XEROX

Technical Information

WRITER'S MASTER COPY DO NOT REMOVE

Xerox 900/9300 Meta-Symbol

Technical Manual

90 08 27C

July 1971

Xerox Data Systems

701 South Aviation Boulevard
El Segundo, California 90245
213 679-4511

XEROX

Xerox 900/9300 Meta-Symbol

Technical Manual

90 08 27C July 1971

Price: $18.50

©1965, 1966, 1967, 1971, Xerox Data Systems, Inc.

Printed in U.S.A.

REVISION

This publication is a revision of the Xerox META-SYMBOL/Technical Manual, Publication Number 90 08 27B (dated October, 1967). Sections 6 and 7 have been added, as well as Appendixes B and C. All changes in the text from that of the previous manual are indicated by a vertical line in the margin of the page.

NOTICE

The specifications of the software system described in this publication are subject to change without notice. The availability or performance of some features may depend on a specific configuration of equipment such as additional tape units or larger memory. Customers should consult their XDS sales representative for details.

CONTENTS

Section		Page
	INTRODUCTION	v
1	META-SYMBOL ASSEMBLY SYSTEM OVERALL DATA FLOW (900 Series Only)	1-1
	META-SYMBOL ASSEMBLY SYSTEM OVERALL DATA FLOW (9300 Only)	1-1
2	DETAILED DESCRIPTION OF THE META-SYMBOL ASSEMBLY SYSTEM (900 Series Only)	2-1
	DETAILED DESCRIPTION OF THE META-SYMBOL ASSEMBLY SYSTEM (9300 Only)	2-1
3	INDIVIDUAL DESCRIPTION AND FLOWCHARTS	
	a. Basic Tape Loader (900 Series Only)	3-1
	MSCONTRL (900 Series Only)	3-3
	MSCONTRL (9300 Only)	3-3
	b. ENCODER (900 Series Only)	3-39
	ENCODER (9300 Only)	3-39
	S4B	3-94
	MON1 (900 Series Only)	3-114
	c. PREA (900 Series Only)	3-117
	PREA (9300 Only)	3-117
	SRNK	3-155
	d. PAS1 (900 Series Only)	3-169
	PAS1 (9300 Only)	3-169
	Programmed Operators (900 Series Only)	3-240
	e. PAS2 (900 Series Only)	3-299
	PAS2 (9300 Only)	3-299
	FNSH (900 Series Only)	3-341
	FNSH (9300 Only)	3-341

Section		Page
4	ITEM AND TABLE FORMATS USED IN META-SYMBOL (900 Series Only . .	4-1
	ITEM AND TABLE FORMATS USED IN META-SYMBOL (9300 Only)	4-1
5	OPERATIONAL INFORMATION (900 Series Only)	5-1
6	META-SYMBOL CONCORDANCE OPTION (900 and 9300 Series)	6-1
7	ITEM AND TABLE FORMATS USED BY THE CONCORDANCE PROGRAM (900 and 9300 Series)	7-1
	APPENDIX A. LISTING OF SUBROUTINES.	A-1
	APPENDIX B. HOW TO MAKE A 900 META-SYMBOL SYSTEM	B-1
	Figure B-1. META-SYMBOL Overlay Structures	B-16
	APPENDIX C. 900 SERIES META-SYMBOL ENCODED FORMAT	C-1

INTRODUCTION

The META-SYMBOL Technical Manual will be an aid in maintaining the 900 Series and 9300 programming systems as well as being a reference manual suitable for an operations guide in using the system. No definition or explanation of the source language is provided; it is assumed that the reader is familiar with the language as well as the XDS computers and their peripheral equipment. The language is described in Xerox Data Systems publication XDS 90 05 06B, SYMBOL and META-SYMBOL Reference Manual

This manual contains four major parts. The first part (Section 1) gives an overall picture of the assembly system and the monitor META-SYMBOL relationships. The second part (Sections 2 and 3) is a detailed explanation of the assembly system, explaining the various programs and routines used. The third part (Section 4) describes item and table formats. The combination of parts 2 and 3 is a basic maintenance manual for the assembly system. The fourth part (Sections 5 and 6) is a self-standing operations reference manual for machine room use. The Appendixes – whose titles are self-explanatory – give further detailed information.

The 900 Series and 9300 META-SYMBOL have some differences. The most significant difference is caused by the fundamental differences in the monitors META-SYMBOL operates under. The 900 Series monitor (MONARCH) is not resident, while the 9300 monitor (MONITOR) is resident. The 900 Series META-SYMBOL does its own I/O which is initialized using the UAT (unit assignment table) set up by MONARCH. The 9300 META-SYMBOL does its I/O through MONITOR.

When information is applicable to only the 900 Series or to only the 9300, the fact is noted at the top of the page.

In the sections that follow, references to "the system tape" should be construed as references to "the system file" when META-SYMBOL is operating within the RAD MONARCH environment. In the RAD MONARCH system, S is always RAD-resident; however, X1 and X2 may be optionally assigned to the RAD.

SECTION 1

META-SYMBOL ASSEMBLY SYSTEM OVERALL DATA FLOW

Figure 1 illustrates the overall data and program flow for the META-SYMBOL assembly system. Each of the program boxes represents a separate core overlay of the system. Each overlay in turn represents a group of records (one or more labeled segments) on the MONARCH system tape. The first segment of the assembly system (META) is loaded by MONARCH. All other segments (except MONARCH) are loaded by an absolute tape load program loaded with the META and left in low memory. MONARCH is reloaded at the end of the assembly process by means of the MONARCH bootstrap routine, left residing in high memory.

Although the diagram shows a card-oriented system, the META-SYMBOL assembly system has a complete range of self-initializing I/O capability, dependent on the setting of UAT and MSFNC by MONARCH, which allows the user to relate I/O functions and devices at assembly time.

The processing of control records within the assembly system is performed by MONARCH. From the ASSIGN and METASYM control cards MONARCH sets up two communication regions for the assembler; the first of these is the Unit Assignment Table (UAT) which indicates the unit and channel assignments for the various I/O devices and options which the assembly system may use. The second communication region is a cell, MSFNC, in which MONARCH indicates the I/O functions to be performed for a given assembly as determined from the METASYM control card. After setting MSFNC, MONARCH loads the first overlay of the system: META.

The ENCODER portion of META reads and processes the input program which may be symbolic, or encoded, or encoded with symbolic corrections. The ENCODER outputs an intermediate program tape (X1) and, if requested, a new encoded file. If no additional processing is requested, the ENCODER returns control to MONARCH. If additional processing is required, the ENCODER calls a basic tape loader routine to load the PREA (preassembler) routine.

When loaded, PREA, has at its disposal in core the dictionary for the encoded program and the balanced tree search table for searching the dictionary as constructed by the ENCODER. PREA processes the selected standard system procedures from the system tape, defining only those procedures which are used within the user's program. The preassembler also defines the directives for the assembler and converts the dictionary from the ENCODER format to the format used by the

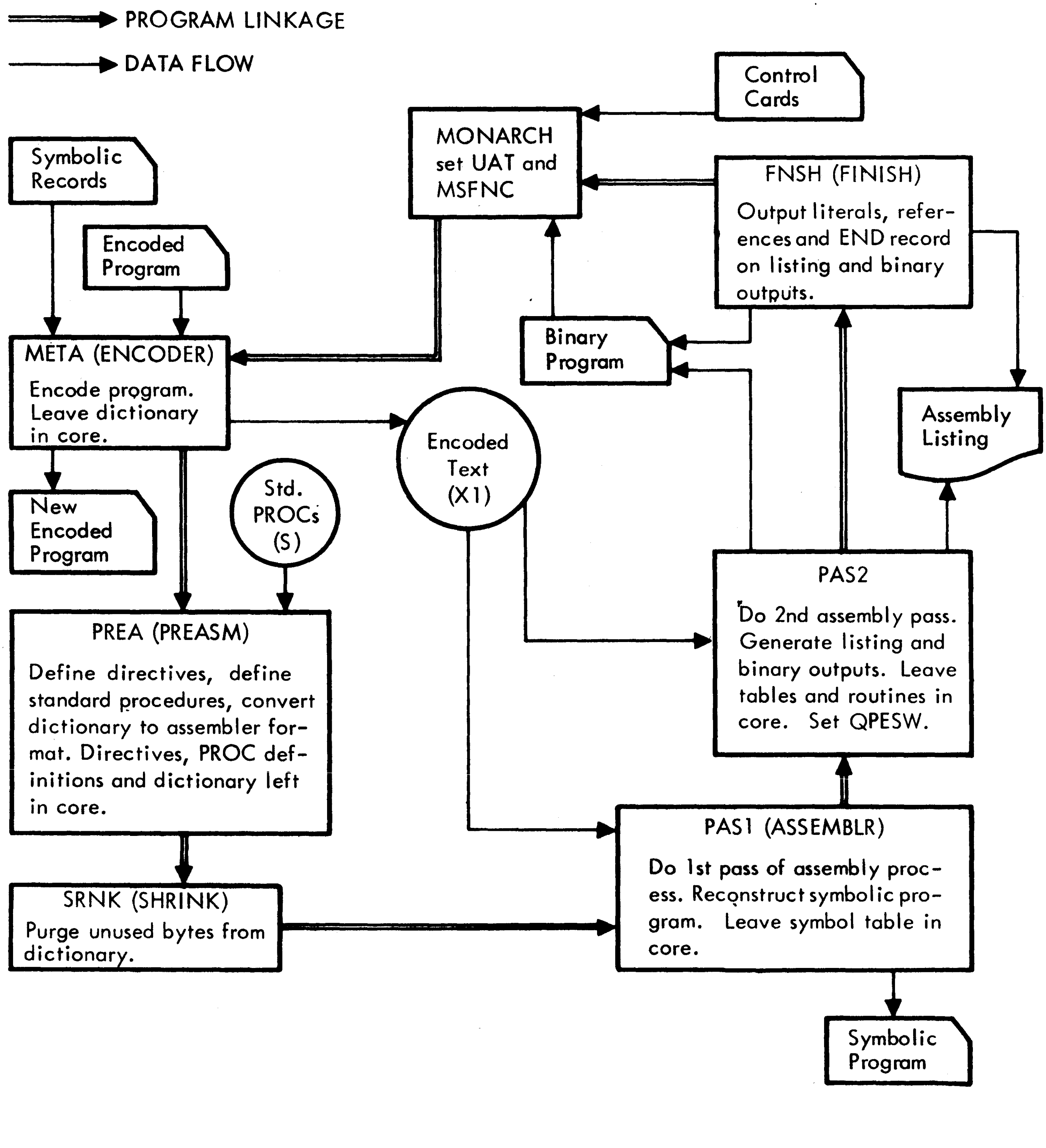


Figure 1-1. MONARCH-META-SYMBOL Data Flow

assembler. (See Section 4, Item Formats.) The preassembler then calls the tape loader to load SRNK (SHRINK).

SHRINK purges the dictionary and byte table constructed by the preassembler to remove unused bytes. The sole purpose of SHRINK is to minimize the table size and thus maximize available working storage.

SHRINK calls the tape loader to load PAS1 (assembler pass 1). The input to the assembler is the encoded text tape (X1) generated by the ENCODER. During pass 1, the assembler defines the labels used within the program and determines program size in order to set the starting location for literals. If a symbolic regeneration is requested, the symbolic program is output during pass 1 of the assembler. At the conclusion of pass 1, the external symbol (entry points) definitions are output in type 1 records, and the external programmed operator definitions are output on type 2 records, provided binary output has been requested. If either listing or binary output has been requested, assembler pass 1 calls the loader to load PAS2 (assembler pass 2). If no additional output has been requested control returns to MONARCH.

PAS2 is the data-generating pass of the assembly system. Using X1 as input, PAS2 generates the binary output records and assembly listing. If errors are detected, cell QPESW is set for MONARCH indicating that errors have been encountered. This cell is important in "assemble-and-go" operation as a measure of the quality of the binary output. During this second assembly pass, literals are defined and references to externally defined symbols are flagged and linked. At the conclusion of the second assembly pass, PAS2 calls the tape loader to load FNSH (FINISH).

FINISH punches and lists the literals, punches and lists the external symbol references, and punches the transfer or end card for the binary program file. Upon completion, MONARCH is reloaded by calling the bootstrap routine which has been retained in high memory.

SUMMARY OF MONARCH-META-SYMBOL COMMUNICATIONS

MONARCH processes the ASSIGN control card and passes on to the assembler the unit and channel assignment information in the UAT.

MONARCH processes the METASYM control card and passes on to the assembler the functions to be performed in the form of entries in MSFNC.

MONARCH loads the first overlay of the assembler.

MONARCH determines maximum machine size for the run and locates the bootstrap routine (QBOOT) and UAT accordingly. The assembler uses the contents of cell 1, which MONARCH sets to BRU QBOOT, to determine the location of QBOOT and hence the available storage.

MONARCH does all tape positioning in the system. The only positioning performed by the assembler is on scratch tapes X1 and X2 (in the event it is necessary to copy symbolic corrections) and on the system tape when specific routines are being loaded. Thus, all inputs and/or outputs may be stacked.

The assembler sets QPESW, program error switch, for MONARCH as a quality indicator for "assemble-and-go" jobs.

The assembler returns control to MONARCH at the conclusion of all runs by branching to the MONARCH bootstrap routine QBOOT.

Following are the interpretations given the UAT settings by the assembler.

MONARCH Symbol	Assemble Interpretation
QSYSI	Scratch tape for corrections (X2)
QMSG	Not used
QSYS	System tape (S)
QSYMI	Symbolic input device (SI)
QSYST	Intermediate output tape (X1)
QBINO	Binary output device (BO)
QSYMO	Listing output device (LO)
QBINI	Encoded input device (EI)
QSYSP	Encoded output device (EO)
QSYSW	Symbolic output device (SO)
QPESW	Error switch

Following is the format of MSFNC.

	C	P	SI	TO	BO	LO	EI	EO	SO
Bits	0	1 2	3 5	6 8	9 11	12 14	15 17	18 20	21 23

The P field indicates by the binary numbers 00, 01, 10, and 11 which of four procedure libraries are to be used. The other fields indicate the presence or absence of a function by a 1 or 0:

C	- compatability mode	LO	- listing output
SI	- symbolic input	EI	- encoded input
TO	- intermediate output	EO	- encoded output
BO	- binary output	SO	- symbolic output

GENERAL RESTRICTIONS AND LIMITATIONS

The META-SYMBOL assembly system requires a minimum configuration of at least 8192 words of core memory and two magnetic tape units or one MAGPAK pair.

If both encoded and symbolic inputs are present for an assembly and if both these inputs are on the same peripheral unit, an additional tape unit is needed. One MAGPAK pair of tapes meets this requirement.

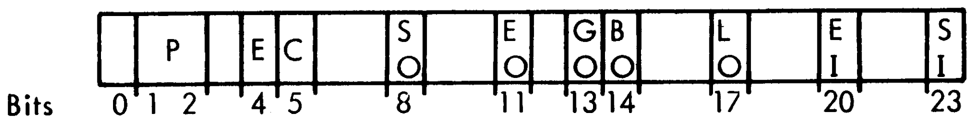
The system does not have the capability to process FORTRAN compatibility directives nor to process local NAME directives.

SECTION 1

META-SYMBOL ASSEMBLY SYSTEM OVERALL DATA FLOW

Figure 1 illustrates the overall data and program flow for the META-SYMBOL assembly system. Each of the program boxes, with the exception of MONITOR, represents a separate core overlay. Each overlay is a labeled absolute binary record on the MONITOR system tape.

MONITOR reads control cards. Assign cards cause MONITOR to set up the I/O linkage. The META card causes MONITOR to read the first overlay (META) into core and branch to it. The functions requested on the META card are passed on to META-SYMBOL by a coded word in index register 2. This word is saved in a cell called OPTION:



P - selects one of four standard procedure sets

E - encoded input = symbolic input

C - compatability

SO - symbolic output

EO - encoded output

GO - GO output

BO - binary output

LO - listing output

EI - encoded input

SI - symbolic input

META has two sections: MSCONTRL and ENCODER. MSCONTRL is never overlayed. It contains the I/O file control routines and a tapeloading subprogram to read each of the succeeding overlays into core.

The ENCODER portion of META reads and processes the input program which may be symbolic, or encoded, or encoded with symbolic corrections. The ENCODER outputs an intermediate program tape (X1) and, if requested, a new encoded file. If no additional processing is requested, the ENCODER returns control to MONITOR. If additional processing is required, the ENCODER calls a basic tape loader routine to load the PREA (preassembler) routine.

When loaded, PREA, has at its disposal in core the dictionary for the encoded program and the balanced tree search table for searching the dictionary as constructed by the ENCODER. PREA

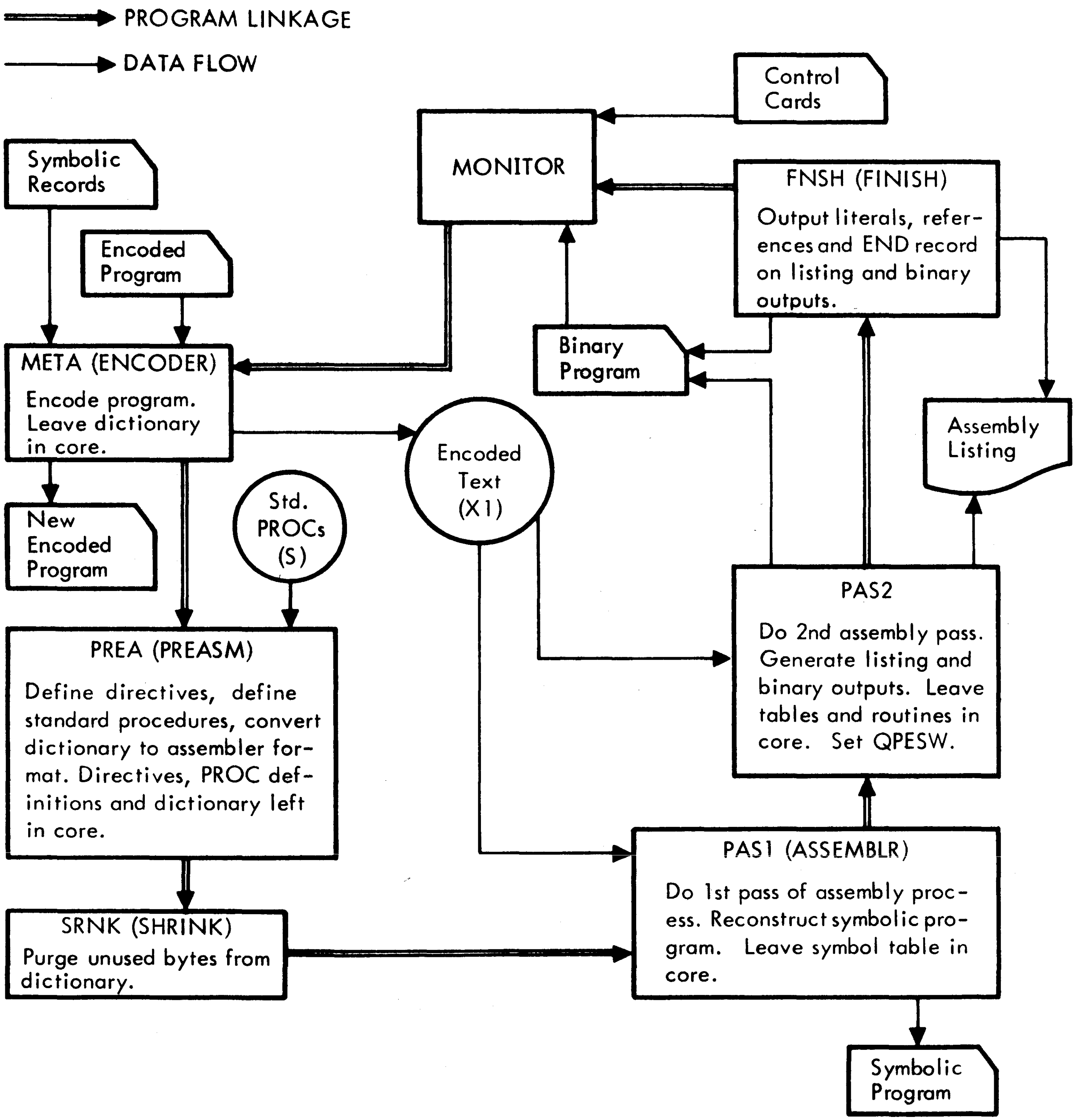


Figure 1-1. MONITOR-META-SYMBOL Data Flow

processes the selected standard system procedures from the system tape, defining only those procedures which are used within the user's program. The preassembler also defines the directives for the assembler and converts the dictionary from the ENCODER format to the format used by the assembler. (See Section 4, Item Formats.) The preassembler then calls the tape loader to load SRNK (SHRINK).

SHRINK purges the dictionary and byte table constructed by the preassembler to remove unused bytes. The sole purpose of SHRINK is to minimize the table size and thus maximize available working storage.

SHRINK calls the tape loader to load PAS1 (assembler pass 1). The input to the assembler is the encoded text tape (X1) generated by the ENCODER. During pass 1, the assembler defines the labels used within the program and determines program size in order to set the starting location for literals. If a symbolic regeneration is requested, the symbolic program is output during pass 1 of the assembler. At the conclusion of pass 1, the external symbol (entry points) definitions are output in type 1 records, and the external programmed operator definitions are output on type 2 records, provided binary output has been requested. If either listing or binary output has been requested, assembler pass 1 calls the loader to load PAS2 (assembler pass 2). If no additional output has been requested control returns to MONITOR.

PAS2 is the data-generating pass of the assembly system. Using X1 as input, PAS2 generates the binary output records and assembly listing. If errors are detected, cell QPESW is set to control the type of return to MONITOR. This cell is important in "assemble-and-go" operation as a measure of the quality of the binary output. During this second assembly pass, literals are defined and references to externally defined symbols are flagged and linked. At the conclusion of the second assembly pass, PAS2 calls the tape loader to load FNSH (FINISH).

FINISH punches and lists the literals, punches and lists the external symbol references, and punches the transfer or end card for the binary program file. Upon completion MONITOR is reloaded by calling the bootstrap routine which has been retained in high memory.

SECTION 2

DETAILED DESCRIPTION OF THE META-SYMBOL ASSEMBLY SYSTEM

PURPOSES OF THE ASSEMBLY SYSTEM

The primary purpose of the assembly system is to provide users of SDS computers a processor capable of translating symbolic lines of code (written in an advanced assembly language) to machine language and to provide the user a listing of the machine language generated as well as a loadable program tape or deck.

Secondary purposes of the assembly system provide:

1. The user the capability to obtain a condensed representation of the symbolic source program (the encoded program).
2. The capabilities to modify symbolically an encoded program and to recover from the encoded program the symbolic program it represents.
3. The capability to assemble a program or group of programs and to load and execute the resulting machine language output in essentially a single operation with a minimum of human intervention.
4. The user the capability to assemble programs written in the SYMBOL, SYMBOL 4, or SYMBOL 8 programming languages.
5. A system capable of running on a wide range of machine configurations. This includes the ability to allow the user to assign peripheral devices to the various assembly functions in a convenient manner at assembly time and with a minimum of restrictions.
6. A processor capable of generating machine code for machines other than that on which META-SYMBOL is operating.

GENERAL CONSIDERATIONS ABOUT META-SYMBOL

Those routines which process the encoded information on the intermediate output tape X1 and convert it to a machine language program are grouped into three separate machine overlays. These overlays, PAS1 (ASSEMBLR), PAS2, and FNSH (FINISH) are the assembler, META-SYMBOL.

META-SYMBOL is a 2-pass assembly system with the separate passes PAS1 and PAS2. FINISH is the end logic of PAS2 and is maintained as a separate overlay for space economy. (See Figure 2.)

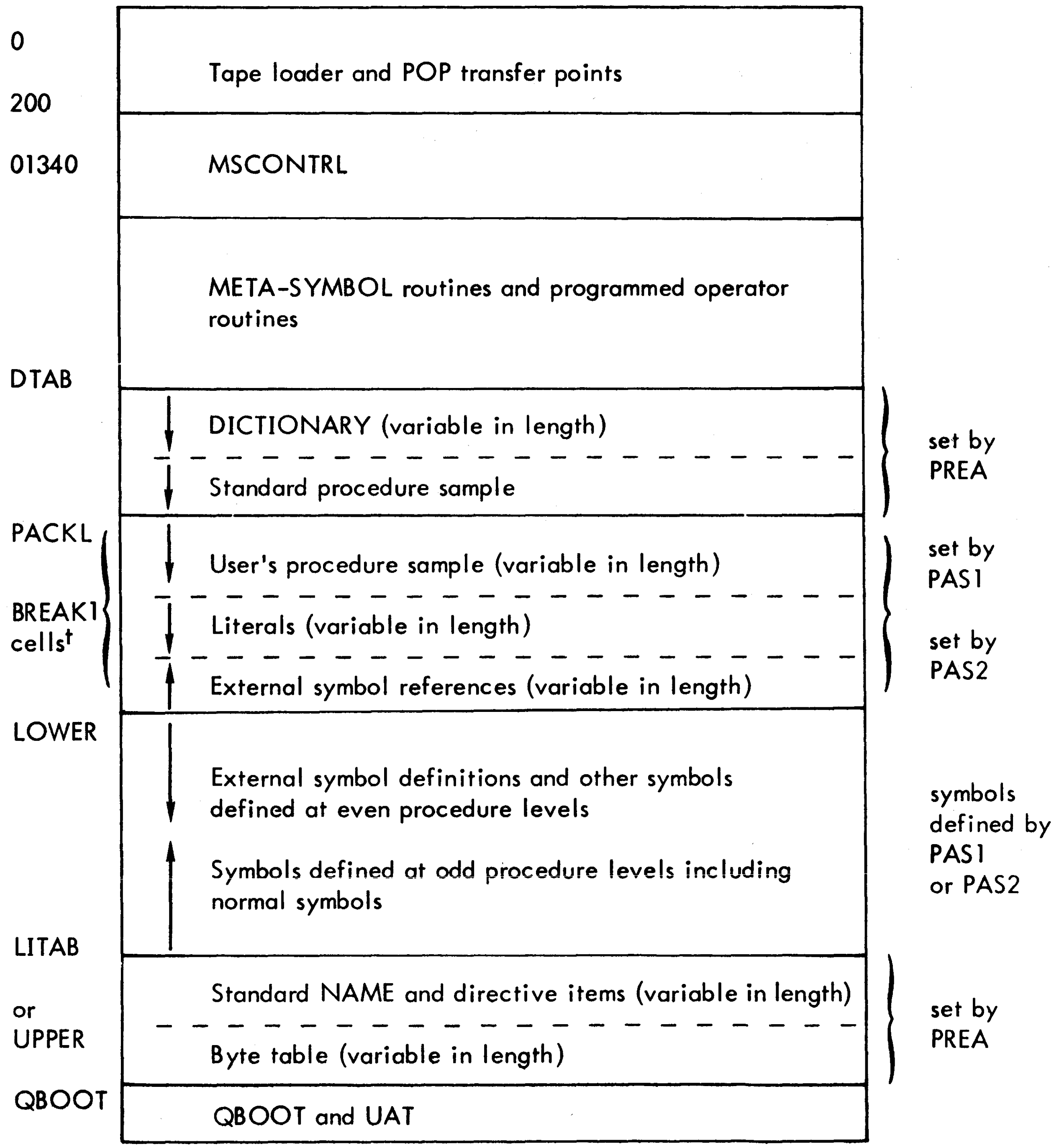


Figure 2-1. META-SYMBOL Core Layout

† Value of BREAK1 depends on machine size.

Many of the functions and routines of PAS1 and PAS2 are identical; therefore, where a routine is present in both programs, within this document it is described with PAS1 and cross-referenced within the PAS2 descriptions.

META-SYMBOL Symbol Table Processing

META-SYMBOL enters symbol definitions into the symbol table from both ends; the determination of which end of the table to use is a function of the current procedure level and the presence or absence of the external symbol flag (S) associated with the symbol.

Each time a procedure reference is encountered, the direction of the symbol table is reversed (normally, symbols are entered from high to low core), and symbols appearing within the procedure are thereby defined at the alternate end of the symbol table. When the procedure reference is completed, the table direction is again reversed. When a leading $ (dollar sign) is found on a label, a flag is set so that the label will be defined at the opposite side of the symbol table.

All symbols defined within a procedure at its normal level are purged when the procedure is completed (this includes the list of parameters for the procedure) by resetting the appropriate pointer for the next available cell in the symbol table (UPPER or LOWER) and relinking the pointers in the byte table for symbols purged. Labels preceded by $ marks are all external (saved) for one procedure level outside the level at which they were defined.

Input/Output Routines Used

All I/O routines used by META-SYMBOL are initialized as to unit and channel assignments. All I/O routines used, except the listing routines, are standard routines in MSCONTRL.

Processing of Procedures

Inherent in the concept of procedure processing is the procedure storage table. This table is sufficient to allow for six levels of procedures or functions and each level has 27_8 cells of information. (See Item Formats, Section 4.)

Normal level for processing code is level 1 [indicated by (PLV = PLVT = PLV1 = 27_8)]. For each current procedure reference level, the level indicators PLV and PLVT are incremented by the length of the table. The entries within this table reflect information to be retained during the processing of the procedure or functions. For example, the location of the next character to be obtained when the procedure is completed, the tentative definition of any label on the procedure reference line, and the value of the location counter when the procedure was referenced are all retained in this table. When discussing the value of parameters saved in the procedure storage table, that value associated with the current level is implied unless specified otherwise.

References to procedures are processed almost as separate programs. A double pass is made over the procedure sample (unless the procedure is defined as a single-pass procedure) during the second assembly pass so that forward references to local symbols within the procedure may be made. In general, any line of code permitted outside the procedure is allowed within the procedure.

META-SYMBOL COMPONENT PROGRAMS

The component programs of the META-SYMBOL assembly system are grouped as five segments:

1. Loader and file control routine
2. ENCODER, S4B, MON1
3. PREA (PREASM), SRNK (SHRINK)
4. PAS1 (ASSEMBLR) (Assembler Pass 1)
5. PAS2 (Assembler Pass 2), FNSH (FINISH)

Each of the segments 2 through 5 are independent entities and may not reference each other; however, all segments may reference the control routine.

Basic Tape Loader

This program loads absolute programs from MONARCH system tape.

MSCONTRL

This program contains the input/output and function control cells initialized by MON1. MSCONTRL is resident in lower memory during the entire assembly process. MSCONTRL contains those I/O routines used by two or more overlays of the assembly system.

ENCODER

This program reads symbolic input, encoded input, or symbolic corrections and encoded input. It also produces the intermediate output tape to be used as input to the assembler and produces new encoded program if requested. It leaves the dictionary and balanced tree search table in core for the preassembler (PREA).

S4B

This program is called by the ENCODER to translate symbolic input from SYMBOL 4 or SYMBOL 8 format to META-SYMBOL format.

MON1

This program is called by the ENCODER to initialize the I/O control cells for the system. MON1 also copies corrections to scratch tape X2 when the symbolic corrections and encoded inputs are on the same input device.

PREA (PRESM)

The preassembler program defines directives, processes the selected standard procedure file, and reformats the dictionary in preparation to starting the assembly process. The standard procedures are located on the system tape between PREA and SRNK.

SRNK (SHRINK)

This program purges the dictionary and byte table left by PREA to remove bytes from the standard procedure deck which are not referred to in the user's program or by that portion of the standard procedures needed to process the user's program.

PAS1 (ASSEMBLR)

This is the first pass of the assembler. PAS1 reads the intermediate input tape constructed by the ENCODER. This pass also defines the symbols used within the user's program, determines the origin of the literals, establishes the origin of the literal and reference tables, processes user PROC and FUNC sample definitions, and defines procedure NAMEs and programmed operators. At the conclusion of PAS1 the external symbol and programmed operator definitions are output on the binary output device. If symbolic output is requested, it is generated by PAS1.

POPs

These are the programmed operators for either the 920/930 computers or 910/925 computers depending on installation. The POPs are loaded separately with the ENCODER, PREASM, and ASSEMBLR.

PAS2

This is the second pass of the assembler. PAS2 generates the binary and listing outputs. During the second pass, symbols are redefined; however, NAME definitions are not redefined. Therefore, no local NAMEs are permitted within nested sample. Literals are generated and references to externally defined symbols are flagged and linked.

FNSH (FINISH)

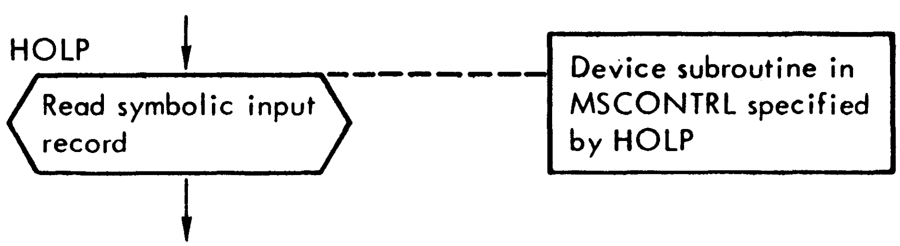
This program outputs the literals and external references on the listing and binary outputs. It also prints the END line and outputs the transfer or end record.

FLOWCHART CONVENTIONS

Included in MSCONTRL and ENCODER are I/O device subroutines which are not called by name:

MSCONTRL	ENCODER
EFC	CRDB
EFPT	CRDH
EOF	HOLP
PCB	RDPT
PCH	RPTB
PPTB	
RMTB	
RMTBU	
WMTB	

For example, the following is a call to a subroutine:



Thus, in location HOLP is a pointer specifying which device subroutine is to be called.

A transfer to MONITR means a branch to the MONARCH monitor, which is described in a separate document.

Branch tables are used throughout META-SYMBOL:

Branch Table	Accessed from	Subroutines Accessed	Page
T1	ENCODER	AL BLANK DOT EORC NU QUOTE SPEC	
Directive Number	PREA	FUN (function) NAM (name) PRO (procedure) SEND (end)	
DIRT (directive routines)	PAS1 and PAS2	AORG BCD DED DO END EQU FORM FUNC NAME ORG PAGE POPD PROC RES TEXTR	
TYP	PAS1 and PAS2	DATAT (end cards) DEF (types 1 and 2) ENDM (END card with transfer address) ENDN (END card without transfer address) POPRD (POP reference or DEF)	

SECTION 2

DETAILED DESCRIPTION OF THE META-SYMBOL ASSEMBLY SYSTEM

PURPOSES OF THE ASSEMBLY SYSTEM

The primary purpose of the assembly system is to provide users of SDS computers a processor capable of translating symbolic lines of code (written in an advanced assembly language) to machine language and to provide the user a listing of the machine language generated as well as a loadable program tape or deck.

Secondary purposes of the assembly system provide:

1. The user the capability to obtain a condensed representation of the symbolic source program (the encoded program).
2. The capabilities to modify symbolically an encoded program and to recover from the encoded program the symbolic program it represents.
3. The capability to assemble a program or group of programs and to load and execute the resulting machine language output in essentially a single operation with a minimum of human intervention.
4. The user the capability to assembly programs written in the SYMBOL, SYMBOL 4, or SYMBOL 8 programming languages.
5. A system capable of running on a wide range of machine configurations. This includes the ability to allow the user to assign peripheral devices to the various assembly functions in a convenient manner at assembly time and with a minimum of restrictions.
6. A processor capable of generating machine code for machines other than that on which META-SYMBOL is operating.

GENERAL CONSIDERATIONS ABOUT META-SYMBOL

Those routines which process the encoded information on the intermediate output tape X1 and convert it to a machine language program are grouped into three separate machine overlays. These overlays, PAS1 (ASSEMBLR), PAS2, and FNSH (FINISH) are the assembler, META-SYMBOL.

META-SYMBOL is a 2-pass assembly system with the separate passes PAS1·and PAS2. FINISH is the end logic of PAS2 and is maintained as a separate overlay for space economy. (See Figure 2.)

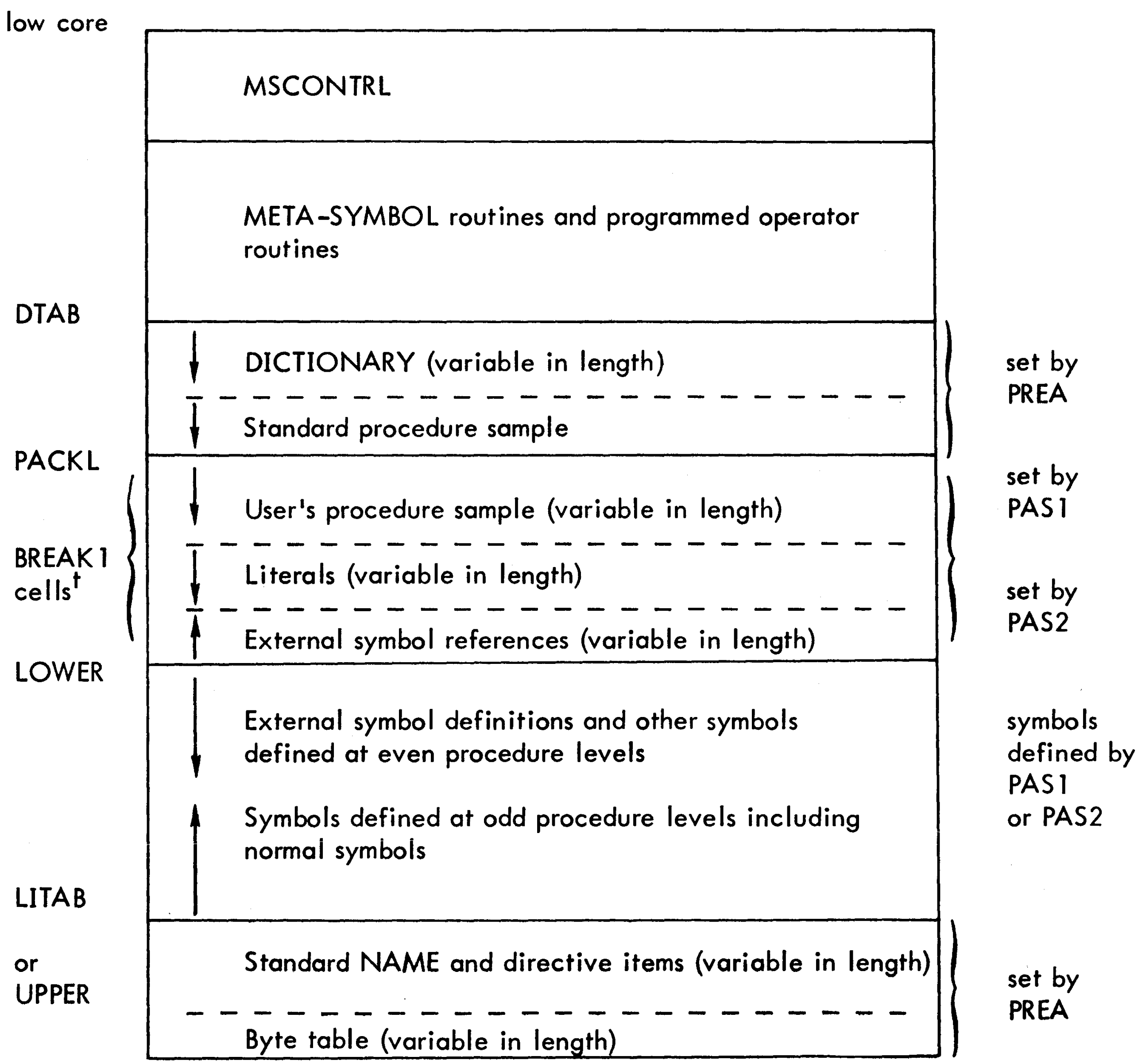


Figure 2-1. META-SYMBOL Core Layout

[t]Value of BREAK1 depends on machine size.

Many of the functions and routines of PAS1 and PAS2 are identical; therefore, where a routine is present in both programs, within this document it is described with PAS1 and cross-referenced within the PAS2 descriptions.

META-SYMBOL Symbol Table Processing

META-SYMBOL enters symbol definitions into the symbol table from both ends; the determination of which end of the table to use is a function of the current procedure level and the presence or absence of the external symbol flag ($) associated with the symbol.

Each time a procedure reference is encountered, the direction of the symbol table is reversed (normally, symbols are entered from high to low core), and symbols appearing within the procedure are thereby defined at the alternate end of the symbol table. When the procedure reference is completed, the table direction is again reversed. When a leading $ (dollar sign) is found on a label, a flag is set so that the label will be defined at the opposite side of the symbol table.

All symbols defined within a procedure at its normal level are purged when the procedure is completed (this includes the list of parameters for the procedure) by resetting the appropriate pointer for the next available cell in the symbol table (UPPER or LOWER) and relinking the pointers in the byte table for symbols purged. Labels preceded by $ marks are all external (saved) for one procedure level outside the level at which they were defined.

Processing of Procedures

Inherent in the concept of procedure processing is the procedure storage table. This table is sufficient to allow for six levels of procedures or functions and each level has 27_8 cells of information. (See Item Formats, Section 4.)

Normal level for processing code is level 1 [indicated by (PLV = PLVT = PLV1 = 27_8)]. For each current procedure reference level, the level indicators PLV and PLVT are incremented by the length of the table. The entries within this table reflect information to be retained during the processing of the procedure or functions. For example, the location of the next character to be obtained when the procedure is completed, the tentative definition of any label on the procedure reference line, and the value of the location counter when the procedure was referenced are all retained in this table. When discussing the value of parameters saved in the procedure storage table, that value associated with the current level is implied unless specified otherwise.

References to procedures are processed almost as separate programs. A double pass is made over the procedure sample (unless the procedure is defined as a single-pass procedure) during the second assembly pass so that forward references to local symbols within the procedure may be made. In general, any line of code permitted outside the procedure is allowed within the procedure.

META-SYMBOL COMPONENT PROGRAMS

The component programs of the META-SYMBOL assembly system are grouped as five segments:

1. Loader and file control routine
2. ENCODER, S4B
3. PREA (PREASM), SRNK (SHRINK)
4. PAS1 (ASSEMBLR) (Assembler Pass 1)
5. PAS2 (Assembler Pass 2), FNSH (FINISH)

Each of the segments 2 through 5 are independent entities and may not reference each other; however, all segments may reference the control routine.

Loader

This program loads absolute programs from MONITOR system tape.

MSCONTRL

MSCONTRL contains the I/O file control routines and the communication cells used by two or more overlays of the assembly system.

ENCODER

This program reads symbolic input, encoded input, or symbolic corrections and encoded input. It also produces the intermediate output tape to be used as input to the assembler and produces new encoded program if requested. It leaves the dictionary and balanced tree search table in core for the preassembler (PREA).

S4B

This program is called by the ENCODER to translate symbolic input from SYMBOL 4 or SYMBOL 8 format to META-SYMBOL format.

PREA (PRESM)

The preassembler program defines directives, processes the selected standard procedure file, and reformats the dictionary in preparation to starting the assembly process. The standard procedures are located on the system tape between PREA and SRNK.

SRNK (SHRINK)

This program purges the dictionary and byte table left by PREA to remove bytes from the standard procedure deck which are not referred to in the user's program or by that portion of the standard procedures needed to process the user's program.

PAS1 (ASSEMBLR)

This is the first pass of the assembler. PAS1 reads the intermediate input tape constructed by the ENCODER. This pass also defines the symbols used within the user's program, determines the origin of the literals, establishes the origin of the literal and reference tables, processes user PROC and FUNC sample definitions, and defines procedure NAMEs and programmed operators. At the conclusion of PAS1 the external symbol and programmed operator definitions are output on the binary output device. If symbolic output is requested, it is generated by PAS1.

PAS2

This is the second pass of the assembler. PAS2 generates the binary and listing outputs. During the second pass, symbols are redefined; however, NAME definitions are not redefined. Therefore, no local NAMEs are permitted within nested sample. Literals are generated and references to externally defined symbols are flagged and linked.

FNSH (FINISH)

This program outputs the literals and external references on the listing and binary outputs. It also prints the END line and outputs the transfer or end record.

FLOWCHART CONVENTIONS

Branch tables are used throughout META-SYMBOL:

Branch Table	Accessed from	Subroutines Accessed	Page
T1	ENCODER	AL BLANK DOT EORC NU QUOTE SPEC	
Directive Number	PREA	FUN (function) NAM (name) PRO (procedure) SEND (end)	
DIRT (directive routines)	PAS1 and PAS2	AORG BCD DED DO END EQU FORM FUNC NAME ORG PAGE POPD PROC RES TEXTR	
TYP	PAS1 and PAS2	DATAT (end cards) DEF (types 1 and 2) ENDM (END card with transfer address) ENDN (END card without transfer address) POPRD (POP reference or DEF)	

a

SECTION 3
INDIVIDUAL DESCRIPTIONS AND FLOWCHARTS

The routines for META-SYMBOL are described inthe following orders:

a. Loader and MSCONTRL
b. ENCODER, S4B, and MON1
c. PREA and SRNK
d. PAS1
e. PAS2

SDS SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Basic Tape Loader

PURPOSE: To load absolute sections of the assembly system into core and to transfer control to them.

ACTION: The tape loader reads records from the system tape until it finds an identification record at level 2 (Δ 2 in characters 1 and 2) with the first four characters of the segment name identical to the contents of the A register at entry. The following records are then loaded until a transfer record is reached, at which point the loader branches to the location indicated as the starting address. All records loaded are checksummed, and a checksum error results in a HALT with an address of 4 displayed in C. Stepping causes the record to be accepted. A tape read error results in a halt with an address of 1 displayed.

PROGRAMMING TECHNIQUES: The tape loader is an absolute routine originated at 3 with a starting location at 4. The routine occupies low memory up to and including cell 177_8 except that cells 100_8 through 135_8 inclusive are not used and are available for programmed operator use. The tape read routine used is not self-initializing and assumes the system tape to be on unit 0 of the W buffer. The tape loader does not supply its own input buffers. Locations 1427_8 through 1504_8 are used as input buffers so programs loading data into this region cannot be loaded by tape loader.

CALLING SEQUENCE: Program ID to A register
BRU 4

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Basic RAD Loader

PURPOSE: Provide linkages to the RAD input/output routines for 900 Series RAD META-SYMBOL.

ACTION: The basic RAD loader contains the following routines, called individually

SCTP	Scan system file for $\Delta 2$ record
RDTP	Read system file
RDF	Read RAD record
WDF	Write RAD record
EFDF	Close RAD file
RWTST	Rewind RAD file
SETUP	Initialize I/O packet for RAD

SCTP is called by the preassembler when scanning for the $\Delta 2$ PROC records, and in the loading operation to load the next overlay from disc.

RDTP is called by the preassembler to read the encoded PROC images and by the SCTP routine to obtain the $\Delta 2$ records.

RDF is a generalized RAD read linkage routine to read records from scratch files X1 and X2, and calls SETUP to initialize the RAD read calling sequence.

WDF is a generalized linkage to the RAD write routine and is used to write the X1, X2 and BO files on the RAD. WDF calls SETUP to initialize the RAD I/O calling sequence.

EFDF is a generalized linking routine to close RAD output files.

RWTST is called from the rewind routine REWW, in MSCONTRL, to rewind and open a disc file on scratch files X1 and X2 or binary output file BO.

SETUP initialized the calling sequence to the RAD I/O routines.

PROGRAMMING TECHNIQUES:

The routines in the Basic RAD Loader have absolute origins. The routine uses space from cell 3 to 177_8 inclusive, except for locations 100-126_8 which are reserved for POPs. The RDTP routine uses the 40_{10} words starting at location 1444_8 as an input buffer area.

CALLING SEQUENCE:

The general calling sequence for the I/O routines on disc is:

LDX	Disc I/O control word
LDA	Buffer location
LDB	Record length
BRM	I/O linkage routine

If the record length is to be taken from the data itself, the B register should contain a minus 1 on entry to the disc I/O linkage routine. The rewind routine is entered by placing the relative UAT location for the file in X2 (e.g., 0 for system file, -2 for scratch file X2, 2 for scratch file X1) and executing a BRM REWW.

The loader is entered by placing the alphanumeric values of the first 4 characters of the segment name in the A register and branching to location 4. For example:

```
LDA  =  'PREA'
BRU     4
```

MEMORY REQUIREMENTS:

Locations 3-77_8, 127_8-177_8

SUBROUTINES USED:

RAD File Management Routine

MEMORY REQUIREMENTS:	See "Programming Techniques" above.
SUBROUTINES USED:	None

SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Resident I/O routines (MSCONTRL)

PURPOSE: To provide I/O control information and standard input/output file control and device handling routines for the assembly system.

ACTION: MSCONTRL as such is never executed; it is merely a collection of routines and control information to be used by the assembly system.

PROGRAMMING TECHNIQUES: MSCONTRL is an absolute program loaded in the first overlay of the system and retained in low core throughout the assembly run. It is the last program loaded with the ENCODER and contains the transfer to ENCODER to start the assembly process. The file handling routines contained in MSCONTRL all assume an input/output control packet which is part of the input/output buffer. These routines, INPUT, OUTPUT, OPEN, CLOSE, READ and WRITE, use a packet of the following format:

INPUT/OUTPUT PACKET FORMAT

Word	Read	Write
0	location from which to load next data word	location into which to store next data word
1	not used (used as temporary by READ)	full word checksum for words stored in buffer
2	last location of buffer	last location of buffer
3	location of input subroutine	location of output subroutine
4	not used	location of end-of-file subroutine
5	not used (used as temporary by READ)	dummy control word (used to initialize control word)

PROGRAMMING TECHNIQUES: (cont.)

Word	Read	Write
6	buffer. First word is control word	buffer. First word is control word
7-45	remainder of buffer	remainder of buffer

For formats of the I/O control cells, see MON1 program description.

The I/O device handling routines in MSCONTRL are all self-initializing as to unit and channel, and none of them depends on the existence of a buffer interlace system. When called, the routines depend on the following information in the machine registers:

A register	address of first word to transmit
B register	number of words to transmit
Index register	standard I/O control word

When entered, the file control routines assume that the index register contains the location of the I/O packet.

CALLING SEQUENCE: MSCONTRL as such is never executed.

MEMORY REQUIREMENTS: MSCONTRL has an absolute origin at location 200_8 and uses core from that point to location 1336_8. Since MSCONTRL has several routines and control words which are addressed by programs not loaded with MSCONTRL and since cell 1337_8 is the origin of the ENCODER routine, any change in the size or ordering of MSCONTRL is likely to necessitate the reassembly of several other major sections of the system.

SUBROUTINES USED: Not applicable.

SDS SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 612001

IDENTIFICATION: I/O file routines (MSCONTRL)

PURPOSE: To provide I/O control information and standard input/output file control routines for the assembly system. Communication cells used by more than one overlay are in FILE.

ACTION: MSCONTRL is an absolute program loaded in the first overlay of the system and retained in low core throughout the assembly run. The file handling routines contained in MSCONTRL all assume an input/output control packet which is part of the input/output buffer. These routines, INPUT, OUTPUT, OPEN, CLOSE, READ and WRITE, use a packet of the following format:

INPUT/OUTPUT PACKET FORMAT

Word	Read	Write
0	location from which to load next data word	location into which to store next data word
1	not used (used as temporary by READ)	full word checksum for words stored in buffer
2	last location of buffer	last location of buffer
3	read flag and location of file description table	not used
4	not used	write flag and location of file description table
5	not used (used as temporary by READ)	dummy control word (used to initialize control word)
6	buffer. First word is control word	buffer. First word is control word
7-45	remainder of buffer	remainder of buffer

CALLING SEQUENCE: MSCONTRL as such is never executed.

MEMORY REQUIREMENTS: 475_8 cells

SUBROUTINES USED: Not applicable.

ENTRY POINTS TO LOADER AND MSCONTRL SUBROUTINES

Entry	Page Description	Page Flowchart	Entry	Page Description	Page Flowchart
ABORT	3-12	3-32	PBC	3-24	3-38
CLOSE	3-8	3-34	PCB		3-37
EFC	3-29	3-38	PCH	3-26	3-38
EFMT	3-21	3-37	PPTB	3-15	3-33
EFPT	3-17	3-33	R	3-1	3-31
GTUNT	3-30	3-38	READ	3-11	3-35
IAW	3-27	3-38	READY		3-36
INEFC	3-28	3-38	REWW	3-13	3-32
INEFPT	3-16	3-33	RMTB	3-22	3-37
INPCB	3-23	3-37	RMTBU	3-18	3-35
INPCH	3-25	3-38	TBOT		3-36
INPPT	3-14	3-32	TYPMSG		3-32
INPUT	3-10	3-34	WMTB	3-20	3-36
LOADER	3-1	3-31	WMTBU	3-19	3-35
OPEN	3-6	3-33	WRITE	3-9	3-34
OUTPUT	3-7	3-33			

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 0420[illegible]
Catalog No. 9300: 612001

IDENTIFICATION: Open a standard I/O file (OPEN)

PURPOSE: To initialize an I/O packet to output a file.

ACTION: OPEN clears word 2 of the packet (checksum) and sets word 1 of the packet to the location of the seventh word (first buffer word) and word 3 to the location of the 46th word of the packet (last word of buffer).

PROGRAMMING TECHNIQUES: OPEN assumes the index register contains the location of the packet. OPEN is an absolute routine assembled as part of MSCONTRL.

CALLING SEQUENCE:
Packet location to index register
BRM   OPEN

MEMORY REQUIREMENTS: 11_8 cells

SUBROUTINE USED: None.

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 0420
Catalog No. 9300: 6120

IDENTIFICATION: Output words to an output file (OUTPUT)

PURPOSE: To store an output word located in the A register into an output buffer and to empty the buffer when filled.

ACTION: OUTPUT stores the contents of the A register into the next buffer location and increments the location. The full word checksum is set in the second word of the packet. When the buffer becomes full, OUTPUT empties the buffer by calling WRITE.

PROGRAMMING TECHNIQUES: The packet location is assumed to be in the index register when OUTPUT is entered. OUTPUT is an absolute program assembled as part of MSCONTRL.

CALLING SEQUENCE:

Word to be output to A register
Location of packet to index register
BRM OUTPUT

MEMORY REQUIREMENTS: 12_8 cells

SUBROUTINES USED: WRITE

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Close an output file (CLOSE)

PURPOSE: To close an output file by emptying the output buffer and writing an end-of-file mark.

ACTION: CLOSE calls WRITE to empty the buffer associated with the packet at the location given by the index register. CLOSE then calls the end-of-file routine at the location indicated in the fifth word of the packet.

PROGRAMMING TECHNIQUES: CLOSE is a standard I/O file maintenance routine using the standard packet format and register assignments. CLOSE is an absolute routine assembled as part of MSCONTRL.

CALLING SEQUENCE:

Location of packet to index register
BRM CLOSE

MEMORY REQUIREMENTS: 6 cells

SUBROUTINES USED:

WRITE
Any of the standard end-of-file device routines

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 04
Catalog No. 9300: 61

IDENTIFICATION: Write the contents of a buffer (WRITE)

PURPOSE: To write the contents of a buffer onto an output file.

ACTION: If the buffer addressed by the index register is empty, WRITE exists; if it is not, the word count is saved and the control word is formed and stored in the seventh word of the packet. The location of the seventh packet word is placed in the A register and the word count in the B register; WRITE calls the I/O routine addressed by the fourth word of the packet. OPEN is called to reinitialize the packet.

PROGRAMMING TECHNIQUES: WRITE uses the standard I/O file control routine packet format and register contents. WRITE is an absolute routine assembled as part of MSCONTRL.

CALLING SEQUENCE: Location of packet to index register
BRM WRITE

MEMORY REQUIREMENTS: 37_8 cells

SUBROUTINES USED: OPEN
Any of standard output device handling routines

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Obtain the next word from an input file (INPUT)

PURPOSE: To obtain in the A register the next word from a specified input file.

ACTION: If the input buffer is empty, INPUT calls READ to obtain the next record. An end-of-file return from READ results in an end-of-file exit from INPUT. The next word of input is loaded into the A register, and the buffer location is incremented.

PROGRAMMING TECHNIQUES: INPUT is a standard file maintenance routine and assumes the presence of an I/O packet addressed by the index register. INPUT is an absolute routine assembled as part of MSCONTRL.

CALLING SEQUENCE:

```
Location of packet to index register
BRM    INPUT
End-of-file return
Normal return
```

MEMORY REQUIREMENTS: 14_8 cells

SUBROUTINES USED: READ

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042(
Catalog No. 9300: 612(

IDENTIFICATION: Read the next record of an input file (READ)

PURPOSE: To obtain the next record from the specified input file and to verify its correctness by computing the checksum.

ACTION: READ loads the A register with the location of the seventh word of the specified I/O packet, loads the B register with 40_{10}, and calls the I/O device routine addressed by the fourth word of the packet. If the read results in an end of file, READ exits through its end-of-file return. READ computes the checksum for the record and verifies the record by comparing the computed and stated checksums. A checksum discrepancy results in a halt with a NOP 2 displayed in C. Stepping causes the record to be accepted as read.

PROGRAMMING TECHNIQUES: READ is a standard file processing routine and assumes a standard packet addressed by the contents of the index register. READ is an absolute program assembled as part of MSCONTRL.

CALLING SEQUENCE: Location of I/O packet to index register
BRM READ

MEMORY REQUIREMENTS: 47_8 cells

SUBROUTINES USED: 900 Series Only: Any of the standard binary input device handling routines

9300 Only: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Write on-line typewriter message and call MONARCH (ABORT)

PURPOSE: To print an assembly system error message and return control to MONARCH.

ACTION: ABORT stores the contents of the A register (error message code) into the skeletal error message and types the error message. The error control switch QPESW in the UAT is set, and control goes to QBOOT to reload MONARCH.

PROGRAMMING TECHNIQUES: The A register contains the error message code when ABORT is entered. ABORT is an absolute program assembled as part of MSCONTRL.

CALLING SEQUENCE:

Error code to A register
BRU ABORT

MEMORY REQUIREMENTS: 25_8 cells

SUBROUTINES USED: None. The typewriter routine used to type the error message in this case is assumed to be part of ABORT.

SDS SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Rewind magnetic tapes (REWW)

PURPOSE: To rewind the magnetic tape specified.

ACTION: REWW constructs a rewind instruction by determining the proper unit and channel designations from the UAT entry and executes that instruction.

PROGRAMMING TECHNIQUES: The index register at entry to REWW contains the location, relative to QSYS, of the UAT entry to be used in determining unit and channel assignments. REWW is an absolute routine assembled as part of MSCONTRL.

CALLING SEQUENCE: UAT relative location to index register
BRM REWW

MEMORY REQUIREMENTS: 15_8 cells

SUBROUTINES USED: None

NOTE: In the RAD MONARCH system, REWW calls RWTST to determine whether the file is allocated to magnetic tape or to the RAD. When the file is RAD-allocated, the File Management Routine is called in order to rewind the file.

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Initialize binary paper tape punch routine (INPPT)

PURPOSE: To initialize with respect to unit and channel the binary paper tape punch routine, PPTB.

ACTION: INPPT obtains the unit and channel assignments by calling GTUNT. It then sets the I/O instructions in PPTB.

PROGRAMMING TECHNIQUES: INPPT is an absolute routine assembled as part of MSCONTRL and is an extension of PPTB.

CALLING SEQUENCE:
I/O control word to index register
BRM INPPT

MEMORY REQUIREMENTS: 25_8 cells

SUBROUTINES USED: GTUNT

SDS SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Punch paper tape binary (PPTB)

PURPOSE: To punch a record on paper tape in the binary mode.

ACTION: PPTB calls IAW to obtain the buffer address and INPPT to initialize its I/O instructions with respect to unit and channel. PPTB then outputs the specified number of words from the specified location by executing a MIW loop. A buffer error results in a halt with a NOP 4 displayed in the C register; stepping permits the routine to conclude as though no error had occurred.

PROGRAMMING TECHNIQUES: PPTB is a device handling routine designed to work with the standard file processing routines. PPTB is an absolute routine assembled as part of MSCONTRL.

CALLING SEQUENCE:

Buffer location to A register
Word count to B register
Control word to index register
BRM PPTB

MEMORY REQUIREMENTS: 16_8 cells

SUBROUTINES USED:

IAW
INPPT

SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Initialize the paper tape end-of-file routine (INEFPT)

PURPOSE: To initialize the end-of-file routine for paper tape, EFPT, as to unit and channel assignments.

ACTION: INEFPT calls GTUNT to obtain the channel and unit assignments which are used to initialize the I/O instructions in EFPT.

PROGRAMMING TECHNIQUES: INEFPT is an absolute routine assembled as part of MSCONTRL and is an extension to EFPT.

CALLING SEQUENCE: I/O Control word to index register
BRM INEFPT

MEMORY REQUIREMENTS: 13_8 cells

SUBROUTINES USED: GTUNT

SDS SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Feed blank paper tape (EFPT)

PURPOSE: To feed blank paper tape following an output paper tape file.

ACTION: EFPT calls INEFPT to set channel and unit assignments and then spaces blank tape.

PROGRAMMING TECHNIQUES: EFPT is designed to work with the standard file processing routines and is an absolute routine assembled as part of MSCONTRL.

CALLING SEQUENCE:
I/O control word to index register
BRM EFPT

MEMORY REQUIREMENTS: 11_8 cells

SUBROUTINES USED: INEFPT

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Initialize the magnetic tape read routine (RMTBU)

PURPOSE: To initialize the I/O instructions in RMTB as to mode, unit, and channel.

ACTION: RMTBU initializes the I/O instructions remotely executed by RMTB as to unit, channel, and mode (decimal or binary). RMTBU calls GTUNT to obtain the unit and channel designation in the proper format to initialize the I/O instructions within RMTB.

PROGRAMMING TECHNIQUES: RMTBU is a logical extension of the RMTB routine and makes use of the fact that RMTB is designed to work with the file processing routines and has the normal contents in the registers when called. RMTBU is an absolute routine assembled as part of MSCONTRL.

CALLING SEQUENCE:

Bits 0 through 9 of I/O control word to bits 14 through 23 of A register
I/O control word to TEMP + 3
BRM RMTBU

MEMORY REQUIREMENTS: 32_8 cells

SUBROUTINES USED: GTUNT

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Initialize magnetic tape write routine (WMTBU)

PURPOSE: To initialize the write end-of-file routine EFMT and the magnetic tape write routine, WMTB, as to mode, unit, and channel.

ACTION: WMTBU initializes the I/O instructions remotely executed by WMTB and EFMT as to unit, channel, and mode. WMTBU calls GTUNT to obtain the channel and unit designations in the format to initialize the I/O instructions within WMTB and EFMT.

PROGRAMMING TECHNIQUES: WMTBU is a logical extension of the routines to write magnetic tape. It assumes on entry that an I/O control word has been stored in WCNT and that the high order ten bits of that control word are in the low order ten bits of the A register. WMTBU is an absolute routine assembled as part of MSCONTRL.

CALLING SEQUENCE:

Control word to WCNT
Bits 0 through 9 of WCNT to bits 14 through 23 of A register
BRM WMTBU

MEMORY REQUIREMENTS: 57_8 cells

SUBROUTINES USED: GTUNT

SDS SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Write magnetic tape (WMTB)

PURPOSE: To write a record of a given size, from a specified buffer to magnetic tape on a given channel and unit and in the mode requested; to check for write errors and if necessary to erase and rewrite the record up to three times.

ACTION: WMTB calls IAW to set the buffer address and WMTBU to initialize the I/O instructions. WMTB tests the tape for ready and, if the tape is at load point, erases forward the required distance. If the tape is at the end-of-tape mark, WMTB exits; otherwise, the record is written by executing a WIM loop the required number of times. An error in writing causes the tape to be erased backward to remove the record; then WMTB rewrites it. If this fails, the record is erased backward and forward and then rewritten. This procedure is followed up to three times before WMTB halts. Stepping will cause the routine to try once more to write the record.

PROGRAMMING TECHNIQUES: WMTB is designed to be used with the standard file processing routines; it is an absolute program assembled as part of MSCONTRL.

CALLING SEQUENCE:

Location of buffer to A register
Number of words to write to B register
I/O control word to index register
BRM WMTB

MEMORY REQUIREMENTS: 76_8 cells

SUBROUTINES USED:

IAW
WMTBU

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Write end-of-file marks on magnetic tape (EFMT)

PURPOSE: To write a tape end-of-file mark on the specified magnetic tape.

ACTION: EFMT calls WMTBU to initialize the I/O instructions. Tape ready and beginning of tape status are checked after which EFMT writes a one-character record of 17_8 to the tape.

PROGRAMMING TECHNIQUES: EFMT is designed to work with the standard file processing routines and is an absolute routine assembled as part of MSCONTRL.

CALLING SEQUENCE:

I/O control word to index register
BRM EFMT

MEMORY REQUIREMENTS: 16_8 cells

SUBROUTINES USED:

WMTBU
Those portions of WMTB to check tape ready status and beginning of tape

SDS SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Read magnetic tape (RMTB)

PURPOSE: To obtain a record of given maximum size from a specified tape unit in the indicated mode and place it in the specified buffer.

ACTION: RMTB calls IAW to set the buffer address and RMTBU to initialize the I/O instructions. RMTB executes a WIM loop until an end of record is reached or until the indicated number of words have been read. If the record is less than a full word long or if the first word is ΔEOF, RMTB takes the end-of-file exit. A read error causes the routine to backspace and reread the tape up to ten times. An error still detected after ten attempts results in a halt. Stepping causes the record to be accepted as read.

PROGRAMMING TECHNIQUES: RMTB is designed to work with the standard file processing routines. RMTB is an absolute routine assembled as part of MSCONTRL.

CALLING SEQUENCE:

Buffer location to A register
Word count to B register
Standard control word to index register
BRM RMTB
End-of-file return
Normal return

MEMORY REQUIREMENTS: 60_8 cells

SUBROUTINES USED:

IAW
RMTBU

SDS SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Initialize the punch cards binary mode routine (INPCB)

PURPOSE: To initialize as to unit and channel the I/O instructions in the punch cards binary mode routine, PCB.

ACTION: INPCB calls GTUNT to get the unit and channel assignments which are used to set the I/O instructions in PCB.

PROGRAMMING TECHNIQUES: INPCB is a logical extension of the PCB routine and is an absolute routine assembled as part of MSCONTRL.

CALLING SEQUENCE:

I/O control word to index register
BRM INPCB

MEMORY REQUIREMENTS: 27_8 cells

SUBROUTINES USED: GTUNT

SDS SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Punch cards binary mode (PCB)

PURPOSE: To punch in the binary mode a record of given size from a specified buffer into a card on the unit and channel indicated.

ACTION: PCB calls IAW to set the buffer address and INPCB to initialize the I/O instructions. PCB then punches the card received by executing 12_{10} times a WIM loop for the number of words to be punched.

PROGRAMMING TECHNIQUES: PCB is designed to work with the standard file processing routines and is an absolute routine assembled as part of MSCONTRL.

CALLING SEQUENCE:

Buffer location to A register
Word count to B register
I/O control word to index register
BRM PCB

MEMORY REQUIREMENTS: 35_8 cells

SUBROUTINES USED:

IAW
INPCB

SDS SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION:	Initialize the routine to punch cards in the BCD or Hollerith mode (INPCH)
PURPOSE:	To initialize the I/O instructions in PCH as to unit and channel.
ACTION:	INPCH calls GTUNT to obtain channel and unit designations which are used to initialize the I/O instructions in PCH.
PROGRAMMING TECHNIQUES:	INPCH is a logical extension of the PCH routine and is an absolute routine assembled as part of MSCONTRL.
CALLING SEQUENCE:	I/O control word to index register BRM INPCH
MEMORY REQUIREMENTS:	22_8 cells
SUBROUTINES USED:	GTUNT

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Punch cards in BCD or Hollerith mode (PCH)

PURPOSE: To punch in the BCD mode a record of given length from a specified buffer to cards on the unit and channel indicated.

ACTION: PCH calls IAW to set the buffer address and INPCH to initialize its I/O instructions. It then outputs the record by executing a WIM loop the required number of times as determined by the word count. This loop is repeated 12 times.

PROGRAMMING TECHNIQUES: PCH is designed to work with the standard file processing routines and is an absolute routine assembled as part of MSCONTRL.

CALLING SEQUENCE:

Word count to B register
I/O control word to index register
Buffer location to A register
BRM PCH

MEMORY REQUIREMENTS: 21_8 cells

SUBROUTINES USED:

IAW
INPCH

SDS SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION:	Set I/O buffer address (IAW)
PURPOSE:	To set cell ADDR to address the last cell of the I/O buffer with an index of 2 and to complement the word count in the B register.
ACTION:	IAW sets cell ADDR with an index of 2 and an address of the last location of the I/O buffer. The contents of the B register are complemented.
PROGRAMMING TECHNIQUES:	IAW is an absolute routine assembled as part of MSCONTRL.
CALLING SEQUENCE:	Buffer location to A register Word count to B register BRM IAW
MEMORY REQUIREMENTS:	10_8 cells
SUBROUTINES USED:	None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION:	Initialize the end-of-file cards routine (INEFC)
PURPOSE:	To set the unit and channel assignments in the I/O instructions to clear the card punch.
ACTION:	INEFC calls GTUNT to obtain the unit and channel assignments which are used to initialize the I/O instructions in EFC.
PROGRAMMING TECHNIQUES:	INEFC is a logical extension of the EFC routine and is an absolute routine assembled as part of MSCONTRL.
CALLING SEQUENCE:	I/O control word to index register BRM INEFC
MEMORY REQUIREMENTS:	15_8 cells
SUBROUTINES USED:	GTUNT

SDS SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Clear the card punch (EFC)

PURPOSE: To feed two cards through the designated card punch.

ACTION: EFC calls INEFC to initialize I/O instructions; it then punches two cards.

PROGRAMMING TECHNIQUES: EFC is designed to work with the standard file processing routines and is an absolute routine assembled as part of MSCONTRL.

CALLING SEQUENCE: I/O control word to index register
BRM EFC

MEMORY REQUIREMENTS: 14_8 cells

SUBROUTINES USED: INEFC

SDS SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Extract unit and channel assignments (GTUNT)

PURPOSE: To obtain the unit and channel assignments from a standard I/O control word for use by the various I/O initialization routines.

ACTION: GTUNT extracts the unit channel and mode bits from the I/O control word in the index register and stores them in CHANL. The channel designation is right adjusted in the index register.

PROGRAMMING TECHNIQUES: GTUNT is an absolute routine assembled as part of MSCONTRL.

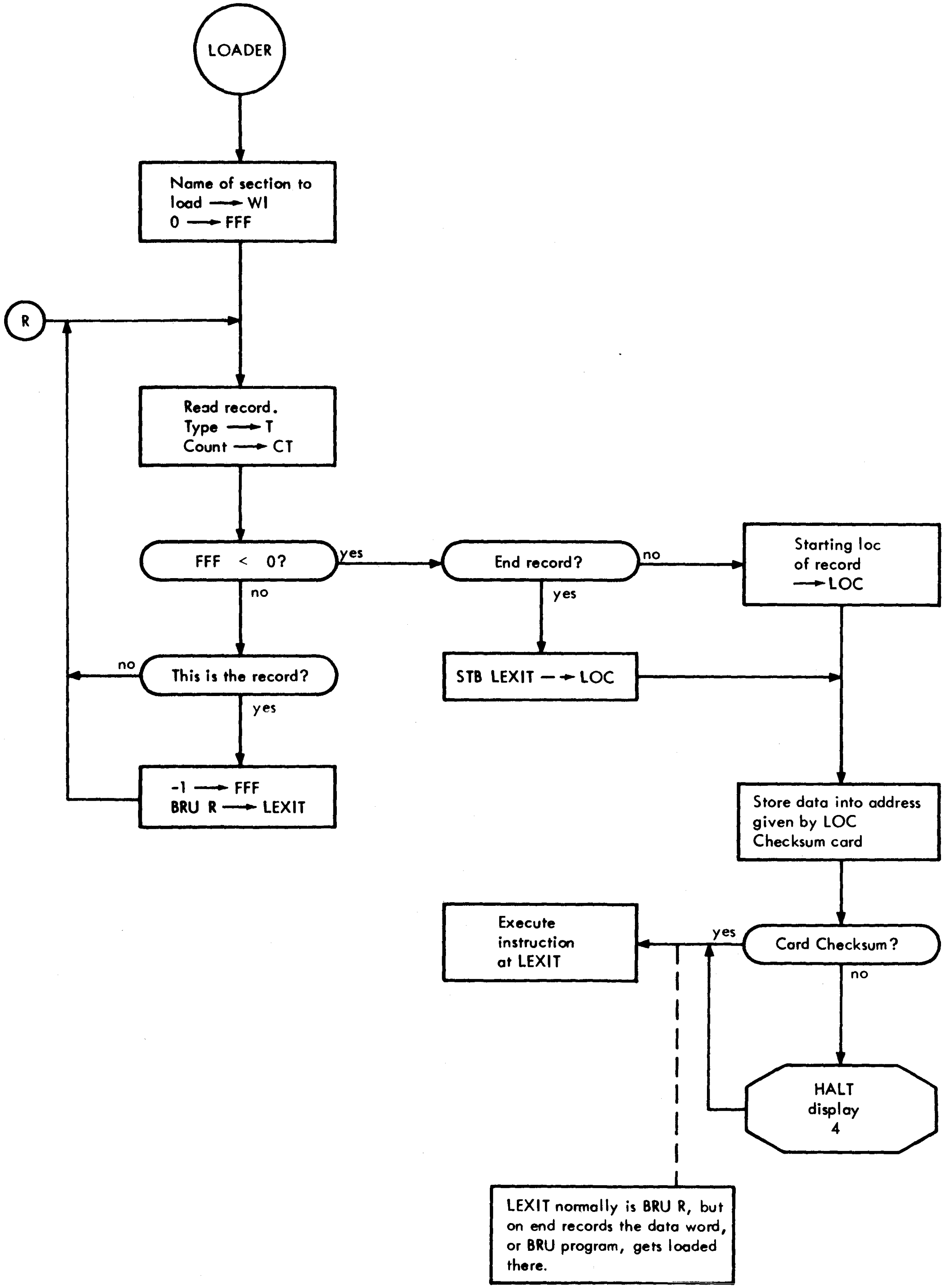
CALLING SEQUENCE:

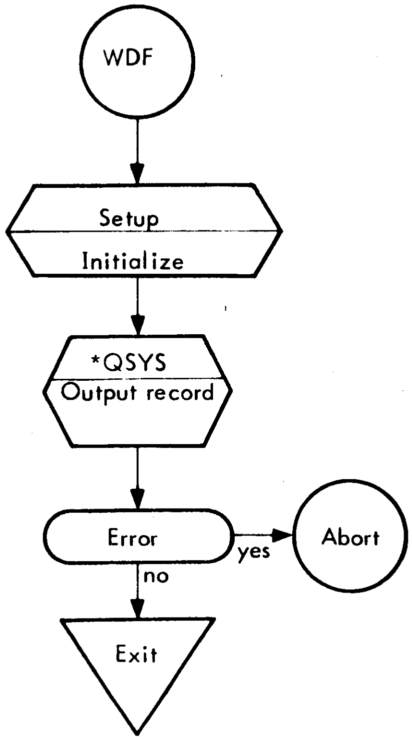
I/O control word to index register
```
BRM    GTUNT
```

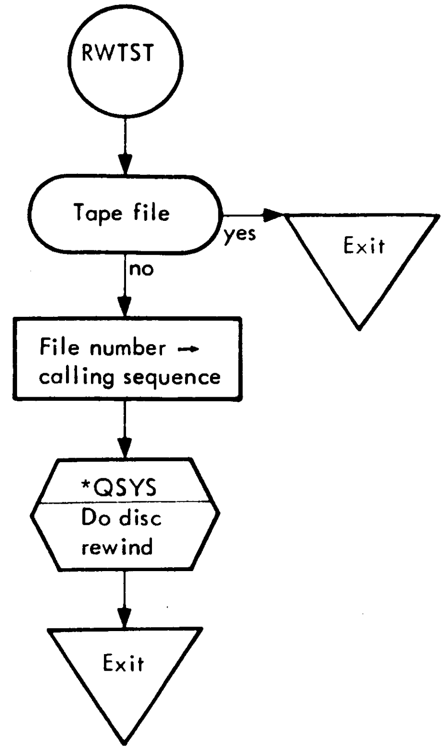
MEMORY REQUIREMENTS: 13_8 cells

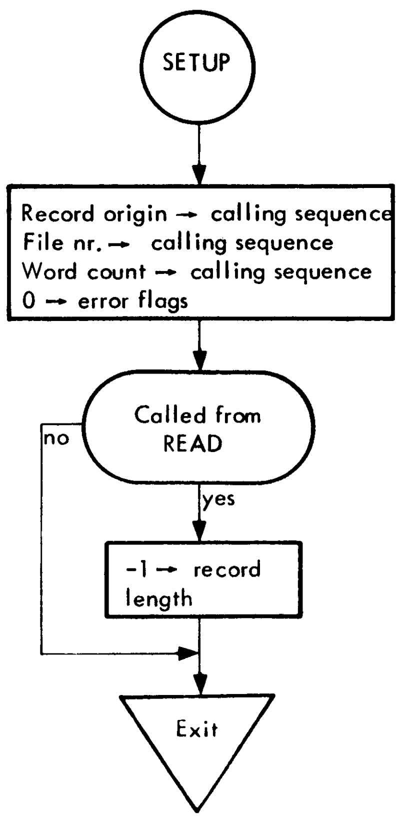
SUBROUTINES USED: None

BASIC TAPE LOADER

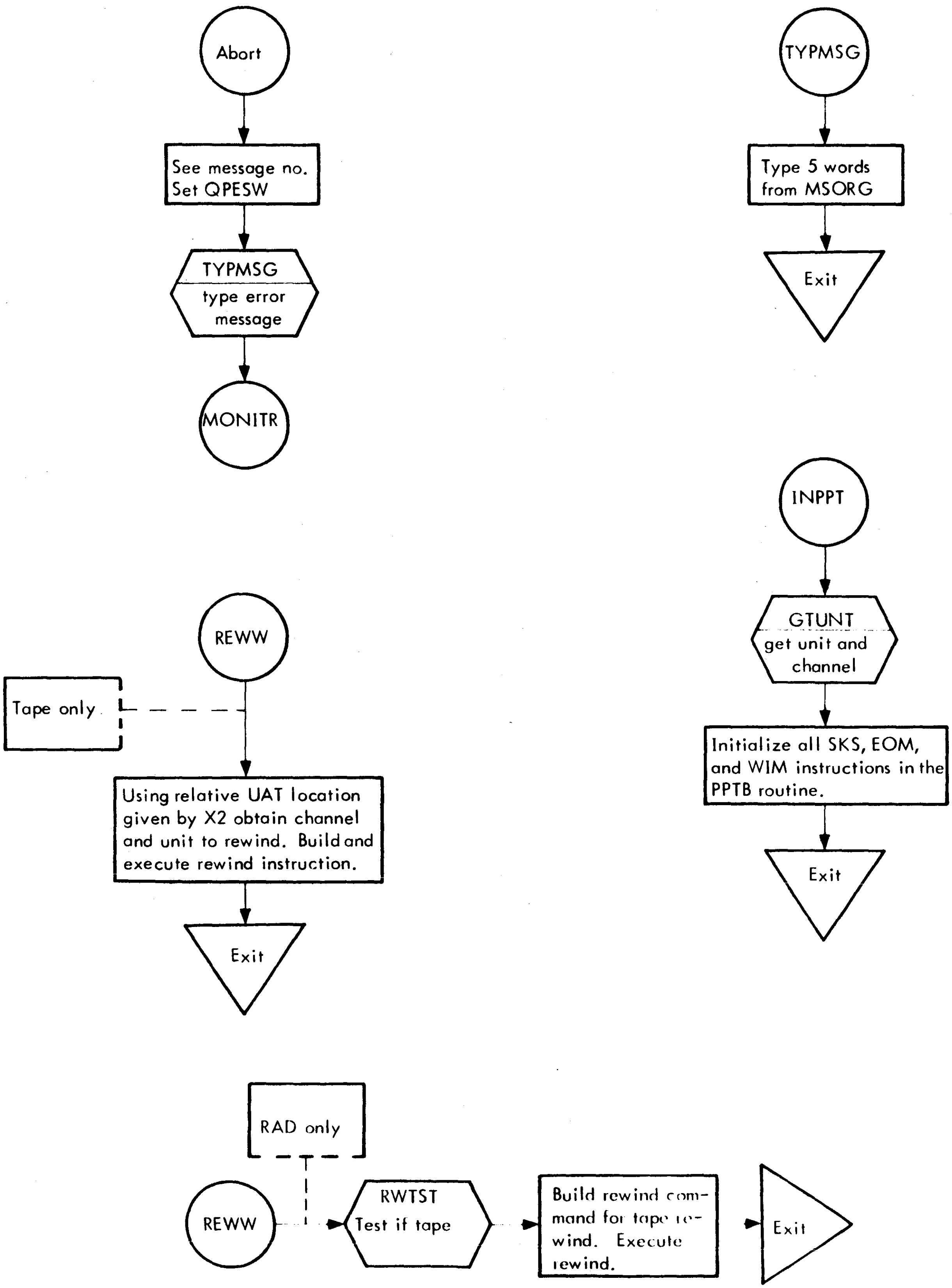



WDF
Setup
Initialize
*QSYS
Output record
Error
yes
Abort
no
Exit

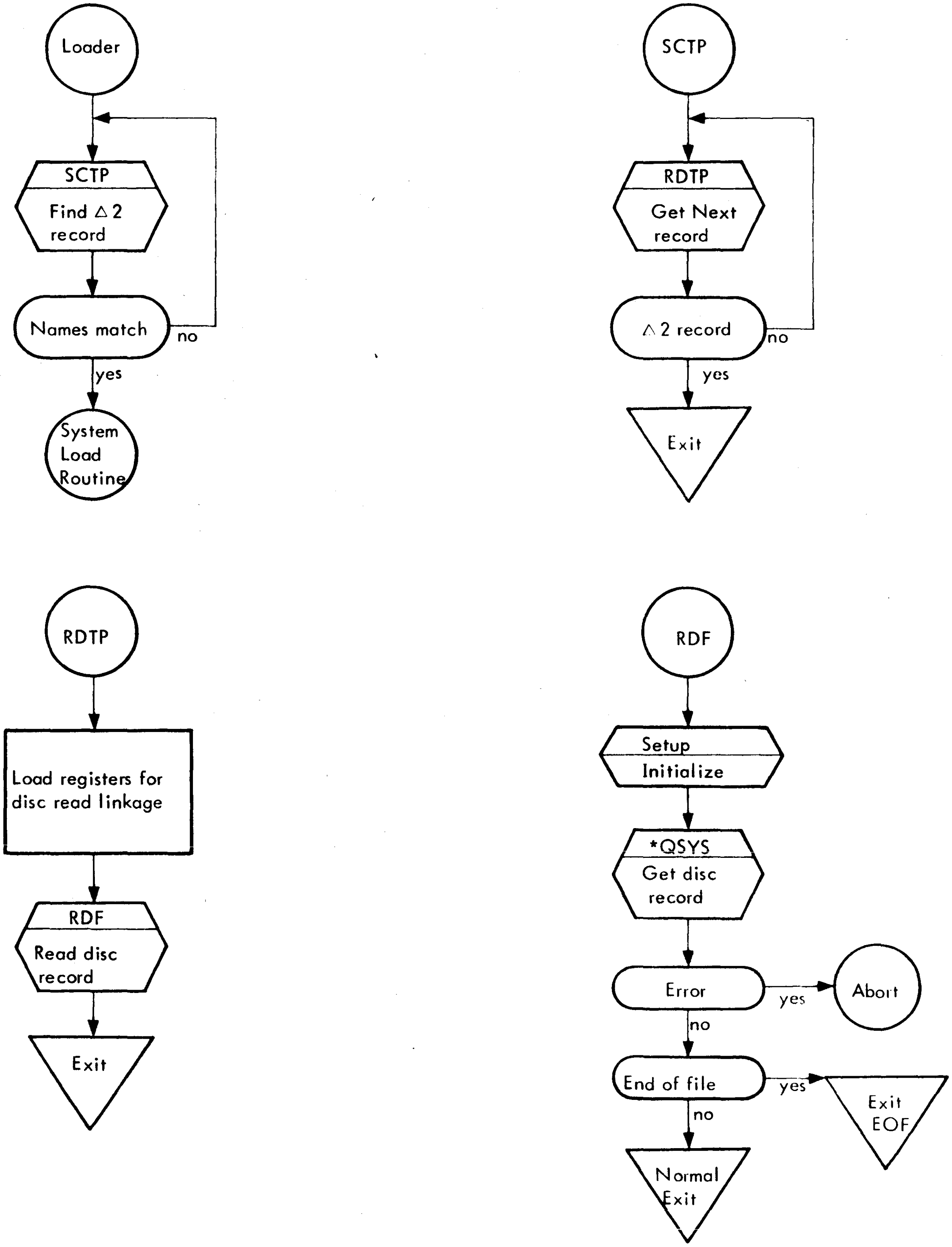

RWTST
Tape file
yes
Exit
no
File number → calling sequence
*QSYS
Do disc rewind
Exit

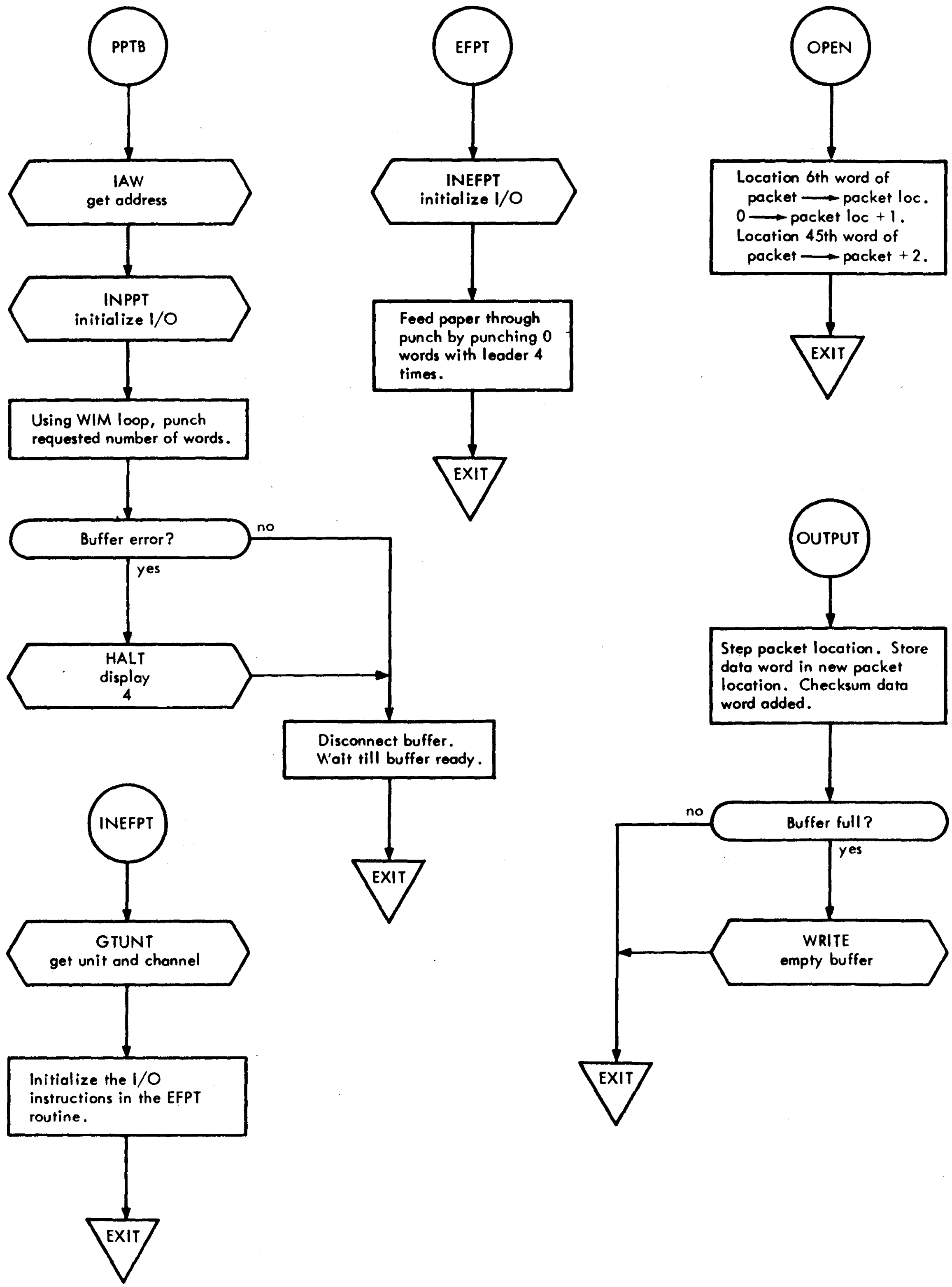

SETUP
Record origin → calling sequence
File nr. → calling sequence
Word count → calling sequence
0 → error flags
Called from READ
no
yes
-1 → record length
Exit

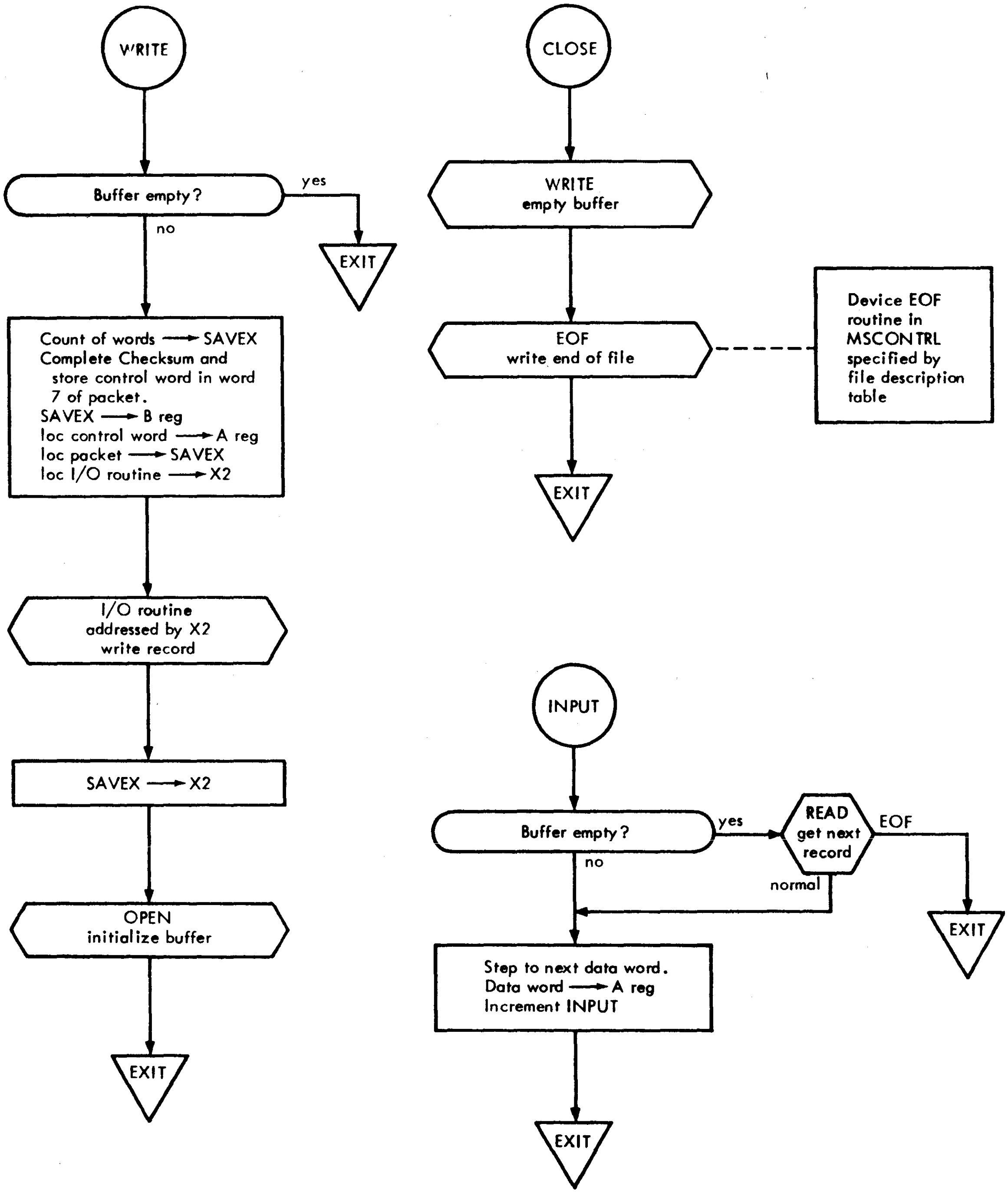
ABORT, TYPMSG, REWW AND INPUT ROUTINES

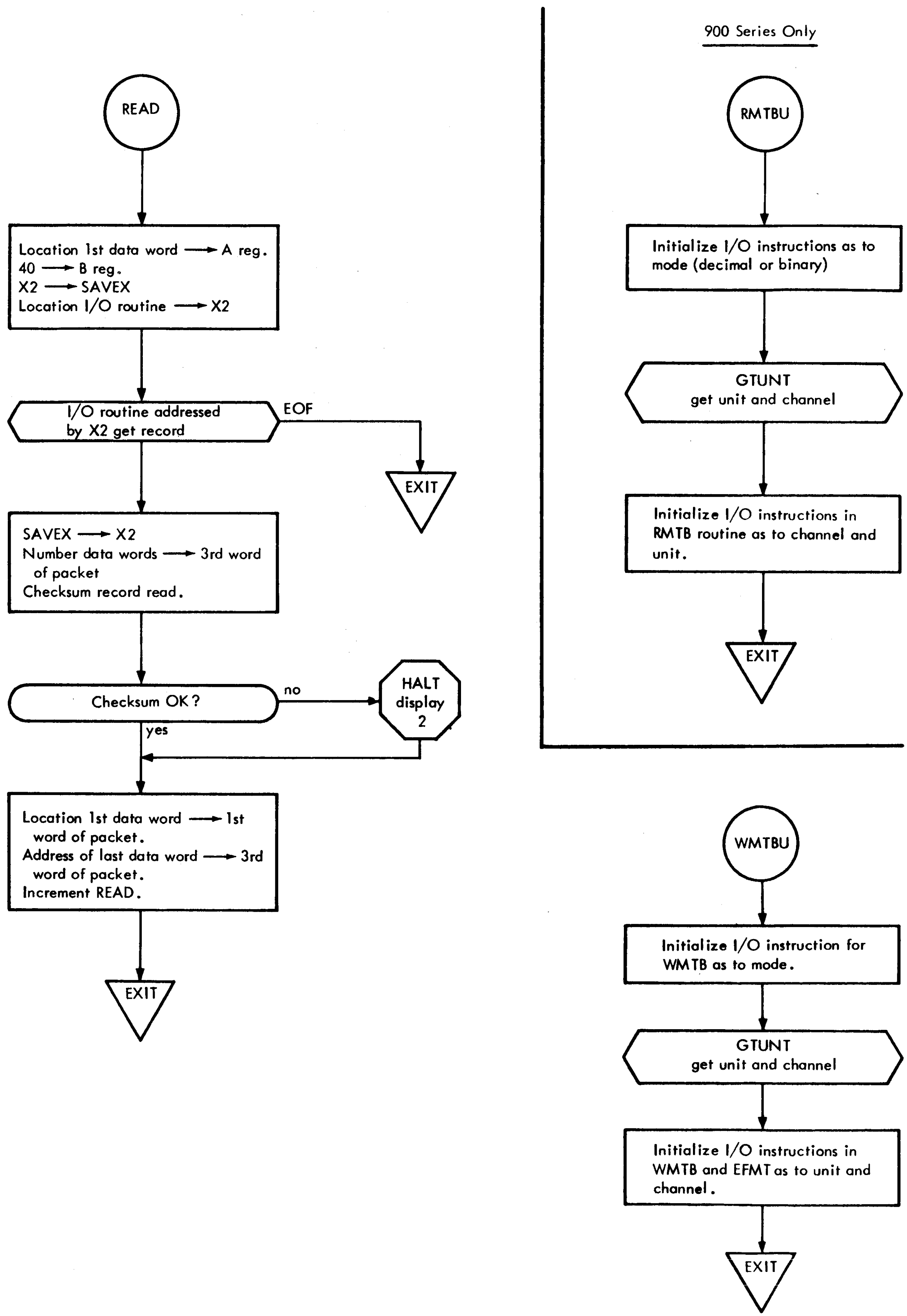


BASIC RAD LOADER

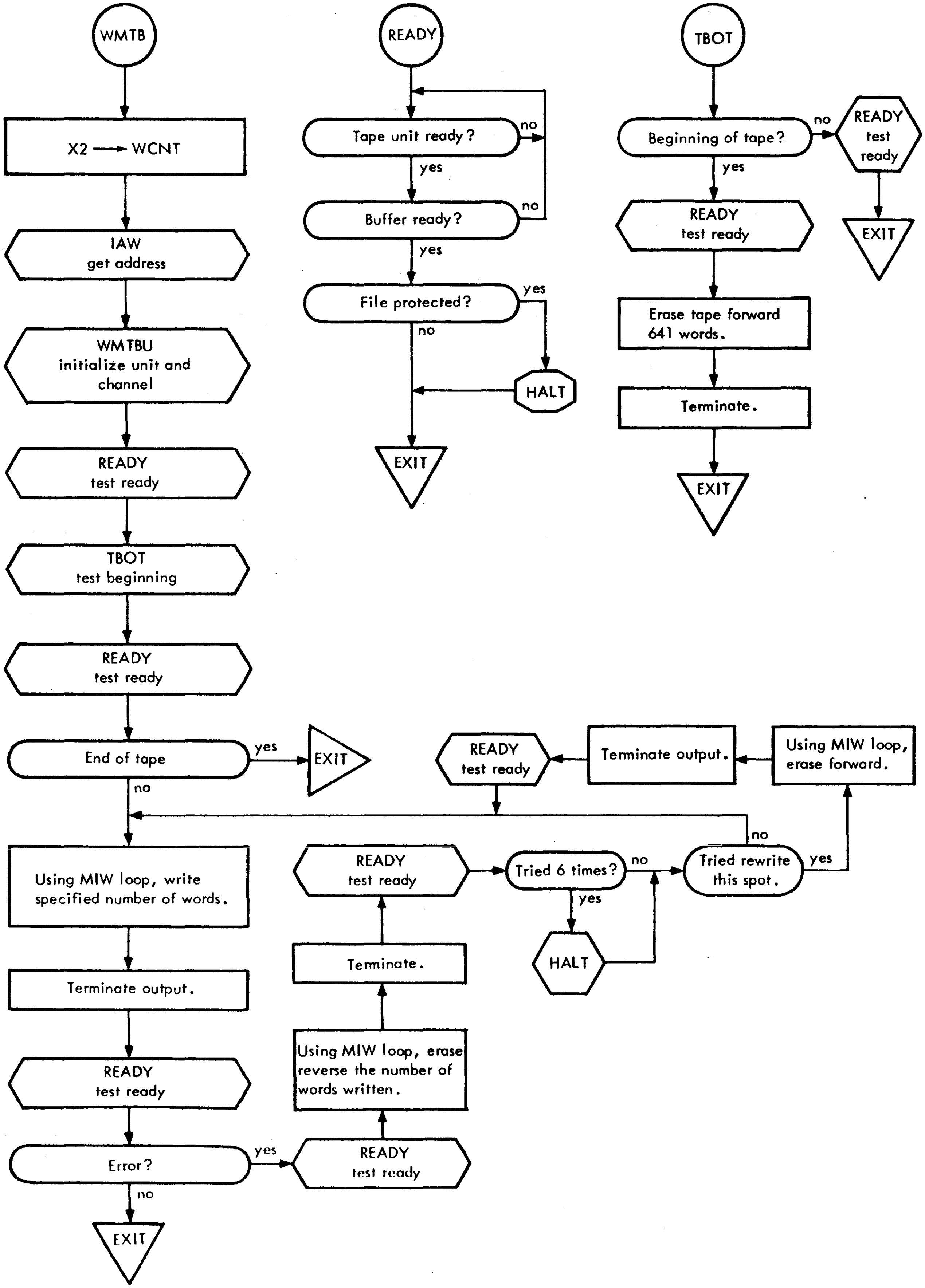


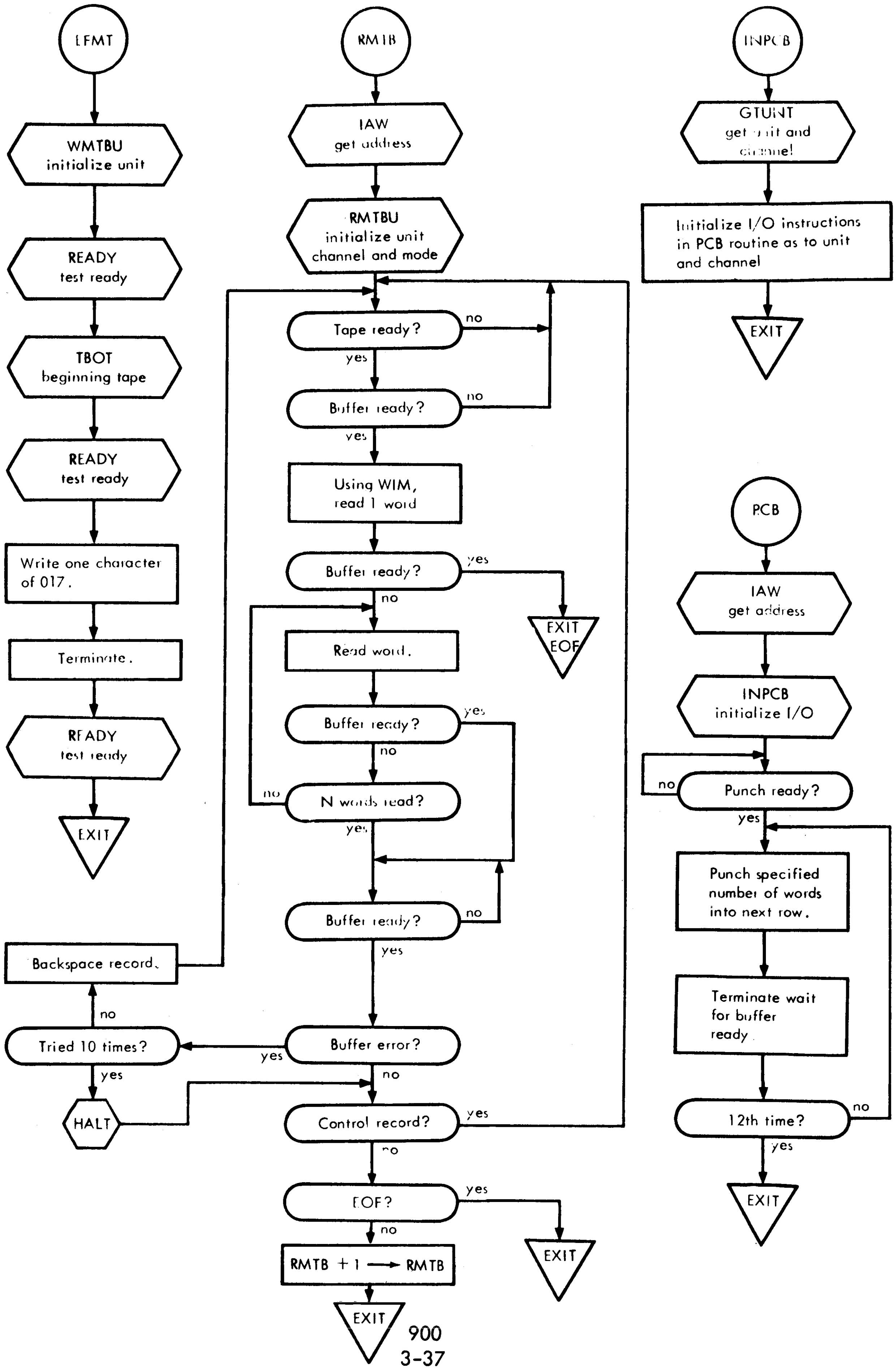

PPTB
IAW
get address
INPPT
initialize I/O
Using WIM loop, punch requested number of words.
Buffer error?
no
yes
HALT
display
4
Disconnect buffer. Wait till buffer ready.
EXIT
INEFPT
GTUNT
get unit and channel
Initialize the I/O instructions in the EFPT routine.
EXIT
EFPT
INEFPT
initialize I/O
Feed paper through punch by punching 0 words with leader 4 times.
EXIT
OPEN
Location 6th word of packet → packet loc.
0 → packet loc + 1.
Location 45th word of packet → packet + 2.
EXIT
OUTPUT
Step packet location. Store data word in new packet location. Checksum data word added.
Buffer full?
no
yes
WRITE
empty buffer
EXIT


WRITE
Buffer empty?
yes
no
EXIT
Count of words ⟶ SAVEX
Complete Checksum and store control word in word 7 of packet.
SAVEX ⟶ B reg
loc control word ⟶ A reg
loc packet ⟶ SAVEX
loc I/O routine ⟶ X2
I/O routine addressed by X2 write record
SAVEX ⟶ X2
OPEN initialize buffer
EXIT
CLOSE
WRITE empty buffer
EOF write end of file
Device EOF routine in MSCONTRL specified by file description table
EXIT
INPUT
Buffer empty?
yes
no
READ get next record
EOF
normal
EXIT
Step to next data word.
Data word ⟶ A reg
Increment INPUT
EXIT


READ
Location 1st data word → A reg.
40 → B reg.
X2 → SAVEX
Location I/O routine → X2
I/O routine addressed by X2 get record
EOF
EXIT
SAVEX → X2
Number data words → 3rd word of packet
Checksum record read.
Checksum OK?
no
yes
HALT display 2
Location 1st data word → 1st word of packet.
Address of last data word → 3rd word of packet.
Increment READ.
EXIT
900 Series Only
RMTBU
Initialize I/O instructions as to mode (decimal or binary)
GTUNT
get unit and channel
Initialize I/O instructions in RMTB routine as to channel and unit.
EXIT
WMTBU
Initialize I/O instruction for WMTB as to mode.
GTUNT
get unit and channel
Initialize I/O instructions in WMTB and EFMT as to unit and channel.
EXIT

MSCONTRL
WMTB, READY AND TBOT ROUTINES

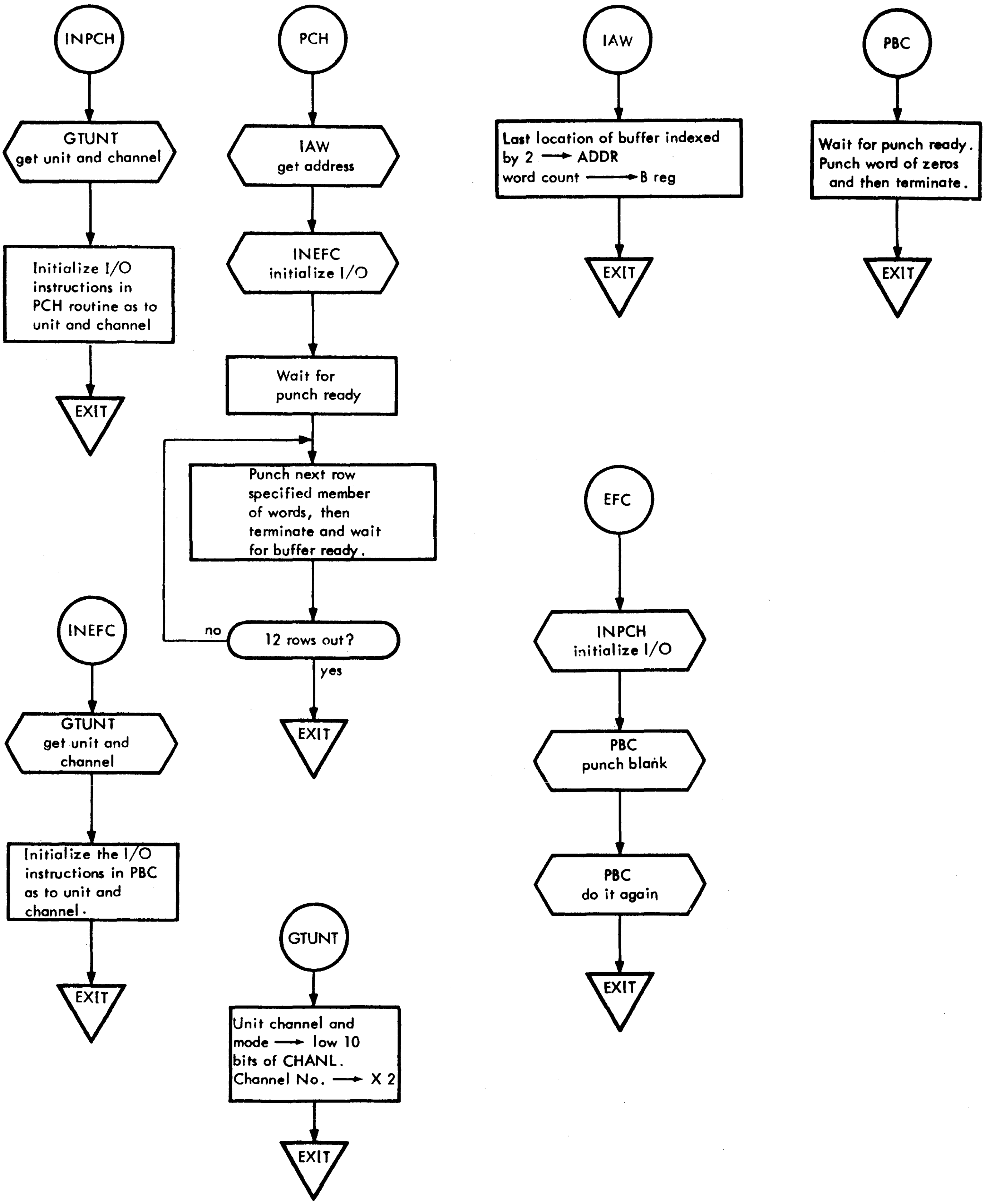



LFMT
WMTBU
initialize unit
READY
test ready
TBOT
beginning tape
READY
test ready
Write one character of 017.
Terminate.
READY
test ready
EXIT
Backspace record.
no
Tried 10 times?
yes
HALT
RMTB
IAW
get address
RMTBU
initialize unit
channel and mode
Tape ready?
no
yes
Buffer ready?
no
yes
Using WIM,
read 1 word
Buffer ready?
yes
no
EXIT
EOF
Read word.
Buffer ready?
yes
no
N words read?
no
yes
Buffer ready?
no
yes
Buffer error?
yes
no
Control record?
yes
no
EOF?
yes
no
RMTB + 1 → RMTB
EXIT
EXIT
INPCB
GTUNT
get unit and
channel
Initialize I/O instructions
in PCB routine as to unit
and channel
EXIT
PCB
IAW
get address
INPCB
initialize I/O
no
Punch ready?
yes
Punch specified
number of words
into next row.
Terminate wait
for buffer
ready
12th time?
no
yes
EXIT

b

ENCODER
S4B
MON1

MSCONTRL
INPCH, PCH, IAW, PBC, INEFC, EFC AND GTUNT ROUTINES



SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: ENCODER

PURPOSE: To encode symbolic programs or to update existing encoded programs.

ACTION: Performs the following functions:

1. Calls MON1 to set up the input/output unit and channel assignments and the input/output function requests in MSCONTRL and, if necessary, to copy symbolic corrections to scratch tape X2.

2. Reads a record of symbolic input and checks for correction (+ in column 1). If the card is a correction, or if there is encoded input only as determined by an end-of-file return from the reader, ENCODER copies the old encoded dictionary into core and builds the APO table of dictionary addresses.

3. ENCODER reads the symbolic input or the encoded text, or both if it be an update run. When correcting an encoded file, the symbolic insertions are inserted into the file by using the same TRANS routine as is used for runs with only symbolic input. Deleted encoded lines are bypassed by calling the DELETE routine, and encoded lines to be retained are passed along by calling the SKIP routine. All symbolic lines are translated to META-SYMBOL language by calling the translation program S4B.

4. As each line of input is obtained, ENCODER scans the line and collects each string of characters into a dictionary entry. (See Section 4, Item Formats.) The encoder builds four types of entries: blank strings, special characters, numeric items, and alphanumeric items. If the string is in the form of a byte of encoded record, this construction step

ACTION:
(cont.)

is eliminated since the dictionary entry is already available. If this is an encoded byte, the program tests APO to see if the new byte value has been determined; if it has not, or if this is a string from a symbolic record, SRCH is called to find the location of the CPO (balanced tree insert table) entry for the string and to obtain the byte value. If SRCH fails to find an equivalent entry in CPO, NSRT is called to enter the string into the dictionary (BPO) and insert a 3-word entry into CPO for later reference. (See Section 4, Item Formats.) The sequence number of a unique string of characters or dictionary entry is the byte value for the entry.

5. As a value for each byte is obtained, it is output on the intermediate output tape (X1). This encoding-updating process continues until an end of file is detected on the input file. Note that an END card does not terminate the encoding process. During the encoding process comments are not encoded in the manner indicated. Comments, as determined by the presence of an * in the first character of the record or by three blank fields (excluding imbedded blank strings in TEXT and BCD variable fields or in alphanumeric expressions), are output as they appear in the source language except that they are preceded by a count byte of six bits indicating the number of comment characters.

6. When an end-of-file condition is detected on the input file, control goes to the END section of ENCODER. Here a check is made to see if there was encoded input; if so, the insert table CPO is moved to the origin of APO and the dictionary is moved to a position immediately following CPO. The intermediate output tape is rewound, and the dictionary is output on the encoded output device by selecting the dictionary entry for each entry in CPO. In this manner dropped bytes caused by deleting encoded lines are purged. As each dictionary entry is output, it is moved to high memory to form a dictionary for

ACTION: (cont.)

PREASM. When all dictionary entries have been output, the encoded text records are copied from X1 to the output file.

7. When the output file has been completed, ENCODER checks the I/O function control cells in MSCONTRL to determine if additional outputs are required. If encoded output is the only output, control is returned to MONARCH; otherwise, the PREASM routine is loaded by branching to the basic tape loader routine.

CALLING SEQUENCE:

ENCODER is one of several independently assembled routines loaded as the first assembler overlay. The last of these routines in order loaded is MSCONTRL. The transfer address for MSCONTRL is to a cell containing a branch to the starting location of ENCODER (TRACOR).

PROGRAMMING CONVENTIONS:

ENCODER is assembled with an origin of 01337, which is just above the MSCONTRL program. Since ENCODER leaves the dictionary and search tables in core for PREASM, it is necessary to provide a few control words to PREASM indicating the location of tables and key words. The following control cells are left by ENCODER in the first locations following MSCONTRL.

1. DTAB. Starting location for PREASM-built dictionary if no POPs are used. (The programmed operator routines overload this cell to account for the additional length of the POP code.)

2. APO or CPO. The next available location in the balanced tree search table for entering items.

3. BPO. The next available cell in the ENCODER-built dictionary.

4. HED. A 3-word control region used in building CPO and BPO. All chain ends in CPO point to HED.

PROGRAMMING CONVENTIONS: (cont.)

5. CORG. The location minus 9 of the first word of CPO table.

6. CSEQ. The next sequence number or byte value to be defined.

ENCODER is designed to be maximally independent of machine configuration.

1. Memory size is determined by examining cell 1 of memory in which MONARCH stores the instruction BRU QBOOT. Hence maximum memory is always used.

2. MON1 sets the delay timing for the paper tape read routines in ENCODER so that in the event of inputs on paper tape proper reading will result.

3. All instructions used which are not common to all machines are either converted to alternate instructions by using procedures (e.g., CLB is LDB = 0) or by generating POP items by use of procedures.

ENCODER has a 2-condition rewind of magnetic tape X2. If corrections are used and copied to X2, ENCODER rewinds X2 in preparation of taking outputs on X2. If running with MAGPAK tapes, X2 is rewound.

MEMORY REQUIREMENTS: Variable

SUBROUTINES USED:

PTCH	OUTC	RPTB	MON1
DEC	SRCH	INCRD	OPEN†
DELETE	NSRT	CRD	REWW†
SKIP	TRAIL	CRDB	READ†
INIT	IN	CRDH	INPUT†
TRANS	OUT	INRDT	GTUNT†
STORE	MVTAB	RDPT	WRITE†
CHAR	RESET	EDC	IAW†
RCRD	TBOUT	EDS	OUTPUT†
INC	INRPT	S4B	CLOSE†

† These routines are described under MSCONTRL.

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 612001

IDENTIFICATION: ENCODER

PURPOSE: To encode symbolic programs or to update existing encoded programs.

ACTION: Performs the following functions:

1. Copies symbolic corrections to scratch tape X2, if necessary.

2. Reads a record of symbolic input and checks for correction (+ in column 1). If the card is a correction, or if there is encoded input only as determined by an end-of-file return from the reader, ENCODER copies the old encoded dictionary into core and builds the APO table of dictionary addresses.

3. ENCODER reads the symbolic input or the encoded text, or both if it be an update run. When correcting an encoded file, the symbolic insertions are inserted into the file by using the same TRANS routine as is used for runs with only symbolic input. Deleted encoded lines are bypassed by calling the DELETE routine, and encoded lines to be retained are passed along by calling the SKIP routine. All symbolic lines are translated to META-SYMBOL language by calling the translation program S4B.

4. As each line of input is obtained, ENCODER scans the line and collects each string of characters into a dictionary entry. (See Section 4, Item Formats.) The encoder builds four types of entries: blank strings, special characters, numeric items, and alphanumeric items. If the string is in the form of a byte of encoded record, this construction step is eliminated since the dictionary entry is already available. If this is an encoded byte, the program tests APO to see if the new byte value has been

ACTION: (cont.)

determined; if it has not, or if this is a string from a symbolic record, SRCH is called to find the location of the CPO (balanced tree insert table) entry for the string and to obtain the byte value. If SRCH fails to find an equivalent entry in CPO, NSRT is called to enter the string into the dictionary (BPO) and insert a 3-word entry into CPO for later reference. (See Section 4, Item Formats.) The sequence number of a unique string of characters or dictionary entry is the byte value for the entry.

5. As a value for each byte is obtained, it is output on the intermediate output tape (X1). This encoding-updating process continues until an end of file is detected on the input file. Note that an END card does not terminate the encoding process. During the encoding process comments are not encoded in the manner indicated. Comments, as determined by the presence of an * in the first character of the record or by three blank fields (excluding imbedded blank strings in TEXT and BCD variable fields or in alphanumeric expressions), are output as they appear in the source language except that they are preceded by a count byte of six bits indicating the number of comment characters.

6. When an end-of-file conditions is detected on the input file, control goes to the END section of ENCODER. Here a check is made to see if there was encoded input; if so, the insert table CPO is moved to the origin of APO and the dictionary is moved to a position immediately following CPO. The intermediate output tape is rewound, and the dictionary is output on the encoded output device by selecting the dictionary entry for each entry in CPO. In this manner dropped bytes caused by deleting encoded lines are purged. As each dictionary entry is output, it is moved to high memory to form a dictionary for PREASM. When all dictionary entries have been output, the encoded text records are copied from X1 to the output file.

ACTION:
(cont.)

7. When the output file has been completed, ENCODER checks the I/O function control cells in MSCONTRL to determine if additional outputs are required. If encoded output is the only output, control is returned to MONARCH; otherwise, the PREASM routine is loaded by branching to the basic tape loader routine.

CALLING SEQUENCE:

ENCODER is one of several independently assembled routines loaded as the first assembler overlay. The last of these routines in order loaded is MSCONTRL. The transfer address for MSCONTRL is to a cell containing a branch to the starting location of ENCODER (TRACOR).

PROGRAMMING CONVENTIONS:

Since ENCODER leaves the dictionary and search tables in core for PREASM, it is necessary to provide a few control words to PREASM indicating the location of tables and key words. The following control cells are left by ENCODER in the first locations following MSCONTRL.

1. DTAB. Starting location for PREASM-built dictionary.
2. APO or CPO. The next available location in the balanced tree search table for entering items.
3. BPO. The next available cell in the ENCODER-built dictionary.
4. HED. A 3-word control region used in building CPO and BPO. All chain ends in CPO point to HED.
5. CORG. The location minus 9 of the first word of CPO table.
6. CSEQ. The next sequence number or byte value to be defined.

ENCODER has a 2-condition rewind of magnetic tape X2. If corrections are used and copied to X2, ENCODER rewinds X2 in preparation of taking outputs on X2. If running with MAGPAK tapes, X2 is rewound.

MEMORY REQUIREMENTS: Variable

SUBROUTINES USED:

PTCH	INC	TBOUT
DEC	OUTC	S4B
DELETE	SRCH	OPEN†
SKIP	NSRT	READ†
INIT	TRAIL	INPUT†
TRANS	IN	WRITE†
STORE	OUT	OUTPUT†
CHAR	MVTAB	CLOSE†
RCRD	RESET	

†These routines are described under MSCONTRL.

ENTRY POINTS TO ENCODER SUBROUTINES

Entry	Page Description	Page Flowchart
ALL	3-49	3-81
ALL2	3-49	3-81
ALL3	3-49	3-81
ALL4	3-49	3-81
ALL5	3-49	3-81
BEGIN		3-75
BLAN1	3-49	3-79
BLANK	3-49	3-79
BLANK1	3-49	3-79
BLANK2	3-49	3-79
CHAR	3-52	3-82
CHAR1	3-52	3-82
CHAR2	3-52	3-82
CHARX	3-52	3-82
CORR		3-75
CORR1		3-76
CORR4		3-76
CORR5		3-76
CORR6		3-76
CORR8		3-76
CORR10		3-76
CORR11		3-76
CRD	3-67	3-91
CRDB	3-68	3-92
CRDH	3-69	3-92
DEC	3-45	3-77
DELETE	3-46	3-77
DOT	3-49	3-81
EDC	3-72	3-93
EDS	3-73	3-93
END		3-89
END2		3-89
EOD	3-49	3-80
EOR	3-49	3-80
EORC		3-79
IN	3-59	3-88
IN1B	3-59	3-88
INAB4		3-75
INC	3-54	3-83
INCRD	3-66	3-91
INIT	3-48	3-78
INRDT	3-70	3-92
INRPT	3-64	3-91
MVTAB	3-61	3-90
NS3	3-57	3-86
NS4	3-57	3-86
NS4A	3-57	3-87
NS4B	3-57	3-87
NS5	3-57	3-87
NS6	3-57	3-87
NS7	3-57	3-87
NS8	3-57	3-86
NS9	3-57	3-87
NS10	3-57	3-87
NSRT	3-57	3-86
NU	3-49	3-81
OUT	3-60	3-88
OUTC	3-55	3-83
PROG		3-76
PTCH	3-44	3-77
RCRD	3-53	3-83
RDPT	3-71	3-92
RESET	3-62	3-90
RPTB	3-65	3-91
SKIP	3-47	3-78
SR1	3-58	3-84
SRCH	3-56	3-84
STORE	3-51	3-82
TBOUT	3-63	3-90
TRACOR	3-39	3-74
TRAIL	3-58	3-88
TRAN	3-49	3-79
TRAN1	3-49	3-79
TRANS	3-49	3-79

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION:	Get input character subroutine (PTCH)
PURPOSE:	To get next character of encoded input dictionary.
ACTION:	Extracts next character from TEMP and stores the remaining characters in TEMP. If a new word is needed as determined by the character count, B, PTCH calls INPUT to obtain next word of dictionary. An end-of-file return from input results in an abort message of '03'.
PROGRAMMING CONVENTIONS:	PTCH is a relocatable routine contained in ENCODER.
CALLING SEQUENCE:	B register should be set to zero BRM PTCH on initial call for each dictionary entry
MEMORY REQUIREMENTS:	24_8 cells
SUBROUTINES USED:	INPUT

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042C
Catalog No. 9300: 612C

IDENTIFICATION: Compute correction number routine (DEC)

PURPOSE: To compute correction numbers for ENCODER symbolic correction logic.

ACTION: Computes correction numbers by successive multiplication. Leaves resulting number in WORD.

PROGRAMMING CONVENTIONS: First character of correction number is in A register on entry. Obtains characters by calling CHAR until end of record or non-numeric digit is obtained. If first character of corrections is +, the plus is ignored. DEC is a relocatable routine assembled as part of ENCODER.

CALLING SEQUENCE:

First character of number to A register
BRM DEC
Result left in WORD

MEMORY REQUIREMENTS: 21_8 cells

SUBROUTINES USED: CHAR

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION:	Delete line of encoded input (DELETE)
PURPOSE:	To delete lines of encoded input when updating encoded files.
ACTION:	Gets input characters by calling IN until end of line is reached; then calls INC until all comments have been passed.
PROGRAMMING CONVENTIONS:	DELETE is a relocatable routine assembled as part of ENCODER.
CALLING SEQUENCE:	BRM DELETE
MEMORY REQUIREMENTS:	13_8 cells
SUBROUTINES USED:	IN INC

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 0420
Catalog No. 9300: 6120

IDENTIFICATION:	Routine to save lines of encoded input (SKIP)
PURPOSE:	To transcribe bytes of encoded input file by calling IN. Each byte is translated to the correct output value by either obtaining the value from the APO table entry for the byte or by using SRCH and NSRT to obtain the value. Bytes are output by calling OUT. Comments are copied by using INC and OUTC.
PROGRAMMING CONVENTIONS:	SKIP is a relocatable routine assembled as part of ENCODER.
CALLING SEQUENCE:	BRM SKIP
MEMORY REQUIREMENTS:	45_8 cells
SUBROUTINES USED:	IN OUT INC OUTC NSRT SRCH

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION:	Table initialization routine (INIT)
PURPOSE:	To initialize cells for ENCODER, SRCH, and NSRT routines.
ACTION:	INIT initializes CORG, HED, and CSEQ.
PROGRAMMING CONVENTIONS:	INIT is a relocatable routine assembled as part of ENCODER.
CALLING SEQUENCE:	BRM INIT
MEMORY REQUIREMENTS:	10_8 cells
SUBROUTINES USED:	None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 0420
Catalog No. 9300: 6120

IDENTIFICATION: Symbolic translation routine (TRANS)

PURPOSE: To convert symbolic lines of code by generating a dictionary entry for each character string within the line and calling SRCH/NSRT to define the entry. The resulting byte value is output to the intermediate output tape (X1) by calling OUT. Comment characters are counted and output as a count followed by the character string.

ACTION: TRANS obtains input symbolic characters by calling the CHAR routine. Character strings are constructed, the initial type being determined by executing a 64-place transfer table T1. As each string or dictionary entry is constructed, it is defined by calling SRCH/NSRT and the resulting byte value is output on X1 by calling OUT. Blank fields are counted and the third blank field or end of symbolic record terminates the line. Blank fields within alphanumeric data strings are not used as terminators. If a comma appears as the terminal non-blank character of a line, the line is interpreted as a continuation. Trailing blanks on the current card plus leading blanks on the following card are treated as a single blank string, and the following card is taken as part of the current record without an end-of-line mark between.

PROGRAMMING CONVENTIONS: TRANS uses transfer table T1 to determine string types by loading the index with the initial character and branching indirectly to T1 modified by the index. TRANS is a relocatable routine assembled as part of ENCODER.

CALLING SEQUENCE: BRM TRANS

MEMORY REQUIREMENTS:	303_8 cells
SUBROUTINES USED:	CHAR NSRT
	OUT SRCH
	OUTC STORE

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 0420
Catalog No. 9300: 6120

IDENTIFICATION: Subroutine to store characters into dictionary entry (STORE)

PURPOSE: To insert characters into a dictionary entry being constructed.

ACTION: STORE positions characters to the next available cell addressed by WORD and merges the characters into the location specified by WORD by adding to memory.

PROGRAMMING CONVENTIONS: Before the initial call for a dictionary entry, cells SHIFT, WORD, and the cell addressed by WORD must be initialized. STORE is a relocatable routine assembled as part of ENCODER.

CALLING SEQUENCE:

Character to A register
BRM   STORE

MEMORY REQUIREMENTS: 20_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Fetch a symbolic input character (CHAR)

PURPOSE: To get the next character of input from the symbolic input file.

ACTION: CHAR extracts the next input character from P2 into the low order six bits of the A register. When the input word P2 is empty, the next word is taken from the input buffer CARD. When CARD is empty, the next record is obtained by calling RCRD. If the end-of-file flag is set, CHAR terminates the encoding operation by exiting from the TRANS routine. On EOF returns from RCRD, CHAR sets the end-of-file flag. After reading a record, CHAR exits with an end-of-record character in the A register.

PROGRAMMING CONVENTIONS: RCRD and CHAR work together since the number of input characters is set by RCRD. CHAR is a relocatable routine assembled as part of ENCODER.

CALLING SEQUENCE: BRM CHAR

MEMORY REQUIREMENTS: 35_8 cells

SUBROUTINES USED: RCRD

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 04201
Catalog No. 9300: 6120C

IDENTIFICATION: Read input symbolic records (RCRD)

PURPOSE: To read symbolic records.

ACTION: On entry, RCRD saves the character count for the current line (P7) in cell P8. The next input record is read by calling the proper routine to read symbolic input as indicated by HOLP. An end-of-file return from the read results in an end-of-file exit from RCRD. S4B is called to perform any language translation needed on the input record. The number of terminal blank characters in the record is set in P5. The characters remaining in current word count (P) are cleared, and P1 is set to -19 for indexing the input buffer by CHAR. The number of characters to the first blank of a terminal blank string is set in P7.

PROGRAMMING CONVENTIONS: RCRD uses the input I/O routine established by MON1 as determined from the UAT. RCRD is a relocatable routine assembled as part of ENCODER.

CALLING SEQUENCE:

```
BRM   RCRD
end-of-file exit
normal exit
```

MEMORY REQUIREMENTS: 50_8 cells

SUBROUTINES USED:
S4B
I/O routine needed to read symbolic input

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION:	Get comment characters from encoded input file (INC)
PURPOSE:	Get next comment character from encoded input file.
ACTION:	INC sets the input byte size to six bits, calls IN to get the next byte into the A register, and then resets the byte size to its initial value.
PROGRAMMING CONVENTIONS:	INC is a relocatable routine assembled as part of the ENCODER.
CALLING SEQUENCE:	BRM INC
MEMORY REQUIREMENTS:	12_8 cells
SUBROUTINES USED:	IN

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 900 Series: 0420
9300: 6120(

IDENTIFICATION: Output comment characters (OUTC)

PURPOSE: To output comment characters to the encoded output file.

ACTION: OUTC sets the output byte size to six bits, calls OUT to output the character in the A register, and then resets the output byte size.

PROGRAMMING CONVENTIONS: OUTC is a relocatable routine assembled as part of ENCODER.

CALLING SEQUENCE:

```
BRM    OUTC
Output character to A register
```

MEMORY REQUIREMENTS: 11_8 cells

SUBROUTINES USED: OUT

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Search balanced tree table (SRCH)

PURPOSE To search the balanced tree table of bytes, CPO, for a given dictionary entry.

ACTION: SRCH steps through CPO starting at the loaction given in HED+1, looking for an item identical to the input item. See Section 3, Item Formats for an illustration of CPO and dictionary entries. When an identical item is found in CPO, SRCH exits with the sequence number of the dictionary entry in the A register (byte value). Successful search results in a return to the calling location plus 2; an unsuccessful search results in a return to the calling location plus 1. SRCH sets cell U to the last point of imbalance in the path searched and M0 to the last point examined by search. In addition SRCH sets the direction pointer in each CPO item examined to indicate the path taken from that point.

PROGRAMMING CONVENTIONS: SRCH is a relocatable routine assembled as part of ENCODER.

CALLING SEQUENCE:

Set HED to location of dictionary item being searched for
BRM SRCH
Item not found exit
Item found exit

MEMORY REQUIREMENTS: 621_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 04201
Catalog No. 9300: 6120C

IDENTIFICATION: Insert entries in dictionary, BPO, and search table, CPO (NSRT)

PURPOSE: To define unique dictionary entries representing unique character strings of input by inserting a dictionary item in BPO and a corresponding balanced tree search table entry in CPO, and to maintain the balance of CPO.

ACTION: NSRT enters the dictionary item into BPO and a 3-word item into CPO. The CPO entry is inserted such that the first word addresses the dictionary entry in BPO, the second word addresses the item that is just less than the current entry, and the third word addresses the item just larger than the current entry. If the addition of the current item results in a tree that is out of balance, (a tree such that from that point the longest path on one side is more than one item longer than the longest path on the alternate side), NSRT rebalances the tree by adjusting the lesser and greater linkages within the tree from the last point of imbalance. Upon exit the value of the byte inserted is in the A register.

PROGRAMMING CONVENTIONS: NSRT is a relocatable routine assembled as part of ENCODER. NSRT depends upon SRCH having been called to search for the item being inserted prior to entering NSRT.

CALLING SEQUENCE: BRM NSRT

MEMORY REQUIREMENTS: 266_8 cells

SUBROUTINES USED: TRAIL

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Determine path taken by SRCH (TRAIL)

PURPOSE: To determine the location of the item following a given CPO item on the path taken by search.

ACTION: TRAIL sets cell LINK with the location of the item following a CPO entry, indicated by X2 on entry, on the path taken by SRCH. Cell LINK+1 is set with the location of the item following the item given by X2 on the alternate path.

PROGRAMMING CONVENTION: TRAIL is a relocatable routine assembled as part of ENCODER.

CALLING SEQUENCE:

```
BRM    TRAIL
```
Location of CPO entry to index register

MEMORY REQUIREMENTS: 13_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Obtain one byte of encoded input (IN)

PURPOSE: To obtain in the A register the next byte of encoded text input.

ACTION: IN extracts the next INBYTE bits of encoded text from cell INCELL. If INCELL does not contain at least INBYTE bits, IN calls INPUT to obtain the next word of encoded text. An end-of-file return from INPUT is considered a catastrophic error, and results in an abort message '03'. The remaining bits of text in INCELL are retained in INCELL, and INBIT is set to reflect the number of data bits remaining in INCELL. If a byte is zero, it is converted to 2^{INBYTE} and INBYTE is incremented by 1. Upon exit the byte is in the A register.

PROGRAMMING CONVENTIONS: IN is a relocatable program assembled as part of ENCODER.

CALLING SEQUENCE: BRM IN

MEMORY REQUIREMENTS: 46_8 cells

SUBROUTINES USED: INPUT

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Output one byte of encoded text (OUT)

PURPOSE: To output a byte of encoded text located in the A register to the intermediate output file (X1).

ACTION: OUT positions the byte and merges it into the location CELL. If the byte is larger than BYTE bits, BYTE is incremented. When 24 bits of bytes have been placed in CELL, OUT calls OUTPUT to write the contents of CELL on the intermediate output file, X1. To reflect the number of bits of data stored in CELL, BIT is reset each time OUT is called.

PROGRAMMING CONVENTIONS: OUT is a relocatable routine assembled as part of ENCODER.

CALLING SEQUENCE:
Byte to A register
BRM OUT

MEMORY REQUIREMENTS: 34_8 cells

SUBROUTINES USED: OUTPUT

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 0420
Catalog No. 9300: 6120

IDENTIFICATION: Move tables BPO and CPO (MVTAB)

PURPOSE: To move the CPO and BPO tables to a lower location in memory so that the dictionary (BPO) may be reinserted purging bytes lost because of lines being deleted from encoded input.

ACTION: MVTAB moves CPO to the starting location of APO and adjusts the CPO table pointers to reflect the amount of relocation. BPO is then moved to the first locations following CPO. The location of CPO is set in CORG, and the amount of displacement in each table is recorded.

PROGRAMMING CONVENTIONS: MVTAB is a relocatable routine assembled as part of ENCODER.

CALLING SEQUENCE: BRM MVTAB

MEMORY REQUIREMENTS: 60_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 900 Series: 042016
9300: 612001

IDENTIFICATION: Relocate dictionary entries into high core (RESET)

PURPOSE: To store dictionary entries remaining after an update run into the BPO in high core and to alter the CPO pointers to the dictionary items to reflect the move.

ACTION: RESET stores the next location for BPO into the CPO table entry given by the index X2. Then RESET moves the dictionary item from the location indicated by TEMP to the next available location for BPO. The number of words to move, less 1, is given by COUNT. If the location of the item as indicated by TEMP is greater than the next available location for BPO, RESET aborts with an '02' message.

PROGRAMMING CONVENTION: RESET is a relocatable routine assembled as part of ENCODER.

CALLING SEQUENCE:

BRM RESET
Word count -1 to COUNT
Location of dictionary item to TEMP
Location of CPO entry for byte to X2

MEMORY REQUIREMENTS: 24_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 900 Series: 04201
9300: 6120C

IDENTIFICATION: Output encoded dictionary items to the encoded output file (TBOUT)

PURPOSE: To output the number of dictionary characters given by the A register to the encoded output file.

ACTION: TBOUT first tests to see if encoded output is requested and exits if it is not. If an encoded output file is requested, TBOUT obtains the output characters from the location addressed by TEMP and packs them into cell DATA until DATA contains four characters as indicated by the count A. At this time TBOUT calls OUTPUT to write the dictionary word on the encoded output file. When the number of characters to output has been depleted, TBOUT exits.

PROGRAMMING CONVENTIONS: TBOUT is a relocatable routine assembled as part of ENCODER.

CALLING SEQUENCE:

BRM TBOUT
Location of dictionary entry to TEMP
Number of characters to output to A register

MEMORY REQUIREMENTS: 30_8 cells

SUBROUTINES USED: OUTPUT

SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Initialize RPTB routine (INRPT)

PURPOSE: To initialize unit and channel assignments in the read paper tape binary routine, RPTB.

ACTION: INRPT obtains the unit and channel assignments for the device by calling GTUNT. The I/O instructions within RPTB are then set using the unit and channel assignments available.

PROGRAMMING CONVENTIONS: INRPT works as an integral part of RPTB using the unit and channel assignments from UAT as reflected in the I/O control words within MSCONTRL. INRPT is a relocatable routine assembled as part of ENCODER.

CALLING SEQUENCE: BRM INRPT

NOTE: I/O routines used by the META-SYMBOL assembly system have in general special requirements on the contents of the A, B, and X registers; for an explanation of the contents of these registers see MSCONTRL.

MEMORY REQUIREMENTS: 22_8 cells

SUBROUTINES USED: GTUNT

SDS SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Read binary paper tape (RPTB)

PURPOSE: To read into the indicated address the number of words specified (or one record) of encoded input from paper tape.

ACTION: RPTB uses a WIM loop to read from paper tape the specified number of words, or to an end of record, in four character-per-word binary format. IAW and INRPT are called to initialize the buffer address and unit and channel assignments.

PROGRAMMING CONVENTIONS: RPTB is coded to work with the file maintenance programs in MSCONTRL. RPTB is a relocatable binary routine assembled as part of ENCODER. A buffer error results in a HALT displaying '10'. Stepping causes the next record to be read.

CALLING SEQUENCE:

BRM RDPT
Number of words to B register
Location of buffer to A register
Not used
Normal return

MEMORY REQUIREMENTS: 22_8 cells

SUBROUTINES USED:

INRPT
IAW

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION:	Initialize card read routine (INCRD)
PURPOSE:	To initialize the card read routine, CRD, as to unit and channel.
ACTION:	INCRD initializes the I/O instructions in CRD by setting the correct unit and channel bits for each I/O instruction.
PROGRAMMING CONVENTION:	INCRD is a logical extension of the CRD routine and depends on the GTUNT routine having been called to obtain the proper unit and channel assignments. INCRD is a relocatable routine assembled as part of ENCODER.
CALLING SEQUENCE:	BRM INCRD
MEMORY REQUIREMENTS:	23_8 cells
SUBROUTINES USED:	None

SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Card read routine (CRD)

PURPOSE: To read the specified number of words from the next card in the card reader specified into the buffer specified and in the mode specified.

ACTION: CRD calls IAW to initialize the buffer address and INCRD to initialize itself as to unit and channel. CRD then reads the number of words requested into the buffer requested by executing the EOM following the BRM to CRD and entering a WIM loop. A buffer error results in a HALT displaying NOP1. Stepping to the next instruction results in the next card being read.

PROGRAMMING CONVENTIONS: CRD is coded to work with the binary and Hollerith card read routines CRDB and CRDH. CRD is a relocatable routine assembled as part of ENCODER.

CALLING SEQUENCE:

Buffer location to ASV
BRM CRD
Word count to IN1
EOM instruction
End-of-file exit
Normal exit

MEMORY REQUIREMENTS: 35_8 cells

SUBROUTINES USED:

IAW
INCRD

SDS SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Read binary cards (CRDB or CRDN)

PURPOSE: To read cards in the binary mode.

ACTION: CRDB gets the unit and channel by calling GTUNT. An EOM instruction is then initialized for CRD. CRD is called to read the card.

PROGRAMMING TECHNIQUES: CRDB is coded to be used with the file control routines in MSCONTRL. CRDB is a relocatable routine assembled as part of ENCODER.

CALLING SEQUENCE:

Number of words to B register
Location of buffer to A register
BRM CRDB
End-of-file return
Normal return

MEMORY REQUIREMENTS: 14_8 cells

SUBROUTINES USED: CRD

SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Read Hollerith cards (CRDH)

PURPOSE: To read cards in the Hollerith mode.

ACTION: CRDH calls GTUNT to obtain the unit and channel assignments; it then initializes an EOM instruction for the CRD routine. The CRD routine is called to read the card. The first word is tested for ΔEOF indicating end of file.

PROGRAMMING TECHNIQUES: CRDH is designed to work with the file control routines in MSCONTRL. CRDH is a relocatable routine assembled as part of ENCODER.

CALLING SEQUENCE:

Word count to B register
Buffer location to A register
BRM CRDH
End-of-file return
Normal return

MEMORY REQUIREMENTS: 20_8 cells

SUBROUTINES USED: CRD

SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Initialize the RDPT routine (INRDT)

PURPOSE: To initialize with respect to unit and channel the I/O instructions used in the RDPT routine.

ACTION: INRDT calls GTUNT to obtain the unit and channel assignments for the read. INRDT then sets the I/O instructions in RDPT to reflect these assignments.

PROGRAMMING TECHNIQUES: INRDT is a logical extension of the RDPT routine and is a relocatable routine assembled as part of ENCODER.

CALLING SEQUENCE: BRM INRDT

MEMORY REQUIREMENTS: 20_8 cells

SUBROUTINES USED: GTUNT

SDS SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Read paper tape and typewriter (RDPT)

PURPOSE: To read binary-coded decimal records from paper tape or from the typewriter.

ACTION: RDPT calls INRDT to initialize I/O instructions. Characters are then read into memory, one at a time using a WIM instruction. Tabulation characters are converted to blank strings, typewriter blanks (012) are converted to blanks (060), and carriage return characters are interpreted as end-of-record marks. Up to 80 characters per record are read. ΔEOF in the first word of input is taken as end of file.

PROGRAMMING TECHNIQUES: RDPT is designed to work with the file control routines in MSCONTRL. RDPT is a relocatable routine assembled as part of ENCODER.

CALLING SEQUENCE:

```
BRM   RDPT
End-of-file return
Normal return
```

MEMORY REQUIREMENTS: 73_8 cells

SUBROUTINES USED:

EDC
EDS
INRDT

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Store characters into buffer (EDC)

PURPOSE: To store a character into the buffer location specified.

ACTION: EDC subtracts 60_8 from the character furnished in the A register, positions it to the correct character position as determined by EDC1, and stores it into the location addressed by EDWW by adding to memory.

PROGRAMMING TECHNIQUES: EDC assumes the buffer has been cleared to blanks (60_8) prior to being called. EDC is a relocatable routine assembled as part of ENCODER.

CALLING SEQUENCE:
BRM EDC
Character to A register

MEMORY REQUIREMENTS: 21_8 cells

SUBROUTINES USED: None

SDS SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Initialize word and character positions to store characters (EDS)

PURPOSE: To set parameters EDC1 and EDWW for EDC routine.

ACTION: EDS uses the control word supplied in the A register to set the shift parameter, EDC1, and the buffer location, EDWW, for storing characters. The control word has the following format:

character (9 bits)	word position (15 bits)
0 — 8	9 — 23

Character is 0 through 3 giving character positions from left to right to store next character.

Word position is the address in buffer to store next character.

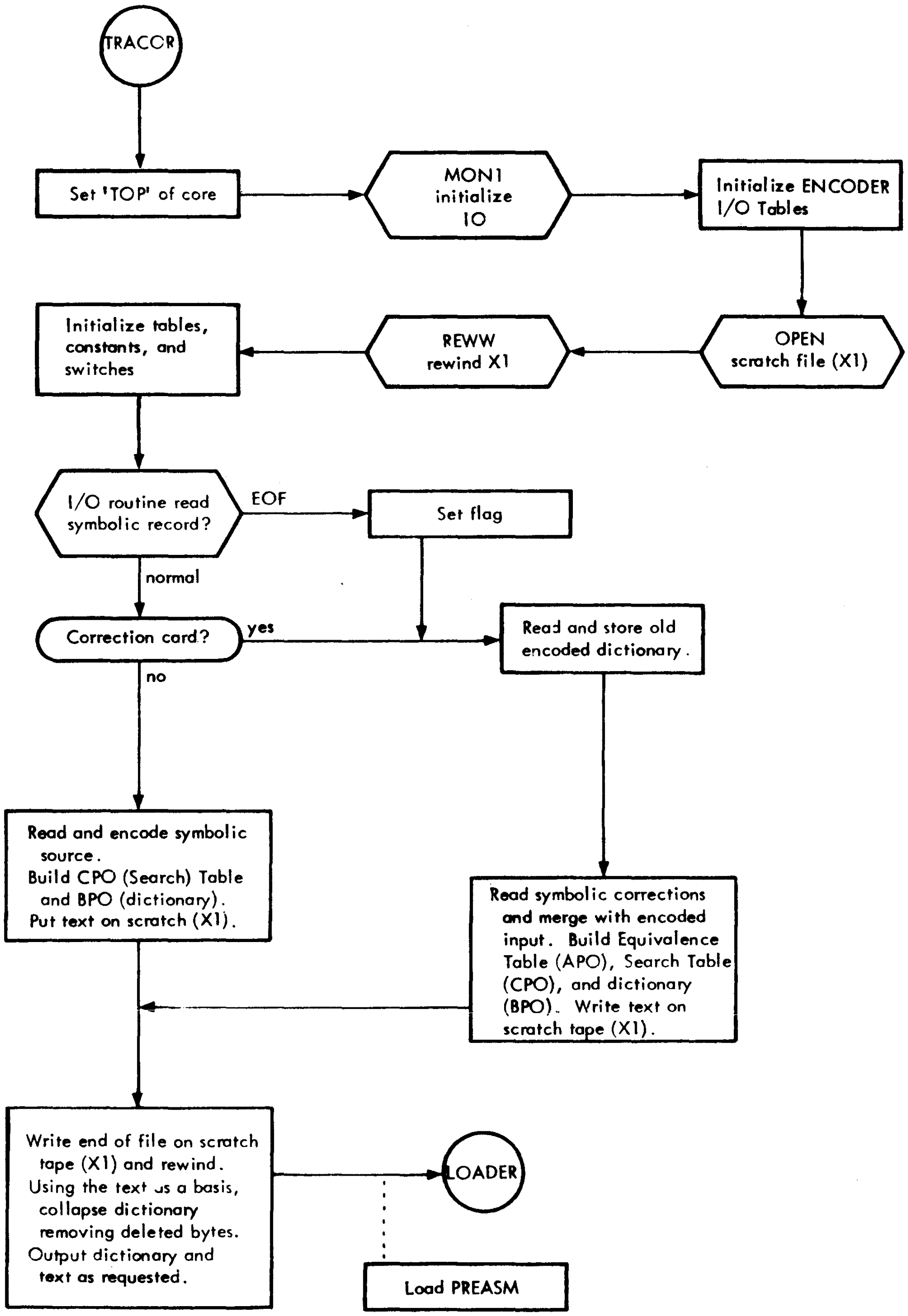
PROGRAMMING TECHNIQUES: EDS is a relocatable routine assembled as part of ENCODER.

CALLING SEQUENCE:
Control word to A register
BRM EDS

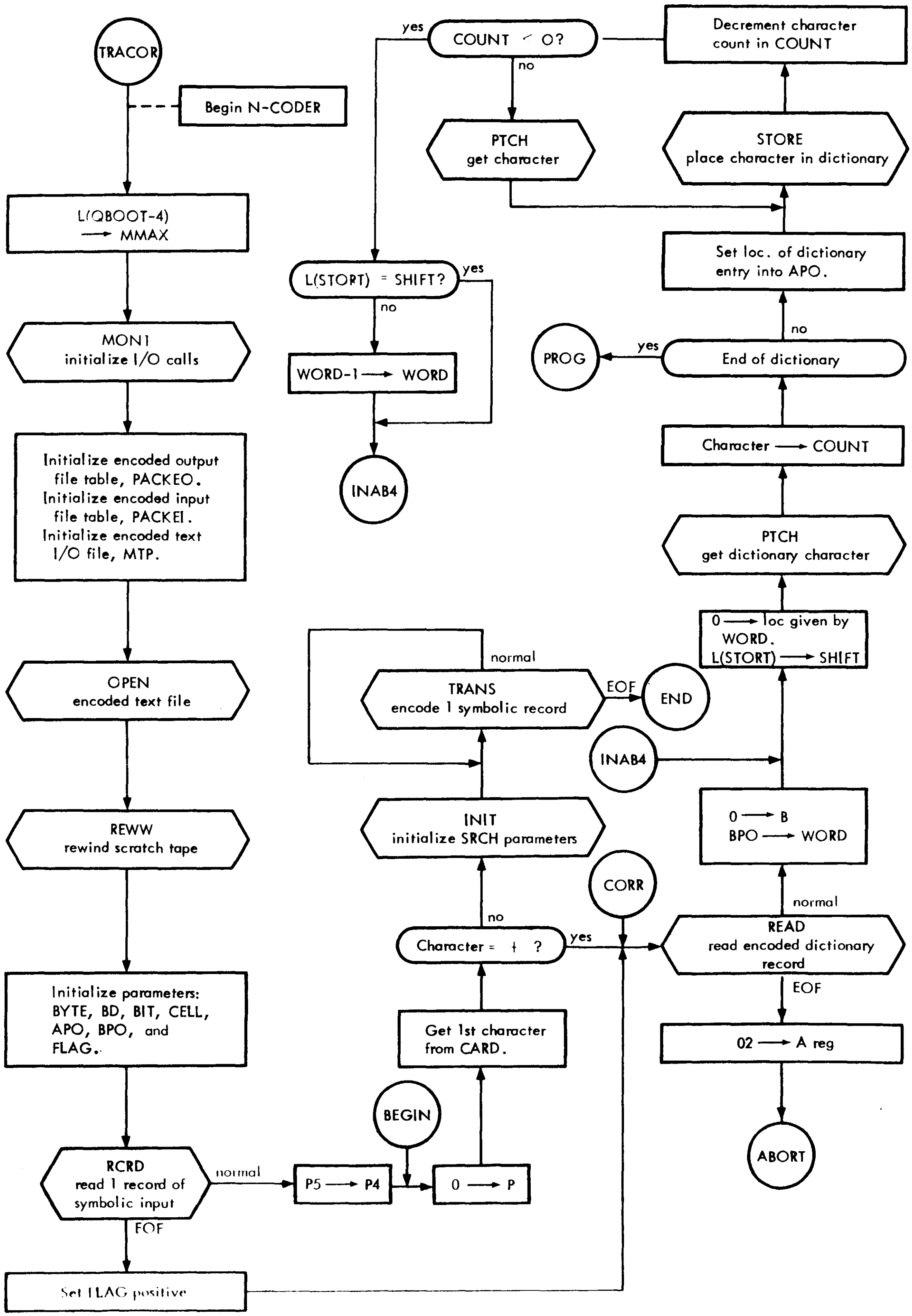
MEMORY REQUIREMENTS: 6 cells

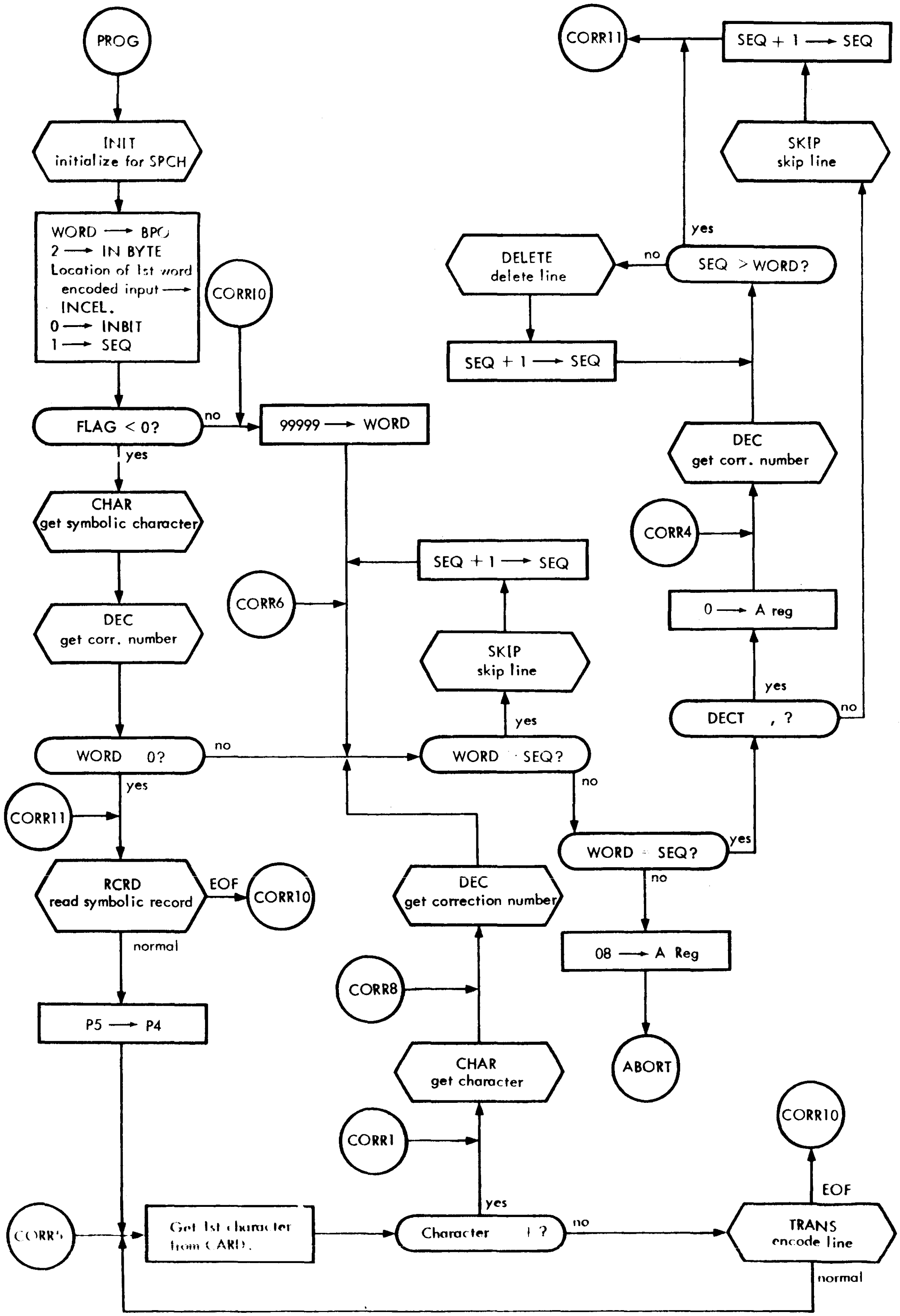
SUBROUTINES USED: None

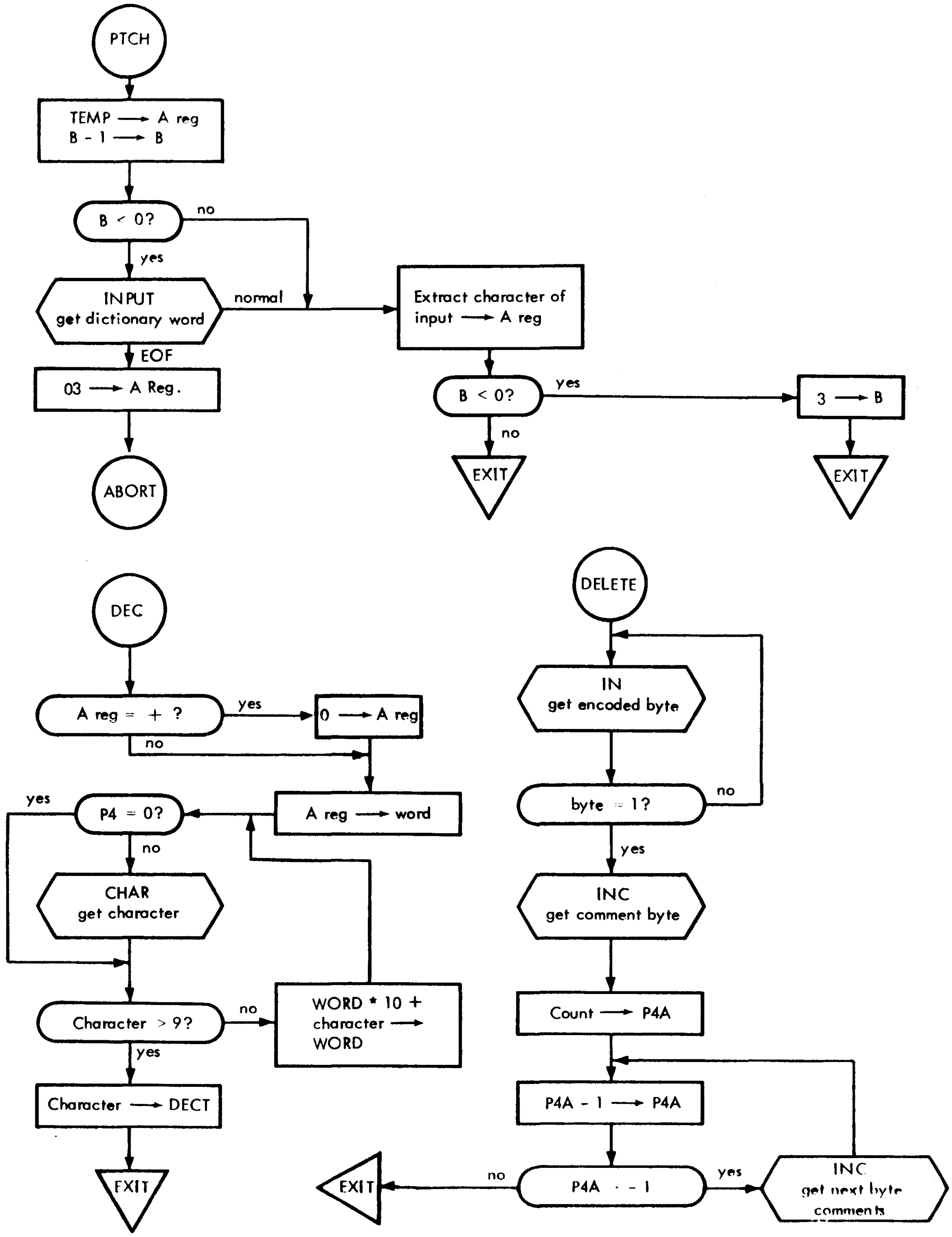
ENCODER
OVERALL FLOW

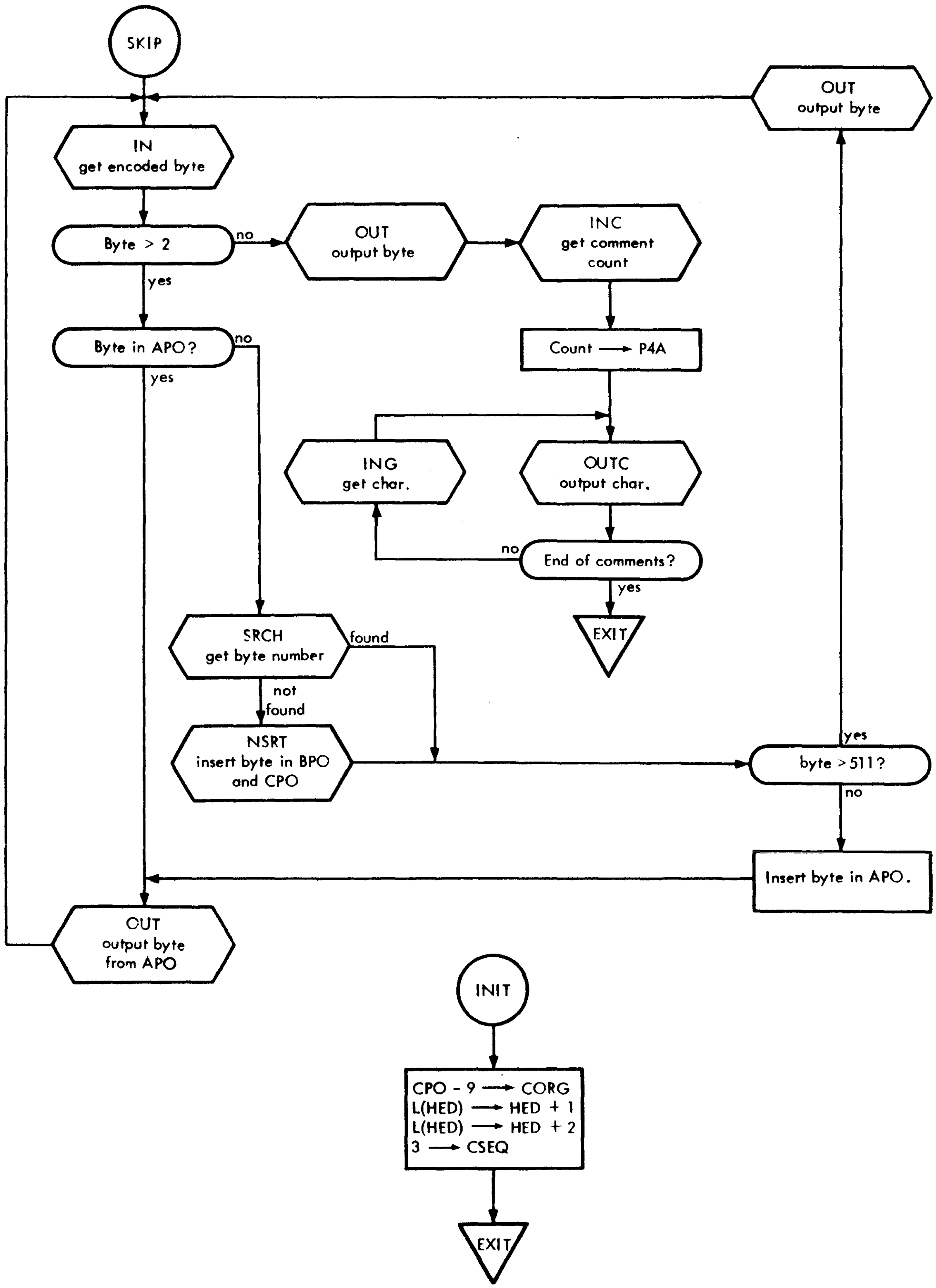


ENCODER
TRACOR, BEGIN, AND CORR
ROUTINES

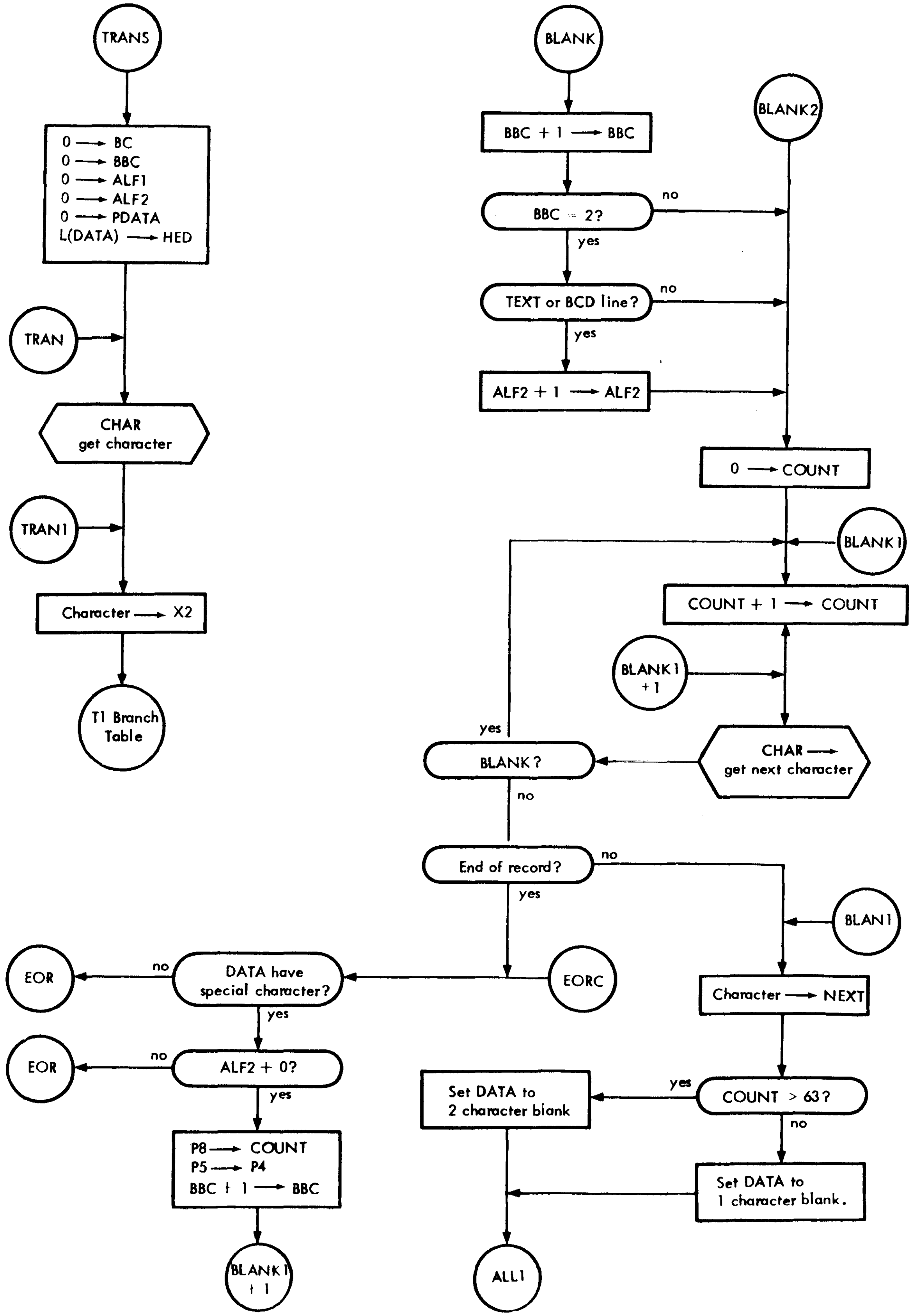


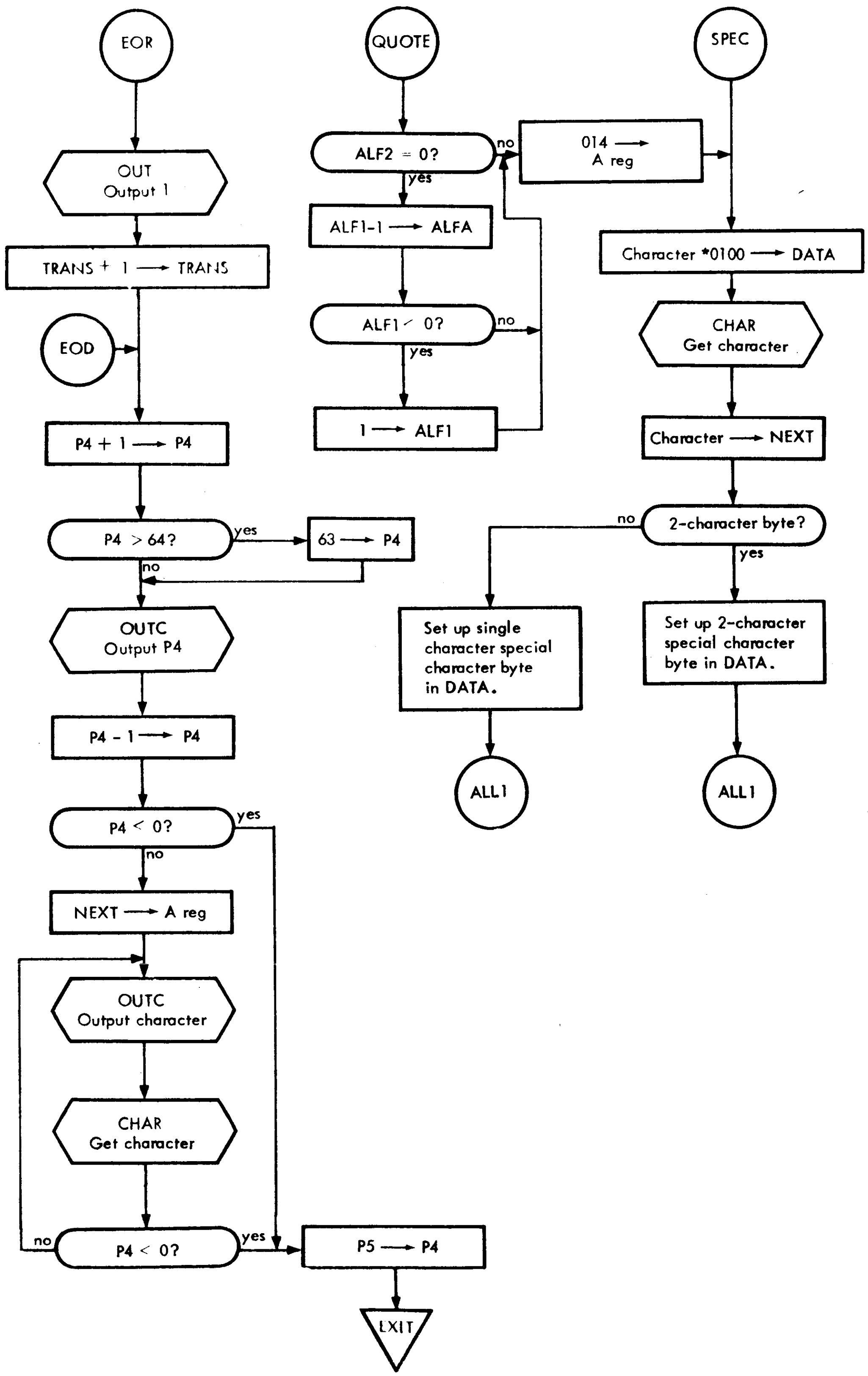

PROG
INIT
initialize for SPCH
WORD → BPO
2 → IN BYTE
Location of 1st word encoded input → INCEL.
0 → INBIT
1 → SEQ
CORR10
FLAG < 0?
no
yes
99999 → WORD
CHAR
get symbolic character
DEC
get corr. number
CORR6
SEQ + 1 → SEQ
SKIP
skip line
yes
WORD 0?
no
yes
WORD SEQ?
no
CORR11
RCRD
read symbolic record
EOF
CORR10
normal
P5 → P4
DEC
get correction number
CORR8
CHAR
get character
CORR1
yes
Get 1st character from CARD.
Character ?
no
TRANS
encode line
EOF
CORR10
normal
CORR11
SEQ + 1 → SEQ
SKIP
skip line
yes
DELETE
delete line
no
SEQ > WORD?
SEQ + 1 → SEQ
DEC
get corr. number
CORR4
0 → A reg
yes
DECT , ?
no
WORD SEQ?
yes
no
08 → A Reg
ABORT

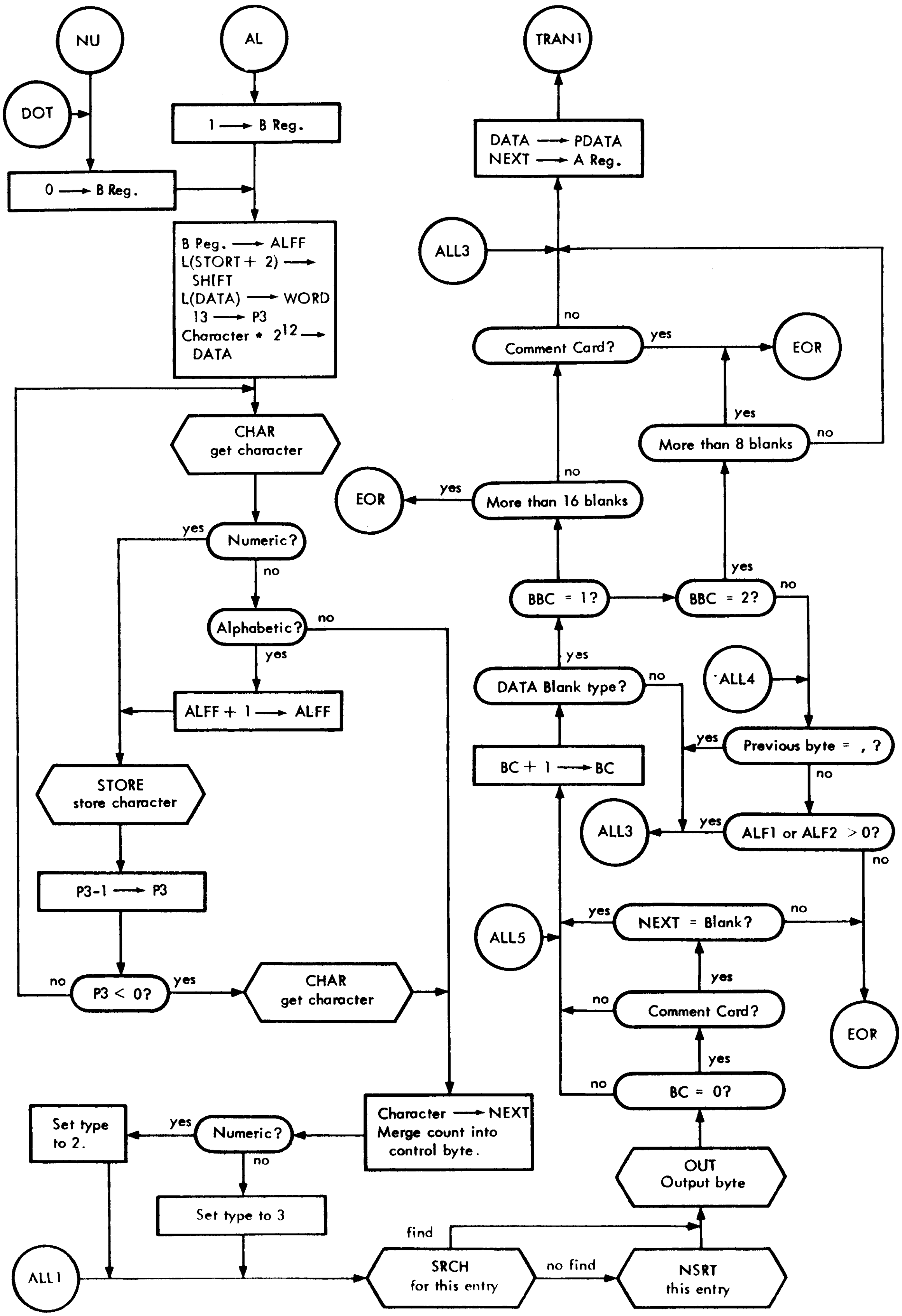

PTCH
TEMP ⟶ A reg
B - 1 ⟶ B
B < 0?
no
yes
INPUT
get dictionary word
normal
EOF
03 ⟶ A Reg.
ABORT
Extract character of
input ⟶ A reg
B < 0?
yes
no
EXIT
3 ⟶ B
EXIT
DEC
A reg = + ?
yes
0 ⟶ A reg
no
A reg ⟶ word
P4 = 0?
yes
no
CHAR
get character
Character > 9?
no
WORD * 10 +
character ⟶
WORD
yes
Character ⟶ DECT
EXIT
DELETE
IN
get encoded byte
byte = 1?
no
yes
INC
get comment byte
Count ⟶ P4A
P4A - 1 ⟶ P4A
P4A = -1
no
yes
EXIT
INC
get next byte
comments

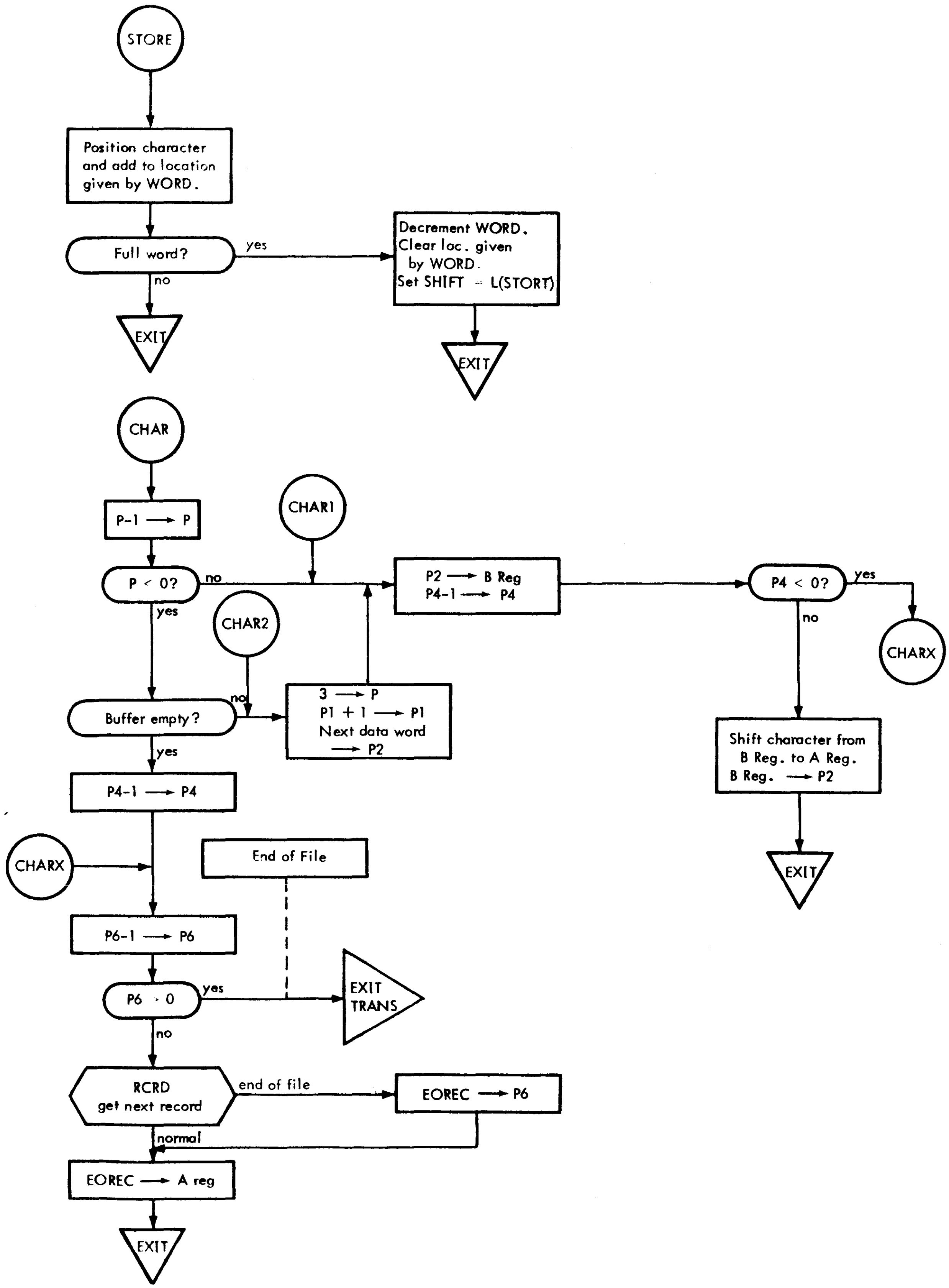

SKIP
OUT
output byte
IN
get encoded byte
Byte > 2
no
yes
OUT
output byte
INC
get comment count
Byte in APO?
no
yes
Count → P4A
ING
get char.
OUTC
output char.
no
End of comments?
yes
EXIT
SRCH
get byte number
found
not found
NSRT
insert byte in BPO and CPO
yes
byte > 511?
no
Insert byte in APO.
OUT
output byte from APO
INIT
CPO - 9 → CORG
L(HED) → HED + 1
L(HED) → HED + 2
3 → CSEQ
EXIT

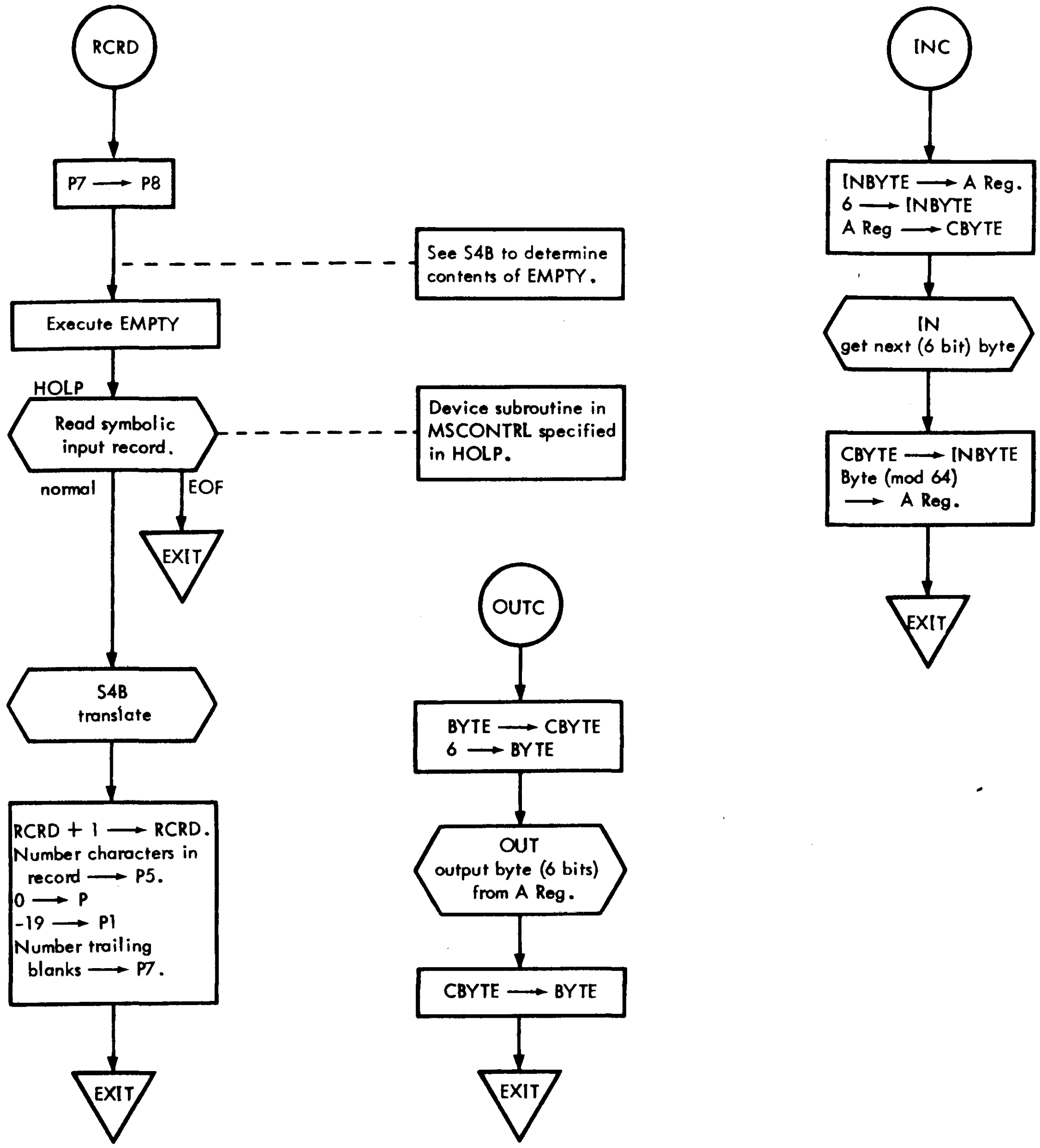
ENCODER
TRANS ROUTINE

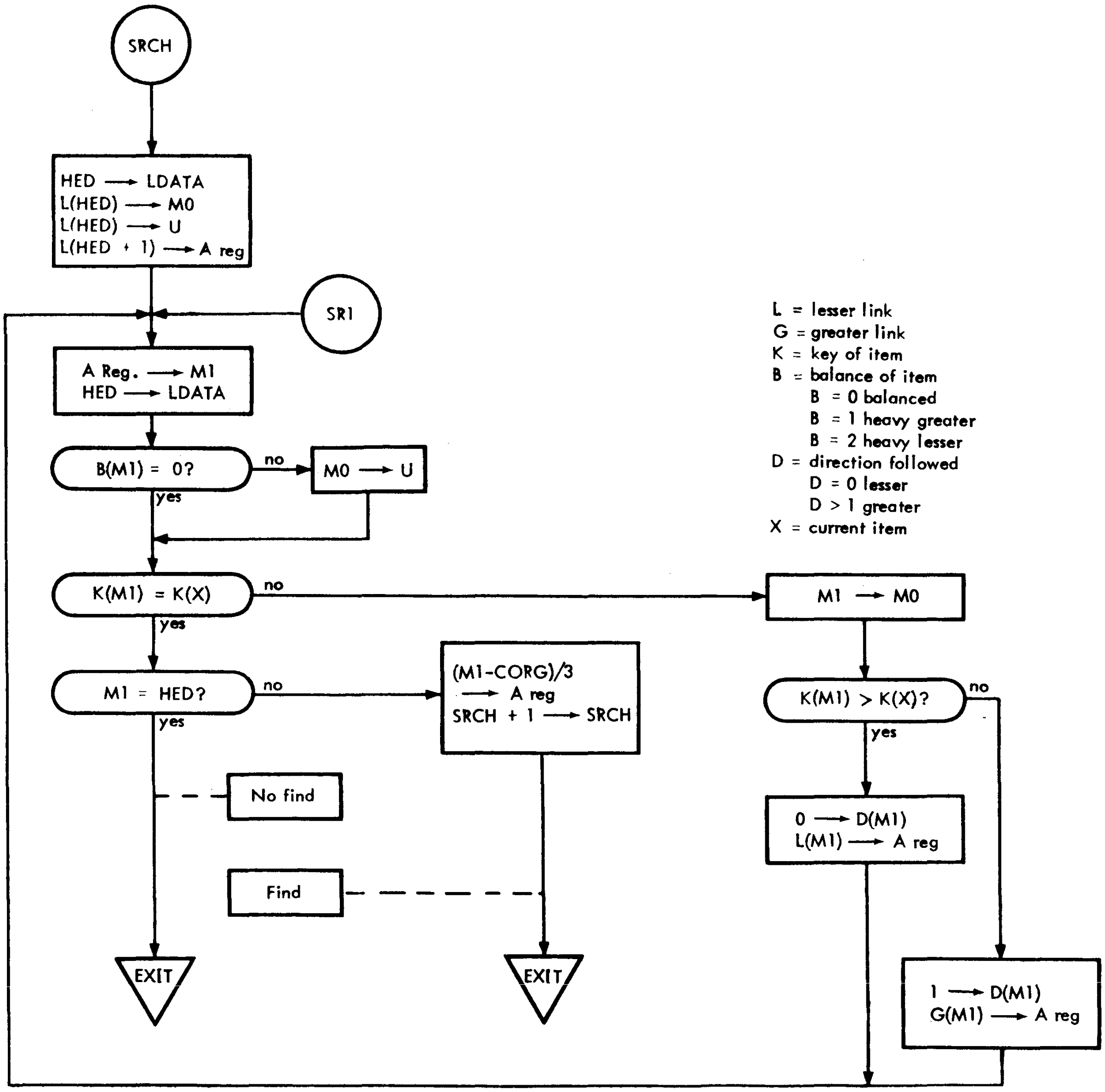



EOR
OUT
Output 1
TRANS + 1 → TRANS
EOD
P4 + 1 → P4
P4 > 64?
yes
63 → P4
no
OUTC
Output P4
P4 - 1 → P4
P4 < 0?
yes
no
NEXT → A reg
OUTC
Output character
CHAR
Get character
no
P4 < 0?
yes
P5 → P4
EXIT
QUOTE
ALF2 = 0?
no
014 →
A reg
yes
ALF1-1 → ALFA
ALF1 < 0?
no
yes
1 → ALF1
SPEC
Character *0100 → DATA
CHAR
Get character
Character → NEXT
2-character byte?
no
yes
Set up single
character special
character byte
in DATA.
Set up 2-character
special character
byte in DATA.
ALL1
ALL1


NU
DOT
AL
1 → B Reg.
0 → B Reg.
B Reg. → ALFF
L(STORT + 2) → SHIFT
L(DATA) → WORD
13 → P3
Character * 2^12 → DATA
CHAR
get character
Numeric?
Alphabetic?
ALFF + 1 → ALFF
STORE
store character
P3-1 → P3
P3 < 0?
CHAR
get character
Character → NEXT
Merge count into control byte.
Numeric?
Set type to 2.
Set type to 3
ALL1
SRCH
for this entry
find
no find
NSRT
this entry
OUT
Output byte
BC = 0?
Comment Card?
NEXT = Blank?
ALL5
EOR
ALF1 or ALF2 > 0?
ALL3
Previous byte = , ?
ALL4
BC + 1 → BC
DATA Blank type?
BBC = 1?
BBC = 2?
More than 16 blanks
EOR
More than 8 blanks
Comment Card?
EOR
ALL3
DATA → PDATA
NEXT → A Reg.
TRAN1
yes
no


STORE
Position character and add to location given by WORD.
Full word?
yes
no
EXIT
Decrement WORD. Clear loc. given by WORD. Set SHIFT = L(STORT)
EXIT
CHAR
P-1 → P
CHAR1
P < 0?
no
yes
P2 → B Reg
P4-1 → P4
P4 < 0?
yes
no
CHARX
CHAR2
Buffer empty?
no
yes
3 → P
P1 + 1 → P1
Next data word → P2
Shift character from B Reg. to A Reg.
B Reg. → P2
EXIT
P4-1 → P4
CHARX
End of File
P6-1 → P6
P6 > 0
yes
EXIT TRANS
no
RCRD
get next record
end of file
EOREC → P6
normal
EOREC → A reg
EXIT


RCRD
P7 → P8
See S4B to determine contents of EMPTY.
Execute EMPTY
HOLP
Read symbolic input record.
Device subroutine in MSCONTRL specified in HOLP.
normal
EOF
EXIT
S4B translate
RCRD + 1 → RCRD. Number characters in record → P5. 0 → P -19 → P1 Number trailing blanks → P7.
EXIT
OUTC
BYTE → CBYTE 6 → BYTE
OUT output byte (6 bits) from A Reg.
CBYTE → BYTE
EXIT
INC
INBYTE → A Reg. 6 → INBYTE A Reg → CBYTE
IN get next (6 bit) byte
CBYTE → INBYTE Byte (mod 64) → A Reg.
EXIT


SRCH
HED → LDATA
L(HED) → M0
L(HED) → U
L(HED + 1) → A reg
SR1
A Reg. → M1
HED → LDATA
B(M1) = 0?
no
yes
M0 → U
K(M1) = K(X)
no
yes
M1 → M0
M1 = HED?
no
yes
(M1-CORG)/3
→ A reg
SRCH + 1 → SRCH
K(M1) > K(X)?
no
yes
No find
Find
0 → D(M1)
L(M1) → A reg
1 → D(M1)
G(M1) → A reg
EXIT
EXIT
L = lesser link
G = greater link
K = key of item
B = balance of item
B = 0 balanced
B = 1 heavy greater
B = 2 heavy lesser
D = direction followed
D = 0 lesser
D > 1 greater
X = current item

Let α denote some byte entry in the table; then:

L (α) is the pointer from α to a lesser item.

G (α) is the pointer from α to a greater item.

K (α) is the key of α.

B (α) is the balance of α.
B (α) = 0 denotes balance.
B (α) = 1 denotes heavy in the greater chain.
B (α) = 2 denotes heavy in the lesser chain.

D (α) is the direction followed from α in searching for an item.
D (α) = 0 denotes lesser chain taken.
D (α) = 1 denotes greater chain taken.

X denotes current item to insert.

F (α) denotes the item following α on the search path taken.

Q (α) denotes the item following α on the path other than that taken.

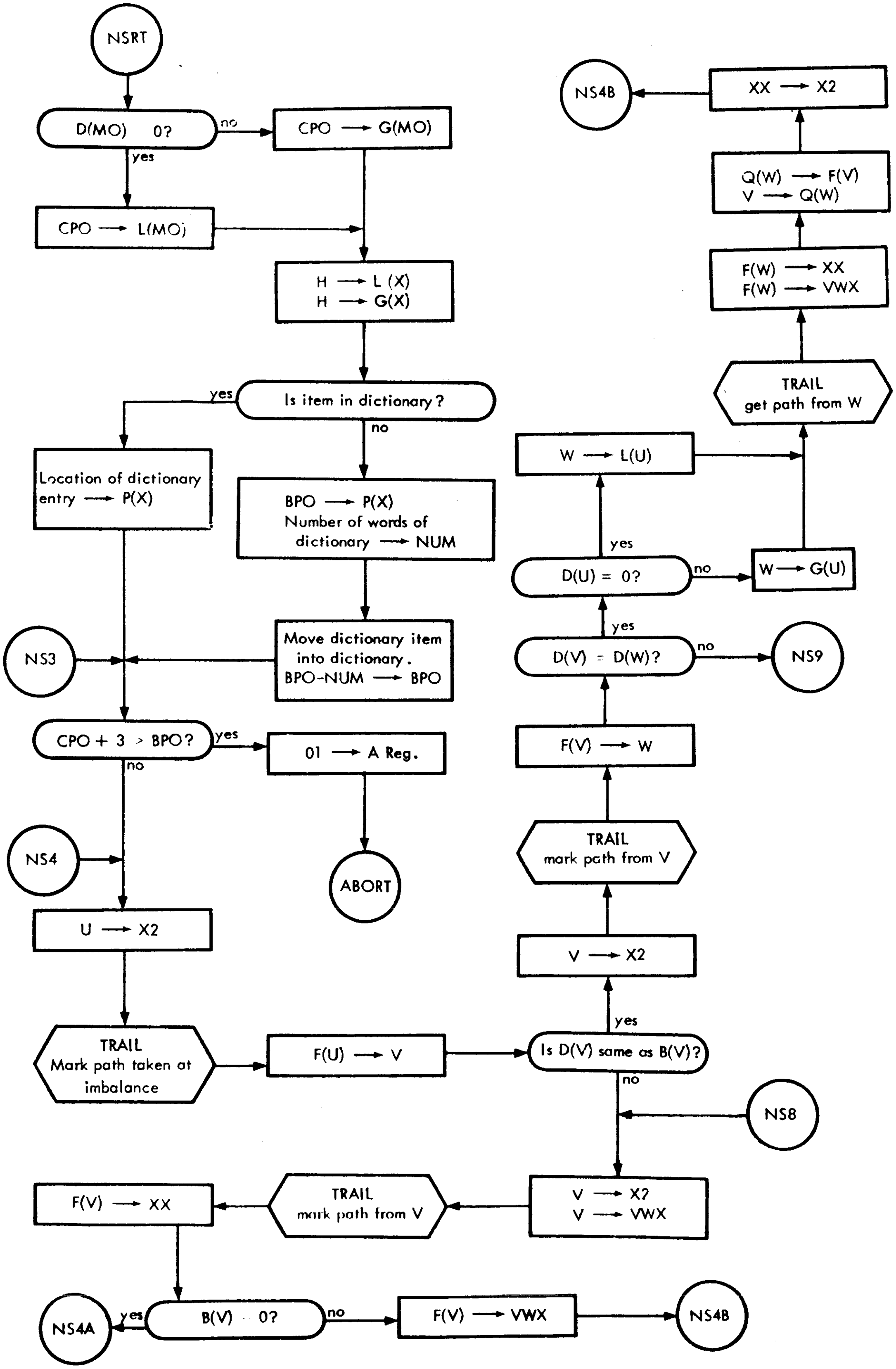
U denotes the last point of imbalance on the last search path.

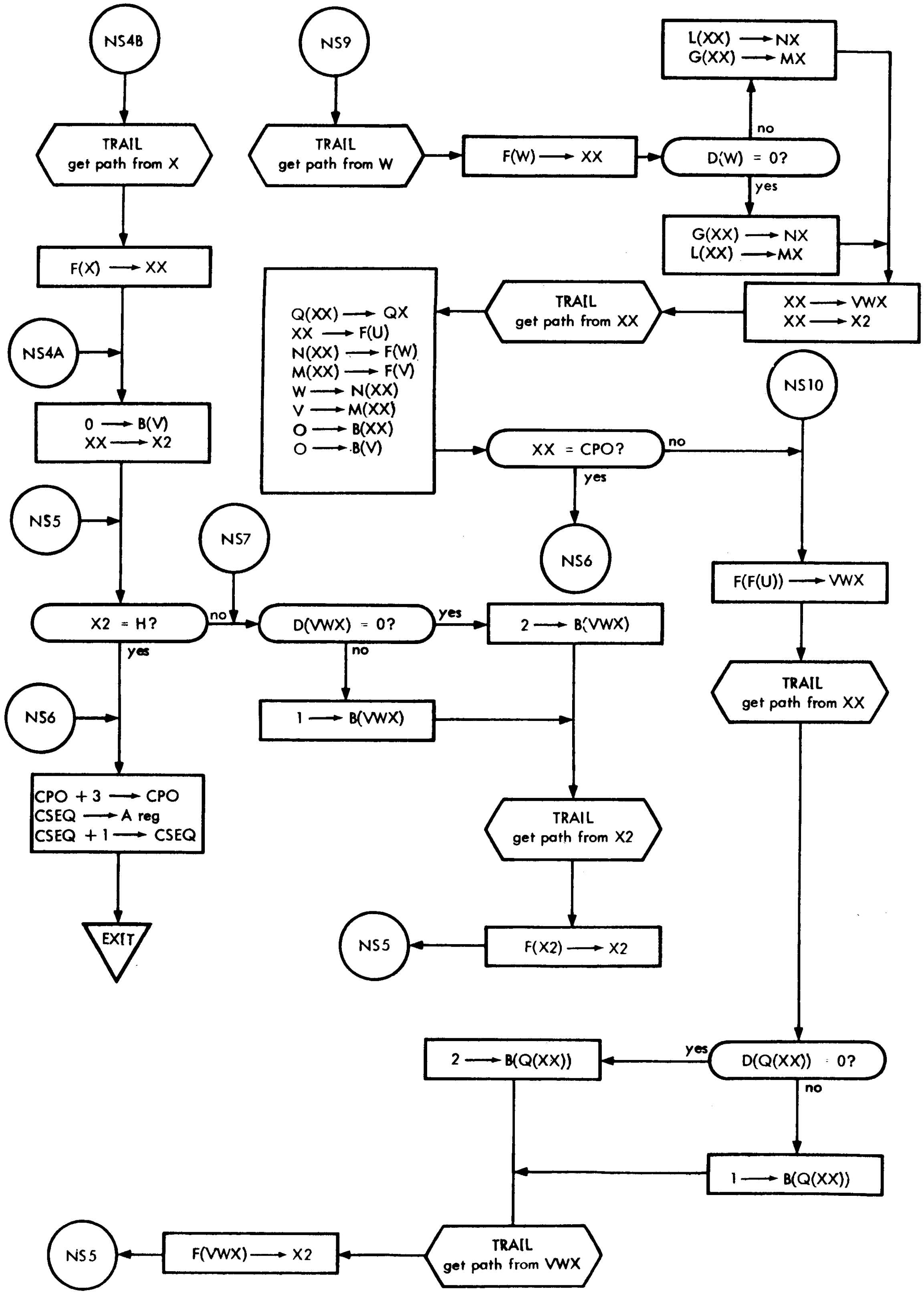
MO denotes the last point examined by SRCH.

M (β) and N (β) are defined such that
If G (α) = β then M (β) = G (β)
and N (β) = L (β)
If L (α) = β then M (β) = L (β)
and N (β) = G (β)

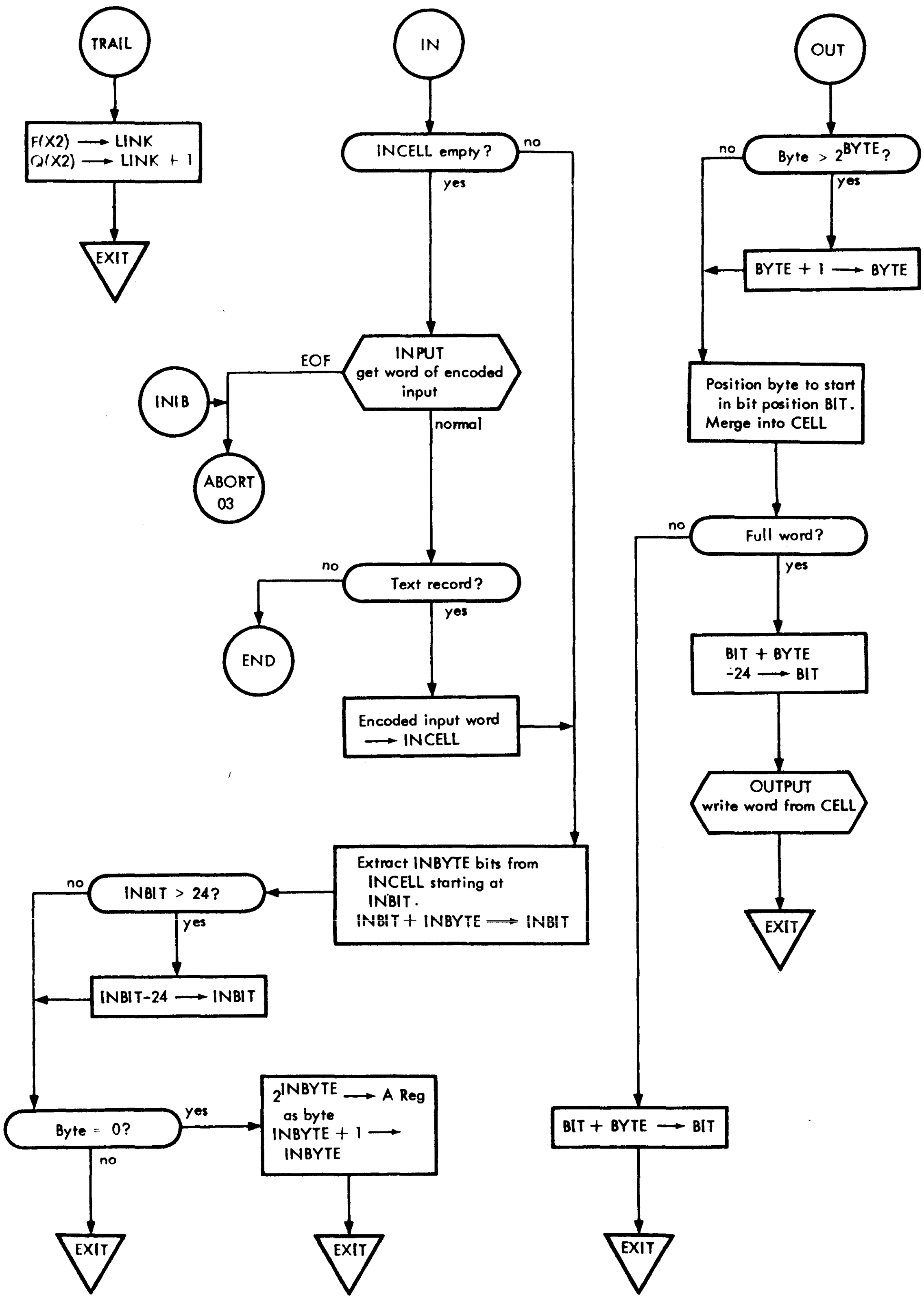
H denotes location of HED.

P (α) denotes location of dictionary entry for byte α.

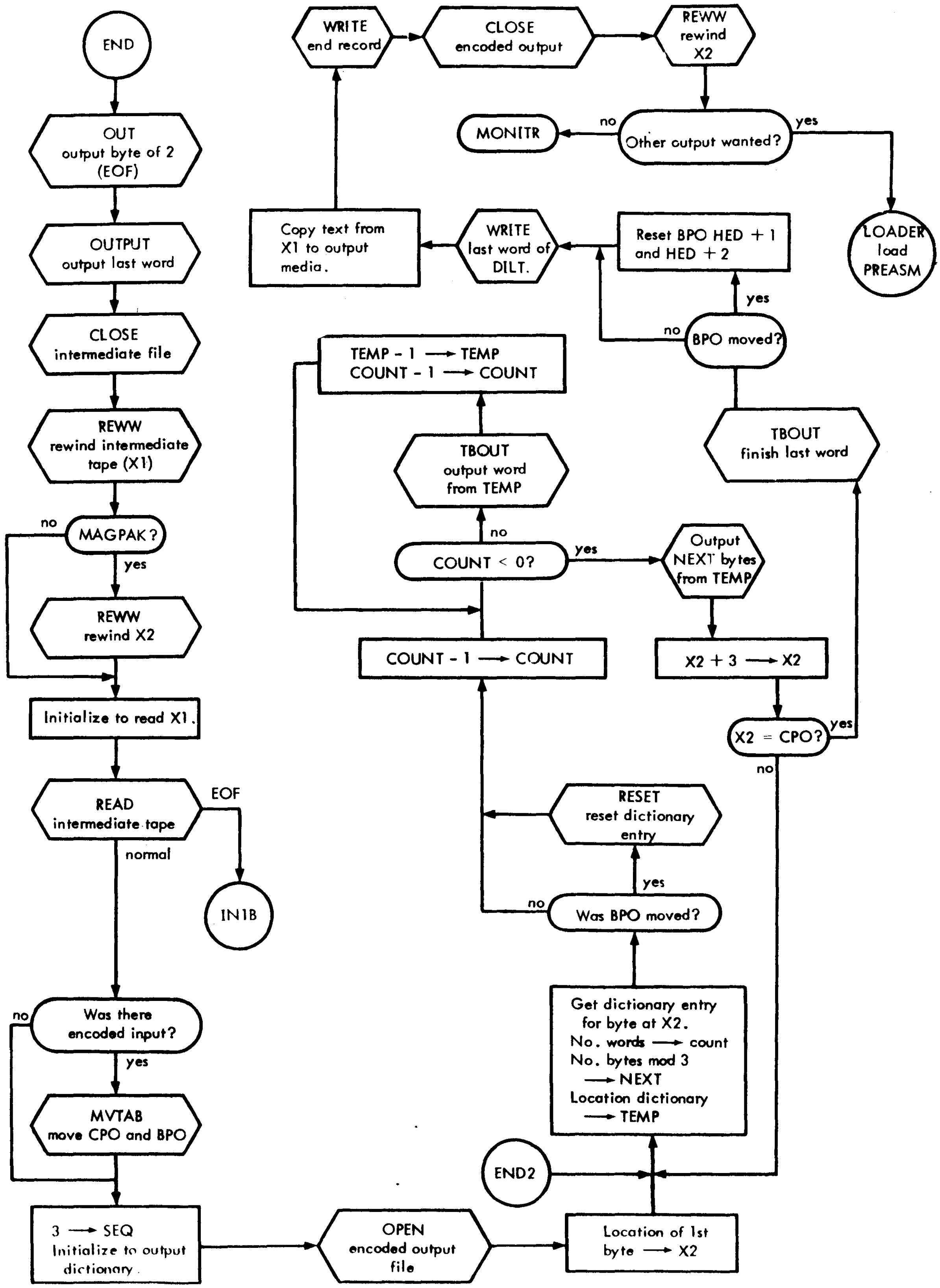

NSRT
D(MO) 0?
no
yes
CPO → G(MO)
CPO → L(MO)
H → L (X)
H → G(X)
Is item in dictionary?
yes
no
Location of dictionary entry → P(X)
BPO → P(X)
Number of words of dictionary → NUM
Move dictionary item into dictionary.
BPO-NUM → BPO
NS3
CPO + 3 > BPO?
yes
no
01 → A Reg.
ABORT
NS4
U → X2
TRAIL
Mark path taken at imbalance
F(U) → V
Is D(V) same as B(V)?
yes
no
NS8
V → X2
V → VWX
TRAIL
mark path from V
F(V) → XX
B(V) 0?
yes
no
NS4A
F(V) → VWX
NS4B
V → X2
TRAIL
mark path from V
F(V) → W
D(V) = D(W)?
no
NS9
yes
D(U) = 0?
yes
no
W → L(U)
W → G(U)
TRAIL
get path from W
F(W) → XX
F(W) → VWX
Q(W) → F(V)
V → Q(W)
XX → X2
NS4B

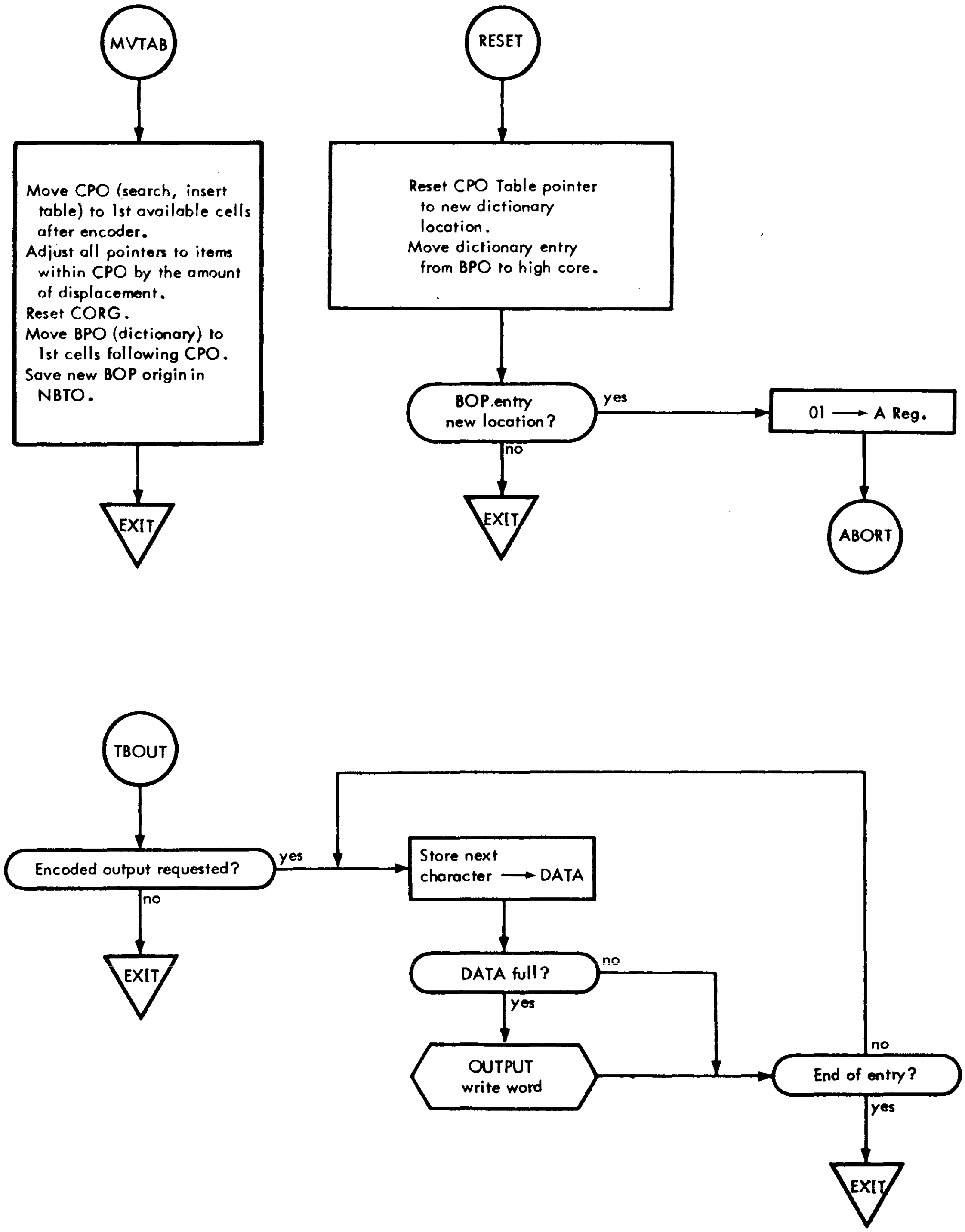

NS4B
TRAIL get path from X
F(X) → XX
NS4A
0 → B(V)
XX → X2
NS5
X2 = H?
no
yes
NS6
CPO + 3 → CPO
CSEQ → A reg
CSEQ + 1 → CSEQ
EXIT
NS9
TRAIL get path from W
F(W) → XX
D(W) = 0?
no
yes
L(XX) → NX
G(XX) → MX
G(XX) → NX
L(XX) → MX
XX → VWX
XX → X2
TRAIL get path from XX
Q(XX) → QX
XX → F(U)
N(XX) → F(W)
M(XX) → F(V)
W → N(XX)
V → M(XX)
O → B(XX)
O → B(V)
XX = CPO?
no
yes
NS6
NS7
D(VWX) = 0?
yes
no
2 → B(VWX)
1 → B(VWX)
TRAIL get path from X2
F(X2) → X2
NS5
NS10
F(F(U)) → VWX
TRAIL get path from XX
D(Q(XX)) = 0?
yes
no
2 → B(Q(XX))
1 → B(Q(XX))
TRAIL get path from VWX
F(VWX) → X2
NS5

ENCODER
TRAIL, IN AND OUT ROUTINES

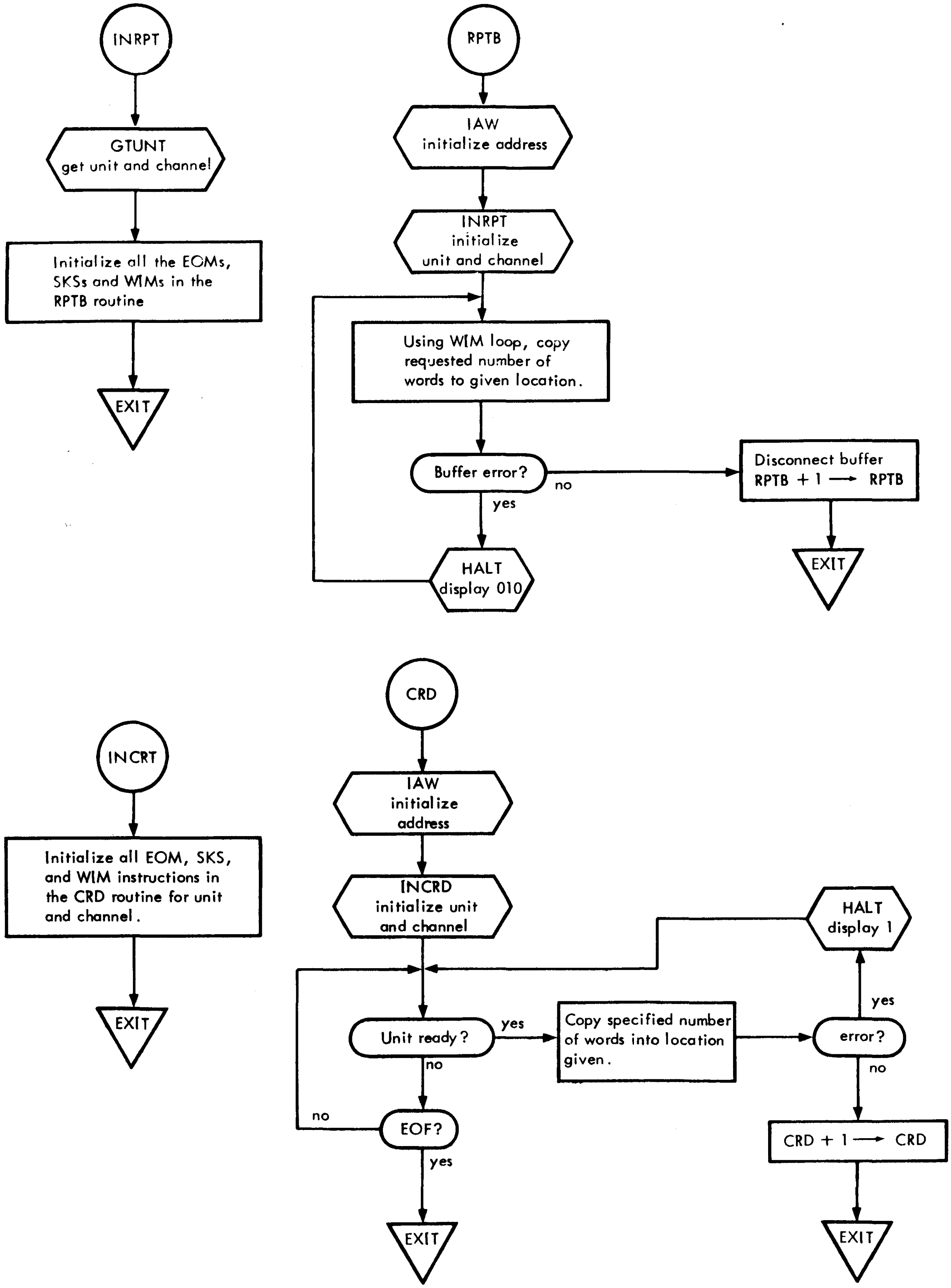


ENCODER
END ROUTINE

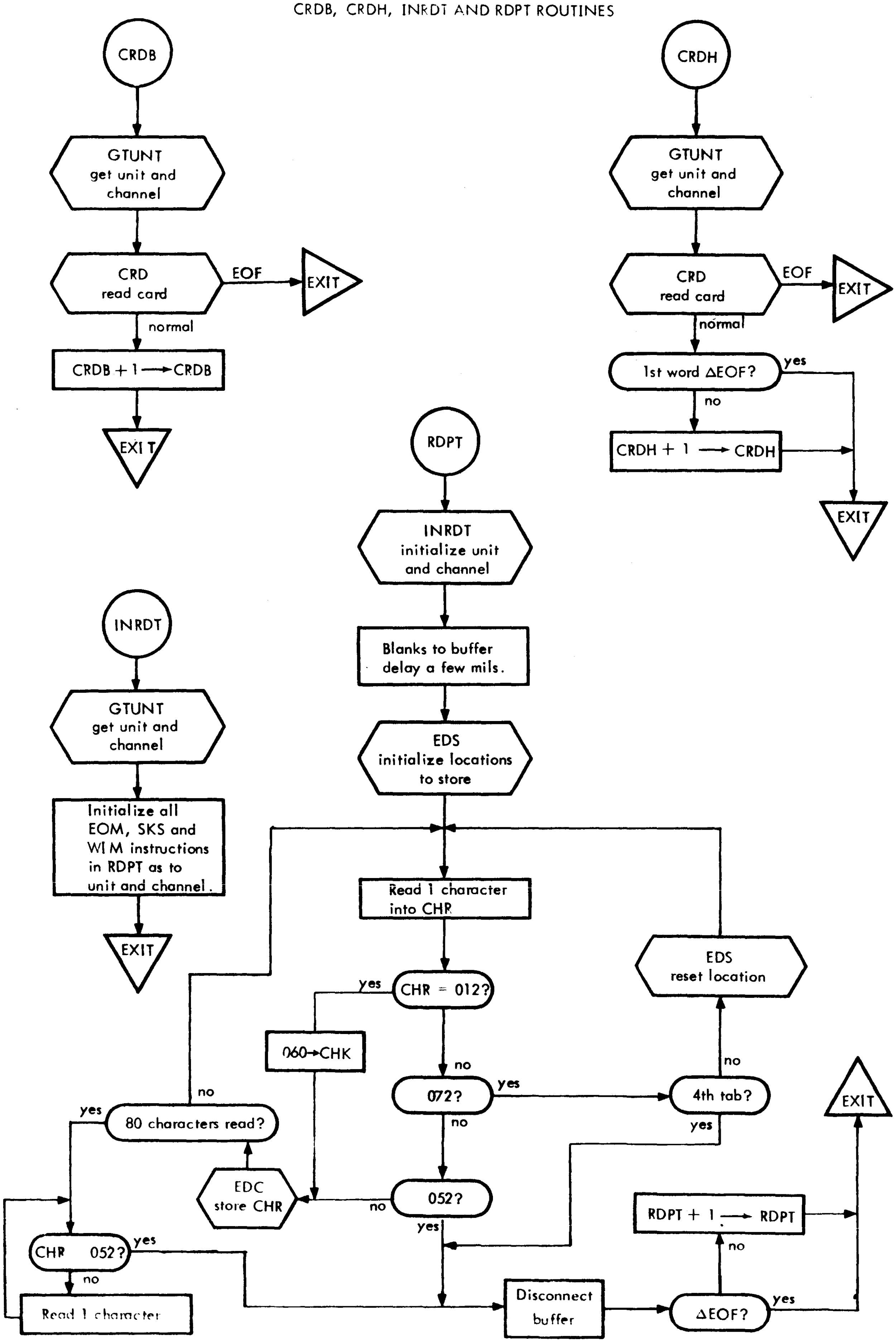



MVTAB
Move CPO (search, insert table) to 1st available cells after encoder.
Adjust all pointers to items within CPO by the amount of displacement.
Reset CORG.
Move BPO (dictionary) to 1st cells following CPO.
Save new BOP origin in NBTO.
EXIT
RESET
Reset CPO Table pointer to new dictionary location.
Move dictionary entry from BPO to high core.
BOP entry new location?
yes
no
EXIT
01 → A Reg.
ABORT
TBOUT
Encoded output requested?
yes
no
EXIT
Store next character → DATA
DATA full?
no
yes
OUTPUT write word
End of entry?
no
yes
EXIT

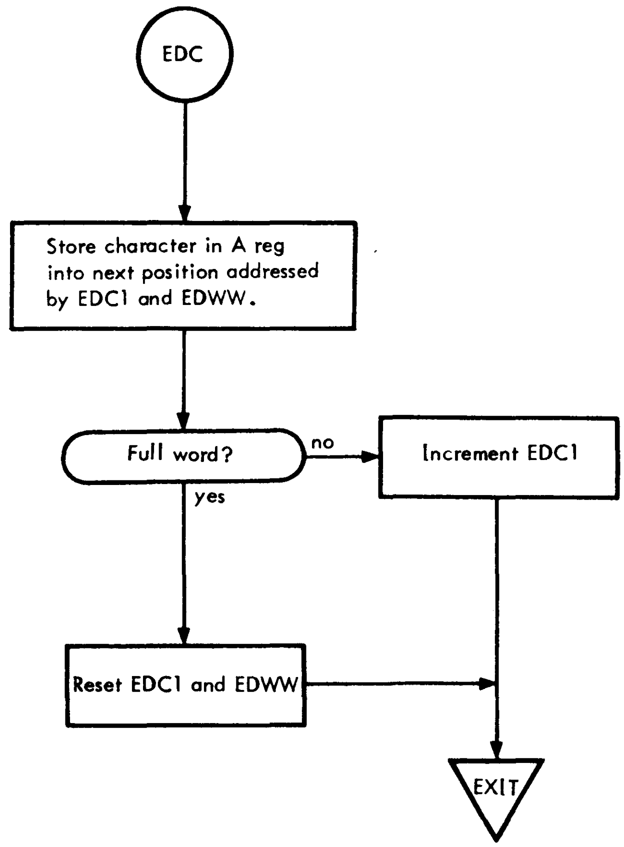
ENCODER
INRPT, RPTB, INCRD AND CRD ROUTINES

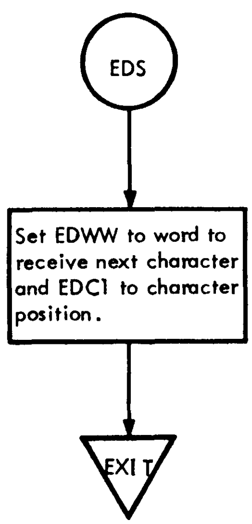


ENCODER
CRDB, CRDH, INRDT AND RDPT ROUTINES



ENCODER
EDC AND EDS ROUTINES





SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 900 Series: 042016
9300: 612001

IDENTIFICATION: Translation program (S4B)

PURPOSE: To translate symbolic input from SYMBOL 4 or SYMBOL 8 language into META-SYMBOL language.

ACTION: S4B tests the control cell MSFNC to determine if translation has been requested. If no trnaslation is requested, S4B exits immediately. The input record is scanned for items to be translated which includes the following:

1. MACROs written in the SYMBOL 8 format are translated to PROCs and NAMEs in the META-SYMBOL format; ENDM is translated to END.
2. Fields which have assumed octal values in the input language are supplied leading zero characters in the META-SYMBOL format.
3. Decimal and binary scale factors are converted to META-SYMBOL format.
4. Indirect flags are moved from the last character of the operation field to the leading character position of the operand field.
5. Operand fields of VFD directives are translated to META-SYMBOL lists.
6. BCI directives are converted to BCD directives.
7. Leading O or H characters on literals are replaced by 0 (zero) and leading and trailing ' respectively.
8. DEC and OCT directives are translated to DATA.
9. Parameter references within macros are translated to META-SYMBOL format.

ACTION: (cont.)

If an input record results in an expansion necessitating the generation of two or more output records, the location EMPTY is set to transfer to the appropriate point in S4B to resume the translation. EMPTY, normally a NOP instruction, is executed by the RCRD routine of ENCODER before reading the next symbolic input record and in this way proper translation results.

PROGRAMMING TECHNIQUES:

S4B as a translation program is designed to work with ENCODER and may be thought of as an extension to the symbolic input section of ENCODER. Note that only symbolic inputs are translated, the assumption being that all encoded outputs have been translated from the original symbolic on the initial encoding run. S4B is a separately assembled relocatable routine to be loaded behind ENCODER and the programmed operator routines. The cell TABLES in ENCODER is set to a value which addresses some location following S4B. Since this cell represents the starting location in lower memory for tables constructed by ENCODER, any increase in the size of S4B may result in the need to reassemble ENCODER. The symbols EMPTY and S4B are defined as external for reference within ENCODER. The symbol CARD within S4B addresses the symbolic input buffer within ENCODER; and, if the location of this buffer (CARD) changes in ENCODER, it is necessary to reassemble S4B to reflect this shift.

CALLING SEQUENCE:

Initial entry to S4B for record
BRM S4B

Subsequent entries to S4B to resume translation of a single symbolic record.

EXU EMPTY

The return in this case is to the location of the EXU plus 4.

MEMORY REQUIREMENTS: 1220_8 cells

SUBROUTINES USED:

TENC	MOVE
OCTC	RESET
NUM	GET
NAME	PUT
PARAMS	

ENTRY POINTS TO S4B SUBROUTINES

Entry	Page Description	Page Flowchart	Entry	Page Description	Page Flowchart
BCI		3-108	OCT8	3-99	3-109
BINS		3-110	OCTC	3-99	3-109
DECS		3-110	PARAMS	3-102	3-112
DECS2		3-110	PUT	3-106	3-113
DED		3-108	RESET	3-104	3-113
ENDM		3-111	S4B	3-94	3-107
GET	3-105	3-113	S4B02	3-94	3-108
LIT	3-94	3-107	S4B03	3-94	3-108
LIT1	3-94	3-107	S4B1	3-94	3-108
MACRO		3-112	S4B2	3-94	3-107
MOVE	3-103	3-113	S4B6	3-94	3-107
NAME	3-101	3-112	TEN	3-98	3-108
NUM	3-100	3-109	TEN3	3-98	3-108
NUM1	3-100	3-109	TENC	3-98	3-108
NUM2	3-100	3-109	VFD		3-111
OCT	3-99	3-109	VFD3		3-111
OCT5	3-99	3-109	VFD4		3-111
OCT6	3-99	3-109			

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Resume translation of DEC directive (TENC)

PURPOSE: To initialize parameters and exit location when resuming the translation of the operand field of a DEC directive which expands to more than 72 characters.

ACTION: TENC sets S4B to the location of the EXU EMPTY instruction plus 4 and then calls RESET to initialize the card buffer for resuming the translation. TENC exits to TEN3 to continue the translation process.

PROGRAMMING CONVENTIONS: TENC is executed only after the DEC translation code is unable to translate the input image into a 72-character META-SYMBOL equivalent because of space. It assumes the ENCODER will remotely execute the instruction at EMPTY. TENC is a relocatable routine assembled as part of S4B.

CALLING SEQUENCE: A BRM TENC is stored in EMPTY, and TENC is called when EMPTY is executed.

MEMORY REQUIREMENTS: 6 cells

SUBROUTINES USED: RESET

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 0420
Catalog No. 9300: 6120(

IDENTIFICATION: Resume translation of OCT directive (OCTC)

PURPOSE: To reset the S4B exit and the symbolic input buffer to resume the translation of the OCT directive.

ACTION: OCTC sets S4B to the location of the EXU EMPTY plus 4 and then calls RESET to initialize the symbolic input buffer to resume translation of the OCT directive.

PROGRAMMING TECHNIQUES: OCTC is called only when during the translation of an OCT directive the symbolic card buffer filled before the translation could be completed. It assumes the ENCODER will remotely execute cell EMPTY. OCTC is a relocatable routine assembled as part of S4B.

CALLING SEQUENCE: A BRM OCTC is stored in EMPTY during the translation of an input OCT directive. OCTC is called by executing EMPTY remotely.

MEMORY REQUIREMENTS: 7 cells

SUBROUTINES USED: RESET

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Convert input numeric fields to META-SYMBOL format (NUM)

PURPOSE: To translate input symbolic numeric fields from SYMBOL 4 or SYMBOL 8 format to META-SYMBOL format.

ACTION: NUM obtains characters one at a time by calling GET until a terminator is obtained. Binary scaling factors are converted from the B notation to */; decimal scaling factors are converted from the E notation to *+. Characters are stored in the input symbolic buffer by calling PUT.

PROGRAMMING TECHNIQUES: NUM is a relocatable routine assembled as part of S4B.

CALLING SEQUENCE: BRM NUM

MEMORY REQUIREMENTS: 112_8 cells

SUBROUTINES USED: GET
PUT

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 04201
Catalog No. 9300: 61200

IDENTIFICATION: Define name lines for MACRO directives being translated (NAME)

PURPOSE: To define NAME directives by which procedures being generated from input MACRO lines may be called.

ACTION: NAME sets the exit from S4B and calls RESET to initialize the symbolic input buffer. NAME then sets NAME in the operation field of the buffer and the label from the MACRO line into the label field in the buffer. Before exiting S4B, NAME sets EMPTY to contain BRM PARAMS for translating the parameters to the MACRO sample.

PROGRAMMING TECHNIQUES: NAME is called only as a result of encountering an input MACRO line. NAME is a relocatable routine assembled as part of S4B.

CALLING SEQUENCE: BRM NAME is stored in EMPTY
NAME is called when EMPTY is executed.
NAME returns to the EXU EMPTY plus 4.

MEMORY REQUIREMENTS: 21_8 cells

SUBROUTINES USED: RESET

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 04201o
Catalog No. 9300: 612001

IDENTIFICATION: Define parameters on MACRO lines (PARAMS)

PURPOSE: To translate macro parameters to a format suitable to META-SYMBOL.

ACTION: PARAMS provides for the translation of parametric values by defining each parameter equal to the corresponding META-SYMBOL subscripted symbol for the parameter. PARAMS sets the S4B exit location and then calls RESET to initialize the input buffer. The next parameter is then placed in the label field, the operation is set to EQU, and the operand field is set to the REFLIST (n), where REFLIST is the label given the PROC line generated in place of the MACRO and n is the current parameter number.

PROGRAMMING TECHNIQUES: PARAMS is called only after generating a NAME line as part of the translation of a MACRO. PARAMS is a relocatable routine assembled as part of S4B.

CALLING SEQUENCE: PARAMS is called by an EXU EMPTY after EMPTY has been set with a BRM PARAMS by the NAME routine. Return is to the EXU EMPTY plus 4.

MEMORY REQUIREMENTS: 44_8 cells

SUBROUTINES USED:
GET
PUT
RESET

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 0420
Catalog No. 9300: 6120

IDENTIFICATION: Save symbolic lines of input (MOVE)

PURPOSE: To move symbolic input lines and clear the symbolic input buffer to blanks in anticipation of having to translate a line and expand its size.

ACTION: MOVE moves the contents of the buffer CARD to the buffer XCRD and stores blanks in CARD. MOVE then initializes the cells GETD, GETW, and GETCT for GET and the cells PUTT and PUTW for the PUT routine.

PROGRAMMING TECHNIQUES: MOVE is a relocatable routine assembled as part of S4B.

CALLING SEQUENCE: BRM MOVE

MEMORY REQUIREMENTS: 23_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Initialize symbolic input buffer when continuing translation (RESET)

PURPOSE: To set the last 16 words of the symbolic input buffer to blank and to initialize the PUT routine parameters.

ACTION: RESET stores blanks in the last 16 words of CARD and also in the label field of CARD. RESET then initializes the PUT parameter words PUTT and PUTW.

PROGRAMMING TECHNIQUES: RESET is a relocatable routine assembled as part of S4B.

CALLING SEQUENCE: BRM RESET

MEMORY REQUIREMENTS: 25_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Get next character of symbolic source (GET)

PURPOSE: To get the next character of symbolic input into the A register.

ACTION: GET extracts the next input character into the low order bit positions of the A register and GETC. The pointers to the next character of source are incremented. If the character is a comma or a blank, GET exits to the location following the call; otherwise, it exits to the location of the call plus 2.

PROGRAMMING TECHNIQUES: GET is a relocatable routine assembled as part of S4B.

CALLING SEQUENCE:

```
BRM   GET
end-of-entry return
normal return
```

MEMORY REQUIREMENTS: 35_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Store a character of translated source (PUT)

PURPOSE: To insert a character located in the A register into the next character position of the symbolic input buffer.

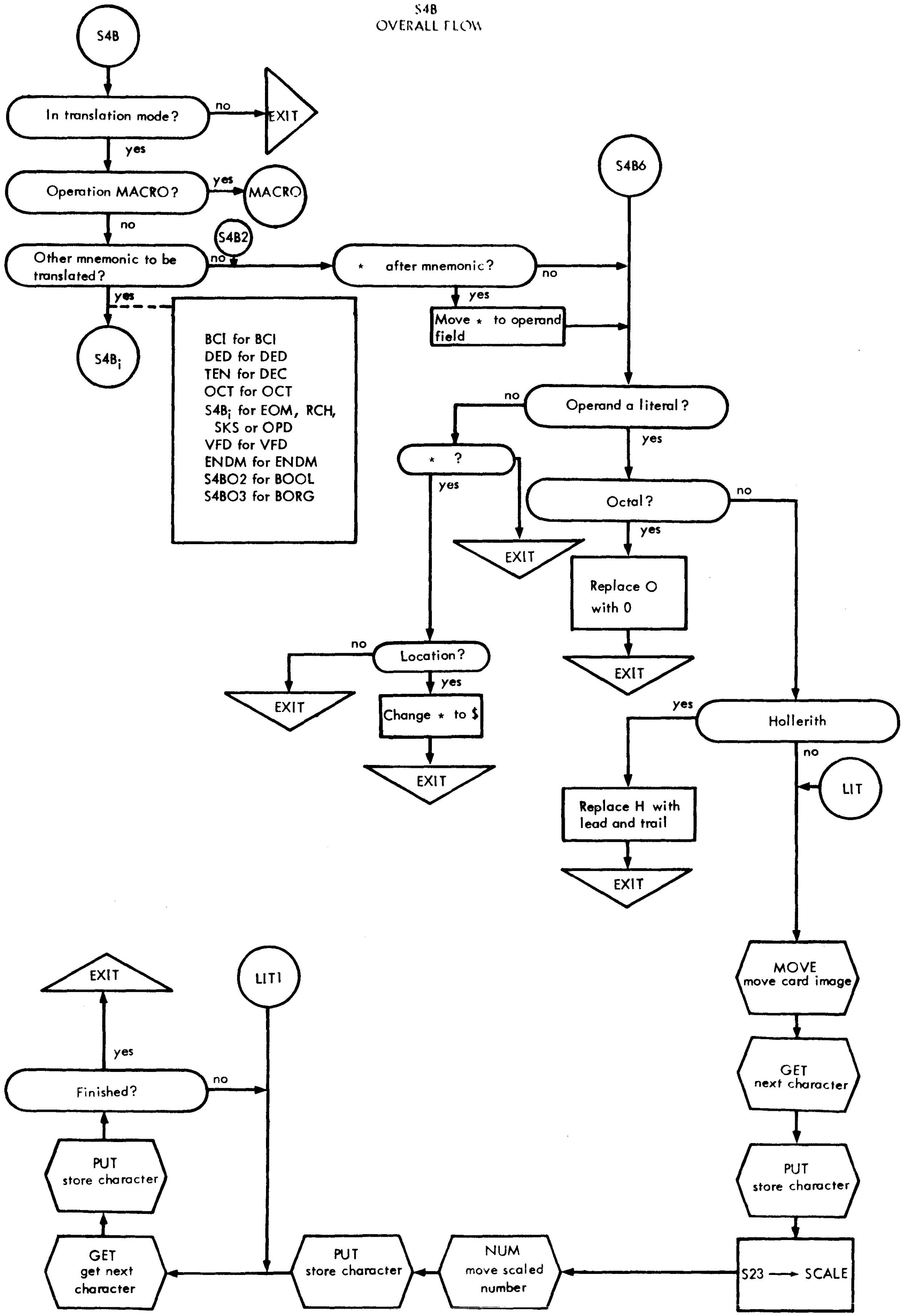
ACTION: PUT positions and stores the input character into the buffer at the location given by PUTW. The buffer location pointers are incremented for the next character.

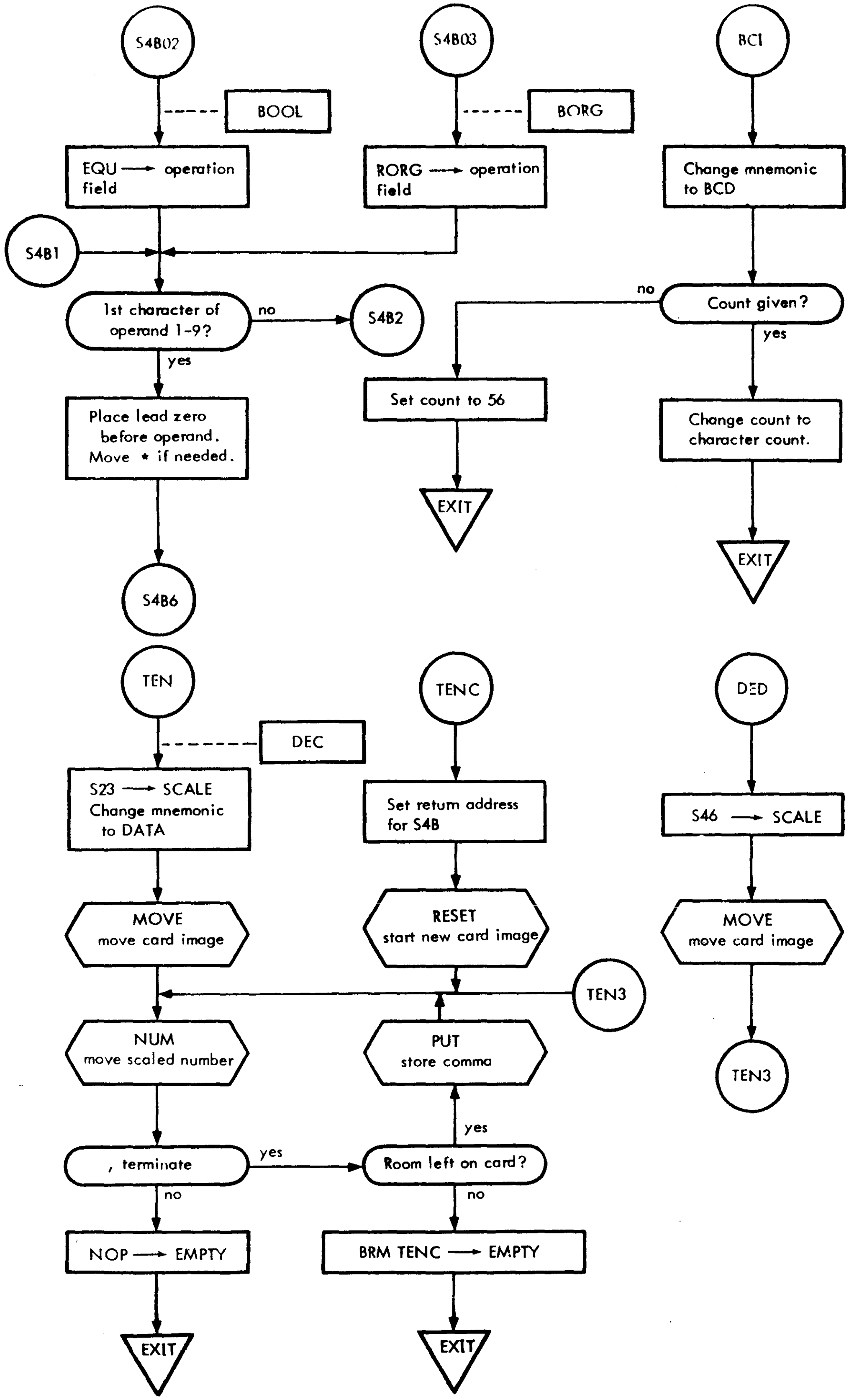
PROGRAMMING TECHNIQUES: Since PUT subtracts 60_8 from the character and adds the result to memory, it assumes the buffer has been cleared to blanks before being called. PUT is a relocatable routine assembled as part of S4B.

CALLING SEQUENCE: Character to A register
BRM PUT

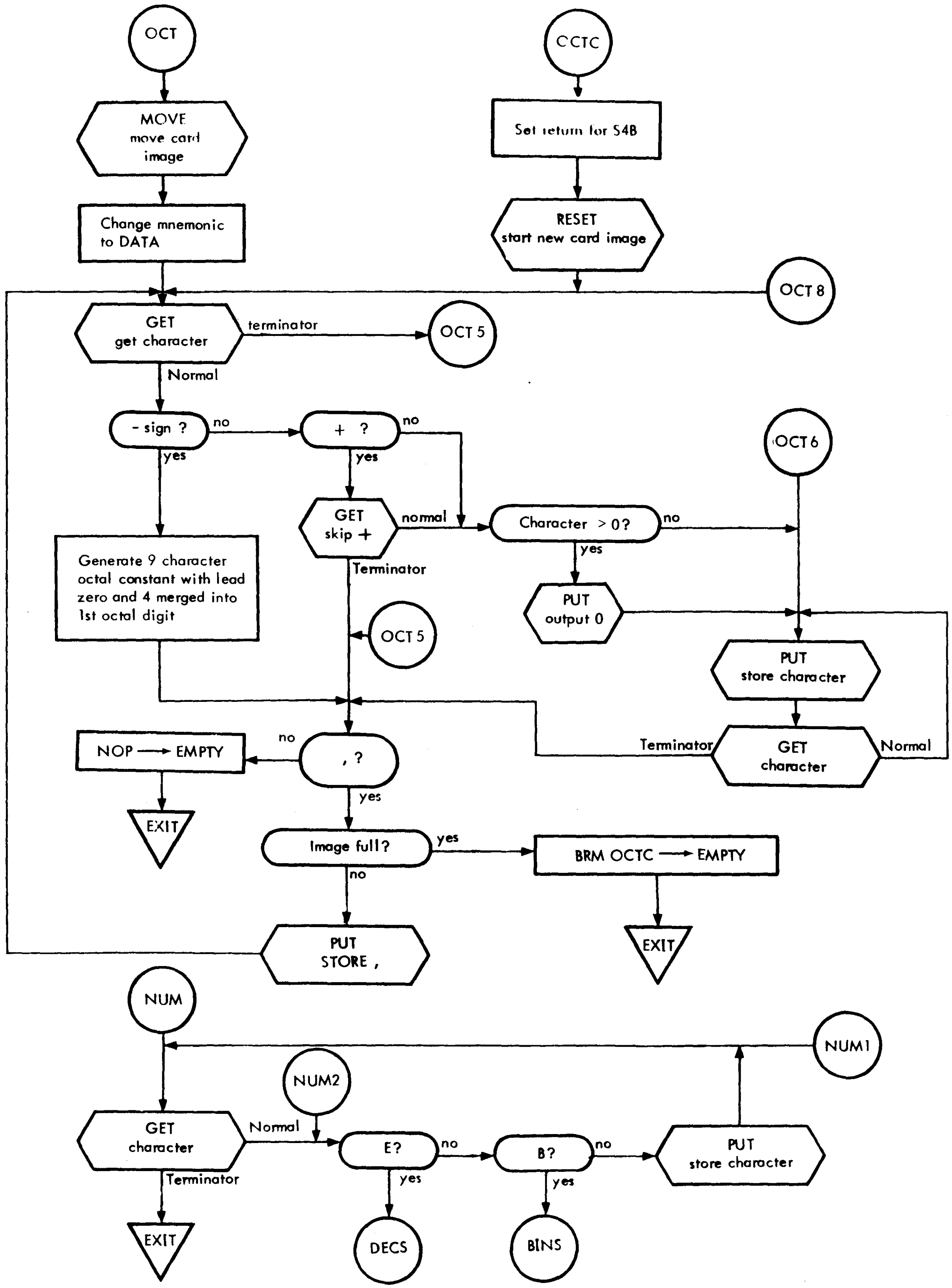
MEMORY REQUIREMENTS: 22_8 cells

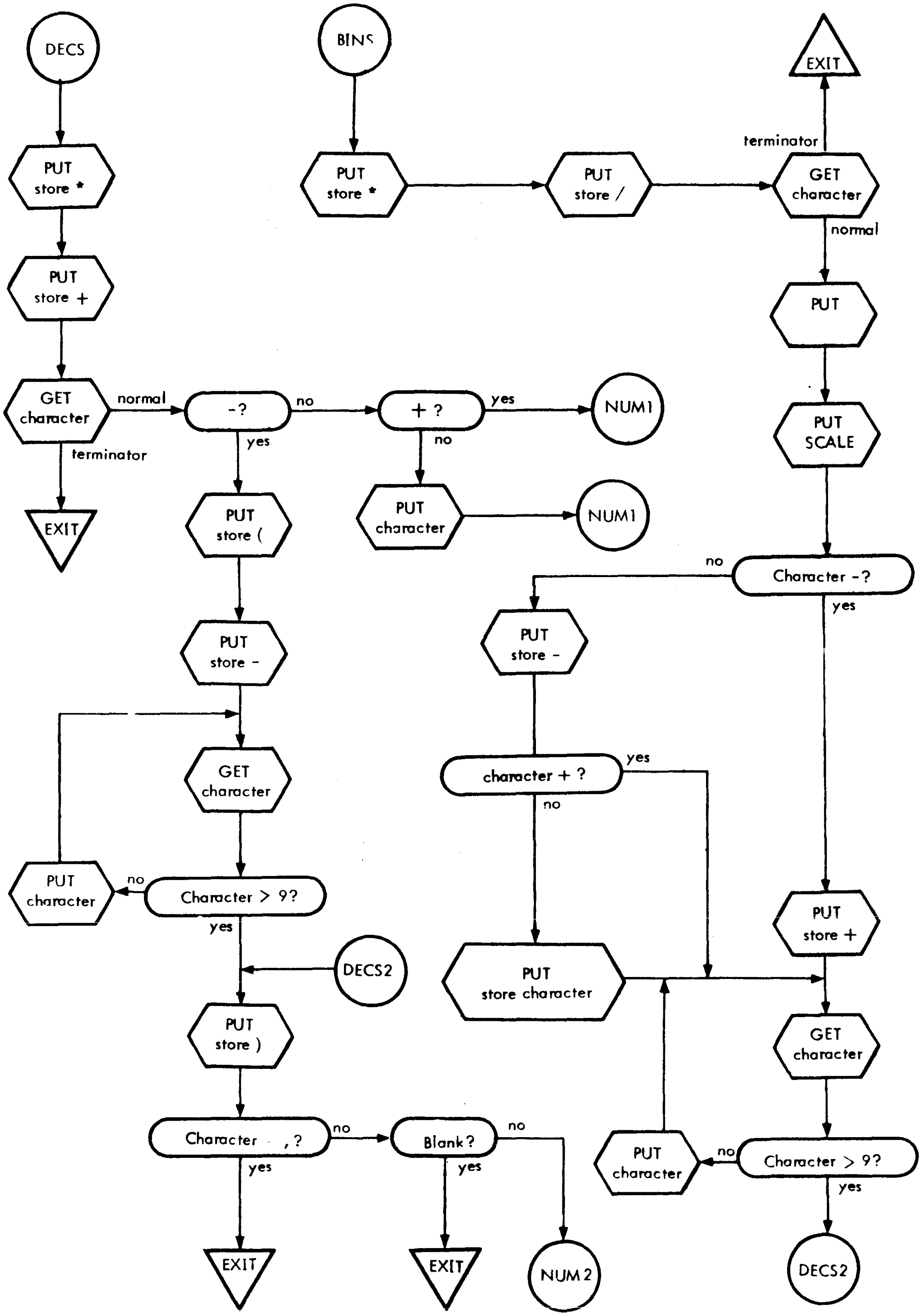
SUBROUTINES USED: None

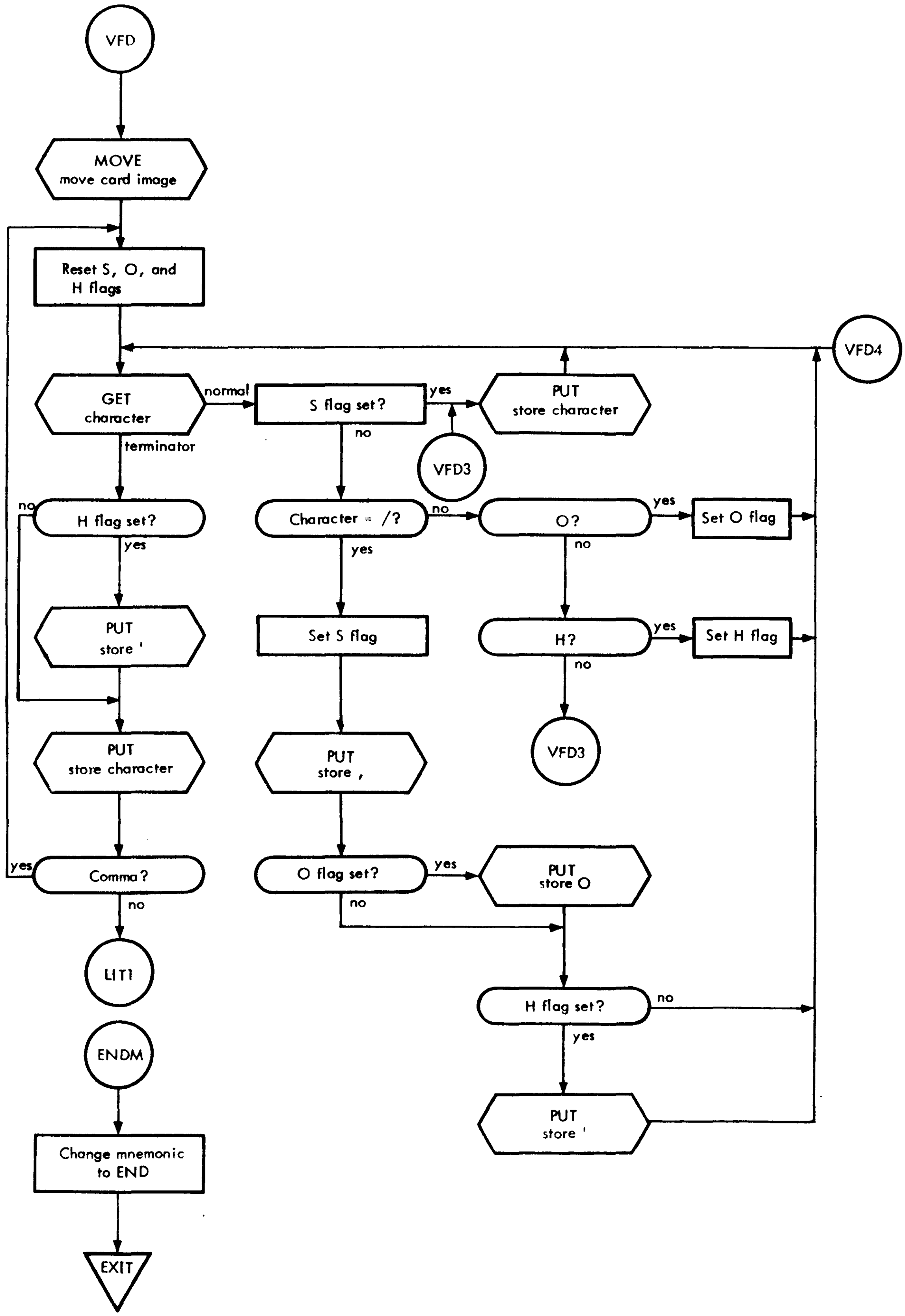

S4B
In translation mode?
no
EXIT
yes
Operation MACRO?
yes
MACRO
no
Other mnemonic to be translated?
no
S4B2
* after mnemonic?
no
yes
Move * to operand field
S4B6
yes
$S4B_i$
BCI for BCI
DED for DED
TEN for DEC
OCT for OCT
$S4B_i$ for EOM, RCH, SKS or OPD
VFD for VFD
ENDM for ENDM
S4BO2 for BOOL
S4BO3 for BORG
Operand a literal?
no
yes
* ?
yes
EXIT
Octal?
no
yes
Replace O with 0
EXIT
Location?
no
yes
EXIT
Change * to $
EXIT
Hollerith
yes
no
Replace H with lead and trail
EXIT
LIT
MOVE
move card image
GET
next character
PUT
store character
S23 ⟶ SCALE
NUM
move scaled number
PUT
store character
LIT1
GET
get next character
PUT
store character
Finished?
no
yes
EXIT

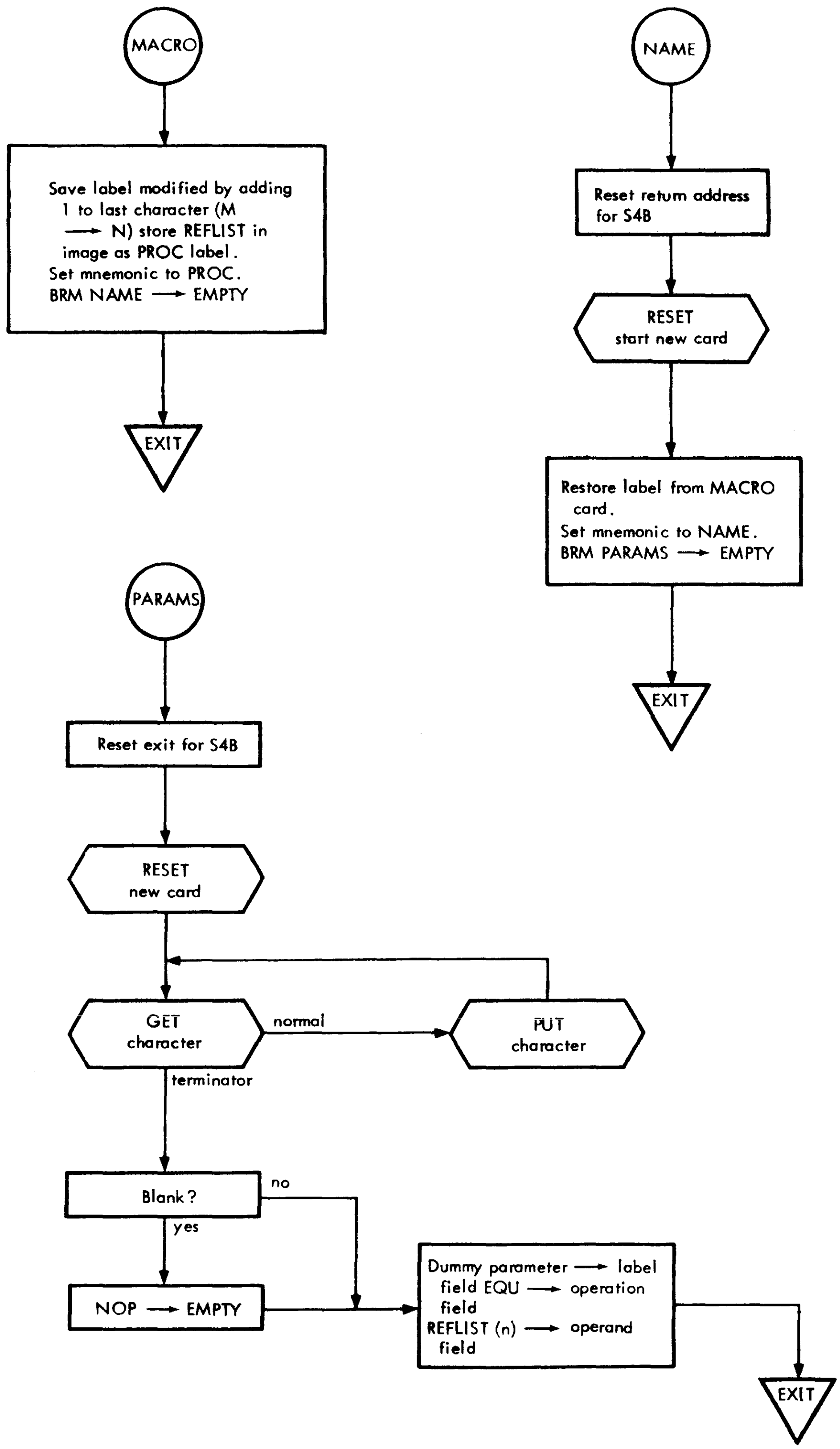

S4B02
BOOL
EQU → operation field
S4B03
BORG
RORG → operation field
BCI
Change mnemonic to BCD
S4B1
1st character of operand 1-9?
no
yes
S4B2
Count given?
no
yes
Set count to 56
Change count to character count.
Place lead zero before operand. Move * if needed.
EXIT
EXIT
S4B6
TEN
DEC
S23 → SCALE Change mnemonic to DATA
TENC
Set return address for S4B
DED
S46 → SCALE
MOVE move card image
RESET start new card image
MOVE move card image
TEN3
NUM move scaled number
PUT store comma
TEN3
yes
, terminate
yes
Room left on card?
no
no
NOP → EMPTY
BRM TENC → EMPTY
EXIT
EXIT

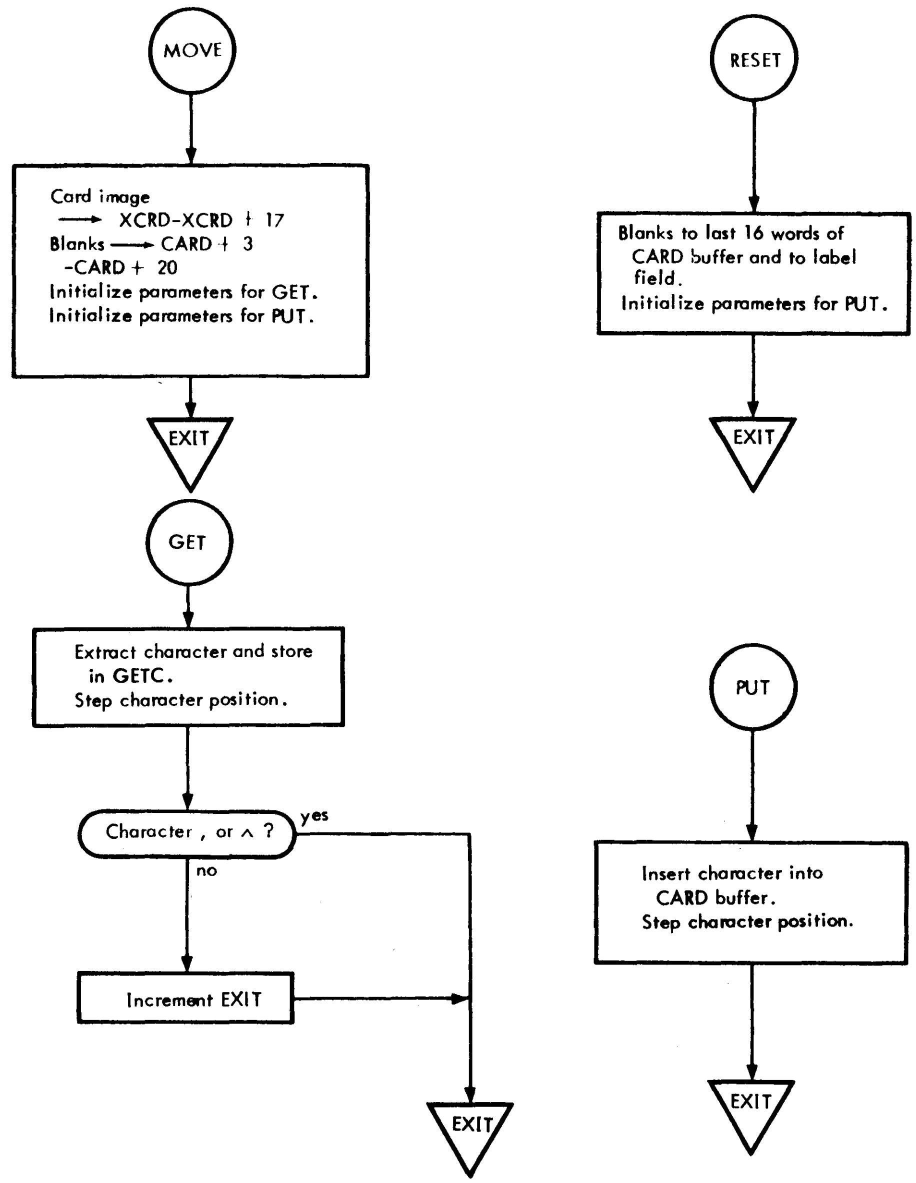
OCTC AND NUM ROUTINES




DECS
PUT store *
PUT store +
GET character
normal
terminator
EXIT
-?
no
yes
+ ?
yes
no
NUM1
PUT store (
PUT character
NUM1
PUT store -
GET character
Character > 9?
no
yes
PUT character
DECS2
PUT store)
Character ,?
no
yes
Blank?
no
yes
EXIT
EXIT
NUM 2
BINS
PUT store *
PUT store /
GET character
terminator
EXIT
normal
PUT
PUT SCALE
Character -?
no
yes
PUT store -
character + ?
yes
no
PUT store character
PUT store +
GET character
Character > 9?
no
yes
PUT character
DECS2


VFD
MOVE
move card image
Reset S, O, and H flags
VFD4
GET
character
normal
S flag set?
yes
PUT
store character
no
terminator
VFD3
no
H flag set?
yes
Character = /?
no
O?
yes
Set O flag
yes
no
PUT
store '
Set S flag
H?
yes
Set H flag
no
PUT
store character
PUT
store ,
VFD3
yes
Comma?
no
O flag set?
yes
PUT
store O
no
LIT1
H flag set?
no
yes
ENDM
PUT
store '
Change mnemonic to END
EXIT


MACRO
Save label modified by adding 1 to last character (M ⟶ N) store REFLIST in image as PROC label. Set mnemonic to PROC. BRM NAME ⟶ EMPTY
EXIT
NAME
Reset return address for S4B
RESET start new card
Restore label from MACRO card. Set mnemonic to NAME. BRM PARAMS ⟶ EMPTY
EXIT
PARAMS
Reset exit for S4B
RESET new card
GET character
normal
PUT character
terminator
Blank?
no
yes
NOP ⟶ EMPTY
Dummy parameter ⟶ label field EQU ⟶ operation field REFLIST (n) ⟶ operand field
EXIT


MOVE
Card image
⟶ XCRD-XCRD + 17
Blanks ⟶ CARD + 3
-CARD + 20
Initialize parameters for GET.
Initialize parameters for PUT.
EXIT
RESET
Blanks to last 16 words of
CARD buffer and to label
field.
Initialize parameters for PUT.
EXIT
GET
Extract character and store
in GETC.
Step character position.
Character , or ∧ ?
yes
no
Increment EXIT
EXIT
PUT
Insert character into
CARD buffer.
Step character position.
EXIT

SDS SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Initialize I/O control cells and copy symbolic corrections (MON1)

PURPOSE: To initialize the file maintenance and I/O device handling control words of MSCONTRL, to copy symbolic corrections to scratch tape X2 if needed, and to set the delay loop timing for paper tape reading.

ACTION: MON1 examines each entry of the unit assignment table (UAT) referenced by the assembly system and each entry in the MSFNC control word to determine which I/O functions and devices are to be used for the run. If a function is requested, MON1 inserts an entry into a control cell of the following format:

Standard I/O Control Word

contents	M	C	U	A
bits	0	1 3	4 9	10 23

where: M is decimal/binary mode flag

C is channel designation

U is unit number

A is address of routine to perform function

The above control words are found in MSCONTRL from MONBO through MONLF. In addition, the control flags USI through USO in MSCONTRL are set by MON1 in the following format:

Standard I/O Control Flag

contents	M	C	U	Code
bits	0	1 3	4 9	10 23

where: M is decimal/binary mode flag

C is channel

ACTION (cont.)

U is unit

Code is
- 0 for no operation
- 1 for cards
- 2 for paper tape
- 3 for magnetic tape

If the contents of the cell is zero, no request has been made for this input or output function.

The Standard I/O control word for a disc file is

NR	A
0 – 9	10 – 23

where

NR is a file number

A is address of I/O linkage routine to perform operation.

If both encoded and symbolic inputs are requested on the same device, MON1 copies the symbolic corrections to scratch tape X2 and changes the control cells for symbolic input in MSCONTRL. If corrections are not copied, MON1 stores a NOP over the rewind call in ENCODER at SETMO. MON1 also sets DELAY for the paper tape read routine depending on the type of machine as determined by executing a shift instruction.

PROGRAMMING TECHNIQUES:

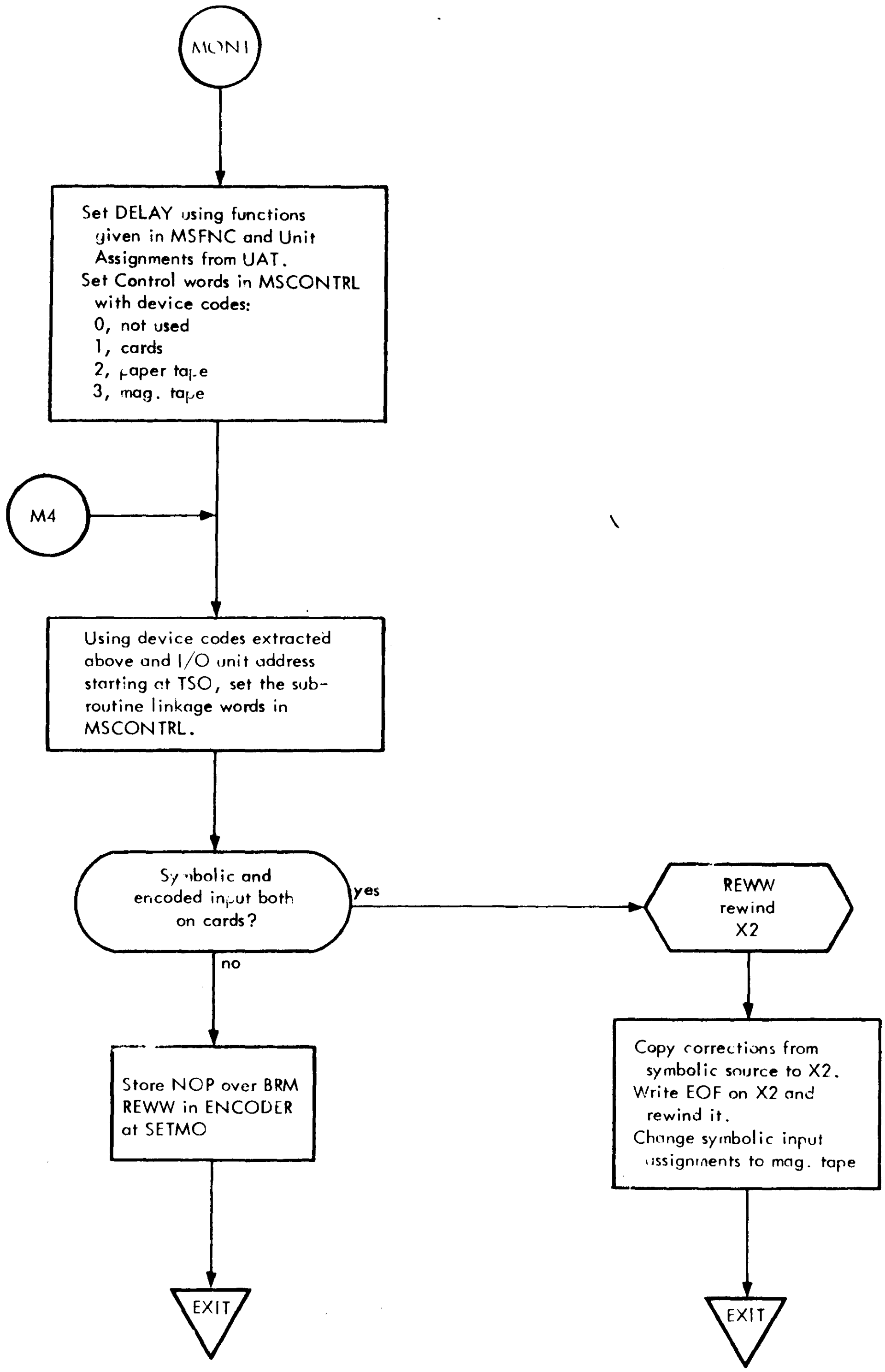
MON1 is dependent on the ordering of the control cells in MSCONTRL, the order of the UAT, and the order of parameters in MSFNC. MON1 is a relocatable routine and is the last relocatable routine loaded with ENCODER. Since MON1 is an initialization routine, ENCODER does not preserve it and overlays MON1 with tables.

CALLING SEQUENCE:

```
BRM    MON1
```

MEMORY REQUIREMENTS:

404_8 cells (all reusable after MON1 has been executed)


MON1
Set DELAY using functions given in MSFNC and Unit Assignments from UAT.
Set Control words in MSCONTRL with device codes:
0, not used
1, cards
2, paper tape
3, mag. tape
M4
Using device codes extracted above and I/O unit address starting at TSO, set the subroutine linkage words in MSCONTRL.
Symbolic and encoded input both on cards?
yes
no
REWW rewind X2
Store NOP over BRM REWW in ENCODER at SETMO
Copy corrections from symbolic source to X2. Write EOF on X2 and rewind it. Change symbolic input assignments to mag. tape
EXIT
EXIT

PREA
SRNK

C

SUBROUTINES USED:

Routine associated with symbolic input (CRDH†, RDPT†, and RMTB††)
WMTB†† REWW††
EFMT††

† These routines are described under ENCODER.

††These routines are described under MSCONTRL.

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Process standard procedure file (PREA or PREASM)

PURPOSE: To define directives, process standard procedures, and reformat dictionary entries and to establish the byte table in preparation for assembling programs.

ACTION: PREASM acts as the link between ENCODER and the assembler programs. ENCODER builds a series of tables during the encoding process in order to translate the symbolic data into compressed encoded information. These tables are inadequate for the assembly program for several reasons.

1. The tables are too extravagant of space. ENCODER needs a 3-word table to find and define unique dictionary entries efficiently; once the dictionary is defined, however, a 1-word pointer to each entry will suffice very well to interpret the encoded text.

2. The dictionary is in the wrong format. The assembly programs will need to make relatively few references to the actual dictionary entries for a byte if they can know the type of information the byte represents.

3. The data is incomplete. ENCODER processes only the user program; in order to complete the assembly operation the assembly routines must also have at their disposal definitions of the directives and standard procedures referenced by the user's program.

PREASM first reads the dictionary from the standard procedures file on the systems tape and then, using the tables and communication cells left by ENCODER, defines all unique bytes in much the same manner as does ENCODER. For each entry in the standard procedure dictionary PREASM makes a 1-word entry in an equivalence table, ETAB, which allows the translation of byte numbers from the standard procedures text to the equivalent

ACTION: (cont.)

byte numbers in the user's program. In a similar manner PREASM defines the directive bytes from dummy dictionary entries assembled as part of PREASM.

This done, PREASM has no further need for the balanced tree search table CPO. The next step is to convert the dictionary into two tables, part being the dictionary characters themselves in packed format and the remainder a 1-word pointer to the character position of the lead dictionary character for the byte. The pointer word also contains the number of characters in the dictionary and the code indicating type of entry.

The dictionary characters are stored in ascending order starting at the location given in DTAB whose address is sufficiently large to allow the largest segment of the assembly system to be loaded below it. The byte table (BTAB) is stored in descending locations starting just above QBOOT at the upper end of memory.

Once the dictionary has been compressed and the byte table has been established, PREASM defines the directives by entering them in the symbol table just below BTAB; these are also stored in descending order.

The text of the standard procedures file are now read, and those procedures to which reference has been made in the user's program are stored in the sample storage area just above the dictionary. Those NAMEs which have been referred to in the user's program are defined by entering NAME items in the symbol table.

When all records from the standard procedures file have been processed, PREASM calls the tape loader routine to load SHRINK.

PROGRAMMING TECHNIQUES:

PREASM is a relocatable program originated at location 1350_8. This leaves sufficient room below PREASM for the resident routines and the communication cells established by ENCODER. PREASM is assembled in two parts

PROGRAMMING TECHNIQUES: (cont.)

and converted to an absolute program by loading the two segments with POPs between them and then punching out an absolute program from the contents of core.

PREASM determines the location of QBOOT and hence the available table space by examining cell 1 which contains the instruction BRU QBOOT, established by MONARCH.

Since the length of ENCODER and S4B combined is larger than PREASM, the tables generated by ENCODER are sufficiently above the end of PREASM to allow room for the equivalence table, ETAB, below them. Should the number of bytes in the standard procedures deck increase sharply or the size of PREASM relative to ENCODER increase sharply, this may not be the case; then ETAB will have to be moved or the origin of the ENCODER tables increased.

As noted above, there are a few words of communication between ENCODER and PREASM. These cells are addressed by absolute addresses within PREASM. PREASM has two communication links with the assembler routines in addition to the tables noted above. These cells, PACKL and LITAB, indicate to the assembler programs the ending locations of the procedure sample and the symbol or item table, respectively. The words are the first two locations in PREASM.

Processing of Standard Procedure Sample

The encoding technique used in the META-SYMBOL assembly system allows for a monotonically increasing byte size. The byte size is incremented whenever the byte represented by the current size is zero. Because procedure NAME lines are not normally saved in the procedure sample area and because the number of NAMEs associated with a procedure may be large, it is possible to have the byte size incremented several times between the end of the PROC line and the first line following the procedure names. Unless the byte size for the PROC line is set to reflect this hidden increase in byte size, the processing of lines of code from the procedure sample area will

PROGRAMMING TECHNIQUES: (cont.)

degenerate to nothing. This single underlying phenomenon will be apparent through the following discourse on sample processing.

NAMEs of procedures are not defined when they appear inside a nested procedure but rather the NAME lines are moved to the procedure sample and the NAMEs defined when the outer procedure is referenced.

Each new line of procedure sample is processed starting at the location LINE. The line is read by calling TEX and then scanning from left to right. The label is saved at LBL. The operation code is obtained and tested to see if it is a directive. If the line is not a directive, control goes to LIN3. If it is a directive, control goes to PRO for a PROC, FUN for a function, NAM for a NAME, or SEND for an END. All other directives go to LIN3. A directive branch table is used to determine the type of directive. Processing stops when an end-of-file is detected.

At LIN3 the line is moved to the sample storage area if the previous line was moved there. If the line is not inside a procedure, or if it is inside a procedure but no NAMEs have been defined for the procedure, the line is ignored.

If the line is the first line following a procedure NAME line, and at least one NAME of the procedure has been referred to by the user's program, the starting location for the procedure is determined and placed in the NAME items saved for this PROC. The procedure line is moved to sample storage followed by the current line, and a flag is set indicating the sample is being saved.

Processing the PROC and FUNC lines. The detection of PROC or FUNC lines results in a count being incremented to indicate the level of procedure nesting. If the PROC or FUNC is not nested, the line is moved to a buffer, PRBYTS, for later insertion into the sample storage area, and a flag is set to indicate if the sample is procedure or function. If the PROC is nested, control goes to LIN3 to be processed like any other line.

PROGRAMMING TECHNIQUES: (cont.)

Processing the NAME line. When a NAME is detected, a test is made to determine if a PROC or FUNC line has been encountered; if one has not, the line is ignored. If the NAME appears in nested sample, it is treated like any other line by transferring control to LIN3A. If the NAME appears in the user's sample, the count of NAMEs saved is incremented and a NAME item inserted in the symbol table. If it does not appear in the user's program, the line is ignored. When inserting NAME items into the symbol table, the NAMEs associated with a procedure are linked so that once the procedure origin has been established it may readily be inserted in all the NAME items. The value associated with the NAME is obtained by calling VAL.

Processing the END lines. When an END directive is detected, the program determines whether the END follows a PROC or FUNC. If not, it is ignored; if it does, the nested procedure count is decremented. If this is the END of an outer PROC, the sample processing flag is turned off and a test is made to see if any NAMEs have been defined. If the sample is being saved, control goes to LIN3A; if not, the line is ignored. If an END is detected within nested PROC, sample control goes to LIN3 after decrementing the nested procedure count.

The following modifications have been incorporated within the RAD MONARCH version of META-SYMBOL:

1. The input buffer has been moved to correspond with the locations used in the Basic RAD Loader routine.
2. Calculation of top of memory and UAT entries is changed slightly because of the larger resident monitor.
3. The initialization of the RDTP routine is bypassed since the system is RAD-resident.

4. The SCTP routine is overlaid with a call to the SCTP routine in the Basic RAD loader.

5. The RDTP routine is overlaid with a call to the RDTP routine in the Basic RAD loader.

6. The DONE code to load the next core overlay simply calls the RAD loader since it searches for the $\Delta 2$ name records anyway, and the special skipping of encoded procedure decks on the system file is therefore avoided.

CALLING SEQUENCE: PREASM is called by the tape loader when the latter executes the transfer address in the last record of the PREASM program file.

MEMORY REQUIREMENTS: Variable, but at least 8192_{10} words of core. PREASM, when it has exhausted its working storage area, calls the ABORT routine to write an error message and return control to the monitor.

SUBROUTINES USED:

TRAIL[†]	GCW	GTCHR
SRCH[†]	GTB	DPDIV
NSRT[†]	GBW	GPDC
ABORT[††]	TEX	PI(RDPD)
GBC[†††]	INC	FETCH
VAL	MRKBYT	PACK
MVPRC	CNVRT	RDTP
MOVE		

[†] These routines are the same as those described under ENCODER except that they are assembled as part of PREASM.

[††] This routine is described under MSCONTRL.

[†††] No flow diagram provided.

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 612001

IDENTIFICATION: Process standard procedure file (PREA or PREASM)

PURPOSE: To define directives, process standard procedures, and reformat dictionary entries and to establish the byte table in preparation for assembling programs.

ACTION: PREASM acts as the link between ENCODER and the assembler programs. ENCODER builds a series of tables during the encoding process in order to translate the symbolic data into compressed encoded information. These tables are inadequate for the assembly program for several reasons.

1. The tables are too extravagant of space. ENCODER needs a 3-word table to find and define unique dictionary entries efficiently; once the dictionary is defined, however, a 1-word pointer to each entry will suffice very well to interpret the encoded text.

2. The dictionary is in the wrong format. The assembly programs will need to make relatively few references to the actual dictionary entries for a byte if they can know the type of information the byte represents.

3. The data is incomplete. ENCODER processes only the user program; in order to complete the assembly operation the assembly routines must also have at their disposal definitions of the directives and standard procedures referenced by the user's program.

PREASM first reads the dictionary from the standard procedures file on the systems tape and then, using the tables and communication cells left by ENCODER, defines all unique bytes in much the same manner as does ENCODER. For each entry in the standard procedure dictionary, PREASM makes a 1-word entry in an equivalence table, ETAB, which allows the translation of byte numbers from the standard procedures text to the equivalent

ACTION: (cont.)

byte numbers in the user's program. In a similar manner PREASM defines the directive bytes from dummy dictionary entries assembled as part of PREASM.

This done, PREASM has no further need for the balanced tree search table CPO. The next step is to convert the dictionary into two tables, part being the dictionary characters themselves in packed format and the remainder a 1-word pointer to the character position of the lead dictionary character for the byte. The pointer word also contains the number of characters in the dictionary and the code indicating type of entry.

The dictionary characters are stored in ascending order starting at the location given in DTAB whose address is sufficiently large to allow the largest segment of the assembly system to be loaded below it. The byte table (BTAB) is stored in descending locations starting at the top of core.

Once the dictionary has been compressed and the byte table has been established, PREASM defines the directives by entering them in the symbol table just below BTAB; these are also stored in descending order.

The text of the standard procedures file are now read, and those procedures to which reference has been made in the user's program are stored in the sample storage area just above the dictionary. Those NAMEs which have been referred to in the user's program are defined by entering NAME items in the symbol table.

When all records from the standard procedure file have been processed, PREASM calls the tape loader routine to load SHRINK.

PROGRAMMING TECHNIQUES:

Since the length of ENCODER and S4B combined is larger than PREASM, the tables generated by ENCODER are sufficiently above the end of PREASM to allow room for the equivalence table, ETAB, below them. Should the number of bytes in the standard procedures deck increase sharply or the size of

PROGRAMMING TECHNIQUES: (cont.)

PREASM relative to ENCODER increase sharply, this may not be the case; then ETAB will have to be moved or the origin of the ENCODER tables increased.

As noted before, there are a few words of communication between ENCODER and PREASM. These cells are addressed by absolute addresses within PREASM. PREASM has two communication links with the assembler routines in addition to the tables noted above. These cells, PACKL and LITAB, indicate to the assembler programs the ending locations of the procedure sample and the symbol or item table, respectively. The words are the first two locations in PREASM.

Processing of Standard Procedure Sample

The encoding technique used in the META-SYMBOL assembly system allows for a monotonically increasing byte size. The byte size is incremented whenever the byte represented by the current size is zero. Because procedure NAME lines are not normally saved in the procedure sample area and because the number of NAMEs associated with a procedure may be large, it is possible to have the byte size incremented several times between the end of the PROC line and the first line following the procedure names. Unless the byte size for the PROC line is set to reflect this hidden increase in byte size, the processing of lines of code from the procedure sample area will degenerate to nothing. This single underlying phenomenon will be apparent through the following discourse on sample processing.

NAMEs of procedures are not defined when they appear inside a nested procedure but rather the NAME lines are moved to the procedure sample and the NAMEs defined when the outer procedure is referenced.

Each new line of procedure sample is processed starting at the location LINE. The line is read by calling TEX and then scanning from left to right. The label is saved at LBL. The operation code is obtained and tested to see if it is a directive. If the line is not a directive, control goes to LIN3. If it is

PROGRAMMING TECHNIQUES: (cont.)

a directive, control goes to PRO for a PROC, FUN for a function, NAM for a NAME, or SEND for an END. All other directives go to LIN3. A directive branch table is used to determine the type of directive. Processing stops when an end-of-file is detected.

At LIN3 the line is moved to the sample storage area if the previous line was moved there. If the line is not inside a procedure, or if it is inside a procedure but no NAMEs have been defined for the procedure, the line is ignored.

If the line is the first line following a procedure NAME line, and at least one NAME of the procedure has been referred to by the user's program, the starting location for the procedure is determined and placed in the NAME items saved for this PROC. The procedure line is moved to sample storage followed by the current line, and a flag is set indicating the sample is being saved.

Processing the PROC and FUNC lines. The detection of PROC or FUNC lines results in a count being incremented to indicate the level of procedure nesting. If the PROC or FUNC is not nested, the line is moved to a buffer, PRBYTS, for later insertion into the sample storage area, and a flag is set to indicate if the sample is procedure or function. If the PROC is nested, control goes to LIN3 to be processed like any other line.

Processing the NAME line. When a NAME is detected, a test is made to determine if a PROC or FUNC line has been encountered; if one has not, the line is ignored. If the NAME appears in nested sample, it is treated like any other line by transferring control to LIN3A. If the NAME appears in the user's sample, the count of NAMEs saved is incremented and a NAME item inserted in the symbol table. If it does not appear in the user's program, the line is ignored. When inserting NAME items into the symbol table, the NAMEs associated with a procedure are linked so that once the procedure origin has been established it may readily be inserted in all the NAME items. The value associated with the NAME is obtained by calling VAL.

PROGRAMMING TECHNIQUES: (cont.)

Processing the END lines. When an END directive is detected, the program determines whether the END follows a PROC or FUNC. If not, it is ignored; if it does, the nested procedure count is decremented. If this is the END of an outer PROC, the sample processing flag is turned off and a test is made to see if any NAMEs have been defined. If the sample is being saved, control goes to LIN3A; if not, the line is ignored. If an END is detected within nested PROC, sample control goes to LIN3 after decrementing the nested procedure count.

The following modifications have been incorporated within the RAD MONARCH version of META-SYMBOL:

1. The input buffer has been moved to correspond with the locations used in the Basic RAD Loader routine.
2. Calculation of top of memory and UAT entries is changed slightly because of the larger resident monitor.
3. The initialization of the RDTP routine is bypassed since the system is RAD-resident.
4. The SCTP routine is overlaid with a call to the SCTP routine in the Basic RAD loader.
5. The RDTP routine is overlaid with a call to the RDTP routine in the Basic RAD loader.
6. The DONE code to load the next core overlay simply calls the RAD loader since it searches for the $\Delta 2$ name records anyway, and the special skipping of encoded procedure decks on the system file is therefore avoided.

CALLING SEQUENCE:

PREASM is called by the tape loader when the latter executes the transfer address in the last record of the PREASM program file.

MEMORY REQUIREMENTS: Variable, but at least 8192_{10} words of core. PREASM, when it has exhausted its working storage area, calls the ABORT routine to write an error message and return control to the monitor.

SUBROUTINES USED:

TRAIL[t]	GCW	GTCHR
SRCH[t]	GTB	DPDIV
NSRT[t]	GBW	GPDC
ABORT[tt]	TEX	PI(RDPD)
GBC[ttt]	INC	FETCH
VAL	MRKBYT	PACK
MVPRC	CNVRT	RDTP
MOVE		

---

[t] These routines are the same as those described under ENCODER except that they are assembled as part of PREASM.

[tt] This routine is described under MSCONTRL.

[ttt] No flow diagram provided.

ENTRY POINTS TO PREASM SUBROUTINES

Entry	Page Description	Page Flowchart	Entry	Page Description	Page Flowchart
CHNG1	3-117	3-141	NS4		3-152
CNV1	3-132	3-147	NS4A		3-153
CNV2	3-132	3-147	NS4B		3-153
CNV3	3-132	3-148	NS5		3-153
CNV6Z	3-132	3-148	NS6		3-153
CNVRT	3-132	3-147	NS7		3-153
CNVT	3-132	3-148	NS8		3-152
DONE		3-142	NS9		3-153
DPDIV	3-134	3-149	NS10		3-153
FETCH	3-137	3-150	NSRT		3-152
FUN		3-143	PACK	3-138	3-150
GBW	3-128	3-145	PI	3-136	3-150
GCW	3-126	3-145	PRE1	3-117	3-140
GPDC	3-135	3-150	PRE2	3-117	3-141
GTB	3-127	3-145	PRE5	3-117	3-140
GTCHR	3-133	3-149	PRE6	3-117	3-141
INC	3-130	3-146	PRE8	3-117	3-140
LIN1		3-142	PRE11	3-117	3-140
LIN2A		3-142	PRE12	3-117	3-142
LIN3		3-142	PREASM	3-117	3-140
LIN3A		3-142	PRO		3-143
LIN5		3-142	RDTP	3-139	3-150
LINE		3-142	RREAD	3-139	3-150
MO1	3-125	3-144	SAMP		3-140
MOVE	3-125	3-144	SEND		3-144
MRKBYT	3-131	3-146	SR1		3-154
MVPRC	3-125	3-144	SRCH		3-154
NA5A		3-143	TEX	3-129	3-146
NAM		3-143	TRAIL		3-154
NS3		3-152	VAL	3-124	3-144

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042C
Catalog No. 9300: 612C

IDENTIFICATION: Determine blank character string lengths (GBC)

PURPOSE: To determine the number of characters in a blank character string.

ACTION: GBC gets the location for the dictionary entry and then calls GTCHR to get the entry which is the number of blank characters. The count is placed in the A register and in BCNT.

PROGRAMMING TECHNIQUES: GBC is a relocatable routine assembled as part of PREASM.

CALLING SEQUENCE:

```
BRM    GBC
```

Byte table entry to B register

MEMORY REQUIREMENTS: 33_8 cells

SUBROUTINES USED: GTCHR

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 900 Series: 042016
9300: 612001

IDENTIFICATION: Evaluate numeric expressions on NAME lines (VAL)

PURPOSE: To evaluate numeric expressions and construct a numeric item which is used in setting the value associated with a procedure NAME.

ACTION: If the byte is not numeric, VAL returns via the nonnumeric exit. VAL sets the character count for the string and the dictionary location for the string. Next VAL calls CNVRT to convert the string to a binary constant. VAL then builds a numeric value item and places its length in the low order bits of the A register.

PROGRAMMING TECHNIQUES: VAL is a relocatable routine assembled as part of PREASM.

CALLING SEQUENCE:
Byte table entry to B register
BRM VAL
nonnumeric return
numeric item return

MEMORY REQUIREMENTS: 57_8 cells

SUBROUTINES USED: CNVRT

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 900 Series: 0420
9300: 6120

IDENTIFICATION: Move lines of sample to procedure storage (MVPRC and MOVE)

PURPOSE: To move a line of code which is part of a procedure definition to procedure sample storage area.

ACTION: MVPRC and MOVE are a common routine. Each of them causes a line of code to be moved from a buffer area to the sample storage area. MRKBYT is called to flag each byte moved so that it will be retained by SHRINK. As bytes are moved, the bite size is tested; if it increases above the byte size currently being used, the byte size used to save sample is increased. At the conclusion of the move SMPBIT is set to indicate the number of bits in the current sample word that have been used.

PROGRAMMING TECHNIQUES: MVPRC and MOVE are relocatable routines assembled as part of PREASM.

CALLING SEQUENCE:

BRM	MVPRC	to move PROC lines
BRM	MOVE	to move all other lines

MEMORY REQUIREMENTS: 65_8 cells

SUBROUTINES USED: MRKBYT

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 900 Series: 042016
9300: 612001

IDENTIFICATION: Obtain next byte table entry (GCW)

PURPOSE: To get the next byte value and byte table entry corresponding to it.

ACTION: GCW obtains the next byte value from BBUF and uses it to index the byte table. The byte table entry is loaded into the B register, and the negative of the byte value is left in the A and index registers.

PROGRAMMING TECHNIQUES: GCW is a relocatable routine assembled as part of PREASM.

CALLING SEQUENCE: BRM GCW

MEMORY REQUIREMENTS: 10_8 words

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 04201
Catalog No. 9300: 61200

IDENTIFICATION: Get the next byte value from the standard procedures file (GTB)

PURPOSE: To extract into the low order bits of the A register the value of the next byte of standard procedure text.

ACTION: GTB extracts from CHAD the next BSIZ bits of standard procedure text. If fewer than BSIZ bits of data remain in CHAD, GTB calls GBW to obtain the next word of input. If a zero byte is obtained, GTB takes 2^{BSIZ} as the value of the byte and steps BSIZ and the related mask BMSK. The byte value obtained is then converted to the equivalent user value by taking the corresponding entry from ETAB as the byte value. If the ETAB entry is greater than the mask SVBMS, the size indicator SVBSZ and the mask SVBMS are increased in size until SVBMS is as large or larger than the byte.

PROGRAMMING TECHNIQUES: GTB is a relocatable routine assembled as part of PREASM.

CALLING SEQUENCE: BRM GTB

MEMORY REQUIREMENTS: 61_8 cells

SUBROUTINES USED: GBW

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Get the next word of standard procedure text (GBW)

PURPOSE: To place into CHAD the next word of standard procedure test.

ACTION: GBW moves the next word of standard procedure text from the input buffer to CHAD. If the buffer is empty, GBW first calls the input routine PI (indirectly through RDPD) to read the next record from the standard procedure file.

PROGRAMMING TECHNIQUES: PI is indirectly addressed through cell RDPD. GBW is a relocatable routine assembled as part of PREASM.

CALLING SEQUENCE: BRM GBW

MEMORY REQUIREMENTS: 21_8 cells

SUBROUTINES USED: PI

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 04201
Catalog No. 9300: 61200

IDENTIFICATION: Obtain the next line of encoded text (TEX)

PURPOSE: To store the byte values for the next line of standard procedure text into consecutive cells starting at BBUF and to skip the comments on the line.

ACTION: TEX calls GTB to obtain the byte values from the input file which are then stored in BBUF. Bytes are moved until an end-of-line byte is encountered, at which point TEX calls INC to obtain comment characters until all comments have been skipped.

PROGRAMMING TECHNIQUES: TEX is a relocatable routine assembled as part of PREASM.

CALLING SEQUENCE: BRM TEX

MEMORY REQUIREMENTS: 23_8 cells

SUBROUTINES USED: GTB
INC

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Get comment characters (INC)

PURPOSE: To get the next comment character from the standard procedures file.

ACTION: INC sets a flag INCFG to cause GTB to suppress stepping of the byte sizes and masks. INC then saves the current byte size and mask and sets the byte size to 6. GTB is called to obtain the next six bits of encoded text, and the byte size and mask are restored.

PROGRAMMING TECHNIQUES: INC is a relocatable routine assembled as part of PREASM.

CALLING SEQUENCE: BRM INC

MEMORY REQUIREMENTS: 21_8 cells

SUBROUTINES USED: GTB

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 0420
Catalog No. 9300: 6120

IDENTIFICATION: Flag bytes to be saved (MRKBYT)

PURPOSE: MRKBYT marks bytes appearing in all lines of procedure sample, including name lines which are saved. The purpose for this flagging is to identify those bytes and only those bytes from the standard procedure file which are needed to process the user's program. SHRINK, when called, will purge all bytes from the dictionary and byte table which neither appear in the user's program or are marked as being needed. This marking is necessary since the appearance of a byte in the dictionary is unique, but the first reference to the byte may not be the instance that resulted in its being needed.

ACTION: MRKBYT sets bit 2 of the byte table entry for each byte in the buffer addressed by the contents of the A register.

PROGRAMMING TECHNIQUES: MRKBYT makes use of the fact that bit 2 of the byte table entry is not used. MRKBYT is a relocatable routine assembled as part of PREASM.

CALLING SEQUENCE:
Buffer location to A register
BRM MRKBYT

MEMORY REQUIREMENTS: 14_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Convert numeric strings to binary values (CNVRT)

PURPOSE: To convert numeric items to binary values.

ACTION: CNVRT converts numeric character strings to their binary value by successive multiplications of 8 or 10 (depending on the value of the first character). GTCHR is used to fetch the characters of the string. Results are left in PROD, PROD1, and PROD2. If the leading character is a dot, the number is converted to floating point by dividing the integer by the appropriate powers of 10 and calculating the exponent. The DPDIV routine is used to perform the divisions. All floating point fractions so calculated are left in normalized form.

PROGRAMMING TECHNIQUES: CNVRT is a relocatable routine assembled as part of PREASM.

CALLING SEQUENCE:

Number of characters in byte to SIZE
Character position of first character to CHAR
Memory location of dictionary word to DLOC
BRM CNVRT

MEMORY REQUIREMENTS: 170_8 cells

SUBROUTINES USED:

GTCHR
DPDIV

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 900 Series: 042
9300: 612

IDENTIFICATION: Extract characters from the packed dictionary (GTCHR)

PURPOSE: To get the next character of a dictionary entry to the A register.

ACTION: GTCHR loads the next character from the dictionary entry into the low order bits of the A register. The dictionary location of the next character as indicated by DLOC and CHAR is established, and SIZE is decremented.

PROGRAMMING TECHNIQUES: GTCHR is a relocatable routine assembled as part of PREASM.

CALLING SEQUENCE:

Character position in word to CHAR
Location of dictionary word to DLOC
Size of byte in characters to SIZE
BRM GTCHR
end of entry
normal exit

MEMORY REQUIREMENTS: 22_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION:	Perform double-precision divisions (DPDIV)
PURPOSE:	To divide the contents of the A and B registers by the contents of the location addressed by the index register and maintain maximum precision.
ACTION:	DPDIV divides the contents of the A and B registers by the single precision divisor addressed by the index register. The remainders are then divided and that remainder divided. The resulting quotient is normalized.
PROGRAMMING TECHNIQUES:	DPDIV assumes that both the dividend and divisor are normalized and leaves the results in the same format. DPDIV is a relocatable routine assembled as part of PREASM.
CALLING SEQUENCE:	Double-precision dividend to A and B registers Location of divisor to X register BRM DPDIV
MEMORY REQUIREMENTS:	36_8 cells
SUBROUTINES USED:	None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042
Catalog No. 9300: 612

IDENTIFICATION: Get characters from the input standard procedure dictionary (GPDC)

PURPOSE: To fetch the next dictionary character into the A register and PDCHR.

ACTION: GPDC extracts the next dictionary character from the input buffer into the low order bits of the A register and to PDCHR. If the buffer is empty, GPDC calls PI to read the next record of input from the standard procedures file. If the record read is not of type 1 (dictionary), GPDC returns through the end-of-dictionary exit.

PROGRAMMING TECHNIQUES: GPDC is a relocatable routine assembled as part of PREASM.

CALLING SEQUENCE:

```
BRM    GPDC
end-of-dictionary return
normal return
```

MEMORY REQUIREMENTS: 36_8 cells

SUBROUTINES USED: PI (indirectly addressed through RDPD).

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Obtain the next record from the standard procedures file (PI)

PURPOSE: To cause the next record to be read from the standard procedures deck and to extract the record type and length.

ACTION: PI calls RDTP to read the next record of standard procedures. PI extracts the record type and stores it in RT; next it extracts the record length and stores it minus 2 in PIWC for the GPDC routine.

PROGRAMMING TECHNIQUES: PI is a relocatable routine addressed through cell RDPD and is assembled as part of PREASM.

CALLING SEQUENCE:

```
BRM    PI
   or
BRM    *RDPD
```

MEMORY REQUIREMENTS: 13_8 cells

SUBROUTINES USED: RDTP

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042
Catalog No. 9300: 612

IDENTIFICATION: Get a character from the unpacked dictionary in core (FETCH)

PURPOSE: To extract the next character from the unpacked dictionary as constructed by ENCODER.

ACTION: FETCH gets the next dictionary character as addressed by FCHWD and FCHSH into the low-order bits of the A and B registers.

PROGRAMMING TECHNIQUES: FETCH is a relocatable routine assembled as part of PREASM.

CALLING SEQUENCE: BRM FETCH

MEMORY REQUIREMENTS: 20_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Pack characters into consecutive bytes of core (PACK)

PURPOSE: To merge a character in the low-order bits of the A register into the next byte position of memory as addressed by PCKSH and PACKL.

ACTION: PACK positions the character using the contents of PCKSH and merges the character into the locations addressed by PACKL. PCKSH and PACKL are incremented as needed. PCKNT indicates the number of characters stored in the current location.

PROGRAMMING TECHNIQUES: PACK is a relocatable routine assembled as part of PREASM.

CALLING SEQUENCE:
Character to A register
BRM PACK

MEMORY REQUIREMENTS: 24_8 words

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 04
Catalog No. 9300: 61

IDENTIFICATION: Read records from the standard procedure file of the system tape (RDTP)

PURPOSE: To read the next record of standard procedures.

ACTION: RDTP uses a WIM loop to read records of up to 40_{10} words each from the systems tape. Records are read in the binary mode. A read error results in the tape being backspaced and the record reread. Up to ten rereads are executed before the routine halts. Stepping from the halt causes the record to be accepted as read.

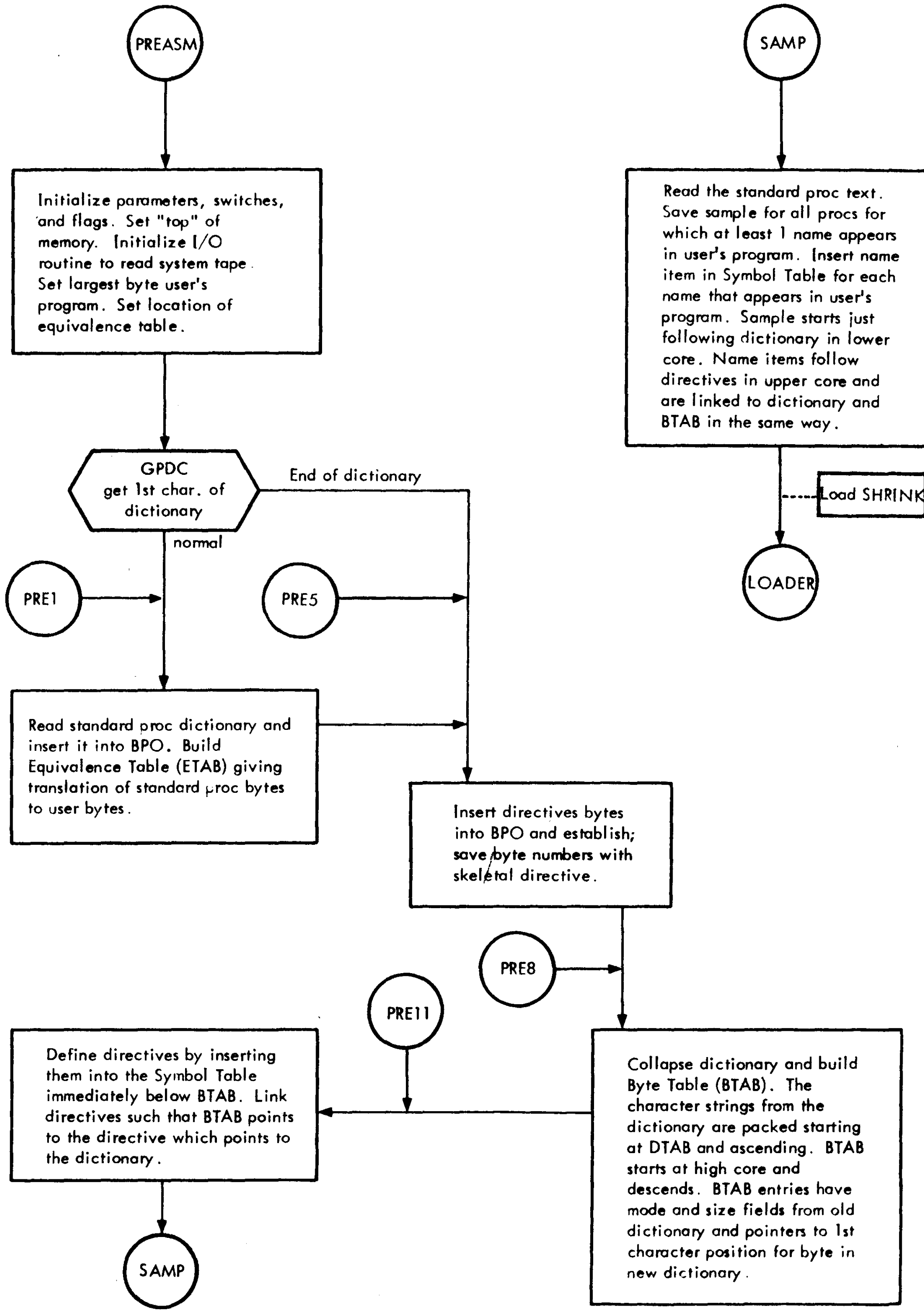
PROGRAMMING TECHNIQUES: RDTP is initialized as to unit and channel assignments by the initialization code of PREASM. RDTP is a relocatable routine assembled as part of PREASM.

CALLING SEQUENCE: BRM RDTP

MEMORY REQUIREMENTS: 32_8 cells

SUBROUTINES USED: None

NOTE: In the RAD MONARCH system this routine is overlaid by a call upon the system file read routine, RDTP, which is contained within the Basic RAD Loader.

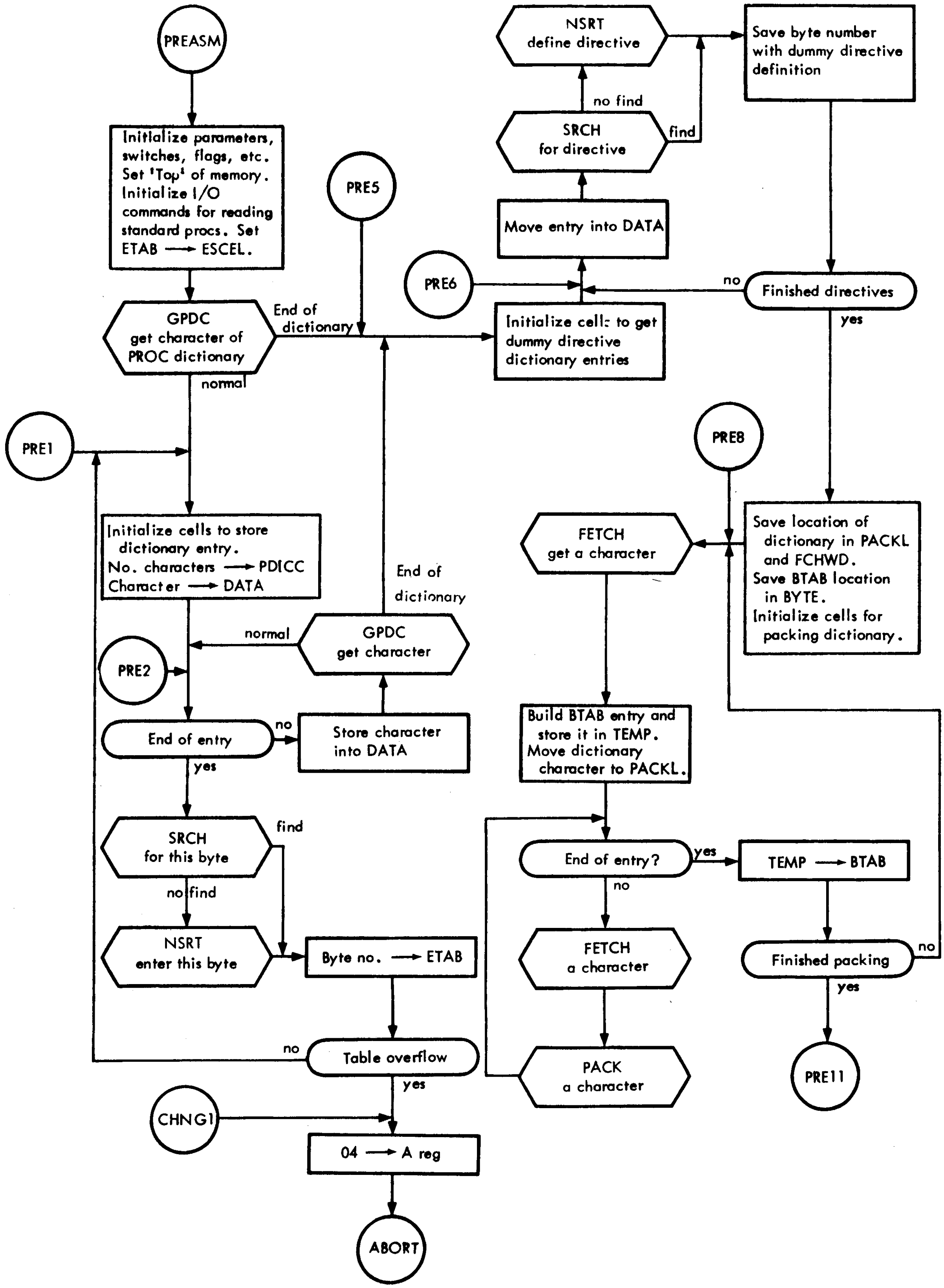

PREASM
Initialize parameters, switches, and flags. Set "top" of memory. Initialize I/O routine to read system tape. Set largest byte user's program. Set location of equivalence table.
GPDC get 1st char. of dictionary
End of dictionary
normal
PRE1
PRE5
Read standard proc dictionary and insert it into BPO. Build Equivalence Table (ETAB) giving translation of standard proc bytes to user bytes.
Insert directives bytes into BPO and establish; save byte numbers with skeletal directive.
PRE8
PRE11
Collapse dictionary and build Byte Table (BTAB). The character strings from the dictionary are packed starting at DTAB and ascending. BTAB starts at high core and descends. BTAB entries have mode and size fields from old dictionary and pointers to 1st character position for byte in new dictionary.
Define directives by inserting them into the Symbol Table immediately below BTAB. Link directives such that BTAB points to the directive which points to the dictionary.
SAMP
SAMP
Read the standard proc text. Save sample for all procs for which at least 1 name appears in user's program. Insert name item in Symbol Table for each name that appears in user's program. Sample starts just following dictionary in lower core. Name items follow directives in upper core and are linked to dictionary and BTAB in the same way.
Load SHRINK
LOADER

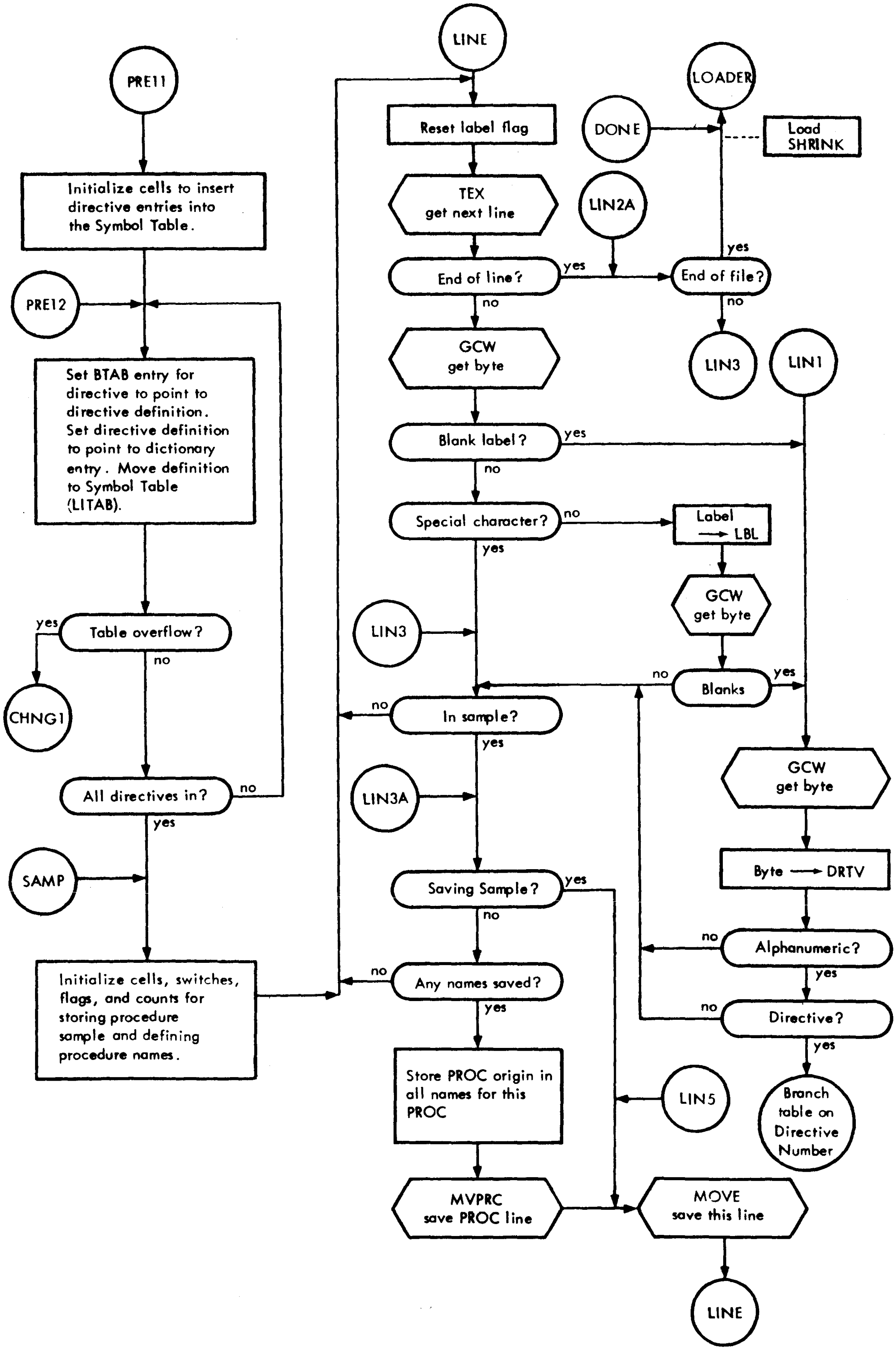
SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

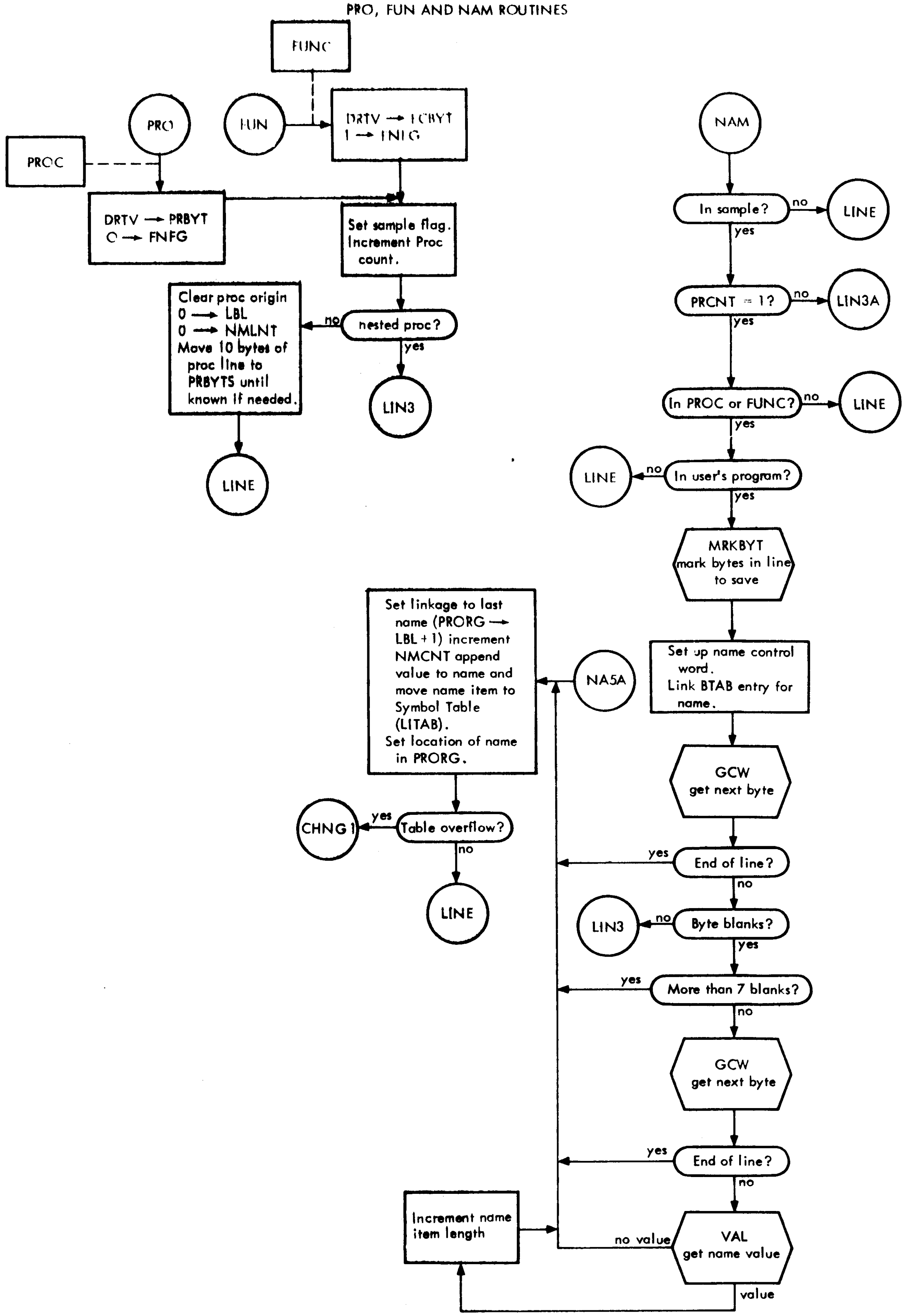
IDENTIFICATION:	Find △2 record on system file (SCTP)*
PURPOSE:	To scan the procedure on the system file for the next △2 record.
ACTION:	SCTP initializes some parameters to the RDTP routine, then calls RDTP to fetch the first word of the next record in the system file. When a △2 record is encountered the routine exits.
CALLING SEQUENCE:	BRM SCTP
MEMORY REQUIREMENTS:	9 cells
SUBROUTINES USED:	RDTP

*In the RAD system this routine is overlaid by a call to the SCTP routine in the Basic RAD Loader.

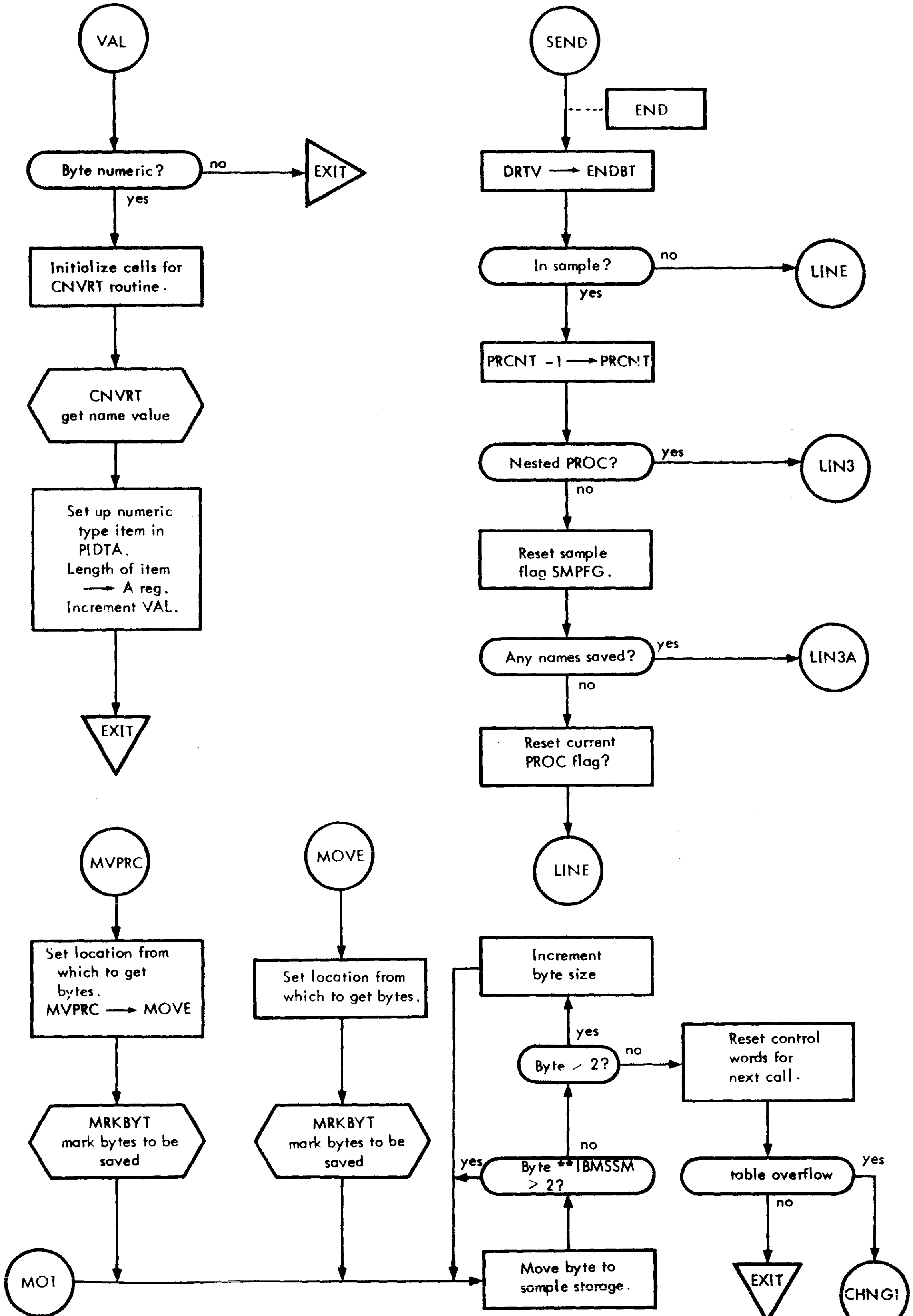

PREASM
Initialize parameters, switches, flags, etc. Set 'Top' of memory. Initialize I/O commands for reading standard procs. Set ETAB ⟶ ESCEL.
GPDC get character of PROC dictionary
End of dictionary
normal
PRE5
PRE6
PRE1
Initialize cells to store dictionary entry. No. characters ⟶ PDICC Character ⟶ DATA
End of dictionary
normal
GPDC get character
PRE2
End of entry
no
yes
Store character into DATA
SRCH for this byte
find
no find
NSRT enter this byte
Byte no. ⟶ ETAB
no
Table overflow
yes
CHNG1
04 ⟶ A reg
ABORT
NSRT define directive
no find
SRCH for directive
find
Move entry into DATA
Initialize cells to get dummy directive dictionary entries
Save byte number with dummy directive definition
no
Finished directives
yes
PRE8
Save location of dictionary in PACKL and FCHWD. Save BTAB location in BYTE. Initialize cells for packing dictionary.
FETCH get a character
Build BTAB entry and store it in TEMP. Move dictionary character to PACKL.
End of entry?
yes
no
TEMP ⟶ BTAB
FETCH a character
PACK a character
Finished packing
no
yes
PRE11

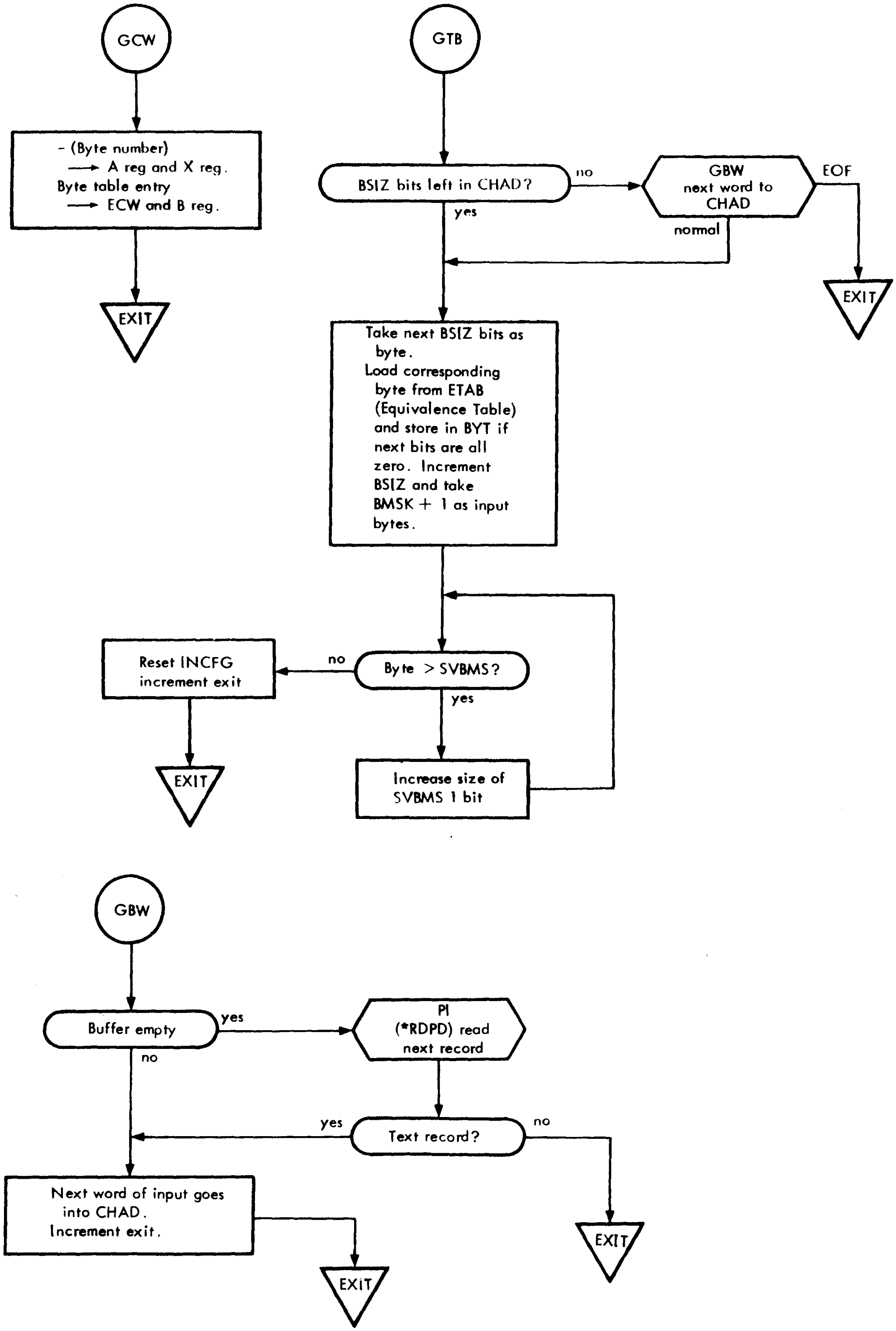

PRE11
Initialize cells to insert directive entries into the Symbol Table.
PRE12
Set BTAB entry for directive to point to directive definition. Set directive definition to point to dictionary entry. Move definition to Symbol Table (LITAB).
Table overflow?
yes
no
CHNG1
All directives in?
no
yes
SAMP
Initialize cells, switches, flags, and counts for storing procedure sample and defining procedure names.
LINE
Reset label flag
TEX
get next line
End of line?
yes
no
GCW
get byte
Blank label?
yes
no
Special character?
no
yes
LIN3
In sample?
no
yes
LIN3A
Saving Sample?
yes
no
Any names saved?
no
yes
Store PROC origin in all names for this PROC
MVPRC
save PROC line
LOADER
DONE
Load SHRINK
LIN2A
End of file?
yes
no
LIN3
LIN1
Label
→ LBL
GCW
get byte
Blanks
no
yes
GCW
get byte
Byte → DRTV
Alphanumeric?
no
yes
Directive?
no
yes
Branch table on Directive Number
LIN5
MOVE
save this line
LINE

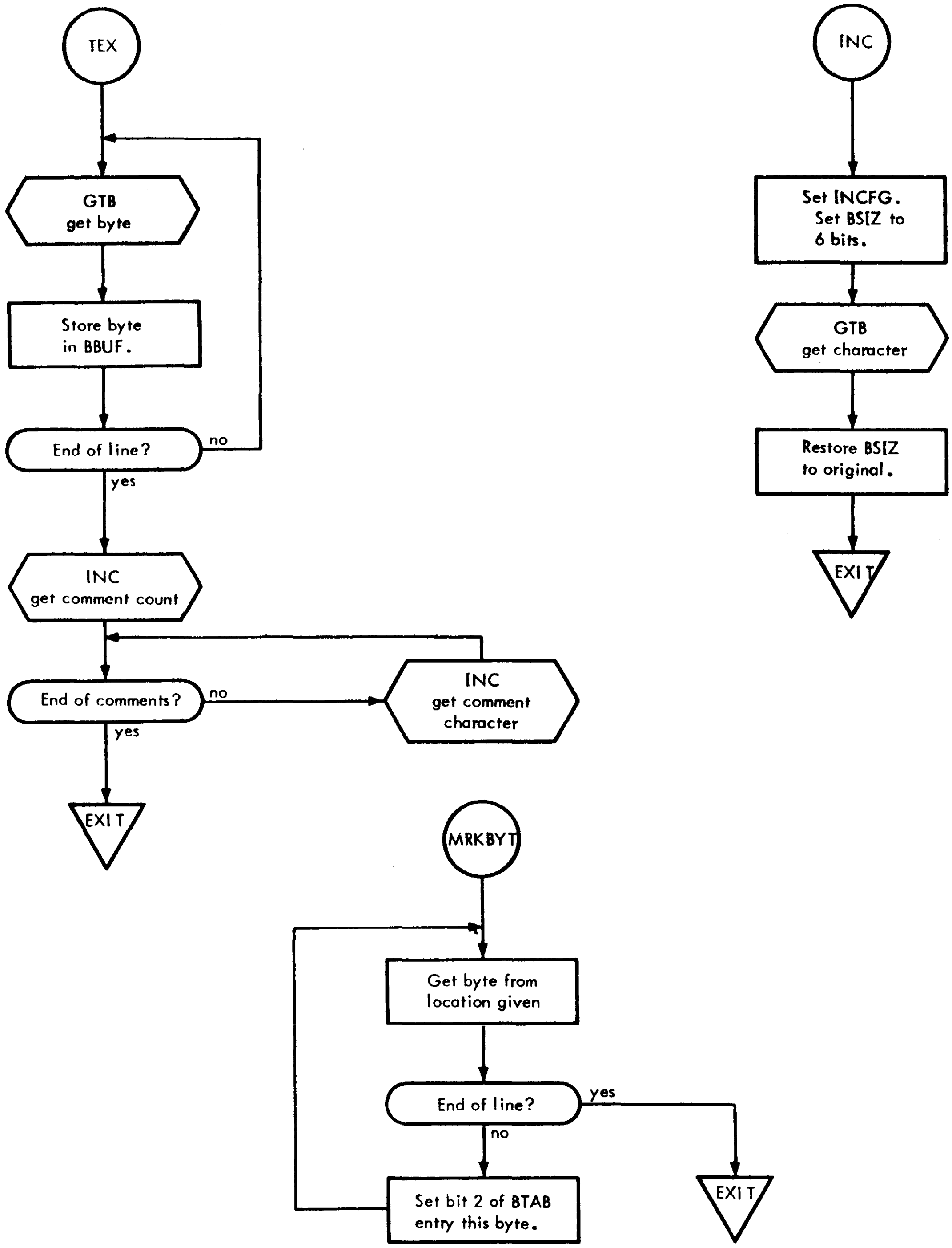
PRO, FUN AND NAM ROUTINES

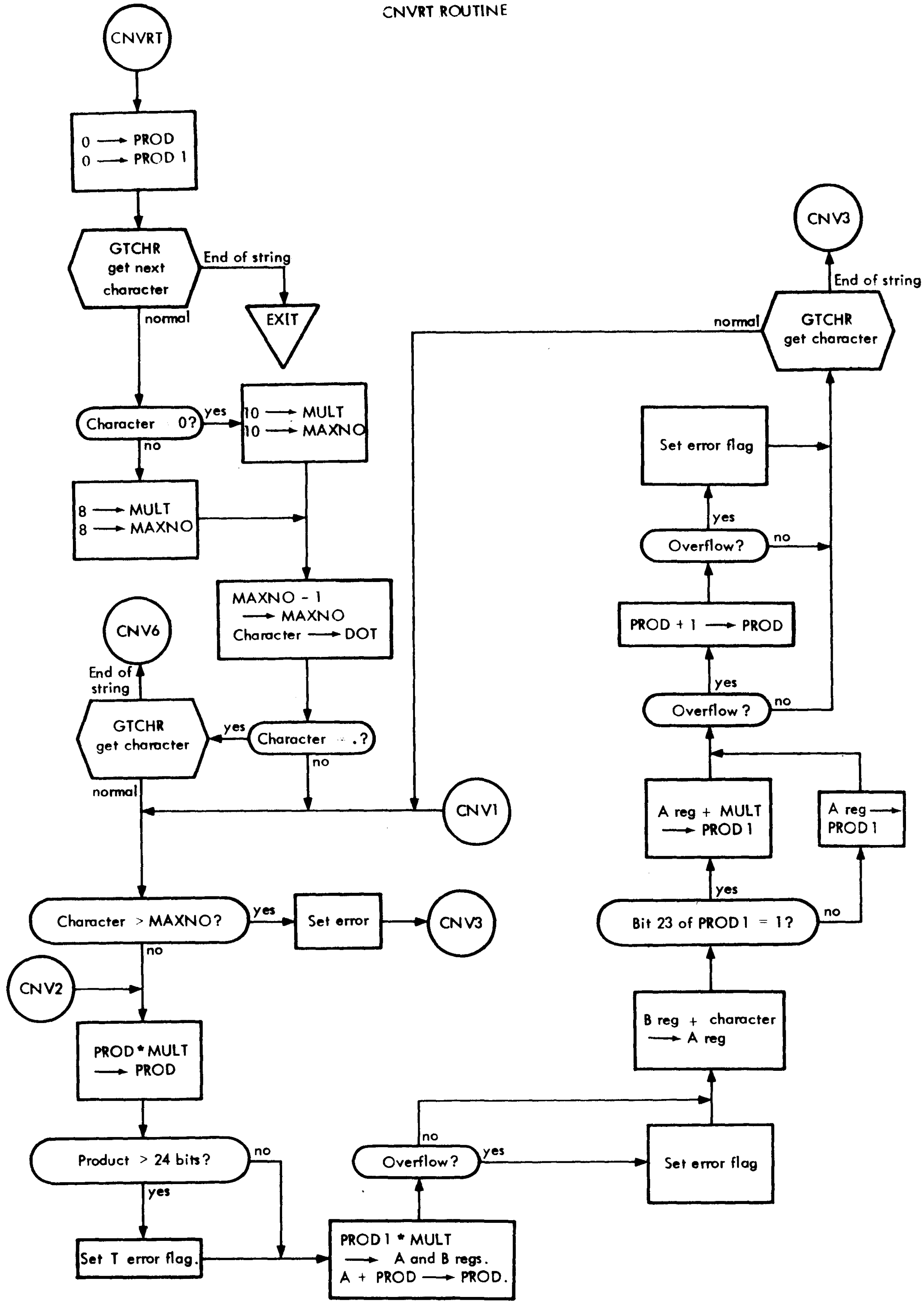


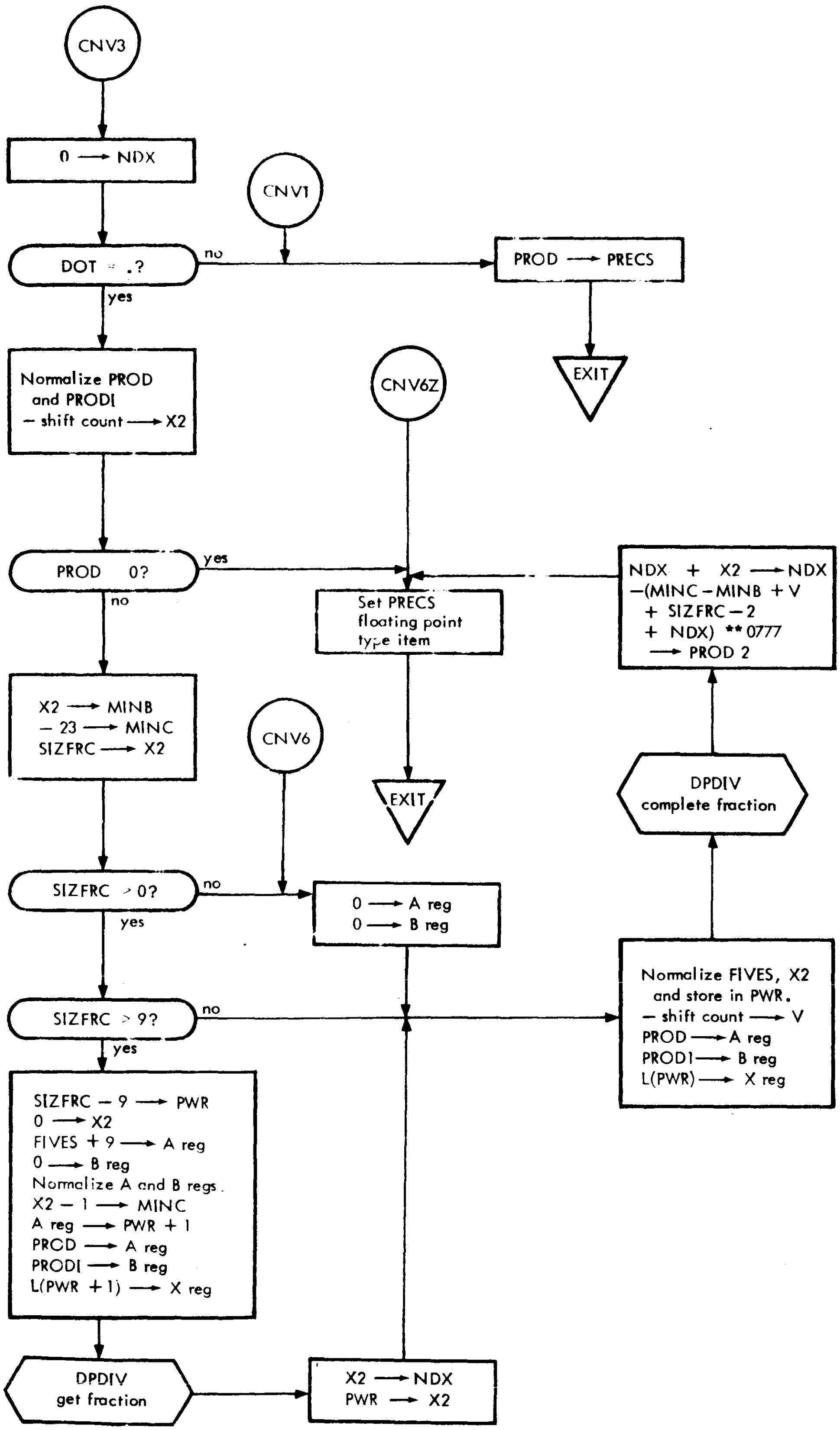
VAL, SEND, MVPRC AND MOVE ROUTINES

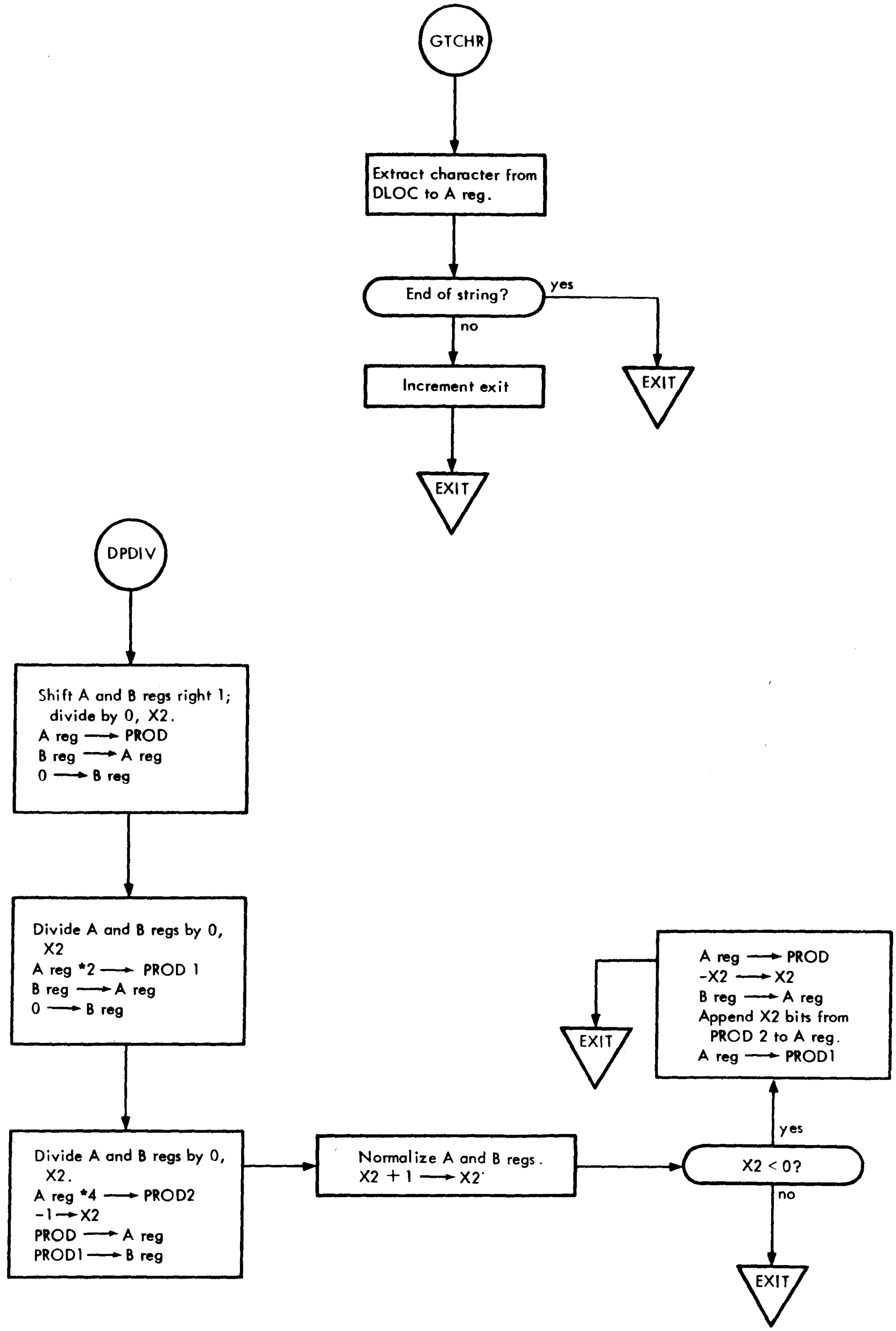



GCW
- (Byte number)
⟶ A reg and X reg.
Byte table entry
⟶ ECW and B reg.
EXIT
GTB
BSIZ bits left in CHAD?
no
yes
GBW
next word to
CHAD
EOF
normal
EXIT
Take next BSIZ bits as byte.
Load corresponding byte from ETAB (Equivalence Table) and store in BYT if next bits are all zero. Increment BSIZ and take BMSK + 1 as input bytes.
Reset INCFG
increment exit
no
Byte > SVBMS?
yes
EXIT
Increase size of
SVBMS 1 bit
GBW
Buffer empty
yes
no
PI
(*RDPD) read
next record
yes
Text record?
no
Next word of input goes
into CHAD.
Increment exit.
EXIT
EXIT

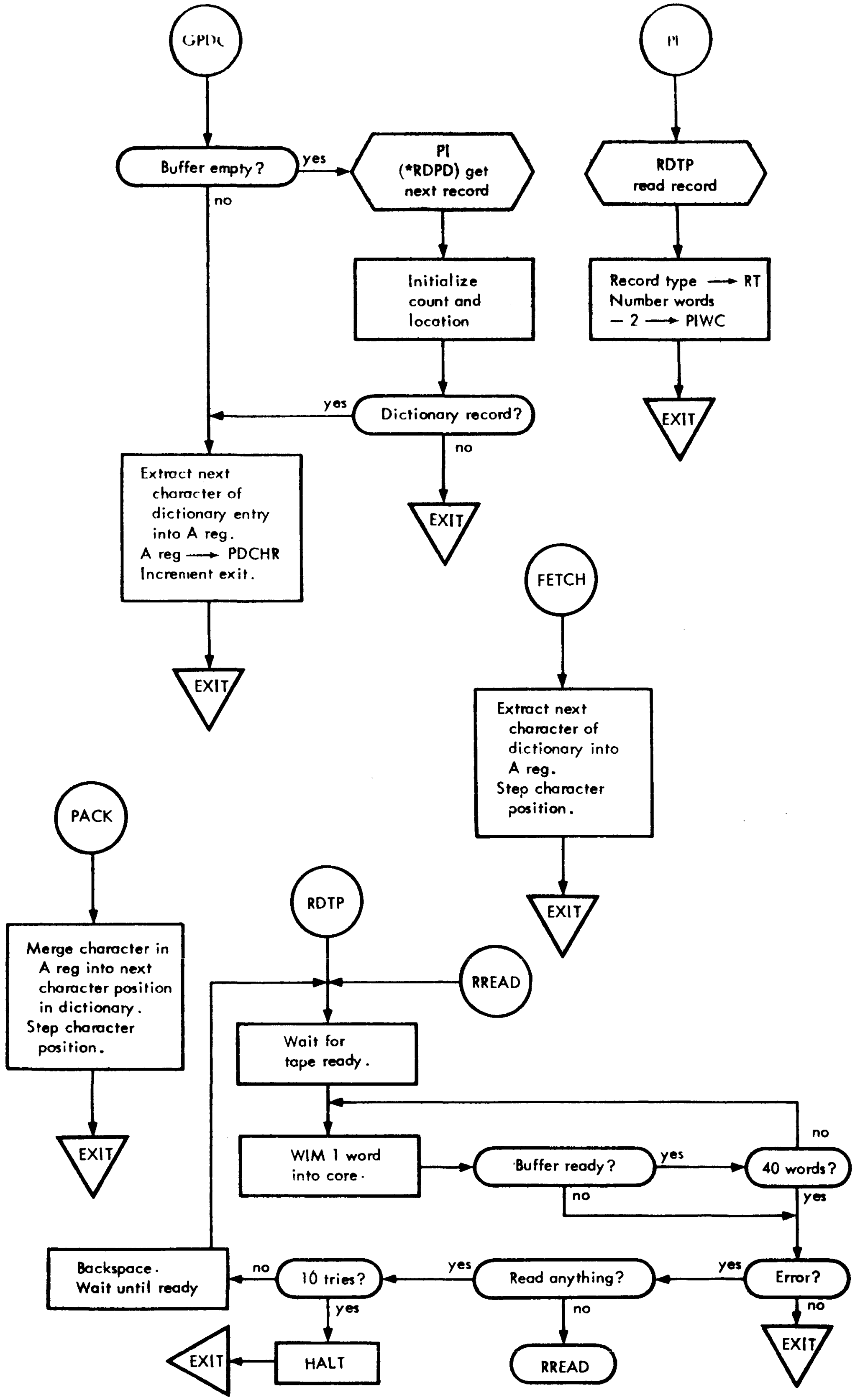

TEX
GTB
get byte
Store byte
in BBUF.
End of line?
no
yes
INC
get comment count
End of comments?
no
yes
INC
get comment
character
EXIT
INC
Set INCFG.
Set BSIZ to
6 bits.
GTB
get character
Restore BSIZ
to original.
EXIT
MRKBYT
Get byte from
location given
End of line?
yes
no
Set bit 2 of BTAB
entry this byte.
EXIT


CNVRT
0 → PROD
0 → PROD 1
GTCHR get next character
End of string
EXIT
normal
Character 0?
yes
10 → MULT
10 → MAXNO
no
8 → MULT
8 → MAXNO
MAXNO - 1 → MAXNO
Character → DOT
CNV6
End of string
GTCHR get character
yes
Character .?
no
normal
CNV1
Character > MAXNO?
yes
Set error
CNV3
no
CNV2
PROD * MULT → PROD
Product > 24 bits?
no
yes
Set T error flag.
PROD 1 * MULT → A and B regs.
A + PROD → PROD.
Overflow?
no
yes
Set error flag
B reg + character → A reg
Bit 23 of PROD 1 = 1?
yes
no
A reg + MULT → PROD 1
A reg → PROD 1
Overflow ?
yes
no
PROD + 1 → PROD
Overflow?
yes
no
Set error flag
GTCHR get character
normal
End of string
CNV3


CNV3
0 → NDX
CNV1
DOT = .?
no
yes
PROD → PRECS
EXIT
Normalize PROD and PROD1 – shift count → X2
CNV6Z
PROD 0?
yes
no
Set PRECS floating point type item
NDX + X2 → NDX –(MINC – MINB + V + SIZFRC – 2 + NDX) ** 0777 → PROD 2
X2 → MINB
– 23 → MINC
SIZFRC → X2
CNV6
EXIT
DPDIV complete fraction
SIZFRC > 0?
no
yes
0 → A reg
0 → B reg
SIZFRC > 9?
no
yes
Normalize FIVES, X2 and store in PWR.
– shift count → V
PROD → A reg
PROD1 → B reg
L(PWR) → X reg
SIZFRC – 9 → PWR
0 → X2
FIVES + 9 → A reg
0 → B reg
Normalize A and B regs.
X2 – 1 → MINC
A reg → PWR + 1
PROD → A reg
PROD1 → B reg
L(PWR + 1) → X reg
DPDIV get fraction
X2 → NDX
PWR → X2


GTCHR
Extract character from DLOC to A reg.
End of string?
yes
no
Increment exit
EXIT
EXIT
DPDIV
Shift A and B regs right 1; divide by 0, X2.
A reg → PROD
B reg → A reg
0 → B reg
Divide A and B regs by 0, X2
A reg *2 → PROD 1
B reg → A reg
0 → B reg
Divide A and B regs by 0, X2.
A reg *4 → PROD2
-1 → X2
PROD → A reg
PROD1 → B reg
Normalize A and B regs.
X2 + 1 → X2
X2 < 0?
yes
no
A reg → PROD
-X2 → X2
B reg → A reg
Append X2 bits from PROD 2 to A reg.
A reg → PROD1
EXIT
EXIT

RDTP, PACK, GPDC, PI and FETCH routines



NSRT ROUTINE (DEFINITIONS)

Let α denote some byte entry in the table. Then:

$L(\alpha)$ is the pointer from α to a lesser item
$G(\alpha)$ is the pointer from α to a greater item
$K(\alpha)$ is the key of α.

$B(\alpha)$ is the balance of α.
$B(\alpha) = 0$ denotes balance
$B(\alpha) = 1$ denotes heavy in the greater chain
$B(\alpha) = 2$ denotes heavy in the lesser chain

$D(\alpha)$ is the direction followed from α in searching for an item.
$D(\alpha) = 0$ denotes lesser chain taken
$D(\alpha) = 1$ denotes greater chain taken

X denotes current item to insert.

$F(\alpha)$ denotes the item following α on the search path taken.

$Q(\alpha)$ denotes the item following α on the path other than that taken.
U denotes the last point of imbalance on the last search path.

MO denotes the last point examined by SRCH.

$M(\beta)$ and $N(\beta)$ are defined such that

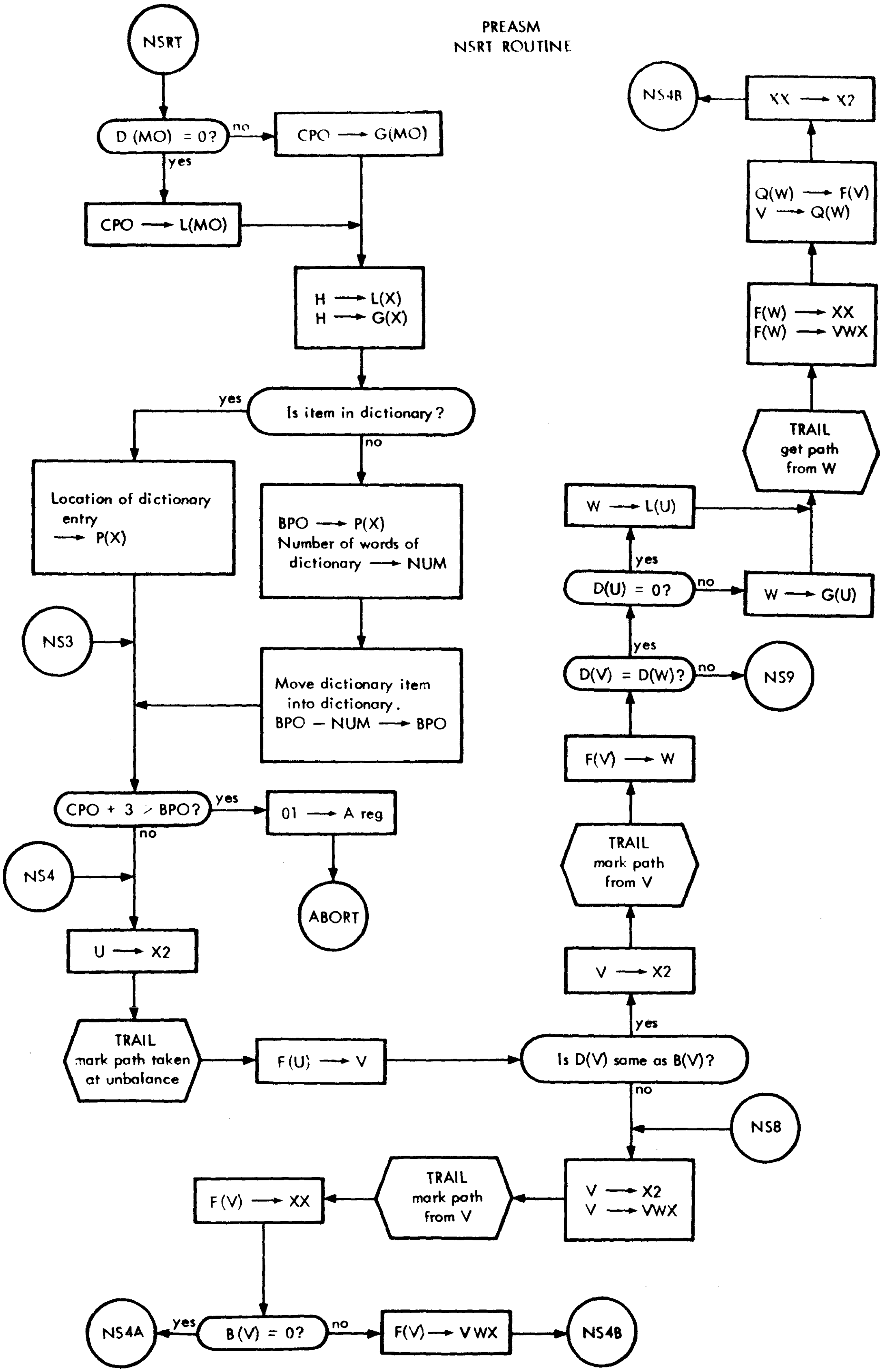
If $G(\alpha) = \beta$ then $M(\beta) = G(\beta)$ and $N(\beta) = L(\beta)$

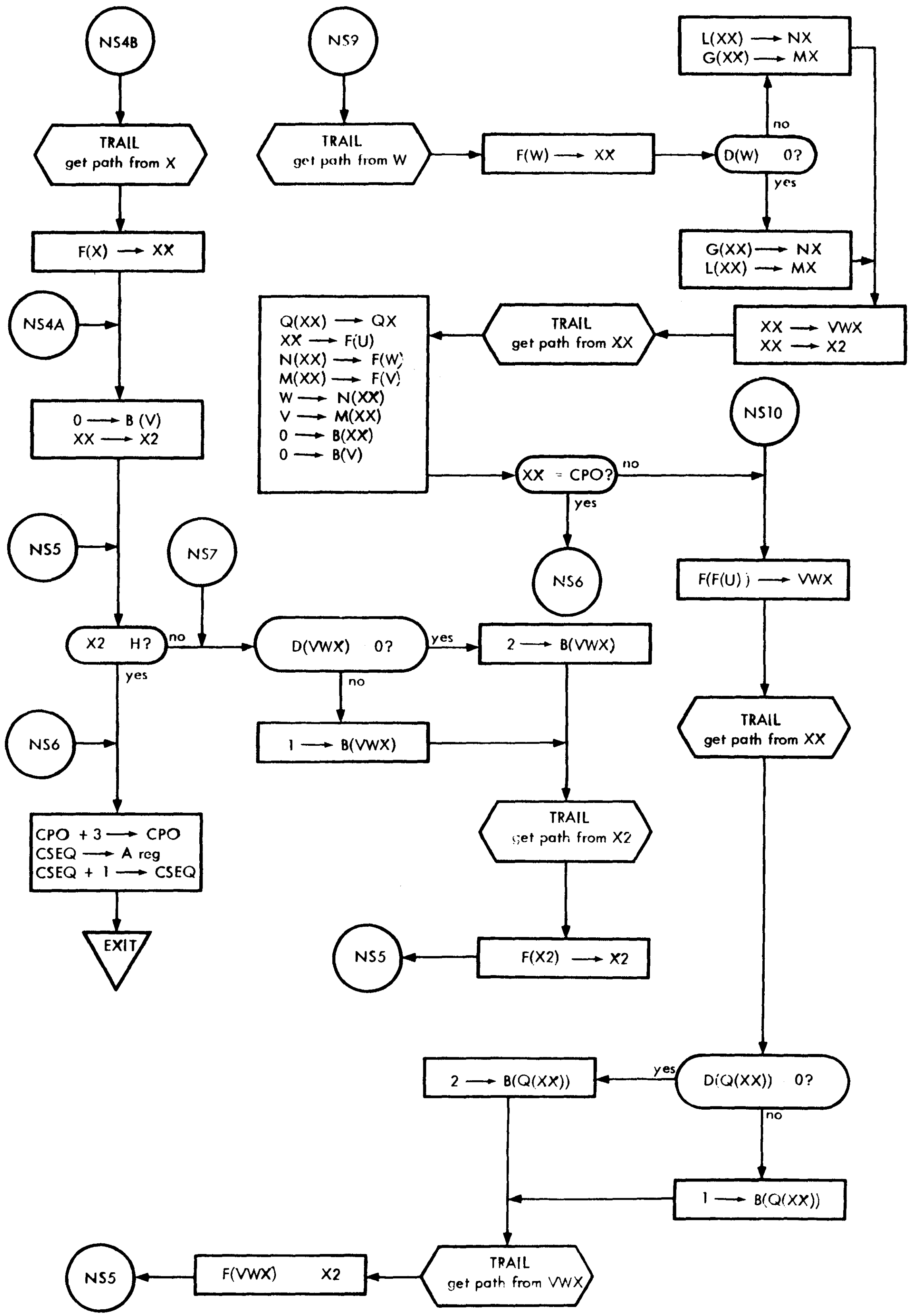
If $L(\alpha) = \beta$ then $M(\beta) = L(\beta)$ and $N(\beta) = G(\beta)$

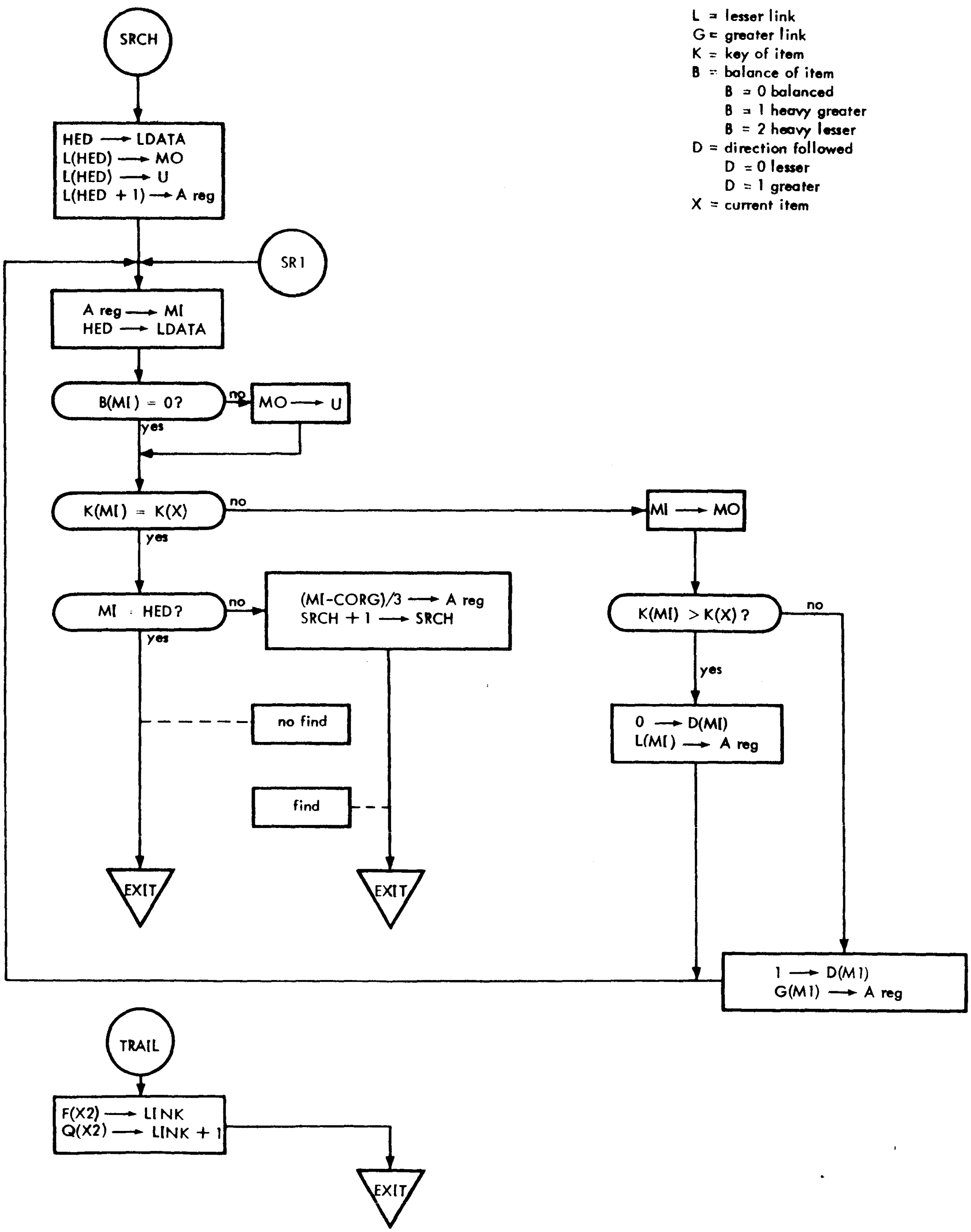
H denotes location of HED.

$P(\alpha)$ denotes location of dictionary entry for byte α.

PREASM
NSRT ROUTINE




NS4B
TRAIL get path from X
F(X) → XX
NS4A
0 → B (V)
XX → X2
NS5
NS7
X2 H?
no
yes
NS6
CPO + 3 → CPO
CSEQ → A reg
CSEQ + 1 → CSEQ
EXIT
NS9
TRAIL get path from W
F(W) → XX
D(W) 0?
no
yes
L(XX) → NX
G(XX) → MX
G(XX) → NX
L(XX) → MX
XX → VWX
XX → X2
TRAIL get path from XX
Q(XX) → QX
XX → F(U)
N(XX) → F(W)
M(XX) → F(V)
W → N(XX)
V → M(XX)
0 → B(XX)
0 → B(V)
XX = CPO?
no
yes
NS6
NS10
F(F(U)) → VWX
D(VWX) 0?
yes
no
2 → B(VWX)
1 → B(VWX)
TRAIL get path from X2
F(X2) → X2
NS5
TRAIL get path from XX
D(Q(XX)) 0?
yes
no
2 → B(Q(XX))
1 → B(Q(XX))
TRAIL get path from VWX
F(VWX) X2
NS5


L = lesser link
G = greater link
K = key of item
B = balance of item
B = 0 balanced
B = 1 heavy greater
B = 2 heavy lesser
D = direction followed
D = 0 lesser
D = 1 greater
X = current item
SRCH
HED → LDATA
L(HED) → MO
L(HED) → U
L(HED + 1) → A reg
SR1
A reg → MI
HED → LDATA
B(MI) = 0?
no
MO → U
yes
K(MI) = K(X)
no
MI → MO
yes
MI = HED?
no
(MI-CORG)/3 → A reg
SRCH + 1 → SRCH
yes
K(MI) > K(X) ?
no
yes
0 → D(MI)
L(MI) → A reg
no find
find
EXIT
EXIT
1 → D(M1)
G(M1) → A reg
TRAIL
F(X2) → LINK
Q(X2) → LINK + 1
EXIT

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Purge unused bytes from the dictionary and byte tables (SRNK or SHRINK)

PURPOSE: To make maximum table space available to the assembly routines by removing those entries in the dictionary and byte table which represent bytes from the standard procedures file that are not needed to assemble a particular program.

ACTION: SHRINK steps through the byte table starting at the first byte following the user's program and examines each byte to see if it has been flagged to be saved. Bytes not flagged are skipped. Bytes to be saved are moved up to follow the previous saved bytes, and the dictionary entry for the byte is moved down to follow the previously saved dictionary entry. As each byte is examined, a translation table is constructed giving the new byte value for the byte.

The byte table is scanned again in its entirety, and the save flags are removed. As each byte is obtained, it is examined to determine if a symbol table entry exists for the byte. The dictionary pointer in each symbol replaces the byte table pointer to the symbol, and the symbol is set to point to the byte table entry. When all save flags have been removed and all symbol table pointers reversed, SHRINK proceeds to the next step.

The symbol table is scanned, and the symbols to be saved are moved up to follow the byte table. If this is a NAME, the sample pointer must be revised; and, if it is the first NAME encountered for a procedure, the procedure sample is moved to the cells following the dictionary. As each byte in the sample is moved, it is translated to the new byte value so that the byte numbers resulting will be a contiguous set. The symbol table pointers are reset to their normal format.

ACTION: (cont.)

After the symbol table and sample have been moved, SHRINK sets the communication cells for the assembler routines and calls the tape loader to load ASSEMBLER.

PROGRAMMING TECHNIQUES:

The SHRINK routine must proceed to accomplish its function in a rigid sequence, since there is no correspondence between the sequence of bytes in the byte table and the order of the appearance of the NAME and directive items in the symbol table. SHRINK assumes that there is at most one symbol for each byte and that the order of NAMEs is also the order in which sample is saved. The first assumption could be violated, but should not be on the standard procedures file, and the second assumption is always true.

SHRINK is loaded over part of PREASM; however, care must be exercised in setting the origin for SHRINK since many communication cells and some PREASM subroutines are used by SHRINK. The external references to SHRINK are satisfied by loading SHRINK with PREASM and then punching the absolute program, to be placed on the system tape, from memory. SHRINK is an absolute routine separately assembled.

CALLING SEQUENCE:

SHRINK is loaded and executed by the tape loader as a separate memory overlay.

MEMORY REQUIREMENTS:

Variable, but at least 8192_{10} words

SUBROUTINES USED:

GTB[t]	GTCR
MOVE[t]	STCR
MVITM	ITMOV
SMPTRN	SAMPLE

---

t These routines are described under PREASM.

ENTRY POINTS TO SHRINK SUBROUTINES

Entry	Page Description	Page Flowchart	Entry	Page Description	Page Flowchart
GTCR	3-160	3-165	SHR10	3-155	3-164
ITMO	3-162	3-166	SHR11	3-155	3-164
ITMOV	3-162	3-166	SHR12	3-155	3-165
MVITM	3-158	3-165	SHRINK	3-155	3-164
SAMPLE	3-163	3-167	SMPTRN	3-159	3-166
SHR3	3-155	3-164	STCL	3-161	3-165
SHR7	3-155	3-164			

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Reverse symbol table linkage (MVITM)

PURPOSE: To relink the byte table and symbol table pointers so the byte table points to the dictionary and the symbol table points to the byte table.

ACTION: MVITM takes the dictionary pointer from the symbol table entry addressed by the A register at entry and places it in the A field of the byte table entry addressed by FBWRD. The location of the byte table location is then placed in the A field of the symbol entry.

PROGRAMMING TECHNIQUES: MVITM is an absolute routine assembled as part of SHRINK.

CALLING SEQUENCE:
Byte table entry to A register
BRM MVITM

MEMORY REQUIREMENTS: 14_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 900 Series: 042016
9300: 612001

IDENTIFICATION: Translate and move procedure sample (SMPTRN)

PURPOSE: To translate the bytes of procedure sample to the new byte values and move the sample to its new origin.

ACTION: SMPTRN sets the parameters to cause GTB to get bytes starting at the old procedure sample and the parameters to cause MOVE to store bytes at the new sample origin. SMPTRN then obtains bytes by calling GTB, translates them to the new value by taking the TRTB table entry for the byte, and stores them in BBUF. As each line is obtained, SMPTRN checks for PROC, FUNC, or END directives to determine the amount of sample to move. MOVE is called to store bytes into sample storage.

PROGRAMMING TECHNIQUES: SMPTRN is an absolute routine assembled as part of SHRINK.

CALLING SEQUENCE: BRM SMPTRN

MEMORY REQUIREMENTS: 106_8 cells

SUBROUTINES USED: GTB
MOVE

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Get dictionary characters (GTCR)

PURPOSE: To get the next dictionary character to the A register and FCHR.

ACTION: GTC takes the next character as indicated by FBDC from the dictionary word addressed by FDW and stores it in the A register and FCHR. The character position is incremented.

PROGRAMMING TECHNIQUES: GTCR is an absolute routine assembled as part of SHRINK.

CALLING SEQUENCE: BRM GTCR

MEMORY REQUIREMENTS: 25_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Store characters into dictionary (STCR)

PURPOSE: To store characters into their new dictionary locations.

ACTION: STCR positions the character given in the A register to the position indicated by TBDC and stores it into the dictionary word addressed by TDW. The other three characters in TDW are preserved. The character position is incremented.

PROGRAMMING TECHNIQUES: STCR is an absolute routine assembled as part of SHRINK.

CALLING SEQUENCE:

```
Character to A register
BRM    STCR
```

MEMORY REQUIREMENTS: 30_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Move symbol table entries (ITMOV)

PURPOSE: To move all symbol table entries to their new location and to relink the byte table and items pointers.

ACTION: ITMOV calls SAMPLE to process the NAME item sample linkage and sample moving. The item is moved from the old location given by FITAB to the location given by TITAB. The byte table entry for the entry is given an associate address of the new location, and the symbol table item is given the associate linkage from the byte table (points to dictionary). The 'from' and 'to' positions are incremented. If the item is to be deleted, it is not moved and only the 'from' pointer is incremented. ITMOV continues processing until all items are moved.

PROGRAMMING TECHNIQUES: ITMOV is an absolute routine assembled as part of SHRINK.

CALLING SEQUENCE: BRM ITMOV

MEMORY REQUIREMENTS: 51_8 cells

SUBROUTINES USED: SAMPLE

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Set NAME item sample pointers (SAMPLE)

PURPOSE: To determine the proper sample pointer word to be associated with a NAME item and, when needed, to call SMPTRN to move procedure sample.

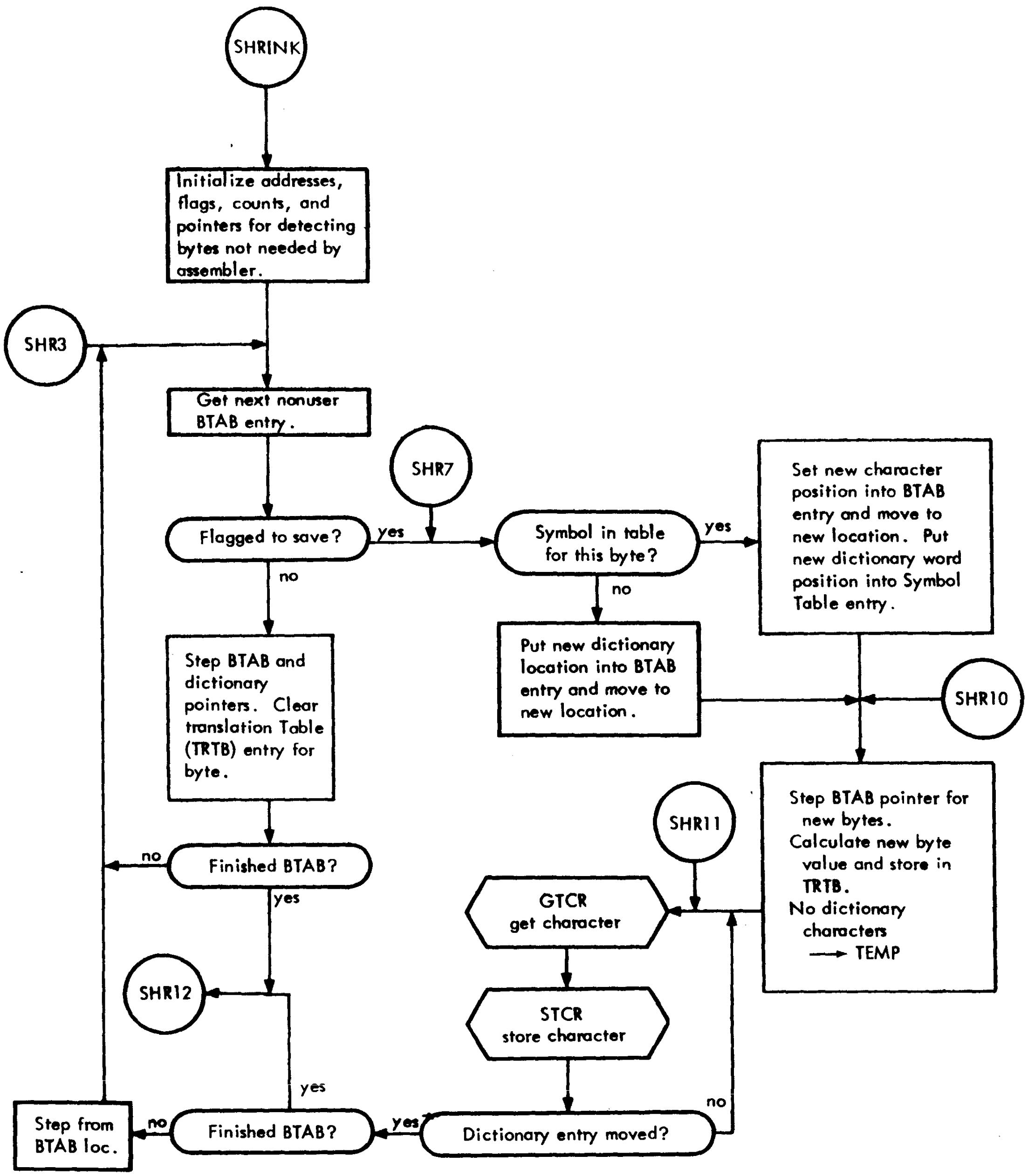
ACTION: SAMPLE tests the symbol table entry at FITAB to determine if it is a NAME. If the item is not a NAME item, SAMPLE exits without taking further action. If it is a name item, SAMPLE tests to see if this is the first occurrence of a NAME item for this procedure by comparing the sample pointer word for the name with the entries in a table giving the old and new sample pointer words from previously processed NAME items. If the NAME is that of a procedure which has been encountered, SAMPLE takes the new sample pointer word from the table and inserts it into the NAME item. If this is a first NAME encountered, SAMPLE determines the current sample position and constructs a new pointer word which is inserted into the NAME item. Entries are made in the PSMPLC table showing the old and new sample positions, and SMPTRN is called to move the procedure sample.

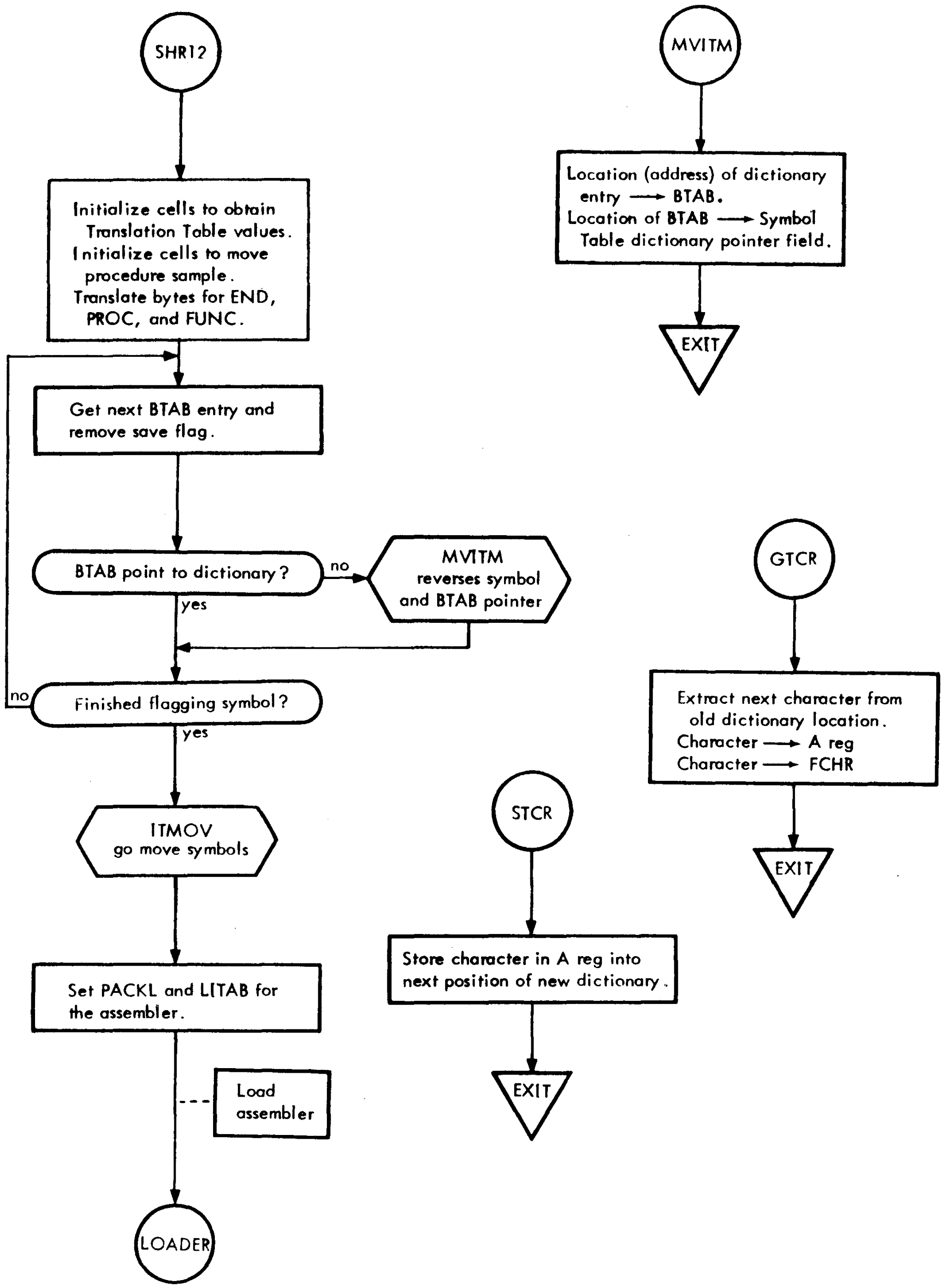
PROGRAMMING TECHNIQUES: SAMPLE is a relocatable routine assembled as part of SHRINK.

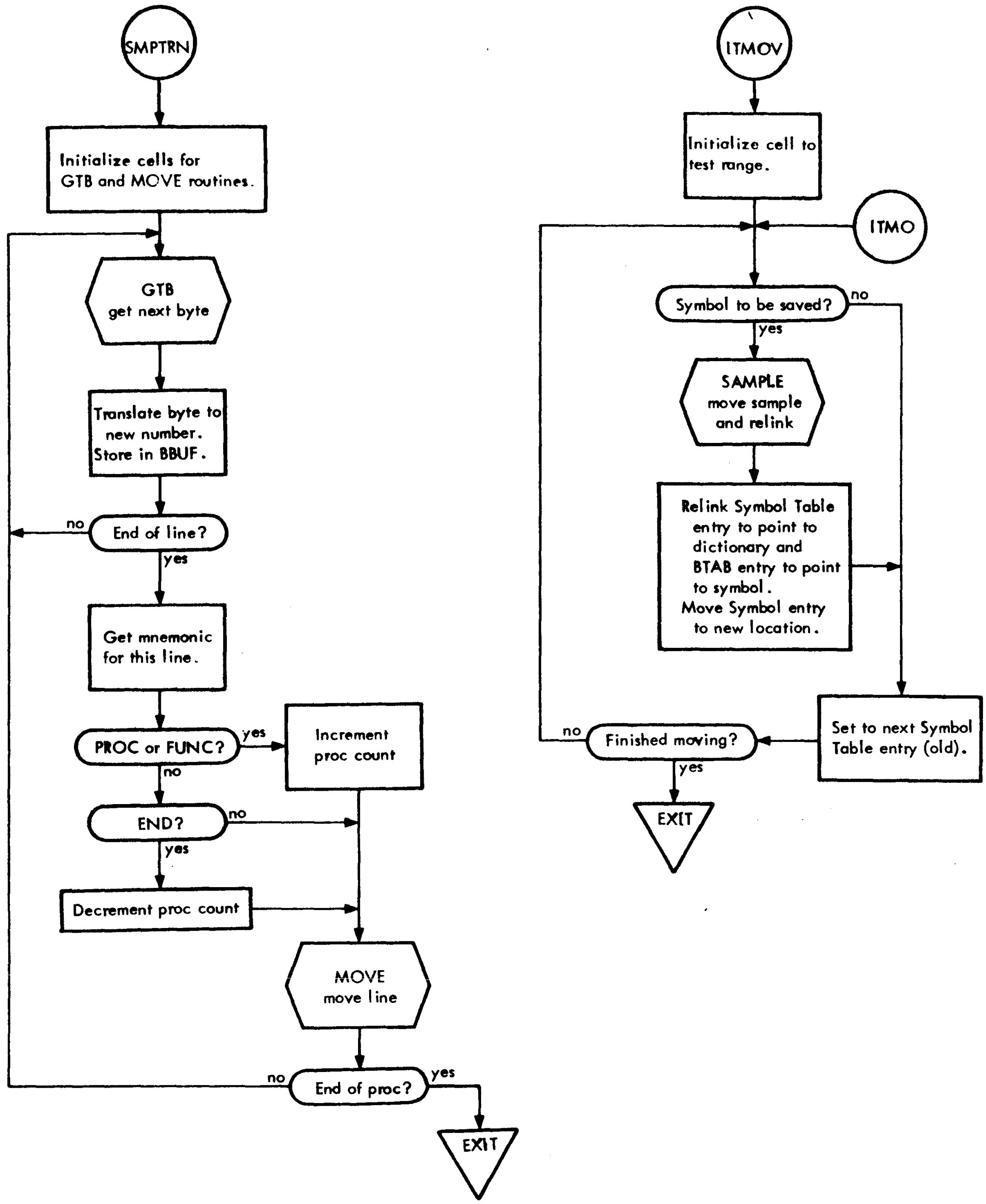
CALLING SEQUENCE: BRM SAMPLE

MEMORY REQUIREMENTS: 70_8 cells

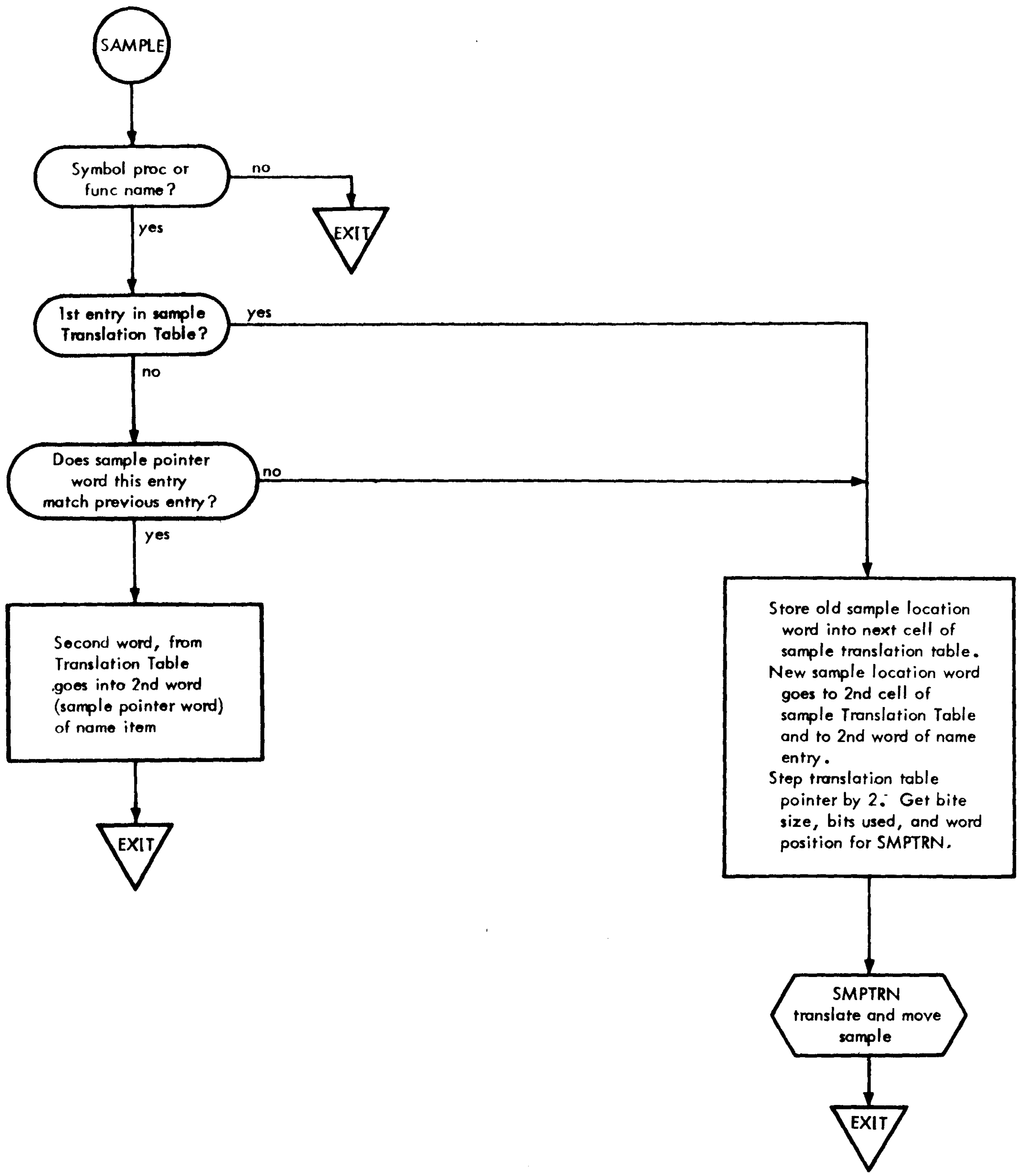
SUBROUTINGES USED: SMPTRN


SHRINK
Initialize addresses, flags, counts, and pointers for detecting bytes not needed by assembler.
SHR3
Get next nonuser BTAB entry.
SHR7
Flagged to save?
yes
no
Symbol in table for this byte?
yes
no
Set new character position into BTAB entry and move to new location. Put new dictionary word position into Symbol Table entry.
Step BTAB and dictionary pointers. Clear translation Table (TRTB) entry for byte.
Put new dictionary location into BTAB entry and move to new location.
SHR10
Step BTAB pointer for new bytes.
Calculate new byte value and store in TRTB.
No dictionary characters ——→ TEMP
SHR11
Finished BTAB?
no
yes
GTCR
get character
SHR12
STCR
store character
yes
no
Step from BTAB loc.
no
Finished BTAB?
yes
Dictionary entry moved?


SHR12
Initialize cells to obtain Translation Table values.
Initialize cells to move procedure sample.
Translate bytes for END, PROC, and FUNC.
Get next BTAB entry and remove save flag.
BTAB point to dictionary?
no
MVITM reverses symbol and BTAB pointer
yes
Finished flagging symbol?
no
yes
ITMOV go move symbols
Set PACKL and LITAB for the assembler.
Load assembler
LOADER
MVITM
Location (address) of dictionary entry → BTAB.
Location of BTAB → Symbol Table dictionary pointer field.
EXIT
GTCR
Extract next character from old dictionary location.
Character → A reg
Character → FCHR
EXIT
STCR
Store character in A reg into next position of new dictionary.
EXIT


SMPTRN
Initialize cells for GTB and MOVE routines.
GTB
get next byte
Translate byte to new number. Store in BBUF.
End of line?
no
yes
Get mnemonic for this line.
PROC or FUNC?
yes
no
Increment proc count
END?
no
yes
Decrement proc count
MOVE
move line
End of proc?
no
yes
EXIT
ITMOV
Initialize cell to test range.
ITMO
Symbol to be saved?
no
yes
SAMPLE
move sample and relink
Relink Symbol Table entry to point to dictionary and BTAB entry to point to symbol. Move Symbol entry to new location.
Set to next Symbol Table entry (old).
Finished moving?
no
yes
EXIT

PAS1
(ASSEMBLR)


SAMPLE
Symbol proc or func name?
no
EXIT
yes
1st entry in sample Translation Table?
yes
no
Does sample pointer word this entry match previous entry?
no
yes
Second word, from Translation Table goes into 2nd word (sample pointer word) of name item
EXIT
Store old sample location word into next cell of sample translation table. New sample location word goes to 2nd cell of sample Translation Table and to 2nd word of name entry. Step translation table pointer by 2. Get bite size, bits used, and word position for SMPTRN.
SMPTRN translate and move sample
EXIT

SDS SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Perform the first assembly pass (PAS1 or ASSEMBLR)

PURPOSE: To perform the first assembly pass over the user's program contained on the intermediate output tape X1. This includes the following functions:

1. To define symbols appearing in the label fields by inserting the appropriate item type (see Section 4, Item Formats) entry into the symbol table.
2. To store the procedure sample contained in the user's program into memory for later reference.
3. To maintain a count of the space needed by the program so that location-dependent labels may be defined and the origin of literals may be determined.
4. If called for, to regenerate the symbolic program from the encoded representation.
5. To output the external symbol definitions to the binary output file.
6. To generate both internal and external programmed operator definitions and to define programmed operator references.
7. To output external programmed operator definitions and programmed operator references to the binary output file.

ACTION: ASSEMBLR rewinds the input tape, X1, and then reads and processes the program contained thereon one line at a time. If symbolic output has been requested, each line is reconstructed and written on the symbolic output file.

Lines are scanned from left to right. When a label is encountered, a tentative definition of the label is made equating its value to the current value

ACTION:
(cont.)

of the location counter. The operation field is obtained and a determination is made of the line type. Directives are processed by executing a directive branch table which causes control to go to the proper directive processing routine. Procedure references are processed at PRL; FORM references, at FRL. If an operation is undefined, it is processed at POPR as a programmed operator reference. A non-symbolic operation is treated as an error.

Before each new line is obtained, a test is made to see of a DO directive has been encountered, but not completed. If there is an active DO directive, control goes to DOAGN to repeat the line or lines already obtained.

Within each of the routines to process the various types of lines, the operand field is evaluated by calling SCAN. When the routines have completed their tasks, control returns to the main control section where any label which has been encountered, but not defined, is defined by calling NSRT to place the label definition into the symbol table. The location counter is incremented as needed, and control returns to LINE to process the next line of user's program.

When all lines have been processed, ASSEMBLR outputs the external symbol definitions, external programmed operator definitions, and programmed operator references to the binary output file.

PAS2 is then loaded by calling the tape loader.

PROGRAMMING
TECHNIQUES:

ASSEMBLR is segmented into five separately assembled parts plus the programmed operators. An absolute version of ASSEMBLR is made for insertion on the system tape by loading the separate routines and punching the absolute program from core. Two cells (PACKL and LITAB), giving the upper and lower table locations used by PREASM, are located just below the origin of ASSEMBLR and are referenced by ASSEMBLR as absolute location. Several of the communication cells between MSCONTRL and ASSEMBLR are also

PROGRAMMING TECHNIQUES: (cont.)

referred to as absolute locations. Changes in any of these communication locations necessitate the reassembly of ASSEMBLR to reflect the changes. ASSEMBLR is a relocatable program originated at 01354. The machine memory size is determined from the contents of cell 1.

CALLING SEQUENCE:

ASSEMBLR is loaded and executed by the tape loader.

MEMORY REQUIREMENTS:

Variable but at least 8192_{10} words

SUBROUTINES USED:

In addition to the routines listed here, the file processing and I/O routines of MSCONTRL may be used.

TEXT	POPR	SCRP	PEEK
IPL	DO	EDC	GNC
MBYT	DOAGN	EDS	GET
SKIP	DODEC	OUTP	GBSL
INC	PRL	FLUSH	MIFT
GCW	FNRL	RESET	GLOP
GTB	DFLST	PAGE	POP
GEC	END	EPRNT	EDTST[t]
LBTST	FRL	DED	PLINE[t]
PLB	BCD	GLOV	RDPI[t]
PLTST	TEXTR	M3WAI	EDIT[t]
EQU	SAM	FLN	EDTV[t]
AORG	NAME	FLM	TYPWRT[t]
ORG	MVPRC	RELTST	EDTL[t]
RES	MOVE	CNVRT	EDE[t]
FORM	SWITCH	DPDIV	HOME[t]
FUNC	GTLBL	SCAN	FLDC[t]
PROC	SRCH	GIT	PRNT[t]
POPD	NSRT	SCANC	MFOI[t]

[t]These routines may be called by ASSEMBLR, but perform no operation needed for first pass processing. In the case of EDTST, return is to the location of its call (BRM EDTST) plus 2.

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

IDENTIFICATION: Perform the first assembly pass (PAS1 or ASSEMBLR)

PURPOSE: To perform the first assembly pass over the user's program contained on the intermediate output tape X1. This includes the following functions:

1. To define symbols appearing in the label fields by inserting the appropriate item type (see Section 4, Item Formats) entry into the symbol table.
2. To store the procedure sample contained in the user's program into memory for later reference.
3. To maintain a count of the space needed by the program so that location-dependent labels may be defined and the origin of literals may be determined.
4. If called for, to regenerate the symbolic program from the encoded representation.
5. To output the external symbol definitions to the binary output file.
6. To generate both internal and external programmed operator definitions and to define programmed operator references.
7. To output external programmed operator definitions and programmed operator references to the binary output file.

ACTION: ASSEMBLR rewinds the input tape, X1, and then reads and processes the program contained thereon one line at a time. If symbolic output has been requested, each line is reconstructed and written on the symbolic output file.

Lines are scanned from left to right. When a label is encountered, a tentative definition of the label is made equating its value to the current value

ACTION: (cont.)

of the location counter. The operation field is obtained and a determination is made of the line type. Directives are processed by executing a directive branch table which causes control to go to the proper directive processing routine. Procedure references are processed at PRL; FORM references, at FRL. If an operation is undefined, it is processed at POPR as a programmed operator reference. A non-symbolic operation is treated as an error.

Before each new line is obtained, a test is made to see of a DO directive has been encountered, but not completed. If there is an active DO directive, control goes to DOAGN to repeat the line or lines already obtained.

Within each of the routines to process the various types of lines, the operand field is evaluated by calling SCAN. When the routines have completed their tasks, control returns to the main control section where any label which has been encountered, but not defined, is defined by calling NSRT to place the label definition into the symbol table. The location counter is incremented as needed, and control returns to LINE to process the next line of user's program.

When all lines have been processed, ASSEMBLR outputs the external symbol definitions, external programmed operator definitions, and programmed operator references to the binary output file.

PAS2 is then loaded by calling the tape loader.

PROGRAMMING TECHNIQUES:

ASSEMBLR is segmented into five separately assembled parts plus the programmed operators. An absolute version of ASSEMBLR is made for insertion on the system tape by loading the separate routines and punching the absolute program from core.

CALLING SEQUENCE: ASSEMBLR is loaded and executed by the tape loader.

MEMORY REQUIREMENTS: Variable but at least 8192_{10} words

SUBROUTINES USED: In addition to the routines listed here, the file processing and I/O routines of MSCONTRL may be used.

TEXT	POPR	SCRP	PEEK
IPL	DO	EDC	GNC
MBYT	DOAGN	EDS	GET
SKIP	DODEC	OUTP	GBSL
INC	PRL	FLUSH	MIFT
GCW	FNRL	RESET	GLOP
GTB	DFLST	PAGE	POP
GEC	END	EPRNT	EDTST
LBTST	FRL	DED	PLINE[t]
PLB	BCD	GLOV	RDPI[t]
PLTST	TEXTR	M3WAI	EDIT[t]
EQU	SAM	FLN	EDTV[t]
AORG	NAME	FLM	TYPWRT[t]
ORG	MVPRC	RELTST	EDTL[t]
RES	MOVE	CNVRT	EDE[t]
FORM	SWITCH	DPDIV	HOME[t]
FUNC	GTLBL	SCAN	FLDC[t]
PROC	SRCH	GIT	PRNT[t]
POPD	NSRT	SCANC	MFOI[t]

[t] These routines may be called by ASSEMBLR, but perform no operation needed for first pass processing. In the case of EDTST, return is to the location of its call (BRM EDTST) plus 2.

ENTRY POINTS TO ASSEMBLR (PASS 1) SUBROUTINES

Entry	Page Description	Page Flowchart	Entry	Page Description	Page Flowchart
AORG	3-188	3-151	DO2	3-193	3-255
BCD	3-202	3-263	DO3	3-193	3-255
CNV1	3-226	3-296	DOA2	3-194	3-256
CNV2	3-226	3-296	DOA3	3-194	3-256
CNV3	3-226	3-297	DOA4	3-194	3-256
CNV6	3-226	3-297	DOA5	3-193	3-255
CNV7	3-226	3-297	DOAGN	3-194	3-256
CNVRT	3-226	3-296	DODEC	3-195	3-257
COAD	3-228	3-284	DOEND	3-193	3-255
COAD2	3-228	3-284	DOERR	3-193	3-255
COAD3	3-228	3-284	DOVFW	3-193	3-255
COAP	3-228	3-285	DPDIV	3-227	3-295
COAS	3-228	3-284	EDC	3-213	3-273
COAS1	3-228	3-284	EDE		3-272
COAS3	3-228	3-284	EDIT		3-272
COBS	3-228	3-286	EDS	3-214	3-273
CODS	3-228	3-285	EDTL		3-272
COEQ	3-228	3-282	EDTST		3-267
COGT	3-228	3-282	EDTV		3-272
COIQ	3-228	3-285	END	3-199	3-260
COLD	3-228	3-283	END1	3-199	3-261
COLS	3-228	3-283	END1A	3-199	3-261
COLS1	3-228	3-283	END1B	3-199	3-261
COLS2	3-228	3-283	END1BA	3-199	3-261
COLS3	3-228	3-283	END2	3-199	3-260
COLS4	3-228	3-283	END3	3-199	3-261
COLS6	3-228	3-283	ENDM		3-274
COLS6A	3-228	3-283	ENDN		3-274
COLSZ	3-228	3-283	ENDP	3-199	3-262
COLT	3-228	3-282	ENDS	3-199	3-260
COLT1	3-228	3-282	EPRNT	3-219	3-276
COLT2	3-228	3-282	EQU	3-187	3-251
COLT3	3-228	3-282	EQU1	3-187	3-251
COXQ	3-228	3-285	EQU3	3-187	3-251
COXQ1	3-228	3-285	EQU4	3-187	3-251
DATAT		3-274	EQU6	3-187	3-251
DED	3-220	3-277	EQU7	3-187	3-251
DEF		3-274	FLDC		3-272
DELST	3-198	3-259	FLM	3-224	3-279
DO	3-193	3-255	FLN	3-223	3-279
DO1	3-193	3-255	FLUSH	3-216	3-275
DO1ZZ	3-193	3-255	FLUSH1	3-216	3-275

ENTRY POINTS TO ASSEMBLR (PASS 1) SUBROUTINES (cont.)

Entry	Page Description	Page Flowchart	Entry	Page Description	Page Flowchart
FNRL	3-196	3-258	GITS2	3-230	3-288
FNRL1	3-196	3-258	GITS3	3-230	3-288
FNRL2	3-196	3-258	GITS4	3-230	3-287
FORM	3-189	3-252	GITS5	3-230	3-288
FRL	3-201	3-263	GITS8	3-230	3-287
FUNC	3-190	3-253	GITS9	3-230	3-288
GBSL	3-237	3-294	GITX	3-230	3-291
GBSL2	3-237	3-294	GLOP	3-239	3-278
GCW	3-181	3-249	GLOV	3-221	3-278
GEC	3-183	3-250	GNC	3-235	3-293
GET	3-236	3-294	GNC3	3-235	3-293
GET1	3-236	3-294	GNCE	3-235	3-293
GET4	3-236	3-294	GNCER	3-235	3-293
GET6	3-236	3-294	GO1	3-198	3-259
GI13	3-230	3-287	GTB	3-182	3-250
GIT	3-230	3-287	GTB1	3-182	3-250
GIT1	3-230	3-287	GTLBL	3-208	3-268
GIT2	3-230	3-291	GTRBL		3-287
GIT3	3-230	3-287	HOME		3-272
GIT4	3-230	3-290	INC	3-180	3-249
GIT9	3-230	3-289	IPL	3-177	3-247
GIT11	3-230	3-287	LBERR	3-185	3-245
GIT31	3-230	3-290	LBTST	3-184	3-249
GIT32	3-230	3-290	LINE		3-243
GIT33	3-230	3-289	LINSYM		3-244
GIT34	3-230	3-290	LN1		3-243
GIT35	3-230	3-288	LN1A		3-243
GIT35A	3-230	3-288	LN4		3-243
GIT37	3-230	3-290	LNDPV		3-244
GIT41	3-230	3-290	LNE		3-244
GIT42	3-230	3-290	LNEN		3-243
GIT43	3-230	3-289	LNERR		3-244
GIT44	3-230	3-289	LNFRM		3-244
GIT99	3-230	3-289	LNLOC		3-244
GIT351	3-230	3-289	LNVAL		3-244
GIT352	3-230	3-289	M3WAI	3-222	3-278
GITA	3-230	3-291	MBYT	3-178	3-248
GITA2	3-230	3-291	MFOI		3-277
GITC	3-230	3-291	MIFT	3-238	3-279
GITE	3-230	3-291	MO1	3-206	3-266
GITL	3-230	3-291	MO5	3-206	3-266
GITS1	3-230	3-288	MO6	3-206	3-266

ENTRY POINTS TO ASSEMBLR (PASS 1) SUBROUTINES (cont.)

Entry	Page Description	Page Flowchart	Entry	Page Description	Page Flowchart
MOVE	3-206	3-266	PRL2A	3-196	3-259
MTASYM		3-242	PRL3	3-196	3-258
MVPRC	3-206	3-266	PRL7	3-196	3-259
NAM1	3-205	3-265	PRNT		3-272
NAM2	3-205	3-265	PROC	3-190	3-253
NAME	3-205	3-265	RDPI		3-278
NMEND	3-205	3-265	RELTST	3-225	3-295
NMERR	3-205	3-265	RES	3-188	3-252
NOEND		3-244	RESET	3-217	3-275
NS1A	3-210	3-270	RET3A		3-248
NS1B	3-210	3-270	RET4		3-248
NS1C	3-210	3-270	RET5		3-248
NS1D	3-210	3-270	RET10		3-248
NS3	3-210	3-270	REZZ		3-248
NS3A	3-210	3-270	SA2	3-203	3-264
NS9	3-210	3-270	SA3	3-203	3-264
NS99	3-210	3-270	SA4	3-203	3-264
NSRT	3-210	3-270	SAM	3-203	3-264
ORG	3-188	3-251	SC2	3-212	3-271
ORG1	3-188	3-251	SC3	3-212	3-271
OUTP	3-215	3-274	SCAN	3-228	3-280
OUTP1	3-215	3-274	SCAN1	3-228	3-280
OUTP2	3-215	3-274	SCAN2	3-228	3-280
PAGE	3-218	3-276	SCAN3	3-228	3-280
PEEK	3-234	3-293	SCAN6	3-228	3-282
PL1	3-185	3-245	SCAN7	3-228	3-280
PLB	3-185	3-245	SCAN9	3-228	3-281
PLB2	3-185	3-245	SCAN21	3-228	3-280
PLB3	3-185	3-245	SCAN23	3-228	3-280
PLBEX	3-185	3-245	SCAN99	3-228	3-281
PLINE		3-272	SCAN998	3-228	3-281
PLT4	3-186	3-246	SCAN999	3-228	3-281
PLT4A	3-186	3-246	SCANC	3-232	3-292
PLT5	3-186	3-246	SCANC1	3-232	3-292
PLT6	3-186	3-246	SCANC2	3-232	3-292
PLTST	3-186	3-246	SCANC3	3-232	3-292
POP	3-201	3-264	SCANC6	3-232	3-292
POPD	3-191	3-254	SCANC8	3-232	3-292
POPR	3-192	3-254	SCANC9	3-232	3-292
PR7	3-203	3-264	SCANC9E	3-228	3-281
PRL	3-196	3-258	SCANR	3-232	3-292
PRL1	3-196	3-258	SCNC11	3-232	3-292

ENTRY POINTS TO ASSEMBLR (PASS 1) SUBROUTINES (cont.)

Entry	Page Description	Page Flowchart	Entry	Page Description	Page Flowchart
SCRP	3-212	3-271	TEXT1	3-176	3-247
SKIP	3-179	3-249	TEXT2	3-202	3-263
SR5	3-209	3-269	TEXT3	3-202	3-263
SR6	3-209	3-269	TEXTR	3-202	3-263
SR7	3-209	3-269	TXT2	3-176	3-247
SR9	3-209	3-269	TXT3	3-176	3-247
SRCH	3-209	3-269	TXT5	3-176	3-247
START	3-169	3-241	TYPWRT		3-272
SWITCH	3-207	3-267	UNDEF		3-244
TEXT	3-176	3-247	WEOFL		3-276

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Obtain next line of text (TEXT)

PURPOSE: To obtain the next line of input to be processed.

ACTION: TEXT takes the following actions:

1. If the line is to be obtained from the procedure sample area, TEXT calls SKIP to skip to the end of the current line.
2. If symbolic output is requested, TEXT reconstructs the line and stores the bytes into BBUF by calling MBYT and writes the line on the symbolic output file.
3. If the line is not to be output as symbolic, TEXT obtains the bytes by calling GTB and stores them in BBUF. SKIP is called to skip over comments.

PROGRAMMING TECHNIQUES: TEXT is a relocatable routine assembled as part of ASSEMBLR. The symbolic output routine is a standard MSCONTRL I/O routine.

CALLING SEQUENCE:

```
BRM    TEXT
end-of-file return
normal return
```

MEMORY REQUIREMENTS: 70_8 cells

SUBROUTINES USED:

IPL	SKIP
EDS	GTB
EDC	MBYT

symbolic output routine

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 04201
Catalog No. 9300: 6120(

IDENTIFICATION: Initialize line reconstruction (IPL)

PURPOSE: To initialize parameters for reconstructing line images.

ACTION: IPL sets the maximum character count and the line length and initializes the buffer locations for fields by calling EDS. The buffer is set to blanks.

PROGRAMMING TECHNIQUES: IPL is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE: BRM IPL

MEMORY REQUIREMENTS: 14_8 cells

SUBROUTINES USED: EDS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Reconstruct symbolic lines (MBYT)

PURPOSE: To reconstruct line images for punching and to enter bytes into byte buffer, BBUF.

ACTION: MBYT obtains bytes by calling GTB. The byte is stored in BBUF, and the byte table entry is obtained and placed in ECW. The dictionary characters represented by the byte are obtained by calling GEC and stored into the image by calling EDC. If the line is continued, the first portion is output to the symbolic file (listing in PAS2). INC is used to obtain comment characters.

PROGRAMMING TECHNIQUES: MBYT is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE: BRM MBYT

MEMORY REQUIREMENTS: 102_8 cells

SUBROUTINES USED:

GTB GEC
IPL EDC
INC GBSL

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 0420l
Catalog No. 9300: 6120(

IDENTIFICATION: Skip to the end of lines (SKIP)

PURPOSE: To skip to the end of the current line.

ACTION: SKIP calls GCW to get consecutive bytes until an end-of-line byte is obtained. Comments are skipped by calling INC to get comment characters.

PROGRAMMING TECHNIQUES: SKIP is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE: BRM SKIP

MEMORY REQUIREMENTS: 22_8 cells

SUBROUTINES USED: GCW
INC

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Get comment characters (INC)

PURPOSE: To fetch the next comment character.

ACTION: INC saves the current byte size and mask and then sets the size to six bits. INC calls GTB to fetch the next six bits of input, after which it restores the byte size and mask. The character is in the A register.

PROGRAMMING TECHNIQUES: INC is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE: BRM INC

MEMORY REQUIREMENTS: 22_8 cells

SUBROUTINES USED: GTB

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 900 Series: 042016
9300: 612001

IDENTIFICATION: Get the next byte table entry (GCW)

PURPOSE: To get the next byte table entry and byte value.

ACTION: GCW gets the next byte from BBUF or by calling GTB if within a procedure. The negative byte value is placed in the A and X registers and in BYT; the byte table location for the byte is placed in ABYT, and the byte table entry is placed in the B register and in ECW.

PROGRAMMING TECHNIQUES: GCW is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE: BRM GCW

MEMORY REQUIREMENTS: 20_8 cells

SUBROUTINES USED: GTB

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Get bytes from the input file (GTB)

PURPOSE: To obtain the negative of the next byte of input and place it in BYT and in the A and index registers.

ACTION: GTB extracts the next BSIZ bits from CHAD, complements the result, and stores it in BYT and the index register. If there are fewer than BSIZ bits remaining in CHAD, INPUT is called to obtain the next encoded text word. If a byte has zero value, the value is taken to the 2^{BSIZ}, and BSIZ and its related mask are incremented.

PROGRAMMING TECHNIQUES: GTB is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE: BRM GTB

MEMORY REQUIREMENTS: 42_8 cells

SUBROUTINES USED: INPUT

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 04201
Catalog No. 9300: 61200

IDENTIFICATION: Get a dictionary character (GEC)

PURPOSE: To fetch to the A register and NCE the character from a dictionary entry.

ACTION: GEC extracts the next character from the location given by ECW and stores it in the A register and NCE.

PROGRAMMING TECHNIQUES: GEC assumes that ECW has the format of the byte table entry and that the right 15_{10} bits of ECW point to the dictionary word. GEC modifies ECW to indicate characters remaining and next character position. GEC is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE:

```
Control word to ECW
BRM    GEC
end-of-string return
normal return
```

MEMORY REQUIREMENTS: 34_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Test for waiting labels (LBTST)

PURPOSE: To define waiting labels and reset the label flag.

ACTION: If LBL at the current PROC level contains a label, LBTST calls NSRT to enter it into the symbol table and then resets LBL to zero.

PROGRAMMING TECHNIQUES: LBTST is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE: BRM LBTST

MEMORY REQUIREMENTS: 11_8 cells

SUBROUTINES USED: NSRT

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 04201
Catalog No. 9300: 6120C

IDENTIFICATION: Process label fields (PLB)

PURPOSE: To scan the label field of a line, set a tentative definition of the label if it is present, and set the cell WLLVL to indicate the procedure level at which the label is to be defined.

ACTION: WLLVL calls GCW to obtain the bytes of the label field and the blank following the label. If the line is a comment, PLB exits with an end-of-line flag in the A register. WLLVL is set to reflect the level at which the label is to be defined. A tentative definition is made for the label, setting it equal to the location counter value; this tentative definition in the form of an address item is placed in LBL through LBL+3. PLTST is called to test for an external label string.

PROGRAMMING TECHNIQUES: PLB is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE:

```
BRM   PLB
end-of-line return
normal return
```

MEMORY REQUIREMENTS: 134_8 cells

SUBROUTINES USED:

GCW GBSL
GEC PLTST

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Process external label strings (PLTST)

PURPOSE: To process strings of external label definitions.

ACTION: PLTST determines if the label is external and if it is either the only field on the line or followed by a second symbol. If it is not, PLTST returns to PLB without taking action. Otherwise, a flag is set for SRCH to accept any type of symbol definition, and SRCH is called to test for the presence of the symbols in the string at the current procedure level. As each symbol definition at the current level is found, it is redefined at a lower level by calling NSRT. Labels not found are ignored.

PROGRAMMING TECHNIQUES: PLTST is an open routine assembled as part of ASSEMBLR and used only in conjunction with PLB.

CALLING SEQUENCE: PLTST is called by PLB and returns either to PLB or to the main line code.

MEMORY REQUIREMENTS: 122_8 cells

SUBROUTINES USED:

GCW NSRT
GBSL GEC
SRCH

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Process EQU directives (EQU).

PURPOSE: To process the EQU directive

ACTION: The operand field of the line is evaluated by calling SCAN. The value returned by SCAN is used to construct an item definition in LBL to LBL+3. If the operation is a reference, LBL is set to zero and return is made to LINSYM. In constructing the item definition, EQU uses the associate set for the tentative definition of the symbol by PLB and the type and mode bits of the operand field. NSRT is called to define the item.

PROGRAMMING TECHNIQUES: EQU is an open subroutine assembled as part of ASSEMBLR.

CALLING SEQUENCE: EQU is called by executing the directive branch table and returns to the main line code.

MEMORY REQUIREMENTS: 107_8 cells

SUBROUTINES USED:

SCAN MFOI
NSRT RDPI

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Process AORG, ORG and RES directives (AORG, ORG and RES)

PURPOSE: To process the indicated directive.

ACTION: Each of these routines calls SCAN to evaluate the operand field.

1. RES stores the resulting value in CCINC.
2. ORG appends the relocation flag and stores the value in CC and LBL+1.
3. AORG removes any relocation flag and stores the value in CC and LBL+1.

PROGRAMMING TECHNIQUES: All of these routines are open routines assembled as part of ASSEMBLR.

CALLING SEQUENCE: Each is called by executing the directive branch table, and each returns to the main line code at LNLOC.

MEMORY REQUIREMENTS: 15_8 cells total

SUBROUTINES USED: SCAN

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Process FORM directives (FORM)

PURPOSE: To process the FORM directive.

ACTION: FORM calls SCAN to evaluate the expressions in the operand field. As each field is obtained, a bit is set in a double form control word and the word is cycled left by the value of expression evaluated by SCAN. When all fields have been evaluated, the form control words are cycled right one bit and placed in a form definition item in LBL through LBL+2. NSRT is called to place the item into the symbol table.

PROGRAMMING TECHNIQUES: FORM is an open routine assembled as part of ASSEMBLR.

CALLING SEQUENCE: FORM is called by executing the directive branch table and returns to the line code at LINSYM.

MEMORY REQUIREMENTS: 55_8 cells

SUBROUTINES USED:
NSRT
SCAN

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Process the PROC and FUNC directives (PROC and FUNC)

PURPOSE: To process the indicated directive.

ACTION: The sample processing flag is set, and the nested sample count is incremented. If the line appears while already processing sample, control goes to SA2 to process the line like any other sample line. A flag is set to indicate PROC or FUNC, and a test is made to determine if ASSEMBLR is processing a PROC or FUNC reference. If a reference is being processed, the line position at the beginning of the line is set in PRPOS to be used in defining following NAME lines. If not inside a reference, the bytes of the line are moved to PRBYTS for later insertion into sample storage (see PREASM for more information on the concept of processing procedure sample).

PROGRAMMING TECHNIQUES: PROC and FUNC are open routines assembled as part of ASSEMBLR.

CALLING SEQUENCE: PROC and FUNC are entered by executing the directive transfer table. Both routines return to the main line code at LINSYM.

MEMORY REQUIREMENTS: 53_8 cells total

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Process POPD directive (POPD)

PURPOSE: To process the POPD directive.

ACTION: The label given on the line is defined as a programmed operator by building a local or external programmed operator item with the operation value of the current programmed operator count. The POP count is then incremented. The item built is placed in LBL and LBL+1, and WLLVL is set to define the item at the lower level.

PROGRAMMING TECHNIQUES: POPD is an open routine assembled as part of ASSEMBLR.

CALLING SEQUENCE: POPD is called by executing the directive transfer table; control returns to the main line code at LNLOC.

MEMORY REQUIREMENTS: 27_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Process undefined operations (POPR)

PURPOSE: To define a programmed operator reference item.

ACTION: POPR defines the waiting label and then constructs a programmed operator reference item at LBL and LBL+1. WLLVL is set to cause the item to be defined at the lower procedure level. NSRT is called to place the POP reference item into the symbol table, LBL is set to zero, and the programmed operator count is incremented.

PROGRAMMING TECHNIQUES: POPR is an open routine assembled as part of ASSEMBLR.

CALLING SEQUENCE: POPR is called by the line code when an undefined operation is detected and return is to the line code at LNLOC.

MEMORY REQUIREMENTS: 41_8 cells

SUBROUTINES USED: NSRT

SDS PROGRAM LIBRARY PROGRAM DESCRIPTION

900 Series: 0420
Catalog No. 9300: 6120(

IDENTIFICATION: Process DO directives (DO)

PURPOSE: To process DO directives.

ACTION: DO calls SCAN to evaluate the expressions in the operand field. These values are placed into a DO table entry. NSRT is called to define the label on the DO line which is given a value of zero. The location of the DO label value is placed in the DO table entry as is the current procedure level. The pointer to the current DO table entry is set. If the DO appears in a PROC or FUNC reference, the next line is obtained by calling SKIP and its location is moved to the DO table. If the line is outside any PROC or FUNC reference, it is obtained by calling TEXT. If a void DO appears outside a PROC reference, the DO line and the line following it are ignored. A void DO within a PROC reference results in the number of lines to 'do' being skipped.

PROGRAMMING TECHNIQUES: DO is an open routine assembled as part of ASSEMBLR.

CALLING SEQUENCE: DO is called by executing the directive branch table and returns to the main line code.

MEMORY REQUIREMENTS: 144_8 cells

SUBROUTINES USED:

SCAN TEXT
NSRT EPRNT
SKIP

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Repeat lines of code (DOAGN)

PURPOSE: To repeat lines of code in the range of a DO directive.

ACTION: DOAGN decrements the line count for the DO and, if all lines have not been finished, returns to LN4 in the main line code to continue processing the line. When all lines have been done, the DO count is decremented and, if not finished, the origin of the first line to do is reset, the DO label value is incremented, and control goes to DOAGN to count the lines. As each line is done, DODEC is called to decrement the line counts on outer active DOs. When the DO count reaches zero, the lines to skip are skipped and the DO table pointer is reset to the next lower DO level.

PROGRAMMING TECHNIQUES: DOAGN is an open routine assembled as part of ASSEMBLR.

CALLING SEQUENCE: DOAGN is called by the line control code when an active DO is detected and control returns to the line code.

MEMORY REQUIREMENTS: 124_8 cells

SUBROUTINES USED:

DODEC	GCW
TEXT	SKIP
SWITCH	

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

IDENTIFICATION: Decrement DO line counts (DODEC).

PURPOSE: To decrement the DO line counts of active DOs outside the current DO.

ACTION: DODEC steps through the DO table, decrementing the DO line counts for active DOs outside the current DO but at the same PROC level.

PROGRAMMING TECHNIQUES: DODEC is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE: BRM DODEC

MEMORY REQUIREMENTS: 24_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Process PROC and FUNC reference lines (PRL and FNRL)

PURPOSE: To process the line referencing a PROC or FUNC.

ACTION: The procedure level is first tested to determine if space exists to process the line; and, if space is not available, the routine is exited. The temporary procedure level, PLVT, is incremented, a flag is set to indicate whether the reference was to a PROC or FUNC, WLLVL is set equal to PLVT, and the symbol table direction is reversed. PLV and the location counter are saved and the pass is set to first. DFLST is called to define the parameter list elements. PLV is set to PLVT; BYT, ECW, and TERM are saved. The starting location of the sample is obtained from the calling NAME item, and SWITCH is called to reset the origin of the next byte of input. The old input position is saved for resuming later. PLB is called to obtain the PROC line label, and a test is made to see if this is a 1- or 2-pass procedure. If it is a 1-pass procedure, PASS is set equal to PASS at the referencing level. The list item is constructed using the element linkage established by DFLST, the list identification is obtained from the PROC label by PLB, and the value is associated with the NAME item. NSRT is called to place the list item into the symbol table. SKIP is called to bypass the remainder of the PROC line.

PROGRAMMING TECHNIQUES: The temporary setting of the procedure level PLVT before defining the list parameters is done so that the parameters will be inserted into the correct table position. Since a FUNC reference is possible before finishing the definition of the list, the PLV flag must remain unaltered so that characters

PROGRAMMING TECHNIQUES: (cont.)

are obtained and labels processed, etc., in the normal manner; however, it must be remembered that this additional reference must be completed. These routines are open routines assembled as part of ASSEMBLR.

CALLING SEQUENCE:

PRL is called by the main line code when a procedure reference is encountered. FNRL is called by SCANC when a function reference is encountered. Both return to the main line code.

MEMORY REQUIREMENTS:

225_8 cells total

SUBROUTINES USED:

DFLST	PLB
SWITCH	GCW
GBSL	SKIP
NSRT	

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Define procedure reference parameters (DFLST)

PURPOSE: To define the parameters on the PROC or FUNC reference line.

ACTION: DFLST calls SCAN to evaluate the parameters and NSRT to place them in the symbol table. The parameters are linked as they are inserted, and the number of parameters and the location of the first parameter are saved to define the list item. A skeletal list item is placed in ICW and VALU.

PROGRAMMING TECHNIQUES: DFLST is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE: BRM DFLST

MEMORY REQUIREMENTS: 42_8 cells

SUBROUTINES USED: SCAN
NSRT

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 0420
Catalog No. 9300: 6120

IDENTIFICATION: Process END directives (END)

PURPOSE: To process END directives.

ACTION: END may process an END directive in any of four ways:

1. The END occurs while processing procedure sample. The procedure sample count is decremented. If it is zero, the sample processing flag is reset and control goes to SA2. If the count is not zero, MOVE is called to save the line and control goes to the line code at LINSYM.

2. The END occurs while processing a procedure reference. The label, if any, on the line is defined by calling NSRT. SWITCH is called to reset the origin of the next byte, SCRP is called to purge symbols, and the parameters which were saved when the PROC was referenced are restored. The label on the calling line is defined, if still present, and control is returned to the main line code.

3. The END occurs while processing a FUNC reference. SCAN is called to evaluate the END line expression. SWITCH is called to restore the origin for the next byte, SCRP is called to purge symbols, the parameters saved at the time of reference are restored, and control goes to SCANR in the SCANC routine to continue the expression evaluation.

4. The END is the end of the program. If no further outputs are wanted, control is returned to the monitor. The END line label is defined by calling NSRT. The symbolic output file is closed. The binary output file is opened, and the external symbol and programmed operators

ACTION:
(cont.)

are output as are the programmed operator reference items. When the external symbols have been completed, they are flushed from the output buffer by calling FLUSH. PAS2 is now loaded.

PROGRAMMING
TECHNIQUES: END is an open routine assembled as part of ASSEMBLR.

CALLING
SEQUENCE: END is called by executing the directive branch table.

MEMORY
REQUIREMENTS: 250_8 cells

SUBROUTINES
USED: End-of-file routine for symbolic output

NSRT FLUSH
OPEN MOVE
GTLBL SWITCH
OUTP SCRP

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 0420
Catalog No. 9300: 6120(

IDENTIFICATION: Process FORM reference lines and programmed operator references (FRL and POP)

PURPOSE: To increment the location counter for FORM and POP references.

ACTION: If the FORM is single-precision, CCINC is set to 1; if it is double-precision, CCINC is set to 2. POP sets CCINC to 1.

PROGRAMMING TECHNIQUES: These are open routines assembled as part of ASSEMBLR.

CALLING SEQUENCE: These routines are entered from the main line code when a POP or FORM reference is encountered. Control returns to the line code at LNFRM.

MEMORY REQUIREMENTS: 22_8 cells total

SUBROUTINES USED: N e

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 900 Series: 042016
9300: 612001

IDENTIFICATION: Process BCD and TEXT directives (BCD and TEXTR)

PURPOSE: To process BCD or TEXT directive lines.

ACTION: If the first operand field character is a < (less than), a > (greater than) character is set as the line terminator and the character count is set to 56_{10}. If it is not, the terminating character is set as 100_8 (impossible), and SCAN is called to obtain the character count.

Characters, obtained by calling GET, are then packed into WORD. When WORD is filled, EDIT is called to output the data words. Characters are thus obtained and output until the count reaches zero or the terminating character is encountered. Blanks (60_8) are translated to 12_8 if the entry is at BCD.

PROGRAMMING TECHNIQUES: This routine is an open routine assembled as part of ASSEMBLR.

CALLING SEQUENCE: This routine is entered by executing the directive transfer table. Return is to the main line code.

MEMORY REQUIREMENT: 116_8 cells

SUBROUTINES USED:

PEEK GBSL
GCW GET
SCAN LBTST
EDIT

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 04201
Catalog No. 9300: 61200

IDENTIFICATION: Process lines of sample (SAM).

PURPOSE: To process lines of procedure or function sample.

ACTION: SAM calls PLB to process the label field of the line and GCW to obtain the operation field. The operation field is tested for a PROC, FUNC, NAME, or END directive; and, if it is any of these, control goes to the appropriate routine. All other lines are processed by SAM starting at SA2.

If this is the first line following the procedure NAME, the location of the PROC line is determined either by using the parameters set by MOVE and the current byte size and mask or by using PRPOS if this line appears in a procedure reference. This origin of the PROC line is then set in each of the NAME items associated with the PROC. If the line appears outside any procedure reference, the PROC line is moved to storage by calling MVPRC and the current line is moved by calling MOVE. If the line is not the first line following the NAME lines and is outside any procedure reference, only the current line is moved. The label on the line is ignored.

PROGRAMMING TECHNIQUES: SAM is an open routine assembled as part of ASSEMBLR.

CALLING SEQUENCE: SAM is called from the main line code when the sample processing flag is ON. Control returns to the main line code at LINSYM.

MEMORY REQUIREMENTS: 151_8 cells

SUBROUTINES USED:	PLB	MVPRC
	GCW	MOVE
	GBSL	

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 04201
Catalog No. 9300: 01200

IDENTIFICATION: Process NAME directives (NAME)

PURPOSE: To process NAME directives.

ACTION: If the name does not follow a PROC, FUNC, or other NAME line, it is an error. If the sample level count is greater than 1, the line is moved to the procedure storage area. The value associated with the NAME is evaluated by calling SCAN, and a NAME item is constructed in LBL through LBL+3. If the value of the operand field is a list, a flag is set in the second word of the NAME item reflecting this fact. NSRT is called to place the NAME item reflecting this fact. NSRT is called to place the NAME item in symbol table. When NAME items are built, they are linked together so that the setting of the procedure origin can be expedited later.

PROGRAMMING TECHNIQUES: NAME is an open routine assembled as part of ASSEMBLR.

CALLING SEQUENCE: NAME is entered by executing the directive branch table. Control returns to the main line code at LINSYM.

MEMORY REQUIREMENTS: 131_8 cells

SUBROUTINES USED:

GBSL	NSRT
SCAN	MFOI

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Move lines to sample storage (MVPRC and MOVE)

PURPOSE: To move a line to the user sample storage area.

ACTION: MVPRC is used to move PROC lines to sample storage; MOVE moves all other lines. Bytes are moved until an end-of-line mark is encountered. If the value of a byte (modulo the byte size) is zero, the byte size and related mask are incremented. If table overflow occurs and no symbols have been entered in the lower side of the symbol table, LOWER is moved up to make more space available.

PROGRAMMING TECHNIQUES: MVPRC sets the origin at which to obtain bytes and then branches to MOVE to move the PROC line. The routines are relocatable routines assembled as part of ASSEMBLR.

CALLING SEQUENCE:

```
BRM    MOVE
    or
BRM    MVPRC
```

MEMORY REQUIREMENTS: 111_8 cells total

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 900 Series: 0420
9300: 6120

IDENTIFICATION: Reset line origins (SWITCH)

PURPOSE: To reset the origin to obtain the next byte of input to the location specified.

ACTION: SWITCH packs the current position into the format of a NAME item pointer word. The contents of the A register, in the same format, are unpacked and used to set the new parameters. The contents of the B register is placed in CHAD. On exit the old position in packed format is in the A register and the old CHAD contents in the B register.

PROGRAMMING TECHNIQUES: SWITCH is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE:

```
Current location to A register
CHAD to B register
BRM    SWITCH
```

MEMORY REQUIREMENTS: 41_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Get symbols from the dictionary (GTLBL)

PURPOSE: To find the dictionary entry, given the location of a symbol table entry, and to move the dictionary characters to the location given by WORD.

ACTION: GTLBL determines the dictionary location associated with an item at the location given by the index at entry. The dictionary characters are then obtained by calling GEC and packed into the locations addressed by WORD.

PROGRAMMING TECHNIQUES: GTLBL assumes that TPFLG has been set to indicate the direction of the entry and that a dictionary pointer word follows the symbol table item specified. GTLBL is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE:

Direction of symbol table entry to TPFLG
Location of entry to index register
Location for resulting label to WORD
BRM GTLBL

MEMORY REQUIREMENTS: 50_8 cells

SUBROUTINES USED: GEC

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Search symbol table (SRCH)

PURPOSE: To search for a specified entry in the symbol table.

ACTION: SRCH examines the entries in the symbol table chain for an item of the same type and at the same level as the current item. If SRFG is positive, the type fields are not compared. On exit SRLNK points to the item found or the last item in the chain if the item is not found.

PROGRAMMING TECHNIQUES: SRCH assumes that the direction of the table, WLLVL and TBLOC, are all properly set when SRCH is entered. SRCH is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE:

Location of item to search for to index
BRM SRCH
item-not-found return
item-found return

MEMORY REQUIREMENTS: 66_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Insert items into the symbol table (NSRT)

PURPOSE: To store items into the symbol table and to connect their linkages if they are not already in the table.

ACTION: If PLVT does not equal WLLVL, the table direction is reversed. If the item is not a list element, SRCH is called to determine if it is already in the table; and if it is, a test is made to determine whether the item has the same value as the current item. If the values are not the same and both items are not absolute values or mnemonics, the error bit is set in both items and the item is reinserted. If the items are different but absolute values of the same length, the new item value replaces the old value. If they differ and are of different lengths, the new item is inserted as though the old item had not been found. Special tests are made when inserting mnemonic items for the presence of a programmed operator reference item. If one is found in the chain, it is given the subtype of seven so that it will not be output. SRLNK is set to the location of any new item inserted. A pointer to the byte table entry is inserted following items not at level one of the symbol table. The table direction is restored before exit.

PROGRAMMING TECHNIQUES: NSRT is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE: Location of item to insert to index.
BRM NSRT

MEMORY REQUIREMENTS:	310_8 cells
SUBROUTINES USED:	SRCH

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Purge items from the symbol table (SCRP)

PURPOSE: To remove local symbols and lists from the symbol table at the conclusion of a procedure reference.

ACTION: SCRP steps through the symbol table entries for the current procedure level and reconnects the chain linkages to bypass these symbols. The pointers to the next available cell in the table are reset to the table origin of this level. The direction of the table is reversed.

PROGRAMMING TECHNIQUES: SRCP is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE: BRM SCRP

MEMORY REQUIREMENTS: 102_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Store characters into buffer (EDC)

PURPOSE: To store a character into the buffer location specified.

ACTION: EDC subtracts 60_8 from the character furnished in the A register, positions it to the correct character position as determined by EDC1, and stores it into the location addressed by EDWW by adding to memory.

PROGRAMMING TECHNIQUES: EDC assumes the buffer has been cleared to blanks (60_8) prior to being called. EDC is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE:

Character to A register
BRM EDC

MEMORY REQUIREMENTS: 21_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Initialize word and character positions to store characters (EDS)

PURPOSE: To set parameters EDC1 and EDWW for the EDC routine.

ACTION: EDS uses the control word supplied in the A register to set the shift parameter, EDC1, and the buffer location, EDWW, for storing characters. The control word has the following format:

character (9 bits)	word position (15 bits)
0 — 8	9 — 23

Character is 0 through 3, giving character positions from left to right to store next character.

Word position is the address in buffer to store next character.

PROGRAMMING TECHNIQUES: EDS is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE:
Control word to A register
BRM EDS

MEMORY REQUIREMENTS: 6 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 04201(
Catalog No. 9300: 61200

IDENTIFICATION: Output a universal binary output item (OUTP)

PURPOSE: To store an output item into the output buffer and call the I/O routines to write the record.

ACTION: If no binary output has been requested, OUTP exits without taking any action. If the output buffer is full or if the output item is of a different type than the previous item, OUTP calls FLUSH to empty the buffer. The item is stored into the output buffer and the relocation flags and checksum are accumulated for it. OUTP uses a branch table to transfer to the correct segment of code to process the various item types.

PROGRAMMING TECHNIQUES: OUTP is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE:

Item type to CTYP
Output data to WORD[†]
BRM OUTP

MEMORY REQUIREMENTS: 141_8 cells

SUBROUTINES USED:

FLUSH
RESET

[†]For item types one and two, WORD addresses the location of the datum. The relocation flags WMODR, WMODC, and WMODP indicate whether the datum has the particular relocation quality.

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Empty to binary output buffer (FLUSH)

PURPOSE: To empty the binary output buffer.

ACTION: FLUSH sets the accumulated relocation words into the buffer, sets the output card type into the control word, and calls OUTPUT with each word in the buffer to write the data on the binary output file. When all words are out, FLUSH calls WRITE to write the record.

PROGRAMMING TECHNIQUES: FLUSH is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE: BRM FLUSH

MEMORY REQUIREMENTS: 62_8 cells

SUBROUTINES USED: OUTPUT
WRITE

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Initialize control cells for OUTP (RESET)

PURPOSE: To initialize the OUTP control cells for a new record.

ACTION: RESET initializes the output control cells for a new record.

PROGRAMMING TECHNIQUES: RESET is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE: BRM RESET

MEMORY REQUIREMENTS: 13_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 900 Series: 042016
9300: 612001

IDENTIFICATION: Process the PAGE directive (PAGE)

PURPOSE: To process the PAGE directive.

ACTION: PAGE calls EDTST to determine whether listing is being done. If so, the HOME routine is called.

PROGRAMMING TECHNIQUES: PAGE is an open routine assembled as part of ASSEMBLR.

CALLING SEQUENCE: PAGE is called by executing the directive branch table. PAGE returns to the line code at LINSYM.

MEMORY REQUIREMENTS: 4 cells

SUBROUTINES USED: EDTST
HOME

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Print errors and symbolics (EPRNT)

PURPOSE: To cause the error flags and symbolics to be written on the listing.

ACTION: If listing is to be performed on the line, EPRNT calls EDE to edit error flags and PRNT to print the line.

PROGRAMMING TECHNIQUES: EPRNT is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE: BRM EPRNT

MEMORY REQUIREMENTS: 10_8 cells

SUBROUTINES USED:
EDTST
EDE
PRNT

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Process DED directives (DED)

PURPOSE: To process DED directive lines.

ACTION: DED calls SCAN to evaluate the expressions in the operand field. The values are then placed into WORD and WORD+1, a double-precision FORM control word is moved to WRD2 and WRD2+1, and the data are output by calling EDIT. Before editing the data, LBTST is called to define any waiting label. When all expressions have been evaluated, control goes to LNFRM. The location counter is incremented by 2 for each expression output.

PROGRAMMING TECHNIQUES: DED is an open routine assembled as part of ASSEMBLR.

CALLING SEQUENCE: DED is called by executing the directive branch table. Control is returned to the main line code at LNFRM.

MEMORY REQUIREMENTS: 46_8 cells

SUBROUTINES USED:

SCAN	LBTST
MFOI	EDIT
RDPI	GLOV

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 04201
Catalog No. 9300: 61200

IDENTIFICATION: Obtain the low order value word set by SCAN (GLOV)

PURPOSE: To get the low order value word from VALU or VALU+1.

ACTION: If the value is a 3-word address item as indicated by ICW, the value is taken from VALU+1; if not, the value is taken from VALU. The resulting value is in the A register, and the contents of ICW is in the B register at exit.

PROGRAMMING TECHNIQUES: GLOV is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE: BRM GLOV

MEMORY REQUIREMENTS: 13_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Make 3-word address items (M3WAI)

PURPOSE: To expand an address item into three words if necessary.

ACTION: M3WAI removes bits 9 through 23 of the VALU and stores it in VALU+1. Bits 9 through 23 of VALU are set to zero. The item length is set to three words.

PROGRAMMING TECHNIQUES: M3WAI is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE: BRM M3WAI

MEMORY REQUIREMENTS: 13_8 words

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 900 Series: 04201(
9300: 61200

IDENTIFICATION: Negate floating point numbers (FLN)

PURPOSE: To negate a floating point number and to normalize the result.

ACTION: FLN takes the negative of the floating point number at the location given by the index register, by complementing the fraction and adding 1. The resulting number is then normalized as needed to correct for overflow or underflow.

PROGRAMMING TECHNIQUES: FLN is a relocatable routine assembled as part of ASSEMBLR

CALLING SEQUENCE: Location of floating point number to the index register.

BRM FLN

MEMORY REQUIREMENTS: 35_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Multiply floating point numbers (FLM)

PURPOSE: To obtain the product of two floating point numbers.

ACTION: FLM obtains the product of $a2^n * b2^m$ as $ab2^{n+m}$ where the product ab is taken as $(h+i)*(j+k) \cong hj + ij + hk$.

If the result is over- or under-normalized, the resulting exponent is corrected.

PROGRAMMING TECHNIQUES: FLM is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE:
Location of multiplicand to A register
Location of multiplier to B register
BRM FLM
The product replaces the multiplicand.

MEMORY REQUIREMENTS: 106_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Test item relocations (RELTST)

PURPOSE: To determine the relocation status of a pair of items.

ACTION: If the item addressed by MODA is relocatable, RELTST sets bit 22 of the A register; if the item at ICW is relocatable, RELTST sets bit 23 of the A register. A is stored in RELFG.

PROGRAMMING TECHNIQUES: RELTST is designed to be used by SCAN and is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE: BRM RELTST

MEMORY REQUIREMENTS: 26_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Convert numeric strings to binary values (CNVRT)

PURPOSE: To convert numeric items to binary values.

ACTION: CNVRT converts numeric character strings to their binary value by successive multiplications of 8 or 10 (depending on the value of the first character). GEC is used to fetch the characters of the string. Results are left in VALU, VALU1, and VALU2. If the leading character is a dot, the number is converted to floating point by dividing the integer by the appropriate powers of 10 and calculating the exponent. The DPDIV routine is used to perform the divisions. All floating point fractions so calculated are left in normalized form.

PROGRAMMING TECHNIQUES: CNVRT is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE:

Byte table entry for numeric byte to ECW
BRM CNVRT

MEMORY REQUIREMENTS: 170_8 cells

SUBROUTINES USED:

GEC
DPDIV

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 04201
Catalog No. 9300: 61200

IDENTIFICATION: Perform double-precision divisions (DPDIV)

PURPOSE: To divide the contents of the A and B registers by the contents of the location addressed by the index register and maintain maximum precision.

ACTION: DPDIV divides the contents of the A and B registers by the single-precision divisor addressed by the index register. The remainders are then divided and that remainder divided. The resulting quotient is normalized.

PROGRAMMING TECHNIQUES: DPDIV assumes that both the dividend and divisor are normalized and leaves the results in the same format. DPDIV is a relocatable routine assembled as part of the ASSEMBLR.

CALLING SEQUENCE:

Double-precision dividend to A and B registers
Location of divisor to X register
BRM DPDIV

MEMORY REQUIREMENTS: 36_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Evaluate expressions (SCAN)

PURPOSE: To evaluate an expression and leave the control word of the results in the B register and ICW and the value in VALU through VALU+2 with the low order portion of the value in the A register.

ACTION: SCAN obtains the items in the expression by calling GIT and the connectors by calling GNC. The items and connectors are obtained in pairs. If the connector obtained is of higher priority than the previous connector, the item value and the connector are saved in the SCAN operations table and the table pointers are incremented. If the connector is of lower priority, the previous operation is performed. The type of operation to be performed is determined by executing an operations branch table which carries control to the various operation routines.

The operation routines perform the indicated operation between a pair of operands, one of which is located in the SCAN operations table and the other of which is located in ICW and VALU to VALU+2. The first item is always the one in the SCAN operations table.

The result of the operation is placed in the cells ICW and VALU to VALU+2, and the pointers to the operations branch table are decremented to point to the previous item.

When a leading = (equals) mark is encountered, SCAN sets a flag indicating that the expression is to be interpreted as a literal. A leading * (asterisk) mark causes a flag to be set which will result in the value of the expression being interpreted as an address quantity. This * flag will also be output

ACTION: (cont.)

with the resulting value so that expressions of the format P (*i) may be properly interpreted.

When the last operation to be performed is a terminator, SCAN tests for the literal flag being set; and if it is, SCAN takes zero as the value of the expression. If the * flag is ON, the value is converted to a 3-word address value and the sign bit of VALU is set.

Upon exit the contents of TERM are

0 if blank terminated
1 if comma terminated
2 if right parenthesis terminated

The cell STAR contains 1 if the expression had a leading * and 0 otherwise.

PROGRAMMING TECHNIQUES:

The SCAN operations table is really a series of short tables each of which is indirectly addressed. The table positions are incremented or decremented by incrementing or decrementing the indirect pointer words. SCAN is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE:

Byte table entry for the first byte of the expression to ECW
BRM SCAN

MEMORY REQUIREMENTS:

1266_8 cells

SUBROUTINES USED:

GCW	MIFT	RELTST
GIT	GLOV	FLM
GNC	GLOP	FLN

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Get next item of an expression (GIT)

PURPOSE: To obtain the value of an item and store it in VALU through VALU+2 with its control word in ICW.

ACTION: GIT evaluates the following types of items:

alphanumeric constants
location counter reference
function references
subscripted symbols (parameter)
symbolic items
numeric items
lists
list count
parenthetical expression

1. Alphanumeric constants are evaluated by obtaining the characters from the dictionary which comprise the constant and packing them together into VALU and VALU+1.

2. The value of location counter references is the current value of CC.

3. Function references are evaluated by calling SCANC (which in turn calls FNRL).

4. Subscripted symbols are evaluated by calling SCANC to obtain the subscripts and by stepping through the list to extract the proper element.

5. Symbolic items are obtained by picking the item out of the symbol table. If a symbolic item is undefined, the resulting value is taken as zero.

ACTION:
(cont.)

6. Numeric items are evaluated by calling CNVRT. If a numeric item is a mixed floating point number, the integer and fractional parts are obtained by separate calls on CNVRT and the parts are then combined by GIT.

7. Lists are obtained by inserting the elements of the list into the symbol table by calling SCANC and by generating a list item giving the location of the first element and the number of elements.

8. List counts are evaluated by finding the appropriate list item and extracting the element count from it.

9. Parenthetical expressions are obtained by calling SCANC. GIT does not differentiate between lists and parenthetical expressions; the distinction is made by SCANC.

PROGRAMMING TECHNIQUES: GIT works with the SCAN and SCANC routines and is really a major section of the overall expression evaluation processing. GIT is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE:

Byte table entry for first byte to ECW
BRM GIT

MEMORY REQUIREMENTS: 472_8 words

SUBROUTINES USED:

GCW	SCANC
GLOV	MIFT
CNVRT	GBSL
PEEK	GET

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Process lists and parenthetical expressions (SCANC)

PURPOSE: To evaluate parenthetical expressions, define elements of lists, evaluate function references, and obtain values of subscripts.

ACTION: SCANC increments all the SCAN table level pointers in anticipation of calling the SCAN routine. If the mode field of the contents of the B register is non-zero, SCANC calls FNRL to evaluate a function reference. If it is zero, SCAN is called to evaluate the expression. If there is an operator at the SCAN level at which SCANC was called, the resulting value is not taken as an element and SCANC decrements the SCAN operation table pointers and exits. (GIT takes advantage of this test when calling SCANC to obtain subscripts by setting an artificial value in the SCAN operations table.) Similarly, if the literal flag is set, the value is not a list element.

The element of a list is inserted into the symbol table by calling NSRT. The next element is obtained by calling SCAN. When all elements have been inserted and linked, SCANC constructs a list item in ICW and VALU, decrements the SCAN operation table pointers, and exits.

PROGRAMMING TECHNIQUES: SCANC is designed to be recursive with the SCAN routine. Since it automatically steps the SCAN operations table pointers, SCANC serves as the device for forcing parenthetical expressions to be completed before other operations. SCANC is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE: Control word to B register
BRM SCANC

CALLING SEQUENCE: (cont.)	FNRL returns control to SCANC at SCANR.
MEMORY REQUIREMENTS:	157_8 cells
SUBROUTINES USED:	SCAN NSRT FNRL GLOV PEEK GCW GEC

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Peek at next dictionary character (PEEK)

PURPOSE: To peek at the next character in the dictionary entry without obtaining the character. PEEK is normally used when a conditional test is needed but the contents of ECW are not to be destroyed.

ACTION: PEEK locates the dictionary entry for the byte addressed by ECW then extracts the character addressed from the dictionary. The result is left in the A register.

PROGRAMMING TECHNIQUES: PEEK is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE:

Byte table entry to ECW
BRM PEEK

MEMORY REQUIREMENTS: 24_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 04201
Catalog No. 9300: 61200

IDENTIFICATION: Get next connector (GNC)

PURPOSE: To get the operator table entry for the next connector into TERM and the A register.

ACTION: GNC obtains the next connector characters by calling GEC. The characters are left-adjusted in the A register and compared to bits 0 through 11 of the operator table OTBL. When a match is found, bits 12-23 of the OTBL entry are placed in TERM. GCW is called to get the first byte of the following item. TERM is loaded into the A register before exit.

PROGRAMMING TECHNIQUES: GNC is designed to work with SCAN and is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE:
Byte table entry for connector to ECW
BRM GNC

MEMORY REQUIREMENTS: 44_8 cells

SUBROUTINES USED:
GEC
GCW

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Get next character of a line (GET)

PURPOSE: To get the next dictionary character for a line of input.

ACTION: If the end of line has been reached, GET exits with a blank. GET gets the next character for a byte by either using blank and reducing BCNT if the string is blank or by calling GEC for nonblank strings. When the end of a byte is reached, GET gets the next byte by calling GCW. If it is blank, GBSL is also called. The character is in the A register at exit.

PROGRAMMING TECHNIQUES: GET is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE:

```
Byte table entry to ECW
BRM    GET
```

MEMORY REQUIREMENTS: 44_8 cells

SUBROUTINES USED:

GCW
GBSL
GEC

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 0420'
Catalog No. 9300: 6120(

IDENTIFICATION: Count blank string lengths (GBSL)

PURPOSE: To determine the size of blank strings.

ACTION: GBSL calls GEC to obtain the characters in the dictionary entry representing the blank count. The count is placed in BCNT.

PROGRAMMING TECHNIQUES: GBSL is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE: Byte table entry for blank byte to ECW
BRM GBSL

MEMORY REQUIREMENTS: 15_8 cells

SUBROUTINES USED: GEC

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Fetch symbol table entries (MIFT)

PURPOSE: To move the item addressed by the contents of the index register to ICW through ICW+3.

ACTION: MIFT moves four words starting at the location specified in the index register to ICW to ICW+3. If the address specified is greater than LOWER, the items are taken in descending order from the starting point. If the item moved is a 2-word address item, M3WAI is called to expand it to three words.

PROGRAMMING TECHNIQUES: MIFT is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE: Location of item to index register
BRM MIFT

MEMORY REQUIREMENTS: 27_8 cells

SUBROUTINES USED: M3WAI

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 900 Series: 0420[illegible]
9300: 6120[illegible]

IDENTIFICATION: Get low order parameter values (GLOP)

PURPOSE: To get the low order parameter word into the A register.

ACTION: If the item addressed by MODA is a 3-word address item, the value is loaded from the cell addressed by HOA; otherwise, the value is taken from the cell addressed by LOA.

PROGRAMMING TECHNIQUES: GLOP is designed to be used by the SCAN routine and is a relocatable routine assembled as part of ASSEMBLR.

CALLING SEQUENCE: BRM GLOP

MEMORY REQUIREMENTS: 11_8 cells

SUBROUTINES USED: None

SDS SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Programmed Operators (POPs)

PURPOSE: To define those instructions not common to all machines in the 900 series, but which are used in the assembly system.

ACTION: The POPs are executed individually as referred to by the system.

PROGRAMMING TECHNIQUES: There are two sets of programmed operators used, one set for the 910/925 and one for the 920/930. Both are coded in the intersection of the instruction sets so no nesting results. The transfer words in cells 100 to 117 have absolute origins; the routines to simulate the machine instructions are relocatable. In addition to the programmed operators, cell 1337 (DTAB) is set by POPs to reflect the increase in space for code needed when POPs are used. In this way better memory utilization results than if DTAB were fixed sufficient for the largest set.

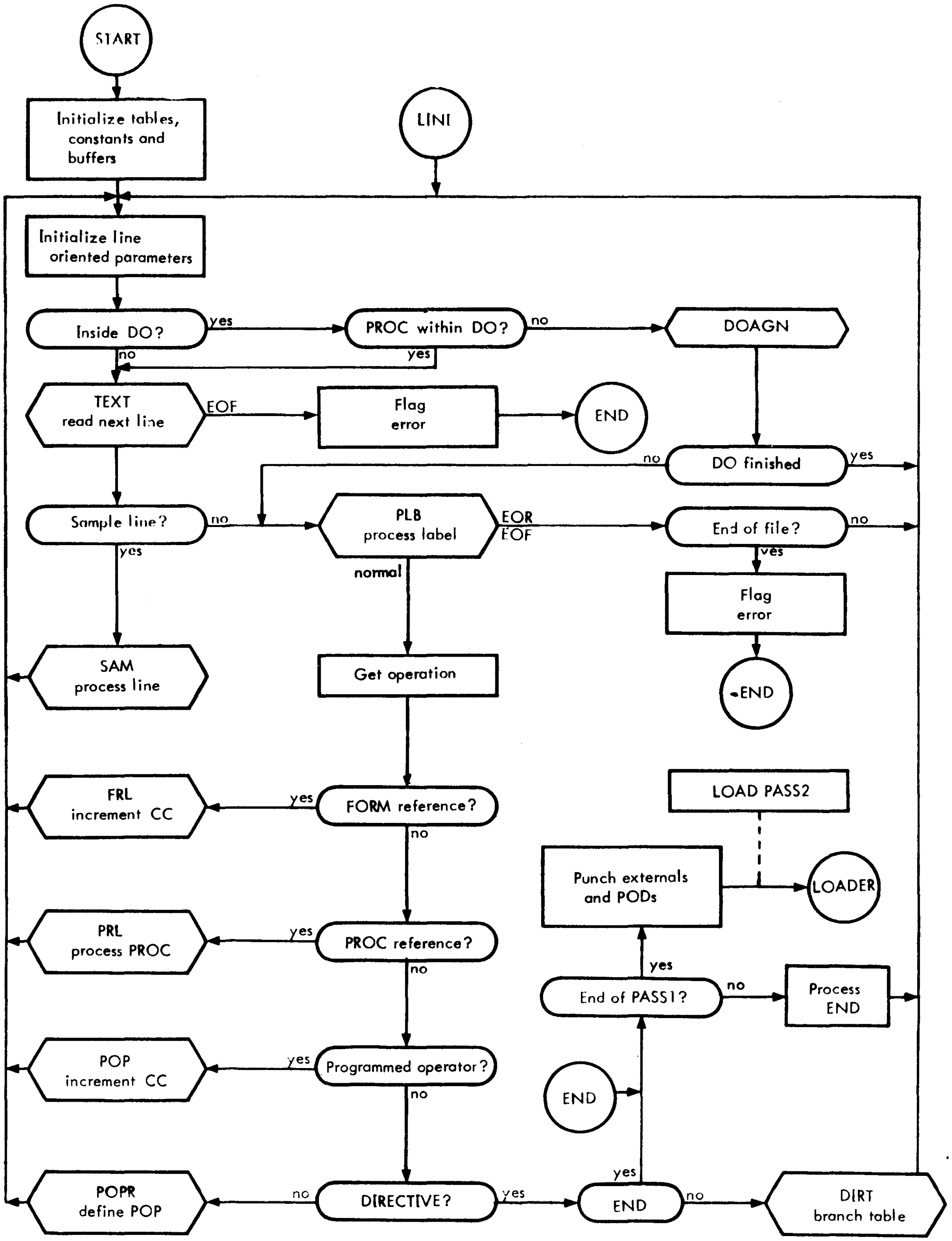
CALLING SEQUENCE: Not applicable

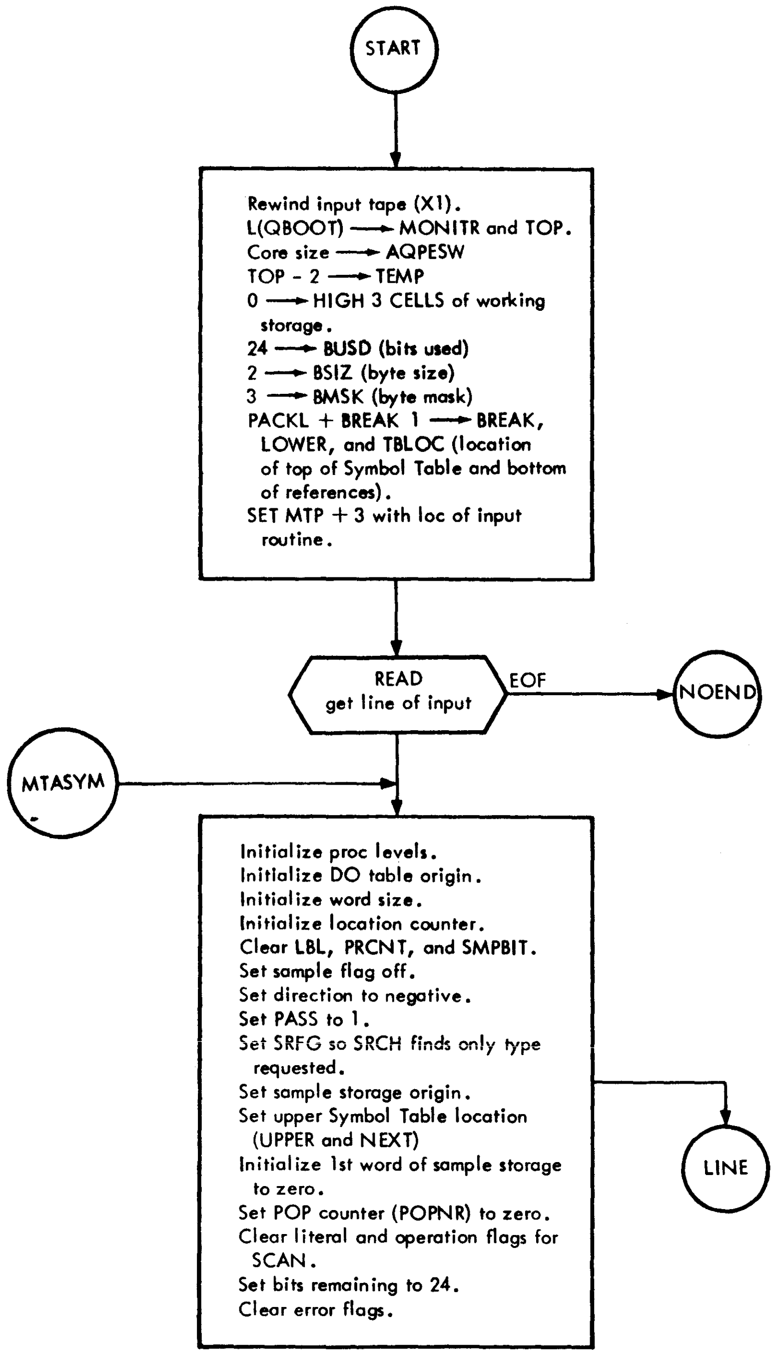
MEMORY REQUIREMENTS:

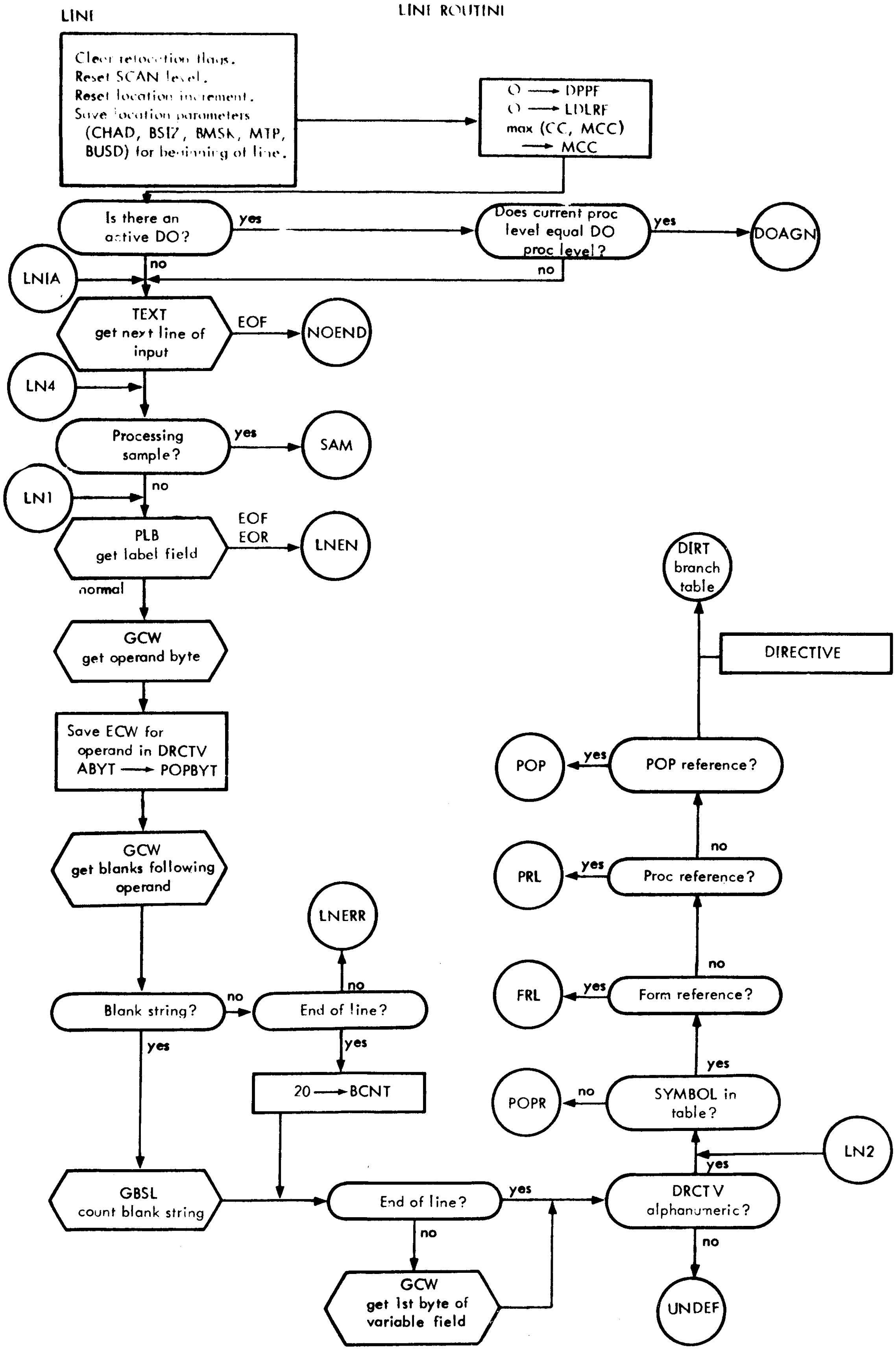
920 cells	100_8 through 126_8	plus 64_8 cells
910 cells	100_8 through 117_8	plus 257_8 cells

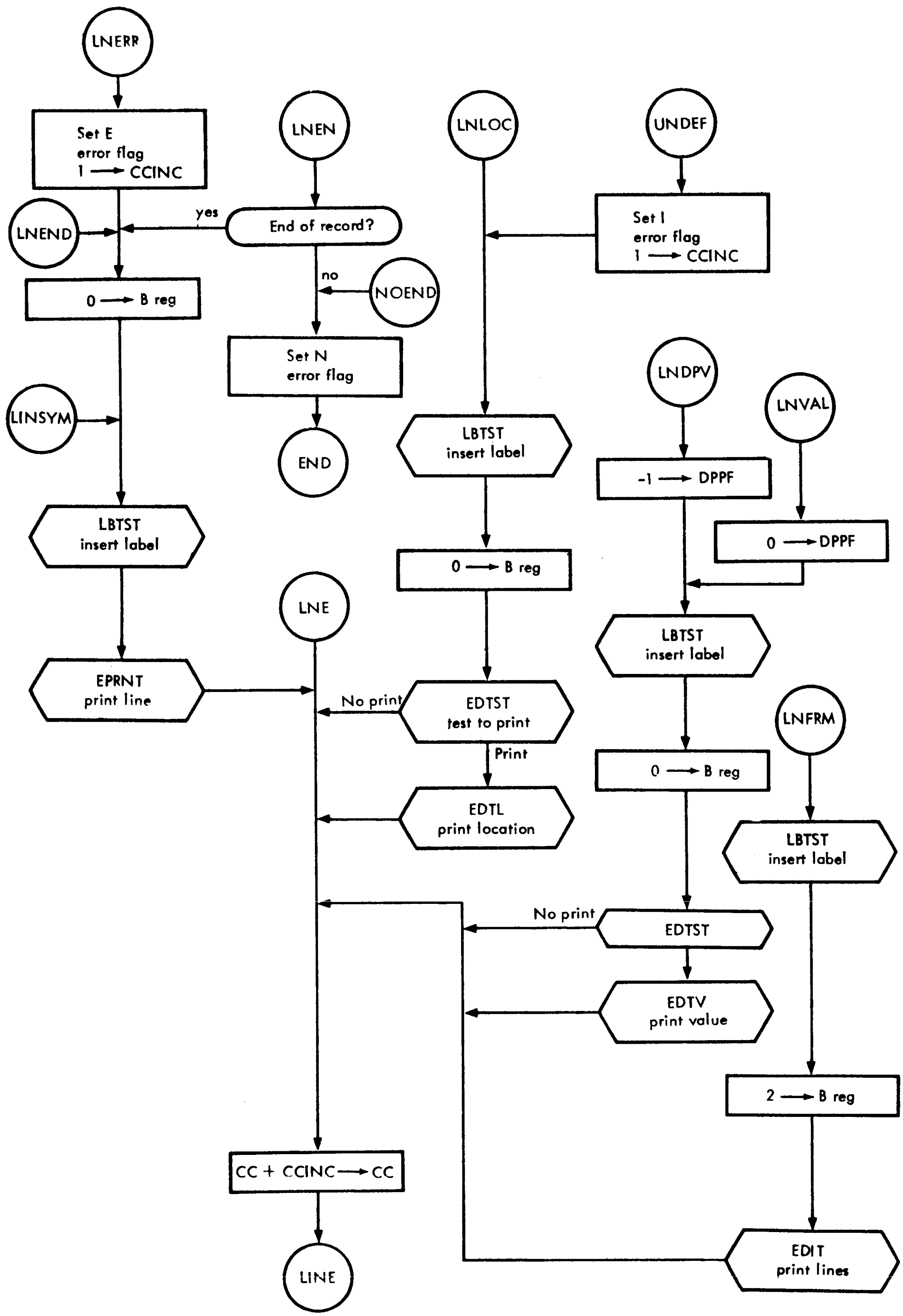
SUBROUTINES USED: Not applicable

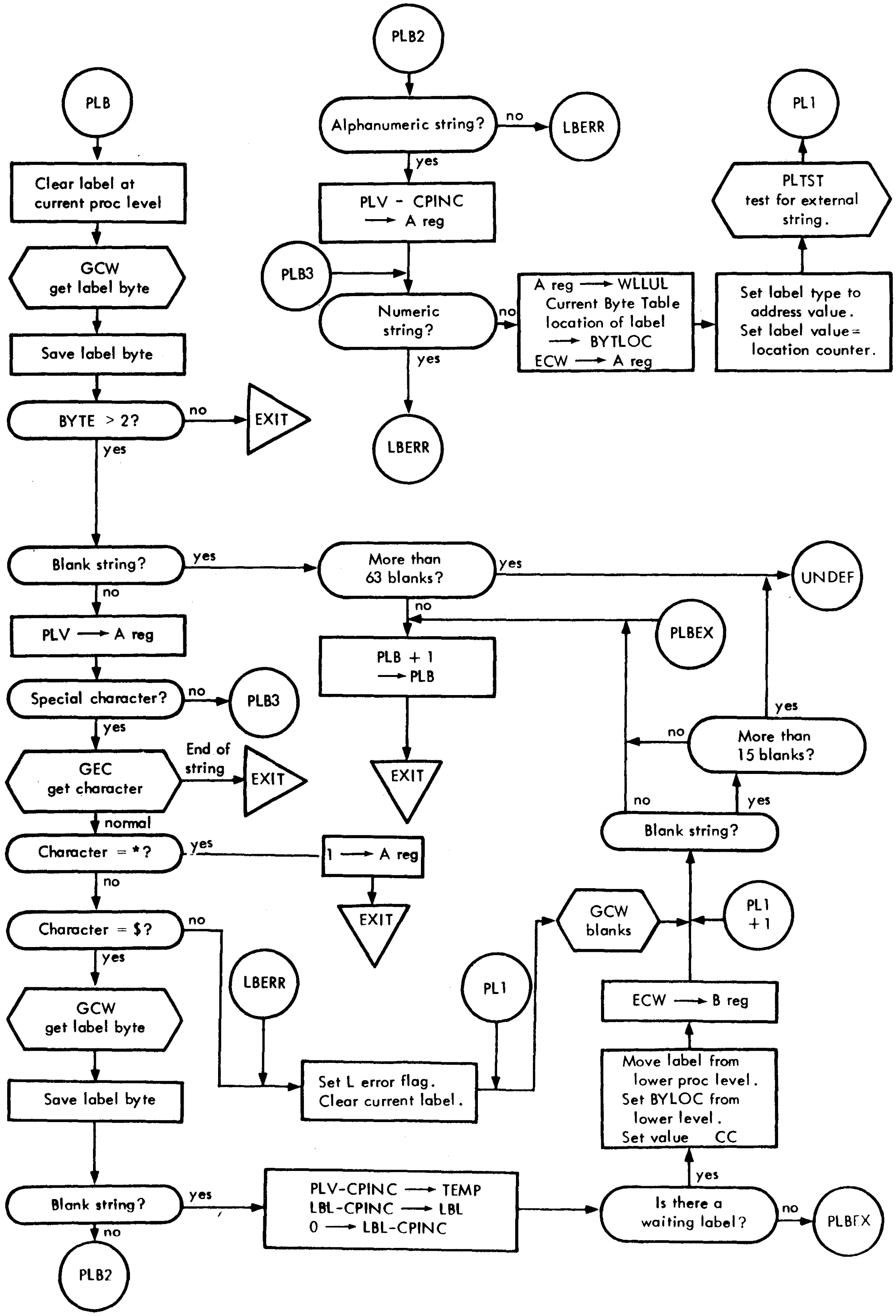
ASSEMBLR
OVERALL FLOW

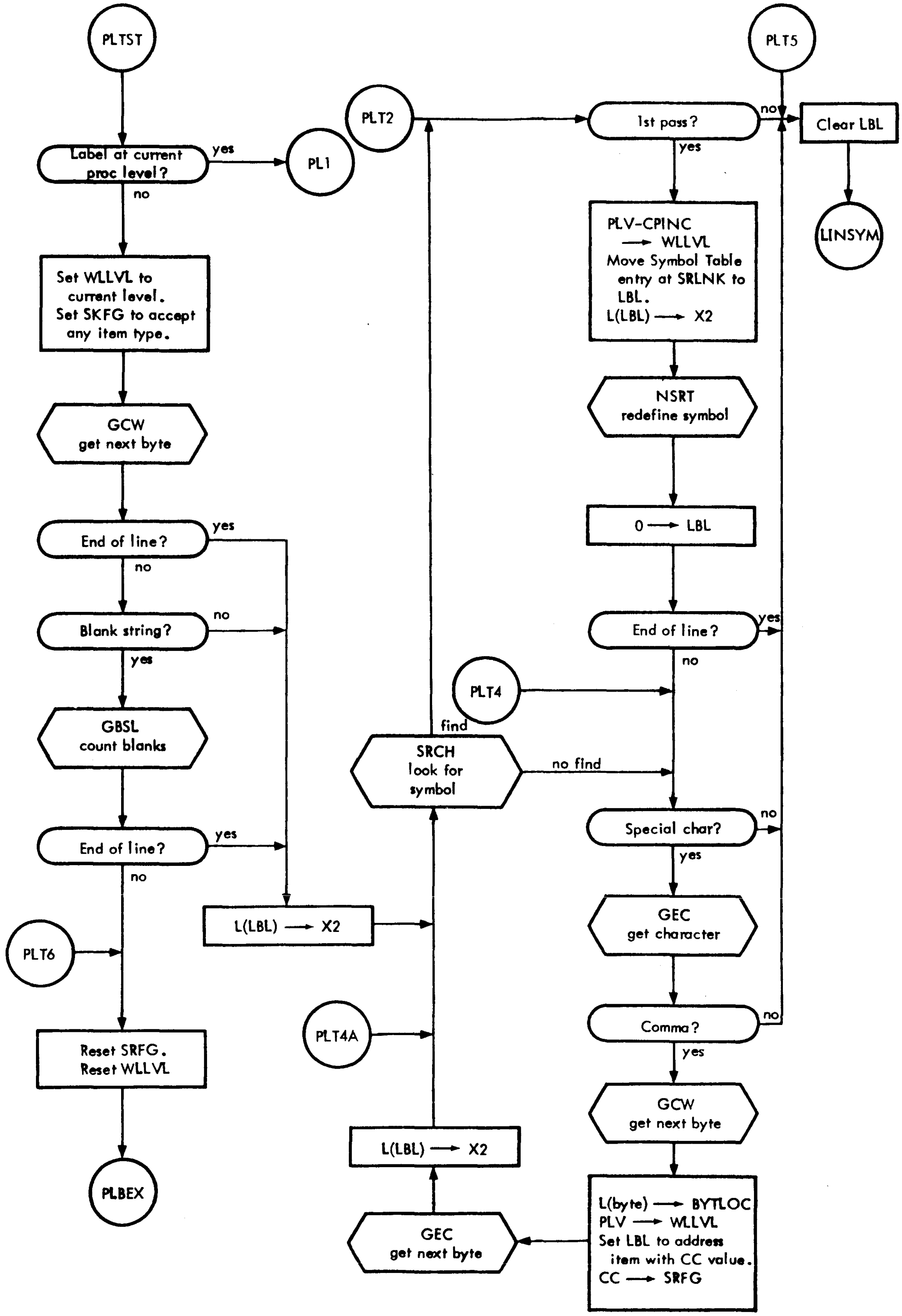


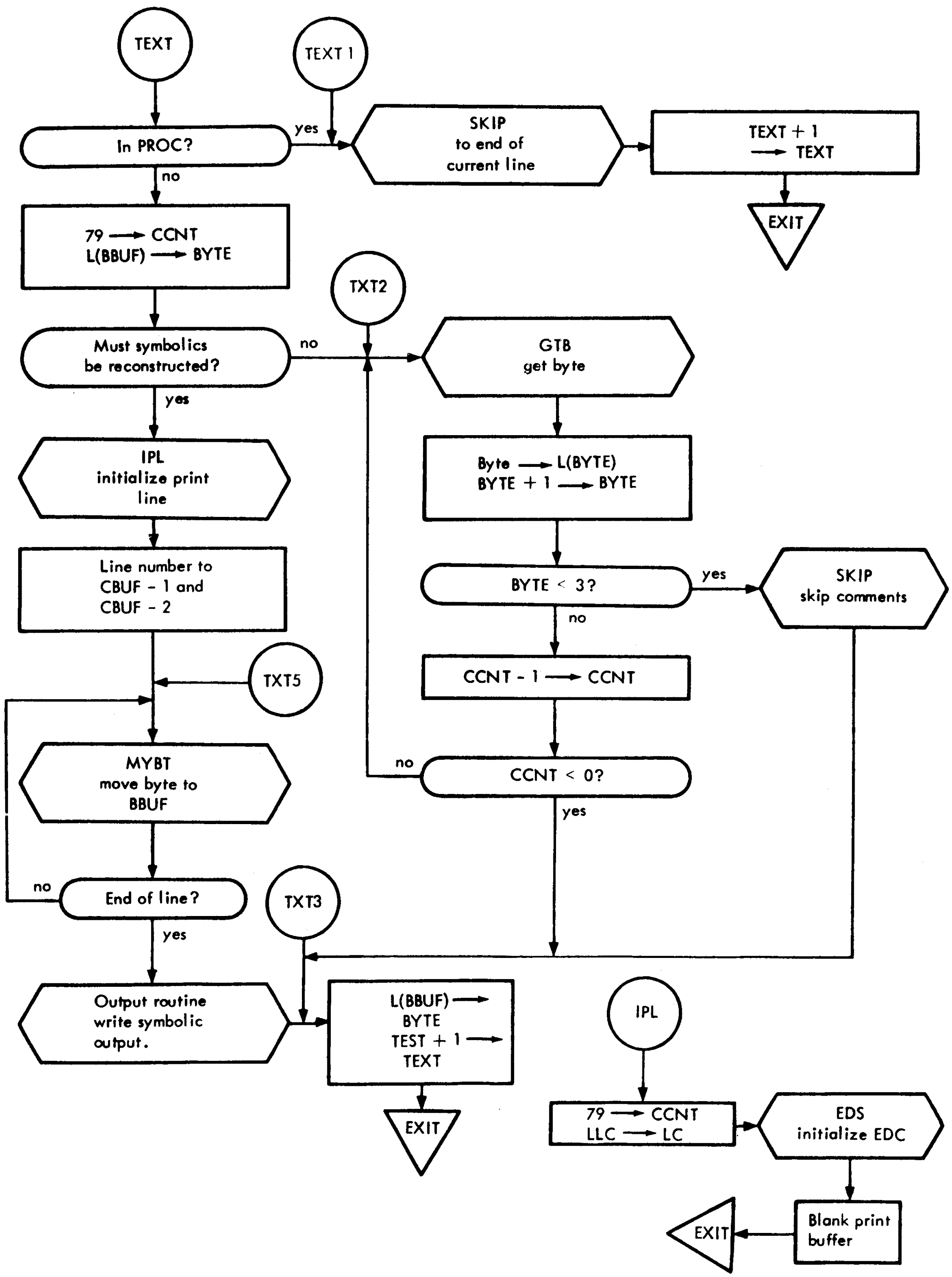

START
Rewind input tape (X1).
L(QBOOT) ⟶ MONITR and TOP.
Core size ⟶ AQPESW
TOP - 2 ⟶ TEMP
0 ⟶ HIGH 3 CELLS of working storage.
24 ⟶ BUSD (bits used)
2 ⟶ BSIZ (byte size)
3 ⟶ BMSK (byte mask)
PACKL + BREAK 1 ⟶ BREAK, LOWER, and TBLOC (location of top of Symbol Table and bottom of references).
SET MTP + 3 with loc of input routine.
READ
get line of input
EOF
NOEND
MTASYM
Initialize proc levels.
Initialize DO table origin.
Initialize word size.
Initialize location counter.
Clear LBL, PRCNT, and SMPBIT.
Set sample flag off.
Set direction to negative.
Set PASS to 1.
Set SRFG so SRCH finds only type requested.
Set sample storage origin.
Set upper Symbol Table location (UPPER and NEXT)
Initialize 1st word of sample storage to zero.
Set POP counter (POPNR) to zero.
Clear literal and operation flags for SCAN.
Set bits remaining to 24.
Clear error flags.
LINE

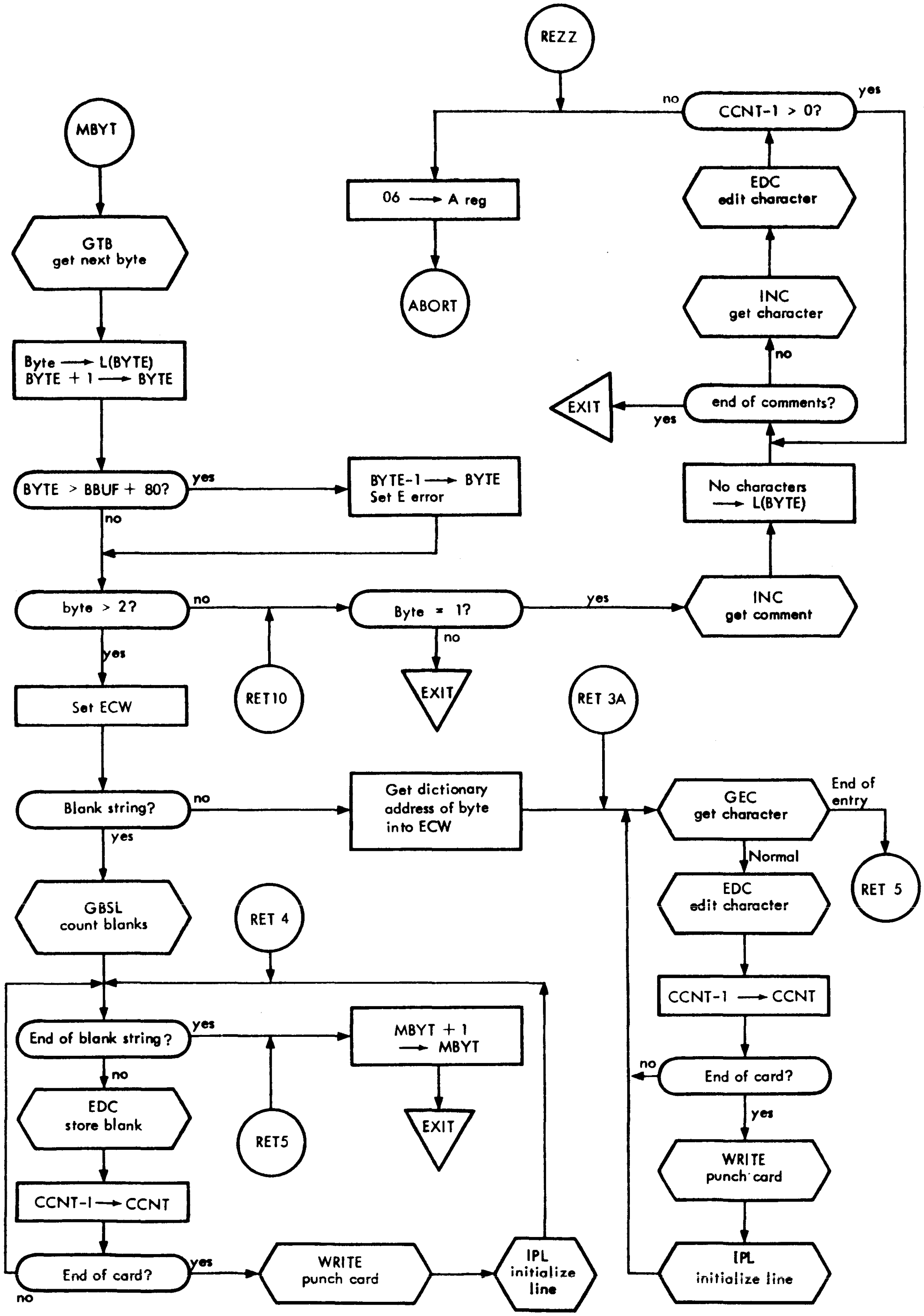

LINE
Clear relocation flags.
Reset SCAN level.
Reset location increment.
Save location parameters (CHAD, BSIZ, BMSK, MTP, BUSD) for beginning of line.
0 → DPPF
0 → LDLRF
max (CC, MCC) → MCC
Is there an active DO?
yes
Does current proc level equal DO proc level?
yes
DOAGN
no
no
LNIA
TEXT get next line of input
EOF
NOEND
LN4
Processing sample?
yes
SAM
no
LN1
PLB get label field
EOF
EOR
LNEN
normal
GCW get operand byte
Save ECW for operand in DRCTV
ABYT → POPBYT
GCW get blanks following operand
Blank string?
no
End of line?
no
LNERR
yes
20 → BCNT
yes
GBSL count blank string
End of line?
yes
no
GCW get 1st byte of variable field
DRCTV alphanumeric?
no
UNDEF
yes
LN2
SYMBOL in table?
no
POPR
yes
Form reference?
yes
FRL
no
Proc reference?
yes
PRL
no
POP reference?
yes
POP
DIRECTIVE
DIRT branch table

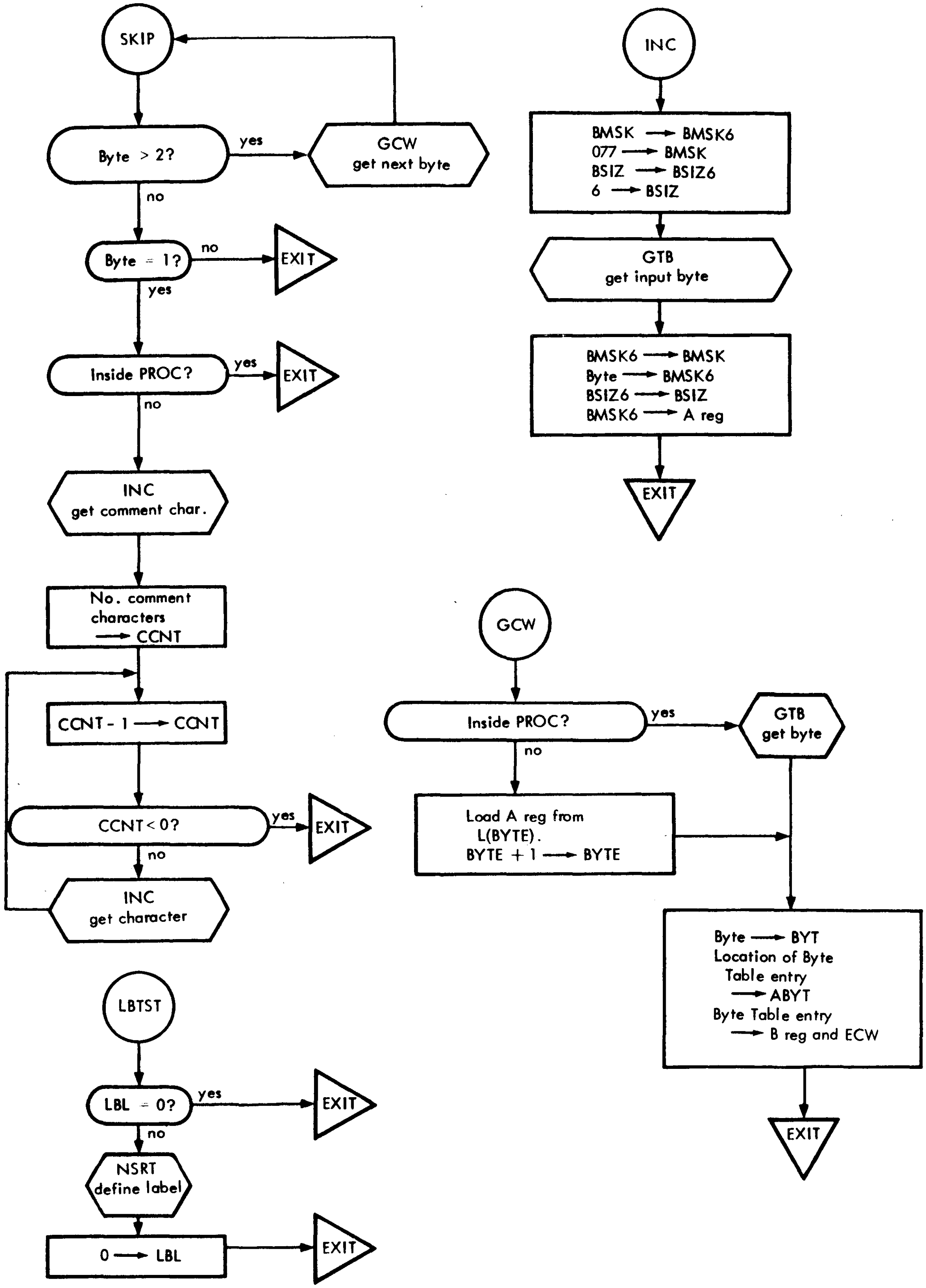

LNERR
Set E
error flag
1 → CCINC
LNEN
End of record?
yes
no
LNEND
NOEND
0 → B reg
Set N
error flag
END
LINSYM
LBTST
insert label
EPRNT
print line
LNE
LNLOC
LBTST
insert label
0 → B reg
EDTST
test to print
No print
Print
EDTL
print location
UNDEF
Set I
error flag
1 → CCINC
LNDPV
-1 → DPPF
LNVAL
0 → DPPF
LBTST
insert label
0 → B reg
EDTST
No print
EDTV
print value
LNFRM
LBTST
insert label
2 → B reg
EDIT
print lines
CC + CCINC → CC
LINE

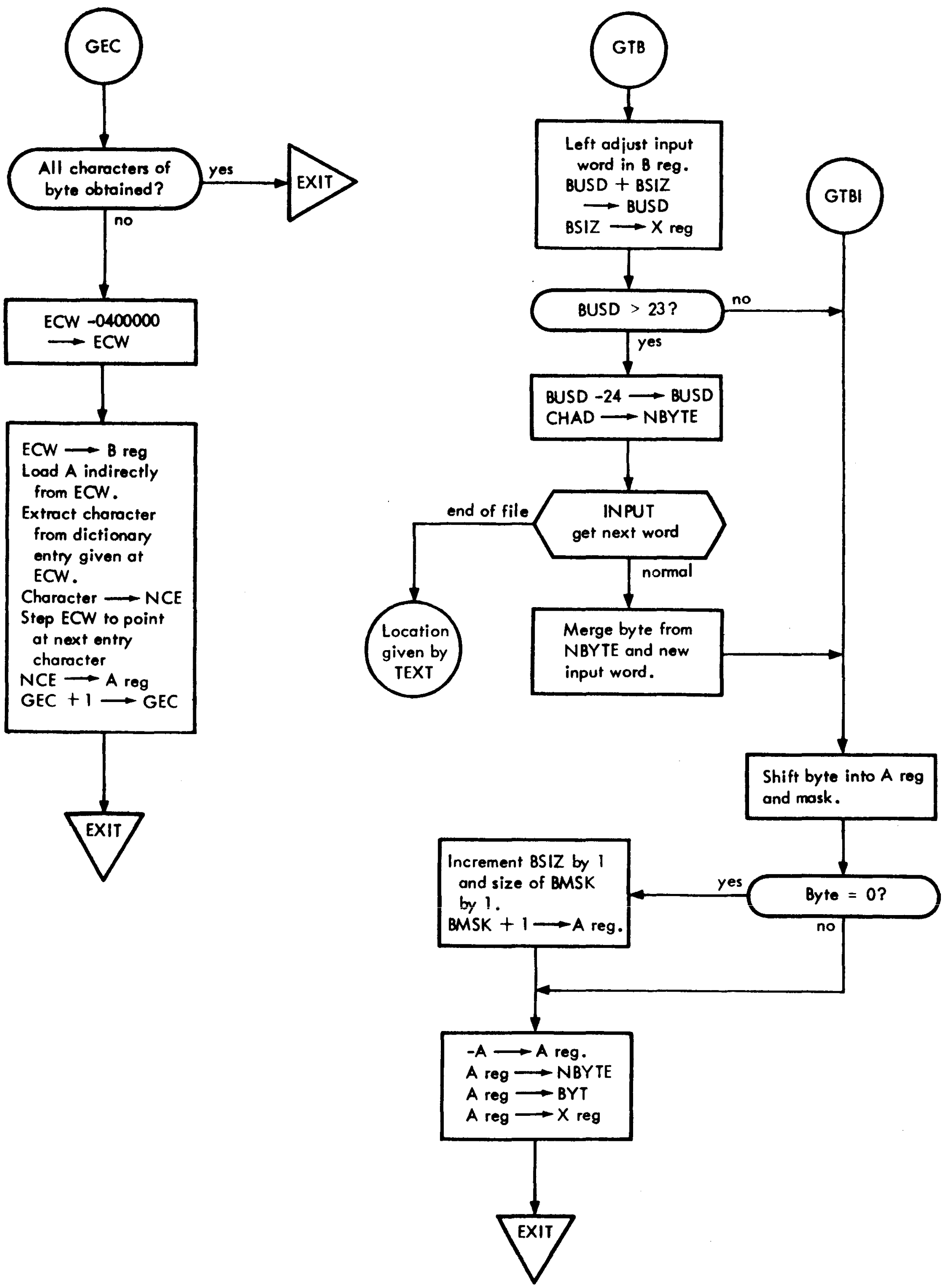

PLB2
Alphanumeric string?
no
LBERR
yes
PLV - CPINC → A reg
PLB3
Numeric string?
no
A reg → WLLUL
Current Byte Table location of label → BYTLOC
ECW → A reg
Set label type to address value.
Set label value = location counter.
PLTST
test for external string.
PL1
yes
LBERR
PLB
Clear label at current proc level
GCW
get label byte
Save label byte
BYTE > 2?
no
EXIT
yes
Blank string?
yes
More than 63 blanks?
yes
UNDEF
no
PLV → A reg
PLBEX
PLB + 1 → PLB
Special character?
no
PLB3
yes
More than 15 blanks?
GEC
get character
End of string
EXIT
EXIT
normal
Blank string?
Character = *?
yes
1 → A reg
no
EXIT
GCW
blanks
PL1
+1
Character = $?
no
yes
LBERR
PL1
ECW → B reg
GCW
get label byte
Set L error flag.
Clear current label.
Move label from lower proc level.
Set BYLOC from lower level.
Set value CC
Save label byte
Blank string?
yes
PLV-CPINC → TEMP
LBL-CPINC → LBL
0 → LBL-CPINC
Is there a waiting label?
no
PLBEX
PLB2

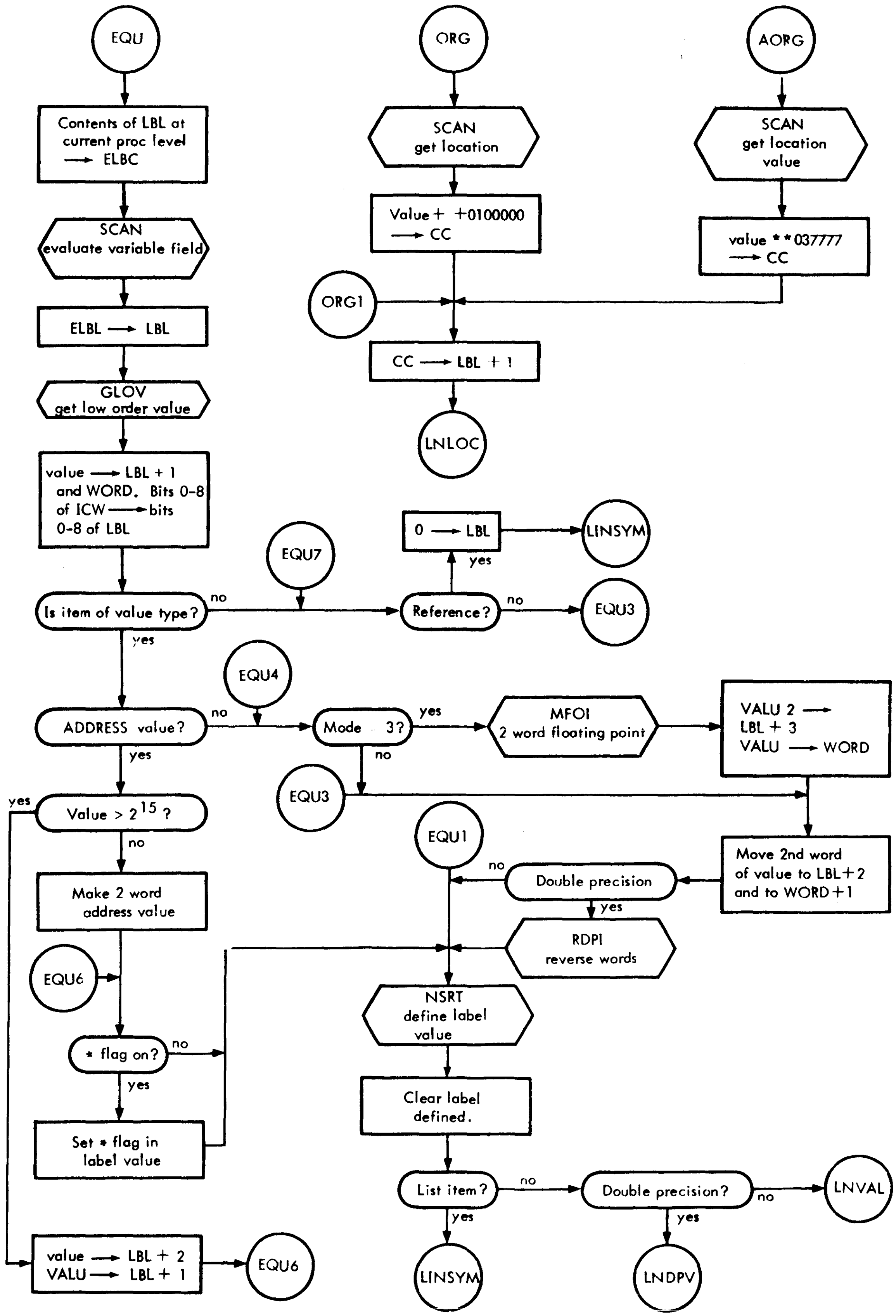

PLTST
Label at current proc level?
yes
PL1
no
Set WLLVL to current level. Set SKFG to accept any item type.
GCW get next byte
End of line?
yes
no
Blank string?
no
yes
GBSL count blanks
End of line?
yes
no
PLT6
Reset SRFG. Reset WLLVL
PLBEX
L(LBL) ⟶ X2
PLT2
find
SRCH look for symbol
no find
PLT4A
L(LBL) ⟶ X2
GEC get next byte
PLT5
1st pass?
no
yes
Clear LBL
LINSYM
PLV-CPINC ⟶ WLLVL
Move Symbol Table entry at SRLNK to LBL.
L(LBL) ⟶ X2
NSRT redefine symbol
0 ⟶ LBL
End of line?
yes
no
PLT4
Special char?
no
yes
GEC get character
Comma?
no
yes
GCW get next byte
L(byte) ⟶ BYTLOC
PLV ⟶ WLLVL
Set LBL to address item with CC value.
CC ⟶ SRFG

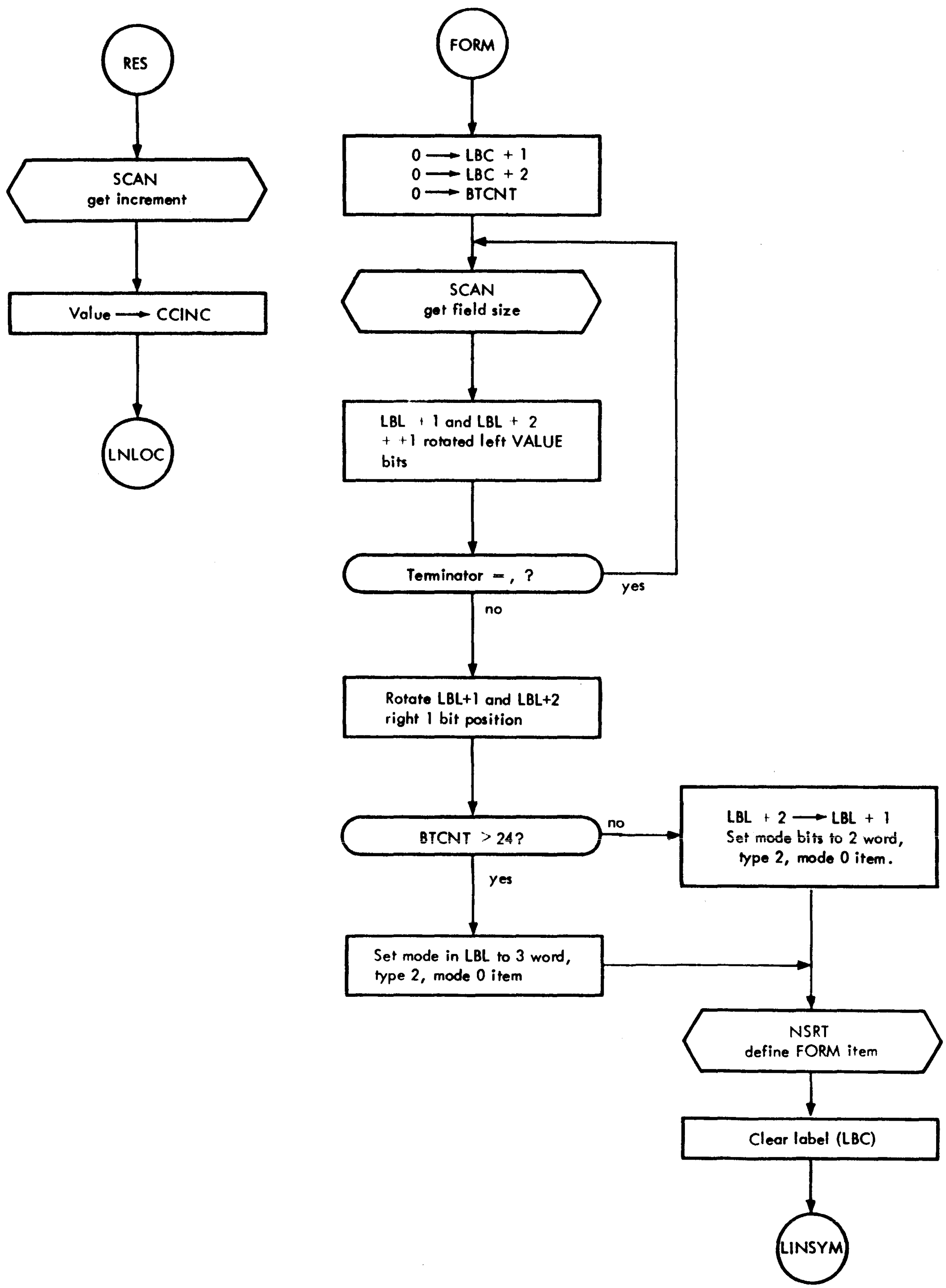

TEXT
TEXT 1
In PROC?
yes
no
SKIP
to end of
current line
TEXT + 1
→ TEXT
EXIT
79 → CCNT
L(BBUF) → BYTE
TXT2
Must symbolics
be reconstructed?
no
yes
GTB
get byte
IPL
initialize print
line
Byte → L(BYTE)
BYTE + 1 → BYTE
Line number to
CBUF - 1 and
CBUF - 2
BYTE < 3?
yes
SKIP
skip comments
no
CCNT - 1 → CCNT
TXT5
MYBT
move byte to
BBUF
no
CCNT < 0?
yes
no
End of line?
TXT3
yes
Output routine
write symbolic
output.
L(BBUF) →
BYTE
TEST + 1 →
TEXT
IPL
EXIT
79 → CCNT
LLC → LC
EDS
initialize EDC
Blank print
buffer
EXIT

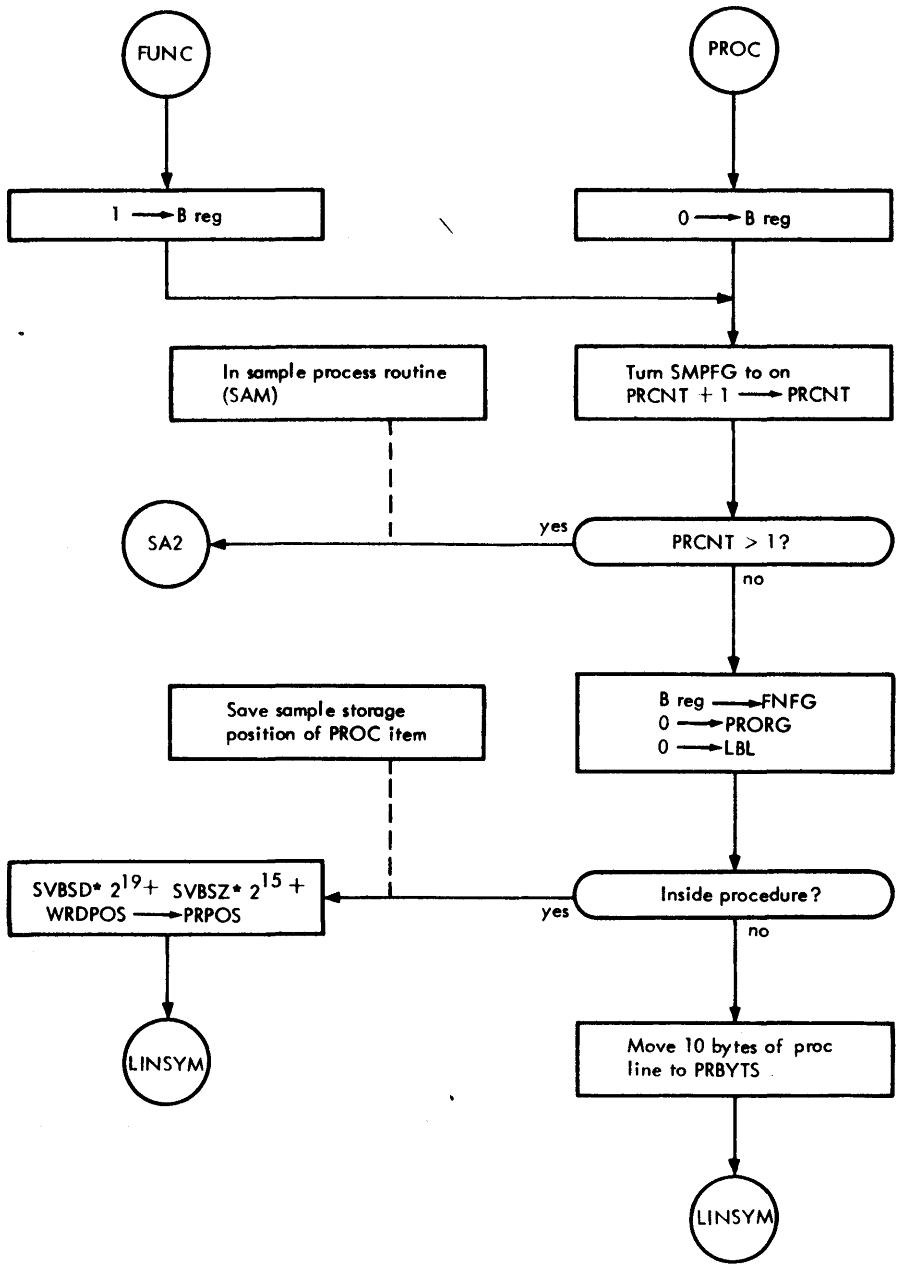

REZZ
CCNT-1 > 0?
no
yes
06 → A reg
ABORT
EDC
edit character
INC
get character
no
end of comments?
yes
EXIT
No characters
→ L(BYTE)
MBYT
GTB
get next byte
Byte → L(BYTE)
BYTE + 1 → BYTE
BYTE > BBUF + 80?
yes
no
BYTE-1 → BYTE
Set E error
byte > 2?
no
yes
Byte = 1?
yes
no
INC
get comment
Set ECW
RET10
EXIT
RET 3A
Blank string?
no
yes
Get dictionary
address of byte
into ECW
GEC
get character
End of
entry
Normal
RET 5
GBSL
count blanks
RET 4
EDC
edit character
CCNT-1 → CCNT
End of blank string?
yes
no
MBYT + 1
→ MBYT
EDC
store blank
RET5
EXIT
End of card?
no
yes
WRITE
punch card
CCNT-1 → CCNT
End of card?
yes
no
WRITE
punch card
IPL
initialize
line
IPL
initialize line

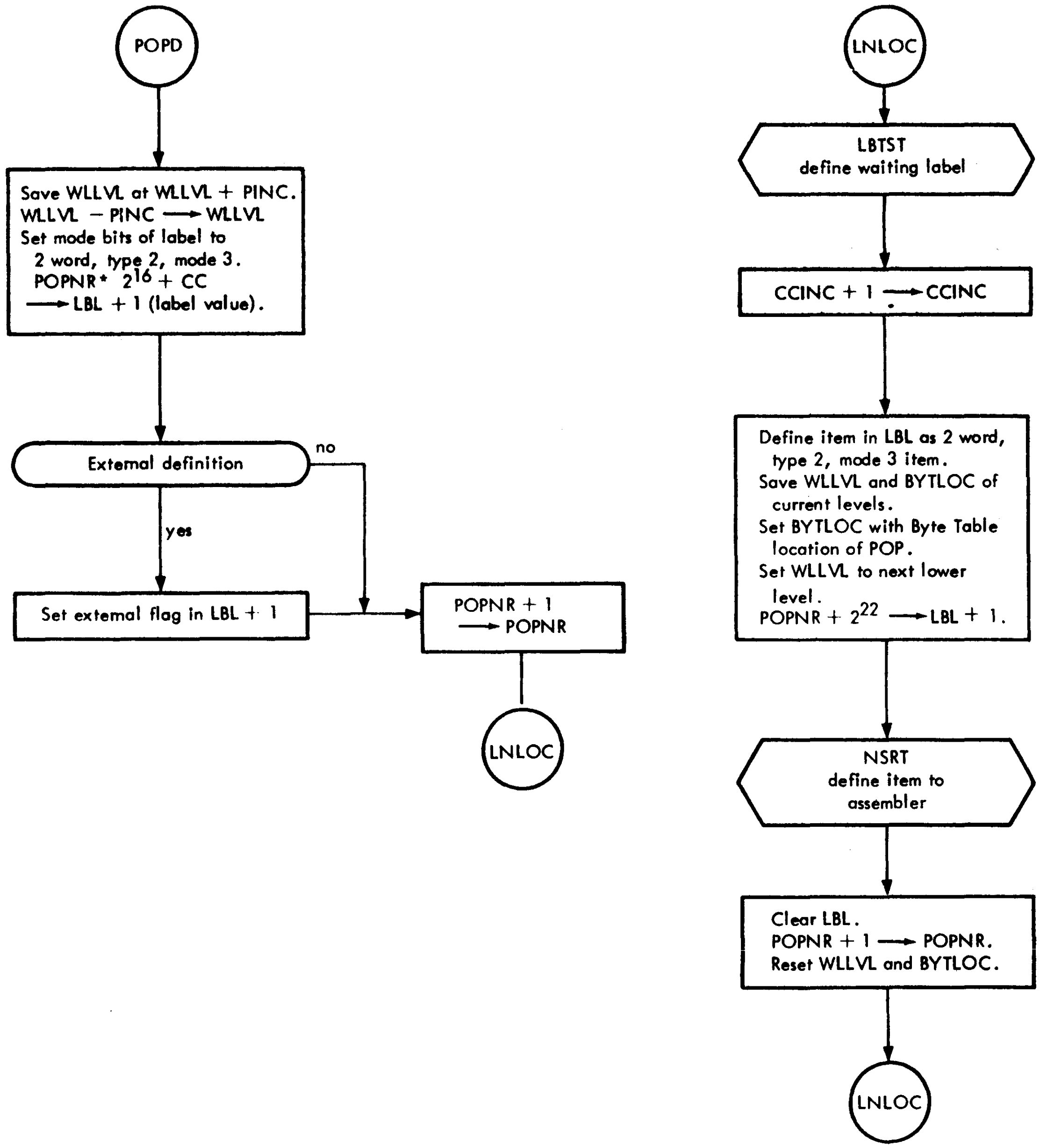

SKIP
Byte > 2?
yes
GCW
get next byte
no
Byte = 1?
no
EXIT
yes
Inside PROC?
yes
EXIT
no
INC
get comment char.
No. comment
characters
→ CCNT
CCNT - 1 → CCNT
CCNT < 0?
yes
EXIT
no
INC
get character
LBTST
LBL = 0?
yes
EXIT
no
NSRT
define label
0 → LBL
EXIT
INC
BMSK → BMSK6
077 → BMSK
BSIZ → BSIZ6
6 → BSIZ
GTB
get input byte
BMSK6 → BMSK
Byte → BMSK6
BSIZ6 → BSIZ
BMSK6 → A reg
EXIT
GCW
Inside PROC?
yes
GTB
get byte
no
Load A reg from
L(BYTE).
BYTE + 1 → BYTE
Byte → BYT
Location of Byte
Table entry
→ ABYT
Byte Table entry
→ B reg and ECW
EXIT

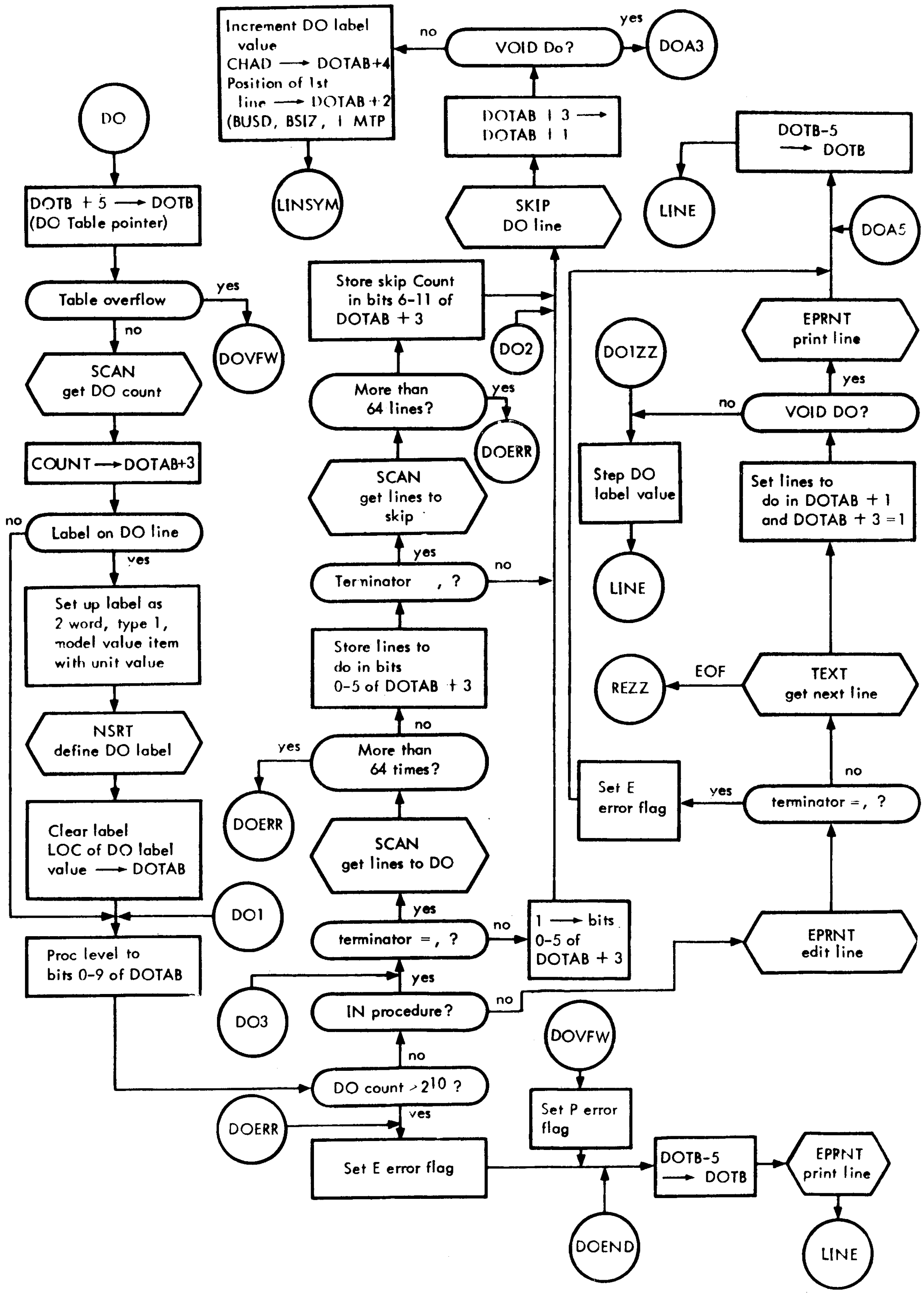

GEC
All characters of byte obtained?
yes
EXIT
no
ECW -0400000 → ECW
ECW → B reg
Load A indirectly from ECW.
Extract character from dictionary entry given at ECW.
Character → NCE
Step ECW to point at next entry character
NCE → A reg
GEC + 1 → GEC
EXIT
GTB
Left adjust input word in B reg.
BUSD + BSIZ → BUSD
BSIZ → X reg
GTBI
BUSD > 23?
no
yes
BUSD -24 → BUSD
CHAD → NBYTE
end of file
INPUT get next word
normal
Location given by TEXT
Merge byte from NBYTE and new input word.
Shift byte into A reg and mask.
Increment BSIZ by 1 and size of BMSK by 1.
BMSK + 1 → A reg.
yes
Byte = 0?
no
-A → A reg.
A reg → NBYTE
A reg → BYT
A reg → X reg
EXIT

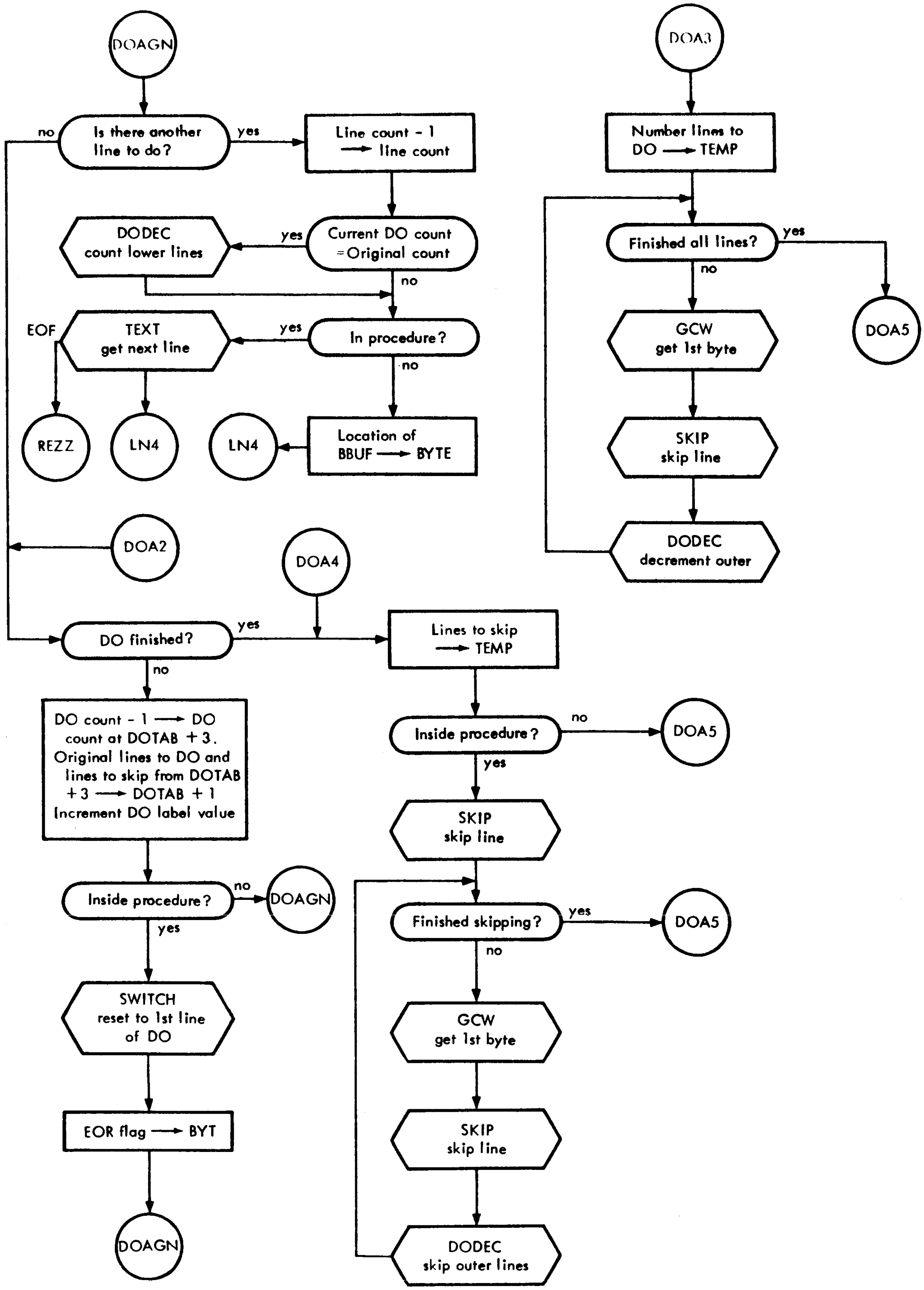

EQU
Contents of LBL at current proc level → ELBC
SCAN evaluate variable field
ELBL → LBL
GLOV get low order value
value → LBL + 1 and WORD. Bits 0-8 of ICW → bits 0-8 of LBL
Is item of value type?
no
yes
EQU7
Reference?
yes
no
0 → LBL
LINSYM
EQU3
ADDRESS value?
no
yes
EQU4
Mode 3?
yes
no
MFOI 2 word floating point
VALU 2 → LBL + 3 VALU → WORD
EQU3
Value > 2^15 ?
yes
no
Make 2 word address value
EQU6
* flag on?
no
yes
Set * flag in label value
value → LBL + 2 VALU → LBL + 1
EQU6
EQU1
Double precision
no
yes
Move 2nd word of value to LBL+2 and to WORD+1
RDPI reverse words
NSRT define label value
Clear label defined.
List item?
no
yes
Double precision?
no
yes
LNVAL
LINSYM
LNDPV
ORG
SCAN get location
Value + +0100000 → CC
ORG1
CC → LBL + 1
LNLOC
AORG
SCAN get location value
value ** 037777 → CC

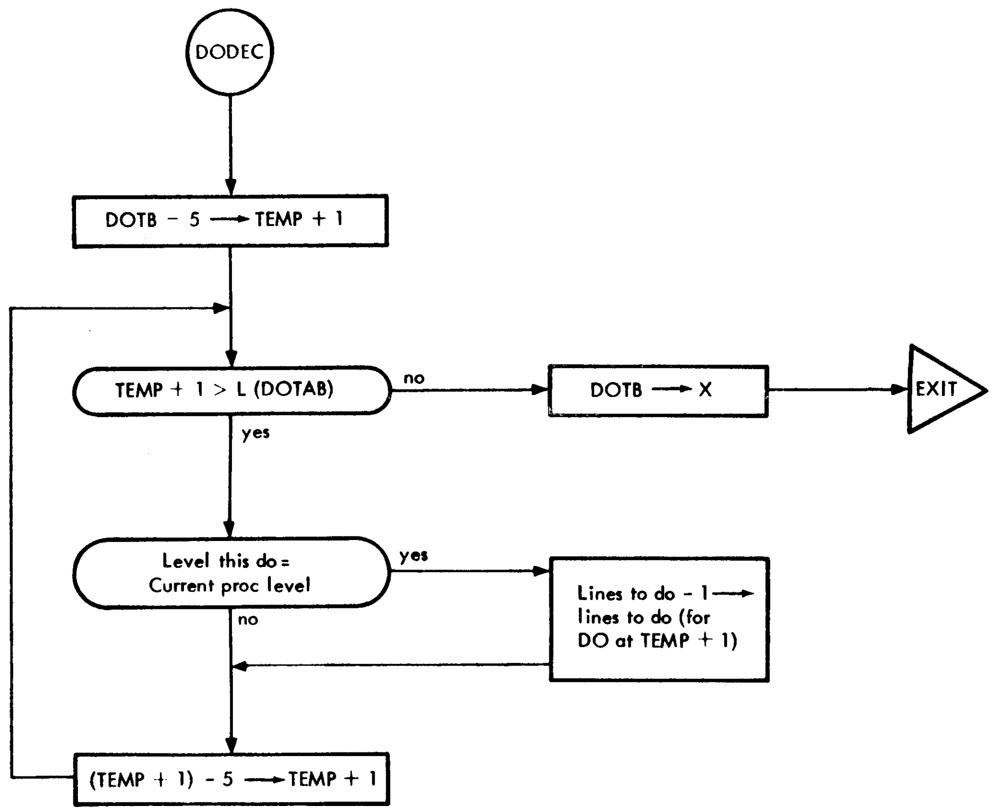

RES
SCAN
get increment
Value ⟶ CCINC
LNLOC
FORM
0 ⟶ LBC + 1
0 ⟶ LBC + 2
0 ⟶ BTCNT
SCAN
get field size
LBL + 1 and LBL + 2
+ +1 rotated left VALUE
bits
Terminator = , ?
yes
no
Rotate LBL+1 and LBL+2
right 1 bit position
BTCNT > 24?
no
yes
LBL + 2 ⟶ LBL + 1
Set mode bits to 2 word,
type 2, mode 0 item.
Set mode in LBL to 3 word,
type 2, mode 0 item
NSRT
define FORM item
Clear label (LBC)
LINSYM

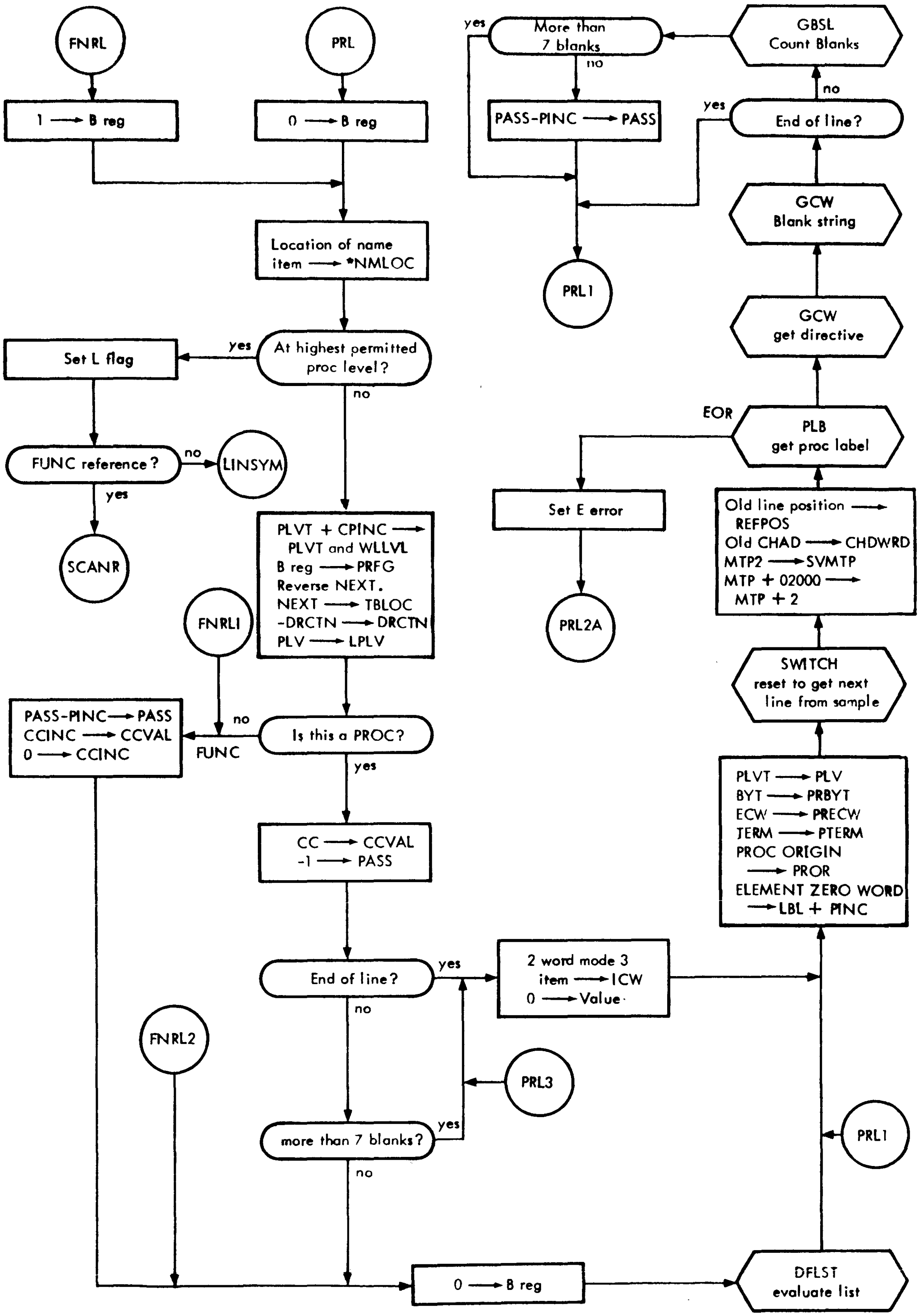

FUNC
PROC
1 → B reg
0 → B reg
In sample process routine (SAM)
Turn SMPFG to on
PRCNT + 1 → PRCNT
SA2
yes
PRCNT > 1?
no
Save sample storage position of PROC item
B reg → FNFG
0 → PRORG
0 → LBL
SVBSD* 2^19 + SVBSZ* 2^15 + WRDPOS → PRPOS
yes
Inside procedure?
no
LINSYM
Move 10 bytes of proc line to PRBYTS
LINSYM

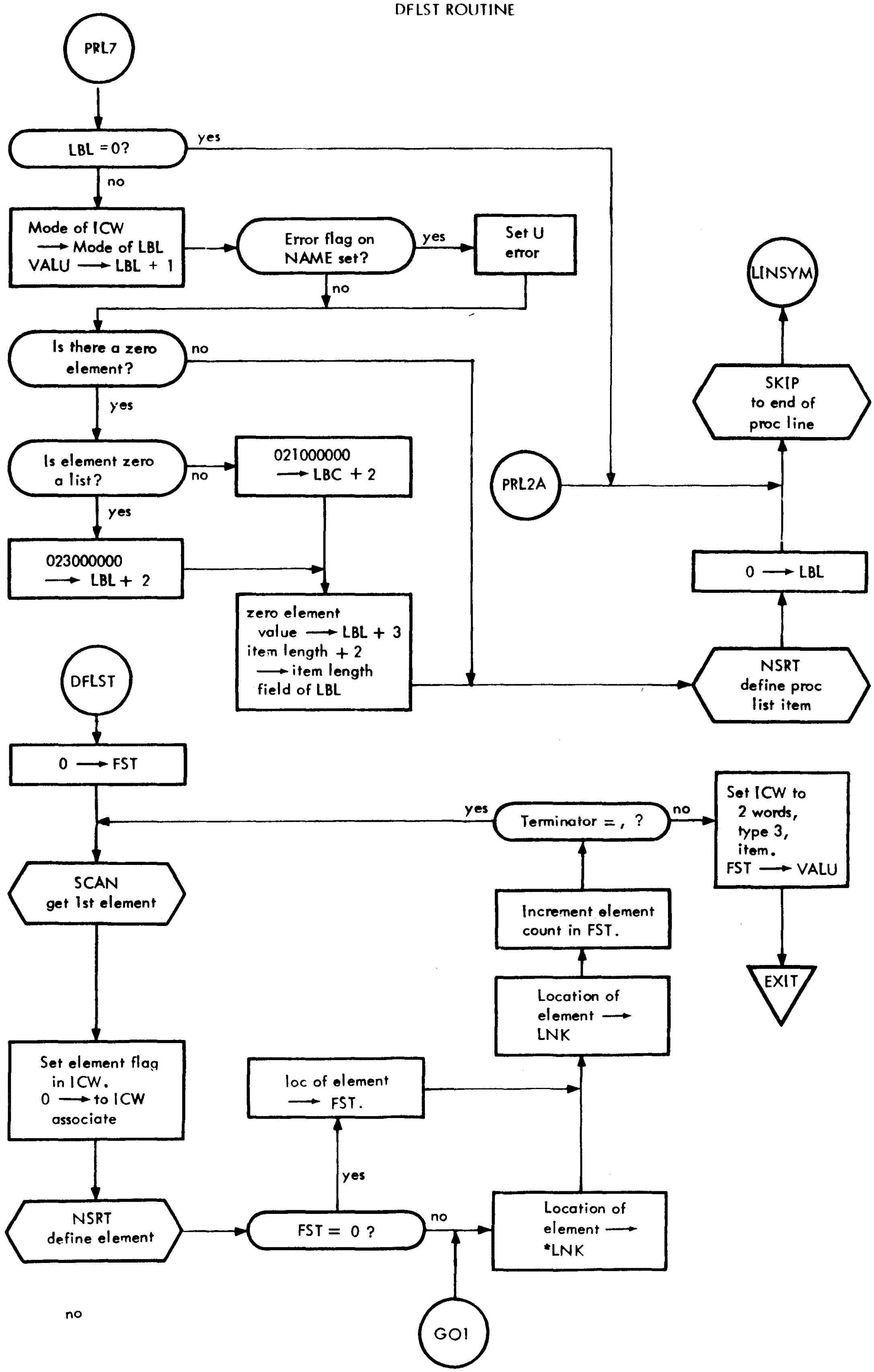

POPD
Save WLLVL at WLLVL + PINC.
WLLVL − PINC ⟶ WLLVL
Set mode bits of label to
2 word, type 2, mode 3.
POPNR* 2^{16} + CC
⟶ LBL + 1 (label value).
External definition
no
yes
Set external flag in LBL + 1
POPNR + 1
⟶ POPNR
LNLOC
LNLOC
LBTST
define waiting label
CCINC + 1 ⟶ CCINC
Define item in LBL as 2 word,
type 2, mode 3 item.
Save WLLVL and BYTLOC of
current levels.
Set BYTLOC with Byte Table
location of POP.
Set WLLVL to next lower
level.
POPNR + 2^{22} ⟶ LBL + 1.
NSRT
define item to
assembler
Clear LBL.
POPNR + 1 ⟶ POPNR.
Reset WLLVL and BYTLOC.
LNLOC


DO
DOTB + 5 → DOTB (DO Table pointer)
Table overflow
yes
no
DOVFW
SCAN get DO count
COUNT → DOTAB+3
Label on DO line
no
yes
Set up label as 2 word, type 1, model value item with unit value
NSRT define DO label
Clear label LOC of DO label value → DOTAB
DO1
Proc level to bits 0-9 of DOTAB
Increment DO label value CHAD → DOTAB+4 Position of 1st line → DOTAB+2 (BUSD, BSIZ, + MTP
LINSYM
VOID Do?
no
yes
DOA3
DOTAB + 3 → DOTAB + 1
SKIP DO line
Store skip Count in bits 6-11 of DOTAB + 3
DO2
More than 64 lines?
yes
DOERR
SCAN get lines to skip
yes
Terminator , ?
no
Store lines to do in bits 0-5 of DOTAB + 3
no
More than 64 times?
yes
DOERR
SCAN get lines to DO
yes
terminator =, ?
no
1 → bits 0-5 of DOTAB + 3
yes
DO3
IN procedure?
no
no
DO count > 210 ?
yes
DOERR
Set E error flag
DOTB-5 → DOTB
LINE
DOA5
EPRNT print line
yes
VOID DO?
no
DO1ZZ
Step DO label value
LINE
Set lines to do in DOTAB + 1 and DOTAB + 3 = 1
REZZ
EOF
TEXT get next line
no
Set E error flag
yes
terminator =, ?
EPRNT edit line
DOVFW
Set P error flag
DOTB-5 → DOTB
EPRNT print line
DOEND
LINE

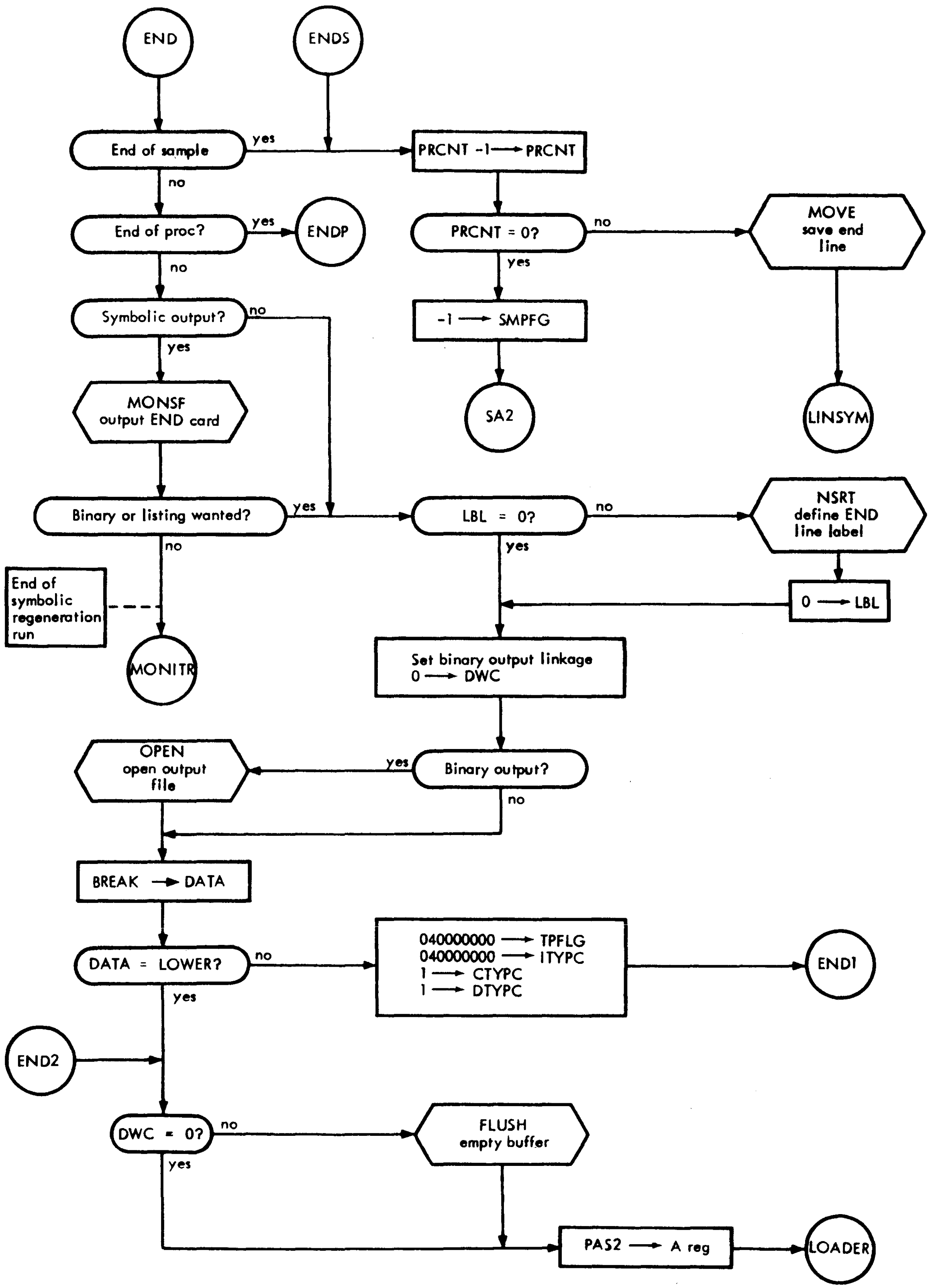

DOAGN
Is there another line to do?
no
yes
Line count - 1 ⟶ line count
Current DO count = Original count
yes
DODEC count lower lines
no
In procedure?
yes
TEXT get next line
EOF
REZZ
LN4
no
Location of BBUF ⟶ BYTE
LN4
DOA2
DOA4
DO finished?
yes
Lines to skip ⟶ TEMP
no
DO count - 1 ⟶ DO count at DOTAB + 3. Original lines to DO and lines to skip from DOTAB + 3 ⟶ DOTAB + 1 Increment DO label value
Inside procedure?
no
DOAGN
yes
SWITCH reset to 1st line of DO
EOR flag ⟶ BYT
DOAGN
Inside procedure?
no
DOA5
yes
SKIP skip line
Finished skipping?
yes
DOA5
no
GCW get 1st byte
SKIP skip line
DODEC skip outer lines
DOA3
Number lines to DO ⟶ TEMP
Finished all lines?
yes
DOA5
no
GCW get 1st byte
SKIP skip line
DODEC decrement outer

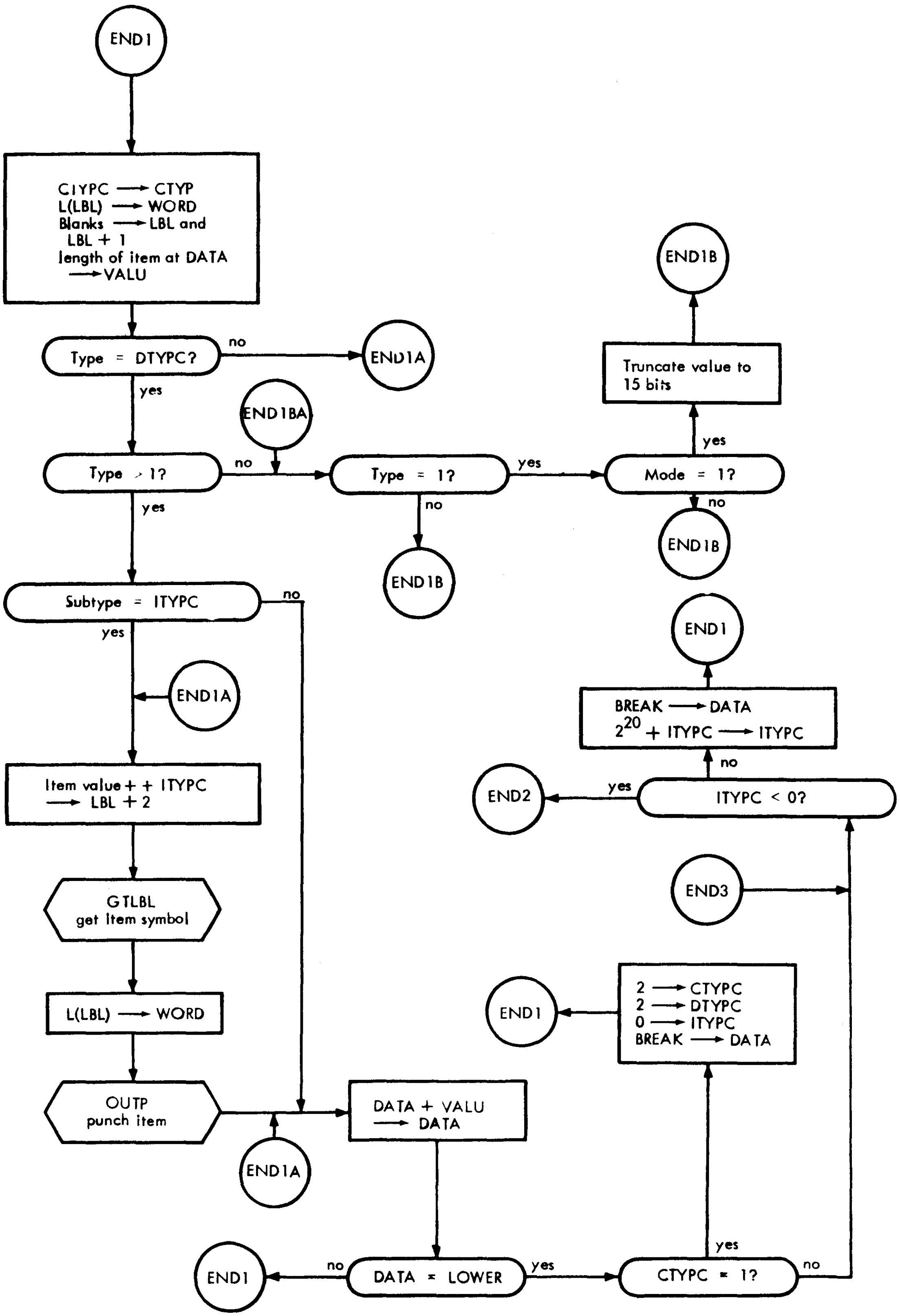

DODEC
DOTB - 5 ⟶ TEMP + 1
TEMP + 1 > L (DOTAB)
no
DOTB ⟶ X
EXIT
yes
Level this do =
Current proc level
yes
Lines to do - 1 ⟶
lines to do (for
DO at TEMP + 1)
no
(TEMP + 1) - 5 ⟶ TEMP + 1

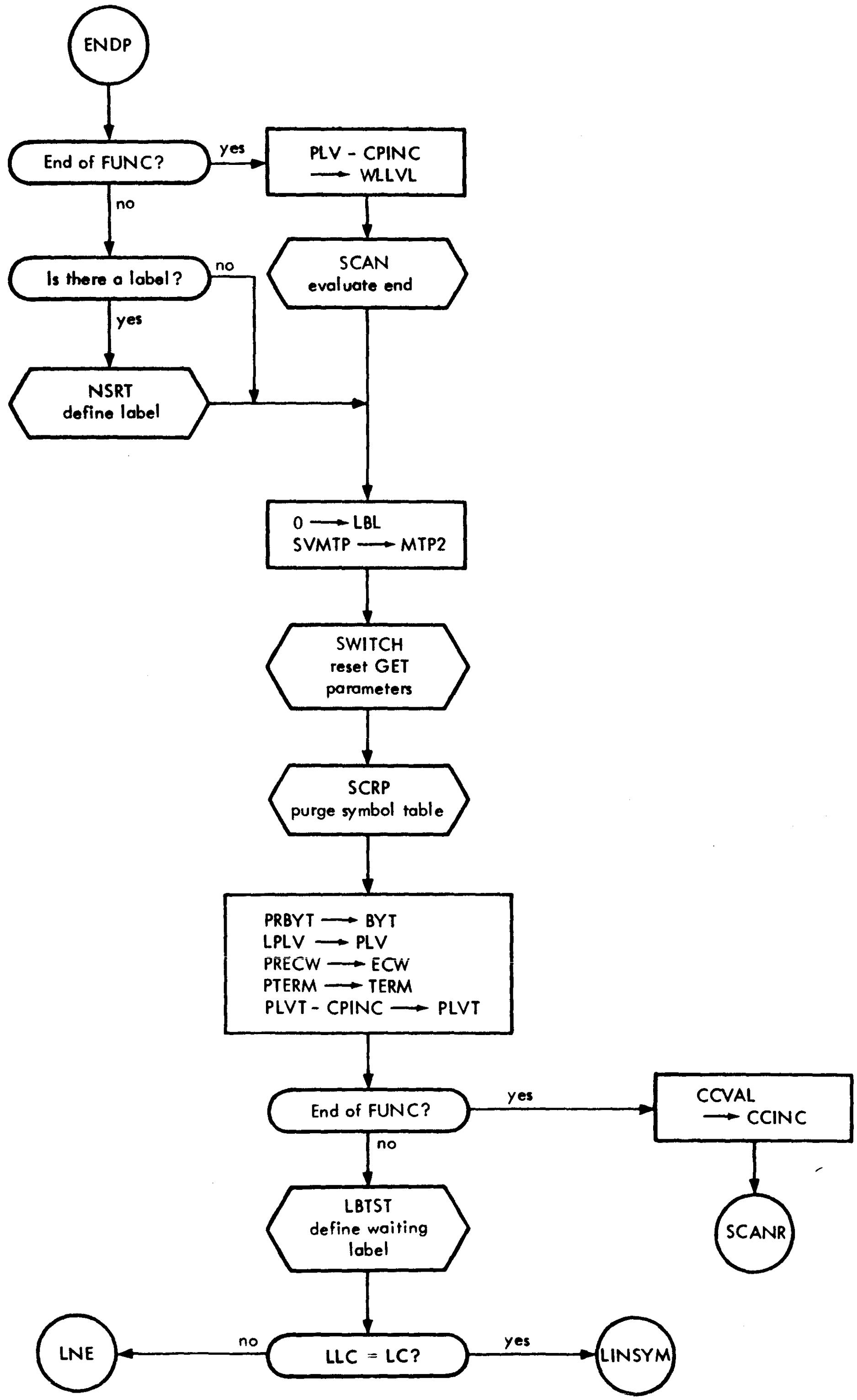

FNRL
1 → B reg
PRL
0 → B reg
Location of name item → *NMLOC
At highest permitted proc level?
yes
no
Set L flag
FUNC reference?
no
LINSYM
yes
SCANR
PLVT + CPINC → PLVT and WLLVL
B reg → PRFG
Reverse NEXT.
NEXT → TBLOC
-DRCTN → DRCTN
PLV → LPLV
FNRL1
PASS-PINC → PASS
CCINC → CCVAL
0 → CCINC
FUNC
Is this a PROC?
CC → CCVAL
-1 → PASS
End of line?
2 word mode 3 item → ICW
0 → Value
FNRL2
PRL3
more than 7 blanks?
0 → B reg
More than 7 blanks
PASS-PINC → PASS
PRL1
GBSL
Count Blanks
End of line?
GCW
Blank string
GCW
get directive
EOR
PLB
get proc label
Set E error
PRL2A
Old line position → REFPOS
Old CHAD → CHDWRD
MTP2 → SVMTP
MTP + 02000 → MTP + 2
SWITCH
reset to get next line from sample
PLVT → PLV
BYT → PRBYT
ECW → PRECW
TERM → PTERM
PROC ORIGIN → PROR
ELEMENT ZERO WORD → LBL + PINC
PRL1
DFLST
evaluate list

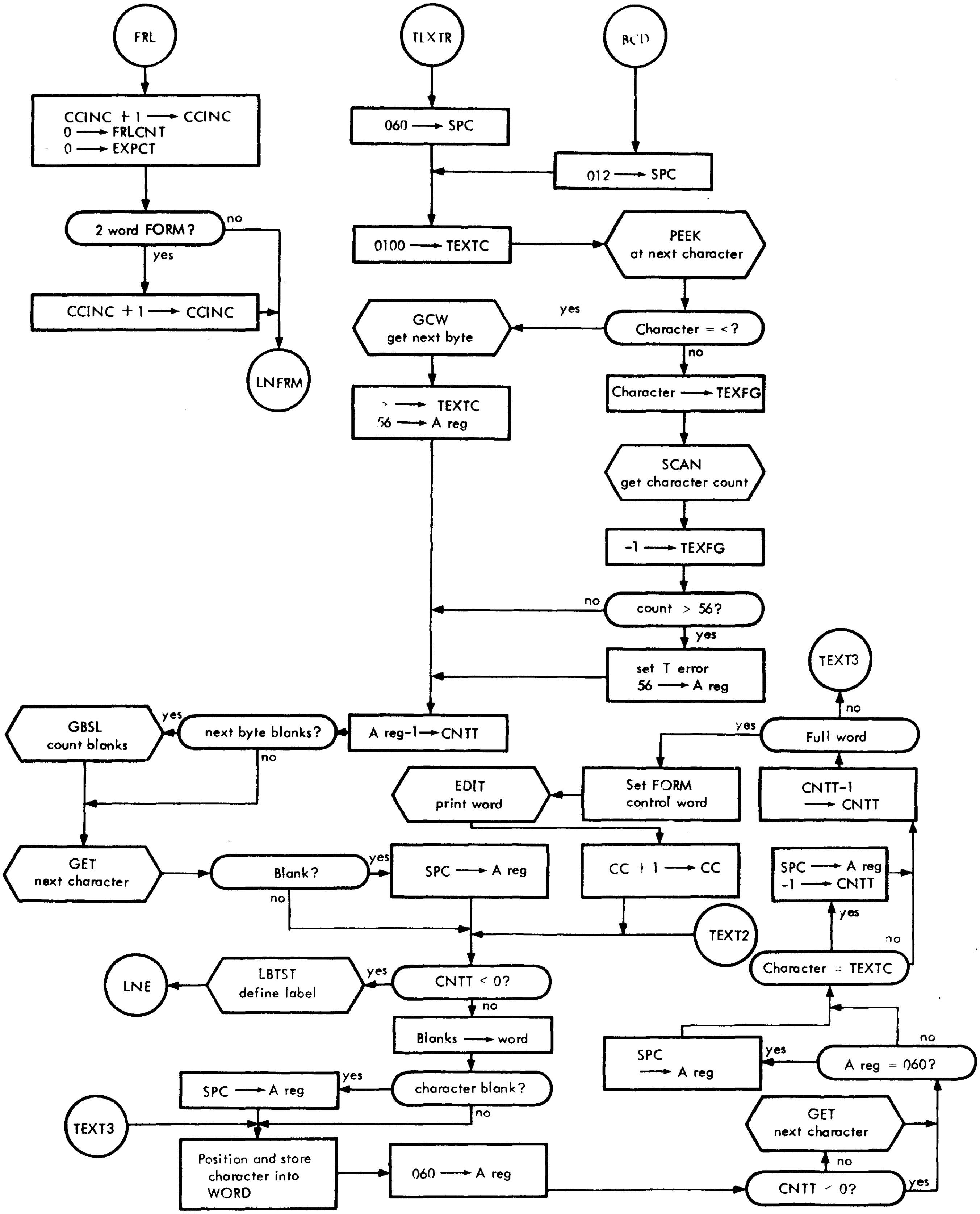

PRL7
LBL = 0?
yes
no
Mode of ICW → Mode of LBL
VALU → LBL + 1
Error flag on NAME set?
yes
Set U error
no
LINSYM
Is there a zero element?
no
yes
SKIP to end of proc line
Is element zero a list?
no
021000000 → LBC + 2
yes
PRL2A
023000000 → LBL + 2
0 → LBL
zero element value → LBL + 3
item length + 2 → item length field of LBL
DFLST
NSRT define proc list item
0 → FST
yes
Terminator = , ?
no
Set ICW to 2 words, type 3, item.
FST → VALU
SCAN get 1st element
Increment element count in FST.
EXIT
Location of element → LNK
Set element flag in ICW.
0 → to ICW associate
loc of element → FST.
yes
NSRT define element
FST = 0 ?
no
Location of element → *LNK
no
GO1

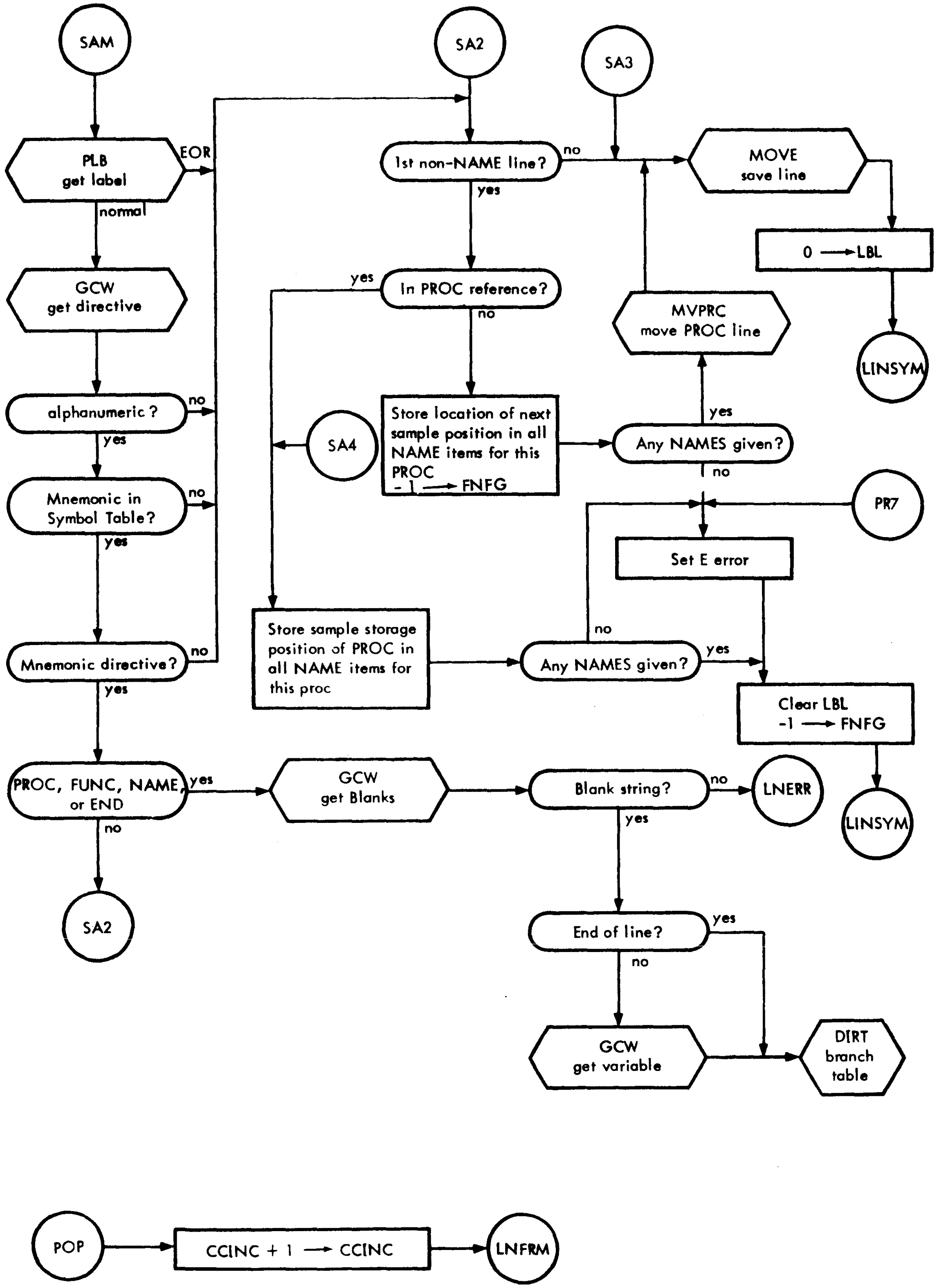
ASSEMBLR
END ROUTINE

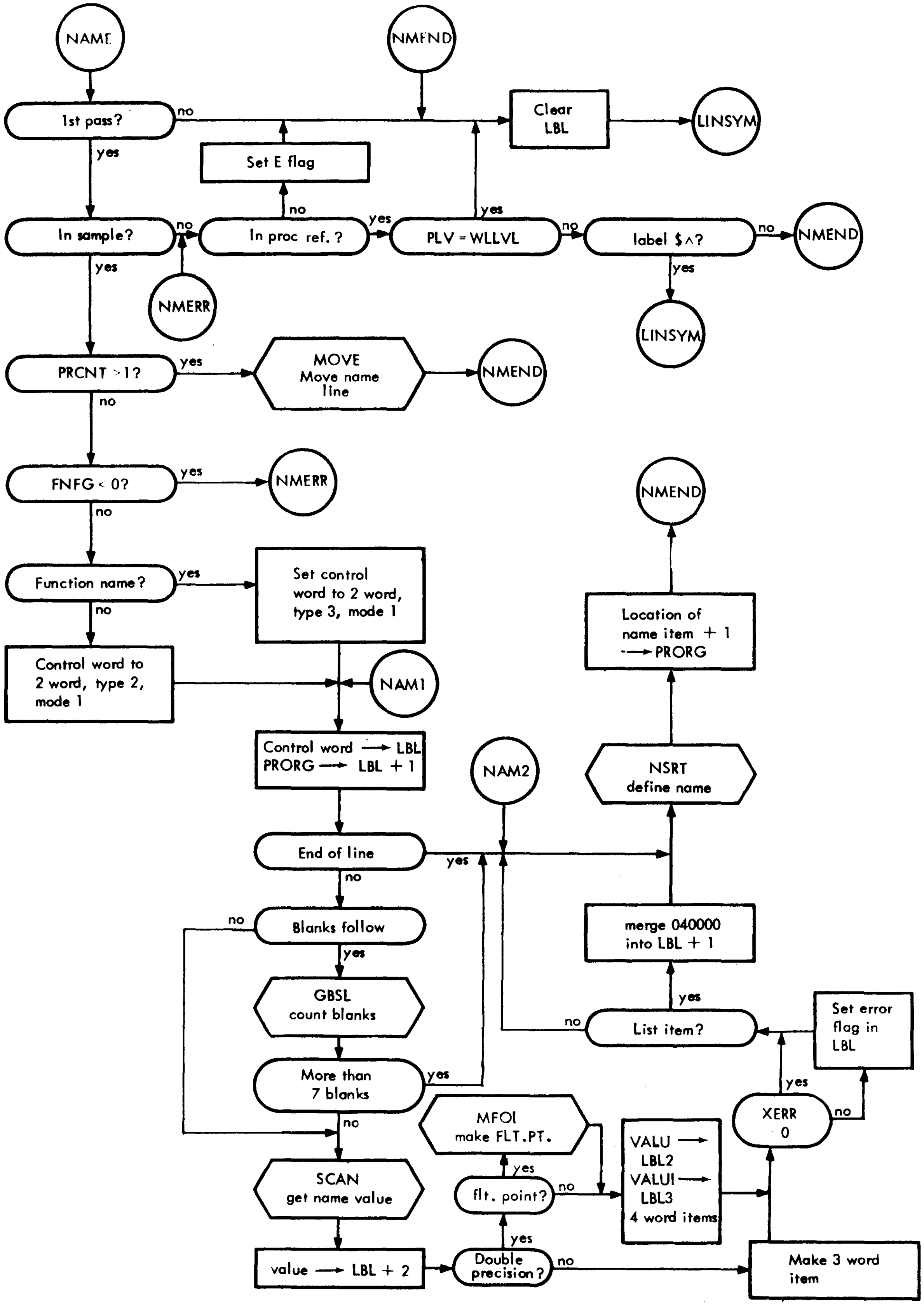


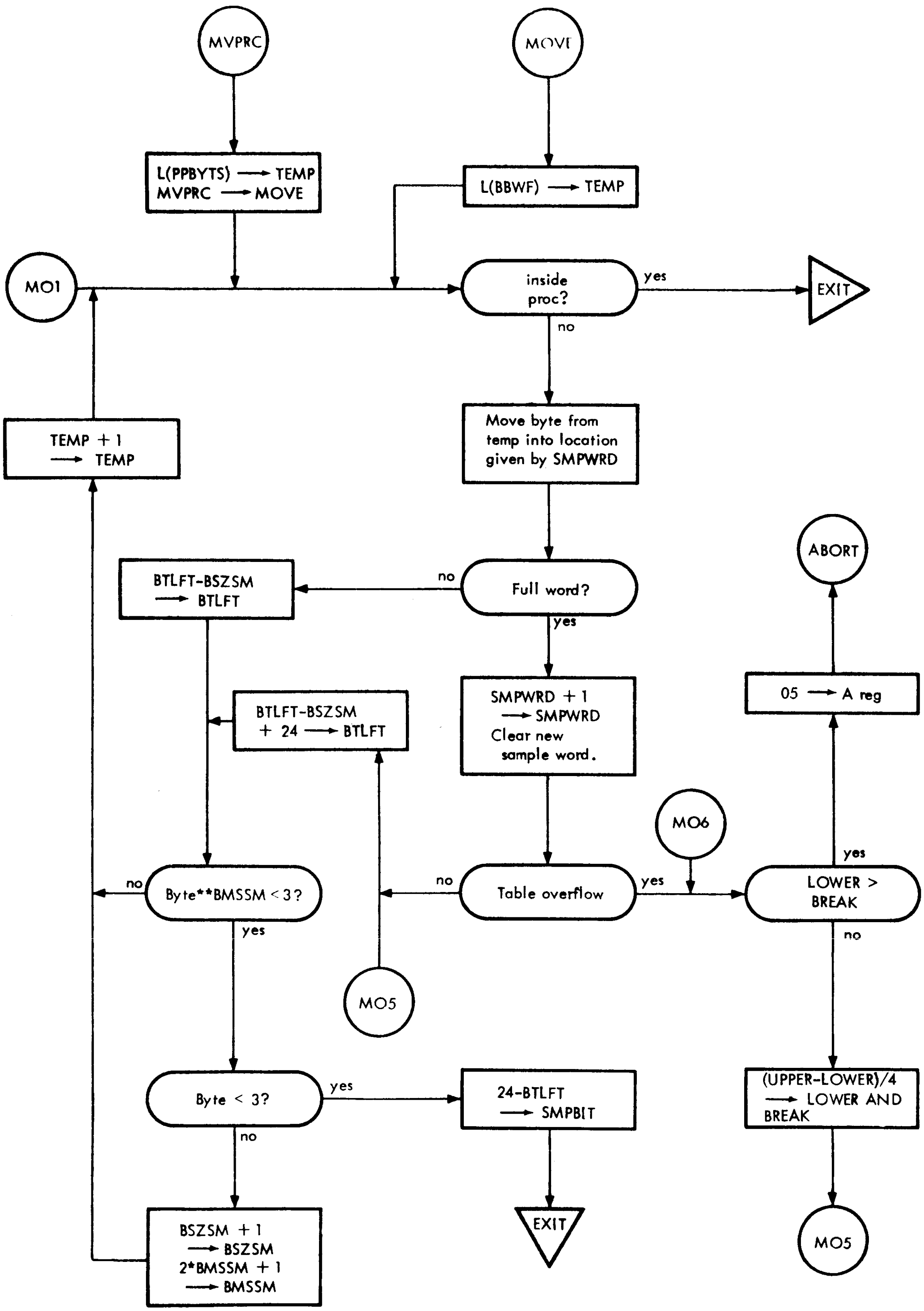

END1
CTYPC ⟶ CTYP
L(LBL) ⟶ WORD
Blanks ⟶ LBL and
LBL + 1
length of item at DATA
⟶ VALU
Type = DTYPC?
no
yes
END1A
END1BA
Type > 1?
no
yes
Type = 1?
yes
no
END1B
Mode = 1?
yes
no
END1B
Truncate value to
15 bits
END1B
Subtype = ITYPC
no
yes
END1A
Item value + + ITYPC
⟶ LBL + 2
GTLBL
get item symbol
L(LBL) ⟶ WORD
OUTP
punch item
END1A
DATA + VALU
⟶ DATA
END1
BREAK ⟶ DATA
2^20 + ITYPC ⟶ ITYPC
no
END2
yes
ITYPC < 0?
END3
END1
2 ⟶ CTYPC
2 ⟶ DTYPC
0 ⟶ ITYPC
BREAK ⟶ DATA
END1
no
DATA = LOWER
yes
CTYPC = 1?
yes
no

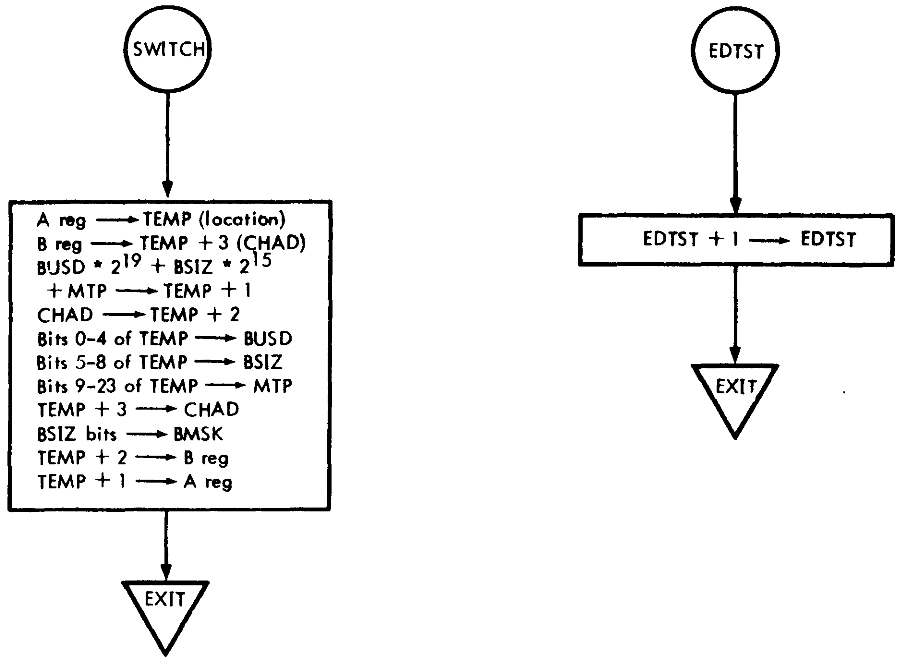

ENDP
End of FUNC?
yes
no
PLV - CPINC → WLLVL
SCAN evaluate end
Is there a label?
no
yes
NSRT define label
0 → LBL
SVMTP → MTP2
SWITCH reset GET parameters
SCRP purge symbol table
PRBYT → BYT
LPLV → PLV
PRECW → ECW
PTERM → TERM
PLVT - CPINC → PLVT
End of FUNC?
yes
no
CCVAL → CCINC
SCANR
LBTST define waiting label
LLC = LC?
no
yes
LNE
LINSYM

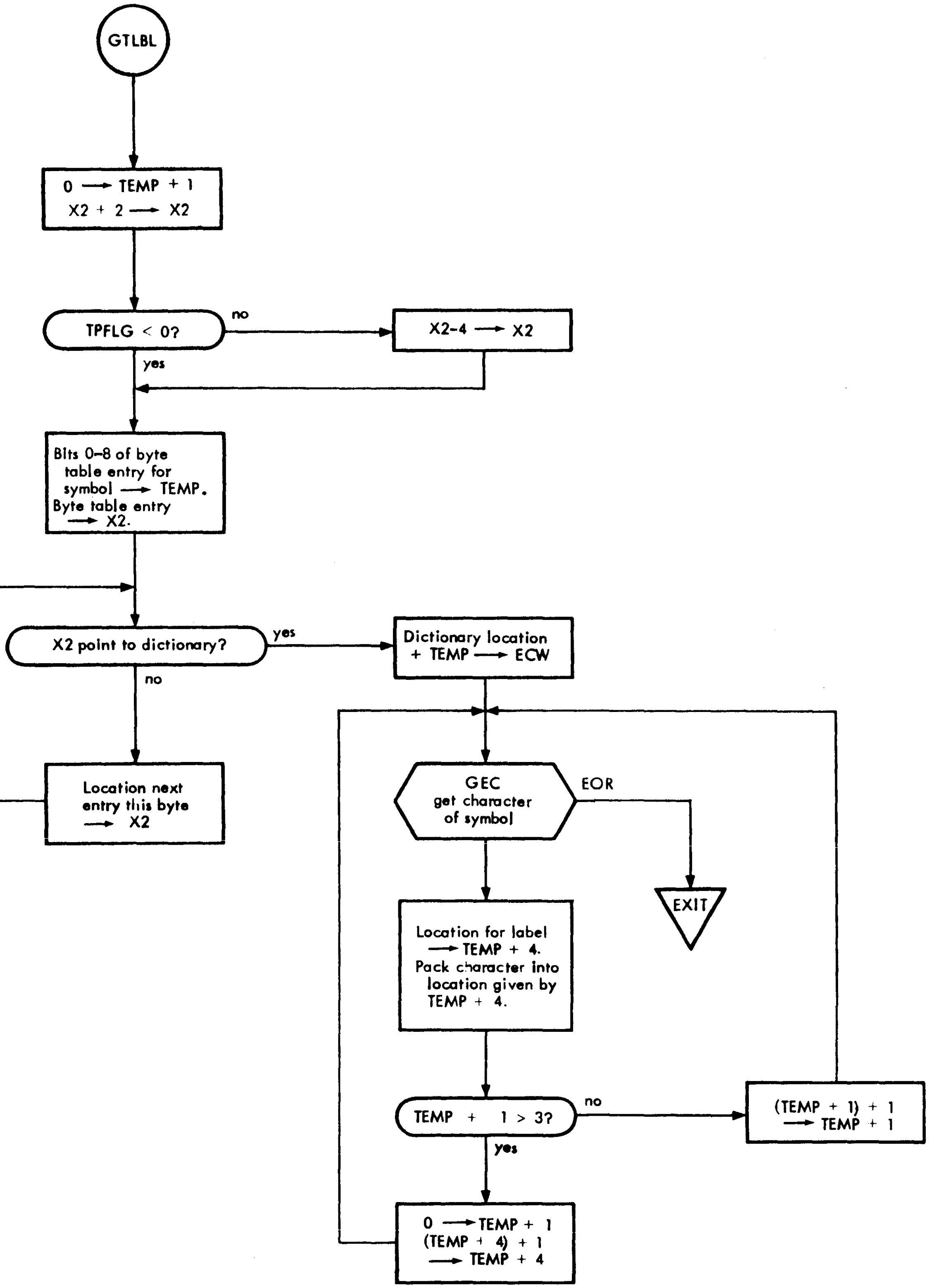

FRL
CCINC + 1 → CCINC
0 → FRLCNT
0 → EXPCT
2 word FORM?
yes
no
CCINC + 1 → CCINC
LNFRM
TEXTR
060 → SPC
BCD
012 → SPC
0100 → TEXTC
PEEK
at next character
Character = <?
yes
GCW
get next byte
> → TEXTC
56 → A reg
no
Character → TEXFG
SCAN
get character count
-1 → TEXFG
count > 56?
no
yes
set T error
56 → A reg
A reg-1 → CNTT
next byte blanks?
yes
GBSL
count blanks
no
GET
next character
Blank?
yes
no
SPC → A reg
TEXT3
no
Full word
yes
Set FORM
control word
EDIT
print word
CNTT-1
→ CNTT
CC + 1 → CC
SPC → A reg
-1 → CNTT
yes
no
TEXT2
Character = TEXTC
CNTT < 0?
yes
LBTST
define label
LNE
no
Blanks → word
character blank?
yes
SPC → A reg
no
TEXT3
Position and store
character into
WORD
060 → A reg
SPC
→ A reg
yes
A reg = 060?
no
GET
next character
no
CNTT < 0?
yes

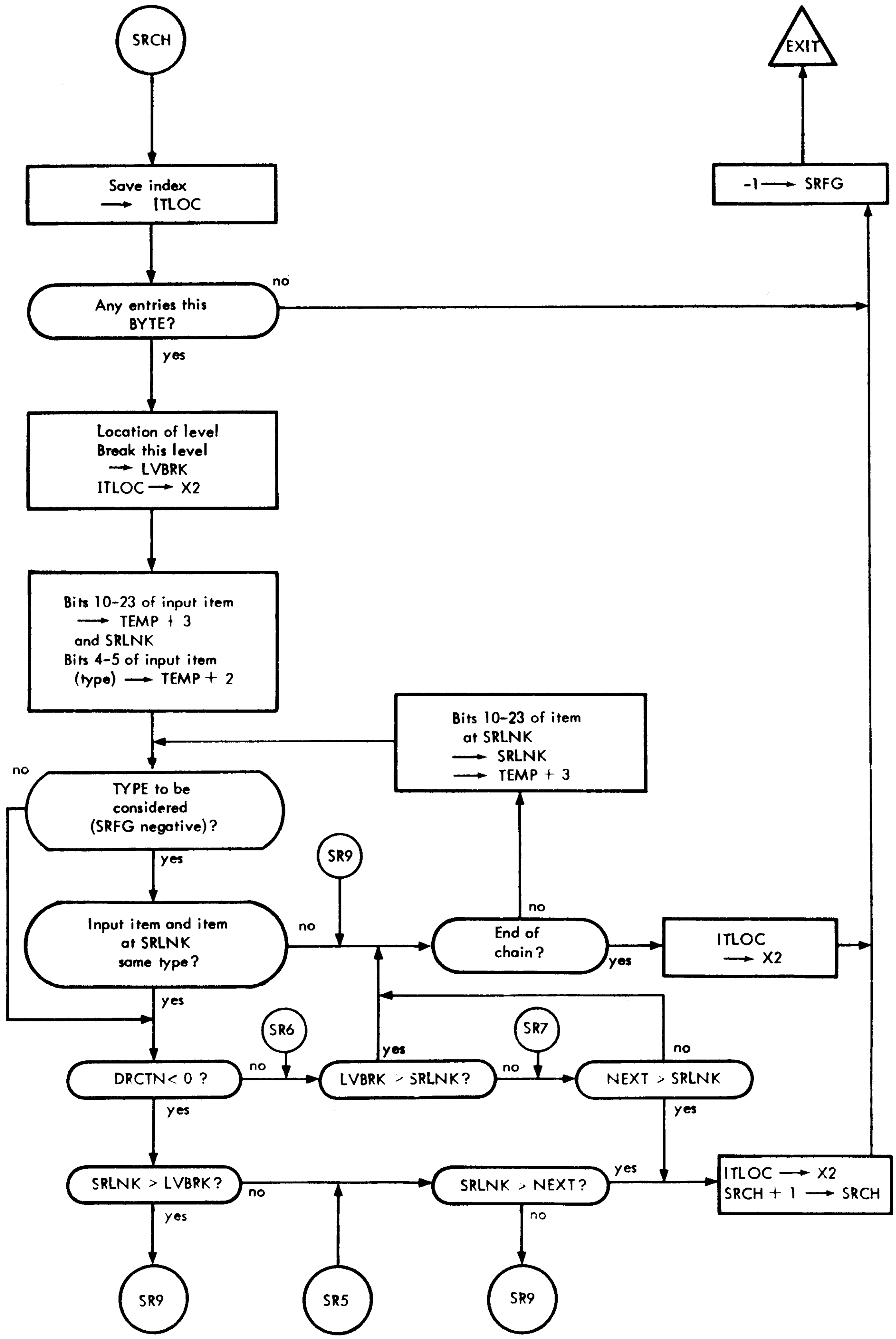

SAM
SA2
SA3
PLB
get label
EOR
normal
1st non-NAME line?
no
yes
MOVE
save line
0 ⟶ LBL
GCW
get directive
yes
In PROC reference?
no
MVPRC
move PROC line
LINSYM
alphanumeric ?
no
yes
SA4
Store location of next sample position in all NAME items for this PROC
-1 ⟶ FNFG
yes
Any NAMES given?
no
Mnemonic in Symbol Table?
no
yes
PR7
Set E error
Store sample storage position of PROC in all NAME items for this proc
no
Mnemonic directive?
no
yes
Any NAMES given?
yes
Clear LBL
-1 ⟶ FNFG
PROC, FUNC, NAME, or END
yes
no
GCW
get Blanks
Blank string?
no
yes
LNERR
LINSYM
SA2
End of line?
yes
no
GCW
get variable
DIRT
branch
table
POP
CCINC + 1 ⟶ CCINC
LNFRM

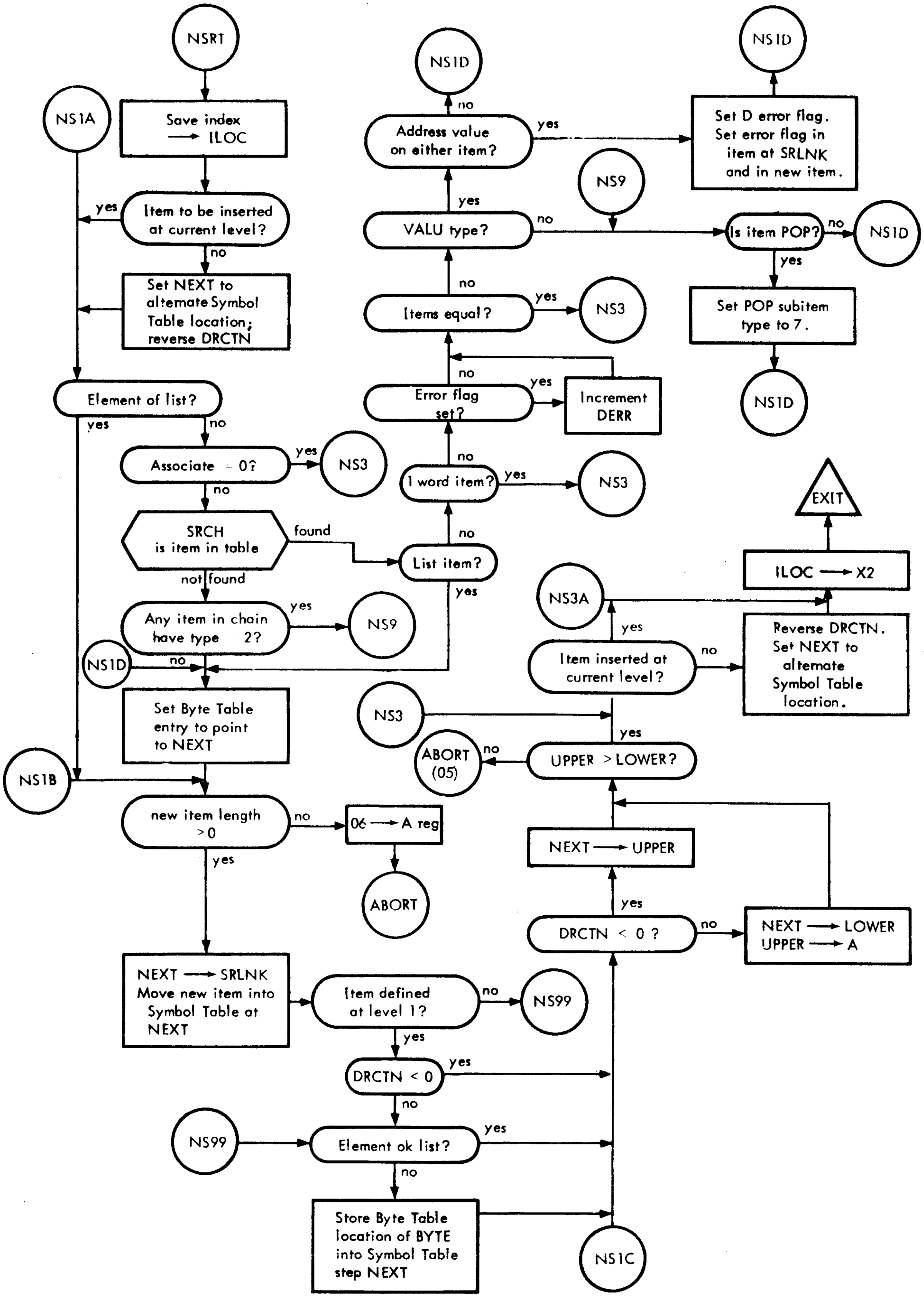

NAME
NMEND
1st pass?
no
yes
Set E flag
Clear LBL
LINSYM
In sample?
no
In proc ref.?
no
yes
PLV = WLLVL
yes
no
label $∧?
no
NMEND
yes
LINSYM
NMERR
yes
PRCNT > 1?
yes
MOVE Move name line
NMEND
no
FNFG < 0?
yes
NMERR
no
NMEND
Function name?
yes
Set control word to 2 word, type 3, mode 1
no
Location of name item + 1 → PRORG
Control word to 2 word, type 2, mode 1
NAM1
Control word → LBL
PRORG → LBL + 1
NAM2
NSRT define name
End of line
yes
no
Blanks follow
no
yes
merge 040000 into LBL + 1
GBSL count blanks
yes
List item?
no
Set error flag in LBL
More than 7 blanks
yes
no
XERR 0
yes
no
MFOI make FLT.PT.
VALU → LBL2
VALUI → LBL3
4 word items
SCAN get name value
flt. point?
yes
no
value → LBL + 2
Double precision?
yes
no
Make 3 word item

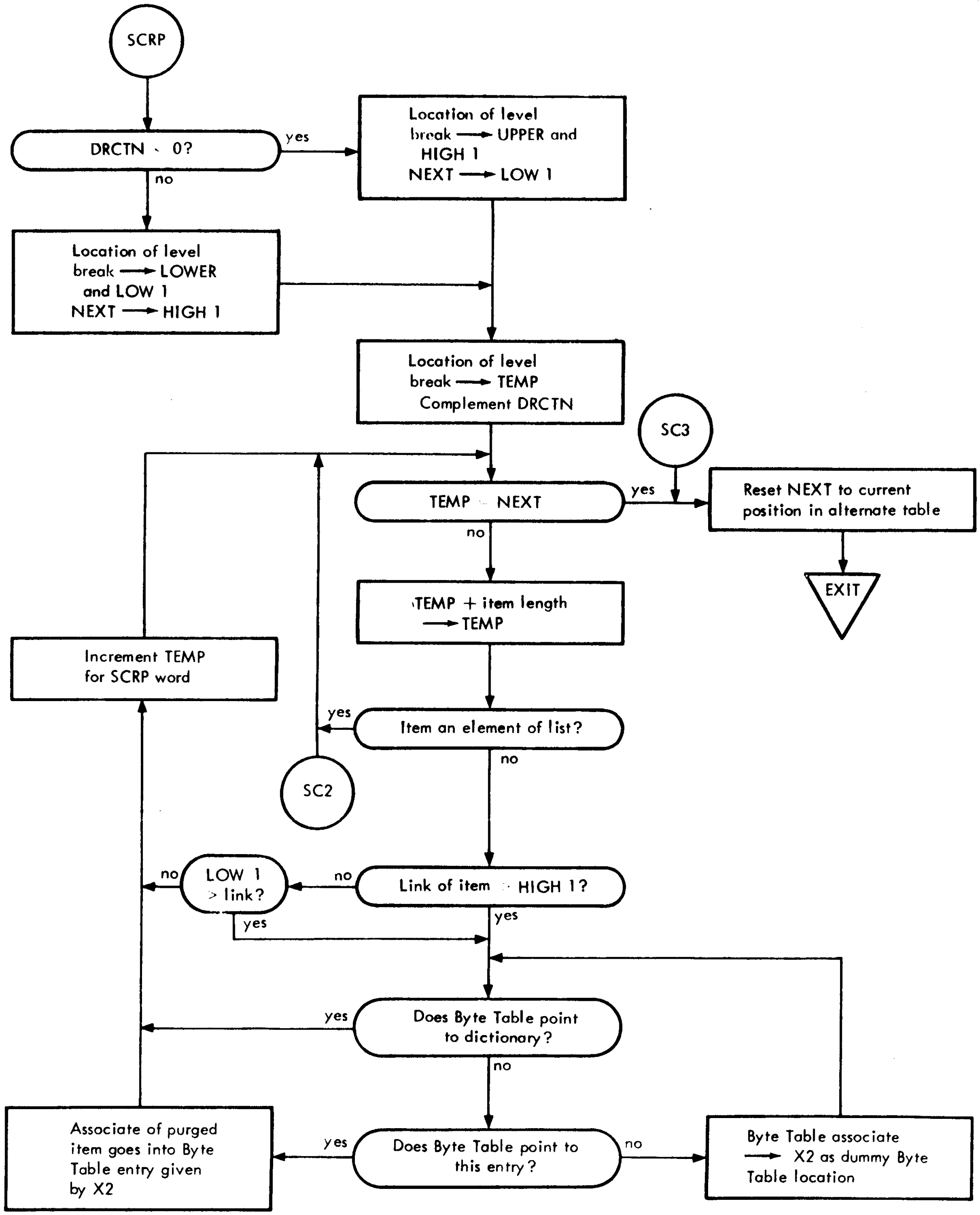

MVPRC
MOVE
L(PPBYTS) → TEMP
MVPRC → MOVE
L(BBWF) → TEMP
MO1
inside proc?
yes
EXIT
no
Move byte from temp into location given by SMPWRD
TEMP + 1 → TEMP
ABORT
BTLFT-BSZSM → BTLFT
no
Full word?
yes
05 → A reg
BTLFT-BSZSM + 24 → BTLFT
SMPWRD + 1 → SMPWRD
Clear new sample word.
MO6
yes
no
Byte**BMSSM < 3?
no
Table overflow
yes
LOWER > BREAK
yes
no
MO5
Byte < 3?
yes
24-BTLFT → SMPBIT
(UPPER-LOWER)/4 → LOWER AND BREAK
no
BSZSM + 1 → BSZSM
2*BMSSM + 1 → BMSSM
EXIT
MO5

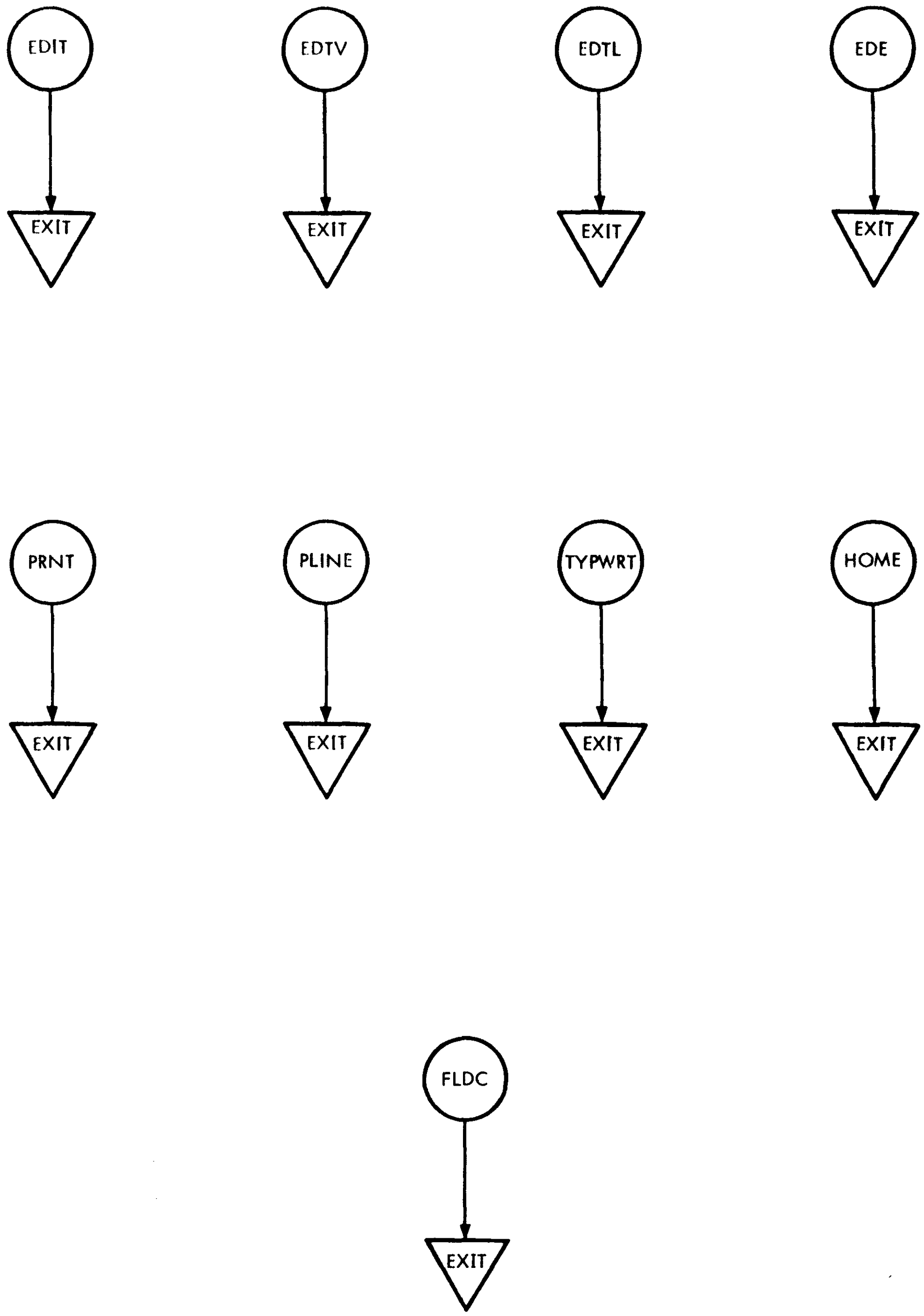

SWITCH
A reg ⟶ TEMP (location)
B reg ⟶ TEMP + 3 (CHAD)
BUSD * 2^19 + BSIZ * 2^15 + MTP ⟶ TEMP + 1
CHAD ⟶ TEMP + 2
Bits 0-4 of TEMP ⟶ BUSD
Bits 5-8 of TEMP ⟶ BSIZ
Bits 9-23 of TEMP ⟶ MTP
TEMP + 3 ⟶ CHAD
BSIZ bits ⟶ BMSK
TEMP + 2 ⟶ B reg
TEMP + 1 ⟶ A reg
EXIT
EDTST
EDTST + 1 ⟶ EDTST
EXIT

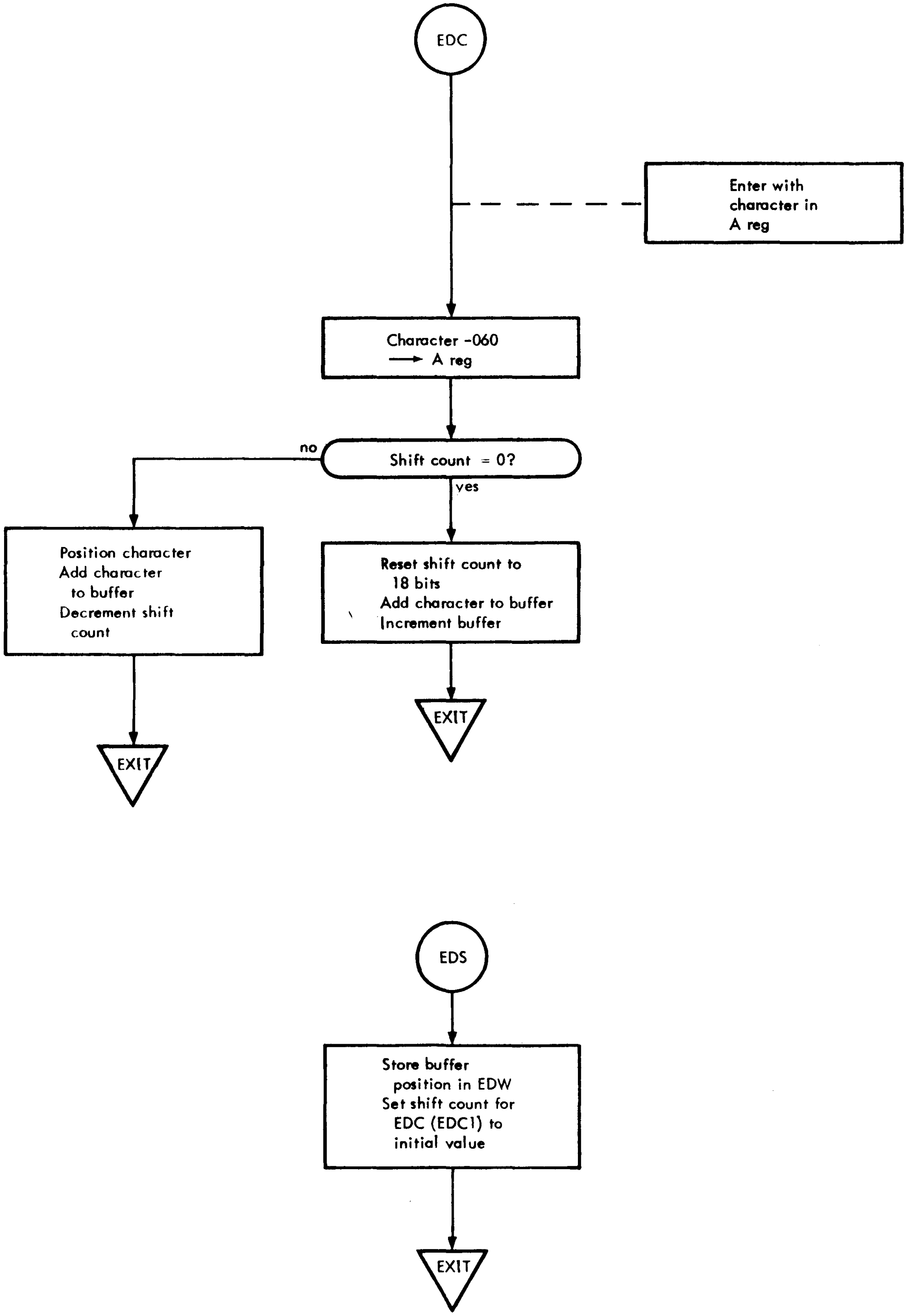

GTLBL
0 → TEMP + 1
X2 + 2 → X2
TPFLG < 0?
no
X2-4 → X2
yes
Bits 0-8 of byte table entry for symbol → TEMP. Byte table entry → X2.
X2 point to dictionary?
yes
Dictionary location + TEMP → ECW
no
Location next entry this byte → X2
GEC get character of symbol
EOR
EXIT
Location for label → TEMP + 4. Pack character into location given by TEMP + 4.
TEMP + 1 > 3?
no
(TEMP + 1) + 1 → TEMP + 1
yes
0 → TEMP + 1
(TEMP + 4) + 1 → TEMP + 4

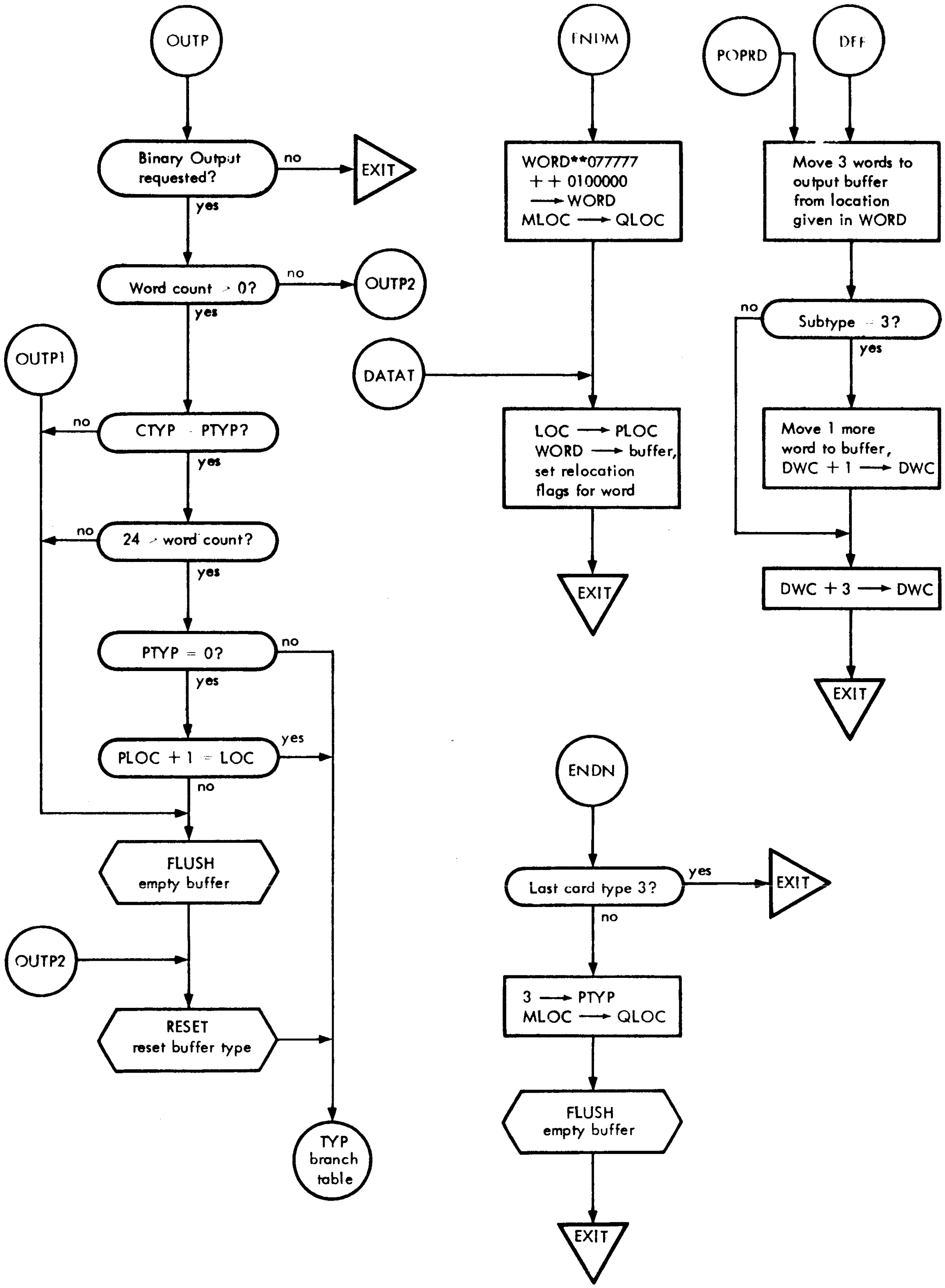

SRCH
Save index
→ ITLOC
Any entries this BYTE?
no
yes
Location of level
Break this level
→ LVBRK
ITLOC → X2
Bits 10-23 of input item
→ TEMP + 3
and SRLNK
Bits 4-5 of input item
(type) → TEMP + 2
Bits 10-23 of item
at SRLNK
→ SRLNK
→ TEMP + 3
TYPE to be considered (SRFG negative)?
Input item and item at SRLNK same type?
End of chain?
ITLOC
→ X2
DRCTN< 0 ?
LVBRK > SRLNK?
NEXT > SRLNK
SRLNK > LVBRK?
SRLNK > NEXT?
ITLOC → X2
SRCH + 1 → SRCH
-1 → SRFG
EXIT
SR9
SR6
SR7
SR5
yes
no

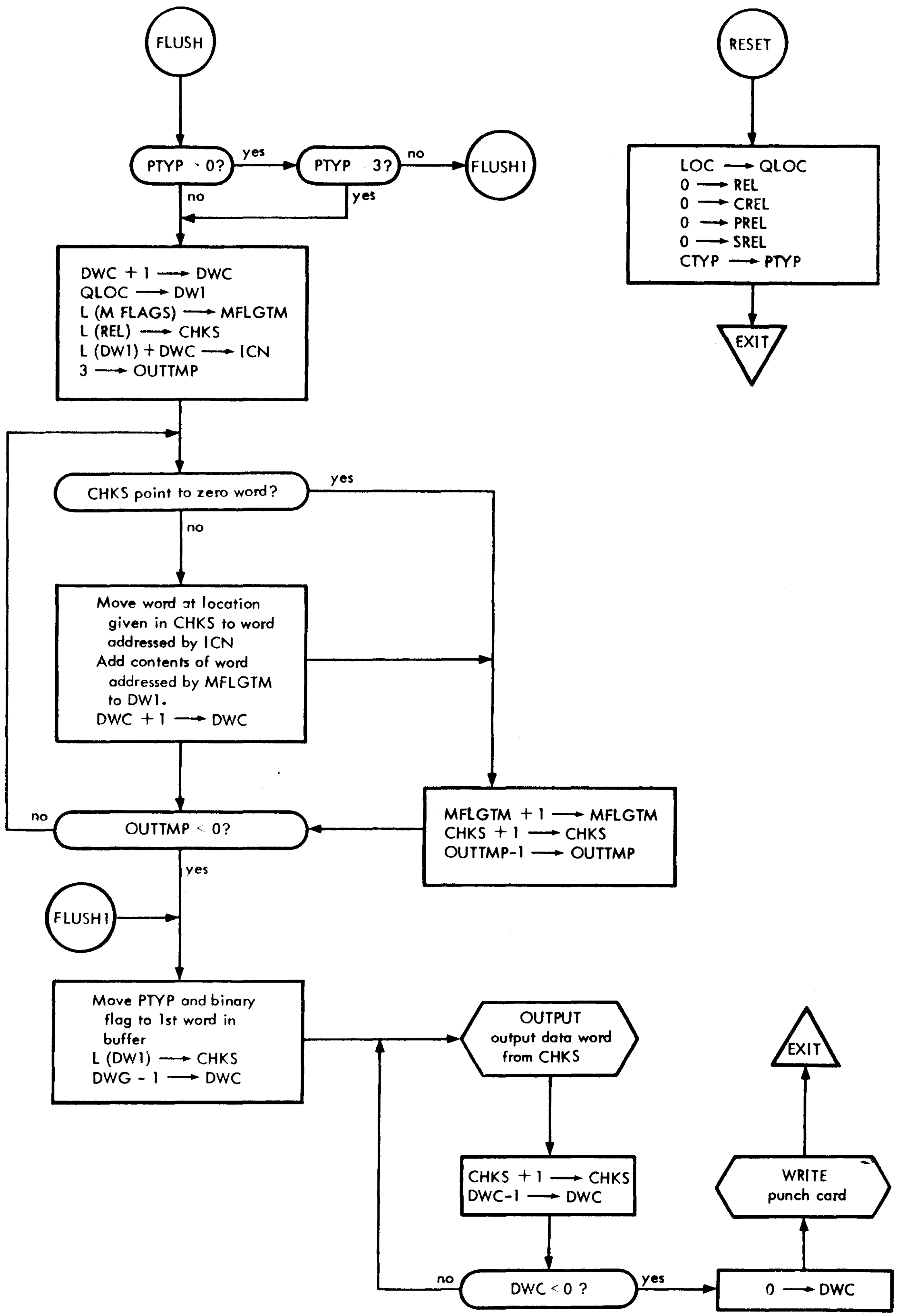

NSRT
Save index → ILOC
NS1A
Item to be inserted at current level?
yes
no
Set NEXT to alternate Symbol Table location; reverse DRCTN
Element of list?
yes
no
Associate = 0?
NS3
SRCH is item in table
found
not found
Any item in chain have type 2?
NS9
NS1D
Set Byte Table entry to point to NEXT
NS1B
new item length > 0
06 → A reg
ABORT
NEXT → SRLNK Move new item into Symbol Table at NEXT
Item defined at level 1?
NS99
DRCTN < 0
Element ok list?
Store Byte Table location of BYTE into Symbol Table step NEXT
NS1C
List item?
1 word item?
Error flag set?
Increment DERR
Items equal?
VALU type?
Address value on either item?
Set D error flag. Set error flag in item at SRLNK and in new item.
Is item POP?
Set POP subitem type to 7.
EXIT
ILOC → X2
NS3A
Item inserted at current level?
Reverse DRCTN. Set NEXT to alternate Symbol Table location.
UPPER > LOWER?
ABORT (05)
NEXT → UPPER
DRCTN < 0 ?
NEXT → LOWER UPPER → A

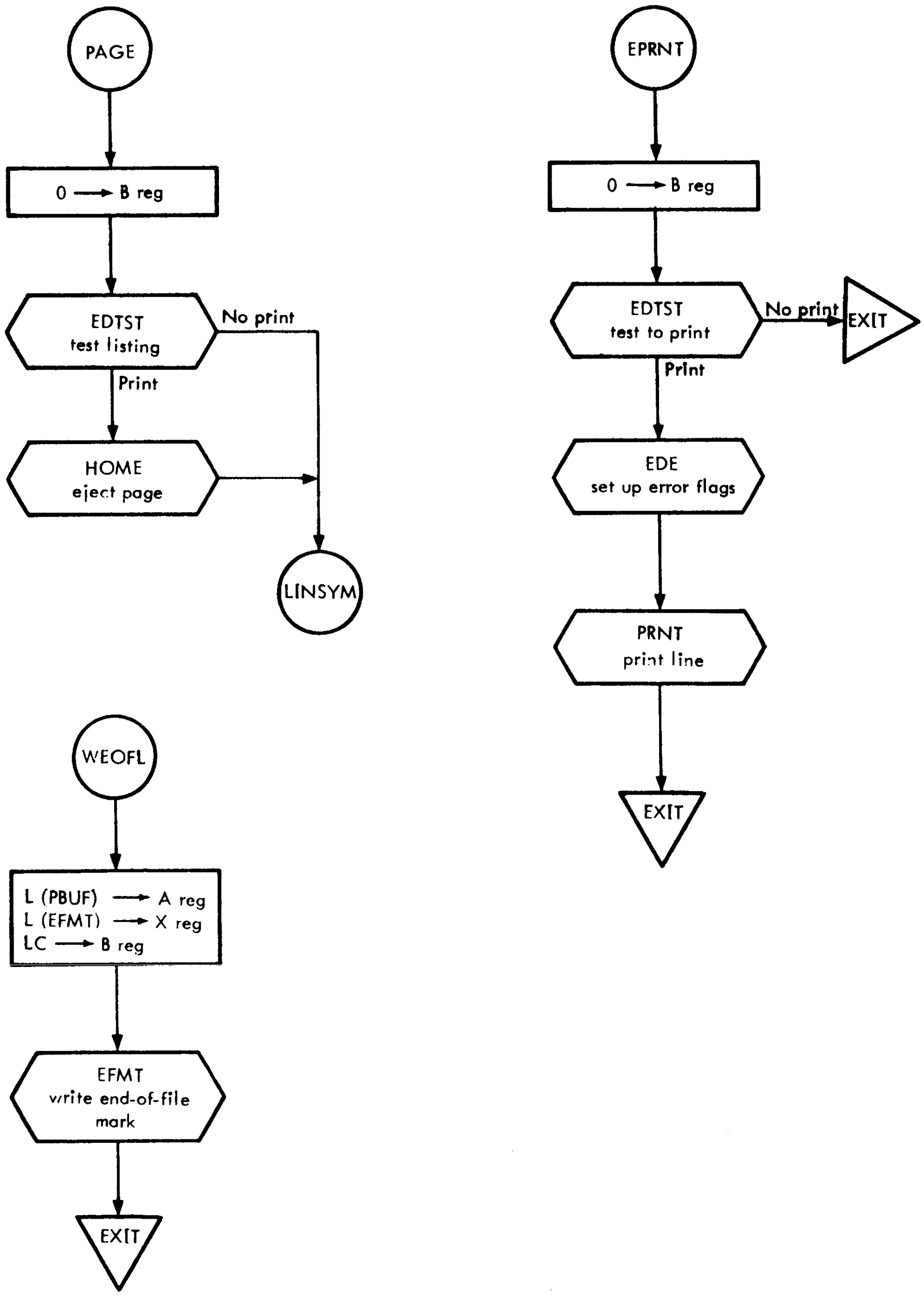

SCRP
DRCTN · 0?
yes
no
Location of level break → UPPER and HIGH 1
NEXT → LOW 1
Location of level break → LOWER and LOW 1
NEXT → HIGH 1
Location of level break → TEMP
Complement DRCTN
SC3
TEMP · NEXT
yes
no
Reset NEXT to current position in alternate table
EXIT
TEMP + item length → TEMP
Increment TEMP for SCRP word
yes
Item an element of list?
no
SC2
no
LOW 1 > link?
no
Link of item · HIGH 1?
yes
yes
yes
Does Byte Table point to dictionary?
no
Associate of purged item goes into Byte Table entry given by X2
yes
Does Byte Table point to this entry?
no
Byte Table associate → X2 as dummy Byte Table location

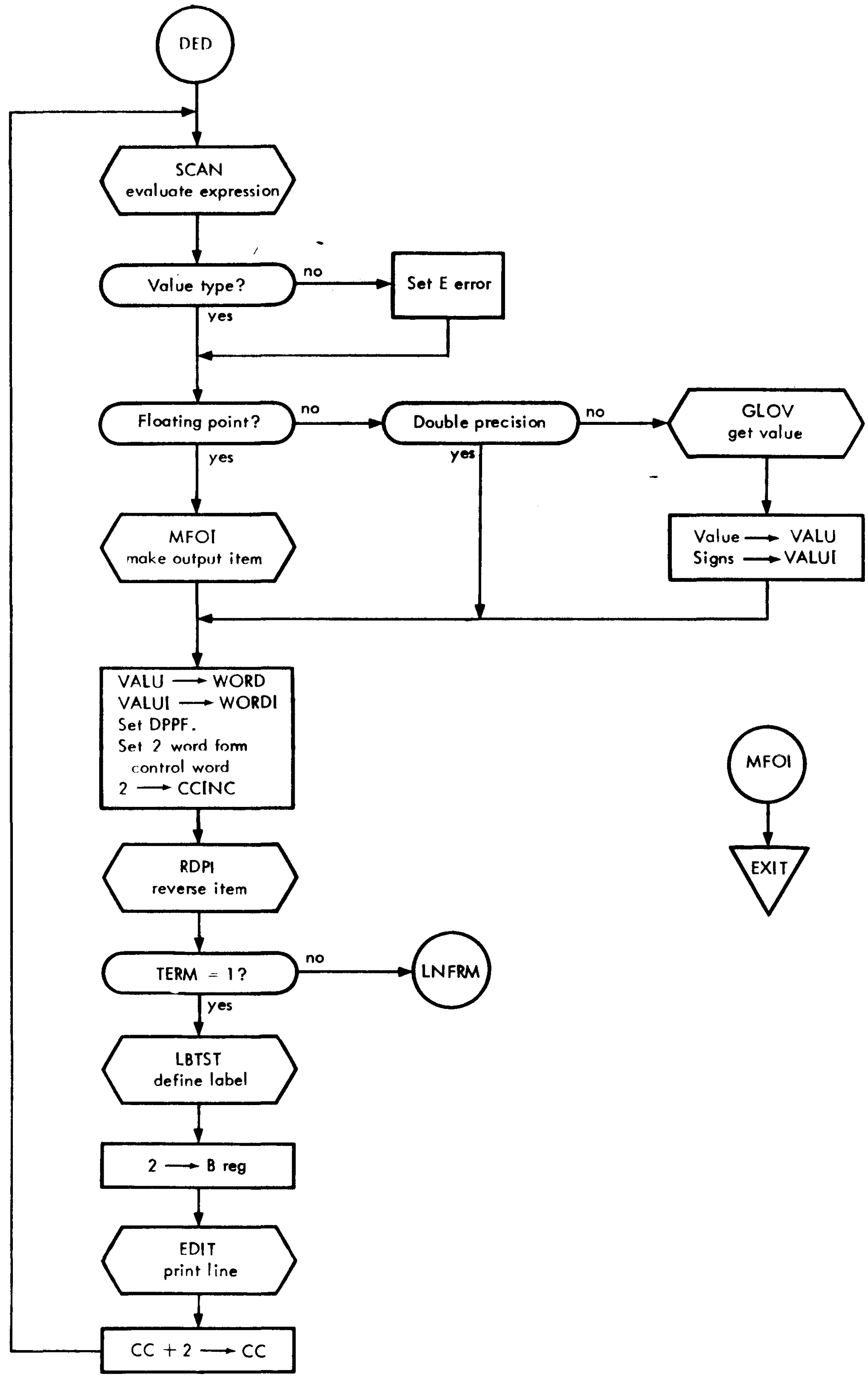

EDIT
EXIT
EDTV
EXIT
EDTL
EXIT
EDE
EXIT
PRNT
EXIT
PLINE
EXIT
TYPWRT
EXIT
HOME
EXIT
FLDC
EXIT

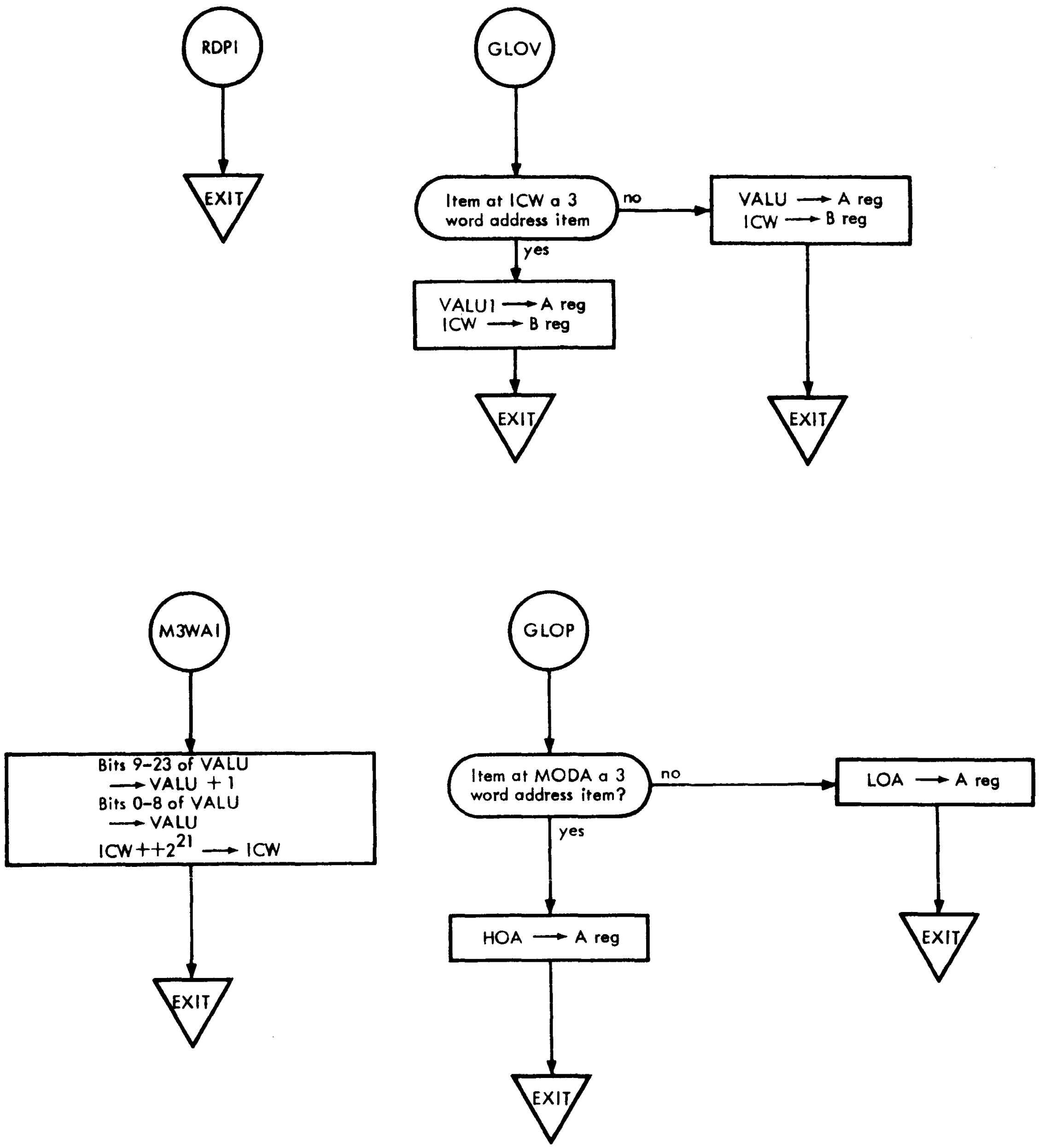

EDC
Enter with character in A reg
Character -060 → A reg
Shift count = 0?
no
yes
Position character
Add character to buffer
Decrement shift count
EXIT
Reset shift count to 18 bits
Add character to buffer
Increment buffer
EXIT
EDS
Store buffer position in EDW
Set shift count for EDC (EDC1) to initial value
EXIT

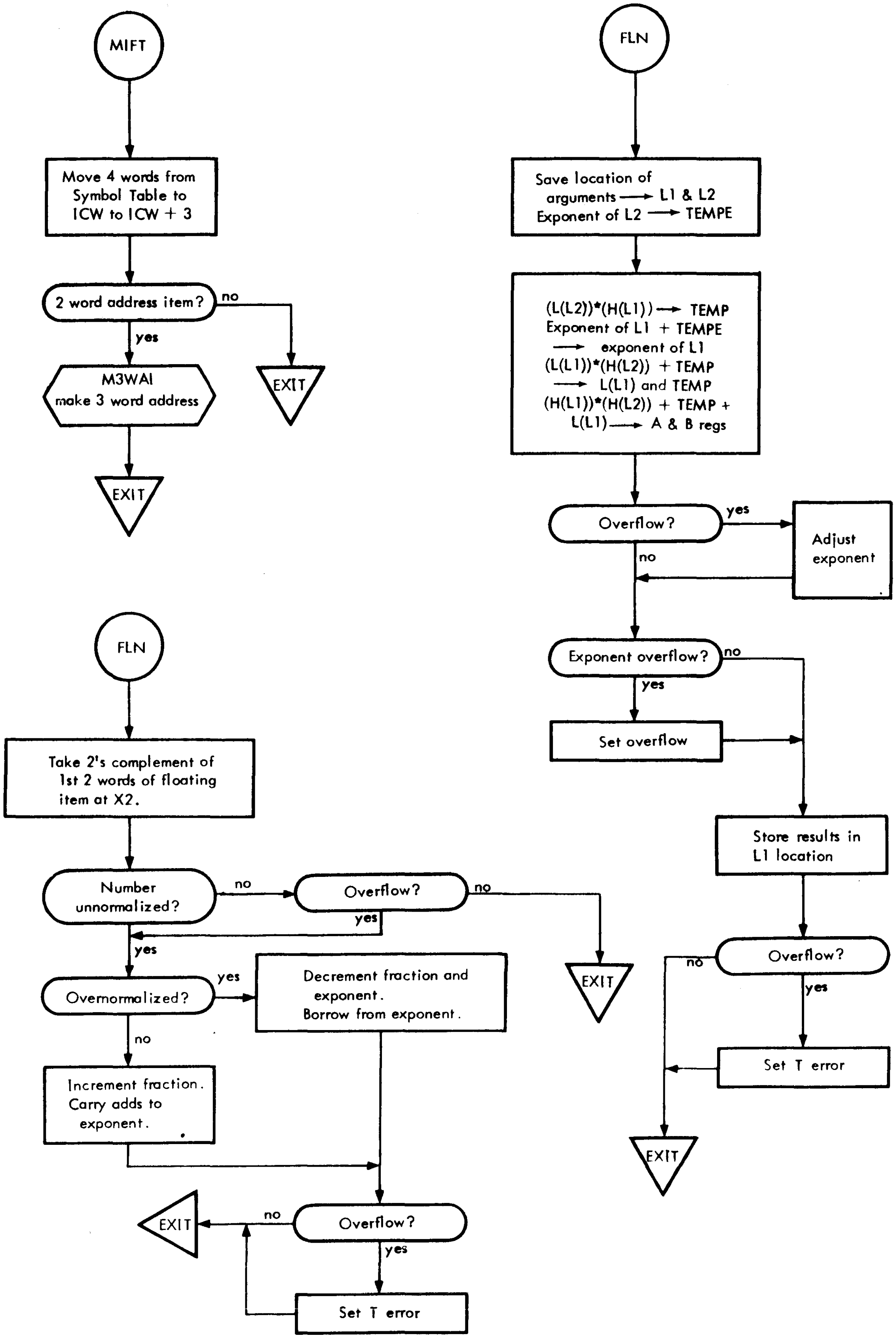

OUTP
Binary Output requested?
no
EXIT
yes
Word count > 0?
no
OUTP2
yes
OUTP1
CTYP = PTYP?
no
yes
24 > word count?
no
yes
PTYP = 0?
no
yes
PLOC + 1 = LOC
yes
no
FLUSH
empty buffer
OUTP2
RESET
reset buffer type
TYP
branch
table
ENDM
WORD**077777
+ + 0100000
→ WORD
MLOC → QLOC
DATAT
LOC → PLOC
WORD → buffer,
set relocation
flags for word
EXIT
POPRD
DEF
Move 3 words to
output buffer
from location
given in WORD
Subtype = 3?
no
yes
Move 1 more
word to buffer,
DWC + 1 → DWC
DWC + 3 → DWC
EXIT
ENDN
Last card type 3?
yes
EXIT
no
3 → PTYP
MLOC → QLOC
FLUSH
empty buffer
EXIT


FLUSH
PTYP 0?
yes
PTYP 3?
no
FLUSH1
no
yes
DWC + 1 ⟶ DWC
QLOC ⟶ DW1
L (M FLAGS) ⟶ MFLGTM
L (REL) ⟶ CHKS
L (DW1) + DWC ⟶ ICN
3 ⟶ OUTTMP
CHKS point to zero word?
yes
no
Move word at location given in CHKS to word addressed by ICN
Add contents of word addressed by MFLGTM to DW1.
DWC + 1 ⟶ DWC
no
OUTTMP < 0?
MFLGTM + 1 ⟶ MFLGTM
CHKS + 1 ⟶ CHKS
OUTTMP-1 ⟶ OUTTMP
yes
FLUSH1
Move PTYP and binary flag to 1st word in buffer
L (DW1) ⟶ CHKS
DWG - 1 ⟶ DWC
OUTPUT
output data word from CHKS
CHKS + 1 ⟶ CHKS
DWC-1 ⟶ DWC
no
DWC < 0 ?
yes
0 ⟶ DWC
WRITE
punch card
EXIT
RESET
LOC ⟶ QLOC
0 ⟶ REL
0 ⟶ CREL
0 ⟶ PREL
0 ⟶ SREL
CTYP ⟶ PTYP
EXIT

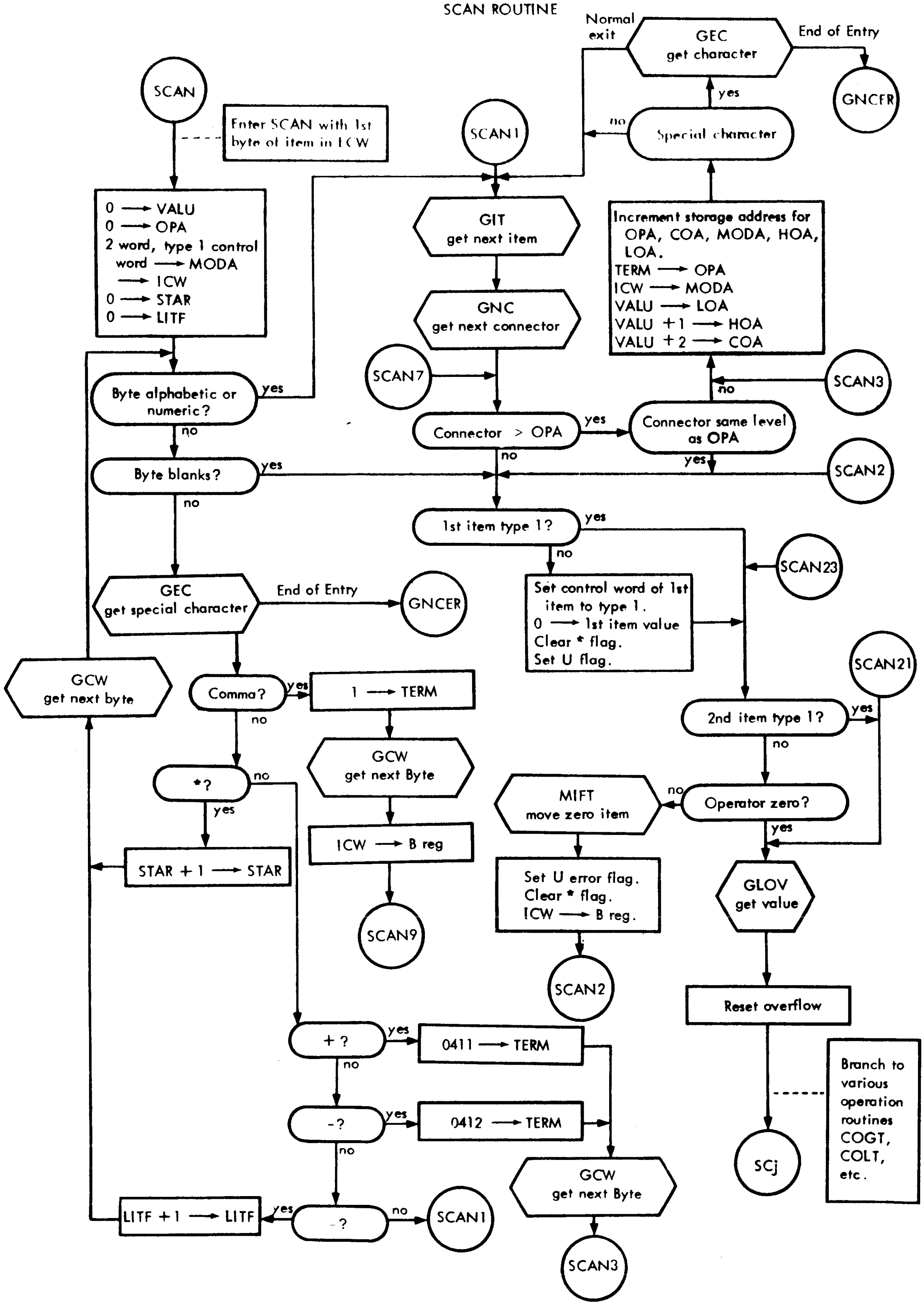

PAGE
0 → B reg
EDTST
test listing
No print
Print
HOME
eject page
LINSYM
WEOFL
L (PBUF) → A reg
L (EFMT) → X reg
LC → B reg
EFMT
write end-of-file
mark
EXIT
EPRNT
0 → B reg
EDTST
test to print
No print
EXIT
Print
EDE
set up error flags
PRNT
print line
EXIT

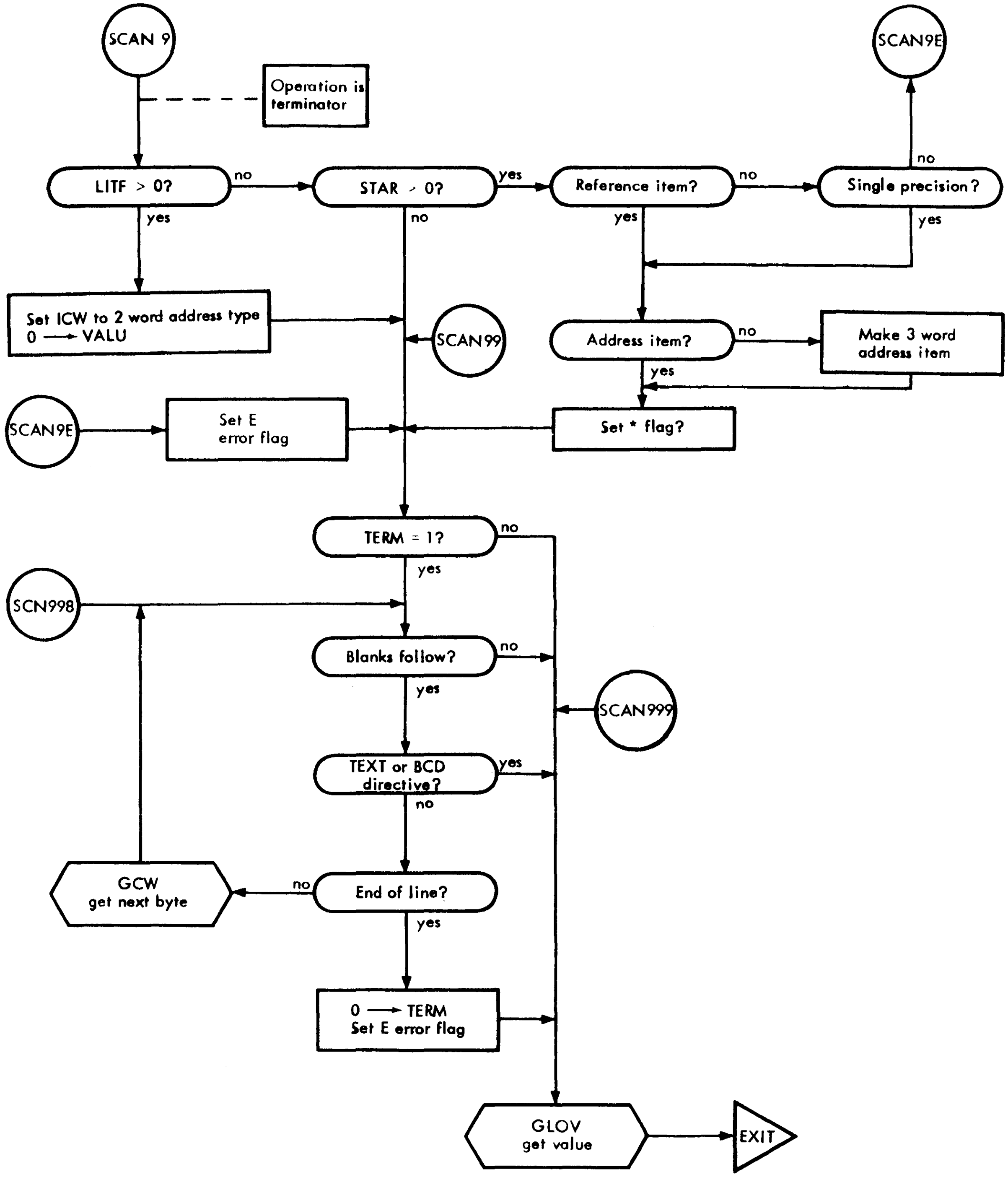

DED
SCAN
evaluate expression
Value type?
no
Set E error
yes
Floating point?
no
Double precision
no
GLOV
get value
yes
yes
MFOI
make output item
Value → VALU
Signs → VALUI
VALU → WORD
VALUI → WORDI
Set DPPF.
Set 2 word form
control word
2 → CCINC
RDPI
reverse item
TERM = 1?
no
LNFRM
yes
LBTST
define label
2 → B reg
EDIT
print line
CC + 2 → CC
MFOI
EXIT

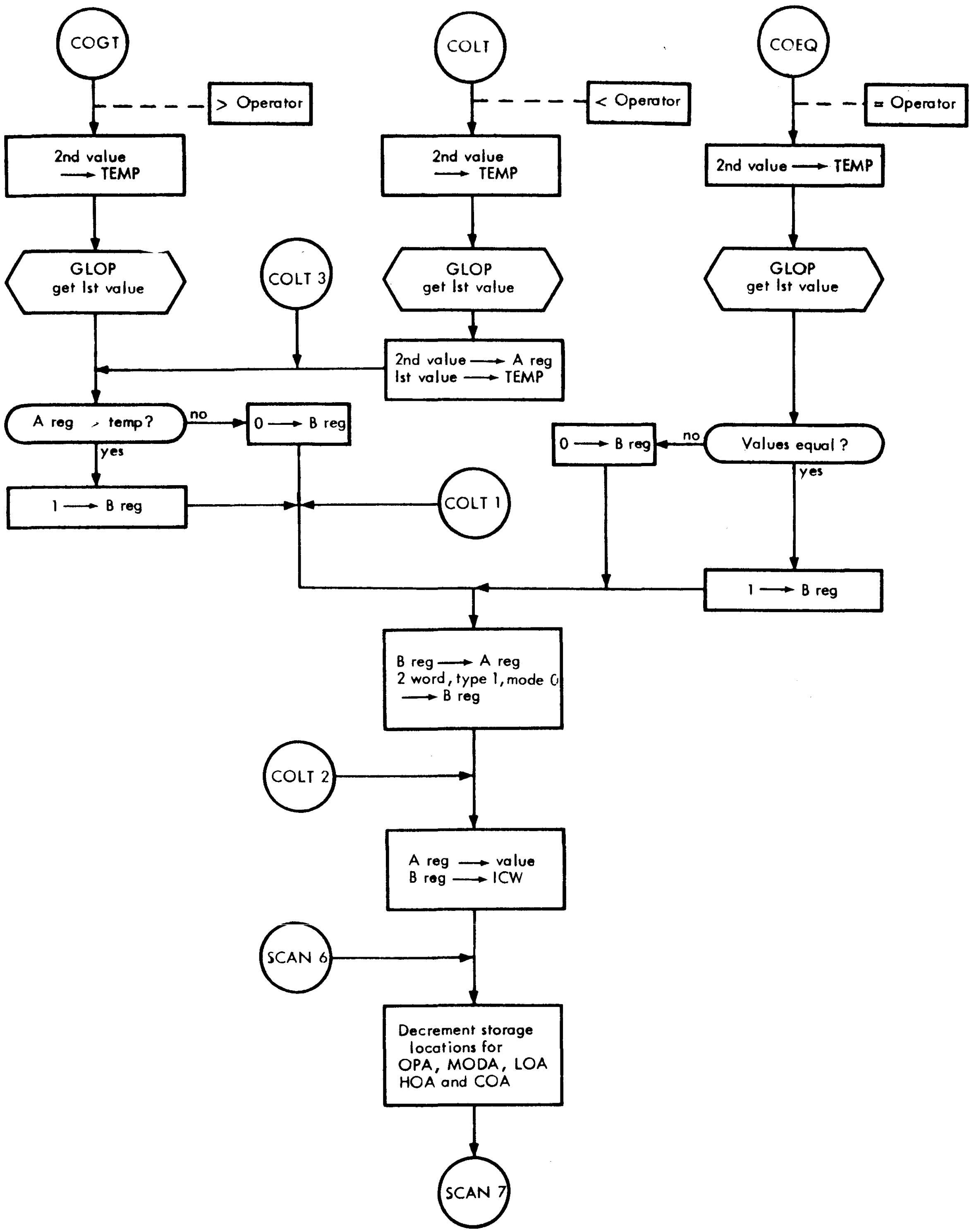

RDPI
EXIT
GLOV
Item at ICW a 3 word address item
no
yes
VALU → A reg
ICW → B reg
VALU1 → A reg
ICW → B reg
EXIT
EXIT
M3WAI
Bits 9–23 of VALU → VALU + 1
Bits 0–8 of VALU → VALU
ICW++2^21 → ICW
EXIT
GLOP
Item at MODA a 3 word address item?
no
yes
LOA → A reg
HOA → A reg
EXIT
EXIT

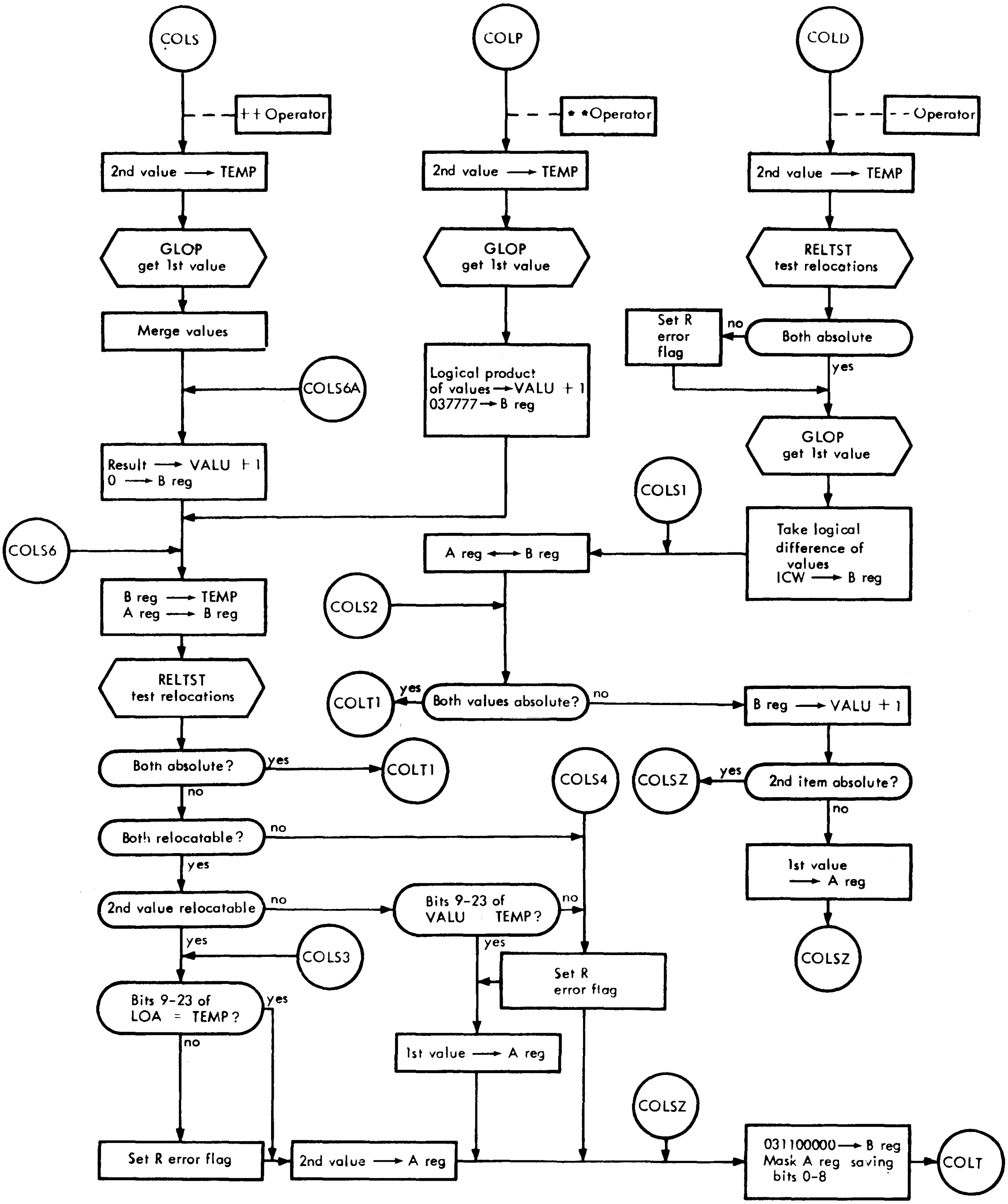

MIFT
Move 4 words from Symbol Table to ICW to ICW + 3
2 word address item?
no
yes
M3WAI
make 3 word address
EXIT
EXIT
FLN
Save location of arguments ⟶ L1 & L2
Exponent of L2 ⟶ TEMPE
(L(L2))*(H(L1)) ⟶ TEMP
Exponent of L1 + TEMPE ⟶ exponent of L1
(L(L1))*(H(L2)) + TEMP ⟶ L(L1) and TEMP
(H(L1))*(H(L2)) + TEMP + L(L1) ⟶ A & B regs
Overflow?
yes
Adjust exponent
no
Exponent overflow?
no
yes
Set overflow
Store results in L1 location
Overflow?
no
yes
Set T error
EXIT
FLN
Take 2's complement of 1st 2 words of floating item at X2.
Number unnormalized?
no
Overflow?
no
yes
yes
EXIT
Overnormalized?
yes
Decrement fraction and exponent.
Borrow from exponent.
no
Increment fraction. Carry adds to exponent.
Overflow?
no
EXIT
yes
Set T error

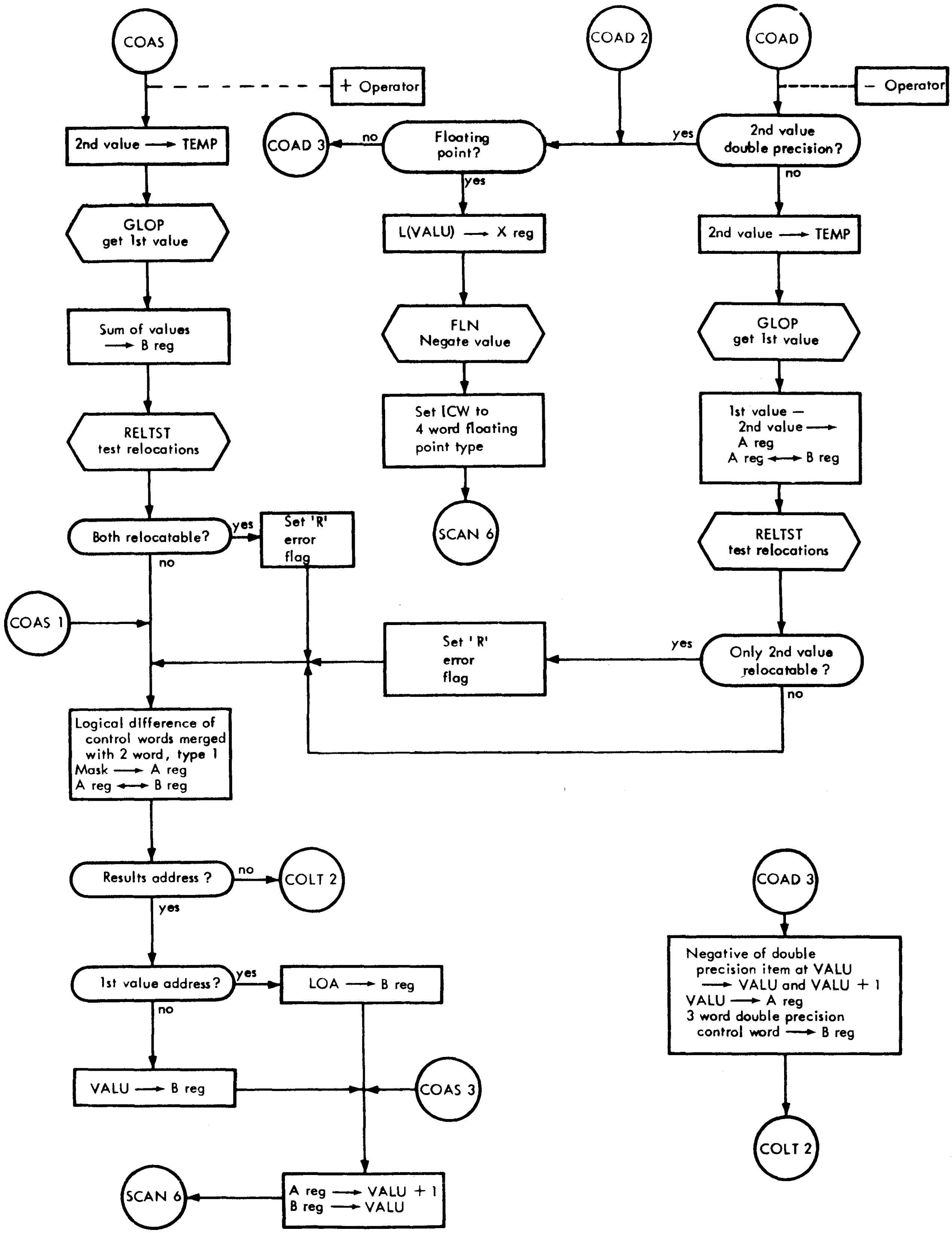
ASSEMBLR
SCAN ROUTINE

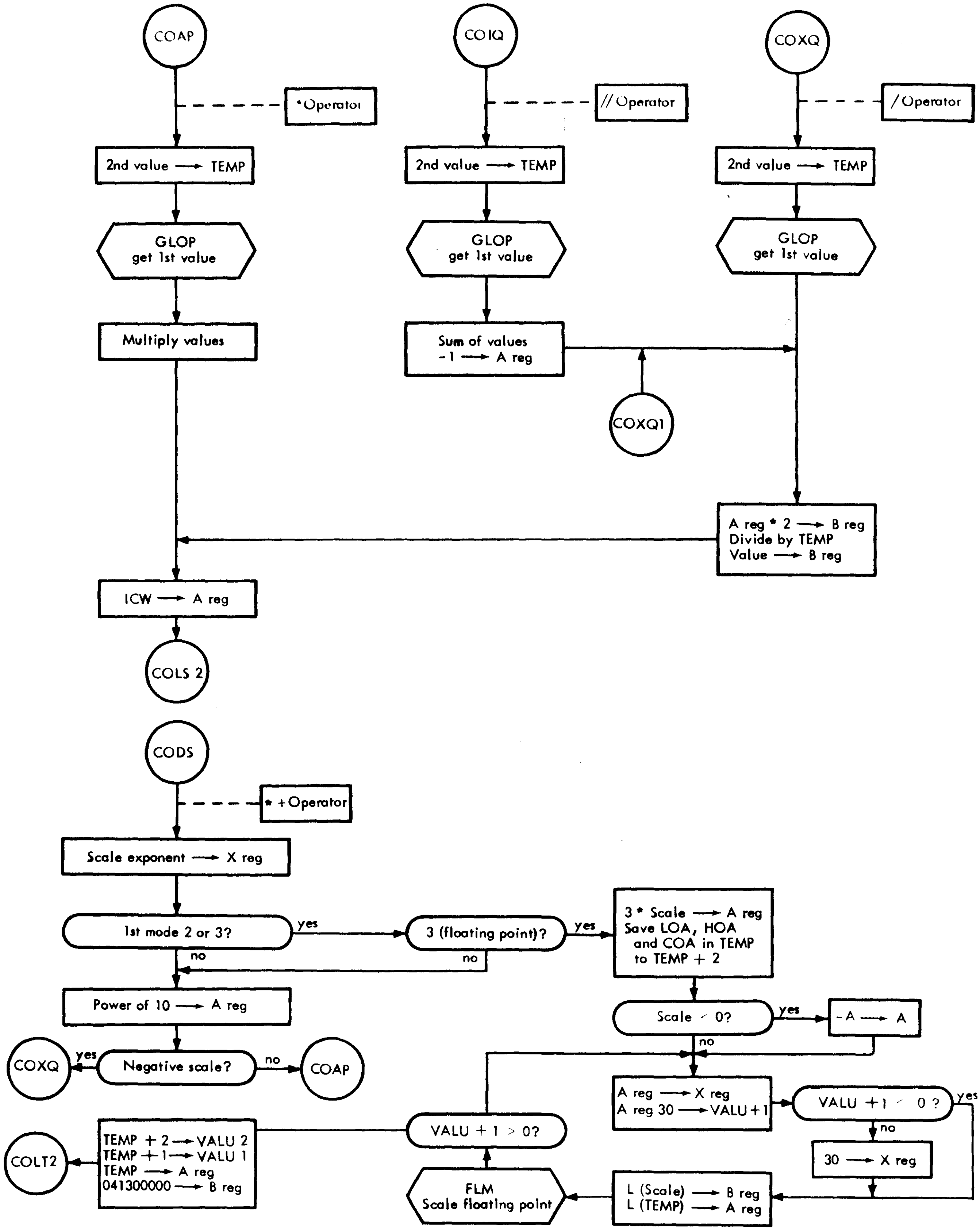


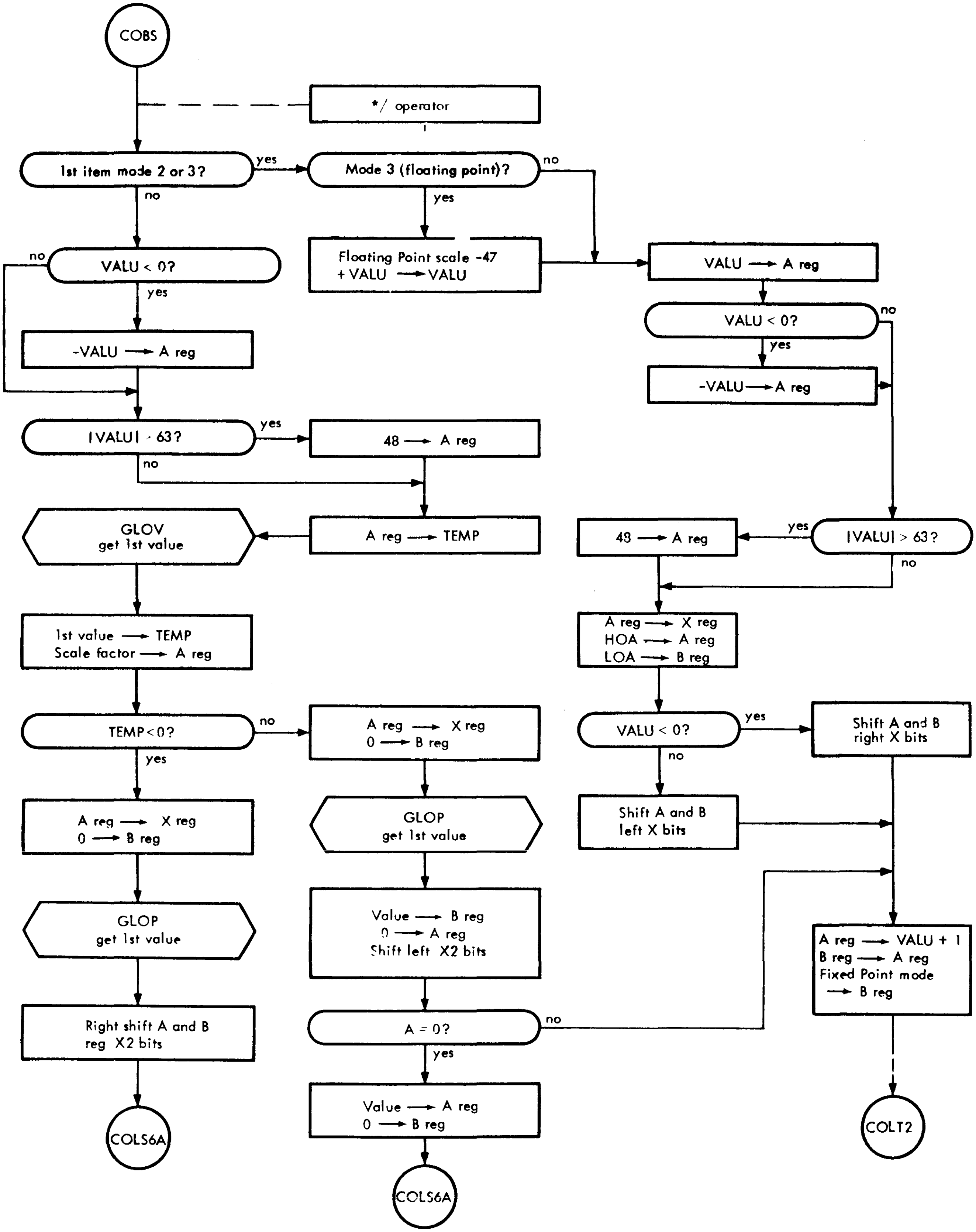

SCAN 9
Operation is terminator
LITF > 0?
no
yes
STAR ≠ 0?
yes
no
Reference item?
no
yes
Single precision?
no
yes
SCAN9E
Set ICW to 2 word address type
0 → VALU
SCAN99
Address item?
no
yes
Make 3 word address item
SCAN9E
Set E error flag
Set * flag?
TERM = 1?
no
yes
SCN998
Blanks follow?
no
yes
SCAN999
TEXT or BCD directive?
yes
no
GCW
get next byte
no
End of line?
yes
0 → TERM
Set E error flag
GLOV
get value
EXIT

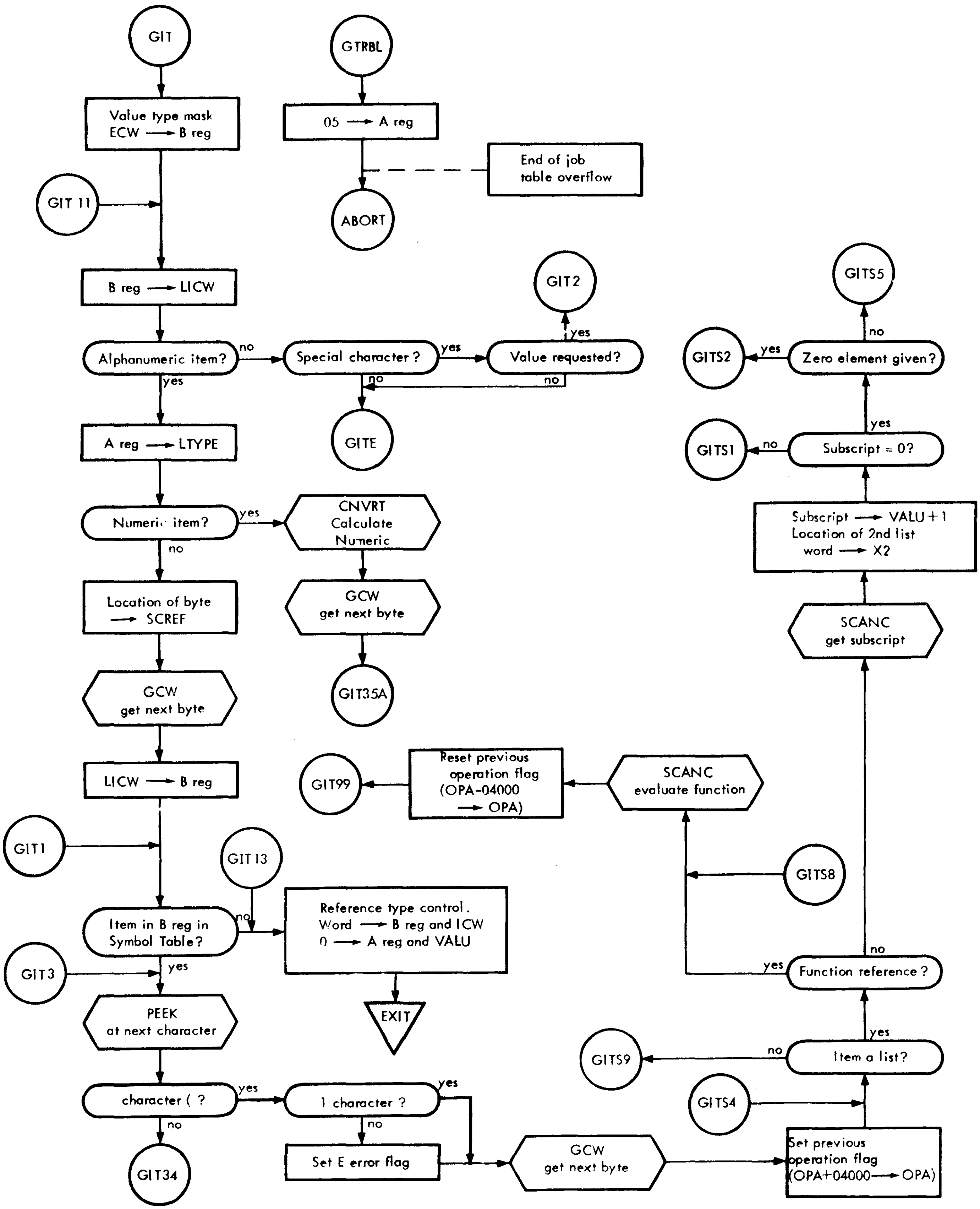

COGT
> Operator
2nd value → TEMP
GLOP get Ist value
COLT 3
A reg > temp?
no
yes
0 → B reg
1 → B reg
COLT
< Operator
2nd value → TEMP
GLOP get Ist value
2nd value → A reg
Ist value → TEMP
COLT 1
COEQ
= Operator
2nd value → TEMP
GLOP get Ist value
Values equal ?
no
yes
0 → B reg
1 → B reg
B reg → A reg
2 word, type 1, mode 0
→ B reg
COLT 2
A reg → value
B reg → ICW
SCAN 6
Decrement storage locations for OPA, MODA, LOA HOA and COA
SCAN 7

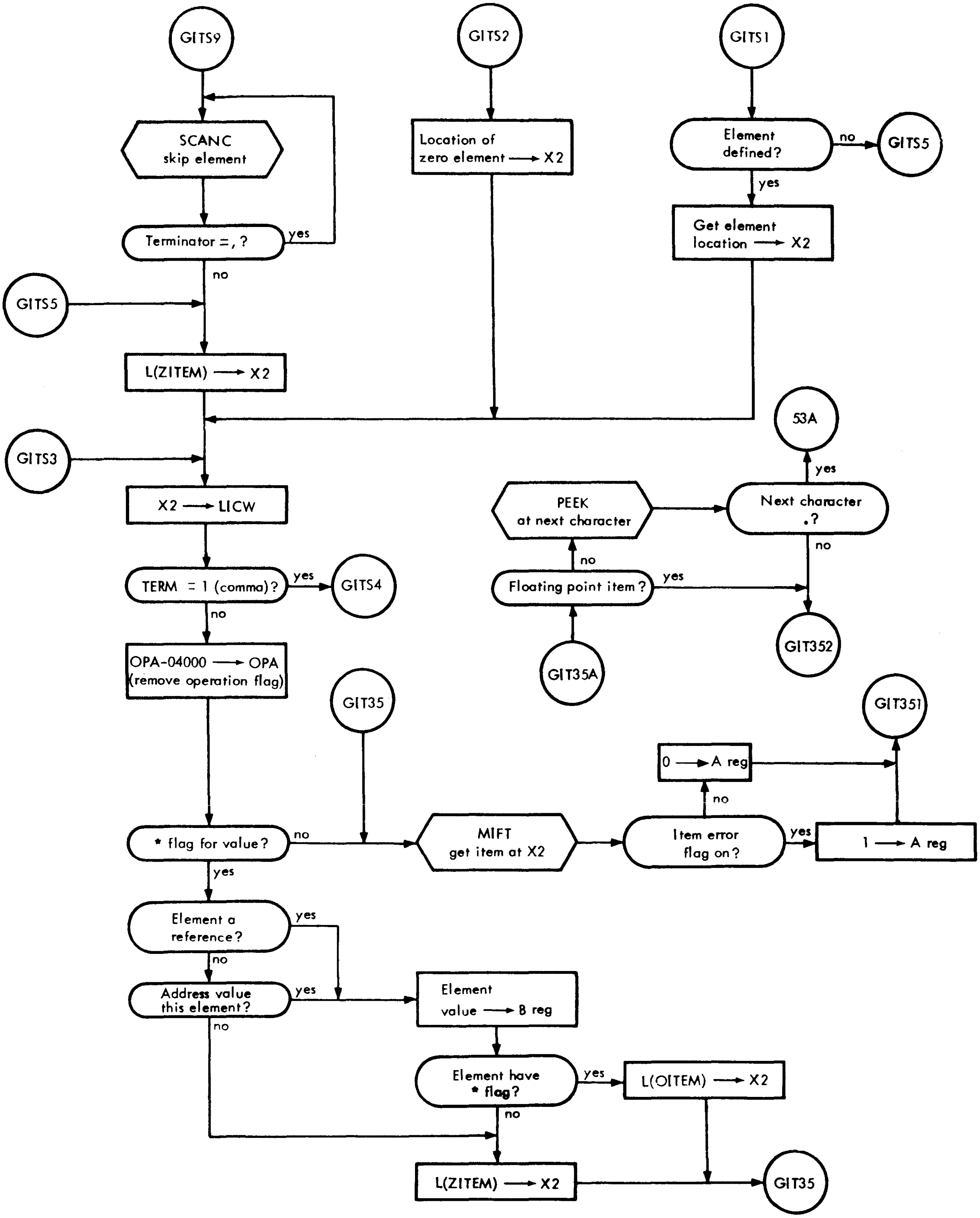

COLS
+ + Operator
2nd value → TEMP
GLOP
get 1st value
Merge values
COLS6A
Result → VALU + 1
0 → B reg
COLS6
B reg → TEMP
A reg → B reg
RELTST
test relocations
Both absolute?
yes
COLT1
no
Both relocatable?
no
yes
2nd value relocatable
no
yes
COLS3
Bits 9-23 of
LOA = TEMP?
yes
no
Set R error flag
2nd value → A reg
COLP
* *Operator
2nd value → TEMP
GLOP
get 1st value
Logical product
of values → VALU + 1
037777 → B reg
A reg ↔ B reg
COLS2
COLT1
yes
Both values absolute?
no
COLS4
Bits 9-23 of
VALU TEMP?
no
yes
Set R
error flag
1st value → A reg
COLD
- - Operator
2nd value → TEMP
RELTST
test relocations
Set R
error
flag
no
Both absolute
yes
GLOP
get 1st value
COLS1
Take logical
difference of
values
ICW → B reg
B reg → VALU + 1
COLSZ
yes
2nd item absolute?
no
1st value
→ A reg
COLSZ
COLSZ
031100000 → B reg
Mask A reg saving
bits 0-8
COLT

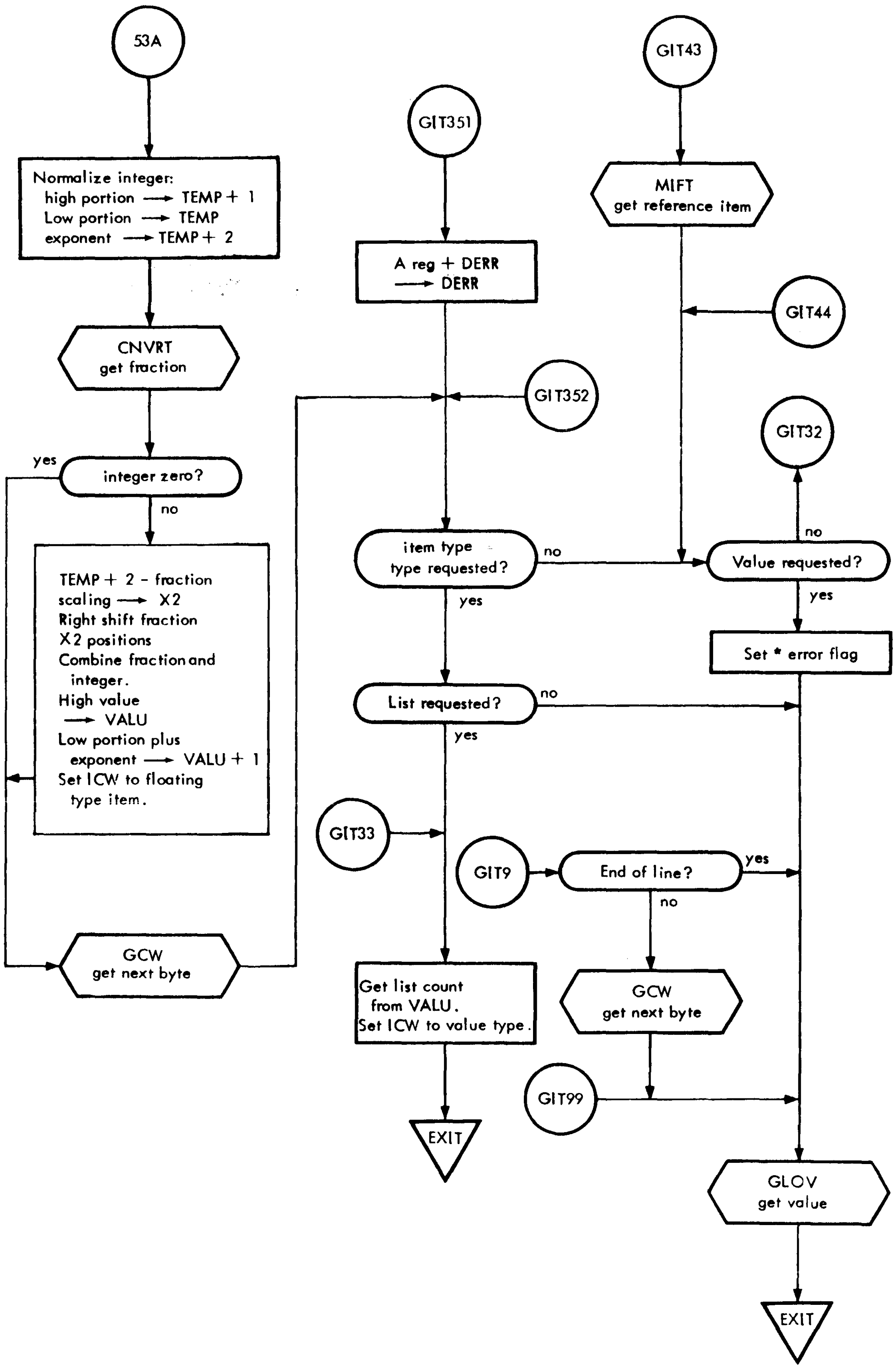

COAS
+ Operator
2nd value → TEMP
GLOP
get 1st value
Sum of values
→ B reg
RELTST
test relocations
Both relocatable?
yes
no
Set 'R'
error
flag
COAS 1
Logical difference of
control words merged
with 2 word, type 1
Mask → A reg
A reg ↔ B reg
Results address ?
no
COLT 2
yes
1st value address?
yes
LOA → B reg
no
VALU → B reg
COAS 3
A reg → VALU + 1
B reg → VALU
SCAN 6
COAD 2
COAD
– Operator
2nd value
double precision?
yes
no
Floating
point?
COAD 3
yes
L(VALU) → X reg
FLN
Negate value
Set ICW to
4 word floating
point type
SCAN 6
2nd value → TEMP
GLOP
get 1st value
1st value –
2nd value →
A reg
A reg ↔ B reg
RELTST
test relocations
Only 2nd value
relocatable ?
yes
no
Set 'R'
error
flag
COAD 3
Negative of double
precision item at VALU
→ VALU and VALU + 1
VALU → A reg
3 word double precision
control word → B reg
COLT 2

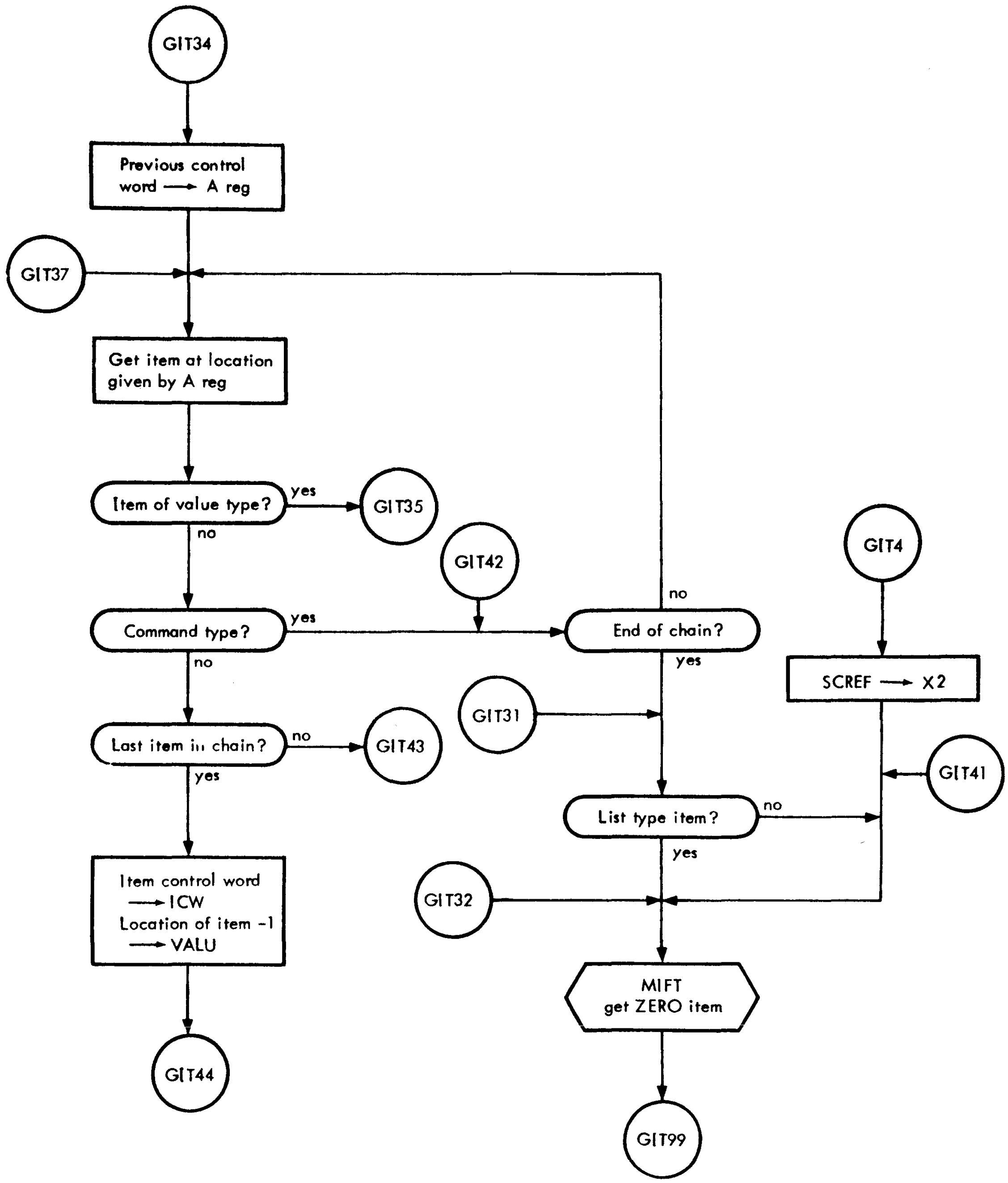

COAP
*Operator
2nd value → TEMP
GLOP get 1st value
Multiply values
COIQ
// Operator
2nd value → TEMP
GLOP get 1st value
Sum of values -1 → A reg
COXQ1
COXQ
/ Operator
2nd value → TEMP
GLOP get 1st value
A reg * 2 → B reg
Divide by TEMP
Value → B reg
ICW → A reg
COLS 2
CODS
* + Operator
Scale exponent → X reg
1st mode 2 or 3?
yes
no
3 (floating point)?
yes
no
3 * Scale → A reg
Save LOA, HOA and COA in TEMP to TEMP + 2
Power of 10 → A reg
Scale < 0?
yes
no
-A → A
COXQ
yes
Negative scale?
no
COAP
A reg → X reg
A reg 30 → VALU + 1
VALU + 1 < 0 ?
yes
no
30 → X reg
COLT2
TEMP + 2 → VALU 2
TEMP + 1 → VALU 1
TEMP → A reg
041300000 → B reg
VALU + 1 > 0?
FLM
Scale floating point
L (Scale) → B reg
L (TEMP) → A reg

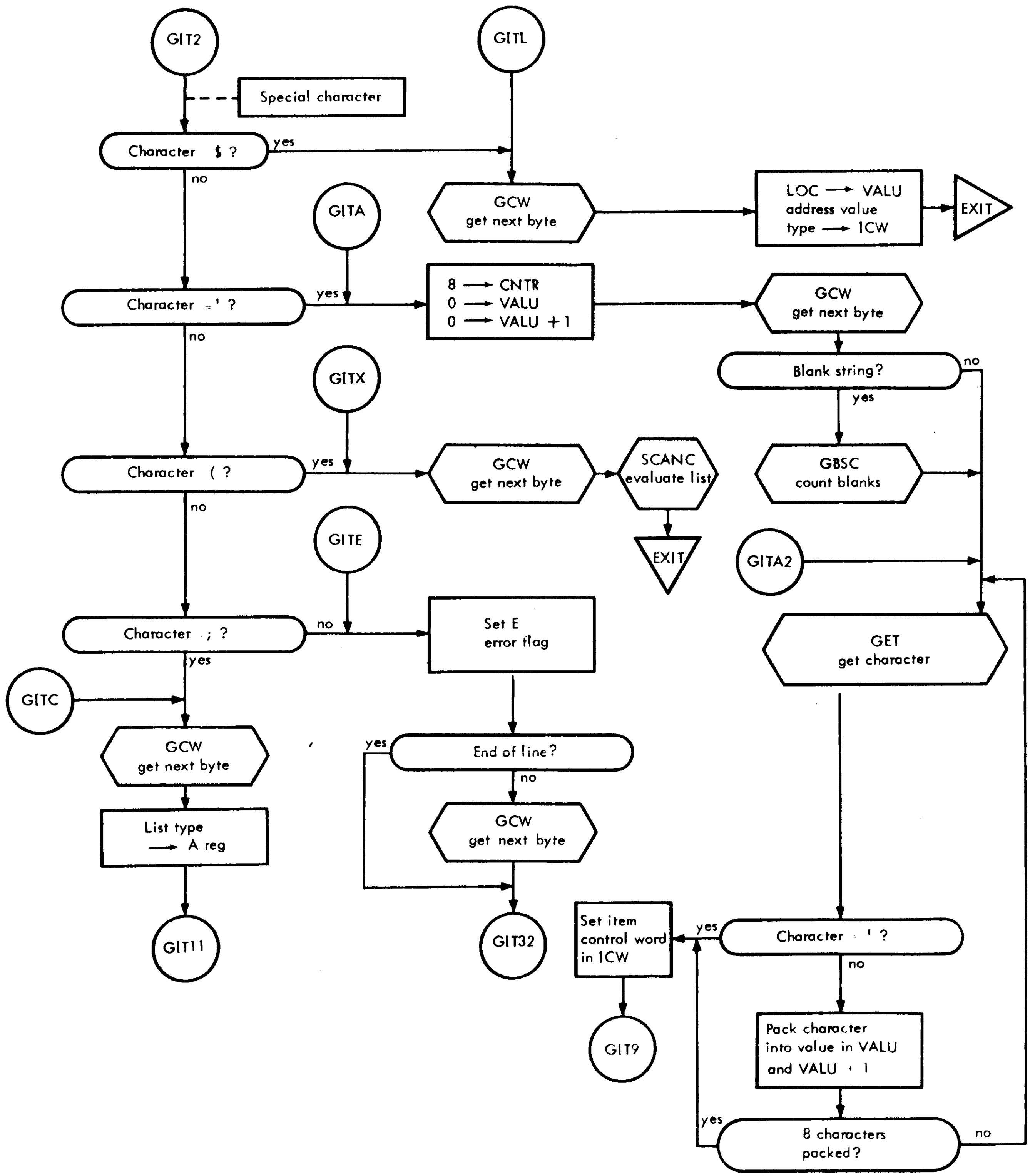

COBS
*/ operator
1st item mode 2 or 3?
yes
no
Mode 3 (floating point)?
no
yes
Floating Point scale -47
+ VALU → VALU
VALU → A reg
VALU < 0?
no
yes
-VALU → A reg
VALU < 0?
no
yes
-VALU → A reg
\|VALU\| > 63?
yes
no
48 → A reg
A reg → TEMP
GLOV
get 1st value
1st value → TEMP
Scale factor → A reg
TEMP < 0?
no
yes
A reg → X reg
0 → B reg
GLOP
get 1st value
Right shift A and B
reg X2 bits
COLS6A
A reg → X reg
0 → B reg
GLOP
get 1st value
Value → B reg
0 → A reg
Shift left X2 bits
A = 0?
no
yes
Value → A reg
0 → B reg
COLS6A
\|VALU\| > 63?
yes
no
48 → A reg
A reg → X reg
HOA → A reg
LOA → B reg
VALU < 0?
yes
no
Shift A and B
right X bits
Shift A and B
left X bits
A reg → VALU + 1
B reg → A reg
Fixed Point mode
→ B reg
COLT2

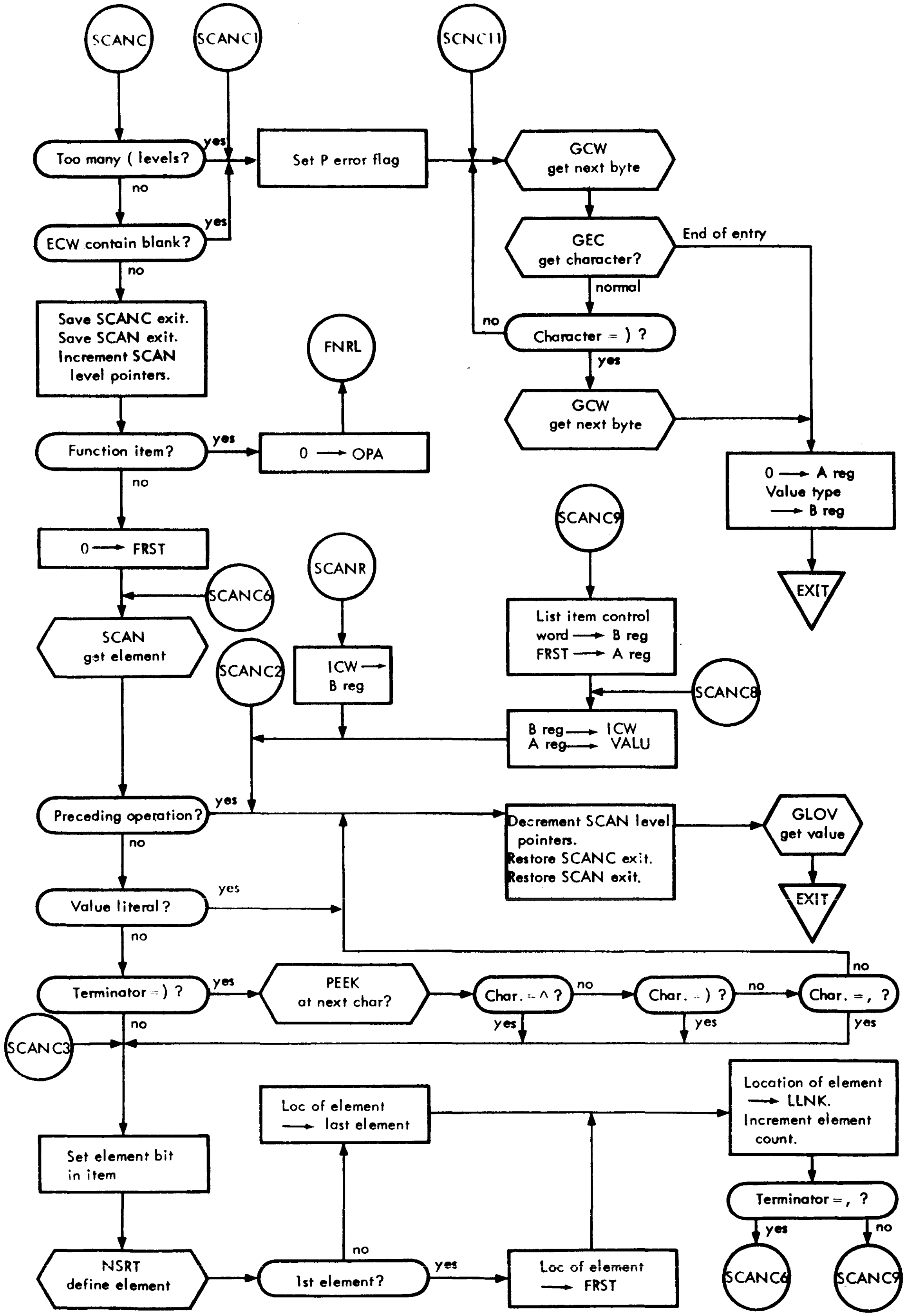

GIT
Value type mask
ECW → B reg
GIT 11
B reg → LICW
Alphanumeric item?
no
yes
A reg → LTYPE
Numeric item?
yes
no
Location of byte → SCREF
GCW
get next byte
LICW → B reg
GIT1
GIT 13
Item in B reg in Symbol Table?
no
yes
GIT3
PEEK
at next character
character (?
yes
no
GIT34
GTRBL
05 → A reg
End of job
table overflow
ABORT
Special character ?
yes
no
GITE
CNVRT
Calculate
Numeric
GCW
get next byte
GIT35A
Value requested?
yes
no
GIT2
GIT99
Reset previous
operation flag
(OPA-04000 → OPA)
SCANC
evaluate function
GITS8
Reference type control.
Word → B reg and ICW
0 → A reg and VALU
EXIT
1 character ?
yes
no
Set E error flag
GCW
get next byte
Set previous
operation flag
(OPA+04000 → OPA)
GITS4
Item a list?
yes
no
GITS9
Function reference ?
yes
no
SCANC
get subscript
Subscript → VALU+1
Location of 2nd list
word → X2
Subscript = 0?
no
yes
GITS1
Zero element given?
yes
no
GITS2
GITS5

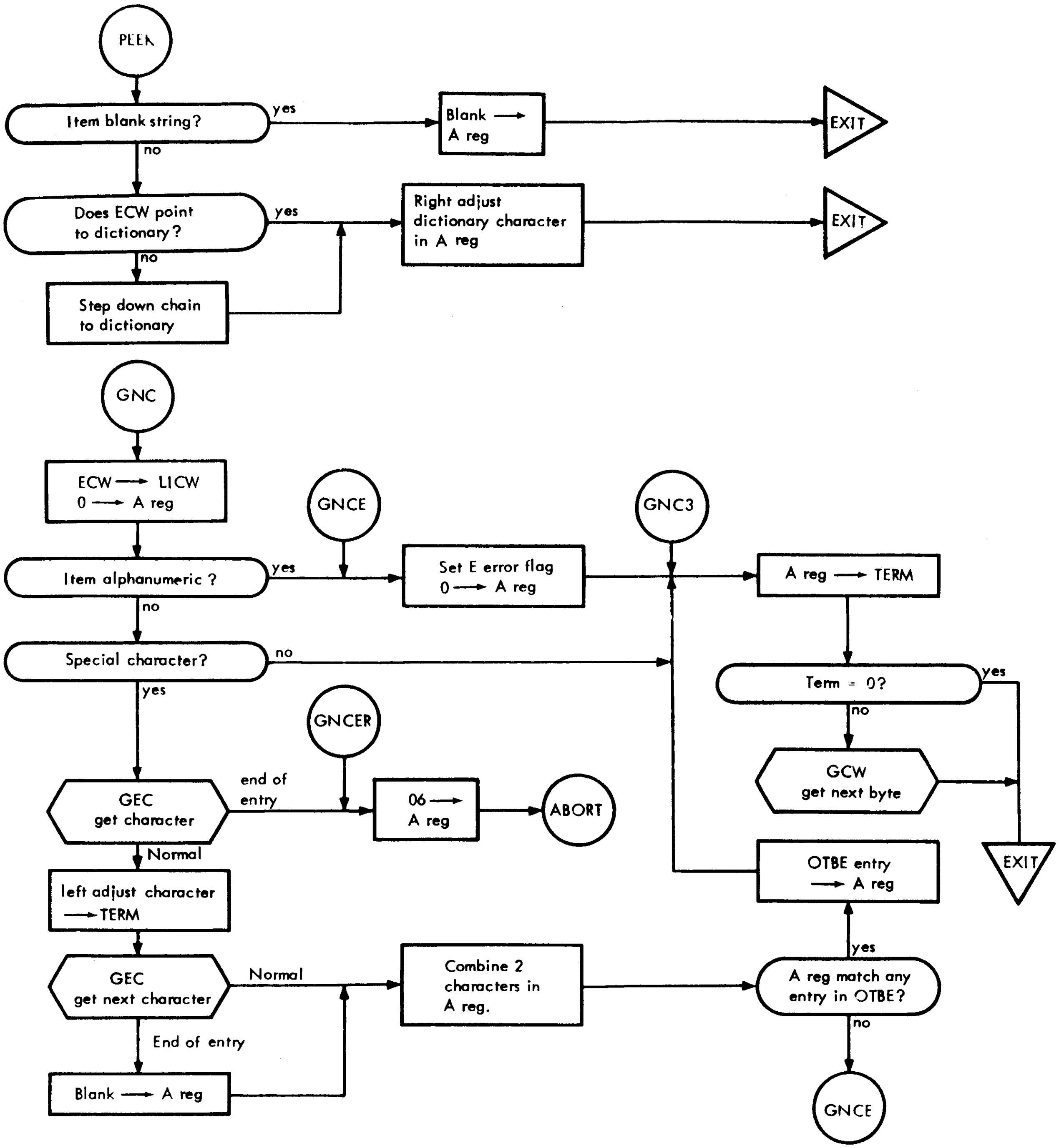

GITS9
GITS2
GITS1
SCANC
skip element
Location of
zero element ——→ X2
Element
defined?
no
GITS5
yes
Get element
location ——→ X2
Terminator =, ?
yes
no
GITS5
L(ZITEM) ——→ X2
53A
GITS3
yes
X2 ——→ LICW
PEEK
at next character
Next character
.?
no
no
TERM = 1 (comma)?
yes
GITS4
Floating point item?
yes
no
GIT352
OPA-04000 ——→ OPA
(remove operation flag)
GIT35A
GIT35
GIT351
0 ——→ A reg
no
* flag for value?
no
MIFT
get item at X2
Item error
flag on?
yes
1 ——→ A reg
yes
Element a
reference?
yes
no
Address value
this element?
yes
Element
value ——→ B reg
no
Element have
* flag?
yes
L(OITEM) ——→ X2
no
L(ZITEM) ——→ X2
GIT35

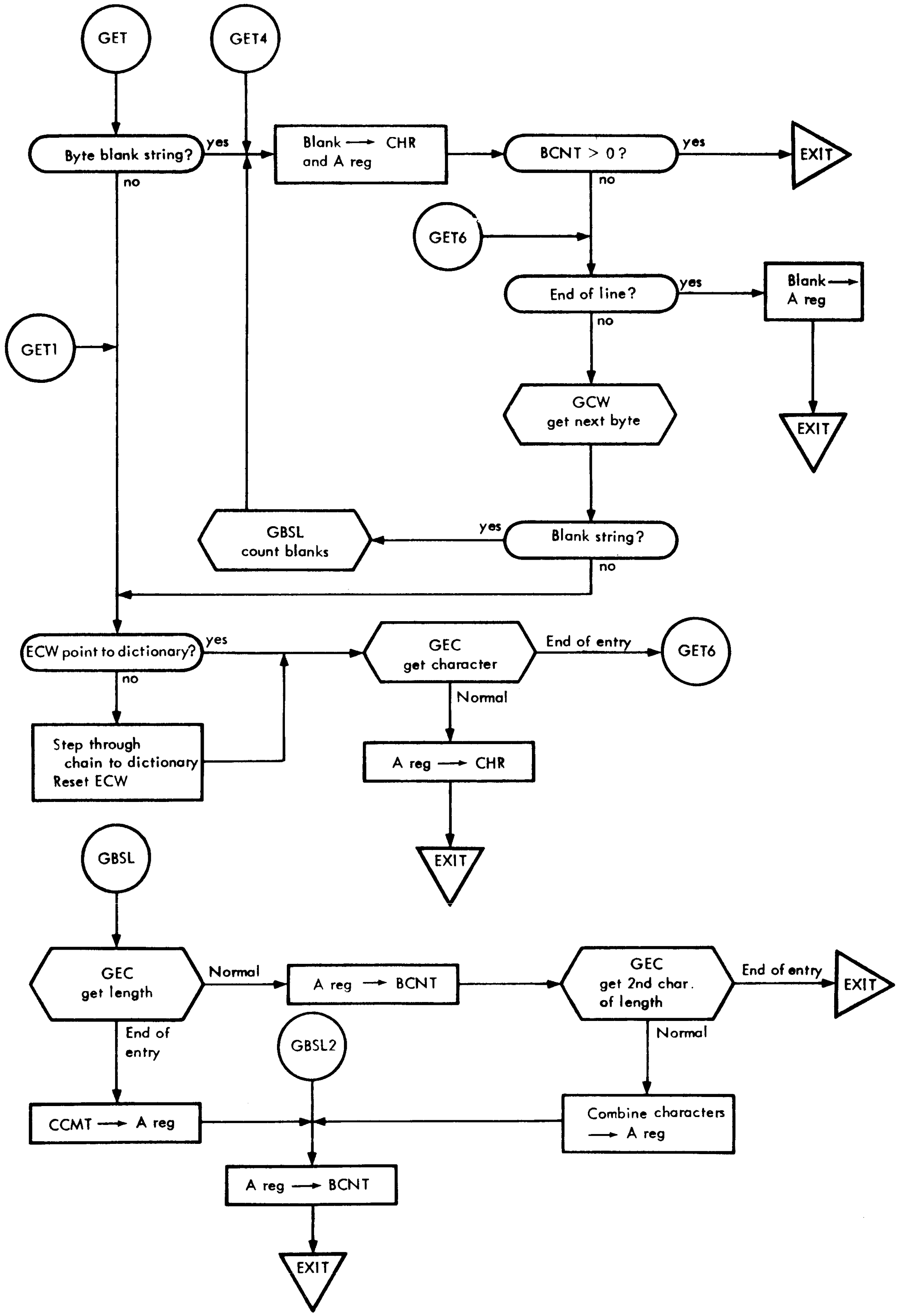

53A
Normalize integer:
high portion → TEMP + 1
Low portion → TEMP
exponent → TEMP + 2
CNVRT
get fraction
integer zero?
yes
no
TEMP + 2 - fraction scaling → X2
Right shift fraction X2 positions
Combine fraction and integer.
High value → VALU
Low portion plus exponent → VALU + 1
Set ICW to floating type item.
GCW
get next byte
GIT351
A reg + DERR → DERR
GIT352
item type type requested?
no
yes
List requested?
no
yes
GIT33
Get list count from VALU.
Set ICW to value type.
EXIT
GIT43
MIFT
get reference item
GIT44
GIT32
Value requested?
no
yes
Set * error flag
GIT9
End of line?
yes
no
GCW
get next byte
GIT99
GLOV
get value
EXIT

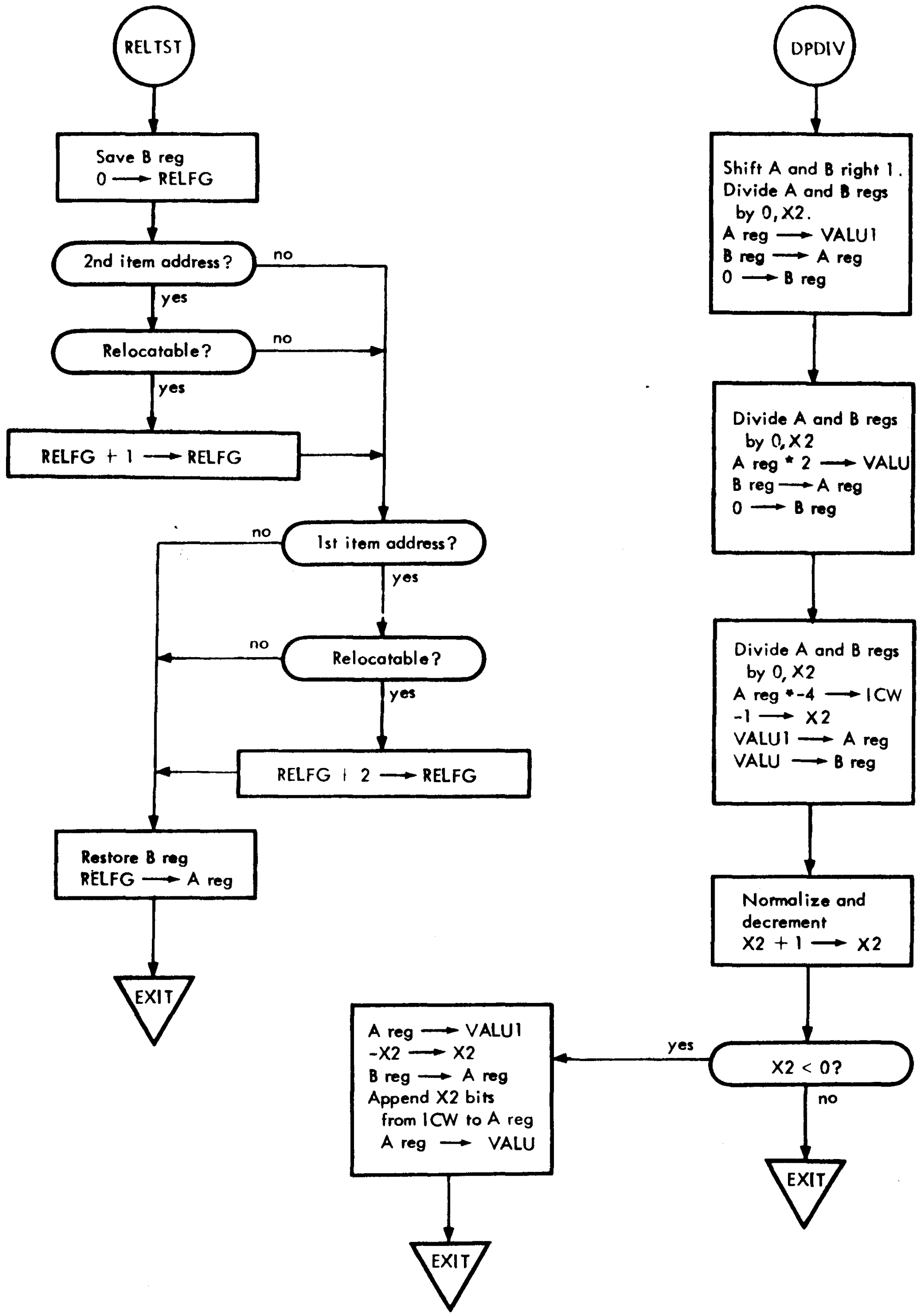

GIT34
Previous control word ⟶ A reg
GIT37
Get item at location given by A reg
Item of value type?
yes
GIT35
no
GIT42
Command type?
yes
End of chain?
no
yes
GIT4
SCREF ⟶ X2
no
GIT31
Last item in chain?
no
GIT43
yes
GIT41
List type item?
no
yes
Item control word ⟶ ICW
Location of item -1 ⟶ VALU
GIT32
MIFT
get ZERO item
GIT44
GIT99

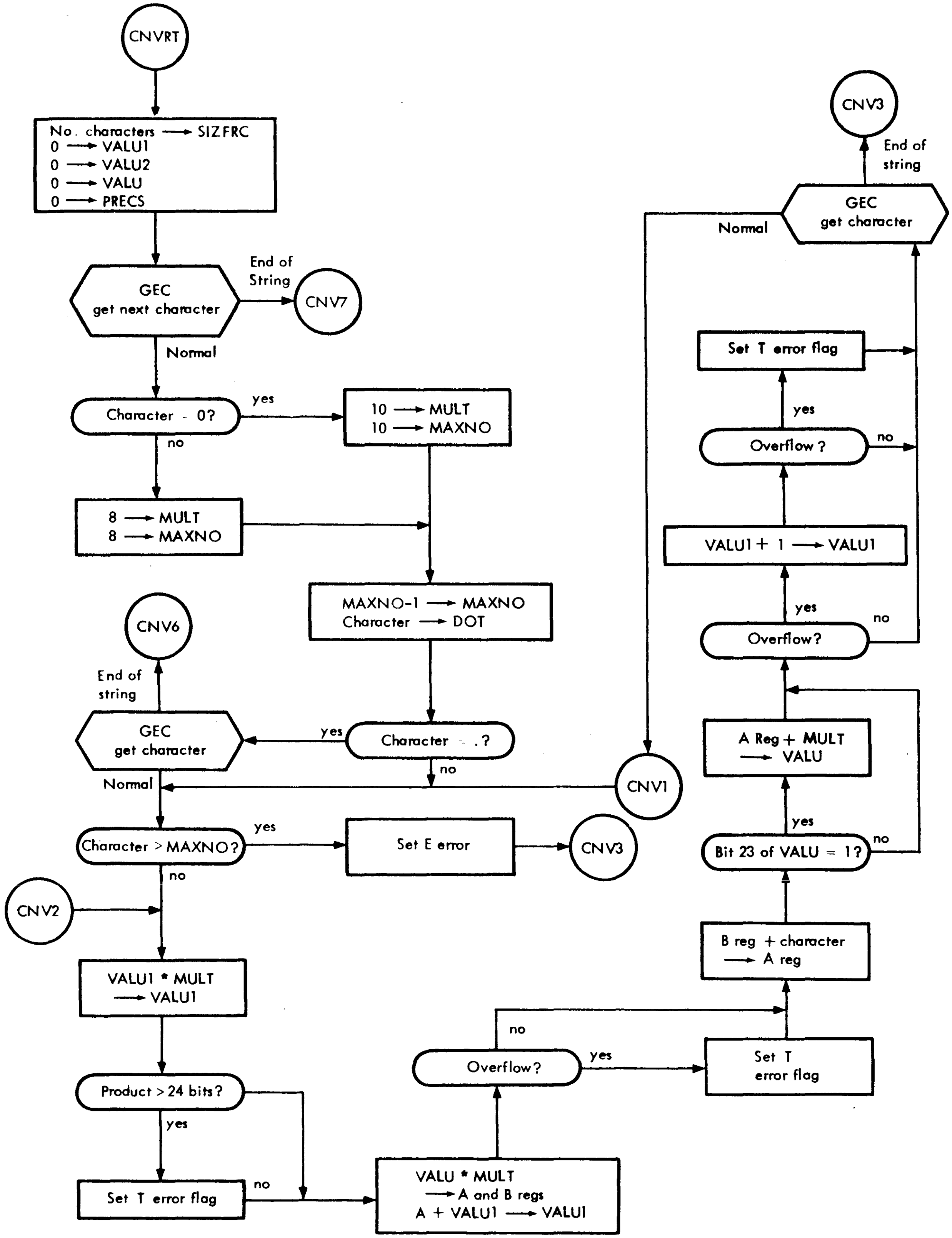

GIT2
GITL
Special character
Character $?
yes
no
GCW
get next byte
LOC → VALU
address value
type → ICW
EXIT
GITA
Character ' ?
yes
no
8 → CNTR
0 → VALU
0 → VALU +1
GCW
get next byte
Blank string?
no
yes
GITX
Character (?
yes
no
GCW
get next byte
SCANC
evaluate list
EXIT
GBSC
count blanks
GITE
GITA2
Character ; ?
no
yes
Set E
error flag
GET
get character
GITC
GCW
get next byte
End of line?
yes
no
List type
→ A reg
GCW
get next byte
GIT11
GIT32
Set item
control word
in ICW
Character ' ?
yes
no
GIT9
Pack character
into value in VALU
and VALU + 1
8 characters
packed?
yes
no


SCANC
SCANC1
SCNC11
Too many (levels?
yes
no
Set P error flag
GCW
get next byte
ECW contain blank?
yes
no
GEC
get character?
End of entry
normal
no
Character =) ?
yes
Save SCANC exit.
Save SCAN exit.
Increment SCAN
level pointers.
FNRL
GCW
get next byte
Function item?
yes
0 → OPA
no
0 → A reg
Value type
→ B reg
SCANC9
0 → FRST
SCANR
EXIT
SCANC6
SCAN
get element
List item control
word → B reg
FRST → A reg
SCANC2
ICW →
B reg
SCANC8
B reg → ICW
A reg → VALU
Preceding operation?
yes
no
Decrement SCAN level
pointers.
Restore SCANC exit.
Restore SCAN exit.
GLOV
get value
Value literal?
yes
no
EXIT
Terminator =) ?
yes
PEEK
at next char?
Char. = ^ ?
no
Char. =) ?
no
Char. = , ?
no
yes
yes
yes
no
SCANC3
Loc of element
→ last element
Location of element
→ LLNK.
Increment element
count.
Set element bit
in item
Terminator = , ?
yes
no
NSRT
define element
no
1st element?
yes
Loc of element
→ FRST
SCANC6
SCANC9


PEEK
Item blank string?
yes
no
Blank → A reg
EXIT
Does ECW point to dictionary?
yes
no
Right adjust dictionary character in A reg
EXIT
Step down chain to dictionary
GNC
ECW → LICW
0 → A reg
GNCE
GNC3
Item alphanumeric ?
yes
no
Set E error flag
0 → A reg
A reg → TERM
Special character?
no
yes
Term = 0?
yes
no
GNCER
GCW
get next byte
GEC
get character
end of entry
06 → A reg
ABORT
Normal
OTBE entry
→ A reg
EXIT
left adjust character
→ TERM
yes
GEC
get next character
Normal
Combine 2 characters in A reg.
A reg match any entry in OTBE?
End of entry
no
Blank → A reg
GNCE

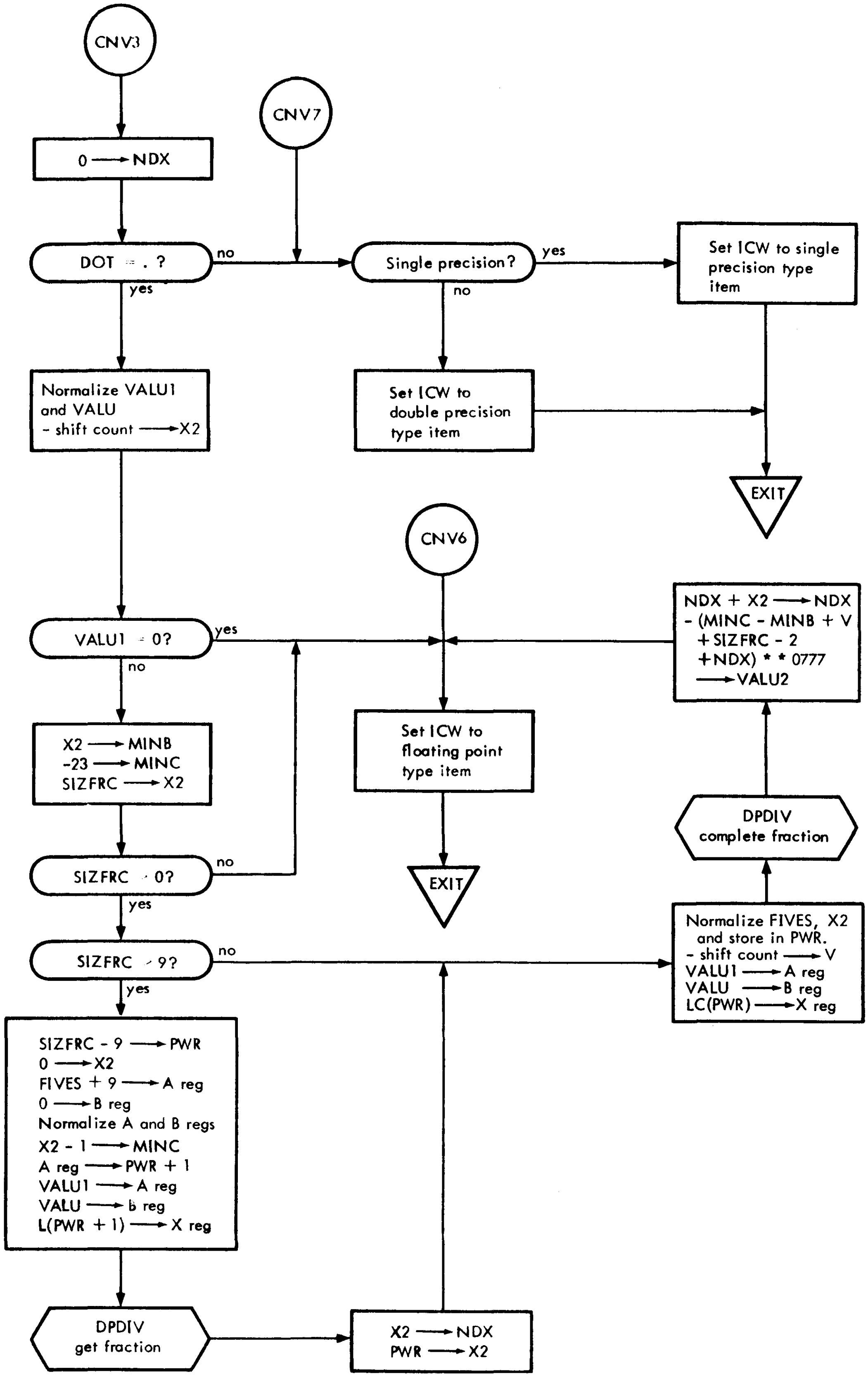

GET
GET4
Byte blank string?
yes
no
Blank → CHR and A reg
BCNT > 0 ?
yes
no
EXIT
GET6
End of line?
yes
no
Blank → A reg
EXIT
GET1
GCW get next byte
GBSL count blanks
yes
Blank string?
no
ECW point to dictionary?
yes
no
GEC get character
End of entry
GET6
Normal
Step through chain to dictionary Reset ECW
A reg → CHR
EXIT
GBSL
GEC get length
Normal
A reg → BCNT
GEC get 2nd char. of length
End of entry
EXIT
End of entry
GBSL2
Normal
CCMT → A reg
Combine characters → A reg
A reg → BCNT
EXIT


RELTST
Save B reg
0 → RELFG
2nd item address?
no
yes
Relocatable?
no
yes
RELFG + 1 → RELFG
1st item address?
no
yes
Relocatable?
no
yes
RELFG + 2 → RELFG
Restore B reg
RELFG → A reg
EXIT
DPDIV
Shift A and B right 1.
Divide A and B regs by 0, X2.
A reg → VALU1
B reg → A reg
0 → B reg
Divide A and B regs by 0, X2
A reg * 2 → VALU
B reg → A reg
0 → B reg
Divide A and B regs by 0, X2
A reg *-4 → ICW
-1 → X2
VALU1 → A reg
VALU → B reg
Normalize and decrement
X2 + 1 → X2
X2 < 0?
yes
no
EXIT
A reg → VALU1
-X2 → X2
B reg → A reg
Append X2 bits from ICW to A reg
A reg → VALU
EXIT


CNVRT
No. characters → SIZFRC
0 → VALU1
0 → VALU2
0 → VALU
0 → PRECS
GEC
get next character
End of String
CNV7
Normal
Character = 0?
yes
no
10 → MULT
10 → MAXNO
8 → MULT
8 → MAXNO
MAXNO-1 → MAXNO
Character → DOT
CNV6
End of string
GEC
get character
yes
Character = .?
no
Normal
CNV1
Character > MAXNO?
yes
Set E error
CNV3
no
CNV2
VALU1 * MULT
→ VALU1
Product > 24 bits?
yes
Set T error flag
no
VALU * MULT
→ A and B regs
A + VALU1 → VALU1
Overflow?
no
yes
Set T
error flag
B reg + character
→ A reg
Bit 23 of VALU = 1?
yes
no
A Reg + MULT
→ VALU
Overflow?
yes
no
VALU1 + 1 → VALU1
Overflow ?
yes
no
Set T error flag
GEC
get character
Normal
End of
string
CNV3

PAS2

e


CNV3
0 → NDX
CNV7
DOT = . ?
no
yes
Single precision?
yes
no
Set ICW to single precision type item
Normalize VALU1 and VALU - shift count → X2
Set ICW to double precision type item
EXIT
CNV6
VALU1 = 0?
yes
no
NDX + X2 → NDX - (MINC - MINB + V + SIZFRC - 2 +NDX) * * 0777 → VALU2
X2 → MINB
-23 → MINC
SIZFRC → X2
Set ICW to floating point type item
DPDIV complete fraction
SIZFRC = 0?
no
yes
EXIT
SIZFRC > 9?
no
yes
Normalize FIVES, X2 and store in PWR. - shift count → V
VALU1 → A reg
VALU → B reg
LC(PWR) → X reg
SIZFRC - 9 → PWR
0 → X2
FIVES + 9 → A reg
0 → B reg
Normalize A and B regs
X2 - 1 → MINC
A reg → PWR + 1
VALU1 → A reg
VALU → B reg
L(PWR + 1) → X reg
DPDIV get fraction
X2 → NDX
PWR → X2

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Program to complete the second assembly pass (PAS2)

PURPOSE: To complete the second assembly pass over the intermediate output tape X1. More specifically, PAS2 is to accomplish the following items:

1. To process each line of input and detect any errors thereon.
2. To generate the machine language (binary) output represented by each line in the user's program.
3. To list the machine language code generated and the errors together with the symbolic source line.
4. To redefine symbols used as needed and to search for duplicate symbol definitions.
5. To generate literals as requested.
6. To generate items for externally defined symbols which will allow for their definition at load time.

ACTION: The main flow of PAS2 is very similar to the ASSEMBLR logic. When PAS2 is loaded, it takes the table locations generated by Pass 1 and from them sets the origin of the literal and reference tables. The cells to obtain inputs are initialized and the input tape X1 is rewound. The error flags are set to zero and the print buffer is set to blanks. The routines to perform the listing are initialized with respect to hardware device, channel, and unit. The first record of the input text is read, and control goes to the main line processing code to process the individual lines of input.

In the line processing code, a line is obtained by calling TEXT. If the line is a line of procedure sample, SAM is called and, if a DO directive is

ACTION:
(cont.)

active, DOAGN is called. PLB is called to process the label and establish a tentative definition; the operation is obtained and the proper routine is called to process the remainder of the line. Normally, control returns to the main line code where the label is now defined by inserting it into the symbol table; the line is listed, and the binary output is written on the output file. The location counter is incremented for the word generated, and control returns to the beginning of the main line code to fetch the next line of input. When all lines have been processed, FINISH is loaded by calling the tape loader.

PROGRAMMING TECHNIQUES:

PAS2 is the largest overlay in the META-SYMBOL assembly system. DTAB, as set in ENCODER, and POPs must be sufficiently large to allow PAS2 to be loaded below it. The first cells of PAS2 and ASSEMBLR are common to both routines. Many of these cells are set by ASSEMBLR (SMPWRD and UPPER for example) and used by PAS2; therefore, care must be exercised in introducing new constants or control cells to this region. FINISH, which follows PAS2, uses some of the routines in PAS2 (for example, PRNT) and must be loaded so as not to destroy the routines it uses or any of the memory cells used by them. Finally the tape loader has been assigned storage in the routines it loads to use as input buffer. None of the routines loaded by tape loader can depend on the contents of those cells assigned to tape loader for its buffer. PAS2 is a relocatable program assembled in one piece and originated at 1354_8. PAS2 depends on the POPs having been loaded by ASSEMBLR and does not contain the POP code.

CALLING SEQUENCE:

PAS2 is loaded and executed by the tape loader after completion of ASSEMBLR.

MEMORY REQUIREMENTS: Variable, but a minimum of 8192_{10} cells.

SUBROUTINES USED:

TEXT	EDIT	INTYP	RES[†]	RESET[†]
MBYT	EDTV	MFOI	FORM[†]	PAGE[†]
PLB	EDTL	RDPI	EDC[†]	EPRNT[†]
EQU	EDL	SCAN	EDS[†]	MIFT[†]
PROC	EDE	GIT[†]	GET[†]	FLM[†]
FUNC	EDR	IPL[†]	DPDIV[†]	RELTST[†]
NAME	EDF	SKIP[†]	SCANC[†]	SWITCH[†]
SAM	FLDC	INC[†]	DO[†]	SRCH[†]
POPD	PRNT	GCW[†]	DOAGN[†]	NSRT[†]
POPR	PLINE	GTB[†]	OUTP[†]	SCRP[†]
FNRL	HOME	GEC[†]	FLUSH[†]	DED[†]
PRL	TYPWRT	GNC[†]	GBSL[†]	GLOV[†]
END	TYPE	CNVRT[†]	PEEK[†]	FLN[†]
FRL	TYCC	LBTST[†]	DFLST[†]	GLOP[†]
POP	LNCT	ORG[†]	BCD[†]	PLTST[†]
EDTST	THOME	AORG[†]	TEXTR[†]	

[†] These routines are the same as those described under ASSEMBLR except that they are assembled as part of PAS2.

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

IDENTIFICATION: Program to complete the second assembly pass (PAS2)

PURPOSE: To complete the second assembly pass over the intermediate output tape X1. More specifically, PAS2 is to accomplish the following items:

1. To process each line of input and detect any errors thereon.
2. To generate the machine language (binary) output represented by each line in the user's program.
3. To list the machine language code generated and the errors together with the symbolic source line.
4. To redefine symbols used as needed and to search for duplicate symbol definitions.
5. To generate literals as requested.
6. To generate items for externally defined symbols which will allow for their definition at load time.

ACTION: The main flow of PAS2 is very similar to the ASSEMBLR logic. When PAS2 is loaded, it takes the table locations generated by Pass 1 and from them sets the origin of the literal and reference tables. The cells to obtain inputs are initialized and the input tape X1 is rewound. The error flags are set to zero and the print buffer is set to blanks. The routines to perform the listing are initialized with respect to hardware device, channel, and unit. The first record of the input text is read, and control goes to the main line processing code to process the individual lines of input.

In the line processing code, a line is obtained by calling TEXT. If the line is a line of procedure sample, SAM is called and, if a DO directive is

ACTION (cont.) active, DOAGN is called. PLB is called to process the label and establish a tentative definition; the operation is obtained and the proper routine is called to process the remainder of the line. Normally, control returns to the main line code where the label is now defined by inserting it into the symbol table; the line is listed, and the binary output is written on the output file. The location counter is incremented for the word generated, and control returns to the beginning of the main line code to fetch the next line of input. When all lines have been processed, FINISH is loaded by calling the tape loader.

PROGRAMMING TECHNIQUES: PAS2 is the largest overlay in the META-SYMBOL assembly system. DTAB, as set in ENCODER, must be sufficiently large to allow PAS2 to be loaded below it.

CALLING SEQUENCE: PAS2 is loaded and executed by the tape loader after completion of ASSEMBLR.

MEMORY REQUIREMENTS: Variable, but a minimum of 8192_{10} cells.

SUBROUTINES USED:

TEXT	END	PRNT	GNC†	DO†	MIFT†
MBYT	FRL	HOME	CNVRT†	DOAGN†	FLM†
PLB	POP	MFOI	LBTST†	OUTP†	RELTST†
EQU	EDTST	RDPI	ORG†	FLUSH†	SWITCH†
PROC	EDIT	SCAN	AORG†	GBSL†	SRCH†
FUNC	EDTV	GIT	RES†	PEEK†	NSRT†
NAME	EDTL	IPL†	FORM†	DFLST†	SCRP†
SAM	EDL	SKIP†	EDC†	BCD†	DED†
POPD	EDE	INC†	EDS†	TEXTR†	GLOV†
POPR	EDR	GCW†	GET†	RESET†	FLN†
FNRL	EDF	GTB†	DPDIV†	PAGE†	GLOP†
PRL	FLDC	GEC†	SCANC†	EPRNT†	PLTST†

† These routines are the same as those described under ASSEMBLR except that they are assembled as part of PAS2.

ENTRY POINTS TO PAS2 (ASSEMBLY PASS 2) SUBROUTINES

Entry	Page Description	Page Flowchart	Entry	Page Description	Page Flowchart
AORG	3-188	3-353	DO1ZZ	3-193	3-356
BCD	3-202	3-364	DO2	3-193	3-356
CNV1	3-226	3-399	DO3	3-193	3-356
CNV2	3-226	3-399	DOA2	3-194	3-357
CNV3	3-226	3-400	DOA3	3-194	3-357
CNV6	3-226	3-400	DOA4	3-194	3-357
CNV7	3-226	3-400	DOA5	3-193	3-356
CNVRT	3-226	3-399	DOAGN	3-194	3-357
COAD	3-337	3-387	DODEC		3-358
COAD2	3-337	3-387	DOEND	3-193	3-356
COAD3	3-337	3-387	DOERR	3-193	3-356
COAP	3-337	3-388	DOVFW	3-193	3-356
COAS	3-337	3-387	DPDIV	3-227	3-398
COAS1	3-337	3-387	ED	3-318	3-370
COAS3	3-337	3-387	EDC	3-213	3-373
COBS	3-337	3-389	EDE	3-321	3-371
CODS	3-337	3-388	EDF	3-323	3-372
COEQ	3-337	3-385	EDIT	3-318	3-370
COGT	3-337	3-385	EDITP	3-318	3-370
COIQ	3-337	3-388	EDL	3-320	3-372
COLD	3-337	3-386	EDR	3-322	3-372
COLP	3-337	3-386	EDS	3-214	3-373
COLS	3-337	3-386	EDTL	3-320	3-371
COLS1	3-337	3-386	EDTST	3-317	3-366
COLS2	3-337	3-386	EDTV	3-319	3-371
COLS3	3-337	3-386	END	3-313	3-361
COLS4	3-337	3-386	ENDF	3-313	3-361
COLS6	3-337	3-386	ENDM		3-375
COLS6A	3-337	3-386	ENDN		3-375
COLSZ	3-337	3-386	ENDS	3-313	3-361
COLT	3-337	3-385	EPRNT	3-219	3-379
COLT1	3-337	3-385	EQU	3-308	3-353
COLT2	3-337	3-385	EQU3	3-308	3-353
COLT3	3-337	3-385	EQU4	3-308	3-353
COXQ	3-337	3-388	EQU6	3-308	3-353
COXQ1	3-337	3-388	EQU7	3-308	3-353
DATAT		3-375	FINISH	3-341	3-401
DED	3-220	3-380	FLDC	3-324	3-373
DEF		3-375	FLM	3-224	3-382
DFLST	3-198	3-360	FLN	3-223	3-382
DO	3-193	3-356	FLUSH	3-216	3-376
DO1	3-193	3-356	FLUSH1	3-216	3-371

ENTRY POINTS TO PAS2 (ASSEMBLY PASS 2) SUBROUTINES (cont.)

Entry	Page Description	Page Flowchart	Entry	Page Description	Page Flowchart
FNRL	3-311	3-359	GIT44	3-339	3-392
FNRL1	3-311	3-359	GIT99	3-339	3-392
FNRL2	3-311	3-359	GIT351	3-339	3-392
FORM	3-189	3-354	GIT352	3-339	3-392
FRERR	3-315	3-363	GITA	3-339	3-394
FRL	3-315	3-362	GITA2	3-339	3-394
FRL4	3-315	3-362	GITC	3-339	3-394
FRL4A	3-315	3-363	GITE	3-339	3-394
FRL4B	3-315	3-362	GITL	3-339	3-394
FRL4C	3-315	3-362	GITS1	3-339	3-391
FRL4E	3-315	3-362	GITS2	3-339	3-391
FRL5	3-315	3-362	GITS3	3-339	3-391
FRL5A	3-315	3-362	GITS4	3-339	3-390
FRL5B	3-315	3-362	GITS5	3-339	3-391
FRL6	3-315	3-363	GITS8	3-339	3-390
FRL8	3-315	3-363	GITS9	3-339	3-391
FRND	3-315	3-362	GITX	3-339	3-394
FUNC	3-309	3-355	GLOP	3-239	3-381
GBSL	3-237	3-397	GLOV	3-221	3-381
GVSL2	3-239	3-397	GNC	3-235	3-396
GCW	3-181	3-351	GNC3	3-235	3-396
GEC	3-183	3-352	GNCE	3-235	3-396
GET	3-236	3-397	GNCER	3-235	3-396
GET1	3-236	3-397	GO1	3-198	3-360
GET4	3-236	3-397	GTB	3-182	3-352
GET6	3-236	3-397	GTB1	3-182	3-352
GIT	3-339	3-390	GTLBL	3-208	3-402
GIT1	3-339	3-390	GTRBL		3-390
GIT2	3-339	3-394	HOME	3-328	3-374
GIT3	3-339	3-390	INC	3-180	3-351
GIT9	3-339	3-392	INTYP	3-334	3-378
GIT11	3-339	3-390	IPL	3-177	3-349
GIT31	3-339	3-393	LBERR	3-307	3-348
GIT32	3-339	3-393	LBTST	3-184	3-351
GIT33	3-339	3-392	LINE		3-346
GIT34	3-339	3-393	LINSYM		3-347
GIT35	3-339	3-391	LN1		3-346
GIT35A	3-339	3-391	LN1A		3-346
GIT37	3-339	3-393	LN4		3-346
GIT41	3-339	3-390	LNCT	3-332	3-378
GIT42	3-339	3-393	LNDPV		3-347
GIT43	3-339	3-392	LNE		3-347

ENTRY POINTS TO PAS2 (ASSEMBLY PASS 2) SUBROUTINES (cont.)

Entry	Page Description	Page Flowchart	Entry	Page Description	Page Flowchart
LNEN		3-347	PRL1	3-311	3-359
LNEND		3-347	PRL2A	3-311	3-360
LNERR		3-347	PRL3	3-311	3-359
LNFRM		3-347	PRL7	3-311	3-360
LNLOC		3-347	PRNT	3-325/3-326	3-374
LNVAL		3-347	PROC	3-3-9	3-355
M3WAI		3-381	RDIP	3-336	3-381
MBYT	3-306	3-350	RELTST	3-225	3-398
MFOI	3-335	3-380	RES	3-188	3-354
MIFT	3-238	3-382	RESET	3-217	3-376
NAME	3-309	3-355	RET3A	3-306	3-350
NOEDT	3-317	3-366	RET5	3-306	3-350
NOEND		3-347	RET10	3-306	3-350
NRST	3-210	3-368	REZZ	3-306	3-350
NS1A	3-210	3-368	SA2	3-309	3-355
NS1B	3-210	3-368	SAM	3-309	3-355
NS1C	3-210	3-368	SC2	3-212	3-369
NS1D	3-210	3-368	SC3	3-212	3-369
NS3	3-210	3-368	SCAN	3-337	3-383
NS3A	3-210	3-368	SCAN1	3-337	3-383
NS9	3-210	3-368	SCAN2	3-337	3-383
NS99	3-210	3-368	SCAN21	3-337	3-383
ORG	3-188	3-353	SCAN23	3-337	3-383
ORG1	3-188	3-353	SCAN3	3-337	3-383
OUTP	3-215	3-375	SCAN6	3-337	3-385
OUTP1	3-215	3-375	SCAN7	3-337	3-383
PAGE	3-218	3-379	SCAN9	3-337	3-384
PEEK	3-234	3-396	SCAN9E	3-337	3-384
PL1	3-307	3-348	SCAN98	3-337	3-384
PLB	3-307	3-348	SCAN99	3-337	3-384
PLB2	3-307	3-348	SCANC	3-232	3-395
PLB3	3-307	3-348	SCANC1	3-232	3-395
PLBEX	3-307	3-348	SCANC2	3-232	3-395
PLINE	3-327	3-374	SCANC3	3-232	3-395
POP	3-316	3-365	SCANC6	3-232	3-395
POP1	3-316	3-365	SCANC8	3-232	3-395
POP2	3-316	3-365	SCANC9	3-232	3-395
POP3	3-316	3-365	SCANF	3-337	3-384
POP4	3-316	3-365	SCANK	3-337	3-384
POPD	3-310	3-355	SCANL	3-337	3-384
POPR	3-310	3-355	SCANR	3-232	3-395
PRL	3-311	3-359	SCN998	3-337	3-384

ENTRY POINTS TO PAS2 (ASSEMBLY PASS 2) SUBROUTINES (cont.)

	Page			Page	
Entry	Description	Flowchart	Entry	Description	Flowchart
SCNC11	3-232	3-395	TEXT3	3-202	3-364
SCRP	3-212	3-369	TEXTR	3-202	3-364
SKIP	3-179	3-351	THOME	3-333	3-378
SR5	3-209	3-367	TXT2	3-305	3-349
SR6	3-209	3-367	TXT3	3-305	3-349
SR7	3-209	3-367	TXT5	3-305	3-349
SR9	3-209	3-367	TYCC	3-331	3-378
SRCH	3-209	3-367	TYPE	3-330	3-377
SWITCH	3-207	3-366	TYPWRT	3-329	3-377
TEXT	3-305	3-349	UNDEF		3-347
TEXT1	3-305	3-349	WEOFL	3-343	3-402
TEXT2	3-202	3-364			

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 0420
Catalog No. 9300: 6120

IDENTIFICATION: Obtain next line of text (TEXT)

PURPOSE: To obtain the next line of input to be processed.

ACTION: TEXT takes the following actions:

1. If the line is to be obtained from the procedure sample area, TEXT calls SKIP to skip to the end of the current line.
2. If the line is to be listed, it is reconstructed by calling MBYT. The line is not output on the symbolic output file.
3. If the line is not to be listed, TEXT obtains the bytes by calling GTB and stores them in BBUF.

PROGRAMMING TECHNIQUES: TEXT is a relocatable routine assembled as part of PAS2.

CALLING SEQUENCE:

```
BRM    TEXT
end-of-file return
normal return
```

MEMORY REQUIREMENTS: 70_8 cells

SUBROUTINES USED:

IPL	SKIP
EDS	GTB
EDC	MBYT

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Reconstruct symbolic lines (MBYT)

PURPOSE: To reconstruct line images for printing and to enter bytes into byte buffer, BBUF.

ACTION: MBYT obtains bytes by calling GTB. The byte is stored in BBUF, and the byte table entry is obtained and placed in ECW. The dictionary characters represented by the byte are obtained by calling GEC and are stored into the image by calling EDC. The first portion of continued lines is listed. INC is used to obtain comment characters.

PROGRAMMING TECHNIQUES: MBYT is a relocatable routine assembled as part of PAS2.

CALLING SEQUENCE: BRM MBYT

MEMORY REQUIREMENTS: 102_8 cells

SUBROUTINES USED:

GTB	GEC
IPL	EDC
INC	GBSL

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042(
Catalog No. 9300: 612(

IDENTIFICATION: Process label fields (PLB)

PURPOSE: To scan the label field of a line, set a tentative definition of the label (if it is present), and set the cell WLLVL to indicate the procedure level at which the label is to be defined.

ACTION: WLLVL calls GCW to obtain the bytes of the label field and the blank following the label. If the line is a comment, PLB exits with an end-of-line flag in the A register. WLLVL is set to reflect the level at which the label is to be defined. A tentative definition is made for the label, setting it equal to the location counter value; this tentative definition in the form of an address item is placed in LBL through LBL+3.

PROGRAMMING TECHNIQUES: PLB is a relocatable routine assembled as part of PAS2.

CALLING TECHNIQUE:

```
BRM   PLB
end-of-line return
normal return
```

MEMORY REQUIREMENTS: 134_8 cells

SUBROUTINES USED:

GCW	GBSL
GEC	PLTST

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 900 Series: 042016
9300: 612001

IDENTIFICATION: Process EQU directives (EQU)

PURPOSE: To process the EQU directive.

ACTION: The operand field of the line is evaluated by calling SCAN. The value returned by SCAN is used to construct an item definition in LBL to LBL+3. If the operation is a reference, LBL is set to zero and return is made to LINSYM. In constructing the item definition, EQU uses the associate set for the tentative definition of the symbol by PLB and the type and mode bits of the operand field. NSRT is called to define the item. When an undefined value appears in the operand field, the U error flag is set, the * flag is reset, a zero value is assumed, and control returns to the main line code at LNVAL.

PROGRAMMING TECHNIQUES: EQU is an open subroutine assembled as part of ASSEMBLR.

CALLING SEQUENCE: EQU is assembled as part of PAS2 and is called by executing the directive branch table. Return is to the main line code.

MEMORY REQUIREMENTS: 107_8 cells

SUBROUTINES USED:

SCAN MFOI
NSRT RDPI

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 04201
Catalog No. 9300: 61200

IDENTIFICATION: Process lines of procedure sample (PROC, FUNC, NAME and SAM)

PURPOSE: To skip sample lines and at the same time keep sufficient track of the sample nesting to determine when the end of the sample is reached.

ACTION: PROC and FUNC set the sample processing flag, increment the nested sample count, and go to SA2. NAME is synonymous with SA2. SAM calls PLB to process the label and then tests the operation field for a directive that is a NAME, PROC, FUNC or END. If the operation field contains one of these, SAM executes the proper routine by using the directive branch table; otherwise, control goes to SA2 where the label flag (LBL) is reset and control is returned to the main line routine at LINSYM.

PROGRAMMING TECHNIQUES: All these routines are open routines assembled as part of PAS2.

CALLING SEQUENCE: PROC, FUNC, and NAME are called by using the directive branch table. SAM is called by the main line code when the sample processing flag is ON.

MEMORY REQUIREMENTS: 52_8 words total

SUBROUTINES USED:
PLB
GCW
GBSL

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION:	Process undefined mnemonics and POPD directives (POPD and POPR)
PURPOSE:	To cause the lines with undefined mnemonics or POPD directives to be ignored.
ACTION:	POPR defines any waiting label, increments CCINC, and goes to POPD where LBL is reset before returning to the main line code at LNLOC.
PROGRAMMING TECHNIQUES:	POPD and POPR are open routines assembled as part of PAS2.
CALLING SEQUENCES:	POPD is called by using the directive branch table. POPR is called by the line code when an undefined operation is encountered.
MEMORY REQUIREMENTS:	5 cells total
SUBROUTINES USED:	LBTST

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 04201
Catalog No. 9300: 61200

IDENTIFICATION: Process PROC and FUNC reference lines (FNRL and PRL)

PURPOSE: To process the line referencing a PROC or FUNC.

ACTION: The procedure level is tested to determine if space exists to process the line; if it does not, the routine is exited. The temporary procedure level (PLVT) is incremented, a flag is set to indicate whether the reference was to a PROC or FUNC, WLLVL is set equal to PLVT, and the symbol table direction is reversed. PLV and the location counter are saved, and the pass is set to first. DFLST is called to define the parameter list elements. PLV is set to PLVT; BYT, ECW, and TERM are saved. The starting location of the switch is called to reset the origin of the next byte of input. The old input position is saved for resuming later. PLB is called to obtain the PROC or FUNC line label, and a test is made to determine whether the PROC is a 1-pass or a 2-pass PROC. If it is a 1-pass PROC, the PASS for this level is set equal to the PASS at the next lower procedure level. The list item is constructed using the element linkage established by DFLST, the list identification is obtained from the PROC label by PLB, and the value is associated with the NAME item. NSRT is called to place the list item into the symbol table. SKIP is called to bypass the remainder of the PROC line.

PROGRAMMING TECHNIQUES: The temporary setting of the procedure level PLVT before defining the list parameters is done so that the parameters will be inserted into the correct table position. Since a FUNC reference is possible before finishing the definition of the list, the PLV flag must remain unaltered so that characters are obtained and labels processed, etc., in the normal manner: however, it must

PROGRAMMING TECHNIQUES: (cont.)

be remembered that this additional reference must be completed. These routines are open routines assembled as part of PAS2.

CALLING SEQUENCE:

PRL is called by the main line code when a procedure reference is encountered.

FNRL is called SCANC when a function reference is encountered. Both return to the main line code.

MEMORY REQUIREMENTS:

225_8 cells total

SUBROUTINES USED:

DFLST	PLB
SWITCH	GCW
GBSL	SKIP
NSRT	

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 04201
Catalog No. 9300: 61200

IDENTIFICATION: Process END directives (END)

PURPOSE: To process END directives.

ACTION: There are four separate cases involved in processing END lines:

1. The END of the program. Control goes to tape loader to load FINISH.
2. The END of user sample. The sample level count is decremented and, if zero, the sample processing flag is reset. Control goes to LINSYM in the main line code.
3. The END of a PROC reference. If this is the first pass of a 2-pass PROC, the PASS is set to second; SWITCH is called to reset the line origin to the first line of the PROC; the location counter is reset; error flags for * and U errors are cleared; and GCW is called to get the first byte of the PROC line from sample. Control then goes to the start of the main line code. If this is the second pass of the PROC, any waiting label is defined by calling NSRT; SWITCH is called to reset the line origin to the point at which the PROC was entered; SCRP is called to purge local symbols from the table; the external parameters are restored; PLV and PLVT are decremented; and control is returned to the end of the main line code.
4. The END of a function reference. SCAN is called to define the operand field of the END line. SWITCH is called to reset the line origin to the point of entry, SCRP is called to purge the symbol table, the parameters are reset, PLV and PLVT are decremented, and control goes to SCANR in the SCANC routine.

PROGRAMMING TECHNIQUES: END is an open routine assembled as part of PAS2.

CALLING SEQUENCE: END is called by executing the directive branch table.

MEMORY REQUIREMENTS: 117_8 cells

SUBROUTINES USED:

SWITCH	SCAN
SCRP	NSRT
GCW	

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 04201
Catalog No. 9300: 6120(

IDENTIFICATION: Process FORM reference lines (FRL)

PURPOSE: To process FORM reference lines.

ACTION: The FORM control word is obtained and saved. CCINC is set to the number of words generated by the FORM. The form control word is normalized to determine the number of bits to be generated; this number is set in BITSS. The normalized FORM goes to WRD2 and WRD2+1. SCAN is called to evaluate the expressions in the operand field and FLDC to determine the field size for each expression. The data are positioned and stored into WORD and WORD+1. If an expression is relocatable, WMODR is set. If an expression is a reference, the value is taken from the location indicated as the value and the value of the location counter is placed into the location indicated as the value. In this way the references are linked for the loader.

PROGRAMMING TECHNIQUES: FRL is an open routine assembled as part of PAS2.

CALLING SEQUENCE: FRL is entered when the line code encounters a FORM reference. Return is to the line code at LNFRM.

MEMORY REQUIREMENTS: 307_8 cells

SUBROUTINES USED:

SCAN MFOI
FLDC GLOV

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Process programmed operator references (POP)

ACTION: CCINC is incremented, and the programmed operator item is obtained. The operation code from the programmed operator is set to WORD. If the programmed operator is an external reference, type IERR is set. SCAN is called to obtain the address and index fields which are inserted into WORD. If the address is a reference, the contents of the cell addressed by VALU is used as the value and the location counter is stored in the cell addressed by VALU. WMODP is set (as is WMODR, if needed).

PROGRAMMING TECHNIQUES: POP is an open routine assembled as part of PAS2.

CALLING SEQUENCE: POP is called by the main line code when a programmed operator item is encountered. POP returns to the line code at LNFRM.

MEMORY REQUIREMENTS: 130_8 cells

SUBROUTINES USED: SCAN GLOV

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 900 Series: 04201
9300: 61200

IDENTIFICATION: Test to list line (EDTST)

PURPOSE: To determine if the current line should be listed.

ACTION: Lines are listed only if:

1. Listing is requested, and
2. The pass at the current level is the second pass, and
3. A procedure or function reference is not being processed, or
4. A procedure or function reference is being processed and data have been generated at this point for output.

PROGRAMMING TECHNIQUES: EDTST is a relocatable routine assembled as part of PAS2.

CALLING SEQUENCE:

Data generated flag to B register
BRM EDTST
Listing-to-be-done return
Do-not-list return

MEMORY REQUIREMENTS: 21_8 cells

SUBROUTINES USED: None.

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Edit full lines for listing (EDIT)

PURPOSE: To format a line for listing and cause it to be listed and to cause the data generated to be output.

ACTION: EDTST is called to determine whether listing is to be done. EDE, EDL, and EDR are called to format the error flags, location, and data, respectively. PRNT is called to output the line to the listing. The binary data are output by calling OUTP.

PROGRAMMING TECHNIQUES: EDIT assumes that a FORM control word for formatting the data has been placed in WRD2 and WRD2+1, that the datum is in WORD and WORD+1, and that double-precision flag (DPPF) is negative if the datum is double-precision. EDIT is a relocatable routine assembled as part of PAS2.

CALLING SEQUENCE:

```
Control words set as noted
BRM    EDIT
```

MEMORY REQUIREMENTS: 43_8 cells

SUBROUTINES USED:

EDTST	EDS
EDE	PRNT
EDL	OUTP
EDR	

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION:	Edit value fields (EDTV)
PURPOSE:	To format the value field of a line and cause the line to be listed.
ACTION:	EDTV calls EDE to format the error flags; EDF is called to format the value; and PRNT is called to output the line to the listing.
PROGRAMMING TECHNIQUES:	EDTV assumes that the datum to be output is in WORD and WORD+1 and that DPPF is negative if the datum is double-precision. EDTV is a relocatable routine assembled as part of PAS2.
CALLING SEQUENCE:	Control words set as indicated above BRM EDTV
MEMORY REQUIREMENTS:	22_8 cells
SUBROUTINES USED:	EDE EDF EDS PRNT

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Edit locations (EDTL and EDL)

PURPOSE: To format the location field of the listing. EDTL also formats the errors and causes the line to be listed.

ACTION: EDTL calls EDE to format the error flags, EDL to format the location, and PRNT to output the line. EDL calls EDS to initialize the buffer position to store the location and EDF to place the location characters in the buffer.

PROGRAMMING TECHNIQUES: Both routines are relocatable routines assembled as part of PAS2.

CALLING SEQUENCE:

```
BRM    EDTL
   or
BRM    EDL
```

MEMORY REQUIREMENTS: 16_8 cells total

SUBROUTINES USED:

by EDTL: EDE
EDL
PRNT

by EDL: EDS
EDF

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Format error flags (EDE)

PURPOSE: To format the error flags for the listing and to set QPESW.

ACTION: QPESW is incremented if any error flags other than I or * have been set. The error flags are tested, and for each one set the equivalent letter code is placed in the listing by calling EDC. The flags are reset when found set.

PROGRAMMING TECHNIQUES: EDE is a relocatable routine assembled as part of PAS2.

CALLING SEQUENCE: BRM EDE

MEMORY REQUIREMENTS: 23_8 cells

SUBROUTINES USED: EDS
EDC

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 900 Series: 042016
9300: 612001

IDENTIFICATION: Format data fields (EDR)

PURPOSE: To format the data field for the listing under the control of a FORM control word.

ACTION: EDR normalizes the FORM control word and determines the number of bits of data. The datum in WRD1 and WRD 1+1 is positioned, and FLDC is called to determine the field size. The proper number of bits of data are loaded into the B register and low order character of the A register. EDF is called to insert the field into the listing buffer. EDC is called to insert a blank character between each field processed.

PROGRAMMING TECHNIQUES: The FORM control word is assumed to be in WRD2 and WRD2+1. The datum is assumed to be in WRD1 and WRD1+1. EDR is a relocatable routine assembled as part of PAS2.

CALLING SEQUENCE: Data and form control word as indicated

```
BRM    EDR
```

MEMORY REQUIREMENTS: 107_8 cells

SUBROUTINES USED:
FLDC
EDF
EDC

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 900 Series: 04201(
9300: 61200

IDENTIFICATION: Insert data fields into listing buffer (EDF)

PURPOSE: To insert the data contained in the A and B registers into the listing buffer.

ACTION: EDF calls EDC to store in the A register the individual characters which are shifted from the B register until all characters are stored as determined by CNTR.

PROGRAMMING TECHNIQUES: EDF is a relocatable routine assembled as part of PAS2.

CALLING SEQUENCE:

Character count to CNTR
First character to A register
Remainder of field left-adjusted in B register
BRM EDF

MEMORY REQUIREMENTS: 12_8 cells

SUBROUTINES USED: EDC

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 900 Series: 042016
9300: 612001

IDENTIFICATION:	Determine field sizes of a FORM word (FLDC)
PURPOSE:	To determine the size of a field in a FORM control word.
ACTION:	FLDC removes the sign bit of a FORM control word in WRD2 and WRD2+1 and normalizes the result to determine the field size. BITSS contains the number of bits remaining in the control word and is decremented by the size of this field. The result is in the A register.
PROGRAMMING TECHNIQUES:	FLDC is a relocatable subroutine assembled as part of PAS2.
CALLING SEQUENCE:	FORM control word to WRD2 and WRD2+1 FORM length to BITSS BRM FLDC end-of-FORM return normal return
MEMORY REQUIREMENTS:	32_8 cells
SUBROUTINES USED:	None

SDS SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: List one line of output (PRNT)

PURPOSE: To list one line.

ACTION: PRNT calls the listing output routine to write the line. The left portion (nine words) of the listing buffer are cleared to blanks, and LC is set to print data only.

PROGRAMMING TECHNIQUES: The I/O routine called is set by the initialization code for PAS2. PRNT is a relocatable routine assembled as part of PAS2.

CALLING SEQUENCE: BRM PRNT

MEMORY REQUIREMENTS: 14_8 cells

SUBROUTINES USED: List output routine, normally PLINE

SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 612001

IDENTIFICATION: List one line of output (PRNT)

PURPOSE: To list one line.

ACTION: PRNT calls the listing output routine to write the line. The left portion (nine words) of the listing buffer are cleared to blanks, and LC is set to print data only.

PROGRAMMING TECHNIQUES: The I/O routine called is set by the initialization code for PAS2. PRNT is a relocatable routine assembled as part of PAS2.

CALLING SEQUENCE: BRM PRNT

MEMORY REQUIREMENTS: 14_8 cells

SUBROUTINES USED: List output routine

SDS SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Write listing on the printer (PLINE)

PURPOSE: To write the line of listing output to the on-line printer.

ACTION: PLINE executes a MIW loop to output the required number of words to the printer.

PROGRAMMING TECHNIQUES: PLINE is initialized as to channel and unit assignments by the initialization code for PAS2. If a buffer error or print fault occurs, PLINE halts. Stepping causes processing to resume. PLINE is a relocatable routine assembled as part of PAS2.

CALLING SEQUENCE:
Word count to B register
Buffer location to A register
BRM PLINE

MEMORY REQUIREMENTS: 26_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 900 Series: 042016
9300: 612001

IDENTIFICATION: Home paper on the printer (HOME)

PURPOSE: To space to the top of the next page on the on-line printer or call the proper routine if the listing is other than on the printer.

ACTION: If the listing is on the on-line printer HOME ejects the page by skipping to the proper channel.

PROGRAMMING TECHNIQUES: HOME is initialized by the initialization code of PAS2 as to unit and channel assignments if the printer is to be used. If not, a branch instruction is inserted in HOME to cause control to go to the proper routine for homing the page. HOME is a relocatable routine assembled as part of PAS2.

CALLING SEQUENCE: BRM HOME

MEMORY REQUIREMENTS: 10_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Write a line of listing on the typewriter (TYPWRT)

PURPOSE: To output lines of listing on the on-line typewriter.

ACTION: TYPWRT determines the number of characters to output, returns the carriage by calling TYCC, and tabs to the correct starting point by again calling TYCC. Characters are output by calling TYPE. If a line is longer than 72_{10} characters, it is output in two lines. LNCT is called to maintain a line count.

PROGRAMMING TECHNIQUES: TYPWRT is initialized by INTYP, which sets the control linkage to call TYPWRT when typed listing is indicated. TYPWRT is a relocatable routine assembled as part of PAS2.

CALLING SEQUENCE: BRM TYPWRT

MEMORY REQUIREMENTS: 47_8 cells

SUBROUTINES USED:
LNCT
TYPE
TYCC

SDS SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Type a specified number of words (TYPE)

PURPOSE: To output to the on-line typewriter the number of words indicated in the index register from the location specified in the A register.

ACTION: TYPE outputs to the typewriter from the location specified by the A register the number of words indicated by the index register (count is in negative form). Blanks are converted to 12_8, and a MIW loop is used to output the words.

PROGRAMMING TECHNIQUES: TYPE is initialized by INTYP as to unit and channel assignments. TYPE is a relocatable routine assembled as part of PAS2.

CALLING SEQUENCE:

Buffer location to A register
Negative word count to index register
BRM TYPE

MEMORY REQUIREMENTS: 26_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Output one character to the typewriter (TYCC)

PURPOSE: To output the high-order character of the A register to the typewriter.

ACTION: TYCC writes the high-order character in the A register on the typewriter.

PROGRAMMING TECHNIQUES: TYCC is initialized as to unit and channel by INTYP. TYCC is a relocatable routine assembled as part of PAS2.

CALLING SEQUENCE: Character to A register
BRM TYCC

MEMORY REQUIREMENTS: 14_8 cells

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Keep listing line counts (LNCT)

PURPOSE: To count lines output on the typewriter and call THOME when 50 lines have been typed.

ACTION: LNCT increments the line count and, if it is greater than 50, calls THOME.

PROGRAMMING TECHNIQUES: LNCT is a relocatable routine assembled as part of PAS2.

CALLING SEQUENCE: BRM LNCT

MEMORY REQUIREMENTS: 7 cells

SUBROUTINES USED: THOME

SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Home paper on typewriter (THOME)

PURPOSE: To space paper on the typewriter

ACTION: THOME spaces the typewriter listing 66 - CTR lines by calling TYCC with a carriage return character.

PROGRAMMING TECHNIQUES: THOME is a relocatable routine assembled as part of PAS2.

CALLING SEQUENCE: BRM THOME

MEMORY REQUIREMENTS: 12_8 cells

SUBROUTINES USED: TYCC

SCIENTIFIC DATA SYSTEMS

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No. 042016

IDENTIFICATION: Initialize the typewriter routines (INTYP)

PURPOSE: To set the linkage to use the typewriter routines for listing output and to initialize the typewriter routines as to channel and unit assignments.

ACTION: INTYP sets the location of TYPWRT into PRINT and the branch to HMTW into the HOME routine. The unit and channel assignments for listing are obtained and the I/O instructions in the various typewriter routines set.

PROGRAMMING TECHNIQUES: INTYP is a relocatable routine assembled as part of PAS2.

CALLING SEQUENCE: BRM INTYP

MEMORY REQUIREMENTS: Nil. INTYP resides in an output buffer.

SUBROUTINES USED: None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 04201
Catalog No. 9300: 61200

IDENTIFICATION: Make 2-word floating-point values (MFOI)

PURPOSE: To convert the 3-word internal floating-point items into items with two value words.

ACTION: MFOI rounds the floating-point value to 37_{10} fractional bits. If overflow occurs, FLN is called to rescale the result. The exponent is moved into the low-order bits of the low-order data word.

PROGRAMMING TECHNIQUES: MFOI is a relocatable routine assembled as part of PAS2.

CALLING SEQUENCE: Floating-point item to VALU through VALU+2
BRM MFOI

MEMORY REQUIREMENTS: 25_8 words

SUBROUTINES USED: FLN

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION:	Reverse double-precision data words (RDPI)
PURPOSE:	To reverse double-precision values for output.
ACTION:	The data words in WORD and WORD+1 are reversed.
PROGRAMMING TECHNIQUES:	This routine must be 'NOPed' for 9300 format outputs.
CALLING SEQUENCE:	Double precision value to WORD and WORD+1 BRM RDPI
MEMORY REQUIREMENTS:	6 cells
SUBROUTINES USED:	None

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 042016
Catalog No. 9300: 612001

IDENTIFICATION: Evaluate expressions (SCAN)

PURPOSE: To evaluate an expression and leave the control word of the results in the B register and ICW and the value in VALU through VALU+2 with the low order portion of the value in the A register.

ACTION: SCAN obtains the items in the expression by calling GIT and the connectors by calling GNC. The items and connectors are obtained in pairs. If the connector obtained is of higher priority than the previous connector, the item value and the connector are saved in the SCAN operations table and the table pointers are incremented. If the connector is of lower priority, the previous operation is performed. The type of operation to be performed is determined by executing an operations branch table which carries control to the various operation routines.

The operation routines perform the indicated operation between a pair of operands one of which is located in the SCAN operations table and the other of which is located in ICW and VALU to VALU+2. The first item is always the one in the SCAN operations table. The result of the operation is placed in the cells ICW and VALU to VALU+2, and the pointers to the operations branch table are decremented to point to the previous item.

When a leading = (equals) mark is encountered, SCAN searches the literal table to find the literal location. If the literal is not in the table, it is inserted. The value of a literal is the location of the literal in the object program. A leading * (asterisk) mark causes a flag to be set which will result in the value of the expression being interpreted as an address quantity.

ACTION: (cont.)

This * flag will also be output with the resulting value so that expressions of the format P (*i) may be properly interpreted.

When the last operation to be performed is a terminator, SCAN tests for the literal flag being set; if it is, SCAN takes zero as the value of the expression. If the * flag is ON, the value is converted to a 3-word address value and the sign bit of VALU is set.

Upon exit, the contents of TERM are

0 if blank terminated

1 if comma terminated

2 if right parenthesis terminated

The cell STAR contains 1 if the expression had a leading * and 0 otherwise.

PROGRAMMING TECHNIQUES:

The SCAN operations table is really a series of short tables each of which is indirectly addressed. The table positions are incremented or decremented by incrementing or decrementing the indirect point words. SCAN is a relocatable routine assembled as part of PAS2.

CALLING SEQUENCE:

Byte table entry for the first byte of the expression ECW
BRM SCAN

MEMORY REQUIREMENTS:

1266_8 cells

SUBROUTINES USED:

GCW	MIFT	RELTST
GIT	GLOV	FLM
GNC	GLOP	FLN

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 0420
Catalog No. 9300: 6120(

IDENTIFICATION: Get next item of an expression (GIT)

PURPOSE: To obtain the value of an item and store it in VALU through VALU+2 with its control word in ICW.

ACTION: GIT evaluates the following types of items:

alphanumeric constants
location counter reference
function references
subscripted symbols (parameters)
symbolic items
numeric items
lists
list count
parenthetical expressions

1. Alphanumeric constants are evaluated by obtaining the characters from the dictionary which comprise the constant and packing them together into VALU and VALU+1.

2. The value of location counter reference is the current value of CC.

3. Function references are evaluated by calling SCANC (which in turn calls FNRL).

4. Subscripted symbols are evaluated by calling SCANC to obtain the subscripts and by stepping through the list to extract the proper element.

5. Symbolic items are obtained by picking the item out of the symbol table. When an undefined symbol is encountered, the reference table is searched for the symbol. If the symbol is not in the table, a reference

ACTION: (cont.)

item with zero value is inserted into the table. The value of the reference is taken as the location of the reference value in the reference table.

6. Numeric items are evaluated by calling CNVRT. If a numeric item is a mixed floating point number, the integer and fractional parts are obtained by separate calls on CNVRT and the parts are then combined by GIT.

7. Lists are obtained by inserting the elements of the list into the symbol table by calling SCANC and generating a list item giving the location of the first element and the number of elements.

8. List counts are evaluated by finding the appropriate list item and extracting the element count from it.

9. Parenthetical expressions are obtained by calling SCANC. GIT does not differentiate between lists and parenthetical expressions; the distinction is made by SCANC.

PROGRAMMING TECHNIQUES:

GIT works with the SCAN and SCANC routines and is really a major section of the overall expression evaluation processing. GIT is a relocatable routine assembled as part of PAS2.

CALLING SEQUENCE:

Byte table entry for first byte to ECW
BRM GIT

MEMORY REQUIREMENTS:

472_8 words

SUBROUTINES USED:

GCW	SCANC
GLOV	MIFT
CNVRT	GBSL
PEEK	GET

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

IDENTIFICATION: Program to output literals and references (FNSH or FINISH)

PURPOSE: To output the literals, references, and END records to the binary and listing outputs.

ACTION: The system tape is rewound, the transfer value is obtained by calling SCAN (left in core from PAS2), and the END line is listed by calling EPRNT if no transfer address and EDTV if there is a transfer address. The literals are taken from the literal table and output to the listing and binary files by calling EDIT. When the literals are completed, the references are obtained and output. GTLB is used to reconstruct the symbols, and EDTL is called to list them. OUTP is called to write the references to the binary file. When all the references are out, the END record is written on the binary output file by calling OUTP; the binary output file is closed; and, the last page is ejected for a listing or an end of file written for magnetic tape.

PROGRAMMING TECHNIQUES: FINISH is loaded over parts of the PAS2 code. When the FINISH absolute deck was made, the external references from PAS2 were loaded with FINISH since the table's origins and certain subroutines from PAS2 are used by FINISH. Care must be exercised, therefore, when changing either PAS2 or FINISH to preserve these communications. FINISH is an absolute program separately assembled.

CALLING SEQUENCE: FINISH is loaded and executed as a separate overlay of the assembly system by the tape loader.

MEMORY REQUIREMENTS: Same as for PAS2

SUBROUTINES USED:

SCAN[†]	EDIT[†]	CLOSE[†††]
EPRNT[†]	RDPI[†]	HOME[†]
GLOV[†]	GTLBL[††]	GEC[†]
EDTST[†]	EDTL[†]	
EDTV[†]	OUTP[†]	

ENTRY POINTS TO FINISH SUBROUTINES

Entry	Page Description	Page Flowchart
FINISH	3-341	3-401
GTLBL	3-208	3-402

[†]These routines are part of PAS2.

[††]GTLBL is described under ASSEMBLR.

[†††]REWW and CLOSE are described under MSCONTRL.

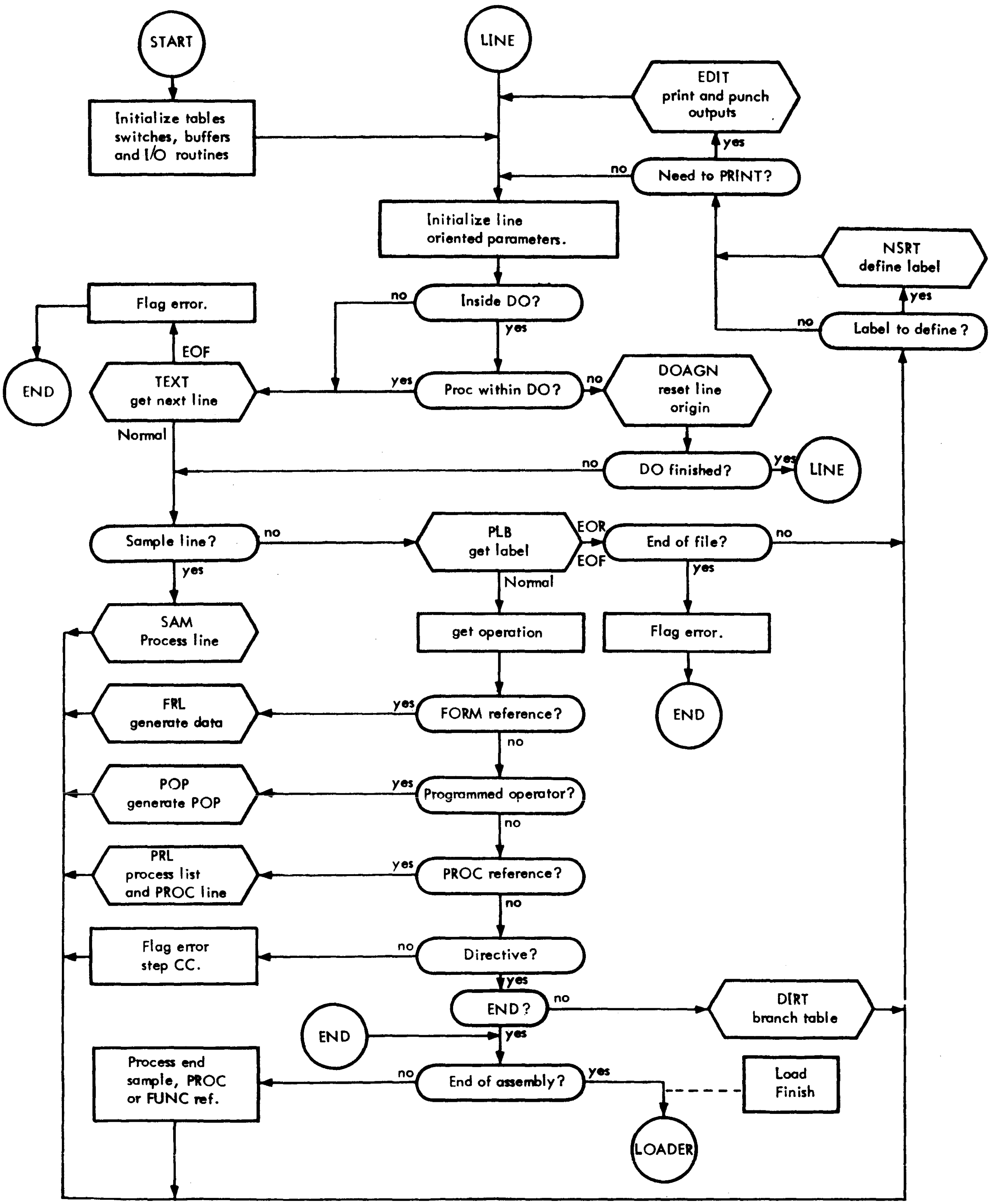
SDS SCIENTIFIC DATA SYSTEMS

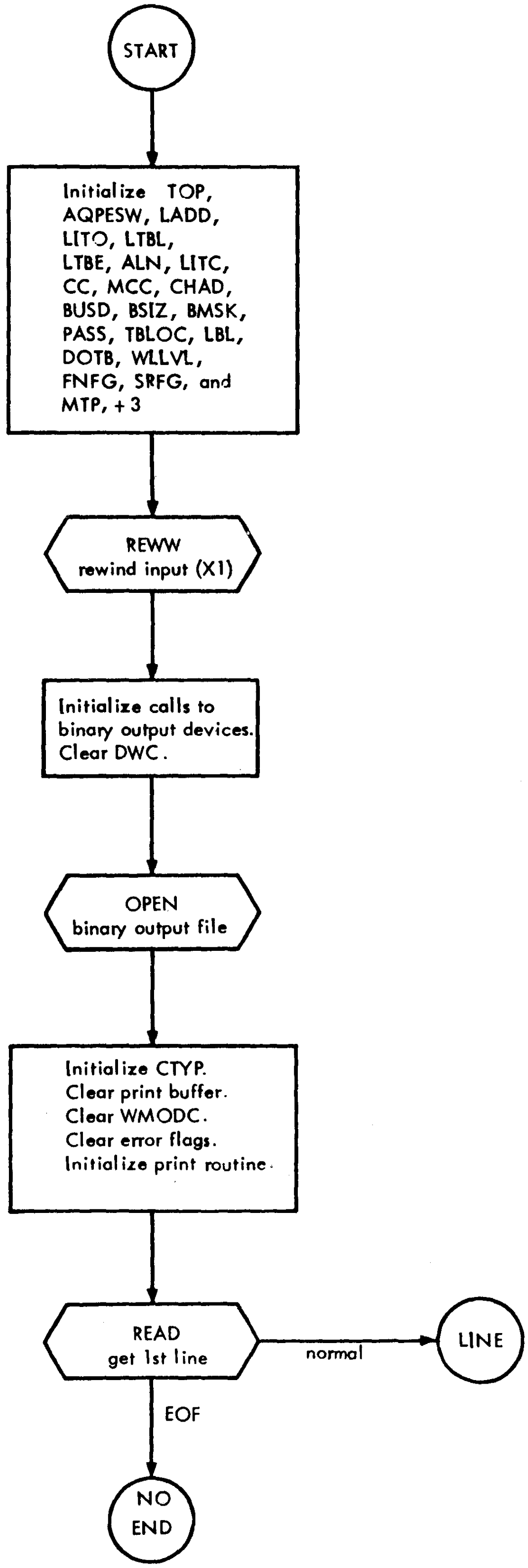
SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

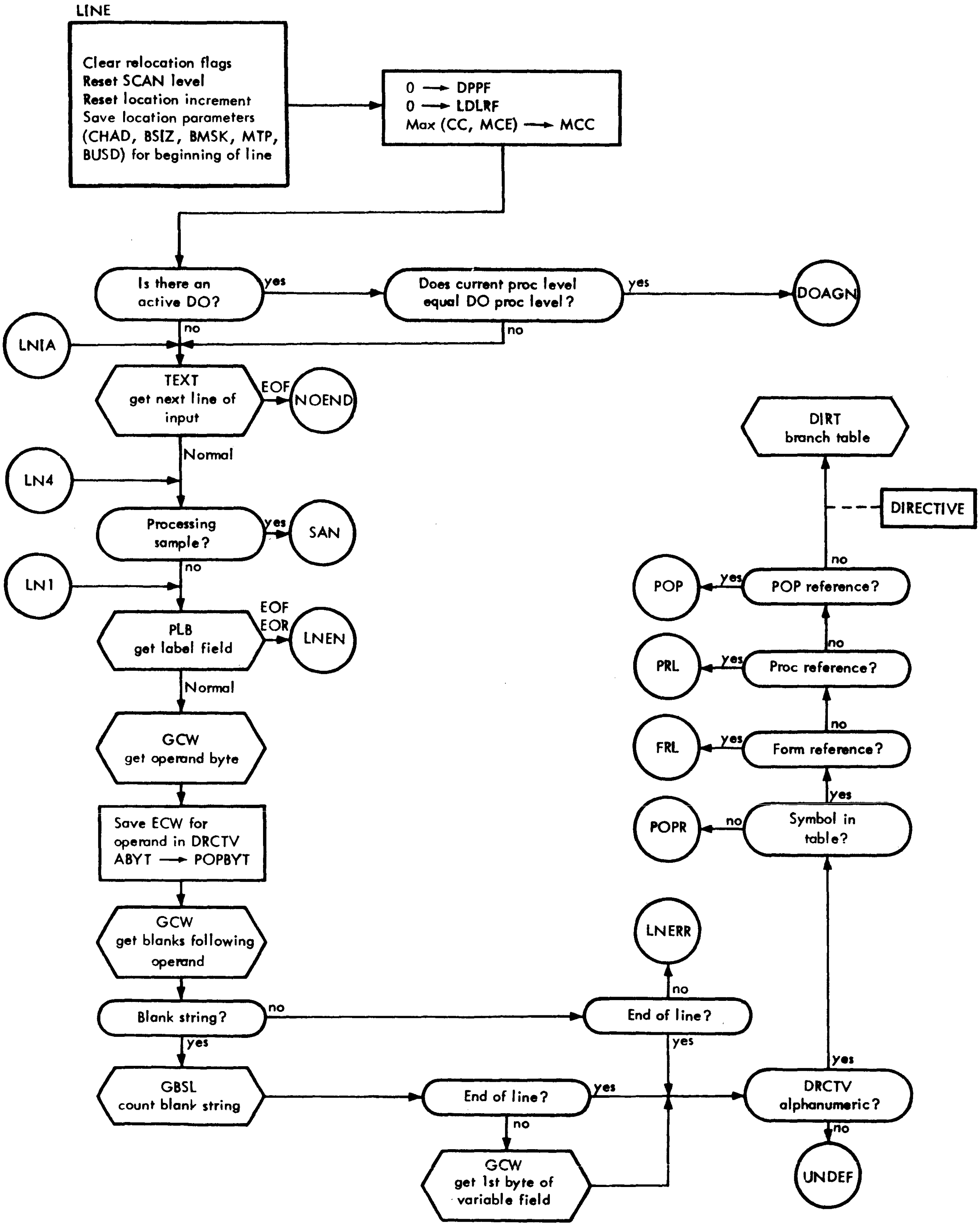
Catalog No. 042016

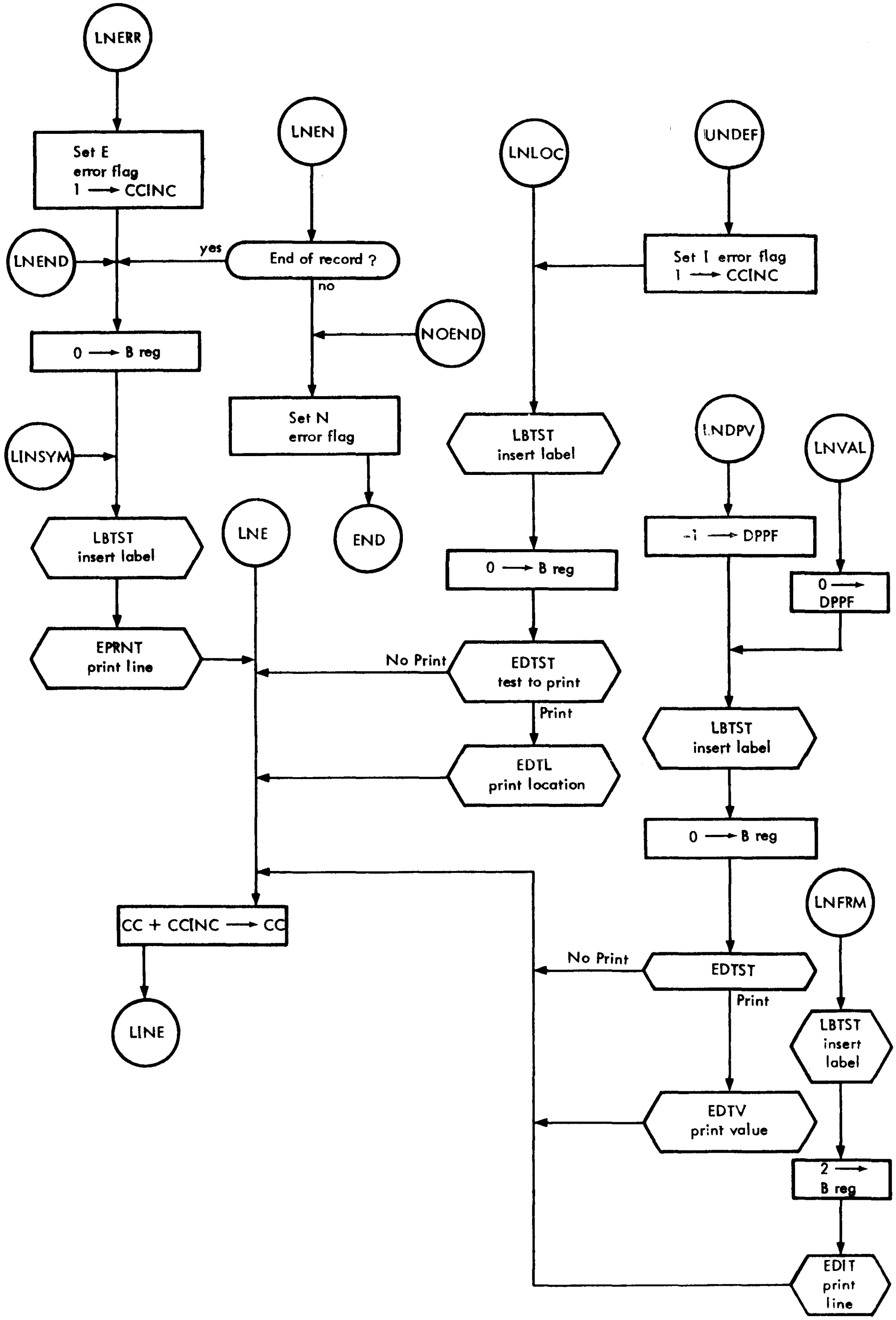
IDENTIFICATION:	Write end-of-file marks on listing output (WEOFL)
PURPOSE:	To write an end-of-file mark on the listing output on magnetic tape.
ACTION:	WEOFL calls the end-of-file routine associated with listing output (EFMT) to write an end-of-file mark.
PROGRAMMING TECHNIQUES:	WEOFL is an absolute routine assembled as part of FINISH.
CALLING SEQUENCE:	BRM WEOFL
MEMORY REQUIREMENTS:	6 cells
SUBROUTINES USED:	End-of-file routine for magnetic tape

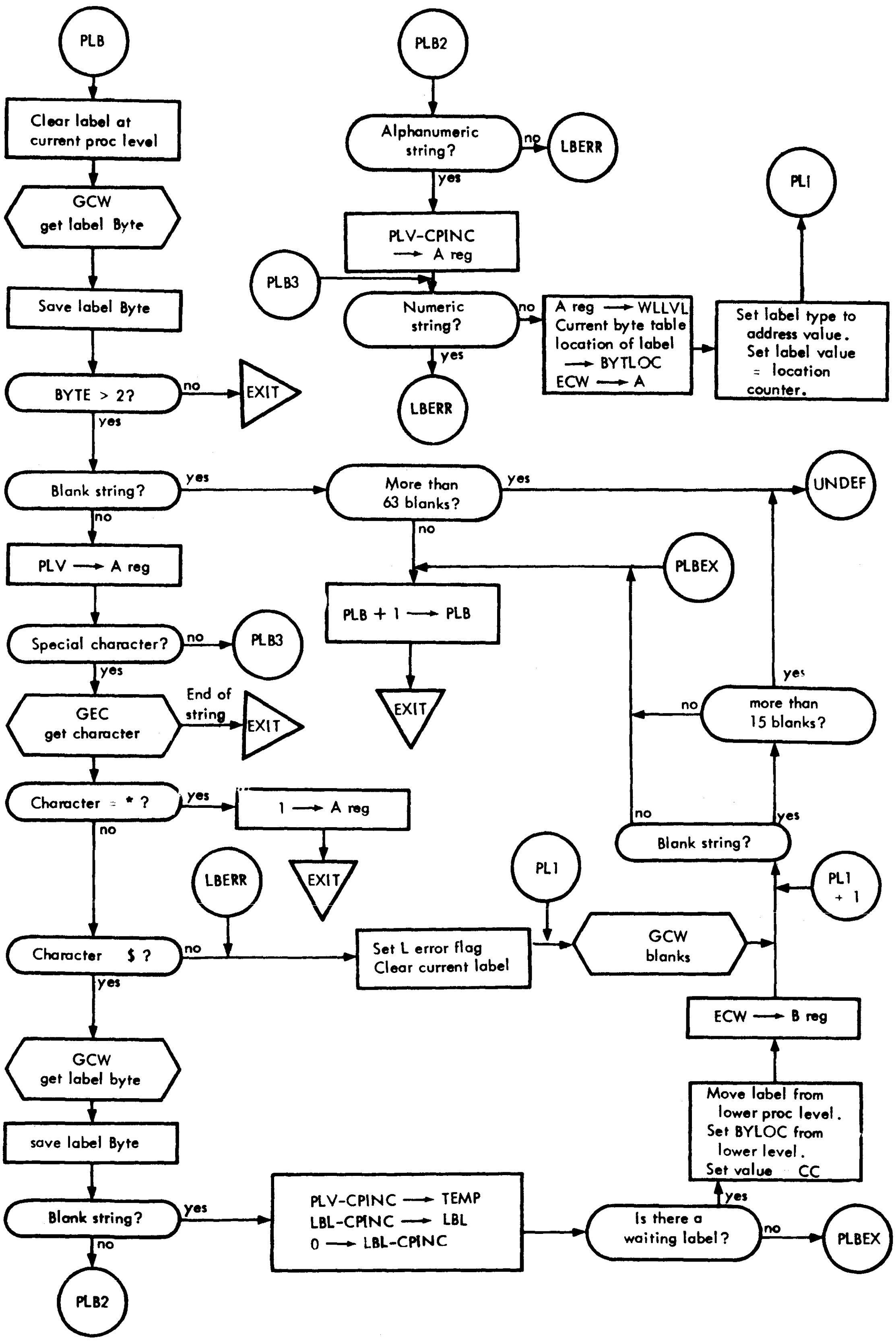
PAS 2
OVERALL FLOW

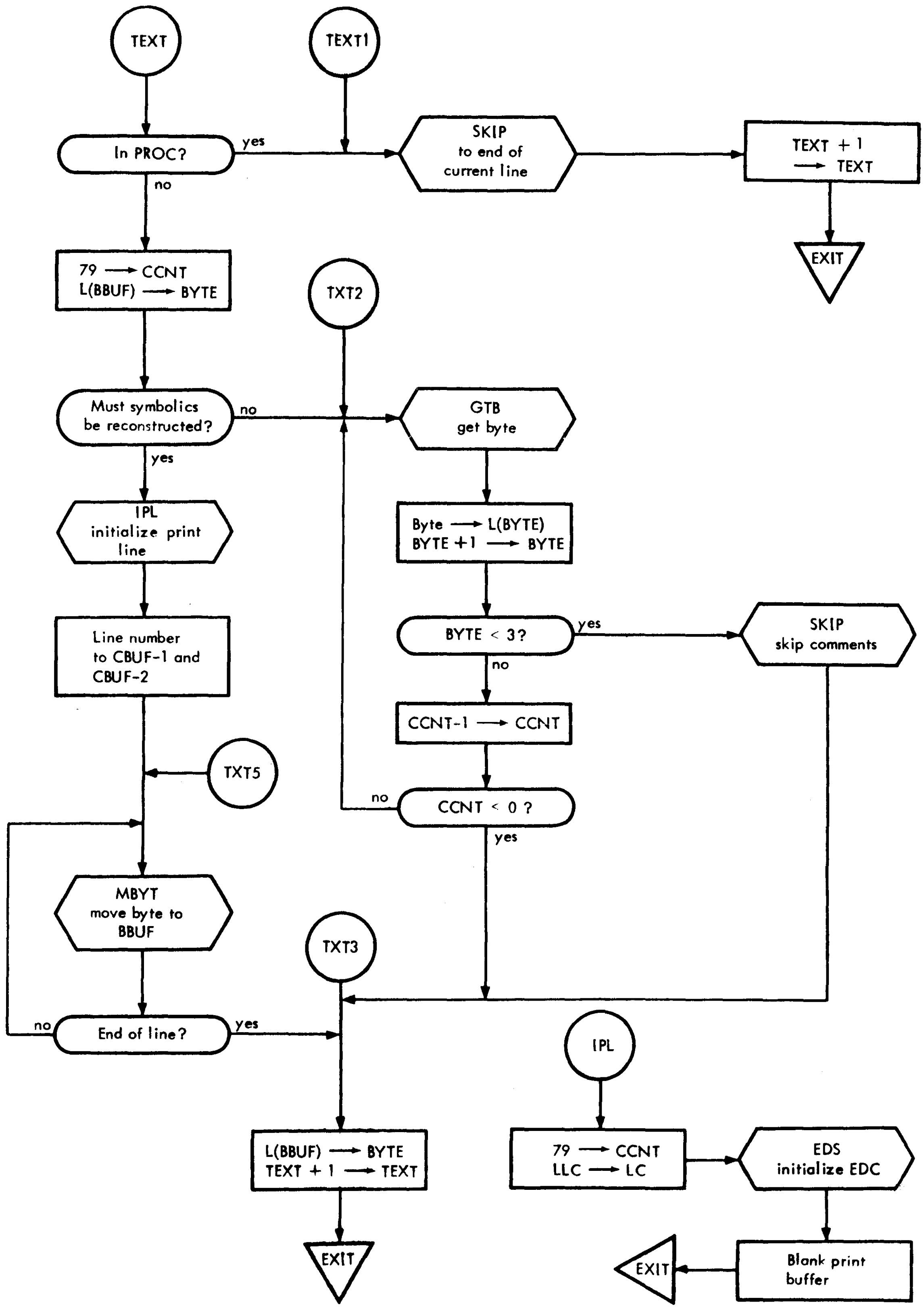


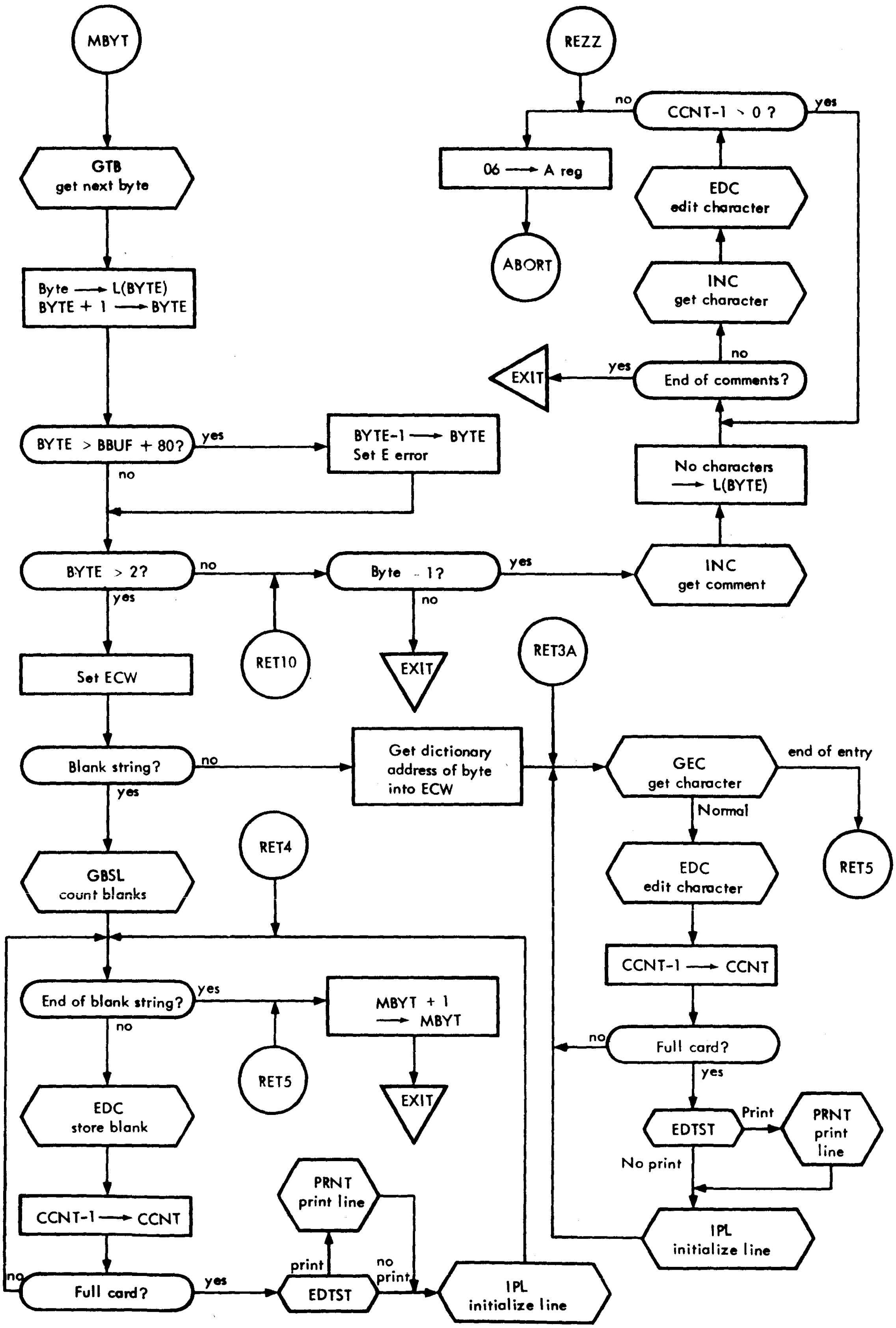

START
Initialize TOP, AQPESW, LADD, LITO, LTBL, LTBE, ALN, LITC, CC, MCC, CHAD, BUSD, BSIZ, BMSK, PASS, TBLOC, LBL, DOTB, WLLVL, FNFG, SRFG, and MTP, +3
REWW
rewind input (X1)
Initialize calls to binary output devices.
Clear DWC.
OPEN
binary output file
Initialize CTYP.
Clear print buffer.
Clear WMODC.
Clear error flags.
Initialize print routine.
READ
get 1st line
normal
LINE
EOF
NO END

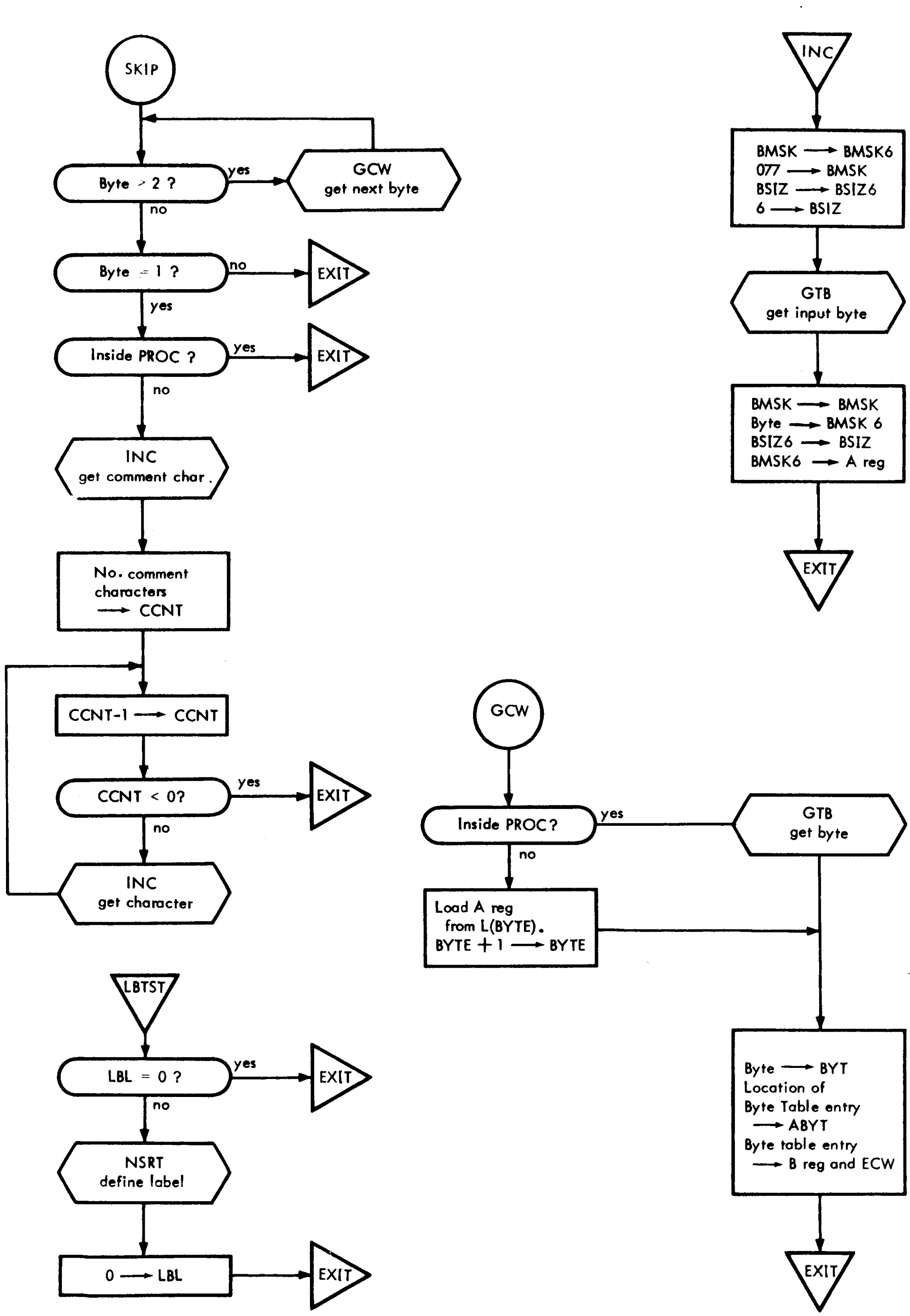

LINE
Clear relocation flags
Reset SCAN level
Reset location increment
Save location parameters
(CHAD, BSIZ, BMSK, MTP,
BUSD) for beginning of line
0 ⟶ DPPF
0 ⟶ LDLRF
Max (CC, MCE) ⟶ MCC
Is there an active DO?
yes
Does current proc level equal DO proc level?
yes
DOAGN
no
no
LN1A
TEXT
get next line of input
EOF
NOEND
Normal
LN4
Processing sample?
yes
SAN
no
LN1
PLB
get label field
EOF
EOR
LNEN
Normal
GCW
get operand byte
Save ECW for operand in DRCTV
ABYT ⟶ POPBYT
GCW
get blanks following operand
Blank string?
no
End of line?
no
LNERR
yes
yes
GBSL
count blank string
End of line?
yes
no
GCW
get 1st byte of variable field
DRCTV
alphanumeric?
yes
no
UNDEF
Symbol in table?
no
POPR
yes
Form reference?
yes
FRL
no
Proc reference?
yes
PRL
no
POP reference?
yes
POP
no
DIRT
branch table
DIRECTIVE


LNERR
Set E error flag
1 → CCINC
LNEN
LNLOC
UNDEF
LNEND
yes
End of record ?
no
Set I error flag
1 → CCINC
NOEND
0 → B reg
Set N error flag
LBTST insert label
LNDPV
LNVAL
LINSYM
LBTST insert label
LNE
END
-1 → DPPF
0 → B reg
0 → DPPF
EPRNT print line
No Print
EDTST test to print
Print
LBTST insert label
EDTL print location
0 → B reg
LNFRM
CC + CCINC → CC
No Print
EDTST
Print
LINE
LBTST insert label
EDTV print value
2 → B reg
EDIT print line

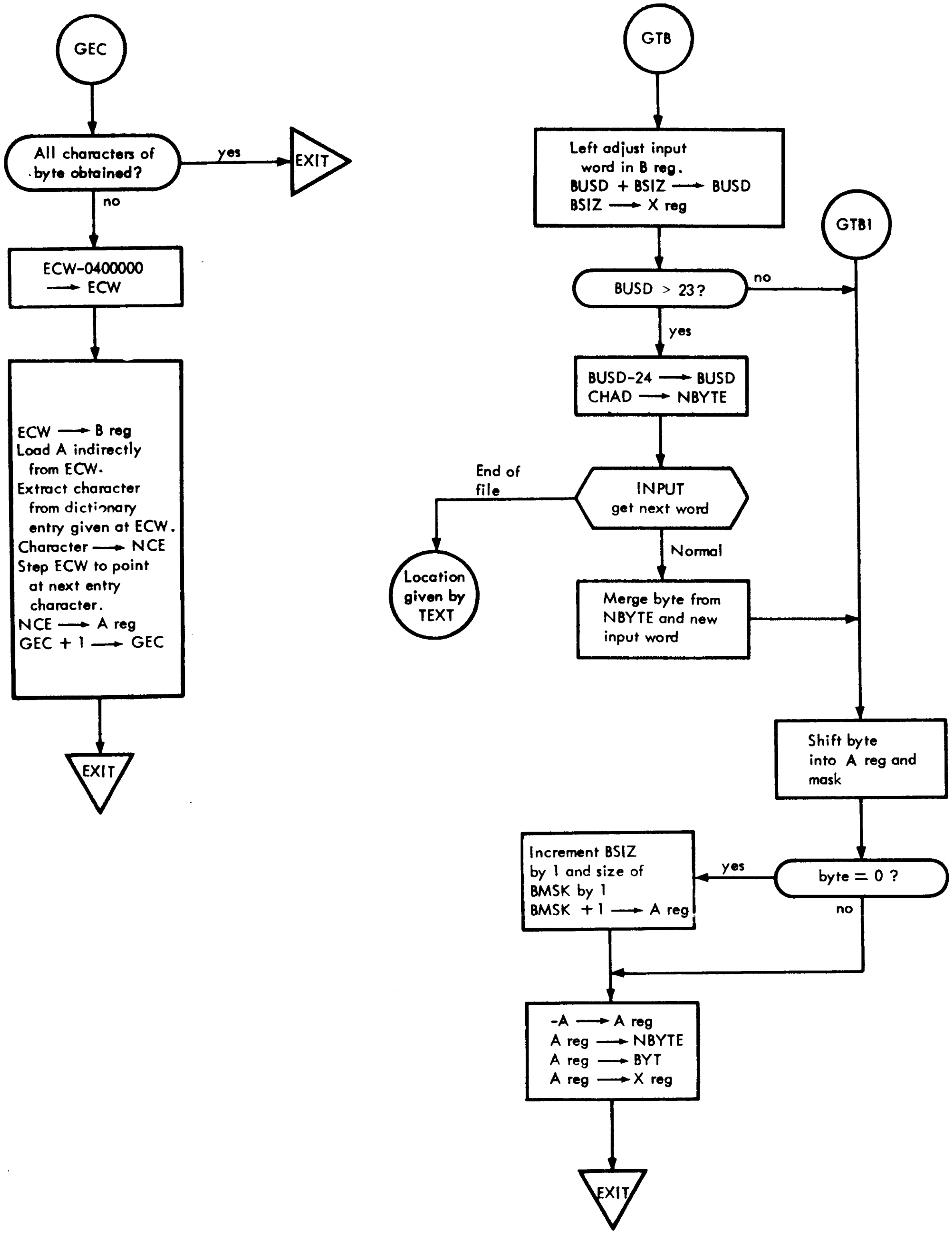

PLB
Clear label at current proc level
GCW get label Byte
Save label Byte
BYTE > 2?
no
EXIT
yes
Blank string?
yes
no
PLV → A reg
Special character?
no
PLB3
yes
GEC get character
End of string
EXIT
Character = * ?
yes
1 → A reg
EXIT
no
Character $?
no
yes
GCW get label byte
save label Byte
Blank string?
yes
no
PLB2
PLB2
Alphanumeric string?
no
LBERR
yes
PLV-CPINC → A reg
PLB3
Numeric string?
no
yes
LBERR
A reg → WLLVL
Current byte table location of label → BYTLOC
ECW → A
Set label type to address value.
Set label value = location counter.
PL1
More than 63 blanks?
yes
UNDEF
no
PLBEX
PLB + 1 → PLB
EXIT
more than 15 blanks?
yes
no
Blank string?
no
yes
PL1 + 1
LBERR
PL1
Set L error flag
Clear current label
GCW blanks
ECW → B reg
Move label from lower proc level.
Set BYLOC from lower level.
Set value CC
yes
PLV-CPINC → TEMP
LBL-CPINC → LBL
0 → LBL-CPINC
Is there a waiting label?
no
PLBEX

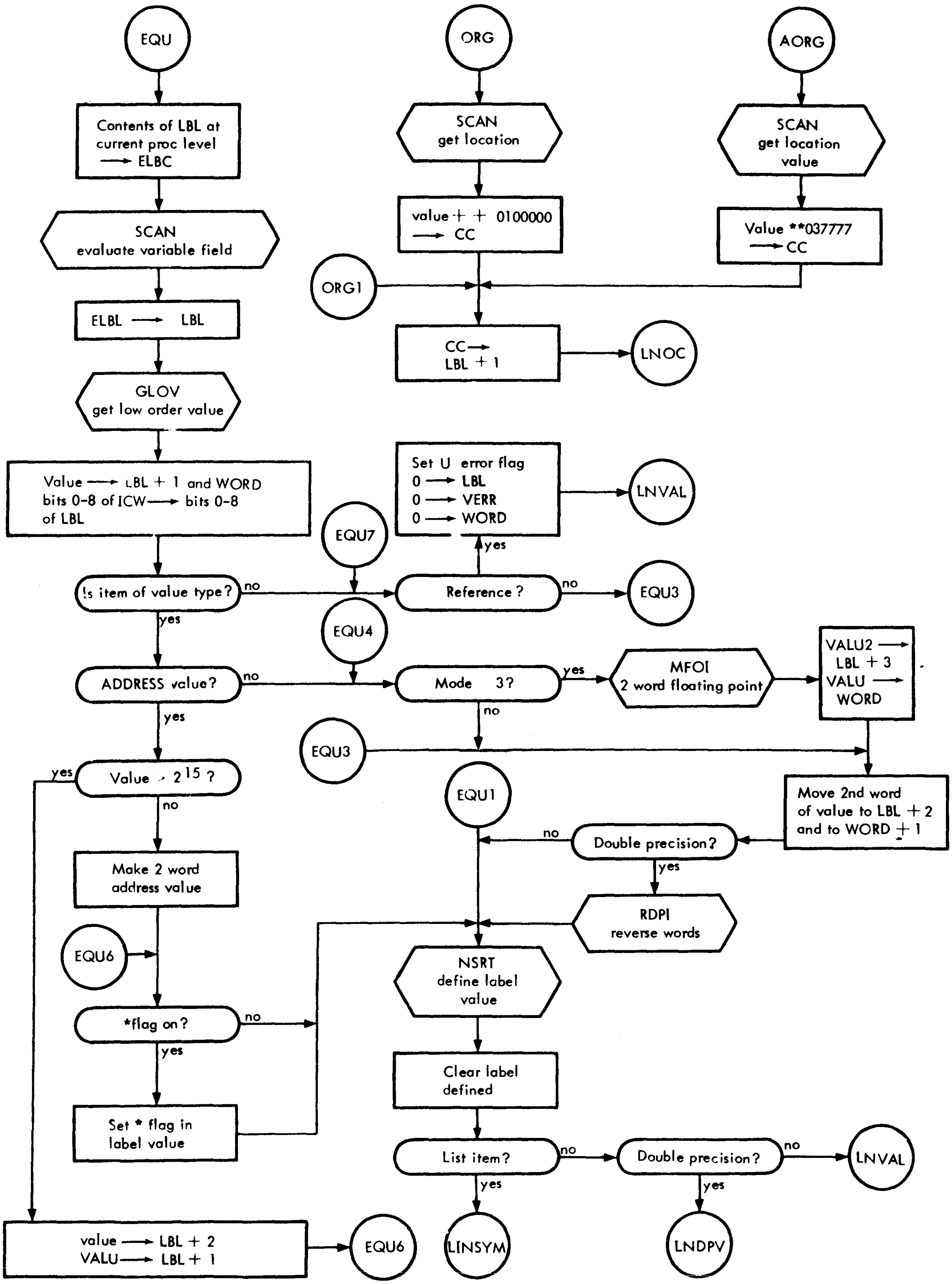

TEXT
TEXT1
In PROC?
yes
no
SKIP
to end of
current line
TEXT + 1
→ TEXT
EXIT
79 → CCNT
L(BBUF) → BYTE
TXT2
Must symbolics
be reconstructed?
no
yes
GTB
get byte
IPL
initialize print
line
Byte → L(BYTE)
BYTE +1 → BYTE
Line number
to CBUF-1 and
CBUF-2
BYTE < 3?
yes
no
SKIP
skip comments
CCNT-1 → CCNT
TXT5
CCNT < 0 ?
no
yes
MBYT
move byte to
BBUF
TXT3
End of line?
no
yes
IPL
L(BBUF) → BYTE
TEXT + 1 → TEXT
79 → CCNT
LLC → LC
EDS
initialize EDC
EXIT
Blank print
buffer
EXIT

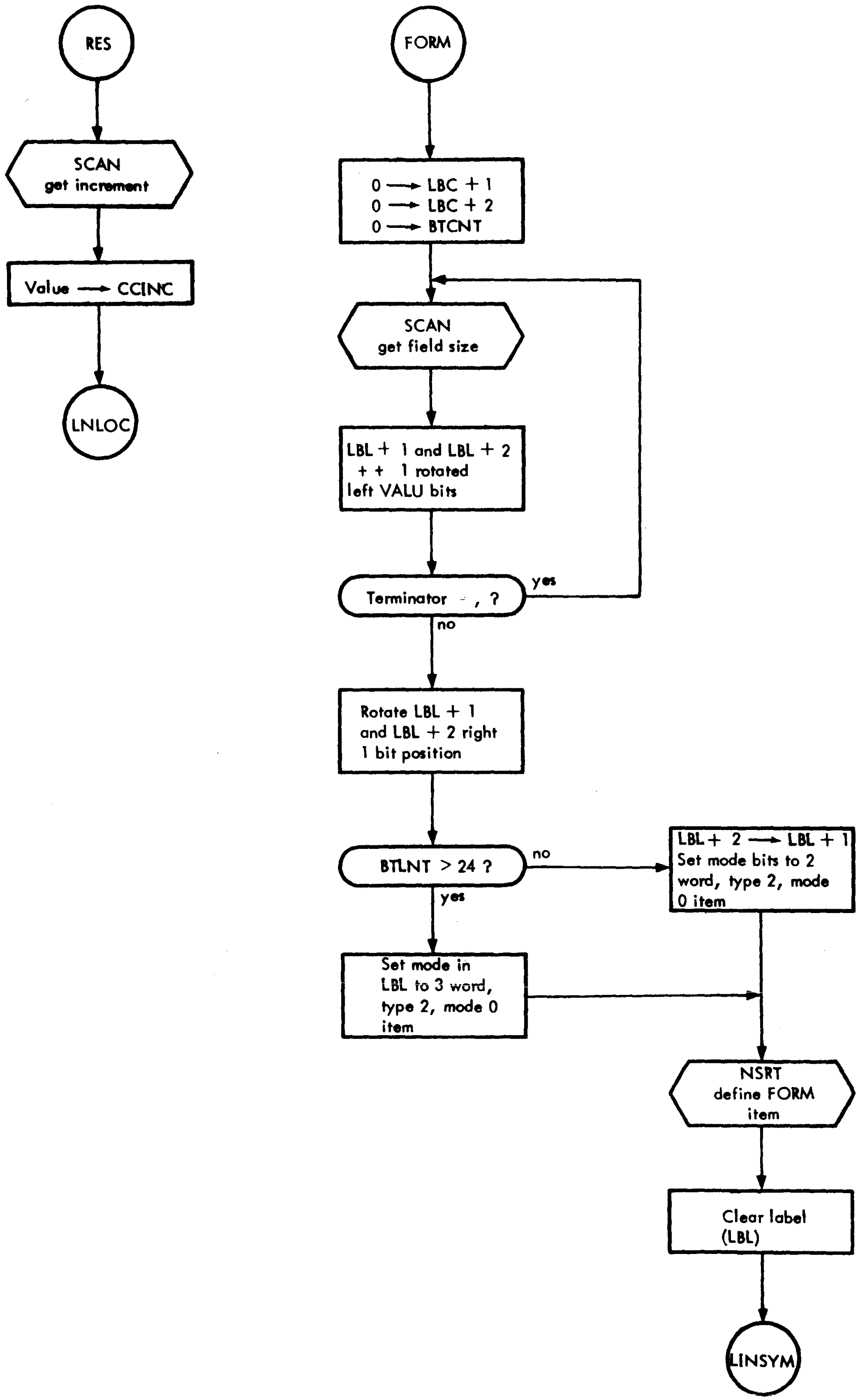

MBYT
GTB
get next byte
Byte → L(BYTE)
BYTE + 1 → BYTE
BYTE > BBUF + 80?
yes
no
BYTE-1 → BYTE
Set E error
BYTE > 2?
no
yes
Byte = 1?
yes
no
RET10
EXIT
Set ECW
Blank string?
no
yes
Get dictionary address of byte into ECW
GBSL
count blanks
RET4
End of blank string?
yes
no
MBYT + 1 → MBYT
RET5
EXIT
EDC
store blank
CCNT-1 → CCNT
PRNT
print line
Full card?
no
yes
print
no print
EDTST
IPL
initialize line
REZZ
CCNT-1 > 0 ?
no
yes
06 → A reg
ABORT
EDC
edit character
INC
get character
End of comments?
yes
no
EXIT
No characters → L(BYTE)
INC
get comment
RET3A
GEC
get character
end of entry
Normal
RET5
EDC
edit character
CCNT-1 → CCNT
Full card?
no
yes
EDTST
Print
PRNT
print line
No print
IPL
initialize line


SKIP
Byte > 2 ?
yes
no
GCW
get next byte
Byte = 1 ?
no
EXIT
yes
Inside PROC ?
yes
EXIT
no
INC
get comment char.
No. comment characters → CCNT
CCNT-1 → CCNT
CCNT < 0?
yes
EXIT
no
INC
get character
LBTST
LBL = 0 ?
yes
EXIT
no
NSRT
define label
0 → LBL
EXIT
INC
BMSK → BMSK6
077 → BMSK
BSIZ → BSIZ6
6 → BSIZ
GTB
get input byte
BMSK → BMSK
Byte → BMSK 6
BSIZ6 → BSIZ
BMSK6 → A reg
EXIT
GCW
Inside PROC?
yes
no
GTB
get byte
Load A reg from L(BYTE).
BYTE + 1 → BYTE
Byte → BYT
Location of Byte Table entry → ABYT
Byte table entry → B reg and ECW
EXIT

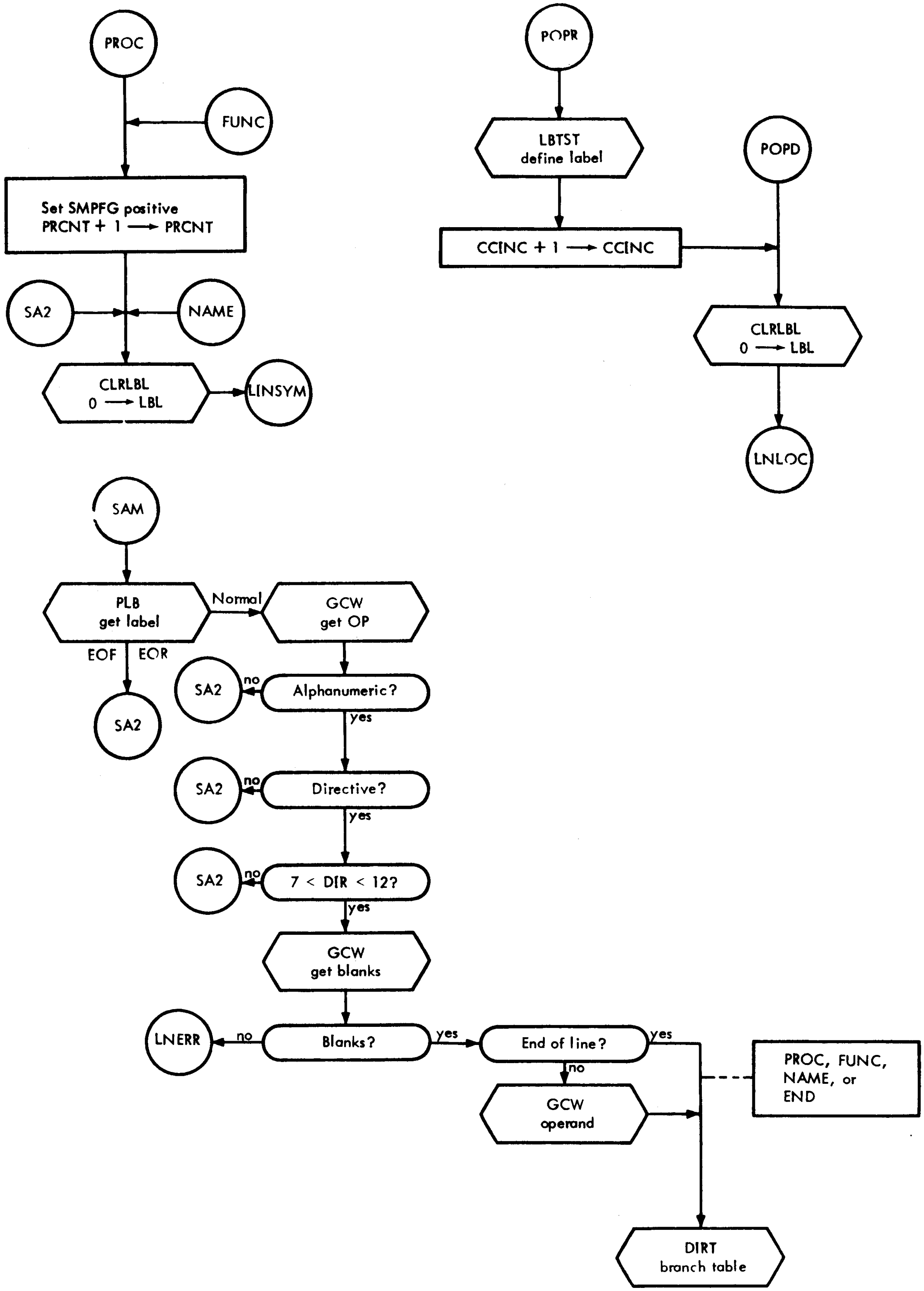
PAS2
GEC AND GTB ROUTINES



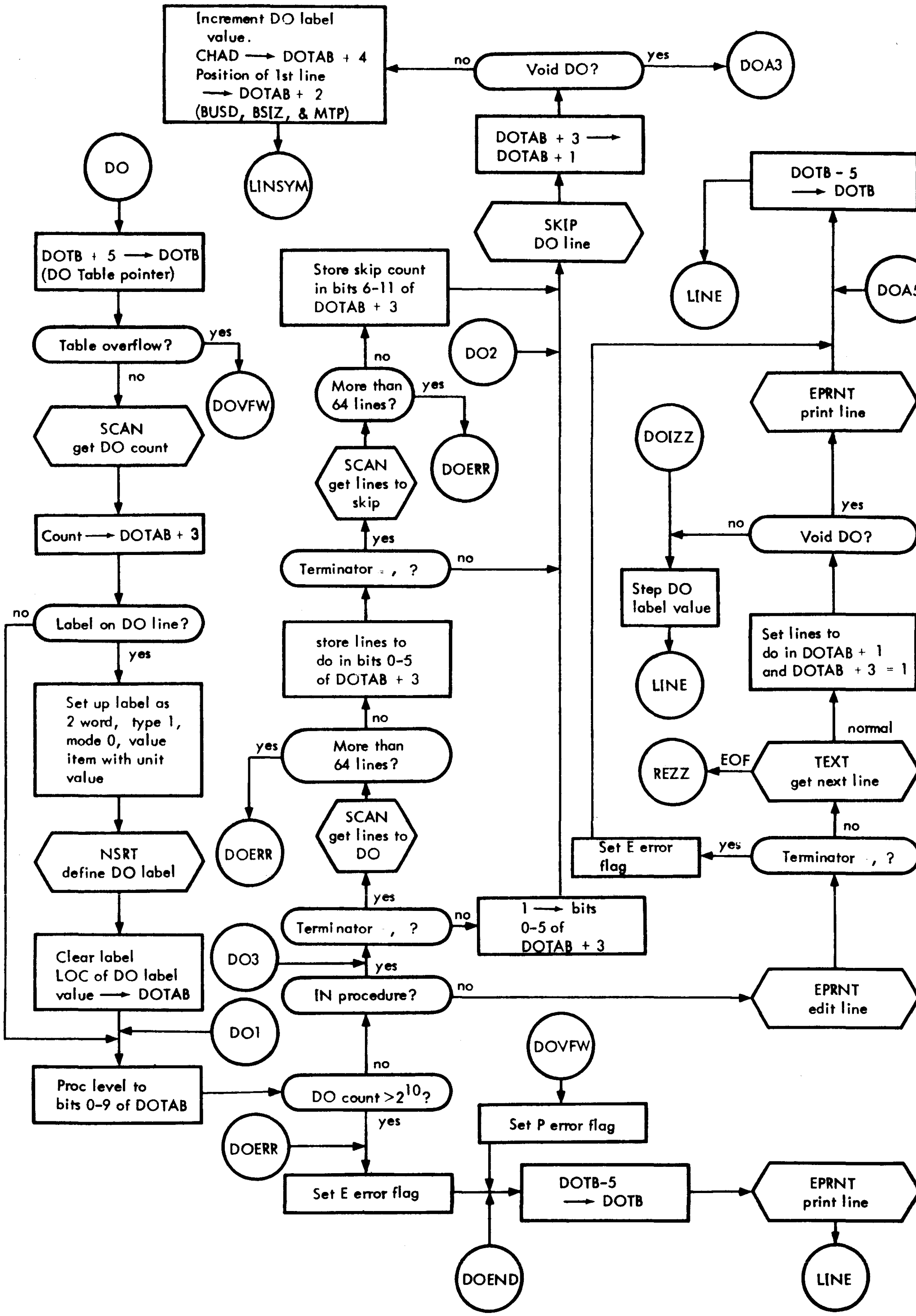

EQU
Contents of LBL at current proc level ⟶ ELBC
SCAN evaluate variable field
ELBL ⟶ LBL
GLOV get low order value
Value ⟶ LBL + 1 and WORD bits 0-8 of ICW ⟶ bits 0-8 of LBL
Is item of value type?
ADDRESS value?
Value > 2 15 ?
Make 2 word address value
EQU6
*flag on?
Set * flag in label value
value ⟶ LBL + 2 VALU ⟶ LBL + 1
EQU6
ORG
SCAN get location
value + + 0100000 ⟶ CC
ORG1
CC ⟶ LBL + 1
LNOC
AORG
SCAN get location value
Value **037777 ⟶ CC
Set U error flag 0 ⟶ LBL 0 ⟶ VERR 0 ⟶ WORD
LNVAL
EQU7
Reference ?
EQU3
EQU4
Mode 3?
MFOI 2 word floating point
VALU2 ⟶ LBL + 3 VALU ⟶ WORD
EQU3
Move 2nd word of value to LBL + 2 and to WORD + 1
EQU1
Double precision?
RDPI reverse words
NSRT define label value
Clear label defined
List item?
Double precision?
LNVAL
LINSYM
LNDPV
yes
no


RES
SCAN
get increment
Value ⟶ CCINC
LNLOC
FORM
0 ⟶ LBC + 1
0 ⟶ LBC + 2
0 ⟶ BTCNT
SCAN
get field size
LBL + 1 and LBL + 2
+ + 1 rotated
left VALU bits
Terminator , ?
yes
no
Rotate LBL + 1
and LBL + 2 right
1 bit position
BTLNT > 24 ?
no
yes
LBL + 2 ⟶ LBL + 1
Set mode bits to 2
word, type 2, mode
0 item
Set mode in
LBL to 3 word,
type 2, mode 0
item
NSRT
define FORM
item
Clear label
(LBL)
LINSYM

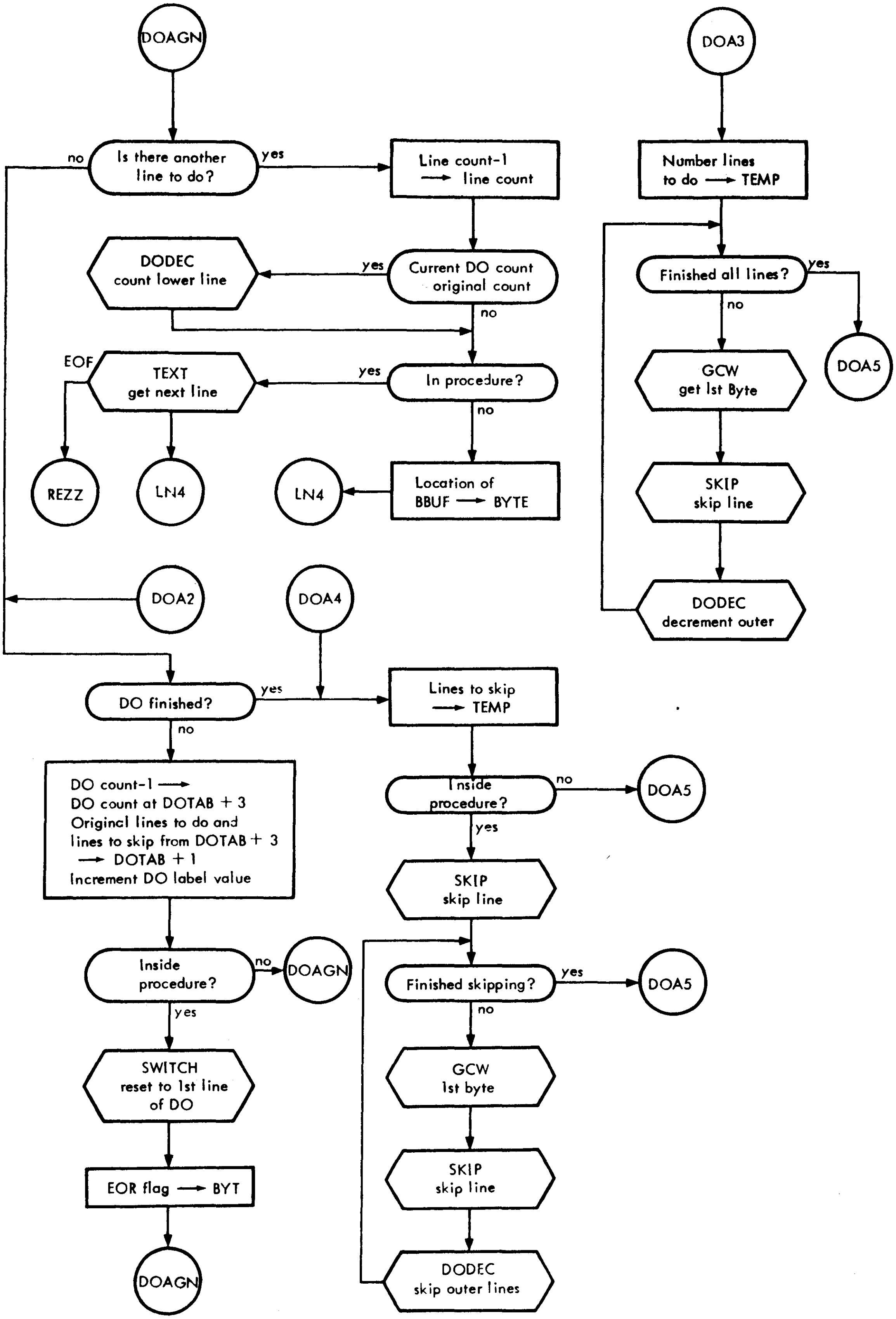
NAME, SAM, PROC FUNC, POPD AND POPR ROUTINES

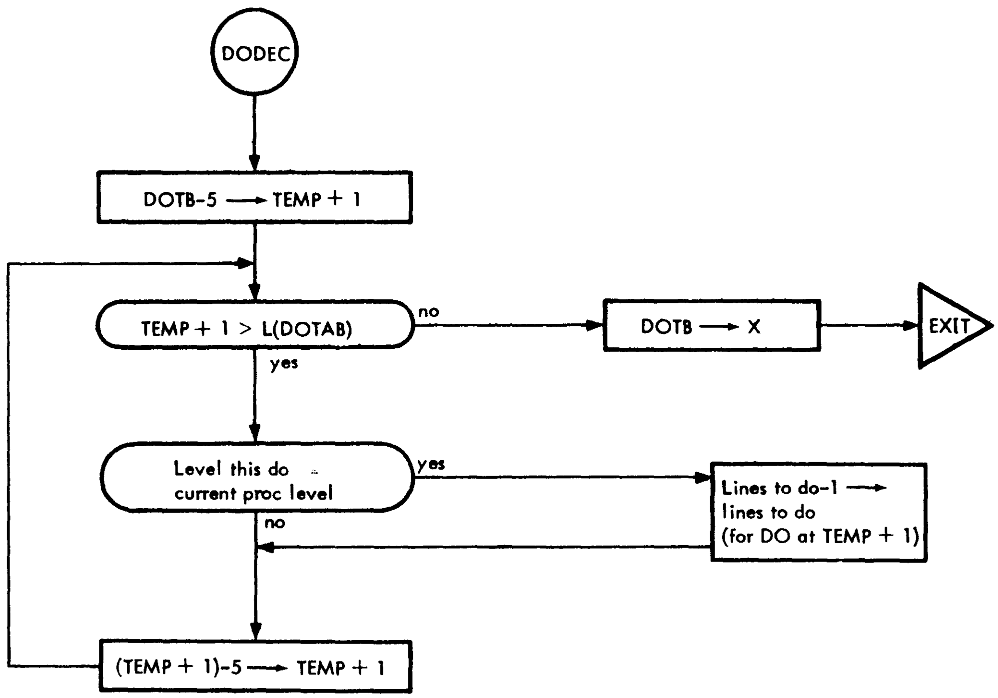


DO ROUTINE

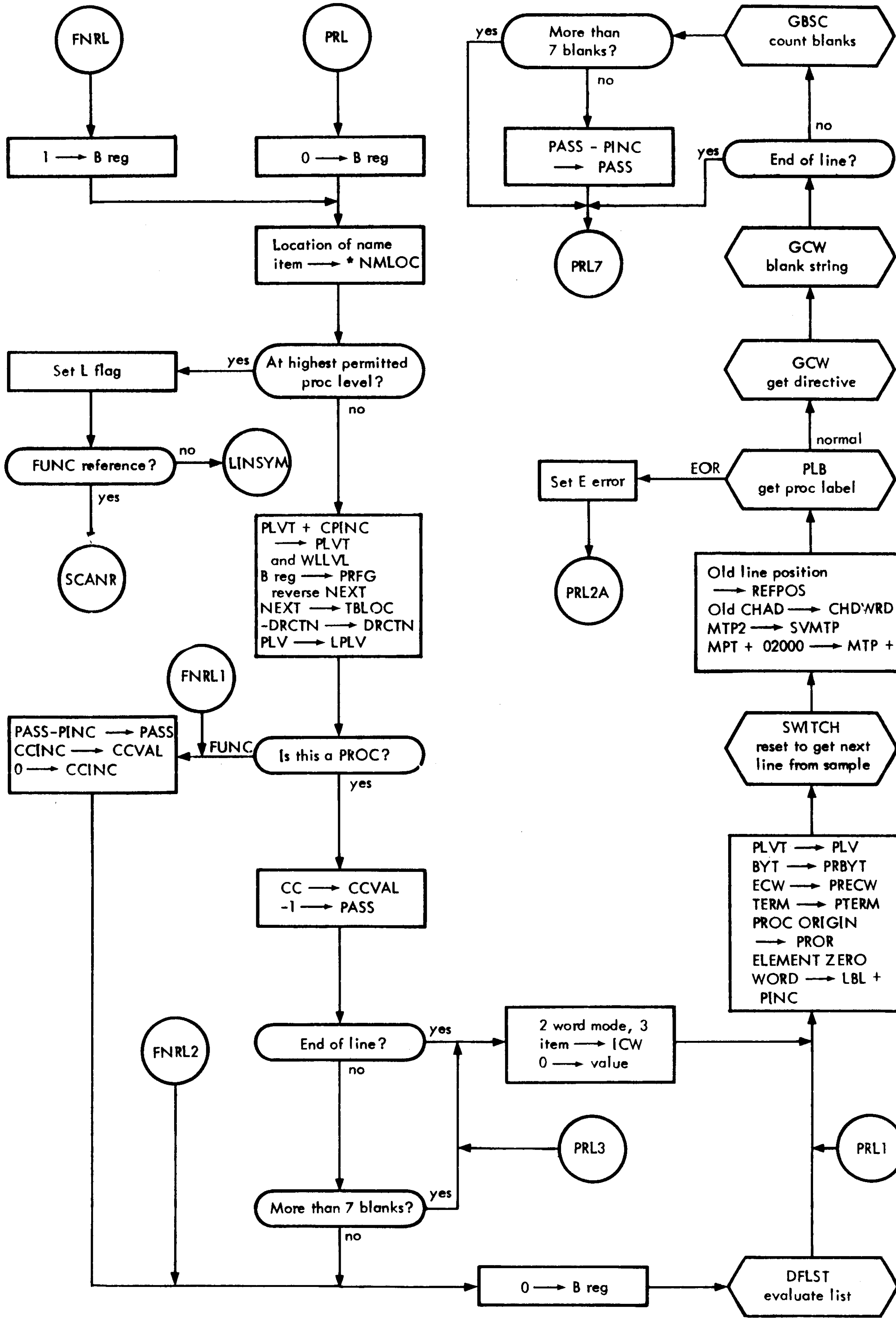


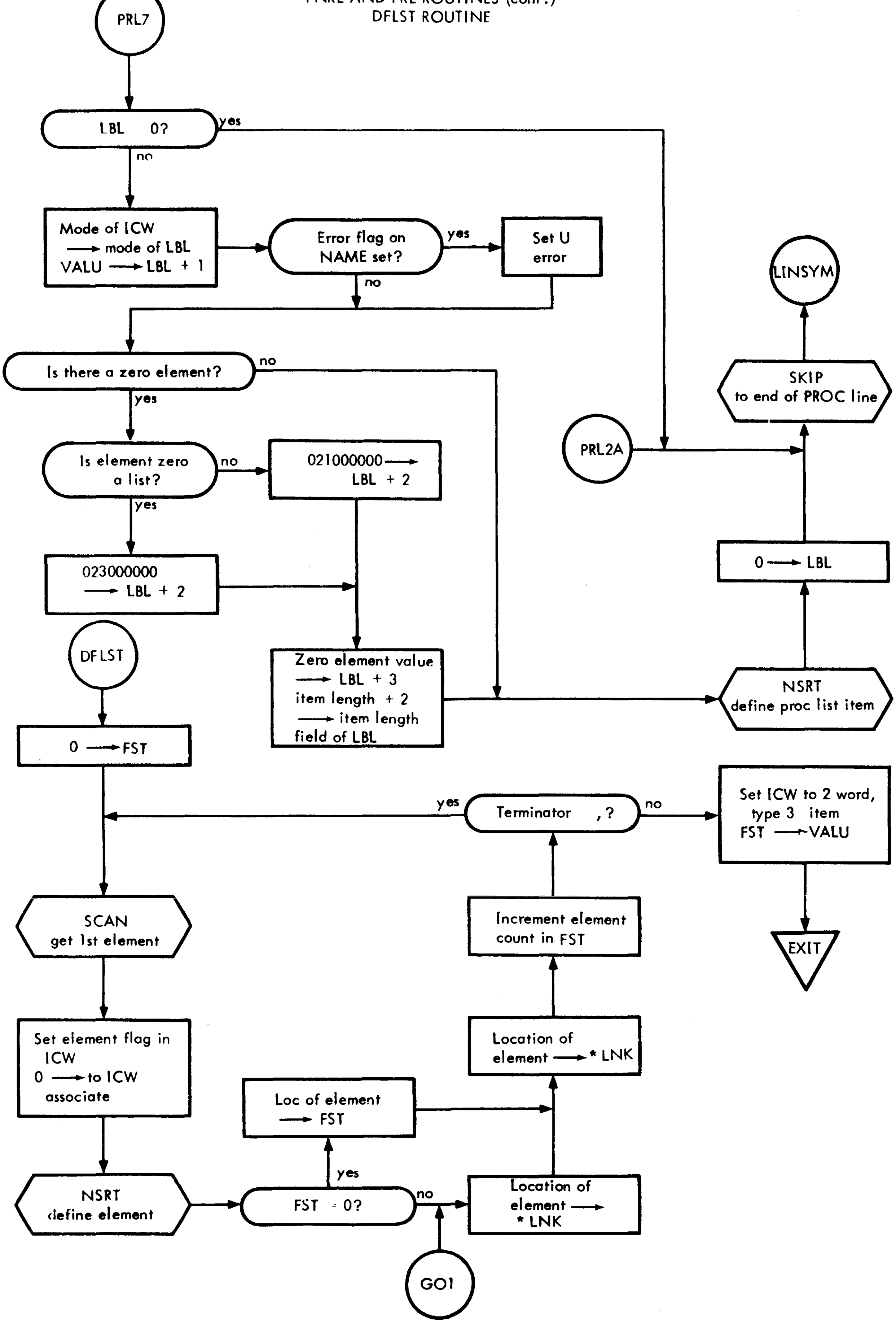
Increment DO label value. CHAD → DOTAB + 4 Position of 1st line → DOTAB + 2 (BUSD, BSIZ, & MTP)
Void DO?
no
yes
DOA3
LINSYM
DOTAB + 3 → DOTAB + 1
DO
DOTB - 5 → DOTB
SKIP DO line
DOTB + 5 → DOTB (DO Table pointer)
Store skip count in bits 6-11 of DOTAB + 3
LINE
DOA5
Table overflow?
yes
no
DO2
DOVFW
More than 64 lines?
EPRNT print line
SCAN get DO count
SCAN get lines to skip
DOERR
DOIZZ
Void DO?
Count → DOTAB + 3
Terminator , ?
Step DO label value
Label on DO line?
store lines to do in bits 0-5 of DOTAB + 3
Set lines to do in DOTAB + 1 and DOTAB + 3 = 1
LINE
Set up label as 2 word, type 1, mode 0, value item with unit value
More than 64 lines?
normal
REZZ
EOF
TEXT get next line
SCAN get lines to DO
NSRT define DO label
DOERR
Set E error flag
Terminator , ?
Terminator , ?
1 → bits 0-5 of DOTAB + 3
Clear label LOC of DO label value → DOTAB
DO3
IN procedure?
EPRNT edit line
DO1
DOVFW
Proc level to bits 0-9 of DOTAB
DO count > 2^10?
Set P error flag
DOERR
Set E error flag
DOTB-5 → DOTB
EPRNT print line
DOEND
LINE

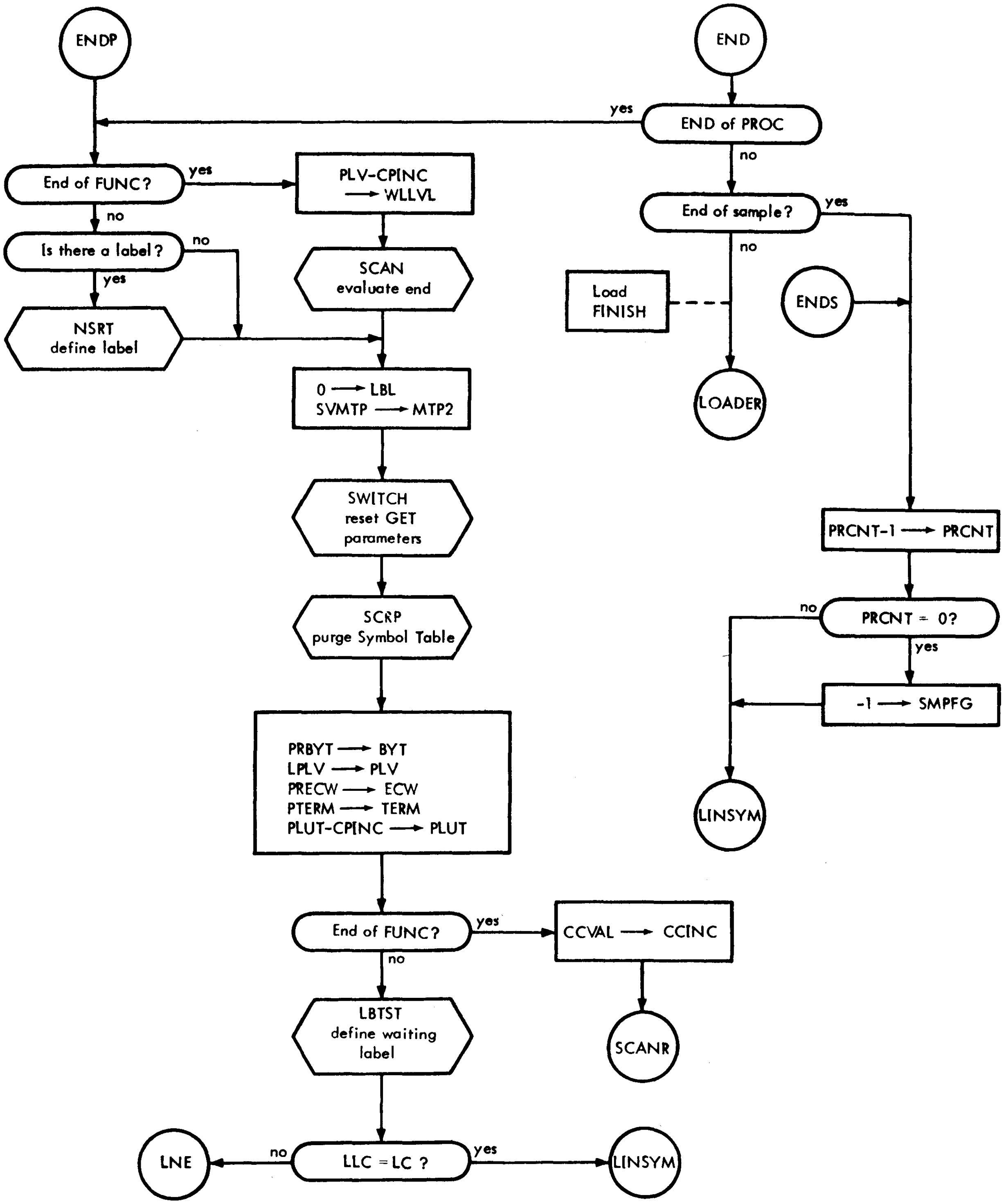

DOAGN
Is there another line to do?
no
yes
Line count-1 → line count
DODEC count lower line
yes
Current DO count original count
no
EOF
TEXT get next line
yes
In procedure?
no
REZZ
LN4
LN4
Location of BBUF → BYTE
DOA2
DOA4
DO finished?
yes
no
Lines to skip → TEMP
DO count-1 →
DO count at DOTAB + 3
Original lines to do and lines to skip from DOTAB + 3 → DOTAB + 1
Increment DO label value
Inside procedure?
no
DOA5
yes
SKIP skip line
Inside procedure?
no
DOAGN
yes
Finished skipping?
yes
DOA5
no
SWITCH reset to 1st line of DO
GCW 1st byte
SKIP skip line
EOR flag → BYT
DODEC skip outer lines
DOAGN
DOA3
Number lines to do → TEMP
Finished all lines?
yes
no
DOA5
GCW get 1st Byte
SKIP skip line
DODEC decrement outer

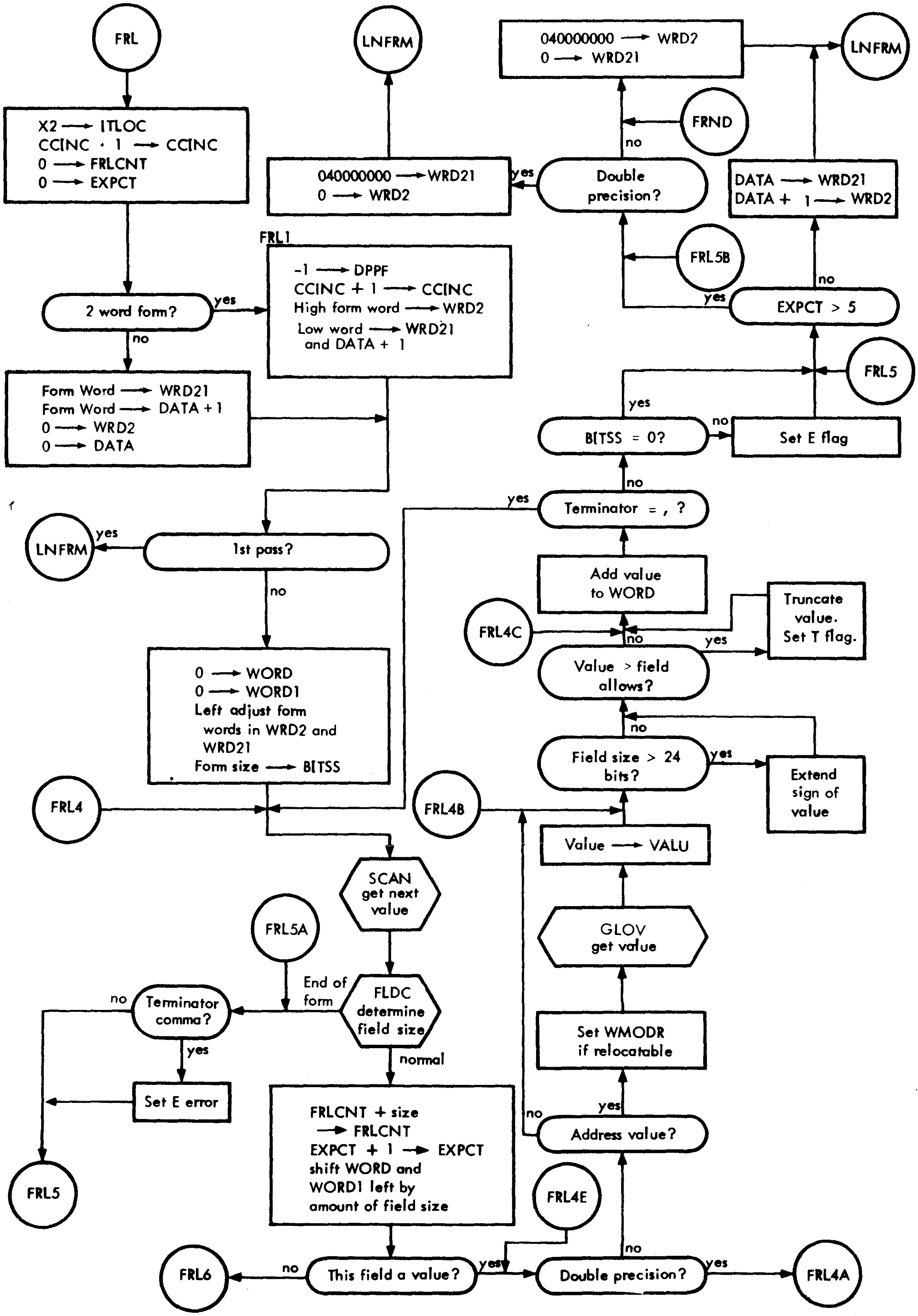

DODEC
DOTB-5 ⟶ TEMP + 1
TEMP + 1 > L(DOTAB)
no
yes
DOTB ⟶ X
EXIT
Level this do =
current proc level
yes
no
Lines to do-1 ⟶
lines to do
(for DO at TEMP + 1)
(TEMP + 1)-5 ⟶ TEMP + 1

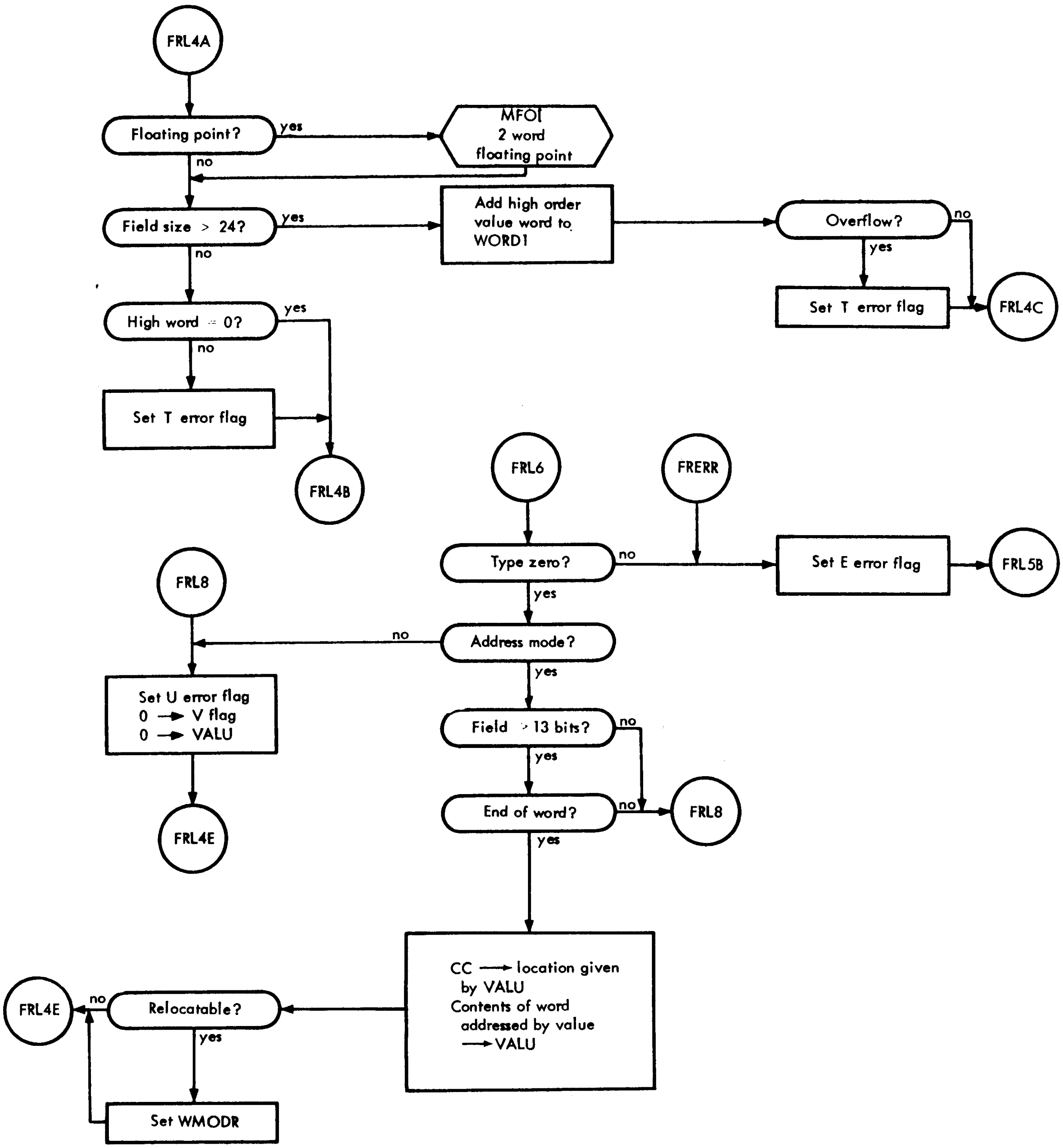
FNRL AND PRL ROUTINES

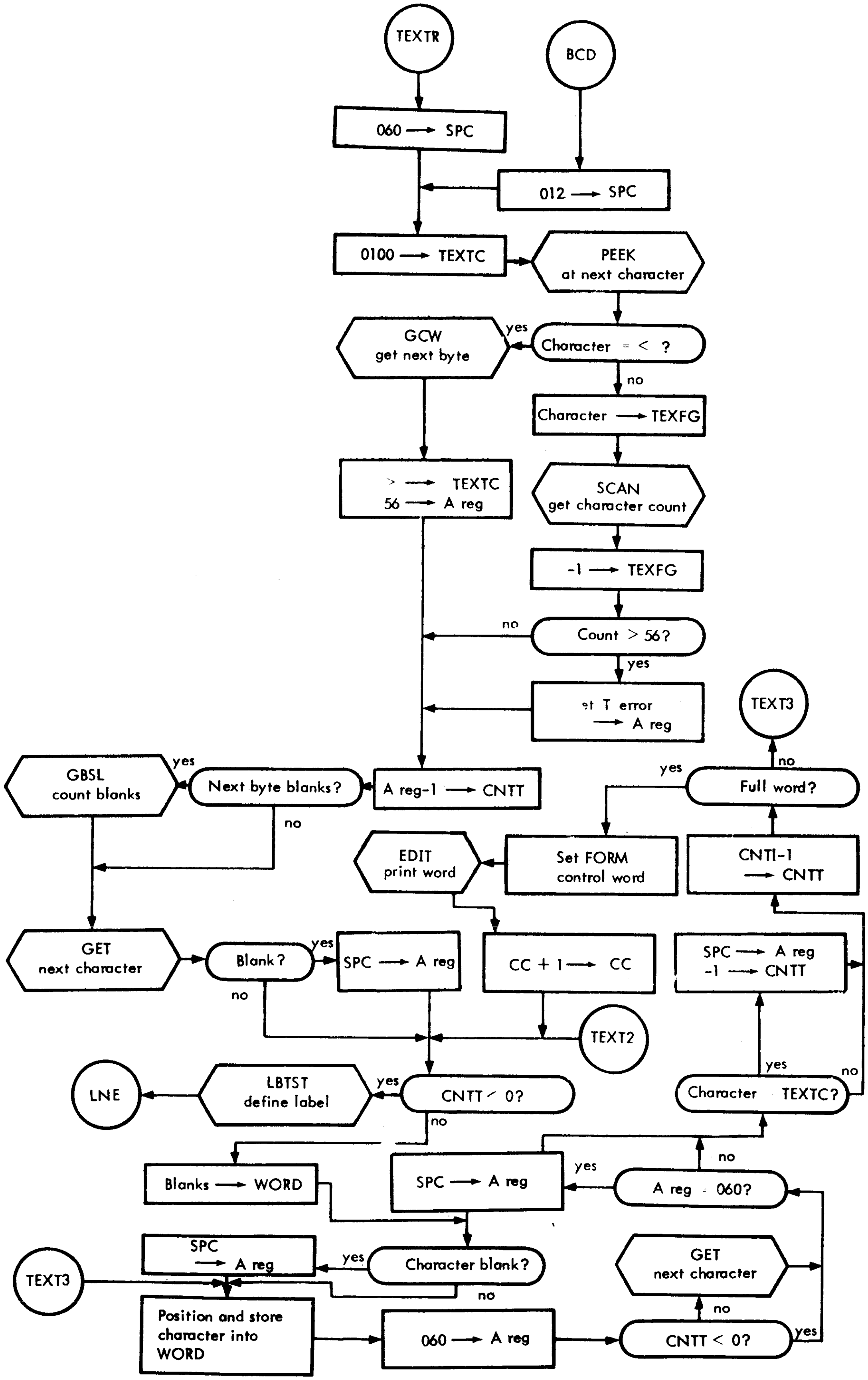


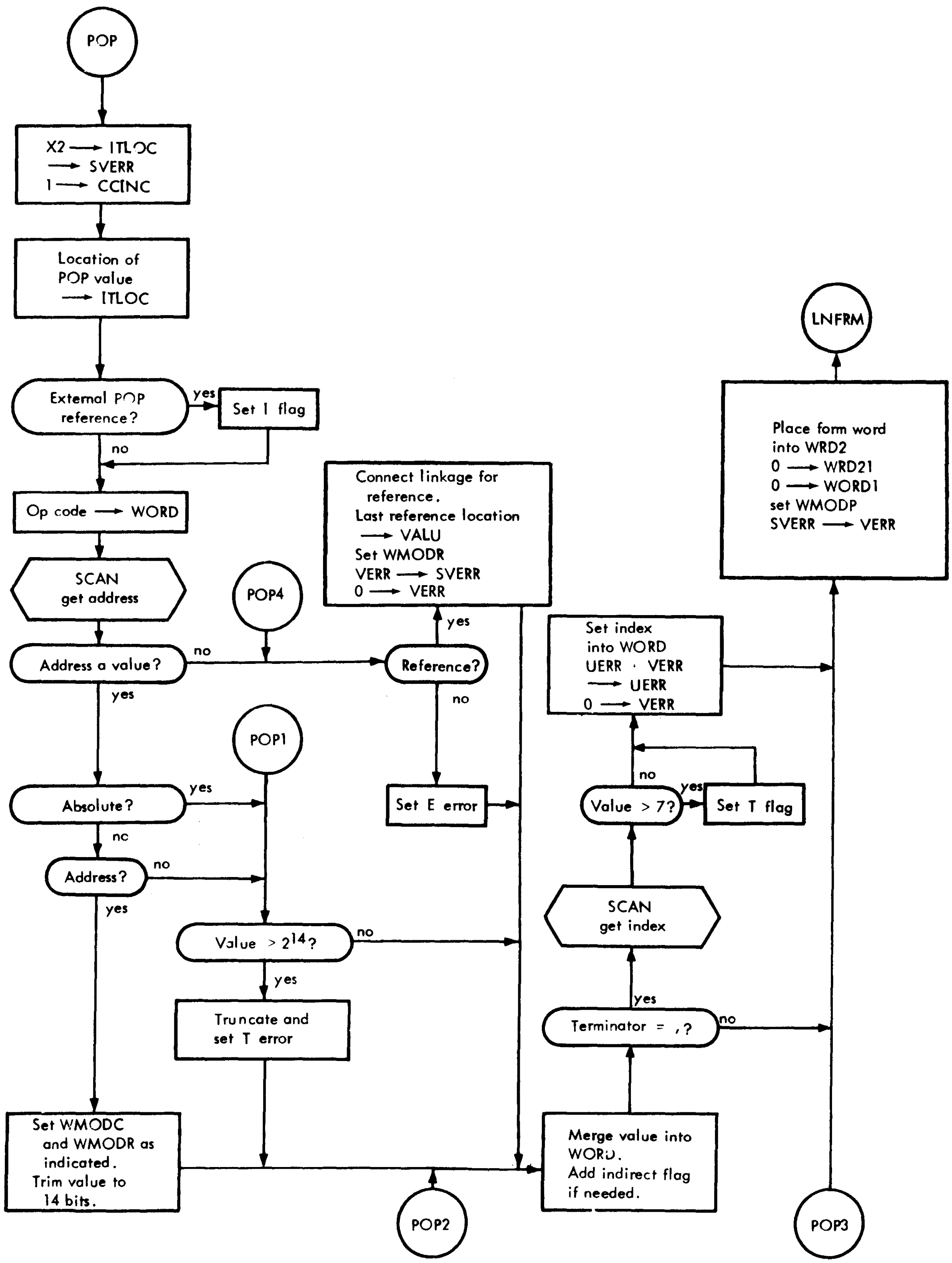

PRL7
LBL 0?
yes
no
Mode of ICW → mode of LBL
VALU → LBL + 1
Error flag on NAME set?
yes
Set U error
no
LINSYM
Is there a zero element?
no
yes
SKIP
to end of PROC line
PRL2A
Is element zero a list?
no
021000000 → LBL + 2
yes
023000000 → LBL + 2
0 → LBL
DFLST
Zero element value → LBL + 3
item length + 2 → item length field of LBL
NSRT
define proc list item
0 → FST
yes
Terminator , ?
no
Set ICW to 2 word, type 3 item
FST → VALU
SCAN
get 1st element
Increment element count in FST
EXIT
Set element flag in ICW
0 → to ICW associate
Location of element → * LNK
Loc of element → FST
yes
NSRT
define element
FST = 0?
no
Location of element → * LNK
GO1

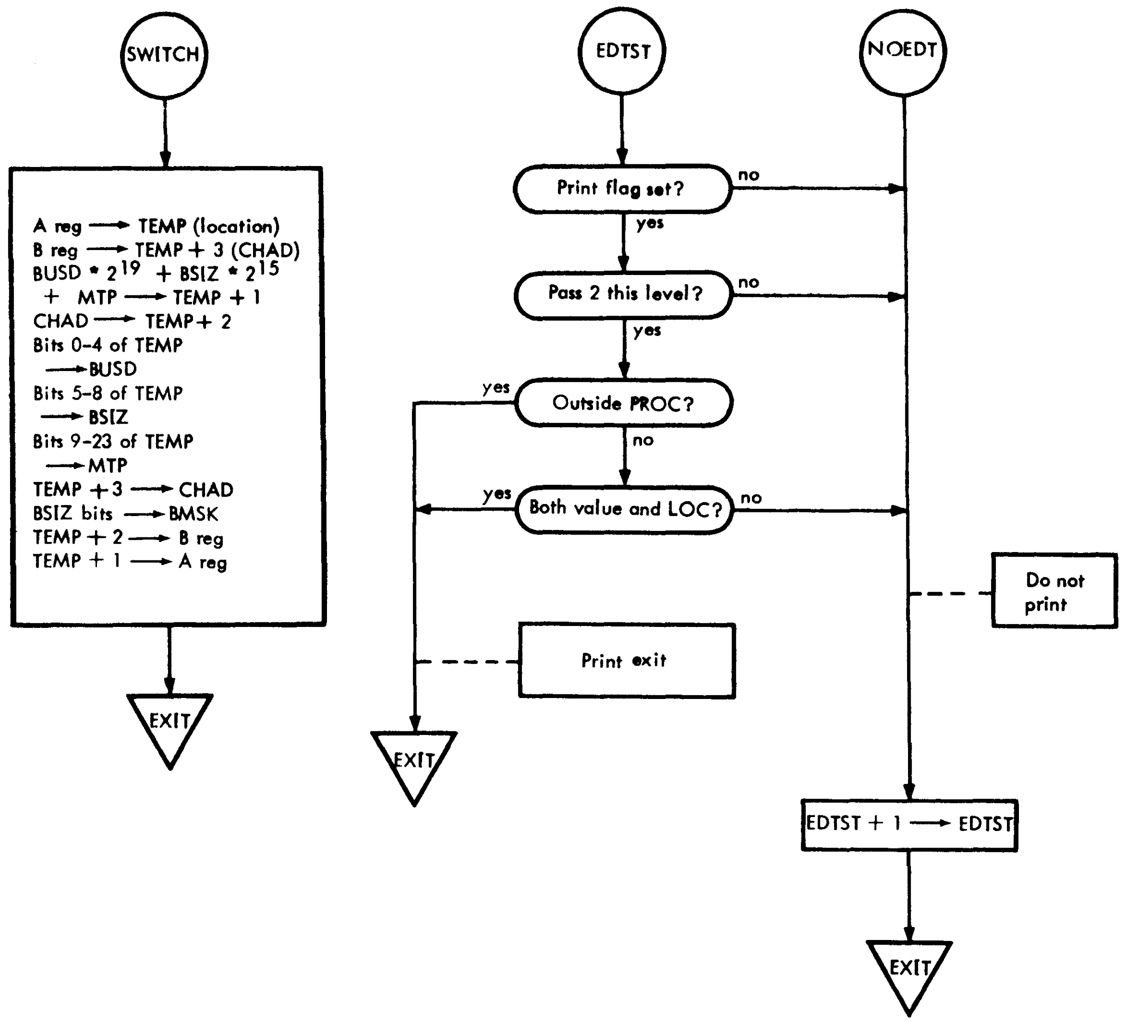

ENDP
END
END of PROC
yes
no
End of FUNC?
yes
no
PLV-CPINC → WLLVL
End of sample?
yes
no
Is there a label?
no
yes
SCAN
evaluate end
Load
FINISH
ENDS
NSRT
define label
LOADER
0 → LBL
SVMTP → MTP2
SWITCH
reset GET
parameters
PRCNT-1 → PRCNT
no
PRCNT = 0?
yes
SCRP
purge Symbol Table
-1 → SMPFG
PRBYT → BYT
LPLV → PLV
PRECW → ECW
PTERM → TERM
PLUT-CPINC → PLUT
LINSYM
End of FUNC?
yes
no
CCVAL → CCINC
LBTST
define waiting
label
SCANR
LNE
no
LLC = LC ?
yes
LINSYM

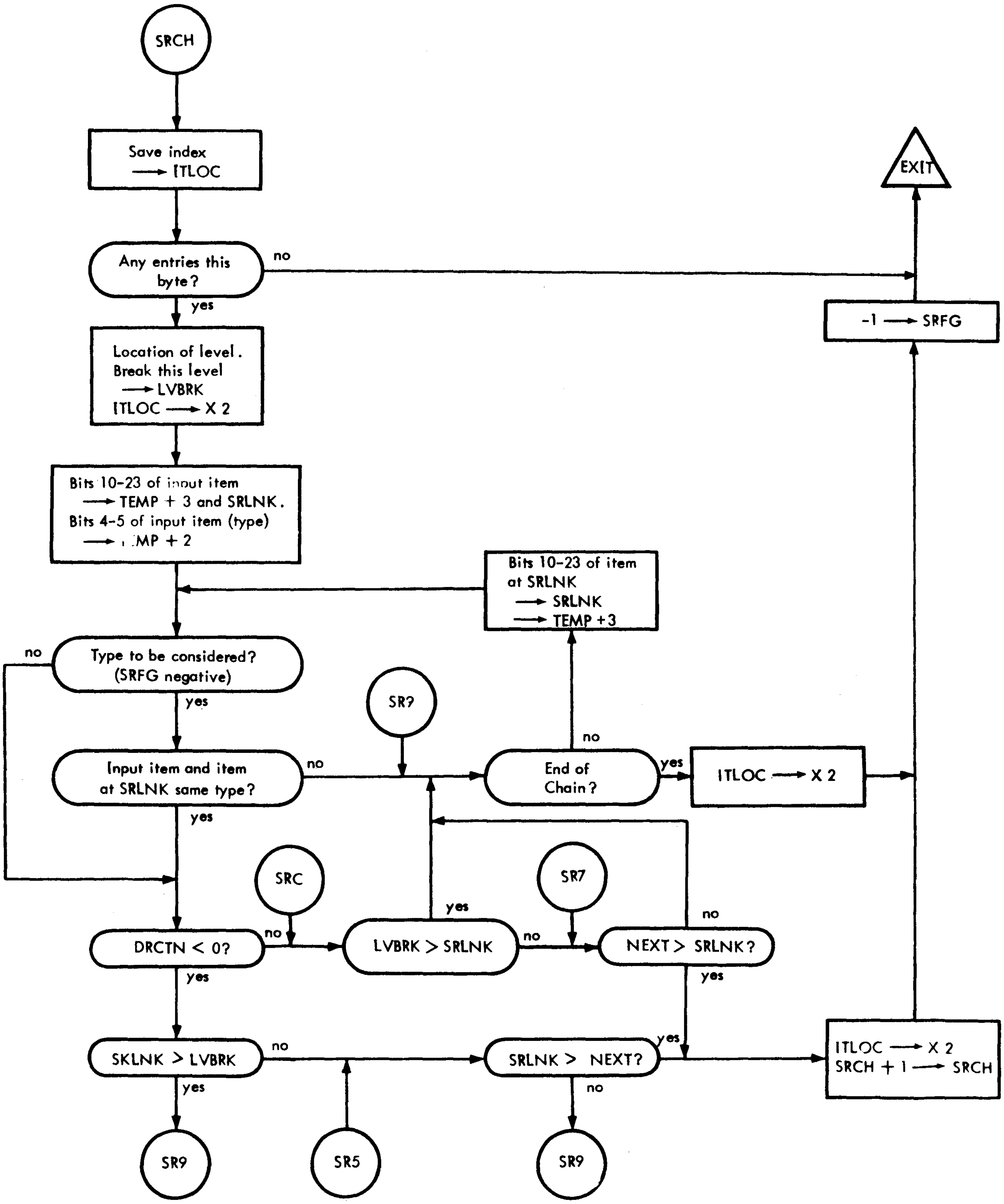

FRL
X2 → ITLOC
CCINC + 1 → CCINC
0 → FRLCNT
0 → EXPCT
2 word form?
yes
no
FRL1
-1 → DPPF
CCINC + 1 → CCINC
High form word → WRD2
Low word → WRD21
and DATA + 1
Form Word → WRD21
Form Word → DATA + 1
0 → WRD2
0 → DATA
LNFRM
040000000 → WRD21
0 → WRD2
040000000 → WRD2
0 → WRD21
LNFRM
FRND
no
Double precision?
yes
DATA → WRD21
DATA + 1 → WRD2
FRL5B
yes
EXPCT > 5
no
FRL5
yes
BITSS = 0?
no
Set E flag
no
yes
Terminator = , ?
LNFRM
yes
1st pass?
no
0 → WORD
0 → WORD1
Left adjust form
words in WRD2 and
WRD21
Form size → BITSS
Add value
to WORD
FRL4C
Truncate
value.
Set T flag.
no
yes
Value > field
allows?
no
Field size > 24
bits?
yes
Extend
sign of
value
FRL4
FRL4B
Value → VALU
SCAN
get next
value
GLOV
get value
FRL5A
End of
form
FLDC
determine
field size
normal
no
Terminator
comma?
yes
Set E error
FRL5
Set WMODR
if relocatable
yes
no
Address value?
FRLCNT + size
→ FRLCNT
EXPCT + 1 → EXPCT
shift WORD and
WORD1 left by
amount of field size
FRL4E
FRL6
no
This field a value?
yes
no
Double precision?
yes
FRL4A

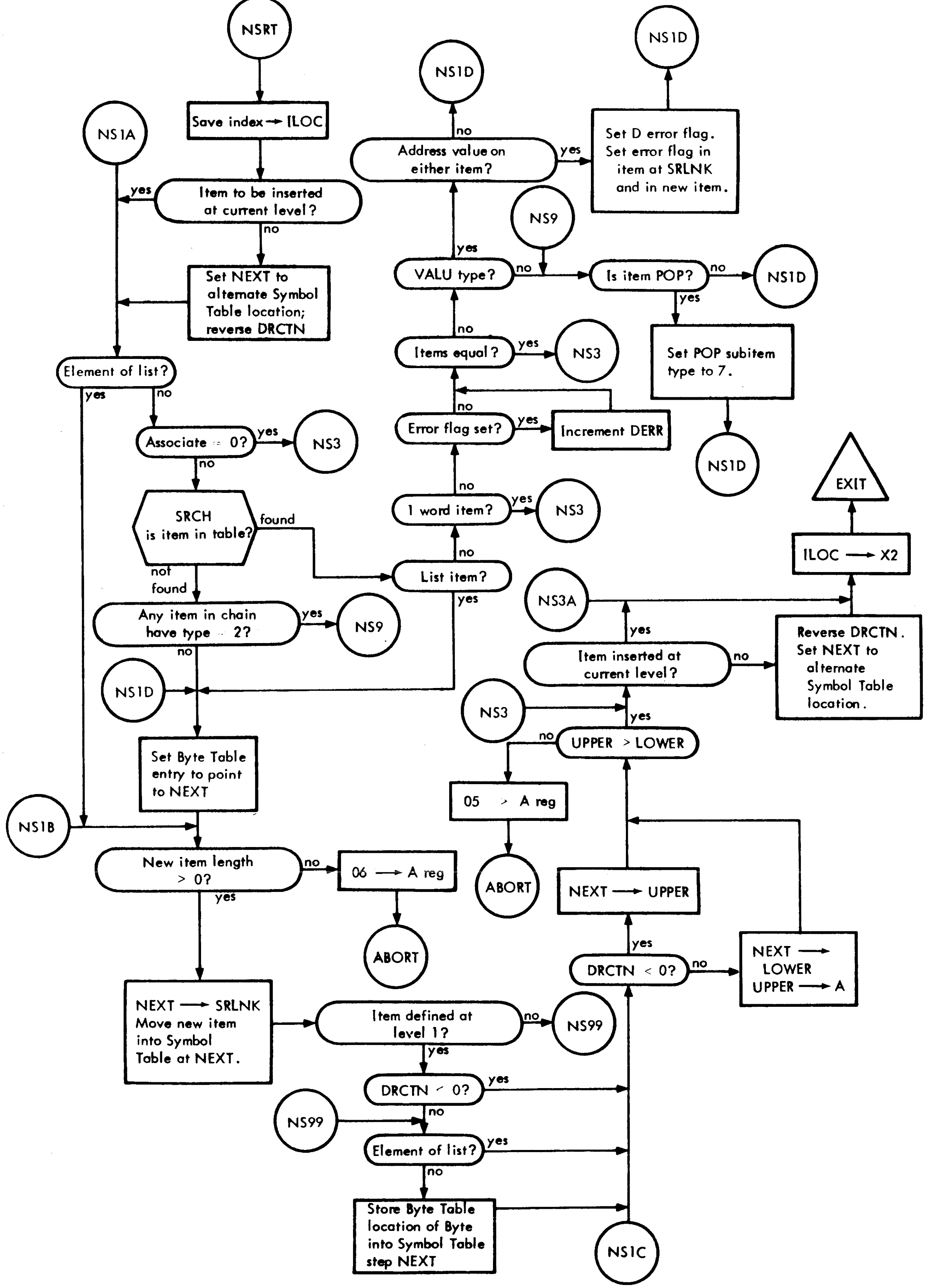

FRL4A
Floating point?
yes
no
MFOI
2 word
floating point
Field size > 24?
yes
no
Add high order value word to WORD1
Overflow?
no
yes
Set T error flag
FRL4C
High word = 0?
yes
no
Set T error flag
FRL4B
FRL6
FRERR
Type zero?
no
yes
Set E error flag
FRL5B
FRL8
Address mode?
no
yes
Set U error flag
0 → V flag
0 → VALU
FRL4E
Field > 13 bits?
no
yes
End of word?
no
yes
FRL8
CC → location given by VALU
Contents of word addressed by value → VALU
Relocatable?
no
yes
FRL4E
Set WMODR

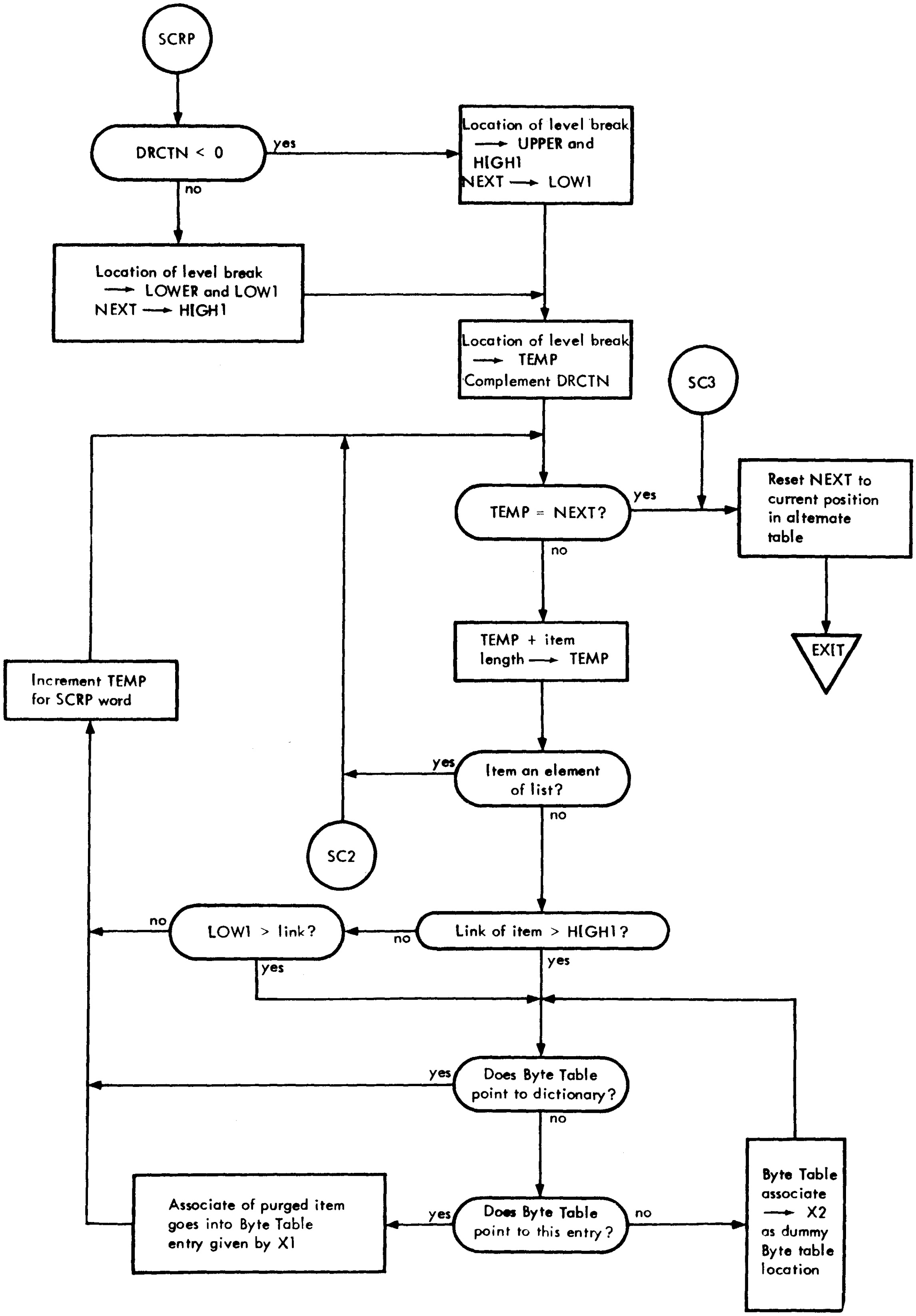

TEXTR
BCD
060 → SPC
012 → SPC
0100 → TEXTC
PEEK at next character
Character = < ?
yes
no
GCW get next byte
Character → TEXFG
SCAN get character count
> → TEXTC
56 → A reg
-1 → TEXFG
Count > 56?
no
yes
Set T error → A reg
TEXT3
A reg-1 → CNTT
Next byte blanks?
GBSL count blanks
Full word?
EDIT print word
Set FORM control word
CNTI-1 → CNTT
GET next character
Blank?
SPC → A reg
CC + 1 → CC
SPC → A reg
-1 → CNTT
TEXT2
LNE
LBTST define label
CNTT < 0?
Character TEXTC?
Blanks → WORD
SPC → A reg
A reg = 060?
SPC → A reg
Character blank?
GET next character
TEXT3
Position and store character into WORD
060 → A reg
CNTT < 0?

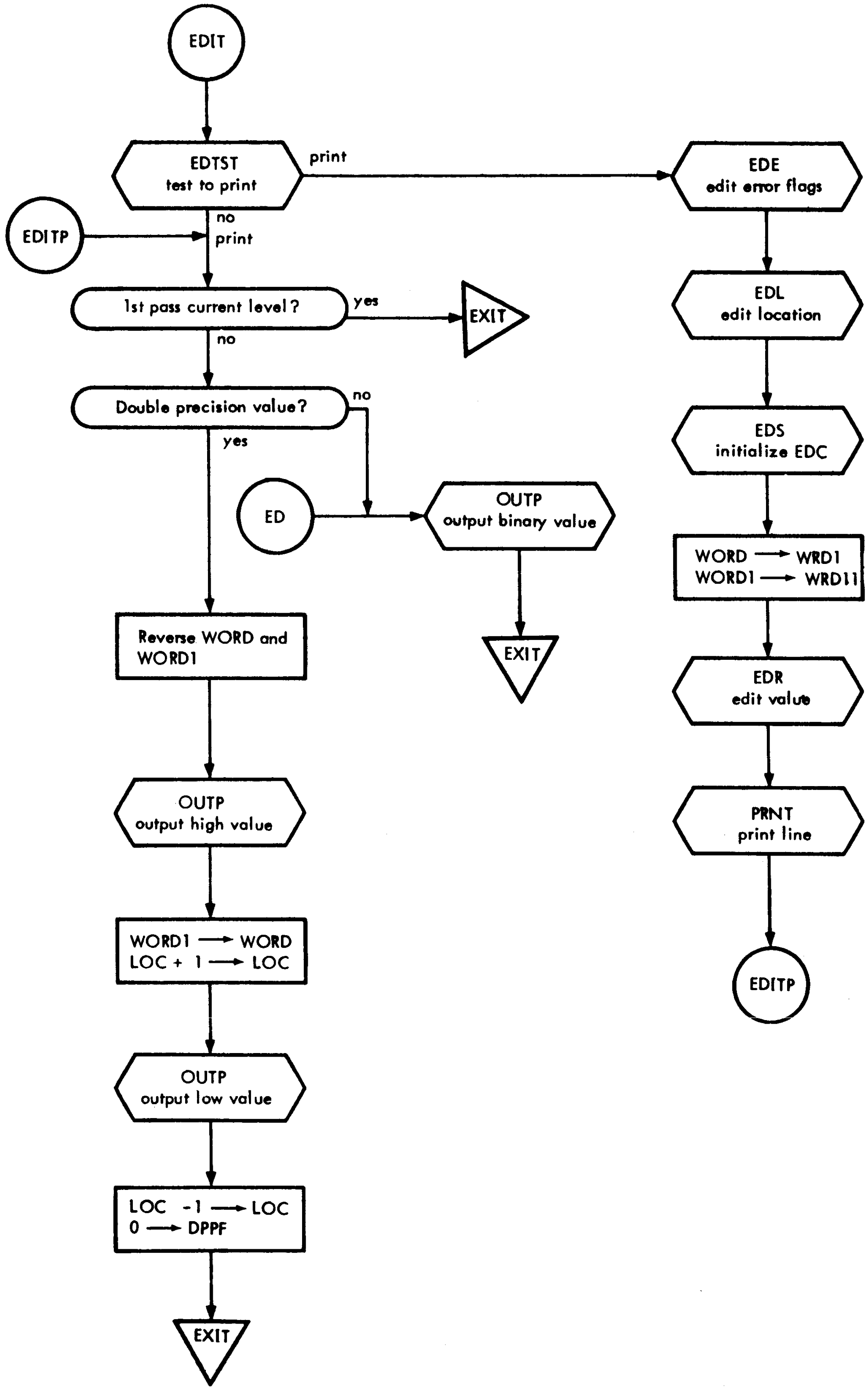

POP
X2 → ITLOC
→ SVERR
1 → CCINC
Location of POP value → ITLOC
External POP reference?
yes
Set I flag
no
Op code → WORD
SCAN get address
Address a value?
no
yes
POP4
Reference?
yes
no
Connect linkage for reference.
Last reference location → VALU
Set WMODR
VERR → SVERR
0 → VERR
Set E error
POP1
Absolute?
yes
no
Address?
no
yes
Value > 2^14?
no
yes
Truncate and set T error
Set WMODC and WMODR as indicated.
Trim value to 14 bits.
POP2
Merge value into WORD.
Add indirect flag if needed.
Terminator = ,?
yes
no
SCAN get index
Value > 7?
yes
no
Set T flag
Set index into WORD
UERR · VERR → UERR
0 → VERR
Place form word into WRD2
0 → WRD21
0 → WORD1
set WMODP
SVERR → VERR
LNFRM
POP3

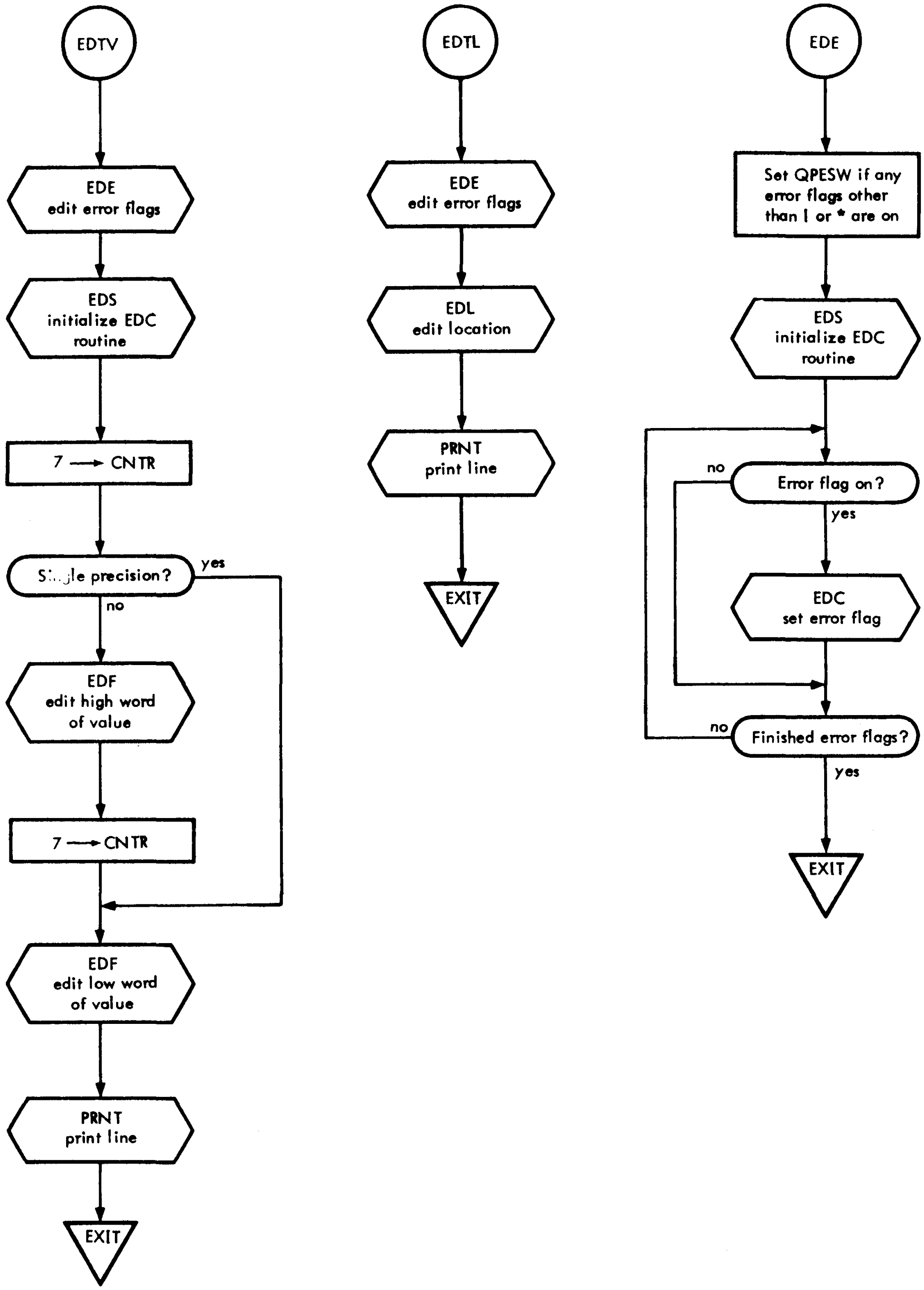

SWITCH
A reg → TEMP (location)
B reg → TEMP + 3 (CHAD)
BUSD * 2^19 + BSIZ * 2^15 + MTP → TEMP + 1
CHAD → TEMP + 2
Bits 0-4 of TEMP → BUSD
Bits 5-8 of TEMP → BSIZ
Bits 9-23 of TEMP → MTP
TEMP + 3 → CHAD
BSIZ bits → BMSK
TEMP + 2 → B reg
TEMP + 1 → A reg
EXIT
EDTST
Print flag set?
no
yes
Pass 2 this level?
no
yes
Outside PROC?
yes
no
Both value and LOC?
yes
no
Print exit
EXIT
NOEDT
Do not print
EDTST + 1 → EDTST
EXIT


SRCH
Save index
→ ITLOC
Any entries this byte?
no
yes
EXIT
-1 → SRFG
Location of level. Break this level
→ LVBRK
ITLOC → X 2
Bits 10-23 of input item
→ TEMP + 3 and SRLNK.
Bits 4-5 of input item (type)
→ TEMP + 2
Bits 10-23 of item at SRLNK
→ SRLNK
→ TEMP +3
Type to be considered? (SRFG negative)
SR?
Input item and item at SRLNK same type?
End of Chain?
ITLOC → X 2
SRC
SR7
DRCTN < 0?
LVBRK > SRLNK
NEXT > SRLNK?
SKLNK > LVBRK
SRLNK > NEXT?
ITLOC → X 2
SRCH + 1 → SRCH
SR9
SR5
SR9

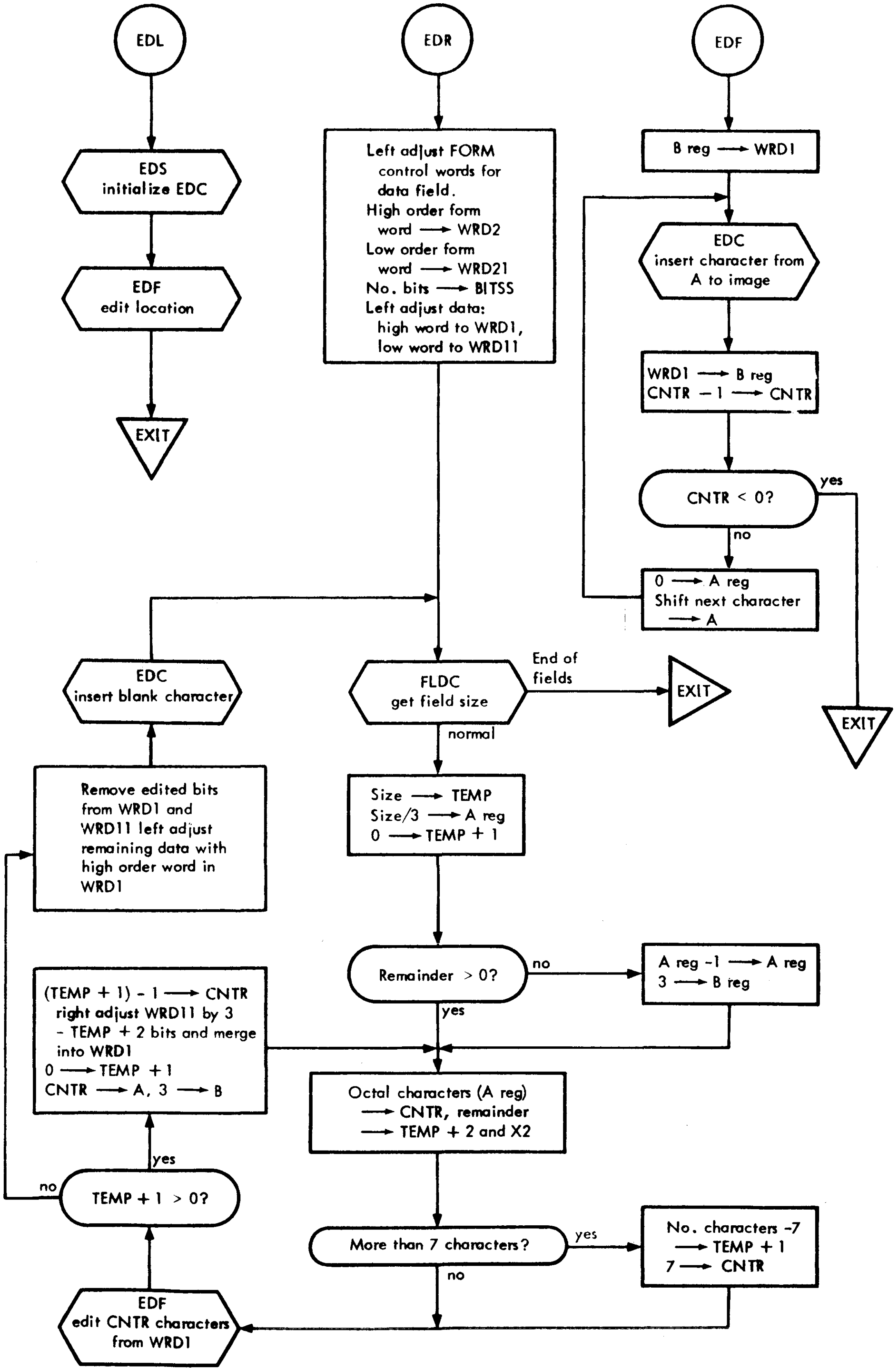

NSRT
Save index → ILOC
NS1A
Item to be inserted at current level?
yes
no
Set NEXT to alternate Symbol Table location; reverse DRCTN
Element of list?
yes
no
Associate = 0?
yes
NS3
no
SRCH is item in table?
found
not found
Any item in chain have type = 2?
yes
NS9
no
NS1D
Set Byte Table entry to point to NEXT
NS1B
New item length > 0?
no
06 → A reg
ABORT
yes
NEXT → SRLNK Move new item into Symbol Table at NEXT.
Item defined at level 1?
no
NS99
yes
DRCTN < 0?
yes
no
NS99
Element of list?
yes
no
Store Byte Table location of Byte into Symbol Table step NEXT
NS1C
NS1D
Address value on either item?
no
yes
Set D error flag. Set error flag in item at SRLNK and in new item.
NS1D
NS9
VALU type?
yes
no
Is item POP?
no
NS1D
yes
Set POP subitem type to 7.
NS1D
Items equal?
yes
NS3
no
Error flag set?
yes
Increment DERR
no
1 word item?
yes
NS3
no
List item?
yes
EXIT
ILOC → X2
NS3A
yes
Item inserted at current level?
no
Reverse DRCTN. Set NEXT to alternate Symbol Table location.
NS3
yes
UPPER > LOWER
no
05 → A reg
ABORT
NEXT → UPPER
yes
DRCTN < 0?
no
NEXT → LOWER
UPPER → A


SCRP
DRCTN < 0
yes
no
Location of level break → UPPER and HIGH1
NEXT → LOW1
Location of level break → LOWER and LOW1
NEXT → HIGH1
Location of level break → TEMP
Complement DRCTN
SC3
TEMP = NEXT?
yes
no
Reset NEXT to current position in alternate table
EXIT
TEMP + item length → TEMP
Increment TEMP for SCRP word
Item an element of list?
yes
no
SC2
LOW1 > link?
no
yes
Link of item > HIGH1?
no
yes
Does Byte Table point to dictionary?
yes
no
Byte Table associate → X2 as dummy Byte table location
Associate of purged item goes into Byte Table entry given by X1
Does Byte Table point to this entry?
yes
no

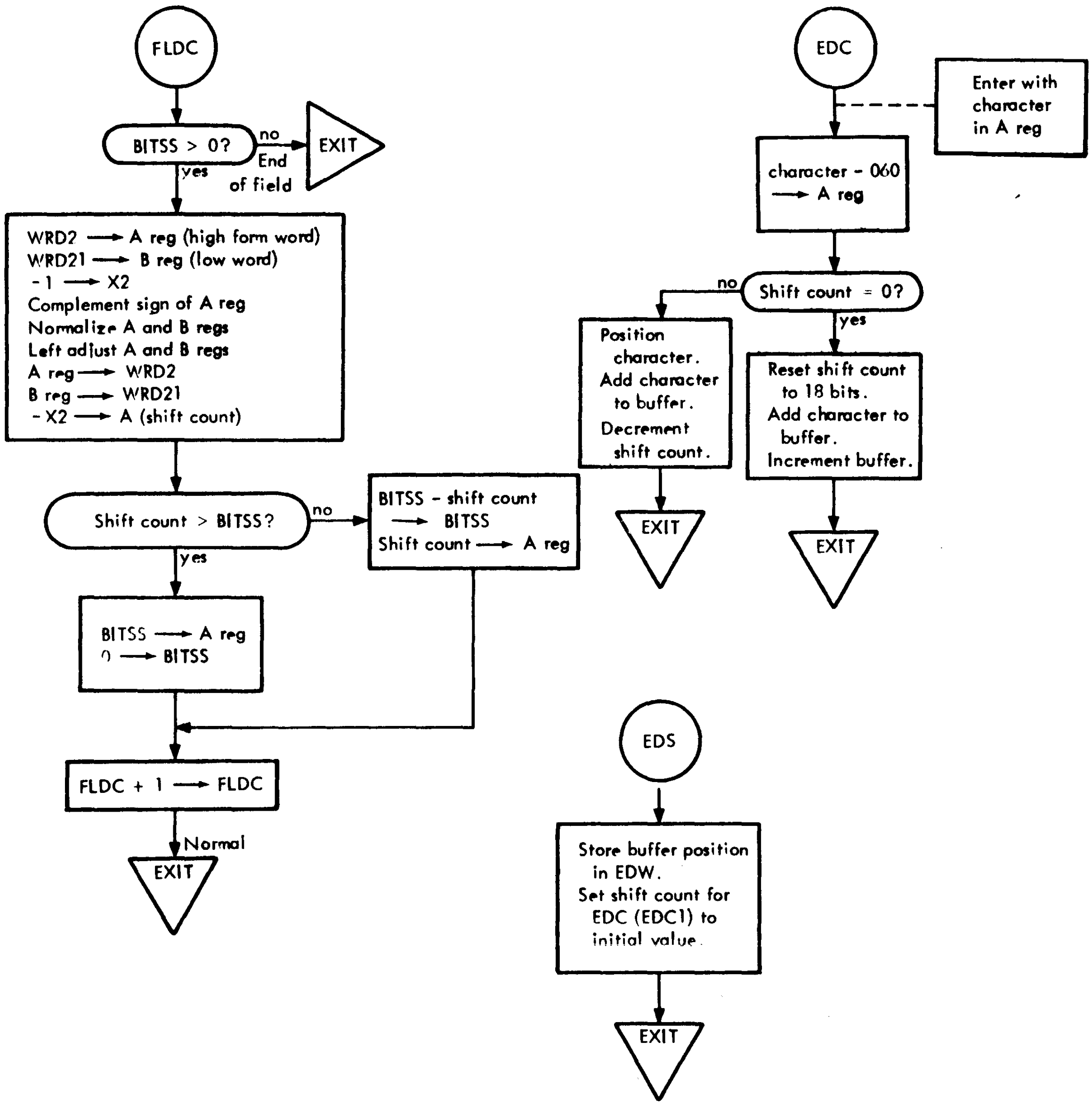

EDIT
EDTST
test to print
print
no
print
EDITP
1st pass current level?
yes
EXIT
no
Double precision value?
no
yes
ED
OUTP
output binary value
EXIT
Reverse WORD and WORD1
OUTP
output high value
WORD1 → WORD
LOC + 1 → LOC
OUTP
output low value
LOC -1 → LOC
0 → DPPF
EXIT
EDE
edit error flags
EDL
edit location
EDS
initialize EDC
WORD → WRD1
WORD1 → WRD11
EDR
edit value
PRNT
print line
EDITP

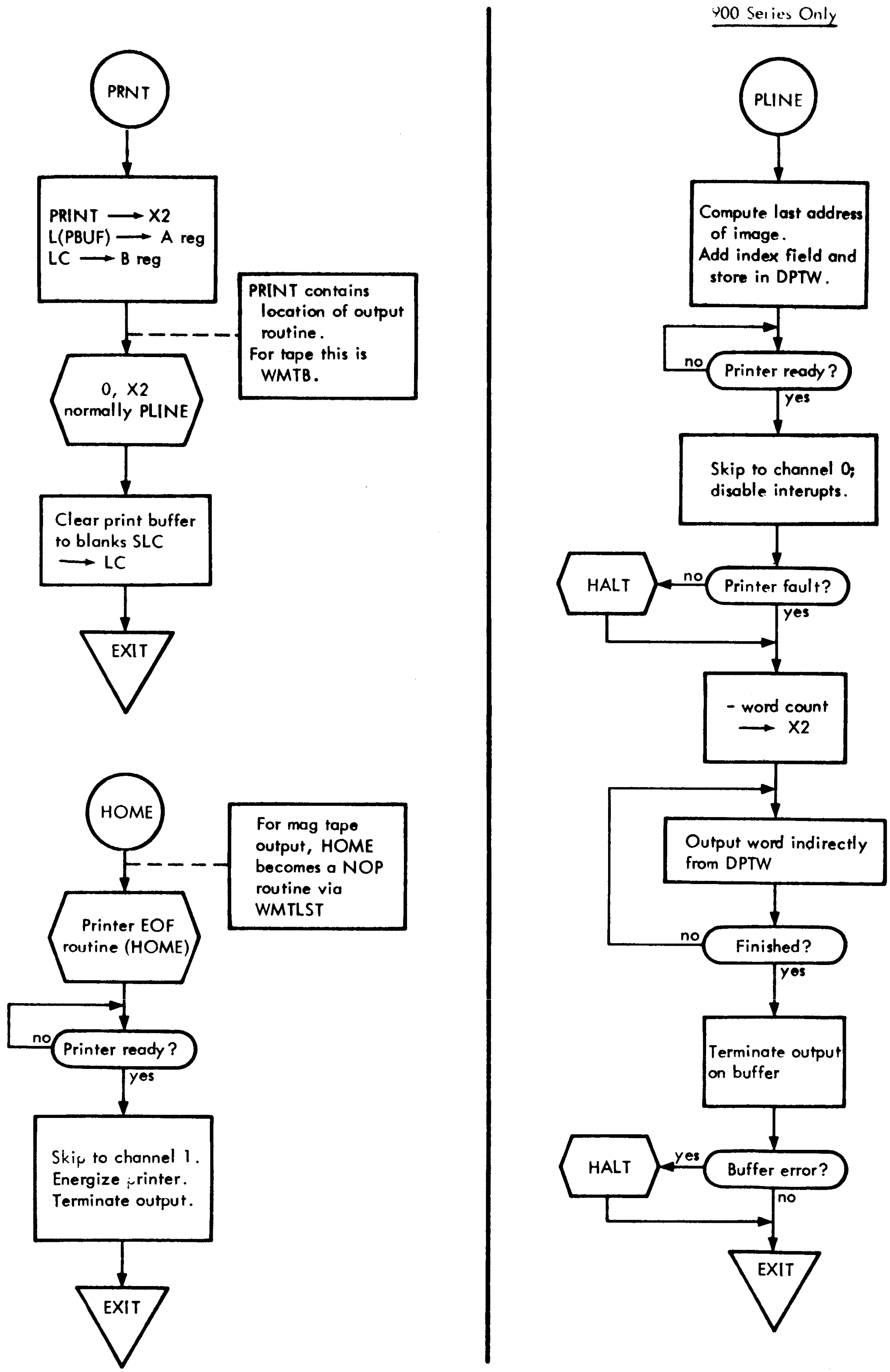

EDTV
EDE
edit error flags
EDS
initialize EDC
routine
7 → CNTR
Single precision?
yes
no
EDF
edit high word
of value
7 → CNTR
EDF
edit low word
of value
PRNT
print line
EXIT
EDTL
EDE
edit error flags
EDL
edit location
PRNT
print line
EXIT
EDE
Set QPESW if any
error flags other
than I or * are on
EDS
initialize EDC
routine
Error flag on?
no
yes
EDC
set error flag
Finished error flags?
no
yes
EXIT

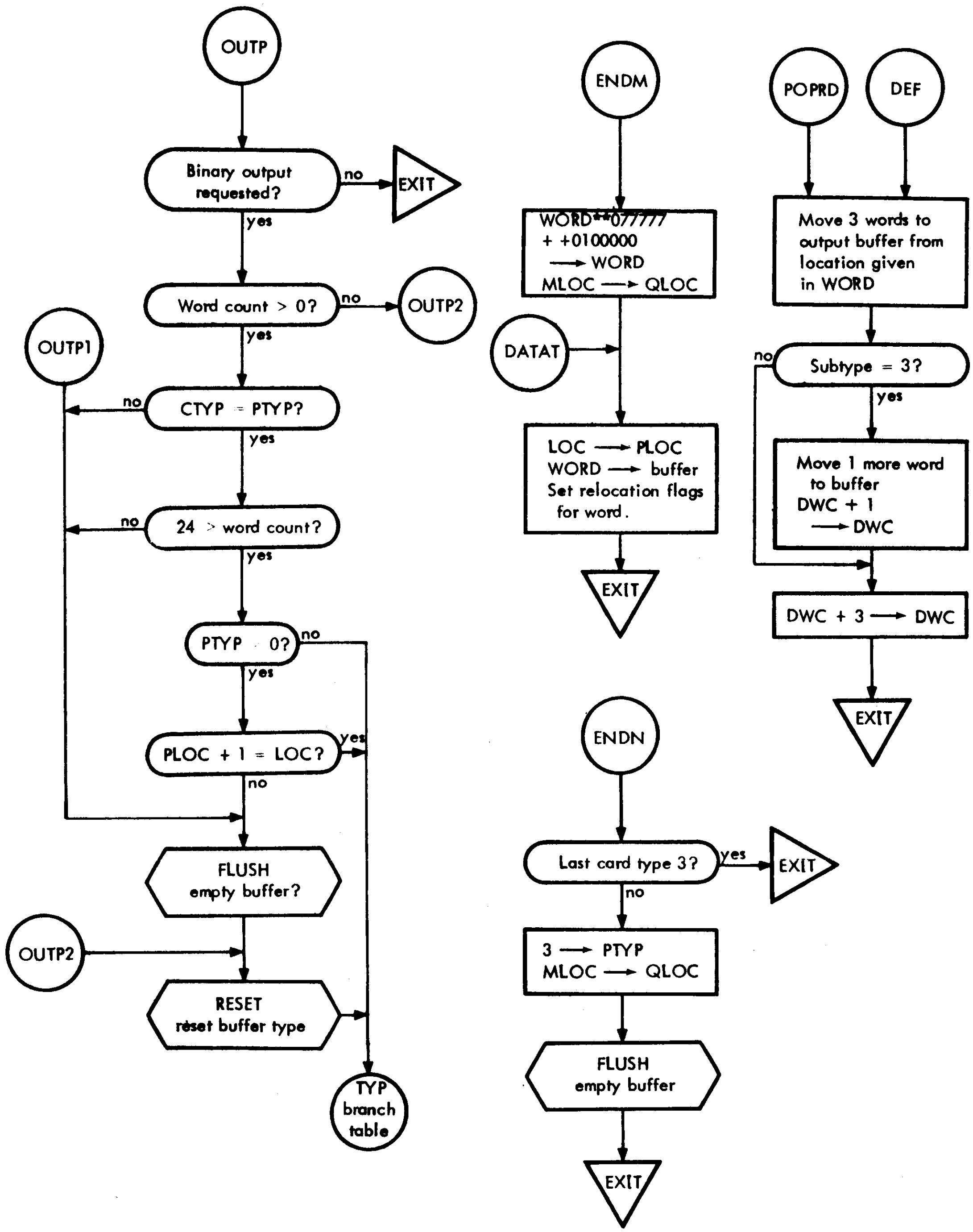
EDL, EDR AND EDF ROUTINES

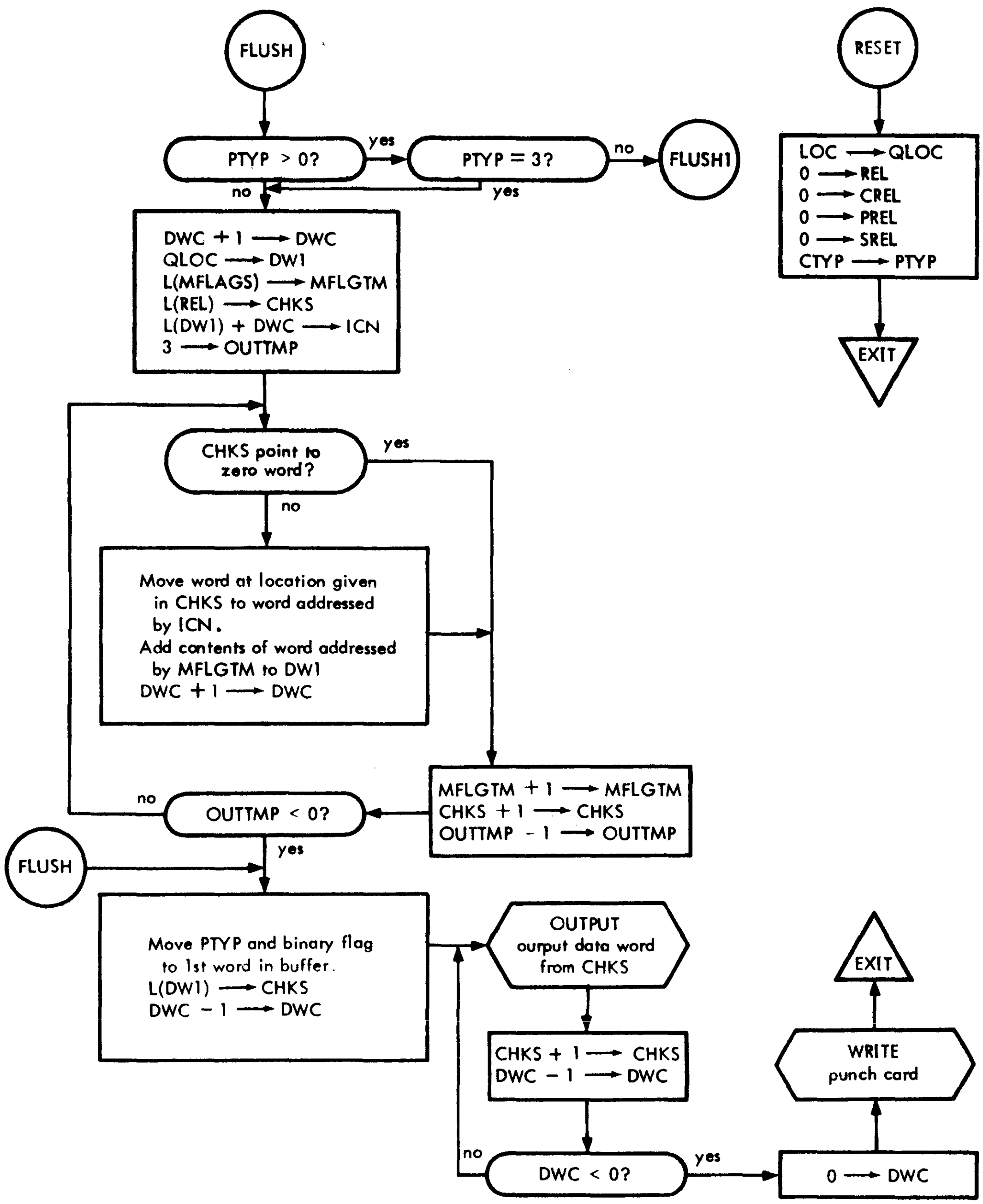


FLDC, EDC AND EDS ROUTINES

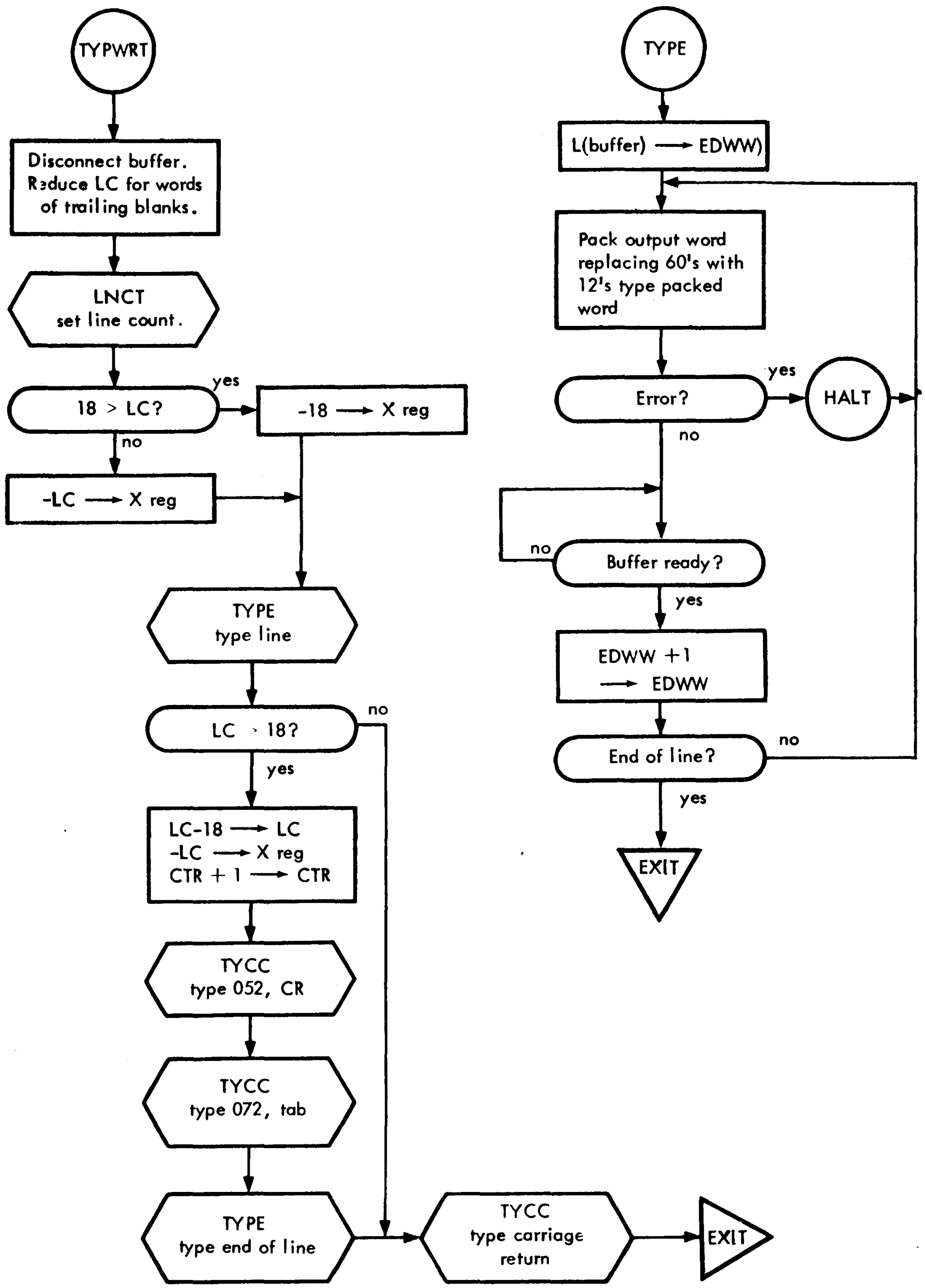



900 Series Only
PRNT
PRINT → X2
L(PBUF) → A reg
LC → B reg
PRINT contains location of output routine. For tape this is WMTB.
0, X2
normally PLINE
Clear print buffer to blanks SLC → LC
EXIT
HOME
For mag tape output, HOME becomes a NOP routine via WMTLST
Printer EOF routine (HOME)
Printer ready?
no
yes
Skip to channel 1. Energize printer. Terminate output.
EXIT
PLINE
Compute last address of image. Add index field and store in DPTW.
Printer ready?
no
yes
Skip to channel 0; disable interupts.
HALT
no
Printer fault?
yes
- word count → X2
Output word indirectly from DPTW
no
Finished?
yes
Terminate output on buffer
HALT
yes
Buffer error?
no
EXIT

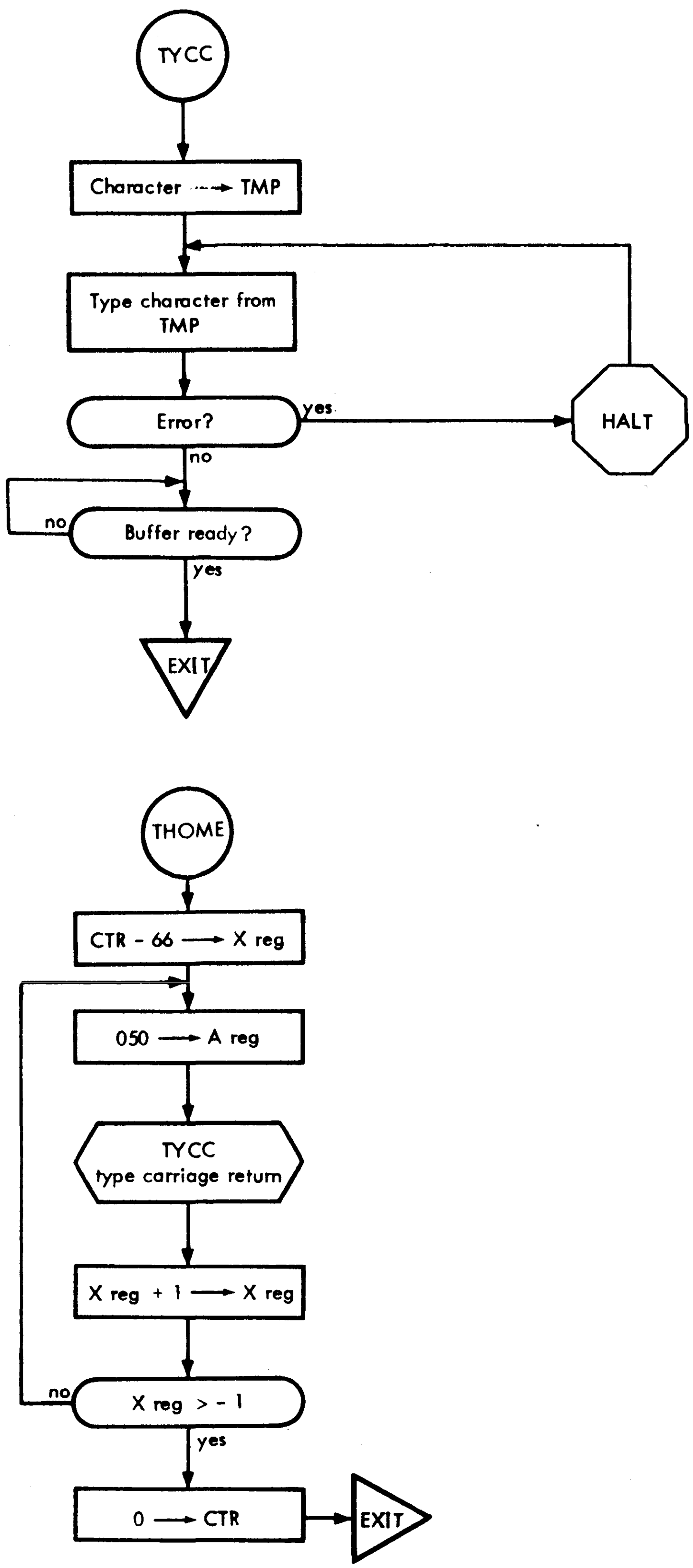

OUTP
Binary output requested?
no
EXIT
yes
Word count > 0?
no
OUTP2
yes
OUTP1
CTYP = PTYP?
no
yes
24 > word count?
no
yes
PTYP 0?
no
yes
PLOC + 1 = LOC?
yes
no
FLUSH empty buffer?
OUTP2
RESET reset buffer type
TYP branch table
ENDM
WORD**077777 + +0100000 → WORD
MLOC → QLOC
DATAT
LOC → PLOC
WORD → buffer
Set relocation flags for word.
EXIT
POPRD
DEF
Move 3 words to output buffer from location given in WORD
Subtype = 3?
no
yes
Move 1 more word to buffer
DWC + 1 → DWC
DWC + 3 → DWC
EXIT
ENDN
Last card type 3?
yes
EXIT
no
3 → PTYP
MLOC → QLOC
FLUSH empty buffer
EXIT

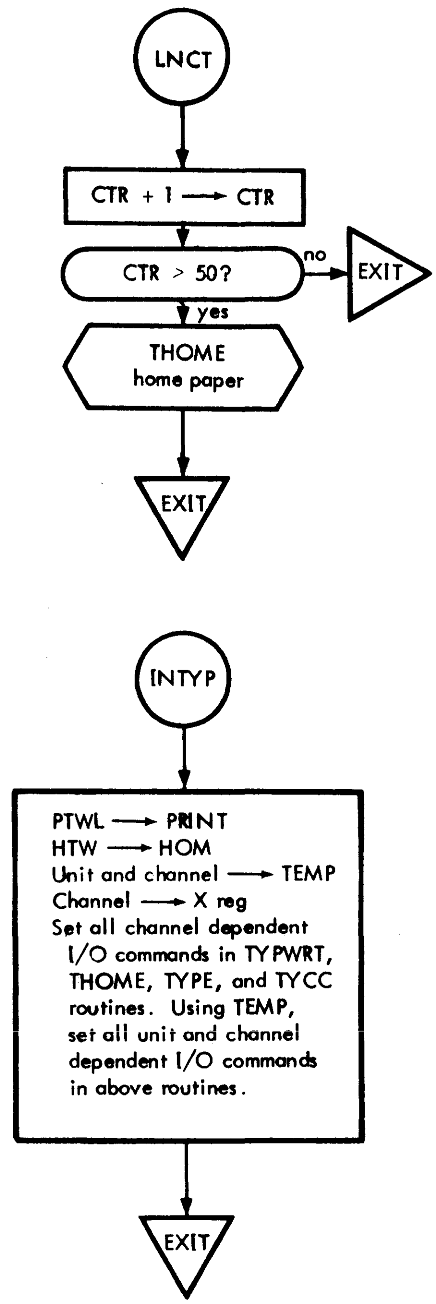

FLUSH
PTYP > 0?
yes
no
PTYP = 3?
no
yes
FLUSH1
DWC + 1 → DWC
QLOC → DW1
L(MFLAGS) → MFLGTM
L(REL) → CHKS
L(DW1) + DWC → ICN
3 → OUTTMP
CHKS point to zero word?
yes
no
Move word at location given in CHKS to word addressed by ICN.
Add contents of word addressed by MFLGTM to DW1
DWC + 1 → DWC
MFLGTM + 1 → MFLGTM
CHKS + 1 → CHKS
OUTTMP - 1 → OUTTMP
OUTTMP < 0?
no
yes
FLUSH
Move PTYP and binary flag to 1st word in buffer.
L(DW1) → CHKS
DWC - 1 → DWC
OUTPUT
output data word from CHKS
CHKS + 1 → CHKS
DWC - 1 → DWC
DWC < 0?
no
yes
0 → DWC
WRITE
punch card
EXIT
RESET
LOC → QLOC
0 → REL
0 → CREL
0 → PREL
0 → SREL
CTYP → PTYP
EXIT

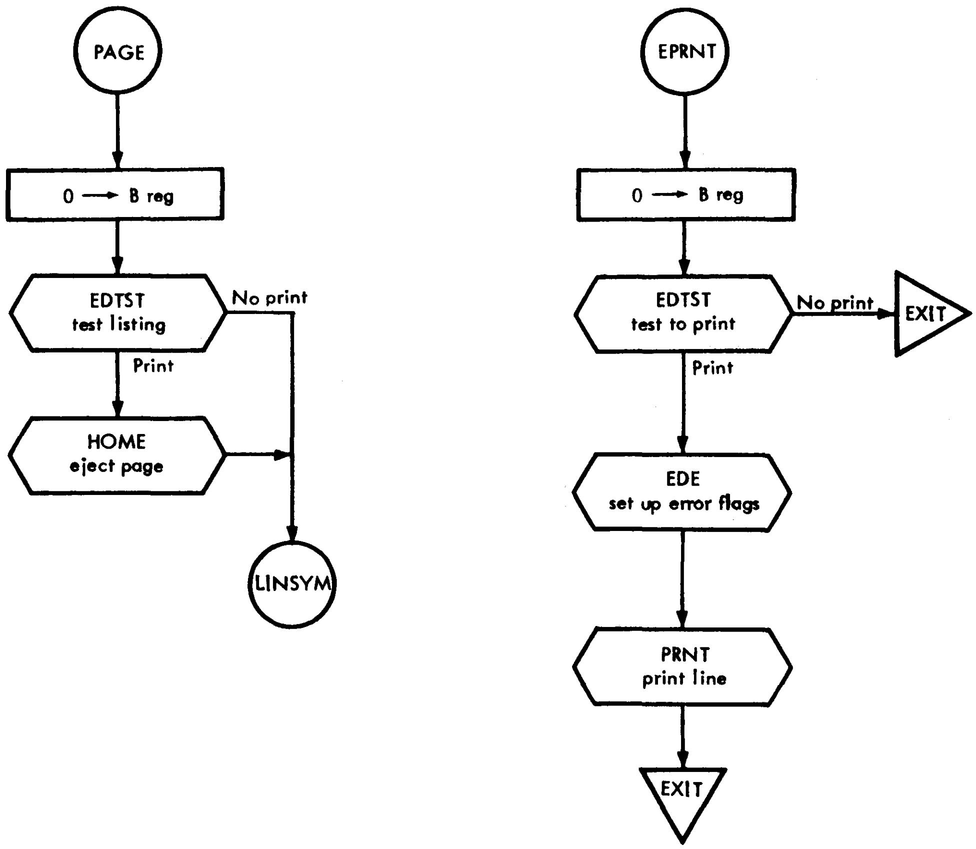
PAS2
TYPWRT AND TYPE ROUTINES

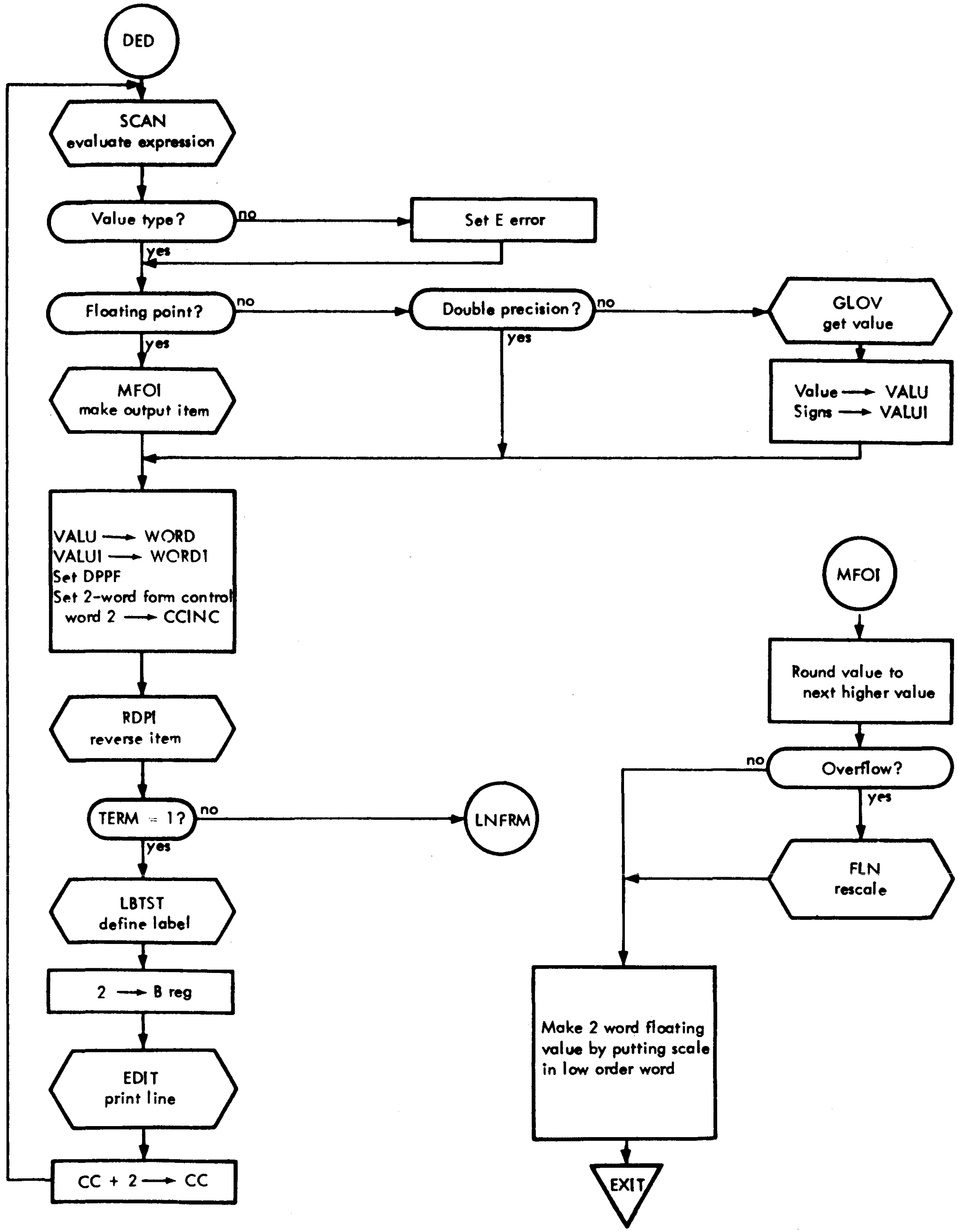


PAS2
INTYP, LNCT, TYCC, AND THOME ROUTINES

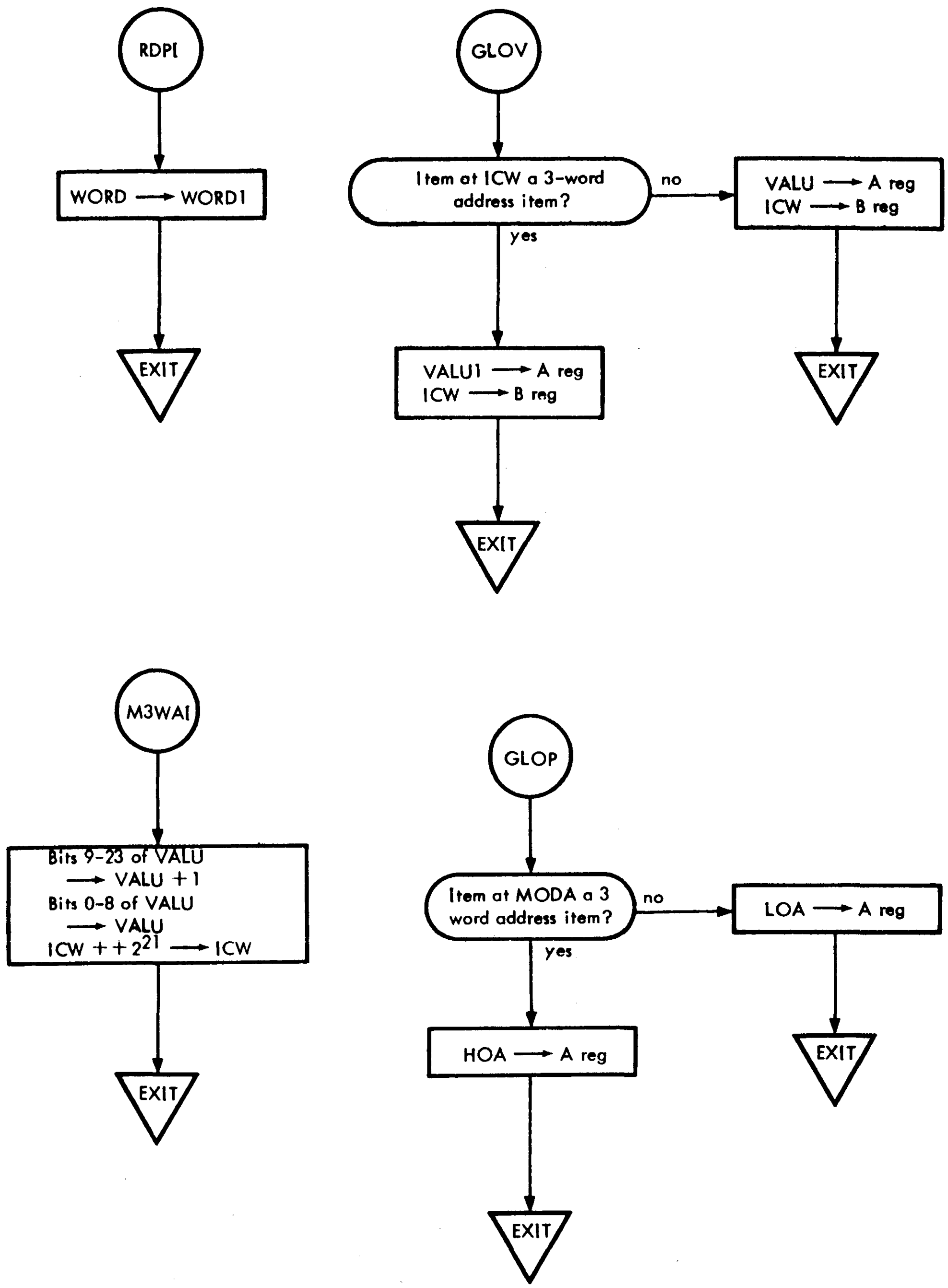


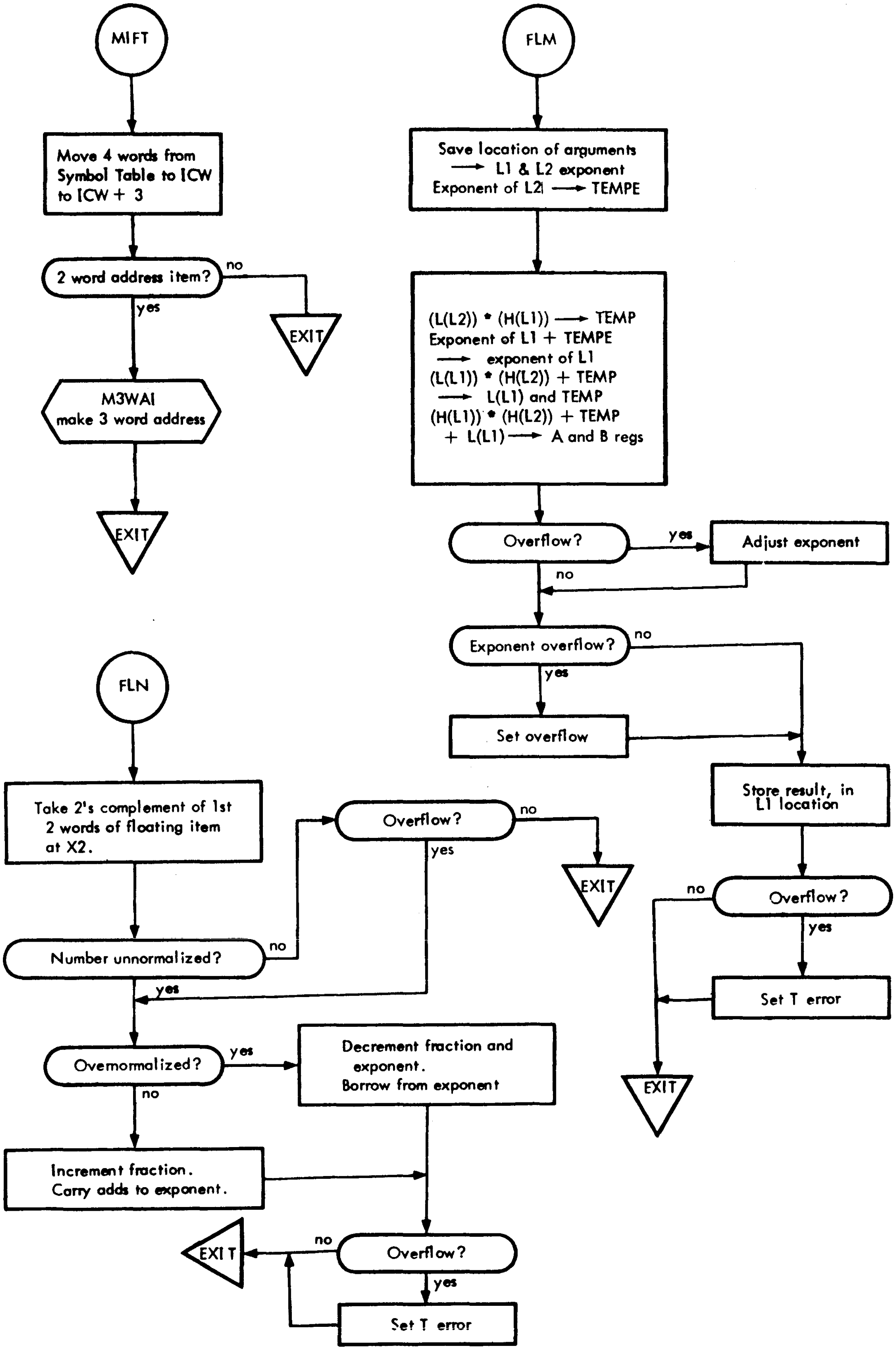



PAGE
0 ⟶ B reg
EDTST
test listing
No print
Print
HOME
eject page
LINSYM
EPRNT
0 ⟶ B reg
EDTST
test to print
No print
EXIT
Print
EDE
set up error flags
PRNT
print line
EXIT

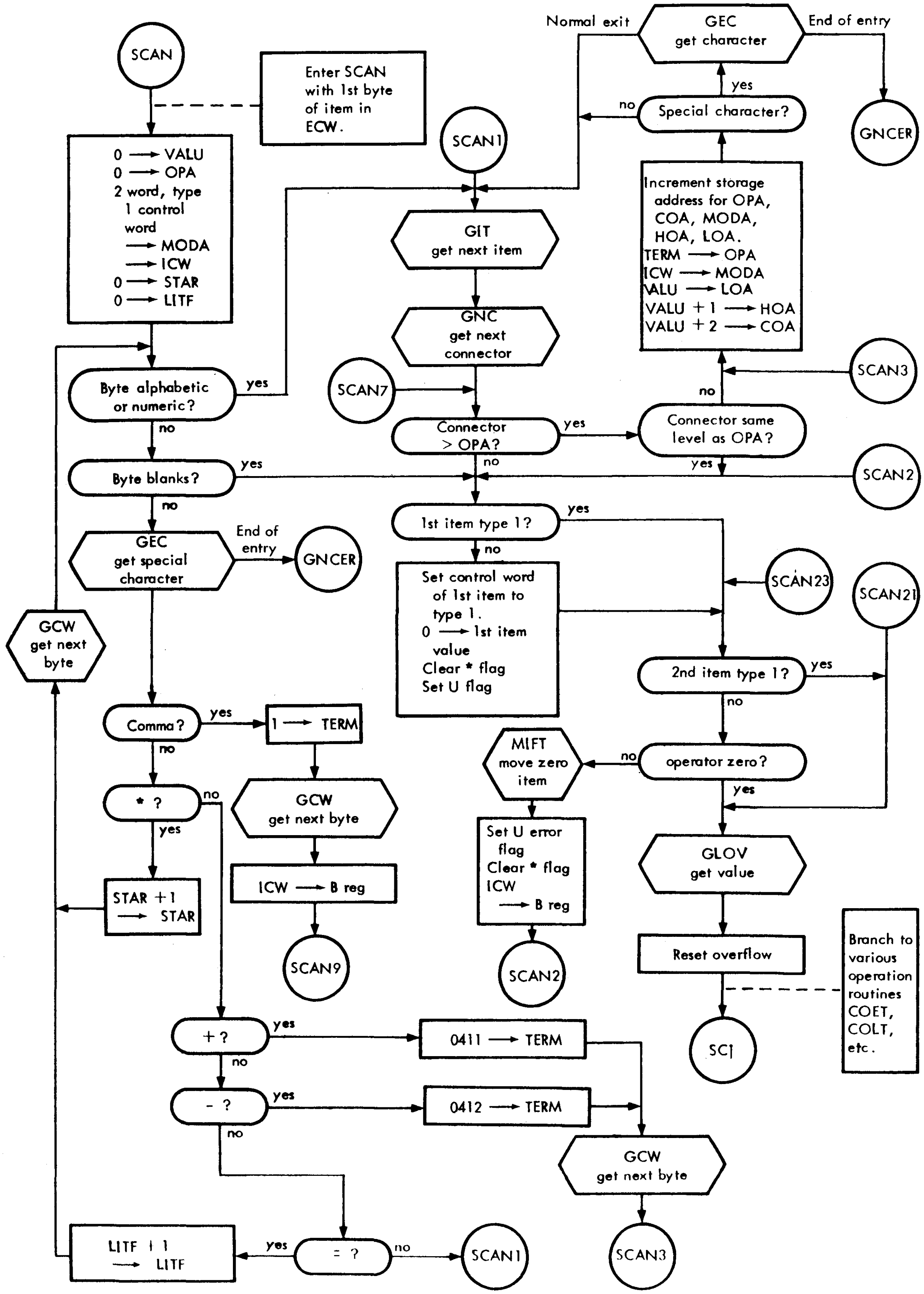

DED
SCAN
evaluate expression
Value type?
no
Set E error
yes
Floating point?
no
Double precision?
no
GLOV
get value
yes
Value → VALU
Signs → VALUI
yes
MFOI
make output item
VALU → WORD
VALUI → WORD1
Set DPPF
Set 2-word form control
word 2 → CCINC
RDPI
reverse item
TERM = 1?
no
LNFRM
yes
LBTST
define label
2 → B reg
EDIT
print line
CC + 2 → CC
MFOI
Round value to
next higher value
Overflow?
no
yes
FLN
rescale
Make 2 word floating
value by putting scale
in low order word
EXIT

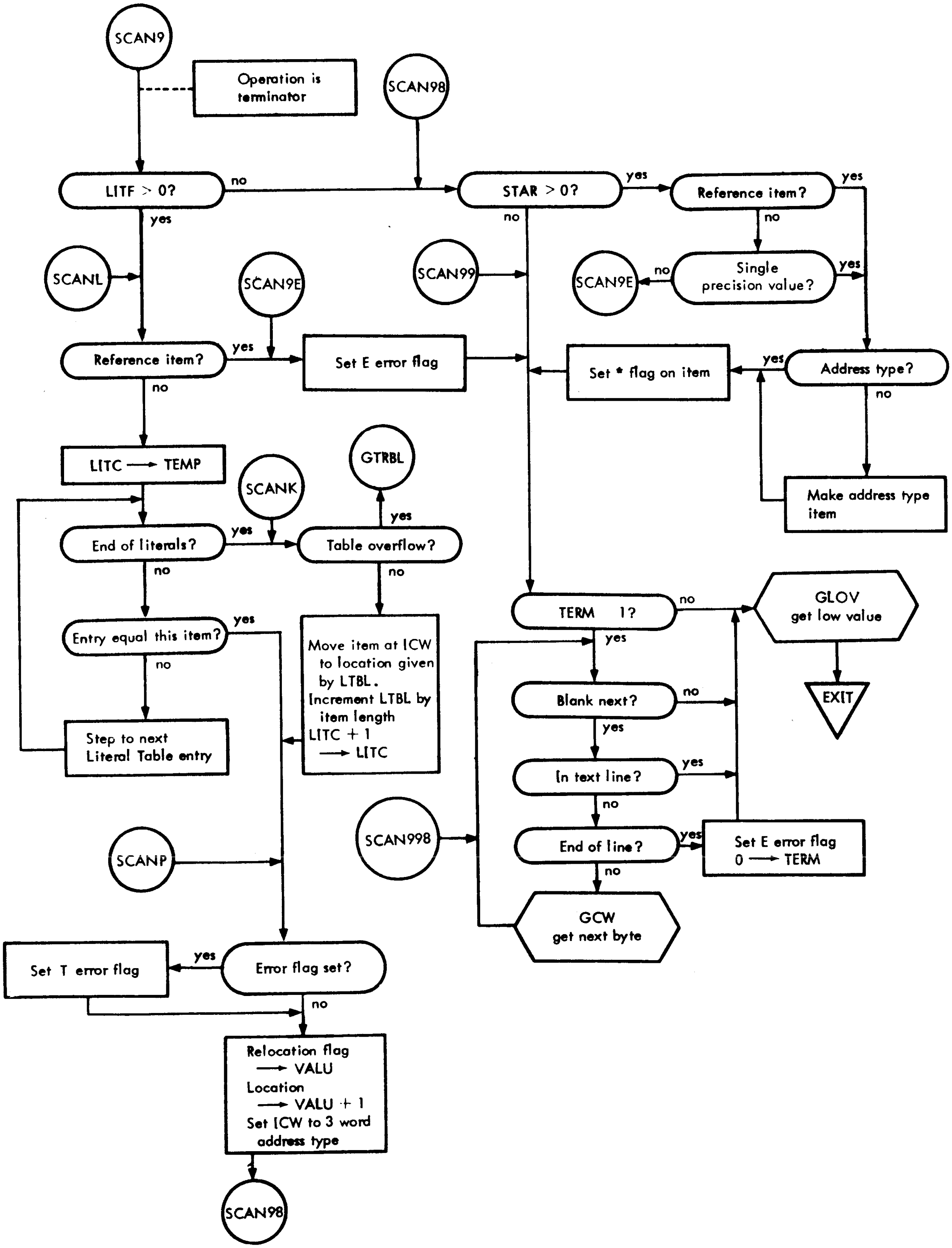
GLOP, RDPI, GLOV, AND M3WAI ROUTINES

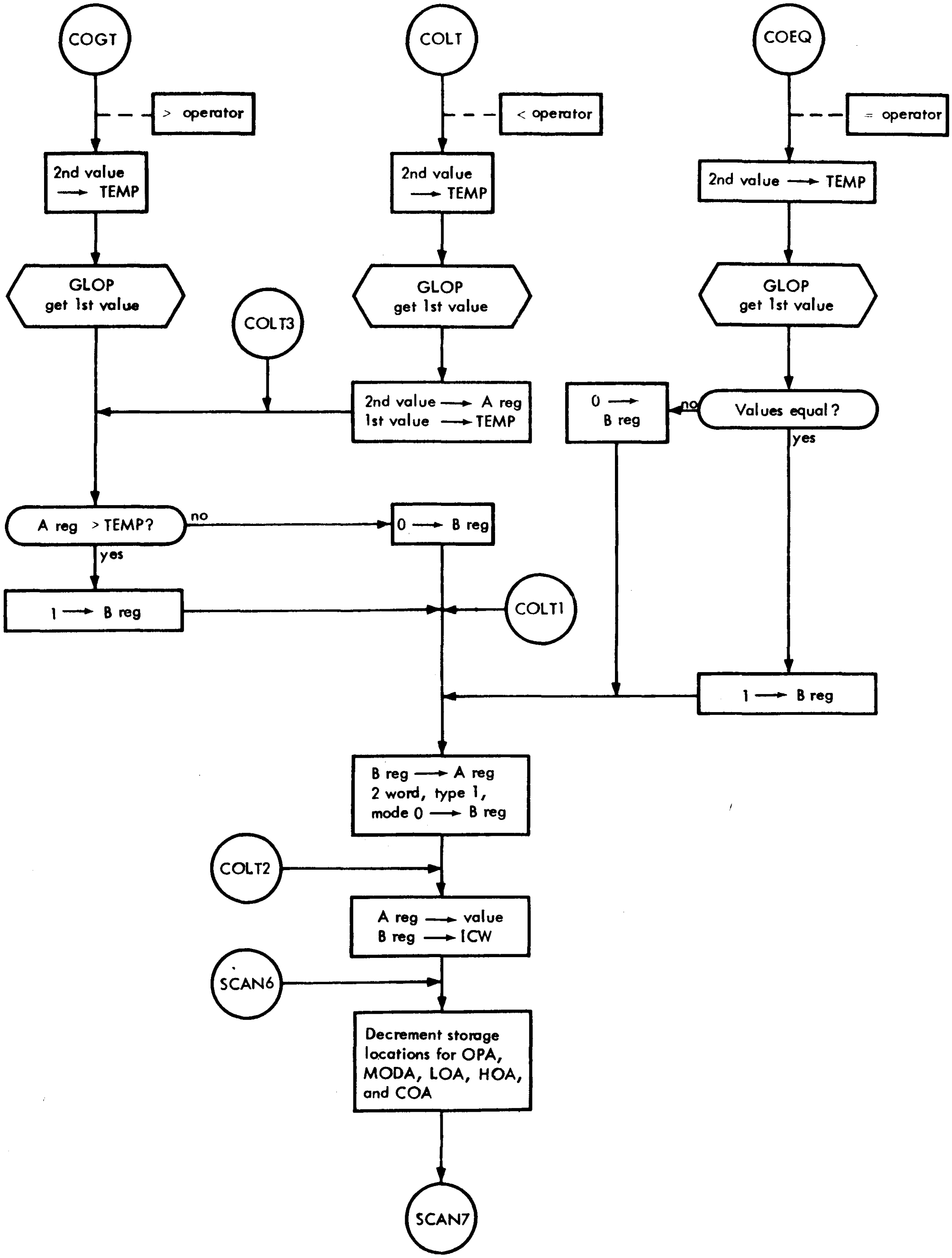


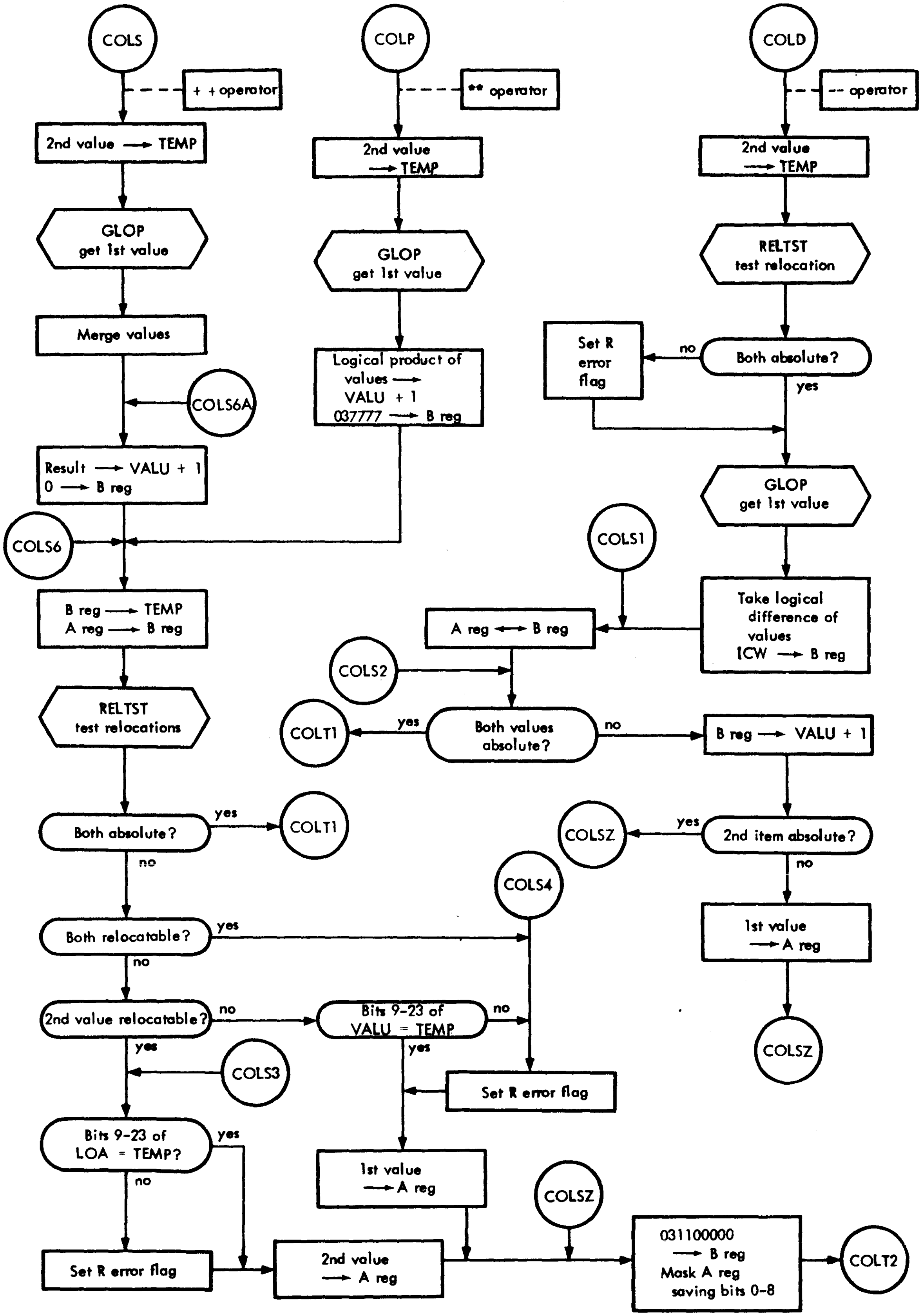

MIFT
Move 4 words from Symbol Table to ICW to ICW + 3
2 word address item?
no
EXIT
yes
M3WAI
make 3 word address
EXIT
FLM
Save location of arguments → L1 & L2 exponent
Exponent of L2 → TEMPE
(L(L2)) * (H(L1)) → TEMP
Exponent of L1 + TEMPE → exponent of L1
(L(L1)) * (H(L2)) + TEMP → L(L1) and TEMP
(H(L1)) * (H(L2)) + TEMP + L(L1) → A and B regs
Overflow?
yes
Adjust exponent
no
Exponent overflow?
no
yes
Set overflow
Store result, in L1 location
Overflow?
no
yes
Set T error
EXIT
FLN
Take 2's complement of 1st 2 words of floating item at X2.
Overflow?
no
EXIT
yes
Number unnormalized?
no
yes
Overnormalized?
yes
Decrement fraction and exponent.
Borrow from exponent
no
Increment fraction.
Carry adds to exponent.
Overflow?
no
EXIT
yes
Set T error

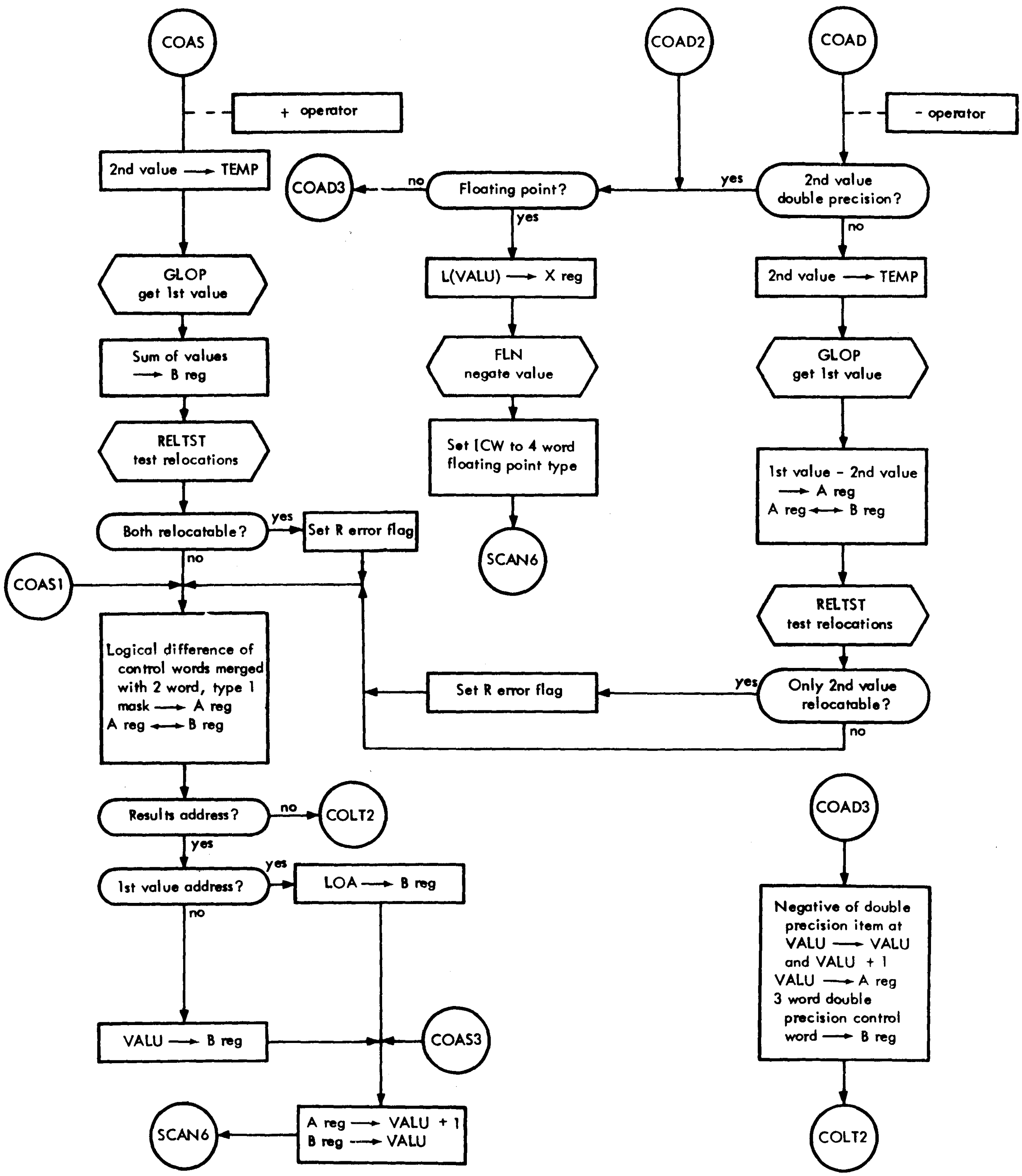
PAS2
SCAN ROUTINE

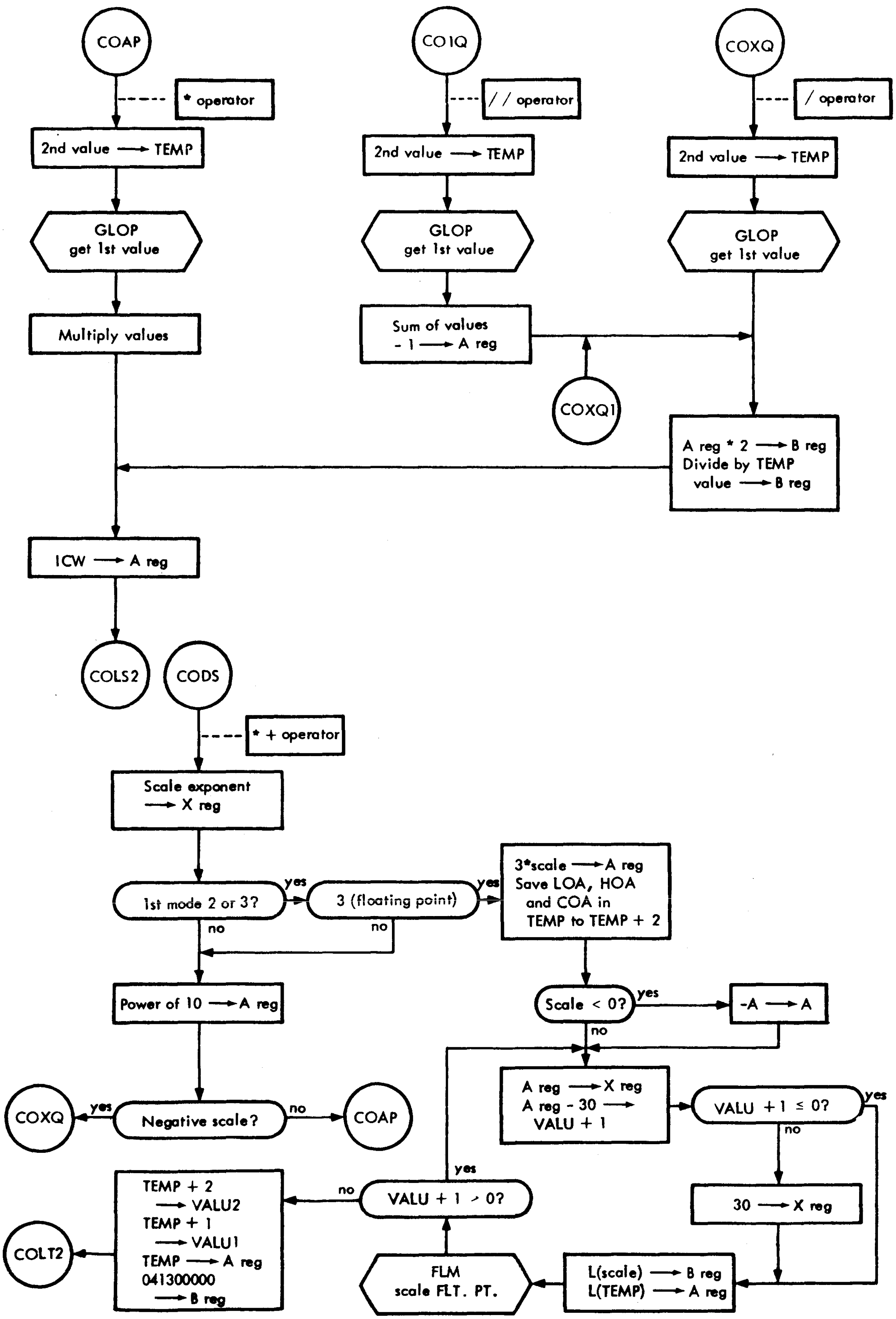


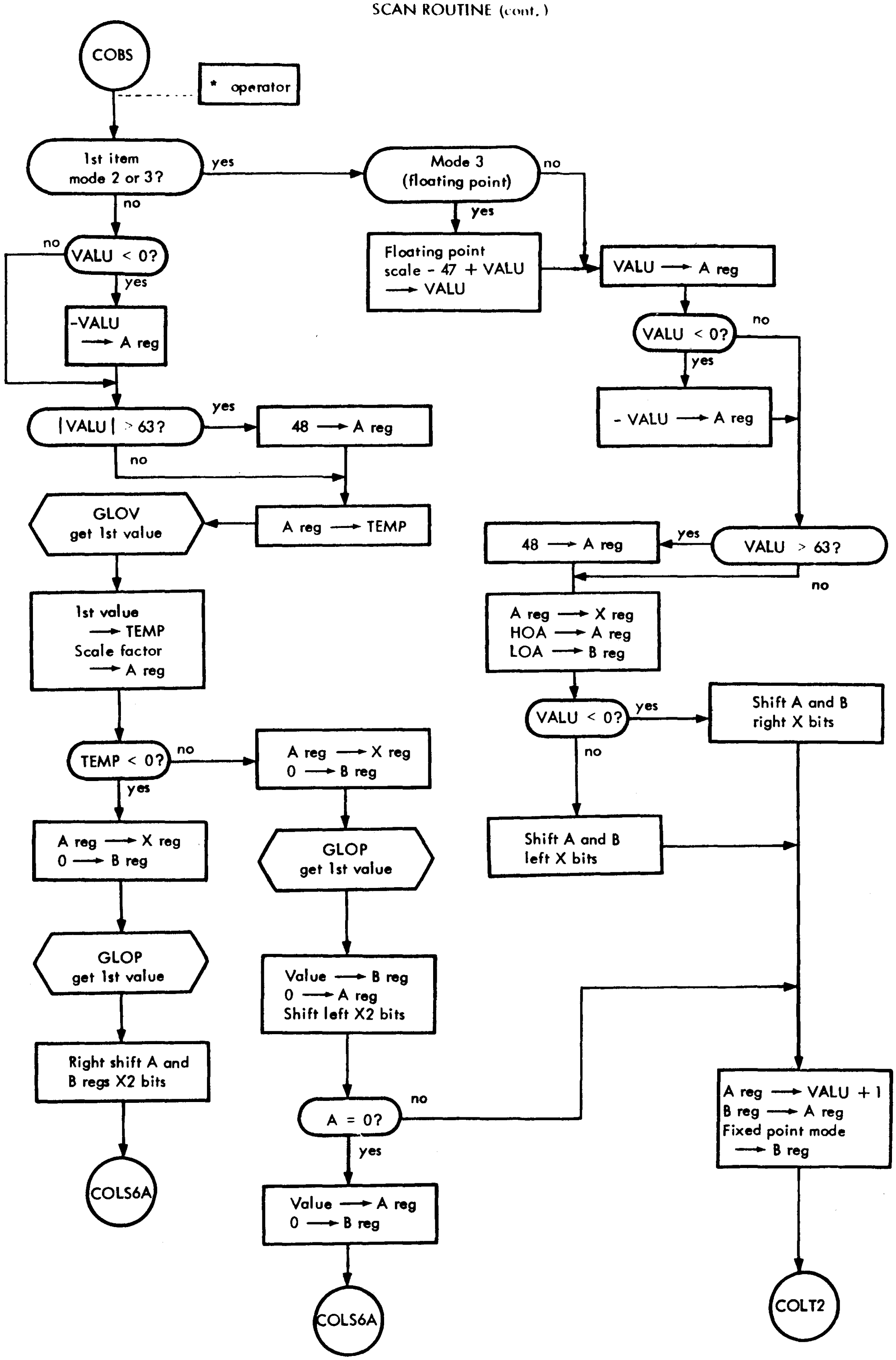

SCAN9
Operation is terminator
SCAN98
LITF > 0?
no
yes
STAR > 0?
yes
no
Reference item?
yes
no
SCANL
SCAN9E
SCAN99
SCAN9E
no
Single precision value?
yes
Reference item?
yes
no
Set E error flag
Set * flag on item
yes
Address type?
no
LITC → TEMP
GTRBL
SCANK
Make address type item
End of literals?
yes
no
Table overflow?
yes
no
TERM 1?
no
yes
GLOV get low value
EXIT
Entry equal this item?
yes
no
Move item at ICW to location given by LTBL. Increment LTBL by item length LITC + 1 → LITC
Blank next?
no
yes
Step to next Literal Table entry
In text line?
yes
no
SCAN998
End of line?
yes
no
Set E error flag 0 → TERM
SCANP
GCW get next byte
Set T error flag
yes
Error flag set?
no
Relocation flag → VALU Location → VALU + 1 Set ICW to 3 word address type
SCAN98


COGT
> operator
2nd value → TEMP
GLOP get 1st value
COLT
< operator
2nd value → TEMP
GLOP get 1st value
2nd value → A reg
1st value → TEMP
COEQ
= operator
2nd value → TEMP
GLOP get 1st value
Values equal?
no
yes
0 → B reg
COLT3
A reg > TEMP?
no
yes
0 → B reg
1 → B reg
COLT1
1 → B reg
B reg → A reg
2 word, type 1,
mode 0 → B reg
COLT2
A reg → value
B reg → ICW
SCAN6
Decrement storage locations for OPA, MODA, LOA, HOA, and COA
SCAN7

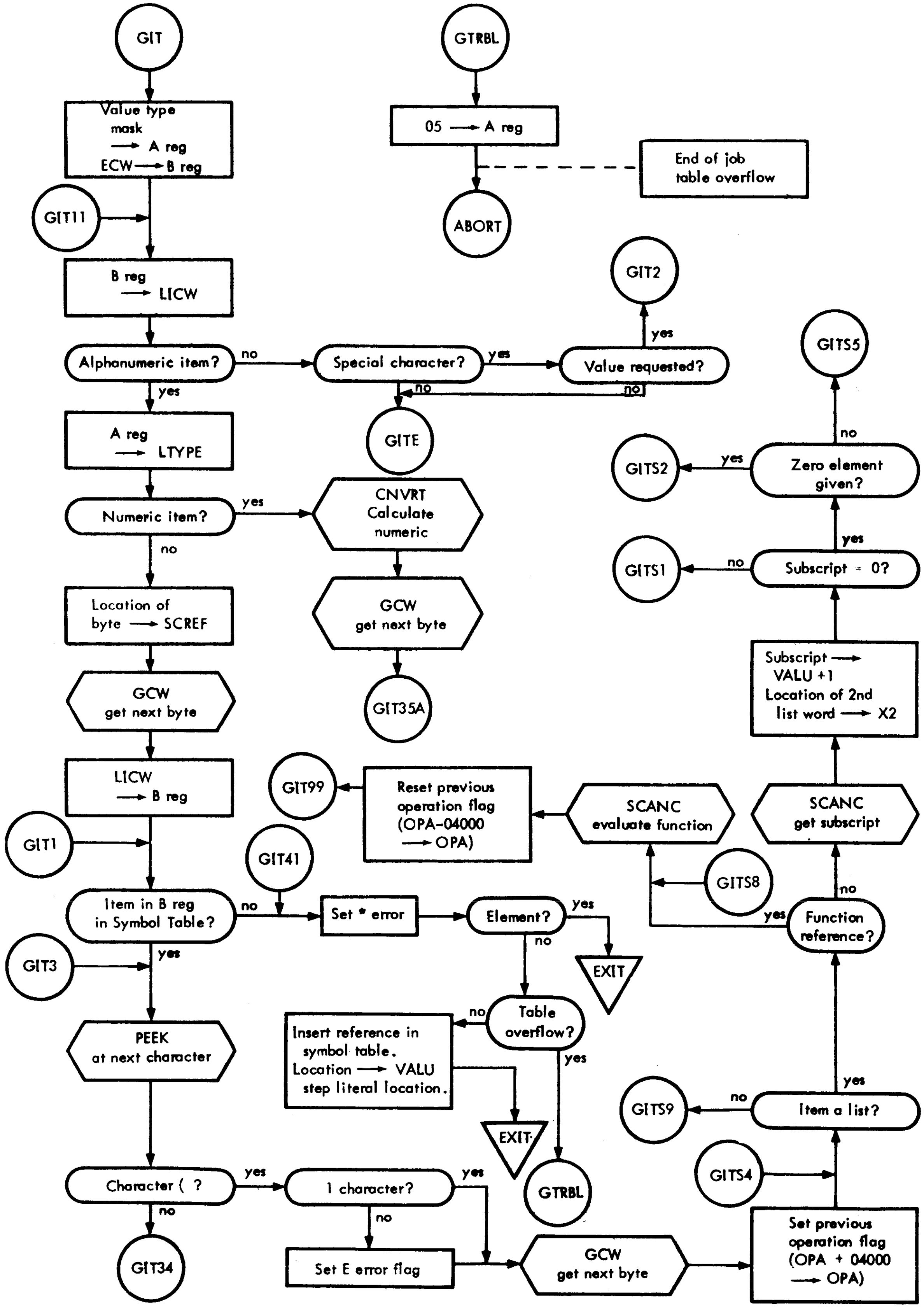

COLS
+ + operator
2nd value → TEMP
GLOP get 1st value
Merge values
COLS6A
Result → VALU + 1
0 → B reg
COLS6
B reg → TEMP
A reg → B reg
RELTST test relocations
Both absolute?
yes
COLT1
no
Both relocatable?
yes
no
2nd value relocatable?
no
yes
COLS3
Bits 9-23 of LOA = TEMP?
yes
no
Set R error flag
COLP
** operator
2nd value → TEMP
GLOP get 1st value
Logical product of values → VALU + 1
037777 → B reg
COLS2
COLT1
yes
Both values absolute?
no
COLS4
Bits 9-23 of VALU = TEMP
no
yes
Set R error flag
1st value → A reg
2nd value → A reg
COLSZ
031100000 → B reg
Mask A reg saving bits 0-8
COLT2
COLD
-- operator
2nd value → TEMP
RELTST test relocation
Set R error flag
no
Both absolute?
yes
GLOP get 1st value
COLS1
Take logical difference of values
ICW → B reg
A reg ↔ B reg
B reg → VALU + 1
COLSZ
yes
2nd item absolute?
no
1st value → A reg
COLSZ

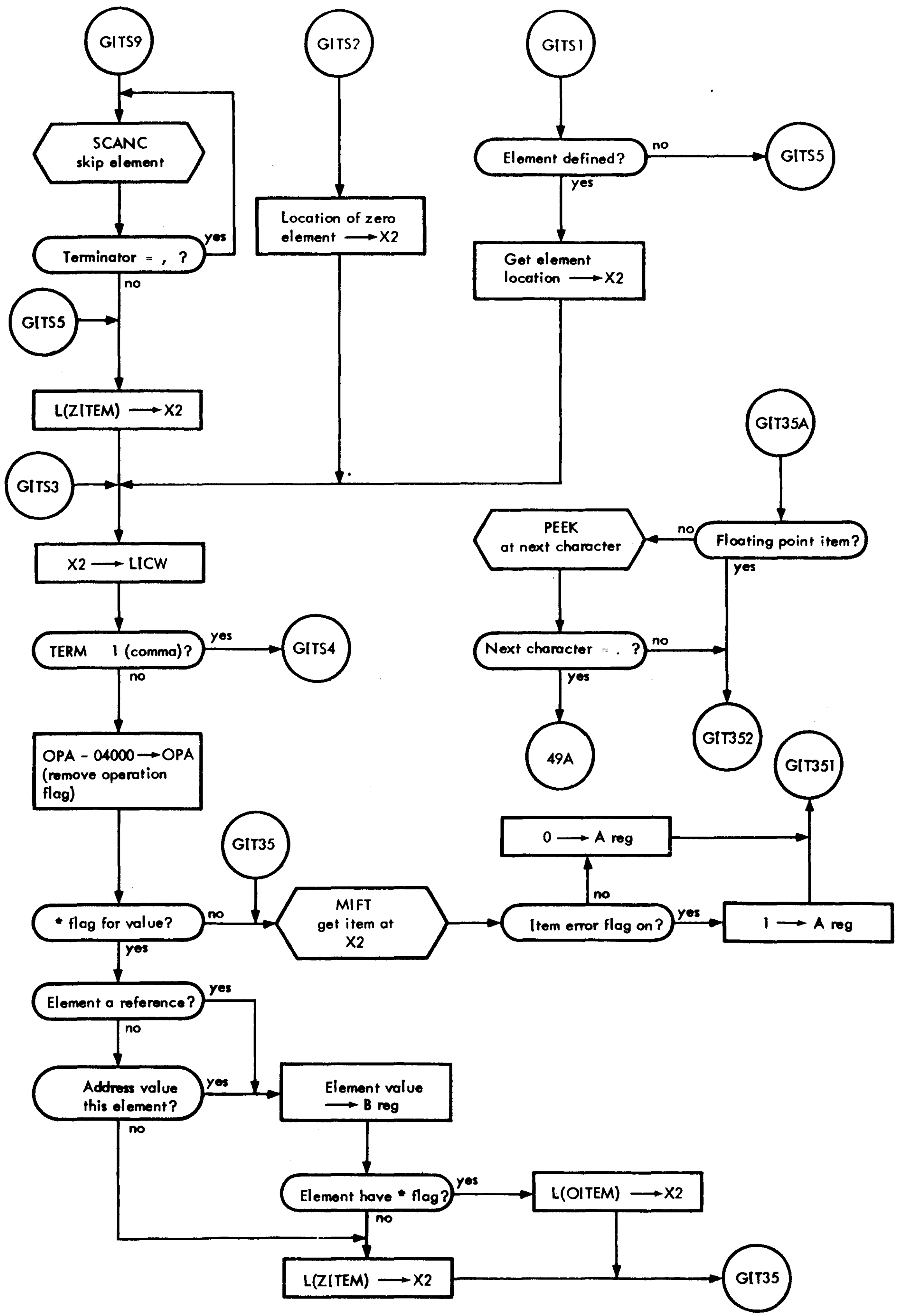

COAS
+ operator
2nd value → TEMP
GLOP get 1st value
Sum of values → B reg
RELTST test relocations
Both relocatable?
yes
Set R error flag
no
COAS1
Logical difference of control words merged with 2 word, type 1 mask → A reg
A reg ↔ B reg
Results address?
no
COLT2
yes
1st value address?
yes
LOA → B reg
no
VALU → B reg
COAS3
A reg → VALU + 1
B reg → VALU
SCAN6
COAD2
COAD3
no
Floating point?
yes
L(VALU) → X reg
FLN negate value
Set ICW to 4 word floating point type
SCAN6
Set R error flag
COAD
- operator
yes
2nd value double precision?
no
2nd value → TEMP
GLOP get 1st value
1st value - 2nd value → A reg
A reg ↔ B reg
RELTST test relocations
yes
Only 2nd value relocatable?
no
COAD3
Negative of double precision item at VALU → VALU and VALU + 1
VALU → A reg
3 word double precision control word → B reg
COLT2

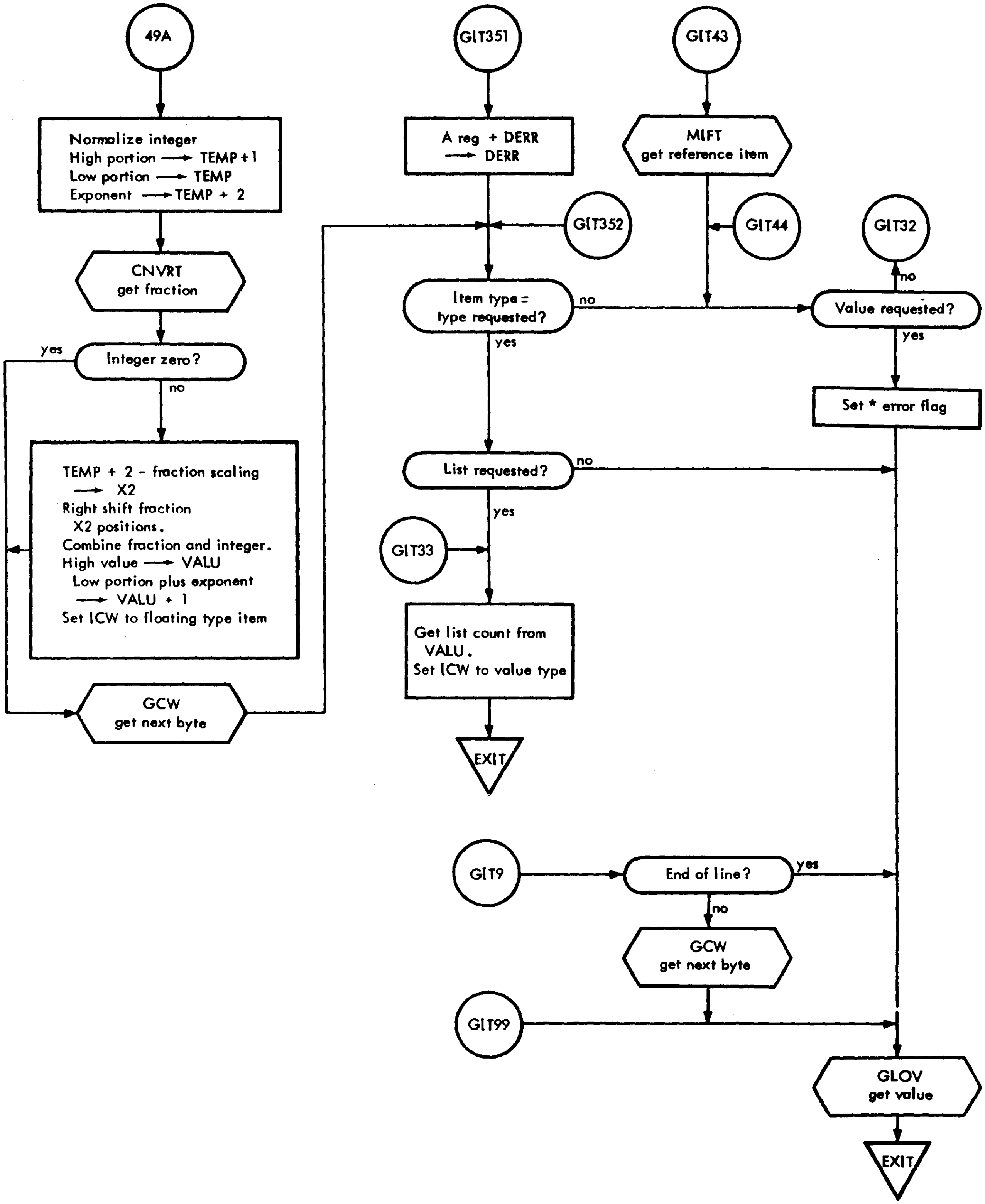

COAP
* operator
2nd value → TEMP
GLOP get 1st value
Multiply values
CO1Q
// operator
2nd value → TEMP
GLOP get 1st value
Sum of values - 1 → A reg
COXQ1
COXQ
/ operator
2nd value → TEMP
GLOP get 1st value
A reg * 2 → B reg Divide by TEMP value → B reg
ICW → A reg
COLS2
CODS
* + operator
Scale exponent → X reg
1st mode 2 or 3?
yes
no
3 (floating point)
yes
no
3*scale → A reg Save LOA, HOA and COA in TEMP to TEMP + 2
Scale < 0?
yes
no
-A → A
Power of 10 → A reg
COXQ
yes
Negative scale?
no
COAP
A reg → X reg A reg - 30 → VALU + 1
VALU + 1 ≤ 0?
yes
no
30 → X reg
COLT2
TEMP + 2 → VALU2 TEMP + 1 → VALU1 TEMP → A reg 041300000 → B reg
no
VALU + 1 > 0?
yes
FLM scale FLT. PT.
L(scale) → B reg L(TEMP) → A reg

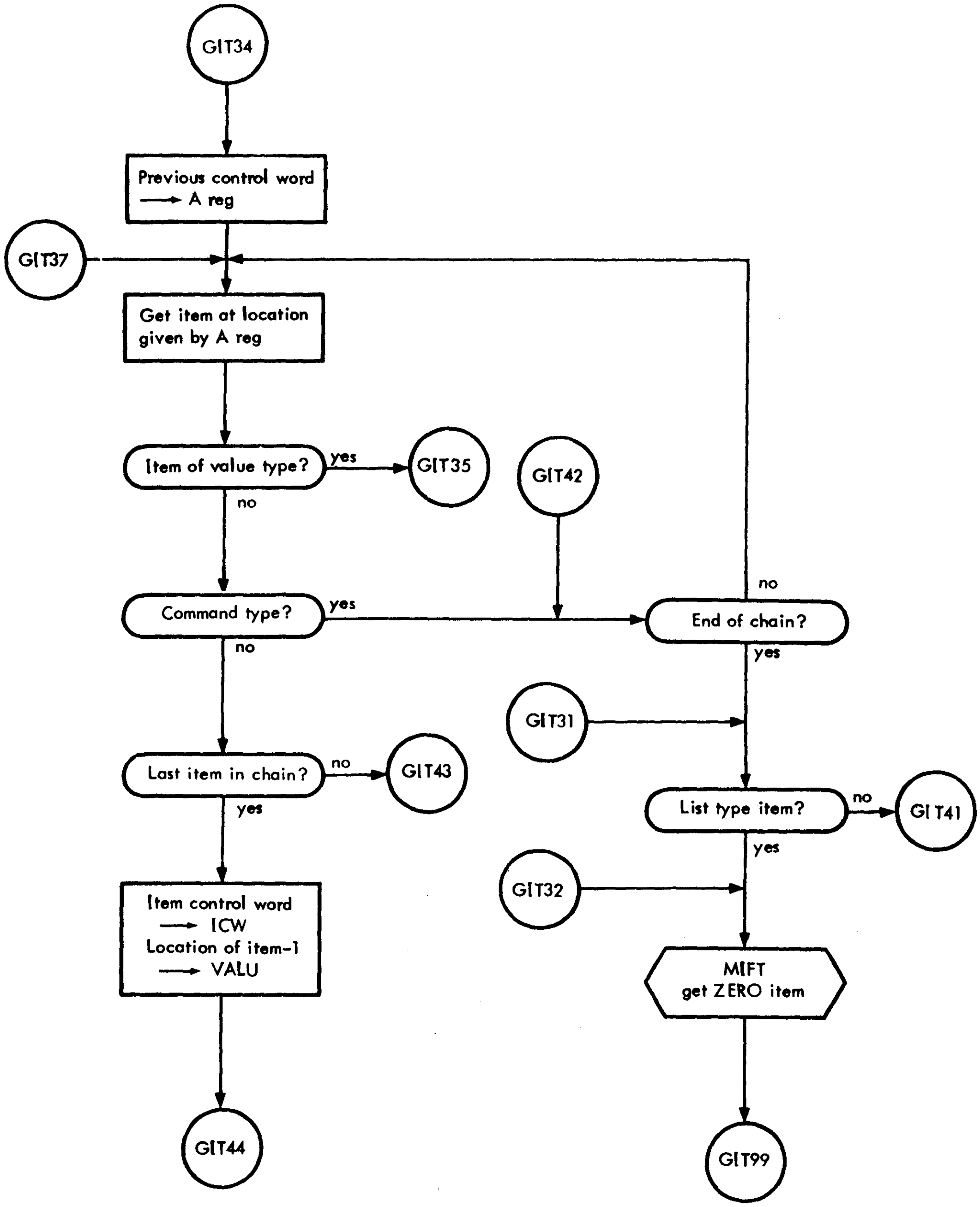

COBS
* operator
1st item mode 2 or 3?
yes
no
Mode 3 (floating point)
no
yes
Floating point scale - 47 + VALU → VALU
VALU → A reg
VALU < 0?
no
yes
-VALU → A reg
|VALU| > 63?
yes
48 → A reg
no
GLOV get 1st value
A reg → TEMP
- VALU → A reg
VALU < 0?
no
yes
VALU > 63?
yes
48 → A reg
no
1st value → TEMP
Scale factor → A reg
A reg → X reg
HOA → A reg
LOA → B reg
VALU < 0?
yes
Shift A and B right X bits
no
TEMP < 0?
no
yes
A reg → X reg
0 → B reg
A reg → X reg
0 → B reg
GLOP get 1st value
Shift A and B left X bits
GLOP get 1st value
Value → B reg
0 → A reg
Shift left X2 bits
Right shift A and B regs X2 bits
A = 0?
no
yes
A reg → VALU + 1
B reg → A reg
Fixed point mode → B reg
COLS6A
Value → A reg
0 → B reg
COLS6A
COLT2

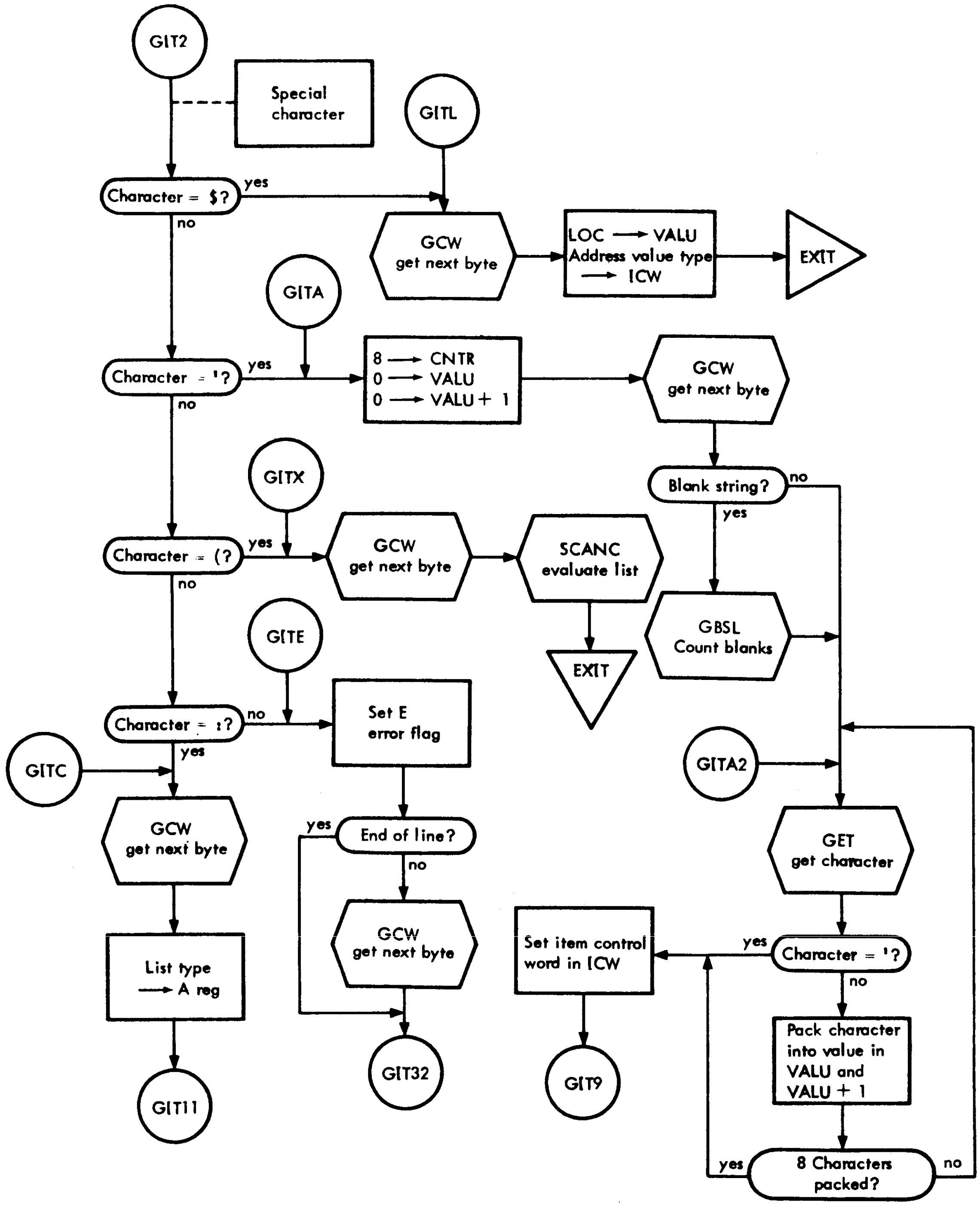
PAS2
GIT ROUTINE

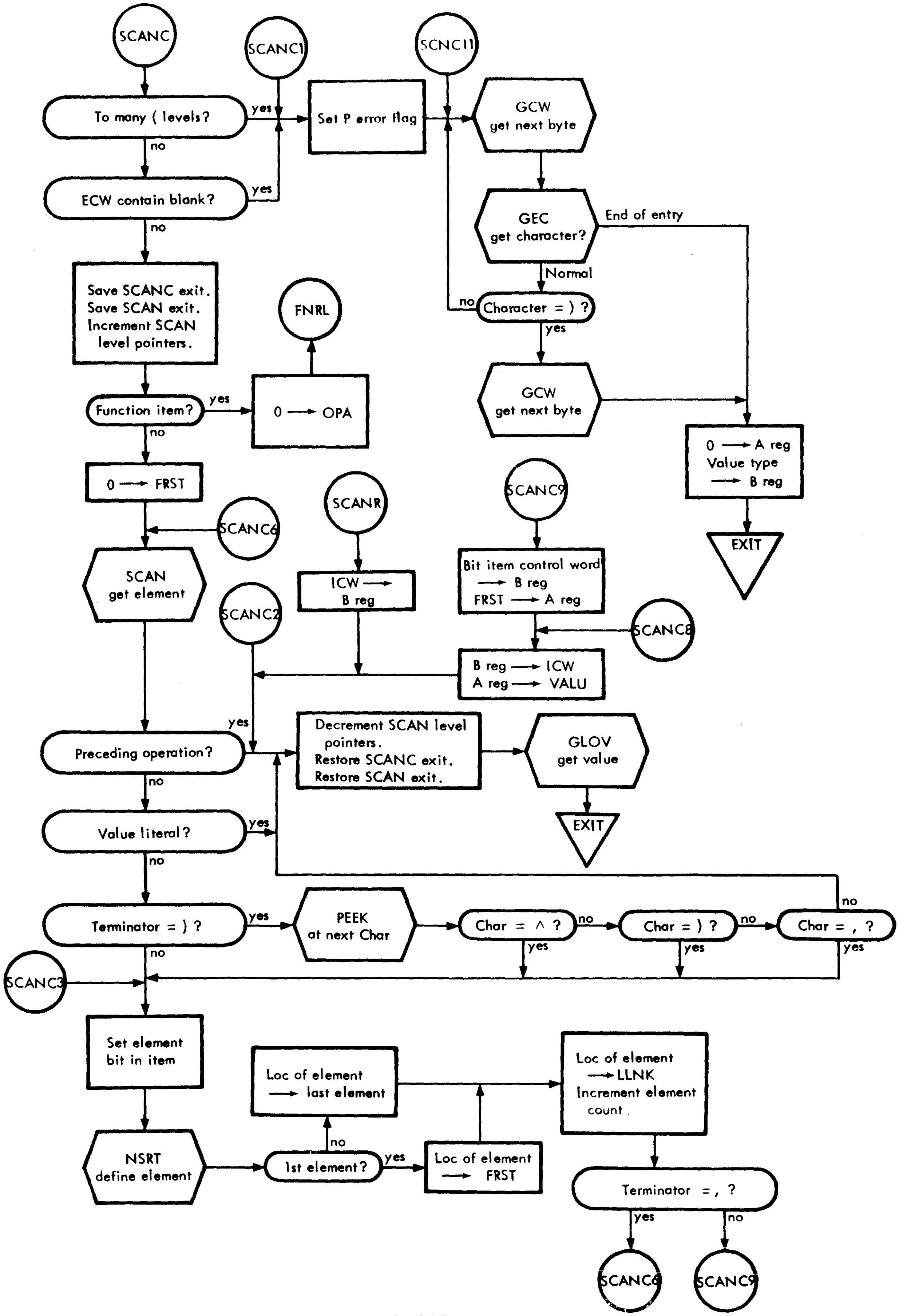


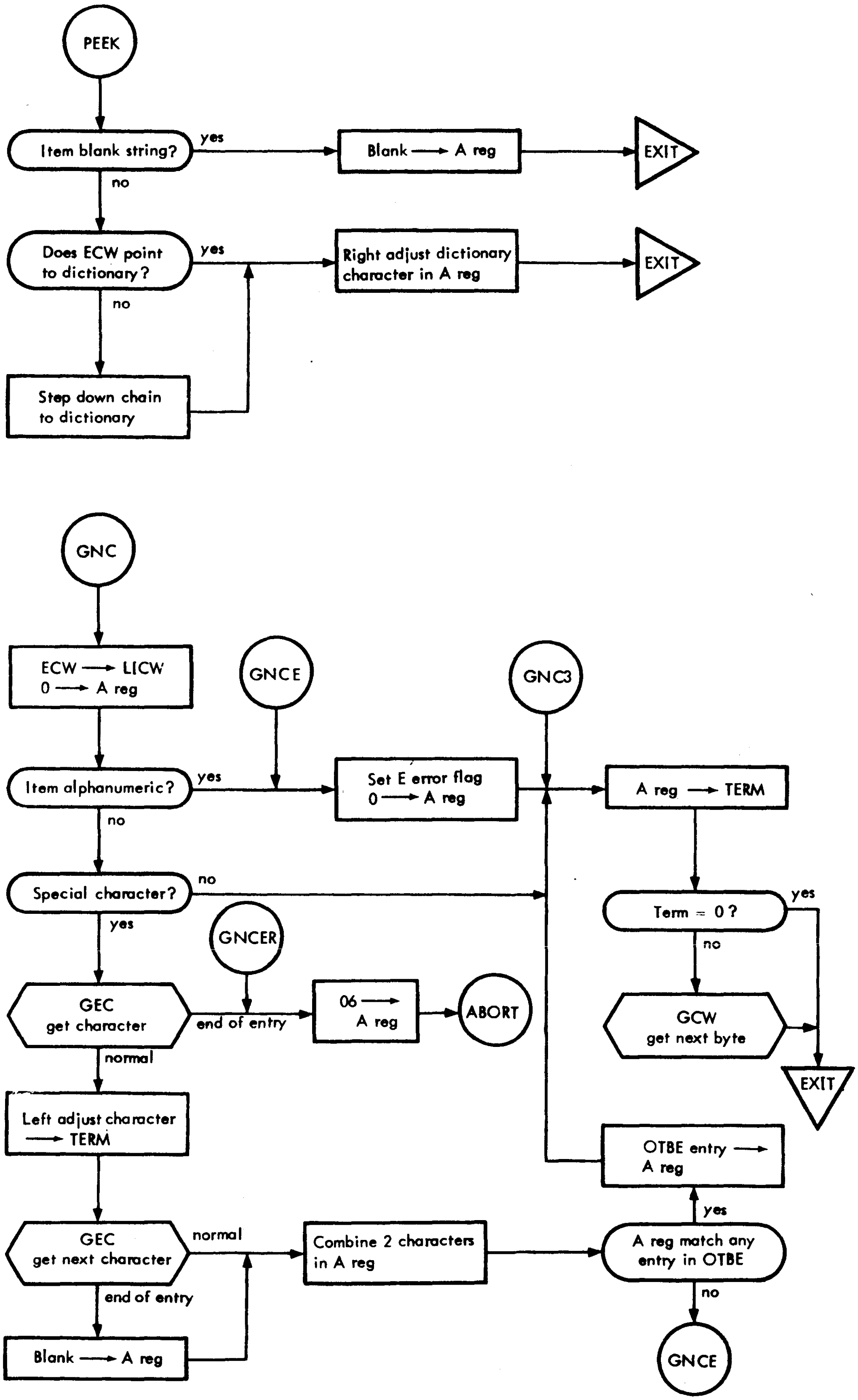

GITS9
GITS2
GITS1
SCANC
skip element
Element defined?
no
yes
GITS5
Location of zero
element ——→X2
Terminator = , ?
yes
no
Get element
location ——→X2
GITS5
L(ZITEM) ——→X2
GIT35A
GITS3
PEEK
at next character
no
Floating point item?
yes
X2 ——→ LICW
TERM 1 (comma)?
yes
no
GITS4
Next character = . ?
no
yes
49A
GIT352
GIT351
OPA - 04000 ——→OPA
(remove operation
flag)
0 ——→ A reg
GIT35
* flag for value?
no
yes
MIFT
get item at
X2
Item error flag on?
no
yes
1 ——→ A reg
Element a reference?
yes
no
Address value
this element?
yes
no
Element value
——→ B reg
Element have * flag?
yes
no
L(OITEM) ——→X2
L(ZITEM) ——→X2
GIT35

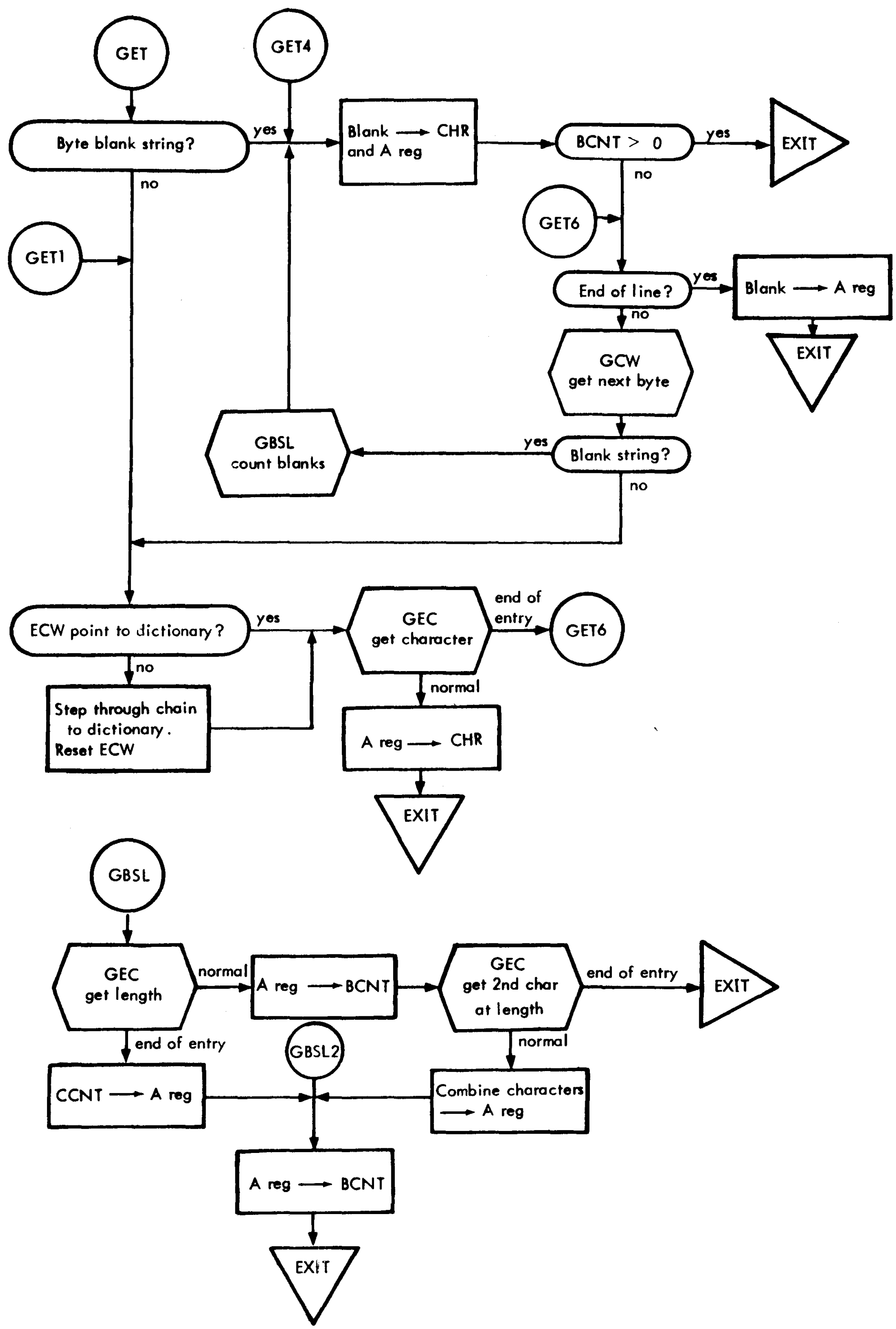

49A
Normalize integer
High portion → TEMP + 1
Low portion → TEMP
Exponent → TEMP + 2
CNVRT
get fraction
Integer zero?
yes
no
TEMP + 2 - fraction scaling → X2
Right shift fraction X2 positions.
Combine fraction and integer.
High value → VALU
Low portion plus exponent → VALU + 1
Set ICW to floating type item
GCW
get next byte
GIT351
A reg + DERR → DERR
GIT352
Item type = type requested?
no
yes
List requested?
no
yes
GIT33
Get list count from VALU.
Set ICW to value type
EXIT
GIT43
MIFT
get reference item
GIT44
GIT32
no
Value requested?
yes
Set * error flag
GIT9
End of line?
yes
no
GCW
get next byte
GIT99
GLOV
get value
EXIT

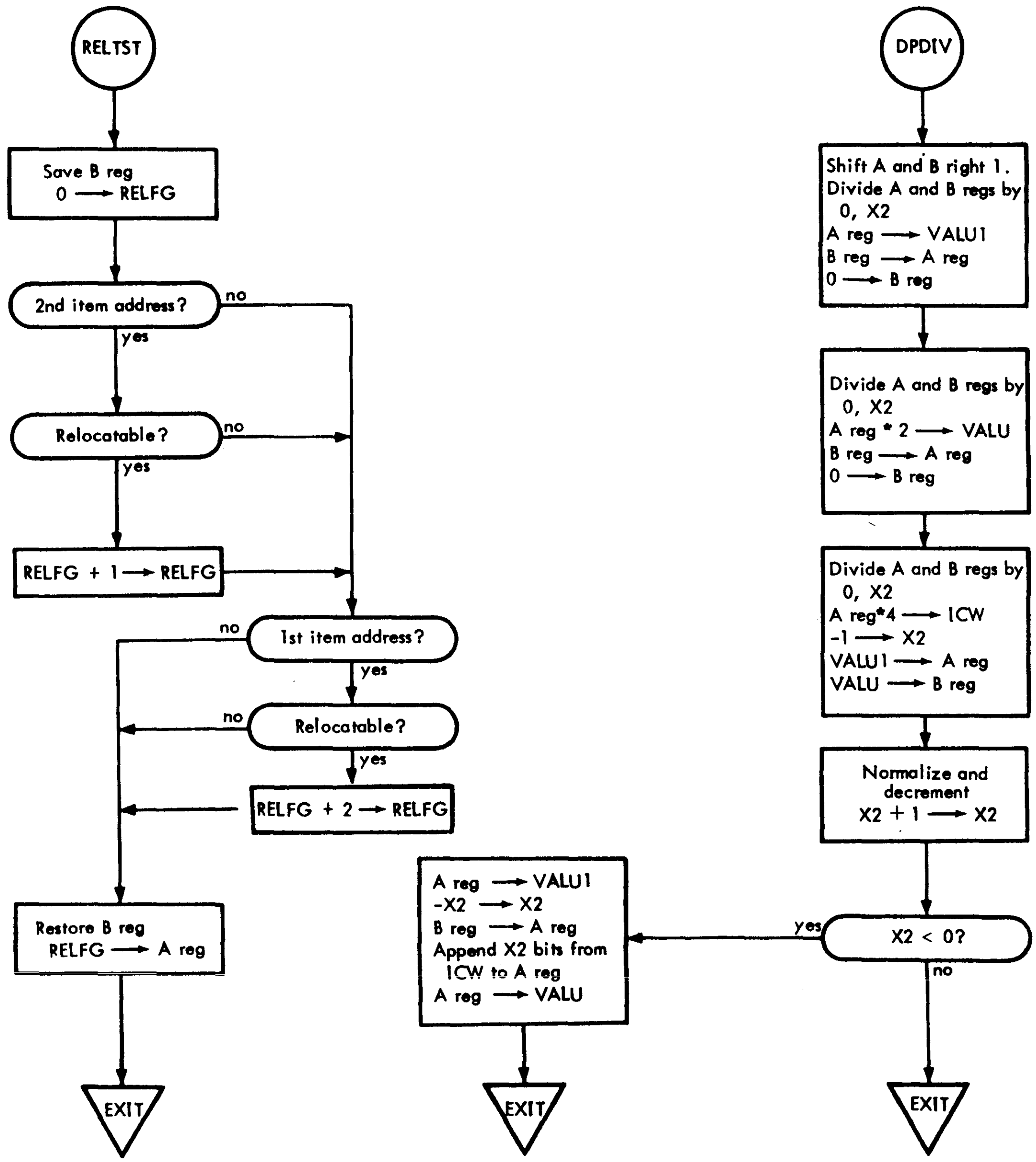

GIT34
Previous control word → A reg
GIT37
Get item at location given by A reg
Item of value type?
yes
GIT35
no
GIT42
Command type?
yes
no
End of chain?
no
yes
GIT31
Last item in chain?
no
GIT43
yes
List type item?
no
GIT41
yes
GIT32
Item control word → ICW
Location of item-1 → VALU
MIFT
get ZERO item
GIT44
GIT99

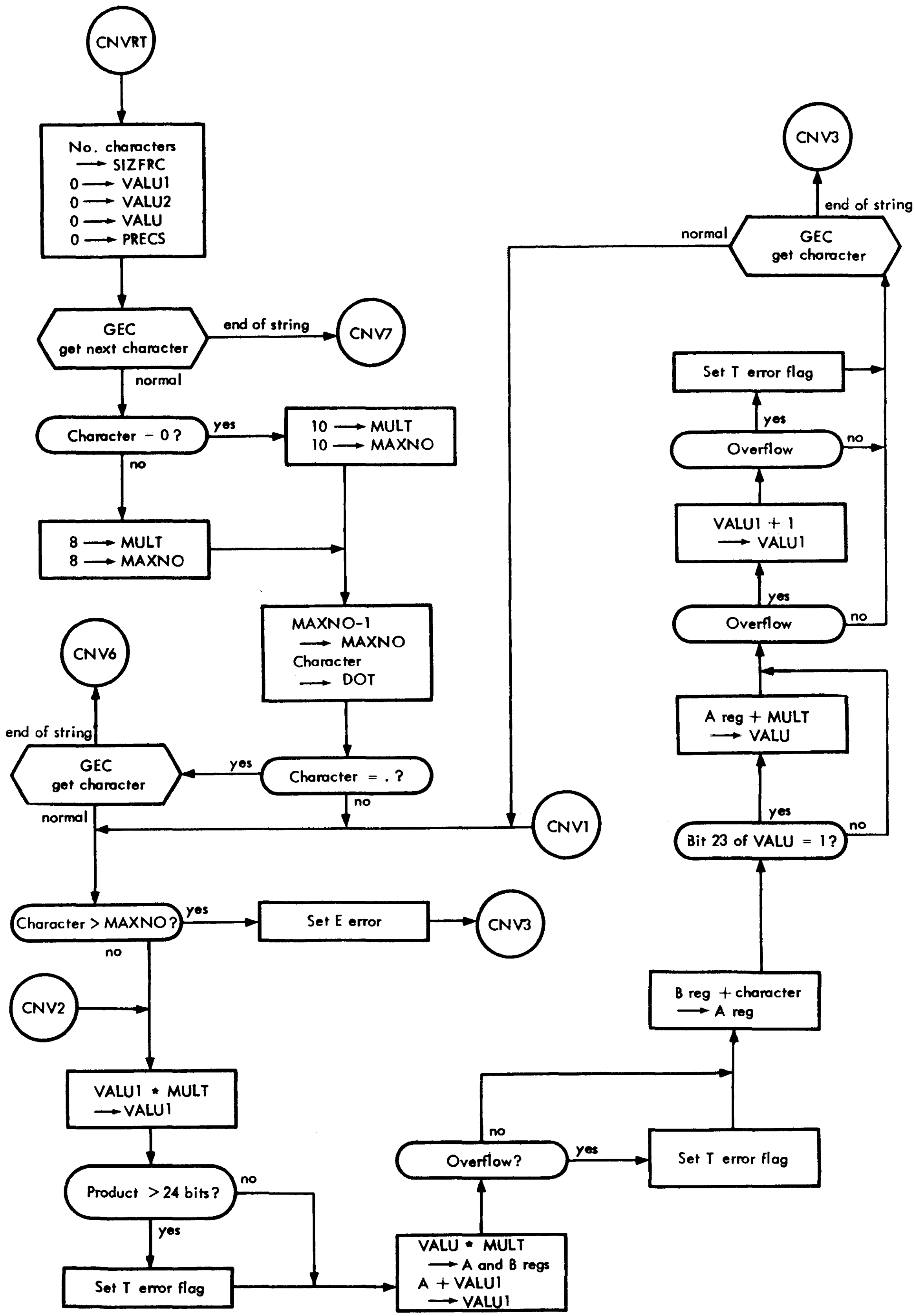

GIT2
Special character
GITL
Character = $?
yes
no
GCW get next byte
LOC → VALU Address value type → ICW
EXIT
GITA
Character = '?
yes
no
8 → CNTR
0 → VALU
0 → VALU + 1
GCW get next byte
Blank string?
no
yes
GITX
Character = (?
yes
no
GCW get next byte
SCANC evaluate list
EXIT
GBSL Count blanks
GITE
Character = :?
no
yes
Set E error flag
GITA2
GITC
GCW get next byte
End of line?
yes
no
GET get character
GCW get next byte
Set item control word in ICW
Character = '?
yes
no
List type → A reg
Pack character into value in VALU and VALU + 1
GIT32
GIT9
GIT11
8 Characters packed?
yes
no

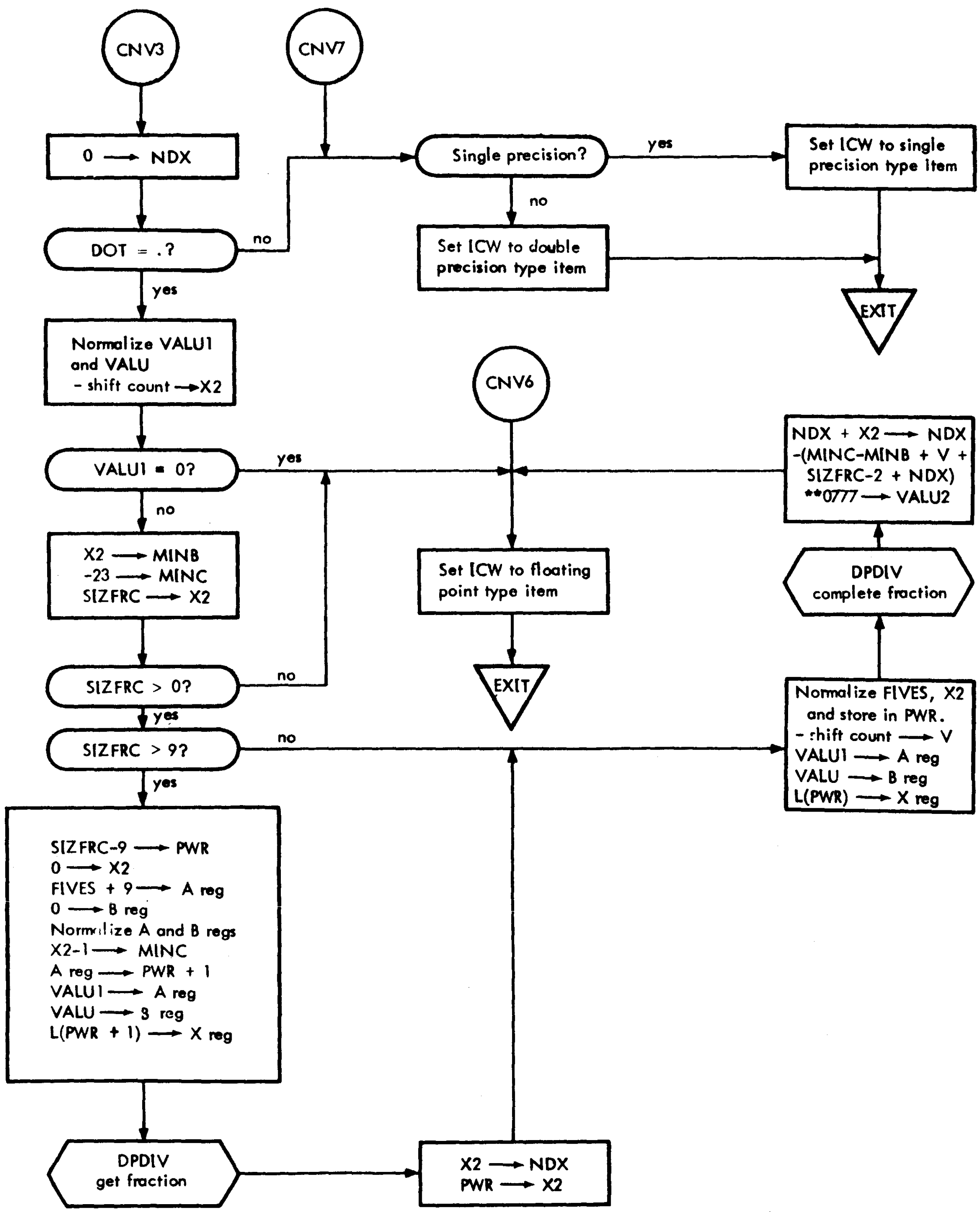

SCANC
SCANC1
SCNC11
To many (levels?
yes
no
Set P error flag
GCW
get next byte
ECW contain blank?
yes
no
GEC
get character?
End of entry
Normal
Save SCANC exit.
Save SCAN exit.
Increment SCAN
level pointers.
FNRL
no
Character =) ?
yes
GCW
get next byte
Function item?
yes
no
0 → OPA
0 → A reg
Value type
→ B reg
0 → FRST
SCANR
SCANC9
EXIT
SCANC6
SCAN
get element
ICW →
B reg
Bit item control word
→ B reg
FRST → A reg
SCANC2
SCANC8
B reg → ICW
A reg → VALU
yes
Preceding operation?
no
Decrement SCAN level
pointers.
Restore SCANC exit.
Restore SCAN exit.
GLOV
get value
Value literal?
yes
no
EXIT
no
Terminator =) ?
yes
PEEK
at next Char
Char = ∧ ?
no
Char =) ?
no
Char = , ?
yes
yes
yes
no
SCANC3
Set element
bit in item
Loc of element
→ last element
Loc of element
→ LLNK
Increment element
count
no
NSRT
define element
1st element?
yes
Loc of element
→ FRST
Terminator = , ?
yes
no
SCANC6
SCANC9

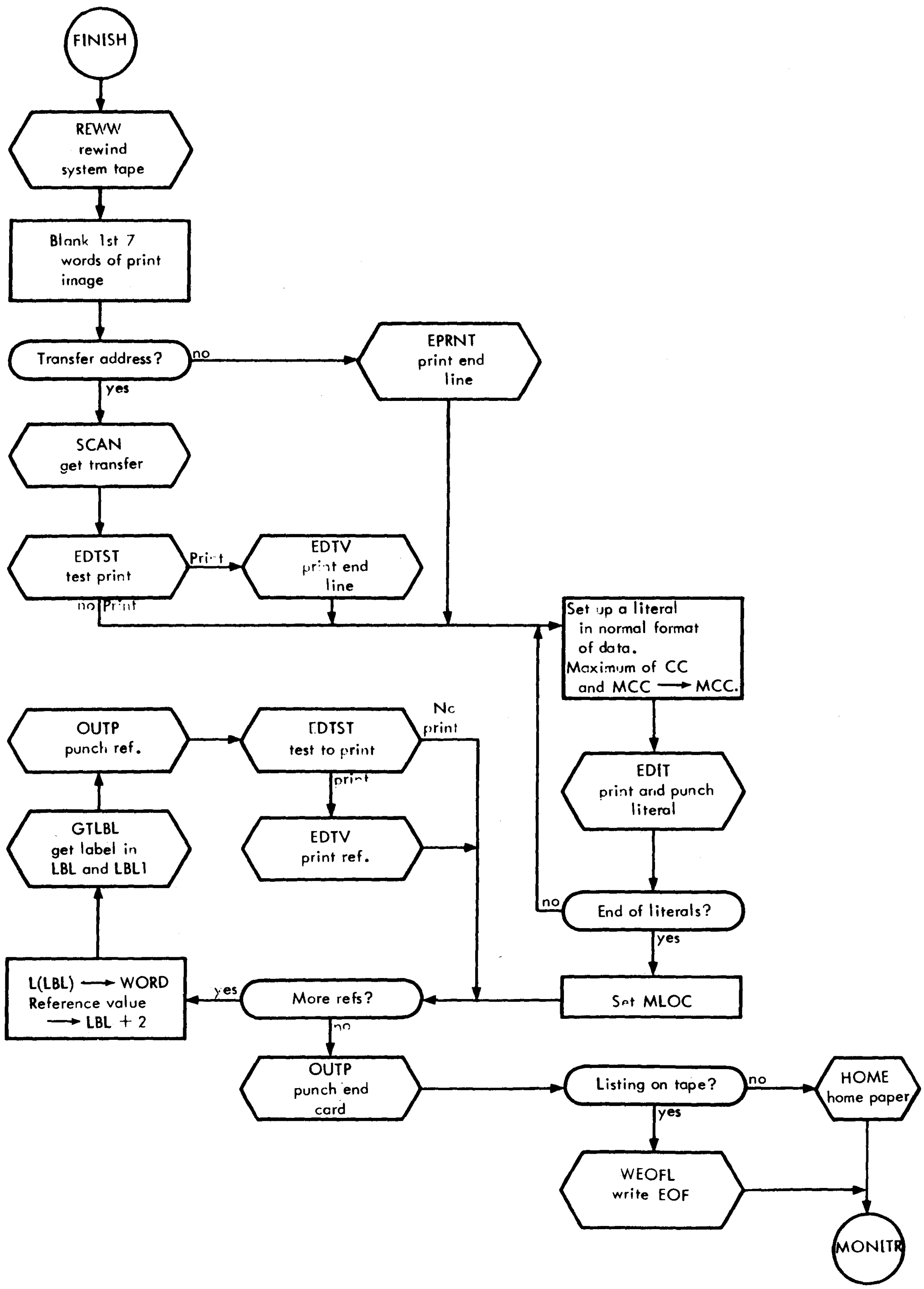

PEEK
Item blank string?
yes
Blank → A reg
EXIT
no
Does ECW point to dictionary?
yes
Right adjust dictionary character in A reg
EXIT
no
Step down chain to dictionary
GNC
ECW → LICW
0 → A reg
GNCE
GNC3
Item alphanumeric?
yes
Set E error flag
0 → A reg
A reg → TERM
no
Special character?
no
Term = 0 ?
yes
yes
GNCER
no
GEC
get character
end of entry
06 → A reg
ABORT
GCW
get next byte
normal
EXIT
Left adjust character → TERM
OTBE entry → A reg
yes
GEC
get next character
normal
Combine 2 characters in A reg
A reg match any entry in OTBE
end of entry
no
Blank → A reg
GNCE

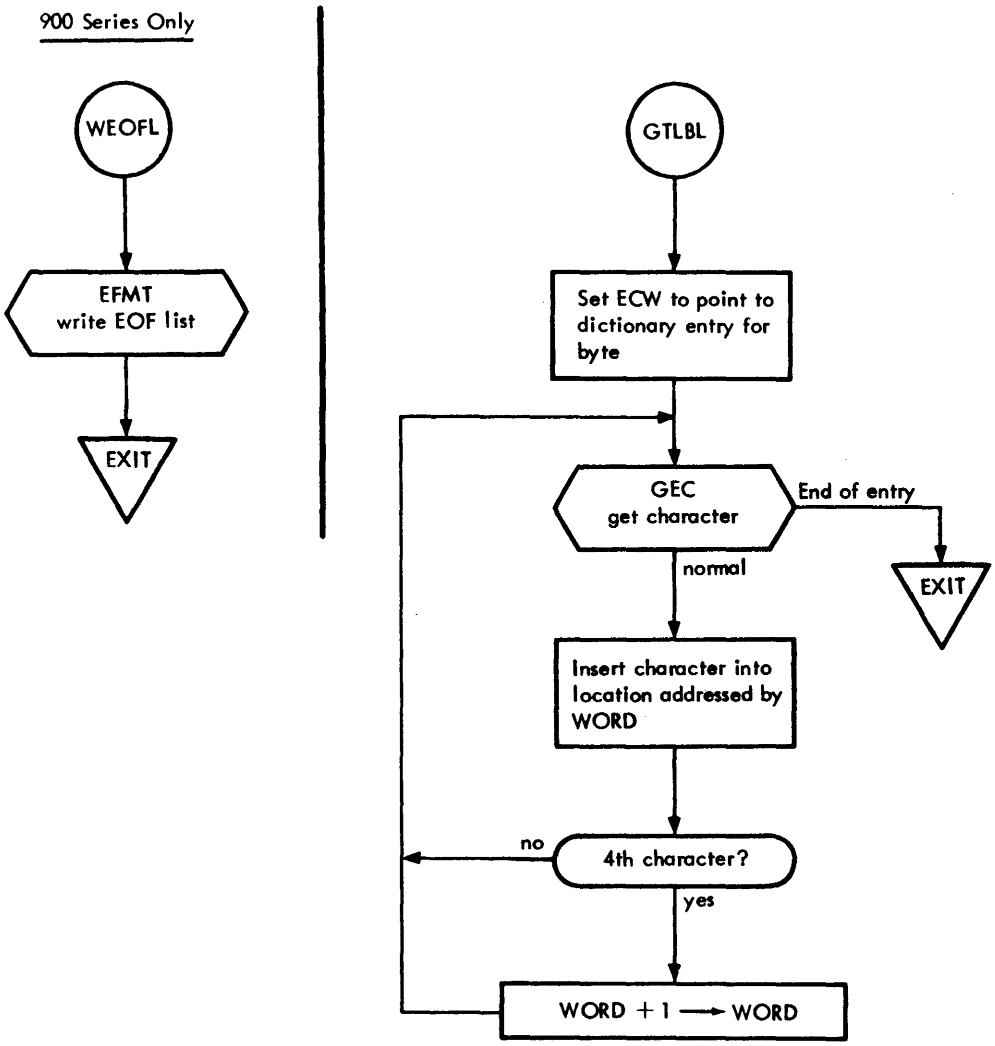

GET
GET4
Byte blank string?
yes
no
Blank → CHR and A reg
BCNT > 0
yes
EXIT
no
GET6
GET1
End of line?
yes
Blank → A reg
EXIT
no
GCW get next byte
GBSL count blanks
yes
Blank string?
no
ECW point to dictionary?
yes
GEC get character
end of entry
GET6
no
normal
Step through chain to dictionary. Reset ECW
A reg → CHR
EXIT
GBSL
GEC get length
normal
A reg → BCNT
GEC get 2nd char at length
end of entry
EXIT
end of entry
GBSL2
normal
CCNT → A reg
Combine characters → A reg
A reg → BCNT
EXIT


RELTST
Save B reg
0 → RELFG
2nd item address?
no
yes
Relocatable?
no
yes
RELFG + 1 → RELFG
1st item address?
no
yes
Relocatable?
no
yes
RELFG + 2 → RELFG
Restore B reg
RELFG → A reg
EXIT
DPDIV
Shift A and B right 1.
Divide A and B regs by
0, X2
A reg → VALU1
B reg → A reg
0 → B reg
Divide A and B regs by
0, X2
A reg * 2 → VALU
B reg → A reg
0 → B reg
Divide A and B regs by
0, X2
A reg*4 → ICW
-1 → X2
VALU1 → A reg
VALU → B reg
Normalize and
decrement
X2 + 1 → X2
X2 < 0?
yes
no
A reg → VALU1
-X2 → X2
B reg → A reg
Append X2 bits from
ICW to A reg
A reg → VALU
EXIT
EXIT


CNVRT
No. characters
→ SIZFRC
0 → VALU1
0 → VALU2
0 → VALU
0 → PRECS
GEC
get next character
end of string
CNV7
normal
Character = 0 ?
yes
10 → MULT
10 → MAXNO
no
8 → MULT
8 → MAXNO
MAXNO-1
→ MAXNO
Character
→ DOT
CNV6
end of string
GEC
get character
yes
Character = . ?
no
normal
CNV1
Character > MAXNO ?
yes
Set E error
CNV3
no
CNV2
VALU1 * MULT
→ VALU1
Product > 24 bits?
no
yes
Set T error flag
VALU * MULT
→ A and B regs
A + VALU1
→ VALU1
Overflow?
no
yes
Set T error flag
B reg + character
→ A reg
Bit 23 of VALU = 1?
yes
no
A reg + MULT
→ VALU
Overflow
yes
no
VALU1 + 1
→ VALU1
Overflow
yes
no
Set T error flag
normal
GEC
get character
end of string
CNV3


CNV3
CNV7
0 → NDX
Single precision?
yes
Set ICW to single precision type item
no
DOT = .?
no
Set ICW to double precision type item
yes
EXIT
Normalize VALU1 and VALU
- shift count → X2
CNV6
NDX + X2 → NDX
-(MINC-MINB + V + SIZFRC-2 + NDX)
**0777 → VALU2
VALU1 = 0?
yes
no
X2 → MINB
-23 → MINC
SIZFRC → X2
Set ICW to floating point type item
DPDIV
complete fraction
SIZFRC > 0?
no
EXIT
yes
SIZFRC > 9?
no
Normalize FIVES, X2 and store in PWR.
- shift count → V
VALU1 → A reg
VALU → B reg
L(PWR) → X reg
yes
SIZFRC-9 → PWR
0 → X2
FIVES + 9 → A reg
0 → B reg
Normalize A and B regs
X2-1 → MINC
A reg → PWR + 1
VALU1 → A reg
VALU → B reg
L(PWR + 1) → X reg
DPDIV
get fraction
X2 → NDX
PWR → X2


FINISH
REWW
rewind
system tape
Blank 1st 7
words of print
image
Transfer address?
no
yes
EPRNT
print end
line
SCAN
get transfer
EDTST
test print
Print
EDTV
print end
line
no Print
Set up a literal
in normal format
of data.
Maximum of CC
and MCC ⟶ MCC.
OUTP
punch ref.
EDTST
test to print
No
print
print
EDIT
print and punch
literal
GTLBL
get label in
LBL and LBL1
EDTV
print ref.
no
End of literals?
yes
L(LBL) ⟶ WORD
Reference value
⟶ LBL + 2
yes
More refs?
Set MLOC
no
OUTP
punch end
card
Listing on tape?
no
HOME
home paper
yes
WEOFL
write EOF
MONITR


900 Series Only
WEOFL
EFMT
write EOF list
EXIT
GTLBL
Set ECW to point to dictionary entry for byte
GEC
get character
End of entry
EXIT
normal
Insert character into location addressed by WORD
4th character?
no
yes
WORD + 1 → WORD

Field	Read	Write
B	Binary mode flag	Binary mode flag
C	Channel	Channel
U	Unit	Unit
I O routine	Location of input routine	Location of output routine
EOF routine	Not used	Location of EOF routine
Dummy control word	Temporary storage	To initialize control word
Control word	First word read	First word to write
Words 7-45	Remainder of record read	Remainder of record to write

ITEM AND TABLE FORMATS USED BY ENCODER

Dictionary Item Format

bits / word	0 3	4 5	6 11	12 17	18 23
0	L	T	1st	2nd	3rd
1	4th				
2					
3				14th	15th

where:

L is the number of characters in entry

T is type of string:

0 - blank

1 - special

2 - numeric

3 - alphanumeric

Entries are full words, as many as needed to represent the string, with a maximum of four words.

The 1st through 15th are characters comprising the string (except for type 0 (blank) strings, where the following one or two characters give the string length).

CPO (Search Table) Item Format

words \ bits	0	1–3	4–5	6–8	9–23
0	D		B		DICTIONARY
1					LESSER
2					GREATER

where:

D is direction taken from item:

0 if lesser

1 if greater

B is balance of table from item:

0 if in balance

1 heavy greater

2 heavy lesser

DICTIONARY is location of dictionary entry for item.

LESSER is location of item smaller than this item.

GREATER is location of item larger than this item.

APO (Dictionary Address Table) Item Format

bits	0–8	9–23
	VALU	DICTIONARY

where:

VALU is byte value of entry.

DICTIONARY is location of dictionary item for entry.

ITEM TABLE FORMATS USED IN META-SYMBOL

Byte Table Entry

Entry for byte b is in STBL – b. Byte table consist of one word with the following format:

C		N	T	F	A	field
(2)	(1)	(4)	(2)	(1)	(14)	no. of bits
0 1	2	3 6	7 8	9	10 23	bit number

where:

C is character position of first character of string in dictionary.

N is number of characters in dictionary string entry.

T is type:

0 – Blank string

1 – Special character string

2 – Numeric character string

3 – Alphanumeric character string

F – The interpretation of F depends upon which routine is operative:

PREA and SRNK

F is flag for interpreting A field.

PAS1, PAS2, FNSH

F is used to detect illegal forward references. F is set to 1 when item is defined during second pass.

A – If F = 0, A is address of word in dictionary containing first character of string. If F = 1, A is address of item in item table with the string of this byte as key. That item will also have an F and an A field which are interpreted in the same manner. Eventually they will be an item with 0 in the F field, and the A field of this item will locate the word in the dictionary containing the first character of string.

Dictionary Table Entry

Entry for string s follows entry for string s–1.

Dictionary strings, with control characters removed, are packed one following the other without regard to word boundaries. The first character of a string is stored in the character position following the position of the last character of the previous string.

Symbol Table Entries

First word is control word. Interpretation of remainder of item is determined by control word.

field	L	I	T	E	M	F	A
no. of bits	(3)	(1)	(2)	(1)	(2)	(1)	(14)
bits	0–2	3	4–5	6	7–8	9	10–23

where:

L is length of entry, including control word.

I is item flag: 0 if item; 1 if element of list.

T is type: 1 if value; 2 if command; 3 if list; 0 if reference.

E is error flag.

M is mode. Interpretation is determined by type (T).

F – The interpretation of F depends upon which routine is operative:

PREA and SRNK

F is flag for interpreting A field.

PAS1, PAS2, FNSH

F is used to detect illegal forward reference. F is set to 1 when item is defined during second pass.

A – If F = 0, A is address of word in dictionary containing first character of string. If F = 1, A is address of another item in table (either next item with the same key, if I = 0, or next element of list, if I = 1). In this case, A is called the associate.

Value item (T = 1). The mode of a value item has the following interpretation:

M = 0 Single-precision absolute.
M = 1 Single-precision address.
M = 2 Double-precision absolute.
M = 3 Double-precision floating point.

If M = 0, 2, or 3, the datum (or value) follows in the next one to three words. If M = 1 and L = 2, the next word has the following format:

field	S	not used	C	R	V
no. of bits	(1)	(6)	(1)	(1)	(15)
bits	0	1–6	7	8	9–23

where:

S is the asterisk flag: 1 if definitions of item was preceded by an asterisk.

C is the common flag: 1 if common bias is to be added.

R is the relocation flag: 1 if relocation bias is to be added.

V is the value of address quantity.

If M = 1 and L = 3, the following two words have this format:

field	S		C	R	
field	V				
bits	0	1–6	7	8	9–23

where: S, C, R and V have the same meaning as above.

If the mode is 3, a 3-word floating-point value follows.

WORD 1	Least significant 24 bits of fraction.
WORD 2	Most significant 24 bits of fraction.
WORD 3	Exponent.

If the mode is 2, the 2-word double-precision value follows.

WORD 1	Least significant 24 bits of value.
WORD 2	Most significant 24 bits of value.

Command Item (T = 2). The mode of a command item determines the sub-type.

Form Command (M = 0). Form pattern is in next word. Form pattern is a word with a 1 in the first bit position of each field and zeros elsewhere.

Procedure Name (M = 1). The control word is followed by the sample control word:

field	P	B	Z	W
no. of bits	(5)	(4)	(1)	(14)
bits	0–4	5–8	9	10–23

where:

P is starting bit position of sample in sample storage word.

B is size of first byte of sample.

Z - If an implied parameter follows (as determined by L in the control word) and if Z = 0, the parameter is a 1-word absolute value; if Z = 1, it is a list word (see list word type).

W is the address of word in sample storage containing first bit of sample.

If an implied parameter is present, it follows in the next word.

Directive (M = 2). The control word is followed by a word containing an index to the directive branch table entry to perform the directive task.

POP Definition (M = 3). The control word is followed by a programmed operator definition word:

	S	N		A
bits	0 1	2 7	8	9 23

where:

S is subtype: 0 - local POP definition

1 - POP reference

2 - external POP definition

N is programmed operator code.

A is value of location counter for POP definitions and zero for POP reference.

List Type (T = 3). This type refers to items which can be referred to in a functional notation. This includes both list items and function names. The mode determines which sub-type the item is.

List Item (M = 0). The control word for a list item is followed by a list word:

N		S	field
(8)	(2)	(14)	no. of bits
0 7	8 9	10 23	

where:

N is number of elements in list.

S is address of first element of list. This is element number 1. If the length of a list item is greater than 2, a sub-item follows the list word. The sub-item is element 0.

Function Name (M = 1). The control word for a function name item is followed by a sample control word as described under procedure name item.

Literal Table Entries

First word is control word. Interpretation of remainder of item is determined by control word.

L	0 0 1	E	M	R	A	field
(3)		(1)	(2)	(1)	(14)	no. of bits
0 2	3 5	6	7 8	9	10 23	

where:

L is length of entry, including control word.

E is truncation error flag.

M is mode. The mode of a literal item has the following interpretation:

M = 0 Single-precision absolute.

M = 1 Single-precision address.

M = 2 Double-precision absolute.

M = 3 Double-precision floating point.

If M = 0, 2, or 3, the datum (or value) follows in the next one or two words. If M = 1, the next word has the following format.

	not used				
S		C	R	V	field
(1)	(6)	(1)	(1)	(15)	No. of bits
0	1 6	7	8	9 23	

where:

S is asterisk flag: 1 if definition of item was preceded by an asterisk.

C is common flag: 1 if common bias is to be added.

R is relocation flag: 1 if relocation bias is to be added.

V is value of address quantity.

R is relocation flag: 1 if A is relocatable.

A is location the literal will occupy when program is loaded.

DO Table (DOTAB) Format

DOTAB	proc level of DO (9)		(1)	location of DO label value (14)	number of bits
+1	lines left to do (6)	lines left to skip (6)	(1)	(11)	number of bits
+2	bits used byte word (5)	byte size (4)	(1)	location of first byte of first line (14)	number of bits
+3	lines to do (6)	lines to skip (6)	(1)	DO count (11)	number of bits
+4	contents of CHAD for first line (24)				number of bits

Procedure Storage Table Values

PTERM Terminator of reference parameter list (TERM):

0 if blank
1 if comma
2 if right parenthesis

FST

CNT	1st ELEMENT LOC
0 – 7	8 – 23

LNK	Location of last element in list.
PRECW	Byte table entry from ECW at end-of-parameter list definition.
LPLV	Value of PLV when proc was entered.
TBLOC	Origin of first symbol table entry at current PROC level.
SVMTP	Location of last word in input buffer at lower PROC level.
PRORG	Location of last NAME item sample pointer word.
PROR	Sample table location of procedure sample for current PROC.
CHDWRD	CHAD the current word of input after processing reference list.
PRPOS	Sample location of PROC line encountered when processing from the sample storage area.
REFPOS	Location of next input byte following procedure reference parameter list.
CCVAL	Value of CC (location center) at start of PROC reference.
PRFG	PROC/FUNC flag: negative if neither; zero if PROC reference; 1 if FUNC reference.
PASS	Pass at current PROC level: negative if first; positive if second.
PRBYT	Value of BYT after processing reference parameter list.
LBL	Symbol table control word for a label waiting to be defined. Zero if no waiting label.
LBL1-LBL3	Value of waiting label.
ELBL	Contents of label on EQU line before calling SCAN.
BYTLOC	Location in BYTE table of byte for current waiting label.
WLLVL	Procedure level at which a waiting label is defined.

Formats of Certain SCAN Communication Cells

ICW. This is the control word for an item evaluated by SCAN; it is the symbol table control word format without dictionary or symbol table pointer.

	L	I	T	E	M	0	zero
bits	0 2	3	4 5	6	7 8	9	10 23

where:

L is length

I is element of list

T is type

E is error

M is mode

VALU through VALU+2. This is the value associated with the item at ICW.

TERM terminator of expression:

0 if blank
1 if comma
2 if right parenthesis

STAR leading * flag: 1 if leading * on expression; zero otherwise

Sample Procedure and Function Entries, in order of occurrence. Procedure and function samples are packed one after the other. A sample follows the preceding sample in the next bit position without regard to word boundaries. The first bit of a sample is stored in the bit position following the position of the last bit of the previous sample.

The first line in the sample is the procedure of function line. If the sample is within another sample, the NAME lines will follow. Otherwise, the next line is the line following the last NAME line. The remaining lines of the sample follow, through the END line.

SECTION 4

ITEM AND TABLE FORMATS USED IN META-SYMBOL

STANDARD I/O CONTROL WORD

contents	M	C	U	A
bits	0	1 3	4 9	10 23

where:

- M is a decimal/binary mode flag; -1 for binary
- C is channel designation
- U is unit number
- A is location of I/O routine to perform the function

Standard I/O control word - RAD

NR	A
0 9	10 23

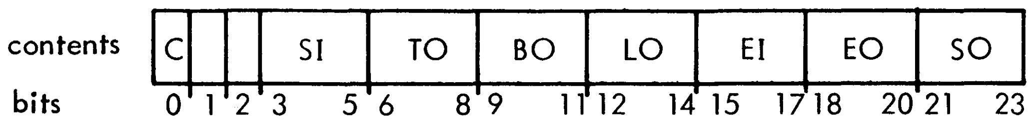
where:

- NR is file number
- A is address of I/O linkage routine to perform the function.

STANDARD I/O CONTROL FLAG

contents	M	C	U	Code
bits	0	1 3	4 9	10 23

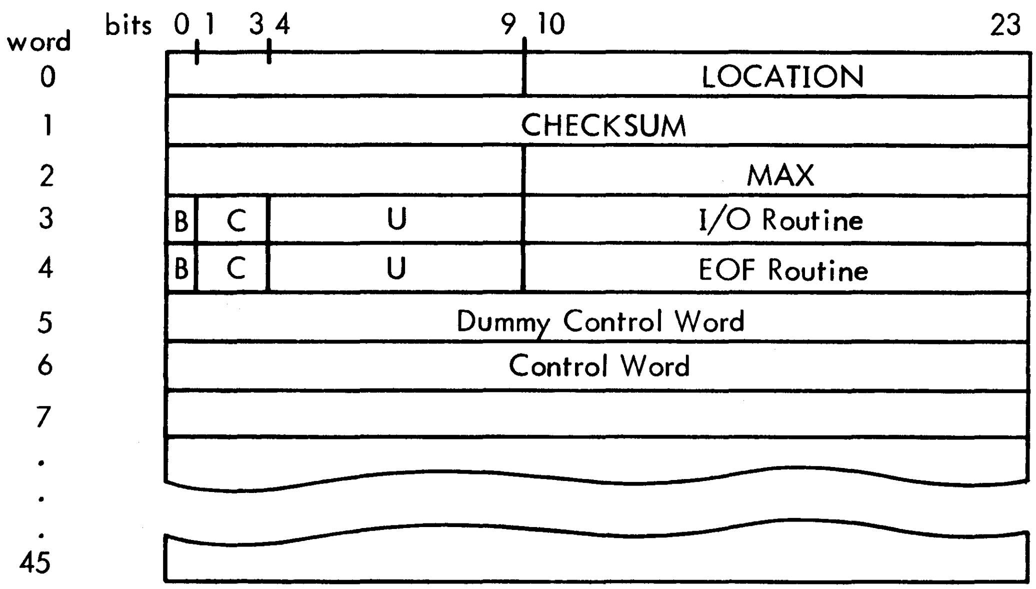
MSFNC FORMAT



A nonzero field indicates the function is to be performed.

C – compatibility mode
SI – symbolic input
TO – intermediate output[†]
BO – binary output
LO – listing output
EI – encoded input
EO – encoded output
SO – symbolic output

STANDARD INPUT/OUTPUT PACKET FORMAT



where the fields have the following meaning:

Field	Read	Write
LOCATION	Location of next data word	Location for next data word
CHECKSUM	Temporary storage	Exclusive 'OR' of words
MAX	Last location of buffer	Last location of buffer

[†] TO is always set to nonzero.

where:

M is decimal/binary mode flag; -1 for binary

C is channel designation

U is unit number

Code is 0 for no operation

1 for card operation

2 for paper tape operation

3 for magnetic tape operation

If the entire cell is zero, the function is not to be completed.

SECTION 4

ITEM AND TABLE FORMATS USED IN META-SYMBOL

ITEM AND TABLE FORMATS USED BY ENCODER .

Dictionary Item Format

bits / word	0 – 3	4 5	6 – 11	12 – 17	18 – 23
0	L	T	1st	2nd	3rd
1	4th				
2					
3				14th	15th

where:

L is the number of characters in entry

T is type of string:

0 - blank

1 - special

2 - numeric

3 - alphanumeric

Entries are full words, as many as needed to represent the string, with a maximum of four words.

The 1st through 15th are characters comprising the string (except for type 0 (blank) strings, where the following one or two characters give the string length).

CPO (Search Table) Item Format

bits / words	0	1 – 3	4 5	6 – 8	9 – 23
0	D		B		DICTIONARY
1					LESSER
2					GREATER

where:

D is direction taken from item:

0 if lesser

1 if greater

B is balance of table from item:

0 if in balance

1 heavy greater

2 heavy lesser

DICTIONARY is location of dictionary entry for item.

LESSER is location of item smaller than this item.

GREATER is location of item larger than this item.

APO (Dictionary Address Table) Item Format

	VALU	DICTIONARY
bits	0 8	9 23

where:

VALU is byte value of entry.

DICTIONARY is location of dictionary item for entry.

ITEM TABLE FORMATS USED IN META-SYMBOL

Byte Table Entry

Entry for byte b is in STBL - b. Byte table consists of one word with the following format:

C	F	N	T	A	field
(2)	(1)	(4)	(2)	(15)	no. of bits
0 1		3 6	7 8	9 23	bit number

where:

C is character position of first character of string in dictionary.

N is number of characters in dictionary string entry.

T is type:

0 - Blank string

1 - Special character string

2 - Numeric character string

3 - Alphanumeric character string

F - The interpretation of F depends upon which routine is operative:

PREA and SRNK

F is flag for interpreting A field.

PAS1, PAS2, FNSH

F is used to detect illegal forward references. F is set to 1 when item is defined during second pass.

A - If F = 0, A is address of word in dictionary containing first character of string. If F = 1, A is address of item in item table with the string of this byte as key. That item will also have an F and an A field which are interpreted in the same manner. Eventually they will be an item with 0 in the F field, and the A field of this item will locate the word in the dictionary containing the first character of string.

Dictionary Table Entry

Entry for string s follows entry for string s-1.

Dictionary strings, with control characters removed, are packed one following the other without regard to word boundaries. The first character of a string is stored in the character position following the position of the last character of the previous string.

Symbol Table Entries

First word is control word. Interpretation of remainder of item is determined by control word.

L	F	I	T	E	M	A	field
(2)	(1)	(1)	(2)	(1)	(2)	(15)	no. of bits
0 1	2	3	4 5	6	7 8	9 23	

where:

L is length of entry, including control word.

I is item flag: 0 if item; 1 if element of list.

T is type: 1 if value; 2 if command; 3 if list; 0 if reference.

E is error flag.

M is mode. Interpretation is determined by type (T).

F – The interpretation of F depends upon which routine is operative:

PREA and SRNK

F is flag for interpreting A field.

PAS1, PAS2, FNSH

F is used to detect illegal forward references. F is set to 1 when item is defined during second pass.

A – If F = 0, A is address of word in dictionary containing first character of string. If F = 1, A is address of another item in table (either next item with the same key, if I = 0, or next element of list, if I = 1). In this case, A is called the associate.

Value item (T = 1). The mode of a value item has the following interpretation:

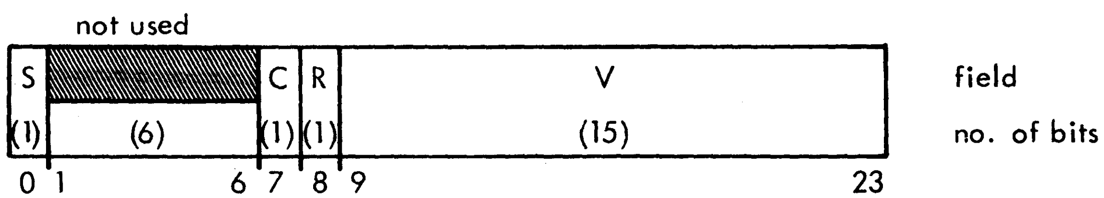
M = 0 Single-precision absolute.

M = 1 Single-precision address.

M = 2 Double-precision absolute.

M = 3 Double-precision floating point.

If M = 0, 2, or 3, the datum (or value) follows in the next one to three words. If M = 1 and L = 2, the next word has the following format:



where:

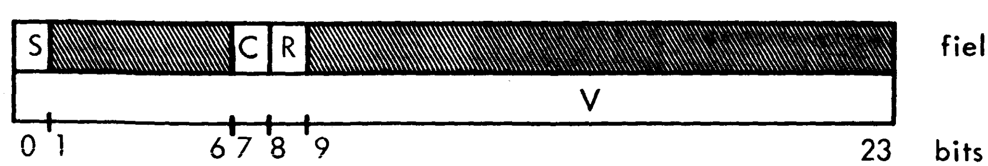
S is the asterisk flag: 1 if definitions of item was preceded by an asterisk.

C is the common flag: 1 if common bias is to be added.

R is the relocation flag: 1 if relocation bias is to be added.

V is the value of address quantity.

If M = 1 and L = 3, the following two words have this format:



where:

S, C, R and V have the same meaning as above.

If the mode is 3, a 3-word floating-point value follows.

WORD 1	Least significant 24 bits of fraction.
WORD 2	Most significant 24 bits of fraction.
WORD 3	Exponent.

If the mode is 2, the 2-word double-precision value follows.

WORD 1	Least significant 24 bits of value.
WORD 2	Most significant 24 bits of value.

Command Item (T = 2). The mode of a command item determines the sub-type.

Form Command (M = 0). Form pattern is in next word. Form pattern is a word with a 1 in the first bit position of each field and zeros elsewhere.

Procedure Name (M = 1). The control word is followed by the sample control word:

P	B	Z	W	field
(5)	(4)	(1)	(14)	no. of bits
0 – 4	5 – 8	9	10 – 23	

where:

P is starting bit position of sample in sample storage word.

B is size of first byte of sample.

Z - If an implied parameter follows (as determined by L in the control word) and if Z = 0, the parameter is a 1-word absolute value; if Z = 1, it is a list word (see list word type).

W is the address of word in sample storage containing first bit of sample.

If an implied parameter is present, it follows the next word.

Directive (M = 2). The control word is followed by a word containing an index to the directive branch table entry to perform the directive task.

POP Definition (M = 3). The control word is followed by a programmed operator definition word:

	S	N	A
bits	0 1	2 7	8 23

where:

S is subtype: 0 - local POP definition

1 - POP reference

2 - external POP definition

N is programmed operator code.

A is value of location counter for POP definitions and zero for POP reference.

List Type (T = 3). This type refers to items which can be referred to in a functional notation. This includes both list items and function names. The mode determines which sub-type the item is.

List Item (M = 0). The control word for a list item is followed by a list word:

N		S	field
(8)	(2)	(14)	no. of bits
0 7	8 9	10 23	

where:

N is number of elements in list.

S is address of first element of list. This is element number 1. If the length of a list item is greater than 2, a sub-item follows the list word. The sub-item is element 0.

Function Name (M = 1). The control word for a function name item is followed by a sample control word as described under procedure name item.

Literal Table Entries

First word is control word. Interpretation of remainder of item is determined by control word.

field	L	R	0 0 1	E	M	A
no. of bits	(2)	(1)		(1)	(2)	(15)
bits	0 1	2	3 5	6	7 8	9 23

where:

L is length of entry, including control word.

E is truncation error flag.

M is mode. The mode of a literal item has the following interpretation:

M = 0 Single-precision absolute.

M = 1 Single-precision address.

M = 2 Double-precision absolute.

M = 3 Double-precision floating point.

If M = 0, 2, or 3, the datum (or value) follows in the next one or two words. If M = 1, the next word has the following format.

field	S	not used	C	R	V
no. of bits	(1)	(6)	(1)	(1)	(15)
bits	0	1 6	7	8	9 23

where:

S is asterisk flag: 1 if definition of item was preceded by an asterisk.

C is common flag: 1 if common bias is to be added.

R is relocation flag: 1 if relocation bias is to be added.

V is value of address quantity.

R is relocation flag: 1 if A is relocatable.

A is location the literal will occupy when program is loaded.

DO Table (DOTAB) Format

DOTAB	proc level of DO (9)	(1)	location of DO label value (14)		number of bits
+1	lines left to do (6)	lines left to skip (6)	(1)	(11)	number of bits
+2	bits used byte word (5)	byte size (4)	(1)	location of first byte of first line (14)	number of bits
+3	lines to do (6)	lines to skip (6)	(1)	DO count (11)	number of bits
+4	contents of CHAD for first line (24)				number of bits

Procedure Storage Table Values

PTERM Terminator of reference parameter list (TERM):

0 if blank
1 if comma
2 if right parenthesis

FST

CNT	1st ELEMENT LOC
0 – 7	8 – 23

LNK	Location of last element in list.
PRECW	Byte table entry from ECW at end-of-parameter list definition.
LPLV	Value of PLV when proc was entered.
TBLOC	Origin of first symbol table entry at current PROC level.
SVMTP	Location of last word in input buffer at lower PROC level.
PRORG	Location of last NAME item sample pointer word.
PROR	Sample table location of procedure sample for current PROC.
CHDWRD	CHAD the current word of input after processing reference list.

PRPOS	Sample location of PROC line encountered when processing from the sample storage area.
REFPOS	Location of next input byte following procedure reference parameter list.
CCVAL	Value of CC (location center) at start of PROC reference.
PRFG	PROC/FUNC flag: negative if neither; zero if PROC reference; 1 if FUNC reference.
PASS	Pass at current PROC level: negative if first; positive if second.
PRBYT	Value of BYT after processing reference parameter list.
LBL	Symbol table control word for a label waiting to be defined. Zero if no waiting label.
LBL1- LBL3	Value of waiting label.
ELBL	Contents of label on EQU line before calling SCAN.
BYTLOC	Location in BYTE table of byte for current waiting label.
WLLVL	Procedure level at which a waiting label is defined.

Formats of Certain SCAN Communication Cells

ICW. This is the control word for an item evaluated by SCAN; it is the symbol table control word format without dictionary or symbol table pointer.

	L	0	I	T	E	M	zero
bits	0 1	2	3	4 5	6	7 8	23

where:

L is length

I is element of list

T is type

E is error

M is mode

VALU through VAL+2. This is the value associated with the item at ICW.

TERM terminator of expression:

0 if blank
1 if comma
2 if right parenthesis

STAR leading * flag: 1 if leading * on expression; zero otherwise

Sample Procedure and Function Entries, in order of occurrence. Procedure and function samples are packed one after the other. A sample follows the preceding sample in the next bit position without regard to word boundaries. The first bit of a sample is stored in the bit position following the position of the last bit of the previous sample.

The first line in the sample is the procedure of function line. If the sample is within another sample, the NAME lines will follow. Otherwise, the next line is the line following the last NAME line. The remaining lines of the sample follow, through the END line.

SECTION 5

OPERATIONAL INFORMATION

The META-SYMBOL assembly system encompasses several core overlays and much communication between segments. The purpose of this section is to summarize the steps taken in modifying portions of the system, to explain how to make system tapes, to define error messages and error halts, and to suggest items to be checked in the event of trouble.

UPDATING META-SYMBOL ON MONARCH SYSTEM TAPES

Use the standard MONARCH ASSIGN, UPDATE, and COPY control cards. Insert in the update deck the binary (encoded in the case of PROC) decks to be changed and do a normal update.

When updating a section of META-SYMBOL, all portions of the labeled segment must be updated. For example, to insert a new PROC deck, one must also insert the PREASM absolute deck preceding it.

Binary patches may be inserted at the end of the absolute binary deck just preceding the END card.

If PREASM, SHRINK, ASSEMBLR, PAS2, or FINISH is modified through reassembly, it is necessary to convert the program to absolute before placing it on the updated system tape.

The order of the deck is as follows:

ENCODER
POPS (910 or 920 depending on object machine)
S4B
MON1
Basic Tape Loader
MSCONTRL
PREASM (absolute deck combining parts 1 and 2 and the POPS)
Standard procedure deck (910, 920, or 9300)
SHRINK (absolute deck)
ASSEMBLR (absolute deck of pass 1 containing pass 1 parts 1-5 and POPS)
PAS2 (absolute deck of pass 2)
FINISH (absolute deck of FINISH)

MAKING THE ABSOLUTE PROGRAM DECKS

Following is a list of steps needed to make the various absolute decks:

1. PREA

   Load with zero relocation bias PREASM part 1, the POP deck (910, or 920), and PREASM part 2. Load the absolute program maker (Cat. No. 000018B) and dump from cell 100 through 126. Dump from 1505 through the end of PREASM part 2.

2. SRNK

   Remove the END card from PREASM part 2 and load PREASM part 1, POP, PREASM part 2, and SHRINK at relocation bias zero. Load the absolute program maker and dump from 3615 through the end of SHRINK.

3. PAS1

   Load the POPs with a relocation bias sufficient to put them after PAS2. After the POPs are loaded, clear the relocation bias to zero (clear the A register) and load parts 1 to 5 of the assembler pass 1. Load the absolute program maker and dump from 100 to 126 and from 1600 through the end of the POPs.

4. PAS2

   Remove the external symbol definition cards (type 1 cards) from the beginning of the relocatable deck. The balance is a loadable absolute deck.

5. FNSH

   Take the external symbol definitions from PAS2 and place them in front of the FINISH relocatable deck. Load the external definitions and FINISH. Load the absolute program maker and dump from 4506 to the end of the FINISH program.

ERROR MESSAGE CODES

The standard abort message is "META-SYMBOL ERROR XX."

Where XX has the following meanings:

XX	Interpretation
01	Insufficient space to complete encoding of input.
02	Corrections to encoded deck but encoded input file is empty.
03	End of file detected while reading encoded input.
04	Insufficient space to complete preassembly operations.
05	Insufficient space to complete the assembly.
06	Data error. META-SYMBOL does not recognize the data as anything meaningful.
07	Requested output on a device which is not available.
08	Corrections out of sequence.
09	End of file detected by ENCODER when trying to read intermediate output tape X1.
10	Not used
11	Byte larger than dictionary (bad encoded deck).
12	Not ENCODED deck
13	Checksum error reading system tape.
14	Preassembler overflow (ETAB)
15	Not used
16	Data error causing META-SYMBOL to attempt to process procedure sample beyond end of table.

Errors 05, 06, and 16 are accompanied by a printout which shows the value of certain internal parameters at the time of the abort:

LINE NUMBER	BREAK	
BREAK1	SMPWRD	
LOCATION COUNTER	LTBE	SECOND PASS ONLY
UPPER	LTBL	
LOWER		

The last six of these are useful in determining the nature of the assembler overflow.

PAS1 Overflows

During PAS1 all memory not in use is allocated to four partially dynamic tables. UPPER is set to top of available memory; SMPWRD is set to bottom of available memory; LOWER and BREAK are set to bottom of available memory + BREAK1.

Odd procedure level symbols are saved in decending order from top of memory (note main program is considered level 1); UPPER is updated to continuously point to the next high available cell. Even procedures and external definitions are built upward from the original value of LOWER, and LOWER is modified. If LOWER > UPPER, one type of PAS1 overflow has occurred.

User procedure sample is built upward from SMPWRD, and SMPWRD is modified. If SMPWRD > BREAK, the second type of overflow has occurred.

PAS2 Overflows

At the beginning of PAS2, LTBE is set equal to BREAK, LOWER is set equal to BREAK, and LTBL is set equal to SMPWRD which is just above user sample. During PAS2 the area between LOWER and UPPER is used in a manner similar to that of PAS1 and can overflow if LOWER > UPPER.

External reference tables are built down from BREAK using LTBE as a pointer. Literals are built up from SMPWRD using LTBL as a pointer. If LTBL > LTBE, overflow has occurred.

I/O ERROR MESSAGES AND HALTS

When an I/O error is detected a simple message is typed and the computer halts. The action taken if the halt is cleared depends on the type of error and the device involved. There are three types of error. The message consists of a 2-letter indication of the type of error and a 2-digit indication of the I/O device. The letter indicators are defined below; the 2-digit number is the unit address number used in EOM selects (see Reference Manual for appropriate 900 Series Computer).

Buffer Error (BE)

1. Examples:

   BE11 buffer error while reading magnetic tape 1

   BE42 buffer error while writing magnetic tape 2

2. Action on clearing halt.

   a. Magnetic tape input

      Since ten attempts are made to read the record before halting, continueing causes META-SYMBOL to accept the bad record.

   b. Paper tape or card input

      Try again.

   c. Magnetic tape output

      Try again.

   d. Output other than magnetic tape.

      Continue.

Checksum Error (CS)

1. Examples:

   CS06 checksum error card reader.

   CS11 checksum error reading magnetic tape 1.

2. Action on clearing halt

   Accept bad record.

Write Error (FP) - Trying to write on file protected tape

1. Example:

   FP42 Magnetic tape 2 file protected

2. Action on clearing halt

   Checks again.

SECTION 6
META-SYMBOL CONCORDANCE OPTION

DESCRIPTION OF THE OVERALL PROCESSING

The program to provide the concordance listing is loaded as two separate overlays following FINISH on the MONARCH system tape.

If a request for a concordance has been made, FINISH saves the locations of the dictionary and symbol tables and then calls the tape loader to load the first overlay of the concordance program, CONCRD.

If exceptions to the normal case are indicated, CONCRD reads the exception control records consisting of EXCLUDE or INCLUDE records from the symbolic input device and retains the list of symbols to be included or excluded. The intermediate output tape, X1, is then scanned to extract the symbols and line numbers to appear in the concordance listing. Symbols to appear in the listing are converted from the encoded to symbolic format and are retained in core. The line numbers containing symbolic definitions or references to appear in the listing are written to the scratch file on X2. For each definition or reference to appear on a line, a pointer word giving the location of the symbol in core and a flag indicating definition or reference is written on X2. As each symbol reference is encountered, a count of the number of half words needed to retain the reference line number is kept with the symbol.

When the entire encoded input file has been processed, the tapes X1 and X2 are rewound and the second overlay, CON2, of the concordance routine is loaded.

CON2 rewinds the system tape and then passes through the symbol table and determines the total core requirement to retain the reference line numbers for the program. If the number of words needed to retain the reference line numbers exceeds the core available, the symbols that appear at the end of the listing are ignored and a recount is made. This process of elimination of later symbols is continued until a subset of symbols is obtained for which all reference line numbers can be retained. Space is then allocated for reference line numbers for each symbol or each symbol in the subset, the scratch tape X2 is read, and the reference and definition line numbers are stored into blocks for each symbol.

The symbol table is then searched for the lowest remaining entry, using a modified linear search technique, and the listing is formatted and output. When the symbol table becomes empty the core requirements for any remaining symbol reference line numbers is determined and these are read and processed. Again, if there is insufficient space for all of them they are taken in segments.

When all symbols have been listed, control returns to MONARCH.

XDS PROGRAM LIBRARY PROGRAM DESCRIPTION

900 Series: 850086
850088
Catalog No. 9300: 860083

IDENTIFICATION: CONCRD

PURPOSE: To process the exception control records, scan the encoded program file, generate the concordance scratch output file, determine space requirements needed to retain the reference lines number for each symbol, and expand the symbols to appear on the concordance output from their encoded format.

ACTION: CONCRD performs the following functions:

1. CONCRD determines the locations of the unit assignment table entries for the various I/O functions and calls INIT to initialize the I/O routines.
2. GETXC is called to process the INCLUDE and EXCLUDE records.
3. RECON is called to initialize the parameters to process the encoded input file.
4. The LINE routine is called to process the encoded input file, determine which symbols to include in the listing, reconstruct the symbols to be included, output the scratch tape X2 and maintain the reference line number storage requirements for each symbol.
5. CONCRD rewinds the encoded input tape X1 and scratch tape X2 before calling the tape loader to load the second overlay of the concordance routine X, CON2.

PROGRAMMING TECHNIQUES: I/O assignments are determined from the unit assignments maintained by MONARCH. CONCRD overlays 2 cells of the tape loader in order to reset the calling locations of the typewriter error message routine and the abort routine used by the loader. CONCRD is given three words of control information by FINISH, which are located in lower core. CONCRD is an absolute program, part of which is origined just below the start of the encoded dictionary; this part is initialization code that may be destroyed after the initialization process is completed. CONCRD is coded in 910-925 subset code.

CALLING SEQUENCE: Control is transferred to CONCRD by the tape loader upon completion of the loading process.

MEMORY REQUIREMENTS: CONCRD uses all available memory.

SUBROUTINES USED:
REWW
INIT
GETXC
RECON
LINE

XDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 850086
850088
Catalog No. 9300: 860083

IDENTIFICATION: LINE

PURPOSE: To obtain and process lines of encoded program text, generate the symbol table of reconstructed symbols, output the reference and definition line numbers and pointers to the scratch tape, and maintain a count of the memory storage requirements associated with each symbol.

ACTION: LINE calls PLBL to obtain the label and operation bytes for the line. The label, if any, is tested for inclusion into the concordance listing and, if it is to be included, the symbol is entered into the symbol table. The byte table pointer is changed to point to the symbol table entry, the line number of the label is output to the scratch tape, followed by a pointer word indicating the location of the symbol table entry.

The operation field is processed and tested for special action (PROC, FUNC, NAME, END, TEXT, BCD, etc.). If the operation is to appear in the listing it is counted as a reference to the appropriate symbol. If the symbol is not in the symbol table it is inserted.

The operand field is then scanned by calling VFLD and the reference line number is output, followed by the symbol pointer word. If the referenced symbol does not appear in the symbol table it is inserted, together with a flag indicating that the symbol definition is unknown.

Comments are skipped by calling SKIP.

Line continues processing text lines until the program END line has been processed.

PROGRAMMING TECHNIQUES:	LINE is an open routine called by the CONCRD program and assembled as part of CONCRD.
CALLING SEQUENCE:	BRU LINE return is to location STOP in CONCRD.
MEMORY REQUIREMENTS:	364_8 cells plus constants.
SUBROUTINES USED:	PLBL STC TSTTYP COMP TSTOP OUTPUT GTDC TSTEX GET VFLD SKIP

XDS PROGRAM LIBRARY PROGRAM DESCRIPTION

900 Series: 850086
850088
Catalog No. 9300: 860083

IDENTIFICATION: TSTTYP

PURPOSE: To determine the type of label definition.

ACTION: TSTTYP locates the symbol table entry generated by META-SYMBOL for a given symbol and from this determines the type of definition associated with a symbol.

PROGRAMMING TECHNIQUES: TSTTYP is a closed routine assembled as part of CONCRD.

CALLING SEQUENCE:

byte number to LBYTE
byte table entry to LBCDE
    BRM TSTTYP
on exit the type code is in LTYPE

MEMORY REQUIREMENTS: 66_8 cells plus constants.

SUBROUTINES USED: None

XDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 850086
850088
Catalog No. 9300: 860083

IDENTIFICATION: PLBL

PURPOSE: To process the line of text through the operation field retaining the byte number and byte table entry for the label and operation code.

ACTION: PLBL obtains the bytes for the line of text by calling GTB. The label and operation code bytes are retained, as is the byte table entry for each. The current sample level is retained as the label level unless the label is external, in which case it is reduced by one. The first nonblank byte of the operand field is obtained to be analyzed by the VFLD routine.

PROGRAMMING TECHNIQUES: PLBL is a closed routine assembled as part of CONCRD.

CALLING SEQUENCE:
BRM PLBL
end of line return
normal return

MEMORY REQUIREMENTS: 111_8 cells plus constants and storage cells.

SUBROUTINES USED:
GTB
GTDC

XDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 850086
850088
Catalog No. 9300: 860083

IDENTIFICATION: TSTOP

PURPOSE: To reconstruct the operation code and test it for certain operations (PROC, FUNC, NAME, END, TEXT, BCD, FORM, POPD, OPD).

ACTION: TSTOP obtains the symbolic operation code and tests it against a list of directives. If the operation matches, control goes to the code to process that class of directive. PROC and FUNC cause the sample level to be incremented and the label type to be set to list. END decrements the sample level. Name decrements the label level by 1 and sets the label type to operator. FORM, POPD, and OPD set the label type to operator. BCD and TEXT cause flags to be set to prevent the BCD message from being interpreted as symbolics.

PROGRAMMING TECHNIQUES: TSTOP is a closed routine assembled as part of CONCRD.

CALLING SEQUENCE: BRM TSTOP

MEMORY REQUIREMENTS: 166_8 cells plus constants and storage cells.

SUBROUTINES USED: GTDC
STC
GET

XDS PROGRAM LIBRARY PROGRAM DESCRIPTION

900 Series: 850086
850088
Catalog No. 9300: 860083

IDENTIFICATION: TSTEX

PURPOSE: To test each operator that is to be deleted from the listing and, if it is not critical (see TSTOP), to purge the entry from the byte table.

ACTION: TSTEX tests the operation code against a list of special directives. If the operation is not any of these, the byte table entry for the symbol is set to zero.

PROGRAMMING TECHNIQUES: TSTEX is a closed routine assembled as part of CONCRD.

CALLING SEQUENCE:

location of symbol to CLOC
symbol length to A register

```
      BRM TSTEX
```

MEMORY REQUIREMENTS: 33_8 cells plus constants and storage cells.

SUBROUTINES USED: None

XDS PROGRAM LIBRARY PROGRAM DESCRIPTION

900 Series: 850086
850088
Catalog No. 9300: 860083

IDENTIFICATION: VFLD

PURPOSE: To process the operand fields of the program text.

ACTION: VFLD scans the operand field for symbolic items. As they are encountered they are tested for inclusion in the concordance listing. If the symbol is to appear in the listing and has not been previously encountered, it is reconstructed and inserted into the symbol table. The space requirement (one or two halfwords) is tallied in the symbol control word and the location of the symbol is output to the scratch file. If the line number of the current line has not been output to the scratch file, it is output preceding the symbol table pointer. Alphanumeric data is skipped and, if the line is a TEXT or BCD line, only the count field is processed.

PROGRAMMING TECHNIQUES: VFLD is a closed routine assembled as part of CONCRD.

CALLING SEQUENCE:
First byte of field to NBYT
First byte table entry to BCW

    BRM VFLD

MEMORY REQUIREMENTS: 210_8 cells plus constants and storage cells.

SUBROUTINES USED:

GTDC	SKPQT
STC	GET
COMP	TSTTYP
GTB	TSTEX
OUTPUT	

XDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 850086
850088
Catalog No. 9300: 860083

IDENTIFICATION: COMP

PURPOSE: To compare a symbol with the entries in a table of symbols.

ACTION: COMP compares a symbol at CLOC with length SLNG with the entries in a table of symbols at CMTB. CMLN gives the numbers of symbols in the table CMTB.

PROGRAMMING TECHNIQUES: COMP is a closed routine assembled as part of CONCRD.

CALLING SEQUENCE:

symbol to CLOC
length to SLNG
table address to CMTB
table length to CMLN
    BRM COMP
not found exit
symbol found exit
on exit cell TEMP+2 contains the location of the symbol entry if found.

MEMORY REQUIREMENTS: 51_8 cells plus constants and storage cells.

SUBROUTINES USED: None

XDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 850086
850088
Catalog No. 9300: 860083

IDENTIFICATION: SKPQT

PURPOSE: To skip an alphanumeric constant until an apostrophe (') is encountered.

ACTION: SKPQT obtains bytes by calling GTB until an apostrophe is obtained.

PROGRAMMING TECHNIQUES: SKPQT is a closed routine assembled as part of CONCRD.

CALLING SEQUENCE:

BRM SKPQT

end of line return

normal return

MEMORY REQUIREMENTS: 16_8 cells plus constants.

SUBROUTINES USED:

GTB
GTDC

XDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 850086
850088
Catalog No. 9300: 860083

IDENTIFICATION:	SKIP
PURPOSE:	To skip to the end of lines of text, including any comments.
ACTION:	SKIP calls GTB until an end of line is detected; it then calls GCM until the comments have been passed.
PROGRAMMING TECHNIQUES:	SKIP is a closed routine assembled as part of CONCRD.
CALLING SEQUENCE:	BRM SKIP
MEMORY REQUIREMENTS:	23_8 cells plus constants and storage cells
SUBROUTINES USED:	GTB GCM

XDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 850086
850088
Catalog No. 9300: 860083

IDENTIFICATION: GTDC

PURPOSE: To get the first symbolic character for a byte, given the table entry for the byte.

ACTION: GTDC stores the symbol length for the byte in LN, sets CNT to the previous dictionary character position for STC. The location of the dictionary entry is then determined and the first word of symbolics obtained, positioned, and placed in DWRD. GTC is called to extract the first character of the entry, which is placed in CHR and the A register at exit.

PROGRAMMING TECHNIQUES: GTDC is a closed subroutine assembled as part of CONCRD.

CALLING SEQUENCE:

byte table entry to BCW
    GRM GTDC
exit character in CHR and A register

MEMORY REQUIREMENTS: 44 octal cells plus constants and storage cells.

SUBROUTINES USED: GTC

XDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 850086
850088
Catalog No. 9300: 860083

IDENTIFICATION: GET

PURPOSE: To get the second and following symbolic characters for a byte.

ACTION: GET decrements the character count, LN, and, if the string is empty, exits through the end of string exit. If additional characters remain, GET calls GTC to obtain the next character, which is placed in CHR and the A register on a normal exit.

PROGRAMMING TECHNIQUES: GET is a closed subroutine assembled as part of CONCRD.

CALLING SEQUENCE:
BRM GET
end of string exit
normal exit

MEMORY REQUIREMENTS: 13_8 cells plus constants and storage cells.

SUBROUTINES USED: GTC

XDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 850086
850088
Catalog No. 9300: 860083

IDENTIFICATION: OUTPUT

PURPOSE: To output a word to the scratch file on unit X2.

ACTION: OUTPUT stores the contents of the A register into the output buffer OBUF. If the buffer is filled, the WMTB routine is called to write the buffer to the scratch file. The buffer is then cleared to zero and the location for the next data word is initialized.

PROGRAMMING TECHNIQUES: OUTPUT is a closed routine assembled as part of CONCRD. It assumes a standard I/O calling sequence to call the tape write routine.

CALLING SEQUENCE: word to output to A register

BRM OUTPUT

MEMORY REQUIREMENTS: 22_8 cells plus constants and storage cells.

SUBROUTINES USED: I/O routine associated with writing scratch tape, WMTB.

XDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 850086
850088
Catalog No. 9300: 860083

IDENTIFICATION: CLOSE

PURPOSE: To close the scratch output file X2.

ACTION: CLOSE empties the output buffer OBUF by calling the WMTB routine and then writes an end-of-file on X2.

PROGRAMMING TECHNIQUES: CLOSE is a closed routine assembled as part of CONCRD. CLOSE uses standard I/O calling sequences to perform the I/O functions.

CALLING SEQUENCE: BRM CLOSE

MEMORY REQUIREMENTS: 15_8 cells plus constants and storage cells.

SUBROUTINES USED: WMTB
EFMT

XDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 850086
850088
Catalog No. 9300: 860083

IDENTIFICATION:	RECON
PURPOSE:	To initialize parameters for reading the encoded input file.
ACTION:	RECON initializes the input buffer location, byte size, byte table location, and related parameters for interpreting the encoded text file, X1.
PROGRAMMING TECHNIQUES:	RECON is a closed routine assembled as part of CONCRD.
CALLING SEQUENCE:	BRM RECON
MEMORY REQUIREMENTS:	37_8 cells plus constants and storage cells.
SUBROUTINES USED:	None

XDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 850086
850088
Catalog No. 9300: 860083

IDENTIFICATION: GTC

PURPOSE: To get the next symbolic character from the specified location.

ACTION: GTC extracts the next character from DWRD. If DWRD is empty as determined by CNT, the next word is obtained from the location address by BUF. If the buffer is empty (which is not possible when obtaining characters from the dictionary) the next input record is obtained by calling INPUT.

PROGRAMMING TECHNIQUES: GTC is a closed routine assembled as part of CONCRD.

CALLING SEQUENCE:

Number of characters in string, - 1 to CNT
word containing next character, left-adjusted in DWRD
location of word containing character to BUF

```
    BRM GTC
```

on exit the character is in CHR and the A register. CNT, DWRD, and BUF are reset to obtain the next character.

MEMORY REQUIREMENTS: 55_8 cells plus constants and storage cells.

SUBROUTINES USED: INPUT

XDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 850086
850088
Catalog No. 9300: 860083

IDENTIFICATION: GTB

PURPOSE: To obtain the next byte of encoded input from the file on X1.

ACTION: GTB extracts the next BSZ bits from BWRD. When BWRD becomes empty, the next word is taken from the location given by BLOC. When the buffer becomes empty, INPUT is called to obtain the next encoded record. GTB steps the byte size when a zero byte is encountered.

PROGRAMMING TECHNIQUES: GTB is a closed routine assembled as part of CONCRD.

CALLING SEQUENCE: BRM GTB

on exit BCW contains the byte table entry for the byte, BYT contains the byte number, NBYT contains the negative of the byte number. The contents of BCW are in the B register, the byte number is in the A register, and the X register contains NBYT.

MEMORY REQUIREMENTS: 103_8 cells plus constants and storage cells.

SUBROUTINES USED: INPUT

XDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 850086
850088
Catalog No. 9300: 860083

IDENTIFICATION: GCM

PURPOSE: To obtain comment characters from the encoded input file.

ACTION: GCM gets the next six bits of encoded information from the encoded input file. BWRD contains the current encoded word addressed by BLOC. BIT contains the number of bits BWRD which have been used. INPUT is used to obtain the next encoded record when the input buffer becomes empty.

PROGRAMMING TECHNIQUES: GCM is closed routine assembled as part of CONCRD.

CALLING SEQUENCE: BRM GCM

MEMORY REQUIREMENTS: 50_8 cells plus constants and storage cells.

SUBROUTINES USED: INPUT

XDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 850086
850083
Catalog No. 9300: 860083

IDENTIFICATION:	STC
PURPOSE:	To store the character in the A register into the character position indicated by SCHR in the word addressed by SLOC.
ACTION:	STC positions the character in the A register to the character position indicated by SCHR and adds the character to the word addressed by SLOC. When the word becomes filled, SLOC is incremented and the new location is cleared.
PROGRAMMING TECHNIQUES:	STC is a closed routine assembled as part of CONCRD.
CALLING SEQUENCE:	character position to SCHR word position to SLOC character to A register BRM STC
MEMORY REQUIREMENTS:	23_8 cells plus constants and storage cells
SUBROUTINES USED:	None

XDS PROGRAM LIBRARY PROGRAM DESCRIPTION

900 Series: 850086
850088
Catalog No. 9300: 860083

IDENTIFICATION:	INPUT
PURPOSE:	To read and checksum an encoded input record from X1.
ACTION:	INPUT reads a maximum 40-word record from X1 into the encoded input buffer CBFE and checksums the image.
PROGRAMMING TECHNIQUES:	INPUT is a closed routine assembled as part of CONCRD. INPUT uses the standard META-SYMBOL calling sequence to call RMTB.
CALLING SEQUENCE:	BRM INPUT end of file exit normal exit
MEMORY REQUIREMENTS:	47_8 cells plus constants, storage cells, and buffer.
SUBROUTINES USED:	RMTB

XDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 850086
850088
Catalog No. 9300: 860083

IDENTIFICATION: GETXC

PURPOSE: To process the concordance exception control records (INCLUDE, EXCLUDE, and ΔEOF).

ACTION: GETXC initializes the cells to locate the lists of exclusions or inclusions, then tests to see if exceptions are to be processed. If there are no exceptions, control returns to CONCRD; otherwise the exceptions are processed and tables of symbols to be excluded and/or included are built. GSYM is called to obtain the symbols on the control card. The appearance of *ALL results in flags (NONE for an EXCLUDE and ALL for an INCLUDE) being set, indicating a general exception.

PROGRAMMING TECHNIQUES: GETXC is a closed routine assembled as part of CONCRD. It is origined in middle core to be overlaid by tables after it has been executed.

CALLING SEQUENCE: BRM GETXC

MEMORY REQUIREMENTS: 171_8 cells, all resuable, plus constants and storage cells.

SUBROUTINES USED: GSYM TYPMSG
I/O routine associated with symbolic input.

XDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 850086
850088
Catalog No. 9300: 860083

IDENTIFICATION:	GSC
PURPOSE:	To get the next symbolic character of the exception control record.
ACTION:	GSC extracts the next character from the symbolic input buffer and steps the indicators to obtain the next character.
PROGRAMMING TECHNIQUES:	GSC is a closed routine assembled as part of CONCRD. It is origined in middle core to be overlaid by tables.
CALLING SEQUENCE:	BRM GSC end of line exit normal exit
MEMORY REQUIREMENTS:	27_8 cells, all reusable, for table plus constants and storage cells.
SUBROUTINES USED:	None

XDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 850086
850088
Catalog No. 9300: 860083

IDENTIFICATION:	GSYM
PURPOSE:	To obtain the next symbol from the exception control record.
ACTION:	GSYM calls GSC to obtain characters from the control record. Leading blanks are ignored. COMMA, blank, or end of record terminate the symbol. STC is called to pack the characters into core. The symbol size is set in SIZE.
PROGRAMMING TECHNIQUES:	GSYM is a closed routine assembled as part of CONCRD. It is origined in middle core to be overlaid after the exception records have been processed.
CALLING SEQUENCE:	BRM GSYM
MEMORY REQUIREMENTS:	25_8 cells, all reusable, plus constants and storage cells.
SUBROUTINES USED:	STC GSC

I/O AND I/O INITIALIZATION ROUTINES

The input/output device routines used in CONCRD and their attendant initialization routines are basically a subset of the routines found in MSCONTRL, ENCODER, and other portions of META-SYMBOL.

Unit and channel assignment are taken from the Unit Assignment Table maintained by MONARCH. To find unit assignments, the contents of cell 1, which is set by MONARCH, is used as an index to the table location.

XDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 850087
850089
Catalog No. 9300: 860084

IDENTIFICATION: CON2

PURPOSE: To determine the space needed for each symbol to retain all reference line numbers for the symbol, to read the concordance scratch tape X2, to build the reference line number table in accordance with the space allocation, to search the symbol table for the alphanumeric sequence in which to print the concordance, and to edit and print the concordance listing.

ACTION: CON2 calls the allocation routine ALLOC to determine which symbols are to be processed in this edit pass, and to allocate the storage requirements for the reference line numbers associated with each symbol. STRNO is then called to read the scratch tape X2 and to store the reference and definition line numbers into blocks, each of which contains all the line numbers associated with a given symbol. SRCH is then called to fetch the lowest entry in the table and EDIT is called to format and print the concordance listing for the symbol. When each symbol is output, its symbol table entry is purged. When all symbols have been output, control returns to MONARCH.

PROGRAMMING TECHNIQUES: Communication between CONCRD and CON2, which are separate core overlays, is maintained in locations between 200_8 and 300_8. The program CON2 has an absolute origin that starts at location 300_8.

CALLING SEQUENCE: Control is transferred to CON2 by the tape loader when the program has been loaded. Control returns to MONARCH when the concordance listing has been completed.

MEMORY REQUIREMENTS:	All available core storage.
SUBROUTINES USED:	REWW STLNO INPRT SRCH ALLOC EDIT I/O routine to perform end-of-file action on the listing output (EFMT, HOME, or THOME).

XDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 850087
850089
Catalog No. 9300: 860084

IDENTIFICATION: ALLOC

PURPOSE: To allocate the available memory to allow space for the references to each symbol to be stored together in a single block of core. If space is not available for all references, to determine the number of symbols for which space is available. To set a parameter indicating which symbols are to be included in this edit pass and allocate core accordingly.

ACTION: ALLOC scans the symbol table established by CONCRD and determines the space needed for reference line numbers for the concordance. If the space needed is greater than that available, those symbols appearing last in the collating sequence are dropped and a recount is made. This process is repeated until a subset of the symbols that appear at the beginning of the collating sequence has been selected and can be processed with the available storage capacity. ALLOC then scans the symbol table, and for each symbol which is to appear in this edit pass sets a pointer to the first location for the symbol's reference line number block. An initial entry is then made in the block, indicating the location (relative) in which to store the line number containing the next reference to the symbol. ALLOC exits when the linkages have all been set.

PROGRAMMING TECHNIQUES: ALLOC sets a pointer to a table of masks. Any symbol that has an absolute value larger than the indicated mask is excluded from this edit pass. ALLOC is a closed routine assembled as part of CON2.

CALLING SEQUENCE: BRM ALLOC

MEMORY REQUIREMENTS: 134_8 cells plus constants and storage cells.

SUBROUTINES USED: None

XDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 850087
850089
Catalog No. 9300: 860084

IDENTIFICATION: STLNO

PURPOSE: To read the concordance scratch tape X2 and to establish the reference line number table.

ACTION: STLNO reads the scratch tape X2 by calling the magnetic tape read routine RMTB. The data is then processed and the reference and definition line numbers for each symbol are stored in the space allocated for them. When entering line numbers, only those symbols which are less than the allocation mask are considered.

PROGRAMMING TECHNIQUES: STLNO uses the standard META-SYMBOL call sequence to call the RMTB I/O routine. STLNO is a closed routine assembled as part of CON2.

CALLING SEQUENCE: BRM STLNO

MEMORY REQUIREMENTS: 142_8 cells plus constants and storage cells.

SUBROUTINES USED: RMTB

XDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 850087
850089
Catalog No. 9300: 860084

IDENTIFICATION: SRCH

PURPOSE: To obtain the lowest entry in the symbol table.

ACTION: SRCH is a modified linear search routine capable of comparing variable length entries. When SRCH is entered, the origin of the symbol table is entered in LAST and the previous contents of LAST are placed in STRT as the location of the first symbol to consider. Symbols following STRT are then compared to the symbol addressed by STRT until an entry is found that precedes STRT in the collating sequence. The contents of STRT are then moved to LAST and the location of the lower entry is placed in STRT. When the end of the table is reached, the routine exits with STRT pointing to the lowest entry.

PROGRAMMING TECHNIQUES: SRCH is a closed routine assembled as part of CON2.

CALLING SEQUENCE: BRM SRCH

on exit STRT points to lowest symbol

MEMORY REQUIREMENTS: 174_8 cells plus constants and storage cells.

SUBROUTINES USED: None

XDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 850087
850089
Catalog No. 9300: 860084

IDENTIFICATION: EDIT

PURPOSE: To format the line images for the concordance listing and to cause the line to be written to the listing output media.

ACTION: EDIT extracts the symbol type from the symbol table entry and translates this to a one- or two-character alphanumeric type flag. The defining line number is converted to BCD code and inserted into the image. The symbol is moved into the print buffer and padded with trailing blanks. The reference line numbers are obtained, converted to BCD by calling CNVRT, and inserted into the image by calling STRNO. When the entire list of references has been processed, any partial line image is output by calling the listing output routine, the buffer is set to blanks, and an exit is made from EDIT.

PROGRAMMING TECHNIQUES: EDIT is a closed routine assembled as part of CON2. The standard META-SYMBOL call sequence is used to call the listing output routine.

CALLING SEQUENCE:
location of symbol table entry to STRT
location of symbol to CFT
    BRM EDIT

MEMORY REQUIREMENTS: 160_8 locations plus constants and storage cells.

SUBROUTINES USED: CNVRT STRNO
I/O routine associated with listing output (PRNT, TYPWRT, WMTB).

XDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 850087
850088
Catalog No. 9300: 860084

IDENTIFICATION: STRNO

PURPOSE: To insert reference line numbers into the concordance print image and to cause the line image to be output when filled.

ACTION: STRNO places the reference line number contained in the A register on entry into the next available position in the print image: if the line image is complete, the listing output routine is called to write the line and the buffer is cleared to blanks.

PROGRAMMING TECHNIQUES: STRNO is a closed routine assembled as part of CON2.

CALLING SEQUENCE: line number to A register

BRM STRNO

MEMORY REQUIREMENTS: 43_8 cells plus constants and storage cells.

SUBROUTINES USED: I/O routine for listing output (PRNT, TYPWRT, WMTB).

XDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

900 Series: 850087
850089
Catalog No. 9300: 860084

IDENTIFICATION:	CNVRT
PURPOSE:	To convert binary line number to BCD with lead blanks.
ACTION:	By successive division, CNVRT generates the BCD number from the binary number contained in the A register on entry. The BCD result is then edited and lead zeros are replaced by blanks. Results are left in the A register.
PROGRAMMING TECHNIQUES:	CNVRT is a closed routine assembled as part of CON2.
CALLING SEQUENCE:	binary number to A register BRM CNVRT
MEMORY REQUIREMENTS:	40_8 cells plus constants and storage cells.
SUBROUTINES USED:	None

I/O ROUTINES AND INITIALIZATION ROUTINES

The input/output routines and attendant initialization routines used in CON2 are basically the same as those used in MSCONTRL, PAS2, and other portions of META-SYMBOL. Unit and channel assignments are taken from the Unit Assignment Table maintained by MONARCH.

SECTION 7

ITEM AND TABLE FORMATS USED BY THE CONCORDANCE PROGRAM

SYMBOL TABLE ENTRY FORMAT

	0	1	2 - 5	6 - 19	20 - 23
control word	D	H	T	W	L
symbol – from 1-4 words left-adjusted with trailing zeros					

where

D is a one-bit flag, 1 if symbol definition line number is unknown.

H is not used.

T is type code:

0 – absolute

1 – relocatable

2 – list

3 – operation

4 – external-absolute

5 – external-relocatable

6 – external-list

7 – external-operation

W is the number of halfwords of reference line numbers.

L is the number of characters in symbol.

After the ALLOC routine has been executed, the control word is given the following format:

0 - 1	2 - 5	6 - 19	20 - 23
H	T	A	L

where

H is the high 2 bits of the definition line number.

T is type, as above.

A is the address of first word of reference line number block for this symbol.

L is length, as above.

If the first word of the symbol is zero, the symbol has been previously output to the concordance listing.

REFERENCE LINE NUMBER BLOCK FORMAT

0 11	12 23
DEF	SIZE
NO.	NO.
NO.	NO.
NO.	

where

DEF is the low 12 bits of the definition line number.

SIZE is the number of words in the block.

NO. are reference line numbers packed two per word unless the line number is greater than $2^{12}2$, in which case the high 10 bits all contain 1's and the line number is in the low 14 bits.

RECORD FORMAT OF CONCORDANCE SCRATCH TAPE X2

The record size maximum is 40 words. Words have the following format:

0 1	2 8	9 23
L	D	A

where

L is the line number flag. If L = 1, the line number is A. If L is zero, A is the location in the symbol table of a symbol entry.

D is the definition flag if L = 0 and D is not zero. The symbol at A was defined at the last preceding line number. If L = 0 and D = 0 the symbol at A was referenced at the last preceding line number.

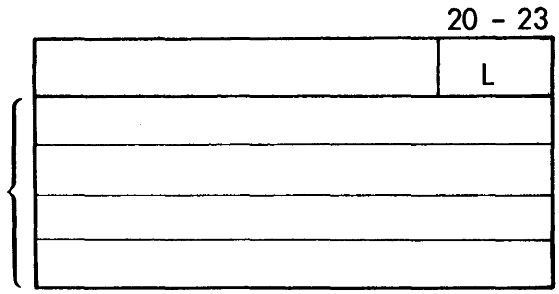
BYTE TABLE FORMAT

During the concordance run, entries in the byte table are modified to reflect definitions of concordance symbols. If a byte table entry is zero, the symbol represented by the entry is to be excluded from the corcordance.

When a symbol is inserted into the concordance symbol table, the associate field (bits 9 to 23) of the byte table entry is modified to point to the new symbol location.

EXCLUDE AND INCLUDE TABLE FORMATS

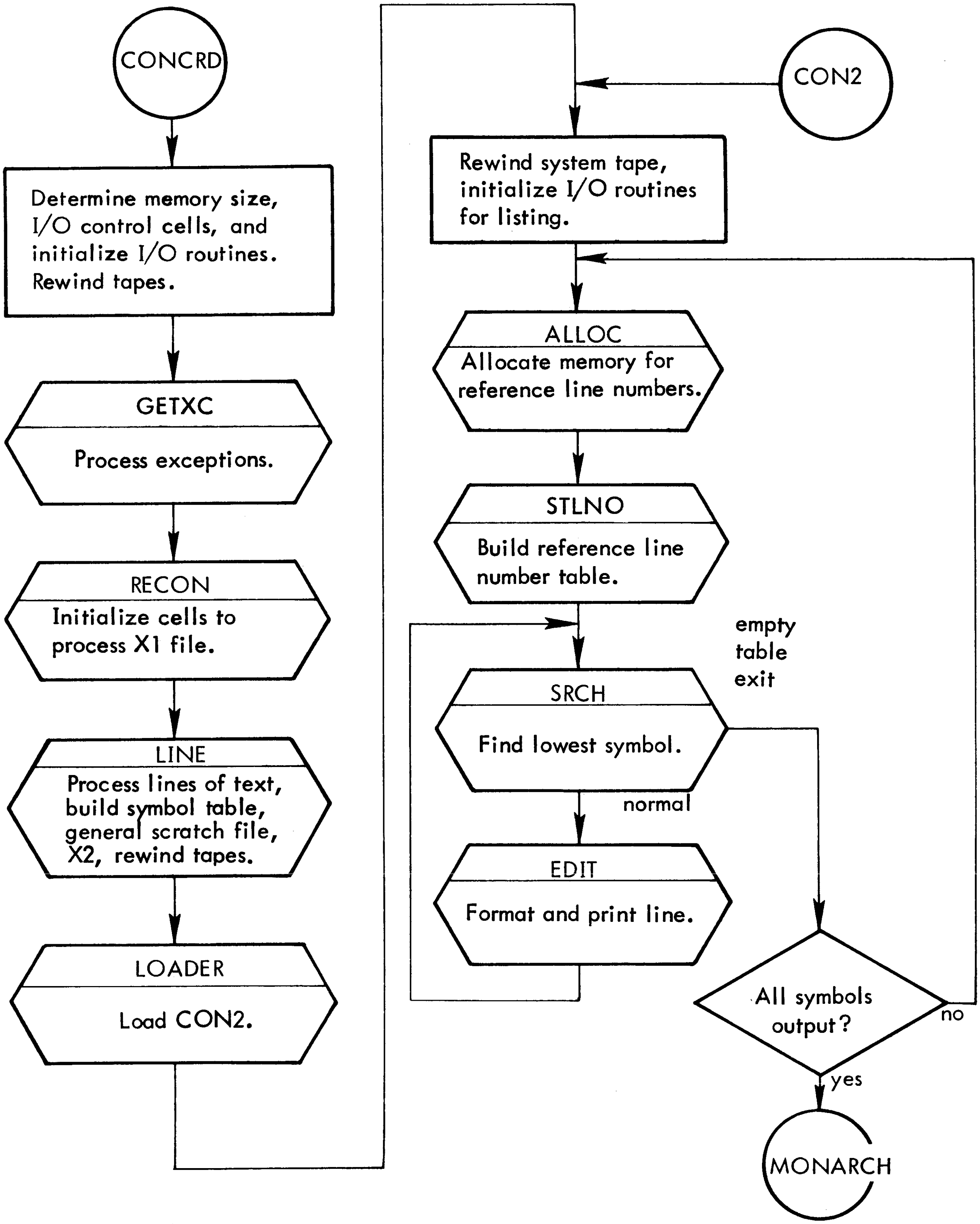
symbol – from 1-4 words with trailing zero characters



where

L is the symbol length in characters.

CONCORDANCE OVERALL FLOW



LISTING OF SUBROUTINES

Subroutine	Contained In	Page Des	Page Chart
ABORT	MSCONTRL	3-12	3-32
AORG	PAS1	3-188	3-151
	PAS2	3-188	3-353
BCD	PAS1	3-202	3-364
	PAS2	3-202	3-364
CHAR	ENCODER	3-52	3-82
CLOSE	MSCONTRL	3-8	3-34
CNVRT	PAS1	3-226	3-296
	PAS2	3-226	3-399
	PREA	3-132	3-147
CRD	ENCODER	3-67	3-91
CRDB	ENCODER	3-68	3-92
CRDH	ENCODER	3-69	3-92
DEC	ENCODER	3-45	3-77
DED	PAS1	3-220	3-277
	PAS2	3-220	3-380
DELETE	ENCODER	3-46	3-77
DFLST	PAS1	3-198	3-259
	PAS2	3-198	3-360
DO	PAS1	3-193	3-255
	PAS2	3-193	3-356
DOAGN	PAS1	3-194	3-256
	PAS2	3-194	3-357
DODEC	PAS1	3-195	3-257
DPDIV	PAS1	3-227	3-295
	PAS2	3-227	3-398
	PREA	3-134	3-149
EDC	ENCODER	3-72	3-93
	PAS1	3-213	3-273
	PAS2	3-213	3-373
EDE	PAS1		3-272
	PAS2	3-321	3-371
EDF	PAS2	3-323	3-372
EDIT	PAS1		3-272
	PAS2	3-318	3-370
EDL	PAS2	3-320	3-372
EDR	PAS2	3-322	3-372
EDS	ENCODER	3-73	3-93
EDS	PAS1	3-214	3-273
	PAS2	3-214	3-373
EDTL	PAS1		3-272
	PAS2	3-320	3-371
EDTST	PAS1		3-267
	PAS2	3-317	3-366
EDTV	PAS1		3-272
	PAS2	3-319	3-371
EFC	MSCONTRL	3-29	3-38
EFMT	MSCONTRL	3-21	3-37
EFPT	MSCONTRL	3-17	3-33
END	PAS1	3-199	3-260
	PAS2	3-313	3-361
EPRNT	PAS1	3-219	3-276
	PAS2	3-219	3-379
EQU	PAS1	3-187	3-251
	PAS2	3-308	3-353
FETCH	PREA	3-137	3-150
FLDC	PAS1		3-272
	PAS2	3-324	3-373
FLM	PAS1	3-224	3-279
	PAS2	3-224	3-382
FLN	PAS1	3-223	3-279
	PAS2	3-223	3-282
FLUSH	PAS1	3-216	3-275
	PAS2	3-216	3-376
FNRL	PAS1	3-196	3-258
	PAS2	3-311	3-359
FORM	PAS1	3-189	3-252
	PAS2	3-189	3-354
FRL	PAS1	3-201	3-263
	PAS2	3-315	3-362
FUNC	PAS1	3-190	3-253
	PAS2	3-309	3-355
GBC	PREA	3-123	none
GBSL	PAS1	3-237	3-294
	PAS2	3-237	3-397
GBW	PREA	3-128	3-145
GCW	PAS1	3-181	3-249
	PAS2	3-181	3-351

		Page				Page	
Subroutine	Contained In	Des	Chart	Subroutine	Contained In	Des	Chart
GCW	PREA	3-126	3-145	INRDT	ENCODER	3-70	3-92
GEC	PAS1	3-183	3-250	INRPT	ENCODER	3-64	3-91
	PAS2	3-183	3-352	INTYP	PAS2	3-334	3-378
GET	PAS1	3-236	3-294	IPL	PAS1	3-177	3-247
	PAS2	3-236	3-397		PAS2	3-177	3-349
	S4B	3-105	3-113	ITMOV	SRNK	3-162	3-166
GIT	PAS1	3-230	3-287				
	PAS2	3-339	3-390	LBTST	PAS1	3-184	3-249
GLOP	PAS1	3-239	3-278		PAS2	3-184	3-351
	PAS2	3-239	3-381	LNCT	PAS2	3-332	3-378
GLOV	PAS1	3-221	3-278				
	PAS2	3-221	3-381	M3WAI	PAS1	3-222	3-278
GNC	PAS1	3-235	3-293	MBYT	PAS1	3-178	3-248
	PAS2	3-235	3-396		PAS2	3-306	3-350
GPDC	PREA	3-135	3-150	MFOI	PAS1		3-277
GTB	PAS1	3-182	3-250		PAS2	3-335	3-380
	PAS2	3-182	3-352	MIFT	PAS1	3-238	3-279
	PREA	3-127	3-145		PAS2	3-238	3-382
GTCHR	PREA	3-133	3-149	MOVE	PAS1	3-206	3-266
GTCR	SRNK	3-160	3-165		PREA	3-125	3-144
GTLBL	FNSH	3-208	3-402		S4B	3-103	3-113
	PAS1	3-208	3-268		SRNK	3-125	3-144
	PAS2	3-208	3-402	MRKBYT	PREA	3-131	3-147
GTUNT	MSCONTRL	3-30	3-38	MVITM	SRNK	3-158	3-165
				MVPRC	PAS1	3-206	3-266
HOME	PAS1		3-272		PREA	3-125	3-144
	PAS2	3-328	3-374	MVTAB	ENCODER	3-61	3-90
IAW	MSCONTRL	3-27	3-38	NAME	PAS1	3-205	3-265
IN	ENCODER	3-59	3-88		PAS2	3-309	3-355
INC	ENCODER	3-54	3-83		S4B	3-101	3-112
	PAS1	3-180	3-249	NSRT	ENCODER	3-57	3-86
	PAS2	3-180	3-351		PAS1	3-210	3-270
	PREA	3-130	3-146		PAS2	3-210	3-368
INCRD	ENCODER	3-66	3-91	NUM	S4B	3-100	3-109
INEFC	MSCONTRL	3-28	3-38				
INEFPT	MSCONTRL	3-16	3-33	OCTC	S4B	3-99	3-109
INIT	ENCODER	3-48	3-78	OPEN	MSCONTRL	3-6	3-33
INPCB	MSCONTRL	3-23	3-37	ORG	PAS1	3-188	3-251
INPCH	MSCONTRL	3-25	3-38		PAS2	3-188	3-353
INPPT	MSCONTRL	3-14	3-32	OUT	ENCODER	3-60	3-88
INPUT	MSCONTRL	3-10	3-34	OUTC	ENCODER	3-55	3-83

Subroutine	Contained In	Page Des	Page Chart	Subroutine	Contained In	Page Des	Page Chart
OUTP	PAS1	3-215	3-274	RELTST	PAS2	3-225	3-398
	PAS2	3-215	3-375	RES	PAS1	3-188	3-252
OUTPUT	MSCONTRL	3-7	3-33		PAS2	3-188	3-354
				RESET	ENCODER	3-62	3-90
PACK	PREA	3-138	3-150		PAS1	3-217	3-275
PAGE	PAS1	3-218	3-276		PAS2	3-217	3-376
	PAS2	3-218	3-379		S4B	3-104	3-113
PARAMS	S4B	3-102	3-112	REWW	MSCONTRL	3-13	3-32
PBC	MSCONTRL	3-24	3-38	RMTB	MSCONTRL	3-22	3-37
PCB	MSCONTRL		3-37	RMTBU	MSCONTRL	3-18	3-35
PCH	MSCONTRL	3-26	3-38				
PEEK	PAS1	3-234	3-293	SAM	PAS1	3-203	3-264
	PAS2	3-234	3-396		PAS2	3-309	3-355
PI (RDPD)	PREA	3-136	3-150	SAMPLE	SRNK	3-163	3-167
PLB	PAS1	3-185	3-245	SCAN	PAS1	3-228	3-280
	PAS2	3-307	3-348		PAS2	3-337	3-383
PLINE	PAS1		3-272	SCANC	PAS1	3-232	3-292
	PAS2	3-327	3-374		PAS2	3-232	3-395
PLTST	PAS1	3-186	3-246	SCRP	PAS1	3-212	3-271
POP	PAS1	3-201	3-264		PAS2	3-212	3-369
	PAS2	3-316	3-365	SKIP	ENCODER	3-47	3-78
POPD	PAS1	3-191	3-254		PAS1	3-179	3-249
	PAS2	3-310	3-355		PAS2	3-179	3-351
POPR	PAS1	3-192	3-254	SMPTRN	SRNK	3-159	3-166
	PAS2	3-310	3-355	SRCH	ENCODER	3-56	3-84
PPTB	MSCONTRL	3-15	3-33		PAS1	3-209	3-269
PRL	PAS1	3-196	3-258		PAS2	3-209	3-367
	PAS2	3-311	3-359	STCR	SRNK	3-161	3-165
PRNT	PAS1		3-272	STORE	ENCODER	3-51	3-82
	PAS2	3-325/326	3-374	SWITCH	PAS1	3-207	3-267
PROC	PAS1	3-190	3-253		PAS2	3-207	3-366
	PAS2	3-309	3-355				
PTCH	ENCODER	3-44	3-77	TBOUT	ENCODER	3-63	3-90
PUT	S4B	3-106	3-113	TENC	S4B	3-98	3-108
				TEX	PREA	3-129	3-146
RCRD	ENCODER	3-53	3-83	TEXT	PAS1	3-176	3-247
RDPI	PAS1		3-278		PAS2	3-305	3-349
	PAS2	3-336	3-381	TEXTR	PAS1	3-202	3-263
RDPT	ENCODER	3-71	3-92		PAS2	3-202	3-364
RDTP	PREA	3-139	3-150	THOME	PAS2	3-333	3-378
READ	MSCONTRL	3-11	3-35	TRACOR	ENCODER	3-39	3-74
RELTST	PAS1	3-225	3-295	TRAIL	ENCODER	3-58	3-88

Subroutine	Contained In	Page Des	Page Chart
TRAIL	PREA	3-58	3-154
TRANS	ENCODER	3-49	3-79
TYCC	PAS2	3-331	3-378
TYPE	PAS2	3-330	3-377
TYPWRT	PAS1		3-272
	PAS2	3-329	3-377

Subroutine	Contained In	Page Des	Page Chart
VAL	PREA	3-124	3-144
WEOFL	FNSH	3-343	3-402
	PAS1		3-276
WMTB	MSCONTRL	3-20	3-36
WMTBU	MSCONTRL	3-19	3-35
WRITE	MSCONTRL	3-9	3-34

APPENDIX B

HOW TO MAKE A 900 META-SYMBOL SYSTEM

This appendix describes the aspects of the system that the user needs to know to generate a working META-SYMBOL system, and in particular, emphasizes the pitfalls the user must avoid if he wishes to modify META-SYMBOL successfully. For deeper understanding, the reader should consult a set of META-SYMBOL listings and a system map of the MONARCH tape. Although the 910 and 920 systems do not operate interchangeably, the listings are identical; the difference lies in the use of POPS and method of creating system overlap (910 = 910/925; 920 = 920/930 throughout the discussion). The discussion which follows describes the generation of both 910 and 920 systems. (Note: Although the 9300 META-SYMBOL operates *like* 900 META-SYMBOL, its method of generation is so radically different as to merit only this cursory note.)

The present META-SYMBOL Assembler has eight overlays. Because of space considerations, only "common" I/O is resident (MSCONTRL); the I/O for LO to the printer, being used only in PAS2, FINISH, and CON2, is written in-line in these passes. (The ramifications of this may be seen in the present Unbuffered Printer Update Packages.) Of the eight overlays, the first is loaded by the MONARCH Loader, and intercommunication between the programs that make up this overlay is by external references and definitions. The last seven overlays, on the other hand, are absolute decks with no external references or definitions, since the small resident system overlay loader (TAPE LOADER) can load only the restricted absolute, unblocked format. All intercommunication between overlays is through absolute locations, assembled into the routines of each overlay as absolute EQU's. Even through "relocatable" decks are used in constructing absolute overlays, the whole system is extremely sensitive to relocation of any segment or change in size and arrangement of tables.

THE ROUTINES OF META-SYMBOL

The routines of META-SYMBOL are listed below, numbered as individual assemblies and identified by the overlay in which they are used. (The POPS are indicated only as separate assemblies, although in essence they are included in each overlay. The procedure will be explained later.)

1.	920 POPS	
2.	910 POPS	
3.	ENCODER	
4.	S4B	
5.	MON1	OVERLAY 1
6.	TAPELOADER	
7.	MSCONTRL	
8.	PREASSEMBLER PART1 (P1)	OVERLAY 2
9.	PREASSEMBLER PART2 (P2)	
10.	SHRINK	
11.	ASSEMBLER PART1 (M1)	
12.	ASSEMBLER PART2 (M2)	
13.	ASSEMBLER PART3 (M3)	OVERLAY 4
14.	ASSEMBLER PART4 (M4)	
15.	ASSEMBLER PART5 (M5)	
16.	PAS2	OVERLAY 5
17.	FINISH	OVERLAY 6
18.	CONCORDANCE PART1 (CONCRD)	OVERLAY 7
19.	CONCORDANCE PART2 (CON2)	OVERLAY 8

Assembling the Routines of META-SYMBOL

Each routine may be assembled with META910 or META920, except for the 910 POPS, which must be assembled with META 910, and the 920 PSEUDO POPS, which must be assembled with META920. Each routine is preceded by PROCedures that define 920 instructions with operation codes between octal 100 – 117. This causes any 920 instruction to POP on either 910 or 920. For example, the OP code for CAB is 100, for SKR, 107. This is true for each routine. These arbitrary POP codes are generated no matter whether the routine is assembled with META910 or META920. Of course, the 910 POPS and 920 POPS should contain no POPS themselves; I flags on any instructions in these routines, indicates they have been incorrectly assembled.

Note that although POP codes are generated for 920 instructions and I flags occur on these instructions, these codes are unique; nowhere is a POP reference/definition item generated or used. For example, for SKR exp it is as though the op code 0107 were merged with the value of exp. The machinery in the PROCS preceding each routine that generates the I flag without producing a POP reference item is worthy of the user's attention. Again, it is important to note that POPS for 920 instructions are unique, forced, and exist in META910 and META920.

How POPS are used in the META-SYMBOL Assembler

As noted, a 920 instruction not in the 910 subset will POP on both 910 and 920 systems through a unique POP transfer location in 100 – 117 that is identical for each routine and for both 910 and 920 systems. Let us trace the execution of an ADM instruction first in the 920 system and then in the 910 system. If we looked at the ADM instruction in memory at location L, it would be 0112 in both systems. On the 920, POP code 112 causes a transfer to location 0112, which contains a BRM CHANGE, where CHANGE is located in the relocatable section of the 920 PSEUDO POPS. The PSEUDO POPS then replaces the POP instruction at location L with the actual 920 instruction for ADM, retaining the index, indirect, and address characteristics, and executes the instruction. Thus, when a POP instruction is encountered on the 920, it is replaced by the actual instruction. In loops containing a POP instruction, the POP occurs only the first time and the instruction itself is executed all other times in location L of that overlay. On a 910 system, the ADM instruction at location L is a 0112. When the POP occurs, the instruction is simulated by the 910 POPS, and no modification takes place.

Note that both the 910 POPS and 920 PSEUDO POPS contain both AORGS and RORGS. The AORGS define the absolute section 100 – 117 where the POP transfers are located. The RORGS define the relocatable section of both packages, which will be located at different points in memory for different overlays.

DTAB

In ENCODER and 910 POPS there is a cell labeled DTAB DATA N. It is AORGed at 01372. This cell is extremely important, since it contains the address of the top of the longest overlay in the META-SYMBOL system, and is used for the beginning of certain tables. At present, since PAS2 is the longest, the value in DTAB would be calculated as the last location in PAS2 plus the length of the relocatable section of the POPS being used in that system. For example, if PAS2 ended at 013500, DTAB for 920 would contain 013500 + 048 (length of relocatable section of 920 PSEUDO POPS) 013548. We would probably set DTAB to 013600 to allow a little leeway, depending on the tightness of the system. On 910, DTAB = 013500 + 0260 = 013760, or 014000 for safety. DTAB may be set too high, but not too low. It must clear the top of PAS2+POPS. The DTAB value for 920 is assembled into the DTAB cell in ENCODER, the 910 value is assembled in the DTAB cell in the 910 POPS. (Note: The 920 POPS contains no DTAB.) As description of the system continues, the determination of DTAB value will also be more clearly seen (see also Figure B-1).

OVERLAY 1

The routines in OVERLAY 1 in the order of loading by the MONARCH loader are as follows ($\Delta 1$ and $\Delta 2$ records are indicated also):

$\Delta 1$ METASYM

$\Delta 2$ ENCODER

- ENCODER (BIN)
- 910 POPS OR 920 PSEUDO POPS (BIN)

$\Delta 2$ MON1

- S4B (BIN)
- MON1 (BIN)

$\Delta 2$ MSCONTRL

- TAPELOADER (BIN)
- MSCONTRL (BIN)

ENCODER is ORGed at 01372. Although it is a relocatable program, it is loaded at 0 and its ORG effectively absolutely positions it at 01372. Note that its references to MSCONTRL and TAPELOADER are absolute through EQU's. These must be changed in all overlays if change is necessary. The last definition in ENCODER is ZTABLES EQU $+01640. This value of ZTABLE can be changed only with discretion. ENCODER is the routine that reads in

Symbolic/Encoded cards, builds a dictionary in core, merges corrections where necessary, and outputs an encoded bit string to tape X1. ENCODER contains the 920 value for DTAB.

910/920 POPS

The 910 POPS or 920 PSEUDO POPS are loaded so that the transfer vector has an AORG 0100 and the relocatable section is located above ENCODER. These function thus for the first overlay only and are repositioned for succeeding overlays. If the 910 POPS are loaded, a new 910 value for DTAB (AORG 1372) overlays the 920 value loaded in ENCODER. If 920 POPS are loaded, the initial 920 value in DTAB is unchanged.

S4B

S4B (RORG 0) is relocated above the POPS. If the C option is called, it translates from old Symbol 4 code to Modern META-SYMBOL code; it translates such items as VFD to FORM, etc. The actual translation is done during encoding and the ENCODED or Source Output (including LO) contains the translation into META-SYMBOL language.

MON1

MON1 is a relocatable routine with RORG 0, loaded just above S4B. It is the I/O initialization section of META-SYMBOL. By querying the MONARCH Unit Assignment Table and MSFNC (0273 in MSCONTRL, the cell that the MONARCH Action routine initialized with parameters on the META control card), it initializes the unit and channel numbers in all resident I/O in MSCONTRL. After initialization, MON1 is overlaid by Encoder tables.

TAPE LOADER

The TAPE LOADER (AORG 2) is a short loader used to load overlays from the systems tape. It reads only absolute subset of the 900 Standard Binary Format, unblocked records only; it can search the system tape for $\Delta 2$ labels.

MSCONTRL

MSCONTRL (AORG 0200) contains the resident I/O information. It is responsible for all input/output except the printing to the line printer or typewriter done by PAS2 or

Concordance (CON2) when listing. If PAS2 puts listing out to magnetic tape for instance, the magnetic tape routine in MSCONTRL is used. MSCONTRL also contains the ABORT logic for typing out the META-SYMBOL ABORT message and returning to MONARCH. MSCONTRL, which is the last program of OVERLAY 1 to be loaded, contains an end transfer to ENCODE, a cell containing a BRU to TRACOR, the entry point of ENCODER. Thus ENCODER is the first program to be executed after the loading of the first overlay.

OVERLAY 2

PREASSEMBLER PART 1 is a relocatable program with an origin of octal 1403. Loading this at 0 effectively positions this overlay absolutely in the correct place. Looking down to approximately line 167 of the listing of PREASSEMBLER PART 1, we find an ORG 01540 followed by some EOM's and SKS's. If we follow the octal addressing, we note that this section effectively overlays the preceding reserve area. In addition, around line 348, just preceding the label PREASSEM, there is another ORG at PIERT + 2. This is the initialization section of PREASSEMBLER, and is overlaid later by quantities placed into the reserve section defined at the beginning of the program. Further on, at about line 416, there is another ORG at CHNG1+2 following the comment "END OF INITIALIZATION CODE". This is the actual operating portion of the PREASSEMBLER. Note that the lowest portion in memory where meaningful coding exists is octal 1540, where those EOM's and SKS's are established. The relocatable section of the POPS will be loaded between P1 and P2. The second portion of the PREASSEMBLER is RORGed at 0. It is also relocatable and is loaded after PREASSEMBLER PART 1 and the POPS. Note that PREASSEMBLER PART 2 has as its last cell the label $LLITX and the unique literal 01234567. It uses this to find the end of its own string of literals and thus begin its tables.

Procedures

Since the PROCS go on the tape just as they come in ENCODED form, it is not necessary to alter them. However, there is a machine definition card that must precede every PROC deck on the system tape. The description of this card is contained in the SYMBOL and META-SYMBOL Reference Manual, under the heading "System Procedures". The PREASSEMBLER searches the tape for the $\Delta 2$ label of the proper set of PROCS, loads it into memory, and builds all the symbol tables accordingly as it makes its first pass through the byte string on X1.

OVERLAY 3

The next program is SHRINK, AORG at octal 4000. It has external references to many labels in PREASSEMBLER PARTS 1 and 2, and overlays only a portion of part 2 (i.e., the portion from octal 4000 to the end of SHRINK). Note that the second to the last label in SHRINK is called PSMPLC EQU $+0100. This effectively allows room for the literals and gives SHRINK some working storage. The purpose of SHRINK is to purge unwanted procedures from the procedure sample table so that more table space can be allowed for the rest of the assembly.

OVERLAY 4

M1 through M5 are the portions of the first pass of the Assembler. This pass was split into portions only because it could not be assembled in 8K as a single overlay. Note that M1 is RORGed at octal 1407. Although it is a relocatable program, loading it at 0 effectively places it correctly in memory. M2 through M5 have RORGS of 0 and are located consecutively following M1. The loading of these five programs, plus the POPS, constitutes the whole of ASSEMBLER PART 1.

OVERLAY 5

PAS2 is an absolutely origined program (AORG 01407). It is put on the tape just as it comes from the assembly, with definition cards removed. The definition cards from PAS2 are used to satisfy the external references in FINISH, the next overlay.

OVERLAY 6

FINISH is an absolutely origined overlay with an AORG of octal 4700. It overlays a portion of PAS2 and makes references to routines in PAS2.

OVERLAYS 7 AND 8

CONCORDANCE PARTS 1 and 2 are both assembled absolute and they go on the system tape exactly as they come from the assembly.

This concludes one rough run over the META-SYMBOL decks. If he wishes, the user can familiarize himself with the system by going through and marking all cells that are absolute, by noting all the intercommunication that is done by absolute cells, and by mapping the origins of each overlay.

CONSTRUCTION OF THE OVERLAYS

This section describes both how the overlays are to be formed in memory, and the absolute overlay created for the META-SYMBOL tape. A following section will describe the actual System Make procedure in more detail.

Overlay 1

Overlay 1 consists of the binary decks as they come from assembly in the following order:

Δ1 META SYM

Δ2 ENCODER

(Binary deck of ENCODER)

(Binary deck of 910 or 920 POPS)

Δ2 MON1

(Binary deck of S4B)

(Binary deck of MON1)

Δ2 MSCONTRL

(Binary deck of TAPE LOADER)

(Binary deck of MSCONTRL)

These routines make up the first overlay. When a META-SYMBOL card is encountered by the MONARCH System, it goes to the META-SYMBOL action routine that searches the system tape for the Δ1 METASYM label. Ignoring Δ2's, it loads decks up to the end transfer, which is on MSCONTRL. Before this, the cell MSFNC is initialized by the action routine according to the parameters on the META-SYMBOL card. ENCODER is loaded by the MONARCH loader at 0 and its ORG of octal 1372 positions it in memory. The POPS, which have a relocatable origin of 0 and an absolute origin of 0100; S4B, which has a relocatable origin of 0; followed by MON1, which has a relocatable origin of 0, are then loaded, following ENCODER. This completes the relocatable section of the first overlay. TAPE LOADER is then loaded, starting at absolute origin of octal 2. Finally MSCONTRL is loaded, with an absolute origin of octal 200. All references and definitions are satisfied by MONARCH loader and control is transferred to the end transfer location of MSCONTRL, initiating the META-SYMBOL system. From here on, the MONARCH system is not used; the META-SYMBOL TAPE LOADER takes care of loading all overlays necessary for the execution of the META-SYMBOL assembly. Control is returned to MONARCH only on completion of the assembly and/or CONCORDANCE or in an ABORT situation. The first overlay is the only one on the system tape that contains external references and definitions. It is also the only overlay loaded by the MONARCH loader.

Overlay 2

Overlay 2 consists of PREASSEMBLER PART 1 (P1), the POPS (910 or 920), and P2. It is formed by loading PREASSEMBLER PART 1, the suitable POPS, and PREASSEMBLER PART 2 into memory with the MONARCH loader and dumping out in absolute version 100 to 117, which contain the POP transfer locations, and octal 1540 through the top PART 2 of PREASSEMBLER. Generally, in making the absolute decks, reserve locations are not to be dumped; only meaningful data is output. The reserves are often used as intercommunication between two different overlays; by dumping them in making the absolute decks we may overlay some meaningful data meant to be left in memory between overlays. Therefore, although PREASSEMBLER PART 1 is ORGed at 01403, only from 01540, the first meaningful data, is dumped.

Overlay 3

The third overlay, SHRINK, is formed by loading P1, POPS, and P2, along with the SHRINK deck, to satisfy all references and definitions, and then dumping from the beginning (octal 4015) to the end of SHRINK.

Overlay 4

The fourth overlay, ASSEMBLER is formed by loading the POPS into memory at a position where they are not overlaid by PAS2 and yet lie under the value of DTAB. After the MONARCH loader has been used to load the POPS, and M1 through M5, then the portion 0100 to 0117 is dumped absolutely and the portion from 01705, which is the first meaningful data cell of ASSEMBLER PART 1, through the top of POPS is dumped. This forms the ASSEMBLER overlay.

Overlay 5

PAS2 is formed by stripping the definition cards from the front of the binary deck as it came from the assembly and using this absolute deck as the overlay. Note that when the deck is read into CORE, it uses the POPS left there by PAS1. Remember that DTAB was calculated so that if the POPS were loaded directly beneath DTAB, PAS2 could load in without overlaying the POPS. Therefore, the binary deck for PAS2 is used as it comes from the assembly but without definition cards.

Overlay 6

FINISH overlays a portion of PAS2; it makes references to labels and subroutines in PAS2. Note that it is an absolutely origined deck. To form the FINISH overlay, the definitions from PAS2 are attached to the FINISH binary deck which is loaded into memory, and the portion from the beginning (04700) to the end of FINISH is punched out.

Overlays 7 and 8

The overlays for CONCORDANCE PARTS 1 and 2 are put on the system tape exactly as they come from the assembly.

ACTUALLY MAKING THE SYSTEM

Assume that the user is at a machine with a card punch, a set of binary decks, a MONARCH System, and a copy of Program Catalog Number 850643, Binary Dump to Paper Tape or Cards. The user then loads this into memory with the MONARCH loader at DTAB or above, but where it does not conflict with the MONARCH loader tables. It remains resident in memory during the making of all overlays. Note the entry location for this dump routine. Also note that for the punching of cards, break points 3 and 4 must be set; otherwise, a tape with bootstrap is punched.

Overlay 2 PREASSEMBLER

First, boot the MONARCH system. Using the ΔLOAD STOP commands, load PREASSEMBLER PAS1 at 0, load the 910 or 920 POPS, and load PREASSEMBLER PART 2. Note that the loader stops after the loading of each of these decks. After PART 2 has been loaded, the C register contains the transfer address and the B register contains the last location plus 1. Next, transfer to the dump program. Dump location 100 through 117 with no transfer address (i.e., set X = 0). Now dump location 1540 through the top of PART 2 with the transfer address in the X register. The deck punched out is now the absolute deck for PREASSEMBLER. This is preceded by a Δ2 PREASSEM card in the system deck. The PROC decks follow, with their Δ2 cards and the machine identification card discussed earlier.

Overlay 3 SHRINK

To form the SHRINK overlay, first set up a binary deck as follows: P1, POPS, P2 with its end card removed, and the SHRINK deck. This effectively loads SHRINK as though it were part of P2. If the end transfer were left on PART 2, the SHRINK deck could not be loaded. Next, boot MONARCH in. Using the ΔLOAD STOP function, and a bias of 0, load P1, POPS, and then the third deck, consisting of P2 plus SHRINK. Note the ending location and transfer address of SHRINK. Then, using the punch program, punch from the beginning of meaningful data in SHRINK, 04015, through the end of SHRINK, with transfer address in the X register. It is not necessary to punch out the POPS at this time as they will be left there from the PREASSEMBLER overlay. SHRINK does not overlay the POPS. This constitutes the SHRINK overlay deck for the system and is now put in the system deck with a Δ2 SHRINK card preceding it.

Overlay 4 ASSEMBLER

Assuming that the calculation for DTAB has been done, the user must now calculate a bias for the POPS, approximately 42 or 260 octal locations below DTAB, depending on which POP system he is using. If the MONARCH loader with the stop function is being used, load the POPS at this bias. When the loader stops, reset the bias in the A register to 0 and load overlays M1 through M5. Because of the ORG on M1, M1 through M5 will be located correctly in memory. After having loaded the POPS in at its bias below DTAB, and loaded M1 through M5, punch out locations 100 through 117 for the POPS transfer locations, and 1705 through the top of the POPS with an end transfer as determined from the values in the C and B registers. The user will now have an absolute deck consisting of 100 to 117 and 1705 through the top of the POPS with an end transfer. This constitutes the absolute overlay of the ASSEMBLER.

Overlay 5 PAS2

To form overlay 5, PAS2, strip the DEF cards (type 1) from the front of the binary deck, that came from the assembly, and use the remaining deck as the absolute overlay. Look at the listings, being careful to determine that the last location on PAS2 lies below the current bias for the POPS. When that check is made, the deck is ready to go on the system.

Overlay 6 FINISH

To make the FINISH overlay, put the DEF cards from PAS2 onto the front of the FINISH binary deck. Load the deck with the MONARCH loader into 0, STOP. Its absolute origin biases it correctly. After loading, determine the final location in B and the transfer location from C. Punch from the beginning (04705) through the end of FINISH, with the transfer address. Do not punch the POP locations or any of PAS2, since these will still be in CORE from the previous overlay at execution time. This absolute deck is now the overlay for FINISH.

Overlays 7 and 8 CONCORD and CON2

The CONCORDANCE PARTS 1 and 2 overlays are the binary decks from the assembly.

These constitute the overlays for META-SYMBOL for MONARCH tape. Make sure that every overlay is preceded with the proper Δ2 label card. Note that it is possible to remake a single overlay and to replace it by using the UPDATE procedure on the MONARCH tape. However, take care that such things as the values of DTAB and the linkages with the POPS for that overlay are properly taken care of. Table B1 describes the final overlay deck structure for system update.

REVIEW

A final review follows of the making of overlays to indicate the exact nature of the decks used to update the META-SYMBOL processor on the MONARCH tape.

Following the Δ1 METASYM ID card is the Δ2 ENCODER ID, the binary decks for ENCODER and 910 or 920 POPS, a 2 MON1 ID, the S4B and MON1 binary decks, a Δ2 MSCONTRL ID, and binary decks of TAPELOADER and MSCONTRL. This constitutes OVERLAY 1 and is loaded by the MONARCH loader upon encountering a ΔMETA control card. Overlay 2 is preceded by a Δ2 PREASSEM ID. The absolute overlay was formed by loading P1, the POPS, and P2, and dumping 0100 – 117 and 01540 to top of P2 with an end transfer into P2. It is a single absolute deck.

The six procedure decks follow, each one an encoded deck preceded by a Δ2 PROCXXXX ID and a machine identification card. Overlay 3 is preceded by a Δ2 SHRINK ID. The absolute deck is formed by loading P1, POPS, P2 (without end card), and SHRINK binary deck, and dumping 04015 (beginning of SHRINK) to the end of SHRINK, with transfer address. It is a single absolute deck. Overlay 4 is preceded by an Δ2 ASSEMBLER ID. The absolute deck is formed by calculating DTAB, loading the POPS at a bias below DTAB, resetting the bias to zero, loading M1 – M5, and dumping 0100 – 117 and 01705 to the top of the POPS, with end transfer into M5. It is a single absolute deck. Overlay 5 is preceded by a Δ2 PAS2 ID. It is a single absolute deck from the assembly with the definition cards removed (type 1). Overlay 6 is preceded by a Δ2 FINISH ID. The absolute overlay is formed by putting the definition cards from PAS2 on the front of the FINISH deck, loading it at 0, and dumping from the beginning (04700) to the top of FINISH, with end transfer.

It is a single absolute deck. Overlay 7 is preceded by a $\Delta 2$ CONCRD ID. It is formed by using the binary deck direct from assembly. Overlay 8 is preceded by a $\Delta 2$ CON2 ID. It is formed by using the binary deck direct from assembly.

This completes the description of ABS overlays for the creation of the META-SYMBOL tape. The only difference, then, between the 910 and the 920 tapes is the POPS that are used. The size of the POPS makes the size of the overlays differ and makes the value for DTAB differ for the two systems. Only a 910 system with a 910 POPS will run on a 910/925. Only a system containing 920 PSEUDO POPS will run on the 920/930. Although the MONARCH system tapes run interchangeably on both systems, processors do not.

During the creation of the system a careful map of loading and dumping should be kept in case the system does not function correctly. Remember that reserve locations at the beginning of the overlays are not punched out. Also, the listings must be studied carefully to determine that useful information is not neglected. For instance, remember the situation of the PREASSEMBLER where the origins are reset in the body of PART 1 of the PREASSEMBLER. Assemblies to create the binary decks may be done with either Meta 910 or 920, since any instructions not in the 910 subset are automatically forced to POP by the procedure definition at the beginning of the deck (except 910/920 POPS). If a reassembly is done, check that the POP operation codes on the listing correspond with the actual transfers in the POPS. None of the overlays uses the POP machinery of the MONARCH loader. All POP operation codes must be generated absolutely at assembly time, or the system will not function.

The complete discussion here has been oriented to creating a system using card input and output. It seems that this could be done equivalently on paper tape, with two exceptions. In the making of the SHRINK overlay, the end card was removed from P2 to orient properly the loading of the SHRINK overlay with P1, POPS, and P2, so that definitions and references could be satisfied. Also, PAS2 definitions were used on the FINISH deck. On paper tape this might be difficult, although possible.

CONCLUSION

Hopefully, by using this discussion, the information in the SYMBOL and META-SYMBOL Reference Manual and the META-SYMBOL Technical Manual (Section 5, "Operational

Information"), the user may successfully create his own META-SYMBOL system. It is suggested that the user try to recreate an existing system before trying any modifications.

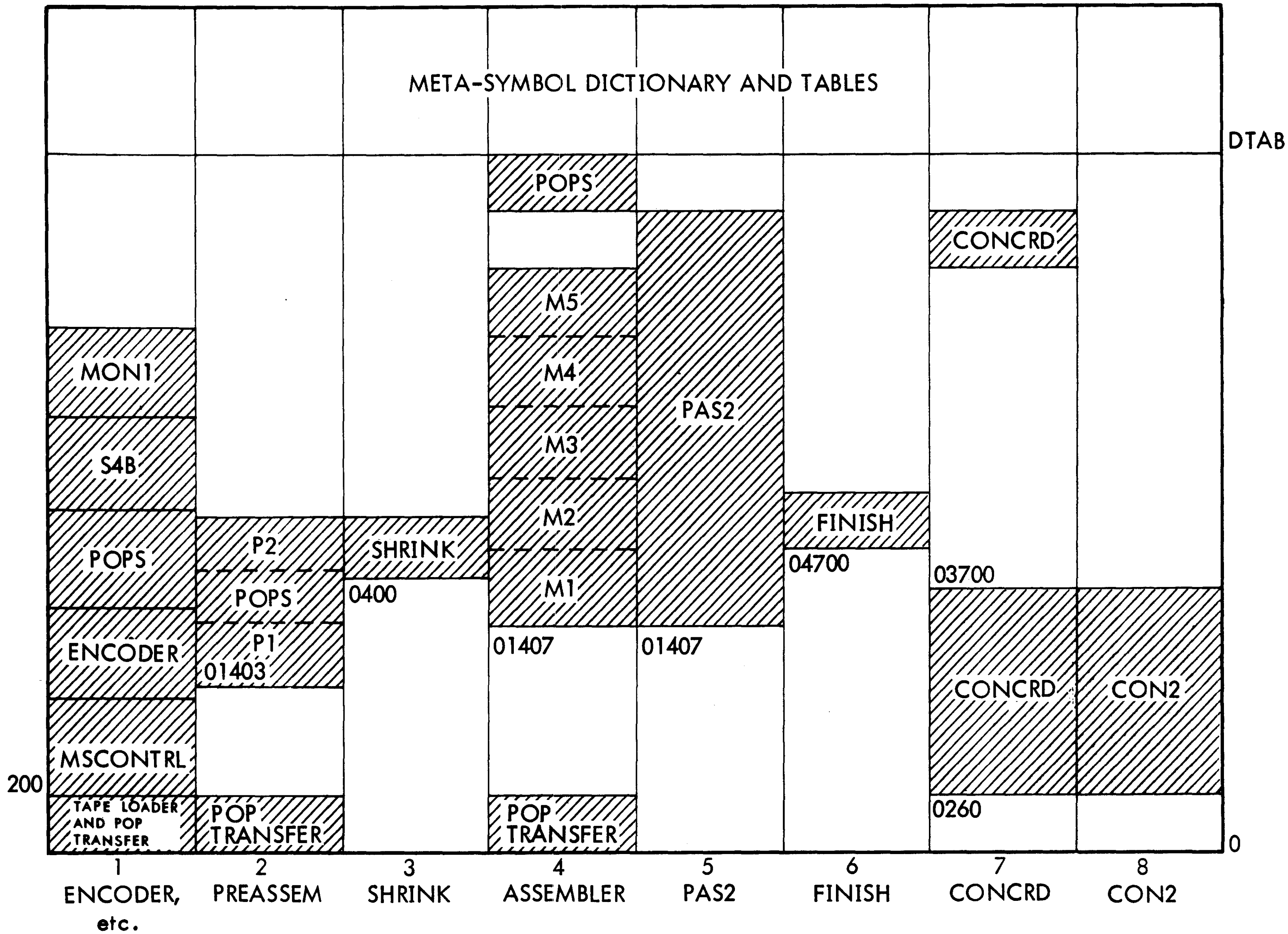


FIGURE B-1. META-SYMBOL OVERLAY STRUCTURES

Table B-1. The META-SYMBOL Update Package

Δ1	METASYM
Δ2	ENCODER
	ENCODER BINARY (as assembled)
	910 or 920 POPS BINARY (as assembled)
Δ2	MON1
	S4B BINARY (as assembled)
	MON1 BINARY (as assembled)
Δ2	MSCONTRL
	TAPE LOADER BINARY (as assembled)
	MSCONTRL BINARY (as assembled)
Δ2	PREASSEM
	PREASSEM ABSOLUTE (P1+POPS+P2) (loaded and dumped)
Δ2	PROC910
	910 PROC (as assembled + machine identification card)
	etc.
	.
	.
	.
Δ2	PROCB93H
	9300 BUSINESS PROCS
Δ2	SHRINK
	SHRINK ABSOLUTE (P1+POPS+P2+SHRINK loaded – SHRINK dumped)
Δ2	ASSEMBLER
	ASSEMBER ABSOLUTE (M1 – M5+POPS, POPS biased below DTAB) (loaded and dumped)
Δ2	PAS2
	PAS2 BINARY (as assembled, less definition cards [type 1])
Δ2	FINISH
	FINISH ABSOLUTE (FINISH+PAS2 DEFS) (loaded and FINISH dumped)

$\Delta 2$	CONCRD
	CONCORDANCE PT1 BINARY (as assembled)
$\Delta 2$	CON2
	CONCORDANCE PT2 BINARY (as assembled)

APPENDIX C

META-SYMBOL ENCODED I/O FORMAT – 900 or 9300 SERIES

An encoded program is an almost exact, but less voluminous, representation of original source code. The principle of its organization is relatively simple. The entire source program is broken down into a set of unique sequences of characters (called character strings) and a table of these unique character strings, called the dictionary, is established. The actual program is then represented to the dictionary by an ordered set of references called the text. Source code is obtained by replacing each dictionary reference with the character string to which it points.

Embedded in the text are punctuation flags which indicate such conditions as end-of-line, end-of-file, and length-of-comment. Also embedded in the text are the actual comments that appear in the source code. Comment fields are excluded from the character string definition and dictionary formation process.

Example 1. Organization of an Encoded Program

The organization of an encoded program may be illustrated by the following two lines of code:

LABEL$_{\wedge\wedge}$ LDA$_{\wedge\wedge}$ 076$_{\wedge\wedge\wedge}$ COMMENTS$_{\wedge}$ HERE

$_{\wedge\wedge\wedge\wedge\wedge}$END

These two lines may be represented by a text with the following dictionary:

Dictionary	
Reference Number	**Character String**
1	LABEL
2	∧∧
3	LDA
4	076
5	∧∧∧
6	∧∧∧∧∧
7	END

Text

1 2 3 2 4 5	END OF LINE FLAG	13 (LENGTH OF COMMENT)	$\overbrace{\text{COMMENTS}_{\wedge}\text{ HERE}}^{\text{13 characters}}$

beginning of next line

6 7	END OF LINE FLAG	0 (LENGTH OF COMMENT)	END OF FILE

The text is read by replacing its reference numbers, one at a time, with the character strings to which they correspond. Note that duplicate items ("∧∧" in our example appear only once in the dictionary.

DETAILED DESCRIPTION

Dictionary

The dictionary is a table of unique source character strings. Source code is divided into character strings in the following way:

A line of source code is moved, one character at a time, into a character string accumulator. The type (blank, special, numeric, or alphanumeric) of the first character is determined, then the type of each subsequent character is compared with that of the first before it is placed in the accumulator. If an unequal compare is made, the new character becomes the first of the next string and the string being accumulated is terminated. The treatment of alphanumeric characters is an exception. Alphabetic and numeric characters are treated as the same type during character string accumulation. However, an alpha "switch" is set whenever an alphabetic character is accumulated. When the string is terminated, this switch is tested. If it is on, the string is alphanumeric, if off, numeric. A character string is arbitrarily terminated when it contains 15 characters.

Each entry in the dictionary specifies four items of information:

1. Number of characters in the string
2. Type of character string
3. The character string itself
4. Byte number (position of entry in the table; initialized at three).

The dictionary is in a packed format. Only its first entry is guaranteed to start at a word boundary. Each entry comprises from 12 to 96 bits. The entry format is as follows:

bit 1 – 4	5 – 6	7 – 12	13 – 18	19 – 90	91 – 96
L	T	1st char.	2nd char.	…	15th char.

where:

L is the number of characters in the string $(1 \leq L \leq 15)$

T is the type of character string:

0 – blank

1 – special

2 – numeric

3 – alphanumeric

Each entry is just long enough to contain L, T, and exactly L characters. If T = 0, the character string is interpreted as a binary count of the number of spaces (internal "60's") represented by the entry, in which case a "one-character" string may represent as many as 077 spaces.

"Byte number" is not explicitly entered. It is inferred from the position of the entry in the dictionary. The byte numbers 0, 1, and 2 are reserved as punctuation. Byte number 3 is associated with the first dictionary entry, byte number 4 with the second, and so on.

Example 2. Dictionary

Suppose the first four character strings in the dictionary are LABEL, ∧∧, LDA, and 076. The beginning of the dictionary would look like the following:

Word	1	2	3	4
Appearance	27432122	25430402	17432421	16000706

These words would be interpreted in the following manner:

It is known that the first entry starts on a word boundary. So, the first six bits (octal 27) are known to specify the first character string length and type.

Octal 27 in binary is 010111; therefore, we have $L = 0101_2 = 5$, and $T = 11_2 = 3$. This means that the next five characters (10_{10} octal digits or 30_{10} bits) are to be interpreted as an alphanumeric character string. Accordingly, we interrupt 4321222543 to be the character string LABEL.

The next six bits are the length and type of the second dictionary entry. Octal 04 in binary is 000100; therefore, L = 1 and T = 0. Note that T = 0. This means that the next character (two octal digits or six bits) contains a count of the number of spaces represented by this entry. That next character is 02, and it is interpreted as the character string.

The next six bits are the length and type of the third entry. Octal 17 in binary is 001111; therefore, L = 3 and T = 3, indicating a three-character alphanumeric string. The next three characters are LDA.

For the last entry, L = 3, T = 2; therefore, the next three characters (or 18 bits) are interpreted as a three-character numeric string. We interpret 000706 to be the character string 076.

Text

Note: Statements made without explanation in this section will be better understood after study of the next section, which describes the dictionary-text generation algorithm.

The text is an ordered set of byte numbers with embedded punctuation and comments. This data is in a highly packed format.

The first item ('byte') of the text always points to the first entry in the dictionary. That is, the first byte always contains the byte number 3. The minimum number of bits required to contain the number 3 is two. Accordingly, the size of the first text byte is arbitrarily set at two bits, and the byte is said to have a byte size of two.

The text (excluding comments) is written with a monotonically increasing byte size. For any given size, the first attempt to write a byte number too large for that size will always occur when the byte number is an integral power of 2 (i.e., 2^n). Specifically, for a byte size of P bits, the first such attempt will occur on the byte number 2^P (e.g., if P = 4, the byte number will be $2^4 = 020_8 = 10000_2$). When this condition arises, a P-bit byte containing the byte number 0 is written. This is the last P-bit byte in the text. The next byte written will have P+1 bits.

Dictionary-Text Generation Algorithm

Consider the general case.

1. A character string is defined.
2. The dictionary is searched for the presence of that same character string.

3. If that string is already present, its dictionary byte number is placed in a byte of the currently used size. Suppose that the current byte size is 7 and the character string being considered already has a byte number of 5. The byte 0000101 is then added to the text. Processing then returns to step 1.

4. If that string is not already present, it is entered into the dictionary along with its length and type code. The corresponding byte number is equal to 1 plus the highest previously used byte number (it occupies the next available dictionary position). This byte number is then used to define a byte of the currently used byte size. Suppose the current byte size is 5, and the byte number is 036, then the byte 11110 is added to the text.

Consider the case in which a newly defined byte number is an integer power of 2 (i.e., 2^n). This will always be a number which is too large to be contained in a byte of the current size. Extending the illustration immediately above:

Byte Size	Octal Byte No.	Binary Byte No.	Byte
5	036	11110 (five bits)	11110
5	037	11111 (five bits)	11111
5	040(2^5)	100000 (six bits)	?

What happens is the following:

1. A byte of the current size containing the value 0 is added to the text.
2. The current byte size is incremented (BS + 1→BS).
3. When later read back, an all-zero byte is interpreted as a byte number = 2^{BS}. In our example, we defined a byte number of 040 when the current byte size was 5, so we added the byte 00000 to the text. When being read (later) the byte size is known (remember this is an ordered set of bytes) so our all-zero byte is interpreted as 2^5 = 040 and is also recognized as a signal that subsequent bytes have a length of six bits. Further, the six-bit zero byte 000000 implies the byte number 2^6 = 0100 and signals that subsequent bytes will be seven bits long. Note that if, at the time the current byte size is 010, a character string identical to that with byte number 040 (=2^5) is encountered, the byte 00100000 is added to the text.

Punctuation and Comments

Recall that the initial byte number was defined as 3. The byte numbers 0, 1, and 2 are reserved as special flags (or punctuation). The significance of the byte number 0 has already been discussed. The byte number 1 indicates end-of-line and will be more fully discussed. The byte number 2 is an end-of-file flag.

Source input is expected to be in a format which is compatible with that described in the META-SYMBOL Reference Manual. If the third (operand) field is nonblank, it will be followed by a blank string whose termination signals an end-of-line condition. If the third field is blank, the termination of its character string indicates end of line.

End of line is indicated in the text by a byte of the currently-being-written size which contains the byte number 01. If this condition arises while bytes are being written with a length of four bits, the byte 0001 is added to the text. If there is no comment present, the blank string that signaled the end-of-line condition will include the end of the source record. Such a blank string is not entered in the dictionary or referenced by the text.

When the operand field contains a list that is continued on a subsequent physical record, the line containing that operand field is extended to include the entire list. The blank field that terminates the physical record of such a to-be-continued operand field is encoded through column 80, and its character string also includes any leading blanks on the continuation record. In this situation, a single encoded line will represent more than one physical source record.

The end-of-line byte is unconditionally followed by a six-bit count of the number of comment characters on that line. This count may be zero. If the count is not zero, it is immediately followed by the actual comment characters in XDS internal format. The comment characters (or the count if it is zero) are immediately followed by the first byte of the next line of code.

Asterisk-comment (*) lines are treated like any other line, except that the end-of-line flag is added to the text just before the first nonblank comment character is processed. If there are no nonblank characters, the end-of-line flag immediately follows the byte that references the asterisk.

After the last line of source is encoded, an end-of-file flag is written. This flag is a byte of the currently-being-written size which contains the byte number 2.

Additional Comments

1. Only the first 72 characters of each source record (card) are encoded (except in the case of a continued list).

2. The first word of each encoded record is a control word in the following format:

0 2	3 8	9 11	12 23
T	L	E = 7	folded checksum

where:

T is the record type:

0 – if part of program text

1 – if part of program dictionary

3 – if last record in file

L is the number of words in the record, including the control word.

E is always seven, which signifies an encoded record.

3. An encoded file comprises a dictionary followed by its text. The dictionary is terminated with a zero length entry. The text is terminated by a byte with byte number 02.

4. An encoded program is described as an "almost" exact representation of original source code because of the restriction on comment length. The maximum count that can be represented in six bits is 077 = 63_{10}. This means that any comment character beyond the 63rd is truncated at the time the program is encoded.

5. META-SYMBOL assembles with source code in encoded form. Therefore, comment 4 above applies to any program assembled by META-SYMBOL.

6. The special character "delete-code" ( ), internal code 077, cannot be encoded properly by the META-SYMBOL routine ENCODER. According to the design of ENCODER, when an end-of-line condition is encountered, the internal code 077 is inserted into the next-character-to-be-processed save location. Its presence there subsequently cues end-of-line flagging and comment field addition to the

text. Internal 077 does not become an actual part of the source input; it is merely inserted into a temporary cell through which every character encoded passes.

When the character "delete code" is actually present in the source program, one of two things happens. If it is used in a comment field, it is processed normally since comment fields are not encoded. If it is used anywhere else, it cues end-of-line processing. That is:

a. The character string being accumulated is terminated.

b. The appropriate dictionary and text entries are made.

c. An end-of-line byte is added to the text.

d. A comment count is output, followed by the actual comment. In this situation, the balance of the source line, from the "delete code" on, is treated as a comment.

e. If an end-of-line condition arises while a blank string is being accumulated, it is assumed that the end of the source record has been reached and that the blank string represents trailing blanks on the source line. In this case, that string causes no entries into the dictionary or text. The balance of the line (including the "delete code") is, however, treated as a comment.

During PAS2, any attempt by the subroutine GET to access a character from a given line, after the end-of-line flag has been detected, results in the default accessing of the character blank (060). For an example of this situation, try using a "delete code" within the range of a TEXT directive.

Conclusion

The following example illustrates the foregoing discussion:

Example 3. Encoding of Source Program

Let us encode the following five line source program.

```
1.  AB6D^^^EQU^^^^^012345^^^^^^^^ ------------------------> ^^^
    1   5                                                   72
```

```
2.  $ENTRY^^^NOP^^^^^^^^^^^^^^^^^^^^^INSTRUCTION^LINE^COMMENT^^^-->^^^
    1   5                                                          72

3.  ^^^^LDA^^^^^AB6D^^^^^ ---------------------------------------->^^^
    1   5                                                          72

4.  *^^^ASTERISK^LINE^COMMENT^^^ ------------------------------->^^^
    1   5                                                          72

5.  ^^^END^^^ ENTRY^^^^^ ----------------------------------------->^^^
    1   5                                                          72
```

The first character string is AB6D; it is a four-character alphanumeric string. Its dictionary entry is 2321220624. The leading 23 indicates $L = 0100_2 = 4$ and $T = 11_2 = 3$. It is the first entry in the dictionary, so it can be referenced by a byte number of 03. Accordingly, the text begins with a two-bit byte equal to 3 (i.e., 11).

The second string is ^^^, a three-character blank field. Its dictionary entry is 0403. The leading 04 indicates L = 0001, T = 0. The character 03 indicates three blanks. It is the second entry in the dictionary, and can be referenced by a byte number of 4. Accordingly, a two-bit byte (we are still in a two-bit-per-byte mode) indicating the byte number 04 is added to the text. In this case, two bits cannot contain the value 04, so an all-zero byte of two bits (i.e., 00) is added. The writing of an all-zero byte means that the next byte will be one bit longer.

At this point, there are two entries each in the dictionary and text. They are:

Dictionary – 23212206240403

Text (a bit string) – 1100

The third character string is EQU, a three-character alphanumeric string. Its dictionary entry is 17255064. The leading 17 indicates $L = 0011_2 = 3$, $T = 11_2 = 3$. This is the third entry in the dictionary. It is referenced by the byte number 05. The last byte written in the text was a two-bit all-zero byte. Therefore, bytes are now to be written in a three-bit-per-byte mode. Accordingly, the byte 101 is added to the text.

The fourth string is another set of blanks, ^^^^^. Its dictionary string is 0405. The leading 04 indicates L = 1, T = 0. Since T = 0, the 05 must specify a five-character sequence of blanks. The terminology in this situation may be confusing. The actual "character-string" is the single character with the internal representation 05. Since this string is specified as type 0, the character 05 is understood to represent the set of characters ^^^^^. This dictionary entry is associated with the byte number 06. Accordingly, the three-bit byte 110 is added to the text. The text now contains the bits 1100101110.

The next string is the six-character set 012345. A six-character numeric string implies L = 06 = 0110_2, T = 02 = 10_2. Therefore, the dictionary entry is 32000102030405. This is the fifth dictionary entry, so it is referenced by byte number 07. The three-bit byte 111 is added to the text. The text now contains the bits 1100101110111.

The last string processed was the operand field; therefore, end-of-line processing begins. The blank string which immediately follows the operand field includes the end of the source record, so it is not included in the dictionary or text. The next byte added to the text is an end-of-line flag. Since we are currently writing bytes of three bits each, the end-of-line flag is the byte 001. This byte is immediately followed by a six-bit character count specifying the length of the comment on the source line. There is no comment on this line, so the character count 000000 is added to the text.

At this point, the dictionary and text are as follows:

```
(Byte Number)    (3)        (4)      (5)     (6)          (7)
Dictionary - 23212206240403172550640405320000102030405
Text (bit string) - 11 001 01 110 111 001 000000

                    3  4  5   6   7  EOL comment length
```

Encoding of the second source line now begins. The first string is the single special character "$". Its dictionary entry is 0553 (05 implies L = 1, T = 1). Its byte number is 010, so the appropriate three-bit byte is added to the text. The new text byte is 000, indicating a byte number equal to 2^3 and further indicating that the byte size is to be increased by one bit.

The next string is the five-character alphanumeric (type 3) string ENTRY. Its dictionary entry is 272545635170 and its byte number is 011. The four-bit byte 1001 is added to the text.

The next string is a three-character blank field. Such a string is already in the dictionary (the second entry), so no additional entry is made. A four-bit byte (the current size) containing the byte number of that dictionary entry, 04, is therefore added to the text. That byte is 0100.

The three-character alphanumeric string NOP is then entered into the dictionary as 17454647 (L = 3, T = 3). Its byte number, 012, is then added to the text in a four-bit byte, 1010.

There is no operand field present, so end-of-line processing begins. The blank string which immediately follows the last string (the operation field) is terminated by a comment. This blank string is 027 characters long and is entered into the dictionary as 0427 (L = 1, T = 0). The byte number of the blank string, 013, is added to the text as 1011. Then the four-bit end-of-line byte, 0001, is added to the text. Next, the six-bit comment character count is added. INSTRUCTION^ LINE^ COMMENT. has a length of 031, so the count 011001 is used. Immediately following the count in the text is the actual 031 character (0226 bit) comment in XDS internal format, unfortunately not constrained to respect standard character and word boundaries (i.e., a character may begin on any of the three bits of an octal digit).

The third line has four character strings:

Source	Dictionary Entry	Byte Number	Text Addition
^^^^	0404	014	1100
LDA	17432421	015	1101
^^^^^	previously defined	06	0110
AB6D	previously defined	03	0011

There are no comments, so the end-of-line byte 0001 and the character count 000000 are added to the text, concluding the third line.

The fourth line is an asterisk-comment line. These lines are encoded normally, except that end-of-line processing begins with the first nonblank character (if any) after the leading asterisk.

This line has two character strings:

Source	Dictionary Entry	Byte Number	Text Addition
*	0554	016	1110
^^^	previously defined	04	0100

Then comes the end-of-line byte, 0001, followed by the comment length 010110 (026) and the 0204 bit comment.

The last line of code has four character strings:

Source	Dictionary Entry	Byte Number	Text Addition
^^^	previously defined	04	0100
END	17254524	017	1111
^^^	previously defined	04	0100
ENTRY	previously defined	011	1001

These bytes are followed by the end-of-line byte 0001 and the comment length 000000.

The last record of a source input file is followed by a ΔEOF record. This record cues the generation of an EOF byte. This byte is of the currently being written size (four bits) and contains the byte number 02. Our file is ended with the byte 0010. Our dictionary is terminated with the entry 00 (L = 0, T = 0).

A detailed dictionary and text for the program just encoded is given below.

Dictionary

(byte number)

Actual Entry –

$$\underbrace{\overbrace{2321220624}^{(03)}}_{(\mathrm{AB6D})}\underbrace{\overbrace{0403}^{(04)}}_{(03^{\dagger})}\underbrace{\overbrace{17255064}^{(05)}}_{(\mathrm{EQU})}\underbrace{\overbrace{0405}^{(06)}}_{(05^{\dagger})}\underbrace{\overbrace{32000102030405}^{(07)}}_{(012345)}\underbrace{\overbrace{0553}^{(010)}}_{(\$)}$$

(character string)

$$\underbrace{\overbrace{272545635170}^{(011)}}_{(\mathrm{ENTRY})}\underbrace{\overbrace{17454647}^{(012)}}_{(\mathrm{NOP})}\underbrace{\overbrace{0427}^{(013)}}_{(027^{\dagger})}\underbrace{\overbrace{0404}^{(014)}}_{(04^{\dagger})}\underbrace{\overbrace{17432421}^{(015)}}_{(\mathrm{LDA})}\underbrace{\overbrace{0554}^{(016)}}_{(*)}\underbrace{\overbrace{17254524}^{(017)}}_{(\mathrm{END})}\overbrace{00}$$

Text – See next page.

† Number of characters in "blank string".

Text – for Example 3, Encoding of Source Program

```
                                         0 char.                                    031 char.
Interpretation    ——→ 03 04 05 06 07 EOL comment 010 011 04 012 013 EOL  comment    I      N
Actual Bit String ——→ 110010111101110010000000000100101001010101110001011001011001100101
Octal Appearance  ——→ 6  2  7  3  4  4  0  0  2  2  4  5  2  6  1  3  1  3  1  4  5

   S      T      R      U      C      T      I      O      N      ∧      L      I      N      E
110010110011101001110100010011110011011001100110100101110000100011011001100101010101
 6  2  6  3  5  1  6  4  2  3  6  3  3  1  4  6  4  5  6  0  4  3  3  1  4  5  2  5

                                                                                    0 char.
          C      O      M  .   M      E      N      T            014  015  06  03  EOL  comment  016
110000010011100110100100100100010101100101110011011011110011010110001100010000001110
 6  0  2  3  4  6  4  4  4  4  2  5  4  5  6  3  3  3  6  3  2  6  1  4  2  0  1  6

          026 char.
04  EOL   comment    A      S      T      E      R      I      S      K             L      I      N
010000010101100100011100101100110101011010010110011100101000101100001000110110011001010
 2  0  2  5  4  4  3  4  5  4  6  5  3  2  2  6  3  4  5  0  5  4  1  0  6  6  3  1  2

                                                                                        0 char.
   E      ∧      C      O      M      M      E      N      T           04  017  04  011  EOL  comment  EOF
101011100000100111001101001001001000101011001011100110110110100111101001001000100000000100
 5  3  4  0  4  7  1  5  1  1  1  0  5  3  1  3  4  6  6  6  4  7  5  1  1  0  4  0  0  4
```

Xerox Corporation
701 South Aviation Boulevard
El Segundo, California 90245

XEROX

Reader Comment Form

We would appreciate your comments and suggestions for improving this publication.

Publication No.	Rev. Letter	Title	Current Date

How did you use this publication?	Is the material presented effectively?
☐ Learning ☐ Installing ☐ Sales ☐ Reference ☐ Maintaining ☐ Operating	☐ Fully Covered ☐ Well Illustrated ☐ Well Organized ☐ Clear
What is your overall rating of this publication?	**What is your occupation?**
☐ Very Good ☐ Fair ☐ Very Poor ☐ Good ☐ Poor	

Your other comments may be entered here. Please be specific and give page, column, and line number references where applicable. To report errors, Please use the Xerox Software Improvement or Difficulty Report (1188) instead of this form.

Your Name & Return Address

2190(12/72)

Thank You For Your Interest. (fold & fasten as shown on back, no postage needed if mailed in U.S.A.)

Staple

Staple

Fold

First Class
Permit No. 229
El Segundo,
California

BUSINESS REPLY MAIL

No postage stamp necessary if mailed in the United States

Postage will be paid by

Xerox Corporation
701 South Aviation Boulevard
El Segundo, California 90245

Attn: Programming Publications

Fold