportional to the analog voltage (Ref. 5). The pulse duration then gates a string of pulses into a counter where they are counted to arrive at the digital value of the analog voltage. Even with high pulse rates (2 or 3 megacycles), the time required to count to 3-decimal digit accuracy (400 microseconds) is longer than for other electronic analog converters. The pulse width converter is slower than the electronic converters of the sampling or free running types.

**Digital-Analog Conversion.** Conversion from digital to analog is much simpler than the corresponding conversion from analog to digital. The digital number is set up in a register. The bits in the register control the switching of current increments into a ladder network which sums the voltage drops to produce an output proportional to the number placed in the register. The digital-to-analog conversion does not require comparisons and decisions in order to set the current switches properly. Instead the switches are simply set to agree with the bits of the number in the register. Furthermore, the switches are set simultaneously rather than sequentially so that conversion is much faster digital to analog than it is analog to digital.

**Converter Buffering.** The construction of electronic analog-digital and digital-analog converters is such that a buffering action is obtained at no extra cost. In the sampling type analog-digital converter, once the current switches are set, the number can be read out at any time between conversions. Since the current switch settings will remain constant until the next conversion, they may be sensed at any time for readout, and until sensed they will store the number. Buffering action of the free running converter is obtained by stopping the decision pulses so that the converter no longer follows the input. When the number has been transferred from the buffer, a subsequent conversion must be delayed until the free running converter has had time to catch up to the input voltage. In the same way, the digital-to-analog converter can accept a number at any time for conversion. Once the current switches are set, the analog output will hold constant while the register accepts another number for subsequent conversion.

**EXAMPLE.** A typical electronic analog-to-digital converter will occupy from 6 to 28 inches of rack space, weigh from 50 to 200 pounds, and require from 25 to 500 watts of power. The converter can be obtained in sizes from 10 bits to 18 bits with options as to whether binary or coded decimal representation is preferred. The static accuracy is about 0.1 per cent. The conversion speed will vary from 2 to 5 microseconds per bit.

### Converter Bandwidth.

1. *Free Running.* The highest frequency a free-running converter can follow is that with a maximum slope of one least count in the time required to set a bit.

EXAMPLE. If the bit setting time is 4 microseconds and the converter has a 10-bit register, the maximum slope would be one least count ( $2^{-10}$ ) divided by the time to set the bit  $4(10^{-6})$ , and this in turn must equal the maximum slope  $2\pi f_{\max}$  of the sine wave of maximum frequency. So that,

$$f_{\max} = \frac{2^{-10}/4(10^{-6})}{2\pi} = 39 \text{ cycles per second.}$$

2. *Sampling.* The sampling converter must sample the voltage before it has changed appreciably. The sampling process takes a finite length of time. It is the speed of conversion, the time required to set the desired number of bits, which determines the converter bandwidth and not the number of sampling operations which can be performed in a given amount of time. Most converters recover very rapidly so that data can be sampled at a very high rate. To obtain appreciable accuracy from any of the samples, the input must be of quite low frequency content so that the additional samples which could be taken would be redundant and of little value.

EXAMPLE. What is the highest frequency that a converter with a speed of 4 microseconds per bit can sample to 10-bit accuracy? For 10-bit accuracy, the input must not change more than one bit in the least significant place. The maximum change is then  $2^{-10}$ . This change can occur in the length of time required to set 10 bits, or in 40 microseconds. The maximum slope  $2\pi f_{\max}$  of the sine wave of maximum frequency for 10-bit conversion must equal the maximum allowed change during conversion. So that,

$$f_{\max} = \frac{2^{-10}/40(10^{-6})}{2\pi} = 3.9 \text{ cycles per second.}$$

### Improving Bandwidth by Computation.

1. *Free Running.* The example above showed that a 4-microsecond free-running converter could follow about 40-cycle data to full 10-bit accuracy. If data are transferred twice per cycle to the digital computer, then at 40 cycles per second, 2 samples per cycle, and 10 bits per sample, 800 bits per second are transferred to the computer. The converter makes 250,000 one-bit decisions each second in following the input, which requires a single decision every 4 microseconds.

The bandwidth can be increased somewhat by reading data which are only approximate from the converter and by using the digital computer program to recover the data from the noise introduced by conversion. It is not possible to increase the bandwidth of the free-running converter very much by these techniques since the converter is either following the input voltage or it is not. Consequently, there are 10 bits of information in each sample, or none. The converter will catch up at the peak values and can provide accurate readings at the crests of the waveform even if it cannot keep up on the steep rise at either side. These peak values can be used to reconstruct the wave. The program can decide whether readings are meaningful or not by differencing to see if the converter was traversing at full speed or if it had settled down to following the input. However, if the input is much above the 40-cycle limit in harmonic content, the converter will produce a triangular waveform intersecting the data waveform more or less randomly. Since the converter is traversing at full speed at all times, it is difficult to determine the intersections with the data and hence the meaningful values.

2. *Sampling.* The example above showed that a 10-bit sampling converter with a conversion time of 40 microseconds has a bandwidth of about 4 cycles per second. If the converter can recover and be ready to sample within 100 microseconds (many sampling converters can), it would be possible to sample an input every 100 microseconds. Each such sample would require ten decisions in setting the bits for the conversion. In theory then, the converter is capable of making 100,000 decisions per second. However, in transferring 4-cycle data to a computer to 10-bit precision, it is in fact delivering only 80 bits per second, an even lower efficiency than that noted for the free-running converter. The sampling converter, however, can accept a much greater range of frequencies at its input before it reaches the point of delivering no information with each sample. Since the sampling converter sets the most significant digit first, the input can change very rapidly before the converter will fail to set the most significant bit correctly. For example, a 1000-cycle signal would cause at least 4 bits to be set properly. If the high-frequency input is sampled more than the ideal two samples per cycle and a least square fit is made to the data in the computer, it is possible to reconstitute very high-frequency data from its approximate sample values. Since the digital computer is often hard put to complete the required computations in the time available, there may be little time left for reconstituting the input data from its approximate samples.

3. *Pulse Width.* The output of any converter can be considered as representing precisely the value which the input voltage assumed at

some time in the sampling interval. Usually the exact time at which this coincidence occurred is not known. If the converter is fast enough, this timing uncertainty is not important. If the coincidence is assumed to occur at the start of the sampling interval, an amplitude error results which can be held to prescribed limits by restricting the bandwidth of the input voltage being converted.

The pulse width converter has a relatively long conversion interval but its design is such that it is quite easy to compute the instant at which sampling occurred. If the perturbations of the nominal sampling interval are computed, the converter output can be accepted at full design accuracy. The counter in a pulse width converter is gated on as a standard ramp voltage crosses zero and off when the ramp reaches the input voltage level.

The converter output is therefore proportional to the time from the start of the sampling interval until the coincidence with the input voltage occurred. This proportionality affords a simple method of computing the sampling instant.

**Converter Selection.** In spite of the bandwidth advantage of the free-running converter, most combined systems use the sampling converter. This is because the sampling converter recovers more rapidly after serving as a buffer than does the free-running converter which must have time to catch up to the input voltage. While this recovery time has little effect on the bandwidth of either converter, it provides a better buffering action for the sampling converter. The improved buffering action is important from the computer programming point of view.

### 3. CONTROL AND TIMING

**Definitions.** In combined systems, the digital computer accepts inputs from the analog computer and computes results which are transmitted back to the analog computer. The time interval between two successive transfers of the same output variable to the analog computer will be called the *cycle time*. The time interval between accepting data and delivering results will be called the *delay time*.

**Program Control.** For many digital computers, the time required to complete a program will vary from computation to computation, since the execution time of operations such as multiply depends on the magnitudes of the numbers being used in the computation. The variation in computation time for some digital computers could amount to  $\pm 10$  per cent or more. If all conversions from analog to digital or digital to analog were initiated under program control, these variations in program execution time would produce corresponding variations in the cycle times and

delay times for each computation. For many system studies, variations in cycle time produce effects not present in the system under simulation and their net effect on the simulation is difficult to estimate in assessing the value of the simulation study. These uncertainties can be eliminated by reducing the simulation time scale by 10 per cent and inserting sufficient idle time in the digital program to pad out each computation to a fixed length. In order to do this easily, the digital computer must provide interlocks with its input-output buffer.

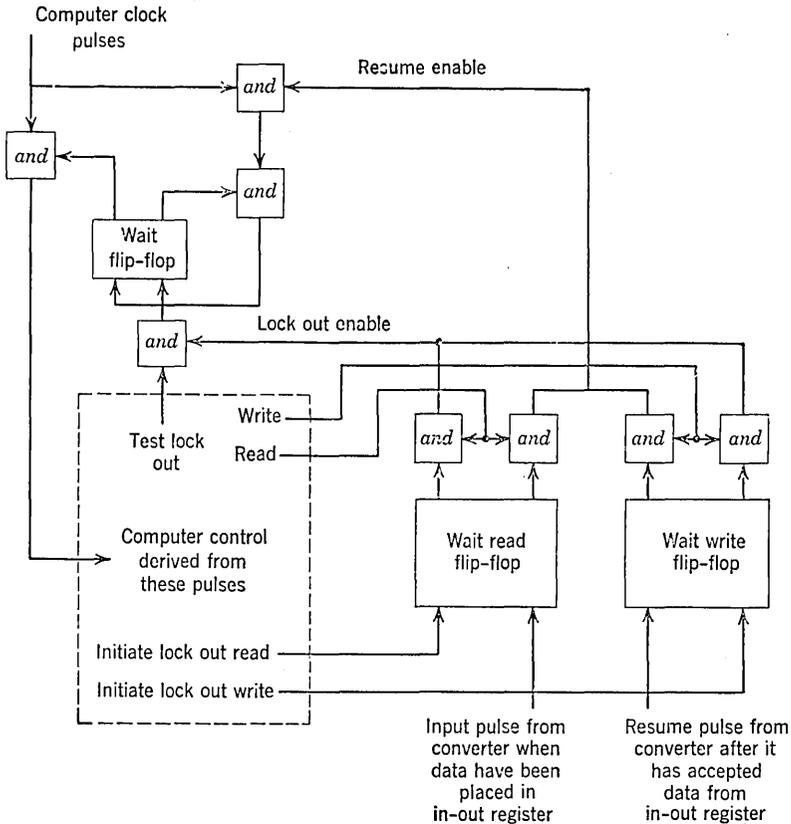


FIG. 2. Block diagram of input-output buffer and interlocks.

**Input-Output Buffer and Interlocks.** Many of the commercially available digital computers either provide or can be modified to provide an interlocked in-out register, such as that shown in Fig. 2. The interlocks halt the digital computer if it attempts to read data from the in-out register before the external equipment has placed the data in the register.

The interlocks are released as soon as data have been placed in the register at which time computation resumes.

Furthermore, the conversion equipment must be provided with a clock which can be manually set to control conversion at a fixed rate, independent of the operation of the digital computer. With the conversion interval set slightly longer than the maximum execution time of the digital computer program, the interlocks will hold the digital computation waiting for the converter clock to transfer data and release the interlocks. In this way, each segment of computation is started in synchronism with the converter clock. Where different computations, each with its own cycle time, must be interleaved, the same techniques are available.

**Sequencing Computations.** Suppose a computation  $X$  is to be performed each conversion interval  $T$ , while a computation  $Y$  is to be performed once in each double interval,  $2T$ . (See Fig. 3a, b.) Note that  $T$  is in seconds of problem time. This can be accomplished if the digital program is written to convert A/D and D/A as follows:

1. Read and convert data A/D for the first computations  $X_1$  and  $Y_1$ .
2. Compute  $X_1$ .
3. Write  $X_1$  results to the converter buffer.
4. Complete half of the  $Y_1$  computation.
5. Convert  $X_2$  data A/D and  $X_1$  results D/A.
6. Read in data for computation  $X_2$ .
7. Write  $X_2$  results to converter.
8. Complete  $Y_1$  computation.
9. Write  $Y_1$  results to converter.
10. Convert  $Y_1$  and  $X_2$  D/A,  $Y_2$  and  $X_3$  A/D, and repeat the cycle.

This sequence completes two  $X$  calculations ( $X_1$  and  $X_2$ ) and one  $Y$  calculation ( $Y_1$ ) in the interval  $2T$ . (See Fig. 3c.)

The analog computer time scale is adjusted so that  $2T$  seconds of problem time equal the nominal execution time of the entire digital computer program. The digital computer program will control conversions and provide the computations  $X$  and  $Y$  with approximate cycle time of  $T$  and  $2T$  seconds of problem time. These cycle times will not be exactly periodic as pointed out earlier. The cycle times can be made exactly periodic by setting the analog computer time scale so that the maximum execution time of one cycle of the digital program is equal to  $2T$  seconds of problem time. The conversions must also be placed under the control of the external clock which is set to convert every  $T$  seconds. In this way, the interlocks on the input-output register insert sufficient idle time at each conversion to accommodate the uncertainty in execution time of the digital computer program. (See Fig. 3d.) Note that while the

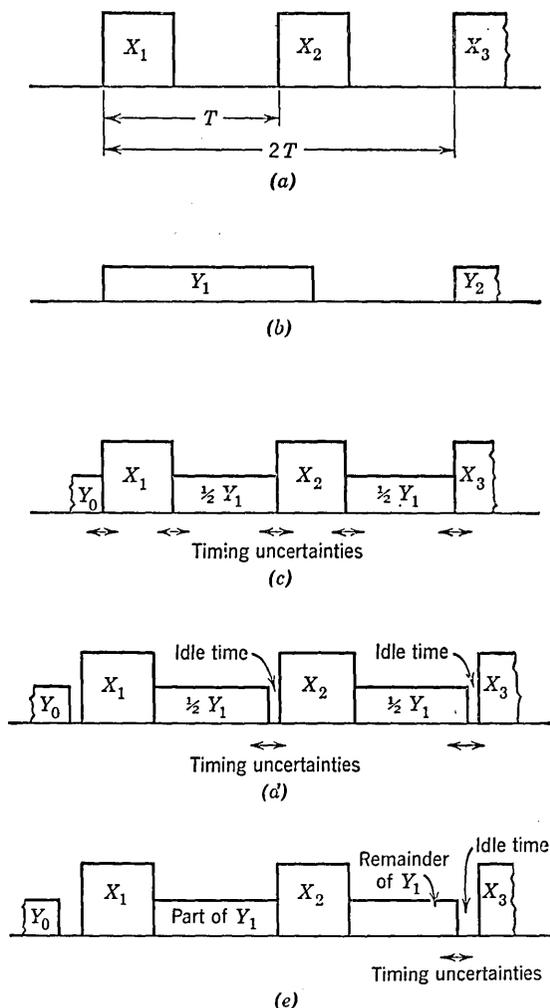


FIG. 3. Time sequencing of computations: (a)  $X$  computation time; (b)  $Y$  computation time; (c) combined  $X$  and  $Y$  computation; (d) combined computations with idle time; (e) combined computation with  $Y$  computation divided unequally.

actual execution time of the program is longer, the analog computer's time scale has been adjusted so that the longer interval still equals  $T$  seconds of the problem time.

A difficulty sometimes occurs in dividing the  $Y$  computation into equal parts. Frequently, a long operation such as a multiply or divide will occur at the point where the computation is to be split. In this case, it is necessary to rewrite the program by performing the arithmetic opera-

tions in a different order, or to split the program unequally. An unequal split of the program necessitates inserting more idle time into the program during the cycle than would otherwise be necessary. If there are several computations to be interleaved, the basic period of the digital program may be quite long, containing many short computation intervals so that additional idle time inserted to offset the program segmenting difficulties can mount up.

**Program Interrupt.** A feature available on several of the current large-scale computers is useful in attacking this problem. The program interrupt feature provides an input for an interrupt pulse from the converter clock. The interrupt pulse forces a jump in the digital computer program at the completion of the instruction currently being executed with provision for returning to the main program at the point where interrupted at a later time. With this feature, the interleaved  $X$  and  $Y$  computations would be programmed as two separate computations. The first computation is programmed to read data for the  $Y$  computation, compute  $Y$ , and write the results to the converter, with a jump back to the start. The second computation is programmed to read data for  $X$ , compute  $X$ , and write the results to the converter with a final jump to an unspecified location. The procedure is as follows.

1. The program is started in the  $Y$  loop.
2. The in-out interlocks halt the program immediately, since  $Y$  data have not been delivered by the converter.
3. At the first clock pulse, data transfer into the in-out register and release the interlocks.
4. At the same time, the program interrupt is initiated.
5. At the end of one of the early  $Y$  computations, the interrupt becomes effective, forces a jump to the start of the  $X$  computation, and sets the address in the jump instruction at the end of the  $X$  computation to return to the  $Y$  computation at the point of interruption.
6. In this way, the program executes one or two instructions of the  $Y$  loop, jumps out to the  $X$  computation, completes the  $X$  computation, and returns to the  $Y$  computation. It continues until the next converter clock pulse initiates data transfers between the two computers and interrupts the  $Y$  computation, forcing a jump which inserts the second  $X$  computation.
7. The program then returns to the  $Y$  loop, which it completes.
8. The program then jumps to the start of the  $Y$  computation, where the interlocks hold the program until new  $Y$  data are available from the converter.

With this approach, the analog time scale is set so that the maximum execution time of one  $Y$  computation and two  $X$  computations is equal to  $2T$  seconds. The converter clock is set to interrupt at  $T$  seconds. The in-out interlocks synchronize the start of the  $Y$  computation, while the program interrupt inserts  $X$  computations at the appropriate time. (See Fig. 3e.)

**Data Transfers.** Cycle time limits the highest frequency component of the response which the digital computer program can faithfully reproduce. Delay time represents a transport lag in the loop through the digital computer. In general, such a lag has a destabilizing effect on the system. Both the cycle time and the delay time are determined by the execution time of the digital computer program. It is important to eliminate as many housekeeping, or nonarithmetic, instructions as possible from the computer program, so that cycle and delay times may be minimized. For this reason, the conversion equipment should be designed to transfer data into the digital computer and accept results from the digital computer as rapidly as possible. The conversion equipment should be designed to require a minimum of control from the digital computer program. The control which the program must supply should be simple and direct so that valuable computer time is not lost in setting up data transfers (Ref. 7).

#### 4. MODES OF OPERATION

**Sampled Data.** The combined system is particularly useful for the simulation of large and complicated control systems, particularly if the system includes sampled or digital effects such as a radar which obtains information at discrete times or a digital guidance computer. The division of the computational task between the two computers is somewhat arbitrary. The guiding principle is that high-accuracy computations must be performed digitally whereas low-accuracy computations may be performed by either analog or digital techniques. Furthermore, it is advisable, where possible, to place high-frequency effects on the analog computer and low-frequency effects on the digital computer.

**Computational Assists.** The combined system is useful for the same class of problems that are normally placed on an analog computer. The digital computer can very easily provide computational assistance in areas which are difficult for the analog computer. As an example, a simple time delay is difficult to simulate by analog techniques; the digital computer need only store sampled values for a specified time and then transmit a delayed value each time a new value is read in. A function of two variables is awkward for an analog computer to supply but easy

for a digital computer. While these examples are almost trivial, a properly designed combined system can be efficiently used for applications as simple as this.

**Time-Sharing Modes.** A properly designed combined system can, of course, operate as an independent analog computer and an independent digital computer. It could also act as an independent analog computer and an independent combined analog-digital computer, by splitting the

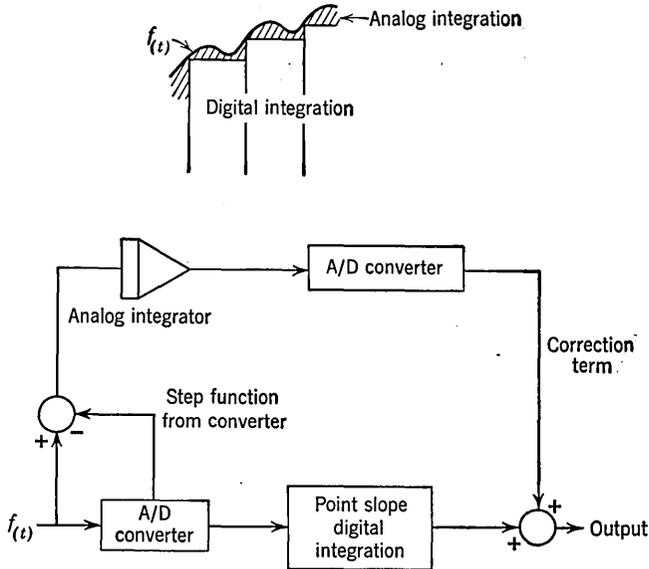


FIG. 4. Combined analog-digital integration.

analog equipment between the two computations. It is even possible by means of the program interrupt feature to time-share the digital computer so that an independent combined computation and an independent digital computation are performed simultaneously. In order to do this the digital computation system must reserve a block of cells for the analog computation so that the analog program can be in high-speed storage at all times. The program interrupt feature can then slip a small computation for the analog computer, such as a function of two variables, into the digital program, whenever the analog computer requires it, without appreciably affecting the execution time of the digital computation that is being executed independently.

**Digital Monitoring.** The combined system has interesting applications in addition to the more obvious ones, suggested above, in which the computational load is split between the two computers in such a way

as to perform one part of the computation by analog techniques while the remainder is computed digitally. The more novel applications employ both analog and digital techniques for the same computational task. The analog computation is performed continuously with the periodic digital computation used to check and modify the analog computation so that the values are correct at the sampling instants.

**Integrations.** Greenstein (Ref. 8) has suggested a method of integration which employs both analog and digital techniques. The input is sampled and integrated digitally as shown in Fig. 4. The sampled values are subtracted from the input waveform in an analog circuit which gives the sampling error. The sampling error function is integrated in an analog circuit, sampled, and added as a correction to the results of the digital integration. The combined analog digital integration produces results which are more precise than an analog integration and more precise than an all digital integration with the same step size and an integration formula of corresponding complexity.

## REFERENCES

1. G. G. Bower, Analog-to-digital converters—What ones are available and how they are used, *Control Eng.*, **4**, 107-118 (1957).
  2. T. C. Fletcher and N. C. Walker, Analog measurement and conversion to digits, *ISA Journal*, **2**, 345 (1955).
  3. B. M. Gordon, High-speed reversible voltage—digital translators and their applications, 10th Annual Instrument—Automation Conference and Exhibit, Sept. 1955.
  4. M. Palevsky, Hybrid analog-digital computing systems, *Instruments and Automation*, **30**, 1957, 1877-80 (1957)
  5. F. H. Blecker, Transistor circuitry for analog-to-digital conversion, WESCON, August 23, 1956, Los Angeles, Calif.
  6. H. Freeman, Cycle and delay time considerations in a real time digital computer, *Am. Inst. Elec. Engrs. General Meeting*, CP 57-677, Montreal, June 24-28, 1957.
  7. W. F. Bauer and G. P. West, A system for general purpose analog-digital computation, *J. Assoc. Comp. Mach.*, Jan. 1957
  8. J. L. Greenstein, Application of AD-DAverter system in combined analog digital computer operation, *Am. Inst. Elec. Engrs. Pacific General Meeting*, June 1956.
  9. H. K. Skramstad, Combined analog digital simulation of sampled data systems, *Am. Inst. Elec. Engrs. Summer General Meeting*, Montreal, June 24-28, 1957.
  10. G. P. West, Combined analog and digital techniques, *Proc. 4th Annual High-Speed Computer Conference*, Louisiana State University, March 5-8, 1957.
- Note.* References 1 and 2 contain additional references.



## Simple Turing Type Computers

*Joseph O. Campeau*

1. Basic Concepts	31-01
2. Functional Requirements	31-02
3. Machine Description	31-03
4. Mechanization	31-07
5. Programming	31-09
6. Communication with no Auxiliary Storage	31-13
7. Machine Comparisons	31-15
References	31-16

### 1. BASIC CONCEPTS

Time sharing is a basic design feature of general purpose digital computers. For example, the typical digital computer operates by time-sharing an arithmetic element among all the arithmetical computations (Ref. 1). The time sharing saves equipment and results in a slower computational rate than would otherwise be achieved with less time sharing, for example, as in an analog computer.

A complete set of logical elements is required to design the arithmetic unit and other parts of a digital computer. (See Chap. 17). The complete set of Boolean elements that is usually used in the logical design of digital computers consists of *and*, *or*, and *not* elements. The Sheffer stroke and Pierce function each are complete and can be used to form all logical Boolean functions.

The concept of time sharing can be carried one step further. Consider a digital system which will operate by time-sharing an *and* gate, an *or*

gate, and a *not* gate among all the required logical Boolean computations (and hence also among all the arithmetic computations).

Turing first described a general class of machines (see Ref. 2) and pointed out that, by using a few simple operations of the proper type, any digital computation could be carried out. In this chapter, machines of the logically programmed type will be called *Turing machines* in spite of the fact that the machine has more than one "tape" and the tapes are finite. This name will be used, although it will be recognized that Turing did not in his paper suggest the use of logical operations but rather used arithmetical examples to illustrate his ideas. Such a machine can be used (a) for computation or (b) to imitate other digital systems. Recent work has been concerned with the design of Turing machines (see Refs. 3 and 4). Much has been written on the logical implications of machines of this type, for example see Ref. 5.

## 2. FUNCTIONAL REQUIREMENTS

In order that any Turing type machine be able to imitate any general purpose computer or be used for computation it must have at least the following three characteristics:

(a) A *storage* equal in bit capacity to or greater than the machine it is to imitate. This can be satisfied many different ways, but one which requires a minimum amount of hardware is the use of a delay element with enough capacity to hold all the variables involved. This can be mechanized by using a circulating register on a magnetic drum, and will be called the *main storage*.

(b) The communication ability to take any two bits located anywhere in the storage and pass them through a processing center and place the result anywhere in the storage. *Information communication* can be realized in one of two ways. The first technique is more direct and will be discussed at greater length. The second technique requires less equipment, but it is more complicated in its operation.

(c) A *processing center* which can combine two bits of information to produce a third *and* or *or* function, be able to accept a single bit and produce its complement, and accept inputs to the Turing machine and produce outputs (or the equivalent of these operations).

**Communication Techniques.** The first technique, in its simplest form, utilizes an auxiliary storage  $S_2$  of a single bit capacity. When any particular bit is to be moved relative to all the other bits in the main storage  $S_1$ , it is transferred to  $S_2$ . By then waiting the appropriate number of bit times until the second bit involved emerges from  $S_1$ , it is possible to present the two desired bits of information to the processing center at the same time. The output from the processing center can then

be set into  $S_2$  and by again waiting the appropriate number of bit times this output can be placed anywhere in  $S_1$ .

In this form  $S_2$  could consist of a single flip-flop, for example.

A less simple version of the  $S_2$  communication technique involves a multibit delay  $S_2$ . Provided that the lengths of  $S_1$  and  $S_2$  have no common multiples other than unity, it can be shown that any bit in  $S_2$  will eventually come next to any bit in  $S_1$ .

Thus whether  $S_2$  is of single or multibit capacity it still serves the same basic purpose of information communication, and allows any particular bit to be placed anywhere at all in  $S_1$ .

The multibit delay can be mechanized by using a second circulating register on the drum. This communication technique allows the construction of a Turing machine which will have no storage external to the surface of a drum.

The second information communication technique does not require an  $S_2$  but makes use of a tap on  $S_1$ . This technique will be described in Sect. 6.

### 3. MACHINE DESCRIPTION

**Operations.** The following three operations will perform all the functions mentioned in the previous sections. *Note.* The main storage  $S_1$  and auxiliary storage  $S_2$  systems are said to be *synchronous* when they operate on a common clock, *asynchronous* if they do not.

(a) *Do Nothing* (DN). Circulate the main and auxiliary storages ( $S_1$  and  $S_2$ ).

(b) *Combine* (C). Combine the two bits coming from  $S_1$  and  $S_2$  in the operation  $\bar{A} + \bar{B}$ , i.e., the Sheffer stroke,  $A|B$ , (the Pierce function,  $A \downarrow B$ , will also work) and place the result in  $S_2$ . Transfer the bit which came from  $S_2$  to  $S_1$ .

(c) *Input-Output* (I). (1) For synchronized systems, read the input bit into  $S_2$ , read out the bit coming from  $S_2$ . (2) For asynchronous systems, read the two input bits into  $S_1$  and  $S_2$  and read out the two bits emerging from  $S_1$  and  $S_2$ .

**Use of Operations.** DN operation is used for communication. The machine will circulate  $S_1$  and  $S_2$  until the two desired bits emerge at the same time and then either the C or the I operations will be used. The C operation can be used to mechanize the *and*, *or*, and *not* operations, as well as provide communication between  $S_1$  and  $S_2$ . The I operation is self-explanatory.

**Sheffer Stroke.** In this chapter a machine will be developed by using the  $\bar{A} + \bar{B}$  function, although a similar treatment can be used for the  $\bar{A} \cdot \bar{B}$  operation. The  $\bar{A} + \bar{B}$  operation is summarized in Table 1.

TABLE 1. SHEFFER STROKE FUNCTION,  $\bar{A} + \bar{B}$

A	B	$\bar{A} + \bar{B}$
0	0	1
0	1	1
1	0	1
1	1	0

**Notation.** Assume a single bit  $S_2$ . A three-row matrix is used: (a) the first row refers to information coming from  $S_1$ ; (b) the second row, for a given column, will specify the operation which combines the bits in the first and third rows, that is, C, I, or DN; (c) the third row will refer to information from  $S_2$ .

**And Function.** The program which will form the *and* function for desired bits  $A$  and  $B$  in  $S_1$  and place the result back into  $S_1$  is shown in matrix form in Table 2.

TABLE 2. PROGRAM FOR THE *and* FUNCTION

Main storage, $S_1$	0	A		1	B		1	X	
Operation	C DN	C DN	C DN	C DN	C DN	C DN	C DN	C DN	C
Auxiliary storage, $S_2$	X 1...	1	$\bar{A}$ ...	$\bar{A}$	A...	A	$\bar{A} + \bar{B}$	$\bar{A} + \bar{B}$	AB... AB

Initially it is assumed that  $S_2$  is in some arbitrary state  $X$ , and that at least one 0 and two 1's are available from known locations in  $S_1$ . The ... are used to indicate that a certain number of bit times must elapse before the required information is available. Recall that the C operation transfers the variable that came from  $S_2$  to  $S_1$ . The operation is then as follows.

1. The C operation combines the 0 with  $X$  to produce  $\bar{0} + \bar{X} = 1$  in  $S_2$ .
2. The DN operation is then used to hold the 1 in the  $S_2$  until  $A$ , the first bit, emerges from  $S_1$ . Then  $A$  coming from  $S_1$  is combined with the 1 in  $S_2$  and  $\bar{1} + \bar{A} = \bar{A}$  is set into  $S_2$ .
3. This is held in  $S_2$  by the DN operation until the 1 emerges from  $S_1$ . This is combined with the  $\bar{A}$  from  $S_2$  and the result,  $\bar{1} + \bar{A} = A$ , is set into  $S_2$ .
4. This is held in  $S_2$  by the DN operation. When  $B$ , the bit to be combined with  $A$ , comes from  $S_1$ ,  $B$  is combined with  $A$  to produce  $\bar{A} + \bar{B}$ . This is set into  $S_2$ .
5. The DN operation is again used to hold  $\bar{A} + \bar{B}$  in  $S_2$ . When the second 1 comes from  $S_1$ ,  $\bar{A} + \bar{B}$  is combined with 1 and the result  $\bar{1} + \overline{(\bar{A} + \bar{B})} = AB$  is set into  $S_2$ .
6. The operation DN is used to hold the desired *and* function,  $AB$ , in  $S_2$  until the desired location (arbitrary state,  $X$ ) in  $S_1$  emerges. At this time, in column eleven, the C operation is used to transfer  $AB$  from  $S_2$  to  $S_1$ .

**Complement.** The matrix form of the *complement* program is shown in Table 3.

TABLE 3. PROGRAM FOR COMPLEMENT

$S_1$	0		$A$	$X$
Operation	C DN		C DN	C
$S_2$	$X$ 1...		1 $\bar{A}$ ...	$\bar{A}$

**Or Function.** The matrix form of the *or* function program is shown in Table 4.

TABLE 4. PROGRAM FOR *or* FUNCTION

$S_1$	0	$A$	$X$	0	$B$	$\bar{A}$	$X$
Oper- ation	C DN	C DN	C DN	C DN	C DN	C DN	C
$S_2$	$X$ 1... 1	$\bar{A}$ ...	$\bar{A}$ $\bar{X}+A$ ...	$X$ 1... 1	$\bar{B}$ ...	$\bar{B}$ $A+B$ ...	$A+B$

Notice that once  $\bar{A}$  has been developed it is sent from  $S_2$  to  $S_1$  by the combine operation at the fifth column. At the eleventh column the  $\bar{A}$  coming from the main memory is combined with the  $\bar{B}$  in  $S_2$  to give  $A + B$ . The  $X$  indicates an arbitrary value in  $S_1$  which will be replaced by  $A + B$ .

**Regeneration.** The combine operation also requires that the bit emerging from  $S_2$  be set into  $S_1$ . This means that whenever a bit from  $S_2$  is combined, that bit will no longer be in  $S_2$ . It will have been used up in the operation. It is therefore necessary to have a process which will generate extra quantities from a given quantity to replenish this variable as it is "used up" by the combine operation.

*Note.* For clarity, the DN operations will be omitted from programs. The use of this operation for communication is indicated in Tables 2 to 4.

To develop an extra  $A$  (where  $A$  is any Boolean variable) in  $S_2$ , given a single  $A$ , four 1's, and two 0's on the drum, first develop  $\bar{A}$  using the program of Table 5. This program generates an extra  $\bar{A}$  needed in the process without using up an  $A$ , any 1's or 0's.

TABLE 5. PROGRAM FOR GENERATING THE COMPLEMENT OF A VARIABLE  $A$ 

$S_1$	0	1	$X$	$A$	1	$X$
Operation	C	C	C	C	C	C
$S_2$	$X$	1	0	1	$\bar{A}$	$A$

This process uses up one 0, two 1's, and an  $A$ . It produces one 0, two 1's, an  $\bar{A}$  and an  $A$ . Since the process ended with all the input quantities plus the  $\bar{A}$ , it is self-sustaining.

**Generating Extra Variables.** Now given the  $\bar{A}$ , the  $A$ , four 1's, and two 0's, an extra  $A$  can be generated by the program of Table 6.

TABLE 6. PROGRAM FOR GENERATING AN EXTRA VARIABLE

$S_1$	0	1	$X$	$X$	0	1	$X$	$A$	1	1	$\bar{A}$	$X$
Operation	C	C	C	C	C	C	C	C	C	C	C	C
$S_2$	$X$	1	0	1	$\bar{X}$	1	0	1	$\bar{A}$	$A$	$\bar{A}$	$A$

The inputs to this process were four 1's, two 0's, an  $A$ , and an  $\bar{A}$ . The outputs were four 1's, two 0's, two  $\bar{A}$ 's, and two  $A$ 's. Thus the program is self-sustaining and also produced the extra desired  $A$ .

**Generation of 1's from 0's.** Initially  $S_1$  will be set to all zeros or at least there will be two 0's in known locations in  $S_1$ . A 1 can be generated from zeros in a self-sustaining manner.

The program which does this is shown in Table 7:

TABLE 7. PROGRAM FOR GENERATING 1'S FROM 0'S

$S_1$	0	$X$	0	1	$X$	1
Operation	C	C	C	C	C	C
$S_2$	$X$	1 . . .	$\bar{X}$	1	0	1 0

The steps are as follows:

1. The process starts with a 0 and produces a 1 in  $S_2$  by  $\bar{0} + \bar{X} = 1$ .
2. The 1 in  $S_2$  is stored in  $S_1$  by  $X + \bar{1} = \bar{X}$ .
3. A second 1 is then produced in  $S_2$  by repeating the first step on a second 0.
4. Combine the 1 from  $S_1$  with the 1 in  $S_2$  to form 0,  $\bar{1} + \bar{1} = 0$ . Note that in this process the 1 in  $S_1$  has been replaced by the 1 from  $S_2$ .
5. The 0 in  $S_2$  is used to replace the original 0 in  $S_1$  by  $\bar{0} + \bar{X} = 1$  and at the same time starts the sequence again.

Thus the process produces 1's from 0's and can regenerate 0's.

**Self-Sustaining Quality.** Even though the combine operation "uses up" the quantity coming from  $S_1$ , given all 0's initially in  $S_1$ , 1's can be generated and for each  $A$  at least two  $A$ 's can be produced. By using these processes any desired number of 1's, 0's, and  $A$ 's (any variable) can be generated, so that the machine as a whole is "self-sustaining."

**Adder Logic.** As an example of the use of the DN and C operations, the sum and the carry logics will be generated. The sum and carry logics, given the previous carry  $K$  and the bits of the two binary numbers being added together,  $A$  and  $B$ , is:

$$S = \bar{A}B\bar{K} + \bar{A}\bar{B}K + ABK + A\bar{B}\bar{K}$$

$$\tilde{K} = AB + BK + AK$$

where  $S$  is the sum and  $\tilde{K}$  is the new carry. The program to generate these two functions is given in Table 8. The C operation has been dropped out, since it is always implied. It is also understood that enough extra  $A$ 's,  $B$ 's, 1's, 0's, etc., will be generated using the processes described above to make this process self-sustaining.

TABLE 8. SUM AND CARRY LOGICS FOR AN ADDER

$S_1$	$A$	$1$	$B$	$0$	$A$	$\bar{B}$	$\bar{A} + \bar{B}$	$K$	$0$
$S_2$	$1$	$\bar{A}$	$A$	$\bar{A} + \bar{B}$	$1$	$\bar{A}$	$A + B$	$AB + \bar{A}\bar{B}$	$\bar{K} + A\bar{B} + \bar{A}B$
$S_1$	$AB + \bar{A}\bar{B}$		$\bar{K}$	$\bar{K} + A\bar{B} + \bar{A}B$			$0$		
$S_2$	$1$	$A\bar{B} + \bar{A}B$		$K + AB + \bar{A}\bar{B}$		$S = A\bar{B}\bar{K} + ABK + \bar{A}B\bar{K} + \bar{A}\bar{B}K$			
$S_1$	$B$	$\bar{A}$	$K$	$\bar{A} + \bar{B}$					
$S_2$	$1$	$\bar{B}$	$A + B$	$\bar{K} + \bar{A}\bar{B}$	$\tilde{K} = AB + BK + KA$				

**Input-Output.** With regard to the input-output operation for two synchronous systems, only a single bit need be transferred. The communication for both inputs and outputs can be a train of 0's until some information, (either a 1 or a 0) is to be transmitted. At this time first a 1 is transmitted followed immediately by the information bit. Thus to send 1 0 1 and then 1 1 after a slight delay, send the following sequence:

0 0 0 1 1 1 0 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0.

In the case of two asynchronous systems, it is necessary to transmit two bits each for input and output. Assume that the duration of the transmitted signal will always be at least one bit time long (receiver time). The information transmitted as (a) the information bit (either 1 or 0) and (b) the clocking bit. The information bit will be set and then the clocking bit will go from 0 to 1. The receiver samples the information line when the clocking line goes from 0 to 1.

**4. MECHANIZATION**

The specific Turing machine described above can be mechanized by using a drum storage. A single program register of length  $2p$  bits with two read stations  $p$  bits apart will be required, where  $p \gg m$  and  $m$  is the length of the  $S_1$  register. Assume asynchronous inputs and outputs. The order code for the machine is given below:

	$P1$	$P2$
Combine	0	0
Do nothing	{ 0	1
	{ 1	0
Input-output	1	1

The logic for the machine using set-reset flip-flops (see Chap. 17) is as follows:

Auxiliary Storage ( $S_2$ ) Flip-Flop  $A$

$$SA = \overline{P1} \overline{P2} \overline{A} + P1 P2 U1$$

$$RA = \overline{P1} \overline{P2} AR + P1 P2 \overline{U1}$$

Main Storage ( $S_1$ ) Write Flip-Flop  $W$

$$SW = (P1\overline{P2} + \overline{P1}P2)R + A \overline{P1}\overline{P2} + P1P2U2$$

$$RW = \overline{SW}$$

where  $R$  is the main storage ( $S_1$ ) read flip-flop, and  $U1$  and  $U2$  are the two input bits.

Outputs

$$V1 = P1 P2 A$$

$$V2 = P1 P2 R$$

The computer block diagram is shown in Fig. 1.

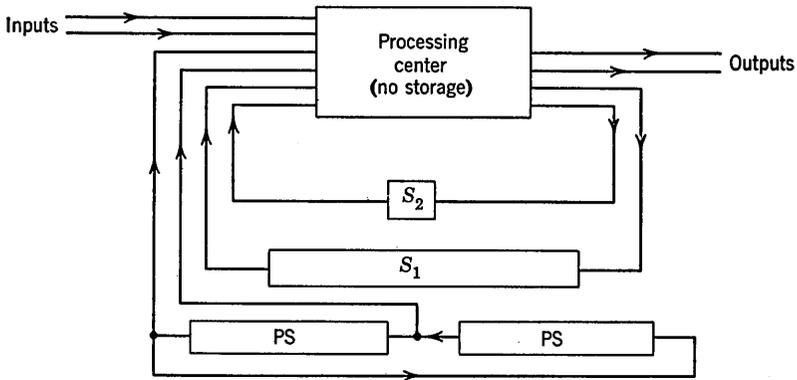


FIG. 1. Block diagram of Turing type computer:  $S_1$ , main storage;  $S_2$ , auxiliary storage; PS, program storage.

Note that only one program storage (PS) with two reading heads on it was required. Since only three different operations are used rather than four, it is possible to use only one program register as follows. By making the order code for the DN operation either of the two codes 01 or 10, it

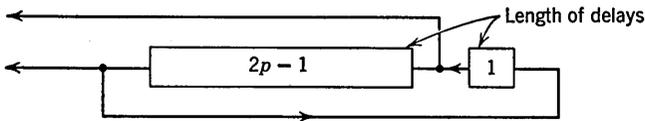


FIG. 2. Alternate form of program storage.

is possible to make the machine operate satisfactorily even though the same bit of information will be used alternately for  $P1$  and  $P2$ . As a final point it is interesting to note that it is also possible to construct a program storage as in Fig. 2.

In using this type storage it will always be necessary to place at least one DN operation between each C and I operation although any number of C or I operations can be placed adjacent to one another.

## 5. PROGRAMMING

While the logic for the Turing machine as well as the information in the program channels is fixed, it is nevertheless possible to have this machine imitate any general purpose digital computer within the capacity of the program and main storages in all respects except speed of operation.

**Terminology.** The actual physical logic for the Turing machine will be called the *first level logic*. The fixed program in the program memory will be called the *first level program*. The logic for the digital system that the Turing machine is imitating will be called the *second level logic*. Note that the Turing machine's first level program is used to imitate the second level logic. The second level machine will be the computer that the Turing machine is imitating.

Finally it is instructive to note that the second level computer will have its own program. For example, the second level machine might be Frankel's computer (see Ref. 1), and this computer has a program. The program for the second level machine will be called the *second level program*. This program will be changeable and will be located physically in the main storage. It is instructive to note that while the first level program (located in the program channels) cannot be changed without the use of external filling equipment, the second level program can be altered without the addition of any extra physical equipment to the Turing machine. This is true since the Turing machine already contains within itself the ability to alter the contents of its own main storage.

**First Level Program Design Example.** In order to illustrate the techniques involved in designing a first level program a simple digital system will now be considered. This digital system is to accept asynchronous input pulses and count them. At every fourth pulse it is to reset the counter to zero and send out a counting pulse.

The first step in the process is to write out a second level logic which will accomplish the desired task. This logic must be written in terms of delay elements each of which has one input and one output. The particular system under consideration will require two delay elements called  $Q1$  and  $Q2$ . The input pulse will be called  $U$  and the output pulse will be called  $V$ . The logic for the system is:

$$N1 = \overline{Q2}U + Q1\bar{U}$$

$$N2 = Q1U + Q2\bar{U}$$

$$U = U1_n\overline{U1_{n-1}}$$

$$V = U1_n\overline{U1_{n-1}}\overline{Q1}Q2$$

where  $N1$  and  $N2$  are the next states of  $Q1$  and  $Q2$  and  $U1_n$  and  $U1_{n-1}$  are the present and previous values of  $U1$  input line.

The counter  $Q1Q2$  counts the sequence shown in Table 9. It repeats its

TABLE 9. COUNTER SEQUENCING

Time	$Q1$	$Q2$
1	0	0
2	1	0
3	1	1
4	0	1
5	0	0

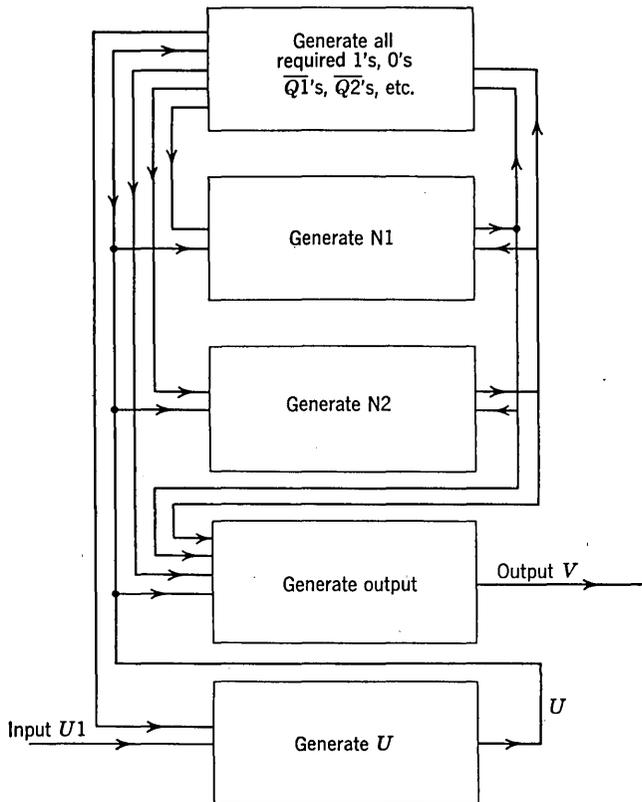


FIG. 3. Sequence of operations to be performed.

pattern every fourth count. The signal  $U1_n\overline{U1_{n-1}} = U$  occurs once each time the asynchronous input  $U1$  goes high. Assume that contact bounce will not last for a complete circulation time of the main storage. There does exist a logic which will bring in three bits to the Turing machine as inputs. One can be the break and a second the make signal so that contact bounce difficulties can be eliminated entirely.

First write in block diagram form a flow chart (see Fig. 3) to indicate the order in which the operations are to be performed. The first level programs are shown in Table 10.

TABLE 10. FIRST LEVEL PROGRAMS

a. Generate  $U$

Second level logic:  $U = U1_n\overline{U1_{n-1}}$

Program:

$X$	$\overline{U1_{n-1}}$	$1$	
$I$	$C$	$C$	
$X$	$U1_n$	$\overline{U} = \overline{U1_n} + U1_{n-1}$	$U = U1_n\overline{U1_{n-1}}$

b. Generate  $V$

Second level logic:  $V = U1_n\overline{U1_{n-1}}\overline{Q1}Q2 = U\overline{Q1}Q2$

Auxiliary storage holds  $U$  from  $U$  generation

Program:

$\overline{Q1}$	$1$	
$C$	$C$	
$U$	$\overline{U} + Q1$	
$Q2$	$1$	
$C$	$C$	$I$
$U\overline{Q1}$	$\overline{U} + \overline{Q2} + Q1$	$V = \overline{Q1}Q2U$

c. Generate  $N1$

Second level logic:  $N1 = \overline{Q2}U + Q1\overline{U}$

Program:

$0$	$Q2$	$U$	$0$	$U$	$Q1$	$\overline{U} + Q2$	$0$
$C$	$C$	$C$	$C$	$C$	$C$	$C$	$C$
$X$	$1$	$\overline{Q2}$	$\overline{U} + Q2$	$1$	$\overline{U}$	$\overline{Q1} + U$	$N1 = \overline{Q2}U + Q1\overline{U}$

d. Generate  $N2$

Second level logic:  $N2 = Q1U + Q2\overline{U}$

Program:

$Q1$	$1$	$U$	$0$	$U$	$Q2$	$\overline{Q1} + U$
$C$	$C$	$C$	$C$	$C$	$C$	$C$
$1$	$\overline{Q1}$	$Q1$	$\overline{Q1} + \overline{U}$	$1$	$\overline{U}$	$Q2 + U$

$N2 = Q1U + Q2\overline{U}$

The total program must also regenerate used up quantities in order to be self-sustaining.



mentioned above will give the instructions during the  $i$ th ( $i = 1, \dots, 40$ ) revolution of the main storage in row  $41 - i$ . The matrix is 23 by 40 and if the program positions are numbered  $1, \dots, 920$ , then element  $Q_{ij}$  in the matrix will give the program operation which is to occur at program position

$$23(40 - i) + j \quad (i = 1, \dots, 40; j = 1, \dots, 23).$$

The program matrix has been shown below in Table 11. C and I stand for combine and input-output, respectively. A blank space will stand for do nothing.

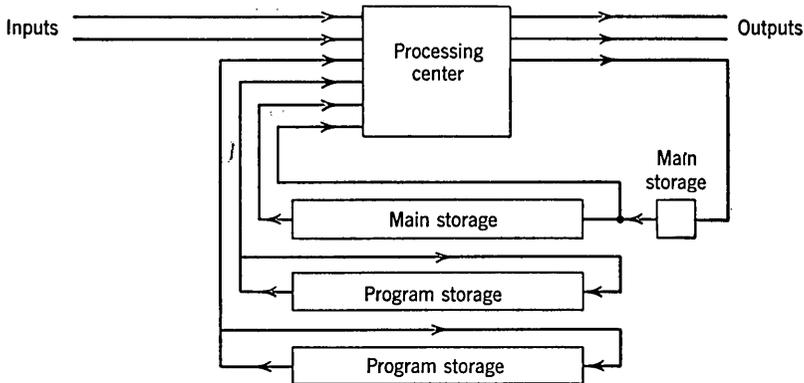


Fig. 4. Block diagram of a Turing type computer having one main storage with a tap.

## 6. COMMUNICATION WITH NO AUXILIARY STORAGE

The Turing machine described in this chapter made use of an auxiliary storage for information communication. It was shown that this could either be a single bit storage (a flip-flop) or a multi-bit storage (provided the main and auxiliary storages had no common multiple other than unity).

There is a communication technique which does not require the use of an auxiliary storage but instead requires only a tap in the main storage. One structure a machine of this type can have is as shown in Fig. 4.

The logic for the machine in terms of set-reset flip-flops is:

Write flip-flop  $W$ :

$$\begin{aligned} SW &= P1P2R + P1R\bar{W} + \bar{P1}\bar{W}P\bar{2} + \bar{P1}\bar{W}U2 \\ RW &= P1P2\bar{R} + \bar{P2}RW + \bar{P1}P2\bar{U1}W \end{aligned}$$

Outputs,  $V1$  and  $V2$ :

$$\begin{aligned} V1 &= P1P2WR \\ V2 &= P1P2\bar{W}R \end{aligned}$$

The program is on fixed channels on the drum and is not altered. The order code for the machine is:

<i>Operation</i>	<i>P1</i>	<i>P2</i>
C, combine ( $\bar{A} + \bar{B}$ )	0	0
I, input-output	0	1
$\oplus$ , <i>exclusive or</i> ( $A \oplus B$ )	1	0
DN, do nothing	1	1

The DN, C, and I operations are essentially the same as described earlier. The *exclusive or* operation,  $A \oplus B$  (or  $A\bar{B} + \bar{A}B$ ) operates on the bits coming from the main storage as shown in Table 12.

TABLE 12.

<i>A</i>	<i>B</i>	<i>Exclusive Or, A <math>\oplus</math> B</i>
0	0	0
0	1	1
1	0	1
1	1	0

The result is set into the main storage. Only the communication mechanism will be described for this particular machine.

What must be shown is that any two bits in the storage can be brought adjacent to each other so that they can be operated on. Basically the problem is one of having to move a single bit of information "by" the other bits and next to the bit with which it is to be combined.

First assume initially every other bit in the storage is a 0 and that the following sequence exists in the storage:

A    0    B    0    F    0    D

It is desired to move *A* past *B* and *F* so that *A* and *D* can be operated on. After such a move the storage sequence would then be:

A    0    B    0    F    A    D

The process which does this is shown in Table 13. The  $\oplus$  is used for the exclusive or operation.

Thus the final result was that *A* and *D* are adjacent and can be operated on.

The communication technique described above made use of a single bit delay on the storage. It can also be shown that as long as the tap distance and the overall length of the main storage have no common multiples other than unity that this technique will also work. Thus again it is possible to build a machine with no logical flip-flop, i.e., all storage on the drum.

TABLE 13. PROGRAM FOR SHIFTING VARIABLES TO MAKE THEM ADJACENT

Read	$A$	$0$	$B$	$0$	$F$	$0$	$D$	
Operation	DN	$\oplus$	DN	$\oplus$	DN	$\oplus$	DN	
Write		$A$	$A$	$B$	$B$	$F$	$F$	$D$
Read	$A$	$A$	$B$	$B$	$F$	$F$	$D$	
Operation	DN	DN	$\oplus$	$\oplus$	$\oplus$	$\oplus$	DN	
Write		$A$	$A$	$A \oplus B$	$A$	$A \oplus F$	$A$	$D$
Read	$A$	$A$	$A \oplus B$	$A$	$A \oplus F$	$A$	$D$	
Operation	DN	DN	DN	DN	$\oplus$	DN	DN	
Write		$A$	$A$	$A \oplus B$	$A$	$F$	$A$	$D$
Read	$A$	$A$	$A \oplus B$	$A$	$F$	$A$	$D$	
Operation	DN	DN	DN	$\oplus$	DN	DN	DN	
Write		$A$	$A$	$A \oplus B$	$B$	$F$	$A$	$D$
Read	$A$	$A$	$A \oplus B$	$B$	$F$	$A$	$D$	
Operation	DN	DN	$\oplus$	$\oplus$	DN	DN	DN	
Write		$A$	$A$	$B$	$0$	$F$	$A$	$D$
Read	$A$	$A$	$B$	$0$	$F$	$A$	$D$	
Operation	DN	$\oplus$	DN	DN	DN	DN	DN	
Write		$A$	$0$	$B$	$0$	$F$	$A$	$D$

## 7. MACHINE COMPARISONS

**Definition of a Measure.** One measure of performance of Turing type machines is the sum of the number of bits transferred to and from the processing center per bit time of computer processing. Define number of bits as the logarithm to the base two of the total number of states represented. For example, two binary lines are required to represent three states, but the number of bits being transferred is  $\log_2 3 = 1.57$ . This measure ( $\alpha$ ) is not dependent on the clock frequency of the computer. In determining  $\alpha$  all information transfers from the storage should be considered as well as all inputs and outputs to the computer.

In considering information processing note that in certain instances such as circulating storage registers on drums and delay lines or composed of flip-flops there is a continuous transfer of information from one storage cell to the next. Even though there is movement of information, it will not be considered that information processing occurs, provided that the transfer *always* occurs.

**Comparisons.** For example, a circulating register on the drum closed on itself would have  $\alpha = 0$ . Thus,  $\alpha = 0$  means no processing of data is occurring, even though information is circulating.

The first machine considered which made use of an auxiliary storage

for communication has an  $\alpha = 8 + \log_2 3 = 9.6$ . Four of the bits come from the input and output (assuming asynchronous inputs and outputs), 2 come from the main storage, 2 from the auxiliary storage, and  $\log_2 3$  come from the program.

The second machine considered which did not make use of an auxiliary storage for communication had  $\alpha = 9$ . (4 for input-output, 2 for program, and 3 for main storage.)

Thus the second machine is seen to have a slightly smaller  $\alpha$  measure than the first.

A theoretical minimum  $\alpha$  for a general purpose digital computer can be developed. For asynchronous inputs and outputs at least four lines are required. The processing center must have at least two inputs from the memory and one output to the memory. Thus a minimum possible value of  $\alpha = 7$  for asynchronous inputs and outputs is obtained although in this paper the minimum value machine presented had  $\alpha = 9$ .

The machine with  $\alpha = 9$  appears to be the Turing machine with the least  $\alpha$  which still can be a general purpose digital computer with asynchronous inputs and outputs.

## REFERENCES

1. S. P. Frankel, The logical design of a simple general purpose computer, *I.R.E. Trans. Elec. Comp.*, March 1957.
2. A. M. Turing, On computable numbers, with an application to the entscheidungsproblem, *Proc. London Math. Soc.*, **42**, 230-265 (1936).
3. W. E. Smith, A digital system simulator, *Proc. West. Jt. Comp. Conf.*, Feb. 26-28, 1957.
4. J. Blankenbaker, Logically micro-programmed computers, *I.R.E. Trans. Elec. Comp.*, **4**, June 1958.
5. C. E. Shannon and J. McCarthy, Editors, *Automata Studies*, Annals of Mathematics Studies, No. 34, Princeton University Press, New Jersey (1956).

# INDEX

- A0, A1, A2, 2-158
- A3, Arithmetic, 2-158
- ABC, 2-158
- ABC I, 2-159
- Academy of Sciences of the Soviet Union, 2-188
- Acceptance tests, simulation, 2-239
- Access requirements, files, 4-06
- Access time, magnetic drum, 19-05
  - storage, (table) 4-13
- Accounting, casualty insurance, 8-08
  - report preparation, 8-09
  - data processing, 8-19
  - dividend, 8-02
  - inputs and outputs, 8-03, 8-09, 8-12, 8-15
  - life insurance, 8-01
  - processing steps, billing, 8-13
    - casualty insurance, 8-10
    - life insurance, data, 8-03
    - payroll and salary distribution, 8-15
  - quantity of data, 8-03, 8-10, 8-12, 8-15
  - salary distribution, 8-15
  - staff, 8-02, 8-09, 8-13, 8-16
- Accounting machine, line printers, 5-35
- Accuracies, analog computers, 22-37
  - electrolytic tank, 25-11
  - quarter square multiplier, 23-09
  - resistance networks, 25-19
  - resistance paper, 25-06, 25-07
  - servo multiplier, 23-04
  - time division multiplier, 23-07
- ACM, 1-02, 2-186, 2-190, 2-261
- A-C network analyzers, 25-20
- ACOM, 2-156
- Acoustic delay line storage, 19-29
- A-C resolvers, 23-32
- Across-through relations, (table) 24-04
- Across variable, 24-03, 25-02
- Adams method, 2-229
- Adams and Milne method, 10-09
- Adders, binary, 18-08
  - decimal, 18-27
  - full, 18-09
  - half, 15-22, 18-08
  - incremental computation, 29-25
  - Kirchhoff, 18-10
  - mechanical, 27-02
  - operational digital, 29-10
  - parallel, 18-11
  - symbols, 18-03, 18-10
  - transistor, 16-11
  - Turing computers, 31-06
- Addition, see *Adders*
- Addresses, 2-56
  - degenerate, 12-28
  - floating, 2-161
  - indirect, 2-193
  - modification methods, 2-60
  - number per instruction, 12-27
  - relative, present, 2-192
  - sorting, 2-146
  - symbolic, 2-160
  - system, requirements, 2-56
- Addressing, 5-07
- Add-subtract, 18-16
- ADES II, 2-157
- Adjoint system, 26-07
- AFAC, 2-156
- AIEE publications, 2-261
- Aircraft production, scheduling, 9-07
- ALGEBRAIC, 2-159
- Algebraic equations, simultaneous linear, 10-02
- Algebraic language, 2-186
  - Soviet compiler, 2-228
- Algorithms, 2-190

- Allison, G. M., 2-156, 2-157  
 Allstate Insurance Company, 8-08  
 Alphabetic information, 5-39  
 Alphanumeric codes, 5-05  
   Datatron 205, (table) 2-108  
 AMA publications, 2-261  
 Ambiguous-word logic, 2-54  
 American Airlines, 9-01  
 American Institute of Electrical Engineers, 2-261  
 American Management Association, 2-261  
 AM-FM multiplier, 23-10  
 Amplifiers, operational, 22-08, 22-16  
   characteristics, 22-16  
   chopper, 22-31  
   design, 22-16  
   drift, 22-17  
   drift corrected, 22-30  
   drift errors, 22-36  
   errors, 22-33  
   gain, 22-09, 22-16  
   gain errors, 22-33  
   integration, 22-09  
   noise errors, 22-36  
   sign inversion, 22-09  
   summation, 22-09  
   summing integrators, 22-09  
   transfer impedance, (table) 22-18  
   pulse, 14-21  
   torque, 27-04  
 Amplitude scale factors, 22-10  
 Analog computers; see also *Analogs*  
   accuracy, 22-37  
   A-C resolvers, 23-32  
   amplifiers, 22-04  
   analog-digital computing systems, see *Analog-digital computing systems*  
   applications, 21-09  
   classification, 21-02  
   comparison with digital computers, (table) 21-02  
   continuous, 21-04  
   curve followers, 23-14  
   diagrams, 22-03, 21-11  
   differential analyzers, 21-04  
   differential equation solution, 22-01  
   differentiation, 22-03  
   digital differential analyzers, 21-04, 28-02; see also *Digital differential analyzers*  
   Analog computers, digital operational computers, 28-11  
   digital techniques, 28-01, 28-15, 29-01; see also *Operational digital systems*  
   diode switching, 23-23  
   direct, 21-03, 21-10  
   discrete, 21-04  
   discrete variable representation, 28-14  
   division, 23-12  
   electronic and mechanical, 27-02  
   engineering problems, steps in the solution of, 21-06  
   equations, solution, 27-09  
   errors, 22-33  
   field problems, 25-01; see also *Field problems; Analogs*  
   function generators, 23-14; see also *Function generators*  
   function multipliers, 23-01  
   general purpose, 21-02  
   hybrid systems, 29-01  
   implicit computation, 23-32  
   indirect, 21-04, 21-09  
   initial conditions, 22-10  
   input requirements, 21-05  
   integration, 22-04, 22-07, 22-09, 23-13, 23-25, 27-03  
   limiters, 23-23  
   linear elements, 22-01, 22-03  
   long-time, 21-04  
   mechanical, 27-01; see also *Mechanical computers*  
   multiplication, 22-04, 23-01, 27-02  
   noise, 26-01; see also *Noise*  
   noise generators, 26-12; see also *Noise generators*  
   nonlinear electronic elements, 23-01  
   notation, 22-03  
   operating frequencies, 22-12  
   operational amplifiers, 22-08; see also *Amplifiers, operational*  
   operational digital techniques, 28-11, 29-01; see also *Operational digital techniques*  
   output requirements, 21-05  
   parallel operation, 21-02  
   passive computing elements, 22-04  
   problem setup, 21-07, 22-12  
   recorders, 23-34

- Analog computers, rectangular to polar transformations, 23-32
- relays, 23-23, 23-29
- sawtooth wave generation, 23-29
- scale factors, 21-07, 22-10, 27-14
- servodriven potentiometers, 23-31
- short-time, 21-04
- simulation, 23-25; see also *Simulation*
- special purpose, 21-02
- square wave generation, 23-29
- statistical problems, (table) 26-10
- statistical techniques, 26-01
- summers, 22-04, 22-09
- summing integrator, 22-04, 22-09
- switching devices, 23-23
- symbols, (table) 21-10, 22-03
- systems, 29-02
- time delays, 23-35
- transfer function representation, 22-13
- transfer impedance, short circuit, (table) 22-18
- trigonometric devices, 23-31
- variables, 22-02, 22-11
- Analog-digital computing systems, 30-01
- applications, 30-01, (table) 30-03
- buffers, 30-09
- control and timing, 30-08
- converters, 20-44, 30-04
- data transfer, 30-13
- digital monitoring, 30-14
- input-output buffer and interlock control, 30-09
- integration, 30-15
- interlocks, 30-09
- interrupt feature, 30-12
- problem types, (table) 30-03
- sequencing, 30-10
- time sharing modes, 30-14
- Analog-digital converters, 20-44, 30-04
- applications, 20-44, (table) 20-45
- bandwidth, 30-06
- buffering action, 30-05
- codes, 20-45, (table) 20-46
- code wheels, 20-52, (table) 20-56
- combined systems, 20-45, 30-04
- digital-analog, shaft position, 20-60
- time interval, 20-57
- voltage, 20-58
- digital codes, (table) 20-46
- electronic, 30-04
- Analog-digital converters, equipment, 20-44, (table) 20-45
- feedback methods, 20-47, 20-60
- free running, 30-04, 30-06
- frequency conversion, 20-48
- incremental pattern methods, 20-56
- incremental systems, 29-18
- indirect, 20-47, 20-57
- mechanical, 30-04
- plotters, 20-63, (table) 20-65
- pulse width, 30-05, 30-07
- ramp method, 20-49
- RC* networks, 20-60
- resistance networks, 20-58
- sampling, 30-04, 30-06
- shaft position, 20-51
- ambiguities, 20-52
- coded patterns, 20-51
- code wheel converters, 20-55, (table) 20-56
- cyclic binary codes, 20-52
- digital-analog, 20-61
- dual brush method, 20-52
- incremental, 20-56
- indirect, 20-57
- mechanical brush, (table) 20-56
- multispeed coders, 20-55
- photoelectric, 20-56
- V brush method, 20-54
- specifications, 20-44
- successive approximation method, 20-63
- techniques, 20-44
- time interval, 20-47, 20-57
- voltage, 20-48, 20-58, 20-62
- cascaded stages, 20-50
- cathode ray tube, 20-48
- ramp method, 20-49
- Analogs, 24-01; see also *Analog computers*
- across-through relations, (table) 24-04
- across variables, 24-03, 24-12
- conducting liquid, 25-06
- conducting sheet, 25-05, (table) 25-07
- continuous, 21-04
- electric, 25-05
- correspondences, (table) 24-09
- direct, transfer function representation, 22-13
- discrete element electric, 25-11
- duality, 24-08

- Analogs, electrical analogy, 24-01  
   electrolytic, 25-08  
   electro-optical, 25-23  
   elements, 24-04; see also *Elements*  
   energy, 24-07  
   fluid mappers, 25-22  
   junction technique, 24-04, 24-05, 24-09  
   nonelectric, 25-22  
   path technique, 24-04, 24-06, 24-11  
   quantities, (table) 24-09  
   relation to duals, 24-10  
   resistance paper, 25-05  
   stretched membranes, 25-22  
   symbols, 24-04  
   terminology, 24-03  
   through variables, 24-03, 24-12  
 Analogy basis, discrete electric analogs, 25-11  
 Analyzers, digital differential, 12-12, 21-04, 28-02, 29-13  
 And circuits, 14-34, 17-04, 17-12  
   magnetic cores, 15-17  
   transistors, 16-10, 16-18  
 AN/FSQ-7, 13-03  
 Arithmetic, classification of subroutines, 2-180  
   incremental computation, 29-21  
   instructions, see *Instructions*  
   operations, 2-59  
   operators, 2-229  
   programming, 2-02  
   programming boxes, 2-131  
 Arithmetic checks, (table) 13-03, 13-06  
 Arithmetic and control unit, 12-06, 18-01  
   adder, 16-11, 18-08, 18-11; see also *Adders*  
   adder-subtractor, 18-16  
   binary counters, 18-04  
   binary operations, 18-03  
   checking, 18-32  
   comparison, 18-31  
   control unit design, 18-33, 18-35  
   counting, 18-26  
   decimal operations, 18-25  
   design, 18-35  
   division, 18-21  
   equipment, 5-38  
   external control, 18-34  
   extract, 18-31  
   fixed point, 18-32  
   Arithmetic and control unit, flip-flop logic, 18-02  
   floating point, 18-32  
   logical design, 17-26  
   logical transfer, 18-31  
   magnetic cores, 15-22  
   multiplication, 18-16, 18-27; see also *Multiplication*  
   notation, 18-02  
   number system conversion, 18-23  
   programmed control, 18-34  
   readin, 18-07  
   shifting registers, 18-05  
   special operations, 18-30  
   square root, 18-22  
   subtractor, 18-14; see also *Subtraction*  
   symbols, 18-02  
   zero representation, 18-16  
 Arrow, 2-111; see also *Strela*  
 Assemblers, see *Assembly programs*  
 Assembly programs, 2-161; see also *SAP*  
   directory, 2-164  
   IBM 704, Share assembly program (SAP), 2-164, 2-165, 2-167  
   location counter, 2-164  
   one-pass, 2-164  
   PACT, 2-162  
   pseudo instructions, 2-165  
   symbolic addresses, 2-163  
   two-pass, 2-164  
   utility, 2-183  
   X-1, 2-162  
 Association for Computing Machinery, see *ACM*  
 Asynchronous computers, 17-02, 17-05  
 AT3, MATHEMATIC, 2-158  
 Atomic Instrument Company, 20-18  
 Auerbach Electronics Corporation, 16-30  
 AUTOCODER, 2-156, 2-157  
 Autocorrelation function, 26-05  
 Automatic checks, 3-13  
 Automatic coding systems, (table) 2-156; see also *Automatic programming*  
 Automatic data acquisition, 29-17  
 Automatic programming, 2-02, 2-12, 2-159  
   algorithms, 2-190  
   assembly programs, 2-163; see also *Assembly programs*  
   business systems, 7-03  
   combined systems, 2-163

- Automatic programming, compiler, Soviet, 2-228; see also *Operational programming*
- compilers, 2-160, 2-186; see also *Compilers; Translators*
- control instructions, 2-160
- easy mistake discovery, 2-162
- elimination of repetitious coding, 2-161
- floating addresses, 2-161
- future trends, 2-189
- generators, 2-163; see also *Subroutines, generators*
- input languages, 2-160
- integrated systems, 2-184; see also *Integrated systems*
- interpreters, 2-234; see also *Interpreters*
- IT translator, 2-200; see also *IT compiler-translator*
- languages, 2-186; see also *Languages*
- logic machines, 11-08
- mnemonic codes, 2-160
- objectives, 2-159
- preset parameters, 2-161
- prevention of mistakes, 2-163
- program parameters, 2-161
- pseudo instructions, 2-162
- recursive languages, 2-244
- Soviet algebraic language compiler, 2-228
- subroutine generators, 2-182
- subroutines, 2-161, 2-167; see also *Subroutines*
- symbolic addresses, 2-160
- synthetic instructions, 2-162
- systems, (table) 2-156
- translation, 2-160
- translator construction, 2-221; see also *Translator construction*
- translators, 2-186; see also *Translators*
- understandable language, 2-160
- unification of techniques, 2-163
- universal computer language, 2-163
- utility programs, 2-162, 2-183; see also *Utility programs*
- Auxiliary equipment, 7-08
- magnetic tape, 20-42
- punched card, 20-31
- Avidac, 12-09
- B0, FLOWMATIC, 2-158
- Babcock and Wilcox, 2-158
- BACAIC, 2-157
- Backlash, simulation of, 23-27
- BALITAC, 2-157
- Bandwidth, analog-digital converters, 30-06
- BAP, 2-156
- Base, 12-12
- BASE 00, 2-159
- Base counter, MIDAC, 2-118
- B box, see *Index registers*
- BELL L1, L2, L3, 2-157
- Bell Telephone Laboratories, 2-157, 12-09, 16-02, 16-15, 16-30
- relay calculator, 2-167, 12-09, 13-03
- Bendix Computer Division, 2-263, 12-09
- G-15, 2-64
- Intercom system, 2-241
- BESM, 2-111, 2-188
- Biharmonic equation, 25-04, 25-19
- Biharmonic operator, 25-02
- Binary arithmetic processes, 12-21
- multiplication, 18-16
- Binary coded decimal, 5-05, 18-26
- counter, transistors, 16-28
- systems, 5-05, 12-18
- 8421 system, 5-05, 20-46
- Binary coded number systems, 12-18
- Binary codes, 20-46
- Binary counters, 18-04
- Binary-decimal conversion, 2-20
- Binary digit, 3-06
- Binary number system, 5-05
- Binary rate multiplier, 29-05
- Binary scalars, 29-05
- Binary states, 17-02
- BIOR, 2-158
- Biquinary code, (table) 12-21
- Bistable storage elements, 17-05
- Bizmac, 2-56
- B line, see *Index registers*
- Block diagrams, computer solution, 2-04
- logical design, 17-28
- storage, 19-02, 19-06, 19-17
- Blocking oscillators, 14-35
- Blotter type electrolytic models, 25-22
- Boeing, 2-156, 2-157
- Boolean algebra, 17-10
- canonical form, 17-12
- equivalences, (table) 17-11

- Boolean algebra, normal form, 17-13  
 Boolean functions, 31-01  
 Booth method, 18-18  
 Bound variables, 2-51  
 Brain behavior, 11-01  
 Branch instruction, 12-04, 17-26  
 Breakpoint, diode characteristic, 23-15  
     programming notations, 2-61  
 B registers, see *Index registers*  
 Brush Electronics Company, 19-09  
 Buffer storage, 17-25  
     analog-digital converters, 30-05  
     equipment, 5-23  
     input-output, 20-03  
         analog-digital computer systems, 30-09  
     perforated tape punch, 20-25  
 Building blocks, preparation of problems, 2-187  
 Burroughs Corporation, 2-98, 2-158, 2-239, 2-263, 12-09, 16-30, 20-18; see also *Datatron*  
     Datatron, see *Datatron*  
     Datatron 201, 2-158  
     Datatron 205, 2-158  
     UDEC III, 2-158  
 Business data manipulation, 3-06  
 Business data processing, 8-01, 9-01; see also *Accounting*; *Business systems*; *Data processing*; *Inventory*; *Reservations*; *Scheduling*  
 Business systems, definition, 7-02, 7-04  
     design for electronic data processing, 7-01, 7-03  
     economic evaluation, 7-06, 7-12  
     equipment,  
         auxiliary, 7-08  
         communication, 7-06  
         data conversion, 7-06  
         input-output, 7-06  
         programming, 7-08  
         selection, 7-02, 7-04, 7-07  
         storage, auxiliary, 7-05  
     implementation, 7-03, 7-06  
     information flow, 7-02  
     integrated, 7-11  
     payroll, 7-10  
     rate of return, 7-12, (table) 7-13  
     requirements, 7-01  
 CAGE, 2-156  
 Calculating punches, 20-32  
 Call-in,  
     momentary, 2-180  
     permanent, 2-180  
 Cambridge University, 2-160, 2-251  
 Canonical form, 17-12  
 Capacitor-diode storage, 19-32  
 Capacitors, 14-49  
     ceramic, 14-50  
     electrolytic, 14-50  
     mica, 14-49  
     paper, 14-49  
 Capacity, magnetic tape, 5-28  
 Card punches, 5-15, 20-31  
 Cards; see also *Punched cards*  
     magnetic, 20-42  
     punched, 20-30  
 Carnegie Institute of Technology, 2-157, 2-263  
 Carry function, Veitch diagram, 17-22  
 Cascode circuits, transistors, 14-39  
 Case Institute, 2-158  
 Casting out, 3-14, 13-03, 13-07  
 Casualty insurance accounting, 8-08  
 Cathode, followers, 14-29, 17-03  
     temperature, 14-43  
 Cathode ray tubes, analog-digital conversion, 20-48  
     display devices, 5-36, 20-14  
     shaped beam tube, 20-14  
     storage, 19-31  
     Typotron, 20-14  
 Change of control operations, 2-60  
 Change-on-one recording method, 20-39  
 Characteristic roots and vectors, 10-04  
 Character recognition, 5-19  
 Charactron, 5-37, 20-14, 20-16  
 Checker game, 2-246  
 Checking, 5-43, 13-02, 18-32  
     arithmetic, (table) 13-03, 13-06  
     automatic, 2-258, 3-13, 13-03  
     casting out, 3-14, 13-03, 13-07  
     data processing, 3-14  
     data transfers, (table) 13-03, 13-05  
     diagnostic problems, 13-09  
     error detecting-correcting codes, 13-03  
     forbidden combinations, 13-03, 13-06  
     input data, 3-13  
     marginal, 13-09, 14-05, 14-07

- Checking, mathematical, 13-03, 13-05  
 mod 11, 3-14  
 operating, (table) 13-03  
 output data, 3-14  
 parity, 13-03, 13-06, 19-03, 19-07, 19-21  
 programmed, (table) 13-03, 13-04  
 reasonableness, 13-03, 13-04  
 self-checking codes, 13-03, 13-06  
 sums, 13-03, 13-05  
 weighted, 13-03, 13-06
- Checks, see *Checking*; *Digital computers, reliability*
- Chess, 2-246, 11-12
- CHIP, 2-158
- Chopper amplifier, 22-31  
 operation, 22-32
- Circuit design, digital computers, 14-01, 14-13; see also *Components*  
 application notes, 14-44, 14-46, 14-53  
 blocking oscillators, 14-35  
 cascode, 14-39  
 cathode followers, 14-29  
 components, 14-03, 14-43; see also *Components*  
 design criteria, 14-04  
 design philosophy, 14-01  
 diode gates, 14-29, 14-33  
 flip-flops, 14-10, 14-15, 14-37; see also *Flip-flops*  
 magnetic core storage, 19-18; see also *Magnetic cores*  
 marginal checking, 14-05, 14-10  
 pulse amplifiers, 14-21  
 pulse source, 14-21  
 reliability, 13-07, 14-01, 14-19  
 safety margins, 14-04  
 semiconductor diodes, 14-20  
 tolerance plots, 14-06  
 transistors, 14-36; see also *Transistor circuits*  
 vacuum tube gates, 14-26  
 vacuum tubes, 14-20
- Circulating loops, 2-58
- Clock pulse generator, 14-22
- Closed subroutines, 2-162, 2-169
- Code conversion, machine translation, 11-13
- Coded patterns, analog-digital conversion, 20-51
- Coders, 6-11; see also *Programmers*
- Codes, see also *Instructions*; *Numbers*  
 2,4,2,1, 12-20  
 8,4,2,1, 12-21, 20-45, 20-46  
 alphanumeric, 5-05; see also *Instructions*  
 binary, 12-17, (table) 20-46  
 binary coded decimal, 18-25  
 biquinary, 12-21  
 cyclic, 12-17, 20-46, 20-52  
 excess 3, 12-20  
 Gray, 12-17, 20-46  
 Hollerith, 12-08, 12-10  
 IBM, 12-10  
 nonweighted, 12-19  
 one out of ten, 18-26  
 parity check, 12-21  
 punched cards, 12-10  
 reflected, 12-17, 20-46  
 Remington-Rand, 12-08, 12-11  
 self-complementing, 12-20  
 weighted, 12-19
- Code wheel converters, 20-51  
 typical, 20-55, (table) 20-56
- Coding, see *Programming*
- Coding systems, automatic, 2-155, (table) 2-156
- Collating, 3-08
- Collation ratio, 4-06, 4-12
- Collators,  
 magnetic cards, 20-43  
 punched card, 20-32
- Combined systems, analog-digital computing, 30-01  
 automatic programming, 2-163
- Commercial Control Corporation, 20-10, 20-21
- Common language medium, 5-10
- Communications equipment, 7-06
- Comparators, perforated tape, 20-29
- Comparison boxes, 2-132
- Comparison techniques, analog-digital conversion, 20-60
- Compiled logic, 17-40
- COMPILER I, 2-157
- Compilers, 2-186; see also *Fortran*; *IT translator*; *Translators*  
 algorithms, 2-190  
 automatic programming, 2-156  
 building blocks, 2-187  
 filing systems, 2-188

- Compilers, future trends, 2-189  
 Generalized Programming, 2-189  
 indirect addressing, 2-193  
 instruction modification, automatic, 2-190  
 IT translator, 2-200  
 list of, (table) 2-156  
 representation, 2-221  
 Soviet algebraic language, 2-228  
 subroutine hierarchy counters, 2-196  
 theory of algorithms, 2-190  
 USE, 2-164  
 utility programs, 2-184
- Compiler-translator, see *Compilers; Translators*
- Complements, 2-22, 12-22, 17-03, 17-05; see also *Number Systems, complements*
- Completeness, 17-09
- Components, analog, test table, (table) 24-05  
 application notes, 14-44, 14-46, 14-53  
 capacitors, 14-11, 14-49; see also *Capacitors*  
 characteristics, 14-43  
 and circuit design, 14-03  
 connectors, 14-46  
 diodes, 14-45; see also *Diodes*  
 failures, 14-03  
 flip-flops, 14-10  
 inductors, 14-50  
 marginal checking, 14-05, 14-07; see also *Marginal checking*  
 relays, 14-46  
 and reliability, 14-02  
 resistors, 14-11, 14-48; see also *Resistors*  
 transformers, 14-50  
 transistors, 14-51; see also *Transistors*  
 tubes, 14-43; see also *Vacuum tubes*  
 variations, 13-07  
 vulnerability, 14-04
- COMPREHENSIVE, 2-159; see also *CSSR*
- Comprehensive system of service routines, see *CSSR*
- Computation loop, 12-05
- Computer circuits, see *Circuit design, digital computers*
- Computer elements, analog, 22-03  
 Computer elements, mechanical, 27-01  
 passive, 22-04
- Computers, see also *Analog computers; Digital computers*  
 analog, terminology, 1-01  
 analog-digital combined systems, 30-01  
 construction, 12-07  
 definitions, 1-02  
 digital, terminology, 1-01  
 equipment, 12-05  
 interlocks, 30-09  
 operating cost, 20-02  
 reliability, control systems, 14-02  
 terminology, 1-01  
 Turing type, 31-01
- Conditional transfer of control instructions, 5-41, 12-04
- Conducting liquid analogs, 25-06
- Conducting sheet analogs, 25-05, (table) 25-07
- Conductive ink, 5-20
- Conductive rubbers, 25-07
- Conjugate gradient method, 10-04
- Connectors, 14-46
- Consoles, 5-12, 5-40, 12-07
- Constancy interval, 2-51
- Continuous balance method, analog-digital conversion, 20-62
- Control console, 5-12, 5-40
- Control elements, see *Control unit Control Engineering, 2-260*
- Control of a process, machine tool, 29-14  
 operational digital techniques, 29-16, 29-17
- Control in programming, combinations, 2-161, 2-184  
 function, 2-44  
 instructions, 2-160; see also *Instructions*  
 sequential and concurrent, 2-45
- Control systems, and digital computers, 12-01
- Control and timing, analog-digital computer systems, 30-08
- Control unit, 5-02, 12-02, 12-06, 18-01; see also *Arithmetic and control unit*  
 external, 18-34  
 function, 18-33

- Control unit, logical design, 17-26, 18-35  
 programmed, 18-34
- Convair, 2-158
- Conversion, coordinates, 25-03  
 of data, 3-03; see also *Translation, languages*  
 equipment, 5-23  
 of numbers from one system to another, 2-14; see also *Number conversion*
- Converters, analog-digital, 20-44, 29-18, 30-04
- Coordinate systems, field problems, 25-02, 25-10
- Corbie system, 2-185, 2-189
- Core circuits, magnetic, 15-01, 15-05
- Cores, see *Magnetic cores*
- Corporation for Economic and Industry Research, 2-239
- Correlation analyses, multiple linear regression and, 10-10
- Cost, analog-digital conversion, 20-56  
 cathode ray tube display, 20-14  
 code wheel converters, (table) 20-56  
 computing system, 20-02  
 electrophotography, 20-16  
 hand-coded programs, 2-129  
 input-output, 20-02  
 magnetic tape transports, 20-40, (table) 20-42  
 mechanical printers, 20-06, (table) 20-08  
 perforated tape equipment, 20-20  
 photoelectric tape readers, 20-20, (table) 20-27  
 plotters, 20-56  
 printers, 20-08  
 punched card machines, 20-30, 20-31  
 tape punch, 20-20  
 tape readers, 20-26  
 typewriters, 20-10
- Coulomb friction, simulation, 23-27, 23-29
- Counters, binary, 18-03  
 logical design, 17-29, 17-32, 17-37  
 magnetic cores, 15-23  
 scale of 2, 17-07  
 scale of 16, 18-04  
 subroutine hierarchy, 2-195
- Counters, ten state, 17-32  
 transistors, 16-28
- Crossed fields multiplier, 23-10
- Cryogenic films, 19-32
- CSSR, 2-21, 2-159, 2-184
- Curtiss-Wright, 2-157
- Curve followers, 23-14  
 conducting ink type, 23-15  
 manual, 23-14  
 photoelectric, 23-14  
 potentiometer type, 23-14
- Customer billing, 8-11
- Cyclic number systems, 12-17; see also *Gray Code*  
 binary codes, 20-46, 20-52
- Danielewsky's method, 10-04
- Data acquisition, operational digital, 29-17
- DATACODE I, 2-158
- Data conversion equipment, 7-06
- Data handling systems, analog-digital conversion, 20-45
- Data logging, 20-45
- Datamatic 1000, 13-03
- Datamatic Corp., 2-159
- Data processing; see also *Business systems; Data processors*  
 accounting, 8-01, 8-08  
 accuracy control, 8-20  
 billing, 8-13  
 business data manipulation, 3-06  
 characteristics, 5-04  
 checking, 3-13, (table) 13-03  
 collating, 3-08  
 collation ratio, 4-06  
 conversion, 3-03  
 costs, 4-08  
 data collection, 3-02  
 editing, 3-07  
 error rate, 4-07, 4-16  
 files, 3-10, 4-06; see also *Files*  
 information retrieval, see *Information retrieval*  
 information units, business, 3-06  
 input, 4-02, 4-05  
 insurance, 8-01, 8-08  
 interrogation, 3-12  
 language, 3-02  
 medium, 3-02

- Data processing, merge-match, 3-08  
 merging, 3-08  
 off-line, 3-04  
 on-line, 3-04  
 output, 3-04, 4-02, 4-07, 4-14  
 programming, 2-01, 8-14, 8-19  
 real time, 3-05  
 redundancy check methods, (table) 4-15  
 reliability, 4-16  
 reservations, 9-01  
 scanning, 3-09  
 scientific, 3-05  
 sorting, 2-145, 3-07  
 staff, 6-10  
 storage, 4-05, 4-09, 4-11, 4-12  
   access time, (table) 4-13  
 suboperations, 3-05  
 systems analysis, 6-11  
 time, 8-02, 8-09, 8-11, 8-13, (table) 8-19  
 transcription, 3-03  
 transfer checks, (table) 13-03, 13-05  
 transmission, 4-09
- Data processors, 6-01; see also *Data processing*  
 accessibility, 6-03  
 central supply room, 6-04  
 coders, 6-11  
 control console placement, 6-03  
 equipment, 4-04, (table) 4-14, 5-01, 8-01, 8-08, 8-13, 8-16  
 equipment manufacturers' services, 6-02  
 equipment room, 6-03  
 facility requirements, 6-01  
 installation, 6-01, 8-02, 8-08, 8-13  
   floor loading, 6-09  
   soundproofing, 6-09  
 instructional staff, 6-12  
 maintenance, 6-04, 6-13  
 office space, 6-04  
 operators, 6-12  
 personnel requirements, 6-09  
 physical layout, 6-02  
 power requirements, 6-04, 6-05  
 programmers, 6-11  
 refrigeration systems, 6-07  
 tape reels, 6-03  
 training, 6-10, 6-12  
 visitor's area, 6-03, 6-05
- Data reduction, 20-45  
 Data transfer, analog-digital computer systems, 30-13  
 Data transmission, 20-45  
 Datatron, 12-09  
 Datatron 205, 2-26, 2-62, 2-64, 2-186, 2-190, 2-191  
   accounting applications, 8-08  
   alphanumeric code, (table) 2-108  
   B register instructions, 2-105  
   characteristics, (table) 5-42  
   instruction logic, 2-98  
 Datatron 220, 2-62, 2-39  
 D-C amplifier, auxiliary, 22-31  
 drift, 22-17  
   operational, 22-08  
 DCTL, see *Direct-coupled transistor logic*  
 DDA, see *Digital differential analyzer*  
 Dead zone, simulation, 23-25  
 Debugging, computer systems, 7-08  
 Decimal arithmetic, 18-27  
 Decimal codes, 18-25  
 Decimal-hexadecimal conversion, (table) 2-42  
 Decimal operations, 18-25  
 DEC INPUT, 2-159  
 Decision elements, 17-01, 17-02, (tables) 17-03, 17-04  
   one input, (table) 17-03  
   two inputs, (table) 17-04  
 Decision machines, 11-05  
   applications, 11-07  
 Decision operations, 2-59  
   alternative boxes, 2-49  
 Decision procedures, 11-03  
   mechanization, 11-05  
 Decision process, used in translator construction, 2-225  
 Decoder, mechanical, 20-09  
 Deductive procedures, 11-10  
 Delay elements, 17-05  
 Delay line storage, 19-29  
   mercury, 5-26  
 de Morgan's theorems, 17-13  
 Dennison punched price tags, 5-13  
 Dependent variables, analog computers, 22-10  
 Design, see *Circuit design; Logical design*

- Design factors, transistors, 16-04  
 Design philosophy, for reliability, 13-08  
 Design techniques, reliability, 13-02  
 Detroit Edison Company, 8-11  
 Diagnostic problems, 13-09  
 Diagram notation, analog computers, 21-11  
 Difference equation, time, 17-31  
 Differential analyzers, digital, 12-12, 21-04, 28-02, 29-13; see also *Digital differential analyzers*  
   transfer function representation methods, 22-16  
 Differential equations, 10-08, 22-02, 22-13  
   digital differential analyzer solution, 28-03  
   integration, 2-212  
   nonlinear, 27-11  
   ordinary, 10-08  
   ordinary second order, 27-09  
   partial, 10-10  
     field problems, 25-01  
     simultaneous, 27-11  
 Differentiated output, 17-05  
 Differentiation, incremental computation, 29-27  
   repeated, 22-03  
 Diffusion equation, 25-04, 25-17  
 Digital-analog conversion, 20-57, 30-05; see also *Analog-digital conversion*  
   shaft position, 20-61  
   voltage, 20-58  
 Digital analog multiplier, 28-16  
 Digital computers; see also *Data processing*  
   adders, 18-08  
   addresses, 5-07, 12-27  
   alphanumeric, 12-26  
   analog-digital combined systems, 30-01; see also *Analog-digital computing systems*  
   arithmetic, 12-09  
   arithmetic and control unit, 18-01  
   arithmetic and logic unit, 5-02, 5-38  
   binary operations, 18-03  
   brain behavior, simulation, 11-02  
   business, 5-04, 7-05  
   checking, 5-43; see also *Checking; Reliability*  
   Digital computers, chess, 11-12  
   compared with analog computers (table), 21-02, 29-02  
   consoles, 5-12  
   control function, 2-44  
   control unit, 5-02, 5-40, 12-28, 18-01; see also *Arithmetic and control unit*  
   deductive procedures, 11-10  
   design characteristics, 12-25  
   engineering applications, 10-01  
   equipment, 5-01, (table) 5-42  
   externally programmed, 12-29  
   fixed point, 2-23, 12-28  
   fixed word length, 2-22  
   floating point, 2-24, 12-28  
   fundamentals, 12-01  
   general purpose, 5-04, 12-25  
   hybrid systems, 29-01  
   incremental computation, 29-17  
   information representation, 5-05  
   input, 5-02, 5-09, 12-02; see also *Input-output equipment*  
   input-output equipment, 20-01; see also *Input-output equipment*  
   instruction logic, 2-53, 2-63  
   instruction types, 5-08, 5-41  
   internally programmed, 12-29  
   logical, 12-09; see also *Logic, machines*  
   logical design, 17-01; see also *Logical design*  
   magnetic core circuits, 15-01; see also *Magnetic core circuits*  
   magnetic drum storage, see *Magnetic drums*  
   magnetic tapes, 5-18, 20-33  
   mathematical representation, 2-08  
   non-numerical problems, 11-01  
   number triad, (table) 2-64  
   numerical analysis, 10-01; see also *Numerical analysis*  
   on-line, 12-29  
   operating speeds, (table) 2-64, 5-07  
   operational digital techniques, 29-01; see also *Operational digital techniques*  
   organization, 2-04  
   output equipment, (table) 5-33  
   parallel, 12-26  
   point location, 2-23, 12-28

- Digital computers, processing unit, 5-02  
 programming and coding, 2-01; see also *Programming*  
 punched card equipment, 5-13, 20-30  
 punched paper tape, 5-16, 20-19  
 real time, 12-29  
 reliability, 13-01; see also *Checking*;  
*Reliability*  
 scientific, 5-04, 10-01  
 serial, 12-26  
 serial-parallel, 12-26  
 special purpose, 5-04, 9-01, 12-09,  
 12-25  
 storage, 5-24, 19-01, 20-01; see also  
*Storage*  
 and programming, 2-130  
 stored program, 12-29  
 techniques in analog computation, 28-  
 01  
 transistorized, (table) 16-03  
 Turing type, 31-01; see also *Turing*  
*type computers*  
 types, 12-27  
 typical parts, 12-02  
 variable instruction, 2-54  
 variable word length, 2-23, 12-27  
 words, 5-06, 12-27
- Digital control systems, analog-digital  
 conversion, 20-45
- Digital control techniques, 28-17
- Digital differential analyzers, 12-12, 21-04,  
 28-02, 29-13  
 integration, 28-02  
 magnetic drum registers, 28-08  
 mechanization, 28-11  
 operational integrators, 28-03  
 positive and negative increments,  
 28-10  
 scaling, 28-04, 28-06  
 schematic, 28-09  
 serial, 28-08  
 solution of differential equations, 28-03  
 symbols, 28-03
- Digital function generation, analog com-  
 puters, 28-15
- Digital logical operations, 2-59; see also  
*Logical programming*
- Digital numbers, 2-23
- Digital operational techniques, see *Op-  
 erational digital systems*
- Digital sorting, 2-146
- Diodes, 14-09, 14-20, 14-29, 14-45  
 application notes, 14-46  
 capacitor gates, 14-27, 14-29  
 characteristics, 14-45  
 contact potential, 23-23  
 function generators, 23-15, 23-21  
 gates, 14-27  
 germanium, 14-11  
 logic, 14-33  
 magnetic core transfer loops, 15-09  
 protection, 14-34  
 types W and Y, 14-11
- Diode switching, characteristic, thermi-  
 onic, 23-15  
 function generators, 23-23
- Dipole recording method, 20-37
- Direct analog computers, 21-03
- Direct-coupled transistor logic, circuits,  
 16-05
- Discrete analogs, 21-04  
 electric, 25-11
- Discrete plotters, 20-64
- Discrete variable representation, 28-14
- Disjunctive form, 17-13
- Display, cathode ray tube, 5-37
- Dividend accounting, 8-02, 8-04, 8-06
- Divider, see *Division*
- Division, 18-21, 18-29  
 analog computer, 23-12  
 binary methods, 18-21  
 decimal, 18-29  
 incremental computation, 29-26  
 inverse multiplication, 23-12  
 operational digital, 28-13, 29-11
- Document reading equipment, 5-19
- DOUGLAS, 2-156
- Douglas Aircraft Company, 2-156
- Dow Chemical, 2-158
- Down time, 8-02, 8-09
- Drift, analysis, 22-17  
 corrected amplifiers, 22-30  
 d-c amplifier, 22-17  
 errors, 22-36
- Drift free amplification, 22-31
- DRUCO I, 2-157
- DUAL, 2-156
- Dual brush method, analog-digital con-  
 version, 20-52
- Duality, 21-03; see also *Duals*

- Duals, 24-08  
 construction, 24-09  
 physical systems, 24-01  
 relation to analogs, 24-10
- DUMBO, 2-158
- Duplication of the arithmetic section,  
 13-03, 13-06
- Duplication checks, 13-03, 13-04
- DuPont, 20-33
- Dynamic stop, programs, 2-140
- Dynamic system, electric analogy of, 24-01
- Dynamometer multiplier, 23-09
- EASE II, 2-157
- EASIAC, 2-63, 2-111, 2-159, 2-244  
 characters, 2-123  
 demonstration problem, 2-125  
 error printout, 2-125  
 instruction logic, 2-122
- Eastman Kodak Company, 2-156, 11-17,  
 20-13
- EASY FOX, 2-159
- Eccles-Jordan circuit, 14-37, 16-26
- Eckert Mauchly Corporation, 2-83
- Economic evaluation, business systems,  
 7-06, 7-12
- Editing, of data, 3-03, 3-07
- EDSAC, 2-168, 2-180, 2-260, 19-29  
 instruction design, 2-62
- EDSAC II, 2-251
- EDVAC, 2-57
- EIA, 14-11
- Eigenvalues, 10-04
- Eigenvectors, 10-04
- Electric analogy of dynamic system, 24-01
- Electric switching circuits and logic,  
 11-05
- Electrodata, 2-98, 2-158, 2-263, 5-42; see  
 also *Burroughs Corporation; Data-tron*
- Electrofax, 20-17
- Electrography, 20-18
- Electrolyte, properties, 25-08
- Electrolytic models, blotter type, 25-22
- Electrolytic tanks, 25-08  
 accuracy, 25-11  
 double layer, 25-10  
 field plotters, 25-09
- Electronic computers, history, 12-09
- Electronic data processing, see *Data processing; Digital computers*
- Electronic digital equipment, organization of, 5-01
- Electronic Industries Association, 14-11
- Electronic multipliers, 23-05
- Electronic printers, 5-36
- Electro-optical method, 25-23
- Electrophotography, 20-16
- Electrostatic storage, 5-26, 19-31
- Elements, analogs, 24-01  
 electrical, 24-02  
 mechanical, 24-01, 27-01  
 test table, 24-05
- ELI, 2-156, 2-157
- Elimination method, Gaussian, 10-02, 10-03, 10-05
- Emitter follower, transistors, 16-24
- Encoding, problem, 2-10  
 self, 2-11
- Energy analogy, 24-07
- Engineering problems, analog computers,  
 21-06  
 digital computers, 10-01
- Engineering Research Associates, 19-06;  
 see also *ERA*
- Eniac, 12-09, 19-29
- Ensemble, average, 26-03  
 mean square, 26-06, 26-02  
 noise generators, 26-02
- Entry, subroutine, 2-140
- Environment and reliability, 13-08
- Equations, solution of, 27-09
- Equipment, accounting applications, 8-01,  
 8-08, 8-13, 8-16  
 acquisition, 7-07  
 arithmetic and logic unit, 5-38  
 data processing, (table) 4-04, 4-09,  
 (table) 4-14  
 electronic digital, 5-01  
 input-output, 20-01  
 manufacturers' services, 6-02  
 reservations, 9-04  
 selection for business systems, 7-02,  
 7-04  
 typical, (table) 5-42
- Equipotential lines, 25-02
- Equitable Life, 2-156, 2-157
- ERA 1101, drum, 19-07

- ERA 1103 and 1103A, see *Sperry Rand Corporation; Univac*  
 magnetic heads, 19-08
- Ergodic processes, 26-03
- Error function, 26-05
- Errors, checking, (table) 4-15  
 computing elements, 22-33  
 detecting-correcting code, 13-03  
 incremental computation, 29-24  
 IT translator, 2-219  
 printout, EASAC, 2-125  
 programming, 2-128  
 and reliability, 4-07, 4-16
- ESCAPE, 2-157
- Euler process, 2-213
- Excess 3 code, (table) 12-20
- Exclusive or*, 17-04, 17-12  
 magnetic cores, 15-18  
 Veitch diagram, 17-14
- Exit, subroutine, 2-171
- Extension of the IT language, example, 2-214
- External control, 18-34
- External storage requirements, input-output, 20-05
- Extract, 18-31
- Facility requirements, 6-01
- Factor analysis, 10-11
- FAP, 2-157
- Feedback, analog-digital conversion, 20-47, 20-60  
 control, 20-45  
 d-c amplifier, 22-08  
 impedance, 22-08
- Ferramics, 19-23
- Ferranti Ltd., 2-159, 20-28  
 Mark I, 2-190  
 Mercury, 2-186
- Ferrite, cores, 15-07, 19-23  
 plates, 19-29
- Ferroelectric storage, 19-32
- Field, 3-06
- Field problems, 25-01  
 across variable, 25-02  
 analogs, 25-05  
 coordinate systems, 25-02  
 discrete element electric analogs, 25-11  
 electronic, 25-21  
 equations, 25-04, 25-16, 25-18
- Field problems, finite difference expansions, 25-11, (tables) 25-12, 25-14  
 infinite fields, 25-10  
 networks, 25-15  
 nonelectric analogs, 25-22  
 operators, 25-02  
 plotters, 25-09  
 through variable, 25-02
- Files, 3-07  
 access requirements, 4-06  
 flexible, 2-188  
 magnetic cards, 20-43  
 magnetic tape, 8-02  
 maintenance, 3-09  
 problem of, 2-03  
 processing, 3-10  
 size, 4-06
- Finite difference expansions, 25-11, (tables) 25-12
- Fixed point, 2-23, 12-28, 18-32  
 scale factors, 2-24
- FLAC, 2-192
- FLAIR, 2-157
- Flexowriter, 2-109, 20-10
- Flip-flops, 17-05  
 component specifications, 14-11  
 delay, 17-34, 18-02  
 dynamic, 17-05  
 high speed, 14-10, 14-15  
 transistor, 14-37, 14-38  
 indicators, 14-32  
 J-K, 17-08, 17-36  
 logic, 18-02  
 low speed, 14-28  
 relation to outputs, (table) 17-06  
 reliability, 14-17  
 R-S, 17-08, 17-35, 18-02  
 R-S-T, 17-08, 17-36  
 saturation, 16-07  
 set-reset, 18-02, 31-07  
 states, 17-05  
 symbols, 17-30, 18-03  
 transistors, 16-26  
 trigger, 17-07, 17-34, 18-02  
 truth tables, 18-03
- FLIP/SPUR, 2-158
- Floating addresses, 2-161
- Floating points, 2-24, 12-28, 18-32
- Floating variables, 2-51, 2-168
- FLOP, 2-156

- Flow diagrams, 2-47
  - boxes, 2-49
  - compiler language, IT translator, 2-207
  - operations, 2-49
  - programming, 2-47
- FLOW-MATIC compiler, 2-259
- Fluid mappers, 25-22
- Fluorescent ink, 5-20
- Forbidden combinations, 13-03, 13-06
- FORC, 2-156
- Formula translator, see *FORTTRAN*
- FORTTRAN, 2-129, 2-156, 2-157, 2-186, 2-187, 2-190, 2-228, 2-260 (650T), 2-157
  - characters, used with IT translator, 2-202
  - compiler, 2-259
    - language, 2-245
- FOR TRANSIT, 2-157
- Four address codes, 2-56
- Fourier transforms, 26-05
- Four-quadrant multiplication, 23-01, 23-04
- Franklin Institute, 2-158
- Free derivation, 11-10
- Free variables, 2-50, 2-168
- Frequency doubling recording method, 20-40
- Frequency response, analog-digital conversion, 20-48
  - function generator, 23-16
  - quarter square multiplier, 23-08
  - servo multiplier, 23-04
- Function generators, 23-14
  - biased diode, 23-15
  - digital, 28-15
  - interpolation techniques, 23-20
  - mechanical, 27-05, (table) 27-06
  - multiplication, 23-13
  - photoformer, 23-19
  - potentiometers, 23-17
  - resistive materials, 23-21
  - scaling, 23-19
  - table, 28-16
  - trigonometric, 23-21
  - two variables, 23-20
  - uses, 23-20
  - variable-breakpoint diode, 23-15
  - variable density film, 23-22
  - variable reference diode, 23-21
  - variations of diode techniques, 23-16
- G-15, see *Bendix G-15*
- Gain, errors, 22-33
  - integrator, 22-07, 22-10
- Game playing machines, 11-11
- Games, 11-11
  - chess, 11-12
  - and logic, 11-11
  - machine solution, 11-11
- GAMM, 2-186
- Gamma 60, 2-56
- Gas discharge tube, 26-13
- Gates, combined, transistors, 16-11
  - diode capacitor, 14-27, 14-29
  - intrinsic, 17-07
  - pulse, high speed, 14-42
  - symbols, 17-02
  - transistors, 16-09, 16-18
  - vacuum tube, 14-26
- Gaussian distribution, 26-04
- Gaussian elimination method, 10-02, 10-05
- Gaussian noise generator, 26-11
- Gauss-Seidel method, 10-04
- Geiger-Mueller tube, 26-12, 26-13
- General Ceramics Corporation, 19-23
- General Dynamics Corporation, 20-14, 20-16
- General Electric Company, 2-156, 2-157, 20-17
- Generalized programming, see *GP compiler*
- Generalized programming extended (GPX), 2-156, 2-189
- General Motors Corporation, 2-157, 2-185, 2-263
- General purpose computers, analog, 21-02
  - digital, 5-04, 12-25
- Generators; see also *Subroutine, generators*
  - automatic programming, 2-163
  - subroutine, 2-182
- German-Swiss Applied Mathematics Society, 2-186
- Gill method, 10-09
- Glossary of terminology, computers, 1-02
- GP compiler, 2-158, 2-189, 2-190, 2-245, 2-260
- GPX compiler, 2-189

- Gray code, 12-17, 18-24  
 conversion, 18-24
- Greenstadt's method, 10-06
- Grid spacing, 14-43
- Guebard's rings, 25-23
- Guide, 2-157
- Half adder, 15-22, 18-03, 18-08  
 transistors, 16-11
- Haloid Company, 20-16
- Hand programming, 2-02  
 traditional, 2-128
- Harvard University, 12-07, 12-09, 12-29
- Heads, design, 20-34  
 magnetic drum, 19-07
- Heuristics, 11-10, 11-12
- Hexadecimal, decimal conversion, (table)  
 2-42  
 multiplication, (table) 2-44
- Hierarchy counter, permanent subroutine, 2-195
- Hollerith code, 2-165, 12-08, 12-10
- Holloman Air Force Base, 2-157
- Hughes Aircraft Company, 20-14
- Hybrid systems, 28-11, 29-01, 29-04
- Ianov, string language, 2-230  
 transformation rules, example, 2-233
- IBM, 2-156, 2-157, 2-160, 2-263, 12-07
- IBM 026 key punch, 2-201
- IBM 305 magnetic disk, 19-13
- IBM 407 printer, 2-201, 8-08
- IBM 528 reproducer, 8-08
- IBM 604, 16-20, 16-28
- IBM 608, 16-03, 16-20  
 characteristics, 16-23  
 packaging, 16-28  
 power supplies, 16-30
- IBM 650, 2-58, 2-64, 2-134, 2-136, 2-143,  
 2-157, 2-186, 2-200, 2-219, 2-239,  
 2-241, 13-03  
 accounting applications, 8-13  
 instruction logic, 2-93  
 IT translator, 2-201  
 magnetic drum, 19-05  
 optimum programming, 8-14  
 programming examples, 2-130  
 scheduling, 9-12  
 SOAP programs, 2-131  
 multiway switch, 2-138
- IBM 650, SOAP programs, vector inner  
 product, (table) 2-134  
 sorting examples, 2-153
- IBM 700 series, 2-58, 12-09
- IBM 701, 2-156, 2-162, 2-186, 2-236, 2-239,  
 2-250  
 instructions, 2-62
- IBM 702, 2-56, 2-157  
 accounting applications, 8-16, (table)  
 8-18
- IBM 704, 2-20, 2-27, 2-62, 2-63, (table)  
 2-64, 2-156, 2-165, 2-186, 2-187,  
 2-189, 2-191, 2-230, 2-239, 2-241,  
 2-246  
 instruction logic, 2-63  
 logical programming, 2-248  
 SAP program, 2-167
- IBM 705, 2-56, 2-64, 8-11  
 characteristics, (table) 5-42  
 magnetic core storage, 19-27
- IBM 709, 2-62, 2-157, 2-186, 2-193
- IBM card program calculator, 12-08,  
 12-29
- IBM cards, 5-13, 20-30, 20-33
- IBM Programming Research, 2-239
- IBM punched cards, 5-13, 12-10, 20-30
- IBM STRETCH, 2-56, 2-62, 2-191
- IBM typewriter action, 20-08
- ILLIAC, 2-21, 2-59, 2-159, 12-09
- Inclusive or*, 17-12
- Incremental computation, 29-17  
 arithmetic, 29-21  
 control applications, 29-18  
 digital differential analyzer, 28-10, 29-13  
 machine organization, 29-19  
 operations, 29-25  
 program, 29-20  
 storage requirements, 29-18  
 transient solutions, 29-28
- Incremental methods, 10-09
- Incremental pattern methods, analog-  
 digital conversion, 20-56
- Index accumulators, see *Index registers*
- Indexing, operations, 2-60  
 systems, 11-16
- Index registers, 2-55, 2-60, 2-190  
 Intercom, 2-241  
 MIDAC, 2-115  
 programming, 2-136
- Indicial boxes, programming, 2-133

- Indicial instruction logic, 2-54
- Indirect addressing, 2-199
- Indirect analog, 21-03
- Indirect methods, analog-digital conversion, 20-47, 20-57
- Inductance-capacitance network, 25-20
- Inductors, 14-50
- Inference rules, 11-10
- Infinite fields, simulation, 25-10
- Information, averaging, 29-15
  - business units of, 3-06
  - flow patterns, 7-02, 7-03
  - retrieval,
    - category search, 3-12
    - externally controlled, 3-11
    - internally controlled, 3-11
    - interrogation, 3-12
- Inhibitor* circuit, 16-18
- Initial conditions, analog computers, 22-10
- Input impedances, 22-08
- Input programs, 2-183
- Input-output, instructions, see *Instructions*
  - operations, 2-60
- Input-output equipment, 5-09, 7-06, 12-02, 20-01
  - analog-digital computer systems, 30-09
  - analog-digital conversion, 20-44; see also *Analog-digital conversion*
  - automatic control, 20-03
  - auxiliary, 5-22
  - buffer storage, 5-23, 17-25, 20-03
  - cathode ray tube display, 5-36, 20-14
  - consoles, 5-12
  - conversion, 5-23
  - cost, 20-02
  - document reading, 5-19
  - electric typewriters, 5-12, 20-07
  - electrography, 20-18
  - external storage, 20-05
  - general requirements, 20-03
  - inquiry units, 5-13
  - keyboards, 5-10, 20-06
  - magnetic cards, 20-42
  - magnetic tape, 5-18, 20-33
  - magnetography, 20-17
  - mechanical printers characteristics, 20-06, (table) 20-08
  - multiplexing, 5-23
- Input-output equipment, off-line equipment, 20-04
  - on-line equipment, 20-04
  - page reading equipment, 20-06
  - perforated tape, 5-16, 20-19; see also *Perforated tape*
  - photoelectric readers, 20-20, 20-27
  - photography, 20-14
  - plotters, 20-63
  - preparation, automatic, 5-11
    - key-driven, 5-10
  - print readers, 5-21
  - printed page, 20-06
  - printers, 20-06, 20-08; see also *Printers*
  - punched card machines, 5-13, 20-30
  - recording media, 5-09, 20-05
  - typewriters, 20-07, (table) 20-08, 20-10
  - verifiers, 5-12
  - xerography, 20-16
- Inputs, analog computer, 21-05
  - tables, 27-04
  - data check, 3-13
  - data processors, 4-02
  - media, 5-09, (table) 5-10, 20-05
  - simulated, 17-39
  - time difference equations, 17-31
- Inquiry units, 5-13
- Installation, costs, 4-08
  - data processors, 6-01
  - time, 4-09
- Institute for Advanced Study, 2-47, 12-09
- Institute of Radio Engineers, see *IRE*
- Instruction counter, MIDAC, 2-117
- Instruction logic, ambiguous word, 2-54
  - common computers, 2-63
  - Datatron 205, 2-98
  - EASIAC, 2-122
  - IBM 650, 2-93
  - IBM 704, 2-63
  - indicial, 2-54
  - Intercom, 2-241
  - MIDAC, 2-115
  - Royal-McBee LGP-30, 2-109
  - Strela, 2-111
  - TX-0, 2-252
  - Univac II, 2-83
  - Univac 1103A, 2-77
- Instruction modification, automatic, 2-190
  - digits, MIDAC, 2-116

- Instruction modification, types, 2-191  
 Instructions; see also *Instruction logic*  
   accounting, 8-14  
   addresses, number, 2-56, 2-64, 5-07, 12-27  
   arithmetic, Datatron 205, 2-99  
     IBM 650, 2-94, 2-96  
     IBM 704, 2-66  
     LGP-30, 2-110  
     Univac II, 2-85  
     Univac 1103A, 2-81  
   arithmetic and logical, Strela, 2-111  
   branching, IBM 650, 2-96  
   breakpoint, Univac II, 2-91  
   change control, Datatron 205, 2-103  
     MIDAC, 2-120  
   change of information, MIDAC, 2-119  
   common computers, 2-63  
   conditional jump, Univac 1103A, 2-80  
   console, 12-05  
   control, 2-160  
     IBM 704, 2-71  
     Strela, 2-112  
     Univac II, 2-87  
   control-record, Datatron 205, 2-104  
   execution of, 12-04  
   extract, Univac II, 2-87  
   floating point, Univac 1103A, 2-82  
   format, 12-03  
   four address, 2-56  
   indexing, IBM 704, 2-74  
   input-output, 12-05  
     Datatron 205, 2-105  
     IBM 650, 2-93  
     Univac II, 2-89  
     Univac 1103A, 2-81  
   jump, Univac 1103A, 2-80  
   logical, 12-05  
     Datatron 205, 2-101  
     IBM 704, 2-67  
   and logical design, 17-26  
   loop transfer, Datatron 205, 2-103  
   one plus one addresses, 2-58, 2-130  
   operation types, 2-59  
   shift, Datatron 205, 2-101  
     IBM 650, 2-97  
     Univac II, 2-88  
   single address, 2-58  
   storage instructions, IBM 650, 2-95  
   structure, 2-56
- Instructions, subroutine, Strela, 2-113  
   three address, 2-57  
   transfer, MIDAC, 2-120  
     Strela, 2-112  
     Univac II, 2-84, 2-89  
     Univac 1103A, 2-79  
   two address, 2-58  
   types, 2-61, 5-08, 5-40, (table) 5-41, 12-05  
   variable, 2-54
- Integrated systems, 2-07, 2-184  
   automatic features, 2-185  
   interrupt features, 2-185
- Integration, analog, 22-04  
   analog-digital computer systems, 30-15  
   dependent variable, 23-13  
   digital differential analyzer, 28-02, 28-11  
   digital operational, 28-14  
   incremental computation, 29-27  
   Kelvin disk, 27-03  
   mechanical, 27-03  
   network, 22-07  
   repeated, 22-02  
   scaling, 28-06
- Integrators, 22-07, 22-09; see also *Integration*  
   gain, 22-07, 22-10  
   time constants, 22-07, 22-10, 22-36
- Intercom system, 2-63, 2-241  
   on the Bendix G-15, 2-241  
   comparisons, 2-244  
   instruction list, (table) 2-243  
   language, 2-242
- Interlace, magnetic drum, 19-05
- Interlocks, analog-digital computer systems, 30-09
- Internal excitation, resistance networks with, 25-17
- Internal translator, see *IT translator*
- International Business Machines, see *IBM*
- International Telemeter Corporation, 19-27
- Interpolation techniques, function generator, 23-20
- Interpreters, 2-235  
   acceptance tests, 2-239  
   automatic programming, 2-160  
   evaluation, 2-244

- Interpreters, Intercom, 2-241
  - interpretation process, 2-236
  - memory, 2-241
  - punched card, 20-32
  - routines, 2-235
  - simulation of one computer by another, 2-236, 2-239
  - tracing programs, 2-240
  - Turing machines, 2-235, 31-01
  - utility programs, 2-184
- Interprete programming language, 2-245
- Interrogation, 3-12
- Inventory, applications, 9-01
- Inverters, 17-03
  - transistors, 16-23
- IPL, 2-245
- IRE, 1-02, 14-54
- IT compiler-translator, 2-129, 2-156, 2-157, 2-186, 2-187, 2-190, 2-200, 2-244, 2-245, 2-259
  - error cards, 2-219
  - extension operands and statements, 2-187
  - flow diagrams in compiler language, 2-207
  - iteration statements, 2-206
  - language, characters of, 2-202
    - extension of, 2-214
    - formal representation, 2-223
    - rules, 2-201
    - run request, 2-219
    - statement forms, 2-205
  - one-pass version, 2-201
  - SOAP, 2-209, 2-214
  - statements, 2-201
  - subroutine, Runge-Kutta-Gill, 2-189
  - subroutine library, 2-214
  - two-pass version, 2-201
- Item, of information, 3-06
- Iteration loops, 2-45
- Iterative methods, matrix inversion, 10-02
  
- Jacobi's method, 10-06
- JCS 13, 2-156
- J-K flip-flops, 17-08, 17-36; see also *Flip-flop, trigger*
- Johnniac, 12-09
  - magnetic core storage, 19-27
- Joint Computer Conferences, 2-261
- Jordan method, 10-02, 10-03
- Jump instructions, 12-04
- Junction method, 24-04, 24-05
  - construction of duals, 24-09
- Junction transistors, 14-51, 14-52
  
- K5, 2-158
- Kalin-Burkhart computer, 12-09
- Kelvin disk integrators, 27-03
- Kerr constant, 25-23
- Key, 3-06
- Keyboards, 5-10, 20-06
  - byproducts, 5-11
  - special, 5-13
  - speed, 5-11
- Key punches, perforated tape, 20-29
  - punched card, 20-31
- Kimball, punched price tags, 5-13
- Kirchhoff adder, 18-10
- Kirchhoff's laws, 24-04, 25-11
- KOMPILER 2, 3, 2-156
  
- Languages, 2-186
  - algebraic, Soviet compiler, 2-228
  - automatic programming, 2-160, 2-186
  - common, 5-13
  - data processing, 3-02
  - English, structural decomposition, 2-259
  - extensions, 2-187
  - formal, programming, 2-05
  - input, easy-to-correct and easy-to-use, 2-160
  - interpretive program, 2-245
- Intercom, 2-242
  - machine level, 2-56
  - machine translation, 11-13
  - natural, programming with, 2-259
    - commands, 2-259
    - storage requirements, 2-259
  - recursive, 2-244
  - string, 2-190
  - target, 2-186
  - translation, 11-13
  - universal computer, 2-163
  - universal programming, 2-200
- Laplace's equation, 25-04, 25-22
- Laplace transforms, 22-07
  - solution, 25-05

- Laplacian operator, 25-02  
 Larc, see *Univac Larc*  
 Latch, 17-06  
 LGP-30, see *Royal McBee LGP-30*  
 Liapounov-Ianov string programming, 2-230, 2-233  
 Librascope, 2-109, 2-263, 12-09; see also *Royal-McBee*  
 Lichtenberger figures, 25-23  
 Life insurance accounting, 8-01  
 Limiting, 23-23  
   computer representation, 23-23  
   generalized, 23-24  
   integrator outputs, 23-25  
 Lincoln Laboratory, 2-252, 14-01, 14-02, 19-29  
 Linear computing elements, 22-01  
   symbols, 22-04  
 Linear interpolation, potentiometers, 23-17  
 Linearity, potentiometer, 22-04  
 Linear matrix equation solver, 2-174  
 Linear programming, 10-06  
   simplex method, 10-07  
   time for solution, 10-07, 10-08  
   transportation problem, 10-08  
 Linear systems, noise, 26-06  
 Line-a-time printers, 5-33, 5-34, 20-08, 20-10  
 Linkage mechanisms, 27-01  
 Literature searching, 11-16  
 Load correction, potentiometer, 22-05  
 Loading, errors, 22-33  
   potentiometer, 22-04  
 Location counter, 2-164  
 Lockheed Aircraft Corporation, 2-156  
   Missile Systems Division, 2-157  
 Logarithmic multiplier, 23-09  
 Logic; see also *Logical programming*  
   circuits, 14-34  
   connectives, 11-03, 11-04  
   decision procedure, 11-03  
   deductive procedures, 11-10  
   on digital computer, 2-246, 11-02  
   evaluation procedures, 11-03  
   free derivation, 11-03, 11-10  
   and games, 11-11  
   inference rules, 11-10  
   machines, 2-53, 11-02  
   applications, 11-07  
   Logic, machines, automatic programming, 11-08  
   limitations, 11-09  
   and numbers, 2-12  
   programming, 2-02  
   proofs, 11-03, 11-10  
   and switching circuits, 11-05  
   tautological character of expressions, 11-03  
   truth tables, 11-04  
 Logical design, 17-01, 17-30  
   algebraic techniques, 17-10, 17-28, 29-21  
   arithmetic and control unit, 17-27  
   block diagrams, 17-28  
   Boolean algebra, equivalences, (table) 17-11  
   completeness, 17-09  
   control unit, 18-35  
   counter, 17-29, 17-32  
   decision elements, 17-02, (tables) 17-03, 17-04  
   difference equations, 17-31  
   elements, 17-01  
   magnetic cores, 15-03, 15-16  
   transistors, 16-28  
   incremental computer, 29-21  
   instructions, 17-26  
   manual control, 17-27  
   matching input and output, 17-25  
   minterms, 17-12  
   number representation, 17-26  
   simplification, inspection, 17-15  
   Quine method, 17-16  
   simulation, 2-236, 2-239, 17-38, 17-40  
   storage, 17-25  
   switching functions, 17-10, 17-15  
   symbols, (table) 17-02  
   time difference equations, 17-31  
   truth tables, 17-11  
   Turing computers, 31-03  
   Veitch diagrams, 17-13  
 Logical gain, transistor circuits, 16-05, 16-07  
 Logical instructions, see *Instructions*  
 Logical operators, 2-229  
 Logical product, 17-12  
 Logical programming, 2-246  
   checkers, 2-246  
   chess, 2-246

- Logical programming, IBM 701, 2-250
  - IBM 704, 2-248
  - MIDAC, 2-250
  - problem in the propositional calculus, 2-246
  - Univac 1103A, 2-247
- Logical simulation, 2-236, 2-239, 17-38
- Logical sum, 17-12
- Logical transfer, 18-31
- Loop control, 2-46, 2-54
- Loops, iteration, 2-45
  - magnetic core transfer, 15-09
- Los Alamos, 2-156
  
- Machine language programs, 2-131
- Machine tool control, 29-14
- Machine translation, code conversion, 11-13
  - languages, 11-13
  - problems, 11-14, 11-15
- MAGIC, 2-20, 2-131, 2-159, 2-184, 2-186
- Magic number, in number conversion, 2-18
- MAGIC program, dynamic stop, 2-140
  - multiway switch, 2-139
  - vector inner product, 2-136
- Magnacord, 20-43
- Magnavox Company, 20-43
- Magnetic card equipment, 20-43
- Magnetic coatings, (table) 19-09
- Magnetic cores, 5-26, 15-01, 19-13
  - adder, 15-22
  - ambient temperature, 19-26
  - arithmetic, 15-22
  - characteristics, (table) 19-23
  - circuit design, 15-01, 15-05
  - counters, 15-23
  - current regulation, 19-26
  - cycle distributor, 15-21
  - design, 19-21, (table) 19-23
  - drivers, 15-23
  - ferrite, 15-07
  - films, 19-28
  - half-select outputs, 19-24
  - hysteresis loop, 15-02
  - logic, 15-16
  - metal-strip types, 15-04
  - molded ferrites, 15-04
  - multiaperture, 19-29
  - multicoordinate, 19-15
- Magnetic cores, noise in bistable, 15-07
  - nondestructive readout, 19-26
  - nondigital applications, 15-23
  - parameters, 15-07
  - plane, 19-22
  - plates, 19-29
  - principles of operation, 19-13
  - read circuits, 19-13, 19-20
  - reliability, 14-03
  - selection circuits, 15-19, 19-19
  - selection systems, (table) 19-15
  - sense windings, 19-25
  - shift registers, 15-15, 15-19
  - squareness ratio, 19-24
  - stability, 19-26
  - storage, 15-19, 19-13
  - switches, 19-16
  - switching time, 19-23
  - switching voltages, 15-06
  - symbols, 15-01, 15-03
  - systems, 19-16, 19-26, (table) 19-27
  - temperature stability, 15-09
  - thin films, 19-28
  - three-coordinate, 19-13
  - timing control circuits, 15-21
  - transfer loops, 15-09
  - transistor multiplier, 29-08
  - two-coordinate, 19-13
  - waveforms, 15-03, 15-06
  - wiring, 19-22
  - writing, 19-15
- Magnetic disk storage, 5-32, 19-13
- Magnetic drums, 5-26, 5-32, 19-04
  - access time, 19-05
  - coatings, 19-07, (table) 19-09
  - construction, 19-07
  - digital differential analyzer, 28-08
  - forced coding, 19-06
  - heads, 19-07
  - interlace, 19-05
  - parity check, 19-07
  - power loss, 19-11
  - reading circuits, 19-11
  - recording, 19-09; see also *Magnetic tapes, recording*
  - Manchester, 19-10
  - nonreturn to zero, invert (NRZI), 19-10, 19-13
  - nonreturn to zero (NRZ), 19-09
  - return to bias, 19-09

- Magnetic drums, recording, return to zero (RZ), 19-09  
 reliability, 19-12  
 typical operating systems (table), 19-07  
 writing techniques, 19-09, 19-11
- Magnetic films, 19-28
- Magnetic heads, 20-33, (table) 20-34  
 characteristics, (table) 20-34  
 stack, (table) 20-34
- Magnetic ink, 5-20, 5-22
- Magnetic recording densities, 20-40
- Magnetic storage, other techniques, 19-26
- Magnetic tapes, 5-25, 20-33  
 equipment, 5-18, 20-33  
   auxiliary, 20-42  
 features, 5-27, 5-30  
 head design, 20-33  
 input units, 5-19  
 master file, accounting, 8-02  
 operation details, 5-27  
 output equipment, 5-36  
 plastic base, characteristics, 20-33, (table) 20-34  
 preparation, 5-18  
 random access, 5-32  
 reading methods, 5-18, 20-35  
 recording, 5-18, 5-25, 20-35; see also *Magnetic drums, recording*  
   dipole, 20-37  
   frequency doubling, 20-40  
   heads, 20-33, (table) 20-34  
   Manchester, 20-39  
   nonreturn to zero (NRZ), 20-38  
   nonreturn to zero, invert (NRZI), 20-39  
   phase modulation, 20-39  
   polarized dipole, 20-34  
   return to zero (RZ), 20-36  
   waveforms, 20-36, 20-38, 20-40  
 sorting, 2-152  
 speeds, 5-19, 5-29  
 storage capacity, 5-28  
 time delay, 23-34  
 transports, 5-29, 20-40, (table) 20-42  
 units, 5-19  
 writing currents, 20-40
- Magnetography, 20-17
- Magnetostrictive delay lines, 19-31
- Magnetronic Reservoir, 9-04
- Maintenance, 6-13, 13-08  
 console, 13-09  
 programs, 2-258  
 scheduled, 13-08  
 unscheduled, 13-08
- Manchester recording method, 19-10, 20-39
- MANIAC, 2-62, 12-09
- Manpower forecasts, 9-07
- Manual control, logical design, 17-27
- Manuals, installation and specification, 6-02
- Marginal checking, 13-09, 14-05, 14-07  
 circuits, 14-10  
 components, 14-07  
 reliability data, 14-17  
 transistors, 14-36
- Mark I, 12-07, 12-29
- Mark II, 12-09
- Mark sensing, 5-19
- Match-merge, 3-08
- Material equivalence function, 15-19
- Mathematical checks, 13-03, 13-05
- Math-Matic, 2-129, 2-186
- Matrix inversion, 10-02  
 methods, 10-02, (table) 10-03  
 storage requirements, 10-03  
 time required, (table) 10-03
- MATRIX MATH, 2-158
- Matrix order, 10-03, 10-06
- Matrix printers, 20-08, 20-13
- Maximum storage, computers, (table) 2-64
- Maxterms, 17-12
- Mean square ensemble output, 26-06, 26-09
- Mean square value, determination of, 26-20
- Mean value, determination of, 26-18
- Measures of performance, programming, 2-05
- Mechanical computers, 27-01  
 accuracy, 27-02  
 addition and subtraction, 27-02  
 digital, 12-07  
 and electronic, 27-02  
 function generation, 27-05, (table) 27-06  
 input tables, 27-04

- Mechanical computers, integration, 27-03  
multiplication by a constant, 27-02  
output tables, 27-04  
scale factors, 27-14  
schematics, 27-05  
solution of equations, 27-09  
symbols, 27-05  
torque amplifiers, 27-04
- Mechanical elements, 24-01, 24-09  
analog-digital converters, 30-04
- Mechanical perforated tape readers, 20-26
- Mechanical printers, 20-06, (table) 20-08
- Mechanical translation, 2-259
- Mechanism, linkage, 27-01
- Mercury, 2-186
- Mercury delay line, 5-26, 19-30
- Merging, 2-145, 2-147, 3-08
- Michigan Digital Automatic Computer, 2-115; see also *MAGIC*
- Microprogramming, 2-251  
example, 2-256  
macroinstructions and microinstructions, 2-254  
TX-0 development, 2-252
- MIDAC, 2-20, 2-58, 2-64, 2-122, 2-159, 2-171, 2-192, 2-244; see also *MAGIC*  
base counter, 2-118  
instruction logic, 2-115  
logical programming, 2-250  
modification digits, 2-116  
programming examples, 2-130  
subroutine library, 2-172
- Mid-square procedure, 2-145
- Milne, Adams and, method, 10-09
- Minicard, 11-17
- Minimal latency, programming, 2-130
- Minnesota Mining and Manufacturing Company, 19-09
- Minterms, 17-12
- MISHAP, 2-157
- Mistake prevention, automatic programming, 2-163
- M.I.T., 2-21, 2-157, 2-159, 2-160, 2-184, 2-251, 2-252, 2-263, 14-01, 19-06, 19-27  
magnetic core unit, 19-19  
TX-2 core storage, (table) 19-27  
Whirlwind, 2-159
- MITILAC, 2-157
- MJS, 2-158
- Mnemonic codes, 2-160
- Mod 11 check, 3-14
- Modus ponens, 11-10
- Monitoring, digital, 30-14
- Monostable storage elements, 17-05
- Moore School, 2-159
- M-Q register, 12-06
- Multi-address codes, 2-56
- Multiaperture magnetic cores, 19-29
- Multicoordinate selection principles, magnetic cores, 19-15
- Multiple linear regression and correlation analyses, 10-10
- Multiplexing equipment, 5-23
- Multiplication, analog, 23-01  
accuracy, 23-04, 23-07, 23-09  
AM-FM, 23-19  
by constant, 22-04, 27-02  
crossed field, 23-10  
dynamometer, 23-09  
four quadrant, 23-02, 23-04  
logarithmic, 23-09  
mechanical, 27-02  
modification of quarter squares, 23-09  
one-quadrant, 23-02  
probability, 23-10  
quarter squares, 23-08  
scaling, 23-11  
servo, 23-02  
step relay, 23-10  
time division electronic, 23-05  
two-quadrant, 23-02
- digital, 18-16, 18-27  
adding and shifting method, 18-28  
binary, 18-16  
Booth method, 18-18  
decimal, 18-27  
hexadecimal, (table) 2-44  
incremental computation, 29-26  
parallel, 18-19  
precision, 18-20  
roundoff, 18-29  
roundoff correction, 18-20  
Russian peasant method, 18-28  
serial, 18-19  
serial-parallel, 18-19  
Shaw method, 18-16

- Multiplication, digital, stored multiples,  
 18-28  
 table, 18-29  
 digital analog, 28-16  
 digital operational, see *operational digital*  
 operational digital, 28-11, 29-05  
 accuracy, 29-08  
 binary rate, 29-05  
 dynamic rate, 29-09  
 magnetic core transistor, 29-08  
 pulse rates, 29-10  
 Multiplier, see *Multiplication*  
 Multiprecision, 2-22  
 Multispeed coders, 20-55  
 Multivibrator, one-shot, 16-08  
 Multiway switch, IBM 650 SOAP program, 2-138, 2-139  
 MIDAC MAGIC program, 2-139  
 Munich Technische Hochschule, 2-191  
 Mylar tape, 5-25, 20-33, 20-43
- NAA ASSEMBLY, 2-156  
 National Advisory Committee on Aeronautics, 2-193  
 National Bureau of Standards, 2-115, 2-159, 2-185, 2-189, 2-240  
 National Cash Register, 12-09  
 Natural languages, programming, 2-259  
 Naval Ordnance Laboratories, 2-157, 2-159  
 Naval Ordnance Test Station, 2-156  
*Negation*, 17-03  
 magnetic cores, 15-18  
 Networks, a-c analyzers, 25-20  
 four-terminal, 22-16  
 inductance-capacitance, 25-20  
 integrating, 22-07  
 resistance, 25-15  
 resistance-capacitance, 25-19  
*R-L-C*, 25-20  
 short circuit impedance, (table) 22-18  
 summing, 22-05  
 two-terminal, 22-15  
 Newton's method, 2-212  
 New York University, 2-158  
 Nodal point location, finite difference expansions, (tables) 25-12  
 Noise, 26-01; see also *Noise generators*  
 adjoint systems, 26-07  
 aerodynamic, 26-10  
 aircraft systems, 26-10  
 analog computers, 26-01  
 contact, 26-01  
 ensemble, 26-06, 26-09  
 average, 26-03  
 ergodic processes, 26-03  
 errors, 22-36  
 Gaussian distribution, 26-04  
 generators, 26-12  
 linear systems, 26-06  
 magnetic cores, 15-07  
 mean square value, 26-20  
 mean value, 26-18  
 missile systems, 26-10  
 non-Gaussian, 26-18  
 nonlinear systems, 26-09  
 nonstationary, 26-18  
 operational amplifier, 22-16  
 radar, 26-10  
 servos, 26-10  
 sources, 26-12  
 spectral density, 26-05, 26-14, 26-20  
 standard deviation, 26-03  
 stationary processes, 26-03  
 thermal, 26-01  
 transistors, 16-05  
 variance, 26-03  
 white, 26-05  
 Noise generators, 26-12  
 characteristics, 26-12, 26-18  
 demodulator type, 26-15  
 gas discharge tube, 26-13  
 gas tube, 26-14  
 Gaussian, 26-11  
 Geiger-Mueller tube, 26-13  
 radioactive source, 26-12  
 sampling type, 26-14  
 Nondestructive readout, 19-26  
 Nondigital applications, magnetic cores, 15-23  
 Nonelectric field analogs, 25-22  
 Nonlinear differential equation, 27-11  
 Nonlinear electronic computer elements, 23-01  
 Nonlinear potentiometers, 23-17  
 servodriven, 23-31  
 Nonlinear systems, noise, 26-09

- Non-numerical problems; see also *Game playing machines*; *Logic machines*; *Machine translation, languages*  
 characteristics, 11-01  
 literature searching, 11-16; see also *Retrieval systems*
- Nonreturn to zero invert recording, 19-10, 19-13, 20-39
- Nonreturn to zero recording, 19-09, 20-38
- NORC, 2-58, 13-03
- NORC COMPILER, 2-159
- Normal form, 17-13
- Normal random deviates, 2-173
- North American Aviation system, 2-156, 2-185
- Not*, 17-03
- Notation, arithmetic and control, 18-02  
 computer diagrams, 22-03  
 constants and variables, 2-50  
 Turing computers, 31-04
- N-p-n transistors, see *Transistors, junction*
- NRZ recording, 19-09, 20-38
- NRZI recording, 19-10, 19-13, 20-39
- Number conversion, 2-14, 12-14, 18-23  
 in arithmetic of original system, 2-14  
 in arithmetic of second system, 2-17  
 base R to decimal, 12-14  
 binary to decimal, 2-20, 18-24  
 binary to Gray code, 18-25  
 binary powers of 2, (table) 2-43  
 decimal to base R, 12-14  
 decimal to binary, 18-23  
 decimal-hexadecimal, (table) 2-42  
 decimal-octal, (table) 2-28  
 fractional numbers, 12-16  
 Gray code to binary, 18-24  
 hexadecimal-decimal, (table) 2-42  
 mixed numbers, 12-17  
 octal-decimal, (table) 2-28  
 roundoff in, 2-21  
 with scale factors, 2-19  
 trick method, 2-18  
 whole numbers, 12-14
- Numbers, 2-12; see also *Codes*; *Number systems*  
 codes, 12-12, 17-26, (table) 20-46  
 "digital," 2-23  
 as discrete electric signal, 5-07  
 fixed point, 2-23
- Numbers, floating point, 2-24  
 negative, 2-22, 12-22  
 overflow, 12-25  
 precision, 2-22  
 representation, 12-12  
 roundoff, 12-24  
 sign digit, 12-25  
 zero representation, 18-16
- Number systems, 2-13, 5-04, 12-12; see also *Codes*; *Numbers*  
 base, 2-13, 12-12, 12-17  
 binary coded, 12-18  
 binary coded decimal, 5-05  
 comparison, (table) 12-13  
 complements, 2-22, 12-22  
 1's, 12-24  
 10's, 12-23  
 negative, 12-22  
 true, 12-23  
 conversion, see *Number conversion*  
 cyclic, 12-17  
 hexadecimal multiplication, (table) 2-44  
 internal decimal scale factors, 2-21  
 for machines, 12-18  
 nonweighted code, 12-19  
 powers of 2, (table) 2-43  
 radix, 2-13, 2-23, 12-12  
 reflected, 12-17  
 representation, 12-12  
 scale factors, 2-12, 2-24  
 triad, 2-24, (table) 2-64  
 types, (table) 12-13  
 weighted codes, 12-19
- Numerical analysis, 2-02, 2-181, 10-01  
 characteristic roots, 10-04  
 correlation, 10-10  
 differential equations, ordinary, 10-08  
 partial, 10-10  
 time for solution, 10-10  
 factors, 10-11  
 incremental evaluation, 10-09  
 integration, 10-09  
 linear programming, 10-06  
 multiple linear regression, 10-10  
 solution time, eigenvalues, 10-06  
 linear programming, 10-07  
 matrix inversion, (table) 10-03  
 simultaneous linear equations, 10-05  
 statistical, 10-10

- Numerical analysis, storage requirements  
for matrix inversion, (table) 10-03
- Numerical expression, 12-12
- Numerical integration, 10-09
- NYAP, 2-156
- NYDPP, 2-156
- NYU, OMNIFAX, 2-158
- Octal-decimal conversion, (table) 2-28
- Office Management Association of Great  
Britain, 2-261
- Office of the Naval Research Symposia,  
2-261
- Off-line, data processing, 3-04, 20-04
- OMNICODE, 2-157
- One-pass assembly programs, 2-164
- One-pass translators, 2-186
- One-plus-one address codes, 2-58
- One-quadrant multipliers, 23-02
- One-shot multivibrator, transistors, 16-08
- On-the-fly printers, 20-08, 20-10
- On-line, data processing, 3-04
- On-line equipment, 20-04
- Open subroutines, 2-168
- Operating costs, 4-09
- Operating speed, computers, (table)  
2-64
- Operational amplifier, d-c, 22-08  
design, 22-16  
transfer function representation, 22-15
- Operational digital systems, 28-11, 29-01  
adder, 29-10  
analog-digital converters, 29-18  
applications, 29-14  
basic devices, 29-05  
control applications, 28-17  
data acquisition, 29-17  
digital differential analyzers, see *Digital  
differential analyzers*  
division, 28-13, 29-11  
function generators, 28-15  
incremental computation, 29-17  
information averaging, 29-15  
integration, 28-14  
machine organization, 29-19  
multiplication, 28-11, 28-16, 29-05,  
29-09  
process control, 29-17  
rate aiding, 29-15  
simulation, 28-14
- Operational digital systems, square roots,  
29-12  
subtractors, 29-10
- Operational programming, 2-229  
Ivanov transformation rules, 2-233  
operators, 2-229  
Strela, 2-233  
string language of Liapounov-Ivanov,  
2-230, 2-233  
transformation rules, 2-230
- Operations research, 26-10
- Or circuits, 14-34, 17-04, 17-12  
magnetic cores, 15-18  
transistors, 16-10, 16-18  
Turing computer program, 31-05
- Oracle, 12-09
- Ordinary differential equations, 10-08  
analog computer solution, 22-01
- Ordvac, 12-09
- Output, analog computer, 21-05  
tables, 27-04  
data, 4-02, 4-07  
data checks, 3-14  
drivers, transistors, 16-28  
time difference equations, 17-31
- Output equipment, 4-14, 5-02, (table)  
5-33, 20-01; see also *Input-output  
equipment*  
auxiliary, 5-38  
conversion, 5-38  
electric printers, 5-33, (table) 20-08  
low speed, 5-34  
machine readable, 5-34  
magnetic tape, 5-36, 20-33  
printers, 5-33, 20-06
- Overflow, 12-12
- Pacific Mills, 25-07
- Packaging, digital computers, 16-14,  
16-28
- PACT I, IA, 2-156
- PACT Group, 2-156
- PACT system, 2-25, 2-162
- Pade approximations, 23-36
- Page reading equipment, 20-06
- Paper tape, printed, 5-16  
punched, 5-16; see also *Perforated  
tape*
- Paradigm dictionary, 11-14
- Parametron storage, 19-33

- Parity check, 4-15, 5-43, (table) 12-21, 13-03, 13-06  
 core storage, 19-21  
 drum storage, 19-07  
 storage reliability, 19-03
- Partial differential equations, 10-10, 25-01  
 field problems, 25-01, 25-11
- Passive computing elements, 22-04
- Path method, 24-04, 24-06  
 construction of duals, 24-11
- Pattern Instrument Company, 20-11
- Pay checks, data processing, 8-17
- Payroll procedure, 7-10, 8-15
- Peirce function, 17-04, 31-01, 31-03
- Perforated tape, 5-16, 20-19  
 auxiliary equipment, 20-29  
 channels, number, 5-16, 20-19  
 characteristics, (table) 20-20  
 costs, (table) 5-10  
 dimensions, 20-19  
 drivers for photoelectric readers, 20-28  
 equipment, (table) 20-20  
 feed mechanisms, 20-23  
 five-channel, 5-16  
 input buffer, 20-25  
 input units, 5-18  
 mechanical readers, 20-26  
 perforators, 5-10  
 photoelectric tape readers, (table) 20-27, 20-29  
 preparation equipment, 5-16  
 punches, 20-20  
 readers, 5-17, 20-25, (table) 20-27  
 speed, 5-10, 5-17  
 synchronization, 20-24  
 timing, 20-24
- PERM, 2-186, 2-191, 2-192
- Permalloy, 19-07, 19-28
- Phase modulation recording, 20-39
- Phase shift, 22-11  
 operational amplifier, 22-16
- Phileo Corporation, 14-36, 16-02, 16-05, 16-14, 16-30
- Photoelectric readers, cards, 5-15  
 tapes, 20-20, (table) 20-27, 20-29
- Photoformer, 23-19
- Photography, 20-14
- Physical systems, analogs and duals, 24-01
- Physical systems, relation of across and through variables, (table) 24-12
- Piezoelectric delay lines, 19-31
- Plastic tape, 20-33, (table) 20-34
- Plotters, 20-63  
 automatic field, 25-09  
 characteristics, (table) 20-65  
 continuous, 20-64  
 discrete, 20-64  
 operation, 20-65
- P-n-p transistors, see *Transistors, junction*
- Poisson distribution, 26-12
- Poisson's equation, 25-04, 25-17
- Polarized dipole method recording, 20-34
- Polarized light servo, 27-04
- Polarized relays, 23-29
- Polynomial evaluation program, 2-207
- Positional notation, 12-12
- Post mortems, 2-162  
 selective, 2-183
- Potential function, 25-02
- Potentiometers, 22-04  
 errors, 22-33  
 linear, 23-02  
 loading, 23-04  
 multiplying, 23-03  
 nonlinear, 23-17  
 padder system, 23-17  
 servodriven, 23-02, 23-31
- Potter Instrument Company, 20-11, 20-28
- Power, requirements, 6-05
- Power dissipation, transistors, 16-05
- Power supplies, 12-07, 16-14, 16-20, 16-30
- Present address relative, 2-192
- Preset parameters, 2-161, 2-168
- Price tags, punched, 5-13
- Prime implicants, 17-16, (table) 17-18, 17-19  
 reduced table, 17-17, 17-18, 17-20
- Primitive, 17-12
- Princeton, 2-47, 12-09  
 computers, 2-58  
 instruction design, 2-62
- Principia Mathematica*, 2-245
- PRINT I, 2-156
- Printed page, 20-06
- Printers, 20-06  
 electronic, 5-36

- Printers, high-speed, 5-35  
 line, 5-33, 20-08, 20-10  
 matrix, 5-36, 20-08, 20-13  
 mechanical, (table) 20-08  
 nonmechanical, 20-13  
 on-the-fly, 20-08, 20-10  
 wheel, 5-36
- Print readers, 5-21, 20-06
- Probability density function, 26-02
- Probability distribution function, 26-03  
 determination of, 26-18
- Probability logic, 11-09
- Probability multiplier, 23-11
- Probes, electrolytic tanks, 25-09
- Problem setup, analog computer, 21-07
- Problem solving, 11-01, 11-13  
 brain behavior, 11-01
- Process control, operational digital,  
 29-16, 29-17
- Processing unit, 5-02
- Process limited problems, 2-63
- Production scheduling, see also *Scheduling*  
 aircraft, 9-07  
 and management, 9-10
- Program address counter, 17-26
- Program control, analog-digital computer  
 systems, 30-08
- Program interrupt, analog-digital com-  
 puter systems, 30-12
- Program matrix, Turing computers,  
 31-12
- Programmed checks, 3-13, (table) 13-03,  
 13-04
- Programmed control, 18-34, 20-45
- Programmers and coders, 6-11
- Programming, 2-01; see also *Automatic  
 programming; Instructions*  
 address modification methods, 2-60  
 address sorting, 2-146  
 ambiguous-word manipulations, 2-55  
 art of, 2-09  
 automatic, 2-02, 2-12, 2-155  
 basic concepts, 2-03  
 breakpoint notation, 2-61  
 business systems, 7-08, 8-14  
 cost, 2-129  
 debugging, 7-08  
 dynamic stop, 2-140  
 elapsed time, 2-129
- Programming, encoding problem, 2-10  
 errors, 2-128  
 evaluation of a polynomial, 2-127  
 examples, 2-130  
 external, 12-29  
 files, 2-03  
 flow diagrams, 2-47  
 games, 2-246, 11-11  
 hand, 2-02, 2-128  
 incremental computation, 29-20  
 index registers, 2-55, 2-136  
 instruction logic, see *Instruction logic*  
 instructions, structure, 2-56, 12-03  
 types, 2-61, 5-08  
 integrated system, 2-07, 2-184  
 internal, 12-29  
 iteration loops, 2-45  
 languages, 2-02, 2-05, 2-56, 2-186, 2-259;  
 see also *Languages*  
 limitations, 2-02  
 logical, 2-246; see also *Logical pro-  
 gramming*  
 loop control, 2-46, 2-54  
 machine language, 2-131  
 magnetic drum systems, 2-144  
 maintenance of equipment, 2-258; see  
 also *Maintenance*  
 mathematical definition, 2-08  
 measures of performance, 2-05  
 methods, 2-128  
 microprogramming, 2-251; see also  
*Microprogramming*  
 minimal latency, 2-130  
 multiway switch, 2-138  
 natural language, 2-259  
 operations, 2-59; see also *Instructions;*  
*Programming instructions*  
 optimum, 8-14  
 random access storage, 2-144  
 recursive languages, 2-244; see also  
*Languages, recursive*  
 restrictions, 2-129  
 scale factors, 2-12; see also *Scale fac-  
 tors*  
 secondary storage, 2-144  
 segmenting, 2-130  
 self-improvement, 2-05  
 sequencing analog-digital computer  
 systems, 30-10  
 simulation, 2-236, 2-239, 17-40

- Programming, sorting methods, 2-145, 2-146
  - staff, 6-11
  - storage hierarchy, 2-130
  - table look-up, 2-142
  - time, 2-129
  - traditional, 2-128
  - Turing computers, 31-09
  - universal language, 2-200
  - variables, 2-50
  - word length, 2-61
- Program parameters, 2-161, 2-168
- Program transfers, 17-26
- Propositional calculus, logical programming, 2-246
- Pseudo instructions, 2-162
- Public utility, customer billing, 8-11
- Pulse amplifiers, 14-21, 14-23
  - high power, 14-25
  - transistor, 16-15
- Pulse gate, high-speed, 14-42
- Pulse source, 14-21, 14-22
- Pulse train, unitary weighted, 29-05
- Pulse width, analog-digital converters, 30-05, 30-07
- Punched cards, 5-13, 12-10, 12-11, 20-30
  - auxiliary equipment, 20-31
  - codes, 12-08, 12-10
  - costs, (table) 5-10, 20-30
  - dimensions, 5-13, 20-30
  - equipment, 5-13, 5-33, 12-08, 20-30
  - input units, 5-15
  - punches, 5-15
  - readers, 5-15, 20-31
  - sorters, 20-32
  - speed, 5-10, 5-19
- Punched tape, see *Perforated tape*
- Punches, perforated tape, 20-20
  - cards, 5-15
- Purdue University, 2-158
- Quadratic evaluation, SAP program, 2-167
- Quarter squares multipliers, 23-08
- QUEASY, 2-156
- QUICK, 2-156
- Quine simplification, method, 17-16
  - Veitch diagram, 17-22
- Radar, noise, 26-10, 26-11
- Radio Electronic Television Manufacturers Association, 14-11
- Radix, 2-13, 12-12
- RAMAC, 2-260
- Ramo-Wooldridge Corporation, 2-157, 2-158, 2-239, 2-263
  - one-pass assembler, see *RAWOOP*
- Ramp method, analog-digital conversion, 20-49
- Rand Corporation, 2-156, 2-159
- Random access storage, 4-12, 5-26, 5-32, 19-03
  - programming with, 2-144
- Randomization, 2-144
- Random variables, 26-02, 26-11
- Rate aiding, 29-15
- Rate multiplier, binary, 29-05
  - dynamic, 29-09
- RAWOOP, 2-158, 2-164
- RAWOOP-SNAP, 2-157
- Raydac, 13-03, 19-29
- Rayleigh distribution, 26-18
- RCA, 20-17
- RC networks, digital-analog conversion, 20-60
- Readers, magnetic drums, 19-11
  - magnetic tape, 5-18, 20-35
  - perforated tape, 5-17, 20-25
  - punched card, 5-15, 20-31
- Read in, instructions, 5-41
- Read in operation, 18-07
- Real time computers, 3-05, 12-29, 29-02
- Real time simulation, 28-14
- Reasonableness checks, 13-03, 13-04
- Recorders, 23-34
- Recording; see also *Magnetic drums, recording*; *Magnetic tapes, recording*
  - errors, 22-37
  - magnetic drums, 5-18
  - magnetic tapes, 20-36, 20-40
  - media, 5-09, 20-05
  - methods, 19-09, 20-35
  - selective, 3-10
- Records, 3-07
- Rectangular to polar transformation, 23-32
- Recursive languages, 2-244
- Redstone Arsenal, 2-156, 2-157
- Red tape subroutines, 2-181
- Redundance, check, 4-15

- Reflected number systems, (table) 12-17,  
18-24, 20-46
- Refrigeration, closed-loop systems, 6-07  
open-ended systems, 6-08  
ton, 6-07
- REG-SYMBOLIC, 2-156
- RELATIVE, 2-157
- Relative address, 12-28
- Relay computer, 12-09
- Relays, 14-47  
analog computer, 23-23  
construction, 14-47  
drivers, transistors, 16-28  
enclosed, 14-47  
polarized, 23-29  
time delays, 14-47
- RELCODE, 2-158
- Reliability, circuit design, 13-07  
component variations, 13-07  
computer circuits, 14-01, 14-19  
control systems, 14-02  
data processors, 4-08  
design techniques, 13-02, (table) 13-03,  
13-05  
digital computers, 13-01  
duplication of arithmetic section,  
13-03, 13-06  
duplication of machine operation,  
13-03, 13-04  
environment, 13-08  
and errors, 4-07  
evaluation criteria, 13-02  
magnetic drums, 19-12  
maintenance, 19-03  
storage, 19-03  
transistors, 16-03, 16-04  
trouble detection and location, 13-09
- Remington Rand, 2-157, 2-158, 2-160;  
see also *Sperry Rand*; *Univac*  
409-2, 12-08  
card, 20-30, 20-33  
punched card, 5-13, 12-11, 20-32  
Univac, 2-83, 2-239, 2-263
- Remote connection, 2-49
- Reproducers, perforated tape, 20-29  
punches, 20-32
- Reservations, data processing, 9-01  
equipment, 9-04  
storage requirements, 9-04
- Resistance-capacitance networks, 25-19
- Resistance networks, 25-15  
accuracies, 25-19  
applications, 25-17  
digital-analog conversion, 20-58  
equations, 25-16  
with internal excitation, 25-17
- Resistance paper, 25-05  
accuracy, 25-06
- Resistors, carbon, 14-48  
film, 14-48  
marginal checking, 14-07  
potentiometers, 14-49  
wire-wound, 14-49
- Resolvers, a-c induction, 23-32  
d-c, 23-31
- RETMA, 14-11
- Retrieval systems, library problem, 11-16  
mechanized, 11-17  
semantic noise, 11-18
- Return to bias recording, 19-09
- Return on investment, business systems,  
7-12, 7-13
- Return to zero recording, 19-09, 20-36
- R-L-C* networks, 25-20
- Rollback, 2-184
- Root determination, 2-211
- Roundoff, 12-24, 18-29  
correction, 18-20  
in number conversion, 2-21
- Royal-McBee, 2-109, 5-42  
LGP-30, 2-64  
characteristics, (table) 5-42  
instruction logic, 2-109
- R-S* flip-flops, 17-08, 17-35, 18-02, 31-07
- R-S-T* flip-flops, 17-08, 17-36
- Runge-Kutta-Gill, IT subroutine, 2-189  
technique, 2-214
- Runge-Kutta method, 10-09  
extension, 2-188  
IT translator compiler, 2-215  
solution of differential equations, 2-214
- Run request, IT translator language,  
2-219
- Russian language, 11-14
- Russian peasant method, 18-28
- RW 300, 2-239
- RZ recording, 19-09, 20-36
- SAC, 2-158
- Sage, 13-03

- Sage, Air Defense Computers, 2-58  
 SAIL, 2-158  
 Salary distribution, 8-15  
 Sampled data, analog-digital computer systems, 30-13  
 Sampling, analog-digital converters, 30-04, 30-06  
 SAP, 2-156, 2-162, 2-164, 2-186, 2-187, 2-239  
   evaluation of quadratic form, 2-167  
   pseudo instructions, 2-165  
 Saturation flip-flop, 16-07  
 Sawtooth waves, generation, 23-29  
 SBT, see *Transistors, surface barrier type*  
 Scale factors, 21-07, 22-10, 22-12, 27-14  
   conversion of numbers with, 2-19  
   digital differential analyzer, 28-04  
   in fixed point computation, 2-24  
   function generator, 23-19  
   integrators, 28-06  
   internal decimal, 2-21  
   limitations on variables, 27-14  
   multipliers, 23-11  
 Scaler, binary, 29-05  
 Scaling, see *Scale factors*  
 Scanning, 3-09  
 SCAT, 2-157  
 Scheduled maintenance, 13-08  
 Scheduling, aircraft production, 9-07  
   computer flow diagram, 9-11  
   equations, 9-09  
 Scientific computers, 5-04  
   applications, 10-01  
   input-output, 20-02  
 SCRIPT, 2-157  
 SEAC, 2-57, 2-115, 2-159, 19-29  
 Segmenting, 2-130  
 Selection circuits, magnetic cores, 15-19  
 Selective recording, 3-10  
 Self-checking codes, 13-03, 13-06  
 Self-complementing codes, 12-20  
 Semantic ambiguity, 11-14, 11-15  
 Semiconductor diodes, 14-45  
   application notes, 14-46  
   characteristics, 14-45  
 Sense windings, 19-25  
 Sentence decomposition, 2-259  
 Sentential calculus, 11-03  
 Sentential calculus, mechanization of decision procedures, 11-05  
 Sequential access, 19-03  
 Sequential storage, 4-12  
 Service Bureau Corp., 2-156  
 Servomechanisms, multipliers, 23-02, 23-04  
   polarized light, 27-04  
   potentiometers, nonlinear, 23-31  
   trigonometric functions, 23-32  
 Set-reset flip-flop, 17-08, 18-02, 31-07  
 SHACO, 2-156  
 Shaft position, analog-digital conversion, 20-51  
   digital-analog conversion, 20-60  
 Shaped beam tube, 5-37, 20-14  
 Share Assembly Program, see *SAP*  
 Share Cooperative Programming Group, 2-164  
 Shaw method, 18-16  
 Sheffer stroke function, 17-04, 31-01, 31-03  
 Shift, 18-27  
 Shifting register, 18-05  
   magnetic cores, 15-15  
 Short-circuit transfer impedance, 22-16, (table) 22-18  
 SHORT CODE, 2-158  
 Sign digit, 12-25  
 Sign inversion, 22-09  
 Simplex method, 10-07  
 Simulation,  
   absolute value, 23-27  
   backlash, 23-27  
   Coulomb friction, 23-27, 23-29  
   dead zone, 23-25  
   driver program, 17-41  
   logical design, 17-38  
   of one computer by another, 2-236, 31-02  
   output, 17-41  
   programming, 17-40  
   real time, 28-14  
   switching devices, 23-25  
   time delay, 23-34  
   by Turing computer, 31-02  
 Simultaneous linear equations, algebraic, 10-02  
   methods, (table) 10-05  
   storage requirements, (table) 10-05  
   time for solution, (table) 10-05

- Sines and cosines from implicit computation, 23-32
- Single address instructions, 2-58
- SIR, 2-157
- 607, 2-156
- SNAP, 2-158
- SO 2, 2-156
- SOAP, 2-186, 2-201, 2-214, 2-217  
 dynamic stop, 2-140  
 IT translator, 2-209, 2-214  
 multiway switch, 2-138  
 programming examples, 2-131  
 subroutine entry, 2-141
- SOAP I, 2-157
- SOAP II, 2-157, 2-219
- SOHIO, 2-156
- Soroban Engineering, Inc., 20-22
- Sorters, punched cards, 20-32
- Sorting, 2-145, 2-146, 3-07  
 address, 2-146  
 digital, 2-146  
 example, 2-153  
 finding the smallest, 2-152  
 interchanging pairs, 2-152  
 by merging, 2-147  
 number of comparisons, 2-153  
 partial, 2-153  
 sifting, 2-153  
 use of main storage, 2-146
- Source data rate, (table) 4-04
- Soviet, algebraic language compiler, 2-228  
 computers, 2-60  
 Strela, see *Strela*  
 Union, see *USSR*
- Special purpose, analog computers, 21-02  
 digital computers, 5-04, 9-01, 12-09, 12-25
- Spectral density, 26-05  
 determination of, 26-20  
 random telegraph signal, 26-14
- SPEEDCODING, 2-156, 2-157
- Speeds, cathode ray tube display, 20-14  
 code wheel converters, (table) 20-56  
 computers, 5-42  
 electrography, 20-18  
 input media, 5-10  
 keyboard, 5-11  
 magnetic cards, 20-42
- Speeds, magnetic tape, (table) 20-42  
 mechanical printers, (table) 20-08, 20-10, 20-12, 20-13  
 perforated tape equipment, (table) 20-20  
 photoelectric readers, 20-20, (table) 20-27  
 plotters, 20-65  
 punched card equipment, 20-31, 20-32  
 tape punches, 20-20, 20-21, 20-22, 20-23  
 typewriters, 20-08, 20-10
- Sperry Rand Corporation, 2-83, 19-27  
 1103, 2-158  
 1103A, 2-157  
 File Computer, 2-158  
 Larc, 2-158  
 Univac I, 2-158  
 Univac II, 2-158
- Split winding transfer loop, 15-10
- SPUR, 2-157
- Squareness ratio, 19-24
- Square root, 18-22, 18-30  
 decimal, 18-30  
 incremental computation, 29-27  
 methods, 18-23  
 operational digital, 29-12
- Square waves, generation, 23-29
- Squaring circuits, biased diode, 23-08  
 triode, 23-08
- Standard deviation, 26-03
- Standard Oil of Ohio, 2-156
- STAR, 2-158
- Stationary processes, 26-03
- Statistical analysis, 10-10
- Statistical problems, (table) 26-10
- Statistical techniques, 26-01
- Steklov Mathematical Institute, 2-228
- Step relay multiplier, 23-10
- Storage, 5-02, 5-24, 12-02, 12-06, 19-01;  
 see also *Magnetic cores*; *Magnetic drums*; *Magnetic tape*  
 access methods, 4-12, 19-03  
 access time, 5-24  
 acoustic delay lines, 19-29  
 auxiliary, 7-05  
 buffer, 17-25, 20-03  
 capacitor-diode, 19-32  
 characteristics, 5-24, 19-03  
 cost, (table) 5-26  
 cryogenic films, 19-32

- Storage, data processors, 4-11  
 delay lines, 19-29  
 electronic devices, 5-25, (table) 5-26  
 electrostatic, 19-31  
 external, 5-24  
 ferrite plates, 19-29  
 ferroelectric, 19-31  
 hierarchies, 19-02  
 logical elements, 17-01, 17-05  
 magnetic cores, 14-03, 15-19, 19-13; see  
   also *Magnetic cores*  
 magnetic disks, 5-32, 19-13  
 magnetic drums, 19-04; see also *Mag-  
   netic drums*  
 magnetic films, 19-28  
 mercury delay lines, 19-29  
 multiaperture cores, 19-29  
 operations, 2-60  
 parametron, 19-33  
 parity check, 19-03  
 principle of operations, 5-25, (table)  
   5-27, 19-02  
 printouts, 2-183  
 programming with secondary, 2-144  
 random access, 5-32, 19-03  
   programming with, 2-144  
   and sorting, 3-08  
 recording media, 5-09, 20-05  
 reliability, 19-03  
 sorting, 2-146  
 speeds, 5-26  
 transfer programs, 2-184  
 transistor circuits, 16-20  
 tubes, 19-31  
 Turing computers, 31-02  
 twister, 19-33  
 types, 5-24, (table) 5-26  
 uses, 5-25, (table) 5-27
- Stored program, 12-29
- Stream function, 25-02
- Streamlines, 25-02
- Strela, 2-60, (table) 2-64, 2-111, 2-186,  
 2-188, 2-228  
 instruction logic, 2-111  
 string programming, Liapounov-Ianov,  
 2-233
- STRETCH**, see *IBM-STRETCH*
- Stretched membranes, 25-22
- Stretched program, 2-182
- String language, 2-190
- String language, of Ianov, 2-230  
 notation of Liapounov, 2-233
- Stromberg-Carlson, 5-37, 20-14, 20-16
- Suboperations, 17-27
- Subroutine library, IT translator, 2-214  
 MIDAC computer, 2-172
- Subroutines, 2-167; see also *Integrated  
   systems; Utility programs*  
 arithmetic classification, 2-180  
 automatic exit, 2-169  
 automatic programming, 2-161  
 call-in methods, 2-180  
 change of control, 2-169  
 closed, 2-169  
 conditional program, 2-171  
 deferred preparation, 2-171  
 entry, 2-140  
 exit, 2-169  
 floating variables, 2-168  
 free variables, 2-168  
 generators, differential, 2-182  
   print, 2-182  
   stretched program, 2-182  
 hierarchy, 2-245  
   counters, 2-195  
 instructions, see *Instructions*  
 linear matrix solver, 2-174  
 momentary call-in, 2-180  
 normal random deviates, 2-173  
 numerical methods, 2-181  
 open, 2-168  
 packages, 2-219  
 permanent call-in, 2-180  
 preliminary preparation, 2-169  
 program parameters, 2-168  
 recursive use of, 2-244; see also *Lan-  
   guages, recursive*  
 red tape, 2-181  
 specification sheet, Univac 1103, 2-174  
 specifications manual, 2-171  
 structure, 2-168  
 subprogram, 2-171  
 synthetic instructions, 2-168  
 transfer address, 2-171  
 wired in, 2-188
- Subtraction, 18-14  
 decimal, 18-27  
 half, 18-14  
 incremental computation, 29-25  
 mechanical, 27-02

- Subtraction, operational digital, 29-10  
 parallel, 18-15  
 serial, 18-14  
 using complements, 18-14
- Subtractor, see *Subtraction*
- Successive approximation method, analog-digital conversion, 20-63
- Sum function, Veitch diagram, 17-07, 17-22
- Summary punches, 20-32
- Summation, analog, 22-04, 22-09
- SUMMER SESSION, 2-159
- Summing amplifier, 22-09
- Summing network, 22-05
- Superconducting storage unit, 19-32
- Superposition, principle of, 26-06
- Surface barrier transistors, 16-06
- Switching characteristics, thermionic diode, 23-15  
 transistors, 16-02
- Switching circuits, analog computers, 23-22  
 and logic, 11-05  
 transistors, 16-02, 16-05
- Switching concepts, magnetic cores, 15-01
- Switching functions, (table) 17-04, 17-10  
 simplification, 17-15
- Switching speed, cores, 19-23  
 transistors, 16-04, 16-07, 16-14
- SYMB. ASSEM., 2-156
- Symbolic addresses, 2-131, 2-160  
 translated, 2-211
- Symbolic language, retranslation, 2-162
- Symbolic Optical Assembly Program, see *SOAP*
- Symbols, adder, 18-03, 18-10  
 analog computers, (table) 21-10, 22-03  
 analogs, 24-03  
 arithmetic and control, 18-02  
 computers, 1-01  
 digital computers, 1-01  
 digital differential analyzers, 28-03  
 EASIAC, 2-123  
 flip-flop, 17-30, 18-03  
 logical design, (table) 17-02  
 magnetic core circuits, 15-04  
 mechanical computers, 27-05  
 programming, 2-47
- Synchronization, tape punches, 20-25
- Synchronous computers, 17-05
- Syntactical transformation rules, 2-228
- Synthetic instructions, 2-161, 2-168
- Systems, linear, 26-06  
 nonlinear, 26-09
- Systems planning staff, 6-11
- Table look-up, 2-142, 3-09
- Tables, 2-50
- Tabulators, punched card, 20-32
- Tape, see *Magnetic tape*; *Perforated tape*
- Tape limited problems, 2-63
- Tapped potentiometers, function generators, 23-17, 23-20
- Taylor-McLaurin series, 2-181
- Tchebysheff, approximation, 2-182  
 polynomials, 2-181
- Teledeltos paper, 20-18, 25-06, (table) 25-07; see also *Resistance paper*
- Telegraph signal, random, 26-13
- Teleregister Corporation, 9-04
- Teletype Corporation, 20-21, 20-26
- Temperature stability, magnetic cores, 15-09
- Terminology, computers, 1-02
- Terms of consensus, 17-17
- Thermal analyzers, 25-19
- Through variable, 24-03, 25-02
- Time average, ergodic process, 26-03
- Time cards, data processing, 8-17
- Time constants, 22-10  
 divider, 28-13  
 effects on errors, 22-36  
 integration, 22-07  
 integrator, 22-36
- Time delays, approximations, 23-35  
 digital storage, 23-35  
 fourth-order approximations, 23-38  
 magnetic tape, 23-34  
 multiple lag approximations, 23-36  
 Pade approximations, 23-36  
 relays, 14-48  
 simulators, 23-34  
 Taylor series expansion, 23-35  
 true, 23-34  
 two-pen recorders, 23-34
- Time division, multipliers, 23-05
- Time interval, analog-digital conversion, 20-47

- Time interval, digital-analog conversion, 20-57
- Time scale factor, 22-11
- Time sharing, analog-digital computations, 30-10, 30-14  
computers, 31-01
- Time varying systems, 26-06
- Timing, tape punches, 20-24
- Timing control circuits, magnetic cores, 15-21
- Timing diagram, binary rate multiplier, 29-07
- Toggles, 17-05; see also *Flip-flops*
- Tolerance plots, 14-06
- Ton, of refrigeration, 6-07
- Torque amplifiers, 27-04
- Tracing programs, 2-183, 2-240
- Tradic computer, 16-02, 16-15  
power requirements, 16-20
- Traditional programming techniques, 2-128
- Training courses, 6-10, 6-12
- Transac computer, 16-02, 16-05  
packaging, 16-14  
power requirements, 16-14
- TRANSCODE, 2-159
- Transcription, of data, 3-03
- Transfer characteristics, computer circuits, 14-21  
gate tube, 14-28  
pulse amplifier, 14-24, 14-26
- Transfer of control instructions, 5-41, 12-04
- Transfer function representation, 22-13  
differential analyzer method, 22-16  
direct analog method, 22-14  
operational amplifier method, 22-15
- Transfer impedance, short circuit, 22-16, (table) 22-18
- Transfer instructions, see *Instructions*
- Transfer loops, magnetic cores, 15-09
- Transformers, 14-50  
coupling of transistors, 16-17
- Transistor circuits, 14-36, 16-01, 16-03  
adder, 16-11  
advantages, 16-01  
*and* gates, 16-10, 16-18  
combined gates, 16-11  
computers, (table) 16-03, 16-05, 16-15, 16-20, 16-23
- Transistor circuits, counter, 16-28  
design considerations, 14-36, 16-04  
direct coupled, 16-05  
emitter follower, 16-24  
flip-flop, 16-07, 16-26  
high-speed, 14-37, 14-38  
gates, 16-09, 16-18  
*inhibitor* gates, 16-18  
inverters, 16-23  
logical gain, 16-04, 16-05, 16-07  
logic elements, 16-28  
magnetic core loops, 15-13  
magnetic core multiplier, 29-08  
marginal checking, 14-37  
noise immunity, 16-05  
one-shot multivibrator, 16-08  
*or* gates, 16-10, 16-18  
output drivers, 16-28  
power dissipation, 16-05  
pulse amplifiers, 16-15, 16-17  
pulse gate, high speed, 14-42  
reliability, 16-04  
storage, 16-20  
switching speed, 16-02, 16-04, 16-07, 16-14  
transformer coupling, 16-17
- Transistors, 14-51  
alloy junction type, 14-51  
construction, 14-51  
junction, 14-51, 14-52, 16-23  
application notes, 14-53  
point-contact characteristics, 16-16  
reliability, 16-03  
surface barrier type, 14-36, 14-41, 16-06  
switching properties, 16-02  
types, 14-51, 16-02
- Translation, 2-183  
automatic programming, 2-160
- Translator construction, 2-221  
compiler representation, 2-221  
decision processes, 2-225  
permissible symbol pairs, (table) 2-222  
rational procedure, 2-223  
symbol pair technique, 2-221  
syntactical transformation rules, 2-228
- Translators, 2-186; see also *Compilers*  
address relative, present, 2-192  
algorithms, 2-190

- Translators, automatic instruction modification, 2-190  
 filing system, 2-188  
 future trends, 2-189  
 indirect addressing, 2-193, 2-199  
 instruction modification types, 2-191  
 IT translator, 2-187, 2-200; see also *IT translator*  
 language extensions, 2-187  
 languages, 2-186  
 one-pass, 2-186  
 preparation of problems in terms of building blocks, 2-187  
 string language, 2-190  
 subroutine hierarchy counter, 2-195  
 two-pass, 2-186  
 Transportation problem, 10-08  
 TRANS-USE, 2-157  
 Trapping mode, IBM-704, 2-65, 2-241  
 Trigger flip-flop, 17-07, 17-34, 18-02  
 Trigonometric devices, analog computer, 23-31  
   sines and cosines from implicit computation, 23-32  
 Trouble detection and location, 13-09  
 Truth computer, 12-09  
 Truth tables, 17-11  
   adder, 18-10  
   flip-flops, 18-03  
   half adder, 18-08  
   in logic, 11-04  
   magnetic core circuits, 15-17  
 Tubes, see *Vacuum tubes*  
 Turing type computers, 31-01  
   adder logic, 31-06  
   auxiliary storage, 31-13  
   communication technique, 31-02  
   comparisons, 31-15  
   input-output, 31-03, 31-07  
   mechanization, 31-07  
   operations, 31-03  
   programming, 31-09, 31-12  
   Sheffer stroke, 31-01, 31-03  
   universal, 2-235  
 Twister storage, 19-33  
 Two-pass assemblers, 2-164  
 Two-pass translators, 2-186  
 Two-quadrant multipliers, 23-02  
 TX-0 Development, microprogramming, 2-252  
 TX-2 core storage, 19-27  
 Typewriters, 5-12, 20-07, 20-08  
 Typotron, 20-14  
 UDECIN-1, 2-158  
 UDECOM-3, 2-158  
 UGLIAC, 2-158  
 Unconditional transfer of control instructions, 5-41, 12-04  
 UNICODE, 2-157, 2-186, 2-245, 2-260  
 Unitary weighted pulse trains, 29-05  
 United Aircraft Corporation, 2-156, 2-164  
 United Gas Corp., 2-158  
 United States Army, 2-190  
 Unit impulse response, 26-06  
 Units of information, data processing, 3-06  
 Unityper, 8-02  
 Univac, 2-162, 12-09, 13-03, 19-27, 19-29  
 Univac I, 2-58, 2-130, 2-186, 2-239, 2-259  
   accounting applications, 8-01  
 Univac II, 2-58, 2-64, 2-186, 2-239, 2-259  
   instruction logic, 2-83  
   symbology, 2-84  
 Univac 1103, 2-58, 2-176, 2-186, 2-193  
   magnetic core storage, (table) 19-27  
   subroutine, linear matrix solver, 2-174  
 Univac 1103A, 2-55, 2-58, 2-64, 2-164, 2-239, 2-241  
   definitions and conventions, 2-77  
   instruction logic, 2-77  
   logical programming, 2-247  
 Univac 1105, 2-58  
 UNIVAC-LARC, 2-62, 2-189, 2-191  
 Univac M-460, 2-191  
 Univac Scientific, 12-09; see also *Univac 1103 and 1103A*  
 Univac Scientific Exchange, see *USE*  
 UNIV. CODE, 2-159  
 Universal computer language, automatic programming, 2-163  
 Universal programming languages, 2-200  
 Universal Turing Machine, 2-235  
 University of California, 27-01  
   Radiation Laboratories, 2-156, 2-157  
 University of Illinois, 2-159  
 University of Michigan, 2-122, 2-159, 2-212, 2-219, 2-240, 2-260, 2-262  
   Digital Computation Unit, 2-200  
   IT Subroutine Library, 2-214

- University of Pennsylvania, 12-09
- University of Toronto, 2-159
- Unscheduled maintenance, 13-09
- Ural, 2-111
- USE, 2-157, 2-164
- U. S. Rubber Company, 25-07
- U.S.S.R., 2-186, 2-190, 2-228
  - Academy of Sciences, 2-111
- Utility programs, 2-183
  - assembly, 2-183
  - automatic programming, 2-162
  - compiler-translators, 2-184
  - input, 2-183
  - interpretive, 2-184
  - selective post mortems, 2-183
  - storage printouts, 2-183
    - transfers, 2-184
  - tracing, 2-183
- Vacuum tubes, 14-07, 14-20
  - application notes, 14-44
  - cathode temperature, 14-43
  - characteristics, 14-43
  - construction, 14-43
  - gates, 14-26
  - marginal checking, 14-07
  - pentodes, 14-08
  - triodes, 14-08
- Variables, across-through, 24-03
  - analog computers, 22-02
  - programming, 2-50
  - relationships, (table) 24-04
  - systems, 24-09, (table) 24-12
- Variance, 26-03
- V brush method, analog-digital conversion, 20-54
- Vector inner product, IBM 650 SOAP program, 2-135
  - MIDAC MAGIC program, 2-136
- Vectors, characteristic roots and, 10-04
- Veitch diagrams, 17-13
  - geometric relationships, 17-20
  - Quine simplification, 17-22
  - simplification, 17-20
- Verifiers, 5-12
  - punched card, 20-31
- Voltage, analog-digital conversion, 20-48
  - digital-analog conversion, 20-58
- von Neumann-Goldstine method, 10-02, 10-03
- Wave equation, 25-04, 25-19
- Weighed check, 13-03, 13-06
- Weighted codes, 12-19
- Weighting function, 26-06
- Western Union, 20-18, 25-07
- Wheeler entry, 2-141
- Wheeler method, subroutine exit, 2-170
- Whirlwind I, 2-159, 2-171, 19-06
  - instructions, 2-62
  - Library of Subroutines, 2-180
  - maintenance of equipment programs, 2-258
- Williams storage tube, 19-31
- Words, 2-22, 2-56, 5-06, 12-27
  - instructions, 2-61, 2-63
  - machine translation, 11-14
  - size, (table) 2-64
- Wright Air Development Center, 2-158
- Write amplifiers, logical design, 18-38
- Write out instructions, 5-41
- Writing techniques, magnetic cores, 19-15
  - magnetic drums, 19-09, 19-11
- X-1 assembly system, 2-158, 2-162
- Xerography, 20-16
- Xerox, 20-16
- X-y plotters, 20-63

