INDEX

CONFIDENTIAL

The material in this book represents a constantly changing workbook. Because of these conditions, this information may at any time be outdated.

FORTRAN IVA SYSTEM

DESIGN WORKBOOK

JUNE - SEPT., 1963

Computer Usage Company, Inc.

655 Madison Avenue
New York, New York 10021

6266 Manchester Blvd.
Los Angeles, California

Preliminary

FORTRAN IVA System Description

FORTRAN IVA is a programming system for the IBM 7090/7094 designed to speed the processing of all-FORTRAN jobs. This new system will process jobs consisting primarily of FORTRAN compilations approximately five times faster than the IBJOB system running under equivalent conditions. The execution time required for object programs will be approximately the same as that for object programs produced by FORTRAN IV under IBJOB.

The FORTRAN IVA System operates under control of the Basic Monitor (IBSYS) and permits operation from 729 magnetic tape units, 7320 magnetic drum, 7340 hypertapes or 1301 magnetic disc system. Both the compiler and the object system are designed to take advantage of the logic of the IBM 7094-II. The compiler will operate on a 7090, 7094 or 7094-II and produce object programs for any of these same machines.

This system permits a considerable degree of compatibility with programs prepared for use with the IBJOB system. FORTRAN IV statements which may be used in one are acceptable without change to the other. In some cases, the sequencing of the source deck may require slight changes. Since the FORTRAN IVA Loader does not accept input in the format processed by IBLDR, it will be necessary that existing FORTRAN IV programs be recompiled and existing MAP subprograms be reassembled for use with programs and subprograms prepared by the new system. Some small changes may be required in MAP subprograms. Since the same library functions and compiler conversion routines will be used, identical computed results are assured.

Among the new features available with the FORTRAN IVA system are:

(1) The ability to compile directly to a library file of user programs. When this file is maintained on disc or drum, it becomes a valuable method for handling and storing object programs in use at an installation.

(2) An absolute self-loading job deck may be obtained from the Loader thus permitting future production runs to be made simply.

(3) The system provides for semi-automatic overlay of subprograms with minimum manipulations required by the source programmer.

In exchange for the added speed, the user of FORTRAN IVA must be prepared for some slight system limitations.

(1) No program or subprograms may designate more than seven COMMON blocks.

(2) The following statements must all precede the first executable statement: COMMON, DIMENSION, EQUIVALENCE, DATA, BLOCK DATA, TYPE, SUBROUTINE, FUNCTION and arithmetic statement functions. FORMATs may appear anywhere.

(3) The method of processing will limit source programs to approximately 500 to 700 FORTRAN IV statements and 500 to 700 MAP statements. These limits may be readily circumvented by breaking oversize programs into smaller subprograms.

(4) The object program will not utilize the 7090/94 Input/Output Control System. Instead, individual library routines will be brought in to perform all the functions required by FORTRAN IV object programs.

(5) While informative charts and listings will be produced during each compilation, a symbolic assembly listing of the object program will not be available since the compiler produces machine code directly.

Table I lists the control cards and options available within the FORTRAN IVA system.

$F4JOB Name [GO / NOGO / LIBRUN[1]] , [M94 / M90] , [7XR / nXR] , [CHART / NOCHART] , [IOCS / IORT][2] [USER / SYSTE

$FORT4 [3]Progname [LIST / NOLIST] , [DECK / NODECK]

$SAF Progname [DECK / NODECK]

$OBJECT[4] Progname

$LIBEDT[6] [5]Progname [ADD] , [DELETE] , [COPY / NOCOPY] , [CHANGE, Name]

$LIBLST[7]

The following control cards generate information in the Communications region that is used only by the Loader.

$USE [8]Libname [USER / SYSTEM]

$OMIT [9]Progname

$NAME [10]Progname= Revname

$OVERLAY [11][1 / n units] Subprogname, Subprogname .....

$ETC. Subprogname .....

1. Name of a main program is required if this option is used.
2. These options require further definition.
3. Program name is mandatory if this is a main program.
4. Indicates relocatable binary program follows.
5. Program name is required if DELETE or CHANGE option is used. They cannot coexist on the same card.
6. The appearance of the ADD option refers to the entire job. Whereas DELETE refers to the program name. There may be more than one DELETE requested in a job.
7. When a library list is to be performed it must be the only processing request in a job.
8. The program name specified will be used from the library instead of the Load File.
9. The program specified will be omitted from loading.
10. This is a temporary renaming of the program name.
11. The number of tape units available to the loader for preparing object overlays for the specified subprograms.

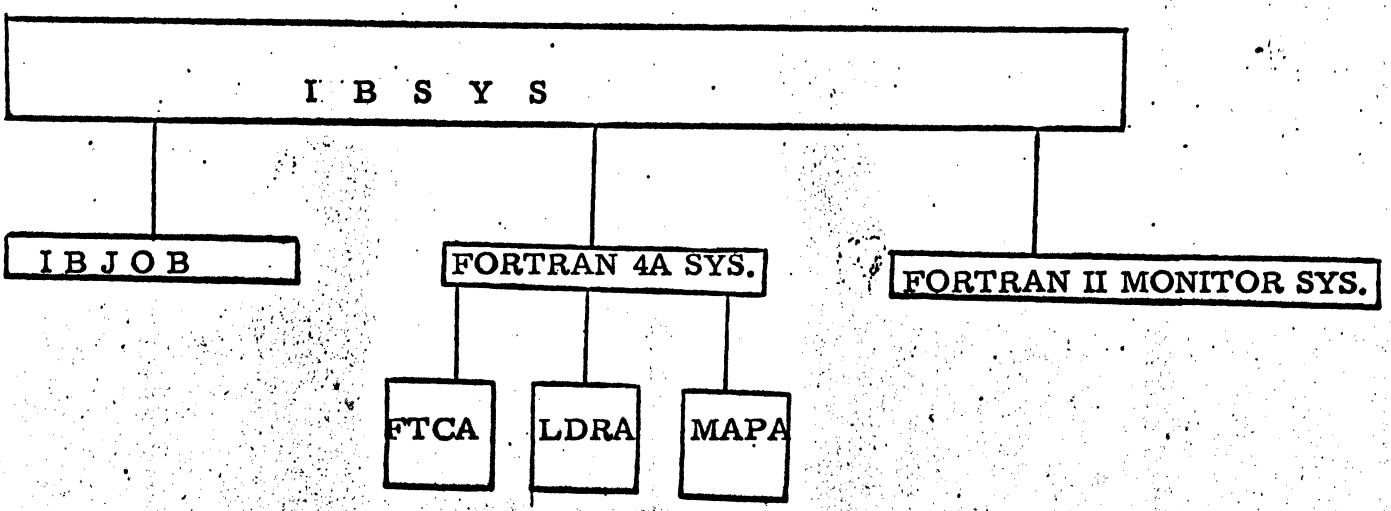
In equal GO or NOGO and/or the appearance of $LIBEDT will result in an update of the library tape. The new Library tape will be used by the Loader. If the options were GO and/or $LIBEDT the Loader operates as if in the LIBRUN mode.

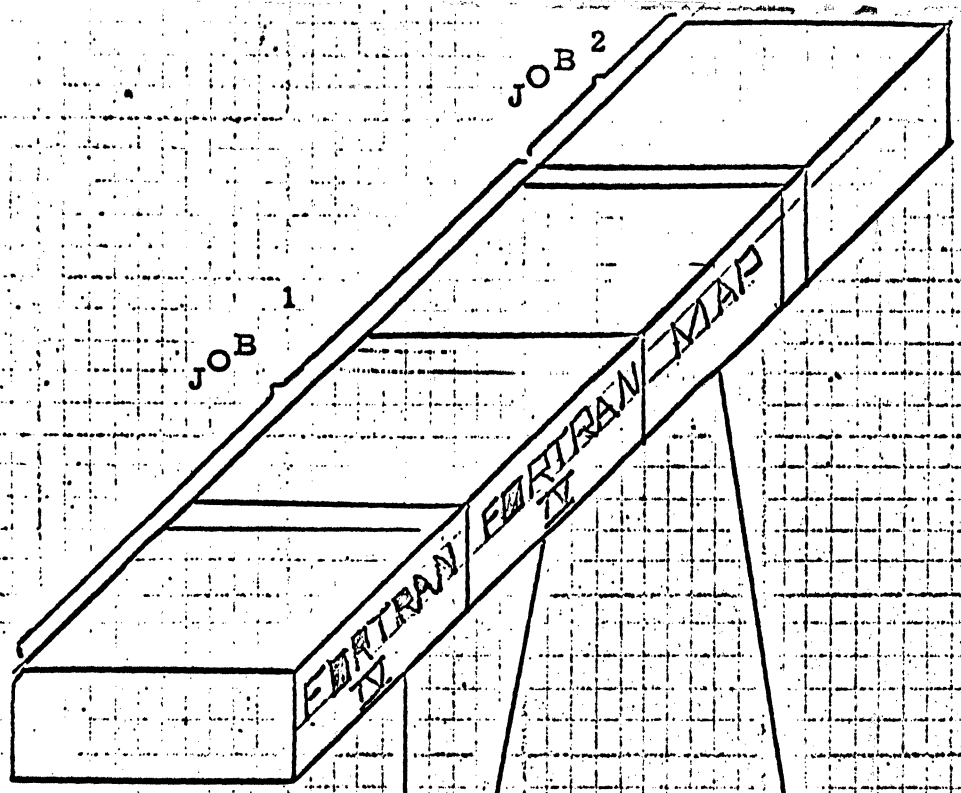
Order of Control Cards

Major Control Cards

$ F 4JOB	precedes every job.
$ FORTA	precedes each compilation.
$ SAF	precedes each assembly.
$ LIBEDT	The order will not effect the processing. The library edit will occur immediately before any loading.

Remaining cards may follow any of the major control cards or may appear after the END card in the last Fortran or Map deck.


I B S Y S
I B J O B
FORTRAN 4A SYS.
FORTRAN II MONITOR SYS.
FTCA
LDRA
MAPA



FTC A

MAP A

LOAD FILE

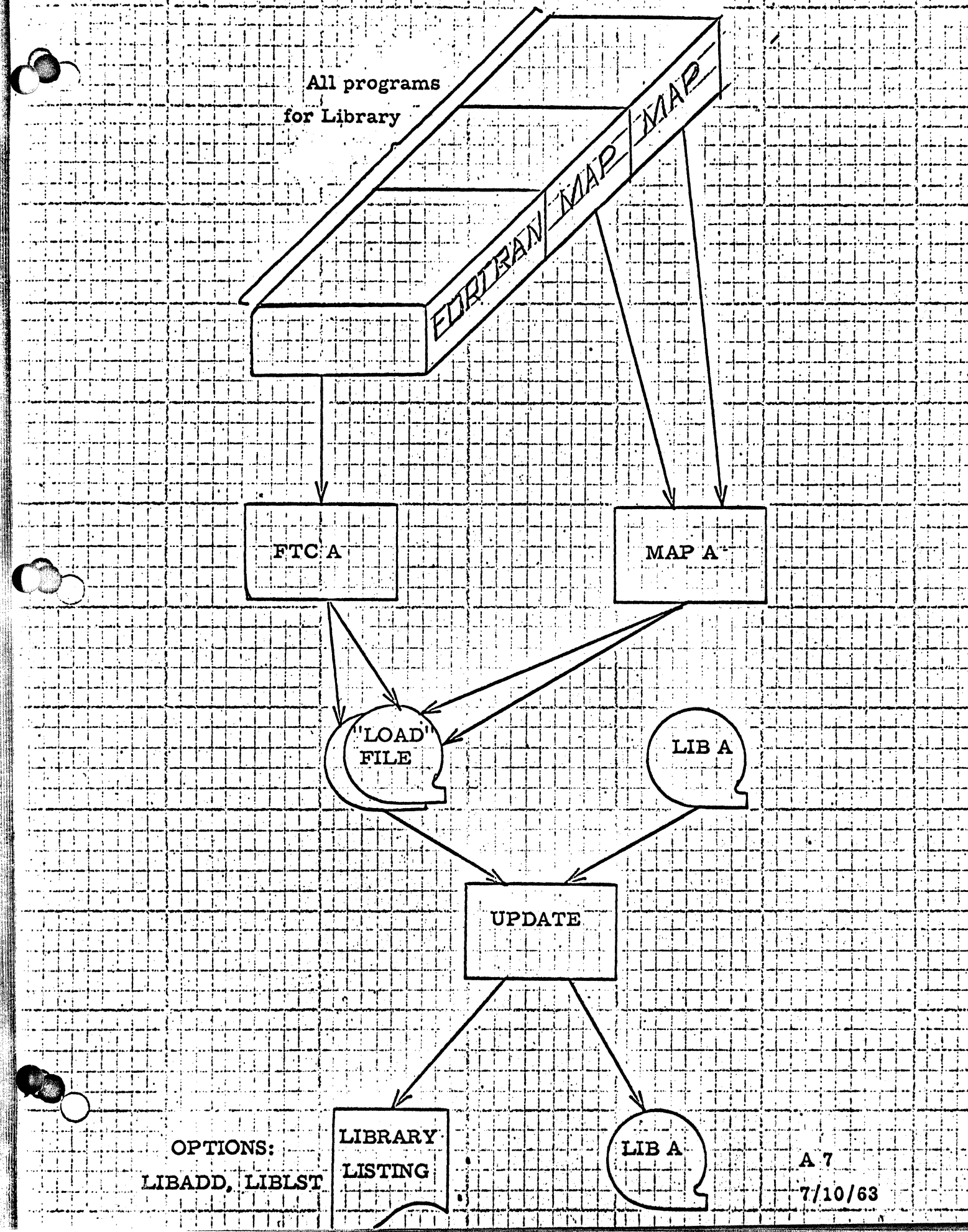
LIB A

LOADER A

Object program execution

ABS self loading deck

OPTIONS: -GO, -DECK


All programs
for Library
FORTRAN
MAP
MAP
FTC A
MAP A
"LOAD"
FILE
LIB A
UPDATE
LIBRARY
LISTING
LIB A
OPTIONS:
LIBADD, LIBLST

Compatability FORTRAN IV → FORTRAN IVA

FORTRAN IV Language — Same--(add logical variable Format)

FORTRAN subprograms--must be recompiled

MAP subprograms--must be reassembled

} in some cases, some reorganization will be required

PREST decks

LDR text, dict decks

} no compatability

Binary tapes written by FORTRAN IV- uncertain

Library- Generally same programs

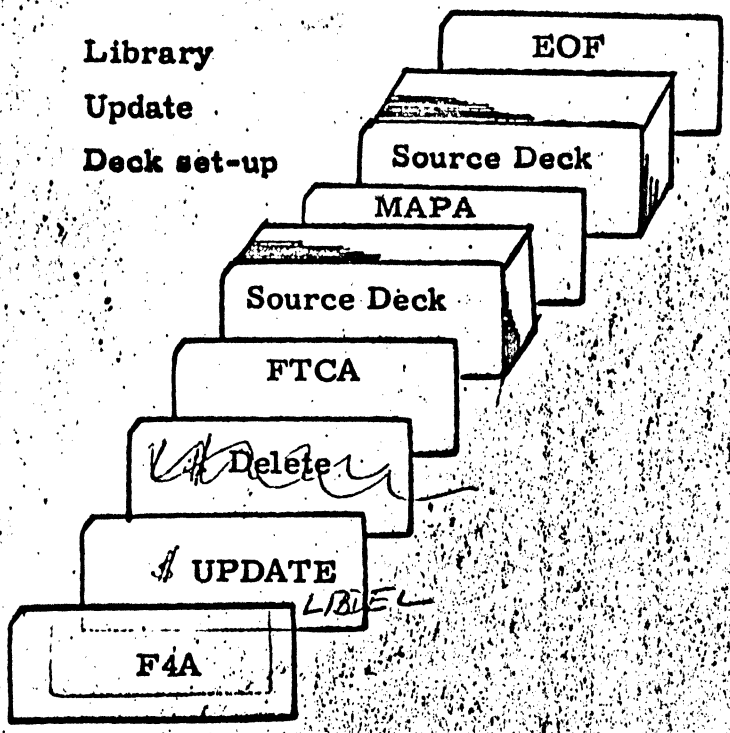
Conversion within compiler- will probably use some routines.

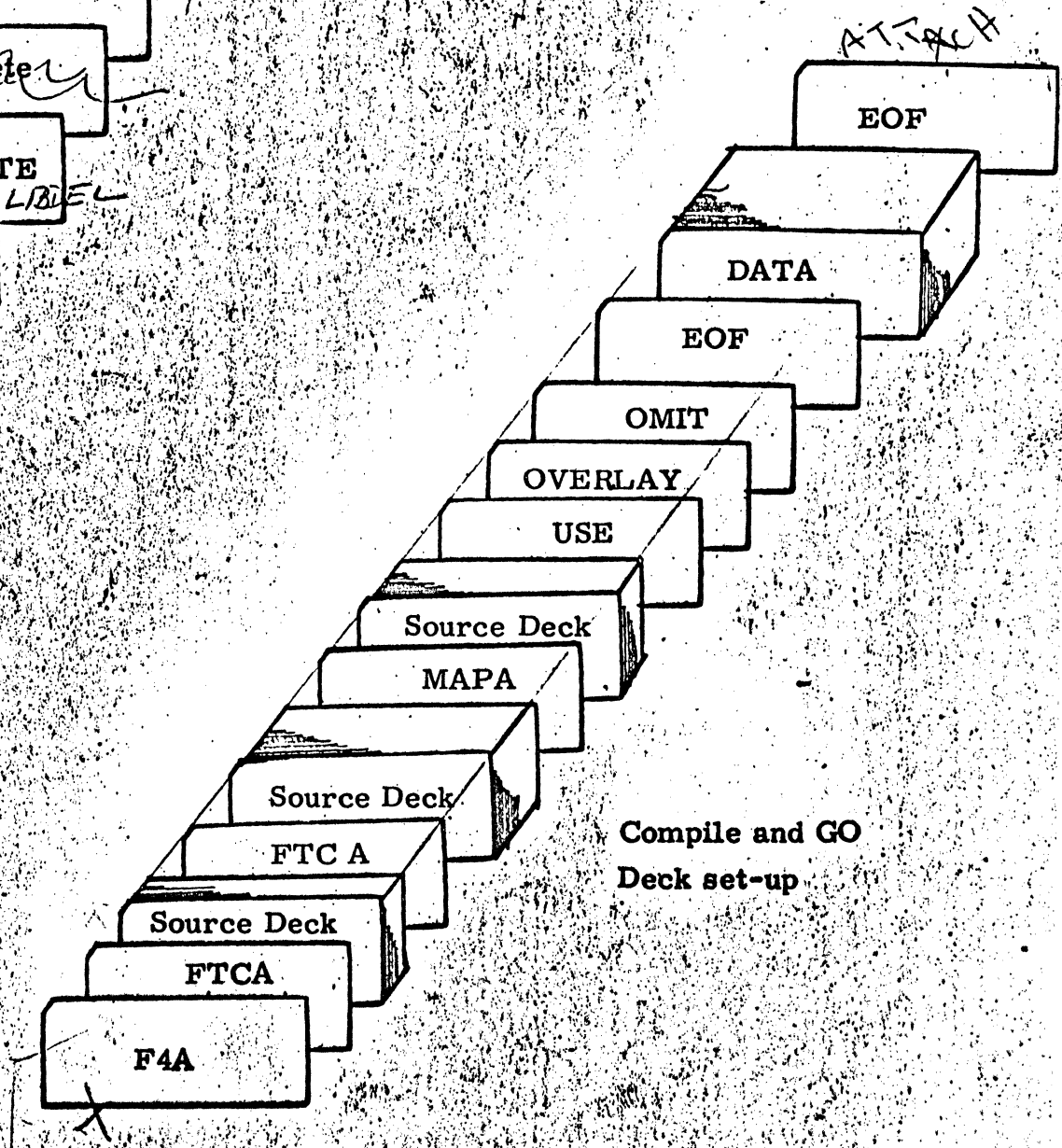
COMPILER SOURCE DECK CHANGES

Sequence: non-executable must precede executable (FORMATs excepted)

Only 7 COMMON sections allowed.

Library
Update
Deck set-up





Compile and GO
Deck set-up

System Input-Output Requirements

Since the system will run under IBSYS--the processor's I-O requirements are taken care of--except for some possible fancy tape manipulations.

Object Program will use selected IOCS sections from library--only required pieces--no buffering or label checking.

Self loader will probably have ability to reassign I-O units at load time.

System Communication Region

Instead of generating and/or passing control cards down to lower system, we will use a System Communications Region which will contain information about each job being processed. Various options will be represented by bit positions.

Discovered information (fatal diagnostics, number of subprograms; name of main program; etc.) will also be passed down through this region.

System Communication Region

Bit Flags

GO/NOGO
Deck/~~NODECK~~ ABS/NOABS
Loadfile/LIBRUN
7090/7094
Liblst/NoLiblst
Libcpy/ NoLibcpy
An External function appeared from compile
A diagnostic failure occurred & Process
A main program has been compiled from Compile

use
omit
NAME
U
(Cards
BLOCK DATA
subprog
from

Counts

n index registers
compilations and assemblies in this job (control cd)

Names

Title (main program)
Libdel (may be more than one)
Libadd (may be more than one)
Omit (may be more than one)
Libchg (2 names) (may be more than one)
Name (2 names) (may be more than one)
Use (2 names) (may be more than one)

from Process
Control Cds

Overlay

3 words (may be more than one)

System Tape Repositioning Method

System Control always reads one card ahead for all processors. Whenever it reads a "next-action" control card, it either:

(a) takes immediate action

or

(b) stacks the requiremtns as a list and works through the list interspersed between subsequent reads and writes.

before passing the card preceding the present control card to the processor.

TAPE POSITIONING

	PHASE 1						PHASE 2						
	A		B		C		A		B		C		
SYSLB1	Rd		Rd		Rd		Rd		Rd		Rd	Rep	IBSYS
SYSIN1		Rd		Rd									SOURCE PROGRAM
SYSOUT1		S-Wr		S-Wr		S-Wr				S-Wr		W	LISTINGS
* SYSUT3		S-Wr		S-Wr	RWD			Rd	RWD				EXTERNAL FILE
~~SYSUT1~~										~~W~~			~~LOAD FILE A~~
SYSUT2											W		LOAD FILE B
~~SYSLB2~~	~~RWD~~					Rd							~~PROGRAM LIBRARY~~

* if spill

one compilation - FORTRAN IVA

Rd	= Read
W	= Write
S-Wr	= Sporadic Write
RWD	= Rewind
Rep	= Reposition

loader need SYSUT2 for punch output

Technical Changes to be Considered

1. Compiler external file will be a "spill" type file. Thus small programs will remain entirely in core.
2. Compiler records will be as few in number as possible.
3. All diagnostics from Phase I will be in a tape record following Phase II. Only when needed will it be read in. Thus, valid batch compilations will be faster and compilations of invalid programs will be slower.

8/8/6

Fortran Compiler Communications Region

Bit Flags

Main/subprogram

List/No list

An adjustable dimension appears *

Double Data Function *

Subprogram without arguments *

External Function Definition

External Function Transmission *

Diagnostic of Level 1

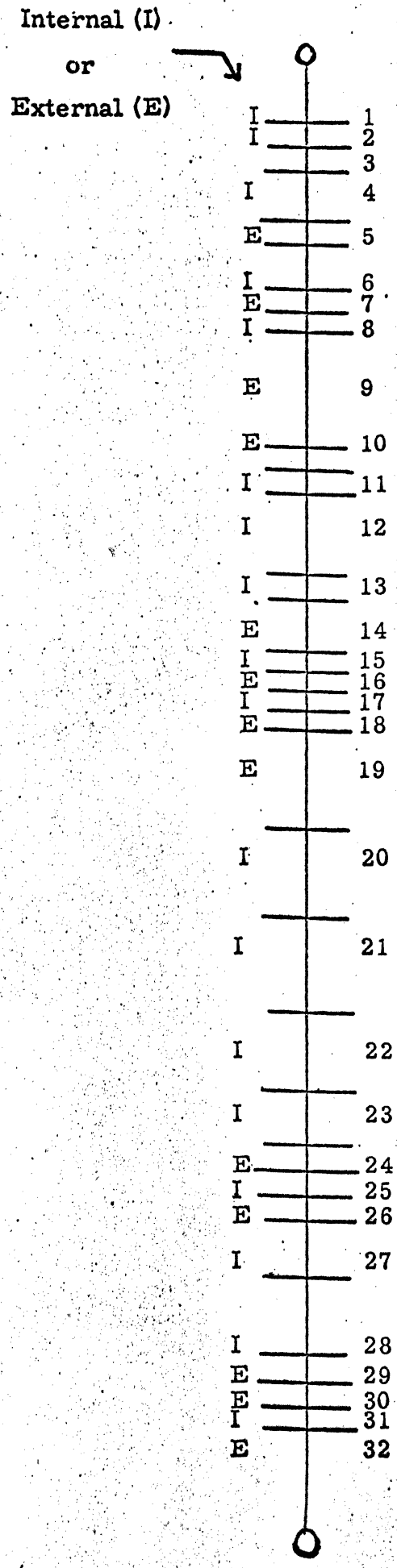
Diagnostic of Level 2

Diagnostic of Level 3

BLOCK DATA subprogram *

Notes: * applies if subprogram only

The System Communication Region is also available to all parts of the compiler.



Executable Source Program

Table Handling Method (provisional)

The buffer table method will be used modified as follows:

(a) Buffers will be grouped

(b) Groups may be characterized by either

(1) common to a phase

(2) same buffer size

For example, there might be four groups:

Group 1	Phase I only--tables with large buffer size
Group 2	Phase I only--tables with small buffer size
Group 3	Phase II only--tables with small buffer size
Group 4	Phase I - II tables with medium buffer size

Compiler Tables

Reference Tables

DIMENSION

COMMON

Equivalence

Assignment Tables

Explicit Data Names (single and double)

Implicit Data Names (single and double)

EFN

Transfer vector

Literals (single and double)

Subscript Temporaries (single and double) } Counts

ASF Temporaries (single and double) } Counts

Algebra Temporaries(single and double) } Counts

Program points

Local Work Tables

ASF Dummy

Table formats

EFN TABLE

EFN				U Definition #		Definition Sequence
bb171						
bbb73						
b1029						
bbbb1						

(1 bit used if last in the range of a DO)

30 bits spare spare

Explicit DATA TABLE

NAME	Type	Relative Address
VECTOR	R	Assigned
MATRIX	I	
DOUBLE	D	at
COMPLEX	C	end of
LOGIC	L	Phase
FUNCT	F	IA

spare bits

Implicit DATA TABLE

NAME	Type	Relative Address
J	I	Assigned as
K	I	encountered in
TEMP1	R	Phase IB or as
STEM	C	generated in
BOOLE	L	Phase IC.

spare bits

Program Point Table

(Phase IB) (Phase IC-Scan 1) (Phase IIB) (Phase IC-Scan 8)

Program Point#	Type	Synonym, argument #, etc.		Final address	Sequence number	Increment
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						

Types

Dummy Argument
Subscripted variable
Equated Subscripted Variable
Dimensioned Dummy Variable
True-logical intra-sequence jump
False-logical intra-sequence jump

External Functions

Defining Program (or subprogram)

```
EXTERNAL    BETTY, BARB
CALL DORIS (BETTY)
CALL DORIS (BARB)
```

Coding

Barb	BCI	BARB
Betty	BCI	BETT
Doris	BCI	DORIS
	TSX	DORIS
	PZE	BETT
	TSX	DORIS
	PZE	BARB

Transmitting subprogram

```
SUBROUTINE DORIS (DUMMY)
CALL DUMMY
```

Dummy	BCI	Dumm
Prolog	CLA	1, 4
	STA	Dumm
"CALL"	TSX	Dumm

Executing subprograms

```
SUBROUTINE BETTY
    }
RETURN
END
SUBROUTINE BARB
    {
RETURN
END
```

Processing External Functions

(see page C 8)

Compiler--

Defining Program flags BARB and BETTY in transfer vector as external (E).

Transmitting Program flags DUMMY as a dummy (D).

Loader

Does not collect "(D)" subprograms

Collects DORIS, BETTY, BARB

Will reject (as diagnostic) OVERLAY card with name corresponding to (E) flagged routine.

Literal Processing

A. In the following cases, no "literal pool" is established and redundancy is not recognized:

1. Data in DATA statements
2. FORMATs
3. Hollerith Arguments of CALLs

B. Two literal pools will be established for Implicit Single Data and for Implicit Double Data.

C. Exceptions:

TRUE and FALSE are standard.

Integer implicit variables L 32768 are placed in the NAME table as BCD numbers left adjusted with trailing blanks (e. g. VECTOR, 17, MATRIX, 4000 are all names).

Diagnostic Processing

1. All diagnostic messages will be classified as immediate or delayed.
2. An immediate message will be written out by the section of the compiler which discovered it.
3. A one-word "message" is stacked in the Compiler Communication Region whenever a delayed diagnostic is discovered.
4. Immediate diagnostics are:

   Trivial messages

   Phase II messages

   All others are delayed.
5. Depending on the permissiveness level of the delayed message(s), the compiler control may advance to the Diagnostic Message Writer which follows Phase II on the IBSYS tape. After the message-writer is read in, the system tape repositioning can be started.
6. The one-word message will have a code # in the address field, sequence number is the decrement field (if applicable) and permissiveness level in the tag field.

Basic Definitions

A File is composed of Sequences.

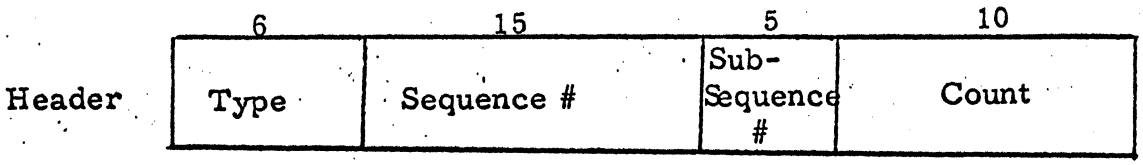
The internal file sequences are successively in Format A, Z and B.

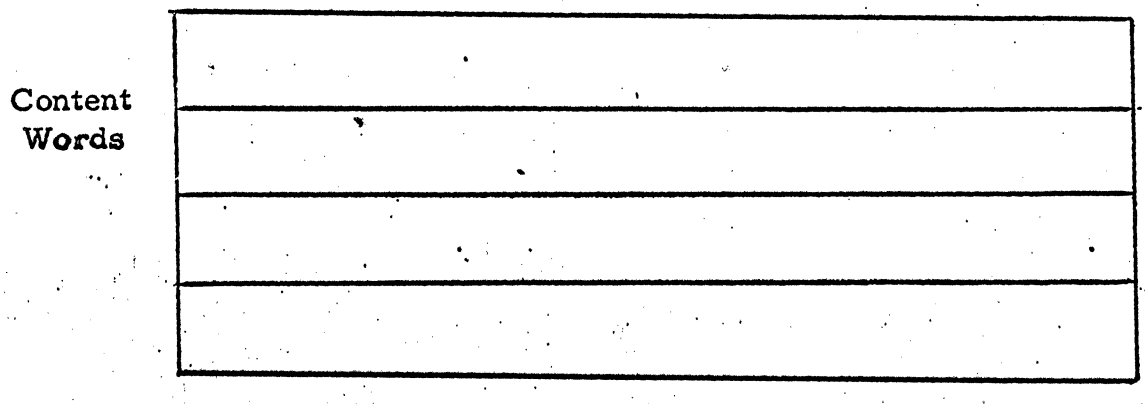
The external file sequences are always in Format Z

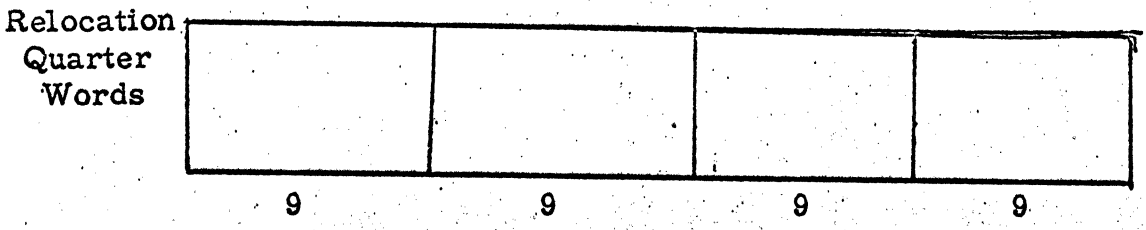
A sequence is a homogeneous contiguous string of information which is processed similarly.

The original assignment of sequence numbers limits the program string to 2047 sequences.

GENERAL SEQUENCE FORMAT







Classes of Formats.

1. A - Format.

   Initial form of internal file. Created by Phase IB to hold information from the source program.

2. Z - Format.

   Binary format with proto-addresses. Exists at the end of Phase I.

3. B - Format.

   Binary format with real addresses. Created by Phase II from Z - format.

4. S - Format

   Word format used by arithmetic and logical statement processors.

FORMAT Z

6	11	4	5	10	
Type	Seq.	0000	00000	Count	Header

Content--7090 commands with photo-addresses in A or D

Type	Code	RQW D/A	
A Title		UD Seq +	
B Common Block Names	00	UD Seq -	
C Counts	01	D Seq +	1-7
D Transfer Vector	10	D Seq -	
E Explicit Unique	11	Exp Data	
F-J Implicit Unique		Imp Data	
K Program String			
L Arith Stat. Functions		Function	5
		I O Unit	6
M-S Labelled Common Regions	Pseudo ops	BSS	7

Constant $> 2^{15}$ (?)

Formats of A, Z, B Sequences

(1) All have headers, content words

(2) RQWs are present in all A sequences

(3) RQWs are present in Z sequences:

J, K, L (prologue, program string, A.S.F. routines)

and in B sequences:

J (program string)

(4) RQWs precede in B format, follow in A and Z

(5) Content words have a set structure for each type of A format. Other formats contain unstructured contents.

(6) The minor subsequence field of the A format header is used for information bits (branch, fix left, etc.

(7) Only the sequence (11 bits) is originally assigned in A format.

Header Type Codes (all sequence formats)

A	Z*	B
A	Title	Preface
B	Counts	
C	Common Names	
D	Transfer Vector	Transfer Vector
E	Explicit Unique	Unique Data
F	Implicit Unique	
G	Implicit Unique (part 2)	
H	Formats	
I	Hollerith Arguments	
J	Prologue	Program String
K	Program String	
L	ASF routines	
M	Common 1	Common 1
N	Common 2	Common 2
O	Common 3	Common 3
P	Common 4	Common 4
Q	Common 5	Common 5
R	Common 6	Common 6
S	**	
T	DO	
U	SSV(single)	
V	SSV(double)	
W	subscript	
X		
Y		
Z	List Element	
1	Assignment	
2	Call Argument	
3	Arithmetic IF	
4	Logical IF	
5	ASF Definition	
6		
7		
8		
9		

* Used for sorting in Phase II -A

** Original assignment S for program string sequences containing an undefined EFN. changed to K in Phase II when EFN's Phase 1 are defined.

S GO TO 27

PHASE II

K

Proto-address/RQW Summary

	A/D	Mnemonic	Fortran IV Equivalent	Applicable Formats	
Locations in Program			also B1n		
Sequence Number (defined)	A	DSN	n An1-	ASZ	
(undefined)	A	USN	nZn	ASZ	
Intra-sequence +	A	STP	*+	Z	
Intra-sequence -	A	STN	*-	Z	
Program Point (Dummy)	A	PPD	nAA	ASZ	
Program Point (Subscripted Variable)	A	PPS		ASZ	B_1
Program Point (Dimensioned Dummy)	A	PDD		ASZ	
Program Point (True/False)	A	PPL			
Locations of Explicit Data					
Explicit Data	A/D	EXD	actual	ASZ	B_0
Unique Data	A/D	UED	name	ASZ	
Common Data 1-6	A/D	CD1 to CD6	"		B_{2-7}
Locations of Implicit Data					
Implicit Single Data*	A/D	ISD	actual name	ASZ	
Implicit Double Variables	A/D	IDD	actual name	ASZ	
Constants in Single Data Literal Pool	A/D	SDC	= n	ASZ	
Constants in Double Data Literal Pool	A/D	DDC	= nD	ASZ	B_0
Algebra Temporaries-single	A/D	ATS	= n	SZ	
Algebra Temporaries-double	A/D	ATD	= nD	SZ	
Arithmetic statement temporaries(all double)	A/D	AST	PP. n	SZ	
Subscript Calculation Temporaries	A/D	SCT		Z	
Locations of Transfer Vector					
Transfer Vector	A	TRV	actual name		B_1
RQW (indicative only)					
Function-open	A/D	OPF		AS	
-defined	A/D	ASF		AS	B
-subprogram	A/D	SPF		AS	format
Symbol-operator	A/D	OPR		AS	
Symbol-delimiter	A/D	DEL		AS	
Null	A/D	NUL		AS	
Dummy Argument		ARG			

Proto-addresses only appear in program string, prologue or ASF definitions.

Data Type: A-Format, Type T

Purpose: To hold data from statements of the form a DO b C = m_1, m_2, m_3; where a and b are sequence numbers, C is a fixed point variable; m_1, m_2, and m_3 are fixed point constants or variables. m_3 may be omitted, in which case computations involving it use the constant value one.

A Format, Type T

FORMAT:

Header

Field	A	B	C	D	E
Contents	T	a	1		
Bits	1 6	7 21	22	23 26	27 36

Content Words

Field	F	G	H	I
Contents		b		C
Bits	1 3	4 18	19 21	22 36

Field	J	K	L	M
Contents		m_1		m_2

Field	N	P	Q	R
Contents		m_3		null

Field	Bits	Contents
A	1-6	Header type code "T"
B	7-21	Sequence number for location of DO
C	22	1 to indicate FIXLEFT entry for C
D	23-26	Minor subsequence number
E	27-36	Number of content words (2 or 3)
F	1-3	Not used
G	4-18	Sequence number of range of DO. RQW: USN or DSN

A- Format, Type T

Field	Bits	Contents
H	19-21	Not used
I	22-36	Control Variable. RQW: EXD, ISD or ARG
J	1-3	Not used
K	4-18	Initial value of control variable. RQW: EXC, SDC, ISD, or ARG
L	19-21	Not used
M	22-36	Final value of control variable. RQW: EXD, SDC, ISD, or ARG
N	1-3	Not used
P	4-18	Increment of control variable. RQW: EXD, SDC, ISD, or ARG. If m_3 is not present in the statement, the third content word is not present in the A-Format.
Q	19-21	Not used.
R	22-36	Not used. RQW: NUL

Data Type: A-Format, Type U or V

Purpose: To hold data from subscripted variables of the form $A(e_1 * s_1 + f_1)$, $A(e_1 * s_1 + f_1, e_2 * s_2 + f_2)$, or $A(e_1 * s_1 + f_1, e_2 * s_2 + f_2, e_3 * s_3 + f_3)$; where A is the name of a dimensioned variable, s_i are fixed point variables, e_i and f_i are fixed point constants.

A-Format, Type Uor V

Header

Field	A	B	C	D
Contents	U or V			
Bits	1 6	7 21	22 26	27 36

Content Words

Field	E	F	G	H	I	J
Contents			A			S_1
Bits	1	23	4 18	19	20 21	22 36

Field	K	L	M	N
Contents		e_1		f_1
Bits	1 3	4 18	19 21	22 36

Field	P	Q	R	S	T
Contents			d_1		s_2

Field	U	V	W	X
Contents		e_2		f_2

Field	Y	Z	AA	AB	AC
Contents			d_2		s_3

Field	AD	AE	AF	AG
Contents		e_3		f_3

A-Format, Type U or V

Field	Bits	Contents
A	1-6	Header type code U or V
B	7-21	Sequence number for location of subscripted variable
C	22-26	Minor subsequence number
D	27-36	Number of content words (1 to 6)
E	1	1 if $e_1 = 1$, $f_1 = 0$ and the content word containing fields K-N is deleted. O otherwise.
F	2-3	Not used
G	4-18	Dimensioned variable. RQW: EXD or ARG.
H	19	Not used
I	20-21	Number of dimensions (1, 2 or 3).
J	22-36	Subscript variable of first dimension RQW: EXD, ISD, ARG or NUL.
K	1-3	Not used
L	4-18	Multiplier of first dimension. RQW: SDC or NUL
M	19-21	Not used
N	22-36	Addend of first dimension. RQW: SDC or NUL
P	1	1 if $e_2 = 1$, $f_2 = 0$ and the content word containing fields U-X is deleted. O otherwise.
Q	2-3	Not used
R	4-18	Size of first dimension RQW: ARG or ISD
S	19-21	Not used
T	22-36	Subscript variable of second dimension
U	1-3	Not used
V	4-18	Multiplier of second dimension.
W	19-21	Not used
X	22-36	Subscript variable of second dimension.
Y	1	1 if $e_3 = 1$, $f_3 = 0$ and the content word containing fields AD - AG is deleted. O otherwise.
Z	2-3	Not used
AA	4-18	Size of second dimension
AB	19-21	Not used.

A-Format, Type U or V

Field	Bits	Contents
AC	22-36	Subscript variable of third dimension.
AD	1-3	Not used
AE	4-18	Multiplier of third dimension.
AF	19-21	Not used
AG	22-36	Addend of third dimension.

Sample formats (A)

Content Words

LISTS and EXPRESSIONS

(The expressions exlude subscripted variables and functions which presumably have been placed in separate sequences with their 20 bit designation placed in the "upper".)

A	Sequence	6
Subsequence for B (I)		/
null		(
C	(pa)	+
D	(pa)	*
E	(pa)	)
A	(pa)	=

A = B (I) / (C + D * E)

W	Sequence	7
A (pa)		,
B (pa)		,
null		(
subsequence for C (I)		,
I (pa)		=
1 (constant)		,
12 (constant)		)

READ (3, 4) A, B, (C(I), I=1, 12)

Sample formats (A)

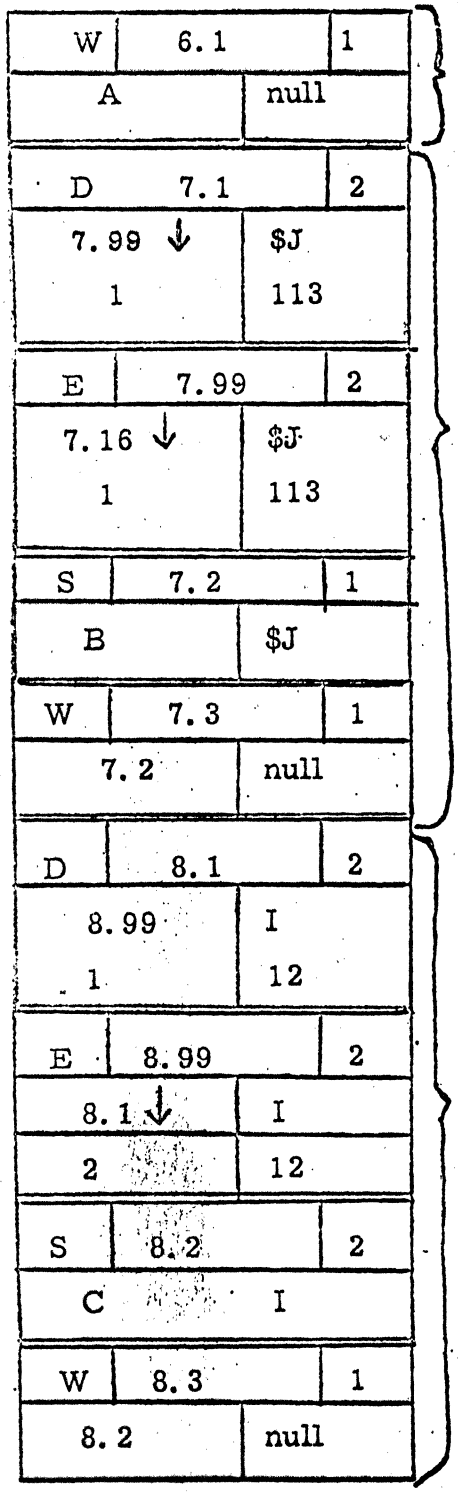
Content Words

LISTS and EXPRESSIONS

(The expressions exlude subscripted variables and functions which presumably have been placed in separate sequences with their 20 bit designation placed in the "upper".)

A	Sequence	6
Subsequence for B (I)		/
null		(
C	(pa)	+
D	(pa)	*
E	(pa)	)
A	(pa)	=

A = B (I) / (C + D * E)



DIMENSION B(113), C(24)
READ (3,4) A, B, (C (I), I = 1, 12)
(list portion only)

This is a word format and not a sequence format. It is used during the expression manipulating scans of Phase III C.

RQW	Mode	flag	pa or symbol

(1) The RQW and pa portion are derived from format A.

(2) S format serves as a "bridge" finally leading to format Z.

(3) The Mode field indicates the hierarchical power of the symbol, when it is an operator.

Logical IF format example

internal

J	Sequence 13	3
false sequence no. (15)		null
A		.GR.
B		null

IF (A.GR.B), ASSIGN 12 TO N

K

external

Z	Sequence 14	3

CALL format example

K

external

Z	Sequence 213	1
0740004 (pa Boxer)		

internal

C	Sequence 214	1
A		null
S	Sequence 215.1	1
B		(I)
C	Sequence 215	1
Sequence 215.1		null
C	Sequence 216	1
3 (constant)		null
C	Sequence 217	1
reference to seq ~~JOB#~~ in format Z.		

external

J	Sequence 001	1
MATTER		

CALL BOXER (A, B(I), 3, 6H MATTER)

Implicit data

Program Points

These are locations within sequences that are referenced:

(1) In subscript computation

(2) In logical expressions ("true" and "false" entries)

(3) In argument insertion by the prologue of a subprogram.

A table is started in Phase IB and is built upon and modified in the Phase IC. Finally, Phase IIB uses this table in assigning addresses.

The format of the table is shown on page C 7.1.

Manipulating the Internal File

The following operations (at least) will be required:

Build (initially)

Merge two (or more) files

(Sort?)

SCAN- {forward / backward} {find every sequence / find only selected sequence or sequences}

(Virtual) Insert

Delete

Replace

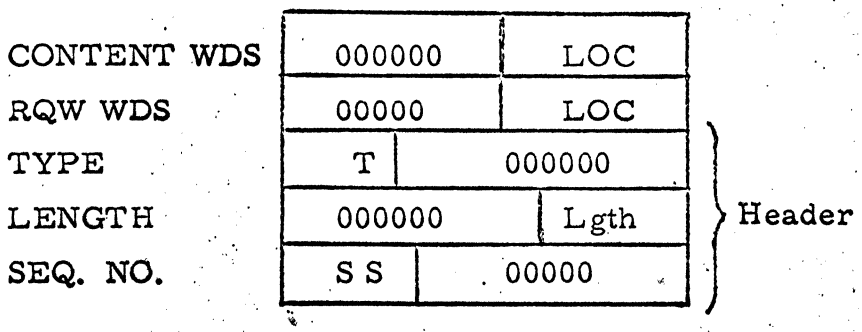
Read in

Write out

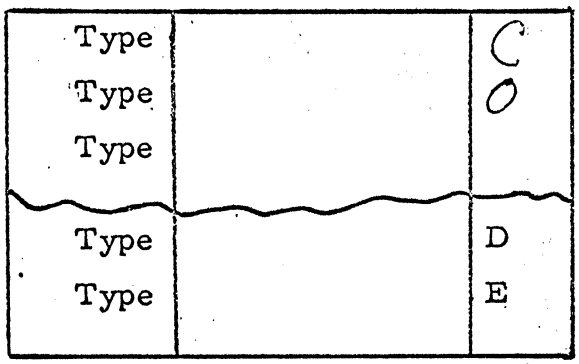
The file will consist of sequences in A, B, S or Z format. In any event, the I. F. Manipulate Routine will be informed which format to use.

Internal File Manipulating Routine

Communication Region

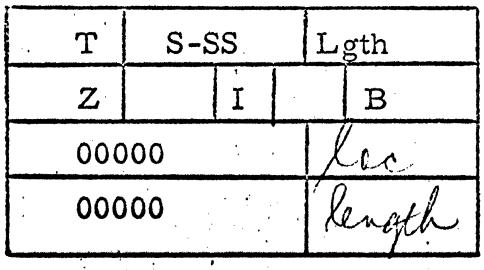


Logic Packet



CODE	
000	AND
001	OR
010	NOT
111	Separation

IMF Routine Control Region



(format, internal/external, forward/backward)

Internal File Manipulating Routine

```
TSX   BUILD, 4
TSX   INSERT, 4
TSX   DELETE, 4
TSX   CHGTYP, 4
```

User fills communication region and then calls IMF Routine

```
TSX   SCAN, 4
PZE   LOC LOGIC,, LGTH LOGIC
TSX   RESCAN, 4
```

IMF Routine fills communication routine

The routine will also sort, merge, read and write.

Internal File Manipulating Methods

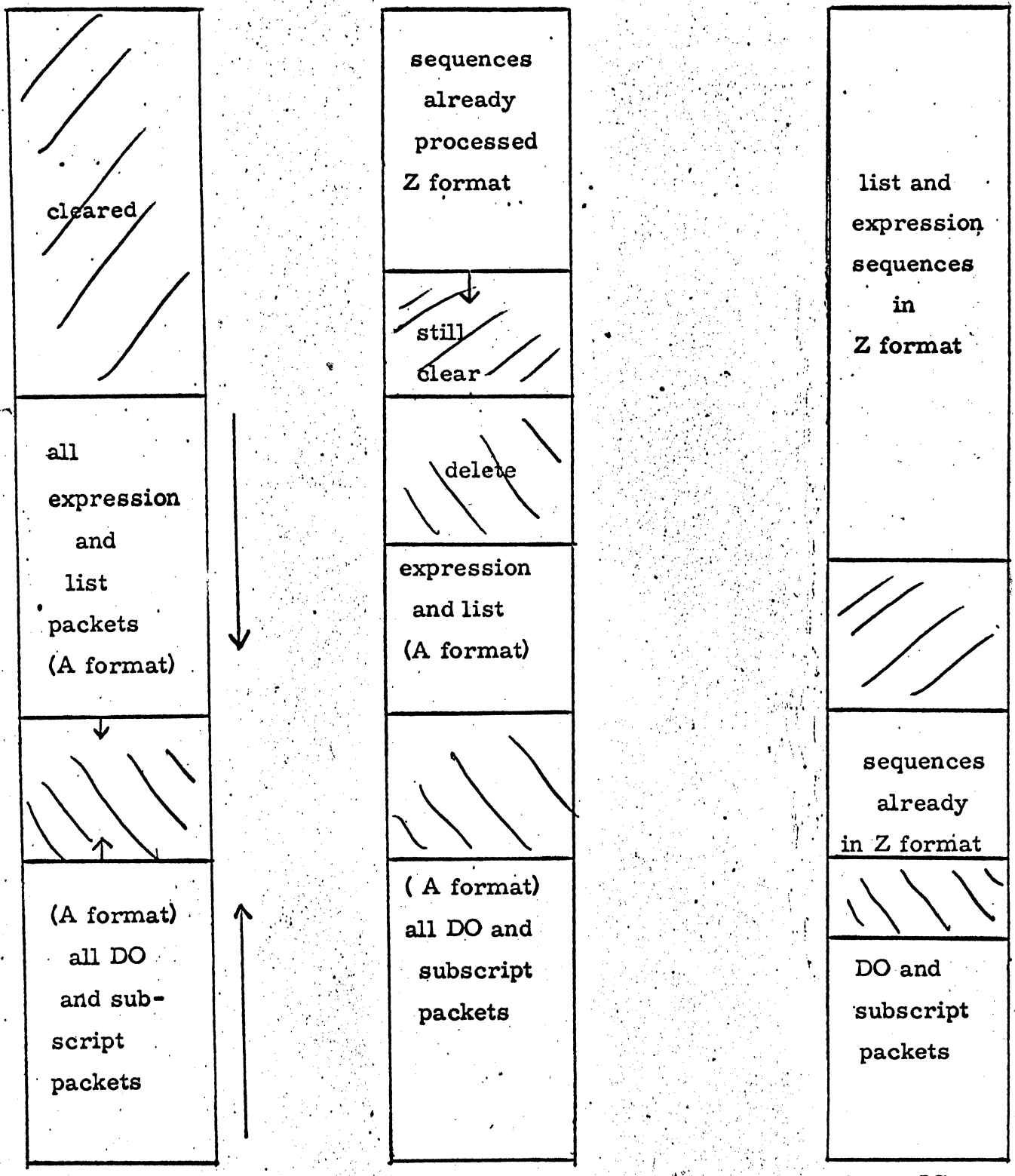
A. Sequencing by number (logical)

1. Original numbers--11 bits + 4 zeros
2. Insert numbers (between sequences) the 4 bits are used
3. Dividing numbers--The 5 subsequence bits

B. Sequencing by linking (physical)

1. Actual physical exchange is possible
2. Special link-headers can be used linking to an addendum file
3. Special delete headers can be used
4. A sequence table can be maintained and rearranged. Accessing will be by indirectly addressing this table.

Internal File Building -- Phase I



RQW Manipulation

For each format, for the program string, arithmetic statement functions and open functions, there is an associated "Relocation Quarter-Word" of 9 bits (RQW).

The interpretation of the RQW varies from format to format but its connection to the associated full word does not.

Thus, the internal file manipulating routine must also build, scan, etc. with these words connected.

FTCA -- Phase I

IA Initialize; Process non-executable statements including FORMATs.

IB Process executable statements and Formats to internal or external file till END statement.

IC Process internal file (11 Scans!)

~~ID Write Implicit Data Tables to code.~~

Internal file and Tables remain in core.

Diagnostics throughout

G 1
7/10/63

Phase I - Narrative Description

A. After suitable initialization, the specification statements are read in and processed to tables (Explicit Data). Where code is generated (subprogram statements, DATA), It is written out on the external file. The Explicit Data allocation is made.

B. Following processing above, the remainder of the source program is handled. A specification statement is now a diagnostic. The table building continues (EFN, Implicit Data, etc.); FORMATs and many types of statements produce sequences for the external file. The remainder build sequences in the internal file in FORMAT A.

C. Next, the internal file processor is brought in and the I. F. converted to code sequences and optimized in (11 Scans). The Implicit Data Table is converted to code sequences for the external file

Diagnostics are issued throughout as they are encountered.

At the end of the Phase, tables and the internal file remain in memory. The external file of sequences is rewound.

Phase I

Internal File	External File
Statements containing expressions (including CALL argument, IF (~~part 1~~), Arithmetic Statement Function definitions and assignment statements).	All others

Phase I (General)

External File Order

(Phase IA) Explicit Data and Reserves
FORMATs
DATA
} in any order

(Phase IB) Executable sequences
FORMATs

Processing of Functions in Expressions

Phase IB--identifies functions, determines the type and replaces them with pa s and RQWs.

Phase IC--Scan 2--Converts to Format S.

Phase IC--Scan 3--Produces proper outputs. Examples:

A = FUNCT(J+K, B)	A = FUN (FUNCT(J+K), B)
$J+K \longrightarrow T_1$	$J + K \longrightarrow T_1$
$FUNCT\ (T_1, B) \longrightarrow A$	$FUNCT\ (T_1, B) \longrightarrow T_2$
	$FUN\ (T_2) \longrightarrow A$

Phase IC--Scan 8-- Produces proper Z format according to type (open, ASF, sub-program).

Parametrizing Subprograms

Whenever a name is a dummy, a program point is generated. It is placed in the program point table with the argument number. (Phase IB)

In Phase IC - Scan 8, the proto-address of the instruction to be filled is placed in the program point table (sequence number in Z format and increment).

The prologue is created in Phase IC Scan 11. The sequence numbering is such that PhaseIIA will sort it ahead of the program string so that it is executed first.

Exception: If a dimensioned variable is also a dummy, special treatment is required. See page ____.

Subroutine: CHSCAN

Purpose:

1. Eliminate blanks (leading, embedded, trailing)
2. Detect illegal characters
3. Detect equal sign
4. Detect presence of parentheses
5. Detect any imbalance of parentheses (indicates whether left or right parentheses are the cause of imbalance)

Entry parameters:

1. Origin of unpacked statement
2. Origin for packed statement

Exit Conditions:

1. Return to 3, 4
2. Index reg 1 will contain
   - 0 - no errors
   - 1 - unequal number of paren.
   - 2 - illegal character
   - 3 - more than one equal sign
3. If IR1 = 1

   PAREN = + x { x number of extra left paren }

   = - x { x number of extra right paren }
4. If IR1 = 3

   EQS W = number of equal sign

   if > 3 always an error

   if 2-3 can be I/O hst

   if 1 can be I/O

   DO

   assignment

Memory Requirements:

64 Table

85 + Program

Timing - in cycles

Init	Per Charac.	Type Charac.
33	16	Blank
33	31	Parenthesis
33	16	Illegal
33	28	Equal sign
33	16	End charac. (77)
33	24	Normal (not any above)
22	8	Word of blanks
	7-8	Epilogue (after ocur. of end ch) for each charac. that must be left justified in Pack area.
	8	Equal sign switch (once only)
	2	Paren switch (once only)
	1	Illeg switch (once only)
	4	Return Transfer (once only)

Phase IB

Building the Internal File

DOs — fairly straightforward packing to format A

ASF definitions
Assignment statements
IF
CALL arguments

(a) isolate SSV and functions
(b) resolve the specific type of (a)
(c) pack the remainder

I-O lists — each element is to be assigned a sequence number. Create subsequences for SSVs and implicit DOs. (see page D 7 for an example).

Proto addresses must be supplied in format A.

Sequence Number Assignment

Phase IB

(Assumption: Arithmetic statement function definitions have their own sequence number rules set by Phase IA. The type code is L.)

A. The following <u>always</u> start a sequence:

(1) A numbered statement

(2) The statement following a logical IF

B. Successive statements destined for the same file with the same type code in the Header do <u>not</u> start a sequence; e.g.:

```
END FILE 3
REWIND 3        } 1 sequence

J = K
A = B + C       } 1 sequence
D = X (5)

ASSIGN 27 to J
GO TO 30        } 1 sequence
```

C. The following statements always generate more than one sequence:

Logical IF

IO statements with lists

CALL statements with arguments

D. Original sequence assignments are for the high order 11 bits in the sequence field.

Subscript variable packets receive subsequence numbers.

Implicit DO packets (from a list element) receive subsequence numbers.

E. The "statement" immediately following the logical IF receives a sequence number (or numbers).

See A formats for internal file

See Z formats for external file

Initial Assignment of Sequence Number to DO Endings

Since the last statement in the range of a DO may itself be assigned, several sequence numbers (e.g. --a logical IF or an I-O list), a special entry in the EFN table must be made.

Given:

```
        DO    20   I=1, 10
        GO TO  20
20      READ (3, 12) A, B, C, D, E (I)
```

The EFN 20 receives two entreies in the EFN table. The flag used to identify a DO end EFN distinguishes the entries.

EFN	UD #	DS#	Flag
20	6	77	1
20	9	71	

RUN IC

General Plan

SCAN	format	Type	Range	Function
1	A	D, S, 2	file	fill in UD#; eliminate redundant SS calculation
2	A⟶S	expressions	sequence	convert to S format
3	S	"	"	generate algebra via Z stacks
4	S	"	"	eliminate redundant subexpressions
5	S⟶Z	"	"	convert, inserting open functions, powers, able and complex
6	Z	"	"	eliminate redundant load and stores, optimize AC/MQ
7	Z	"	"	eliminate redundant temporaries
8	Z	statement lists	"	specialize coding
9	A	D, S, 2	file	DO nest analysis
10	A⟶Z	D, S, 2	nest	code generation
11	Z	S, Z expression		

Phase IC

Scan 1

Examine all T sequences replacing the undefined sequence numbers for beta with defined sequence numbers.

Examine all contiguous U and V packets for identical subscript calculations where found, delete one and make suitable change in the expression or list packets that reference it. All of the above must have the same sequence number.

Phase IC

Scan 2

Move, sequence by sequence, all packets containing expressions.

Convert to S Format--

Pack (and modify) RQW in S word

Add hierarchical power level if operator

type code if operand

delimiter code if delimiter

Phase IC

Scan 3

Words in S format are taken from a sequence in work area 1 and transferred to work area 2 with intermediate residence in the main stack. There is also a Control Stack. The work area 1 contains the words in reverse order, e.g. (Y = A + B) Work 1

B
+
A
=
Y

)

Temporaries (in S format) are created in the process, legal mode combinations checked and decisions regarding exponentiation made.

The basic algorithm:-

Word is operand--push down to Main Stack.

Word is operator--not inside a function argument

Lower than top of Control Stack--pop up both stacks to first operator of higher value.

Equal or higher--push down to both stacks.

Word is operator--inside a function argument and lower than top of Control Stack--pop up both stacks to first comma or right parenthesis.

Word is delimiter--

")" - push down both

, in a function - push down both

= - pop up both to bottom of stack

(- pop up to first right parenthesis

Phase IC

Scan 3 cont'd

Whenever the first symbol in a packet is commutative and the second operand is a temporary cell, exchange the first with the second.

Phase IC

Scan 4

Redundant subexpressions are identified by examining all generated packets. First, test for equivalent number of entries, then for equivalent contents. If all operands (except the last) are temporaries, give them a special analysis. If only the last operand is a temporary and, otherwise, equality exists, the elimination condition exists.

To eliminate, keep the first encountered and store into the last (output) operand of all other (eliminated) packets.

There may be more than two of the same subexpressions and there may be more than one subexpression with redundant appearances.

Phase IC

Scan 5

Conversion To Z Format

The S format words in WORK 2 are now converted to Z format. Here the specific 7090 or 7094 code is originated. Double precision and exponentiation are treated now and open functions parametrized and inserted.

Phase IC

Scan 5

Converting Logical Expressions to Code

S format to Z format

Assumptions:--

1) Phase IB identified .NOT. and associated it with the logical variable.

Multiple .NOT.s must also be processed.

T = .NOT. A .OR. B	The .NOT. is associated with A
T = .NOT. .NOT. A	The .NOT.s are dropped

2) Phase IC Scan 3 makes no special provision for logical processing.

Actions:-- For every element--

a) Normal form	NZT element TRA F (false exit)
b) When preceded by .OR.	ZET element TRA T (true exit)
c) Negated	reverse the first op-code of the corresponding pair
d) IF (only 1 element)	CAL element TZE F
e) Constants	.TRUE. = 1 .FALSE. = 0
f) Logical IF (ending)	F. TRA false sequence true sequence follows

g) Relational Operators

```
CLA   Arith. Expression 1
SUB   Arith. Expression 2
Suffix
```

or

```
CLS   Arith. Expression 1
ADD   Arith. Expression 2
Suffix
```

reversed form

Suffix	.GE.	TMI	F.	A.GE.B	T+
	.LE.	TPL	F	A.LE.B	T-
	.GT.	TZE	F.	A.GT.B	T+0
		TMI	F		
	.LT.	TZE	F.	A.LT.B	T-0
		TPL	F		
	.NE.	TZE	F.		
	.EQ.	TNZ	F.		

h) Logical Arithmetic Statement Function (ending)

T.	CAL	*
	TRA	9999ZZ
F.	ZAC	
9999ZZ	AXT	**,4
	TRA	1,4

i) Relation .AND. Relation

Relation coding with branch to False

Relation coding with branch to False

j) Relation .OR. Relation

Relation coding with branches changed to True

Relation coding with branch to False.

k) Definition of an "opposite exit":

The inverse of the address of the last branch in the coding for a subexpression.

l) Processing logical temporaries: A op B $\longrightarrow$ Temp.

(i) The opposite exits go to Temp instead of T. or F.

(ii) Temp. is entered in the program point table.

m) Processing logical temporaries Temp. op C$\longrightarrow$D:

The temp. is simply eliminated.

n) Processing logical temporaries $Temp_1$ op $Temp_2 \longrightarrow$ D:

-to be specified

Phase IC

Scan 6

Optimization

Here the sequence under development is scanned to eliminate superfluous loads and stores and to arrange AC/MQ optimization.

The algorithms are to be stated later.

Phase IC

Scan 7

Elimination of Redundant Temporaries

Here the sequence is scanned to eliminate the assignment of superfluous temporary cells.

The algorithm is to be stated later.

Phase IC

Scan 8

Specialization and Lists

Here the specific needs of--

CALL argument

IF (arithmetic)

IF (logical)

assignment statements

arithmetic statement function definitions

as well as of elements in a list are processed.

At the conclusion of this scan, the header is restored, the new count inserted and, as a Z format sequence ~~(but with the original A format type code)~~, it is placed in the internal file replacing the sequence used in Scan 2.

Phase IC

Scan 9

Processes D packets. Analyzes DO's on nest by nest basis.

1. Get all DO's in a nest.
2. Assign level numbers.
3. Check proper DO nesting.

Nesting Rule: If the range of a DO includes another DO, then all of the statements in the range of the latter must also be in the range of the former. That is, let a_1, a_2 be the location of two DO statements with ranges b_1 and b_2 respectively. Then, if a_2 is greater than a_1 and less than b_1, then b_2 must be not greater than b_1.

4. Assign index registers to control variables.

Let n = number of index registers available. Index registers are assigned to control variables according to level number of the DO as follows:

Level Number	Index Register
1	1
2	2
...	...
n	n
n+1	1
n+2	2
...	...

Phase IC

Scan 10

1. Produce Z format coding for the DO start and DO end. If m_1, m_2 or m_3 is symbolic appropriate coding must be generated.

2. Sequencing DO commands. Beginning of DO. Sequence number a. Non-repeating sequences get subsequence numbers A - G. Repeating sequences get subsequence numbers H - P. End of DO. Sequence number b. All sequences get subsequence number P so they follow any sequences generated for statement b (by algebra, etc.). Minor subsequence numbers 0-15 are assigned if several DO's end at the same statement.

3. Setting up DO parameters.

a DO b C = m_1, m_2, m_3

a) Constant parameters.

```
aA  AXC   m1 ,  X(C)        Non-repeating
    STZ   C
```

```
aH  FSCA  C,    X(C)        Repeating
    SXA   bP0 ; X(C)        (If save needed)
```

```
bP0 AXT   **,   X(C)        (If needed).
```

```
bP1 TXI   *+1,  X(C),  -m3
    TXH   aH,   X(C),  -m2-1
```

Phase IC

Scan 10

b) Variable parameters

```
aA   LXC   m1,   X(C)
     STZ   C
```

```
aB   LXC   m3,  4
     SXD   bP1, 4
     LXC   m2,  4
     TX1   *+1, 4, -1
     SXD   bP1+1,  4
```

```
aH   FSCA  C,   X(C)
     SXA   bP0,  X(C)
```

```
bP0  AXT   **,  X(C)
```

```
bP1  TXI   *+1, X(C), **
     TXH   aA,  X(C), **
```

X(C) is index register assigned to control variable C.

4. Saves and restores are inserted in the outer DO's when insufficient index registers are available. The scheme is that lower levels have precedence over higher levels so that inner DO's will not have save and restore commands.

Let L = Highest level of a completely nested set of DO's.
n = Number of index registers available.

Then DO's with level L, L-1,....., L-n+1 do not require saving of index registers. DO's with level L-n, L-n-1, ..., 1 require saving index registers.

All completely nested sets of DO's in a nest must be examined.

5. A-Formats of type T, U and V are processed. Addresses are computed for subscripted variables.

General form of subscripted variable:
$A(e_i * S_i + fi)$ $i=1, \ldots n$, $1 = n = 3$.
Address is $A + \sum_{i=1}^{n} (e_i S_i + f_i - 1) g_i$;
$g_1 = 1$, $g_2 = d_1$, $g_3 = d_1 d_2$;
or $A + h + \sum_{i=1}^{n} e_i g_i S_i$,
where $h = \sum_{i=1}^{n} (f_i - 1) g_i$

Phase IC

Scan 10

6. Address computation of subscripted variable not in a DO:

6.1 n = 1. Address of $A(e_1 * S_1 + f_1)$ is

$$A + h + e_1 S_1 \quad , \quad h = f_1 - 1$$

a. $e_1 > 1$

```
LDQ    S1
VLM    e1, 15
PAX    ,  4
TX1    *+1, 4, A+h
SXA    Z, 4
```

Z OP **

b. $e_1 = 1$

```
LXA    S1, 4
TX1    *+1, 4, A+h
SXA    Z, 4
```

Z OP **

6.2 n = 2. Address of $A(e_1 * S_1 + f_1, e_2 * S_2 + f_2)$ is

$$A + h + e_1 S_1 + e_2 d_1 S_2 \quad , \quad h = (f_1 - 1) + (f_2 - 1) d_1$$

$e_1 > 1$	$e_1 = 1$
LDQ S_1 VLM (e_1), 15	CLA S_1

a. d_1 constant

$e_2 d_1 > 1$	$e_2 d_1 = 1$
STO T LDQ S_2 VLM $(e_2 d_1)$, 15 ADD T	ADD S_2

```
PAX  ,  4
TX1  *+1, 4, A+h
SXA  Z,  4
```

Z OP **

b. d_1 variable

```
STO   T
LDQ   S2

VLM   d1, 15
ADD   T
```

$e_2 > 1$	$e_2 = 1$
XCA VLM (e_2), 15	null

$f_2 = 0$	$f_2 = 1$
SUB d_1	null

$f_2 = 2$	f_2 2
ADD d_1	STO T LDQ d_1 VLM (f_1 -1), 15 ADD T

```
PAX,  4
TX1   *+1, 4, A+f1 -1
SXD   Z,  4
```

Z OP **

6.3 n = 3. Address of $A(e_1*S_1+f_1,\ e_2*S_2+f_2,\ e_3*S_3+f_3)$ is

$$A + h + e_1S_1 + e_2d_1S_2 + e_3d_1d_2S_3 ,$$

$$h = (f_1 - 1) + (f_2 - 1)\,d_1 + (f_3 - 1)\,d_1d_2$$

$e_1 > 1$	$e_1 = 1$
LDQ S_1 VLM (e_1), 15	CLA S_1

a) d_1 constant

$e_2d_1 > 1$	$e_2d_1 = 1$
STO T LDQ S_2 VLM (e_2d_1), 15 ADD T	ADD S_2

i) d_2 constant

```
STO  T
LDQ  S3
VLM  (e3d1d2), 15
ADD  T
```

```
PAX  ,  4
TX1  *+1, 4, A+h
SXD  Z,  4
```

K2.12.8

Z OP **

7/29/63

ii) d_2 variable

```
STO   T
LDQ   S3
VLM   (e3d1), 15
ADD   T
```

$f_3 = 0$	$f_3 = 1$
SUB d_1	null

$t_3 = 2$	$f_3 > 2$
ADD d_1	ADD $(f_3-1)d_1$

```
PAX , 4
TX1  *+1, 4, A+(f1 -1) + (f2 -1)d1
SXD  Z , 4
```

Z OP **

b. d_1 variable

Same as 6.2b , plus:

i) d_2 constant

Same as 6.3aii with d_1 replaced by d_2

ii) d_2 variable

$e_3 > 1$	$e_3 = 1$
STO T LDQ S_3 VLM e_3, 15 ADD T	ADD S_3

$f_3 = 0$	$f_3 = 1$
SUB (1)	null

$f_3 \geq 2$
ADD (f_3 - 2)

XCA VLM d_1 XCA VLM d_2

PAX , 4

$f_1 \neq 1$	$f_1 = 1$
TX1 *+1, 4, A+f_1 -1	null

SXD Z, 4

Z OP **

6.4 When dimensioned variable A is a subprogram parameter the instructions

```
PAX , 4
TX1 *+1, 4, A+h
```

become

```
ADD  L(A)
PAX  , 4
TX1 *+1, 4, h
```

6.5 If a subscripted variable is addressed through an index register, the instructions

```
PAX  , 4
TX1  *+1, 4, A+h
SXA  Z , 4
```

```
Z  OP **
```

become

```
PAC  , XR
```

```
Z  OP A+h, XR
```

6.6 Method of addressing subscripted variables not in a DO nest.

Index registers are used for subscripted variables in a sequence. When all available index registers are used, addresses are placed directly in command operands for remaining subscripted variables.

7. Subscripted variables in DO nests.

a. When a subscript is controlled by a DO control variable, the portion of the address due to the subscript is computed at the location of the DO.

b. When a subscript is not controlled by a DO control variable (relative constant case) the portion of the address due to the subscript is computed at the non-repeating portion of the outermost DO of the nest.

c. Method of addressing. Index registers are used to address subscripted variables under the following conditions.

i) A subscript variable is controlled by a DO.
ii) The subscripted variable is contained in the controlling DO.
iii) The assigned index register does not have to be saved and restored.

Phase IC

Scan 11

Generates Prologue

Converts Implicit tables to code in Z format and places them in internal file.

Indexing

Assignment of actual index registers must be delayed as long as possible.

Index register usage for other than DO's and SSV must be avoided if possible.

e.g. computed GO TO, Assign

Arbitrary "saves" and "restores" must be avoided.

Double precision and complex data must be indexed by doubling the normal increment.

Double Word Activities

double precision
- 7094 - hardware commands
- 7090 - special code generation (same as FTC)

complex -- special code generation

K4
7/10/63

Optimization of Object Code

Types to Consider

		F4A
(1)	Logic Transfers	✓
(2)	Pulling out from loops (e.g. Locating indexing return point as deep in inner loop as possible)	✓
(3)	Saving partial results (inter-statement)	✓
(4)	Saving partial results (intra-statement) (AC-MQ)	✓
(5)	Redundant index loading within a loop	✓

Examples

(1)
```
    IF(A) 17, 18, 17
17
```

(2)
```
      DO 12      I = 1, 10
      Y = 16.0
12  A(I) = B(I)
```

(3)
```
    J = 1
    K = J
```

(4)
```
    Y = SINF(A+B) / 1.0 - (SINF(A+B
```

(5)
```
    A(J+13) = B(J+13)
```

PROCESSING OPEN FUNCTIONS

In creating the internal file, Phase IB writes O-type sequences in format A whenever it finds the name in the Function Table coded as O.

In Phase IC, Scan 5, all skeleton coding is brought in at once from the system tape and the substitution performed.

A mechanism will be provided to users to add their open functions to their IBSYS tape.

Example

O	Seq.	Lgth
ISIGN		
K (pa)		
J (pa)		

Format A

ISIGN 5

INSIGN	Lgth
060000	par #1
032400	par #2
060100003721	
053600001042	
062100	par #1

Skeleton (Format Z)

060000	K(P.A.)	ISD
032400	J (P.A.)	ISD
060100003721		
053600001042		
062100	K(P.A.)	

Result (Format Z)

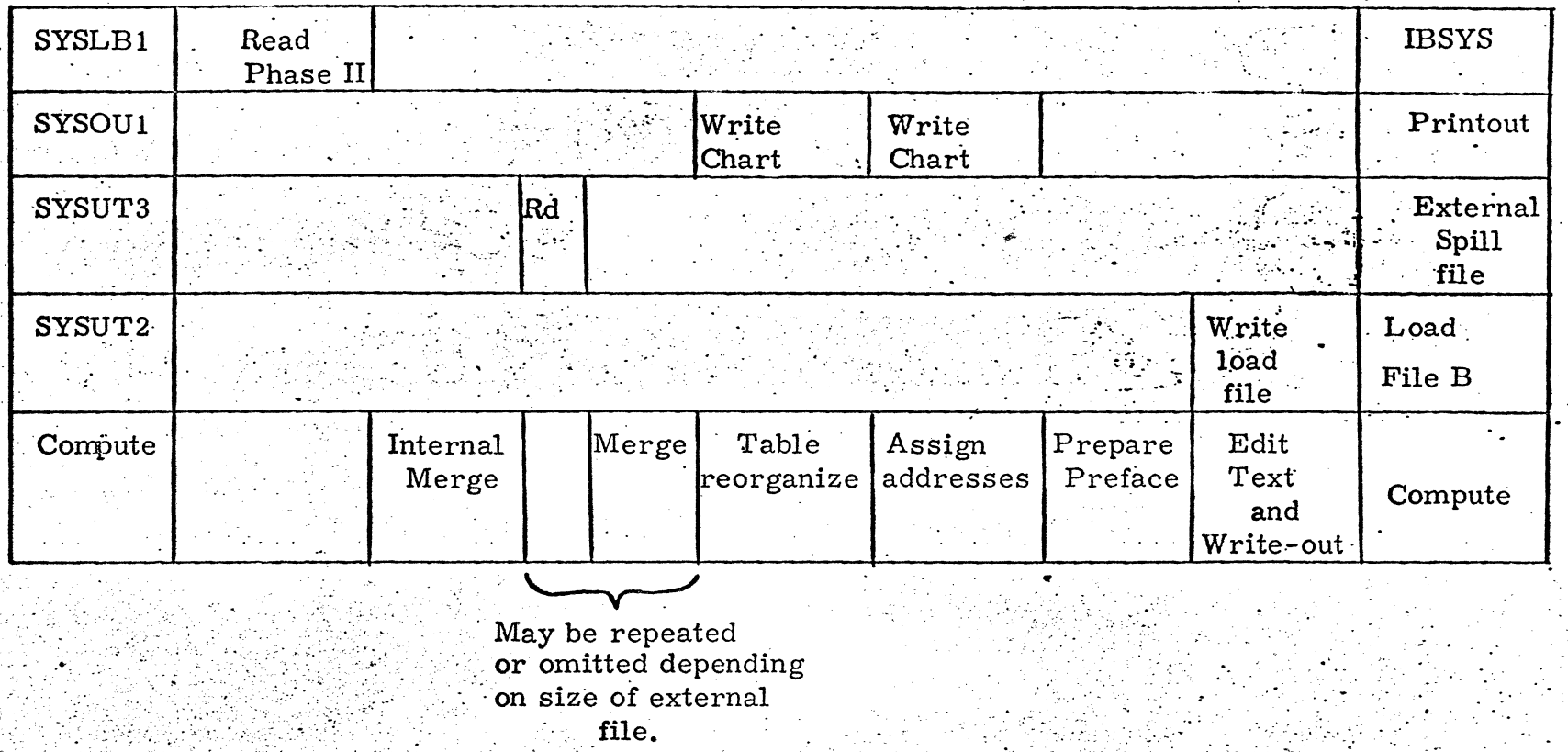
Y = ISIGN (K, J)

PHASE II

A	Replace undefined addresses
	Merge all sections of internal file
	Merge with external file
B	Reorganize tables-assign Data and Constant locations
	Build program string assignment table
	Assign address
C	Prepare preface, place in internal load file 1
	Output text to load file 2

The Chart preparation takes place concurrently with Phase IIB processing.

COMPILER -- PHASE II TIMING



PHASE II A

I While external file is being processed, S sequences have their USNs (undefined sequence numbers) replaced with DSNs (defined). Where no DSN exists-the diagnostic-"EFN xxxxx.is undefined" is issued. This procedure acts like "own-coding" within the merge as described below.

II The Merge program operates on the strings of sequence as follows:

(a) A list is prepared giving the first and last location of the various strings to be merged--

(1) If the external file was not written, the buffer location is given: MERGE N is set to zero.

(2) If it was written out, the buffer location is given but the number of records on tape is also placed in cell MERGE N.

(b) Based on the lengths, the merge-assignment routine sets up the merge.

(c) An n-way internal merge is carried out. RQW's are simply carried along with the sequences. The merge key is the first 26 bits of the sequence header.

(d) After completion, if MERGE N is not zero, a series of two-way internal merges takes place between the already merged file and the buffer MERGE N is reduced by 1 for each cycle until it is zero.

(e) Finally, the merge returns control giving the location of the first and last word of the merged file.

PHASE II A

Merge Details

Key: first 26 bits of header

Length: Exponded to include RQW's (where applicable)

Input to Each "Pass"

n + 1	for first pass	file pieces + buffer
Z	for subsequent passes	buffer + file

The input areas will be variable in word length and sequence counts. The first output area will be a clear area obtained by release of Phase I material that is no longer needed.

For each subsequent merge pass, the output area must be augmented by the buffer size.

Thus, allocation will be a difficult problem unless large amounts of workspace are available.

Phase II B

Table Organization

At the beginning of this section, the pertinent tables may be classified as follows:

Complete

TRV

ARG

Entries Completed -- proto-addresses only

ISD

IDD

SDC

DDC

DSN

PPX

Counts only

ATS
ATD
AST
SCT
} temporaries

To be broken further

EXD

Assignment of Data Locations

1) The Explicit Date (EXD) is broken up according to COMMON lengths into n + 1 tables. The new addresses are given. If the LIST option was chosen, the print lines are set up and written.

	Name	EXC Proto-Address	Real Address	New RQW Code
	VECTOR	0001	0001	B0
(in Common #1)	MATRIX	0501	0001	B2
"	VALUE	0502	0002	B2
(in Common #2)	WORK3	1537	0001	B3

2) Implicit Data Tables are expanded and addresses are assigned

	Name	Proto Address	Address	New RQW Code
ISD	BOX	0001	0328	B0
	BLAH	0002	0329	
	KOOL	0003	0330	
IDD	HEAT	0001	0332	
	COLD	0002	0334	
	PLACE	0003	0336	
	DENOM	0004	0338	B0

3) Implicit Constant Tables are prepared and addresses are assigned

	Proto-address	Address	RQW
SDC	1	0576	B0
	2	0577	
	3	0578	
DDC	1	0580	
	2	0582	
	3	0584	B0

Assignment of Data Locations (cont'd)

4) Tables are created for Erasables and addresses are assigned

	Proto-address	Address	RQW
ATS	1	0671	B0
	2	0672	
	3	0673	
ATD	1	0674	
	2	0676	
	3	0678	
etc.	4	0680	B0

Phase II B

Assignment of Program String Locations

(1) With all of B0, B2 - B7 assigned, the assignment in B1 (program string) must be carried out. The above tables are used only in the Scan B.

(2) Scan A goes through the JKL sequences assigns location for the sequence according to its length and enters the sequence number and location in the Master Sequence Table. The location counter is originally set to the length of the transfer vector.

(3) Next, the DSN and Program Point Table are processed to fill in actual addresses.

(4) In Scan B, the content words have their proto-addresses replaced referencing the B0-2-7 tables and the DSN and PP tables. The RQW's are changed correspondingly. For ARGn, n, 4 is placed in the address field. As each word is completed, its printout form is set up if the LIST option was given.

(5) At this point, assignment is complete and all sequences are in Format B.

Phase II B

Printouts

The printing is carried out entirely in Phase B.

During Table Reorganization and assignment, Part I is printed.

During Program String Assignment, Part II is printed.

Following this, Part III is printed.

All printing is dependent on the LIST option. The program logic is such that processing will be much faster if the LIST option is not given.

Phase II B

Printouts (example)

COMPILATION OF TITLE

VARIABLE	TYPE	BASE	ADDRESS	VARIABLE	TYPE	BASE	ADDRESS
VECTOR	R	0	0000	RESULT	R	3	0000
MATRIX	R	0	0100	HEAT	D	3	0500
LOGIC	I	0	0201	COLD	D	3	0700
BOOL	L	2	0001	POWER	C	4	0000
VALUE	L	2	0002	JIN	I	4	0001
BOX	R	2	0003	INPUT	I	4	0002

Code	CONSTANT	ADDRESS-BASE O		CONSTANT	ADDRESS-BASE O
1S	1	0310	7S	4.71	0316
2S	2	0311	1D	5.34D	0318
3S	6	0312	2D	6.74289D	0320
4S	43	0313	3D	1.E10D	0322
5S	27	0314			
6S	3.74	0315			

Printout (example) cont'd

EFN	SEQUENCE NUMBER	ADDRESS - BASE 1
6	3A	0112
38	7A1	0230
2	14A	0310
147	17B	0330
18	183	0400

(3 columns)

PROGRAM STRING

```
1A    CLA   1D
      ADD   VECTOR, 1
      STO   MATRIX
1B    CLA   2, 4
      STA   13P
        etc.
```

(3 columns)

TRANSFER VECTOR NAMES

BESSEL *DORIS SINE BETTY** EXIT

*DUMMY

**EXTERNAL

Pointout (example) concluded

Master Sequence Table

Sequence	Address-Base 1
1A	0007
1A1	0014
1B	0023
2A	0064
3A	0073
4A	0084

(4 columns)

Program Summary

Section	Length	Base
UNIQUE DATA	230	0
TRANSFER VECTOR	7	1
PROGRAM STRING	1046	1
COMMON 1	200	2
COMMON 2	500	3
COMMON 3	700	4
COMMON 4	0	5
COMMON 5	0	6
COMMON 6	0	7
(name)		

Phase II B

Printouts (cont'd)

	SYMBOLISM				
Sequence Number		nXm			
Program Point	(sequential)	n P			
Erasables	(sequential)	n ES	nED	n	n
Literals	(sequential)	n S	n D	nSC	nAS
Data	(original name)				

Phase II C

- The function of Phase II C is to output the finished program.

1. Preface Preparation--(see R 4.1)
   (a) The TITLE, COMMON and TRV names are placed
   (b) The TRV and Total Preface Count is placed
   (c) The eight counts are placed
   (d) The external bits (if applicable) are set
2. The Preface is built in the proper place in (internal) load file A.
3. The Text is written out (blocking to be specified later) in this sequence.

Unique Data	B0
Program String	RQWs
Program String	B1 (starting with first word following transfer vecto
COMMON # 1	B2
COMMON # 6	B7

Update Program

Handles writing on LIB Tape

Name	LIB ADD	insert
Name	LIB DEL	delete
Name	both	replace

Special Operations

	LIB CPY	(new copy with system routines only)
	LIB LST	(list)
Name	LIB CHG Name 2	(permanent name change)

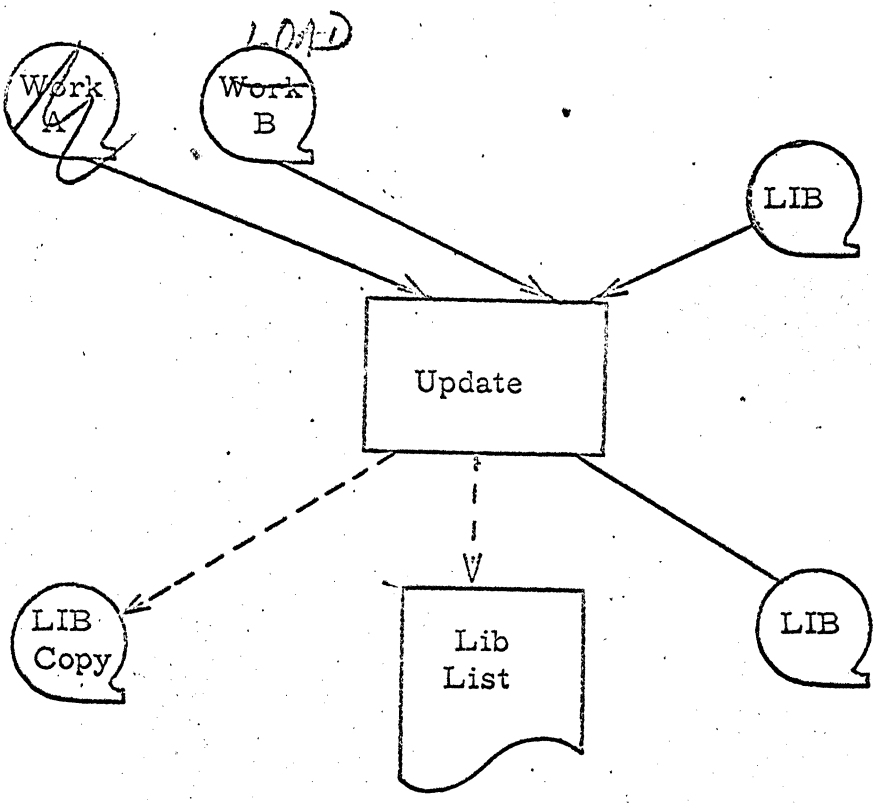
see A4 for control cards

P1 .

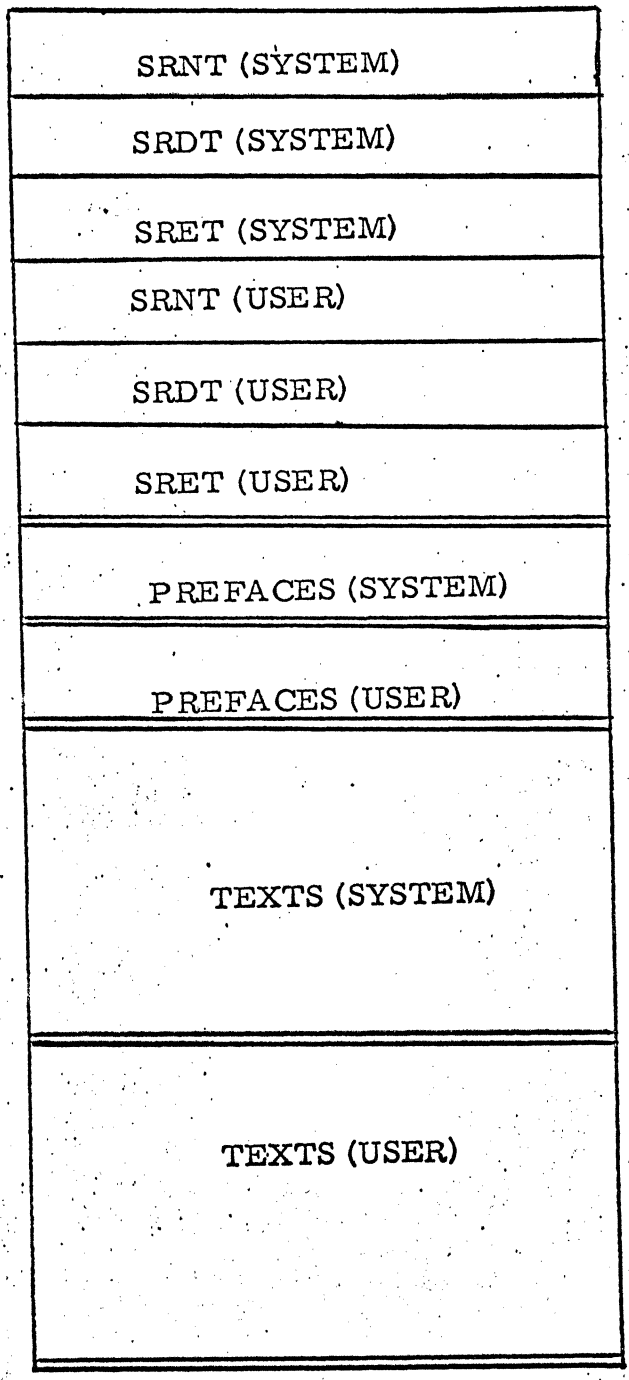
7/10/63

UPDATING

(Load file)



LIBRARY TAPE



Update

1. For LIBADD, LIBDEL and LIBCHG:

   SRNT Table must be altered but kept in order by name.

   SRDT references to SRNT must be altered to reflect changes in SRNT. This implies a change in all references to names beyond the point (or points) of change in SRNT.

   SRET undergoes same consideration as SRDT.

   Each new program preface must be processed for SRDT and SRET entries. Multiple entries are added to SRET (preserving name order) and dropped from the preface. The transfer vector is transformed (using SRNT) into an SRDT entry which is then compared against SRDT to avoid duplication--details of this are still to be resolved. The transfer vector is dropped from the preface--preserving the vector count.

2. For LIBADD

   The appropriate set of prefaces and texts are rewritten with the new prefaces and texts added to the end of their respective files. The new names in SRNT will reference the added record positions.

3. For LIBDEL

   The appropriate set of prefaces and texts are rewritten dropping the ones being deleted.

   SRNT entries referencing relative record position altered by the deletion must be corrected accordingly.

4. LIBCPY

   Strip user table records and preface and text files in creating new tape containing system files only.

   A means of recognizing length and nature (whether user records and files are present on tape) of library for load and update purpose might be necessary.

   Physical blocking of prefaces and texts are indicated.

MAPA

Processor

A "one pass" system since entire symbolic program will reside in memory. This limits the size of MAPA program to 700 input cards.

Output

Must be organized like FORTRAN output. The LIBA may be updated following assembly or the load file will be used by the loader. Relocation codes as in FORTRAN IVA for program strings and data. The output of MAPA is restricted to FORTRAN subprograms (FUNCTION or SUBROUTINE). It cannot serve as independent or "main" program.

General

There will be 9 location counters: transfer vector, data, program string and 6 COMMONs. The user is responsible for using the EVEN operations whenever necessary. The loader will always relocate sections to locations modulo 2.

I. To be included in MAPA:

(a) all machine mnemonics, 1301 and 7909

(b) literals, elements, terms, expressions, Boolean expressions

(c) Psuedo-operations as described by MAP unless specified below.

(d) 7090 double precision routines

(e) Special operations: ZAC, ZSA, ZSD, BRA

(f) Prefix codes

II. Not to be included in MAPA:

(a) Location counter psuedo-operations

USE - BEGIN

(b) Program section psuedo-operations

QUAL - ENDQ - CONTRL - COMMON

(c) Storage allocation and data generation psuedo-operations

LORG - LDIR - RBOOL - LBOOL - LIT - MAX - MIN

(d) File dictionary psuedo-operations

FILE - LABEL

(e) Operation definition psuedo-operations

OPD - OPVFD - OPSYN

(f) Special system macros

SAVE - SAVEN - RETURN

(g) Macro-related psuedo-operations

IFT - IFF - IRP - ORGCRS - NOCRS - MACRO - ENDM

(h) Absolute assembly psuedo-operations

ABS - FUL - PUNCH - UNPNCH - TCD

(i) List control psuedo-operations

LBL - INDEX - PCG - PMC -

(j) Special operations

BFT - BNT - ... - *** - IIB -RIB - SIB

(k) Macro generation mechanism

(l) IOCS operations

(m) Immediate symbols

III. To be included with modified definition

ORG - absolute origin not permissible, symbolic origins must be previously defined

BSS - BES - EQU - SYN - symbolic variable must be previously defined

VFD - symbolic sub-fields must fall in address or decr portion of word

RETURN - expansion of return is:

```
AXT **, 4
TRA 1, 4
```

IV. New psuedo-operations for MAPA

SUB-PROGRAM TYPE PSUEDO-OPERATIONS

These psuedo-operations specify the name and type of sub-program being assembled. The constituents of the SUBR and FUNCT psuedo-operations are:

1. Sub-program name in the location field cannot be blanks
2. SUBR or FUNCT in the operation field
3. The variable field is ignored

This entry must be included at beginning of program deck.

LOCATION COUNTER PSUEDO-OPERATIONS

There are 8 location counters available in MAPA programs. One for program string, one for data areas and 6 common. The normal mode is program. There is no limit to the number of times control is passed between these modes.

The COMMON psuedo-operation

The constituents of the COMMON operation are:

1. A symbol or blanks in the location field
2. COMMON in the operation field
3. The variable field is ignored

The effect of this operation is:

The location counter specified by the label field is activated (initially zero) and will remain in control until another COMMON, PROG or DATA is encountered. The same symbol may be used on COMMONs repeatedly and will continue from previous usage.

The PROG psuedo-operation

The constituents of the PROG psuedo-operation are:

1. The location field is ignored
2. PROG in the operation field
3. The variable field is ignored

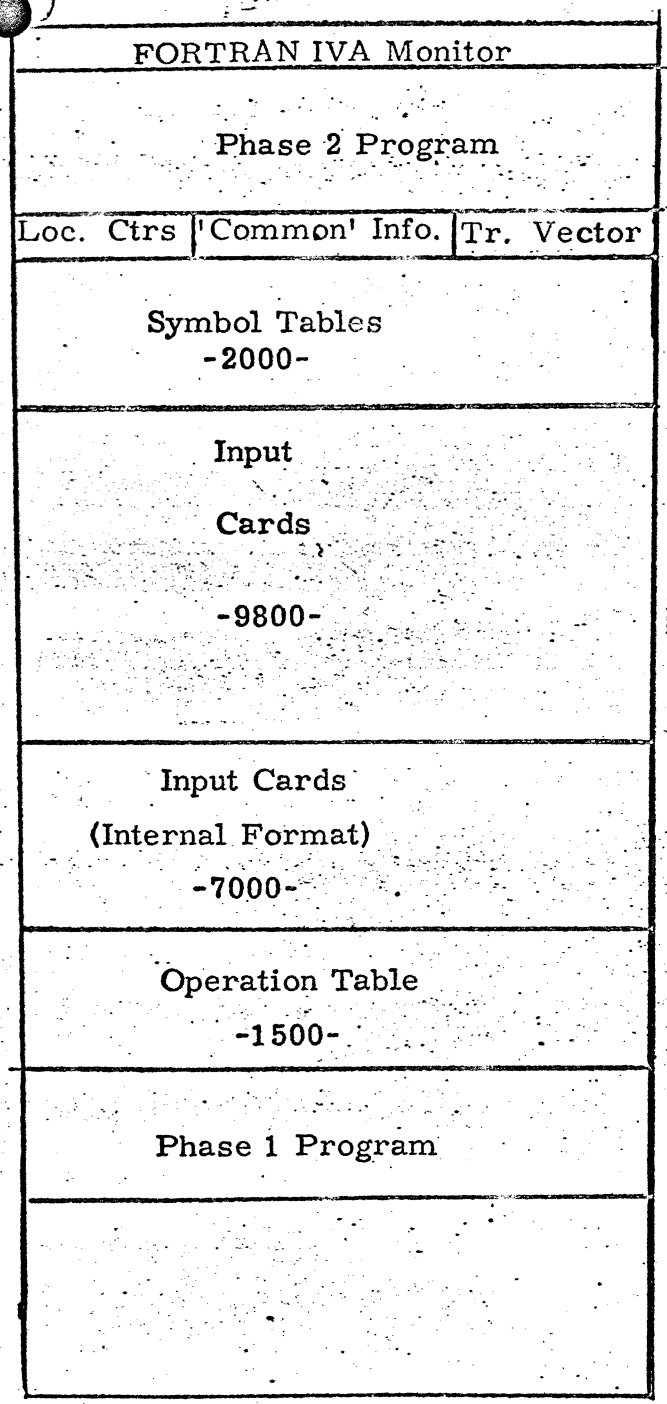
This places the program location counter in control until COMMON or DATA is encountered.

The DATA psuedo-operation

The constituents of the DATA psuedo-operation are:

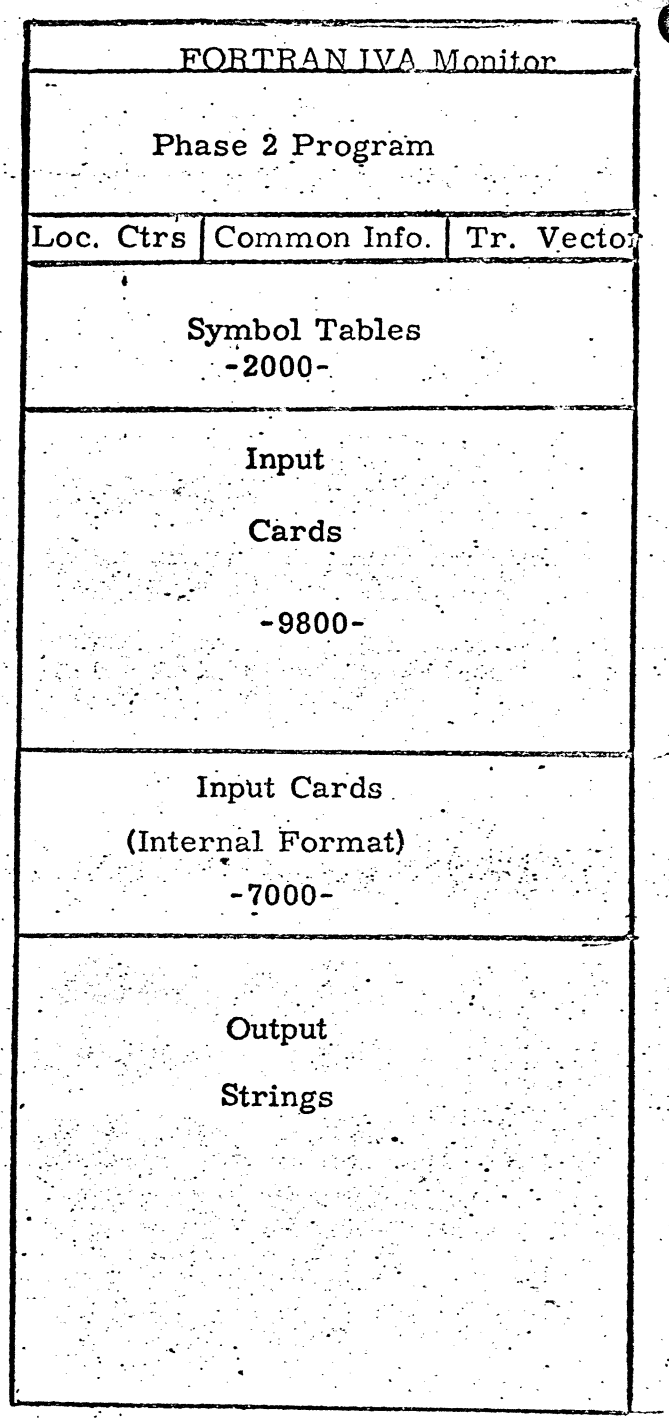
1. The location field is ignored
2. DATA in the operation field
3. The variable field is ignored.

This places the data location counter in control until PROG or data is encountered.



During Phase 1

MAPA Memory Map



During Phase 2

MAPA

Phase 1 will process to binary as much as possible during card read in.

Outline of Phase 1:

1. Stack input card format
2. Set-up Internal format
   a. search operation table obtaining flags and machine OP code.
   b. unblock to internal format per operation type.
   c. if data generating psuedo-op, convert data before placing in internal.
   d. if machine instruction, de-block variable setting indirect and index bits into machine word. Literals to table and literal counter updated.
      Set relocation bits for symbolic variables.
   e. move assigned location and location counter no. to internal format.
3. Construct symbol table
4. Problem area appears to be the expansion of macro-like operations.
5. A space counter for each location counter will be incremented so Phase 2 can allocate memory for output strings.

Phase 2 will complete the assembly.

1. Allocate output strings per space counters for each location counter.
2. Assign literal table location counter to follow data counter.
3. Scan sequentially the internal format:
   a) Search symbol table to complete machine word.
   b) Evaluate expressions in variable.
   c) Set relocation counter bits into RQW where required.
   d) Place in output string per location counter.
   e) Write listing, including diagnostics.
4. After completing scan of internal format:
   Prepare and write preface.
   Write text.
   Transfer control to Monitor.

Elements required in internal format (4 words)

- Input Sequence No.
- Assigned location
- Location counter No.
- Flags from operation table (includes machine op)
- Relocation flag (addr, decr)
- Literal flag (addr, decr)
- No. of words in variable
- Machine word
- Internal form variable } Continuous for data generating psuedo-op
- Diagnostic code

Length of variable + 4 = next internal format

Elements required for Symbol table (2 words)

- Symbol
- Assigned location
- Location counter No. (incl transfer vector)

Elements required for Operation table (2 words)

- Mnemonic
- Flag word same as MAP??

Elements required in STACK area (14 words)

- Input card image
- Input Seq. No. (15 bits)
- List Control Flags (6 bits)

MAPA INTERNAL FORMAT

- WORD # 1 -			- WORD # 2 -
ASSIGNED LOCATION	LOC CTR. NO.	INPUT SEQ. NO.	FLAGS FROM OP TABLE

- WORD # 3 -				- WORD # 4 -
LIT. FLAG	INTERNAL VARIABLE FORM	REL. FLAG	NO. WORDS IN VARIABLE	MACHINE WORD

- WORD # 5 -etc.

INTERNAL VARIABLE OR MACHINE WORD CONTINUED

LOC. CTR. NO.

000 = PROGRAM
001 = COMMON 1
010 = 2
011 = 3
100 = 4
101 = 5
110 = 6
111 = DATA

REL. FLAG

000 = NONE
001 = ADDR
010 = DECR
011 = BOTH
100 = TO TRANSFER VECTOR

LIT. FLAG

00 = NONE
01 = ADDR
10 = DECR
11 = BOTH

NO. WORDS IN VARIABLE + 4 = NEXT INTERNAL FORMAT.

Q 7
7/26/63

Notes

If ETC - Input sequence number in Internal format will be the last ETC card of a series. ETC cards will be evaluated during Phase 1 to create a one entry Internal record.

List control psuedo-ops, *cards, REM cards will not be placed in Internal format.

DUP will be expanded while placing in Internal. DUP itself will not be in Internal.

The first definition of a doubly-defined symbol will be used.

Non-space taking psuedo-ops are in Internal to compute addresses for listing and compile time info only.

PHONEY SUBR xx/xx/xx P

Location Counter			Relocation Counter ADDR/DECR		Location	Operation	Variable	Diag Co
	TRANSFER VECTOR							
T	00000	272525606060			GEE			
						TITLE		
						ENTRY	START	
D						DATA		
D	00000	214570602223			DATA1	BCIb,	ANY BCD INFO.	
D	00012	000000000000	2	P		PZE	K2,, START	
D	00013	076700000000			TBL	ALS	0	
D	00014				BLK	BSS	10	
D	00026	000000000020			DEC4	DEC	16, 25, 33, 62	
D	00032	002000000005	P		INIT	TRA	SYM3	
1					JUNK	COMMON		
1	00000				J1	BSS	9	
1	00011	302160302160			J2	BCI	1, HAbHA	
P						PROG		
P	00000	063400400013	P		START	SXA	RET, 4	
P	00001	050000000033	D			CLA	=6	
P	00002	060100000023	D			STO	BLK+7	
P	00003	007400400000	T			CALL	GEE	
2						COMMON		
2	00000				K2	BSS	3	
						PROG		
P	00005	460000000016	P		SYM3	STQ	K5	
P	00006	450000000032	D			CAL	INIT	
P	00007	060200000002				SLW	2	
P	00010	475400000000				ZAC		
P	00011	477400400000	2			AXC	K2, 4	
P	00012	177777400004	P			TXI	*-6, 4, -1	
					RET	RETURN		
P	00013	077400400000			RET	AXT	**, 4	
P	00014	002000400001				TRA	1, 4	
P	00015				K5	BSS	2	
1					JUNK	COMMON		
1	00012	000032000000		D		PZE	,, INIT	
						END		
	LITERALS							
D	00033	000000000006						

Operation Table Adjective Codes

1X codes are not placed in Internal area

4X-5X codes are space-taking operations

6X codes are non space-taking psuedo ops.

Within psuedo-operation adjective coding sub-operations are assigned.

Adjective Code	Sub-Op	Operation
00		ETC
10		List control operations
	0	UNLIST
	1	LIST
	2	TITLE
	3	DETAIL
	4	EJECT
	5	SPACE
	6	PCC
	7	TTL
11		REM
40		A
41		Prefix
42		I/O Command
43		B
44		C
45		D
46		E
47		Select
50		Disc 4 field channel commands (K)

Adjective Code	Sub-Op	Operation
51		Disc channel commands
52		Boolean variable ?
53		Disc orders
54		Unexpanded instructions (7094 etc.)
55		Storage allocation
	0	BSS
	1	BES
	2	EVEN
56		Data generation
	0	OCT
	1	DEC
	2	BCI
	3	VFD
57		DUP
60		Location Counter
	0	COMMON
	1	PROG
	2	DATA
	3	ORG
61		Symbol definition
	0	EQU
	1	SYN
?	2	BOOL
62		ENTRY
63		NULL
64		END

Codes 55 and 56 - Space Taking Psuedo-operations
BSS, BES, EVEN, OCT, DEC, BCI, VFD

Scan variable to construct machine words.
Convert where necessary.
Insure that VFD with symbolic is in
Address or decr of word.
Increment location counter.
Place symbol and assigned address in tables.
Set relocation flags in Internal.
Set diagnostic flags.
Allow for re-entry if next item is ETC.
Increment space counter.

Code 6X - Non Space-taking Psuedo-operations
COMMON-PROG-DATA-ORG-EQU-SYN-BOOL-ENTRY-NULL

ENTRY - extract symbol from variable field and place in ENTRY list.
Check for duplicates. Diagnostic
Set flags in Internal format.

NULL - Compute symbol address and place in STBL.
Diagnostic if doubly defined and retain initial entry in STBL 1.
Move current location and location counter number to STBL 1.

EQU, SYN - Extract variable to internal format.
If symbolic variable, search STBL obtaining assigned locations.
Evaluate variable expression.
Diagnostic if all variable symbols were not under the same Location counter.
Compute symbol address and place in STBL.
Diagnostic if doubly defined and retain initial entry in STBL 1.
Move evaluated expression and location counter number to STBL 1.
Set Internal format machine word.
Diagnostic if Symbol not previously defined.

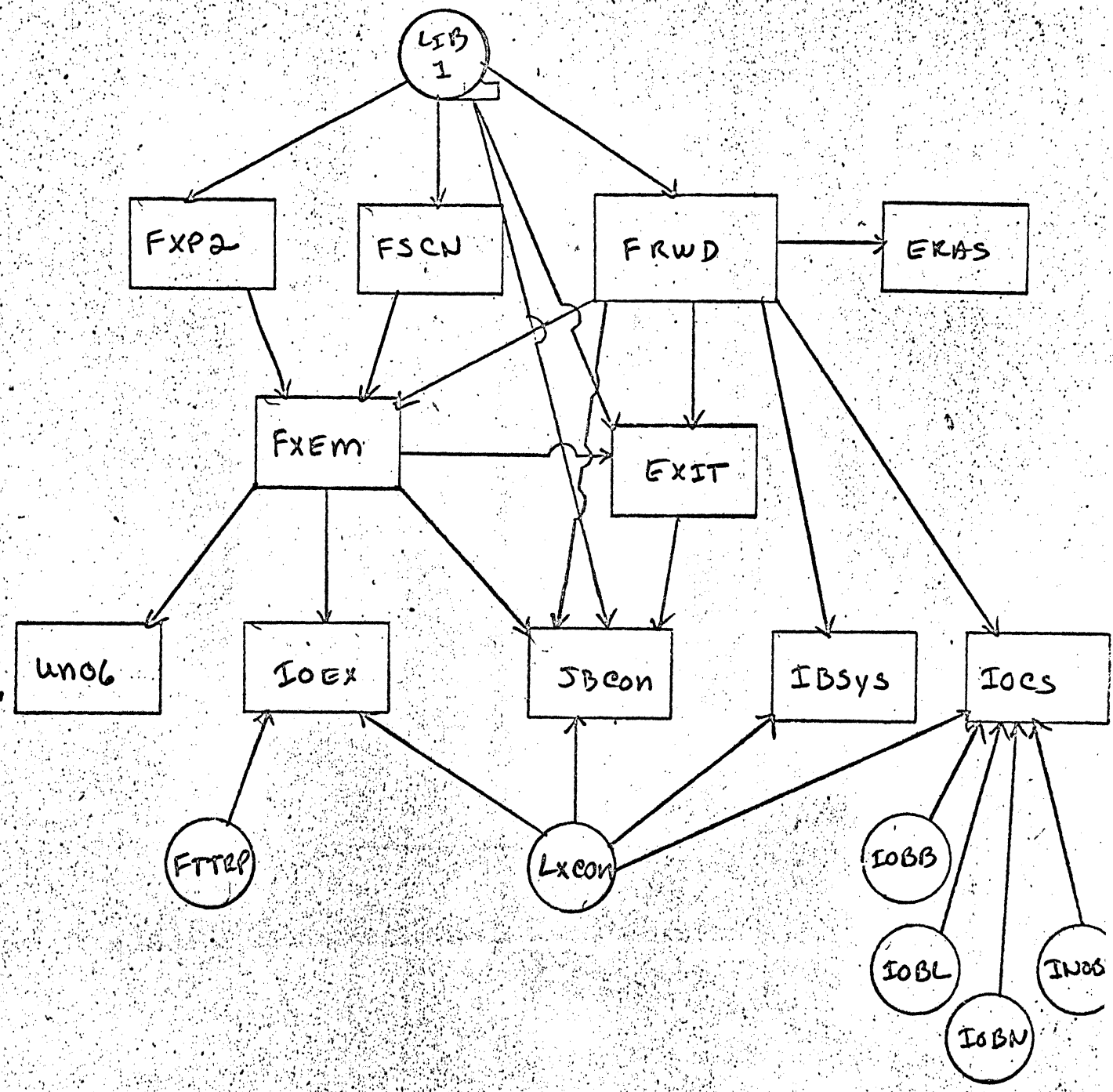
ORG - Extract variable to Internal.
Evaluate variable (see EQU).
If current location counter contents are higher than last used for location counter now in control, replace last used with current location counter.
Initialize current location counter and location counter number per variable.
Set Internal format machine word.

BOOL - ???

COMMON - Move current location counter back to permanent area per current location counter number.
Search COMMON names with input symbol.
If previously defined COMMON, move location counter and number to current area.
Diagnostic if more than 6 COMMONS.
If not previously defined move symbol from input to list of COMMON names, and move location counter number to current, zero current location counter.
Set Internal format machine word.

PROG or DATA - Move current location counter back to permanent area per current location number.
Move location counter from permanent (PROG or DATA) and assign current location counter number.
Set Internal format machine word.

Schematic of output from LOGIC option



LOADER

(no overlay)

Phase A

Reads Control Cards

Reads Load File A (prefaces)

Builds tables (commons, subprogram names)

Reads dependency table from LIB

Builds tables further

Applies USE, NAME, OMIT

Reads in Prefaces from LIB file

Allocates COMMONs

Allocates remainder of program

Prints Allocation Chart

Phase B

Reads texts into core

Links transfer vectors

Relocates and

(a) Punches absolute self-loading deck

(b) Fills core placing STR in unused locations

R 1
7/10/63

Linkage to Subprograms

Present IBJOB System:

A NAMEd subprogram may have symbolic locations that serve as linkage points.

Proposed System

A NAMEd FORTRAN may have linkage only to its prologue.

A NAMEd MAP subprogram may have linkage to specified ENTRY points.

R 2
7/10/63

Library files

File 1: Dependency-Sequence Table
Prefaces

File 2: Texts

(The system subprograms always precede the user subprograms)

Load files

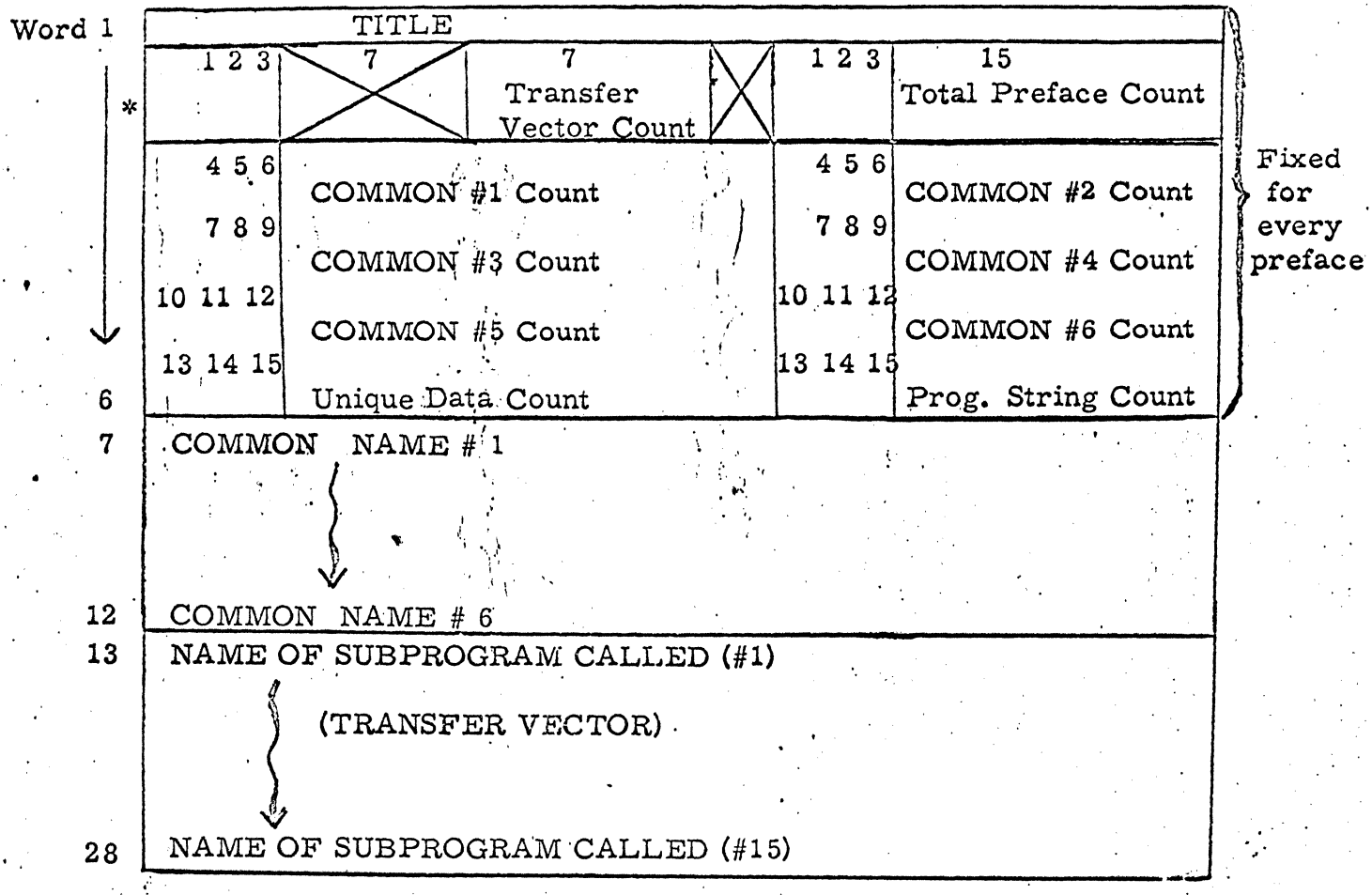
Work File A

Prefaces (no sequence or dependency table)

Work File B

Text

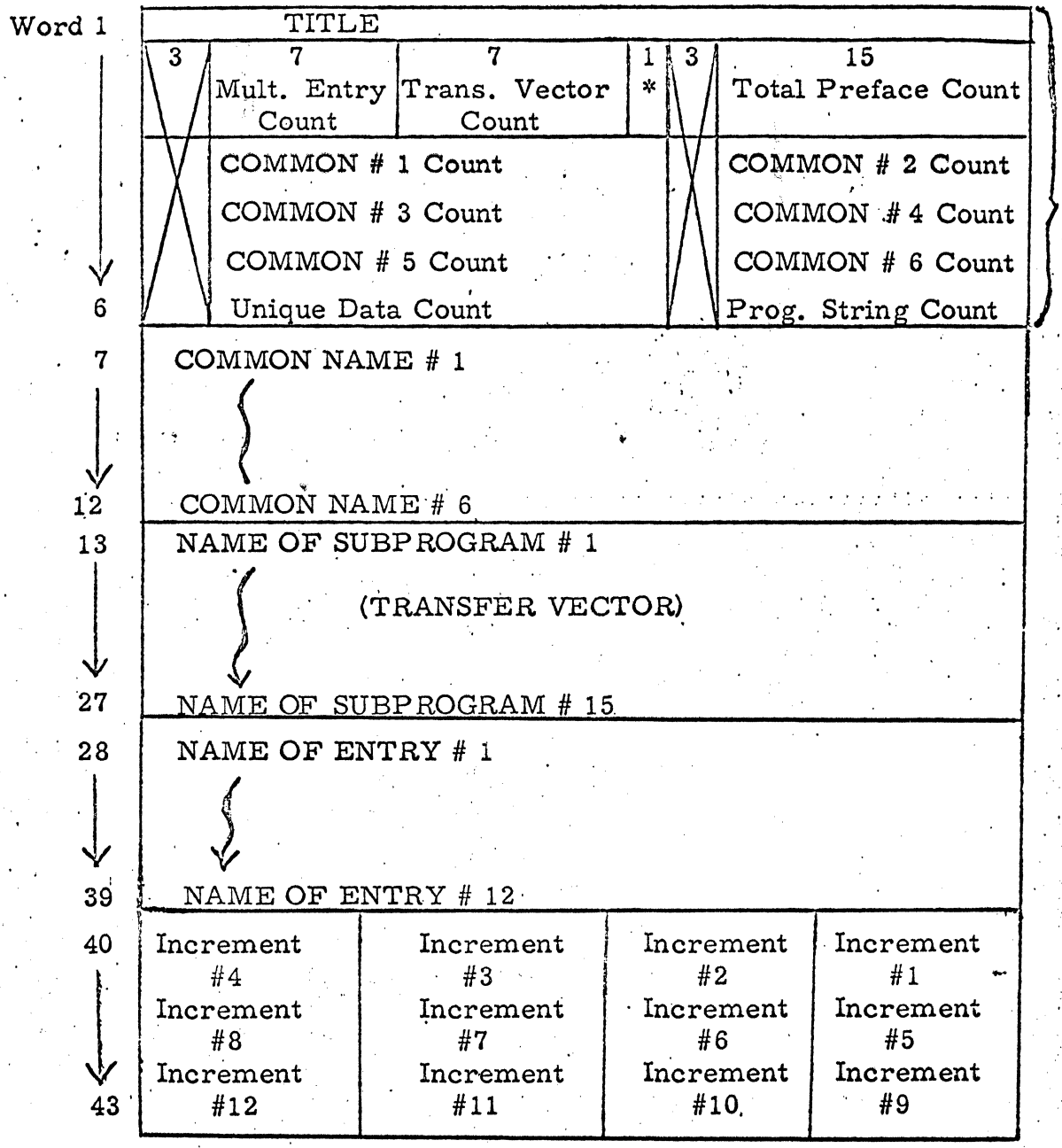
FORTRAN PREFACE



* Each bit of the 2 sets of 15 bits are associated with one of the named subprograms in the transfer vector. The left set are used as external indicators and the right set as dummy name indicators.

** Preface is completely variable in length from word 7 on.

MAP PREFACE

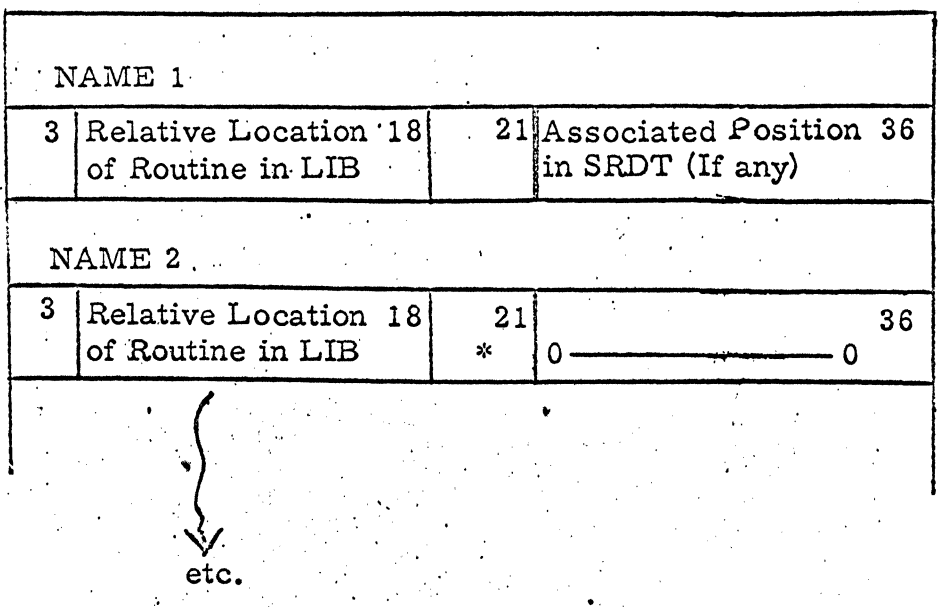


* This bit is set for a MAP text

** Preface is completely variable in length from word 7 on.

Subroutine Name Table (SRNT)

in first file of Library

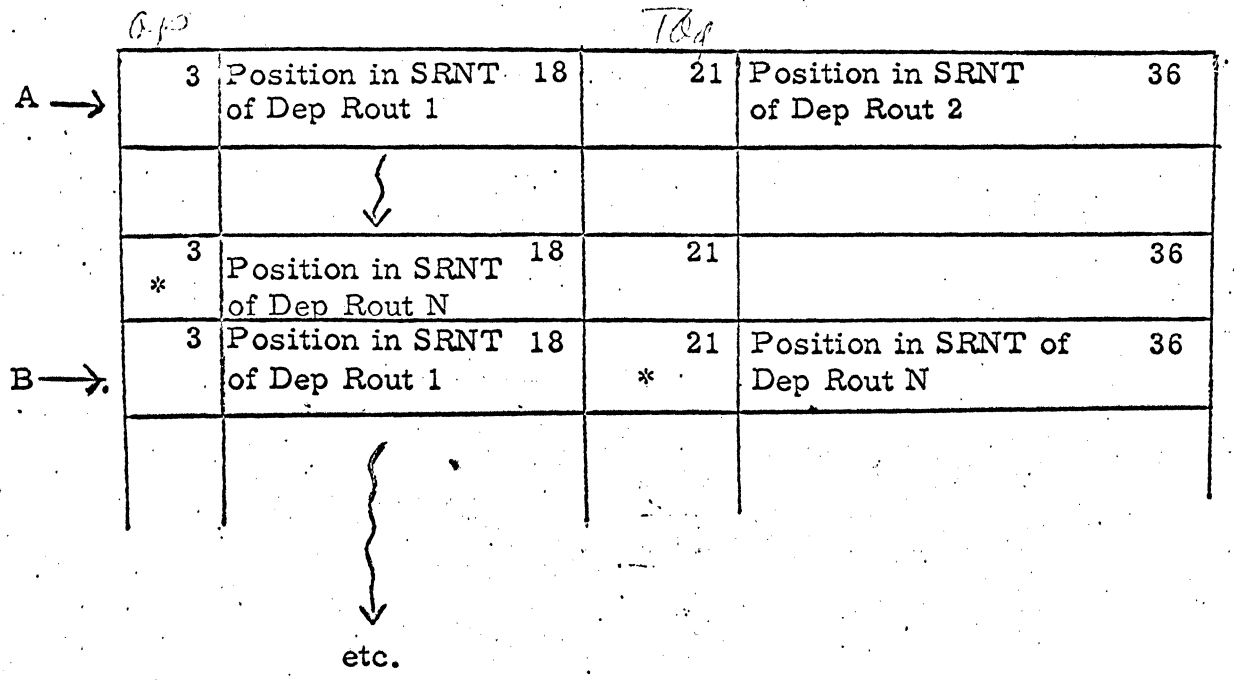


* Bits 19-21 will indicate whether there is any SRDT position # in Bits 22-36.

Table will be in alphabetical order by name, a 2-word entry per routine in the library.

Subroutine Dependency Table (SRDT)

in 1st file at Library



(*) In either OP or Tag field indicates following 15 bits contain SRNT position for last routine dependent upon routine originally directed to this table from SRNT.

A and B represent SRDT positions appearing in SRNT entries, each of which are the initial location of a string of dependent routines.

Elimination of duplicates (where feasible) must be considered.
Does this eliminate need for Transfer Vectors in library prefaces?

Subroutine Entry Table (SRET)

in 1st file of library

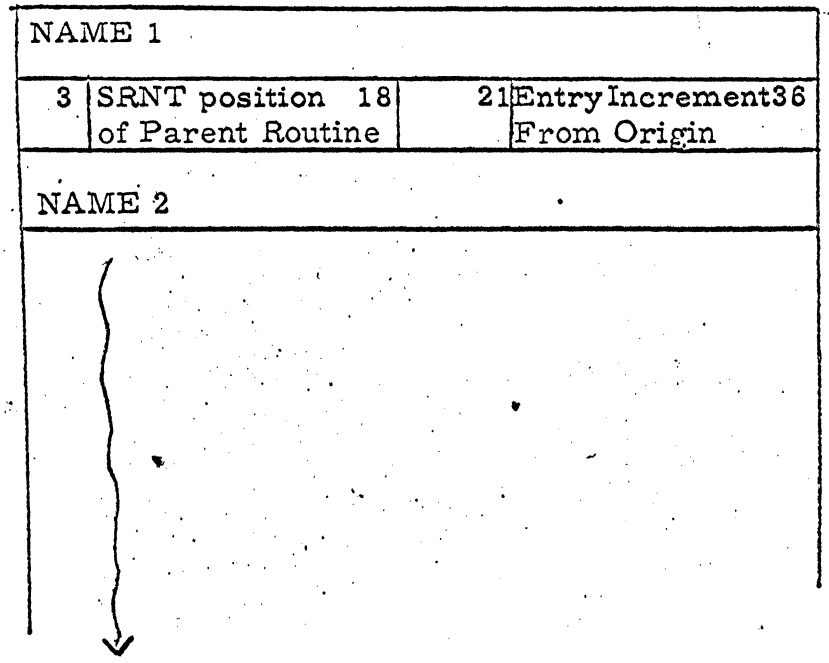
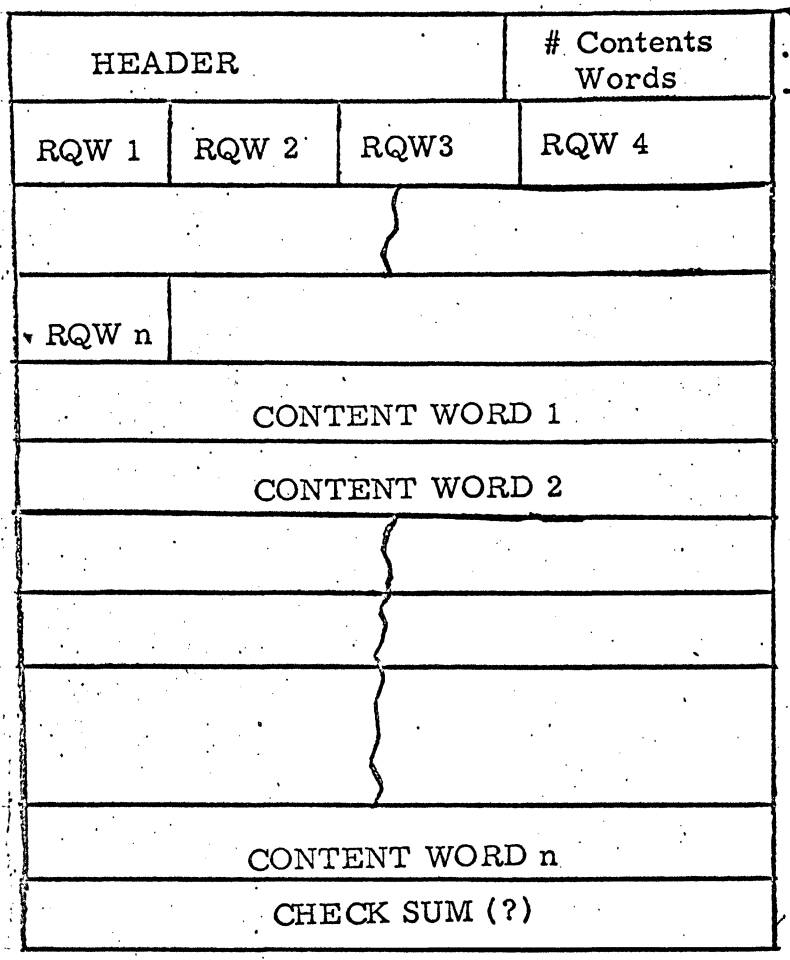


Table will be sorted in alphabetical order by name, a 2-word entry for every "multiple" entry associated with any routine in the library.

Entry names in library prefaces are not necessary.

TEXT FORMAT



Header--indicates whether text is Common Data Block, Unique Data, or program string.

All texts resulting from any one compilation or Assembly to form one physical block on tape--tentative.

The Object System as Loaded

(no overlay)

7094-II considered

(Example)

TR VECTOR	Unique 1
Prog Strg 1	Unique 2
Prog Strg 2	Unique 3
Prog Strg 3	Unique 4
Prog Strg 4	
	Blank Common
——	Common #1
Common #3	
	Common #2

R 5
7/10/63

Chart Printout by Loader

(no overlay)

MAIN PROGRAM: NAME

	OCTAL	DECIMAL
Entry Point	1037	1241

LOCATION OF ROUTINES

NAME	0027	____
SINCOS	1031	____
EXP	1373	____
SUBR1	2001	____
SUBR3	2037	____
FXEM	2077	____

LOCATION OF COMMON REGIONS

/BLANK/	3014	____
/LABEL1/	7120	____
/LABLE2/	10164	____
/LABEL3/	12146	____

UNUSED CORE

FROM	13702	6834
TO	77777	32768

LOADER Error Messages

1. Unequal SIZE of labelled COMMONS
2. Insufficient memory
3. Undefined subprogram requested
4. Ambiguous or Erroneous control card
5 Overlay with EXT function
6 " " ABS

Allocating and Loading Text

COMMON sections are allocated modulo 2. There is no relocation within these sections.

Program string carries relocation information. It is also allocated modulo 2.

Unique data is allocated modulo 2.

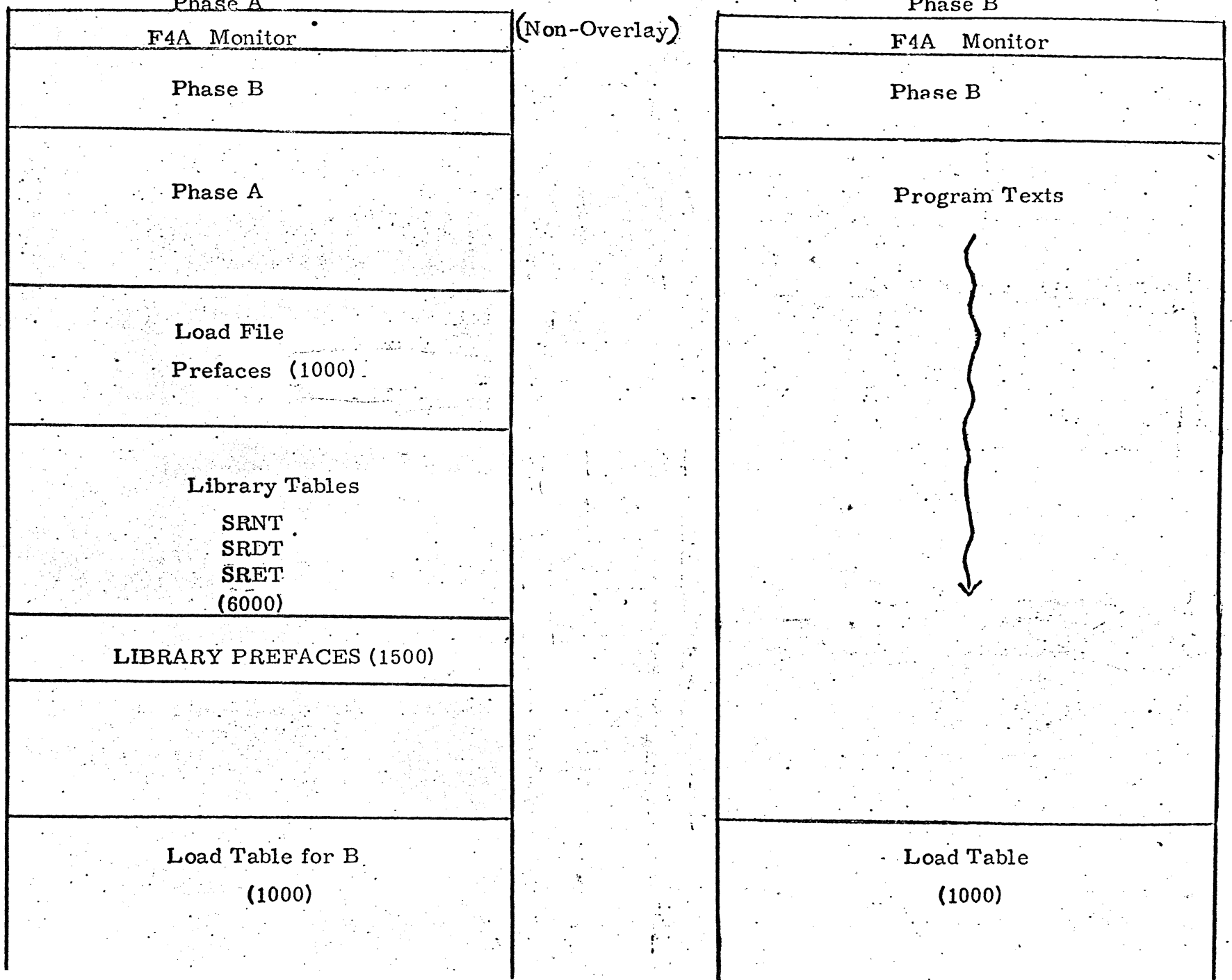
If from a MAP, there is relocation.
If from a FORTRAN, it is loaded without relocation.

R 7
7/24/63

	OVERLAY A	OVERLAY
Control Cards	1: Overlay Card	2: $Origin $Include (Reassigns control sections)
Overlay Areas	1 only - defined by largest chain to overlay.	Multiple - defined by origin cards. (Chains within chains.)
Common	All Blank common resident, labelled common referenced by two or more chains - resident. Unique labelled common - overlay.	No explicit discussion other than its implication as a control section on page 4: any common block can appear only once.
Chain Definition	Loader - determined tree of routines or control sections called by name appearing on overlay card.	All Decks physically following $ Origin card before next $ Origin in input file.
User Intervention	Fully automatic - Load and Go.	Unclear as to where and when control cards are inserted - assumption is made that there is no load-go. User must arrange output decks from FTC or MAP appropriately.
Tape Arrangement at Object Time	1→ n Tapes used for stacking overlay segments. Files prepared by loader prior to execution, user-specified distribution of tapes should be considered.	Up to 7 sysut tapes available - each chain may specify or else get same Roughly same as Overlay A.
Routines Referenced by Other Chains	Proposed: ① Load them in resident core. ② Dupe them in each calling chain on tape. Note: if ① were adopted, LDRA would do so selectively.	At Object time whole chain hierarchy is loaded by a call to any part of the chain. This is in conjunction with unique deck-appearance rule. R 8 Note: All library routines automatically go in resident core. 7/30/63

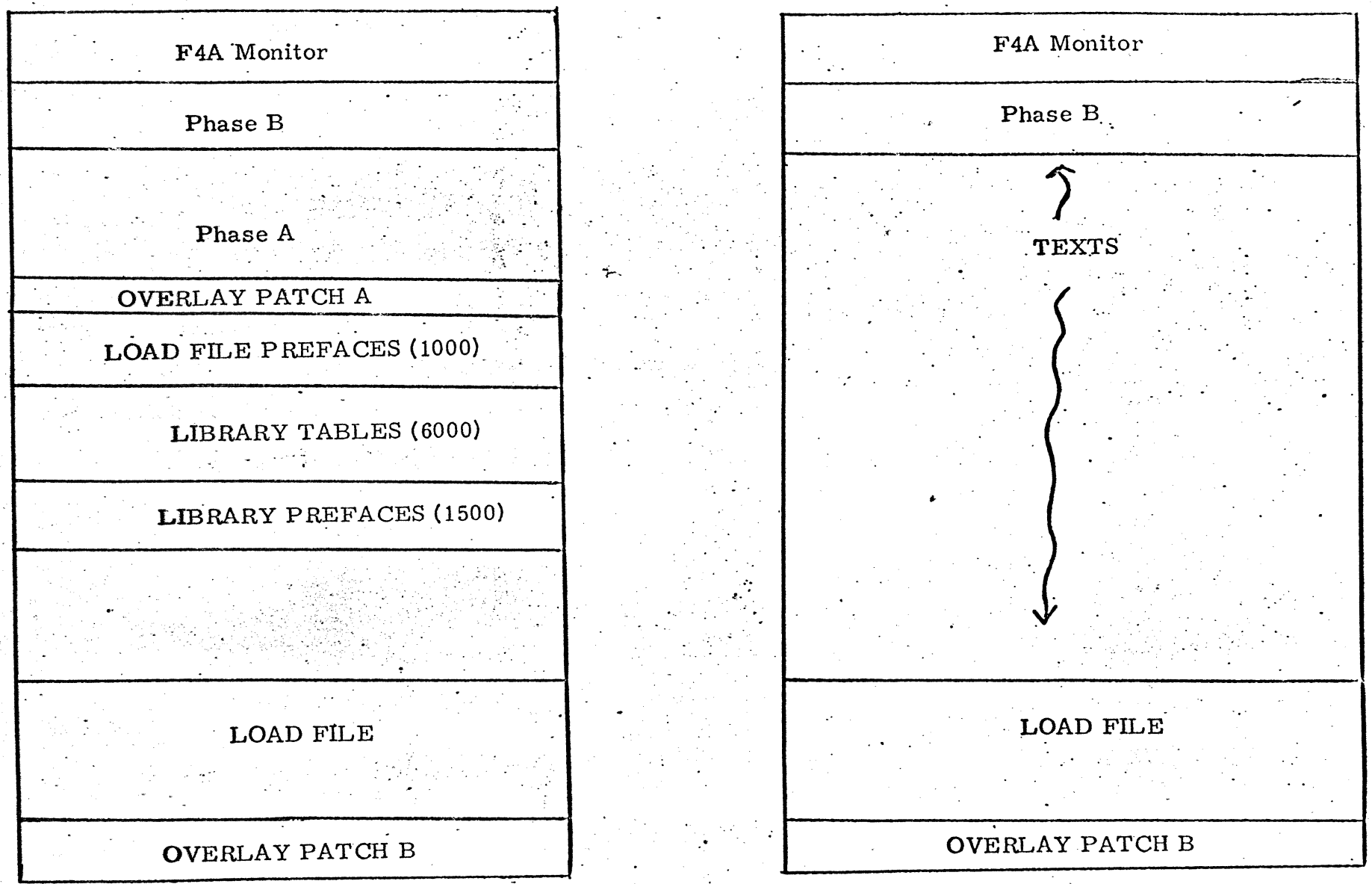
OVERLAY A	OVERLAY (cont.)
External	Same rule should apply. Implication is that External functions can be part but not the beginning of a chain.	Rule: No dummy function name (Parameter of subroutine) can initiate overlay.
Transfer Vector	May be more compact.	Page 5. How do they get the Link # to .OVRLY?
Overlay Candidates	Pending further Analysis - There's no reason why function subprograms may not also be Overlays.	Only subroutines referenced by Calls may initiate overlay. (what about logical if calls?)

LOADER LAYOUT



LOADER MEMORY MAP

(OVERLAY)



Load Table Built By Phase A for Phase B

Single Entry ranges from 1 - 12 words

$	Location of Routine on Load File or LIB.	**	Absolute Core Location For Loading
.	Absolute Address For Unique Data		Absolute Address For TB 001
	↓		
*	Absolute Address For TV 00N		Absolute Address of Common 1
			↓
	Absolute Address of Common N-1	*	Absolute Address of Common N

In OP

$ indicates whether text comes from load file or library. All entries will be in order by tape record location.

In Tag

** indicates whether object transfer vector and/or common locations follow or not

* in OP or Tag indicates following 15 bits contains last transfer vector contents, or last common block location

. in OP indicates Unique Data Location follows

Op and Tag Symbolic notations and equivalent bit settings

Op/Tag	*	= 111
Tag	**	= 000 (no transfer vector, no Common)
		001 (just Transfer Vector)
		010 (just Commons)
		011 (both Transfer Vector and Commons)
Op	$	= 000 (Load File)
		100 (LIBE)
	.	= 001