

BASICPAC
Programming Manual

January, 1961

PHILCO CORPORATION
Government and Industrial Group
Computer Division
3900 Welsh Road
Willow Grove, Pa.


BASICPAC PROGRAMMING MANUAL


Effective January 1961


Prepared by


Mary Ellen Langford

Approved by


Harry K. Mellinger
Section Manager


Hubert J. Nyser


Samuel M. Berkowitz
Manager, Military Computer
Engineering Department

Submitted by Philco Computer Division, Willow Grove, Pa. in partial fulfillment of Article I, Item 17, of Contract No. DA-36-039-SC-78132.

Preface

This document, BASICPAC PROGRAMMING MANUAL, is submitted in accordance with and in partial fulfillment of U. S. Army Signal Corps Contract No. DA-36-039-SC-78132 and Technical Requirement SCL-1943A, Paragraph 3.10.

The Programming Manual is divided into two Sections. Section One provides a general introduction to the BASICPAC system from a programming viewpoint. Section Two provides more specific information on BASICPAC programming techniques, features, and requirements. General reference data is presented in the Appendices.

CONTENTS
SECTION ONE. THE BASICPAC SYSTEM

	Page
I. INTRODUCTION	1-1
A. General	1-1
B. System Configurations	1-1
1. Minimum BASICPAC System	1-1
2. BASICPAC S-109 Shelter System	1-2
3. Maximum BASICPAC System	1-2
4. Busses	1-2
II. BASICPAC CENTRAL PROCESSOR	1-7
A. Memory Unit	1-7
1. Memory Address Register (MA)	1-7
2. Memory Data Register (MO)	1-7
3. Parity	1-7
4. Memory Priority System	1-7
B. Program Unit	1-8
1. Instruction Register (IR)	1-8
2. Program Counter Register (PC)	1-8
3. Program Counter Store (PCS)	1-8
4. Index Registers	1-9
5. Program and Index Adders	1-9
6. Timing	1-9
C. Arithmetic Unit	1-9
1. The A Register	1-9
2. The B Register	1-10
3. The Q Register	1-10
4. The Adder-Subtractor Network	1-11
5. The T-Counter	1-11
D. Control Unit	1-11

CONTENTS (Continued)

	Page
III. DATA AND INSTRUCTIONS	1-11
A. Word Format	1-11
1. Data Words	1-11
a. Numeric Data	1-12
b. Alphanumeric Data	1-12
2. Instruction Words	1-13
a. Computer Instruction Word	1-13
b. Input-Output Instruction Word	1-14
B. Order Codes	1-15
1. Central Processor Order Codes	1-17
a. Arithmetic Orders	1-17
b. Transfer Orders	1-19
c. Logical Orders	1-22
d. Sense Orders	1-26
e. Overflow	1-27
f. Trapping Mode	1-28
2. Input-Output Orders	1-29
a. ICF Behavior	1-30
b. Interrupts	1-31
c. Usage of Control and Converter Input-Output Orders Compared	1-32
IV. CONTROL SYSTEM	1-32
A. Control Unit	1-33
1. Word Selection Register (WSR)	1-33
2. Control Character Buffer (CCB)	1-33
3. Sense Flipflops	1-34
B. Control Panel	1-34
1. Power Area Display and Controls	1-34
2. Register Area Display and Controls	1-36
3. Operation Area Display and Controls	1-39
4. Error Control and Display	1-41
5. Sense Indicators and Switches	1-42

CONTENTS (Continued)

	Page
C. Paper Tape Set	1-43
1. Orders	1-43
2. Speeds	1-44
3. Typewriter Control	1-44
a. Mode Control Keys	1-44
b. Other Control Keys	1-45
4. Paper Tape Reader	1-47
5. Paper Tape Punch	1-47
V. INPUT-OUTPUT CONVERTER	1-49
A. General Description	1-49
B. Function	1-49
1. Instruction Control	1-49
2. Data Control	1-59
3. Error Control	1-59
4. FIELDATA Control Function Processing	1-60
5. Program Interruption by Input-Output Converter	1-61
6. Communication with Central Processor	1-61
a. Computer IR to Input-Output Converter CIS	1-61
b. Input-Output Converter CIS to Computer	1-62
c. Addressable Flipflops	1-62
d. Converter Data Register (CDR) to Memory (MO)	1-63
e. Memory (MO) to Converter Data Register (CDR)	1-63
C. Addressable Flipflops and Registers	1-63
D. Magnetic Tape Transports	1-63
VI. COMMUNICATIONS CONVERTER	1-64
A. Input	1-65
B. Output	1-67

CONTENTS (Continued)
SECTION TWO. BASICPAC PROGRAMMING

	Page
VII. INTRODUCTION	2-1
VIII. NUMBER SYSTEMS	2-1
A. Decimal Number System	2-1
B. Octal Number System	2-4
C. Binary Number System	2-5
D. Number System Conversion	2-6
1. Binary to Octal	2-6
2. Octal to Binary	2-7
3. Binary to Decimal	2-7
4. Decimal to Binary	2-8
E. Binary Arithmetic	2-10
F. Algebraic Arithmetic	2-12
IX. PREPARING A PROGRAM FOR BASICPAC	2-13
A. General	2-13
B. Problem Definition	2-14
C. Problem Analysis	2-15
D. Problem Specification	2-17
X. FLOW CHARTING	2-18
A. Flow Charting Symbols	2-19
B. Examples	2-21
XI. CODING	2-23
A. Transfer of Information	2-24
B. Addition and Subtraction	2-28
C. Multiplication and Division	2-32
D. Transfer of Control	2-38
E. Information Formats	2-45
F. Floating Point Arithmetic	2-46
G. Scaling and Shifting	2-47
Overflow	2-52
H. Logical Instructions	2-52
I. Sense Instructions	2-56

CONTENTS (Continued)

	Page
J. Program Modification and Loops	2-58
K. Index Registers	2-64
L. Subroutines	2-68
M. Interrupt Subroutines	2-71
XII. INPUT-OUTPUT PROGRAMMING	2-74
A. Control Unit Input-Output Equipment	2-74
B. Input-Output Converter	2-77
C. Communications Converter	2-80
XIII. DEBUGGING METHODS	2-84
XIV. FIELDATA ASSEMBLY LANGUAGE	2-89
XV. LIBRARY ROUTINES	2-91

APPENDICES

- A. Microflow Charts
- B. Addressable Registers and Flipflops
- C. Address Assignments for Input-Output Devices
- D. Codes

FIGURES

		Page
I-1	BASICPAC System Block Diagram	1-3
I-2	BASICPAC Central Processor Block Diagram	1-4
IV-1	BASICPAC Control Panel	1-35
V-1	Input-Output Converter Block Diagram	1-50
V-2	Input-Output Converter Flow Diagram, Phase I	1-51
V-3	Input-Output Converter Flow Diagram, Phase II	1-52
V-4	Input-Output Converter Flow Diagram, Phase III	1-54
V-5	Input-Output Converter Flow Diagram, Phase IV	1-58
VI-1	Communications Converter Input Section Block Diagram	1-68
VI-2	Communications Converter Output Section Block Diagram	1-69
VI-3	Communications Converter Output Flow Chart	1-70
VI-4	Communications Converter Input Flow Chart	1-72

TABLES

		Page
I-1	BASICPAC Systems and Major Components	1-5
I-2	Summary of BASICPAC Characteristics	1-6
IV-1	Nixie Indicator Interpretations	1-38
VIII-1	Some Number Systems	2-3

SECTION ONE

THE BASICPAC SYSTEM

SECTION ONE

I. INTRODUCTION

A. GENERAL

The BASICPAC computer is a medium to high speed, general-purpose, solid-state machine with modular expansibility in both memory and input-output capacity. The BASICPAC computer consists of the standard central processor plus additional modular units. (See Figures I-1 and I-2.)

As a member of the FIELDATA family, BASICPAC employs the FIELDATA common-language code, FIELDATA interconnection standards, and the FIELDATA word structure and order catalog. The system is provided with a communications converter for the receipt and transmission of data on a "real-time" basis. Each significant register in each functional unit of the BASICPAC computer is connected to a common major transfer bus which transfers information between different sections of the computer and facilitates the modular expansion of both memory and input-output capacity.

B. SYSTEM CONFIGURATIONS

Three BASICPAC system configurations are defined: the minimum BASICPAC (standard central processor), the BASICPAC S-109 shelter system, and a BASICPAC system expanded to maximum capacity. Table I-1 lists and briefly describes the functions of the BASICPAC major components in the three defined systems.

1. Minimum BASICPAC System

The minimum BASICPAC system is the standard central processor, which consists of five functional units:

- Arithmetic Unit
- Program Unit
- Control Unit
- Memory Unit
- Power Supply Unit

A control panel and a paper tape set provide communication between the operator and the BASICPAC central processor.

Other BASICPAC systems are obtained by adding major BASICPAC components to the standard central processor as shown in Figure I-1 and Table I-1.

2. BASICPAC S-109 Shelter System

The BASICPAC S-109 shelter system is obtained by adding to the standard central processor a communications converter, an input-output converter, and input-output devices associated with these two converters. The S-109 system, designed for mounting in a 2-1/2 ton truck, consists of these units:

Central Processor
Communications Converter
Input-Output Converter (Type A)
Paper Tape Set

The communications converter provides integrated operation of the central processor with remotely located equipments and other central processors via real-time digital data transmitting equipment (e.g., AN/TYC-1 or Kineplex).

The Type A Input-Output converter connects the central processor with four FIELDATA magnetic tape units and one paper tape set.

3. Maximum BASICPAC System

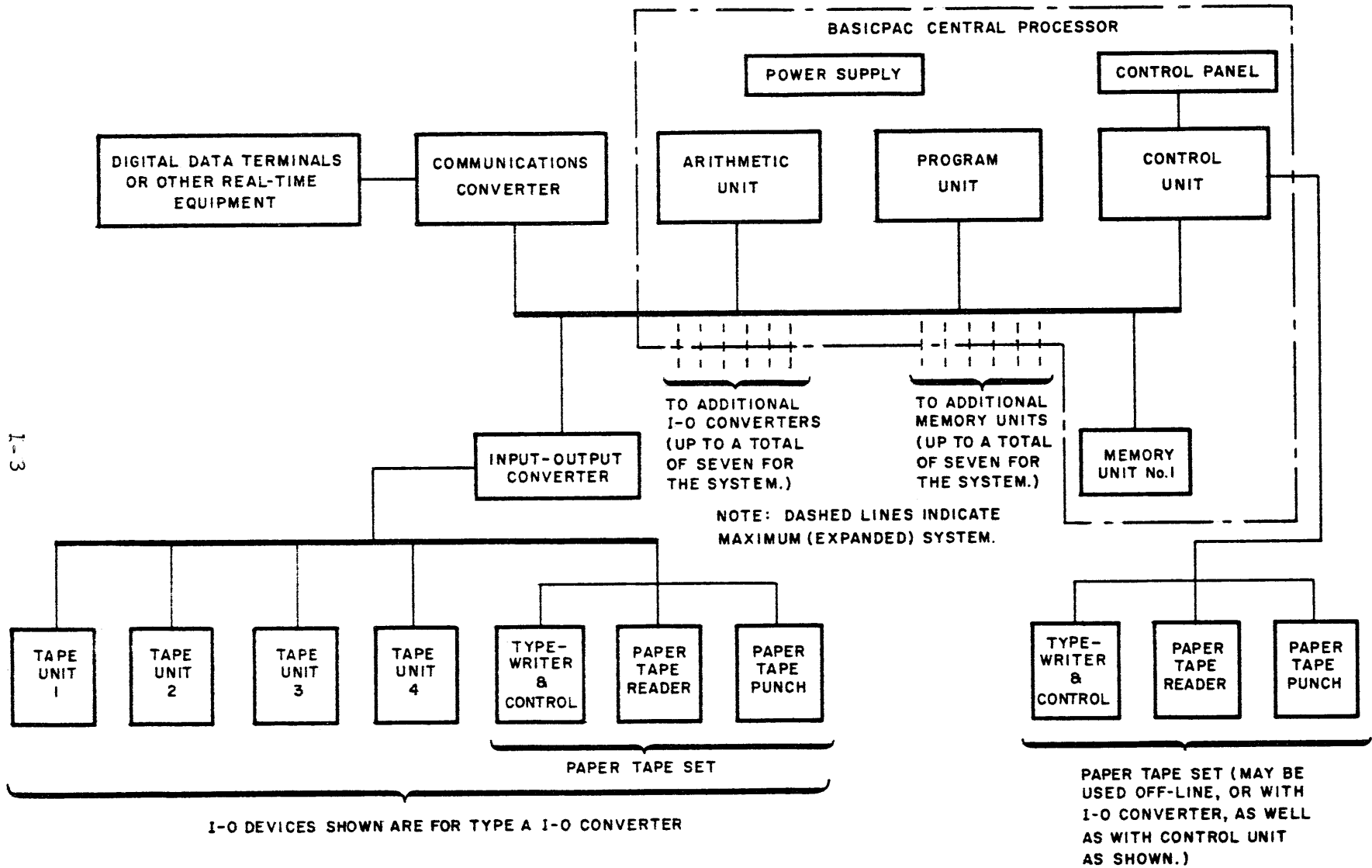
For those applications in which increased input-output and/or memory capacity is required, BASICPAC can be expanded by adding memory units (in 4096-word modules) and input-output converter units.

A maximum BASICPAC system configuration can have seven times 4096 (a total of 28,672) words of internal memory.

4. Busses

The various major components of the BASICPAC system are interconnected by busses, the most important of these being:

- (1) The Major Transfer Bus (MTB), which transfers data between registers in the arithmetic unit, the program unit, the one or several memory units, the control unit, the one or several input-output units, and the communications converter.
- (2) The Address Distribution Bus (ADB), which carries the address of the input-output device or addressable flipflop selected.
- (3) The Memory Selection Bus (MSB), which carries the address of the memory unit selected.



I-3

Figure I-1 BASICPAC System Block Diagram

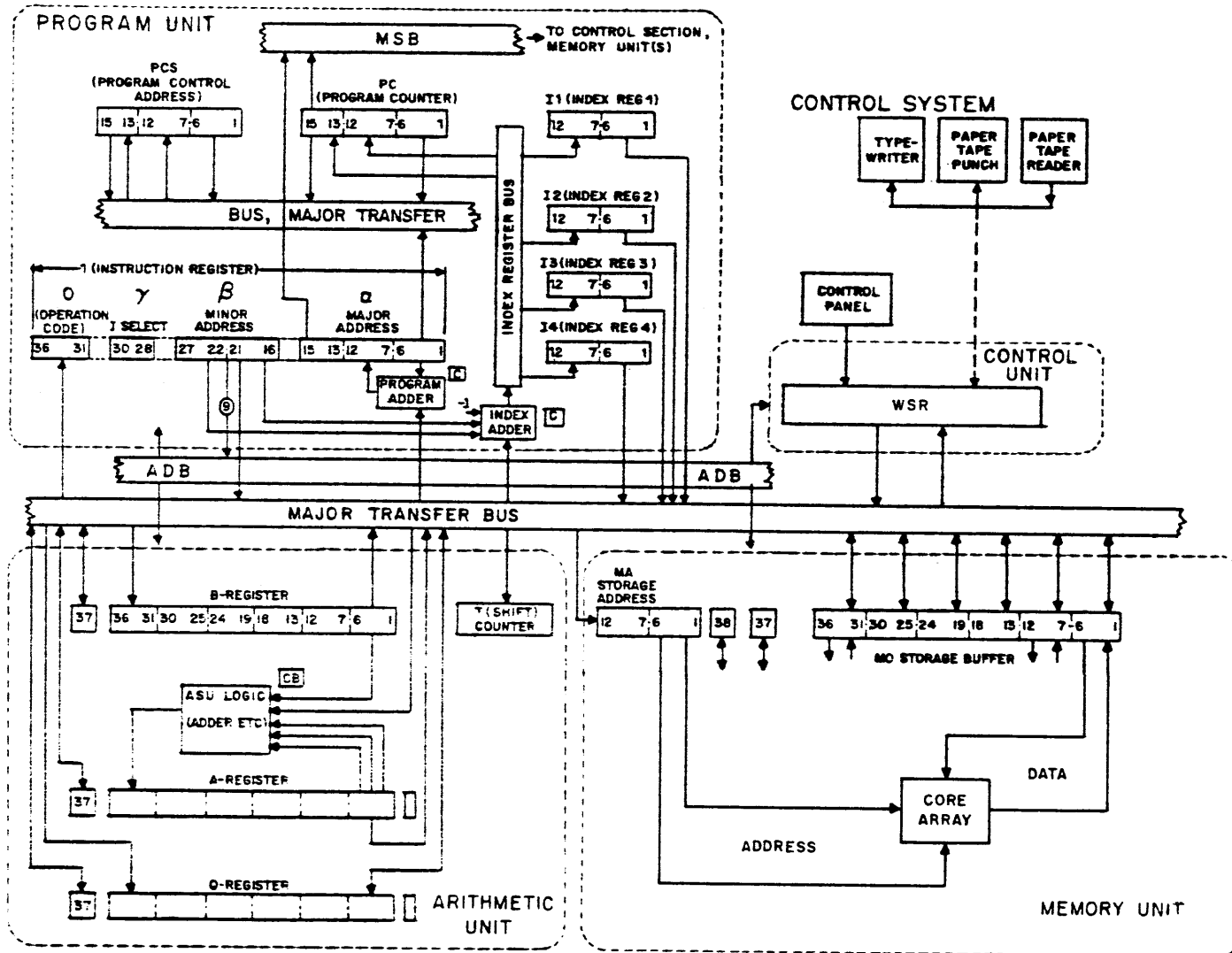


Figure I-2 Block Diagram of BASICPAC Central Processor

TABLE I-1

BASICPAC SYSTEMS AND MAJOR COMPONENTS

Major Components	Function	BASICPAC Central Processor*	BASICPAC S-109 Shelter System*	BASICPAC Maximum System*
ARITHMETIC UNIT	Performs arithmetic operations on binary numbers in a serial-parallel mode, and generates basic timing signals.	1	1	1
PROGRAM UNIT	Provides facilities necessary for storing instructions and for storing and modifying addresses.	1	1	1
MEMORY (STORAGE) UNIT	Provides coincident-current magnetic core storage facility for 4096 38-bit words. Total cycle time, 12 microseconds.	1	1	7
CONTROL UNIT	Provides liaison between BASICPAC and control panel and paper tape set.	1	1	1
POWER SUPPLY UNIT	Furnishes DC power for other major components.	1	1	3
CONTROL PANEL	Provides displays and controls for computer operator.	1	1	1
PAPER TAPE SET	Augments facilities of Control Panel	1	1	1 or more
INPUT-OUTPUT CONVERTER	Provides buffer-synchronization for a selected set of input-output devices		1	7
COMMUNICATIONS CONVERTER	Provides for digital transmission between BASICPAC and up to seven two-way real-time communications channels.	-	1**	1
PACKAGE TESTER	Provides facility for static-testing of logic circuits on individual chassis cards.	1***	1***	1***

- * indicates number of major components required
 ** one limited interrupt input channel, one output channel.
 *** not a functional part of a system, but required for maintenance.

TABLE I-2. SUMMARY OF BASICPAC CHARACTERISTICS

OPERATION TIME (including memory access)

(a)	Clock Frequency	1 μ sec
(b)	Addition	22 to 26 μ sec
(c)	Multiplication	242 μ sec
(d)	Transfer of Control	16 μ sec
(e)	Memory Cycle	12 μ sec
(f)	Reader	30 or 300 char/sec
(g)	Punch	30 char/sec
(h)	Typewriter	10 char/sec

INTERNAL CHARACTERISTICS

(a)	Word Length	38 binary digits, including sign and parity
(b)	Arithmetic	Signed magnitude, fixed point
(c)	Instruction Code	40 orders (expandible to 64 by subroutines)
(d)	Index Registers	4 (expandible to 7)
(e)	Memory Capacity	4096-word memory units; expandible to 7 units (28,672 words)

INPUT-OUTPUT CAPABILITY

(a)	Via Console Unit	Keyboard and paper tape reader and punch.
(b)	Via Input-Output Converter	Magnetic tape, paper tape, keyboard, etc., (several devices per converter); expandible to seven input-output converters operating simultaneously, with one device at a time operating through any one input-output converter.

TABLE I-2 (Continued)

- | | |
|----------------------------------|---|
| (c) Via Communications Converter | Up to six two-way digital communications channels working in real time and one limited interrupt one-way channel. |
|----------------------------------|---|

Total traffic rate is dependent upon programming considerations and upon capacity of input-output devices.

II. BASICPAC CENTRAL PROCESSOR

A. MEMORY UNIT

The BASICPAC Memory Unit provides storage for the program and for data. Each memory unit has a capacity of 4,096 38-bit words (36 data bits, 1 sign bit, 1 parity bit).

1. Memory Address Register (MA) (Not Addressable)

The Memory Address Register (MA) is a 12-bit register which contains the address of the word being accessed.

2. Memory Data Register (MO) (Not Addressable)

The Memory Data Register (MO) is a 38-bit register which serves as a buffer between the computer and the memory cores.

All data is transferred to and from the memory through the data register (MO).

3. Parity

When a word is written into memory a parity bit is generated. When the word is read out of memory the parity bit is checked. Should a parity error occur, a light on the control panel indicates the memory unit in which the error occurred.

4. Memory Priority System

The central computer, the I/O Converters, and the Communications Converter share the memory according to the following priority sequence:

- a. Communications Converter Input
- b. I/O Converters (Numbers 1 through 7 in sequence;
if in the system)
- c. Communications Converter Output
- d. Central Computer

In a system with fewer than the maximum number of converters the absent converters are automatically skipped over. Under maximum traffic conditions, input and output alternate so that no channel is completely neglected.

B. PROGRAM UNIT

The program unit obtains and decodes each instruction, providing modification where necessary and checking for parity error, overflow, and illegitimate addresses or instructions. It contains the following registers and networks:

1. Instruction Register (IR) (Not Addressable)

The 36-bit Instruction Register (IR) receives from memory and temporarily stores each instruction for decoding and execution.

2. Program Counter Register (PC) (Addressable)

The Program Counter is a 15-bit register which contains the address of the next instruction to be executed. Bits 1-12 designate the location of the instruction in a memory unit. Bits 13 to 15 specify one of the seven (0-6) possible memory units. When Bits 1-15 reach the largest memory location value the PC starts over at zero. Instructions occur in a sequential manner (the program counter is automatically advanced by one) except in the case of the seven Transfer and three Sense instructions. For these ten instructions, the program counter may or may not be advanced by one depending on the instruction and the conditions encountered.

3. Program Counter Store (PCS)(Addressable)

The program counter store (PCS) is a 15-bit register used to store the contents of the PC, which contains the address to which the program must return following a subroutine (a discussion of subroutines is given in Section XI). The first instruction (Transfer and Load PCS) transfers control to a subroutine. The last instruction of a subroutine is a "Transfer to PCS" (TRS) which returns

control to the location in the main program whose address is stored in PCS. This permits the programmer to place parts of a program (in subroutine form) outside of the main flow of the program, and to use these subroutines repeatedly.

4. Index Registers (I^Y) (Addressable)

The BASICPAC System contains four index registers, with provisions for adding three more for a total of seven. The index registers are numbered from one to four, and are designated as I¹, I², I³, and I⁴. Each index register is a 12-bit register used primarily for address modification, counting, and looping.

5. Program and Index Adders (Not Addressable)

The Program Adder Network is used to modify the major address of an instruction. The Index Adder Network is used to modify the contents of an Index Register during the execution of an instruction.

6. Timing

The timing signals which control BASICPAC operations are derived from an electronic clock in the arithmetic unit. These timing signals are one microsecond apart, and are called "basic timing units". Each basic computer cycle is divided into an instruction access phase of 12 basic timing units and a variable operand access phase which is an integral number of basic timing units.

C. ARITHMETIC UNIT

The Arithmetic Unit performs arithmetic and logic operations on 36 binary data bits and one algebraic sign bit. This unit includes the A register (Accumulator), the B register, the Q register, the adder-subtractor network (ASU), and the T-counter.

1. The A Register (Addressable)

The A Register (Accumulator) contains 36 data bits and a sign bit (A₃₇). During arithmetic operations the A register contains the following data:

Addition: Augend (Program Placed), then sum.

Subtraction: Minuend (Program Placed), then difference.

Multiplication:	Multiplicand (Program Placed), then major product
-----------------	--

Division:	Divident (Program Placed), then remainder
-----------	---

The A Register is functionally connected with the Q Register during multiplication, division, shifting long and cycling.

2. The B Register (Addressable*)

The B Register is a register which contains 36 data bits and a sign bit (B₃₇). During arithmetic operations the B Register temporarily stores the following data:

Addition:	Not used
-----------	----------

Subtraction:	Not used
--------------	----------

Multiplication:	Multiplier
-----------------	------------

Division:	Divisor
-----------	---------

The B Register also temporarily stores the address of the instruction which was not performed because of a trapping action or an interrupt from the Communications Converter or Input/Output Converter.

3. The Q Register (Addressable)

The Q Register is a register which contains 36 data bits and a sign bit (Q₃₇). During arithmetic operations the Q register contains the following data:

Addition:	(not used)
-----------	------------

Subtraction:	(not used)
--------------	------------

Multiplication:	minor product
-----------------	---------------

Division:	Quotient
-----------	----------

* The B register is addressable only by the α portion of the LOD instruction.

4. The Adder-Subtractor Network (ASU)

The Adder-Subtractor Network performs binary addition and/or subtraction.

5. The T-Counter

The T-Counter is a 7-bit subtracting counter used to count the number of steps in arithmetic or shift instructions. It is automatically preset to the required number of steps and decremented by 1 as each step is performed. When the T-counter reaches 0 the operation stops and a new instruction can begin.

D. CONTROL UNIT

The Control Unit contains the buffer facilities and general control logic for the paper tape set and the operator control panel.

The operator control panel provides manual facilities for entering data and instructions and for starting, stopping, or presetting operational conditions. This panel also contains controls, switches and lights used in maintenance, computer operations, and register display.

The paper tape set augments the functions of the operator control panel.

The control unit, control panel, and paper tape set are further discussed in Section IV.

III. DATA AND INSTRUCTIONS

A. WORD FORMAT

The BASICPAC word consists of 36 bits plus a sign bit and a parity bit. A word can represent a BASICPAC instruction or data. The programmer can use instructions as data, but in general, data cannot be used as instructions.

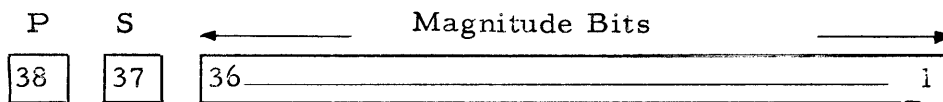
1. Data Words

Data is represented by BASICPAC words in two basic forms: 1) a signed numeric quantity, 2) an alphanumeric quantity. Data can also be represented in any other coded form devised by a programmer for use in a particular problem.

a. Numeric Data

Numeric data is most frequently represented by a word consisting of 36 magnitude bits. Bit 36 is the most significant bit, and bit 1 is the least significant bit.

Bit 37 is the sign bit. A binary zero in this position indicates a positive number; a binary one indicates a negative number. Bit 38 is used for parity checking and is not used by or available to the programmer.

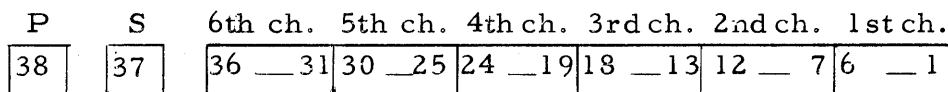


NUMERIC WORD

The position of the binary point (equivalent to the decimal point in a decimal number) is understood to lie between bit 36 and the sign bit, hence all numbers can be considered as fractions greater than minus one but less than plus one. However, the programmer is not restricted to this range since, by the process of scaling (described in Section XI G), he may assume the binary point to lie anywhere within or outside of a word.

b. Alphanumeric Data

For alphanumeric data, the BASICPAC word consists of six characters, each represented by six bits. The use of the sign bit depends on the particular problem.



ALPHANUMERIC WORD

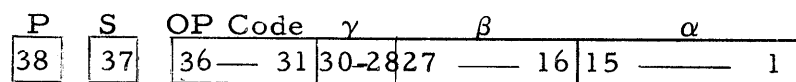
Each character can represent a character of the FIELDATA alphabet. The bit configuration for each character of the standard FIELDATA alphabet and paper tape code are shown in Appendix D.

2. Instruction Words

There are forty BASICPAC instructions, including both computer instructions and input-output instructions. One word format is used for the 31 computer instructions and a different word format for the 9 input-output instructions.

a. Computer Instruction Word

The computer instruction word consists of the format and content as shown below.



COMPUTER INSTRUCTION WORD

The Computer Instruction Word is composed of five sections: alpha (α), beta (β), gamma (γ), operation, and sign.

α (Major Address): Bits 1-15. These bits can designate

- 1) the address of an operand,
- 2) the location of the next instruction in a transfer of control operation.

Bits 1 to 12 can specify a memory (core storage) address, while bits 13 to 15 designate which of the seven possible memory units is to be used. If Bits 13 to 15 read binary 111 then the addressable registers are involved. In this case bits 1 to 6 specify the particular register.

The α portion can also contain data to be transferred to an index register or the magnitude of a shift.

β (Minor Address: Bits 16-27. These bits can designate

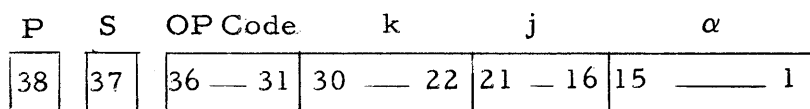
- 1) data to be transferred to an index register,
- 2) a second address,
- 3) an addressable register or an addressable flipflop.
- 4) Bits 16 and 17 control trapping procedure.
- 5) Bits 16-18 control the treatment of overflow.

γ (Index Register Address): Bits 28-30. These bits specify whether an index register is involved and, if so, which one of four is to be used. The legitimate addresses are octal 1, 2, 3, 4. If the index register specified by γ is number 4, then $\gamma+1$ is interpreted as index register 1.

Order Code: Bits 31-36. These bits indicate the order to be performed; e.g., 12_8 , add; 20_8 , multiply.

b. Input-Output Instruction Word

Nine of the forty BASICPAC instructions deal with input-output operations and are specified by a word structured as follows:



I/O INSTRUCTION WORD

Bits 1-15 (α) contain the address of the first memory location into or out of which information is to be transferred. Bits 16-21 contain the address of the I/O device to be used (e.g., paper tape reader, etc.). Bits 22-30 contain the number of words or blocks to be transferred. If bit 30 is set to one on an instruction involving magnetic tape, bits 22-29 represent the number of blocks. If bit 30 is set to zero, bits 22-29 represent the number of words. In I/O operations involving paper tape, bits 22-30 always represent the number of words.

B. ORDER CODES

The BASICPAC order codes are divided into two groups: The central processor order codes which are discussed in Section B-1; and the input-output order codes which are discussed in Section B-2.

The following notation is used throughout this discussion:

$()$	means "The contents of": (A), "the contents of the A register.
\rightarrow	means "replaces" or "enters": (A) \rightarrow Q, "the contents of A enter Q" or "The contents of A replace the contents of Q".
$()_i$	refers to bit(s) i of the specified register or memory location: (A) ₃₇ refers to bit 37 of A. (IR) ₁₋₁₅ refers to bits 1-15 inclusive of the Instruction Register.
$A \xrightarrow{L} A$	indicates a shift of the contents of the specified register in the direction indicated. Subscripts beneath the arrow indicate the number of positions the data is to be shifted.
$A \xrightarrow[3]{R} A$	
\cdot	indicates logical multiplication (see below). $A \cdot Q$, "A and Q" is the bit by bit logical product of the contents of the A and Q registers.
\vee	indicates logical addition (see below). $A \vee Q$, "A or Q", the bit by bit logical sum of the contents of the A and Q registers.

.	0	1
0	0	0
1	0	1

Rules for Logical
Multiplication

v	0	1
0	0	1
1	1	1

Rules for Logical
Addition

The following information is provided in the description of each instruction:

1. Mnemonic code: The standard abbreviation for the instruction.
2. Number code: The machine code number, expressed in the octal system.
3. Name: The name of the instruction.
4. Time: The number of microseconds (μsec) required for the completion of the order.
5. Word Sections: Those sections of the instruction word, if any, which are required for the order.
6. Remarks: An explanation of the operation, with examples in some cases.
7. Legitimate Addresses: Numbers in parenthesis note the references below, which list legitimate addresses for the instruction. When an illegal address is specified in an instruction, the computer proceeds as though a legitimate register whose state or contents were zero had been specified.

REFERENCES

- (1) Core Storage Locations
- (2) A, Q, I^γ , PC, PCS, WSR, CIS', KIW
- (3) A, Q, PCS, WSR
- (4) B
- (5) I^γ , KIW, KOB
- (6) I^γ

1. Central Processor Order Codes

The central processor order codes have been divided functionally into arithmetic, transfer, logical and sense orders. The arithmetic orders deal with addition, subtraction, multiplication and division. The transfer orders vary the normal process of performing sequentially addressed instructions by transferring control to a different portion of memory. The logical orders deal with the shifting and altering of data in memory and other "bookkeeping" functions. The sense orders deal with the setting and interrogation of flipflops to control the flow of the program.

All orders except those with a γ in the Word Sections column can be index modified. When an instruction is index modified, the effective major address is $\alpha + (I^\gamma)$. The exceptions mentioned above are index register instructions which require the specification of an index register.

During the execution of arithmetic or shift instructions, a result may be obtained whose length exceeds the 36-bit capacity of the registers. In this case, only the 36 low-order bits are retained in the register. If any of the excess bits are ones, overflow occurs. The β portion of instructions which may cause overflow; e.g., ADD, ADM, SUB, DVD, SHL and SLL is used to specify computer action in case of overflow. For details, see the descriptions of the individual instructions and section B.1.d.

All unrequired or unused portions of instruction words are customarily filled with zeros.

a. Arithmetic Orders

Mnemonic Code	Code No.	Name	Time	Word Sections
CLA	10	Clear and Add	26 μ s	α

Remarks: $0 + (\alpha) \rightarrow A$

of $\alpha . (1,2)$ * Replace the contents of the A register with the contents

* See page 16 for explanation of these references.

Mnemonic Code	Code No.	Name	Time	Word Sections
---------------	----------	------	------	---------------

ADD	12	ADD	26 μ s	β, α
-----	----	-----	------------	-----------------

Remarks: $(A) + (\alpha) \rightarrow A$

Replace the contents of the A register with the algebraic sum of the contents of the A register and the contents of α . β specifies overflow procedure. (1,2)

ADM	13	Add Magnitude	26 μ s	β, α
-----	----	---------------	------------	-----------------

Remarks: $(A) + |(\alpha)| \rightarrow A$

Replace the contents of the A register with the algebraic sum of the contents of the A register and the absolute value of the contents of α . β specifies overflow procedure. (1,2)

CLS	14	Clear and Subtract	26 μ s	α
-----	----	--------------------	------------	----------

Remarks: $0 - (\alpha) \rightarrow A$

Replace the contents of the A register with the negative of the contents of α . (1,2)

SUB	16	Subtract	26 μ s	β, α
-----	----	----------	------------	-----------------

Remarks: $(A) - (\alpha) \rightarrow A$

Replace the contents of the A register with the algebraic difference of the contents of the A register and the contents of α . β specifies overflow procedure. (1,2)

MLY	20	Multiply	242 μ s	α
-----	----	----------	-------------	----------

Remarks: $(A) \times (\alpha) \rightarrow A, Q$

Compute the product of the contents of the A register and the contents of α . Place the 36 high-order bits of the product in the A register, and place the 36 low-order bits of the product in Q. Both registers have the sign of the product. (1,2).

Mnemonic Code	Code No.	Name	Time	Word Sections
---------------	----------	------	------	---------------

DVD	22	Divide	242 μ s	β, α
-----	----	--------	-------------	-----------------

Remarks: $(A) / (\alpha) \rightarrow Q$

The quotient of the contents of the A register divided by the contents of α is placed in the Q register. The remainder is placed in the A register. β specifies overflow procedure. (1, 2)

Note: If $(A) \geq (\alpha)$, no computation will occur.

DVL	23	Divide Long	242 μ s	β, α
-----	----	-------------	-------------	-----------------

Remarks: $(A) + (Q) \times 2^{-36} / (\alpha) \rightarrow Q$

The contents of the A register and the Q register are treated as "a 72-bit" register and are divided by the contents of α . The quotient is placed in the Q register. The remainder is placed in the A register. β specifies overflow procedure. (1, 2)

b. Transfer Orders

TRU	40	Transfer Unconditional	β, α 16 μ s
-----	----	------------------------	-------------------------------

Remarks: $\alpha \rightarrow PC$

The next instruction performed is taken from memory location α . β specifies trapping procedure. (1).

TRL	41	Transfer and Load	γ, β, α Program Counter
-----	----	-------------------	--

Remarks: $(PC) + 1 \rightarrow PCS$ 26 μ s

$\beta \rightarrow I^\gamma$

$\alpha \rightarrow PC$

The address of the instruction following the TRL instruction is placed in the Program Counter Store Register. The β portion of the instruction word is placed in the index register specified by the γ portion of the instruction. The α portion of the instruction is placed in the Program Counter Register. The next instruction performed is taken from memory location α . This instruction is frequently used in entering closed subroutines. (1)

Mnemonic Code	Code No.	Name	Time	Word Sections
---------------	----------	------	------	---------------

TRS	42	Transfer to Program Counter Store	16 μ s	
-----	----	-----------------------------------	------------	--

Remarks: (PCS) \rightarrow PC

The contents of the Program Counter Store register are placed in the Program Counter Register. The next instruction performed is taken from the memory location specified by the contents of the Program Counter Store register. This instruction is frequently used to exit from closed subroutines. The γ , β and α portions may be used to store constants.

TRX	43	Transfer on Index Register	26 μ s	γ, β, α
-----	----	----------------------------	------------	-------------------------

Remarks: If $I^{\gamma+1} = 0$, $(PC) + 1 \rightarrow PC$

If $I^{\gamma+1} \neq 0$, $I^{\gamma+1} - 1 \rightarrow I^{\gamma+1}$

Then if $I^{\gamma+1} = 0$, $(PC) + 1 \rightarrow PC$

if $I^{\gamma+1} \neq 0$, $I^{\gamma} + \beta \rightarrow I^{\gamma}$ and $\alpha \rightarrow PC$

This instruction is frequently used for n iterations of a series of instructions, any or all of which may be index-modified. The contents of the index register specified by the γ portion of the instruction is examined. If the contents are zero, the next instruction in sequence is performed. If the contents are not zero, they are decreased by one and again examined. If they now equal zero, the next instruction in sequence is performed. If they are still unequal to zero, the contents of the index register specified by the γ portion of the instruction word are increased by the β portion of the instruction word and control is transferred to the instruction in the memory location specified by the α portion of the instruction word.

Note: If there are only four index registers, and if $\gamma = 4$, $\gamma + 1 = 1$. (1)

Mnemonic Code	Code No.	Name	Time	Word Sections
---------------	----------	------	------	---------------

TRP	44	Transfer on Positive Accumulator	16 μ s	α
-----	----	----------------------------------	------------	----------

Remarks: If $(A)_{37} = 0$, $\alpha \rightarrow PC$

If $(A)_{37} = 1$, $(PC) + 1 \rightarrow PC$

If the sign bit of the A register is 0 (meaning positive), the next instruction performed is taken from the memory location specified by the α portion of the instruction word. If the sign bit of the A register is 1 (meaning negative), the next instruction in sequence is performed. (1).

TRZ	45	Transfer on Zero Accumulator	16 μ s	α
-----	----	------------------------------	------------	----------

Remarks: If $(A) = 0$, $\alpha \rightarrow PC$

If $(A) \neq 0$, $(PC) + 1 \rightarrow PC$

If the contents of the A register are equal to plus or minus zero, the next instruction performed is taken from the memory location specified by the α portion of the instruction word. If the contents of the A register are unequal to plus or minus zero, the next instruction in sequence is performed. The sign bit of the A register does not affect this instruction. (1).

TRN	46	Transfer on Negative Accumulator	16 μ s	α
-----	----	----------------------------------	------------	----------

Remarks: If $(A)_{37} = 1$, $\alpha \rightarrow PC$

If $(A)_{37} = 0$, $(PC) + 1 \rightarrow PC$

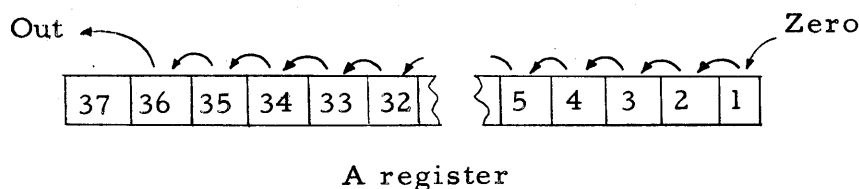
If the sign bit of the A register is 1 (meaning negative), the next instruction performed is taken from the memory location specified by the α portion of the instruction word. If the sign bit of the A register is 0 (meaning positive), the next instruction in sequence is performed. (1).

c. Logical Orders

Mnemonic Code	Code No.	Name	Time	Word Sections
SHL	30	Shift Left	**	β, α

Remarks: $A \xrightarrow[\alpha_{1-7}]{L} A$

Bits 1-7 of the α portion of the instruction specify the number of times the contents of the A register are to be shifted to the left. Bits shifted out of bit 36 of the A register are lost. Zeros replace the vacated low order bits. The sign is not affected. β specifies overflow procedure.

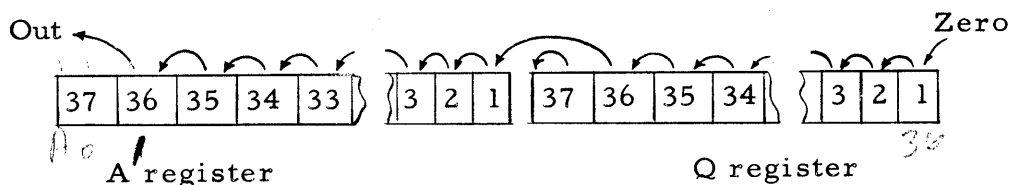


Each arrow indicates the movement of one bit during an SHL instruction.

Mnemonic Code	Code No.	Name	Time	Word Sections
SLL	31	Shift Left Long	**	β, α

Remarks: $A, Q \xrightarrow[\alpha_{1-7}]{L} A, Q$

The A and Q registers are treated as a single 72 bit register. Bits 1-7 of the α portion of the instruction specify the number of times the contents of the A and Q registers are to be shifted to the left. Bits shifted out of bit 36 of the A register are lost. Bit 36 of the Q register enters bit 1 of the A register. Zeros replace the vacated low order bits of the Q register. The sign bits are not affected. β specifies overflow procedure.



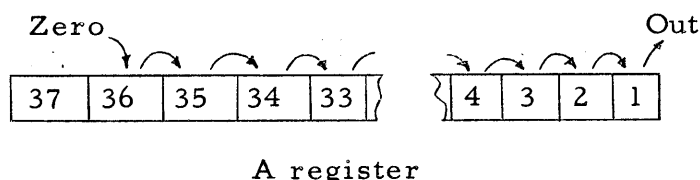
Each arrow indicates the movement of one bit during an SLL instruction.

** $16 \mu s + \alpha 6 \mu s$, α even; $16 \mu s + (\alpha + 1) 6 \mu s$, α odd.

Mnemonic Code	Code No.	Name	Time	Word Sections
SHR	32	Shift Right	**	α

Remarks: $A \xrightarrow[1-7]{R} A$

Bits 1-7 of the α portion of the instruction word specify the number of times the contents of the A register are to be shifted to the right. Bits shifted out of bit 1 of the A register are lost. Zeros replace the vacated high order bits. The sign is not affected.

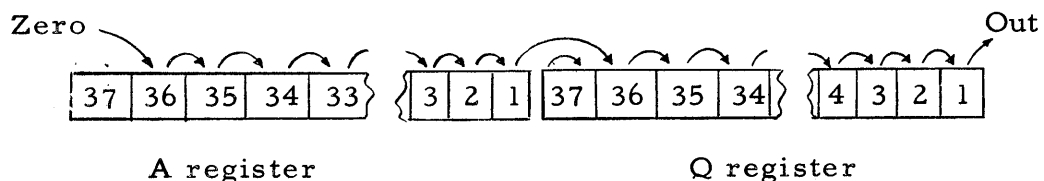


Each arrow indicates the movement of one bit during an SHR instruction.

SRL	33	Shift Right Long	**	α
-----	----	------------------	----	----------

Remarks: $A, Q \xrightarrow[1-7]{R} A, Q$

The A and Q registers are treated as a single 72-bit register. Bits 1-7 of the α portion of the instruction word specify the number of times the contents of the A and Q registers are to be shifted to the right. Bits shifted out of bit 1 of the Q register are lost. Bit 1 of the A register enters bit 36 of the Q register. Zeros replace the vacated high order bits of the A register. The sign bits are not affected.



Each arrow indicates the movement of one bit during an SRL instruction.

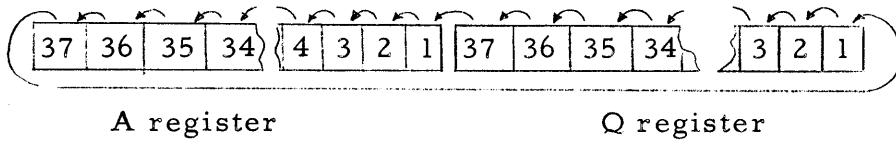
CYL	35	Cycle Long	**	α
-----	----	------------	----	----------

Remarks: $A, Q \xrightarrow[1-7]{L} A, Q$

$A_{37} \longrightarrow Q_1$

The A and Q registers are treated as a single 72-bit circularly connected register. The sign bits are also shifted. Bit 37 of the Q register enters bit 1 of the A register. Bit 37 of the A register enters bit 1 of the Q register.

** $16\mu\text{s} + \alpha 6\mu\text{s}$, α even; $16\mu\text{s} + (\alpha + 1) 6\mu\text{s}$, α odd.



Each arrow indicates the movement of one bit during a CYL instruction.

Mnemonic Code	Code No.	Name	Time	Word Sections
LOD	51	Load	26 μ s	β, α

Remarks: $(\alpha) \rightarrow \beta$

The contents of the register specified by β are replaced with the contents of the register or memory location specified by α . The high order three bits of the register address in β are understood. $(2, 5_\beta), (1, 2, 4_\alpha)$

STR	50	Store	26 μ s	α
-----	----	-------	------------	----------

Remarks: $(A) \rightarrow \alpha$

The contents of the register or memory location specified by α are replaced by the contents of the A register. The contents of the A register are not affected. $(1, 3, 6)$

LDX	53	Load Index Register	20 μ s	γ, β, α
-----	----	---------------------	------------	-------------------------

Remarks: $\beta \rightarrow I^\gamma$
 $\alpha \rightarrow I^{\gamma+1}$

The contents of the index register specified by the γ portion of the instruction are replaced by the β portion of the instruction word. The contents of the index register whose address is one greater than the index register specified by the γ portion of the instruction word are replaced by the α portion of the instruction word. If there are only 4 index registers, and $\gamma = 4, \gamma + 1 = 1$. (anything β) (anything α)

LGA	03	Logical Add	26 μ s	α
-----	----	-------------	------------	----------

Remarks: $(\alpha) \vee (A) \rightarrow A$

The bit by bit logical sum of the contents of the A register and the contents of the register or memory location specified by α is placed in the A register. The sign bits are also logically added. $(1, 2)$

Mnemonic Code	Code No.	Name	Time	Word Sections
---------------	----------	------	------	---------------

LGM 02 Logical Multiply 26 μ s α

Remarks: $(\alpha) \cdot (A) \rightarrow A$

The bit by bit logical product of the contents of the A register and the contents of the register or memory location specified by α is placed in the A register. The sign bits are also logically multiplied (1, 2)

RPA 54 Replace Address 38 μ s α

Remarks: $(A)_{1-15} \rightarrow \alpha_{1-15}$

The configuration of bits 1-15 of the A register replace the configuration of bits 1-15 of the register or memory location specified by α . The contents of the A register are not affected. (1, 3, 6)

MSK 55 Replace Through 38 μ s α

Mask

Remarks: $(A) \cdot (Q) \vee (\alpha) \cdot (\bar{Q}) \rightarrow \alpha$

This instruction is used when it is desired to replace certain bits of the register or memory location specified by α with corresponding bits of the A register. A mask with ones in the bit positions to be changed and with zeros elsewhere must be placed in the Q register before execution of the MSK instruction. The sign of α is affected. The contents of the A and Q registers are not affected. (1, 3, 6)

Ex. To replace the β portion of memory location 3416 with the number 1467₈:

	SN	OP	γ	β	α
before:Q	+	00	0	7777	00000
A	-	12	0	1467	32156
loc. 3416	+	32	7	4453	01276

after:Q	+	00	0	7777	00000
A	-	12	0	1467	32156
loc. 3416	+	32	7	1467	01276

Mnemonic Code	Code No.	Name	Time	Word Sections
HLT	00	Halt	14 μ s	-

Remarks: The central computer operations stop. Any input-output order which is being performed will be completed, but no new orders will be accepted. The γ, β and α portions may be used to store constants.

d. Sense Orders

Mnemonic Code	Code No.	Name	Time	Word Sections
SEN	05	Sense	16 μ s	β, α

Remarks: If $FF\beta = 1, \alpha \rightarrow PC$
 If $FF\beta = 0, (PC) + 1 \rightarrow PC$

If the flipflop specified by β is in the "1" or "set" state, the next instruction performed is taken from the memory location specified by α . If the flipflop is in the "0" or "reset" state, the next instruction in sequence is performed. (Any addressable flipflop β) (1α)

SNS	06	Sense and Set	16 μ s	β, α
-----	----	---------------	------------	-----------------

Remarks: If $FF\beta = 0, 1 \rightarrow FF\beta$ and $\alpha \rightarrow PC$
 If $FF\beta = 1, (PC) + 1 \rightarrow PC$

If the flipflop specified by β is in the "0" or "reset" state, it will be set to 1 and the next instruction performed will be taken from the memory location specified by α . If the flipflop is in the "1" or "set" state, it is not altered and the next instruction in sequence will be performed. (Any addressable flipflop β) (1α)

SNR	07	Sense and Reset	16 μ s	β, α
-----	----	-----------------	------------	-----------------

Remarks: If $FF\beta = 1, 0 \rightarrow FF\beta$ and $\alpha \rightarrow PC$
 If $FF\beta = 0, (PC) + 1 \rightarrow PC$

If the flipflop specified by β is in the "1" or "set" state, it will be reset to 0 and the next instruction performed will be taken from the memory location specified by α . If the flipflop is in the "0" or "reset" state it is not altered and the next instruction in sequence will be performed. (Any addressable flipflop β) (1α)

B.
e. Overflow

Five arithmetic and two logical instructions can cause overflow to occur in the Accumulator. Overflow occurs when the ADD, ADM, SUB, DVD, DVL SHL or SLL instructions produce a number in the Accumulator too large to be accommodated. The result would be a carry into the adjacent bit position, which in this case represents an error. When this indicator contains a 1, overflow has occurred. Overflow can be used by the programmer to indicate an error condition such as improper scaling, or can be used as a programming feature such as to indicate whether certain bits are ones or zeros. No one automatic procedure is ideal for all possible cases, and the BASICPAC programmer has been given complete control of overflow procedures. Note that overflow does not affect the sign bit.

An addressable Overflow alarm flipflop (OA) is used to indicate to the program that overflow has occurred under the conditions set by the program. This flipflop can also halt the computer upon the detection of overflow if so indicated by the program.

Bits 16 - 18 of the β portion of the instructions which may cause overflow determine the procedure to be followed.

<u>$\beta_{18} \beta_{17} \beta_{16}$</u>	<u>Action Before Instruction</u>	<u>If Instruction Causes Overflow</u>
000	Clear OA	Set OA and halt
001	Clear OA	Set OA
010	Clear OA	Set OA and halt
011	Clear OA	No Action
100	Halt if OA = 1	Set OA and halt
101	No action	Set OA
110	Halt if OA = 1	Set OA and halt
111	No action	No action

f. Trapping Mode

During certain operations, especially program debugging, it is very important to be able to trace the flow of control. The trapping mode is a method of computer operation which permits the programmer to examine in detail the flow of control of a program while it is running.

The computer operates in the trapping mode only when the Trapping flipflip (TRA) is set to one and the trapping switch on the control panel is in the "ON" position. The switch on the control panel permits a manual override of the normal trapping action regardless of the state of the TRA flipflop.

When the computer encounters a transfer or sense instruction in the trapping mode, the instruction is not performed. Instead, the address of the instruction being trapped is placed in the α portion of the B register, the Trapping flipflop is reset to zero and control is transferred to memory location 00000. This location must contain an unconditional transfer to a routine which simulates or otherwise processes the trapped instruction and returns control to the main program.

The trapping mode is controlled by bits 16 and 17 of the unconditional transfer instruction and by the previous state of the Trapping flipflop as follows:

<u>TRA before TRU</u>	<u>B17 B16 of TRU</u>	<u>Trapping Action?</u>	<u>Effect on PC</u>	<u>Remarks</u>
0	00	no	No action	No action
0	01	no	No action	1 \rightarrow TRA
0	10	no	No action	No action
0	11	no	No action	0 \rightarrow TRA
1	00	yes	PC-1 \rightarrow B, 0 \rightarrow PC	Trap TRU order
1	01	no	No action	1 \rightarrow TRA
1	10	yes	PC-1 \rightarrow B, 0 \rightarrow PC	Trap TRU order
1	11	no	No action	0 \rightarrow TRA

2. Input-Output Orders

The input-output instructions automatically select the designated input-output device, perform the specified operation, disconnect the device, and return the input-output unit to its initial state. Special consideration is given in this discussion to the sign bit of the data handled by the input-output unit, because certain operations involve the sign bit and others do not. For each of the input-output instructions listed below, the *j* portion of the instruction specifies the device to be used; the *k* portion specifies the amount of information to be processed; and the *α* portion specifies the first memory location to be used in processing the order.

RAN READ ALPHANUMERIC

This instruction assembles a computer word from six 6-bit characters. In interpret sign mode, an additional character (the first in the sequence) is interpreted as the sign. Otherwise the sign bit is automatically set to zero. *

RRV READ REVERSE

Similar to Read Alphanumeric, except that Read Reverse applies only to magnetic tape moving in the reverse direction. *

ROK READ OCTAL

Thirteen paper tape characters are assembled to form a signed (plus or minus) computer data word. The low order bit of the first character is placed in the sign position. Only the low order 3 bits of each of the remaining twelve characters are used. **

WAN WRITE ALPHANUMERIC

This instruction performs the operation as described under Read Alphanumeric except that the flow of information is in the reverse direction. *

WWA REWRITE ALPHANUMERIC

This instruction is applicable only to magnetic tape. With the tape moving in the forward direction, a specified number of words are recorded immediately after two consecutive start of block (SOB) control characters are sensed on the tape. *

* Time: $22 \mu\text{s} + 22 \mu\text{s}/\text{char} + \text{tape time}$

WOK WRITE OCTAL

A computer word is assembled as thirteen FIELDATA characters and punched on paper tape. The sign bit is used in forming the first character. Each set of three data bits is prefixed with the bit pattern 1110 and a parity bit to form one of the remaining twelve characters. *

RWD REWIND

Performs a high-speed rewind of the designated magnetic tape. **

SKP SKIP

The selected magnetic tape is advanced a specified number of blocks. **

BSP BACKSPACE

The same as Skip, except the magnetic tape is moved in the reverse direction. **

a. ICF Behavior

BASICPAC input-output equipment automatically transmits eight bits for each character: six bits of information, one parity bit and one control bit. This control bit and the state of the addressable Interpret Control Function flipflop (ICF) control the performance of input-output orders. When ICF=0, all input output orders are performed exactly as described above. When ICF=1 and an alphanumeric write order is issued, the instruction is performed as described above with the exception that all control bits are transmitted as zero (to indicate "control characters") rather than one (to indicate "data character"). When ICF=1 and an alphanumeric read order is issued, the transmission procedure depends upon the control bit of the first character transmitted. If the control bit is a one (data character), the order is performed as described above. However, if the control bit is a zero, only the first character is read into memory. This character is placed in the low order character position of the memory location specified by the α portion of the read instruction and the equipment proceeds as though the order has been completed. In the case of a magnetic tape order, the tape is moved to the beginning of the next block.

If both ISN and ICF are set to one, ISN will take precedence. When an input-output order involving some unit other than the Communications Converter is completed, the ICF and ISN flipflops are reset automatically.

* $22 \mu\text{s} + 22 \mu\text{s}/\text{char} + \text{tape time}$

** $22 \mu\text{s} + \text{tape time}$

b. Interrupts

When an input-output order is issued either to the Input-Output Converter or to the Communications Converter, both the equipment and the order are examined for errors. If the order is accepted, processing begins and the Central Processor continues its operation. If there is an error or if the input-output order is complete, a deviation from the normal Central Processor operation occurs; a program interrupt is requested. The acceptance or rejection of the interrupt depends upon the state of the addressable Disable Program Interrupt flipflop (DPI). An interrupt request can be accepted only if the Central Processor has completed the current instruction and if DPI=0.

When an interrupt is accepted, the computer automatically sets DPI to one, stores the contents of the Program Counter register (PC) in the α portion of the B register and transfers control to memory location 00001 (F1) or 00002 (F2), depending upon the cause of the interrupt. These locations should contain unconditional transfer instructions to subroutines which will store the current state of the registers in use, determine the cause of interrupt, process any input information and return control to the main program. DPI must be reset by the program to permit new interrupts.

The following conditions will cause an F1 interrupt (interrupt jump to F1):

1. Addressing a non-existent converter
2. Addressing a busy converter
3. Issuing an improper order
4. Initial device malfunction

The following conditions will cause an F2 interrupt (interrupt jump to F2):

1. Completion of transmission (normal interrupt)
2. Device malfunction after the order has been accepted
3. Receipt of a control character when ICF=1
4. Communication Converter malfunction

c. Usage of Control and Converter Input-Output Orders Compared

The paper tape set, consisting of a paper tape reader, a paper tape punch, and a FIELDATA typewriter, can be connected to either the Input-Output Converter or the Control Unit. Connections are made in both cases with standard FIELDATA cable connectors. The same set of input-output orders is used regardless of the connection and functionally the results are identical. Differences arise in the addresses of the devices and in the method of operation.

The paper tape reader, paper tape punch and FIELDATA typewriter have the octal addresses 01, 02, and 03 respectively when connected to the Control Unit; 20, 22, and 26 when connected to the Input-Output Converter.

When a Control Unit device accepts an input-output order, all Central Processor operations are halted until the order has been completed. Processing then continues. When an Input-Output Converter device accepts an order, both input-output and Central Processor operations proceed concurrently until the converter requests an interrupt.

In general, when a Control Unit device detects an error, a flipflop is set and the computer halts; when a converter device detects an error, a flipflop is set and an interrupt is requested.

If a Control Unit device is processing an order, no other operations may proceed concurrently. However, converter devices can be multiplexed with Central Processor operations; if there are n converters present, n devices (one per converter) can operate concurrently with Central Processor.

IV. CONTROL SYSTEM

The BASICPAC control system provides facilities for operator-computer communication. It is composed of the control unit, control panel, and may include paper tape equipment.

The paper tape equipment, which can be connected through either the control unit or input-output converter, includes a paper tape reader, a paper tape punch and a FIELDATA typewriter. It enables the operator to enter and obtain relatively large quantities of information. The paper tape equipment can be controlled either by the program or by the operator through the control panel.

The control panel enables the operator to enter and obtain small quantities of information, to start and stop the computer, to exercise control over program flow, and to issue instructions to the paper tape equipment. This manual discusses only the general purpose control panel. There are, however, special purpose control panels which are designed to be operated by personnel with little or no knowledge of the computer. A special purpose control panel can easily be replaced by a general purpose control panel for maintenance or program debugging.

The control unit contains the electronic equipment used in performing the above functions. It is physically located in the computer cabinetry.

A. CONTROL UNIT

The control unit contains the following registers and flipflops:

1. Word Selection Register (WSR):

WSR is an addressable register used as a buffer for operator-computer communication. WSR receives and transmits information transferred between the central processor and the control unit paper tape equipment. The contents of WSR are always displayed on the control panel. Hence any information transferred to WSR from core storage, registers, or paper tape equipment is automatically displayed. WSR has a 37-bit capacity: 36 data bits and one sign bit.

2. Control Character Buffer (CCB):

CCB is an eight-bit non-addressable buffer register between WSR and the paper tape equipment. The eight bits include six data bits, one parity bit and one control bit; i. e., one FIELDATA character.

When a read order is issued to the paper tape reader, one character at a time is transferred from the reader to the CCB. The CCB transfers the data bits (six alphanumeric or three octal) to the low order bit positions of WSR, which has been shifted to the left to receive them. When enough characters have been transmitted to WSR to complete a word, the contents of WSR are sent to the appropriate memory location, and the process is repeated until the order has been completed.

Write orders are handled in a similar manner. The low order data bits of WSR are transmitted to the CCB and from there to the punch or typewriter, where a FIELDATA character is assembled and typed or punched out to the operator.

3. Sense Flipflops (SFF₁₋₈):

SFF₁₋₈ are one-bit addressable registers which store information for the program. They may be set by the program or by the operator.

B. CONTROL PANEL

The BASICPAC control panel is effectively a plug-in assembly which can be interchanged with or replaced by another control panel by changing cable connections.

The general-purpose control panel is placed on a desk or console structure, and is connected by cables to the control unit in the computer cabinetry.

The illuminated displays of the control panel have the following color assignments:

1. Neon light: Computer conditions to be monitored.
2. Red-covered incandescent lights: conditions requiring operator attention.
3. Amber-covered incandescent lights: conditions of which the operator should be aware.
4. Green-covered incandescent lights: Normal conditions not requiring operator attention.

The two-section front surface is divided into five functional areas: Power, Register, Operation, Error and Sense Flipflops (Figure IV-1).

1. Power Area Display and Controls

The power area switches and indicators concern the BASICPAC computer power supply unit and the neon indicators on the control panel. Only the S-109 type of package tester is included under this power control.

The NEON TEST switch is a spring-loaded pushbutton which operates all the neon lights except the nixie indicators on the control panel.

The POWER ON-OFF control is a guarded three-position switch which establishes or cuts off the computer power supply after momentary displacement from the center position. The green READY indicator is lighted to indicate that the power supply has been turned on.

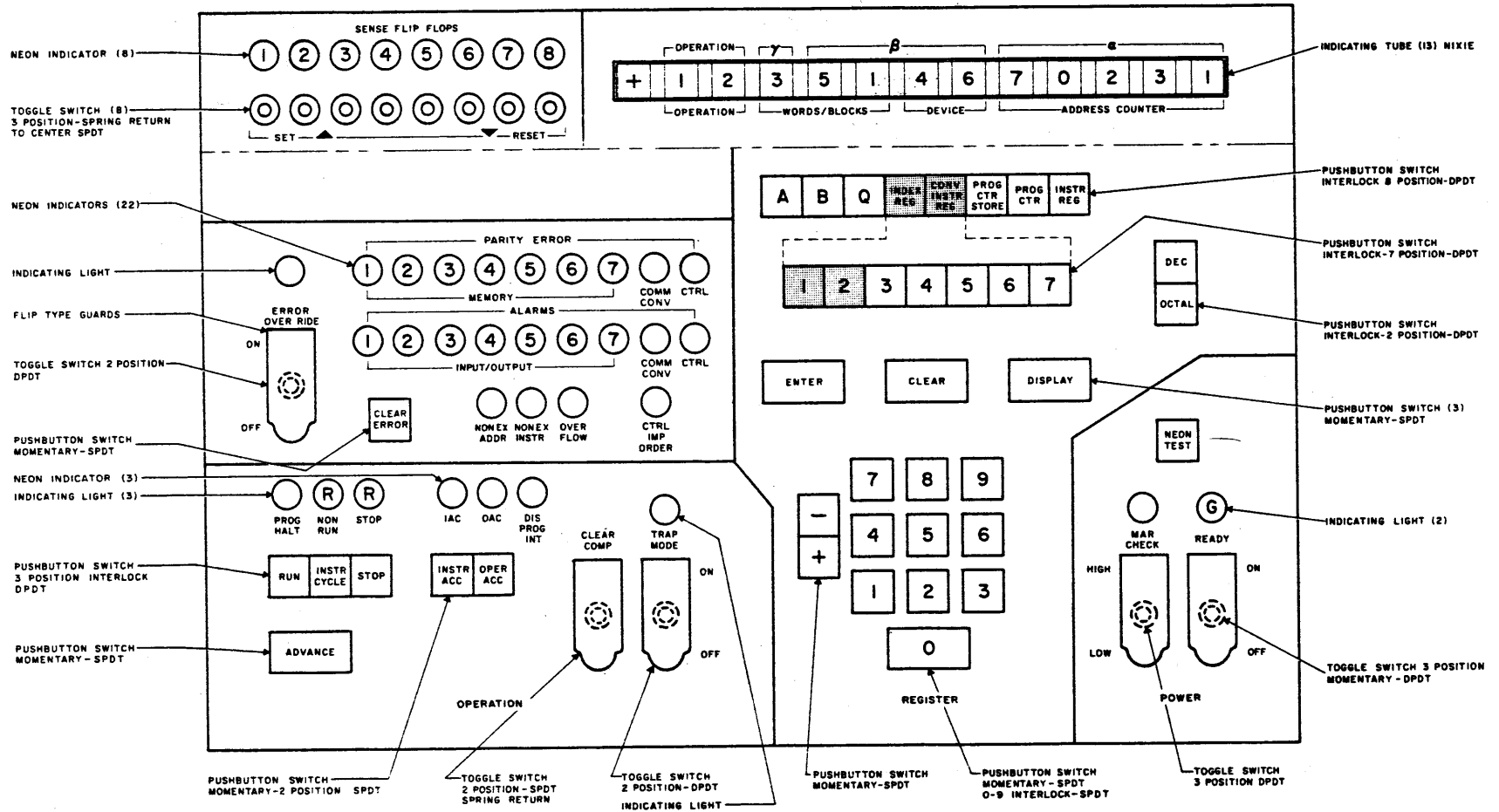


Figure IV-1 BASICPAC Control Panel

The MARGINAL CHECK control is a guarded 3-position switch which controls the marginal check voltage (MCV) in the computer. With the switch in the center position, this MCV is set at its normal (+4 volts) level. With the switch in the high (MCV=+6 volts) or low (MCV=+2 volts) position, the amber MAR CHECK light, located above the MAR CHECK switch on the control panel, is lighted.

2. Register Area Display and Controls

The register area of the control panel permits data entry and display while the computer is in the non-run mode. When the computer is not in the non-run mode the controls in this area are ineffective.

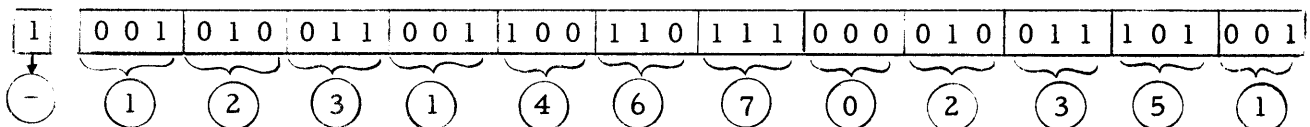
The contents of WSR are displayed by a set of neon tubes located in the top section of the control panel.

Information in the WSR may be displayed octally or decimally, as determined by depressing one of the two switches on the control panel OCTAL and DEC. The information then passes through an octal or decimal decoder network. During an octal display, each nixie tube indicates the contents of 3 bits of the WSR. There are eight possible binary combinations in three bits:

Binary		Octal
000	=	0
001	=	1
010	=	2
011	=	3
100	=	4
101	=	5
110	=	6
111	=	7

The octal number displayed in each nixie tube of the control display range from 0 to 7. The sign bit is displayed as plus or minus.

WSR



OCTAL NIXIE WSR DISPLAY

If the 37 bits of the WSR are to be displayed decimally, the register is divided into 6 groups of 6 bits each. The contents of 4 of the 6 bits in each group are displayed by 6 nixie tubes (every other one), which indicate the contents in FIELDATA 0 to 9:

Binary Displayed		Decimal
1 1 0	0 0 0	0
1 1 0	0 0 1	1
1 1 0	0 1 0	2
1 1 0	0 1 1	3
1 1 0	1 0 0	4
1 1 0	1 0 1	5
1 1 0	1 1 0	6
1 1 0	1 1 1	7
1 1 1	0 0 0	8
1 1 1	0 0 1	9

The nixie tube in the most significant position represents the sign bit of the WSR.

The word displayed may be any of the types shown in Table IV-1. The markings above and below the nixie tubes indicate the two most frequently used types: central processor instruction word and input-output instruction word.

The set of twelve pushbuttons (+, -, 0, 1, 2, ..., 9) located at the bottom of the register area of the control panel permit the operator to alter the contents of WSR. The sign is altered directly by pressing either the + or - button. The data portion of WSR is altered by pressing the desired keys. Each key transfers its binary equivalent into the low order bit positions of WSR while the previous contents of WSR are shifted to the left. When the DEC button is pressed, buttons 0-9 enter their 6-bit FIELDATA code equivalents (60g to 71g) into the low order six bits of WSR. When the OCTAL button is pressed, buttons 0-7 enter their 3-bit octal equivalents (000₂ to 111₂) into the low order three bits of WSR. (8 and 9 are ineffective in octal mode.)

Two rows of interlocked switches and a third row of three separate switches control the disposition of the information in WSR. The eight register selection buttons (A, B, Q, ...) select the computer register into which or from which the contents of WSR are to be transferred. Each of the center two switches in this array (INDEX REG and CONV INSTR REG) concerns a set of registers rather than a single register. The row of

Indicator Operated ¹		WSR Bit	Instruction Word		Data Word	
Decimal ²	Octal		Computer	I/O	Alphanumeric	Binary
1	1	37	-	-	Sign (optional)	Sign
3	2	36	0 (oper. code)	Operation	first character	2 ⁻¹ bit
		35				2 ⁻²
		34				2 ⁻³
5	3	33	γ (index reg. selection)	Words/ Blocks	second character	2 ⁻⁴
		32				2 ⁻⁵
		31				2 ⁻⁶
7	4	30	β (minor address)	Device	third character	2 ⁻⁷
		29				2 ⁻⁸
		28				2 ⁻⁹
9	5	27	α (major address)	Address Counter	fourth character	2 ⁻¹⁰
		26				2 ⁻¹¹
		25				2 ⁻¹²
11	6	24	α (major address)	Address Counter	fifth character	2 ⁻¹³
		23				2 ⁻¹⁴
		22				2 ⁻¹⁵
13	7	21	α (major address)	Address Counter	sixth character	2 ⁻¹⁶
		20				2 ⁻¹⁷
		19				2 ⁻¹⁸
11	8	18	α (major address)	Address Counter	fifth character	2 ⁻¹⁹
		17				2 ⁻²⁰
		16				2 ⁻²¹
13	9	15	α (major address)	Address Counter	fifth character	2 ⁻²²
		14				2 ⁻²³
		13				2 ⁻²⁴
11	10	12	α (major address)	Address Counter	fifth character	2 ⁻²⁵
		11				2 ⁻²⁶
		10				2 ⁻²⁷
13	11	9	α (major address)	Address Counter	fifth character	2 ⁻²⁸
		8				2 ⁻²⁹
		7				2 ⁻³⁰
13	12	6	α (major address)	Address Counter	fifth character	2 ⁻³¹
		5				2 ⁻³²
		4				2 ⁻³³
13	13	3	α (major address)	Address Counter	fifth character	2 ⁻³⁴
		2				2 ⁻³⁵
		1				2 ⁻³⁶

1. Indicators numbered from left (signs indicator is no. 1)
2. Decimal indication for only 10 of the combinations possible for six bits.

Table IV-1. Nixie Indication Interpretations.

seven buttons (1-7) immediately below the eight switch row are used to select a specific index register or Input-Output Converter Instruction Register when necessary. Four index registers are used in the standard system, but there are logical provisions for seven.

The row of three switches located in the center of the register area determines whether information is to be transferred to the selected register from WSR (ENTER) or to WSR from the selected register (DISPLAY). The CLEAR button sets WSR to + zero.

3. Operation Area Display and Controls

The computer can be in one of four phases which are indicated on the control panel by the IAC and OAC lights as follows:

IAC	OAC	PHASE
0	0	0 HALT
1	0	1 INSTRUCTION ACCESS
0	1	2 OPERAND ACCESS
1	1	3 EXTENDED ACCESS (used in the Shift, Multiply and Divide Instructions)

These indicators can provide several types of information to the operator. For example, if a parity error is indicated, the IAC and OAC lights indicate the phase in which the parity error occurred.

The two interlocked pushbutton switches labeled INSTR ACC (Instruction Access) and OPER ACC (Operand Access) are depressed to set the initial phase of the computer. The operand access phase is used to perform an instruction already present in the instruction register.

The Mode operating controls consist of three interlocked pushbutton switches labeled RUN (Run Mode), INSTR CYCLE (Instruction Cycle Mode) and STOP (Stop Mode).

Run Mode, the normal console operation, is entered by depressing the RUN switch. It can be terminated by pressing either the STOP button or the INSTR CYCLE button; however, the current I/O orders and instruction will be completed before the computer enters the Non-Run Mode.

The Instruction Cycle Mode is effectively a Non-Run Mode except that one instruction is performed whenever the ADVANCE button is pressed. This is also known as "step" mode.

When the STOP button is depressed, the Run Mode is released and the ADVANCE bar is disabled, allowing the computer to complete the instruction and any I/O instruction in CIS. The computer then enters into the Non-Run Mode.

The ADVANCE bar is a momentary pushbutton switch used to start the machine after the program has been prepared. If the ADVANCE bar is pressed while the computer is in the Instruction Cycle Mode, only one instruction is completed. The ADVANCE bar is disabled by the following conditions:

- 1) Run Mode and Computer Error
- 2) Instruction Cycle Mode and Error
- 3) Stop Mode
- 4) Programmed halt on overflow and overflow present

The conditions under which the Non-Run Mode exists are: 1) the INSTR CYCLE button is depressed and the H (halt) flipflop is "one"; or 2) the STOP button is depressed the H flipflop is "one", and the I/O converters are neither busy nor requesting interrupt. When this mode occurs, the NON-RUN indicator lights.

The STOP indicator displays the condition of the non-addressable Halt (H) flipflop located in the computer. This light is set by any of the following conditions:

- 1) Error halt when the computer is in the Run Mode.
- 2) Error halt when the computer is in the Instruction Cycle Mode.
- 3) The STOP button is depressed and the computer is not running or is at the end of an instruction.
- 4) The INSTR CYCLE button is depressed and the computer is not running or is at the end of an instruction.
- 5) The machine is in its initial state.

The Program Halt (PROG HALT) light indicates that the computer is in phase 0 and a halt instruction is present in the instruction register. When the Halt phase is indicated, the ADVANCE bar must be depressed to continue.

The Disable Program Interrupt (DIS PROG INT) light indicates that the program will not permit any interrupts (DPI=1).

The TRAP MODE switch is a two-position toggle switch. When this switch is manually set to "OFF", all transfers to the trapping mode are inhibited. When the TRAP MODE switch is set to "ON" the computer will enter into a subroutine at a point determined by the program. This mode is used when there is a need to debug a program in the Run Mode rather than step-by-step.

The Clear Computer (CLEAR COMP) switch is a two-position toggle switch used to preset the computer to the initial state. It operates only if the computer is in the Non-Run Mode. It can be used to interrupt a partially completed I/O order.

4. Error Control and Display

The error control and display section of the control panel consists of indicators to display the location of errors, and controls for clearing error flipflops or continuing despite error. It is used extensively during running operation, program debugging, and system checking.

There are nine parity error neon indicators. Seven of these identify which of the seven memory units has noted a parity error, one indicator is associated with the communications converter and one is associated with the control unit input-output system (CTRL). The parity error flipflops can be cleared only by activating the CLEAR ERROR switch.

Input-Output errors are those errors which occur during the execution of an I/O order if the instruction addresses a converter. The I/O errors are identified by unit (1 to 7). Further indications of the type of converter error (e.g., improper order or device malfunction) are displayed on the individual unit. The I/O alarm flipflops can be reset or cleared either by the issuance of a new I/O order to the same converter or by program control. Similar indicators display an error in the communications converter and in the control unit input-output system.

The Nonexistent Address (NONEX ADDR) indicator shows that a nonexistent address is about to be transferred to the program counter by either a transfer of control or sense instruction. The computer will stop

without performing the instruction unless the error override is set. In this case, the instruction will be performed. However, since any illegal address is treated as though it contained all zeros, IR will then equal zero and the computer will halt. This indicator is reset by the CLEAR ERROR switch.

The Nonexistent Instruction (NONEX INSTR) indicator shows that an order code not implemented in the machine is in the instruction register, in which case the instruction remains in the instruction register and the computer stops, regardless of the position of the Error Override switch. The flipflop is reset by the CLEAR ERROR switch.

The OVERFLOW neon indicates that there has been a carry into a nonexistent position of the accumulator. This state may or may not be intentional, and the program determines whether or not the computer will stop. The instructions in which overflow is possible are ADD, ADM, SUB, DVD, DVL, SHL and SLL.

The Control Unit Improper Order (CTRL IMP ORDER) indicator lights when there is an instruction fault caused by addressing a nonexistent device or issuing an inconsistent order.

The ERROR OVERRIDE switch is a two-position toggle switch which signals the computer to continue despite errors. A light indicates whether or not the ERROR OVERRIDE Switch is on.

The CLEAR ERROR switch is a momentary pushbutton switch which clears error indications for memory parity, control parity, control device alarm, control improper order, non-existent address, and non-existent instruction. It can operate only in the Non-Run Mode.

5. Sense Indicators and Switches

The general sense flipflops (SFF₁₋₈) each hold one bit of information for the computer program. These flipflops can be controlled either by the program or by the operator through the control panel. The state of each sense flipflop is indicated by a correspondingly numbered neon light: if the flipflop is in the "one" state, the indicator light is on; if the flipflop is in the "zero" state, the indicator light is off.

The operator can manually set or reset the sense flipflops by pressing the corresponding three-position toggle switches up or down, respectively. The toggle switches have a spring return to the center position where the flipflops are under the control of the program.

C. PAPER TAPE SET

The paper tape set consists of a paper tape reader, a paper tape punch, and a FIELDATA typewriter. It can be connected either to the control unit or to the input-output converter. The paper tape reader will accept only "read" orders; the paper tape punch and the FIELDATA typewriter will accept only "write" orders.

1. Orders

Orders issued to the control unit paper tape equipment may be considered as a class of computer instructions which differ from other computer instructions only in the length of time required for execution.

When an order is placed in the instruction register (IR) it is examined for errors such as non-existent command, improper address, etc., just as any other instruction is examined. If an error is detected, the control error indicator on the control panel is turned on and the computer halts. If no errors are detected in the instruction, the device addressed is examined for errors (not present, etc.). If a malfunction exists, the device alarm flipflop (DVA) is set to one, the control error indicator in the control panel is lighted, and the computer halts. If the device is ready, processing begins.

The K portion of the instruction word specifies the number of words to be processed and the α portion specifies the corresponding memory locations. As each word is processed, the α portion of IR is incremented by one and the K portion is decremented by one. When K equals zero (or when other conditions are satisfied) the order is assumed to be completed and processing ceases. At this point the altered instruction word can be displayed from IR. α specifies the address of the word which would have been processed next, and K specifies the original number of words to be processed, decremented by the number of words which actually were processed.

Control unit input-output orders can be manually issued through the control panel in Non-Run mode by the same procedure used in issuing other computer instructions. The instruction to be performed is keyed into WSR and entered into IR. Operand Access mode is selected and the Advance bar is pressed. The equipment re-enters Non-Run mode when the order has been completed; another order can then be issued.

Two addressable flipflops supply information to the program concerning the completion of control paper tape orders.

The Control Unit Stop flipflop (0360) indicates whether or not a stop character (57g) has been received.

The Control Unit Control Indicator (0361) flipflop indicates whether or not a control character has been received. Acceptance of a new order automatically resets these flipflops to zero.

2. Speeds

The typewriter operates at a maximum steady output speed of ten characters per second. Except for short bursts, a faster operating speed may jam the type bars together.

On-line, the paper tape reader operates at either the normal speed of 30 characters per second or at high speed of 300 characters per second. Off-line, speeds are either 10 or 30 characters per second, depending upon the mode of operation.

The paper tape punch operates at either 10 or 30 characters per second, depending upon the mode of operation.

3. Typewriter Control

The typewriter control unit contains two sets of flipflop registers (keyboard and main), five mode flipflops which store the designation of the mode to be used, and associated logic.

Pressing a pushbutton control key on the typewriter clears all the mode flipflops, then sets the appropriate mode flipflop and resets the keyboard register.

a. Mode Control

The five modes used in the operation of the paper tape set are:

Keyboard to Punch,

Reader to Printer,

Reader to Punch,

Print and Punch, and

On-Line.

Each mode is selected by depressing the associated control key. The Print and Punch mode can be used only On-Line or with a reader mode.

b. Other Control Keys

In addition to the five mode control keys, four other control keys are used on the typewriter. These are:

1) Step Reader

This key causes the reader to read a single character.

2) Reader Start, Stop

This key starts or stops the reader only when the reader is connected to the punch or typewriter.

3) Tape Feed

This key permits the generation of leader tape when the typewriter is connected off-line to the punch.

4) Back Tape

This control key is active only when the punch and typewriter are connected off-line. It causes the tape in the punch to space back one character.

Alarm

An alarm (similar in appearance to a control key) is active in any of the four off-line modes, and when lit indicates one of the following conditions:

- 1) No tape in reader.
- 2) Reader plastic tape guide not properly positioned.
- 3) Power off at reader and punch, when power on at typewriter.
- 4) Data cables not connected.

Mode Descriptions

On-Line

Control is transferred from typewriter control box to the computer. Data lines are connected directly to the computer. The typewriter keyboard is locked.

Reader to Typewriter

The reader is connected to the typewriter so that the tape is printed as it is read. The punch and typewriter keyboard are disabled.

Reader to Punch

The reader can be connected to the punch either directly or through the control box logic. Tape fed into the reader is reproduced by the punch. The typewriter is disabled.

Keyboard to Punch

The typewriter is connected to the punch. The reader is disabled. Characters struck on the typewriter are punched on tape in FIELDATA paper tape code.

Print and Punch

Data is transmitted to both the typewriter and punch.

Typewriter to Punch

Information cannot be entered directly into the computer via the keyboard. Characters struck on the keyboard are punched in FIELDATA code in paper tape, which is then read into the computer. FIELDATA codes are given in Tables III-1 and III-2.

Tab and Carriage Return are manually set prior to the typing operation.

Character keys are provided for carriage return, space, tab, shift, and backspace. The Master Space key causes the special character ≈ to be printed. The Special key causes a rectangle to be printed.

The Delete key causes a delete character to be punched on tape.

Typewriter from Reader or Computer

Receipt of an upper case control character causes the typewriter to print all subsequent letters in upper case except for symbols which require lower case (e.g., numerals), each of which automatically causes the typewriter to shift to lower case, then return to upper case. The lower-case control character causes a comparable operation. If an upper-case control character is received when the typewriter is

already in upper-case (or a lower-case character in lower case), the typewriter pauses for approximately one-half second, then continues. Careless use of these control characters can therefore slow the operation of the typewriter.

4. Paper Tape Reader

The reader can be connected off-line to the punch and/or typewriter or on-line to the computer. A delete character or a blank (a character having only sprocket holes) is ignored by the reader.

The reader operates at two speeds; normal (30 characters per second), and high (300 characters per second).

Pressing the Normal Speed button when the reader is operating at high speed returns the reader to normal speed.

Pressing the High Speed button when the reader is operating at normal speed steps the reader to high speed. Any malfunction occurring at high speed causes the reader to revert to normal speed.

5. Paper Tape Punch

The punch can be used either to prepare program tapes (off-line) or to punch information from the computer onto tape. The punch rate is 30 characters per second.

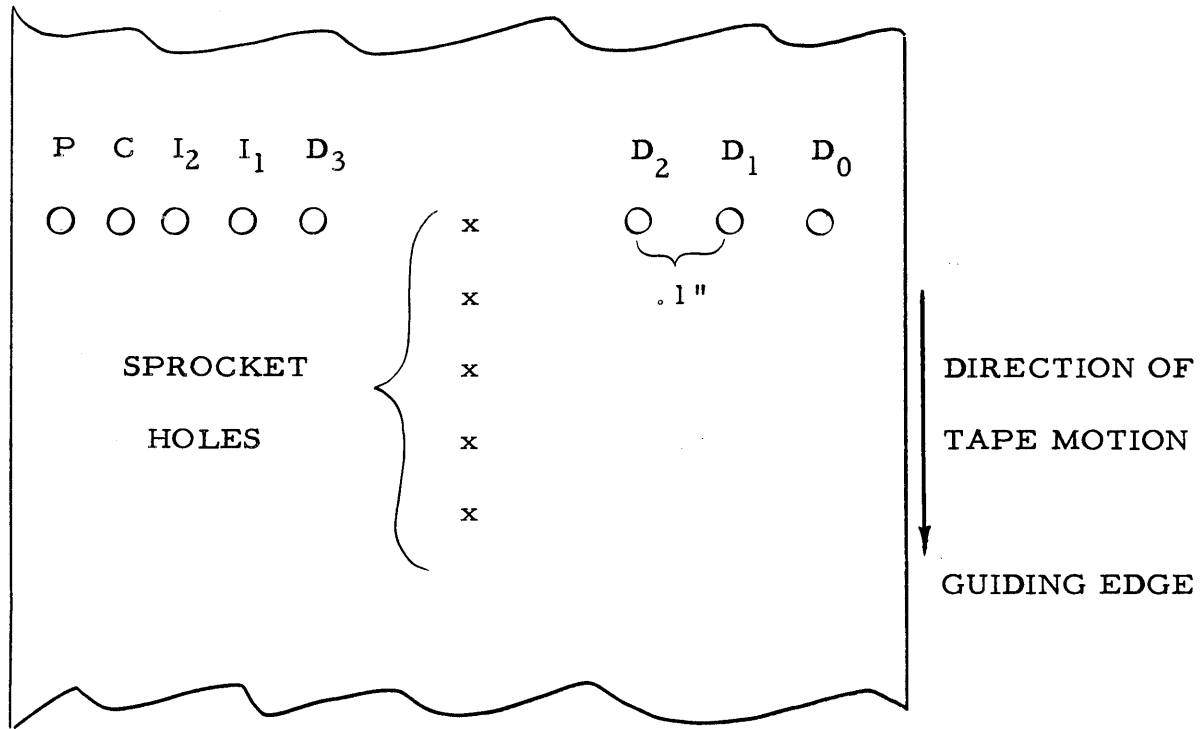
Operation of the punch and typewriter together results in a speed of 10 characters per second.

PAPER TAPE SET OPERATING SPEEDS

UNIT	SPEED(S)
Reader	30 CHAR/SEC, HIGH SPEED = 300 CHAR/SEC
Punch	30 CHAR/SEC, PRINT WHEEL = 15 CHAR/SEC
Typewriter	10 CHAR/SEC

Paper Tape Format

Information is physically positioned on the paper tape as follows:



Tape Characteristics

TAPE WIDTH	8 CHANNEL TAPE 1"
	7 CHANNEL TAPE 7/8"
	5 CHANNEL TAPE 11/16" or 7/8"
SPACE BETWEEN HOLES	.1"
DIAMETER DATA HOLES	.072"
DIAMETER SPROCKET HOLES	.046"

V. INPUT-OUTPUT CONVERTER

A. GENERAL DESCRIPTION

An Input-Output Converter provides communication between the core memory and one or more input-output devices. Only one of the devices attached to an Input-Output Converter can be in communication with the computer core memory at any given time. Magnetic tape units and paper tape equipment are currently available for use with the Input-Output Converters.

B. FUNCTION

The Input-Output Converter performs the following functions:

- (1) Instruction control
- (2) Data control
- (3) Error control
- (4) FIELDATA control function processing
- (5) Program interruption
- (6) Communication with the central processor

1. Instruction Control

An I/O instruction addressed to an Input-Output converter is transferred from the computer instruction register to the instruction register of the appropriate Input-Output Converter.

The converter also receives the contents of the Interpret Sign (ISN) and Interpret Control Function (ICF) flipflops, which were set by instructions given prior to the input-output instruction. These flipflops are automatically reset after their contents have been transferred to the converter.

When the converter unit receives an input-output order in its instruction register it enters the BUSY state, indicated by a "one" in its Converter Busy (CVB) flipflop. No other instruction can be entered into this converter unit until the processing of the current instruction has been completed. However, the computer can sense the contents of the Converter Instruction Register at any time. The CIS and other logical elements of the Input-Output converter are shown in Figure V-1.

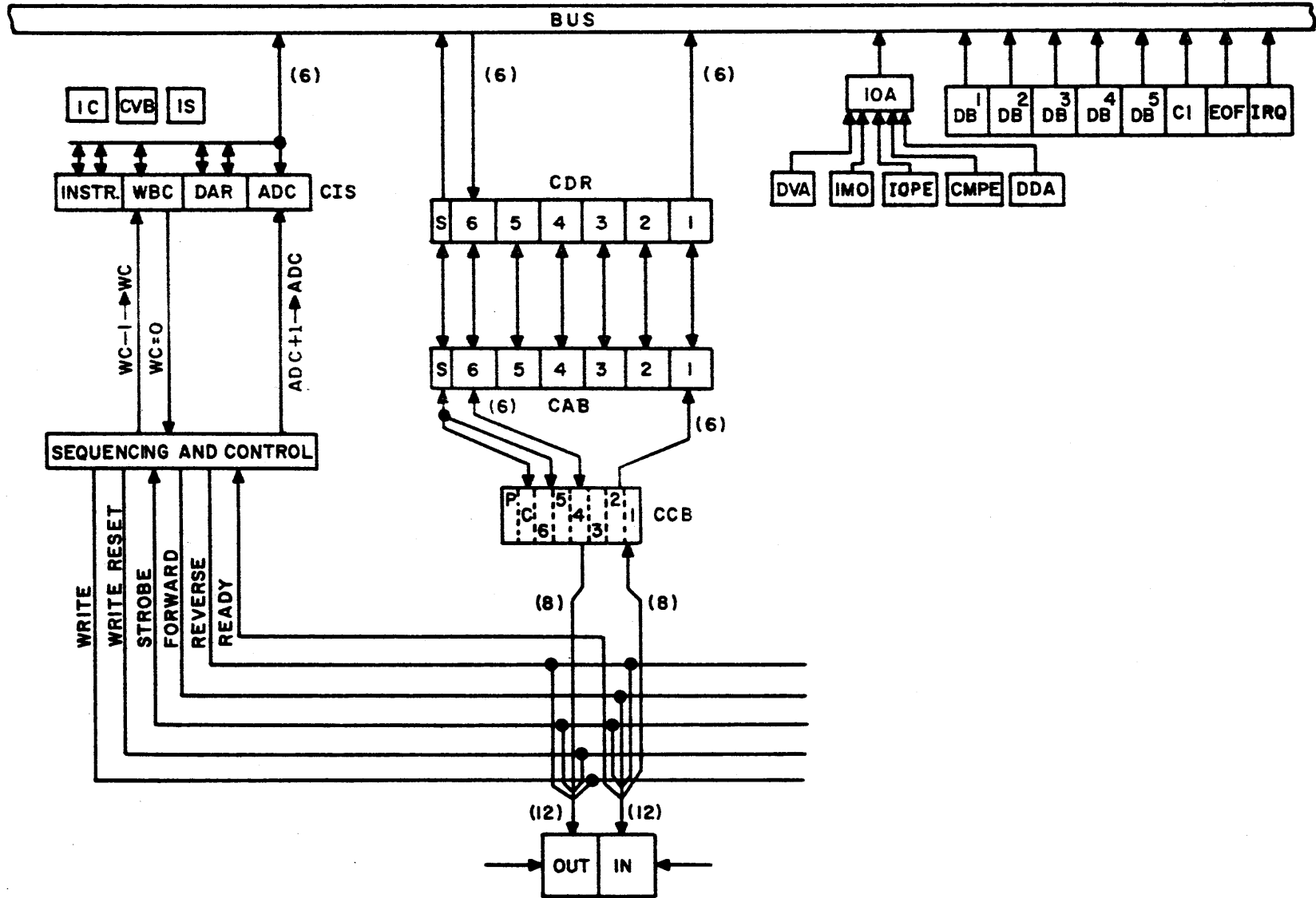


Figure V-1 Input -Output Converter Type A,Block Diagram

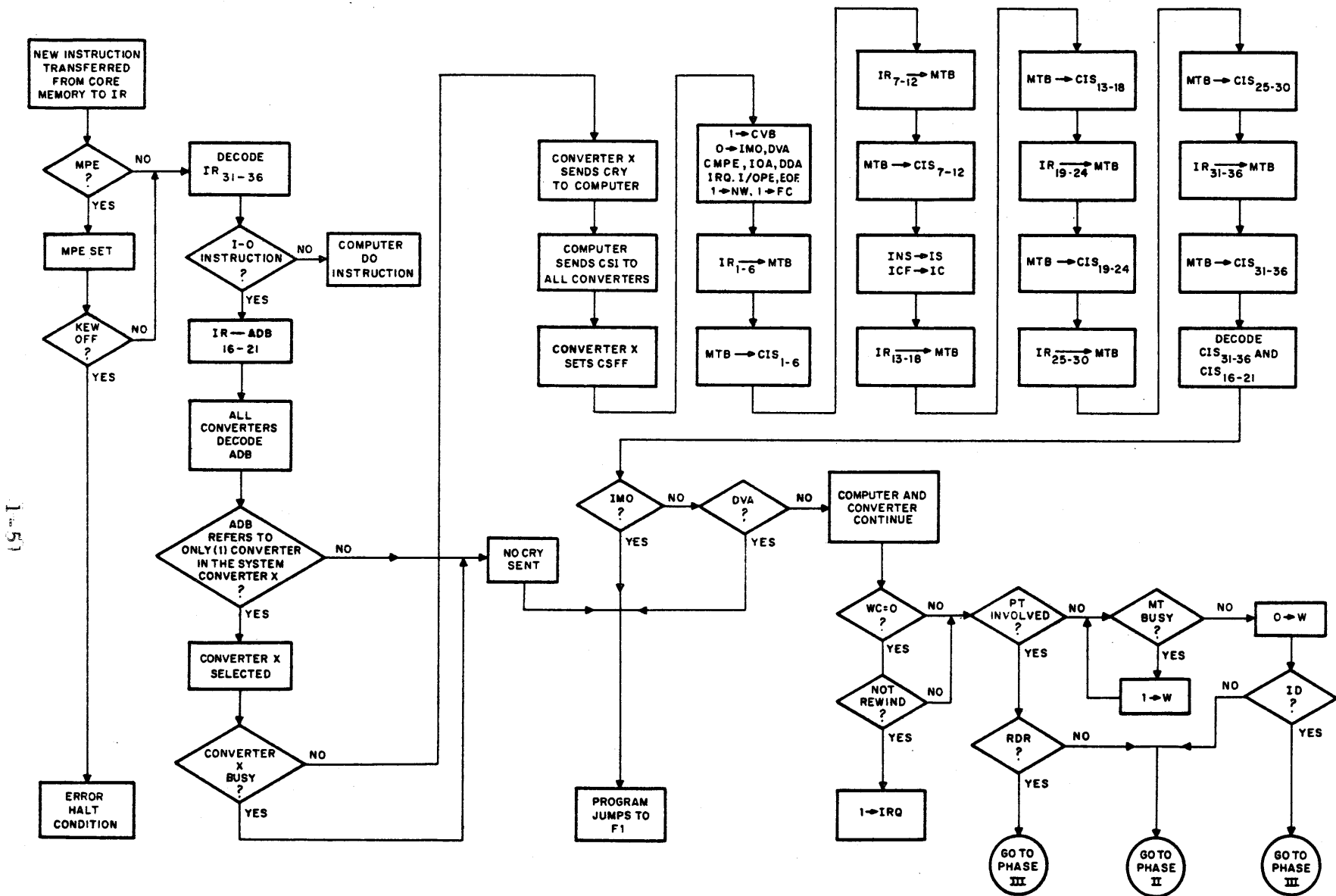


Figure V-2 Input-Output Converter, Flow Diagram Phase I

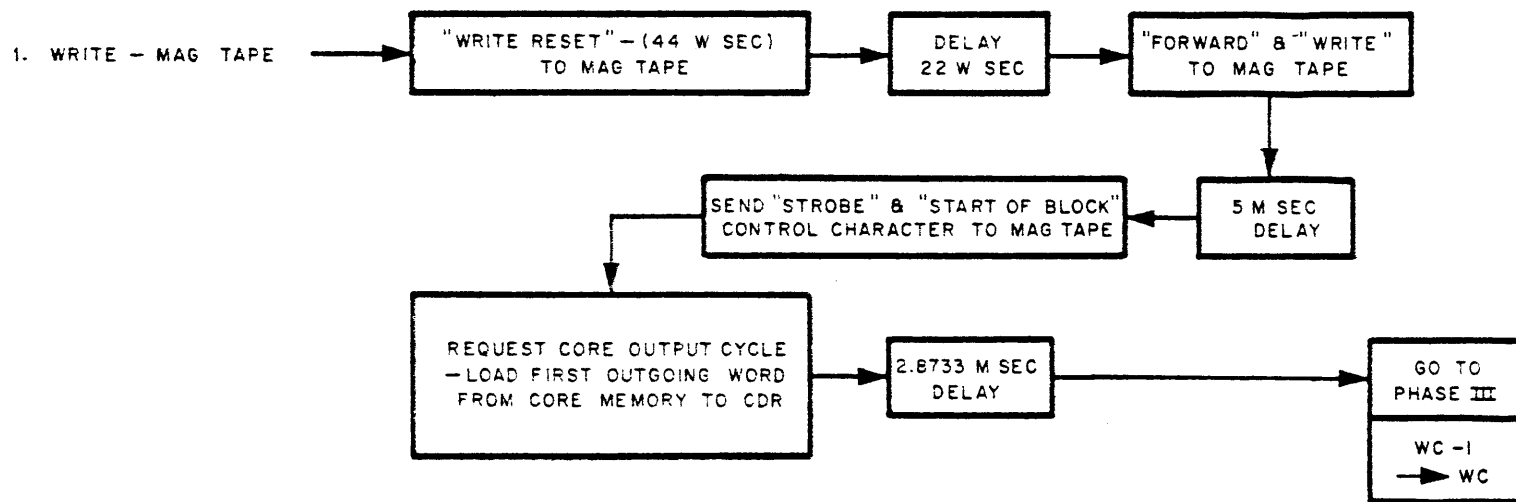


Figure V-3. Input-Output Converter, Flow Diagram, Phase II (Sheet 1 of 2)

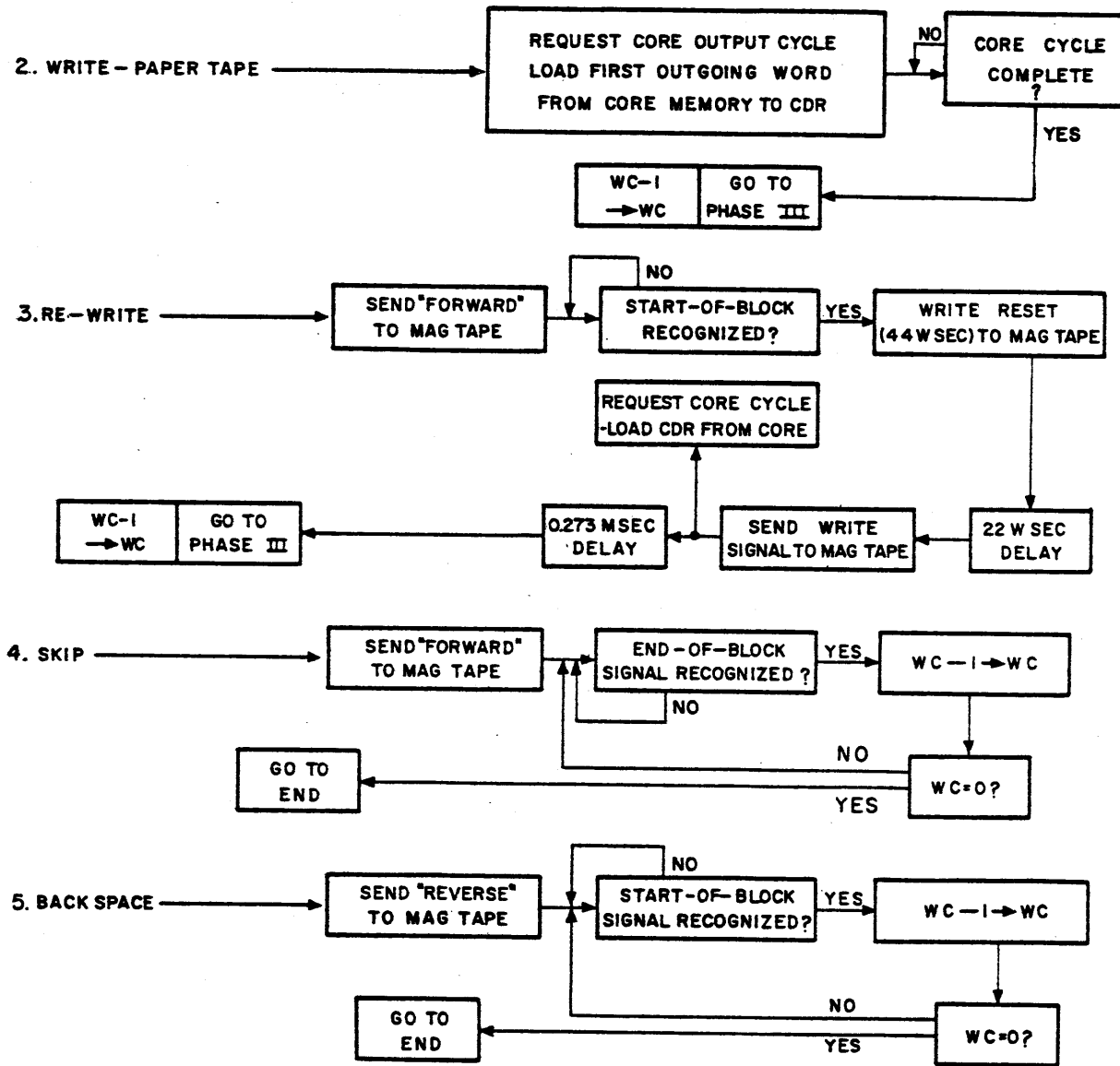


Figure V-2 Input-Output Converter, Flow Diagram, Phase II (Sheet 2 of 2)

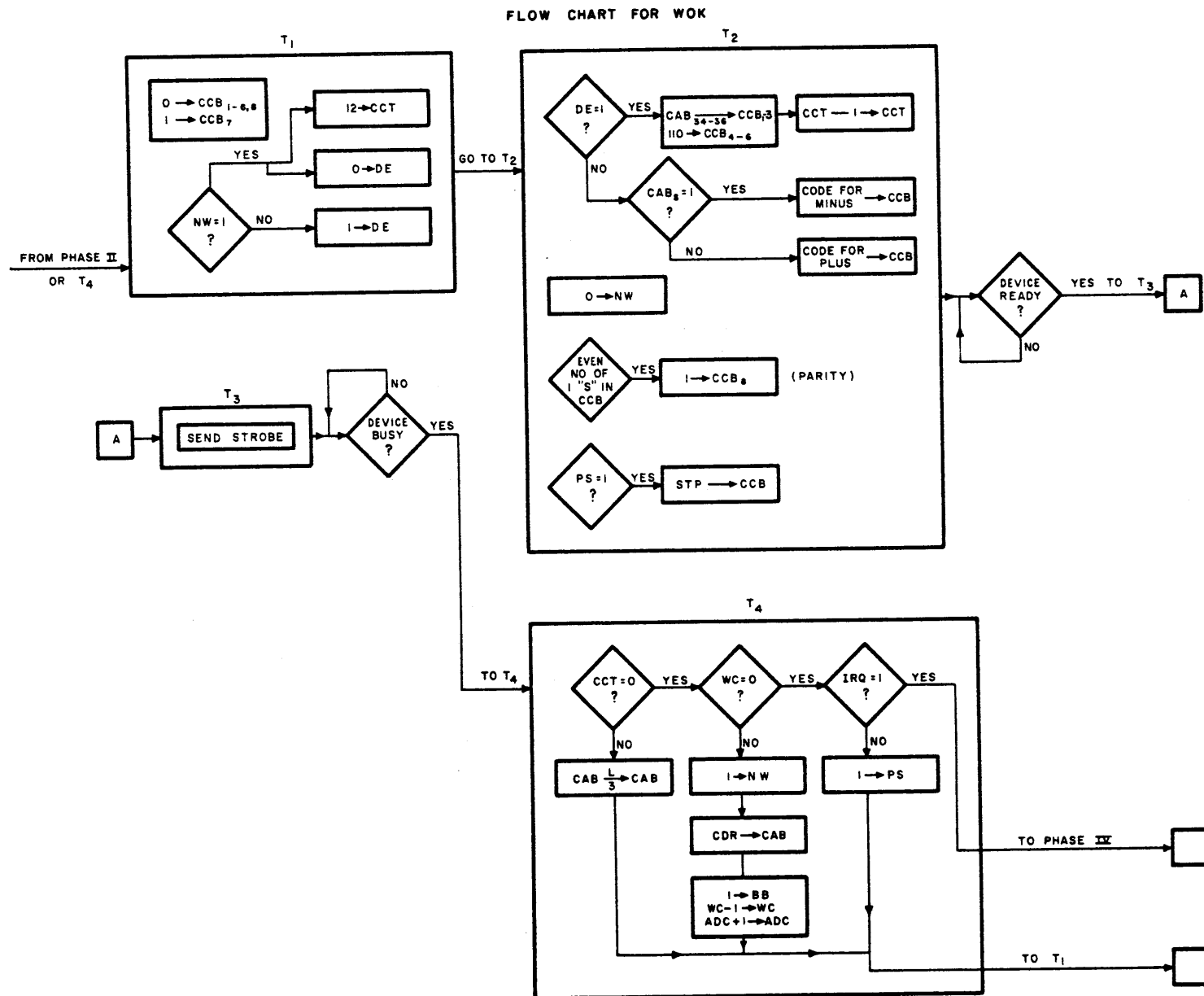


Figure Y-4. Input-Output Converter Flow Diagram, Phase III (Sheet 1 of 4)

FLOW CHART FOR ROK

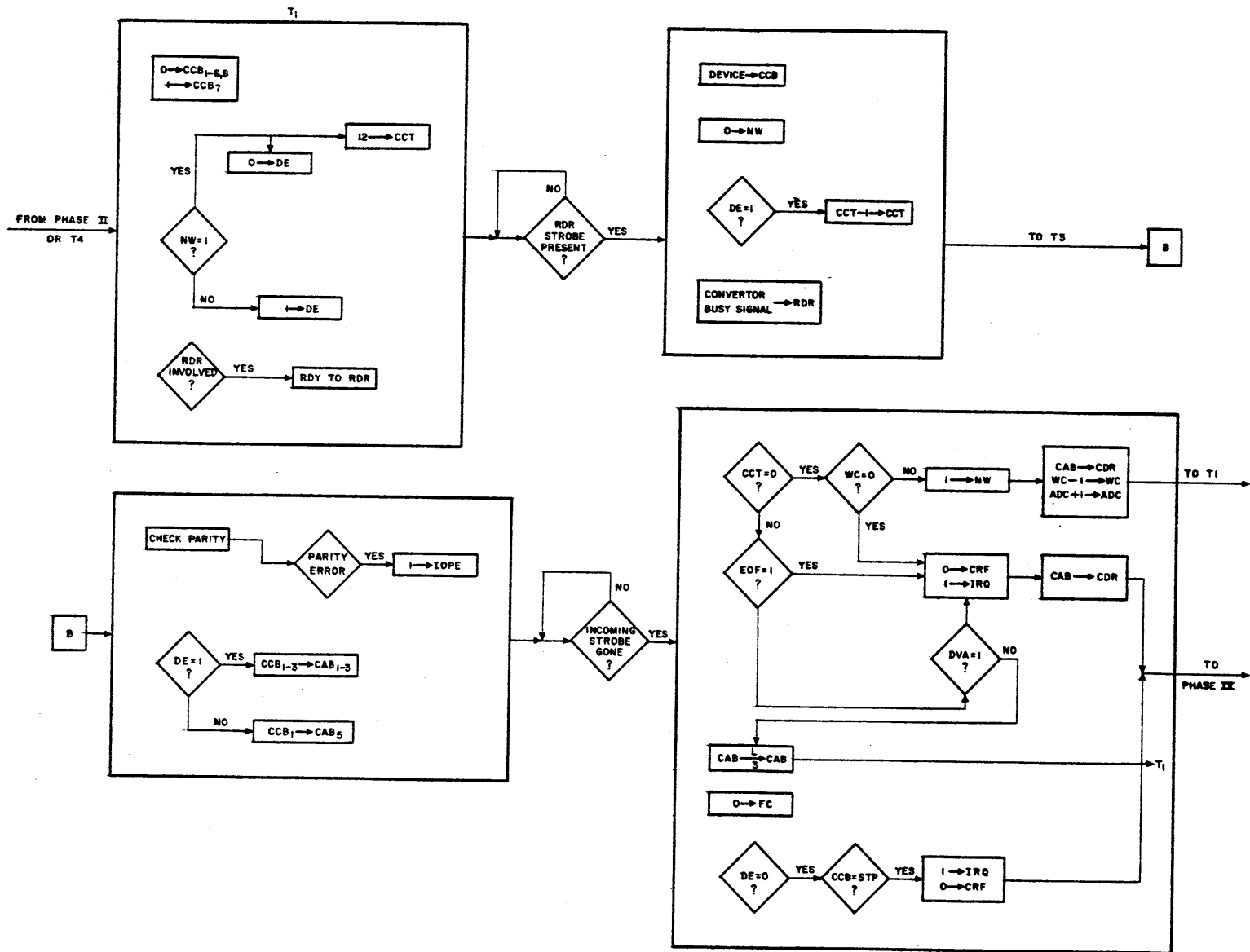


Figure V-4. Input-Output Converter Flow Diagram, Phase III (Sheet 2 of 4)

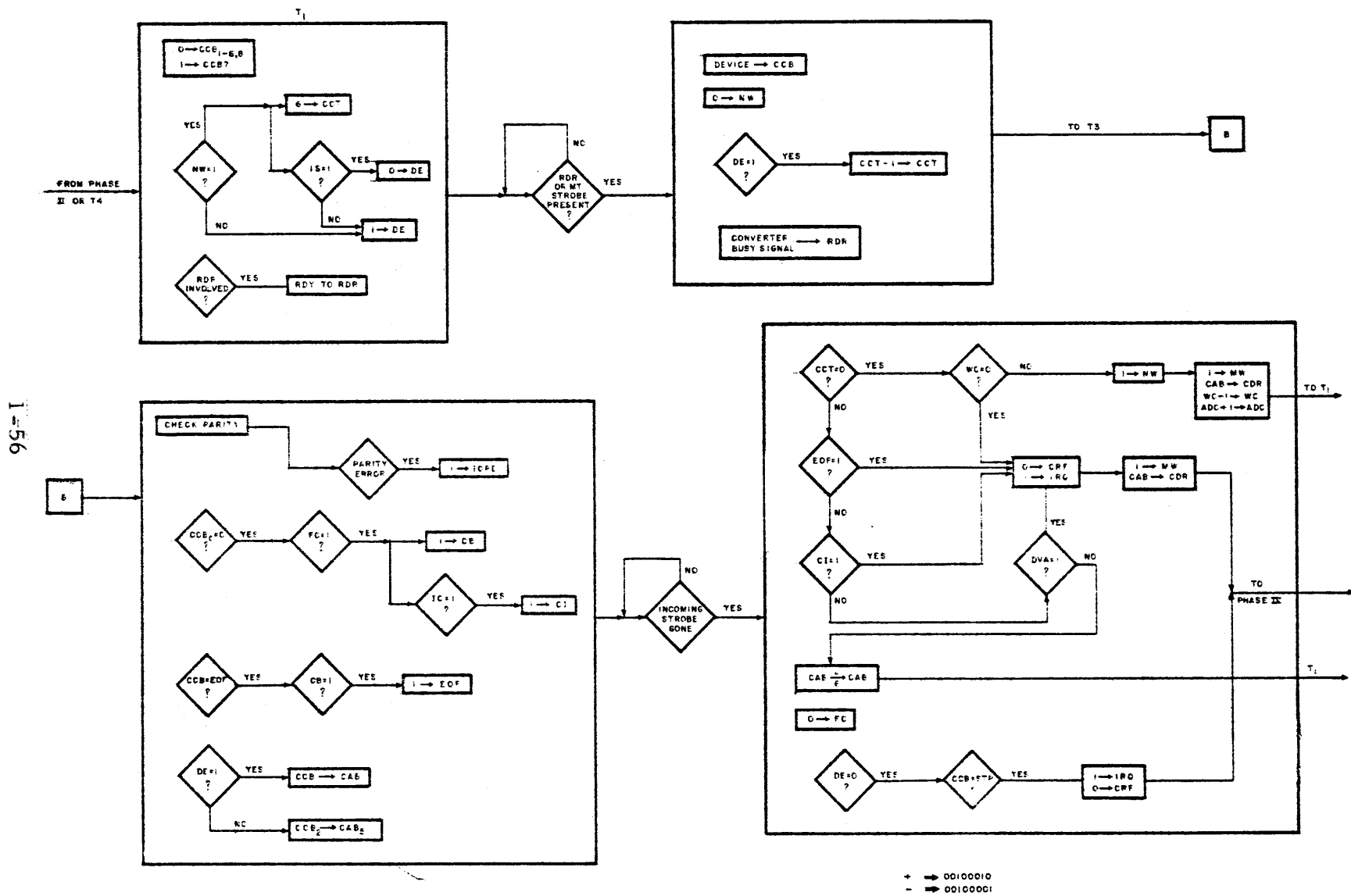
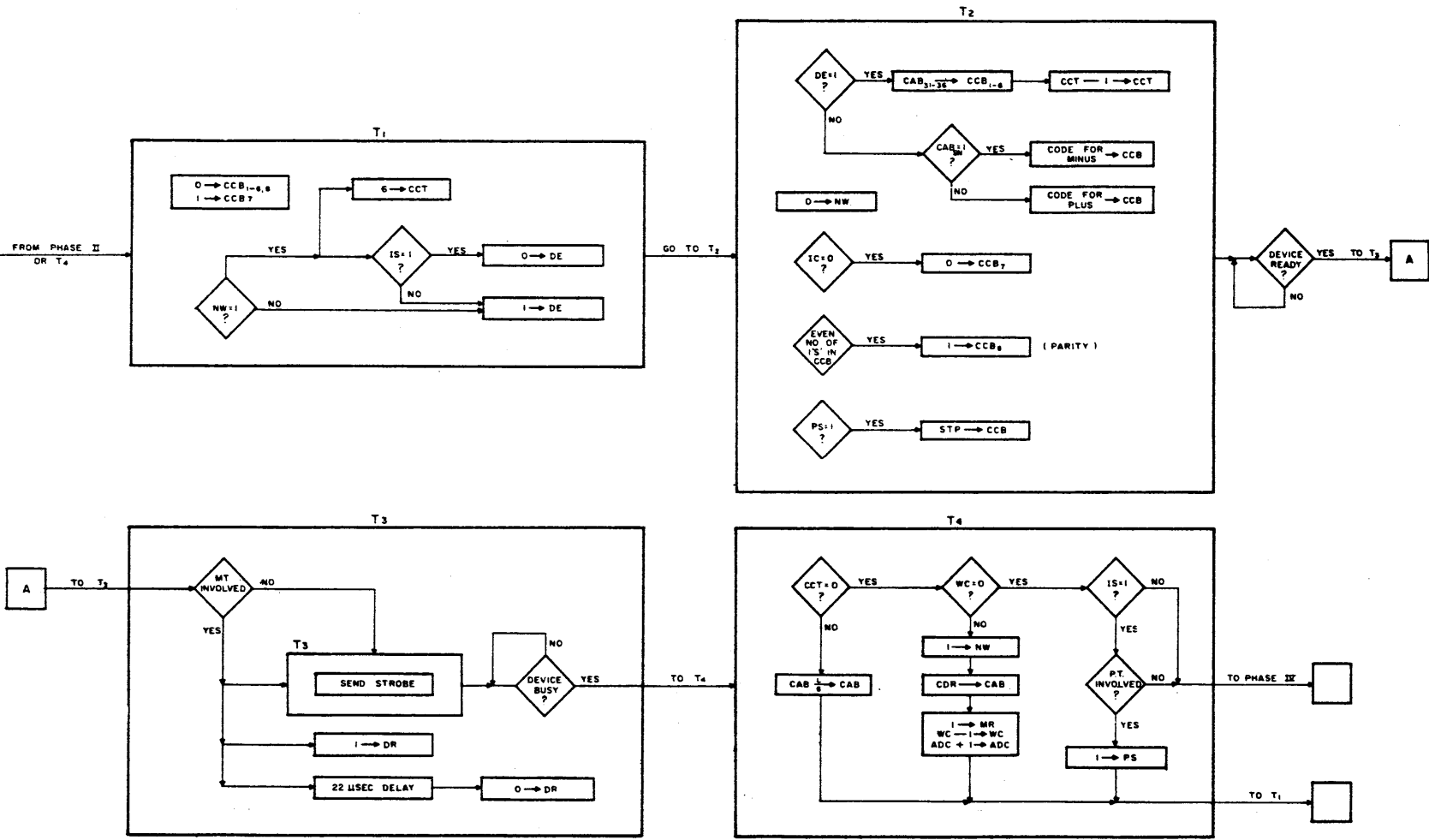


Figure Y-4 Input-Output Converter Flow Diagram, Phase III (Sheet 3 of 4)

FLOW CHART FOR WAN



I-57

Figure V-4 Input-Output Converter, Flow Diagram, Phase III (Sheet 4 of 4)

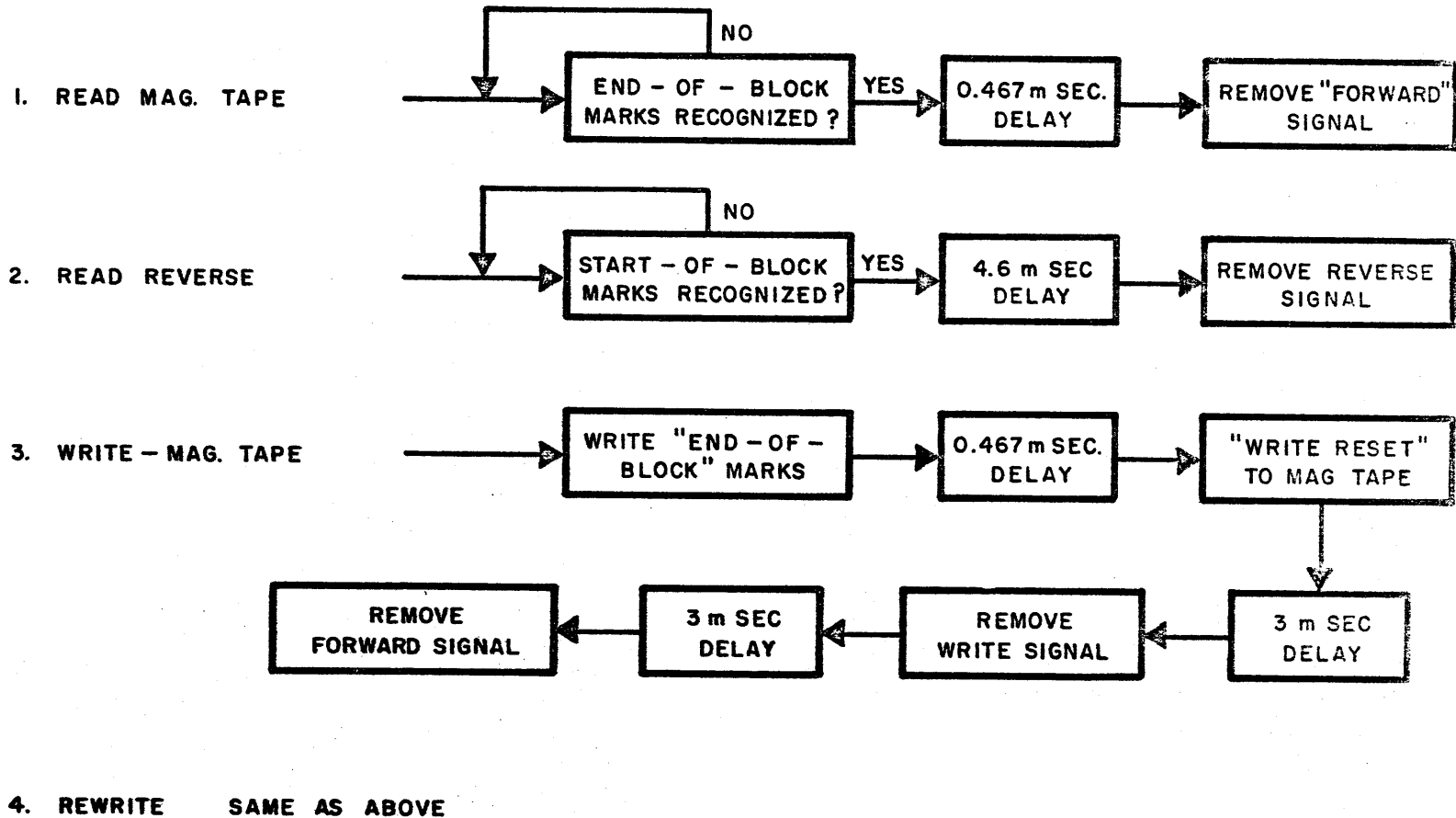


Figure V.-5. Input-Output Converter, Flow Diagram, Phase IV

2. Data Control

(1) Read

Eight-bit characters are received from the input device, checked for parity, and placed in the Converter Assembly Buffer (CAB). When CAB has received a complete word, its contents are transferred to the Converter Data Register (CDR) to await assignment to core memory. Meanwhile CAB is being loaded with the next incoming word.

(2) Write

CAB is initially loaded from core memory. When the selected device is ready it receives from CAB one eight bit character at a time. (One of the 8 bits is the parity bit). At the same time CDR seeks its next assignment from core memory. After CAB sends out its full word, the contents of CDR are transferred to CAB, and processing continues as CDR seeks its next assignment from core memory.

3. Error Control

The following flipflops in each converter unit can be sensed by the program:

(1) Device Malfunction Alarm (DVA) which indicates:

Power off

Tape broken (or absent)

Device in local mode

Device disconnected

Write order with write inhibit

Vacuum failure

Forward order when at end of tape

Reverse order when at beginning of tape

- (2) Improper Order Alarm (IMO) which indicates:

Nonexistent command

Nonexistent device address

Inconsistent command (e. g. , READ when the punch is addressed, OCTAL with magnetic tape, etc.)

- (3) IO Parity Error Alarm (IOPE) which indicates that a parity error was noted during transmission between the converter and the IO device.
- (4) Converter Memory Parity Error Alarm (CMPE) which indicates that a parity error was noted during transmission between the converter and the core memory.
- (5) Data Drop Alarm (DDA) which indicates a loss of data due to excessive multiplexing or, in the case of magnetic tape, detection of two block marks while in control mode.

4. FIELDATA Control Function Processing

If the central processor is to transmit or receive control information through an input-output device, the central processor Interpret Control Function flipflop (ICF) must be set to "ONE" with an SNS instruction. When an input-output instruction is sent to an Input-Output Converter for execution, the contents of ICF are transferred to a corresponding flipflop (IC) in the selected Input-Output Converter.

- (1) Write Instruction

All characters processed under a single write instruction (one block) appear on the output medium either as control characters or as data characters, depending upon the initial state of the IC flipflop in the Input-Output Converter.

If an Input-Output Converter receives a write alphanumeric instruction (WAN) when its IC flipflop is in the "ONE" state, outgoing words appearing in CDR are interpreted as control words. Each character of a control word is sent to the output device as a control character--that is the "c" (seventh) bit of each character is a zero.

If the IC flipflop is in the "ZERO" state, outgoing words appearing in CDR are interpreted as data words.

(2) Read Instruction

If an Input-Output Converter receives an instruction when its IC flipflop is in the ZERO state, ignore control is indicated and incoming characters are interpreted as data, regardless of their "c" bit configuration.

If the IC flipflop is in the "ONE" state, transmission procedure depends upon the "c" bit configuration of the first character in the block. If the "c" bit is one, all information is processed as data. If the "c" bit is zero, the character is recognized as a control character. This character is placed in the low order character position of CDR. The remaining bits of CDR are set to zero and the word is stored in memory. The remainder of the message is not transmitted and a program interrupt is requested. If the "c" bit of the first character is one, the entire order is processed.

5. Program Interruption by Input-Output Converter

When an Input-Output Converter has accepted an order from the central processor, both units resume their operations concurrently. After the converter has entered the end-of-transmission state, it sets the Interrupt Request flipflop (IRQ) to "ONE". At this point, the operation of central processor is interrupted if the Disable Program Interrupt flipflop (DPI) is zero. An interrupt can occur only at the beginning of the instruction access phase. When an interrupt request is granted, DPI is automatically set to "one", the address of the instruction about to be performed is automatically stored in the α portion of the B register, and control is automatically transferred to a special memory location. At this point, normal computer operation is resumed.

6. Communication with Central Processor

a. Computer IR to Input-Output Converter CIS

(1) During the instruction access phase, the instruction is transferred to the IR from memory. The IR is then decoded and, if an IO instruction is present, the device address in IR is placed on the Address Distribution Bus (ADB). After the ADB is decoded, the selected converter signals the central computer if it is ready to accept an instruction. If the "Converter Ready" signal is not present, the central computer performs an interrupt jump.

(2) During the operand access phase (OAC) the computer transmits a SEND signal (CSI) to all converter units. The signal remains active for 12 microseconds. The I/O instruction then is transferred from the IR to the selected CIS through the MTB. For the remainder of operand-access time the selected converter determines whether the instruction is consistent, the selected device exists, and whether the selected device is serviceable. The six j-bits of the instruction designate the assigned device but do not designate the converter as such. Gating in the converter decodes the device address.

(3) If a malfunction or improper order is detected during the first OAC of the I/O instruction, the converter sends a "Converter Initial Malfunction" signal (CIM) to the computer. The computer then performs an interrupt jump.

b. Input-Output Converter CIS to Computer

The following instructions cause the contents of a specified converter instruction register (CIS¹) to be transferred to the computer via the Major Transfer Bus:

LGM	logical multiply
LGA	logical add
CLA	clear add
ADD	arithmetic add
ADM	add magnitude
CLS	clear subtract
SUB	arithmetic subtract
MLY	arithmetic multiply
DVD	arithmetic divide
DVL	arithmetic divide long
LOD	load

c. Addressable Flipflops

(1) Any of the addressable flipflops can be referred to by the following instructions:

SNR (If $FF\beta = 1$, Jump and clear FF)
SNS (If $FF\beta = 0$, Jump and set FF to 1)
SEN (If $FF\beta = 1$, Jump, and do not change the state of FF)

(2) Each converter unit which contains an addressed flipflop signals the computer to jump if the flipflop addressed is in the corresponding state. The state of the flipflop is then changed as required. If this signal is not received, the computer continues to the next instruction.

d. Converter Data Register (CDR) to Memory (MO)

After a full word has been received by the converter assembly buffer (CAB) from an input device, the word is transferred to the converter data register (CDR) to await core memory assignment.

Bits 13-15 of the address counter (ADC) are then gated to the memory selection bus (MSB) to select the desired memory unit.

The core location address then loaded into the memory address register (MA) through MTB. The core readout cycle is then initiated and the memory data register (MO) is loaded from MTB.

The memory cycle then proceeds, and the next unit in priority receives a core memory assignment.

e. Memory (MO) to Converter Data Register (CDR)

When the converter assembly buffer (CAB) requires a new word for transmission to an output device the contents of CDR are transferred to CAB. After a core memory assignment is requested and received, the proper core memory unit is selected. The core address is then sent via MTB to the selected memory unit and core readout is initiated. The information is placed on MTB and loaded into CDR to complete the operation.

C. ADDRESSABLE FLIPFLOPS AND REGISTERS

Each Input-Output Converter has the addressable flipflops listed below:

ADDRESSABLE FLIPFLOPS

Input-Output Alarm	IO	
Converter Busy	CVB	
Improper Order	IMO	
Malfunction	DVA	
Mag Tape ₁ Busy	DB ₁	The state of these flipflops can be sensed but not altered by the program.
Mag Tape ₂ Busy	DB ₂	
Mag Tape ₃ Busy	DB ₃	
Mag Tape ₄ Busy	DB ₄	

Control Indication	CI
End of File	EOF
Input Output Parity Error	IOPE
End of Tape	EOT
Converter to Memory	
Parity Error	CMPE
Beginning of Tape	BOT
Interrupt Request	IRQ
Data Drop Alarm	DDA

D. MAGNETIC TAPE TRANSPORTS

The FIELDATA magnetic tape transports connected to the Input-Output Converter are described in U. S. Army documents SCL-1882A and SCL-1886.

VI. COMMUNICATIONS CONVERTER

The Communications Converter transfers information between the BASICPAC system and real-time communications links. Information is transferred only when the communicator is prepared to transmit or receive and is not program controlled. A Communications Converter can control up to seven input and seven output channels. Any number of input channels can operate at one time. Output channel transmission must be programmed. Access through each channel is on the basis of one character at a time. Two types of characters may be transmitted and received, FIELDATA Control Characters (contain a zero bit in position 6), and Alphanumeric FIELDATA Characters (contain a one bit in position 6). There are sixty-four possible control characters and sixty-four possible alphanumeric characters. Parity is odd.

Control Character

P C I I D D D D

7 6 5 4 3 2 1 0

x	0	x	x	x	x	x	x
---	---	---	---	---	---	---	---

Alphanumeric Character

P C I I D D D D

7 6 5 4 3 2 1 0

x	1	x	x	x	x	x	x
---	---	---	---	---	---	---	---

There are two types of interrupt channels: limited interrupt and regular interrupt channels. A regular interrupt channel requests interrupt when either one data word or one control character has been transmitted. A limited interrupt channel requests interrupt when a specified number (up to 512) of data words or one control character has been transmitted. Limited interrupt applies only to input.

A. INPUT

The seven input channels are numbered 0 through 6. Two consecutively numbered memory locations are associated with each input channel. The even numbered location is used to store incoming data words; the odd numbered location is used to store an incoming control character. Memory location 20_g is reserved for data from channel 0 and 21_g for control characters. Memory locations 22_g and 23_g are reserved for channel 1, etc.

Multi-channel machines include a seven-position input sequencer to keep track of the channels which are prepared to transmit a character into the central processor. The sequencer cycles repeatedly through each position. When the sequencer determines that a character is waiting to be sent on channel i ($i = 0$ to 6), it sends the channel number to the converter address lines (KAA). KAA automatically translates the channel number into the appropriate memory location (odd for control and even for data). If the character to be transmitted over channel i is a control character, it is stored in bits 1-6 of the odd-numbered memory location associated with channel i and the high order bits are set to zero. If the character is a data character, it is stored in the even-numbered memory location associated with channel i .

An input character counter (KIC^i) is used to specify the character position in which an incoming data character is to be stored. This counter is automatically preset by the converter to the number of characters expected (6 or 7 depending on the state of the ISN flipflop) and is counted down as each data character is received by the central processor. When KIC^i has been counted down to zero or when a control character is sensed on the i 'th channel, the program of the central processor is interrupted. At the same time, the converter resets the Converter Input Free flipflop (KIU^i) to indicate that no more information can be transmitted over the i 'th input channel.

The Disable Program Interrupt flipflop is set to prohibit interruption of the current interrupt, and the Communication Input Interrupt Request flipflop (KAI) is set to indicate the source of the interrupt.

An F2 interrupt occurs at this point, and the program jumps to an interrupt subroutine which determines the cause of interrupt and processes the information accordingly. Memory location 00004 is assigned to store information which defines the cause of communications converter input interruption. Bit 37 indicates the presence of parity error. Bits 36 to 34 contain the value of the KIC counter at the time of interrupt in modified Grey code, as shown below. Bits 12 to 1 give the address of the memory location in which the interrupting information received by the central processor is stored.

Since a control character interrupt may occur at any time, bits 36-34 of memory location 00004 will aid the programmer in determining the form of his input information. If bits 36-34 equal 0, a data word has interrupted and there are six or seven data characters, depending on the state of ISN when the channel was permitted to receive. There are no control characters to be processed. Parity error is indicated by bit 37 of memory location 4. If bits 36 to 34 equal 1 to 5, in Grey code, and 6 characters per word had been requested (ISN was 0 when the channel was activated), then from 5 to 1 data characters were accepted before a control character interrupted. Unused character positions of the data word are zero. Parity error is indicated by bit 37 of memory location 4 for data and by bit 37 of the properly numbered memory location for control.

If bits 36 to 34 equal 7 in Grey code and ISN had been set, only 1 control character was received.

If bits 36 to 34 equal 1 to 6 in Grey code and ISN had been set, from 6 to 1 data characters were accepted before a control character caused interrupt.

Grey code (binary)	Binary equivalent	Octal equivalent
000	000	0
001	001	1
011	010	2
010	011	3
110	100	4
111	101	5
101	110	6
100	111	7

Grey Code

Ex: A Grey Code binary configuration of 101 would correspond to the octal number 6.

Only one input channel, channel 0, can be used as a limited interrupt channel. The limited interrupt channel operates somewhat differently from the other input channels, in that up to 512 data words can be received and stored in sequence before a program interrupt is requested. An addressable 15-bit register, KIW, counts the number of incoming data words and specifies the memory location in which each word is to be stored. Prior to transmission, KIW must be set to the address of the first storage location to be used. As each input word is stored, KIW is incremented by one until either a control character is received or until the nine least significant bits of KIW equal 777_8 (512_{10}). At this point a program interrupt is requested.

Since memory location 20_8 is not used to store the incoming data, it is suggested that the programmer store the address placed in KIW in this location also. Upon interrupt, location 20_8 supplies the address of the first data word, and KIW the address of the last data word transmitted. The quantity of information received can then be easily determined.

B. OUTPUT

The central processor sends information to the communications converter under the control of the program. ICF and ISN may be set as desired. The state of ICF determines whether the information to be transmitted from the processor will be treated as data or as control.

The program places the word to be transmitted in the output word buffer (KOU^i) associated with the channel to be used. The converter then transfers the data word from KOU^i to the memory location (10 to 17_8) associated with the desired channel.

In multichannel machines, a seven-position output sequencer cycles repeatedly to keep track of the channel currently transmitting information. An output character counter (KOC^i) is automatically preset to six, seven or one, depending upon the mode of transmission. When a given channel is ready to accept a character, the character is taken from the character position specified by the contents of KOC^i . When a character is transmitted, KOC is counted down and the process is repeated until KOC reaches zero. When transmission is complete, the central processor is interrupted to indicate that one word or one control character has been transmitted on the i 'th channel. The converter also sets the Converter Output Interrupt Request flipflop (KEI) and the DPI flipflop. The converter stores the following information in memory location 3:

Bit 37 = parity error

Bit 1-12 = address (10 to 17_8) of the memory location from which data was transmitted.

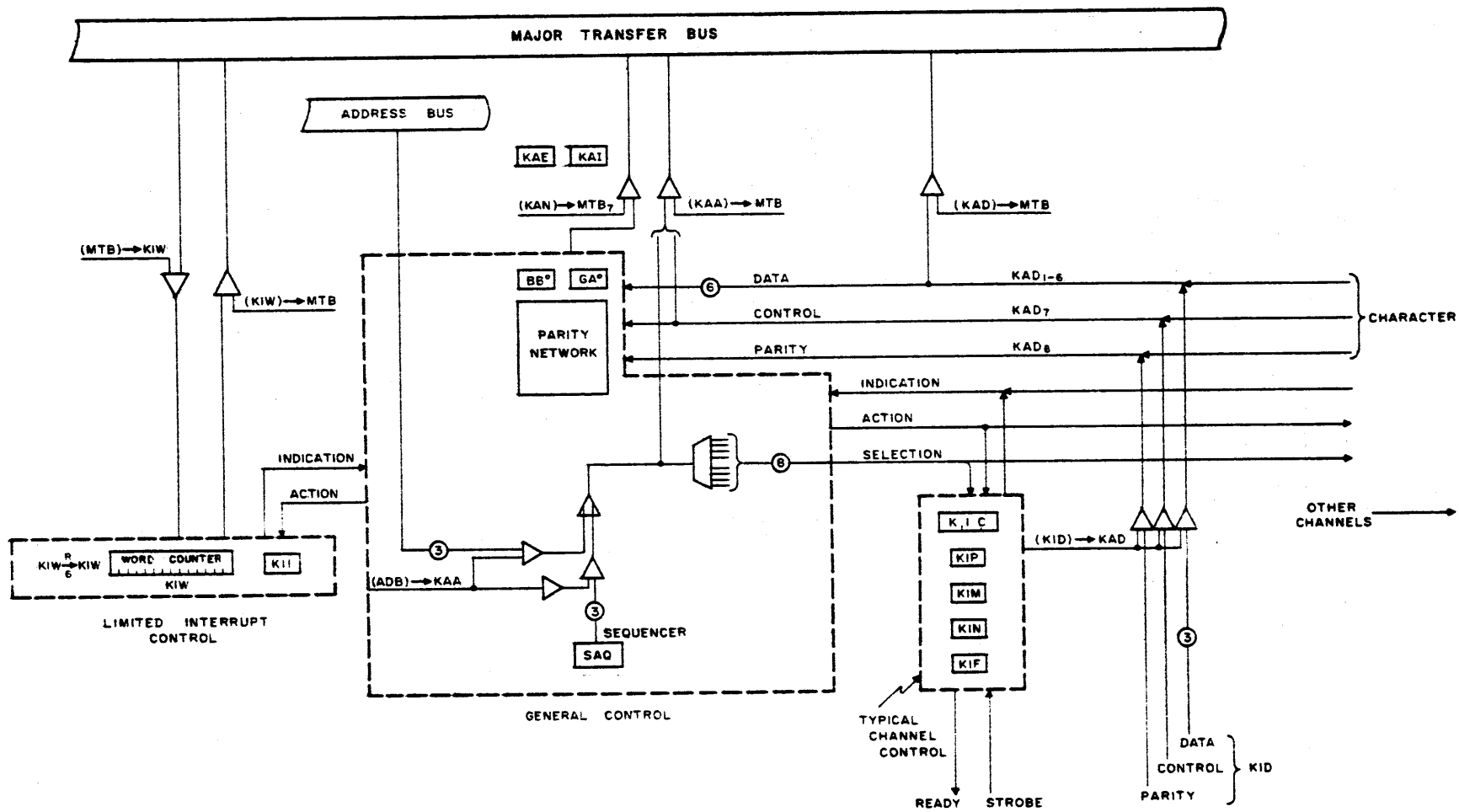


Figure VI-1 Communications Converter, Input Section, Block Diagram

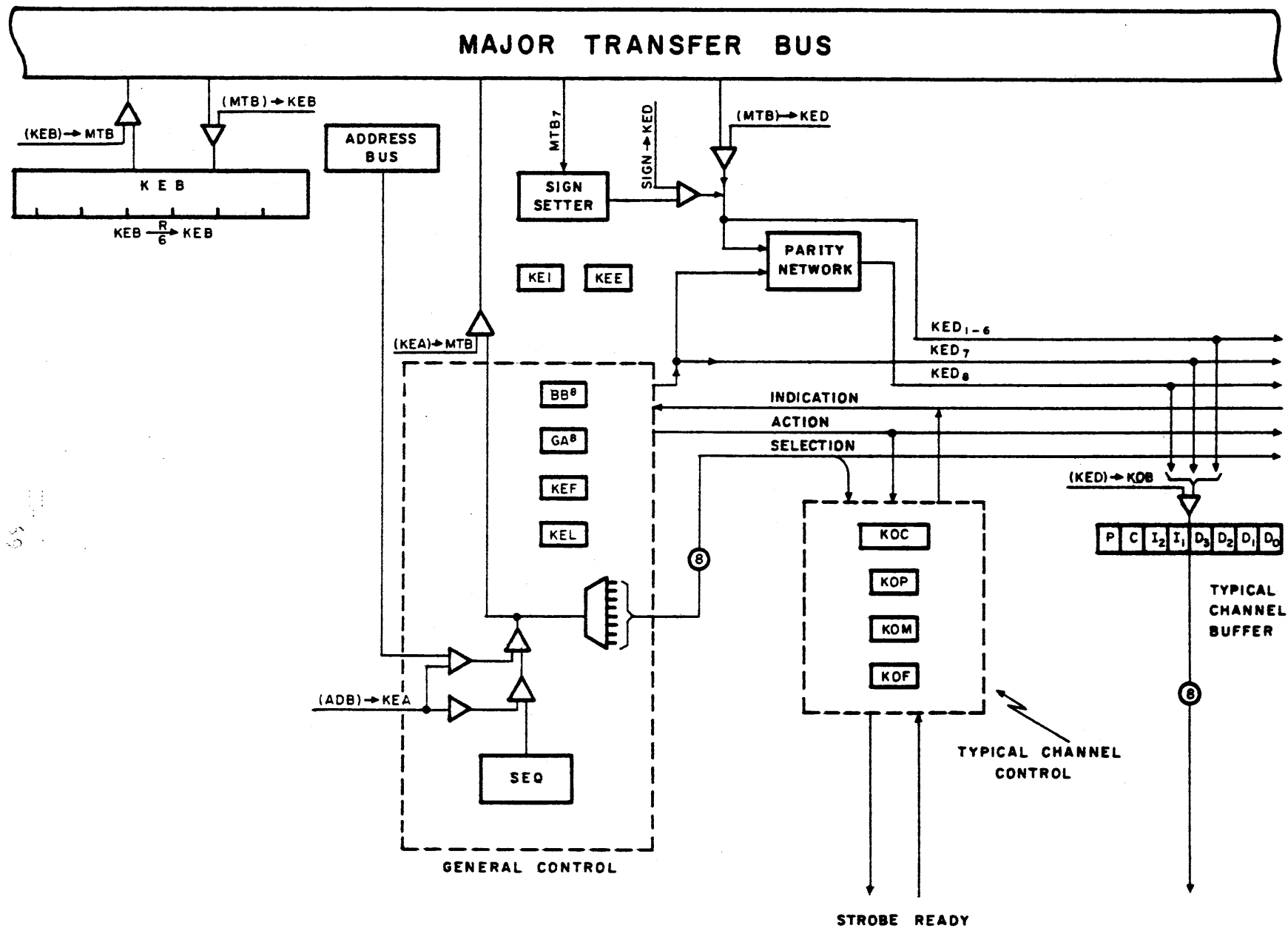
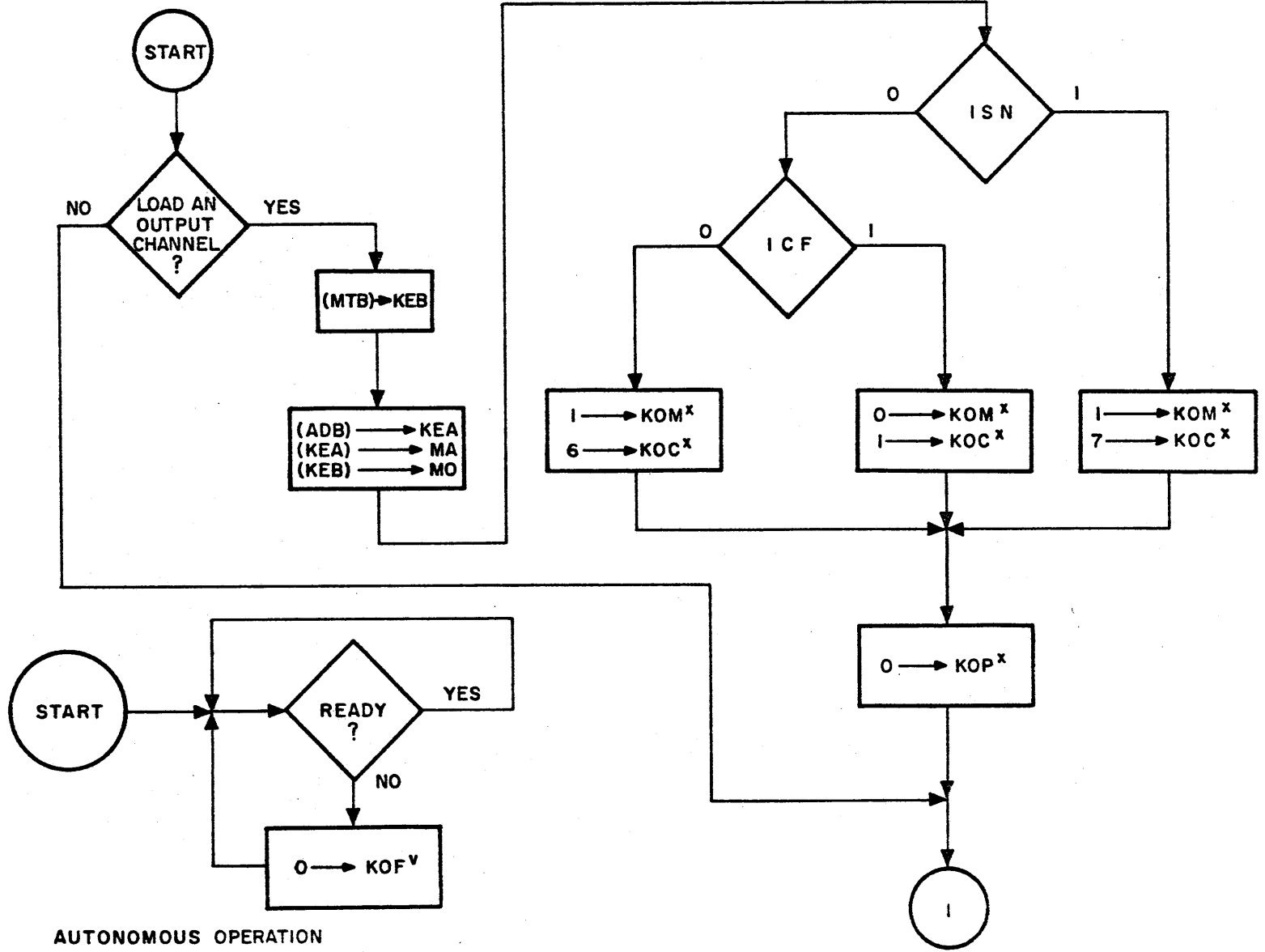


Figure VI-2 Communication Converter, Output Section Block Diagram

07-70



AUTONOMOUS OPERATION
OF TYPICAL CHANNEL UNIT

Figure VI-3. Communications Converter Output Flow Chart (Sheet 1 of 2)

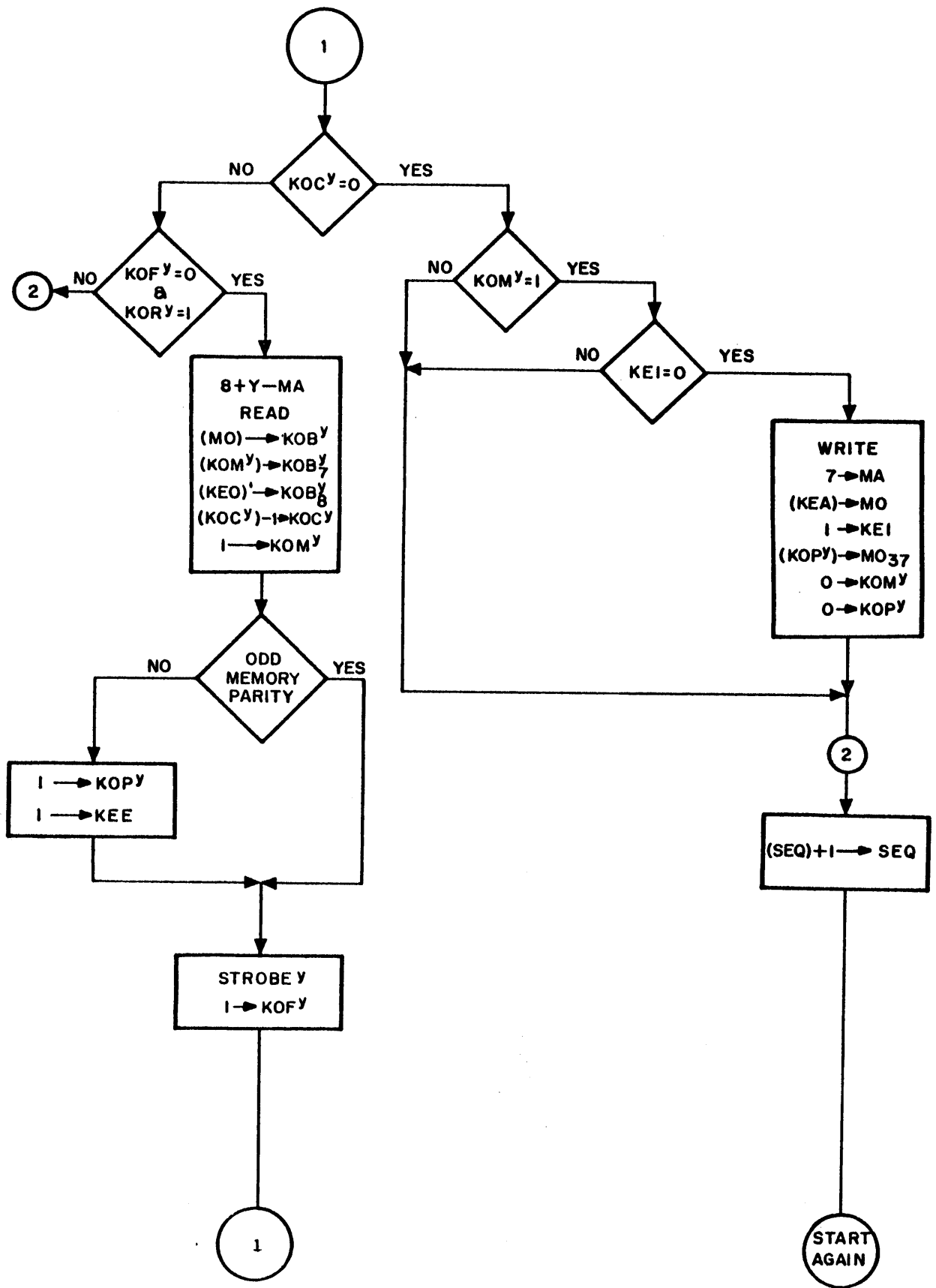


Figure VI-3 Communications Converter Output Flow Chart 1-71
(Sheet 2 of 2)

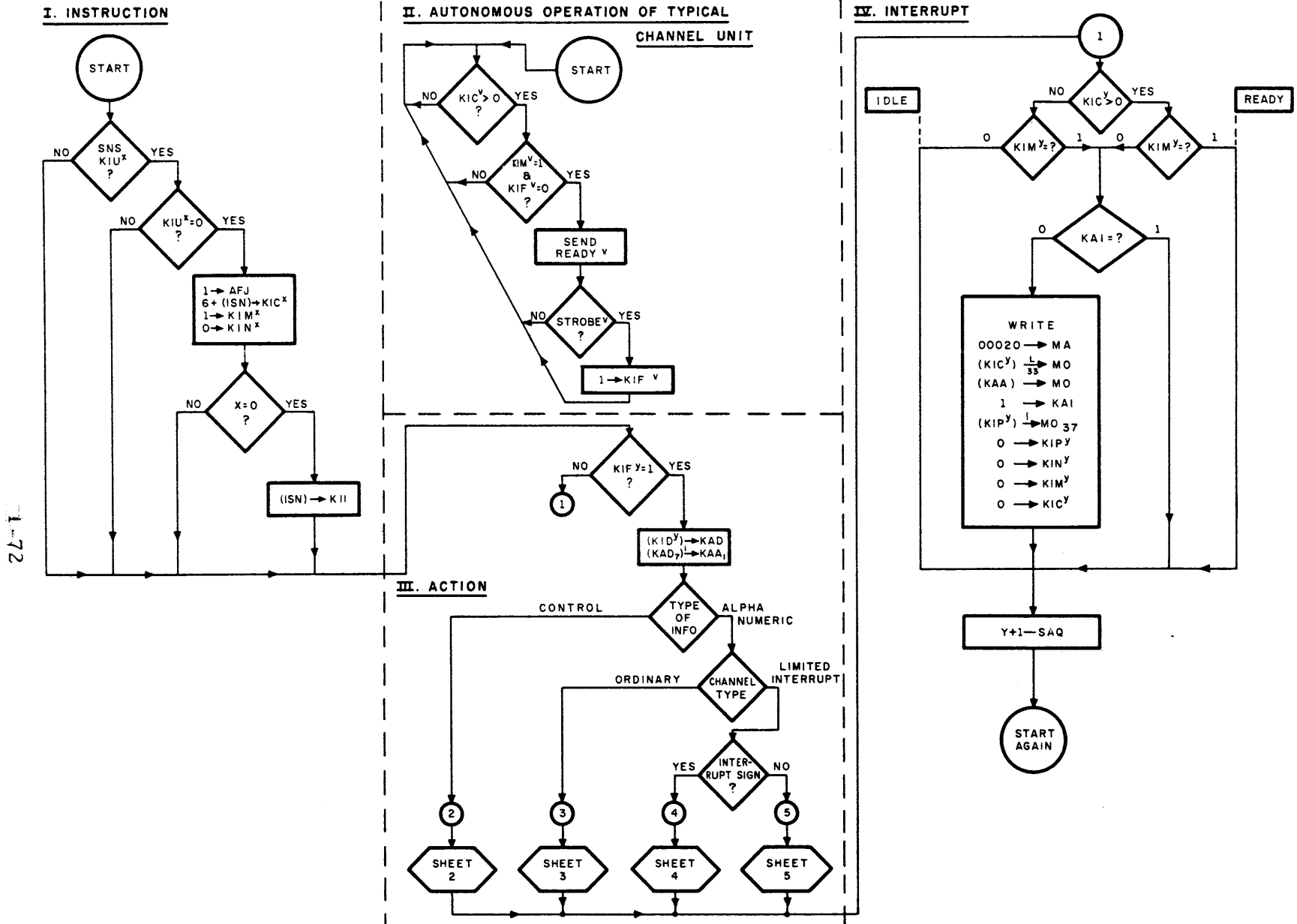


Figure VI-4 Communication Converter Input Flow Chart (Sheet 1 of 5)

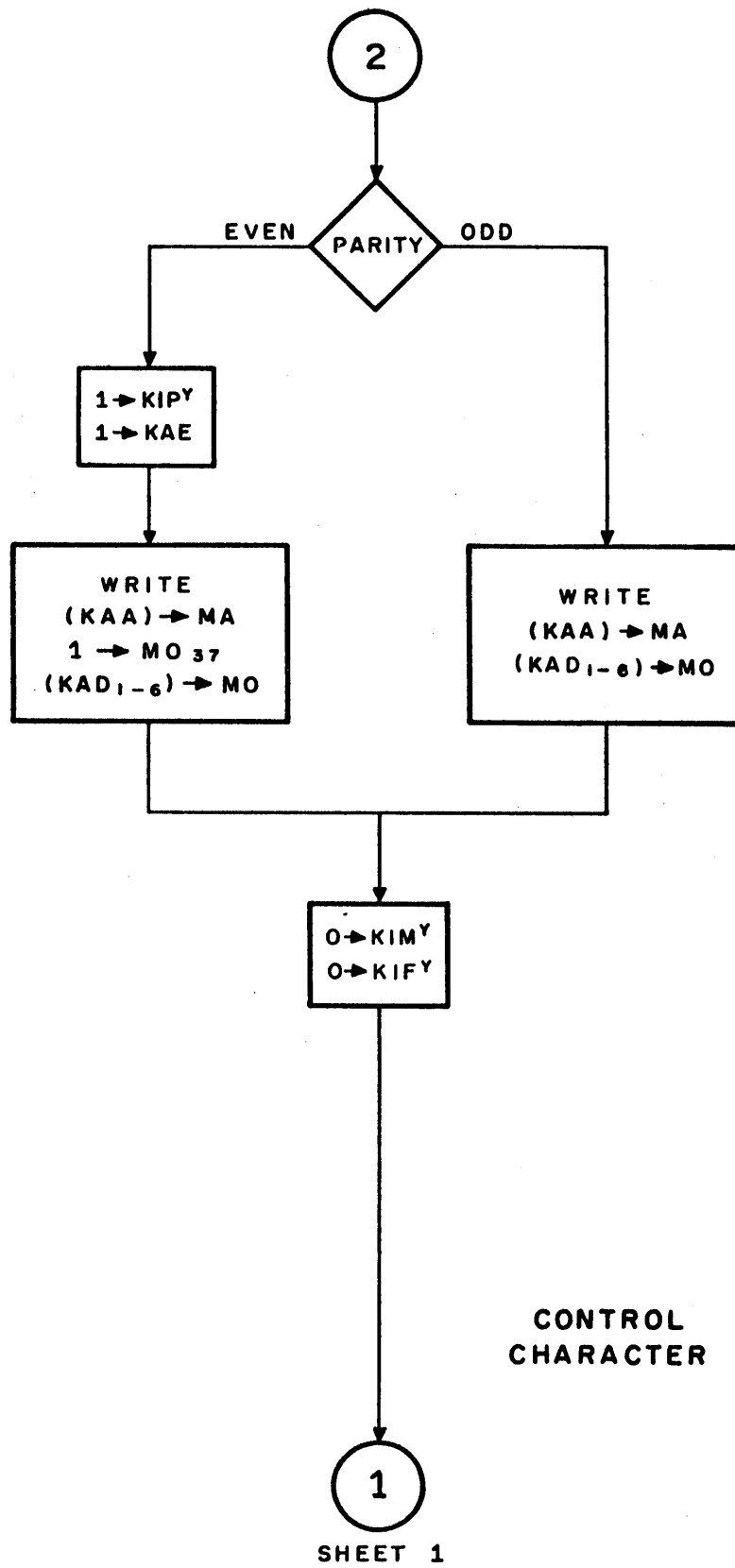
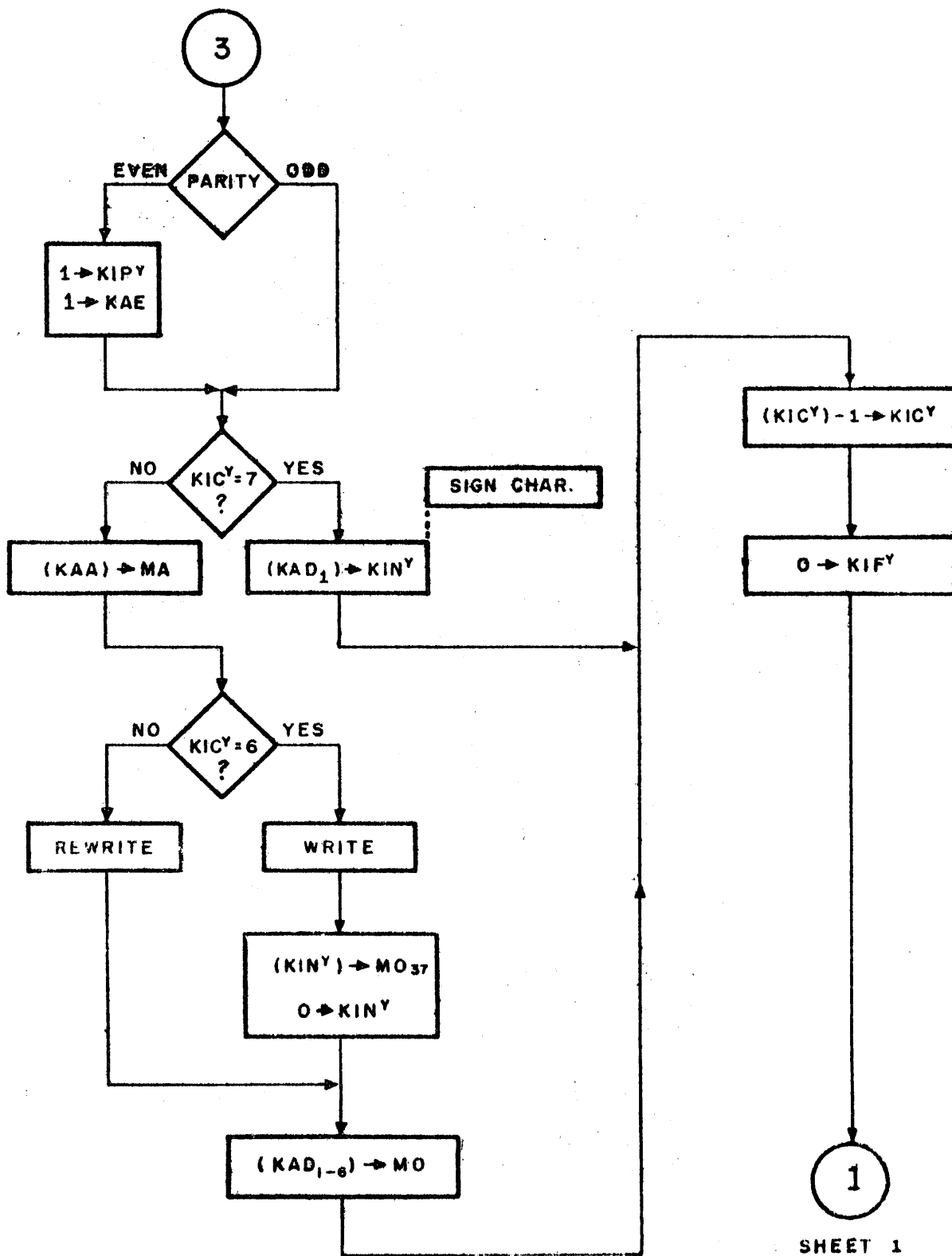


Figure VI-4 Communications Converter Input Flow Chart (Sheet 2 of 5)



**ALPHA-NUMERIC:
ORDINARY CHANNELS**

Figure VI-4 Communications Converter, Input Flow Chart (Sheet 3 of 5) 74

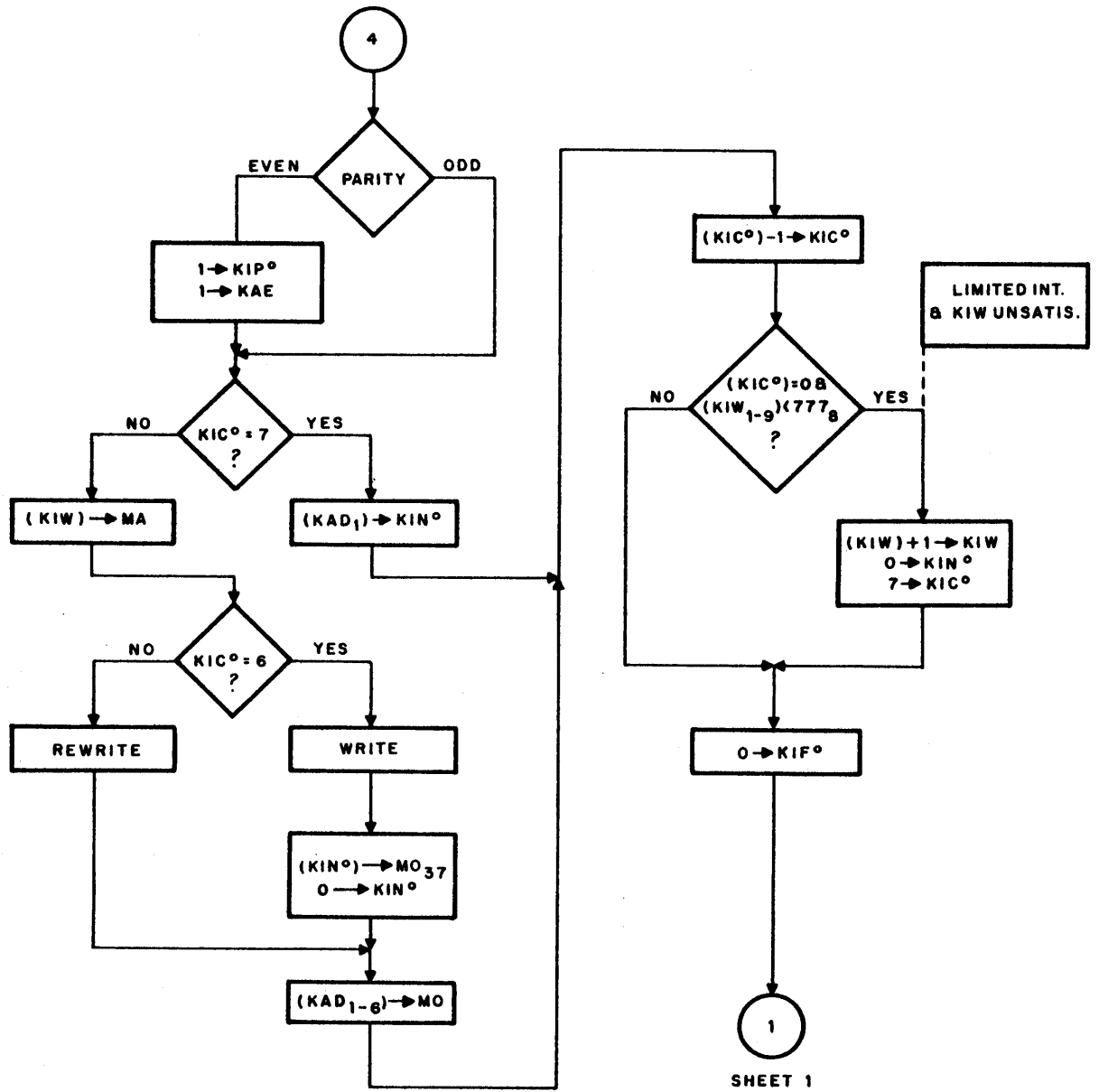


Figure VI-4 Communications Converter, Input Flow Chart (Sheet 4 of 5)

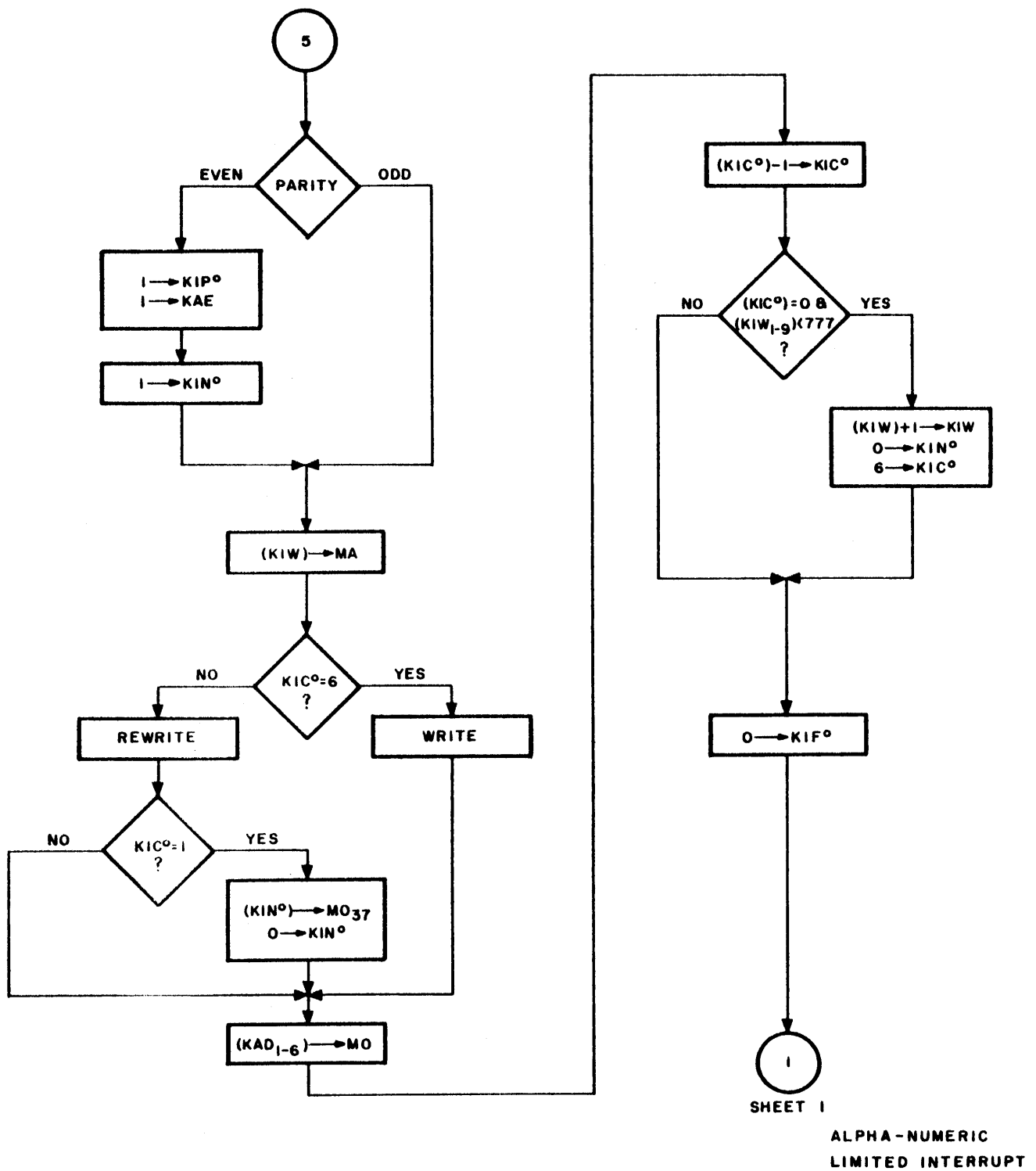


Figure VI-4 Communications Converter, Input Flow Chart (Sheet 5 of 5)

SECTION TWO

BASICPAC PROGRAMMING

SECTION TWO

VII. INTRODUCTION

Section Two contains a detailed discussion of the logical steps followed in preparing programs for BASICPAC. This portion of the manual is intended to be self-instructive, with illustrative examples demonstrating the application of newly introduced principles.

The manual is organized in such a manner that it proceeds from less difficult to more difficult. Careful study will provide the basic knowledge required to prepare programs for BASICPAC.

VIII. NUMBER SYSTEMS

A number system is essentially a means of counting. It consists of a series of unique symbols ("digits"), each of which represents a specific quantity. Number systems have been constructed on various "bases" (number of separate symbols used), and many of these systems have found practical applications. Most people use the decimal number system, containing ten symbols (0 through 9). Most electronic digital computers use the binary number system, containing two symbols (0 and 1). Communication between human and computer is frequently in the octal number system, with eight symbols (0 through 7). The programmer must be able to use all three systems with facility.

Primitive number systems (such as the Roman) assigned symbols according to an arbitrary pattern, with the result that a number can be represented fairly easily in Roman numerals, but cannot be manipulated. (For example, try to multiply CIV by XII.)

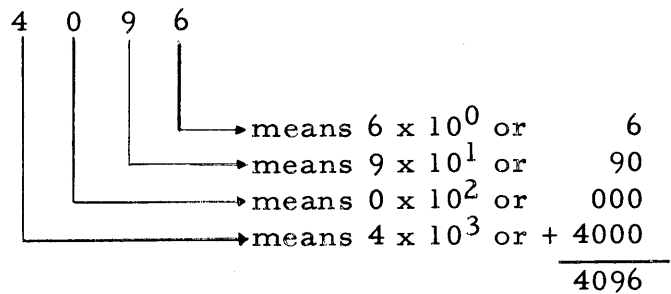
Manipulation of numbers, the basis of mathematics, is feasible only with a positional number system, in which each symbol changes its value according to where it is placed in the number. The decimal digit 1, for instance, represents quite different quantities in the numbers 1, 10, and 100. Note that the same symbol can be used for these different quantities only if there is a symbol for "nothing", an empty position.

A. DECIMAL NUMBER SYSTEM

We are accustomed to a system in which we can count ten things (including "nothing"), then must start over again with the same symbols in a new column:

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	= 1 cycle of ten, + 0
11	= 1 cycle of ten, + 1
.	
.	
20	= 2 cycles of ten, + 0

In this system, when we reach the symbol for 9 we have used all the symbols available. We have gone through the complete cycle once, and we note this fact by a 1 in the next column: 10. When we pass 19 we count two cycles: 20. An adding machine (which is a decimal-type digital computer) has a series of wheels, each with ten teeth. As each wheel completes one revolution it nudges the next wheel one position higher, so that the second wheel counts the number of times the first wheel has completely revolved, the third wheel counts the revolutions of the second wheel, and so on. These wheels correspond to the columns of written numbers. The decimal number 4096 therefore means six units plus nine cycles of ten plus no hundreds of cycles of ten plus four thousands of cycles of ten:



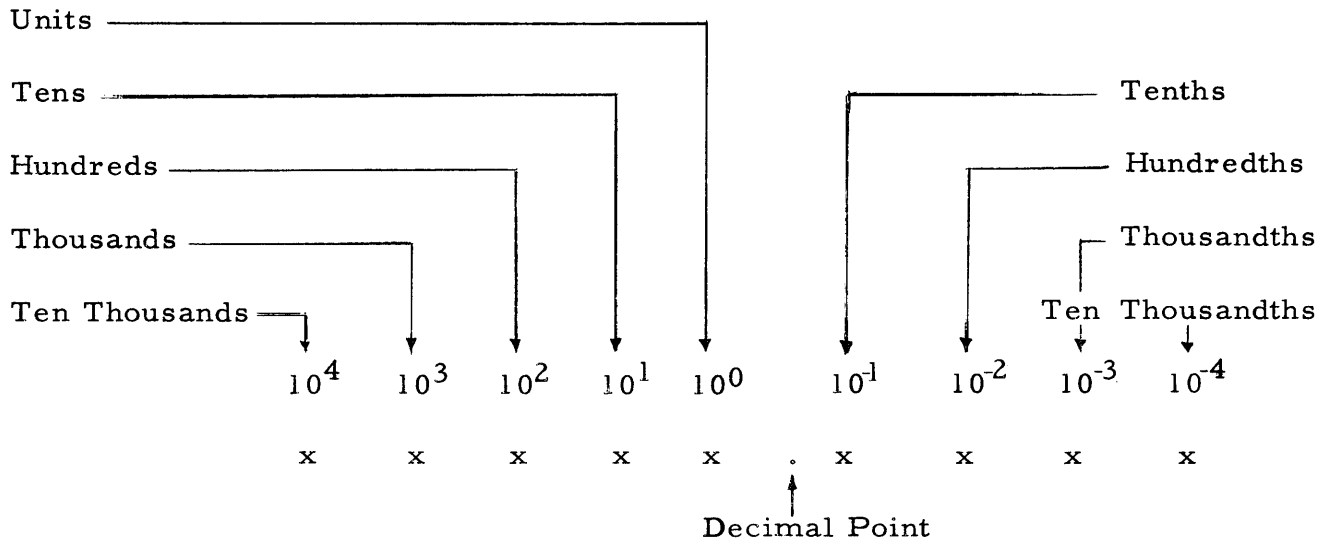
The position of each symbol gives its value in relation to a base of ten, and the symbol itself indicates how many of each value.

The same system is used in moving in the other direction away from the decimal point, giving tenths, hundredths, etc.

TABLE VIII-1
SOME NUMBER SYSTEMS

Things													
Roman (No Base)		I	II	III	IV	V	VI	VII	VIII	IX	X	XI	XII
Duodecimal (Base 12)	0	1	2	3	4	5	6	7	8	9	t	e	— 10
Undecimal (Base 11)	0	1	2	3	4	5	6	7	8	9	t	— 10	11
▶ DECIMAL (Base 10)	0	1	2	3	4	5	6	7	8	9	— 10	11	12
Nonary (Base 9)	0	1	2	3	4	5	6	7	8	— 10	11	12	13
▶ OCTAL (Base 8)	0	1	2	3	4	5	6	7	— 10	11	12	13	14
Septenary (Base 7)	0	1	2	3	4	5	6	— 10	11	12	13	14	15
Senary (Base 6)	0	1	2	3	4	5	— 10	11	12	13	14	15	20
Quinary (Base 5)	0	1	2	3	4	— 10	11	12	13	14	20	21	22
Quaternary (Base 4)	0	1	2	3	— 10	11	12	13	20	21	22	23	30
Ternary (Base 3)	0	1	2	— 10	11	12	20	21	22	100	101	102	110
▶ BINARY (Base 2)	0	1	— 10	11	100	101	110	111	1000	1001	1010	1011	1100

Positional values for the decimal number system are:



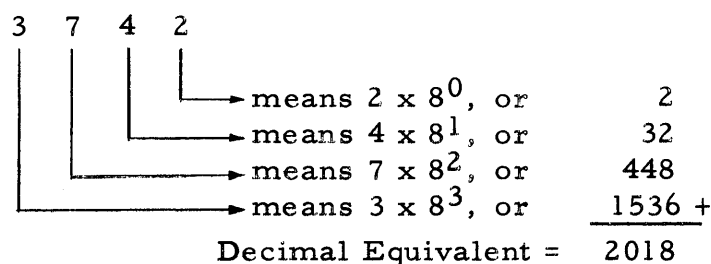
B. OCTAL NUMBER SYSTEM

The octal number system follows the same rules; that is, it is positional in nature. For each column, however, there are only eight possible symbols instead of ten. Octal counting is therefore as follows:

0
1
2
3
4
5
6
7

10 = 1 cycle of eight, + 0
11 = 1 cycle of eight, + 1
12 = 1 cycle of eight, + 2
.
.
.
17 = 1 cycle of eight, + 7
20 = 2 cycles of eight, + 0
etc.

In the octal system we run out of symbols with 7 and must then move to the next column to count complete cycles. "Eight" (a complete cycle from 0 through 7) is therefore represented by 10. Two complete cycles, 20, means we have gone through eight symbols twice, so the octal number 20 is equivalent to the decimal number 16. Note that, in any number system, "10" represents one complete base cycle. Decimal 10 means "ten", octal 10 means "eight", and binary 10 means "two." In the octal system, the first column represents "units". The second column represents "eights", the third "sixty-fours", etc. The positional pattern is exactly the same as in the decimal system, but each column is related to a base of eight instead of ten. The octal number 3742 (written 3742₈) therefore means two units and four eights and seven sixty-fours and three five-hundred-and-twelves:



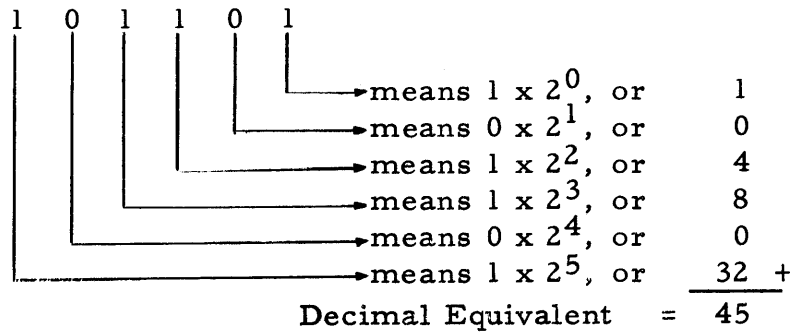
That is, 3742₈ = 2018₁₀

C. BINARY NUMBER SYSTEM

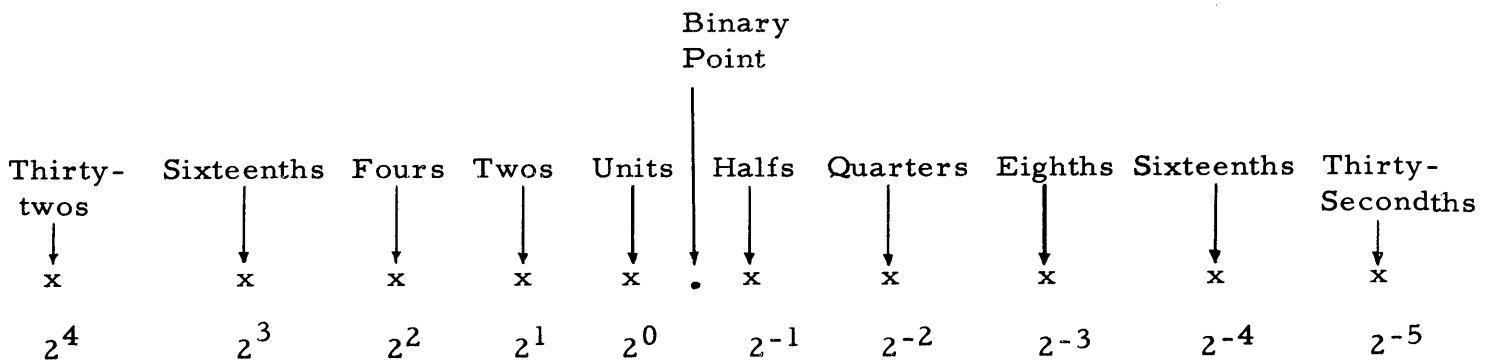
The binary number system follows the same positional pattern, but has only two possible symbols for each column.

0
1
10 = 1 cycle of two, + 0
11 = 1 cycle of two, + 1
100 = 2 cycles of two, + 0
101 = 2 cycles of two, + 1
110 = 3 cycles of two, + 0
111 = 3 cycles of two, + 1
1000 = 4 cycles of two, + 0

Here we run out of symbols at 1, and so we must move into the next column for "two". The first column represents "units". The second column represents "twos", the third "fours", etc. The binary number 101101 therefore means one unit and no twos and one four and one eight and no sixteens and one thirty-two:



The values of the binary position are as follows:



Memorize the first ten binary numbers:

Decimal	Binary
0	= 0000
1	= 0001
2	= 0010
3	= 0011
4	= 0100
5	= 0101
6	= 0110
7	= 0111
8	= 1000
9	= 1001
10	= 1010

D. NUMBER SYSTEM CONVERSION

1. Binary to Octal

A large binary number is difficult to read or understand at a glance, but can readily be converted into the more easily grasped octal equivalent.

To convert from binary to octal,

- 1) Separate the binary number into groups of three digits, starting at the binary point.
- 2) Recall the octal equivalent of each three-digit group. (Same as decimal, but three digits cannot represent a higher number than "seven".)

For example, the binary number 110111101011010 is converted to octal 67532 as follows:

$$\begin{array}{ccccc} \underbrace{110} & \underbrace{111} & \underbrace{101} & \underbrace{011} & \underbrace{010} \\ 6 & 7 & 5 & 3 & 2 \end{array}$$

2. Octal to Binary

To convert an octal number to binary, recall the binary equivalent of each octal digit.

$$\begin{array}{ccccc} 6 & 7 & 5 & 3 & 2 \\ \wedge & \wedge & \wedge & \wedge & \wedge \\ 110 & 111 & 101 & 011 & 010 \end{array}$$

3. Binary to Decimal

To convert a binary number to decimal form,

- 1) Multiply each digit by its positional value,
- 2) Add the results.

For example, the binary number 1100101.1011 is changed to its decimal equivalent 101.6875 as follows:

1	1	0	0	1	0	1	.	1	0	1	1		
								→	1 x .0625	=	.0625		
								→	1 x .125	=	.125		
								→	0 x .25	=	.00		
								→	1 x .5	=	.5		
								→	1 x 1	=	1.		
								→	0 x 2	=	0.		
								→	1 x 4	=	4.		
								→	0 x 8	=	0.		
								→	0 x 16	=	00.		
								→	1 x 32	=	32.		
								→	1 x 64	=	64.		
													101.6825

4. Decimal to Binary

Converting a decimal number to binary is performed differently for whole numbers and for fractions.

Whole Numbers

- 1) Divide the decimal number by 2. The remainder will be either 1 or 0.
- 2) If the remainder is 1, the rightmost binary digit is 1. If 0, the rightmost binary digit is 0.
- 3) Divide the quotient (obtained in step one) by 2.
- 4) Write the remainder as the next binary digit.
- 5) Continue, writing the binary digits from right to left.

For example, the decimal number 76 is converted to binary 1001100 as follows:

$$\begin{array}{r} 38 \\ 2 \overline{)76} \end{array} \quad \text{Remainder} \longrightarrow \underline{0}$$
$$\begin{array}{r} 19 \\ 2 \overline{)38} \end{array} \quad \text{Remainder} \longrightarrow \underline{00}$$
$$\begin{array}{r} 9 \\ 2 \overline{)19} \end{array} \quad \text{Remainder} \longrightarrow \underline{100}$$
$$\begin{array}{r} 4 \\ 2 \overline{)9} \end{array} \quad \text{Remainder} \longrightarrow \underline{1100}$$
$$\begin{array}{r} 2 \\ 2 \overline{)4} \end{array} \quad \text{Remainder} \longrightarrow \underline{01100}$$
$$\begin{array}{r} 1 \\ 2 \overline{)2} \end{array} \quad \text{Remainder} \longrightarrow \underline{001100}$$
$$\begin{array}{r} 0 \\ 2 \overline{)1} \end{array} \quad \text{Remainder} \longrightarrow \underline{1001100}$$

This system can be used to convert any whole number to the equivalent number of a lower base. In each case, divide by the base desired. If the decimal number 76 is first converted to octal (by dividing by 8), fewer steps are required to reach the binary equivalent:

$$8 \overline{) 76} \quad \text{Remainder} \longrightarrow \underline{4}$$

$$8 \overline{) 9} \quad \text{Remainder} \longrightarrow \underline{14}$$

$$8 \overline{) 1} \quad \text{Remainder} \longrightarrow \underline{114}$$

Then convert the octal 114 to binary by inspection:

$$\begin{array}{ccc} \begin{array}{c} 1 \\ \wedge \\ 001 \end{array} & \begin{array}{c} 1 \\ \wedge \\ 001 \end{array} & \begin{array}{c} 4 \\ \wedge \\ 100 \end{array} \quad \text{or } 1001100 \end{array}$$

$$76_{10} = 114_8 = 1001100_2$$

Fractions

- 1) Multiply the decimal fraction by 2.
- 2) If the result is 1.0 or more, the leftmost binary digit is 1. If the result is less than 1, the leftmost binary digit is 0.
- 3) Continue. Do not multiply any number to the left of the decimal point. Write the binary fraction from left to right.
- 4) Stop either when the result of a multiplication is exactly 1, or when the binary fraction reaches the desired length.

For example, the decimal fraction 0.6875 is converted to the binary equivalent 0.1011 as follows:

	0.6875		
	<u> x2</u>		
Ignore	1.3750	greater than 1, so	0. <u>1</u>
			↙
	.3750		
	<u> x2</u>		
	.7500	less than 1, so	0.1 <u>0</u>
			↙
	.7500		
	<u> x2</u>		
Ignore	1.5000	greater than 1, so	0.1 <u>01</u>
			↙
	.5000		
	<u> x2</u>		
	1.0000	exactly 1, so	0.101 <u>1</u> and stop
			↙

This system can be used to convert any fraction to the equivalent fraction of a lower base. In each case, multiply by the base desired.

E. BINARY ARITHMETIC

Arithmetic with binary numbers is quite simple, because only 1 and 0 are used. The rules for binary arithmetic are illustrated below.

Addition:

0	1	0	1
<u>+1</u>	<u>+0</u>	<u>+0</u>	<u>+1</u>
1	1	0	1←0

Carry 1 to left

Borrow 1 from left

Subtraction:

1→0	1	0	1
<u>-1</u>	<u>-0</u>	<u>-0</u>	<u>-1</u>
1	1	0	0

Multiplication:

0	1	0	1
<u>x1</u>	<u>x0</u>	<u>x0</u>	<u>x1</u>
0	0	0	1

Division:

$\frac{0}{1} = 0$	$\frac{1}{0} =$ (not defined)
$\frac{1}{1} = 1$	$\frac{0}{0} =$ (not defined)

The following examples illustrate arithmetic operations for equivalent decimal and binary numbers.

<p>Addition:</p> $\begin{array}{r} 5 \\ +3 \\ \hline 8 \end{array}$ $\begin{array}{r} 29 \\ +22 \\ \hline 51 \end{array}$ $\begin{array}{r} 4.75000 \\ 3.59375 \\ \hline 8.34375 \end{array}$	$\begin{array}{r} 111 \leftarrow \text{carries} \\ 101 \\ + 011 \\ \hline 1000 \\ 111 \leftarrow \text{carries} \\ 11101 \\ +10110 \\ \hline 110011 \\ 1111 \leftarrow \text{carries} \\ 100.11000 \\ 11.10011 \\ \hline 000.01011 \end{array}$
---	---

<p>Subtraction:</p> $\begin{array}{r} 11 \\ -6 \\ \hline 5 \end{array}$ $\begin{array}{r} 54,5390625 \\ 19,0234375 \\ \hline 35,5156250 \end{array}$	$\begin{array}{r} \text{Borrow} \\ 1011 \\ -110 \\ \hline 101 \\ \text{Borrow} \\ \text{Borrow} \quad \text{Borrow} \\ 110110.1000101 \\ -10011.0000011 \\ \hline 100011.1000010 \end{array}$
--	---

<p>Multiplication:</p> $\begin{array}{r} 5 \\ \times 3 \\ \hline 15 \end{array}$ $\begin{array}{r} 4.750 \\ \times 3.625 \\ \hline 23750 \\ 9500 \\ 28500 \\ 14250 \\ \hline 17.218750 \end{array}$	$\begin{array}{r} 101 \\ \times 11 \\ \hline 101 \\ 101 \\ \hline 1111 \\ 100.110 \\ 11.101 \\ \hline 100110 \\ 000000 \\ 100110 \\ 100110 \\ 100110 \\ \hline 10001.001110 \end{array}$
---	--

Division:

$$4 \overline{) 8} \begin{array}{r} 2 \\ \end{array}$$

$$100 \overline{) 1000} \begin{array}{r} 10 \\ \underline{100} \\ 0000 \end{array}$$

$$5 \overline{) 10.625} \begin{array}{r} 2.125 \\ \end{array}$$

$$101 \overline{) 1010.101} \begin{array}{r} 10.001 \\ \underline{101} \\ 0000 \\ 101 \\ \underline{101} \\ 000 \end{array}$$

F. ALGEBRAIC ARITHMETIC

BASICPAC adds and subtracts algebraically. Given an instruction to "add," for instance, it compares the signs of the two numbers to determine whether it must add or subtract. The following rules are observed:

Instruction	Signs	Operation	Example
Add	Alike	Add	$(+3) + (+7) = +10$
Add	Unlike	Change sign of addend, subtract	$(+3) + (-7)$ $(+3) - (+7) = -4$
Subtract	Alike	Subtract	$(+3) - (+7) = -4$
Subtract	Unlike	Change sign of Subtrahend, add	$(+3) - (-7)$ $(+3) + (+7) = 10$

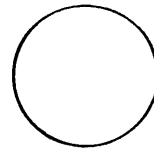
the overall logic of the program has been established, these large segments are analyzed and broken down into more detail. Eventually a detailed flow chart is produced which clearly indicates all significant points in the program and verifies all possible conditions which can occur during program execution.

The detailed flow chart becomes the basis for coding. Once coding is begun, machine logic may necessitate changes in the program logic, and the flow chart may have to be altered. After coding, testing, and debugging, the final corrected flow chart is combined with a narrative description, program listing, and coding sheets to comprise the formal program description.

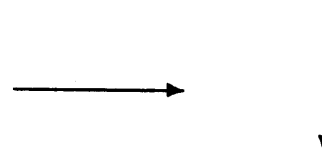
A. FLOW CHART SYMBOLS

Since BASICPAC is a member of the FIELDDATA family, the standard flow charting symbols established by the FIELDDATA Application Systems Techniques organization have been adopted. The following symbols are prescribed in FAST Programming Standard No. 3, adopted on April 27, 1960, and should be used for all application programming:

Start or Halt



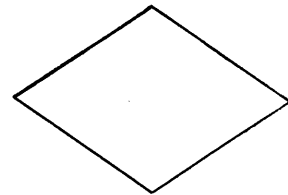
Direction of Flow



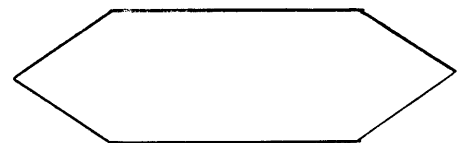
Operation or Function



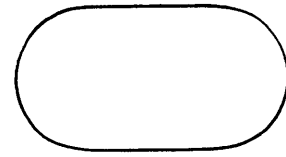
Decision



Closed Subroutine



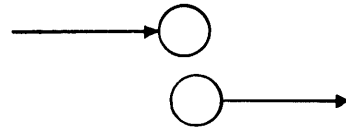
Visual Display



Communication Link

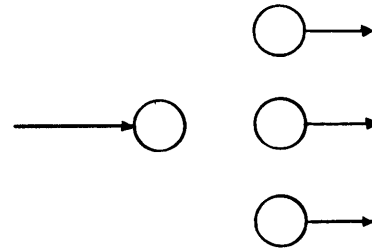


Fixed Connector



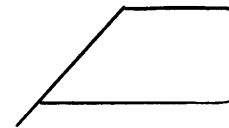
Variable Connector

Used to indicate which of several paths is to be taken; the connectors need not include the instructions which cause the path to vary. When the connector refers to a point on another page, the page number should be shown.



Flag, Explanation, Parameter

List, Etc.



Footnote or Supplemental

Information

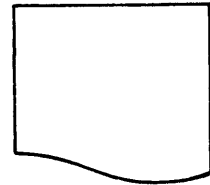


Input/Output

Card

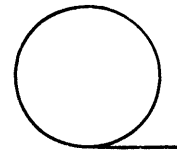


Printed Copy



General Files

The following letters are used to designate the particular type of file and are inserted in the symbol.



M - Magnetic Tape

P - Paper Tape

D - Disk

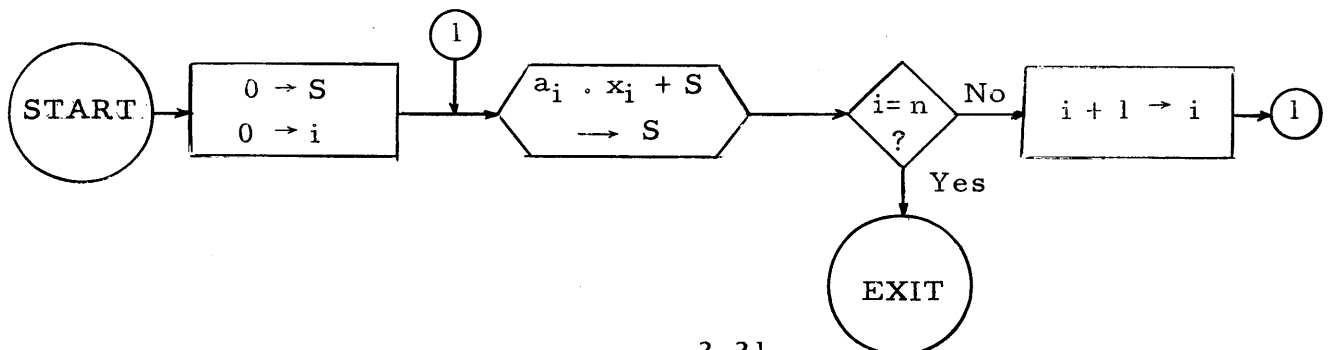
DR - Drum

B. EXAMPLES

To compute

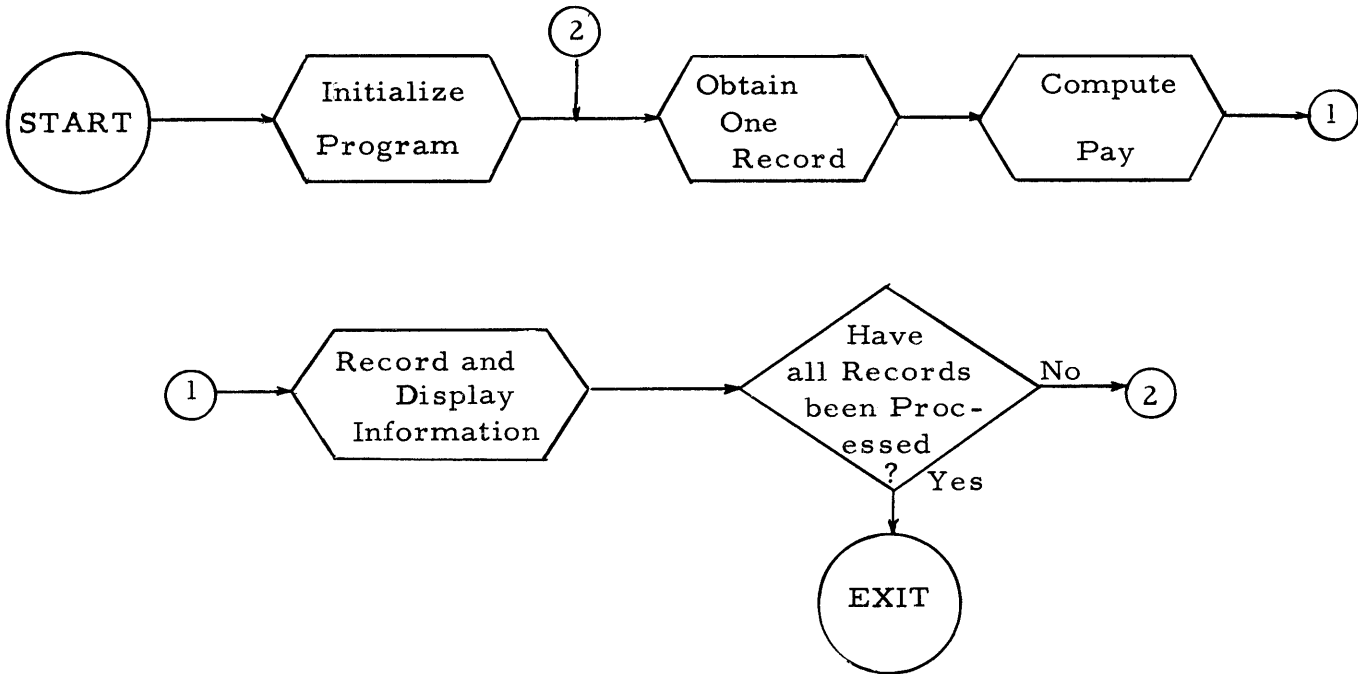
$$S = \sum_{i=0}^n a_i x_i$$

a preliminary flow chart might read:



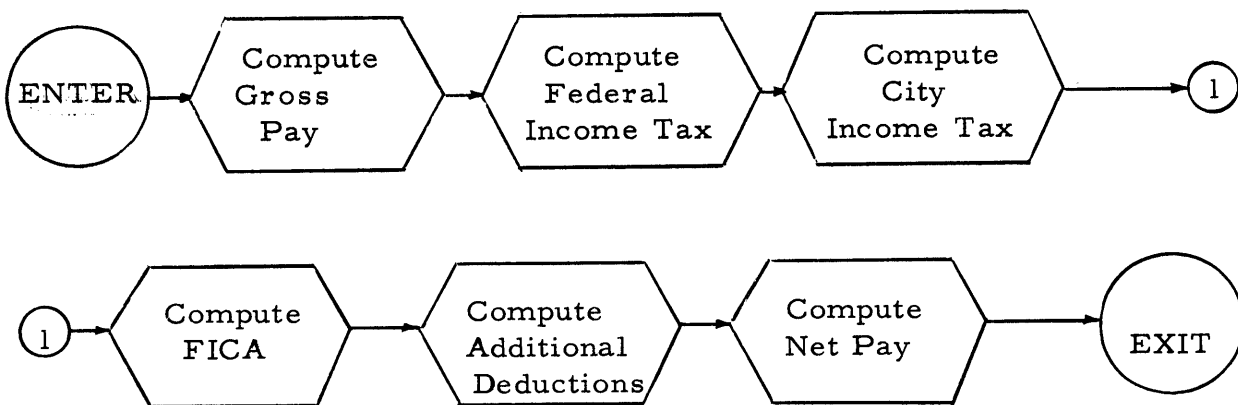
The subroutine " $a_i x_i + s \rightarrow s$ " must be flow charted in detail at a later stage.

To compute a payroll, a repliminary flow chart might read:



The subroutines "Obtain Record", "Compute Pay" and "Record and Display Information" must be described in detail at a later stage.

For example, "Compute Pay" might be flowcharted as:



This process is continued until all steps are described in sufficient detail.

XI. CODING

The previous section discussed the methods of analysis used to prepare a problem for solution on a computer. The flow chart which results from this analysis describes the method of solution in a language readily understood by humans; i. e., English-decimal. Unfortunately, computers cannot understand either of these forms of communication. The information contained in the flow diagram must be restated in a language which is meaningful to the computer by a process called coding.

All stored-program digital computers have a definite set of orders (also called instructions) which they can interpret and execute. These orders must be selected and arranged in such a way as to accomplish the functions described in the flow diagram. The resulting list of orders is called a program. A completed program is placed in the computer storage unit and is executed by the computer one order at a time. The data to be used in solving the problem must also be stored in the memory unit in a form which can be operated upon by the computer.

BASICPAC is a binary computer; that is, all information must be supplied in binary form. A set of 37 binary digits (bits) forms one computer word. A computer word may contain an order (instruction word) or data (data word), as described in Section One.

The memory unit is divided into cells called memory locations, each of which can store 37 bits. A unique numerical address is assigned to each memory location for use in referring to the information stored there. There are 4096 memory locations in one BASICPAC memory unit, numbered from 00000000000_2 to 11111111111_2 . A prefix of three additional bits is used to specify the memory unit (000_2 to 110_2), resulting in complete address of 15 bits.

Octal coding is a shorthand notation frequently used as an alternative to binary coding. The ease of translation between the binary and octal systems and the greater compactness of octal notation simplify the expression of information and instruction words. A thorough understanding of octal arithmetic is of great value to programmers.

Coding in octal or binary is known as fixed or absolute coding. An address is assigned to each instruction word and the content of each location is specified. This method of coding can be very tedious, time-consuming, and susceptible to error, particularly when it is discovered that an instruction was omitted from the beginning of a long program.

Another form of coding evolved to simplify or avoid many of these problems uses a combination of alphabetic and numeric symbols to represent an instruction word. This form of coding, called mnemonic coding, is closely related to the structure of an instruction word and is easily converted into octal coding.

A three-letter mnemonic code is assigned to each order (ADD for 001010, SUB for 001110, etc.) The names of all instructions and the corresponding octal and mnemonic codes are listed in Appendix D.

When coding in binary or octal, the arrangement of the sections of an instruction word is fixed. In mnemonic coding, the parts of an instruction may be written in any convenient order, so long as a given convention is followed. The coding used in the examples which follow use the mnemonic order codes but conform with the structure of the BASICPAC instruction word to illustrate the correspondence between mnemonic and fixed coding. An even more convenient method of coding will be outlined in Section XIV. However, a firm understanding of instruction structure must be acquired first.

All addresses and numbers in these examples are expressed in octal. When decimal quantities are mentioned, it is assumed that their octal equivalents have been already computed. All instructions and data used by the examples are assumed to be present in memory when required. The techniques for entering information into memory and obtaining information from memory will be discussed in later sections.

A. TRANSFER OF INFORMATION

One of the most frequently used steps in programming is the transfer of information from one section of the computer to another. Transfer of information provides the computer with data to be processed, removes data which has been processed, provides information for computer control functions, and provides operands for arithmetic operations. The unit of information transfer is one computer word.

Information can be transferred from memory locations to arithmetic registers, from arithmetic registers to memory locations and from register to register. The arithmetic registers include the A, Q, and WSR registers, each of which is addressable. That is, the programmer can enter information into these registers or obtain information from them by specifying their addresses in an instruction. The names and octal address of all addressable registers are listed in Appendix B. The use of each register will be explained in conjunction with the instructions which utilize it.

When information is transferred it is duplicated in a new section of the computer. The information contained in the original section is unaffected, but the information previously contained by the receiving section is destroyed. The instructions which accomplish the transfer of information are the Load, Store, and Clear and Add instructions.

The Load instruction (LOD, 51) transfers information from any addressable register or memory location to any addressable register. The location from which the information is to be obtained (Y) and the location to which it is to be transferred (X) must be specified as follows:

OP	γ	β	α
LOD		X	Y

(Load X with Y)

Only the contents of the register specified by X will be altered.

The Store instruction (STR, 50) transfers information from the A register to any addressable register or memory location. The instruction is written as follows:

OP	γ	β	α
STR			Y

(Store in Y)

Only the location specified by Y will be altered.

The Clear and Add instruction (CLA, 10) transfers information to the A register from any addressable register or memory location. The instruction is specified as follows:

OP	γ	β	α
CLA			Y

(Clear add Y)

Only the contents of the A register will be altered.

For example, consider the mnemonic coding:

OP	γ	β	α
CLA			1236
LOD		Q	A
STR			1237

This would appear in octal coding as:

OP	γ	β	α
10	0	0000	01236
51	0	0011	70010
50	0	0000	01237

and in binary coding as:

OP	γ	β	α
001 000	000	000 000 000 000	000 001 010 011 110
101 001	000	000 000 001 001	111 000 000 001 000
101 000	000	000 000 000 000	000 001 010 011 111

Before the execution of these instructions, the contents of each location might appear as follows:

A Register

+	102	030
---	-----	-----

Q Register

+	552	376
---	-----	-----

memory loca-
tions 1236

-	123	456
---	-----	-----

1237

+	765	432
---	-----	-----

After the execution of CLA 1236:

A Register

-	123	456
---	-----	-----

Q Register

+	552	376
---	-----	-----

Memory locations

1236

-	123	456
---	-----	-----

1237

+	765	432
---	-----	-----

After the execution of LOD Q A:

A Register

-	123	456
---	-----	-----

Q Register

-	123	456
---	-----	-----

Memory locations

1236

-	123	456
---	-----	-----

1237

+	765	432
---	-----	-----

After the execution of STR 1237:

A Register

-	123	456
---	-----	-----

Q Register

-	123	456
---	-----	-----

Memory locations

1236

-	123	456
---	-----	-----

1237

-	123	456
---	-----	-----

If an address which refers to a non-existent memory location or register is used in a BASICPAC command, the computer proceeds as though an existing location containing positive zero had been specified. Such addresses are called illegal addresses. Illegal addresses such as 70000_g provide a simple method of clearing registers. However, if an illegal address is specified in a STR instruction no action will occur.

For example, the instruction

OP	γ	β	α
CLA			70000

will set the A register to +000 000 000 000.

Summary

1. Information can be transferred between memory locations and addressable registers.
2. Information transfer instructions are LOD, STR, CLA.
3. STR and CLA always move information from or to the A register.
4. LOD can transfer information from addressable registers or memory locations to addressable registers. LOD requires that two addresses be specified.
5. Only the receiving element is altered.

B. ADDITION AND SUBTRACTION

BASICPAC arithmetic instructions require one of the operands to be initially placed in the A register. This operand is replaced by the result. The other operand may be taken from memory or from any addressable register.

The addition and subtraction commands are as follows:

<u>Command</u>			
<u>Octal</u>	<u>Mnemonic</u>	<u>Explanation</u>	<u>Name</u>
10	CLA	$0+(\alpha) \rightarrow A$	Clear Add (α)
12	ADD	$A+(\alpha) \rightarrow A$	Add (α)
13	ADM	$A+ (\alpha) \rightarrow A$	Add Magnitude (α)
14	CLS	$0-(\alpha) \rightarrow A$	Clear Subtract (α)
16	SUB	$A-(\alpha) \rightarrow A$	Subtract (α)

Before execution
of ADD 1723

+	012436705431
+	102030405060

A register

memory location 1723

After execution
of ADD 1723

+	114467312511
+	102030405060

A register

memory location 1723

Before execution
of ADD 1724

+	012436705431
-	102030405060

A register

memory location 1724

After execution
of ADD 1724

-	067371477427
-	102030405060

A register

memory location 1724

Before execution
of SUB 1632

+	012436705431
+	102030405060

A register

memory location 1632

After execution
of SUB 1632

-	067371477427
+	102030405060

A register

memory location 1632

Before execution
of SUB 1633

+	012436705431
-	102030405060

A register

memory location 1633

After execution
of SUB 1633

+	114467312511
-	102030405060

A register

memory location 1633

If the result of any operation is zero, the sign of this result is the sign of the operand originally placed in the A register.

It is assumed that all numbers used in the examples have been properly scaled.

Example 2:

A basic inventory operation consists of adding the quantity of an item of stock purchased to the amount of stock previously on hand, and subtracting from this sum the quantity of stock used.

If the following numbers represent quantities of a particular type of transistor:

Amount on hand	11,463
Amount purchased	5,000
Amount used	7,500

then the updated inventory (new amount on hand) would result from the following arithmetic:

$$11,463 + 5,000 - 7,500 = 8,963$$

With the octal equivalents of this data stored in memory as shown below, the problem is to perform the arithmetic described above and to store the new on-hand amount in memory location 6374:

<u>Memory Location</u>	<u>Contents</u>
6371	Amount on Hand
6372	Amount Purchased
6373	Amount Used

The coding to do this is:

OP	γ	β	α	REMARKS
CLA			6371	Clear Add On-Hand to A
ADD			6372	Add Purchased to A
SUB			6373	Subtract Used from A
STR			6374	Store Result in memory

The instruction CLA 6371 could be replaced by the instruction

OP	γ	β	α
LOD		A	6371

and the effect would be the same.

Example 3:

The following payroll information for an employee is stored in memory as shown:

<u>Memory Location</u>	<u>Contents</u>
3475	Gross Base Pay
3476	Overtime Pay
3477	Social Security Tax
3500	City Income Tax
3501	Federal Income Tax

Compute the employee's net pay and store it in memory location 3502.

Solution:

OP	γ	β	α	REMARKS
LOD		A	3475	Load A with Gross Pay
ADD			3476	Add Overtime Pay to A
SUB			3477	Subtract Social Security Tax
SUB			3500	Subtract City Income Tax
SUB			3501	Subtract Federal Income Tax
STR			3502	Store Net Pay in Memory

Exercises:

1. Add the number in memory location 4132 to each of the numbers in memory locations 6142-6144 inclusive. Each sum should replace its operand in 6142-6144.
2. Compute the sum of the numbers stored in memory locations 5301-5306.

Summary:

1. All addition and subtraction occurs through the A register.
2. The operands are treated algebraically.
3. The addition and subtraction instructions are CLA, ADD, ADM, CLS, SUB.
4. The result is always placed in the A register.
5. A result of zero has the same sign as the operand in the A register.

C. MULTIPLICATION AND DIVISION

The multiplication of two 36-digit numbers (BASICPAC register size) usually produces a product with more than 36 digits. Since this product cannot exceed 64 digits, the A and Q registers are combined to hold the result of a multiplication operation.

The Multiply instruction (MLY, 20) forms the product of the contents of the A register and the contents of any other addressable register or memory location. The 36 most significant bits of the product are placed in the A register and the 36 least significant bits are placed in the Q register. Both registers have the sign of the product.

For example, consider the following instruction:

OP	γ	β	α
MLY			1327

Before execution, the contents of the Q register are random, the A register contains the multiplicand, and the address of the multiplier is specified by the α portion of the instruction word.

<u>Before</u>		<u>After</u>
+ 000123456000	A register	+ 000000000027
- 077700000000	Q register	+ 404740000000
+ 000000220000	Location 1327	+ 000000220000

where $(123456_8) \times (22_8) = 2740474_8$.

Example 1

Compute the total cost of purchasing a quantity of items based on the following information. Store the result in memory location 2732.

<u>Memory Location</u>	<u>Contents</u>
1214	Quantity Purchased
3726	Unit Cost
1423	Percentage Discount
1500 - 1501	Working Storage

No consideration will be given to the actual format of the numbers used in the example. It is assumed that all significant digits of the products will be found in the A register, as can easily be arranged by the techniques described in Section XI.

The necessary arithmetic for this example is:

a. $(\text{Quantity} \times \text{Cost}) - (\text{Quantity} \times \text{Cost} \times \text{Discount})$

or

b. $(\text{Quantity} \times \text{Cost}) \times (1 - \% \text{ discount})$

A number of other arithmetic operations are also possible and each operation can be coded in a variety of ways. Solution a is illustrated below as a straightforward method.

OP	γ	β	α	REMARKS
CLA			1214	Quantity to A
MLY			3726	Unit Cost x Quantity to A
STR			1500	Store temporarily
MLY			1423	Unit Cost x Quantity x Discount to A
STR			1501	Store temporarily
CLA			1500	Unit Cost x Quantity to A
SUB			1501	Subtract Discount
STR			2732	Store result in memory

Another method of computing the same quantity is as follows:

$$\text{Average Unit Cost} = \text{Unit Cost} - (\text{Unit Cost} \times \text{Discount})$$

$$\text{Total Cost} = \text{Average Unit Cost} \times \text{Quantity}$$

The coding for this solution is:

OP	γ	β	α	REMARKS
CLS			3726	0-(Unit Cost) to A
MLY			1423	-(Unit Cost) x Discount to A
ADD			3726	Unit Cost-(Unit Cost x Discount) to A
MLY			1214	Net unit cost x Quantity to A
STR			2732	Store Total Cost in memory

Exercise 1:

Part of a production calculation requires computing the cost of manufacturing parts. Using the information given below, compute the cost by multiplying Number of Assemblies x Number of Parts per Assembly x Unit Cost per Part.

Memory LocationContents

4231	Number of Assemblies
4232	Number of Parts/Assembly
4233	Unit Cost

Exercise 2:

Compute the square of the sum of the quantities stored in locations 6271 and 6272. Show two ways of coding this problem.

$$(a + b)^2 = a^2 + 2ab + b^2$$

The Divide instruction (DVD, 22) divides the contents of the A register by the contents of the register or memory location specified by α . The quotient is placed in the Q register and the remainder is placed in the A register.

The Divide Long instruction (DVL, 23) divides the 72-bit number formed by contents of the A and Q registers by the contents of the register or memory location specified by α . Both the A and Q register must have the same sign. The A register contains the 36 most significant bits of the dividend and the Q register contains the 36 least significant bits. The quotient is placed in the Q register and the remainder is placed in the A register.

For both instructions, the dividend must be smaller than the divisor or the division will not be performed.

Example 1

Memory locations 4371 - 4373 contain the cost of living indices for three years. Compute the average index; i. e., $(\text{Index 1} + \text{Index 2} + \text{Index 3}) \div 3 = \text{Average Index}$. Assume a constant of 3 in memory location 4374.

OP	γ	β	α	REMARKS
LOD		A	4371	Index 1 \rightarrow A
ADD			4372	A + Index 2 \rightarrow A
ADD			4373	A + Index 3 \rightarrow A
DVD			4374	A \div 3 \rightarrow Q

The DVL instruction is generally used to divide a product previously formed or to obtain additional accuracy.

Example 2:

An airplane travels a prescribed distance D a number of times. It travels the course x times in a time period t_1 , y times in a period of time t_2 , and z times in a period of time t_3 . Compute the airplane's average speed R :

$$R = D \left(\frac{x + y + z}{t_1 + t_2 + t_3} \right)$$

Assume the following:

<u>Memory Location</u>	<u>Contents</u>
1260	x
1261	y
1262	z
1263	t_1
1264	t_2
1265	t_3
1266	D
1267	temporary storage

Solution:

OP	γ	β	α	REMARKS
CLA			1263	$0 + t_1 \rightarrow A$
ADD			1264	$A + t_2 \rightarrow A$
ADD			1265	$A + t_3 \rightarrow A$
STR			1267	$t_1 + t_2 + t_3 \rightarrow$ memory
CLA			1260	$0 + x \rightarrow A$
ADD			1261	$A + y \rightarrow A$
ADD			1263	$A + z \rightarrow A$
MLY			1266	$L(x + y + z) \rightarrow A, Q$
DVL			1267	$L(x + y + z) / t_1 + t_2 + t_3 \rightarrow Q$

Exercise 1:

The following information is given:

Memory Location

3744	The number of sales for month 1
3745	The number of sales for month 2
3746	The number of sales for month 3
3747	The total dollar receipts for month 1 sales
3750	The total dollar receipts for month 2 sales
3751	The total dollar receipts for month 3 sales

1. Compute the average number of sales for one month.
2. Compute the average dollar receipts for one month.
3. Compute the average dollar receipt for the average monthly sales.

1.
$$\frac{\text{Sales 1} + \text{Sales 2} + \text{Sales 3}}{3} = \text{Average Sales}$$

2.
$$\frac{\text{Receipts 1} + \text{Receipts 2} + \text{Receipts 3}}{3} = \text{Average Receipts}$$

3.
$$\frac{\text{Average Receipts}}{\text{Average Sales}} = \text{Average Receipt/Sale}$$

Assume, as always, that the operands will be properly aligned throughout. Also assume that the number 3 is stored in memory location 3752.

Summary:

1. The multiplication and division instructions are MLY, DVD, DVL.
2. Both multiplication and division assume the presence of one operand in the A register.
3. The product is placed in the A and Q registers.
4. The quotient is placed in the Q register; the remainder is placed in the A register.

D. TRANSFER OF CONTROL

The instructions which comprise a BASICPAC program are stored in consecutively numbered memory locations for execution. These instructions are performed sequentially, beginning with the smallest address and continuing to the largest. The order of execution of the instructions is controlled by a special register which specifies the address of the next instruction to be performed. As each new instruction has been obtained from memory, the contents of this register are increased by one to supply the address of the next instruction in sequence. This special register is called the Program Counter register (PC).

The sequential method of operation described above permits only one possible sequence of events for each program. Depending upon the answers to questions such as:

1. Have sufficient terms of a series been computed?
2. Is there a transaction for this record?
3. Is there sufficient stock available to fill this order?

A choice can be made of the appropriate action to be initiated.

Although these questions appear dissimilar, they can be generalized as follows:

1. Is a number positive? Negative? Zero?
2. Is one identifying number or name the same as another?
3. Is a number greater than another? Less than another?

Answers to the first set of questions can be obtained by examining an arithmetic register to see if the contents are positive, negative or zero. Similarly, two quantities can be compared by examining their difference.

A group of instructions called the Transfer of Control instructions enables the programmer to choose between the continuation of a sequence of operations and the initiation of a new sequence.

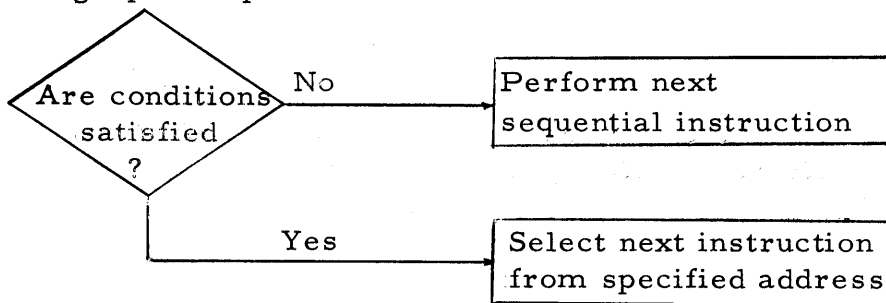
The Transfer of Control instructions which depend upon information supplied are called conditional transfers. An additional instruction called an unconditional transfer transfers control to a new sequence of instructions regardless of any data supplied.

All transfer of control instructions have a mnemonic code of the type TRc, where c specifies the condition to be satisfied.

<u>Octal</u>	<u>Mnemonic</u>	<u>Name</u>
40	TRU	Transfer Unconditionally
44	TRP	Transfer on Positive A register
46	TRN	Transfer on Negative register
45	TRZ	Transfer on Zero A register
41	TRL	Transfer and Load PCS register
42	TRS	Transfer to location in PCS register
43	TRX	Transfer on Index Register

In TRP and TRN, "positive" assumes only that the sign bit is zero; "negative" assumes only that the sign bit is one. In TRZ, zero can be either positive or negative. TRL, TRS, and TRX are discussed in later sections.

A graphic representation of conditional transfer instructions is:



A transfer of control instruction must specify the first address of the alternative coding. This address is then placed in PC, and normal sequential operation continues from this address.

Example 1:

Compute the new loan balance. If it equals zero, add one to the number of cleared balances. If it does not equal zero, add one to the number of active loans.

<u>Memory Location</u>	<u>Contents</u>
3724	Loan balance
3725	Loan payment
3726	Number of active loans
3727	Number of cleared balances

Assume memory location 4212 contains a constant of one.

LOCATION	OP	γ	β	α	REMARKS
	LOD		A	3724	Loan balance \rightarrow A
	SUB			3725	Balance - payment \rightarrow A
	STR			3724	New balance to memory
	TRZ			NOBAL	Transfer control if (A) = 0
	CLA			3726	Add one to number of active loans
	ADD			4212	
	STR			3726	

NOBAL	CLA			3727	Add one to number of cleared loans
	ADD			4212	
	STR			3727	

"NOBAL" in this case was used as the beginning address of the instructions which increment the number of cleared loans. When coding in machine language, the correct numerical address must be placed in the α portion of the TRZ (or any other Transfer of Control) instruction.

Unless this address is known, it is convenient to leave the α portion temporarily blank and continue coding the "condition not satisfied" path. When this path has been completed, the next address (or line on the coding sheet) can be used to begin coding the "condition satisfied" path. The α portion of the transfer of control instruction should be filled at this time.

The example above may be considered in two ways:

1. Has the new balance been reduced to zero?
2. Does the payment equal the balance?

The second way illustrates the method of comparing two quantities by examining their difference.

For if $x > y$, then $x - y > 0$; if $x = y$, then $x - y = 0$; and if $x < y$, then $x - y < 0$.

Example 2:

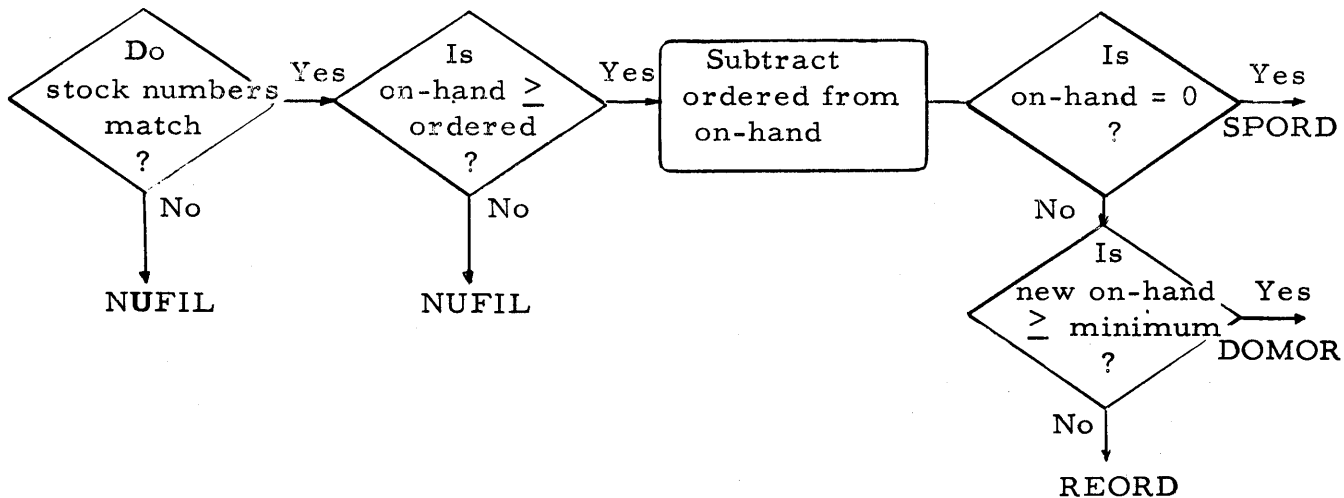
Memory locations 4623 - 4625 contain information for an inventory record. Memory locations 5326 - 5327 contain information for a transaction record.

<u>Memory Location</u>	<u>Contents</u>
4623	Inventory stock number
4624	Inventory amount
4625	Minimum required amount
5326	Transaction stock number
5327	Amount required for sale

1. If the stock numbers are the same (match), perform the processing described below. If they do not match, perform the coding at NUFIL.
2. If they match, determine whether the order can be filled. (Is the inventory amount greater than or equal to the amount ordered?) If the order can be filled, record the new inventory amount and continue. If it cannot be filled, go to NUFIL.
3. If the new inventory amount is greater than or equal to the minimum, go to DOMOR. If the inventory amount is zero, go to SPORD. If it is not, go to REORD.

(SPORD, REORD, DOMOR, and NUFIL are not coded for this example).

The flow chart for this exercise appears below:



LINE	LOCATION	OP	γ	β	α	REMARKS
1		CLA			4623	Inventory stock no. \rightarrow A
2		SUB			5326	A-Transaction stock no. \rightarrow A
3		TRZ			START	Same?
4		TRU			NUFIL	no: go to NUFIL
5	START	CLA			4624	On hand \rightarrow A
6		SUB			5327	On hand - order \rightarrow A
7		TRN			NUFIL	On hand < Order? (A) < 0?
8		STR			4624	no: save new on-hand
9		TRZ			SPORD	new on hand = 0?
10		SUB			4625	no: on hand - minimum \rightarrow A
11		TRP			DOMOR	on hand \geq minimum? A \geq 0?
12		TRU			REORD	no: go to REORD

Analysis of the Coding:

Lines 1 and 2 compute the difference between the two stock numbers.

Line 3 examines their difference. If they are unequal, there is a non-zero difference and the next instruction (line 4) will transfer control to NUFIL. If they are equal, the difference is zero and control will be transferred to START (line 5).

Lines 5 and 6 compute the difference between the amount on hand and the amount ordered. Line 7 examines the difference. If the amount on hand is greater than or equal to the amount ordered, the difference is positive since the quantities are positive and the instruction on line 8 will be performed. If the amount on hand is less than the amount ordered, control will be transferred to NUFIL.

Line 8 replaces the old amount on hand with the new amount on hand for future reference.

Line 9 inquires whether all supplies on hand have been depleted. (Is the new amount on hand zero?) If it is zero, control is transferred to SPORD. If it is not zero, control continues to line 10, which computes the difference between the new amount on hand and the minimum amount required.

Line 11 inquires whether the amount on hand is greater than or equal to the minimum. (Is the difference positive?) If the amount is greater than or equal, control is transferred to DOMOR. If the amount is smaller, the next instruction (line 12) transfers control to REORD.

Exercise:

The following data in memory pertains to an employee:

<u>Memory Location</u>	<u>Contents</u>
4727	Number of Hours Worked
4730	Hourly Pay Rate
4731	Overtime Pay Rate
4732	Number of Exemptions
4733	Union Dues
4734	Hospitalization Contribution
4735	Year to Date Gross Pay
4736	Year to Date Net Pay
4737	Year to Date Social Security Tax
4740	Year to Date Income Tax

Definitions:

Overtime hours = Hours worked in excess of 40.

Gross pay = Hours (not more than 40) x Hourly Rate +
Overtime Hours x Overtime Rate.

Income Tax = $[\text{Gross Pay} - 13 \times \text{Number of Exemptions}]$
 $\times .18$.

Social Security
Tax = $3.00 \times \text{Gross Pay}$.

Net Pay = Gross Pay - Income Tax - Social Security Tax -
Union Dues - Hospitalization Contribution.

This exercise has five parts.

Part 1:

Determine whether the employee worked overtime. If so, store the Overtime Hours in memory location 4741.

Part 2:

Compute Gross Pay, store in 4742, and add it to Year to Date Gross Pay.

Part 3:

Compute Income Tax, store in 4743, and add it to Year to Date Income Tax.

Part 4:

Compute Social Security Tax, store it in 4744, and add it to Year to Date Social Security Tax.

Part 5:

Using the above results compute Net Pay, store it in 4745, and add it to Year to Date Net Pay.

The new Year to Date Social Security Tax must not exceed \$ 144.00. Therefore, do not deduct the full 3.00 if it will cause the Year to Date total to exceed \$ 144.00.

The necessary constants for this routine are stored as follows:

<u>Memory Location</u>	<u>Contents</u>
3000	40
3001	13
3002	.18
3003	.03
3004	144.00

Summary:

1. Transfer of Control instructions permit alternate paths of processing. The information being processed can be used to determine the path to be chosen.
2. All Transfer of Control instructions have the mnemonic form TRc, where c indicates the condition required. The transfer of control instructions are TRU, TRP, TRN, TRZ, TRL, TRS and TRX.
3. When coding, leave the α portion or "go to" address blank until the "go to" coding can be written. This method enables the programmer to keep track of the coding which remains to be written.
4. When subtracting two quantities for comparison, remember that the operation will be performed algebraically, and that a zero result will have the same sign as the operand originally placed in the A register.

5. Transfer of Control instructions do not alter the contents of the arithmetic registers.

A special type of transfer of control instruction is the Halt order (HLT, 00). This instruction transfers control to the operator and all computing stops, but any input-output orders currently being performed will be completed. No new orders will be accepted.

E. INFORMATION FORMATS

In the preceding examples and illustrations, no consideration was given to the numbers actually used by the computer. Decimal numbers and decimal arithmetic were assumed. However, the specific form of information used within the computer is of great importance to the programmer. The programmer has a wide degree of latitude in selecting the formats to be used in his program, subject only to the necessary restriction of no more than 37 bits in a computer word.

There are two generalized forms of computer data words: octal and alphanumeric.

An octal word consists of twelve octal digits and a sign bit. Bits 34-36 form the most significant octal digit, bits 31-33 form the second, etc., and bits 1-3 form the least significant digit. An example of an octal word is shown below.

+	1	2	3	4	5	6	7	0	1	2	3	4
---	---	---	---	---	---	---	---	---	---	---	---	---

Octal Word Format

An octal word may contain numerical data, an information code, or a constant. The quantities 40, 3.00%, and \$144.00 in the previous exercise were constants. That is, they would remain fixed for each employee. They are specified in the program and stored in memory as part of the program, rather than being read into memory with each set of data processed.

The alphanumeric format permits the programmer to obtain and process information within the computer in a form which can be easily converted into English-decimal notation. The alphanumeric word may represent payroll or inventory information, information which is to be typed for the operator, or any other information. The alphanumeric characters are six-bit binary coded characters. The word format is shown below. The FIELDATA alphanumeric code is shown in Appendix D.

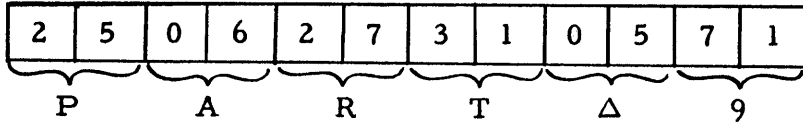
37	36-31	30-25	24-19	18-13	12-7	6-1
SN	F	0	R	M	A	T

Alphanumeric Word Format

Examples:



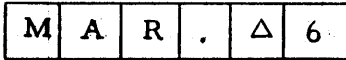
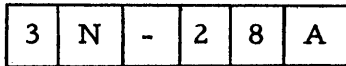
The constant "PART 9" might be a portion of a typewriter display. "Δ" indicates the space character. The alphanumeric word could also be shown in octal form; P=25₈, A=06₈, R=27₈, etc.



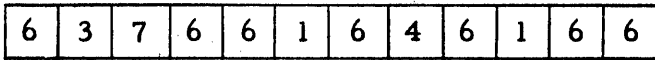
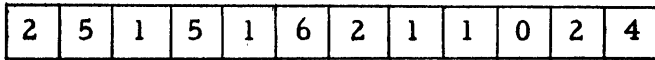
The sign is usually ignored or assumed positive for alphanumeric constants. Note that the decimal digits 8 and 9 can be expressed in alphanumeric form. The symbol "Ø" is used to distinguish the letter O from the numerical 0.

Exercises:

1. Convert to octal form:

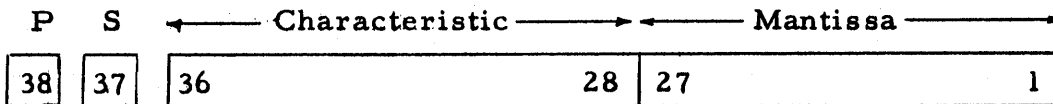


2. Convert to alphanumeric form:



F. FLOATING POINT ARITHMETIC

Floating point operations can be executed in BASICPAC by means of suitable routines. In floating point arithmetic the BASICPAC word is divided into a nine-bit characteristic and a 27-bit mantissa.



The sign and parity bits are identical in function to those in a fixed-point numeric word. All exponents are represented in an excess-256 system, in which the characteristic is equal to the exponent plus 256. The most significant bit of the characteristic can be thought of as the sign of the exponent. Positive and negative exponents in the range -256 to +255 are represented by a positive characteristic ranging from 0 to 511.

A floating point number is said to be normalized when the fractional part (if non-zero) is equal to or greater than 0.5 in absolute value. (The mantissa contains a "one" bit in position 27.) A normalized zero contains zeros in bits 1-36. Normalized floating point numbers permit a retention of the maximum number of significant bits.

G. SCALING AND SHIFTING

BASICPAC interprets all numbers as having absolute values of less than one. (The binary point is located between bits 36 and 37.) However, the programmer is not restricted to this range; he may assume a binary point anywhere within a word or outside of a word. Having made this choice, the programmer must keep track of the assumed point throughout all subsequent operations. This process of representing any desired number by selecting an appropriate binary point is called Scaling, and the number of positions between the computer's point and the assumed point is called the Scale Factor. The scale of a number is that power of two which, when multiplied by the computer number, produces the desired number.

For example:

To represent the number 4, using the computer number .5, the scale factor must be 3; i.e.,

The computer number .5 (binary .1000. . . .)
with a scale factor of 3 .100[^]00. . . (The caret, [^], indicates
the assumed binary point.) This number would be written as the octal
constant +400 000 000 000.

A positive scale factor indicates that the assumed point is to the right of the BASICPAC point. A scale factor of zero indicates that both points coincide. A negative scale factor indicates that the assumed point is to the left of the BASICPAC point. A "B" prefix is used to indicate the scale factor.

<u>Number</u>	<u>Scaled Number</u>	<u>Binary</u>
4.0	4.0B3	.100 [^] 000...
2.0	2.0B2	.10 [^] 000...
4.0 × 2.0	4.0B3 × 2.0B2 =4.0 × 2.0B5 =8.0B5	.01000 [^] 00...
8.0	8.0B5	.01000 [^] 00...
2.0	2.0B2	.10 [^] 000...
8.0 ÷ 2.0	8.0B5 ÷ 2.0B2 = (8.0 ÷ 2.0) B3 = 4.0B3	.100 [^] 00...
x	xBa	
y	yBb	
x × y	xBa × yBb = x × yB (A + B)	
x	xBa	
y	yBb	
x ÷ y	xBa ÷ yBb = x ÷ y B (a-b)	

Exercises

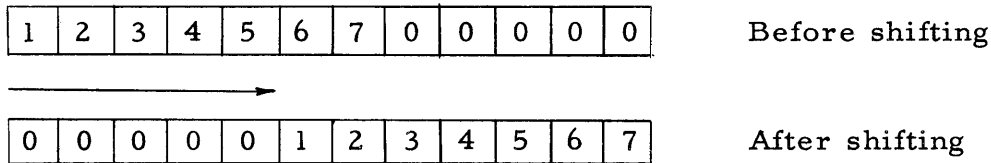
- Determine the products and quotients of the following pairs of numbers:

16.25B 22	3.5B7
6.5B 10	.125B0
.375B 1	2.0B4

2. Show how the above operands and results would appear in binary words. In octal words.

When adding and subtracting numbers, the programmer must be certain that the operands have the same scale factor. Since the value of a number is relative to its position in the word, some method of altering the position must be used to align two operands with different scale factors. This re-positioning could be accomplished by multiplying or dividing by the required power of two. It can also be accomplished by shifting the word the required number of positions.

For example, shift the number in the A register five places to the right:



Three types of shifts are possible in BASICPAC:

1. Ordinary
2. Double-Length
3. Circular

Ordinary shifts can be performed only in the A register. The sign bit is not affected.

Double-length shifts treat the A and Q registers as a single 72-bit register. The signs are not affected.

Circular shifts treat the A and Q register as a single circularly connected register. All bits including the sign bits are treated alike.

For each position shifted, one bit is shifted out of one end of the register and one bit is introduced at the other end. For ordinary and double length shifts, the bit shifted out is lost and zeros are introduced at the other end. For ordinary and double length shifts, the bit shifted out is lost and zeros are introduced at the other end. For circular shifts, the bit shifted out of one end is introduced at the other end.

Shifts can be made either to the right or to the left. The shift instructions are:

<u>Octal</u>	<u>Mnemonic</u>	<u>Name</u>
30	SHL	Shift Left (A)
32	SHR	Shift Right (A)
31	SLL	Shift Left Long (A, Q)
33	SRL	Shift Right Long (A, Q)
35	CYL	Cycle Left (A, Q)

The number of bit positions shifted is specified by the α portion of the instruction word. From 0 to 63 positions can be specified by one instruction. A shift of zero has no effect.

Shifting can be used to align scaled numbers for arithmetic processing or to align alphanumeric data. A shift right of n positions is equivalent to dividing the number by 2^n . A shift left of n positions is equivalent to multiplying a number by 2^n .

Examples:

Add the numbers $74.25B_6$ and 743.6_8B_{21} .

LOCATION	OP	γ	β	α	REMARKS
	CLA			FIRST	$74.25B_6 \rightarrow A$
	SHR			17	$21_{10} - 5 = 16_{10} = 20_8$
	ADD			NEXT	$A + 743.6B_{21} \rightarrow A$
FIRST	74	2	5000	00000	original number
	00	0	0074	25000	shifted number
NEXT	00	0	0743	60000	second number

Notice that if $743.6B_{21}$ had been aligned with $74.25B_6$ a shift left of 15 positions would have been required, and the significant digit "7" would then have been shifted out of the register.

	bit #							
	37	36	35	34	33	32	31	30
Given the number $14B_6$:								
	+	0	0	1	1	1	0 _A	0
shift it right once = $7.B_6$:								
	+	0	0	0	1	1	1 _A	0
or shift $14B_6$ left once = $28B_6$:								
	+	0	1	1	1	0	0 _A	0

Multiplication by numbers which are not powers of two can be done by shifting and adding.

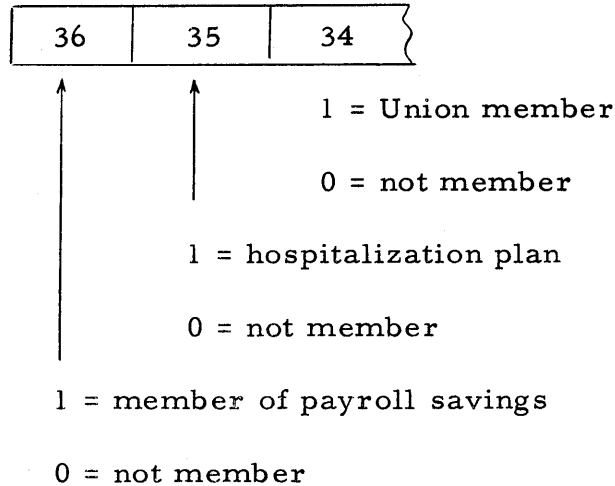
For example:

Multiply 6 x 9 by shifting.

The number 6	110 = 6
shifted left 3 times	110000 = $6 \times 2^3 = 6 \times 8$
plus the original number	110110 = $6 \times 2^3 + 6$
	= $6 \times (2^3 + 1)$
	= $6 \times 9 = 54_{10}$

Shifts can also be used to determine which of a series of possible operations is to be performed.

The high-order three data bits of the word in memory location CODE represent yes or no codes as follows:



The instructions necessary to determine whether or not to make payroll deductions for union dues, hospitalization or payroll savings is shown below.

LINE	LOCATION	OP	γ	β	α	REMARKS
1		LOD			A	code word → A
2		CYL			1	
3		TRN			SAVE	If first bit=1, go to SAVE
4		CYL			1	
5		TRN			HOSP	If next bit=1, go to HOSP
6		CYL			1	
7		TRN			UNION	If third bit=1, go to UNION

Overflow

Five arithmetic (ADD, ADM, SUB, DVD, DVL) and two logical instructions (SHL, SLL) can cause a condition known as overflow. These instructions can produce a number in the Accumulator too large to be accommodated. The result is a carry into the adjacent bit position, which in this case represents the overflow indicator. Overflow can be used by the programmer to indicate an error condition such as improper scaling, or can be used as a programming feature (e.g., to indicate whether certain bits are ones or zeros). No one automatic procedure is ideal for all possible cases, so the BASICPAC programmer has been given complete control of overflow procedures.

The addressable overflow flipflop is used to indicate to the program that overflow has occurred under the conditions set by the program. This flipflop can also halt the computer upon the detection of overflow, if so directed by the program.

Bits 16-18 of the portion of the instructions which can cause overflow determine the procedure to be followed:

<u>β_{18} β_{17} β_{16}</u>	<u>Action Before Instruction</u>	<u>If Instruction Causes Overflow</u>
000	Clear OA	Set OA and halt
001	Clear OA	Set OA
010	Clear OA	Set OA and halt
011	Clear OA	No action
100	Halt if OA=1	Set OA and halt
101	No Action	Set OA
110	Halt if OA=1	Set OA and halt
111	No Action	No Action

It is assumed that the coding at SAVE, HØSP and UNION will perform the required processing and return control respectively to lines 4, 6, and 8. These methods will be discussed at greater length in the section on subroutines.

H. LOGICAL INSTRUCTIONS

The unit of information transfer discussed in the preceding sections was one computer word. The BASICPAC logical instructions allow transfer of only part of a computer word.

The operation of these instructions is based on the rules of logical addition, multiplication, and negation. The rules for logical addition are like the rules for binary addition, except that in logical addition, 1+1=1. There is no carry. The rules for logical multiplication and binary multiplication are identical. The symbols "v" and "." are used to distinguish logical addition and multiplication from arithmetic addition and multiplication.

Logical negation replaces all binary ones with zeros and all binary zeros with ones, forming the "one's complement" of a quantity. Negation is denoted by "-" or "'".

Examples:

If $a = 1, b = 0$
 then $a \cdot b = 1 \cdot 0 = 0$
 $a \vee b = 1 \vee 0 = 1$
 $a \wedge b = 1 \wedge 0 = 0 \vee 0 = 0$
 $a \vee b = 1 \vee 0' = 0 \vee 1 = 1$
 $a \vee b' = 1 \cdot 0' = 1 \cdot 1 = 1$

If $(Q) = 110\ 001\ 101\ 110\dots$
 $(Q)' = 001\ 110\ 010\ 001\dots$

The instruction Logical Add (LGA, 02) performs the logical addition of the contents of the register or memory location specified by α to the contents of the A register and places the result in the A register. The sign bit is affected. This instruction can be used, for example, to insert ones in a yes or no code word, or to change an unknown sign bit to one.

Logical Multiplication (LGM, 03) forms the logical product of the contents of the A register and the contents of the register or memory location specified by α . The result is placed in the A register. The sign bit is affected. This instruction can be used to transfer portions of a word to the A register.

The Mask instruction (MSK, 55) replaces specified portions of a register or memory location with the information in the corresponding portions of the A register. The logical expression of the operation of the mask instruction is:

$$(A) \cdot (Q)' \cdot (\alpha) \rightarrow \alpha$$

A special constant called a mask is placed in the Q register before execution of the MSK instruction to specify the bit positions to be altered. A bit value of one indicates that the corresponding bit position is to be altered; a bit value of zero indicates that the corresponding bit position is not to be changed.

Example:

Assume an employee's rate of pay is to be changed. The rate of pay is stored with other information in memory location PAY in the following format:

(1)	(1)	(1)	(21)	(3)	(9)
U	H	S	Rate of Pay	Tax Code	Bonds

Numbers in parentheses indicate the number of bits occupied by the field. Assume the new rate of pay is stored in location NUPAY in the following form:

Other information	(21) Rate of Pay
-------------------	---------------------

The coding to update the word PAY would be:

LINE	LOCATION	OP	γ	β	α	REMARKS
1		LOD		Q	MASK	mask \rightarrow Q
2		CLA			NUPAY	new rate of pay \rightarrow A
3		SHL			22	$21_{10} - 1 = 20_{10} = 22_8$
4		MSK			PAY	
	MASK	07	7	7777	70000	
	NUPAY	00	0	00XX	XXXXX	X = desired information
	PAY	ZY	Y	YYYY	YZZZZ	Y = rate of pay Z = other information

After the above coding has been performed, the registers and memory locations will appear as follows:

A REGISTER	OX	X	XXXX	X0000
Q REGISTER	07	7	7777	70000
NUPAY	00	0	00XX	XXXXX
PAY	ZX	X	XXXX	XZZZZ

The Replace Address instruction (RPA, 54) replaces bits 1-15 of the specified memory location with bits 1-15 of the A register. The contents of the A register are not affected. This instruction will be discussed in greater detail in the section on subroutines.

Exercises:

Parts of an inventory record and a transaction record are stored in memory in the formats shown below.

Memory Location

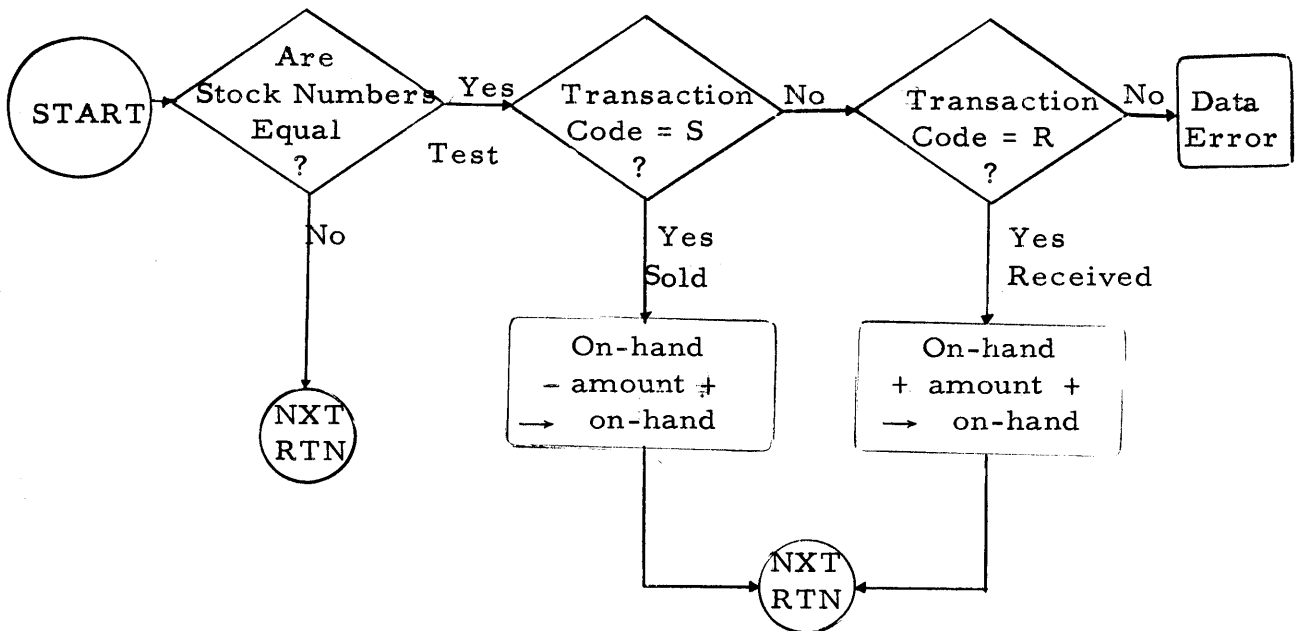
Contents

INV1	Stock number (27)				Other information
INV2	Other data				On hand amount (9)
TRAN1	Stock number (27)	0	0	0	Alphanumeric transaction code
TRAN2	Other data				Amount (9)

The numbers in parentheses indicate the number of bits in each field.
The processing to be performed is:

1. Determine whether the stock numbers are equal; if they are not, go to NXTRTN.
2. If they are equal, test the transaction code to determine whether the transaction indicates an amount sold (S) or received (R). If the transaction code is not R or S the data is in error.
3. For an amount sold, subtract the transaction amount from the on-hand amount. Then go to NXTRTN.
4. For an amount received, add the transaction amount to the on-hand amount. Then go to NXTRTN.

The flow chart of this processing is:



I. SENSE INSTRUCTIONS

The yes-or-no codes discussed previously are valuable programming aids, but these codes require all information to be present in memory. This requirement can be inconvenient under certain circumstances. The decision of whether or not a certain section of coding is to be performed may depend upon conditions external to the computer or conditions which may not be known in advance. The Sense instructions enable the operator to enter information into a set of eight one-bit registers (Sense flipflops) at any time before or during the execution of the program. A set of eight three-position switches on the control panel supply information to the flipflops. The up position of the switch corresponds to a binary value of one, called the "set state"; the down position corresponds to a binary value of zero, called the "reset state". The center position places the flipflops under program control. The switches automatically return to the center position when released by the operator, but the flipflop remains in the new state until altered by either the program or the operator.

Sense instructions can be considered as a special class of conditional Transfer of Control instructions. Instead of being dependent upon the contents of the A register, the behavior of the instructions depends upon the state of the flipflop specified by the β portion of the instruction word. The instructions used with flipflops are the Sense, Sense and Set, and Sense and Reset instructions.

The Sense instruction (SEN, 05) interrogates the specified flipflop. If it is set ($SFF\beta = 1$), control is transferred to the instruction in memory location α . If the flipflop is reset ($SFF\beta = 0$), the next instruction in sequence is performed.

Sense and Set (SNS, 06) interrogates the specified flipflop. If the flipflop is reset ($SFF\beta = 0$), it is set to one and control is transferred to memory location α . If the flipflop is already set, the next instruction in sequence is performed.

Sense and Reset (SNR, 07) interrogates the specified flipflop. If the flipflop is set, it is reset to zero and control is transferred to memory location α . If the flipflop is already set, control is transferred to the next instruction in sequence.

Note that for SNS and SNR a transfer of control occurs whenever the flipflop specified changes its state. A physically or logically non-existent flipflop; e. g., $\beta = 0363$, or IRQ for a non-existent converter, when addressed by the Sense instruction always results in a transfer of control to the next instruction in sequence.

There are other addressable flipflops in BASICPAC, most of which are concerned with input-output and will be discussed in the appropriate section. They can be addressed by any of the Sense instructions and behave exactly like the sense flipflops.

The SNR inquires if SFF1=1. It does, SFF1 is reset to 0 and control is transferred to NEXT. The second time SNR is encountered, SFF1=0 and the next instruction CONTIN is performed. This type of coding, called a loop, will be discussed at greater length in the next section.

Given the following coding, which routine will be performed if SFF6=1, SFF7=0 and SFF8=1? If SFF6=0, SFF7=1, SFF8=0? If SFF6=1, SFF7=1, SFF8=1? Note these values correspond to binary equivalents of 5, 2, and 7.

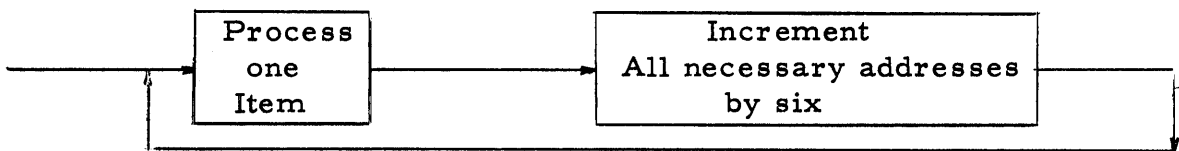
LOCATION	OP	γ	β	α	REMARKS
DØG	SEN		SFF8	LØCA	
	SEN		SFF7	LØCB	
	SEN		SFF6	CAT	(1)
	(0)
LØCB CØW	
	SEN		SFF6	PIG	(3)
	(2)
LØCA	SEN		SFF7	LØCC	
	SEN		SFF6	RAT	(5)
KID	(4)
	
LØCC BAT	SEN		SFF6	GNU	(7)
	(6)
	

J. PROGRAM MODIFICATION AND LOOPS

The preceding examples and exercises assumed that only one set of information was to be processed, whereas inventory and payroll records, for instance, normally contain more than one set of information, each set of which must be processed. The coding for processing one record could be duplicated the required number of times with appropriate changes of addresses for each set, but this method might require an enormous amount of program storage space. It would also be an extremely boring task.

An alternative method codes the processing for the first record, then modifies the coding for each successive record and performs the modified coding. This modification is possible because the program is stored in memory in number-coded form, and the computer can distinguish between a data word or an instruction word only by the fact that control is transferred to the instruction word. Hence, any instruction which can be used to modify data can be used to modify instructions.

For example, assume the entire payroll record for a company is stored consecutively in memory in a consistent format. Each item requires six memory locations. The first word of the first item is stored in memory location 2000. The flow chart for processing these items would be

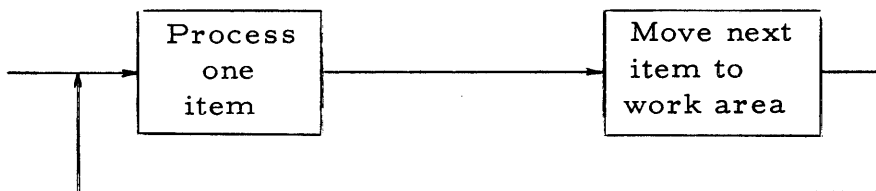


The coding is given below. It is assumed that locations $L\emptyset C1$, $L\emptyset C2, \dots$, $L\emptyset CN$ all refer to addresses in the first item.

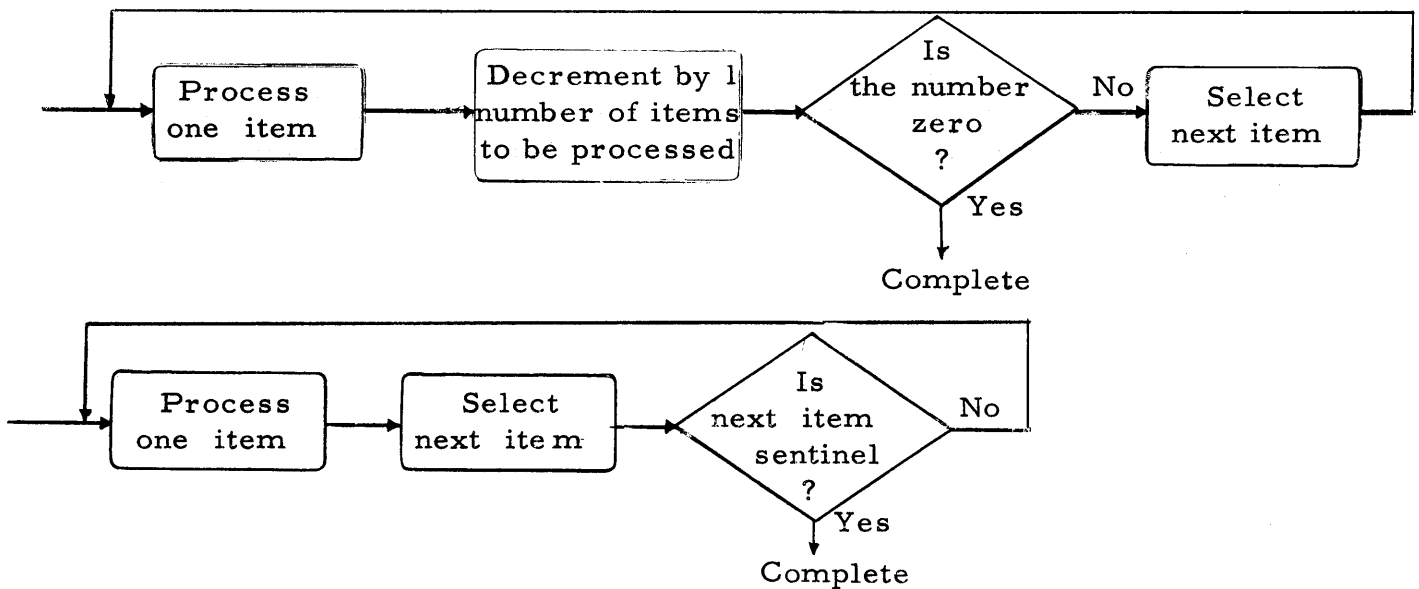
LOCATION	OP	γ	β	α	REMARKS
LØC1	CLA			2000	} process one item
.	
LØC2	ADD	.	.	2003	
.	} modify addresses for next item
LØCN	STR			2005	
	CLA			LØC1	
	ADD			SIX	
	STR			LØC1	
	CLA			LØC2	
	ADD			SIX	
	STR			LØC2	
	CLA			LØC3	
	STR	.	.	LØCN	} modify addresses for next item
BACK	TRU			LOC1	
SIX	00	0	00	00006	

However, each instruction which refers to a part of the first item must be modified.

An alternate method would be to code the processing for a specific work area, then transfer each item into this area for processing. The flow chart would be:



Some method of exiting from these loops must be allowed, or the cyclic nature of BASICPAC memory will result in processing the program as well as the data, or other undesired results. If the number of items to be processed is known in advance, a count can be kept of the number of items processed. Similarly, if an unspecified number of items are to be processed, the loop can be completed by examining each item for a pre-determined sentinel. These two methods are flow charted below.





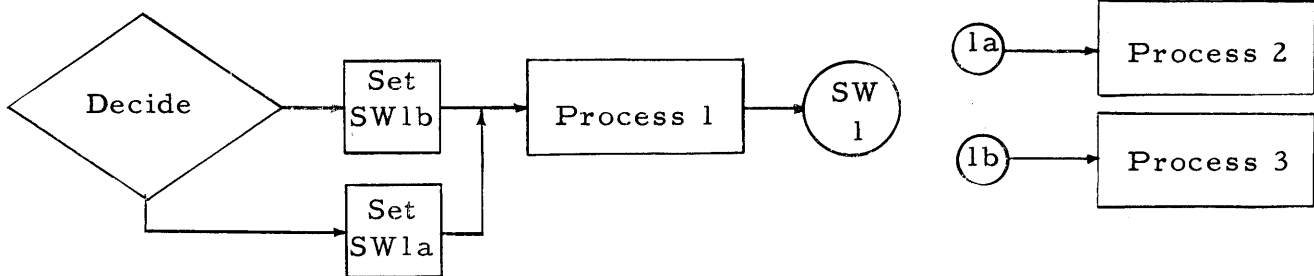
The following sets of coding would then replace BACK in the first example:

BACK	CLA	COUNT
	SUB	ONE
	TRZ	EXIT
	STR	COUNT
	TRU	LOC1
BACK	CLA	ITEM
	SUB	SENTL
	TRZ	EXIT
	TRU	LOC1

If this set of coding is to be performed again, the addresses of instructions which refer to the data must be reinitialized to their original values. This can be accomplished by subtracting a number to reduce each instruction address to the original amount (CLA, SUB, STR), replacing each word with a constant with the correct address (CLA, STR), or by replacing the addresses with the correct values (RPA). Since the RPA instruction does not alter the contents of the A register, one method of reinitializing the loop is to place a constant whose α field equals the address of the first word of the first item, replace the address into all locations which required this address, increment the α portion of the A register by one and replace the addresses of locations requiring this address, and repeat the cycle of incrementing and replacing the address until all addresses have been restored.

The process of altering program addresses can be extended to other uses, such as controlling the program. Alternative methods of processing may be desired, depending upon the results of a comparison. Frequently, some intervening and common processing is required between the comparison and the selection of alternate methods, and the results of the decision must be

remembered. The decision is recorded by setting a "switch." A switch is a transfer of control instruction whose α portion is altered by a preceding series of events. A switch is said to be "set" when a specific address is inserted in the α portion. A switch is shown in flow chart symbology as a circle:  The setting of a switch is shown in a square: 



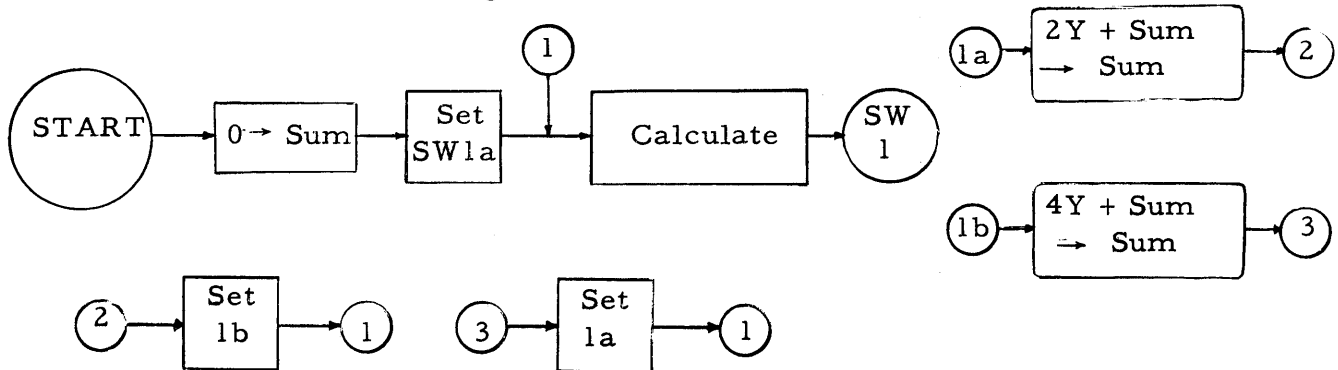
The instructions included in the box PROCESS 1 will always be performed. Either PROCESS 2 or PROCESS 3 would then be performed, depending upon the decision made at DECIDE.

A switch can have any number of alternative paths.

Example:

The following equation is to be evaluated for calculated values of Y:

$$Z = 2Y_1 + 4Y_2 + 2Y_3 + \dots$$



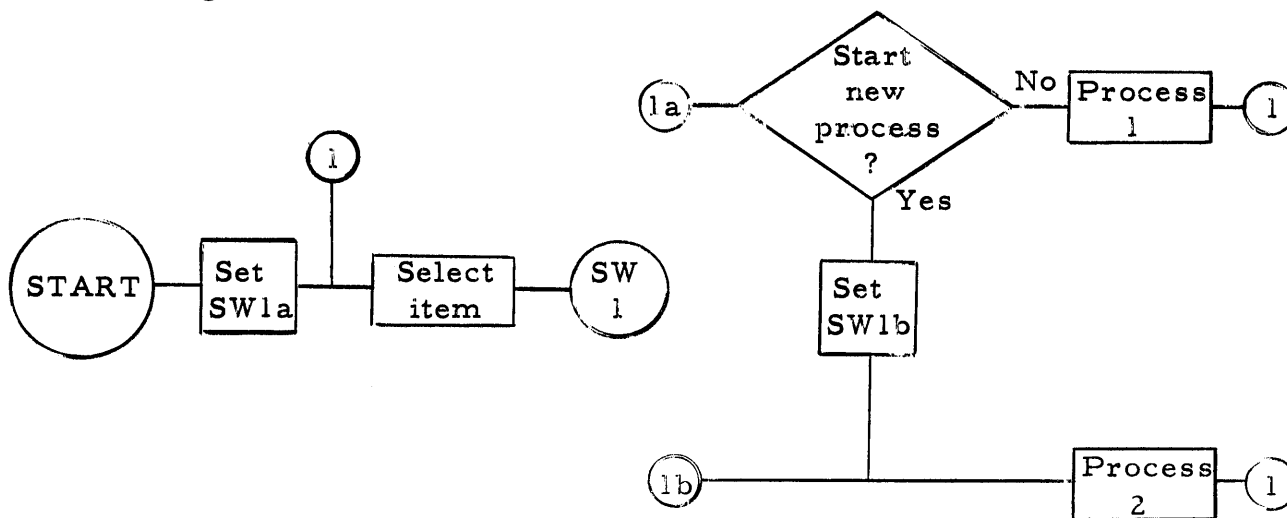
Note that the α portion of location SW1 is SW1; i.e., α will be replaced with the corresponding address. Should the programmer forget to set the switch, this instruction would be repeated indefinitely to indicate the cause.

$$I = \sum_{i=0}^n 2Y_{26ii} + 4Y_{2i}$$

α
 $\pi \beta =$

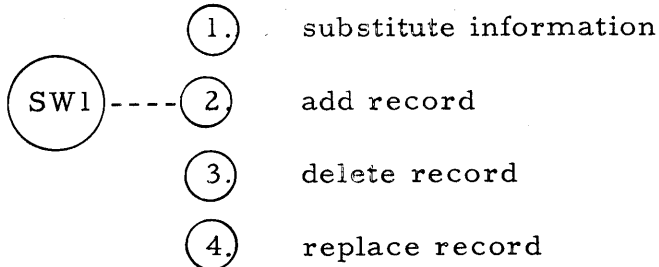
LOCATION	OP	γ	β	α	REMARKS
START	CLA			70 000	$0 \rightarrow A$ (illegal address)
	STR			SUM	$(A) \rightarrow \text{SUM}$
	LOD		A	CON1	
	RPA			SW1	Set switch 1A
CALC	compute Y
	LOD		A	Y	$Y \rightarrow A$
SW1	TRU			SW1	Switch 1
SW1A	SHL			1	$A \times 2 \rightarrow A$
	ADD			SUM	$A + \text{SUM}$
	STR			SUM	$\rightarrow \text{SUM}$
	LOD		A	CON2	
	RPA			SW1	set switch 1B
	TRU			CALC	
SW1B	SHL			2	$A \times 4 \rightarrow A$
	ADD			SUM	$Z + \text{SUM}$
	STR			SUM	$\rightarrow \text{SUM}$
	LOD		A	CON1	
	RPA			SW1	set switch 1A
	TRU			CALC	
CON1				SW1A	constant
CON2				SW1B	constant
SUM					
Y					

A switch can be used to "eliminate" part of a program after it is no longer needed.

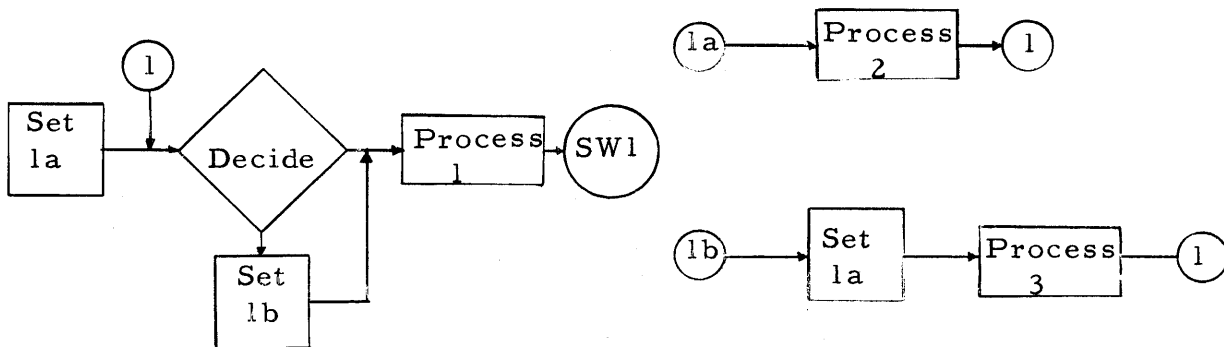


A switch can be used to alternate methods of processing.

Example: A code in a transaction record may indicate any additional processing required.



Another method assumes one path will be used more frequently.



K. INDEX REGISTERS

Two of the most frequently performed functions in coding are counting and address modification. These activities, performed as described in the preceding sections, are both awkward and space consuming. The use of index registers permits a more efficient procedure.

An index register is a twelve-bit addressable register with three special properties:

1. It can be used in addition and subtraction.
2. It can add its contents to the twelve low-order bits of the α portion of the instruction register.
3. The contents of the index register and the contents of the memory location containing the incremented instruction are not altered.

Incrementing the address portion of an instruction by the contents of an index register is called index modification. Instructions which can be altered are said to be indexable.

The sum of the α portion of an instruction and the contents of an index register ($\alpha + (I^\gamma)$) is called the effective address of the instruction. The effective address is either an actual address or a number such as the number of shifts to be performed. If no index register is specified, then the effective address of an instruction is the α portion alone.

There are four index registers included in BASICPAC, although logical provision has been made for seven. Index registers are specified as I^1, I^2, \dots or, more generally, as I^γ . The γ portion of an instruction word specifies the index register to be used.

Consider the instruction:

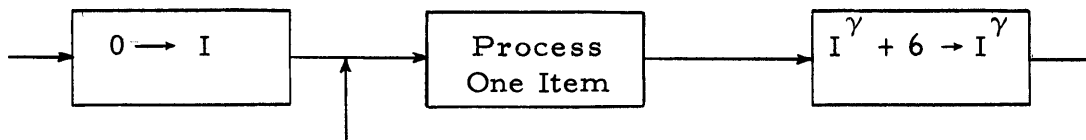
OP	γ	β	α
CLA	2		1432

This instruction has the effective address $1432 + (I^2)$. If, for example, $(I^2) = 32$, the effective address is $1432 + 32 = 1464$, and the above instruction is equivalent to

CLA 1464

Similarly, if $(I^2) = 2000$, the effective address is 3432; if $(I^2) = 0$, 1432; if 201, 1633.

Consider the example in Section XI-J. If the instructions which refer to addresses of data were index modified, the index register could be incremented by six after processing each item, and the series of CLA, ADD, STR instructions to modify each instruction could be omitted. The flow chart would read



The instructions can then be performed as follows:

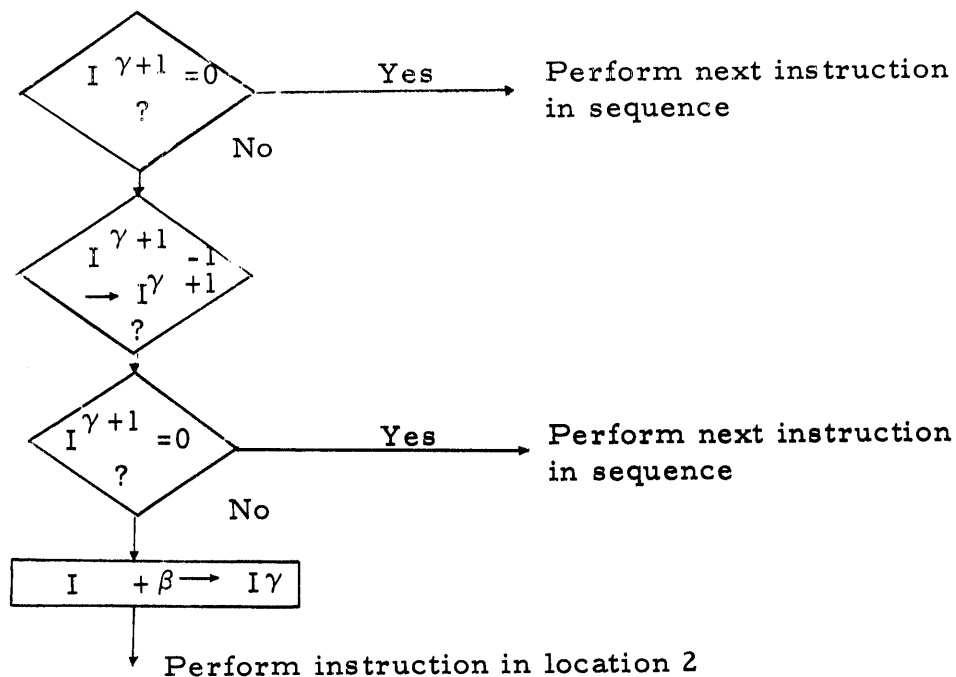
The first time the loop is performed, $(I^\gamma) = 0$, and the effective address for $L\phi C1$ is $2000 + 0 = 2000$; the first item is processed. I^γ is then incremented to equal 6. The effective address this time is $2000 + (I^\gamma) = 2006$, which refers to the second item. The next time, $I^\gamma = 12$, and $2000 + (I^\gamma) = 2012$. Similarly, $L\phi C2$ has an effective address of 2003, 2009, 2015, etc.

Before processing of this type can begin, the contents of the index registers must be set to the desired initial values. The instructions which set and alter the contents of index registers are called index register instructions.

The Load Index instruction (LDX, 53) sets the contents of two index registers to any desired value. The γ portion of the instruction specifies the first index register to be used (I^γ). The actual numbers in the β portion of the LDX instruction are placed in I^γ , and the actual numbers in the α portion are placed in $I^{\gamma+1}$. If there are only four index registers, and $\gamma = 4$, then $\gamma + 1 = 1$.

For example, the instruction LDX 2 200 7007 sets I^2 to 200 and I^3 to 7007.

The Transfer on Index instruction (TRX, 43) is a conditional transfer instruction which depends upon the contents of a specified pair of index registers. The γ portion of the instruction specifies the index registers to be used (I^γ and $I^{\gamma+1}$). The operation of the TRX instruction is shown in the following flow chart:



The TRX instruction can be used both to count and to modify addresses by a predetermined amount. The number in the β portion of a word is added to I^γ . $I^{\gamma+1}$ can be decreased only by one.

For example, if only 400 items are to be processed, the coding in the preceding illustration would read:

LOCATION	OP	γ	β	α	REMARKS
LØC1	LDX	1	0000	4	$0 \rightarrow I^1, 4 \rightarrow I^2$
	CLA	1		2000	$2000+(I^1) \rightarrow A$
LØC2
	ADD	1		2003	$(A) + (2003+(I^1)) \rightarrow A$
LØCN
	STR	1		2005	$(A) \rightarrow 2005+(I^1)$
NEXT	TRX	1	6	LØC1	All items processed yes:

The first time the loop is performed, $I^1 = 0$, $I^2 = 4$. Then the TRX instruction decreases I^2 by 1, increments I^1 by 6, and transfers control to location LØC1. I^1 now = 6, $I^2 = 3$. The next time, $I^1=12$, $I^2=2$, again $I^1 = 18$, $I^2 = 1$. After the processing has been completed, I^2 will be tested for zero and be decreased by 1. I^2 now = 0, so the next instruction in sequence will be performed (location NEXT).

Index registers are cyclic in nature. That is, addition and subtraction are performed modulo $10,000_8$. If $(I^1) = 0132$, and the instruction

OP	γ	β	α
TRX	1	7776	LOOP

is performed, (I^1) will be 0130_8 since $0132 + 7776 = 10130 \equiv 0130 \pmod{10,000_8}$. This fact can be used to decrease index registers by any number during an index modified loop. If the index register is to be decreased by n (where n is expressed in octal), set the β portion of the TRX instruction to $10,000_8 - n$.

Index registers can also be addressed by the α portion of all instructions except the Transfer of Control instructions. The contents of the index register are placed in the low-order 12 bits of the computer word and the remaining bits are set to zero. The sign is always positive. When transferring information from a memory location or larger register into an index register, only the low-order 12 bits are used.

For example:

Assume (A) = -123 456 701 234

(Q) = +000 100 010 001

(I¹) = 1234

OP	γ	β	α	REMARKS
CLA			I ¹	A = +000 000 001 234
ADD			A	A = +000 100 011 235
STR			I ¹	I ¹ = 1235

L. SUBROUTINES

A program or routine is defined as a series of instructions arranged in the sequence necessary to perform a major function. A subroutine is defined as a part of a routine which performs a specific function within the routine. The use of subroutines in flow charting and coding significantly reduces programming time and effort, program testing time, and computer time. Moreover, the use of subroutines enables the programmer to concentrate upon the major processing path of a routine by deferring the programming of minor functions until a more convenient time. A subroutine which is used in many programs need only be coded once. Whenever the function performed by the subroutine is required in a new program it is merely copied into the program. An accumulation of such frequently-used subroutines is called a subroutine library.

A desired end in programming is to code a subroutine once, to transfer control to it whenever its function is to be performed, and return control to the proper place in the program when the function has been completed. A special register called the Program Counter Store register (PCS) is used to keep track of these return addresses.

Two transfer of control instructions are used with PCS to enter and return from subroutines. These instructions are Transfer and Load PCS, and Transfer on PCS.

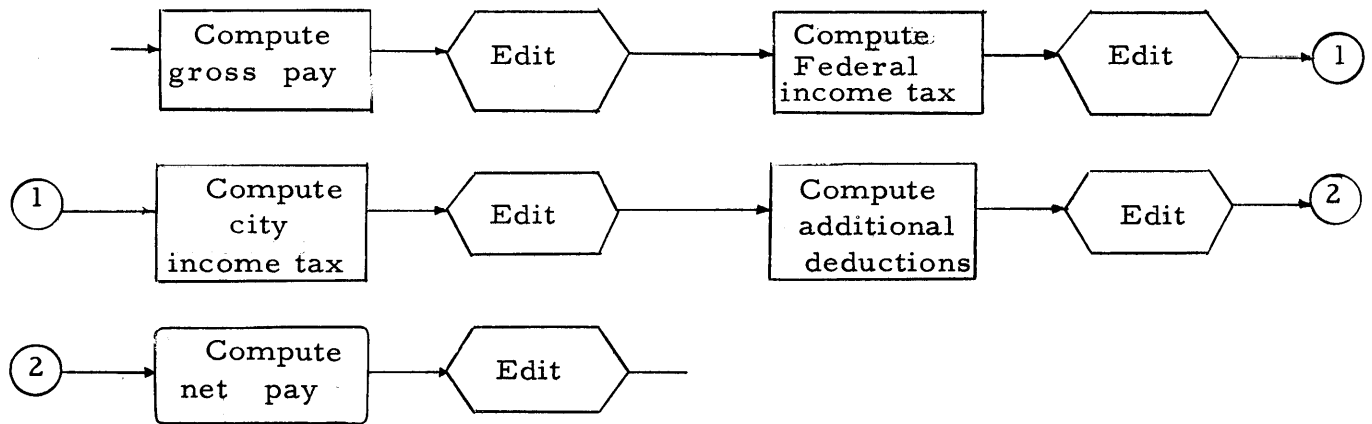
When a Transfer and Load PCS instruction (TRL, 41) is issued, the contents of PC are loaded into PCS, the index register specified (if any) is loaded with the β portion of the instruction word, and PC is loaded with the α portion. Since PC was incremented as soon as the TRL instruction was

obtained from memory, PCS now contains the address of the first instruction consecutive to the TRL. The TRL instruction can be considered an unconditional transfer of control instruction.

The Transfer on PCS instruction (TRS, 42) automatically transfers the contents of PCS to PC, and transfers control to that address. TRS may be considered a special type of TRU instruction in which the contents of PCS rather than the α portion of the instruction word specifies the "go to" address.

Consider the following flow chart:

EDIT is the first address of a subroutine which prepares a word for output. This subroutine is coded only once, and is entered from a number of places in the routine. A subroutine is depicted by a hexagonal figure in flowchart symbology.



The entrances to and exits from the EDIT subroutine might be coded as follows:

LOCATION	OP	γ	β	α	REMARKS
GROSPA	CLA				compute
	gross pay
FIT	LOD		A	AMOUNT	edit result
	TRL			EDIT	compute
CIT	LOD	..	A	FI TAX	Federal Income Tax
	TRL	.		EDIT	edit result
DEDUK	LOD	.	A	CI TAX	compute
	TRL	.		EDIT	city income tax
NETPA	LOD	..	A	SUM	edit result
	TRL	.		EDIT	compute
EDIT	LOD	.	A	NET	additional deductions
	TRL			EDIT	compute
EDIT	TRU			CONTIN	net pay
	STR			WORD	edit
	TRS		edit the (A)
					return control

Note that the word to be edited is placed in the A register before transferring control to EDIT. All information or parameters required by a subroutine must be presented in the method required by the subroutine. An alternative approach consists of loading an index register with the address of the word to be edited. Or the α portion of the A register can contain the address of the first word to be edited and an index register filled with the number of consecutive words to be edited.

The method by which parameters are supplied to a subroutine is called the calling sequence of the subroutine.

It is occasionally convenient to have a number of exits from a subroutine (such as error exits) each of which requires different processing. One technique of providing such exits is to place in sequence after the TRL a series

of unconditional transfers to various locations. The contents of PCS can then be augmented to obtain the required exit. Or, the contents of PCS can be inserted in an index-modified TRU. The index register is loaded with the constant required to produce the correct address.

N. INTERRUPT SUBROUTINES (See also Section XII)

All interrupt subroutines perform three functions:

1. Determine the cause of interrupt.
2. Perform processing as required by the problem.
3. Return control to the main program.

The first step upon entering an interrupt subroutine usually consists of saving the contents of any registers required for the subroutine. The MLY, DVD, DVL, MSK, LGA and LGM instructions should be avoided, since they alter the contents of the B register.

The next step is to obtain the return address from the B register. The α portion of the B register contains the address of the instruction about to be performed when the interrupt occurred. The B register can be addressed only by the α portion of a LOD instruction.

When the return address has been obtained, the reason for interrupt can be determined. It will be assumed that only one Input-Output Converter is present. If there is more than one converter, the coding is simply duplicated with appropriate changes in addresses.

Consider first an F1 subroutine. Possible reasons for an F1 interrupt are:

1. Addressing a non-existent converter.
2. Addressing a busy converter.
3. Issuing an improper order.
4. Initial device malfunction.

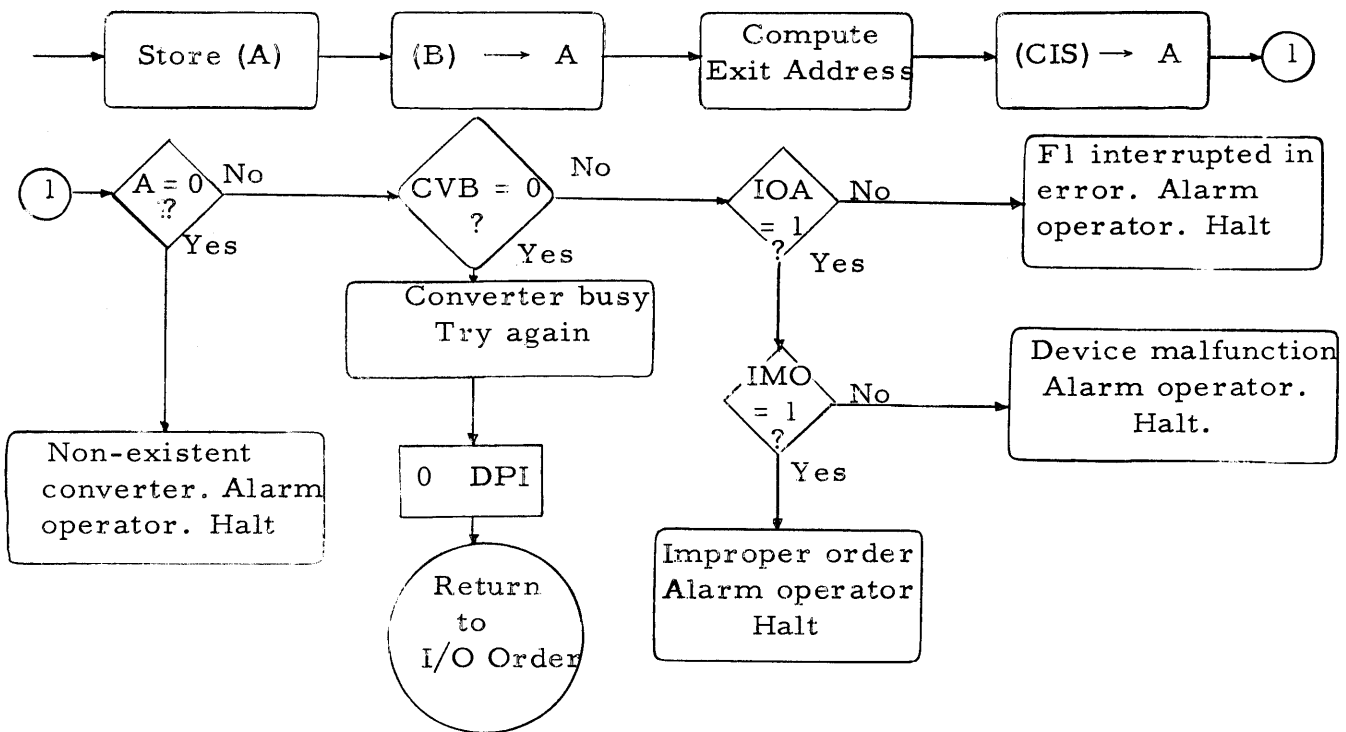
Condition 1 can be determined by checking for non-zero contents of CIS. Conditions 2, 3, and 4 can be determined by interrogating the corresponding addressable flipflops: Converter Busy (CVB), Improper Order (IMO) and Device Alarm (DVA). The Input-Output Alarm flip flop (IOA) will be set if any of the following flipflops are at 1:

CMPE
 IOPE
 DVA
 DDA
 IMO

See Appendix D, Input-Output Converter Flipflop Addresses.

If more than one converter is present, the interrupt subroutine should be supplied with the number of the converter addressed. In certain cases, separate interrupt routines for each converter may require fewer memory locations than one "all-purpose" interrupt routine.

An example of an F1 subroutine flow chart follows:



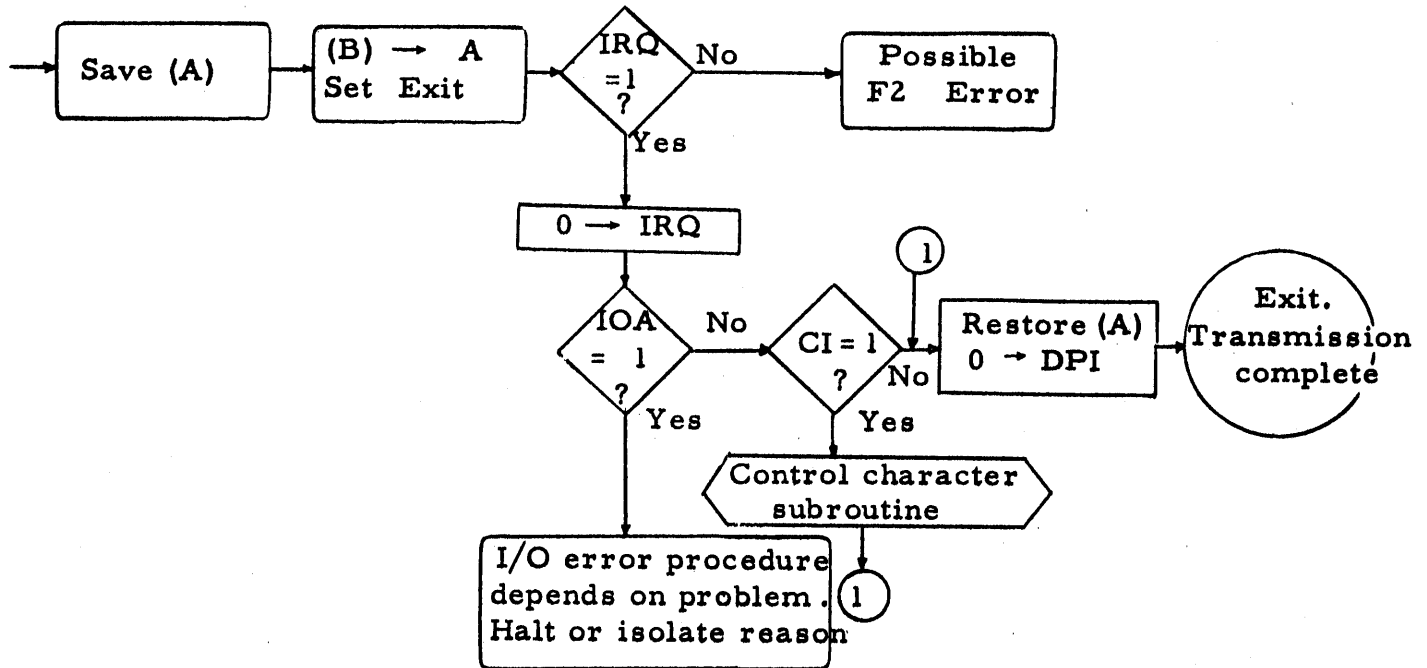
Note that the above flow chart does not require the device address. The return to the I/O order must be preceded by the resetting of DPI to permit future interrupts. However, the return address could be placed in the α portion of an SNR DPI instruction.

An F2 interrupt routine can be as simple or complex as the problem requires. Possible reasons for an F2 interrupt are:

1. Completion of an order (normal case).
2. Error condition.
3. Reception of a control character in interrupt control mode.

These conditions can be determined by interrogating the corresponding addressable flipflops.

A flow chart for an F2 interrupt routine follows:



The IRQ flipflop must be reset before DPI is reset, or an interrupt will be requested for the same converter each time DPI is reset. If more than one converter is present, the IRQ for each converter is interrogated. If none of these are at 1, and if there is no Communications Converter in the system, the F2 jump has been made incorrectly.

When the IRQ of the converter requesting the interrupt has been found the "YES" path of "IRQ=1?" is followed, using the correct addresses for the corresponding flipflops of the converter.

XII. INPUT-OUTPUT PROGRAMMING

A. CONTROL UNIT INPUT-OUTPUT EQUIPMENT

The preceding sections assumed that both the program and the data required for the program were present in memory. However, it is always necessary to initially load the program and frequently necessary to obtain or record information while a program is running. The process of entering information into memory and of obtaining information from memory is called input-output.

BASICPAC input-output devices include paper tape equipment, magnetic tape units, and real-time communication channels. These devices are controlled by three types of equipment:

1. Control Unit
2. Input-Output Converter
3. Communications Converter

This section discusses the operation and programming of the control unit input-output equipment. Section B discusses the Input-Output Converter devices, and Section C discusses the use of the Communications Converter.

The Control Unit input-output equipment consists of a paper tape reader, a paper tape punch, and a FIELDATA typewriter. The paper tape reader is used to enter information into the computer memory. The paper tape punch and the FIELDATA typewriter are used to obtain information from the computer memory. All three devices can be controlled by the program through the input-output orders. The format of the BASICPAC input-output instruction word is shown in Section III A 26.

A list of input-output device addresses is given in Appendix C.

There are two methods of transmitting information: octal mode and alphanumeric mode.

In octal mode, thirteen FIELDATA characters form one computer word. Twelve data characters and one sign character correspond to twelve octal digits and one sign bit. When writing or punching information, the computer automatically translates each octal digit into the equivalent numeral by prefixing to the three bits from the computer word the fixed bit pattern 1110. (See Appendix D, Binary Codes for FIELDATA Characters.) The sign bit is translated into the corresponding FIELDATA characters "+" or "-". When

reading information, only the least significant bit of the sign character and the three least significant bits of the twelve data characters are used in assembling the word to be placed in memory. In both cases the sign bit is processed first, then the data characters from the most significant to the least significant positions.

The Read Octal instruction (ROK, 72) reads the number of words specified by k from the device specified by j into consecutive memory locations beginning at address α . The address of the paper tape reader is 01_8 .

The Write Octal instruction (WOK, 76) writes k words from consecutive memory locations beginning at address α on device j . The address of the paper tape punch is 02_8 ; the address of the typewriter is 03_8 . A stop character (57_8) is automatically punched at the end of each order.

In alphanumeric mode six FIELDATA characters are transmitted, corresponding to the 36 data bits of a computer word. Each set of six data bits forms one FIELDATA character. The treatment of the sign bit depends upon the state of the addressable Interpret Sign flipflop (ISN) when the input-output order is issued. If $ISN = 1$, the sign bit forms a seventh character. If $ISN = 0$, no sign character is generated for output, and the sign of all input information is automatically made positive.

A second addressable flipflop which affects the alphanumeric mode of operation is the Interpret Control Function flipflop (ICF). When $ICF = 0$, alphanumeric information is processed as described above. When $ICF = 1$, all information is punched as control characters. When reading information, if $ICF = 1$ and the first character read is a control character, this character is placed in the low-order character position of a word, the high order bits are cleared to zero, the word is stored in memory and transmission ceases. If $ICF = 1$ and the first character read is a data character, the entire order is processed. If both $ISN = 1$ and $ICF = 1$, ISN takes precedence.

The Read Alphanumeric instruction (RAN, 70) reads k words from device j into consecutive memory locations beginning at address α .

The Write Alphanumeric instruction (WAN, 76) writes k words on device j from consecutive memory locations beginning at address α .

Care must be taken that information is read in the same mode in which it was punched.

When reading in any mode described above, the computer continues processing until either k words have been read or a stop character is read. In the latter case, the stop character is placed in its correct position within

the word being assembled, the remainder of the word is cleared to zero, the word is stored in memory, and transmission ceases. Upon completion of any I/O instruction except those involving the Communications Converter the ICF and ISN flipflops are automatically reset.

When punching in any mode, stop characters are treated as any other character.

The typewriter neither stops processing nor prints a character when the stop code is transmitted.

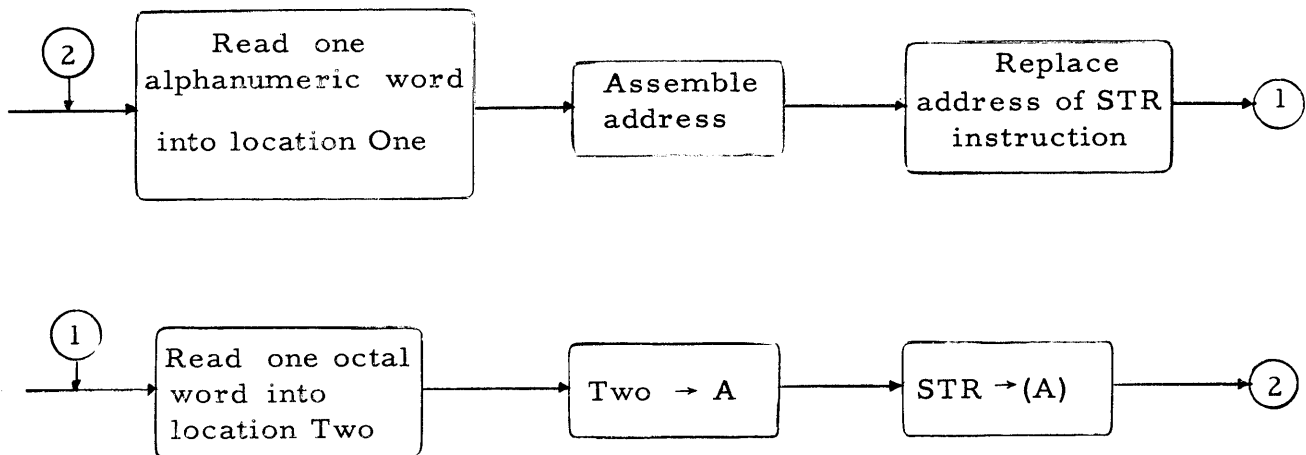
The computer halts if any error occurs during control unit input-output, except in the case of a parity error when the Error Override switch on the control panel is "ON". In this case, the Control Parity Error flipflop (CPE) is set to one and processing continues. CPE can be addressed by any of the Sense instructions.

Example:

When debugging programs, the programmer usually finds that a number of memory locations must be changed. The following routine reads a paper tape which contains a list of corrections in the following format:

C_r XXXXXX \pm YYYYYYYYYYYY

Where X - X is the address in octal FIELDATA characters of the location to be corrected and Y - Y are the desired contents of that location. The program coded below reads in the list of corrections and stores them in the desired memory locations. The flow chart is:



LOCATION	OP	k		j	α	REMARKS
		γ	β			
START	SNR			ISN	NEXT	0 → ISN
NEXT	SNR			ICF	READ	0 → ICF
READ	RAN	0	01	01	ONE	READ one word → ONE
	LDX	1		0	5	0 → I ¹ , 5 → I ²
	LOD			A	ONE	(ONE) → A
LØØP	SRL				3	generate octal address
	SHR				3	in bits 22-36 of Q
	TRX	1		0	LØØP	finished ?
	SLL				17	yes address → A _{α}
	RPA				STØRE	replace address
	ROK	0	01	01	TWØ	read one octal word → TWØ
	LOD			A	TWO	(TWØ) → A
STØRE	STR				()	correct specified location
	TRU				START	
ONE	()	storage for address
TWO	()	storage for word

B. INPUT-OUTPUT CONVERTER

Unlike the Control Unit, the Input-Output Converter does not inhibit central processor operations. The computer continues under program control until either an error is detected or until the input-output order is completed. The normal computer operation is then interrupted.

An interrupt can occur only if the Disable Program Interrupt flipflop (DPI) is at zero and if the computer is obtaining a new instruction word (between instructions). At this time the contents of PC equal the address of the instruction about to be performed. Instead of obtaining the instruction the computer automatically stores the contents of PC in the α portion of the B register, sets DPI to one, and transfers control to a special memory location.

The location to which control is transferred depends upon the reason for interrupt. If an error was detected prior to acceptance of an input-output instruction, control is transferred to memory location 00001. If an error was detected after acceptance of the instruction, or if an order has been completed, control is transferred to memory location 00002. The first type of interrupt is called an F1 interrupt; the second, F2. ("F1" and "F2" are used to refer both to the type of interrupt and to the memory location corresponding to the interrupt.)

The instructions at locations F1 and F2 should be unconditional transfers to subroutines which will determine the cause of the interrupt, perform any required processing, reset DPI to permit new interrupts, and return control to the main program.

Since the actual operation of these subroutines will depend heavily upon the current program, subroutines should be written specially for each program. The unconditional transfer instructions at F1 and F2 must be set by the program before any input-output orders are issued, otherwise the programmer may lose control of his program when an interrupt occurs. (A general discussion of the procedure in writing interrupt subroutines is included at the end of this section.)

A set of addressable flipflops in each converter are used to indicate the existence of errors to the program. The programmer can decide upon the course of action to be taken by using Sense instructions to determine the status of these flipflops. Appendix B lists the addressable error flipflops for all Input Output Converters.

If the coding which follows an input-output order requires the order to be completed before the subsequent coding is executed, the programmer must allow time for the input-output to be completed. Alternately, the programmer can have the program "wait" for the completion of input-output by entering a time delay loop immediately after issuing an input-output order. Completion of input-output interrupts this loop and allows the program to continue.

For example, the following coding issues the input-output order at READ and enters the loop at LOOP. Assuming an F2 interrupt, control is transferred from LOOP to 00002 to F2, the interrupt subroutine. In this routine the contents of A are stored and the return address is computed. Note the negative sign of the input-output order. When the negative instruction is encountered the α portion of location FIND will be READ. At this point the constant TWO is added to the address to produce the machine address of NEXT, the return address. The cause of the interrupt is then determined. Any required processing is then performed, and EXIT transfers control to the main program. The original contents of A should be restored before exiting from the subroutine. This method of processing interrupts assumes that the signs of all input-output orders are negative.

LOCATION	OP	k		j	α	REMARKS
		γ				
00001	TRU				F1	
00002	TRU				F2	
.....						
READ	RAN	0	20	20	INPUT	
LOOP	TRU				LOOP	
NEXT	CLA	2			INPUT	
.....						
TWO	00	0	0000	00002		
ONE	00	0	0000	00001		
F2	STR			STOR A		store (A)
	LOD		A	B		(B) \rightarrow A
	RPA			FIND		(B $_{\alpha}$) \rightarrow FIND α
FIND	CLA			(00000)		
	TRN			OUT		I/O word?
	CLA			FIND		no
	SUB			ONE		try next word
	STR			FIND		
	TRU			FIND		
OUT	CLA			FIND		store return
	ADD			TWO		
	RPA			EXIT		address in EXIT $_{\alpha}$
.....						
	LOD		A	STORA		
	TRU			(00000)		return to program

When an input-output order is issued to an Input-Output Converter device, the converter, the device, and the order are each examined for errors. If the converter is non-existent or busy, if the device is non-existent or busy, or if the order requests a device to do something for which it is not suited an F1 interrupt occurs. If no error condition is present the order is accepted and processing begins.

The accepted order is placed in the Converter Instruction Register (CIS) for execution and the central processor continues operation. As each word is processed, the α portion of CIS is incremented by one and the K portion is decremented by one. When the K portion equals zero the order has been completed. If an error condition is detected during processing, if a control character is received in interpret control mode, or if transmission is completed, the Interrupt Request flipflop (IRQ) on the converter is set to

"one" to indicate to the central processor that an F2 interrupt is being requested. In this context a Stop character is considered a control character regardless of the mode in which it was written.

There is a separate Converter Instruction Register for each Input Output Converter. These registers can be addressed by the LOD, LGA, LGM and arithmetic instructions. The programmer can examine these registers in the interrupt subroutines to obtain information concerning the number of words processed, the last address processed, etc. The CIS of a non-existent converter is an illegal address and therefore appears to have positive zero as its contents.

Paper Tape Equipment

The paper tape set connected to the I/O Converter is physically and functionally identical to the Control Unit paper tape set. The only difference in operation is the interrupt feature. All input-output orders which can be issued to the Control Unit paper tape set can be accepted by the Input-Output Converter paper tape set, assuming correct device addresses. The octal addresses for Input Output Converter #1 paper tape equipment are:

<u>Device</u>	<u>Address (octal)</u>
Paper tape reader	20
Paper tape punch	22
FIELDATA typewriter	26

Magnetic Tape Equipment

Magnetic tape equipment and programming techniques are discussed in U.S. Army documents SCL-1882A and SCL-1886.

C. COMMUNICATIONS CONVERTER

There are no fixed procedures to be followed in programming the Communications Converter. The program prepares for transmission, and transmission occurs whenever both the sending and receiving devices are available. A list of the conditions which must be fulfilled and a few rules of thumb are listed below for input and output.

Input

If there is a limited interrupt channel, KIW should be set at the beginning of the program and reset as required after each input interrupt.

Input can occur when DPI equals one, but no interrupt will occur until DPI is reset.

A change of the state of KIU^i from 0 to 1 indicates that the computer is ready for input. This change in state must be programmed by using the SNR and SNS instruction pair. There is a KIU flipflop for each input channel.

Each KIU^i has two addresses, either of which can be used interchangeably.

When it has been determined that an input caused an interrupt, the contents of memory location 00004 should be saved along with the input words (both data and control) before the input request flipflop KAI is reset. If KAI is reset before the input information is stored or processed, new information may be received which would destroy the first set of information, even though DPI is at one.

On limited interrupts, remember that the address with which KIW is loaded is the first address used, and that the number of words received equals (last address) - (first address) + 1. For example, if KIW is set to 5372_8 , input will continue to location 5777_8 , and up to $5777 - 5372 + 1 = 0405 + 1 = 0406_8$ words can be received before an interrupt will occur.

Output

Use a LOD instruction to load KOU^i with the word to be transmitted. Note that this instruction can be index modified.

Only one word can be transmitted at a time. If a message of more than one word in length is to be transmitted, set an index register to count the number of words to be transmitted and use a TRX instruction to loop over the transmission instructions. Be careful to return control to the TRX instruction after each output interrupt has been processed.

General

F2 must be set to transfer control to the interrupt subroutine.

ICF and ISN need be set to the desired states only once at the beginning of the program. However, if any input-output orders are issued either to the Input-Output Converter or to the Control Unit devices, these flipflops are reset to zero when each input-output order is accepted.

If information is to be transmitted in more than one mode (ISN, ICF), set DPI to one before setting the required flipflops and loading any required registers. Reset DPI to enable transmission when the conditions have been prepared.

The KIU flipflops for non-existent or unconnected channels will always appear to be in the zero state, contrary to the rule for illegitimate flipflop addresses. The following coding is one example of programming output on the Communications Converter. It assumes the existence of a subroutine beginning at GENER which places 64 words of output information in the consecutive memory locations beginning at OUT.

LOCATION	OP	γ	β	α	REMARKS
START	TRL	2	0100	GENER	generate 100_8 words of output. $0 \rightarrow I^1$, $100 \rightarrow I^2$
	LDX	1	0000	00100	
BEGIN	SNS		DPI	NEXT	
NEXT	SNR		ISN	NEXT1	$0 \rightarrow$ ISN
NEXT1	SNR		ICF	NEXT2	$0 \rightarrow$ ICF
NEXT2	SNR		KEI	NEXT3	$0 \rightarrow$ KEI
NEXT3	LOD	1	KOB ^o	OUT	output word \rightarrow KOB ^o
	SNR		DPI	NEXT4	$0 \rightarrow$ DPI
NEXT4	TRU			NEXT4	wait for interrupt
	TRX	1	1	BEGIN	all words transmitted? yes continue
00002	TRU			INTER	
INTER	TRL			ADDRS	obtain address of negative instruction
	ADD			ONE	add one
	RPA			EXIT	set exit address
	SEN		KAI	INPUT	input interrupt?
	SEN		KEI	JUMP	no: output interrupt?
	TRU			IOC	no: to I/O converter test
JUMP	CLA			00003	memory location 3 A
	TRN			PARER	parity error?
	SNR		KEI	EXIT	no: return
EXIT	SNR		DPI	(00000)	$0 \rightarrow$ DPI

The following coding is one example of preparing for input on the Communications Converter. The coding at locations NEXT1 and NEXT2 must be duplicated with corresponding address changes for the KIU flipflops of each channel from which input is expected. It is assumed that channel 10 is a limited interrupt channel, and that all data messages will contain 128 (200_g) words. These initialization procedures must be repeated after each input interrupt.

LOCATION	OP	γ	β	α	REMARKS
START	LOD		A	CON1	
	STR			00002	set F2 jump
	SNS		ISN	NEXT	1 → ISN
NEXT	SNR		ICF	NEXT1	0 → ICF
NEXT1	SNR		KIU ¹	NEXT2	prepare for input
NEXT2	SNS		KIU ²	NEXT3	on channel 11
NEXT3	LOD		KIW	CON2	prepare for
NEXT4	SNR		KIU ⁰	NEXT5	input on
NEXT5	SNS		KIU ⁰	NEXT6	channel 10
NEXT6	process
CON1	TRU			EF TWO	
CON2	00	0	0000	07600	expect 200 _g words input
EF TWO	interrupt routine

The coding methods used for input-output through either the Input-Output Converter or the Communications Converter are closely related to the coding methods used in the interrupt subroutines. Both methods should be carefully planned before any coding is written, and the conventions selected must be consistently followed.

There are as many ways of coding interrupt subroutines as there are programmers. However, three general categories of interrupt routines exist: the special purpose, the general executive, and the all-in-one.

The special purpose subroutine assumes only one type of converter will be used. It is known in advance whether certain registers (A, Q, PCS, I ^{γ} , etc.) will always be immediately used or whether their contents must be stored before using. Only certain information concerning the interrupts is desired, and the routine transfers control directly to the appropriate routine. It is an integral part of the program.

In contrast, the general executive routine may be a library subroutine. It merely collects data concerning the interrupt and returns control to the main program for processing an additional order.

The third type of subroutine is a combination of the other two. It is designed for the system being used, which may include more than one converter or more than one type of converter. It determines the cause of interrupt, checks for errors, processes incoming information, determines where to transfer control and does so. Again it is an integral part of the program, but it performs many more of the interrupt functions than the special purpose subroutines.

XIII. DEBUGGING METHODS

Very few programs run correctly on the first try. Coding errors, tape preparation errors, and, worst of all, logical errors tend to creep into the best-written program like ants into a sugar bin. The detection and correction of these errors, or "bugs", is called debugging.

There are as many ways of debugging a program as there are of coding it. One or more methods may be used at any time in debugging a program, and other programs can be used to help in debugging. This section discusses some of the more widely used methods.

1. Optimist's Method

Read the program into memory and press the ADVANCE bar. This method, generally used by novice programmers on their first programs, usually results in a memory full of garbage.

2. Step Method

Read the program into memory and execute each instruction in Step Mode, checking all relevant registers and memory locations. This method is recommended only if programmer and computer time are available in unlimited quantities.

3. Dump Method

A dump is a program which edits the contents of specified memory locations for output to the FIELDATA typewriter or line printer. The amount of editing and the degree of sophistication vary widely. Generally, the starting address, the format (octal, alphanumeric, mnemonic, floating point, etc.) and either the number of words or the stopping address are specified. The dump can be operated under control

of either the program being debugged or of the operator. A method commonly used under operator control consists of storing a halt instruction at some intermediate point in the program, running the program to the halt instruction, and taking a dump of the results, program, or both.

4. Trace Method

A trace program executes the subject program one instruction at a time and prepares edited output of the contents of any affected registers and memory locations. Trace programs vary widely in scope. Certain types of instructions, instructions in a given address range, or any other set of instructions can be selected for tracing.

5. Trapping Method

This method uses a combination of programming and computer hardware to trace the action of the Transfer of Control instructions. A non-addressable flipflop called the Trapping flipflop (TRA) is used to indicate to the computer that Trapping mode is desired. When the computer is in Trapping mode, all instructions are executed normally except for the Transfer of Control and Sense instructions. When a Transfer of Control or Sense instruction is placed in the instruction register in Trapping mode, the contents of PC are stored in the α portion of the B register, the TRA flipflop is set to zero, and control is transferred to memory location 00000. The trapped instruction is not executed.

The instruction in memory location 00000 transfers control to a special routine which performs any desired processing. The α portion of the B register contains the address of the trapped instruction. Since the TRA flipflop was reset when the order was trapped, normal operation continues until the TRA flipflop is again set to one.

The trapping mode is controlled by bits 16 and 17 of the unconditional transfer instruction and by the previous state of the trapping flipflop as follows:

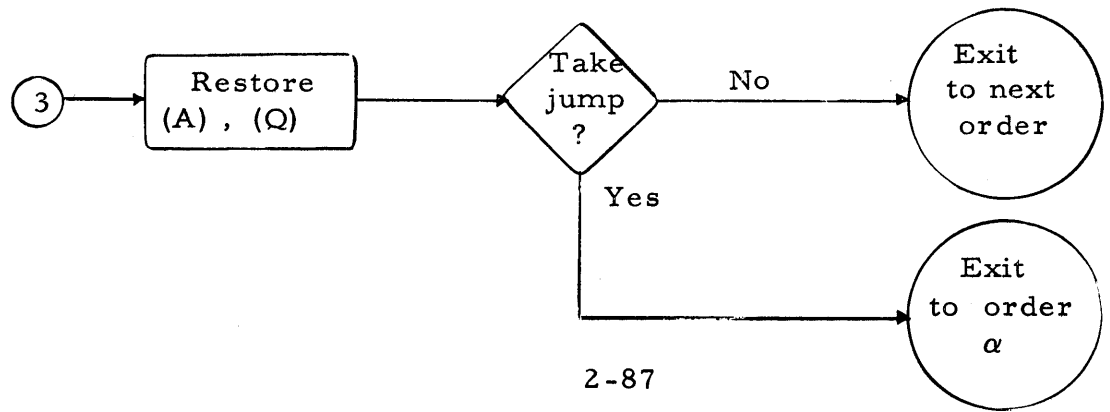
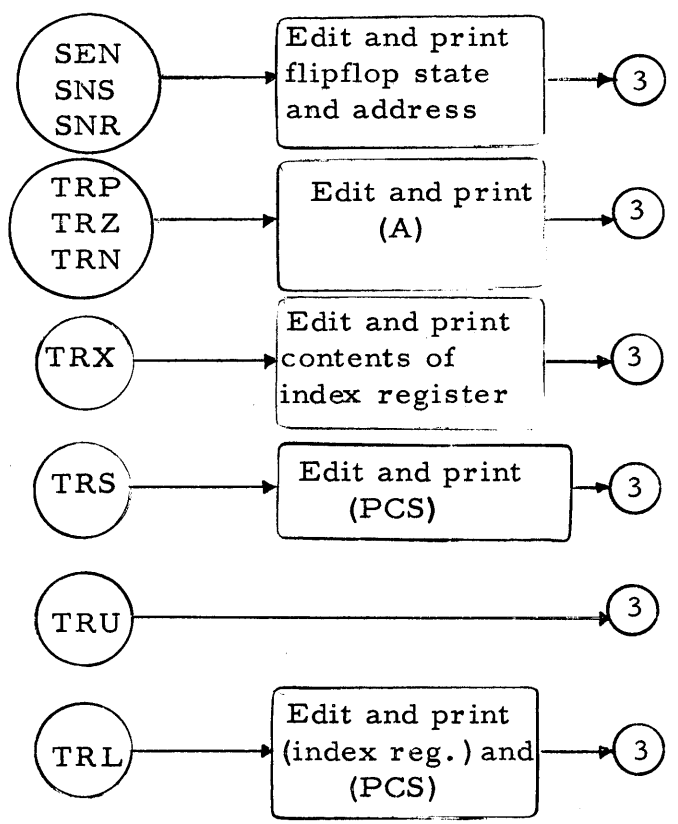
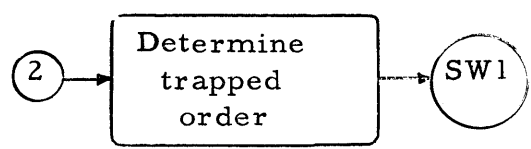
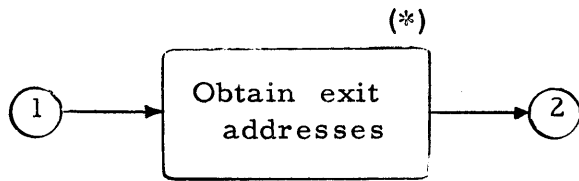
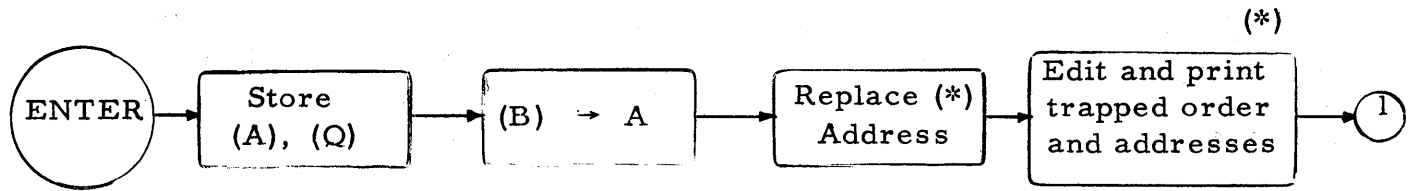
<u>TRA before TRU</u>	<u>β_{17} β_{16} of TRU</u>	<u>Trapping Action?</u>	<u>Effect on PC</u>	<u>Remarks</u>
0	00	no	No Action	No Action
0	01	no	No Action	1 \longrightarrow TRA
0	10	no	No Action	No Action
0	11	no	No Action	0 \longrightarrow TRA
1	00	yes	PC-1 \rightarrow B, 0 \rightarrow PC	Trap TRU order
1	01	no	No Action	1 \longrightarrow TRA
1	10	yes	PC-1 \rightarrow B, 0 \rightarrow PC	Trap TRU order
1	11	no	No Action	0 \longrightarrow TRA

Note that the TRA flipflop must be set to one after each instruction has been trapped.

A switch on the control panel labeled TRAP MODE acts as a trapping over-ride when in the OFF position. The operations described above will occur only when the TRAP MODE switch is in the ON position.

A flow chart for a sample trapping subroutine and a sample of coding for a program which uses the trapping routine are given below.

LOCATION	OP	γ	β	α	REMARKS
START	TRU CLA		1	START+1	Set TRA = 1
					} Process (No transfer of control instructions)
	SUB TRZ			CON 5 REDO	Instruction will be trapped.
REDO	SNS TRU		SFF3 1	REDO+1 REDO+2	TRA = 0 not trapped 1 \longrightarrow TRA } Process
	SEN		SFF3	DONE	Instruction will be trapped.
HERE	TRU TRZ TRU		1 2	HERE CALC FIX	} Both instructions will be trapped.
LOOP	TRU		3	LOOP	0 \longrightarrow TRA for LOOP
	TRX TRU	3	2 1	LOOP CONTIN	1 \longrightarrow TRA for program



Before preparing any program on an input-output medium for debugging, it is suggested that another person thoroughly examine the coding sheets and flow charts for errors. A surprisingly large number of errors can be detected in this way. This process is called "code checking". An extremely thorough method of code checking includes the simulation with paper and pencil of the effect of each instruction on all relevant registers. This process is called "bench checking".

A partial list of possible errors is given below. These errors do not include logical errors, which are the hardest of all errors to detect and correct. Although many items on this list may appear trivial they can cause complicated results.

1. Are all lines of coding consecutively numbered?
2. Are all lines of coding numbered in octal?
3. Are index register settings and number of places to shift specified in octal?
4. Are the correct addresses listed for constants?
5. Are the correct addresses listed for transfers of control instructions?
6. Are constants specified correctly?
7. Are all instructions legitimate?
8. Has the correct overflow procedure been specified where required?
9. Are all branches of the flow chart included in the coding?
10. Have all transfer of control addresses been filled in?
11. Has provision been made for interrupt subroutines?
Are F1 and F2 set early in the program?
12. Are all switches and counters properly set to the required initial state?
13. Does the coding accomplish the functions described in the flow chart? In the same order as the flow chart?
14. Are the correct input-output device addresses used?
With the correct orders?
15. Is the β portion of all LOD instructions specified correctly?

XIV. FIELDATA ASSEMBLY LANGUAGE

The process of coding the concepts described in a flow chart into meaningful sequences of octal digits is to a considerable extent a clerical and bookkeeping task. Locating a program, determining the correct addresses of registers, flipflops and I/O devices, and of entering constants and alphanumeric information can be made easier and less subject to error by using the computer to accomplish this translation. Assembly programs permit a programmer to code his program in a symbolic form.

An assembly program translates a program coded in symbolic form into an absolute form suitable for direct loading into the computer.

The FIELDATA family of computers will possess a group of assembly programs prepared for general use in writing programs. Three assembly programs are currently in existence or in a development stage:

<u>Program</u>	<u>Prepared by</u>
FIELDATA Assembly Program I (FAP I)	Sylvania
FIELDATA Assembly Program II (FAP II)	Philco
FIELDATA Assembly Program III (FAP III)	USASRDL

The FAP II Program was specifically designed for use on a minimum BASICPAC configuration, consisting of a Central Processor with 4096 core registers and a paper tape set. Magnetic tapes are not required. FAP II assembles a program in one or two passes. (If the program is to be assembled in a single pass a more restrictive set of "ground rules" for coding are imposed.)

A complete discussion of the procedures and rules for coding in FAP II language are included in the FAP II Assembly Manual. The programmer is encouraged to use FAP II in the preparation of programs for use on BASICPAC.

The general features of the FAP II program are as follows:

1. FAP II input formats are identical to those of FAP I and III.
2. FAP II permits complete symbolic representation of coding; i. e., ADD TØP, 1, BIG; where "TØP" and "BIG" represent symbolic quantities defined or to be defined.
3. All addressable registers, flipflops, and I/O devices can be referred to symbolically; i. e., QRG in place of 70011.
4. FAP II processes twelve pseudo-ops: ØRG, REM, SYN, DEF, EQU, END, ØCT, DEC, ALZ, ALF, BES and BSS. Provisions

have been made for the inclusion of additional pseudo-ops for a system with more than 4096 core registers and/or magnetic tapes.

5. Address arithmetic is permitted; i. e. , ADD TOP-3,2.

FAP II is an assembly program of the type exemplified by the Symbolic Assembly Program (SAP)*, but specifically adapted to BASICPAC.

*For a description of SAP, see: Grabbe, Ramo, Wooldridge, Handbook of Automation, Computation and Control, Vol. II, 1959, J. Wiley and Sons, Inc.

XV. LIBRARY ROUTINES

1. Diagnostic Routines

CLASSIFICATION

ROUTINE

1. Hardware De-
bugging Routines

D. 1. 1 Manual Diagnostic
Tests

2. Acceptance Tests

D. 2. 1 Operations Test

3. General Equipment
Tests

D. 3. 1 General Test

4. Daily Maintenance
Tests

D. 4. 1. 1 Biased Random
Memory Test

D. 4. 1. 2 Worst Pattern
Memory Test

D. 4. 1. 3 Random Number
Test

D. 4. 1. 4 T-Tests

D. 4. 1. 5 Console Paper
Tape Test

D. 4. 1. 6 Communications
Converter Test

D. 4. 1. 7 I/O Paper Tape Test

D. 4. 1. 8 210 Magnetic Tape Test

2. Input-Output Routines

1. Interpreter Routines

E. 1. 1 Relative Address Interpreter

E. 1. 2 Check Sum Routine

E. 1. 3 Program

E. 1. 4 BASICPAC Output to MOBIDIC
Input Tape Conversion

2. Input-Output Routines (Continued)

CLASSIFICATION

ROUTINE

2. Conversion Routines

E.2.1 Fixed Decimal to
Binary Mixed Numbers

E.2.2 Floating Decimal to
Binary

E.2.3 Fixed Fractional Binary
to Decimal Output

E.2.4 Fixed Integral Binary
to Decimal Output

E.2.5 Double Precision Decimal
to Binary Mixed Numbers

E.2.6 Double Precision Floating
Decimal to Binary

E.2.7 Double Precision Fractional
Binary to Decimal Output

E.2.8 Double Precision Integral
Binary to Decimal Output

E.2.9 Floating Binary to Decimal

E.2.10 Double Precision Floating
Binary to Decimal

3. Service Routines

1. Mathematical Sub-
routines

1.1 Fixed Single Pre-
cision

U.1.1.1 Complex Arithmetic
(Add, Subtract, Multiply,
Divide)

U.1.1.2 2^x , e^x , 10^x

U.1.1.3 $\log_2 X$, $\log_{10} X$,
 $\log_e X$

3. Service Routines (Continued)

CLASSIFICATION

ROUTINE

	U.1.1.4 Square Root
	U.1.1.5 Sine and Cosine
	U.1.1.6 Tangent
	U.1.1.7 Arcsine, Arccosine, Arctangent
1.2 Fixed Double Precision	U.1.2.1 Double Precision Fixed Arithmetic (Add, Sub- tract, Multiply, Divide)
	U.1.2.2 Double Prevision Complex Arithmetic (Add, Subtract, Multiply, Divide)
	U.1.2.3 Fixed Double Pre- cision Square Root
1.3 Floating Single Precision	U.1.3.1 Internal Fixed to Floating Conversion
	U.1.3.2 Internal Double, Precision Fixed to Single Precision Floating Con- version
	U.1.3.3 Internal Floating to Fixed Conversion
	U.1.3.4 Floating Point Arithmetic (Add, Sub- tract, Multiply, Divide)
	U.1.3.5 Floating Point Square Root
	U.1.3.6 Floating Point Complex Arithmetic

3. Services Routine (Continued)

CLASSIFICATION

- U.1.3.7 Floating Point Logarithms
- U.1.3.8 Floating Point Exponentials
- U.1.3.9 Floating Point Sine and Cosine
- U.1.3.10 Floating Point Tangent
- U.1.3.11 Floating Point Arcsine,
Arctangent
- 1.4 Floating Double Precision
 - U.1.4.1 Internal Double Precision
Fixed to Floating Conversion
 - U.1.4.2 Internal Single Precision
Fixed to Double Precision Floating
Conversion
 - U.1.4.3 Internal Double Precision
Floating to Fixed Conversion
 - U.1.4.4 Internal Double Precision
Floating to Single Precision Fixed
 - U.1.4.5 Floating Double Precision
Arithmetic
 - U.1.4.6 Floating Double Precision
Square Root
 - U.1.4.7 Floating Double Precision
Complex Arithmetic
- U.3
 - U.3.1 Random Dump

APPENDIX B

ADDRESS ASSIGNMENTS FOR MEMORY LOCATIONS,
AND ADDRESSABLE REGISTERS AND FLIPFLOPS

TABLE IADDRESS ASSIGNMENTS FOR MEMORY LOCATIONS ¹A. Basic Assignments

<u>Address</u>	<u>Name</u>
00000 - 07777	Memory Unit Zero
10000 - 17777	Memory Unit One
20000 - 27777	Memory Unit Two
30000 - 37777	Memory Unit Three
40000 - 47777	Memory Unit Four
50000 - 57777	Memory Unit Five
60000 - 67777	Memory Unit Six

B. Reserved Assignments

Under certain conditions as specified below, selected memory locations are reserved for specific functions.

1. In Trapping Mode

<u>Address</u>	<u>Name</u>
00000	First instruction of the trapping subroutine.

2. If I/O Converter(s) Connected and Activated

<u>Address</u>	<u>Name</u>
00001	First instruction of the Interrupt subroutine entered whenever errors are detected before acceptance of I/O Converter orders. ²
00002	First instruction of the Interrupt subroutine entered whenever an I/O instruction has been accepted by an I/O Converter. ³

3. If Communications Converter Present and Activated

<u>Address</u>	<u>Name</u>
00002	First Instruction of the Interrupt subroutine entered following completion of any ³ Communications Converter function.
00003	Communications Converter Output Interrupt Word ⁴
00004	Communications Converter Input Interrupt Word ⁵
00010- 00017	Communications Converter Output Storage ⁶
00020- 00037	Communications Converter Input Storage ⁷

TABLE I NOTES

- ¹ Every memory location requires a 15 bit address designator and is addressable only through the α portion of the instruction word. Note that memory locations are sometimes referred to as "registers" or "pseudo registers".
- ² Before execution of an I/O order involving an I/O converter, both the order and the equipment involved are examined. If any of the following conditions exist the program jumps to memory location 00001.
- a. Non-existent I/O converter addressed.
 - b. Busy I/O converter addressed.
 - c. Improper order
 - d. Initial device malfunction.
- ³ The program jumps to memory location 00002 if any of the following conditions exist and if the DPI flip flop is set equal to zero:
- a. Termination of an I/O instruction by an I/O converter when:
 - i. A control character is received in interpret control mode.
 - ii. An error condition is detected in the I/O converter.
 - iii. Transmission is complete.

b. Completion of any Communications Converter function when:

i. A control character is received.

ii. Input or output transmission is complete.

4

This memory location is reserved for storage of the following information when an Interrupt occurs on output from the Communications Converter.

a. The address from which information was transmitted is placed in the α portion of 00003.

b. If parity error has occurred during transmission, a "1" is placed in bit 37 of 00003.

5

This memory location is reserved for storage of the following information when an Interrupt occurs on input to the Communications Converter.

a. The address to which the last character was transmitted is placed in the α portion of 00004.

b. If parity error has occurred during transmission, a "1" is placed in bit 37 of 00004.

c. An indication of the number of characters missing from the expected transmission is placed in bits 34-36 of 00004. (The number of characters is represented by a form of Gray code). If no characters are missing "000" is placed in bits 34-36.

6

There is one memory location reserved for each output channel of the Communications Converter; e.g. Location 00010 for Channel 0; Location 00011 for Channel 1, etc.

7

There is a pair of consecutive memory locations reserved for each input channel of the Communications Converter. The even-numbered location is reserved for data words, the odd-numbered location for control function characters. e.g. Locations 00020 & 00021 for Channel 0; Locations 00022 & 00023 for Channel 1.

TABLE II

ADDRESS ASSIGNMENTS FOR ADDRESSABLE REGISTERS ¹A. Central Processor Registers ²

<u>Address</u>	<u>Address</u>	<u>Address</u>	<u>Name</u>	<u>Code</u>
70001-70007	0001-0007	1-7	Index Registers Nos. 1-7	I ⁷ (i=1,2,...7)
70010	0010	-	Accumulator	A
70011	0011	-	Q Register	Q
70012	-	-	B Register	B
70013	-	-	Program Counter	PC
70014	0014	-	Program Counter Store	PCS
70020	0020	-	Word Switch Register	WSR

B. Input-Output Converter Registers

<u>Address</u>	<u>Address</u>	<u>Name</u>	<u>Code</u>
70030-70036	0030-0036	I/O Converter Instruction Registers	CIS ⁱ ³ (i=1,2,...7)

C. Communications Converter Registers

<u>Address</u>	<u>Address</u>	<u>Name</u>	<u>Code</u>
70021	0021	Address and Word Counter for Communications Con- verter Limited Interrupt Channel (Channel 0)	KIW ⁴
--	0050-0057	Communications Converter Output Channel Registers	KOU ⁱ ⁵ (i=0,1,...7)

TABLE II NOTES

1

All registers except the KOU^1 have been assigned individual 15-bits address designators beginning with an octal "7".

- a. If a register is addressed through the α portion of the instruction word, this 15-bit designator is employed.
- b. If a register is addressed through the β portion of the instruction word, a 12-bit designator is employed. Note that when both α & β addresses exist for a register, they are identical in bits 1-12.

e.g. A is: 70010 if addressed in α
0010 if addressed in β

- c. Index Registers only may also be addressed through the γ portion of the instruction word. A 3-bit octal designator is provided for this purpose. Note that the α , β & γ designators for Index Registers are identical in bits 1-3.

e.g. I^1 is: 70001 if addressed in α
0001 if addressed in β
1 if addressed in γ

2

- a. All Central Processor Registers shown in Table II A, except Registers PC & B can be loaded under program control with the LOD instruction by addressing the register through the β portion of the instruction word.
- b. All Central Processor Registers shown in Table II A are legitimate addresses in the α portion of the LOD instruction.
- c. All Central Processor Registers shown in Table II A, except B, are legitimate addresses in the α portion of the LGM, LGA and arithmetic instructions.
- d. All Central Processor registers shown in Table II A, except Registers B & PC, are legitimate addresses in the α portion of the instructions STR, RPA, and MSK.

- e. The A register is the only register from which information can be stored in memory locations.

3

Each I/O Converter in the system (maximum seven) is assigned an individual CIS register.

e.g. $CIS^1 = 70030$ is provided for I/O Converter No. 1.
 $CIS^2 = 70031$ is provided for I/O Converter No. 2.

- a. Any CIS Register can be loaded under program control with the LOD instruction by addressing the CIS through the β portion of the instruction word.
- b. Any CIS Register is also a legitimate address in the α portion of the LGM, LGA and arithmetic instructions.

4

KIW counts input words and addresses for the limited interrupt channel.

- a. KIW can be loaded under program control with the LOD instruction by addressing the KIW through the β portion of the instruction word.
- b. KIW is a legitimate address in the α portion of all arithmetic instructions.

5

There is one KOU register provided for each output channel of the Communications Converter.

e.g. $KOU^0 = 70050$ is provided for Channel 0.
 $KOU^1 = 70051$ is provided for Channel 1.

- a. Any KOU register can be addressed only in the β portion of the LOD instruction.

TABLE III

ADDRESS ASSIGNMENTS FOR ADDRESSABLE FLIP-FLOPS ¹A. Central Processor Flip-Flops ²

<u>Address</u>	<u>Name</u>	<u>Code</u>
0100	Overflow Alarm	OA
0102	Interpret Sign	ISN
0110-0117	General Sense Flip-Flops	SFF ⁱ (i=1,2...8)
0136	Interpret Con- trol Function	ICF
0137	Disable Program Interrupt	DPI
0360	Control Unit Stop (End of File)	STP (EOF)
0361	Control Unit Con- trol Indicator	IC ^C (CCI)

B. I/O Converter Flip-Flops ³1. Assignments for I/O Converter No. 1 (BasicPac₁ Type
A only) ⁴

<u>Address</u>	<u>Name</u>	<u>Code</u>
0140	Interrupt Request	IRQ ¹
0141	Converter Busy	CVB ¹
0142	Control Indicator	CI ¹
0143	End of File	EOF ¹
0144-0147	Device Busy	DB _i ¹ (i=1,2,3,4)
0150	I/O Alarm	IOA ¹

<u>Address</u>	<u>Name</u>	<u>Code</u>
0151	Converter Memory Parity Error	CMPE ¹
0152	Improper Order	IMO ¹
0153	Device Malfunction Alarm	DVA ¹
0154	I/O Parity Error	IOPE ¹
0155	Data Drop Alarm	DDA ¹
0156	Beginning of Tape	BOT ¹
0157	End of Tape	EOT ¹

2. Assignments Reserved for I/O Converters Nos. 2-7

The following addresses are reserved for the addressable flip flops of the I/O Converters indicated.

<u>Addresses</u>	<u>I/O Converter</u>
0160-0177	No. 2
0200-0217	No. 3
0220-0237	No. 4
0240-0257	No. 5
0260-0277	No. 6
0300-0317	No. 7

C. Communications Converter Flip-Flops ⁵

<u>Address</u>	<u>Name</u>	<u>Code</u>
0370	Input Interrupt	KAI
0371	Input Error	KAE
0372	Output Interrupt	KEI
0420-0437	Communications Converter Input Channel Flip-Flops	KIU ⁱ (i=0,1,2...7) ⁶

TABLE III NOTES

- 1 Every flip-flop has been assigned a 12-bit address designator; flip-flops are always addressed through the β portion of the instruction word.
- 2 All Central Processor Flip-Flops shown in Table III A are capable of being sensed, set and reset.
- 3 All I/O Converter Flip-Flops shown in Table III B, Section 1, except DB_i^1 are capable of being sensed, set and reset. The DB_i^1 can only be sensed.
- 4 The address assignments shown in Table III B, Section 2, are applicable only when a BasicPac type A I/O converter is connected as I/O Converter No. 1. If any other converter is connected as I/O converter No. 1, specific assignments within the address range 0140 - 0157 would apply for the I/O converter involved. If a BasicPac type A I/O converter is connected as I/O converter No. i , where $i = 2, 3, \dots, 7$, new address assignments would be made in the appropriate address range as shown in Table III B, Section 2.
- 5 All Communications Converter Flip-Flops shown in Table III C are capable of being sensed, set and reset.
- 6 A pair of Communications Converter Input channel flip-flops is provided for each communications converter input channel.

e.g. $KIU^1 = 0420$ or 0421 are provided for Channel 0

$KIU^2 = 0422$ or 0423 are provided for Channel 1, etc.

APPENDIX C

ADDRESS ASSIGNMENTS
FOR
INPUT-OUTPUT DEVICES

APPENDIX C

ADDRESS ASSIGNMENTS FOR INPUT-OUTPUT DEVICES

Input-Output address assignments are given below. Octal assignments 04 through 12 are designated as unique spares to be given definite assignments at a late date. Octal assignments 71 through 77 are reserved for specialized spares. They are to be used in any system requiring temporary spares for their system alone. Simple logical testing of the various binary bits will provide information as to the type of input-output devices being addressed.

Device Octal	Address Binary	Assignment	Device Octal	Address Binary	Assignment
01	000001	Console Reader	40	100000	
02	000010	Console Punch	41	100001	
03	000011	Console Typewriter	42	100010	
04	000100	Unique	43	100011	
05	000101	Spares to be	44	100100	
06	000110	assigned	45	100101	
07	000111		46	100110	
			47	100111	
10	001000		50	101000	
11	001001		51	101001	
12	001010		52	101010	
13	001011	Area Display	53	101011	
14	001100	Card Reader	54	101100	
15	001101	Card Punch	55	101101	
16	001110	Line Printer	56	101110	
17	001111	Line Printer	57	101111	
20	010000	8-Ch. Pt. Reader	60	110000	
21	010001	8-Ch. Pt. Reader	61	110001	
22	010010	8-Ch. Pt. Punch	62	110010	
23	010011	8-Ch. Pt. Punch	63	110011	
24	010100	5-Ch. Pt. Reader	64	110100	
25	010101	5-Ch. Pt. Punch	65	110101	
26	010110	Typewriter	66	110110	
27	010111	Typewriter	67	110111	
30	011000	Informer	70	111000	Control Panel Specialized Spares
31	011001	Informer	71	111001	
32	011010	Spare	72	111010	
33	011011	Spare	73	111011	
34	011100	Spare	74	111100	
35	011101	Spare	75	111101	
36	011110	Mobidic B	76	111110	
37	011111	Mobidic B	77	111111	

APPENDIX D

CODES

BASICPAC ORDER CODES

1. Arithmetic Orders

<u>Octal Code</u>	<u>Mnemonic</u>	<u>Instruction</u>
10	CLA	Clear and Add
12	ADD	Add
13	ADM	Add Magnitude
14	CLS	Clear and Subtract
16	SUB	Subtract
20	MLY	Multiply
22	DVD	Divide
23	DVL	Divide Long

2. Transfer Orders

<u>Octal Code</u>	<u>Mnemonic</u>	<u>Instruction</u>
40	TRU	Transfer Unconditional
41	TRL	Transfer and Load PCS
42	TRS	Transfer to PCS
43	TRX	Transfer on Index
44	TRP	Transfer on Positive
45	TRZ	Transfer on Zero
46	TRN	Transfer on Negative

3. Sense Orders

<u>Octal Code</u>	<u>Mnemonic</u>	<u>Instruction</u>
05	SEN	Sense
06	SNS	Sense and Set
07	SNR	Sense and Reset

4. Logical Orders

<u>Octal Code</u>	<u>Mnemonic</u>	<u>Instruction</u>
00	HLT	Halt
02	LGM	Logical Multiply
03	LGA	Logical Add
30	SHL	Shift Left
31	SLL	Shift Left Long
32	SHR	Shift Right
33	SRL	Shift Right Long
35	CYL	Cycle Long
50	STR	Store
51	LOD	Load
53	LDX	Load Index
54	RPA	Replace Address
55	MSK	Replace Through Mask

5. Input-Output Orders

<u>Octal Code</u>	<u>Mnemonic</u>	<u>Instruction</u>
66	SKP	Skip
67	BSP	Backspace
70	RAN	Read Alphanumeric
71	RRV	Read Reverse
72	ROK	Read Octal
74	WAN	Write Alphanumeric
75	WWA	Rewrite Alphanumeric
76	WOK	Write Octal
77	RWD	Rewind

FIELDATA Standard Code

The complete FIELDATA Code, including the alphanumeric characters and the control functions, is presented below in the standard 8-bit form which uses odd parity and the basic pattern for control (1 = data, 0 = control).

7 6 5 4 3 2 1 0
P C I₂ I₁ D₃ D₂ D₁ D₀

ALPHANUMERIC

CONTROL

Character	76543210 PCIIDDDD	Character	76543210 PCIIDDDD	Character	76543210 PCIIDDDD	Character	76543210 PCIIDDDD
Master Sp.	01000000)	11100000	Dial 0	00100000	Blank/Idle	10000000
U. C.	11000001	-	01100001	Dial 1	10100001	Control UC	00000001
L. C.	11000010	+	01100010	Dial 2	10100010	Control LC	00000010
TAB	01000011	<	11100011	Dial 3	00100011	Control Tab	10000011
Car. Ret.	11000100	=	01100100	Dial 4	10100100	Control CR	00000100
Space	01000101	>	11100101	Dial 5	00100101	Control Spa	10000101
A	01000110	.	11100110	Dial 6	00100110	Control A	10000110
B	11000111	\$	01100111	Dial 7	10100111	Control B	00000111
C	11001000	*	01101000	Dial 8	10101000	Control C	00001000
D	01001001	(11101001	Dial 9	00101001	Control D	10001001
E	01001010	"	11101010	SOC	00101010	Control E	10001010
F	11001011	:	01101011	SOB	10101011	Control F	00001011
G	01001100	?	11101100	SOD	00101100	Control G	10001100
H	11001101	!	01101101	SPARE	10101101	Control H	00001101
I	11001110	,	01101110	SPARE	10101110	Control I	00001110
J	01001111	⊕	11101111	STOP	00101111	Control J	10001111
K	11010000	0	01110000	RTT	10110000	Control X	00010000
L	01010001	1	11110001	RTR	00110001	Control L	10010001
M	01010010	2	11110010	NRR	00110010	Control M	10010010
N	11010011	3	01110011	EOBK	10110011	Control N	00010011
O	01010100	4	11110100	EOB	00110100	Control O	10010100
P	11010101	5	01110101	EOF	10110101	Control P	00010101
Q	11010110	6	01110110	EOC	10110110	Control Q	00010110
R	01010111	7	11110111	AKR	00110111	Control R	10010111
S	01011000	8	11111000	RPB	00111000	Control S	10011000
T	11011001	9	01111001	ISN	10111001	Control T	00011001
U	11011010	'	01111010	NISN	10111010	Control U	00011010
V	01011011	;	11111011	CWF	00111011	Control V	10011011
W	11011100	/	01111100	SPARE	10111100	Control W	00011100
X	01011101	.	11111101	SAC	00111101	Control X	10011101
Y	01011110	Special	11111110	SPC	00111110	Control Y	10011110
Z	11011111	Backspace	01111111	DELETE	10111111	Control Z	00011111

FIELDATA Paper Tape Code

The complete FIELDATA Paper Tape Code, including alphanumeric and control characters, is presented below in the 8-bit form for paper tape, which uses even parity and the paper tape pattern for control.

7 6 5 4 3 2 1 0
P C I₂ I₁ D₃ D₂ D₁ D₀

ALPHANUMERIC

CONTROL

Character	76543210 PCIIDDDD	Character	76543210 PCIIDDDD	Character	76543210 PCIIDDDD	Character	76543210 PCIIDDDD
Master Sp.	11000000)	10100000	Dial 0	01100000	Blank/Idle	00000000
U. C.	01000001	-	00100001	Dial 1	11100001	Control UC	10000001
L. C.	01000010	+	00100010	Dial 2	11100010	Control LC	10000010
Tab.	11000011	<	10100011	Dial 3	01100011	Control Tab	00000011
Car. Ret.	01000100	=	00100100	Dial 4	11100100	Control CR	10000100
Space	11000101	>	10100101	Dial 5	01100101	Control Spa	00000101
A	11000110	-	10100110	Dial 6	01100110	Control A	00000110
B	01000111	\$	00100111	Dial 7	11100111	Control B	10000111
C	01001000	*	00101000	Dial 8	11101000	Control C	10001000
D	11001001	(10101001	Dial 9	01101001	Control D	00001001
E	11001010	"	10101010	SOC	01101010	Control E	00001010
F	01001011	:	00101011	SOB	11101011	Control F	10001011
G	11001100	?	10101100	SOD	01101100	Control G	00001100
H	01001101	!	00101101	SPARE	11101101	Control H	10001101
I	01001110	,	00101110	SPARE	11101110	Control I	10001110
J	11001111	⊕	10101111	STOP	01101111	Control J	00001111
K	01010000	0	00110000	RTT	11110000	Control K	10010000
L	11010001	1	10110001	RTR	01110001	Control L	00010001
M	11010010	2	10110010	NRR	01110010	Control M	00010010
N	01010011	3	00110011	EOBK	11110011	Control N	10010011
O	11010100	4	10110100	EOB	01110100	Control O	00010100
P	01010101	5	00110101	EOF	11110101	Control P	10010101
Q	01010110	6	00110110	EOC	11110110	Control Q	10010110
R	11010111	7	10110111	AKR	01110111	Control R	00010111
S	11011000	8	10111000	RPB	01111000	Control S	00011000
T	01011001	9	00111001	ISN	11111001	Control T	10011001
U	01011010	'	00111010	NISN	11111010	Control U	10011010
V	11011011	;	10111011	CWF	01111011	Control V	00011011
W	01011100	/	00111100	SPARE	11111100	Control W	10011100
X	11011101	.	10111101	SAC	01111101	Control X	00011101
Y	11011110	Special	10111110	SPC	01111110	Control Y	00011110
Z	01011111	Backspace	00111111	DELETE	11111111	Control Z	10011111

> <