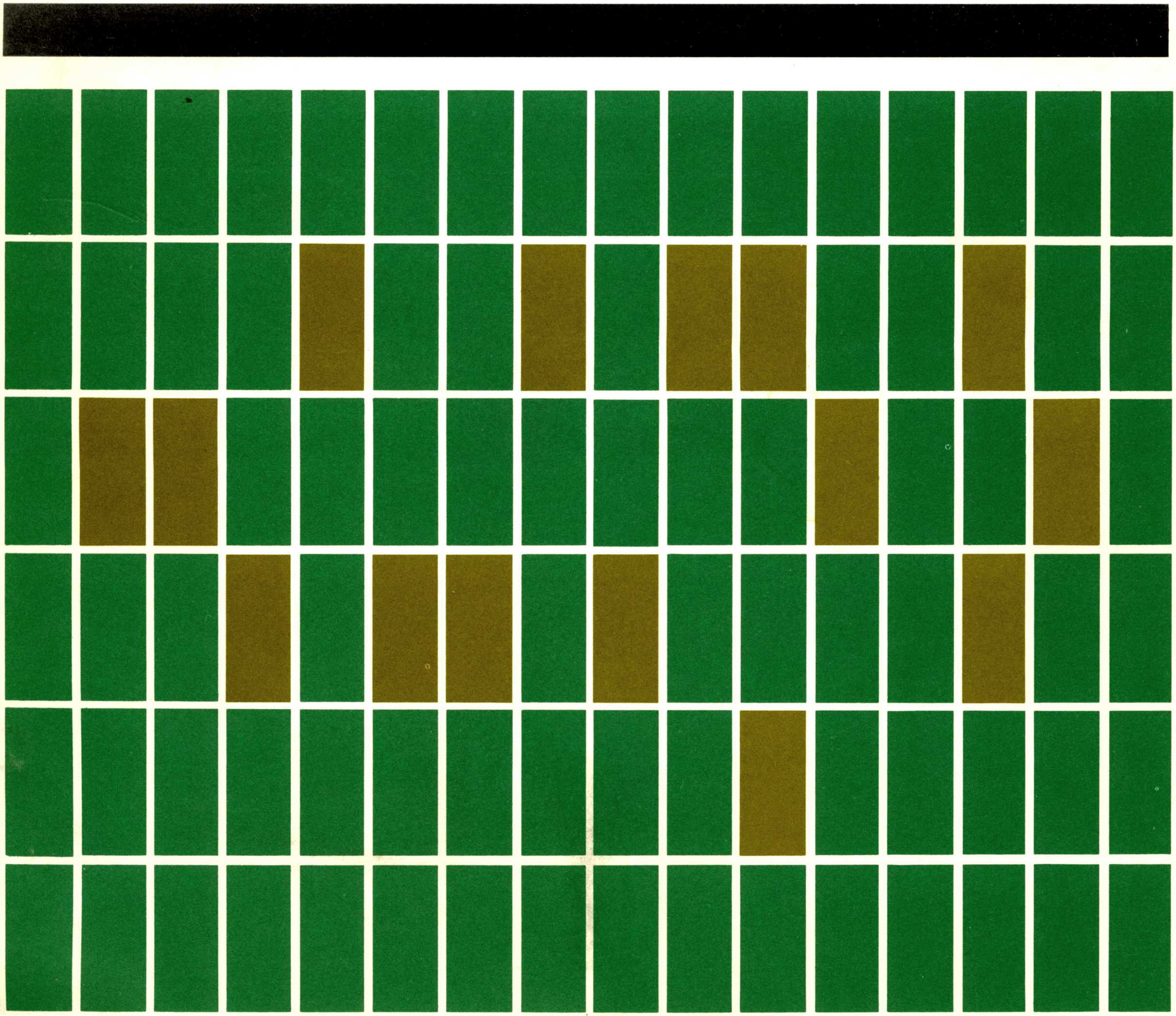
COMPUTER SCIENCES CORPORATION



CONTROL DATA CORPORATION
64/6600 COBOL
VERSION 1.0
INTERNAL REFERENCE SPECIFICATIONS

COMPUTER SCIENCES CORPORATION

At Los Angeles International Airport
650 North Sepulveda Boulevard
El Segundo, California 90245

Los Angeles/San Francisco/Richland, Washington/Houston/Washington/New York/London

NOVEMBER 1967

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd)

TABLE OF CONTENTS (Cont'd)

TABLE OF CONTENTS (Cont'd)

LIST OF FIGURES

LIST OF FIGURES (Cont'd)

LIST OF FIGURES (Cont'd)

LIST OF FIGURES (Cont'd)

LIST OF FIGURES (Cont'd)

LIST OF TABLES

SECTION 1

SECTION 1 - INTRODUCTION

ABSTRACT

This manual is a design document that describes the internal design of the COBOL compiler for the CDC 64/6600. It has several major sections that describe the design from different points of view:

Section 2 describes the form of the processor itself and the methods of producing, changing and operating with it within the SCOPE operating system and describes techniques used broadly throughout the compiler, describing in some cases, the way the source information is processed by different parts of the compiler.

Section 3 describes the major parts of the compiler, giving all the processing in roughly the order in which it occurs.

Section 4 contains the format descriptions of internal tables in one place, showing the ways in which information is encoded.

Section 5 describes the various outputs of the compiler.

Section 6 describes routines that are used by object programs during execution time (by calls from the subprogram library).

REFERENCES

This document presupposes that the reader is familiar with the 64/6600 computers, the ASCENT assembly language for them, the SCOPE 2.0 and 3.0 operating systems, and the COBOL language as described in the External Reference Specifications for this compiler (12/9/66).

DESIGN OBJECTIVES

PRIME OBJECTIVES

1. Early delivery
2. Modularity
3. Reliability
4. Ease of maintenance
5. Object code efficiency

The structure, internal design, and scheduling of implementation stresses these objectives in the order listed.

?

The large memory capacity available in the 64/6600 computer makes new techniques feasible that will result in processing speeds in excess of 6000 statements per minute for normal COBOL source code on the 6600 (assuming source input to the compiler, compiler printing, and relocatable instruction output are not I/O limited).

External Design Objectives

The language to be implemented is covered in the External Reference Specifications (12/9/66). The system provides the data processing user with a complete system for his needs, including report writing, linkage to 64/6600 SORT/MERGE, and the facility to link COBOL object code with relocatable subroutines.

Hardware Configuration

The COBOL system operates on the minimum hardware configuration required by 64/6600 SCOPE Version 3.0. (25K of words of core storage will be available to the compiler, plus disk ~~and tape storage~~.) Additional core and peripheral equipment may improve compiler capacity.

Implementation Language and Operating Systems

The compiler is written in the COMPASS language. It produces relocatable binary output for loading by the 64/6600 SCOPE system.

64/6600 COBOL operates under 64/6600 SCOPE, Version 3.2.

Object time input/output for COBOL programs is provided by the 64/6600 SCOPE system. The COBOL compiler generates appropriate linkage to utilize SCOPE.

64/6600 COBOL generates appropriate linkage to 64/6600 SORT/MERGE, providing the SORT facility within the COBOL system.

Operator Communication

The COBOL compiler requires no communication with the computer operator. Any equipment or files that are not available when the compiler needs them will cause the termination of the compilation run.

Tape assignments, tape-label handling, tape changing, file searching, and similar functions requiring communication with the computer operator are performed by the SORT/MERGE and SCOPE systems.

COBOL statements ACCEPT, DISPLAY, and STOP literal communicate with the operator by means of the standard 64/6600 SCOPE operating system's communication facilities.

Any unexpected arithmetic errors result in error termination of the COBOL object program. The reason for the termination is printed in the user's control listing.

Programmer Communications (Diagnostics)

Information supplied to the compiler by the programmer, i.e., such as compiler options and location of library data, is provided to the COBOL compiler from standard 64/6600 SCOPE control cards, by means of the 64/6600 SCOPE operating system.

Diagnostic information about the source program is available at four levels, which may optionally be printed on the user's listing. These four levels are as follows:

1. Non-DOD messages
2. Precautionary diagnostics
3. Errors
4. Fatal errors

Normally, non-DOD and precautionary diagnostics are not printed in the user's listing format.

General Performance

The primary objective is to provide a COBOL system in which heavy emphasis has been placed on modularity, reliability, and ease of maintenance. Object code efficiency consistent with the speed and power of the 64/6600 has been achieved consistent with a sound compiler design. No special optimization pass was employed in an effort to approach "hand coded" efficiency.

COMPILER STRUCTURE

COMPILER LAYOUT

The following pages describe the overlay format of the compiler. The compiler is constructed in overlays so that maximum control can be utilized to minimize the space needed for the compiler routines. (See Figure 1-1.) SCOPE rules limit overlays to two levels, primary and secondary overlays.

The compiler is placed in absolute form on the system library.

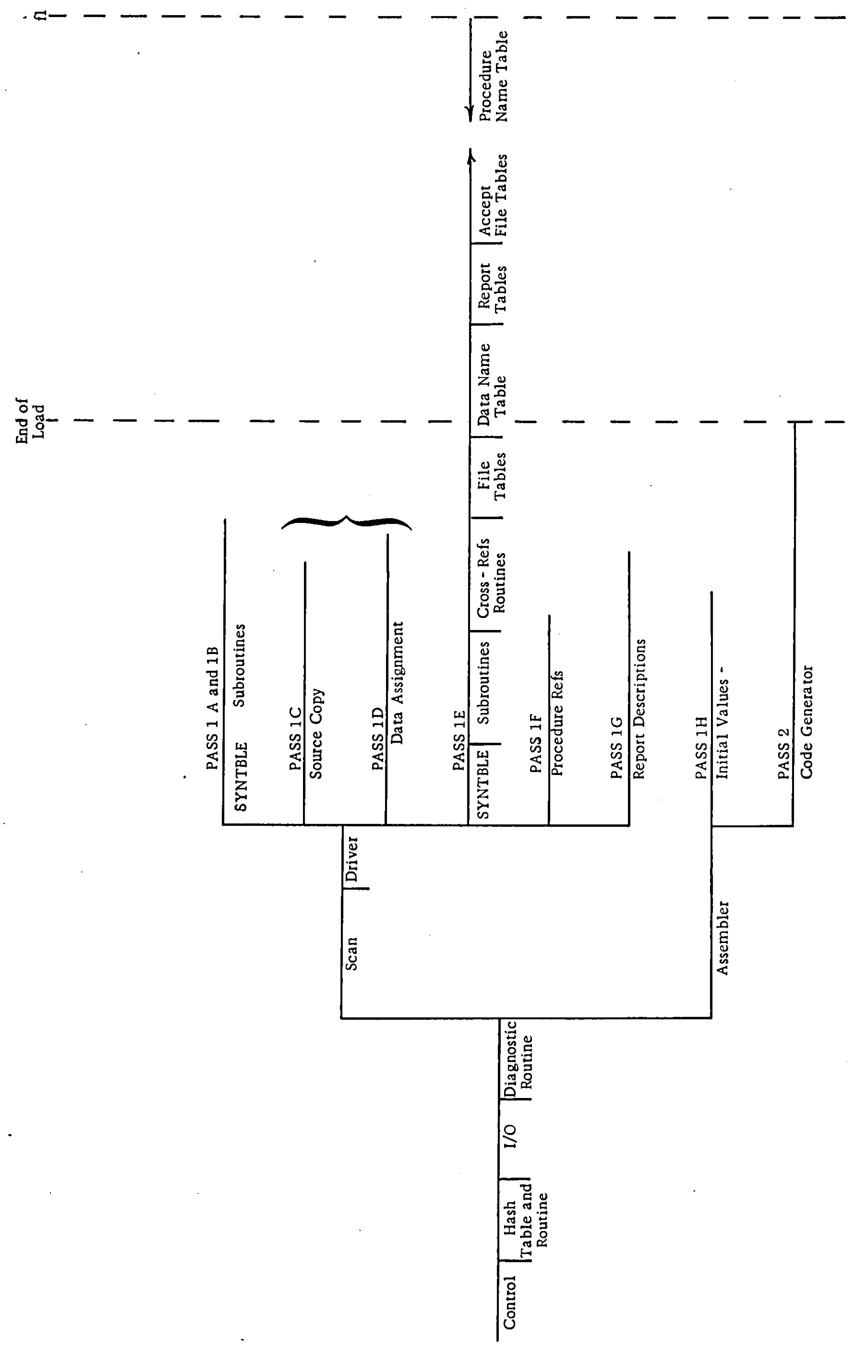


Figure 1-1. Compiler Overlay Scheme

The compiler is designed so that a number of "open-ended" tables are located in one big area at the end of available memory. Since the SCOPE loader allows the compiler access to any area within the field length (fl) specified on the JOB card, the open-ended tables are located above the program load area and below the field length. Obviously, certain amounts of surplus core must be furnished by the user before any significant table work can be done.

SECTION 2

SECTION 2 - PROCESSING TECHNIQUES

TWO-PASS APPROACH

The compiler makes two passes over the Procedure Division code. The first pass consists of the preliminary processing of statements and procedures to establish their references to the Data Division and to each other--or to other compilations. Output from the first pass is a symbol table for procedure names and references, and encoded syntax items in a format which is often referred to as "Trees." The first pass places the syntax items on the disk.

The second pass generates object code and creates relocatable binary elements, placing them on a file ready for subsequent loading and execution by the user.

SYNTAX ANALYSIS

GENERAL

The COBOL compiler consists of several parts. Those parts which are concerned with the scan of the COBOL source code are written in a specially designed source language, the syntax language; this language is processed into a pseudo-machine code in the 6600. An interpretative routine, the Syntax Analysis Driver (SAD) executes this pseudo-code during a COBOL compilation. A part of the compiler written in this language can be called a "subprocessor." Each subprocessor (there are two) undergoes a preprocessing to convert from the syntax language to an octal format acceptable to the 6600 assembler. This preprocessing is done by a separate routine written in the COBOL language. This octal format as output from the preprocessor, is put into loadable form by the assembler. Figure 2-1 illustrates this process and the operation of this subprocessor during a COBOL compilation.

The preprocessor is a routine written in the COBOL language whose input is a SUBPROCESSOR written in syntax language. It converts this to assembly language. This SUBPROCESSOR is then assembled into binary psuedo code for the 6600. The routine SAD and all the necessary routines it uses, including SCAN, DIAG, and certain special punctuation processors, are also assembled.

During operation of the compiler, the source language is processed by the SUBPROCESSOR into an encoded form for use by other parts of the compiler. The SUBPROCESSOR exists as pseudo code that is interpreted by SAD. This technique permits the introduction of a large number of operations into the syntax language, each defined by a subroutine that executes it.

SYNTAX TABLE LANGUAGE STRUCTURE

Each separate syntax language SUBPROCESSOR (there are two in CDC COBOL) is called a syntax table. Each syntax table consists of named sections of variable length. Each section consists of fixed length entries that are numbered within the section. Each such entry

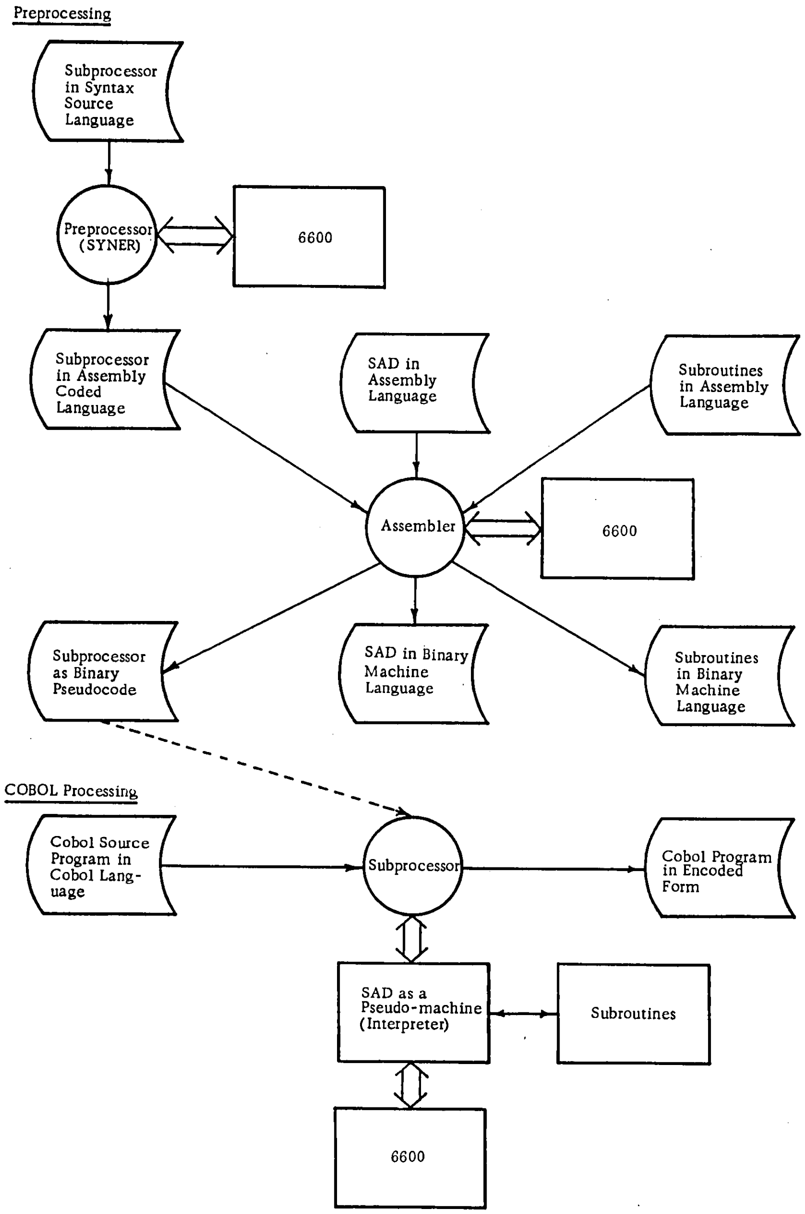
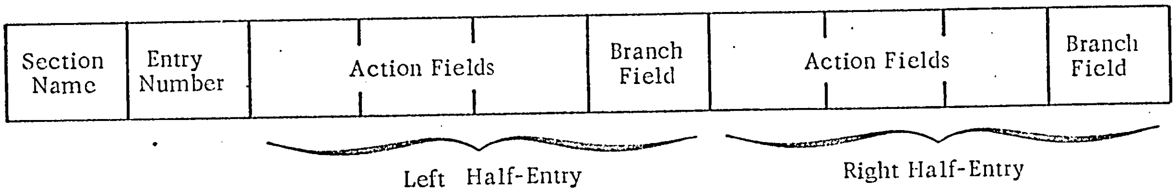


Figure 2-1. Processing Associated With 6600 COBOL Subprocessor in Syntax Language

is written as one line in the original source syntax language. This line can then be further subdivided into two half-entries, each of which has four fields. Three of these fields are of one type and can be called "action" fields. The fourth field in each half-entry (or half-line) is a different type field called a "branch" field. The name of the section and the number of the entry are also written on the line. A line is written in the following form:



Pseudo-Computer Operation (SAD)

The pseudo-computer can be thought of as having eight instructions per entry ("word"). Different type instructions, however, are actually generated for the branch fields and the action fields. The form of these instructions is shown in Table 2-1.

Table 2-1. Pseudo Instructions - Syntax Language

	Source Form		Encoded Form		
	Field	Explanation	Op	Address	Operation Performed
Action Fields	Subroutine name		0	Subroutine number	Execute the subroutine.
	Dnnn	nnn is a diagnostic number.	2	Diagnostic number	Issue the diagnostic.
	$aaa	aaa is a reserved word (in lexicon).	4	Lexicon entry number	Scan next word for this entry.
	Section name		6	Section location	Execute the section.
Branch Fields	YES		7	1	Return from this section skipping to next half-entry after calling instruction.
	NO		7	0	Return from this section to next instruction after calling instruction.
	nn	Entry number must be in this section.	5	Entry number (in table)	Go to first instruction in numbered entry.
			7	2	Return from this instruction to second half instruction after the calling instruction.

The process of "executing" a subroutine or section of the table (pseudo code) involves a transfer of control remembering the calling location. A return is made either to the next instruction (PARAM = 0) or to the first instruction of the nth half word after that (PARAM = n). The NO return from an execution section (of pseudo code) corresponds to PARAM = 0 while the YES return corresponds to PARAM = 1.

Use of the Syntax Language

The syntax language has the ability to be recursive (a section can be executed from within itself, directly or indirectly). It was primarily designed for one particular line format that is the one most commonly used in the CDC COBOL syntax tables. In this form, the first action field is called the "look for" field. It can be a $ field (look for a reserved word), a section-name field (look for some specific type of source word or words), or a subroutine name (look for a particular type word). The remainder of the left half-entry contains "NO-action" fields; the right half-entry contains "YES-action" fields. The "look-for" field is expected to produce a skip to the right half-entry if the thing sought is found; otherwise, sequential return is made to do the NO-actions. The NO-action and YES-action fields do not execute skips, allowing control to pass to the following branch field which transfers control. These branch fields are sometimes called the NO-GO-TO and YES-GO-TO fields, respectively.

DETAILED OPERATION OF SAD

The flowchart in Figure 2-2 describes the Syntax Analysis Driver.

Subroutines Used by SAD

There are several external subroutines called by SAD that deal directly with SCAN's working storage. They are listed below with brief explanations of each:

1. IMPKEYW--Calls SCAN2 then checks bits 8 and 7 of the lexicon number of the current character string for 11, which identifies the word as an imperative key word. If true, the NOSCNFL is set, and a return to SADYES is made; otherwise, return is to SADNOSN.

2. SIMPKEY--Calls SCAN2 then checks bits 8, 7, 6 of the lexicon number for a setting of 111, which indicates a key word that sometimes is imperative and sometimes is conditional. If true NOSCNFL is set and return is made to SADYES; otherwise, return is to SADNOSN.

3. KEYWORD--Calls SCAN2, then checks bit 9 of the lexicon number obtained from SCAN2 for a setting of 1, which indicates a key word that can introduce a new statement. If true, return is to SADYES; otherwise, return goes to SADNOSN.

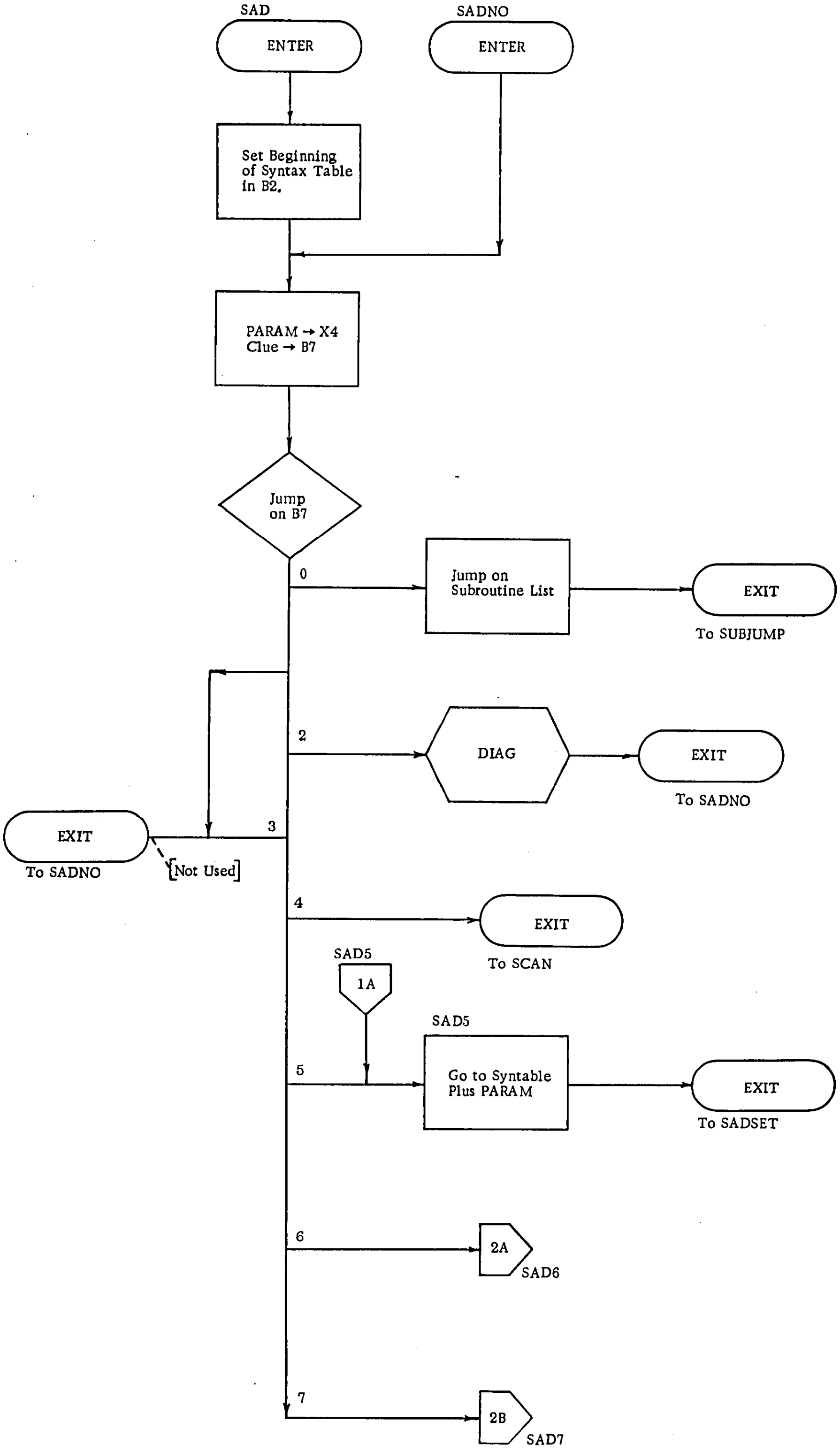


Figure 2-2. Syntax Analysis Driver (SAD) Flowchart (1 of 2)

4. ACCT--Turns on the conditional comma test flag CONCOMA for subsequent testing, then goes to SADNO.

5. DCCT--Clears the CONCOMA flag to zero to discontinue conditional comma testing, then goes to SADNO.

6. CCT--Conditional Comma Test. Checks the CONCOMA flag. If ON, transfer is made to the COMMA subroutine; otherwise, the NO return (JP SADNO) is made to SAD.

7. COMMA--Clears COMAFLG to allow a comma as punctuation before the previous words then checks the XTENDMD flag. If OFF, the SEMIFLG is cleared to allow comma and semicolon to be used interchangeably before the previous word. Otherwise, the SADNO return is made.

8. SEMICOL--First checks SCNOTNW flag to see if semicolon is legal before the next word. If ON, return to SADNO is made; otherwise, the SEMIFLG is cleared and XTENDMD is checked for ON. If ON, return to SADNO is made; otherwise, the COMAFLG is cleared to allow interchangeable use of comma and semicolon before the previous word. Then return is to SADNO.

9. A--Checks the necessity and/or legality of column 8 beginnings. If COL8FLG is ON, it is cleared. If zero, a diagnostic is issued and return is made to SADNO.

10. SNC--Set to Next Card. Sets SKIPOPS to 2, which causes SCAN2 upon subsequent entry to skip to the next source card beginning.

11. SBW--Set Back of Word. Sets the NOSCNFL to enable a reexamination of the previous word returned by SCAN.

12. SCNANW--Semicolon not allowed. Next word sets the SCNOTNW to disallow semicolon use before the next word.

13. NONNLIT--Checks CWIC for 2, which describes a non-numeric literal just returned by SCAN. If true, return is to SADYES; otherwise, return is made to SADNOSN.

14. NUMBER--Calls SCAN2 then checks CWIC = 3. If true, the last character string from SCAN2 was a numeric literal and return is made to SADYES; otherwise, returns to SADNOSN.

15. NAME--Calls SCAN2 then checks CWIC = 0 which describes a name of some type. If true, goes to SADYES; otherwise, returns to SADNOSN.

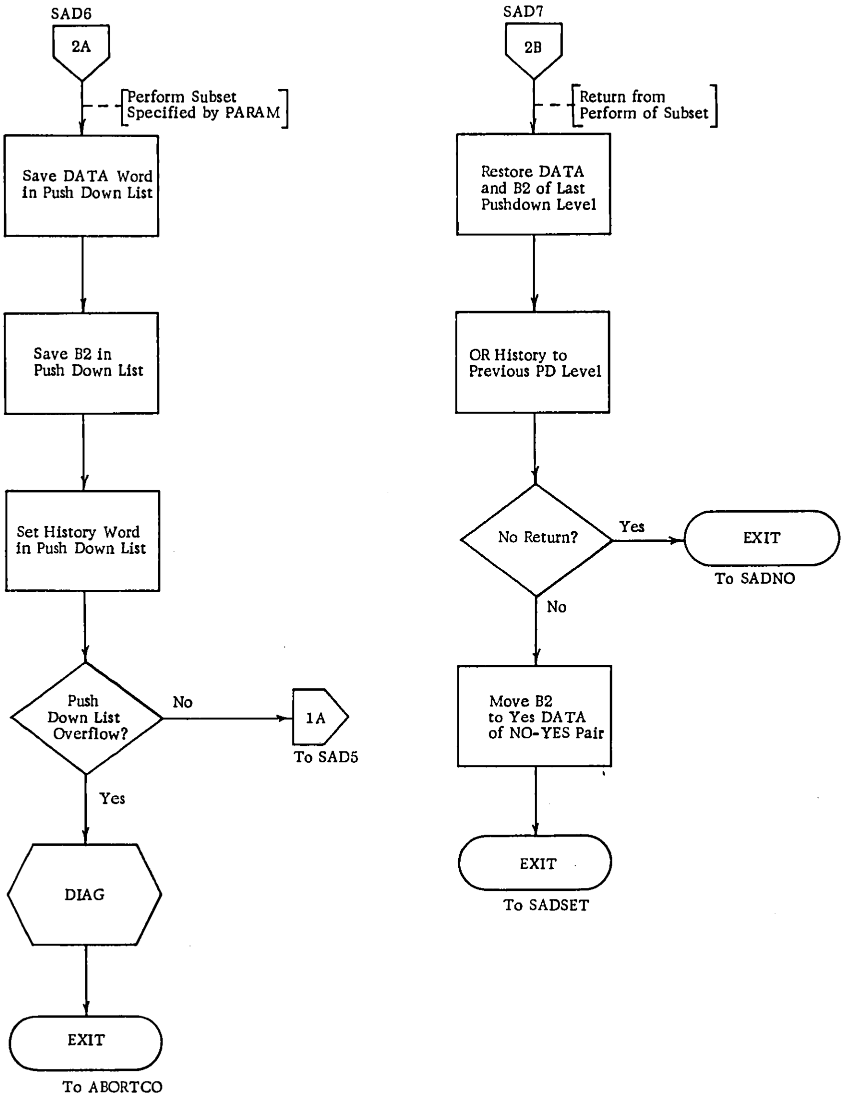


Figure 2-2. Syntax Analysis Driver (SAD) Flowchart (2 of 2)

16. INTEGER--Calls SCAN2 then checks CWIC = 3 and PNTLOC = 0, which indicates an integer. If CWIC = 3 and PNTLOC > 0, indicates a decimal literal; consequently returns to SADNOSN. If integer, return is made to SADYES.

17. SNW--Clear NOSCNFL to enable skipping current word, then returns to SADNO.

SCAN2 AND LEXSRCH

The SCAN2 program separated the next available character string from the source program and forms and identifies a source "word." SCAN2 obtains source cards from the source input file one block at a time when needed. As it inspects each character in a string it classifies it into one of five categories and stores the "word" in a sequential set of cells: (CURNWD, left-justified with the word length count as the high-order character in CURNWD + 0). The word descriptor 12-bit code is right-justified in CWIC.

SCAN2 is constructed to have entry points:

1. To obtain next word.

   Calling sequence:

   ```
   RJ     SCAN2 (Normal)
   ```

   or

   ```
   JP     SCAN (Syntax Analysis Driver)
   ```

2. To reexamine the previous character string.

   Calling sequence:

   ```
   SX6    1
   SA6    NOSNFL (Set No SCAN)
   (Then use same call as in A.)
   ```

3. Special entry used by INCLUDE verb processing.

SCAN2 is also able to:

1. Skip to the beginning of the next logical source card.

   Calling sequence:

   ```
   SX6    2
   SA6    SKIPOPS
   (Then call SCAN2 as in A.)
   ```

2. Skip past the next period followed by a space and give no diagnostic messages if invalid character conditions are sensed prior to the period.

   Calling sequence:

   ```
   SX6    1
   SA6    SKIPOPS
   (RJ SCAN2 or JP SCAN.)
   ```

When SCAN2 is asked to obtain a word, it obtains the next word bounded by blanks or punctuation. As words are obtained, the left delimiters "," and ";" are sensed and the flags COMAFLG and SEMIFLG set nonzero accordingly, so that the subsequent entry to SCAN2 can test the correctness of the punctuation. If the structure or character usage in the word is illegal, SCAN2 puts out diagnostics via DIAG. The type of words that may be obtained are as follows:

1. <u>Non-numeric Literal</u>

   A non-numeric literal is bounded by quotation marks as shown below:

   "(≠)"

   Quotation marks are ≠ signs in the 6400 DISPLAY CODE. The literal not including the quotation mark is stored in CURNWD. The length of the literal is available.

2. <u>Numeric Literal</u>

   A numeric literal is composed of all numeric characters, with or without a leading sign and/or decimal point. A decimal point may not be at the right end. The entire literal, both integral and fractional parts, but not the decimal point will be stored at CURNWD. The sign indication, decimal position and total length is available on exit.

3. Lexicon Words

Lexicon words are found by looking up the words in the lexicon list. The corresponding lexicon control word containing an assigned lexicon code number is available on exit. The =, *, **, /, +, and _ are recognized as lexicon words when they have a space on either side.

4. Name

Name may be of the following types:

a. All words not found in the lexicon list.

b. Words with illegal character (by default).

c. Words with trailing hyphen, imbedded hyphens or leading hyphen on non-numeric words.

5. Period, (and)

The following three characters are treated as separate words by the Syntax Analysis Driver (SAD):

(	as the left delimiter, causes return to SAD.
)	as the right delimiter causes a character backspace and the string recognized previous to the) is identified and return is made.
.	as the right delimiter causes a backspace to itself, and exit is made.

The LEXICON contains a list of COBOL reserved words. A separate LEX list is allocated for each division analysis with the reserved words pertinent to that division. An attempt has been made to recognize all illegal uses of reserved words throughout compilation. This causes some LEXICON overlays to contain the same words, but different LEXICON numbers.

The 12-bit descriptor code found in CWIC upon return from SCAN2 is either the associated lexicon number (if a reserved word), or one of the special control numbers 000 through 004 as follows:

000	Name (or string with invalid character).
001	"Not in" jump return.
002	Non-numeric literal.
003	Numeric literal.

Special syntax analysis external subroutines examine numeric literals and names for specific types. For example:

1. Literals
    a. Unsigned
    b. Signed
    c. Integer or Decimal (PNTLOC = 0)
2. Names
    a. File name
    b. Report name
    c. Routine name
    d. Subroutine name
    e. SWITCHNAM
    f. Other (data name)

SCAN2 returns the following specific information on exit:

1. LEXICON number or control code number in CWIC.
2. Left delimiters occurrence flags for "," and ";". These flags are COMAFLG and SEMIFLG.
3. Sign +1 or -1 in cell SNG; SGN=0 if unsigned.
4. Point location (numeric only). The cell PNTLOC is zero if no decimal point, or a positive integer denoting disposition of the actual decimal point from the rightmost end of the decimal literal.
5. Character count of string in CHARCNT (in binary).
6. Number of words of CURNWD containing character string in bits 59-54 of CURNWD + 0.
7. A margin usage indicator COL8FLG.

SCAN2 exits one of two ways: if called by the Syntax Analysis Driver via:

```
                 JP      SCAN
        SCAN     RJ      SCAN2
```

a test is made upon return to SCAN + 1 to compare the PARAM and the lexicon number of control code found in CWIC. If these two match, X4 is set to 1 and a yes jump is made (JP SADYES). Otherwise, the no jump (JP SADNO) is made.

The second exit from SCAN2 simply returns to the calling external subroutine where CWIC is tested for content.

Special flags are given to SCAN2 in the following cases:

1. PICTURE

   SCAN2 uses CHAR to extract the picture, one character at a time, and stores them one char/word in PICTEMP (in Picture Encoding Routine). No diagnostics are issued for illegal characters and the optional word IS, if used, will be bypassed with no action required by the syntax driver routine. See Picture Processor Description, Section 3, for details.

2. INCL2

   SCAN2 jumps to a special INCLUDE routine, REPLACE, which does the replacing while compiling the INCLUDE sections or paragraphs from the COBOL library.

3. INCL1

   SCAN2 jumps to the PASSI INCLUDE processor, which makes one full pass over the INCLUDED section or paragraph in the COBOL library and sets up the REPBY table for REPLACE.

4. COPYFLG

   COPY FROM LIBRARY sets this flag to inform SCAN2 to duplicate source from the COBOL library.

Upon entry to SCAN2, NOSCNFL is checked for the reexamination option. If OFF, the previous left punctuation is checked for correctness. Diagnostics are issued for incorrect "," and ";" as left delimiters. The starting word column is then checked for correct A and B margin beginning columns. Diagnostics may be issued accordingly.

The source card images are given to OUTPUT for subsequent source listing. CHARBRK sets up the source input card image for OUTPUT.

SCAN2 is the control routine for the subordinate SCANNER routines listed below which perform specific functions:

1. Character Getter Routine--CHAR (MACRO)

CHAR is a MACRO that generates in-line code to load up the next input source character. The CHAR MACRO looks as follows:

```
        CHAR

        SA4      B5          B5 = CHARBUF+N = COLUMN

        SB5      B5 + B6     B6 = 1

        NZ       X4,*+2

    +   RJ       ENDCRD      GET NEXT CARD AND RETURN THE NEXT
                             CHAR IN X4, RIGHT JUSTIFIED.
```

A byte of 00 in X4 indicates end-of-card. ENDCRD is then called to read in the next card and to CHARBRK into CHARBUF.

ENDCRD performs the necessary hyphenation and beginning column usage tests and issues diagnostics accordingly. Column 7 is check for a blank. If blank, Column 8 is checked for nonblank. If nonblank, COL8BE4 is set and B5 is set to Column 8 and exit is made. If Column 8 is blank, ENDCRD positions to the last column checked first nonblank and returns a blank in X4 and B5. If any of the Columns 9, 10, or 11 were nonblank, a diagnostic is issued.

If Column 7 is a hyphen, CHAR positions to the first nonblank character and loads it into X4. If any column between Column 7 and 12 is nonblank, a diagnostic is issued before exit is made.

If something other than a blank or hyphen occurred in Column 7, a diagnostic is issued and Column 8 is checked for nonblank. If nonblank, COL8FLG is set and B5 is positioned to Column 12. Return is then made.

An end-of-card sensed while in the NON-NUMERIC LITERAL mode causes 73-(last column)=N blanks to be stored in CURNWD to compensate for any logical blanks of a continued literal that were suppressed by 2RC (card-to-disk routine).

2. Non-Numerical Literal Getter--NNLIT

SCAN2 calls NNLIT to store all non-numeric literals triggered by a left quote or apostrophe in CURNWD. It, in turn, uses CHAR to feed one character at a time for storage in CURNWD until it finds a right quotation mark followed by a period, space, comma, or semicolon.

This routine truncates (on the right) any literal longer than 255 characters. The quotation marks bounding the literal are neither stored nor counted in the character count.

CHARBRK is called by ENDCRD to get the next source card from either the CHTEMP (normal) or CTEMP2 (Includes and Copies) buffer and breaks it down into one character buffer CHARBUF. When the source block buffer is near empty, SRCRDGT is called to read in the next block, thus keeping a full buffer for the SCAN2 routine. If OUTADD ≠ FRSTOUT (OUT of 00 byte) upon entry to CHARBRK, the previous source card is sent to the OUTPUT buffer for listing. Printing one card behind always necessitates a listing of the last card before end-of-file action is taken.

A missing leading quote on a continuation card of a non-numeric literal causes a diagnostic to be issued and processing continues.

3. Word Getter Routine--NUMLIT

This routine controls the analysis of all words other than pictures and non-numeric literals. It decides if a word is a numeric literal, LEXICON word, or name. It checks for invalid character combinations. Within the word getter routine, there are two subroutine sections, DISCODE and STORIT.

DISCODE examines and identifies each character. The word type is determined and hyphenation is detected and processed.

STORIT stores each legitimate character of the string in CURNWD and test for a maximum of 30 characters. Words greater than 30 characters are truncated on the right and a diagnostic is issued.

Operators, left and right parentheses, and periods are identified instantly and stored in CURNWD. The appropriate LEX number is stored in CWIC.

Numeric literals are identified and 003 put in CWIC.

Alphanumeric words are tagged as names (000 in CWIC).

> Alphabetic words are tested by LEXSRCH for possible match to a reserved word. If a match is found, LEXSRCH returns to INLEX in NUMLIT. Otherwise, the NOTIN return to NUMLIT is taken.
>
> Character strings with invalid character combinations are tagged as names.
>
> Before exiting from NUMLIT, the last word of the stored character string is left-justified in CURNWD and is blank-filled on the right.

SCAN2 employs SRCRDGT to read in the next block of source input cards. SRCRDGT utilizes the File Manager system for its I/O.

Separate LEXICON lists are allocated for the ID-DD and the Procedure Division. LEXDATA and LEXPROC are each divided into four parts.

Part IV is an indexed jump list of 32 words. The low-order addresses of the first 28 words contain the jump addresses in Part I of the one word LEXICON subsets, e.g., ONEA, ONEB, ..., ONEZ, SPEC. The 27th word contains SPEC, which is the subset of COBOL special characters.

Part I is the actual list of one-word reserved subsets arranged in alphabetical order with respect to the subset headings. The elements are arranged in order of predicted descending frequency within each subset; this arrangement optimizes the linear word search.

Part II is the list of two-word reserved word entries ($\geq$ 10 characters). They are arranged in order of predicted descending frequency. No attempt has been made to jump to any particular subset of the two-word entries due to the relatively few entries for comparison.

Part III is a constant section containing the corresponding lexicon numbers for the one- and two-word entries of the reserved word list. They are grouped four 12-bit numbers per word, left-justified in each quadrant.

LEXSRCH examines the subset of the LEXICON corresponding to the word length and first letter of the candidate for reserved word identification. If a match is found, the lexicon number is ascertained and returned in CWIC for future PARAM comparison in SCAN or a syntax table subroutine which had called SCAN2.

A cell called LEXOVLA, in CONTROL, reflects the absolute address of the current LEXICON list referenced by LEXSRCH assembled in SCAN2.

Table 2-2 illustrates the format of the lexicon table.

Note: The last four cells are needed by the LEXSRCH routine to establish LIMITS for its searches.

Table 2-2. Lexicon Table

ONEA	1	A Δ ——— Δ		
	1			
	1			
ONEZ	1	Z Δ ——— Δ		
SPEC	1	· Δ ——— Δ		
	⋮			
	1	/ Δ ——— Δ		
ENDEX	1	** Δ ——— Δ		
TWUWDS	2	T W O Δ W O R D Δ		
	E N T R I E S Δ Δ Δ			
	2	etc.		
	2	E N D Δ O F Δ T Δ		
	O Δ W O R D S Δ Δ Δ			
LEXNRS	00005	01725	00002	01602
	00003	01601	00001	00006
	⋮			
LEXPROC		ONEA		
+1	0	ONEB		
	⋮			
+25		ONEZ		
+26		SPEC		
+27		ENDEX		
+28		ONEA		
+29		LEXNRS		
+30		LEXPROC		
+31		TWUWDS		

LEXDATA and LEXPROC are identical lists of the reserved words. Nonzero lexicon numbers are assigned to the words of each list if the words are legal in the ID-DATA Division or Procedure Division respectively; otherwise, a zero lexicon number is assigned.

LEXSRCH senses incorrect reserved word usage by finding a match in the list whose lexicon number is zero. Upon return to SCAN2, a diagnostic is issued stating that for this Division there is either:

1. Incorrect reserved word usage, or
2. Illegal CDC reserved word usage.

Certain words in the reserved word list are marked as nonstandard COBOL '65. When they are used as optional, precautionary diagnostic "NON-STANDARD RESERVED WORD USAGE" will be issued.

The lexicon numbers consist of a coded 12 bits of which:

Bit 11 = 1	If the reserved word is not DOD COBOL '65 standard.
Bit 9 = 1	If the reserved word in a sentence separates such as ELSE END . DECLARATIVES PROCEDURE, etc. or is a key word.
Bit 8 = 1	If the reserved word is a key word signaling a new statement.
Bit 7 = 1	If the key word can be imperative.
Bit 6 = 1	If the key word can be conditional.

Delimiter Handling

The structure of the COBOL language coupled with the method of 6600 COBOL syntax analysis requires a different approach to certain delimiter tests.

1. Normally the syntax analysis table does not indicate a left-punctuation test. If left punctuation has appeared, a diagnostic message is issued the next time SCAN gets a new word.

2. The syntax analysis table indicates places in the language where a "," or ";" is allowed. This can be implemented by causing SCAN to cancel the fact that a "," or ";" has appeared, so that a diagnostic message will not be issued when SCAN is entered to get the next word.

3. Some parts of the syntax table (in order to save repetition of the code in the table), sometimes allow "," and sometimes not. This condition is handled by the routine CCT (Conditional Comma Test). Normally the "," is not allowed (in which case no attempt is made to cancel the fact that a "," has appeared) but when an indicator has been turned on by ACCT (Allow Conditional Comma Test), CCT cancels the fact that a "," has appeared so that a diagnostic message is not issued. The routine DCCT (Disallow Conditional Comma Test) returns the indicator to OFF (normal).

4. At places in the language where THEN has occurred, it is necessary to disallow a ";" to the right. SCNANW (Semicolon Not Allowed Next Word) sets a flag that prevents the fact that a ";" has appeared from being canceled. The flag remains set until the time SCAN exits with a new word.

5. The DOD rules are very strict as to where a "," and where a ";" is allowed; however, many programmers use the "," and ";" interchangeably. The following usage is allowed in 6600 COBOL:

   When the X option if OFF (so that extended diagnostics are not required) every place in the syntax table that indicates that either a "," or a ";" is allowed, cancels both indicators (one for "," and one for ";") so that the two punctuation signs may be used interchangeably.

6. Starting in Column 8, certain words must appear. Normally a diagnostic message is issued when a word starts before Column 12 the next time SCAN2 is entered to get a word. However, if "A" is indicated, a diagnostic is issued if it does not begin in Column 8, and the fact that it begins prior to Column 12 is canceled.

Copy (From Library)

Library Copies are made from a random file whose name is specified by the S-parameter on the COBOL control card. This file is written by COPYCL (see Appendix C). ITEMCOP is called by SCAN2, when the COPYFLG is set after the E.O.S. is detected on the COPY clause, to integrate the copied source statements at this time.

ITEMCOP searches the COPYCL random file index for a data-name-3 match. If not found, ITEMCOP requests SCAN2 to skip to the next source item. If found, the disk location is input to SCAN2's input routine, which reads the copy item symbolic source into CTEMP2. Compilation of the data-name-3 item proceeds from this buffer while its level numbers, if data-name-3 was not an 01 item, are qualified by the level-number gradient of the data-name to which the COPY clause is subordinate.

Nested copy's from LIBRARY are allowed up to five levels. Overlapping copies or copies including themselves, are not, of course, allowed. Nested copies utilize the same common input buffer, CTEMP2. CHTEMP buffer pointers and random access disk address information is kept in a pushdown list to be used in unnesting the copies. UNNESTC is called to position to finish the previous nested copy, if present, or to return to the source input to continue normal compilation.

INCLUDE Procedure Processing

There are two phases to the INCLUDE procedure processing. The first phase stores the section and/or paragraph name of the library element in PNBUF and PQBUF. Then the REPLACING (item) BY pair, if present, is analyzed and replacement pair clusters are constructed into a table named REPBY. If no replacing option is used, a default replacement pair consisting of the library paragraph name vs. the section and/or paragraph name to which the INCLUDE procedure is subordinate.

The second phase has two subphases. The first subphase consists of a call to ITEMCOP, with INCL1 set to nonzero, to search the RANDOM file, specified by the S-parameter on the COBOL compilation control card, for the request item named by PNBUF/PQBUF. If not found, a diagnostic is issued and the INCLUDE section or paragraph is ignored. If found, the named record is read into CTEMP2. ITEMCOP positions past the headers to the first procedure in the item and jumps to SADYES.

Subphase two of the second phase passes over the library record constructing word clusters, similar to those produced during phase one, pass over the replacing pairs, for each character string. These clusters are formatted as shown in I3 and are written out on a sequential binary file named DDINCL. When an EOR is sensed in the library element, the file is rewound and the INCL2 flag is set to tell SCAN2 to call REPLACE to check for replacing while compiling the cluster from DDINCL.

REPLACE will read a cluster from DDINCL and compare it to the replacement pairs in the table REPBY. If the cluster and all of its qualifiers (if any) match any left-string of qualifiers of a replacement pair in REPBY, then the right string of the REPBY pair is substituted for the string of clusters in DDINCL.

The REPBY table is fixed-length, and therefore, may overflow if the REPLACING (item) BY option is too long. If an overflow should occur, a diagnostic will be issued and the whole INCLUDE procedure will be ignored.

The following list of SCAN2's working storage is saved in the INCLUDE clusters.

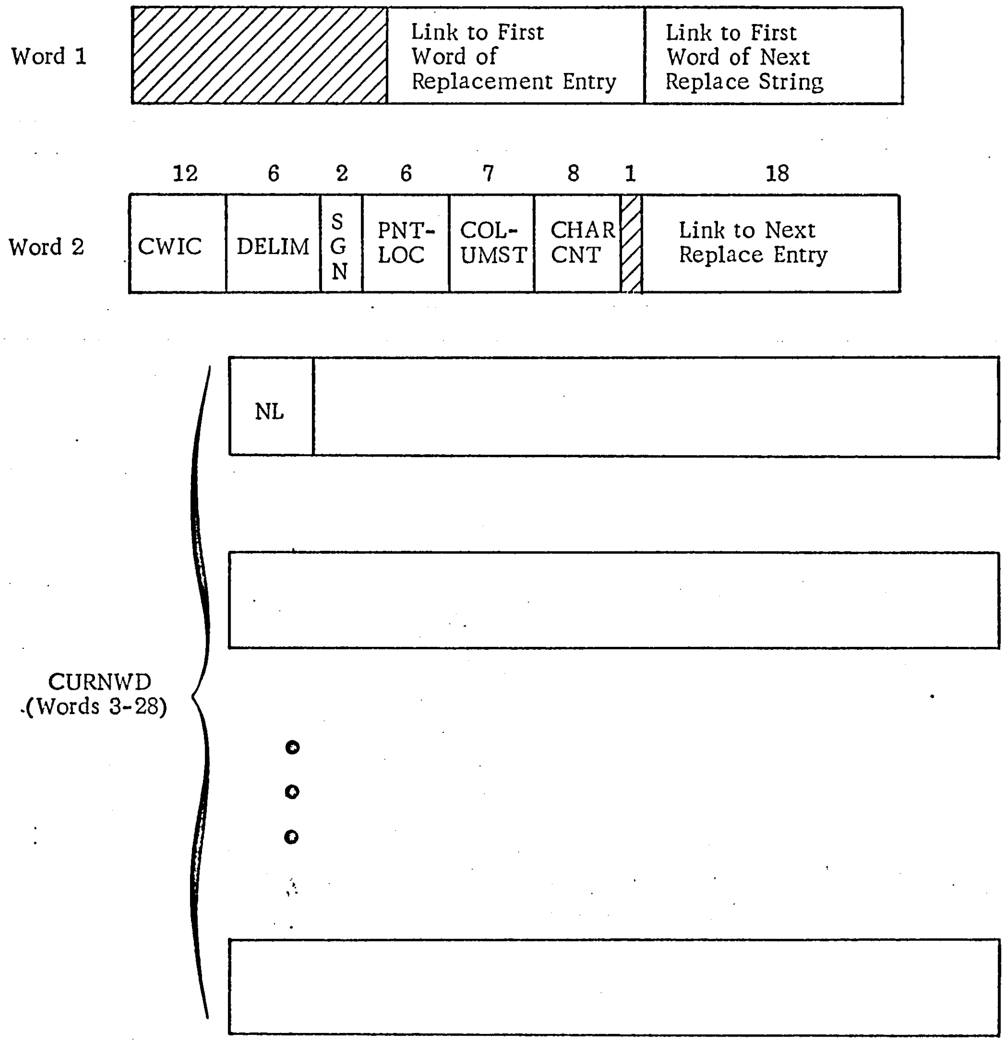
1. CWIC (lexicon or pseudo-lexicon number)
2. Delimiter flags (COMAFLG, SEMIFLG, COL8FLG, COL8BE4, and PUNFLAG)
3. Point location (PNTLOC)
4. Sign (SGN (2 bits))

a. 0 = not signed
b. 1 = + signed
c. 2 = - signed

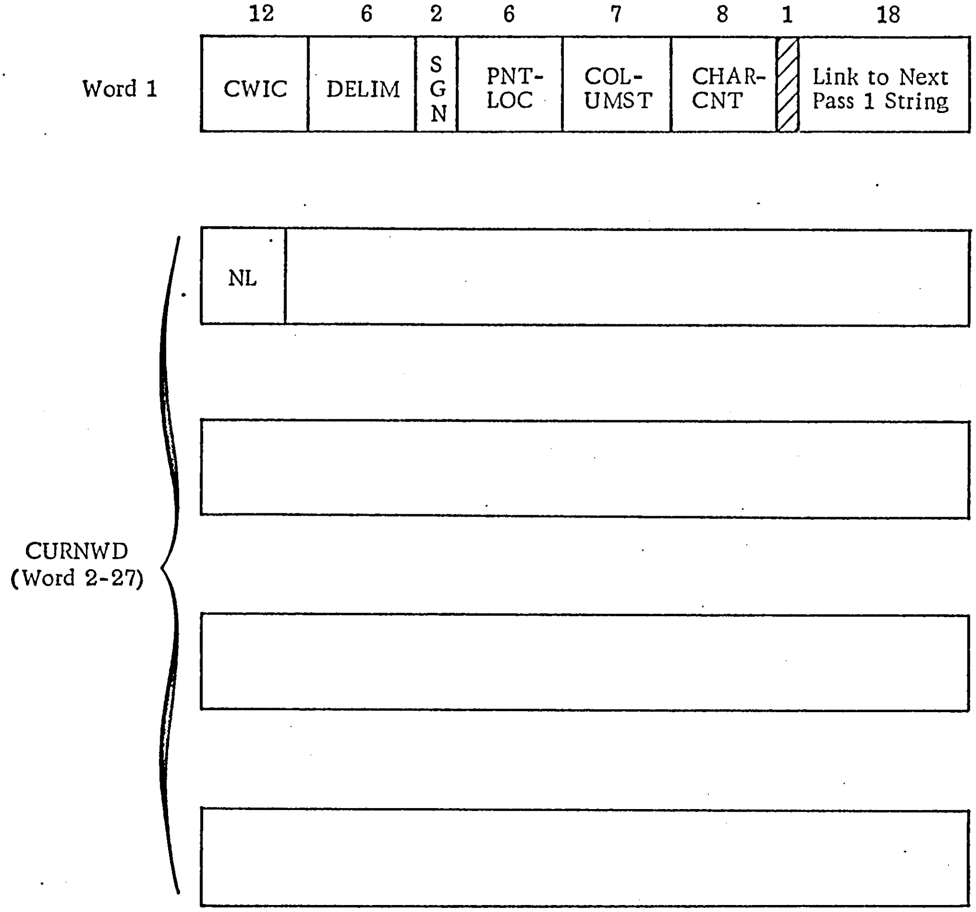
5. Beginning column of word (COLUMST)
6. Number of characters in string (CHARCNT)

The following formats show the left- and right-replacement clusters and the Pass 1 library source clusters:

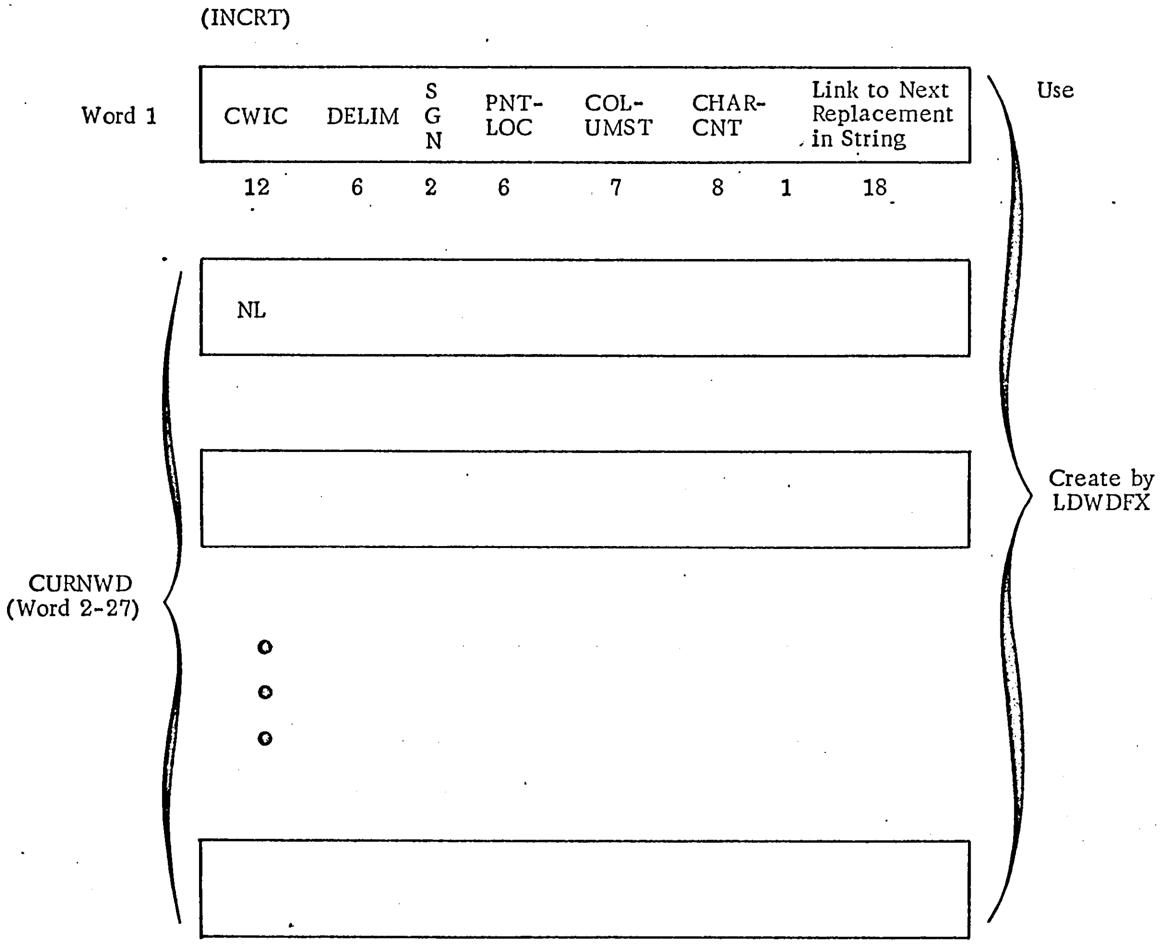
REPLACING (ITEM) BY LEFT CLUSTER FORMAT (REPBY)



INCLUDE PASS 1 CLUSTER FORMAT



REPLACING (ITEM) BY RIGHT CLUSTER FORMAT (INTERSPERSED WITH REPBY)



COMPILER I/O INTERFACE

The compiler performs I/O through the File Manager furnished in the SCOPE system. This causes compiler use of File Environment Tables (FET) similar to those generated for object code.

All printer output is buffered by BLOCKIO and output to the output logical file name specified by the L-option on the COBOL control card. BLOCKIO uses WRITOUT to buffer its input. The output CIO buffer can be written out at any time by an RJ to FLUSH, which does a WRITER on the output file. Registers A5 and A7 are destroyed by either a call to BLOCKIO or FLUSH. Page ejection and header printout is controlled by BLOCKIO.

The format requirements for printer images given to BLOCKIO are as follows: (See printer formats in Section 5.)

1. Carriage control character as first character. (This character is not printed.)

2. End-of-line terminator.

   The standard OUTPUT EOL is at least two zero bytes (six bits/byte). If the first of the two zero bytes is the rightmost byte of the last word of the printer image, then a whole word of zero bytes must follow to complete the EOL requirement.

3. No zero bytes between CC and EOL (i.e., inclusive image must be display coded).

4. Line image $\leq$ 136 characters.

   It should be noted that if the images are > 136 display characters, the PP OUTPUT routine will use the 137th, 273rd, ... character for the carriage control characters that are not printed.

   The CALL to BLOCKIO is as follows:

   ```
   RJ     BLOCKIO
   VFD    A30/BUFFER,N30/M
   ```

   where

   BUFFER is relocatable label address of printer image being output,

   and

   M is number of whole words of display-coded printer image. M should include EOL terminator.

The CALL to FLUSH is as follows:

```
        RJ      FLUSH
```

There are four main compiler CIO buffer areas. They are used for source input/output procedure division TREEOUT/TREEIN, and the Assembler relocatable binary output.

The buffer size of these buffers is determined by the field length of the compilation (assuming a minimum of 53000_8). Each buffer occupies 1/8 of the core space between 50000_8 and the top of the field length. The CIO areas begin at a location half way between 50000_8 and the field length, and run to the end of the field length. The remaining one half of the area is utilized by extending the top of the PNT.

DIAG - DIAGNOSTIC OUTPUT ROUTINE

The diagnostic message output routine retrieves the diagnostic English error message requested by the call and outputs the message to the output file. (See Table 2-3.)

The calling sequence to DIAG consists of the following:

```
        RJ      DIAG
        DATA    N
```

where

N is the message number.

Registers A5 and A7 are destroyed by DIAG.

The maximum diagnostic number is 999. If the number exceeds this limit, a diagnostic is included and issued by DIAG itself. Diagnostic messages have a 10-word maximum length.

The printer-line image for the messages includes the following: (See format in Section 5.)

1. Attention-getting field of asterisks ** **.

2. A 3-digit decimal diagnostic message number within the asterisks. Each number is unique for every spot in the compiler routines where an error can be detected.

3. A 2-digit card column indicator prefaced by a special character 'cc'. This number specifies the card column of the word or position of syntax where the error is sensed. If cc=0, the message applies to the previous card.

Table 2-3. Sample Diagnostic Table Format

<table>
<tr><td>DAGNOS1</td><td colspan="5">DL C000</td><td colspan="5">DL C001</td></tr>
<tr><td></td><td colspan="5">DL C002</td><td colspan="5">DL C003</td></tr>
<tr><td></td><td colspan="5">DL C004</td><td colspan="5">DL C005</td></tr>
<tr><td></td><td colspan="10">⋮</td></tr>
<tr><td></td><td colspan="5">DL C096</td><td colspan="5">DL C097</td></tr>
<tr><td></td><td colspan="5">DL C098</td><td colspan="5">DL C099</td></tr>
<tr><td></td><td colspan="10"></td></tr>
<tr><td>C000</td><td colspan="10">Δ ——— Δ</td></tr>
<tr><td>C001</td><td>X</td><td>Δ</td><td>I</td><td>N</td><td>C</td><td>O</td><td>R</td><td>R</td><td>E</td><td>C</td></tr>
<tr><td></td><td>T</td><td>Δ</td><td>S</td><td>T</td><td>A</td><td>R</td><td>T</td><td>I</td><td>N</td><td>G</td></tr>
<tr><td></td><td>Δ</td><td>C</td><td>O</td><td>L</td><td>U</td><td>M</td><td>N</td><td>Δ</td><td>B</td><td>E</td></tr>
<tr><td></td><td>F</td><td>O</td><td>R</td><td>E</td><td>Δ</td><td>C</td><td>O</td><td>L</td><td>U</td><td>M</td></tr>
<tr><td></td><td>N</td><td>1</td><td>2</td><td>Δ</td><td>Δ</td><td>Δ</td><td>Δ</td><td>Δ</td><td>Δ</td><td>Δ</td></tr>
<tr><td>C002</td><td colspan="10">⋮</td></tr>
<tr><td>C009</td><td colspan="10">Δ ——— Δ</td></tr>
<tr><td></td><td colspan="10">Δ ——— Δ</td></tr>
</table>

4. A 1-letter indicator from the following list:

   a. C - Fatal error with no execution possible.
   b. E - Serious error with probable execution of part of the object code.
   c. T - Trivial precautionary message (extended).
   d. U - Non-DOD error condition message (extended).

5. An English language error comment. (To conserve space, several diagnostic numbers may use the same error message; consequently, the printed number is more specific than the message.)

6. Source line number, indicating the line in the vicinity of the error. (Most error messages will print interspersed in the programmer listing, but some error messages will print before and some after the source line to which they apply.)

An X-parameter on the COBOL control card determines which degree of diagnostics is printed.

Ranges for the diagnostic numbers for the different compiler segments are as follows:

OVERLAYS 0,0 and 1,0	1 - 99
OVERLAYS 1,1 thru 1,4	100 - 499
OVERLAYS 1,5 thru 1,6	500 - 799
OVERLAYS 2,0 thru 2,3	800 - 999

Diagnostic overlay elements (of which there are 10) containing ≤ 100 diagnostic messages per element will be loaded in conjunction with the various passes of the compiler. Their absolute locations (relative to RA) are stored in the CONTROL working storage in the cells named DAGLOC1, DAGLOC2,,, DAGLOC9, DGLOC10.

DIAG first determines in which overlay the requested diagnostic resides. Then the first word of the message is loaded and the degree key is checked for possible extended degree.

If the degree of the potential diagnostic is either T or U, a special flag XTENDMD is checked. If this flag is set, the U and T diagnostic is printed. If not, exit is made from the DIAG routine with no diagnostic issued. All C and E diagnostics are output.

Appendix D contains a list of compiler diagnostic messages.

TREE OUTPUT (TREEOUT) SUBROUTINE

The Tree Output (TREEOUT) subroutine is used for interphase I/O. Each time TREEOUT is called, one tree is written to disk. Each time a section number changes (0-49 is considered no change), the previous logical record is terminated and a new logical record is started. The calling sequence to TREEOUT is:

1. Load X4 Priority Number (0-99)
2. Load B4 with length of tree
3. Load B5 with base address of tree
4. RJ TREEOUT

The format of the trees as they are written to disk is shown in Figure 2-3.

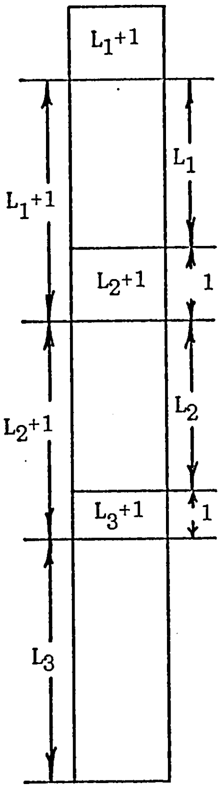


Figure 2-3. Tree to Disk Format

This pictorial diagram represents a logical record of three trees. L_1 is the length of tree 1, etc.

The word immediately preceding the one pointed to by register B5 is destroyed by TREEOUT. Registers preserved by TREEOUT are:

1. X0, X1, X2, X3
2. B1, B2, B3, B6

After the last tree has been written, it is necessary to terminate TREEOUT outputs to the disk by performing the following call:

```
RJ    PHASEND
```

Register preservation is the same as indicated above for TREEOUT.

A flowchart of TREEOUT appears in Table 2-4.

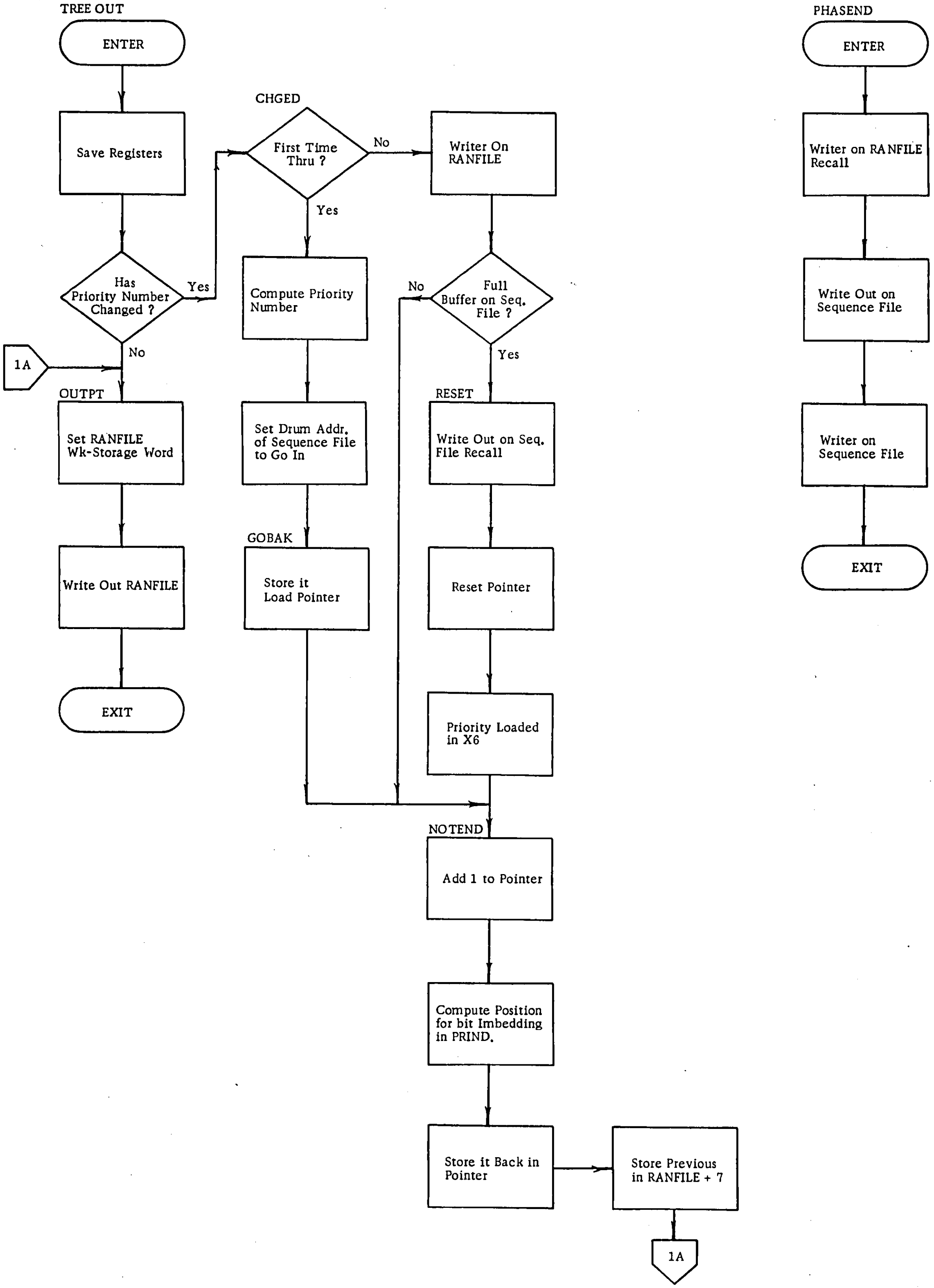


Figure 2-4. TREEOUT Flowchart

TREE INPUT (TREEIN) SUBROUTINE

The Tree Input (TREEIN) subroutine is used for interphase I/O. See Figure 2-5. Each time TREEIN is called, one tree is returned. There are two exceptions:

1. If a priority change, no tree will be returned; instead, special end of priority exit will be taken with the number of the next priority given in X1.

2. If there are no more trees to be input, X1 will contain a negative number.

The calling sequence is:

```
+ RJ      TREEIN
+ Priority change return
+ New tree return
```

Upon exit, B2 contains the address of the new tree. Registers preserved are:

X3 and B1.

TABLE HANDLING

The use of a number of "open-ended" tables are made by various compiler routines. The Data Name table and the Procedure Name table are the two most prominent tables because they are each used commonly by several phases of the compiler. However, many other open-ended tables are also used.

The compiler is designed so that all such tables are located in one big area at the end of available memory. Since the SCOPE loader allows the compiler access to any area within the field length (fl) specified on the JOB card, the open-ended tables are located above the program load area and below the field length. Obviously, certain amounts of surplus core must be furnished by the user before any table work can be done.

Only two open-ended tables can be built at any one time. One begins at one end of available core area and the other begins at the opposite end. The combined length of these tables is limited by one table's extension into a table extending from the other end, in which case every part of available table space is used. Should such a limit be reached in the COBOL compiler, the compilation will be aborted.

Although only two open-ended tables should be "growing" at one time; once a table's maximum extension is determined, another open-ended table may begin from that point, continuing in the same direction. Also, if a table is no longer needed, a new table can begin in its spot. The tables have been carefully designed to make maximum use of the area.

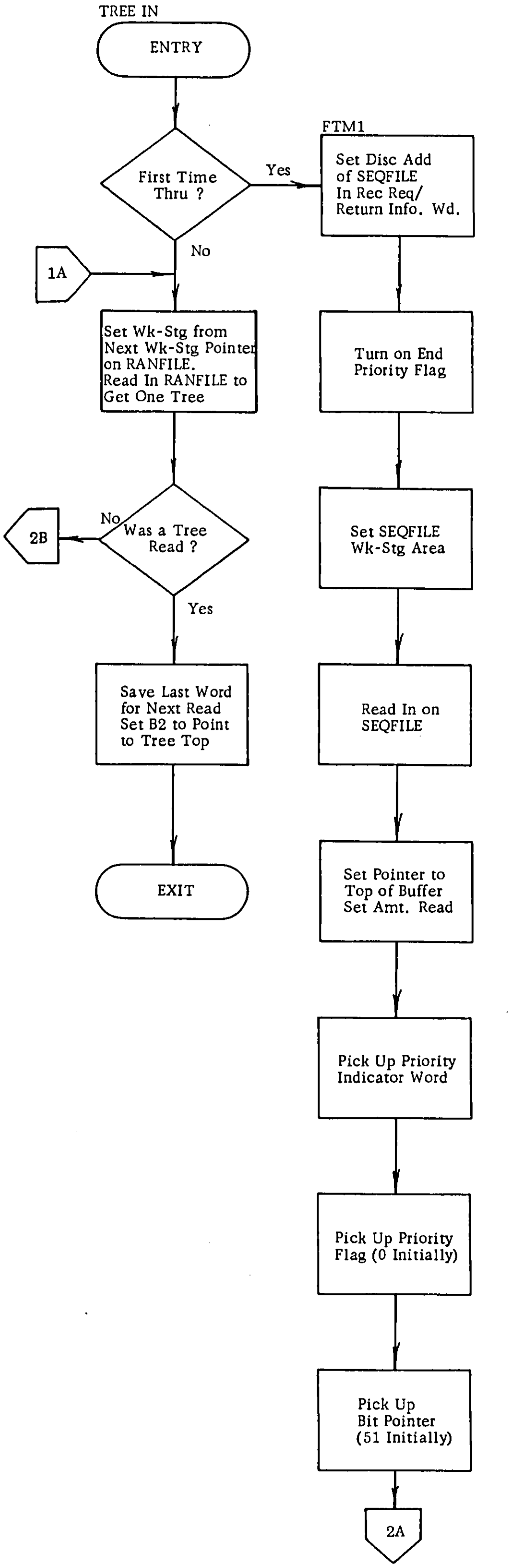


Figure 2-5. TREEIN Flowchart (1 of 2)

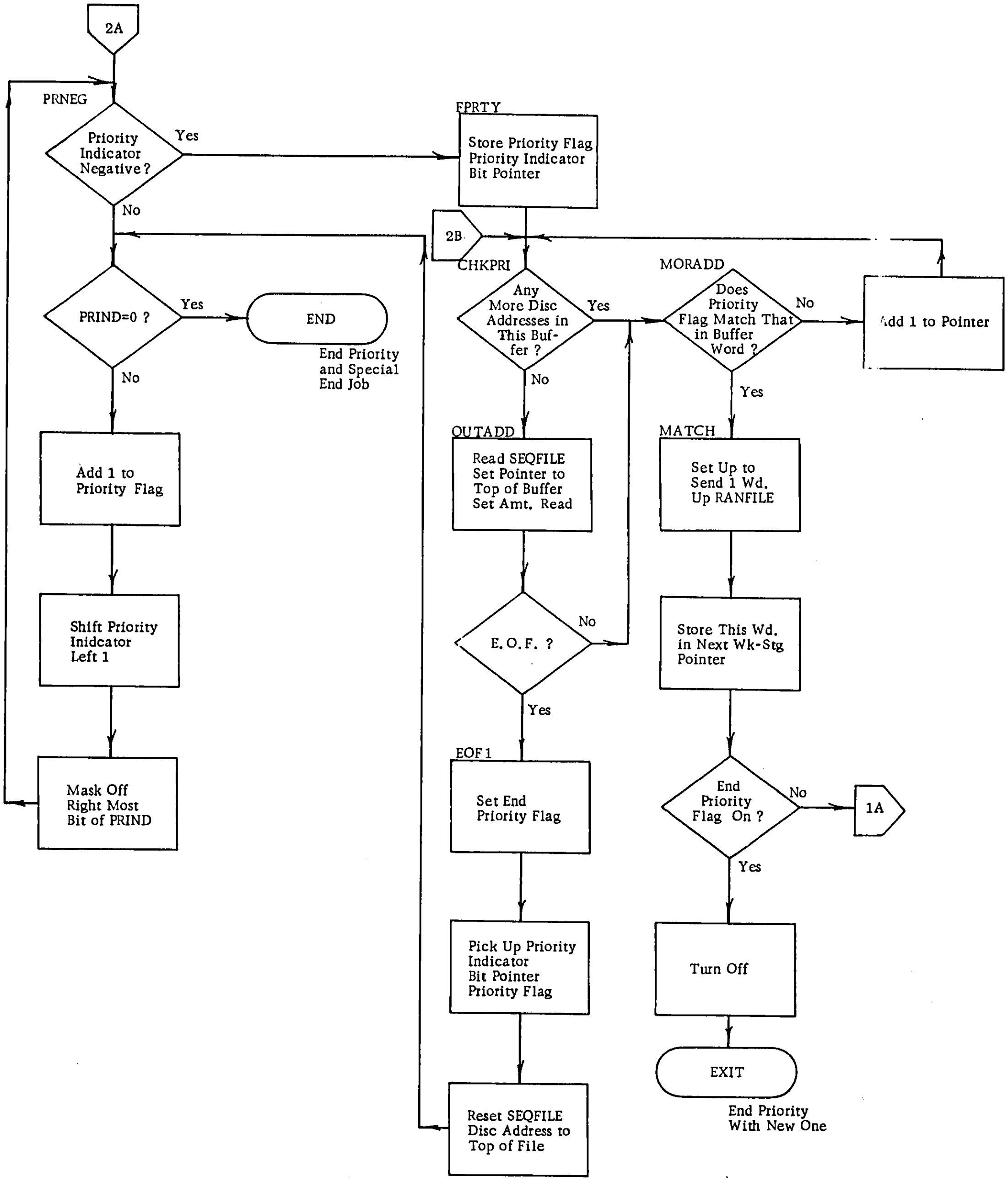


Figure 2-5. TREEIN Flowchart (2 of 2)

Specifically, the Data Name table will begin at the top end of Pass 1B or 1E, whichever is longer, and extend it upward toward RA + FL. When Pass 2 is loaded, it will be allowed to overlay the DNT as the table is no longer needed. Since Pass 2 will need the Procedure Name table, it is not overlayed and is placed above the DNT, extending from the top of the table area downward. The report tables are not needed in Pass 2 and may be overlayed. The DNT length will be determined when the report tables start and thus the latter extend from the former in the area building up towards the top. The Procedure Name table is not started until the length of the report tables is known and extends from the top down toward the completed tables. The PNT is limited by the high end of either the report table or Pass 2, whichever extends the highest. All of these tables exist during Pass 1E.

Several more open-ended tables are used by certain passes. For example, after the report table length has been determined in Pass 1B, subsequent passes build tables upward from the end of the report tables for "local use." After Pass 1E, open-ended tables also extend down from the PNT. Tables not named here, and required for the use of more than one pass, are coordinated so as to not overlay or be overlayed by a "local" table. After Pass 1G, extension upwards is not from the report tables, but from the end of Pass 2 code.

Limited table space is always available to smaller passes between its length and the beginning of the file tables.

Table 2-4 outlines how the table might be stored during the first pass. Indented items do not have a 6-bit identifier.

The initial method used to find entries in the data table is to use a HASH table as an index. As each entry is placed in the data table, its three words of name (30 characters maximum) are hashed as follows:

The three words are added together, ignoring overflow. The resulting sum is equal to its two halves. The resulting value is multiplied by the binary equivalent of 1/511. The first nine bits to the right of the decimal point of the above product is the position in the HASH table of the location of the data table entry. The HASH table will consist of a possible 511 entries. If an entry to the HASH table finds another entry there already, the new data table entry will be linked to the already existing data table entry referenced by the HASH table or to the last link of previous duplicates. Thus strings of entries may be established for duplicate hashing results.

The address of the first word of data name is to be in an address register for the HASH subroutine. The resulting pointer will be in an increment register.

The HASH subroutine will generate a 9-bit hash number from data words, using word one or words one and two of the input data.

Table 2-4. First-Pass Table Storage

Contents of Table	Contents of Table
HL	EDD
NM	EDD2-3
SNI	EP
HL	HL
NM	NM
FD	CN
FD2-27	DDL
HL	HL
NM	NM
FD	CN
FD2-27	DDL
HL	DDL
NM	HL
GDD	NM
GDD2-3	SC
HL	HL
NM	NM
EDD	SC
EDD2-3	GDD
HL	GDD2-3
NM	HL
EDD	NM
EDD2-3	RD
SSC	RD2-00
HL	HL
NM	NM
GDD	RG
GDD2-3	RG2-7
HL	RE
NM	RE2-7
RN	RE
HL	RE2-7
NM	RG
GDD	RG2-7
GDD2-3	RE
EDD	RE2-7
EDD2-3	HL
DDL	NM
DDL	PD
DDL	HL
DDL	NM
HL	PD
NM	PR
EDD	PR
EDD2-3	etc
EP	
DDL	

Legend:

HL - Hash link
NM - Name
GDD - Group data descriptor
EDD - Elementary data descriptor
DDL - Data division literal
PDL - Procedure division literal
SNI - Special name item
SC - Source copy
RN - Renames (66)

EP - Encoded PICTURE
SSC - Subscript coefficient (OCCURS)
CN - Condition name (88)
PD - Procedure def.
PR - Procedure ref.
FD - File table
RD - Report descriptor
RG - Report group
RE - Report elementary

The initial method used to find entries in the data table is to use a HASH table as an index. As each entry is placed in the data table, its three words of name (30 characters maximum) are hashed as follows:

The first three words are added together, ignoring overflow. The resulting sum is equal to its two halves. The resulting value is multiplied by the binary equivalent of 1/511. The first nine bits to the right of the decimal point of the above product is the position in the HASH table of the location of the data table entry. The HASH table will consist of a possible 511 entries. If an entry to the HASH table finds another entry there already, the new data table entry will be linked to the already existing data table entry referenced by the HASH table or to the last link of previous duplicates. Thus strings of entries may be established for duplicate hashing results.

The address of the first word of data name is to be in an address register for the HASH subroutine. The resulting pointer will be in an increment register.

The HASH subroutine will generate a 9-bit hash number from data words, using word one or words one and two of the input data.

The calling sequence is:

```
        RJ      HASH
        Return
```

where register B7 contains the location of a buffer containing the hash data words. The first character of the first word of the buffer must contain the length in words of the data item. If the name or HASH data does not completely fill the last word, blanks should be used to fill the rest of the last word in the item's buffer for display code information and zeros used as filler for binary information.

The return will be to the word following the call with the result in register B7.

The HASH table itself is entered using the HASH result as an index to the table. The HASH table entry format is:

0	Assembler Pointers	PNT Location	DNT Location
6	18	18	18

The DNT field will also be used by the assembler during Pass 2 assemblies.

The compiler data table format is constructed so that each data item has a pointer to the next more dominant item in the hierarchy and also a pointer to other items having the same name which in turn will point to the name itself. Also, each item includes its level number.

A typical data division is shown in Figure 2-6.

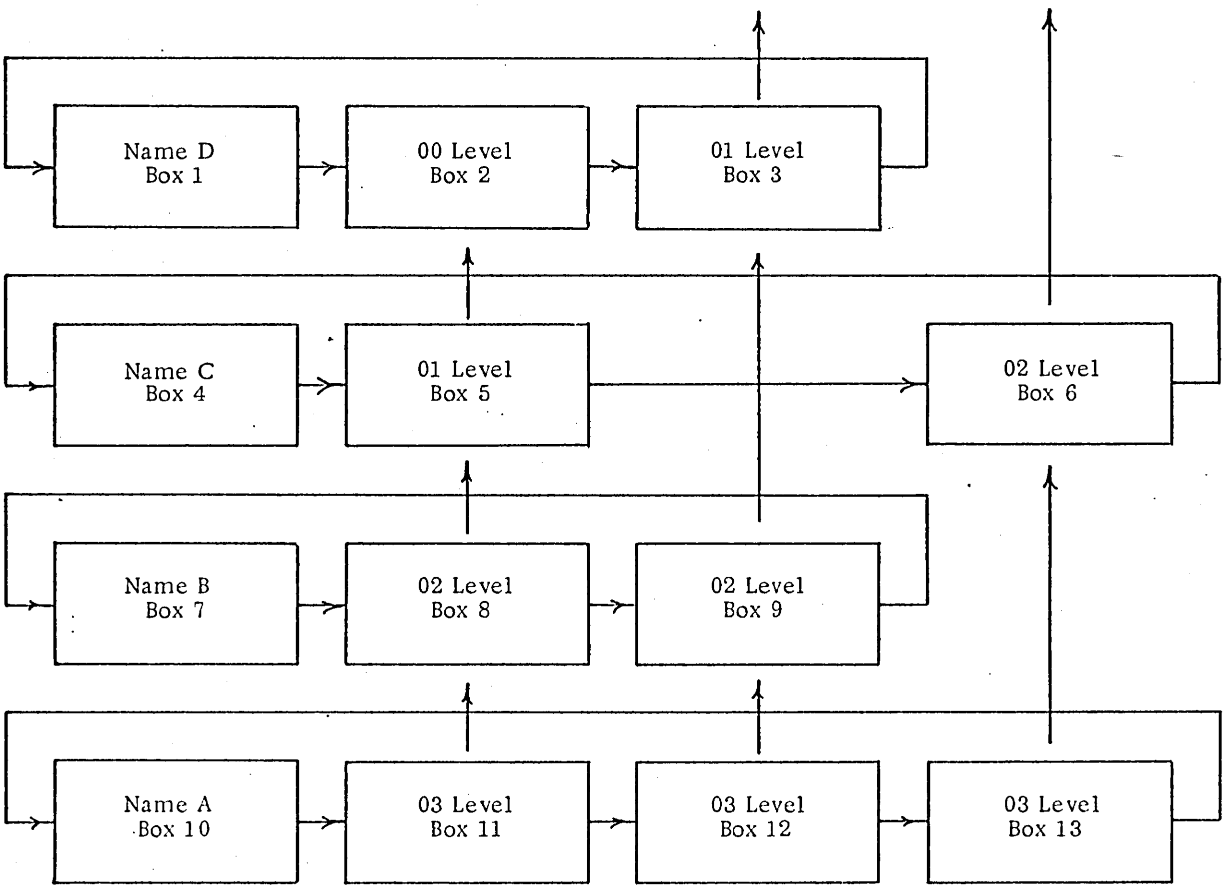


Figure 2-6. Example of Data Division

Suppose it is desired to locate A of B of D:

1. Start at the name A (Box 10).
2. List all A items as candidates (Box 11, 12, 13).
3. Go "up" the hierarchy from one of the candidates (Box 11 to Box 8).
4. Go "around the circle" through all the items of like name until the name (Box 7) is located.
5. See if this name is next in the list of qualifiers (A of B of D). If it is, step to the next name in the qualifier list. (It is, so we step to D).
6. Go "up" the hierarchy again (same as Steps 3, 4, 5). This time the next name in the qualifier list (D) does not match the name "up" the hierarchy. In this case, do <u>not</u> step to the next qualifier but do go "up" the hierarchy (same as Steps 3, 4, 5) (to Box 2).
7. A candidate is disqualified if the hierarchy is exhausted before the list of qualifiers is used up.
8. Otherwise, a candidate is retained.

In the example Box 11 and Box 12, both are retained, but Box 13 is disqualified. (B, the first qualifier, would not be found before the hierarchy is exhausted.)

If no candidates are retained, the item is not defined.

If one candidate is retained, the item is correctly defined and referenced.

If two or more candidates are retained, the item is ambiguously referenced (as in Figure 2-6).

In testing to determine if the name and its qualifying names is defined, it is not necessary to actually compare all the names encountered along the way. In locating A of B of D, set up a table consisting of a pointer to the HL item for each of the names A, B and D. Use the HASH table to find each pointer. Start with the dominant item for one of the A items (Box 8 via 11) and find the location of the HL item for its name. Compare this location with the one from the table for B. If equal, continue up the dominance and down the table. If not equal, hold your position in the table and continue up the dominance.

ITEMCOP - RANDOM FILE ITEM SEARCH ROUTINE

Purpose

ITEMCOP reaches the random file named by the S-parameter for the item specified by a COPY or INCLUDE from library statement. (See Figures 2-7 and 2-8.)

Calling Sequence

1. For COPIES:

   a. COPYDNT - absolute address of last qualifier of item requested.

   b. LEVCOPY - level number of item.
      0 - nonrecord copy.
      1-88 - record copy.

   c. COPYFLG - set nonzero.

2. For INCLUDEs:

   a. COPYDNT - absolute address of procedure or section name.
   b. LEVCOPY - set to -1.
   c. COPYFLG - set nonzero.
   d. INCL1 - set nonzero.

Routines Called

DIAG, SAD, SCAN2

Operation

ITEMCOP reads the random file index (if not already open). The requested record name is hashed and its corresponding record disk address is placed in INCOPY+6. If the disk address is zero, the record does not exist on the random file, and a diagnostic is issued. Exit is then made to SADNO. Otherwise, the named record is read from the random file. If the name of the record matches that of the requested record, INCL1 is checked for INCLUDE. If it is an INCLUDE request, INCLOOD positions to the first procedure of the requested paragraph and returns to SADYES. If INCL1 is zero, QUALCHK positions to the first compliable word of the COPY item. If a record COPY, the level number of the LIBRARY item is returned in binary in NEWLEVN. QUALCHK returns to SADYES.

If the name of the record read does not match that of the requested record name, the linkage word (word 1 of the record) is checked for nonzero. If zero, the element requested, is not in the random file, and a diagnostic is issued and the SADNO exit is taken. Otherwise the nonzero link address represents the absolute disk address of the next record with the same hash index.

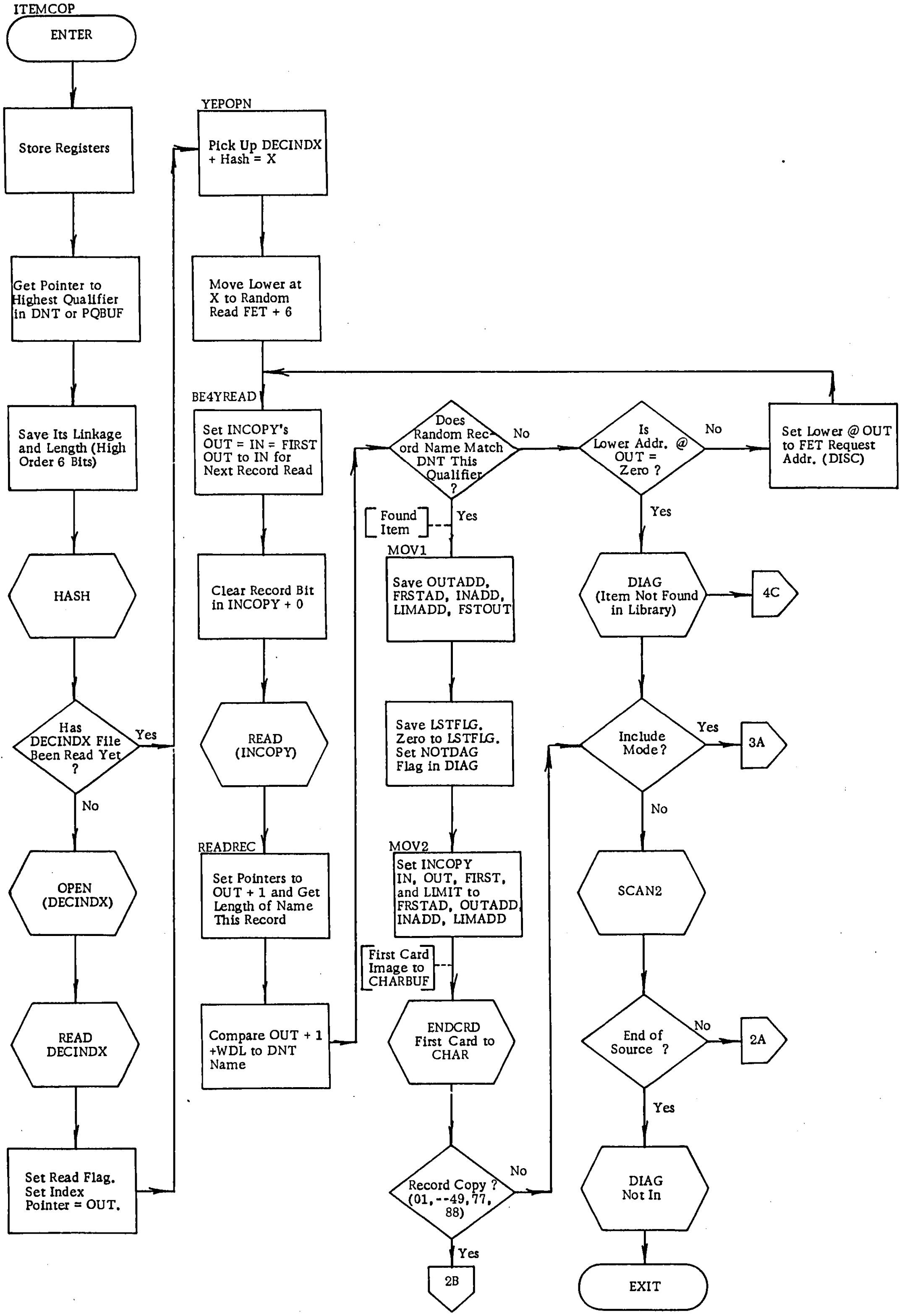


Figure 2-7. ITEMCOP Flowchart (1 of 4)

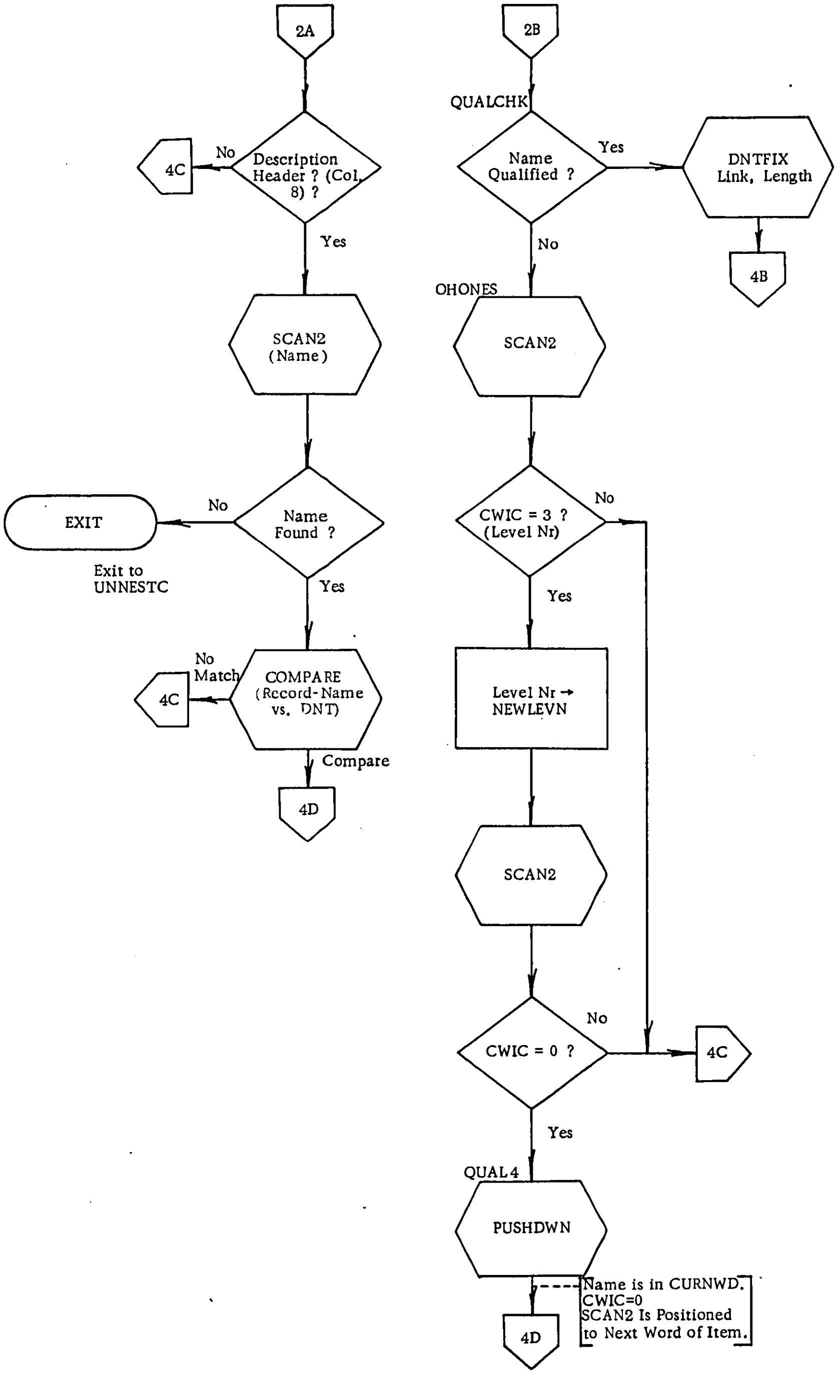


Figure 2-7. ITEMCOP Flowchart (2 of 4)

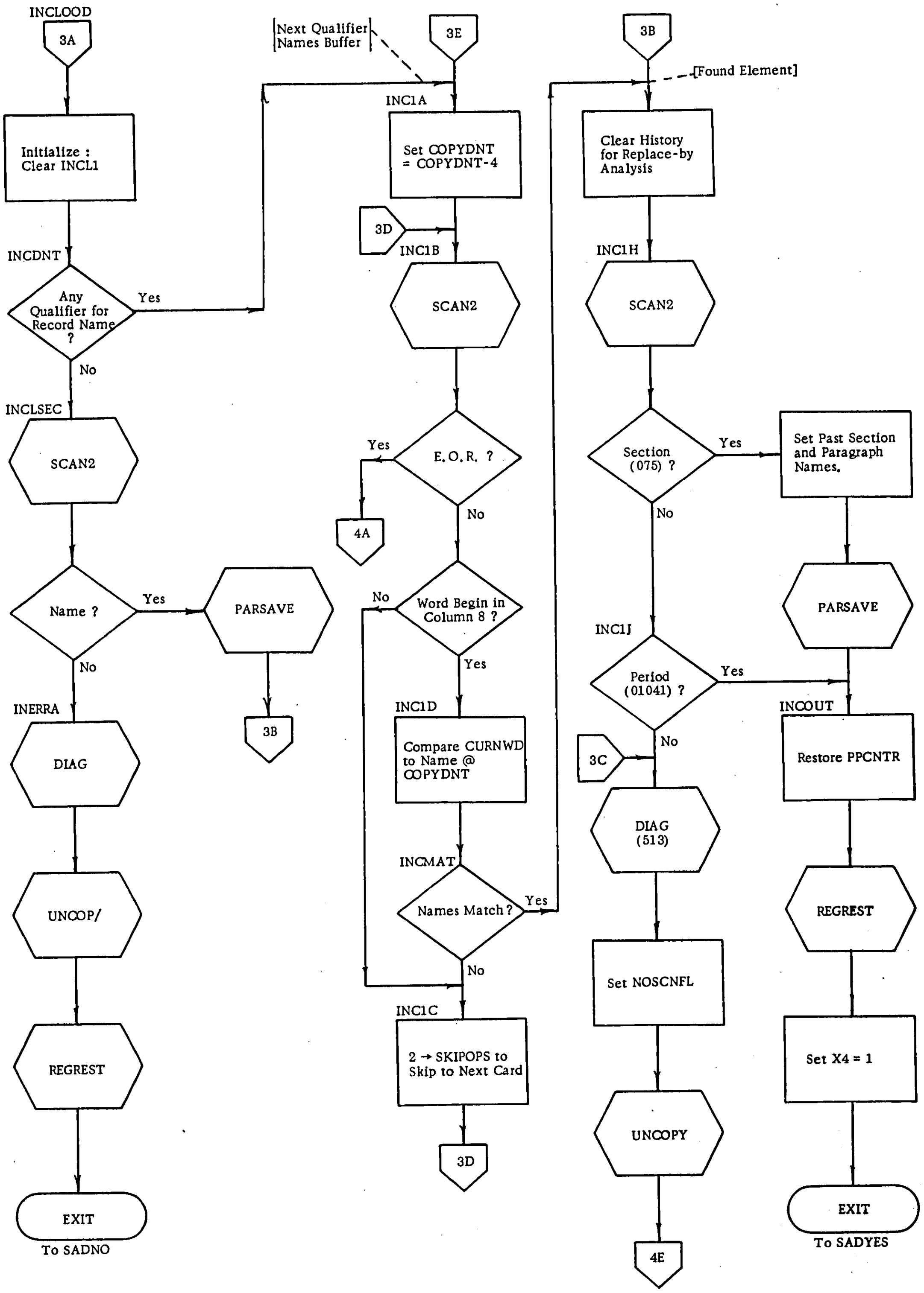


Figure 2-7. ITEMCOP Flowchart (3 of 4)

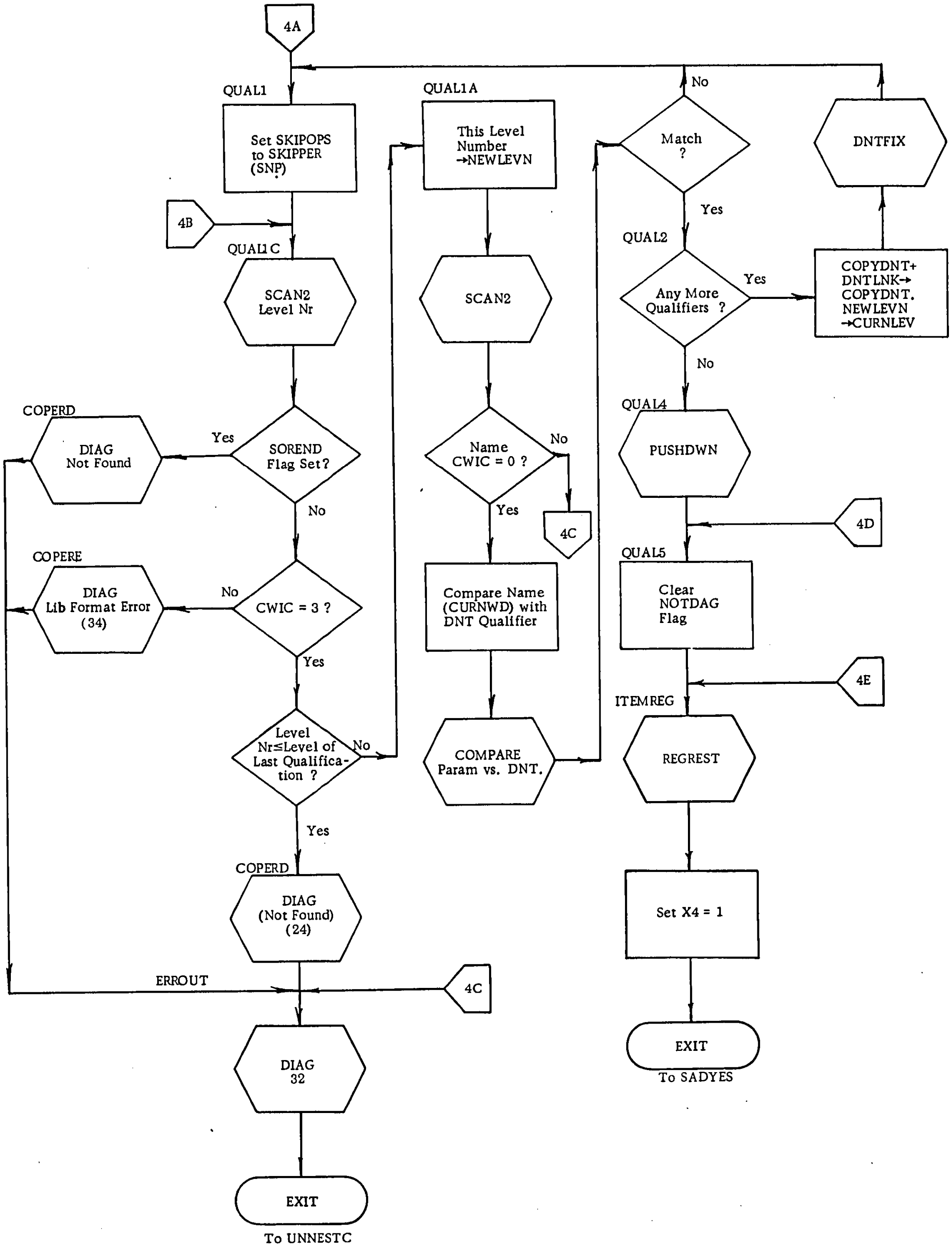


Figure 2-7. ITEMCOP Flowchart (4 of 4)

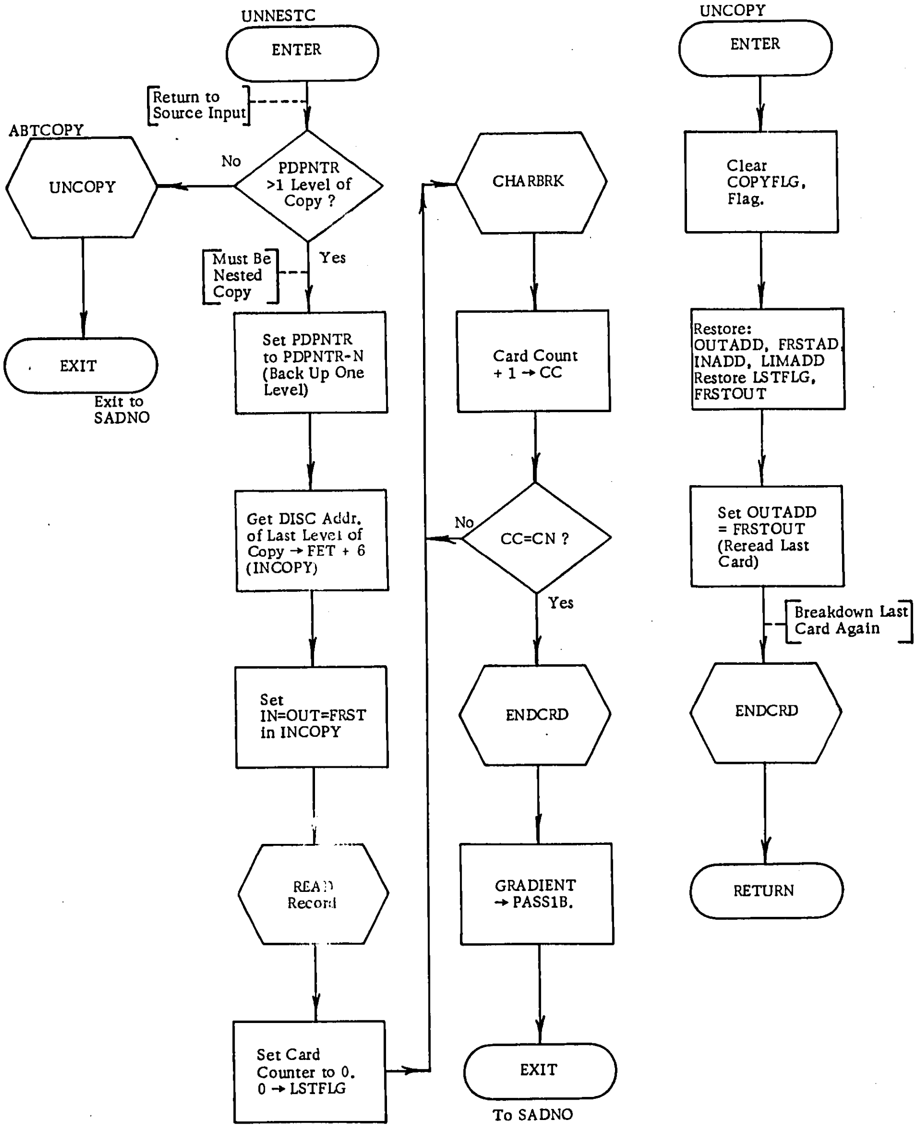


Figure 2-8. UNNESTC Flowchart (1 of 2)

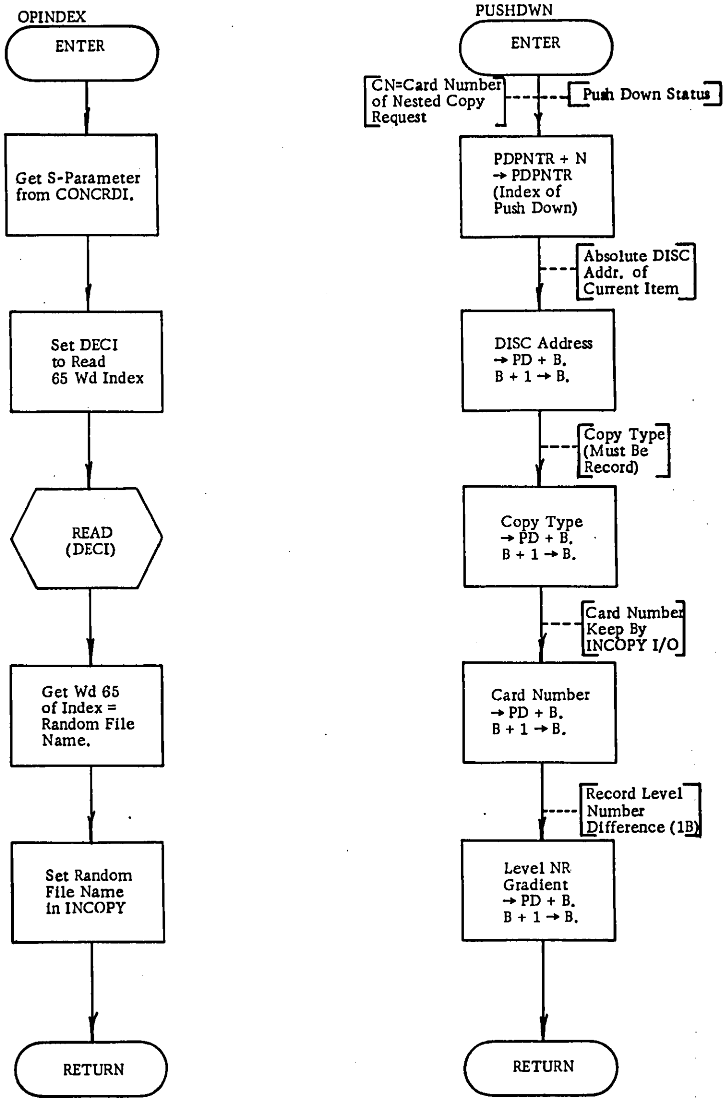


Figure 2-8. UNNESTC Flowchart (2 of 2)

SECTION 3

SECTION 3 - COMPILER COMPONENTS

CONTROL ROUTINE

A general control routine (CONTROL) exists in the compiler to load the overlay passes and to furnish communications between them. The overlays have no other access to code in other overlays or higher level overlays. Initialization routines are required for the syntax overlays and the assembler overlay. The main overlay contains the control routine, several general subroutines, and some subroutines to take snapshot-type dumps, etc.

The general flow is as follows:

1. Calculate initial limits for data table locations
2. Load SCAN overlay
3. Load Pass 1A
4. Go to Pass 1A for initialization
5. Go to SCAN overlay for processing
6. Load Pass 1B
7. Go to Pass 1B for initialization
8. Go to SCAN overlay for processing
9. Load Pass 1C
10. Go to Pass 1C for processing
11. Load Pass 1D
12. Go to Pass 1D for processing
13. Load Pass 1E
14. Go to Pass 1E for initialization
15. Go to SCAN overlay for processing
16. Load Pass 1F
17. Go to Pass 1F for processing
18. Load assembler overlay
19. Go to assembler overlay for initialization
20. Load Pass 1G
21. Go to Pass 1G for processing
22. Load Pass 1H
23. Go to Pass 1H for processing
24. Load Pass 2
25. Go to Pass 2 for processing
26. Clean up

Table values in the CONTROL routine include:

FTBASE	Location of file tables base
DNTBASE	Location of data name table base
RTBASE	Location of report tables base
AFTBASE	Location of accept file tables base
UPTOP	Location of end of "up" tables
DOWNBOT	Location of end of "down" tables

PNTBASE	Location of Procedure Name Table base
PAS2TOP	Location of top of Pass 2
DAGLOC1	Location of diagnostics 000-99
DAGLOC2	Location of diagnostics 100-199
DAGLOC3	Location of diagnostics 200-299
DAGLOC4	Location of diagnostics 300-399
DAGLOC5	Location of diagnostics 400-499
DAGLOC6	Location of diagnostics 500-599
DAGLOC7	Location of diagnostics 600-699
DAGLOC8	Location of diagnostics 700-799
DAGLOC9	Location of diagnostics 800-899
XYZ	Location of syntax tables (current)
SYNSTRT	Location of first entry in syntax table
SYNSUBJ	Location of subjump table
SCANOV	Location of SCAN overlay entrance
LEXOVLA	Location of lexicon list
EAT	Location of 64-word external access table
DAGLOC9	Location of diagnostics 800-899
DGLOC10	Location of diagnostics 900-999
SCNSNAP	Flag* to indicate snapshot dumps in SCAN (debug aid)
SNAPSET	Flag* to indicate snapshot dumps in the current phase

---

*Set by CONTROL upon interpretation of control card parameters.

CONCRDI - CONTROL CARDS INTERPRETER SUBROUTINE

Purpose

CONCRDI is called by control to analyze the COBOL compilation control card and encode its compilation parameters. (See Figure 3-1.)

Calling Sequence

CONCRDI is entered by a return jump.

Routines Called

```
RJ      CONCRDI
DIAG
```

Limitations

This routine is the first executed subroutine of the compiler, therefore, it saves and restores no registers. A parameter, either side of connecting = sign, exceeding seven alphanumeric characters causes a control card error job abortion before control is transferred to control. Parameters other than those specified in the Equipment Reference Specification cause COBOL abortion.

Operation

CONCRDI begins parameter encoding when it reaches a left parenthesis. It identifies the left-hand option mnemonic and then passes over the equal sign, if any, and stores the right-hand parameter (lfn or o) in the corresponding table space in CONTPRM. If the right parameter of an option is not specified, the appropriate default parameter is set in its place. This process continues from left to right until either a period or right parenthesis is scanned, thus terminating the encoding and initiating the setting of necessary listing option flags specified by the L option.

If no parameters are specified CONCRDI returns directly to CONTROL upon sensing a period control card terminator.

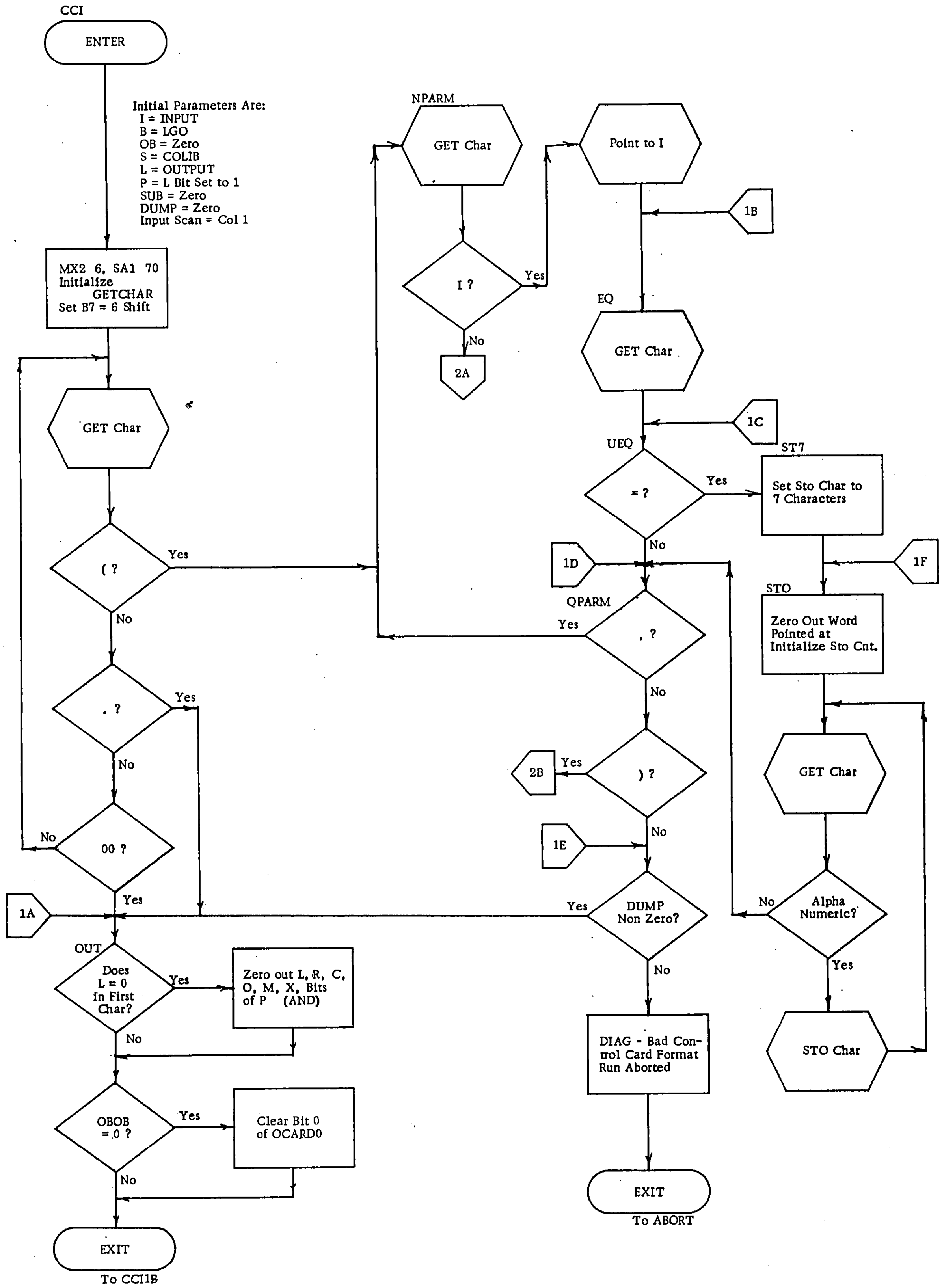


Figure 3-1. CONCRDI Flowchart (1 of 3)

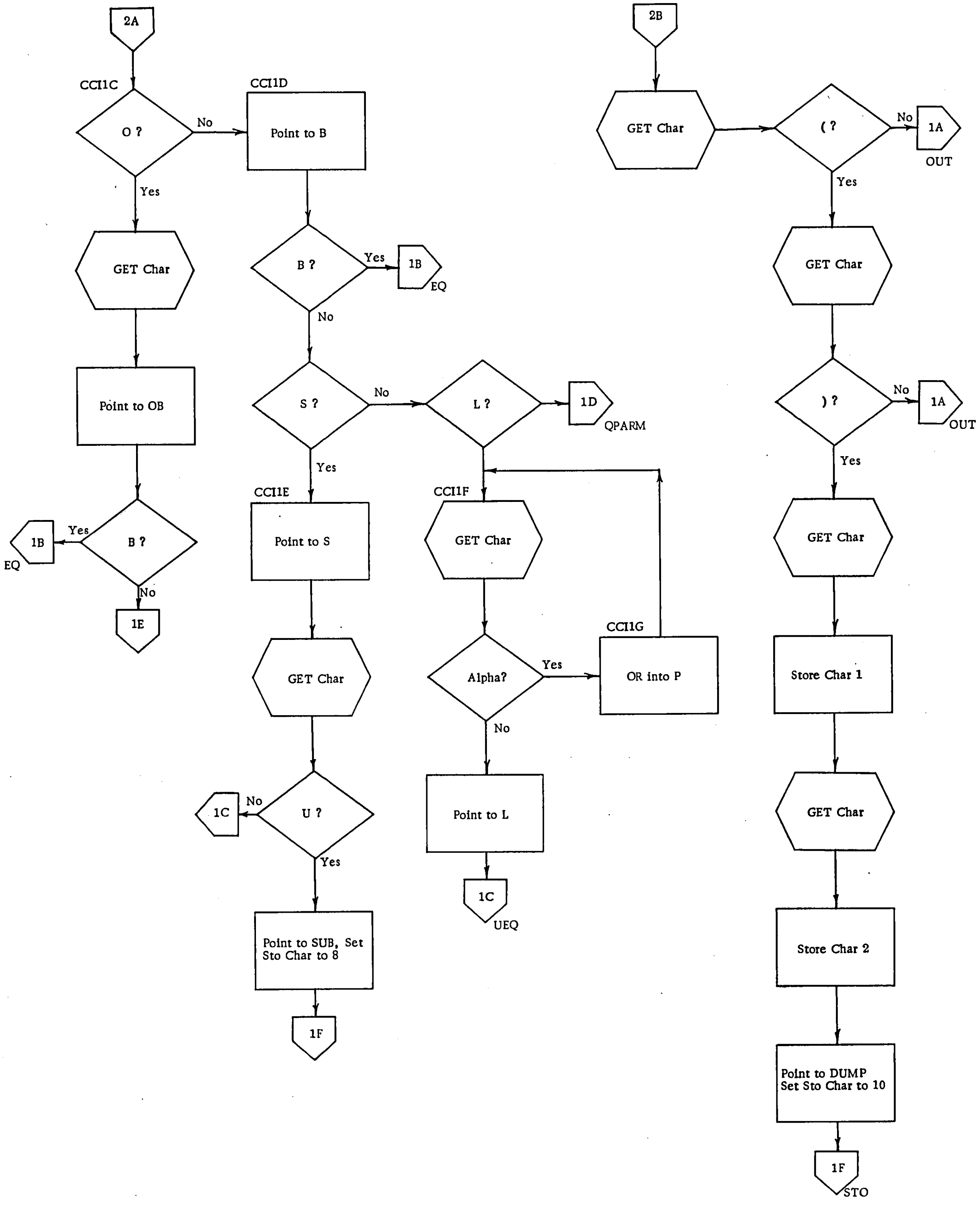


Figure 3-1. CONCRDI Flowchart (2 of 3)

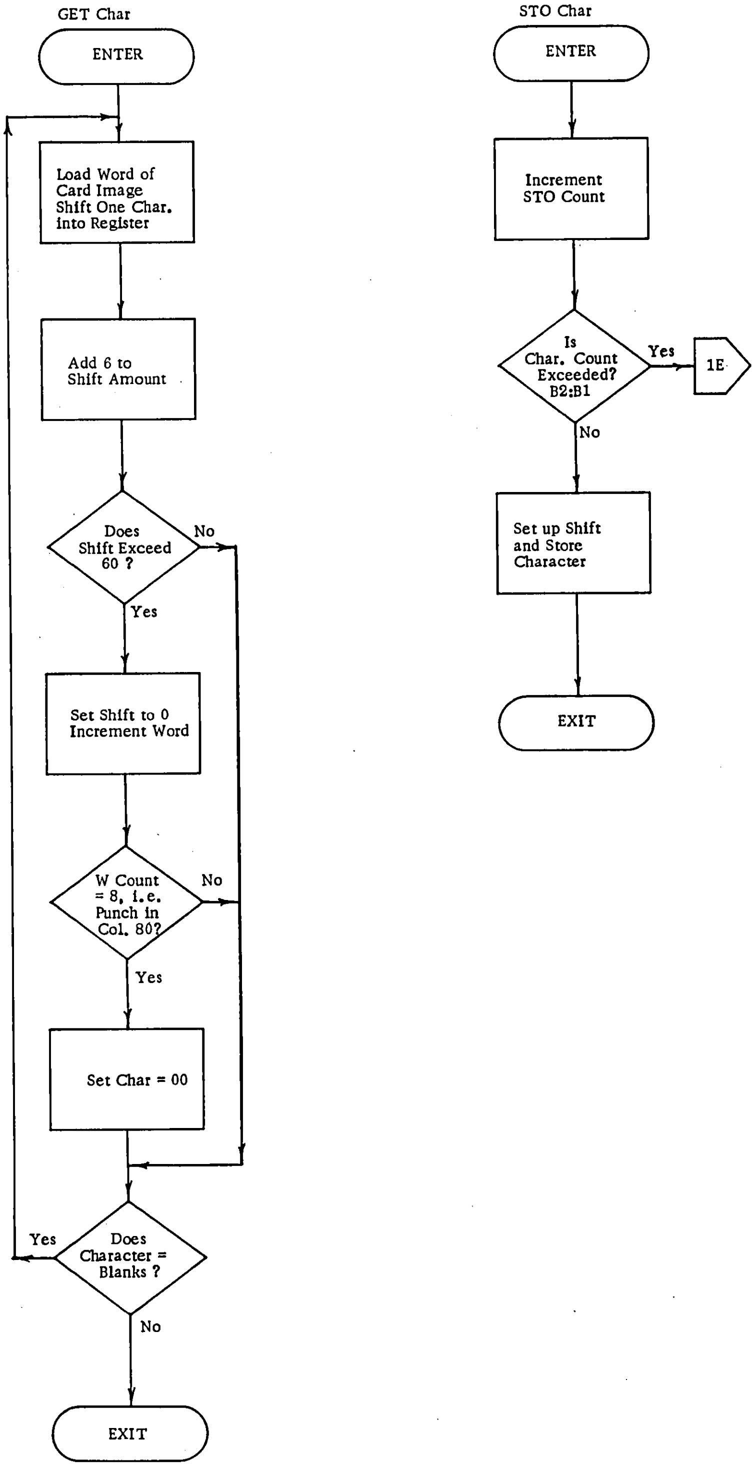


Figure 3-1. CONCRDI Flowchart (3 of 3)

IDENTIFICATION, ENVIRONMENT, AND DATA DIVISION

SYNTAX

The Identification, Environment, and Data Divisions are analyzed by means of a syntax table in theory, not unlike the Procedure Division syntax table. The first three divisions of a COBOL source program must adhere to stringent syntax rules: only certain kinds of statements may appear within any division and section. Therefore, the Identification, Environment, and Data Division syntax table might be thought of almost as a linear search for allowable syntax within a given division or section.

For instance, once a file description (FD) is encountered, the syntax table looks for specific allowable words or clauses that must follow. If an encountered source word is not any of the anticipated words, or is not a section header, then the syntax table issues an appropriate diagnostic, i.e., "Compiler Out of Synchronization" and attempts to get back in synchronization by one of several methods:

1. Skipping past the next period.
2. Searching for a keyword in cc 8.
3. Skipping to the next word.

If an encountered source word is one of the anticipated words, appropriate subroutines are performed to process the word and/or current item.

A brief description of syntax table processing follows, assuming the reader has familiarized himself with the SYNTBLE writeup in Appendix B and under Syntax Analysis in Section 2.

The syntax table items are in alphabetical order, with the first level of analysis beginning with MSTRCON. The syntax levels of processing and structure are shown in Figure 3.2 .

The syntax table begins executing table items at the main level of processing MSTRCON01. This table item "performs" the table item IDNTDIV (second level of processing). IDNTDIV searches for allowable Identification Division COBOL source words. If the word is found, processing returns to the next higher level and the subroutines and/or diagnostics in the yes-action field are performed. If the anticipated word is not found, processing returns and the subroutines and/or diagnostics in the no-action field are performed.

Both the no- and yes-action fields of MSTRCON01 proceed to MSTRCON02 which "performs" the table item ENVDIV, returning to perform the subroutines and/or diagnostics in the no- or yes-action field. Processing continues in a similar manner to the completion of Data Division processing.

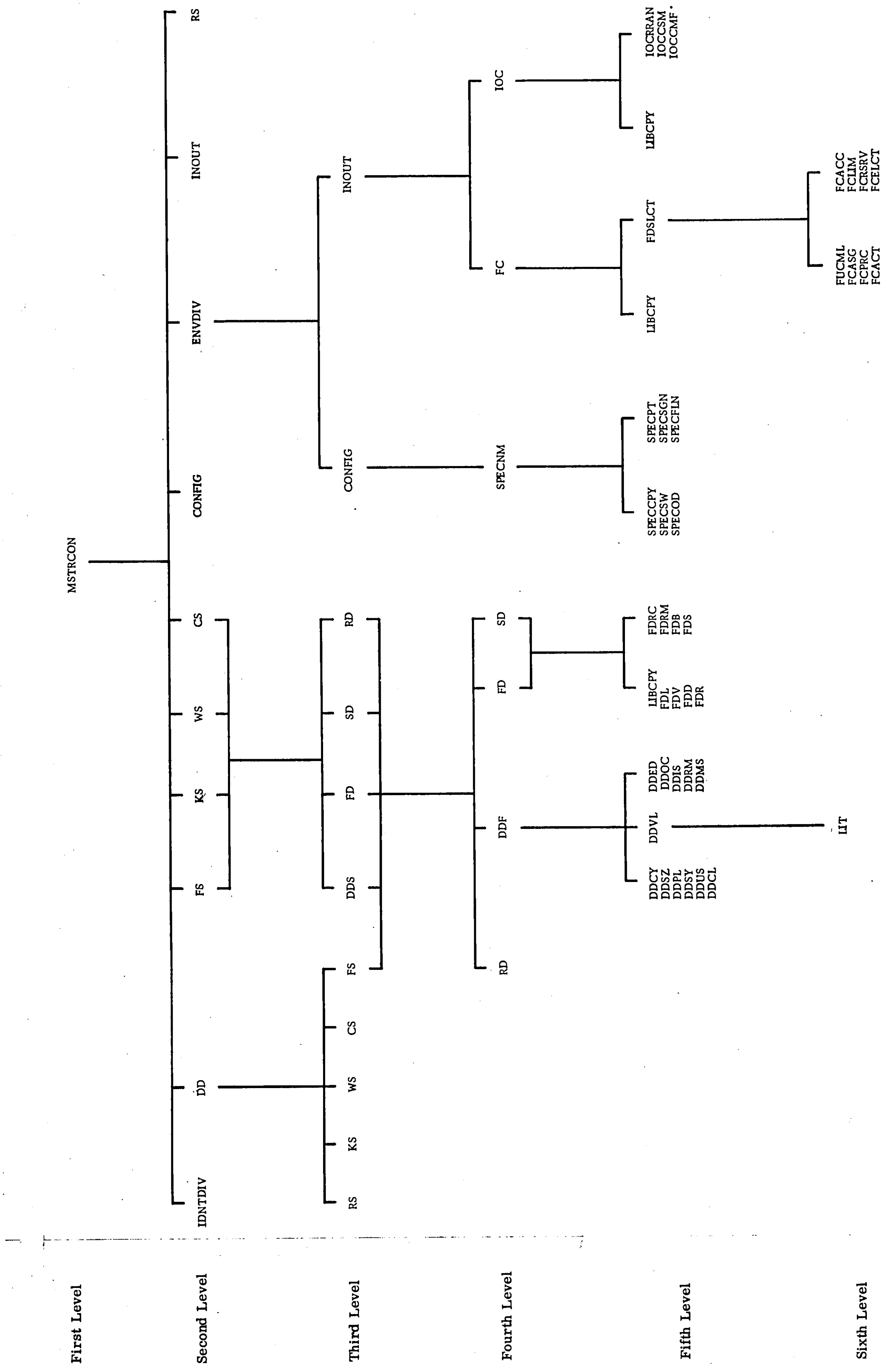
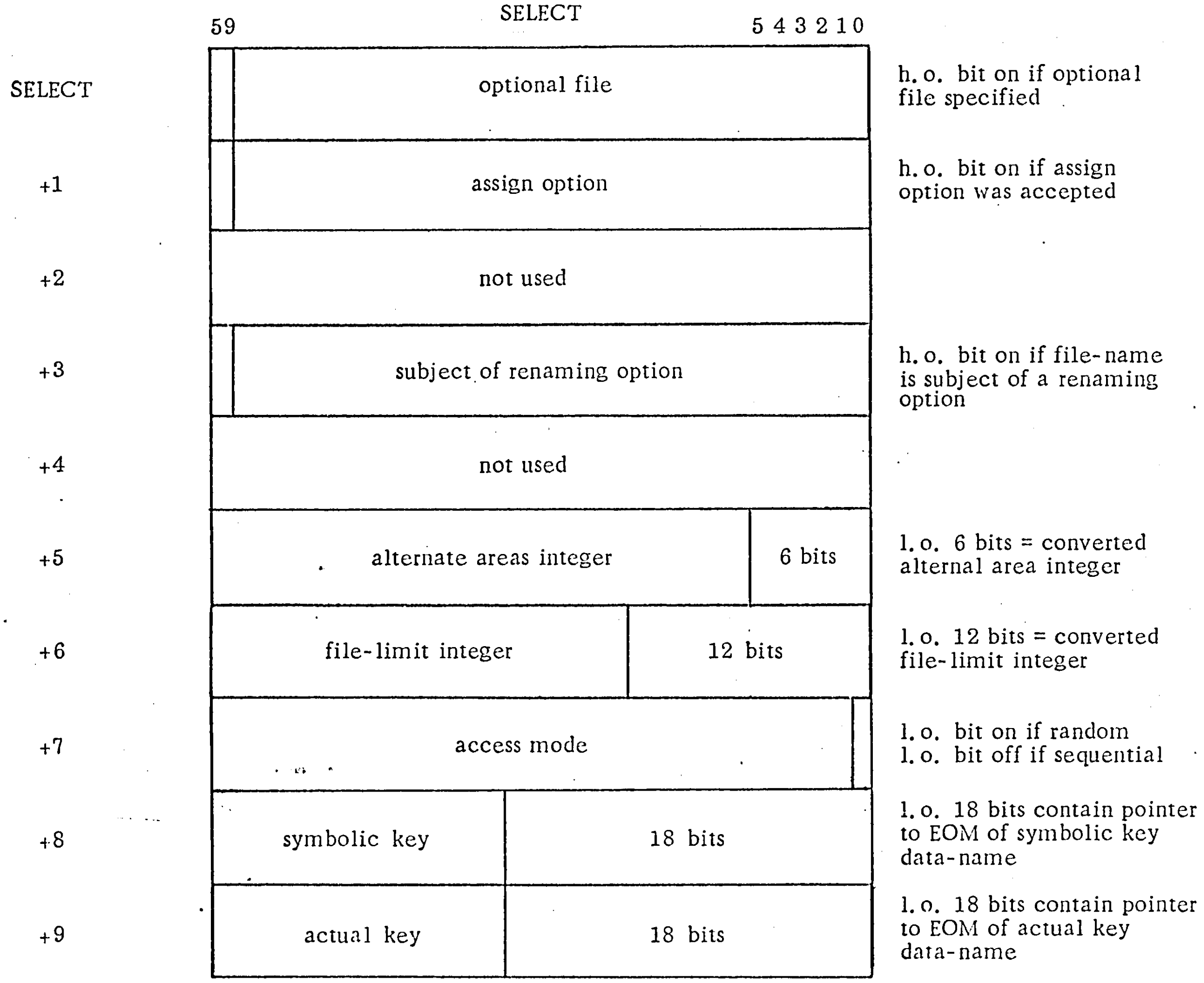


Figure 3-2. Data Division Syntax Levels

PASS 1B AND ELEMENTS

PASS 1B employs several buffers or tables to facilitate the processing of an item. In certain cases, all pertinent information about a single item is stored in these buffers until the item is ready to be processed, after which the buffer is zeroed in preparation for another item. A description and format of the main buffers used in Pass 1B is shown in Figures 3-3, 3-4, and 3-5, and in Table 3-1.

The SELECT buffer is used to store pertinent information about a file-name that is "Selected" in the File-Control paragraph.



SELECT	optional file	h. o. bit on if optional file specified
+1	assign option	h. o. bit on if assign option was accepted
+2	not used	
+3	subject of renaming option	h. o. bit on if file-name is subject of a renaming option
+4	not used	
+5	alternate areas integer | 6 bits	l. o. 6 bits = converted alternal area integer
+6	file-limit integer | 12 bits	l. o. 12 bits = converted file-limit integer
+7	access mode	l. o. bit on if random l. o. bit off if sequential
+8	symbolic key | 18 bits	l. o. 18 bits contain pointer to EOM of symbolic key data-name
+9	actual key | 18 bits	l. o. 18 bits contain pointer to EOM of actual key data-name

Figure 3-3. Select Buffer

The IOCTL buffer is used to store pertinent information about a file-name used in the I/O-Control paragraph.

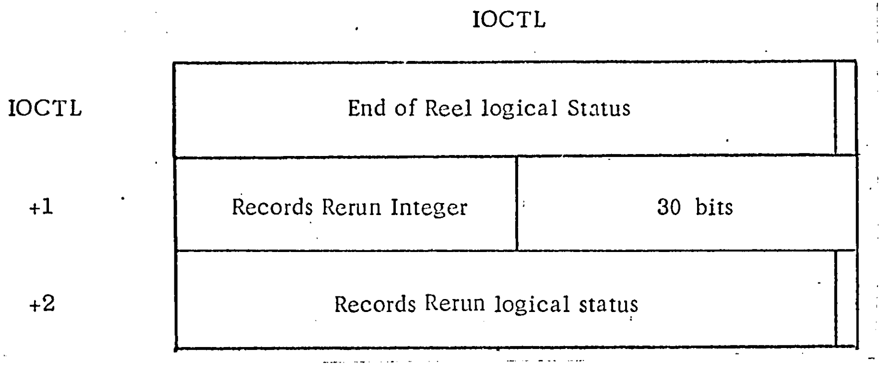


Figure 3-4. IOCTL Buffer

The FDBUFF buffer is used to store pertinent information about a file-name used in a file description in the File Section paragraph.

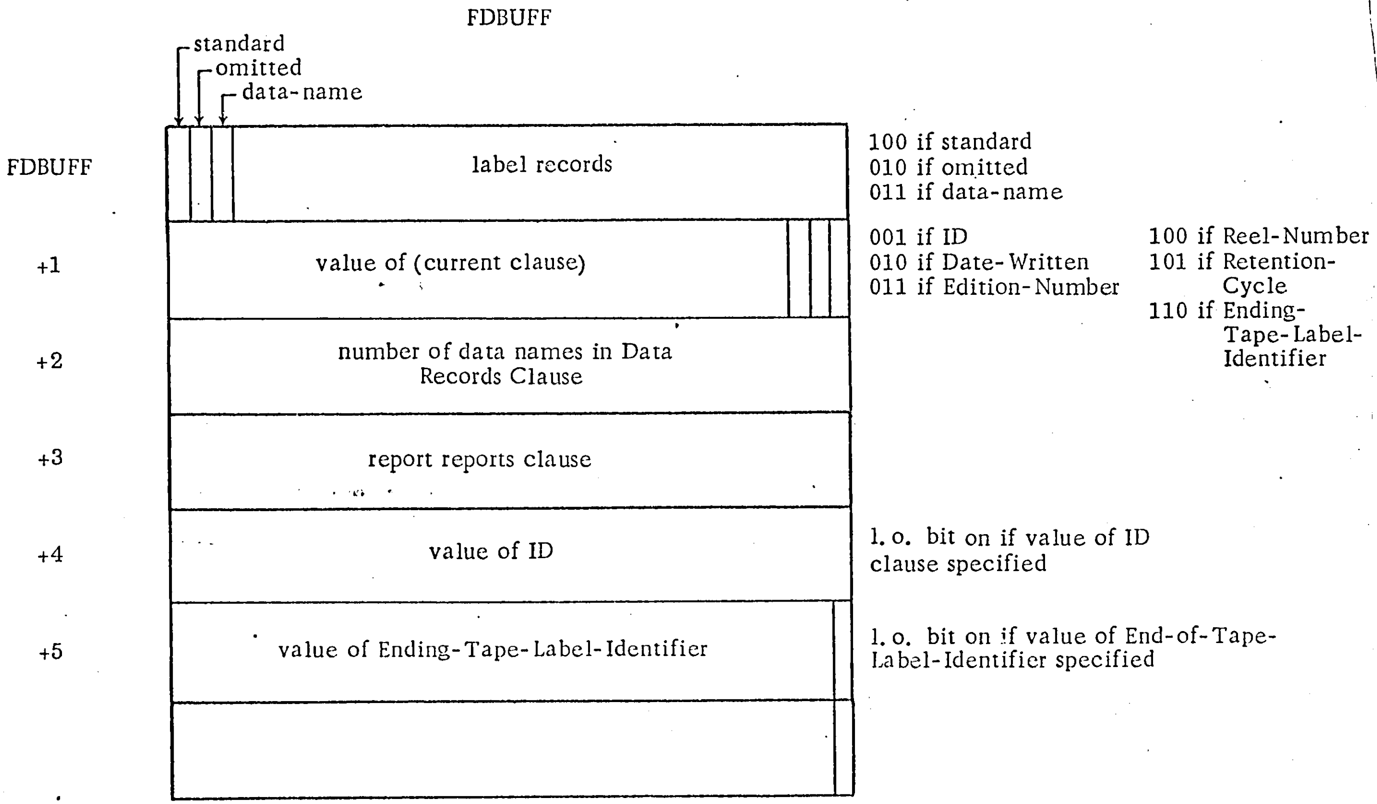


Figure 3-5. FDBUFF Buffer

Table 3-1. SQUASHBU Buffer Table

SQASHBU	Item type	3 bits	T_3 T_2 T_1 (See legend for T and D fields of internal formats)
+1	Data Type	3 bits	D_3 D_2 D_1 (See legend for T and D fields of internal formats)
2	Size	18 bits	Item size
3	occurs ← depending on | integer-1 18 bits | integer-2 18 bits | Occurs Depending On		integer-1 subject of 'TO' option=middle 18 bits integer-2 object of 'TO' option=l.o. 18 bits
4	Redefines Flag		low-order bit=1 if redefines specified
5	Point Location | A V | L R	5 bits count	l.o. 5 bits = point location count A (1 if actual) V (0 is assumed) L (0 if left) R (1 if right)
6	Justified Flag		l.o. bit=1 if justified right specified
7	Synchronized Flag | S 2 | S 1		S (0 if not synchronized) S (0 if synchronized left) 1 (1 if synchronized) 2 (1 if synchronized right)
8	BWZ		l.o. bit=1 if blank when zero specified
9	Editing FOS → | CP | ZS		ZS = 1 if zero suppress CP = 1 if check protect FDS = 1 if float dollar sign
10	Picture Flag	3 bits	l.o. bit on if PICTURE (before PICTUR processing) mural length (after PICTUR processing)
11	Value Flag	6 bits	h.o. bit on if value l.o. 6 bits = number of words of value
12	Insertion Character and Leaving	6 bits	l.o. 6 bits = edit count (before PICTUR processing) l.o. 6 bits = number of insertion characters (after PICTUR processing)
13	File Section/Report Number	6 bits	file or report number 0 - 58 (set external access table)
14	Level Number	6 bits	l.o. 6 bits = converted level number 66 = 628 77 = 63_8 88 = 64_8
15	Signed		h.o. bit=1 if signed item
16	Class	3 bits	C_3 C_2 C_1 (See legend for entry fields of internal formats)
17	Usage	3 bits	U_3 U_2 U_1 (See legend for entry fields of internal formats)
18	Link to Dominant	18 bits	Pointer to the dominant item
19	Link to Same Name	18 bits	Pointer to next item with the same name
20	Line Number	15 bits	Converted source line number
21	Flags	6 bits	G_6 G_5 G_4 G_3 G_2 G_1 (See legend for entry fields of internal formats)
22	Line Clause		
23	Next Group		
24	Report FLAGS		
25	RGD TYPE FLAGS		
26	Report Options		
27	Jump Index to Decl.		
28	Column		
29	Link to Reset; CH Link in RD		
30	3 bits PASS1C | Item Length for PASS1C and PASS1E	3 bits PASS1E	l.o. 3 bits = length for PASS1E bits 36-38 = length for PASS1C
31			
32	Discard Flag		l.o. bit=1 if item is to be discarded
33	Header		non-zero if item is a header

The SQASHBU buffer is used to process the current or previous item. Actually, a double buffering technique is employed, with two SQASHBU buffer areas defined. SQASHBU indexed by B3 references the current item. SQASHBU indexed by B4 references the previous item. (See Table 3-1.)

Pass 1B is comprised of subroutines that handle the Identification, Environment, and Data Divisions. Tables 3-2, 3-3, 3-4, and 3-5 list all of Pass 1B's subroutines, with a brief description of each.

RG Checks Done in Pass 1B

Several special checks and actions are made in Pass 1B on items in the Report Section. Diagnostics are given in appropriate situations. Further checking of the items in the Report Section is done in Pass 1D and Pass 1H. Many of the normal Data Division checks are made on report items in addition to the checks discussed below:

XRDCK 1) Check line position from PAGE clause in RD item.

- a) Heading position (H) must be ≥ 1. If no H, set to 1.
- b) First detail position (FD) must be ≥ H. If no FD, set to H.
- c) Last detail position (LD) must be ≥ FD. If no LD, set to F.
- d) Footing position (F) must be ≥ LD. If no F, set to LD. If no F or LD, set both to page limit.
- e) Page limit must be ≥ F.

SYNT61 2) The COPY clause can only appear on an RD or 01 item. The object of the copy can be in the source program or the library for an 01, but can only be in the library for the RD.

XRECK 3) Every 01 level item in the Report Section must have a TYPE clause and be subordinate to an RD which was listed in the REPORTS clause of an FD.

XRECK 4) The NEXT GROUP clause can only appear on an 01 level item.

XRECK 5) An 01 item with a type designation of DETAIL or DE must have a data name following the level number.

DCKPRE 6) The SELECTED option of the SOURCE clause can appear at the group level only.

Table 3-2. ID and Environment Division Subroutines Performed from SYNTBLE (1 of 3)

Name	Description	Flowchart	Diagnostics Issued	Internal Subroutines Performed	Pertinent Compiler Internal Table Formats
XCN	Insures Configuration Section paragraph succeeds Environment Division header and precedes input/output section paragraph.		123 263		
XED	Insures Environment Division header succeeds Identification Division header and precedes Data Division header.		130		
XFC01	Saves object of select file name in 'FILESV'.				
XFC02	Turn on optional flag in select buffer.				
XFC03	Saves object of renaming filename in 'RENSV'; turns on renaming flag in select buffer.				
XFC06	Stores symbolic key data-name in End-of-Memory (EOM) with pointer to EOM in select buffer.			SETEOM	
XFC07	Stores converted file-limit integer in select buffer.		114		
XFC08	Stores converted alternate area integer in select buffer.		115		
XFC09	Stores actual key data-name in End-of-Memory with pointer to EOM in select buffer.			SETEOM	
XFC10	Turns on random-access flag in select buffer.				
XFC17	Processes the select clause: 1. Object of select data name is in 'FILESV'. 2. Object of renaming data name is in 'RENSV'. 3. Pertinent information in select buffer. 4. Object of assign implementor name is in 'ASGNSVE'.	Figure 3-6	124 133	ZROSLT FIND SETDNT PUTSLCT SETFET	
XFC18	Saves object of assign 7 character implementor name in 'ASGNSVE' after insuring that it has not appeared before as the object of an assign; turns on assign flag in select buffer.		122 110 138		
XFCCK	Insures that all file-names used as object of a renaming clause are selected.		126		
XID	Stores first evel characters of Program ID name in CONTROL element.				

Table 3-2. ID and Environment Division Subroutines Performed from SYNTBLE (2 of 3)

Name	Description	Flowchart	Diagnostics Issued	Internal Subroutines Performed	Pertinent Compiler Internal Table Formats
XID1	Insures Identification Division header precedes any other division headers.		130		
XID2	Stores assigned name 'IDCOBOL' in CONTROL element.				
XINIT	Initializes index to beginning of the Data Name Table (DNT), zeros out core from beginning to end of DNT, initializes index to beginning of End-of-Memory table (EOM), initializes reserved registers B3 and B4.				
XIOC01	Turns on end-of-reel bit in IOCTL buffer.				
XIOC02	Stores converted rerun integer into IOCTL buffer.		118		
XIOC03	Turns on records-rerun flag in IOCTL buffer.				
XIOC05	Stores rerun information from IOCTL buffer into the DNT.				File Table
XIOC06	Saves pointer to FD (see file table of internal formats) in IOCSVE.				
XIOC07	Initializes appropriate cells for multiple file processing.				
XIOC08	Increments positions number (POSCNT) for multiple file clause.		121		File Table
XIOC09	Moves specified position number into the File Table.		121		File Table
XIOC10	Turns on multiple file position flag (MFPOSFG).				
XIOC17	Saves the file number of the first file-name specified in multiple file clause.				File Table
XIOC18	Stores file number saved by XIOC17 into current file-name's file table.				File Table
XIOC19	Zeros out SAMEMSK.				
XIOCK	Saves the high-order 3 characters of the implementor-name to which this file-name was assigned. Store high-order 3 characters as multifile-name.				File Table

Table 3-2. ID and Environment Division Subroutines Performed from SYNTBLE (3 of 3)

Name	Description	Flowchart	Diagnostics Issued	Internal Subroutines Performed	Pertinent Compiler Internal Table Formats
XIOCK1	Insures that the high-order 3 characters of the implementor-name is the same as the other file-name in multiple file clause; Stores 3 characters as multifile-name.		135		
XIOCK3	Turns on flag (IOCTLON) for FILNAM routine.				
XIOCK4	Turns off IOCTLON flag.				
XIOS	Insures that the Input/Output Section follows the Configuration Section.		263		
XSPEC1	Insures switch number is between 1 and 6, converts from display to binary and stores switch number.		106		
XSPEC2	Puts appropriate switch number ON/OFF status item type in the DNT.			ENVSQ	
XSPEC3	Turns on-status bit for switch number to ON position.				
XSPEC5	Replaces the '$' with specified literal in PICTUR element.		106		
XSPEC6	Turns on flag in CONTROL element signifying comma replaces decimal point, replaces the '.' with ',' in PICTUR element.				
XSPEC7	Sets up 'non-numeric literal is mnemonic name' item type option of special-names paragraph.		106		
XSPEC8	Puts mnemonic name and literal item type in DNT.			ENVSQ	
XSPFLN1	Saves first seven characters of implementor name; sets system file type.		361		
XSPFLN2	Turns on flag signifying that the word 'input' was used as implementor-name in special-names paragraph.				
XSPFLN3	Turns on flag signifying that the word 'output' was used as implementor name in special names paragraph.				
XSPFLN4	Puts the mnemonic name used in special names paragraph in the DNT.			ENVSQ	
XSPFLN5	Initializes appropriate flags to zero.				
XSPINIT	Initializes to zero all flags used in special-names paragraph processing.				

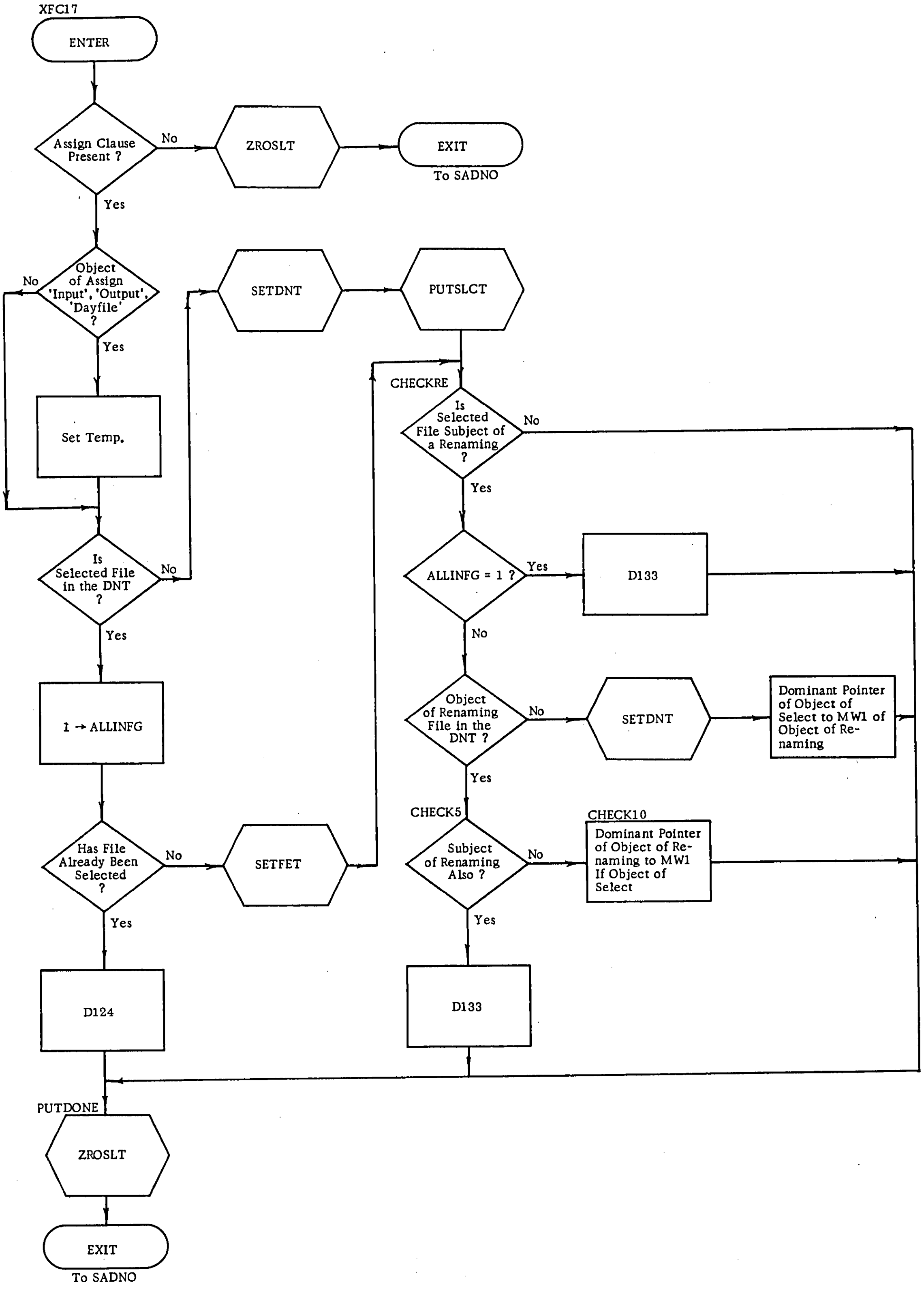


Figure 3-6. XFC17 Flowchart

Table 3-3. Data Division Subroutines Performed from SYNTBLE (1 of 7)

Name	Description	Flowchart	Diagnostics Issued	Internal Subroutines Performed	Pertinent Compiler Internal Table Formats
FDCLNUP	Compares the number of 01 record descriptions under a particular FD to insure that the number is the same as that stated in the Data Records clause.		204		
FILNAM	This subroutine has two return points, one to SADNO in the event that the specified file-name is not in the DNT, or to SADYES in the event that a given file-name is in the DNT.				
SETFD01	If this is an 01 under the current FD, increments the counter of the number of 01 record descriptions of current FD.				
XABORT	Turns on ABORT flag signifying to subsequent passes that the fatal error flag has been set.				
XCLNUP	Processes the last item in the Data Division--that is, the last item before the Procedure Division header--and does appropriate initialization for subsequent passes.		123	FINI88 ZROPREV SWITCH SAVE RESTORE DCKPRE PUTPAT HASH DCOMON	
XCPYCK	Detects nested copy from library clause and exits.				
XCPYLEV	Performs checks on the current library copy item.		206		
XCPYOFF	Exits from library copy mode by unconditional transfer to UNNESTC in ITEMCOP element.				
XCS	Insures Constant Section follows Working-Storage section (if present) and precedes the Report Section (if present).		123 263	FINI88 ZROPREV SWITCH SAVE RESTORE DCKPRE	
XDD	Insures that the Identification Division and Environment Division have preceded Data Division.		130		

Table 3-3. Data Division Subroutines Performed from SYNTBLE (2 of 7)

Name	Description	Flowchart	Diagnostics Issued	Internal Subroutines Performed	Pertinent Compiler Internal Table Formats
XDD1	Does initial checks on the current level number and stores it in the current buffer (SQASHBU + B3)	Figure 3-7	351, 293, 353, 362, 294, 295, 352, 354, 206, 355	SAVE XDTB RESTORE ZROCURR SWITCH DCKPRE	
XDD2	Has 2 return exits, one to SADNO if current integer is not acceptable as a level number; and one to SADYES if current integer is acceptable.		206	XDTB	
XDD4	Saves the current data-name for subsequent squashing into the DNT.				
XDD5	Insures current level number is not a 77 or 88 because FILLER cannot be assigned to these items.		319 289		
XDD6	Saves the word FILLER for subsequent squashing into the DNT.				
XDDCK	Analyzes the current item to diagnose inconsistencies, illegal combination of clauses, etc.	Figure 3-8	132, 146, 202, 295, 298, 299, 300, 301, 302, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 317, 329	SAVE MOVEALL RESTORE	
XDDCL1	Turns on flag signifying Class alphabetic.				
XDDCL2	Turns on flag signifying Class numeric.				
XDDCL3	Turns on flag signifying Class alphanumeric.				
XDDCY1	Sets flag signifying current item is a copy item type. Sets appropriate flags and stores the object of the copy clause in the END OF MEMORY (EOM).			SETEOM	EAT
XDDCY2	Turns off appropriate flags signifying source copies because current item is a library copy.				EAT
XDDCY3	Stores copy qualifier in the END OF MEMORY.			QUALEOM	

Table 3-3. Data Division Subroutines Performed from SYNTBLE (3 of 7)

Name	Description	Flowchart	Diagnostics Issued	Internal Subroutines Performed	Pertinent Compiler Internal Table Formats
XDDCY4	Sets appropriate library copy flap and stores object of the copy clause in the EOM.			SETEOM	
XDDED1	Turns on flag signifying Zero Suppress clause.		364		
XDDED2	Turns on flag signifying Check Protect clause.		389		
XDDED3	Turns on flag signifying Float Dollar sign.		389		
XDDED4	Processes the integer used in the Leaving clause.		251	XDTB	
XDDED5	Turns on flag signifying Blank When Zero clause.				
XDDJS1	Turns on flag signifying Justified clause.				
XDDOC1	Processes the integer (minimum) of the Occurs clause.		253 235	XDTB	
XDDOC2	Processes the integer (maximum-object of THRU) of the Occurs clause.		254 235	XDTB	
XDDOC3	Turns on flag signifying Occurs Depending On option and stores the depending on data-name in the END OF MEMORY.		139	SETEOM	
XDDOC4	Stores the qualifying name of the Occurs Depending On option in the END OF MEMORY.			QUALEOM	
XDDPC1	Sets picture flag for intermediary processing of the COBOL source picture.				
XDDPL1	Turns on flag signifying Point Location clause.				
XDDPL2	Processes the integer used in the Point Location clause		239 238	XDTB	
XDDRF1	Turns on flag signifying Redefines clause and stores the object of the redefine data-name in the END OF MEMORY.			SETEOM	
XDDRF2	Stores the qualifying name of the Redefines clause in the END OF MEMORY.			QUALEOM	

Table 3-3. Data Division Subroutines Performed from SYNTBLE (4 of 7)

Name	Description	Flowchart	Diagnostics Issued	Internal Subroutines Performed	Pertinent Compiler Internal Table Formats
XDDRM1	Has two return points, one to SADNO if the current item is not a 66; or to SADYES in the event that the current item is a 66.		330		
XDDRM2	Puts the data-name used as the object of the Renames clause in the END OF MEMORY.			SETEOM	
XDDRM3	Stores the object of the THRU option of the Renames clause in the END OF MEMORY.			SETEOM	
XDDS	Turns on flag signifying Signed clause.				
XDDSY1	Turns on flag signifying Synchronized clause.				
XDDSY2	Turns on flag signifying Synchronized Right option.				
XDDSZ1	Processes the integer used in the Size clause.		356	XDTB	
XDDUS1	Turns on flag signifying Usage display.				
XDDUS2	Turns on flag signifying Usage computational.				
XDDUS3	Turns on flag signifying Usage computational one.				
XDDUS4	Turns on flag signifying Usage computational -n.				
XDDVL1	Processes the necessary information for initial value of the current item.			SQ88DDL	
XDDVL2	Initializes temporary cell to zero.				
XDDVL3	Turns on flag signifying that the THRU option has been specified in the Value clause and insures this item is an 88.		212		
XDISCRD	Turns on flag signifying that the current item is to be discarded.				
XFD01	Insures file-name is not subject of a renaming option; turns on fill or sort description in External Access Table (EAT).		141		EAT
XFD02	Turns on a flag signifying standard label record under current FD.				File Table

Table 3-3. Data Division Subroutines Performed from SYNTBLE (5 of 7)

Name	Description	Flowchart	Diagnostics Issued	Internal Subroutines Performed	Pertinent Compiler Internal Table Formats
XFD03	Turns on a flag signifying label records are omitted under current FD.				
XFD04	Increments the counter of the number of data-names in the Data Records clause.				
XFD05	Processes the current Value Of clause under the FD, storing the data name in the END OF MEMORY (EOM) with the EOM pointer in the appropriate FET position.			SETEOM	File Table
XFD06	Processes the current Value Of literal of the current FD storing it in the appropriate place in FET.		220		File Table
XFD07	Turns on flag signifying Value Of ID under current FD.				
XFD08	Turns on flag signifying Value Of Date-Written under current FD.				
XFD09	Turns on flag signifying Value Of Edition-Number under current FD.				
XFD10	Turns on flag signifying Value Of Reel-Number under current FD.				
XFD11	Turns on flag signifying Value Of Retention-Cycle under current FD.				
XFD12	Turns on flag signifying nonstandard label under current FD.				File Table
XFD14	Stores the integer (minimum) of the Record Contains clause in FET.		223	XDTB	File Table
XFD15	Turns on flag in FET signifying that record mark has been specified as the object of a Depending On option in the Record Contains clause.				File Table
XFD16	Stores the data-name which is specified as the object of the Depending On option in the Record Contains clause in the END OF MEMORY with the EOM pointer in the appropriate FET position.			SETEOM	File Table
XFD17	Stores integer (maximum) of the Records Contains clause in FET.		223	XDTB	File Table
XFD18	Turns on flag signifying Sort Description in FET.				File Table
XFD19	Turns on flag signifying binary recording mask.				File Table

Table 3-3. Data Division Subroutines Performed from SYNTBLE (6 of 7)

Name	Description	Flowchart	Diagnostics Issued	Internal Subroutines Performed	Pertinent Compiler Internal Table Formats
XFD20	Turns on flag signifying low density.				File Table
XFD21	Turns on flag signifying hyper density.				File Table
XFD22	Stores the integer (maximum) of the Block Contains clause in FET.		227	XDTB	File Table
XFD23	Turns on flag signifying records as specified in the Block Contains clause.				File Table
XFD24	Turns on flag signifying decimal recording mode.				File Table
XFD25	Stores data-name (object of Label Records clause) in LRECDN for subsequent processing.				
XFD26	Turns on VALUOF flag so subroutine XLIT8 processes a literal as the object of Value Of clause.				
XFD27	Turns off VALUOF flag.				
XFD28	Turns on flag signifying Value Of Ending-Tape-Label-Identifier under current FD.		220		
XFDCK	Insures that a Data Records or Report Records clause and Label Records clause has been specified for the current FD.		333, 125, 216, 219, 140		
XFDSET	Initializes FDBUFF.			SETFD	
XFS	Insures File Section paragraph precedes the other Data Division section paragraphs and succeeds the Data Division header.		123 263		
XFSCK	Insures that all selected files which are not the subject of Renaming option have a corresponding FD, SD, or RD.		142		
XKS	Insures the Common-Storage section follows the File Section (if present) and precedes the other section paragraphs of the Data Division.		123 263	FINI88 ZROPREV SWITCH SAVE RESTORE DCKPRE	

Table 3-3. Data Division Subroutines Performed from SYNTBLE (7 of 7)

Name	Description	Flowchart	Diagnostics Issued	Internal Subroutines Performed	Pertinent Compiler Internal Table Formats
XLIT1	Turns on flag signifying initial value is ALL literal.				
XLIT2	Analyzes the initial value literal and turns on appropriate flags signifying Point Location, size of the point location, and sign of the value.	Figure 3-9	220 238 255	STRVALU	
XLIT3	Turns on flag signifying value is Zero.				
XLIT4	Turns on flag signifying value is Space.				
XLIT5	Turns on flag signifying value is Quote.				
XLIT6	Turns on flag signifying value is Low-Value.				
XLIT7	Turns on flag signifying value is High-Value.				
XLIT8	Analyzes the non-numeric initial value literal.		220, 255	STRVALU	
XLIT9	Turns on flag signifying Value is Record-Mark.				
XRMCK	Insures the presence of a valid Recording Mode clause.		210		
XSDCK	Insures the presence of a Data Records clause in the current SD description.		145		
XSDSET	Zeros out FDBUFF.			SETFD	
XWS	Insures the Working-Storage section follows the Common-Storage section (if present) and precedes the Report Section (if present).		123 263	FINI88 ZROPREV SWITCH SAVE RESTORE DCKPRE	
XVLSQ88	If the current item is an 88 item, squash the HL, data name, and CN1, and CN2 into the DNT.			SQASH88	

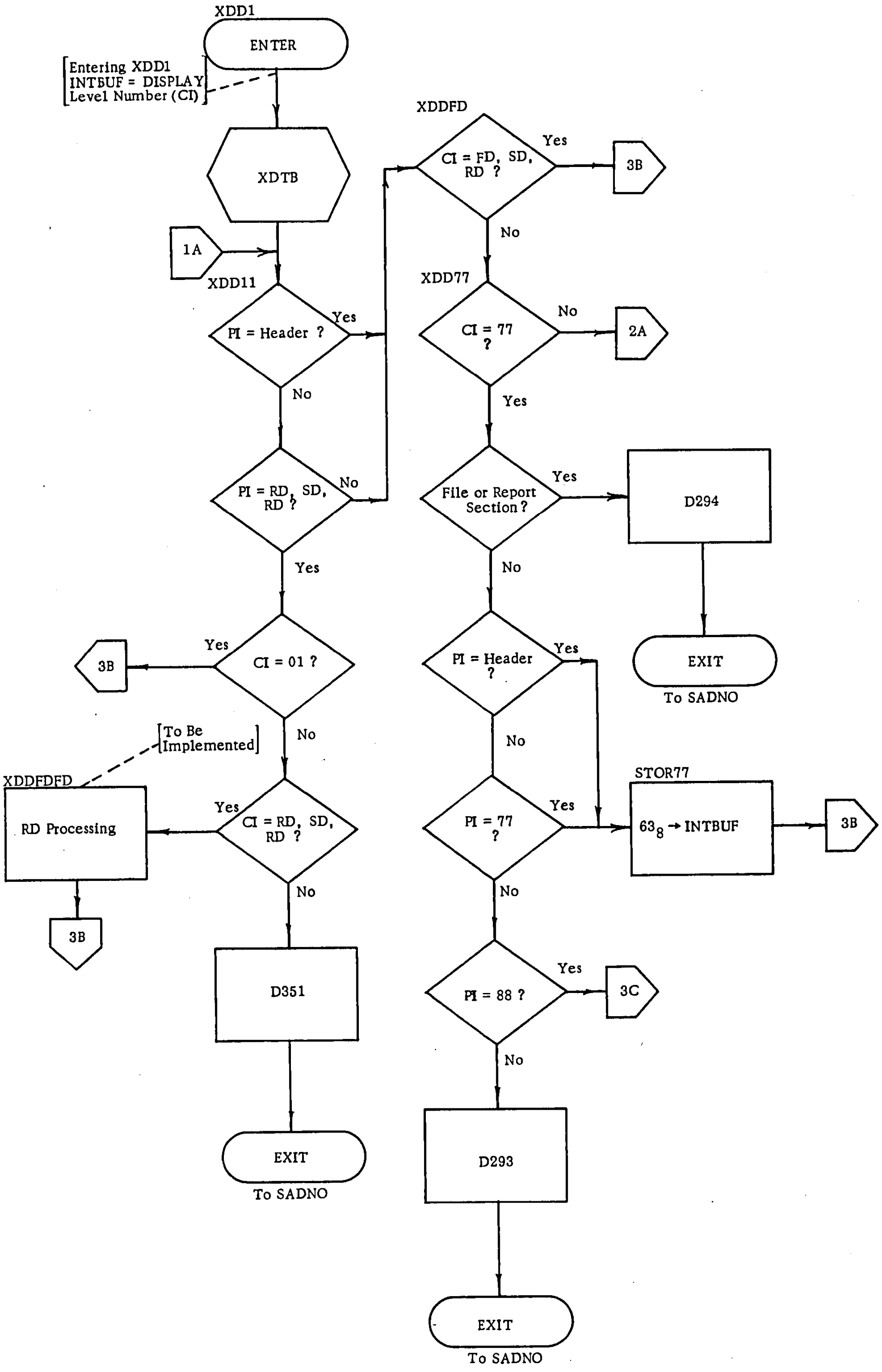


Figure 3-7. XDD1 Flowchart (1 of 3)

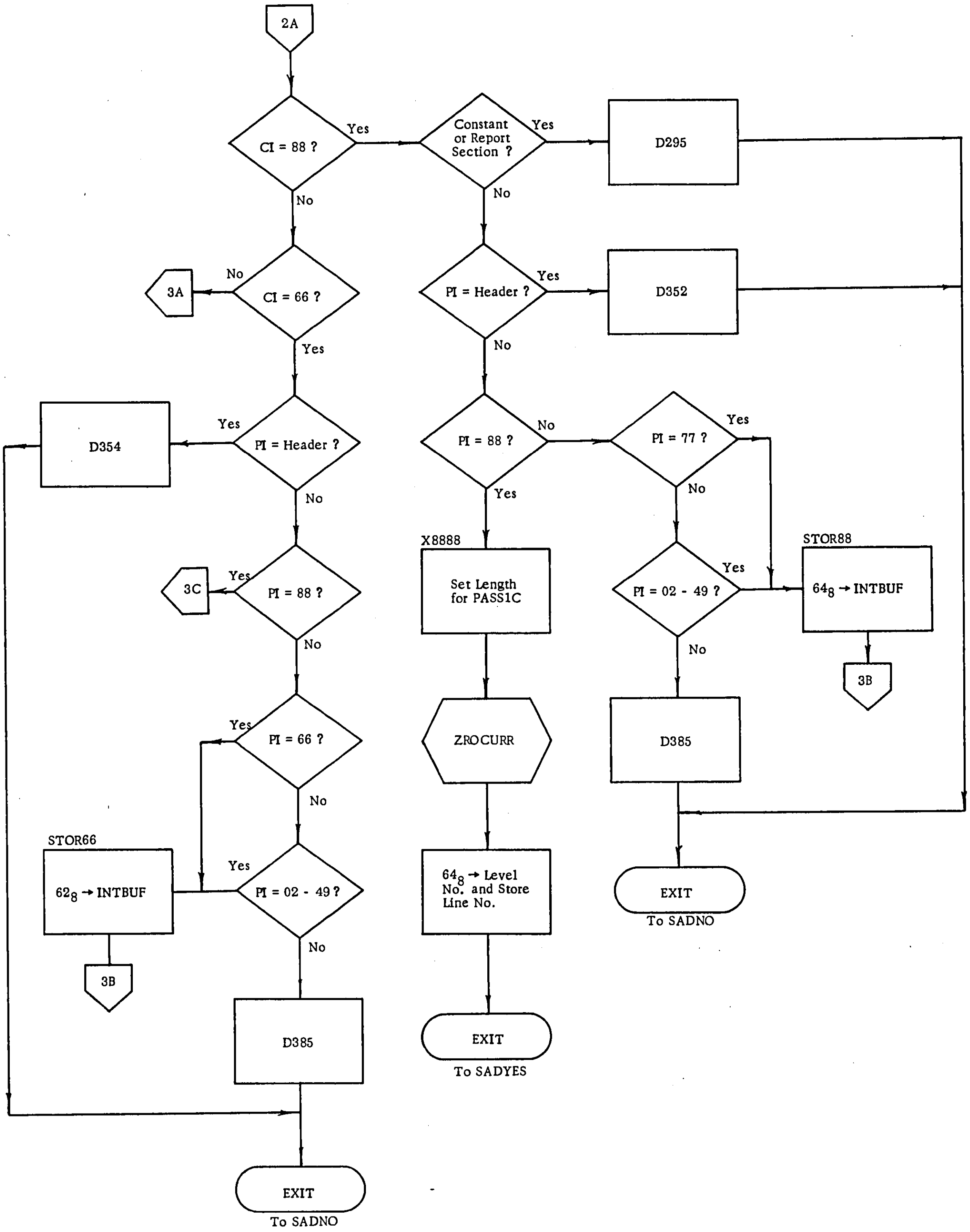


Figure 3-7. XDD1 Flowchart (2 of 3)

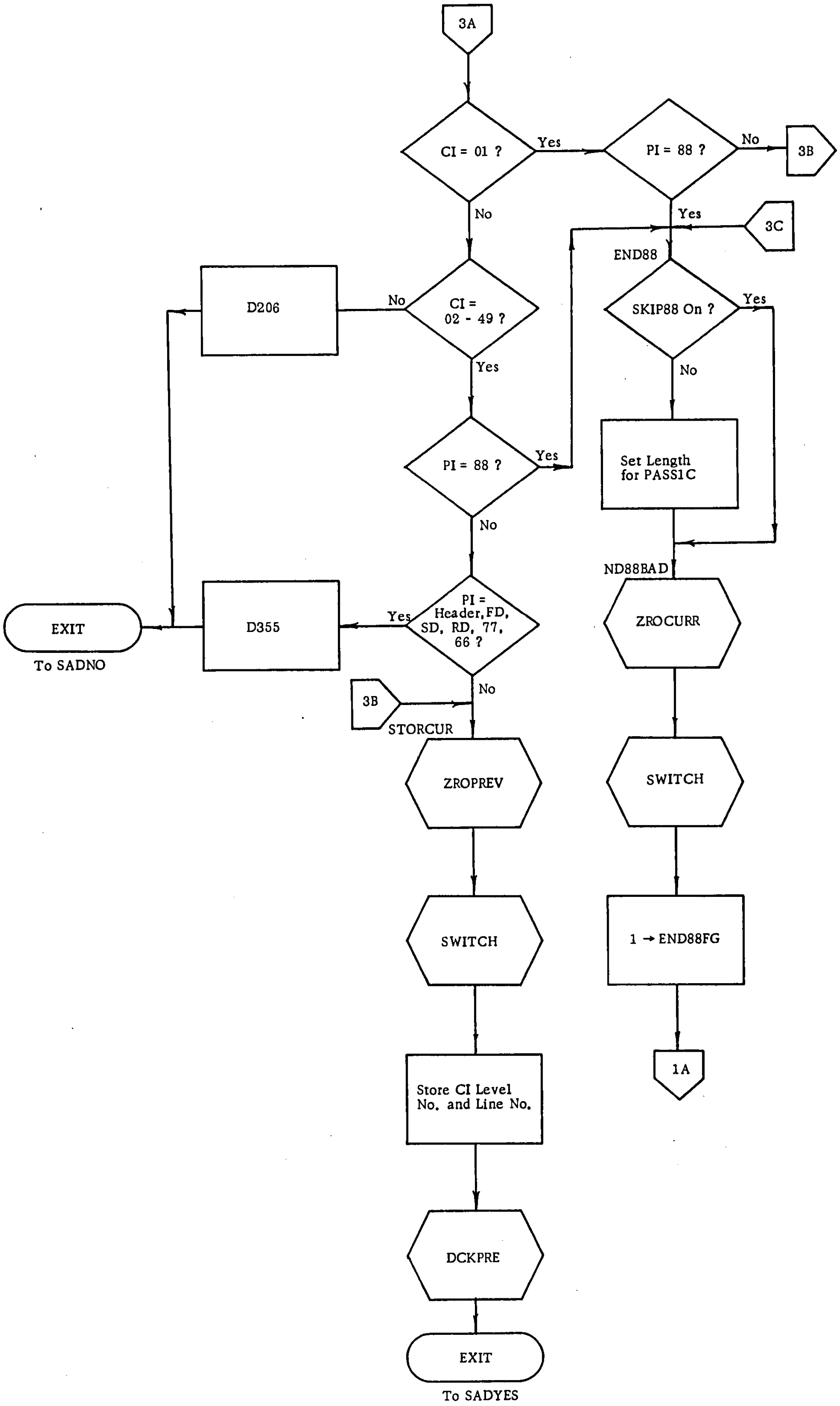


Figure 3-7. XDD1 Flowchart (3 of 3)

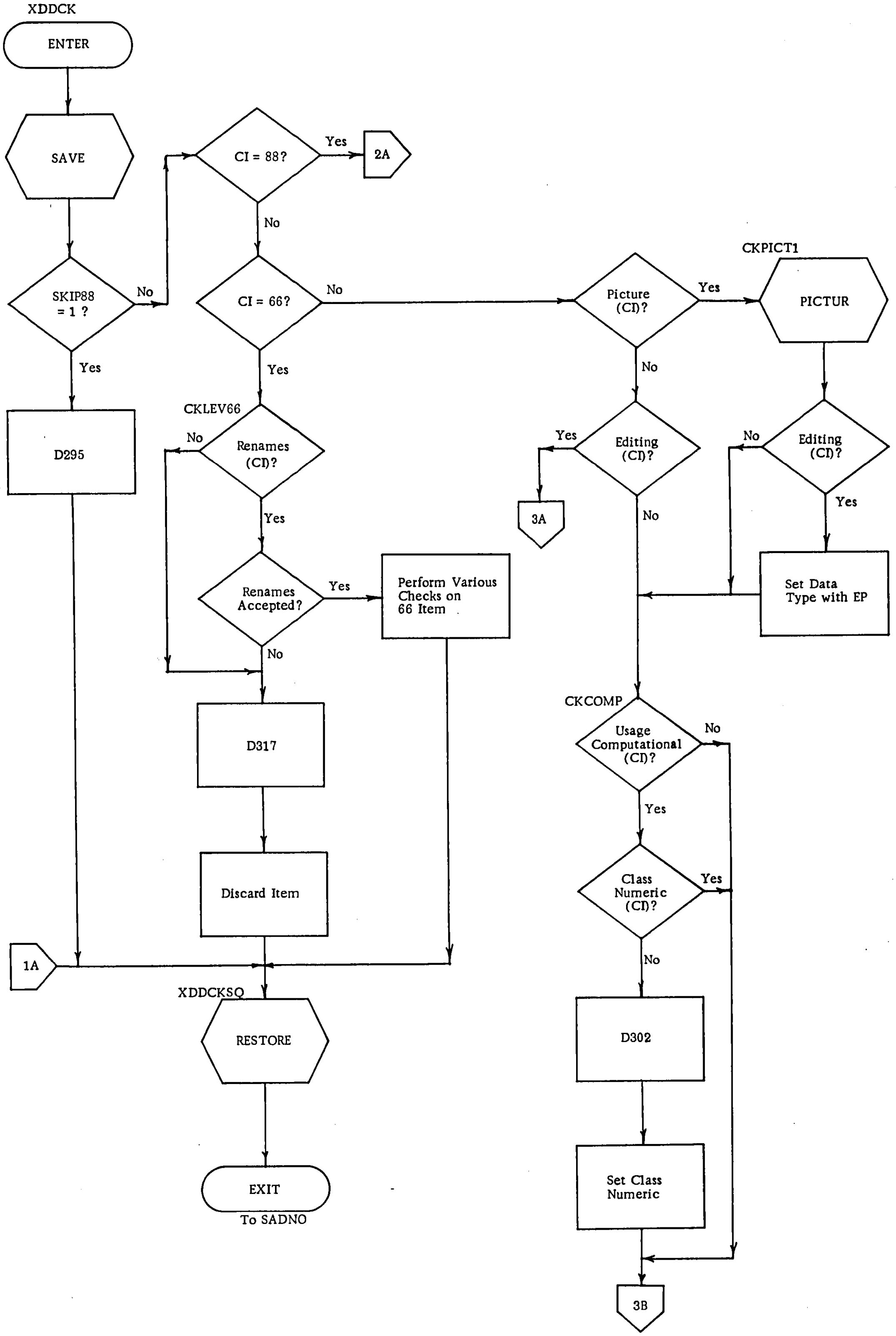


Figure 3-8. XDDCK Flowchart (1 of 4)

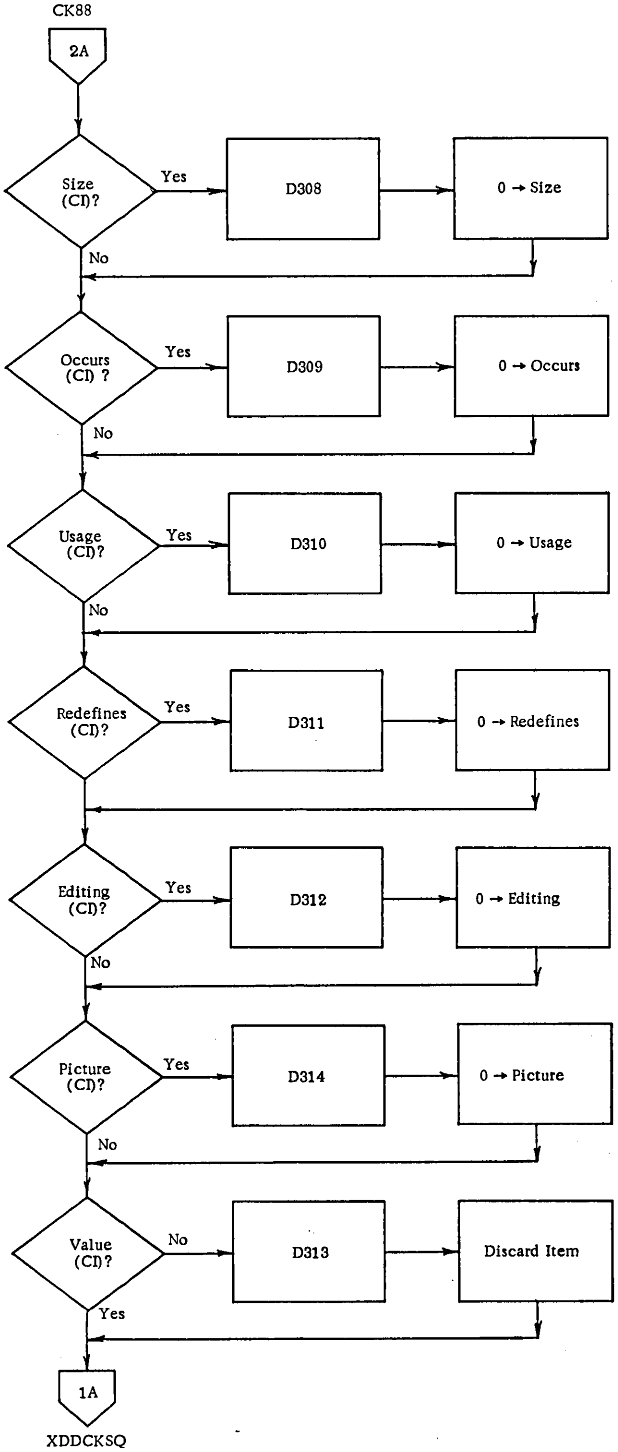


Figure 3-8. XDDCK Flowchart (2 of 4)

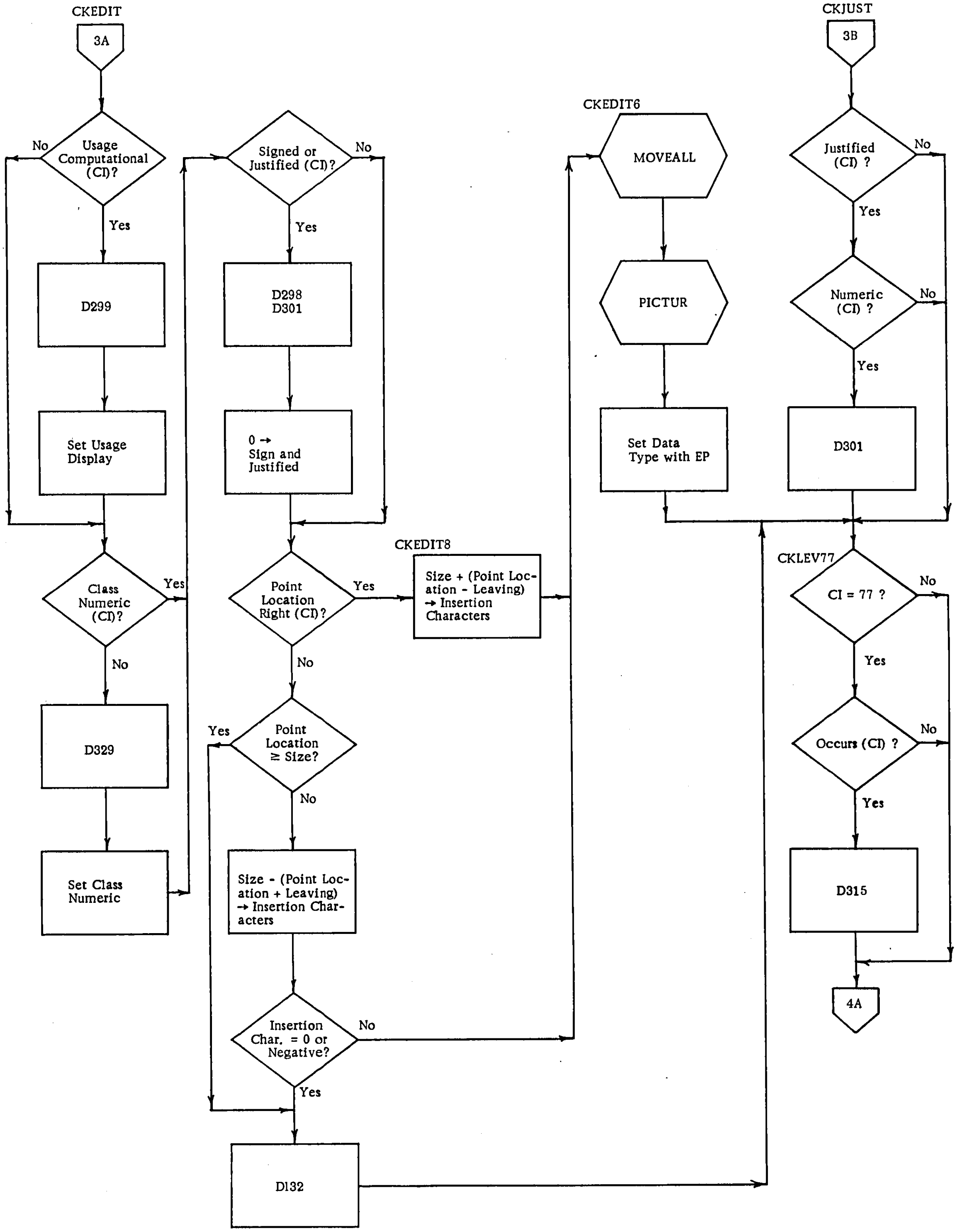


Figure 3-8. XDDCK Flowchart (3 of 4)

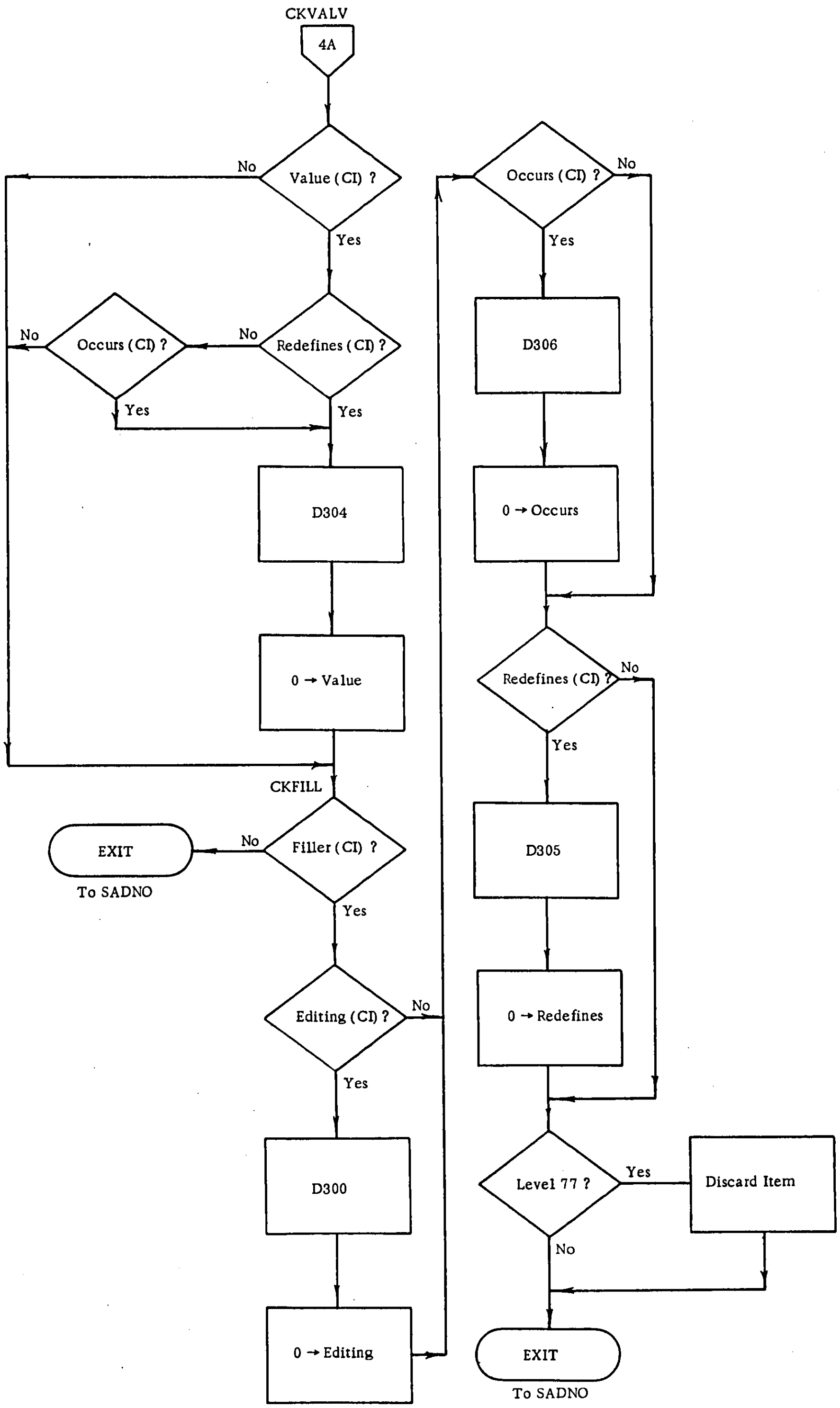


Figure 3-8. XDDCK Flowchart (4 of 4)

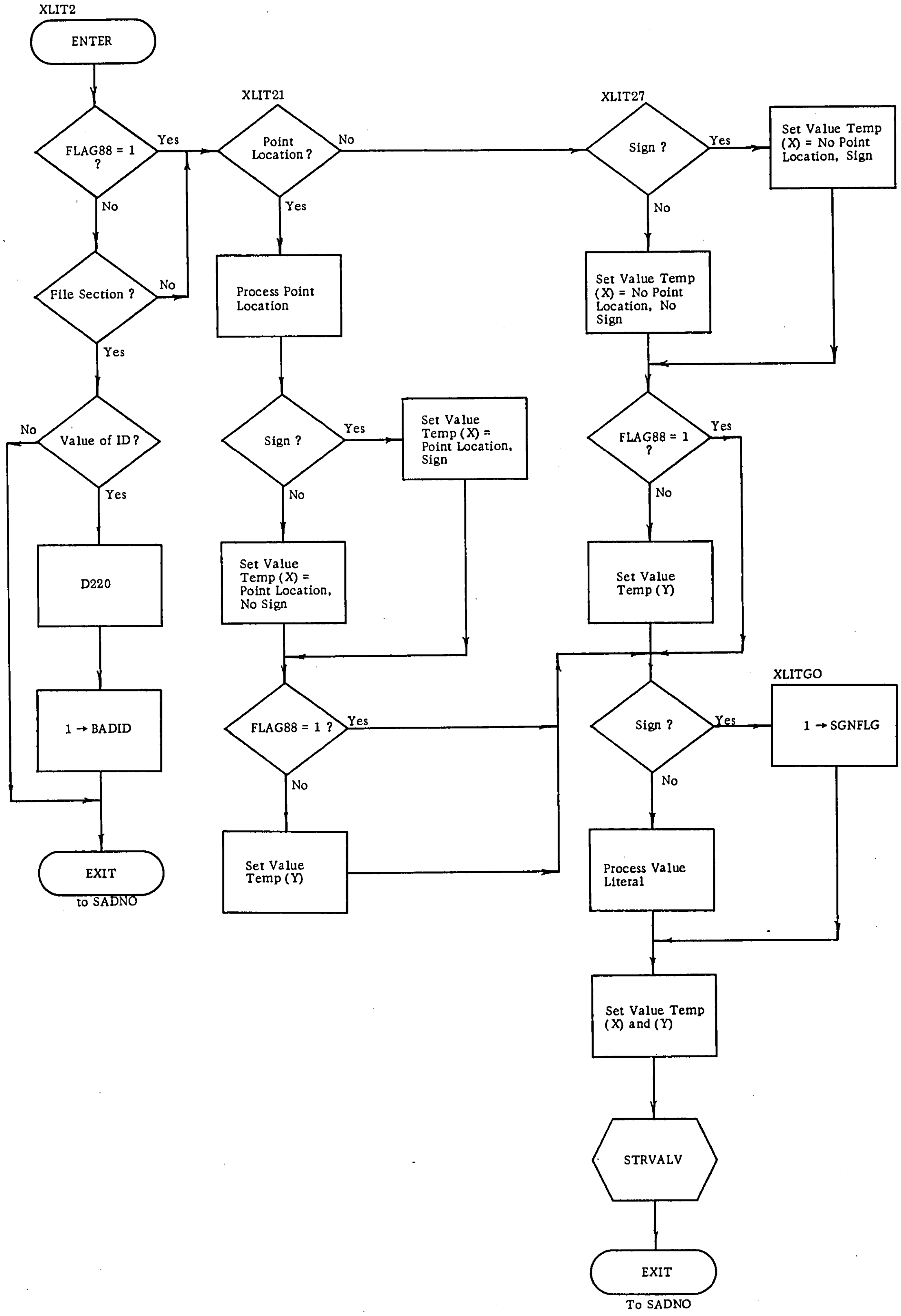


Figure 3-9. XLIT2 Flowchart

Table 3-4. Pass 1B Internal Subroutines (1 of 2)

Name	Description	Flowchart	Diagnostic Issued	Internal Subroutine Performed	Pertinent Compiler Internal Table Formats
DCKPRE	Performs various checks on previous item to insure legality of all specified clauses; determines whether previous item is elementary or group before putting item into the DNT	Figure 3-10	320 321 322 323 324 235	SQUASH SQVAR SPECSQ	
DCOMON	Puts 'Tally' and 'File-Label' into the DNT				
DID	Puts an item's hash link and name into the DNT	Figure 3-11			HL Name
ENVSQ	Puts Special Name intems HL, Name, SNL, SNS, SNF1, SNF2 into the DNT	Figure 3-12			HL Name SNL SNS SNF1 SNF2
FIND	Looks up file-name in DNT; If found, puts pointer to NW1 in B7, otherwise B7 is zero				
FINI88	Is performed when the last 88 item under a conditional variable is encountered; makes the conditional variable an elementary item		235	SQVAR	
MOVEALL	Moves the correct number of nines from the editing clause into PICTEMP in preparation for the PICTUR processor				
PUTPAT	Sets up a File Table for Scope file-name "Input" and "Output" if they have not been specified in the COBOL Source Program				EAT File Table
PUTSLCT	Determines whether file-name is object of Select or Renaming option			SETFET	
QUALEOM	Stores the qualifing name into the end of Memory (EOM)				

Table 3-4. Pass 1B Internal Subroutines (2 of 2)

Name	Description	Flowcharts	Diagnostic Issued	Internal Subroutine Performed	Pertinent Compiler Internal Table Formats
SETDNT	Puts the file-name into the DNT	Figure 3-13			
SETEOM	Puts the name being saved for PASS1B into the EOM				
SETFD	Zeros the FDBUFF buffer work area and appropriate flags				
SETFET	Sets up a File Table for current file-name	Figure 3-14	131 363		
SPECSQ	Is performed when the first 88 item under a conditional variable is encountered; since it cannot be determined at this time whether the condition variable is elementary or group, appropriate space is reserved in the DNT	Figure 3-15			
SQASH88	Puts the condition name into the DNT	Figure 3-16			HL Name CN1 CN2
SQUASH	Puts the item into the DNT by unpacking appropriate information from PASS1B's buffer area "SQASHBU"	Figure 3-17		DID	
SQVAR	Puts the condition variable into the DNT After all condition names have been handled			SQUASH	
SQ88DDL	Is performed only when 88 is encountered; puts the literal into the DNT				DDL
STRVALU	Sets up the current literal and saves for subsequent PASS1B processing				
SWITCH	Changes the index in PASS1B's SQASHBU buffer area to make current item previous				
XDTB	Converts decimal to binary				
ZROCURR	Zeros the current SQASHBU buffer area				
ZROPREV	Zeros the previous SQASHBU buffer area				
ZROSLT	Zeros the SELECT buffer area				

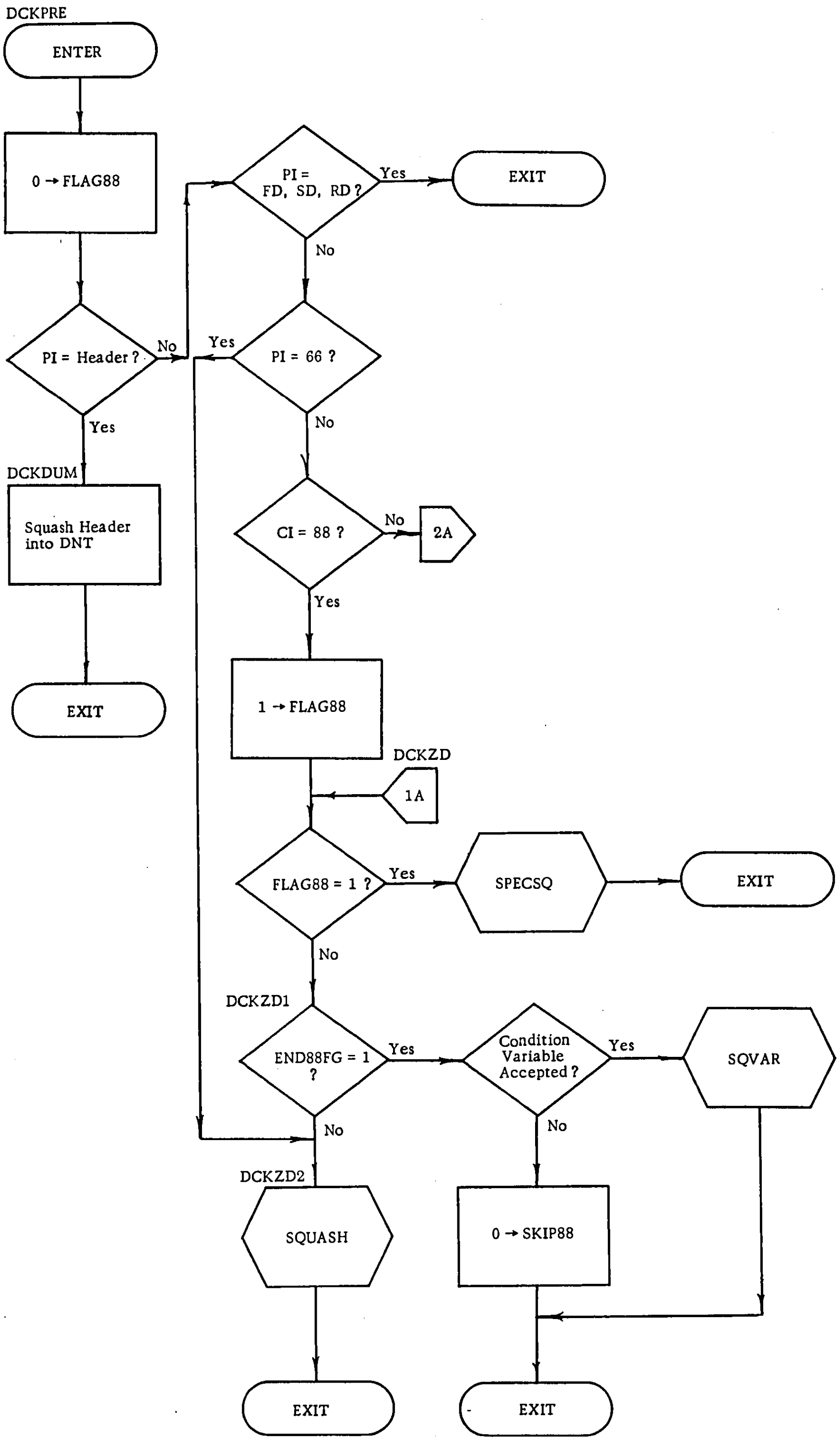


Figure 3-10. DCKPRE Flowchart (1 of 2)

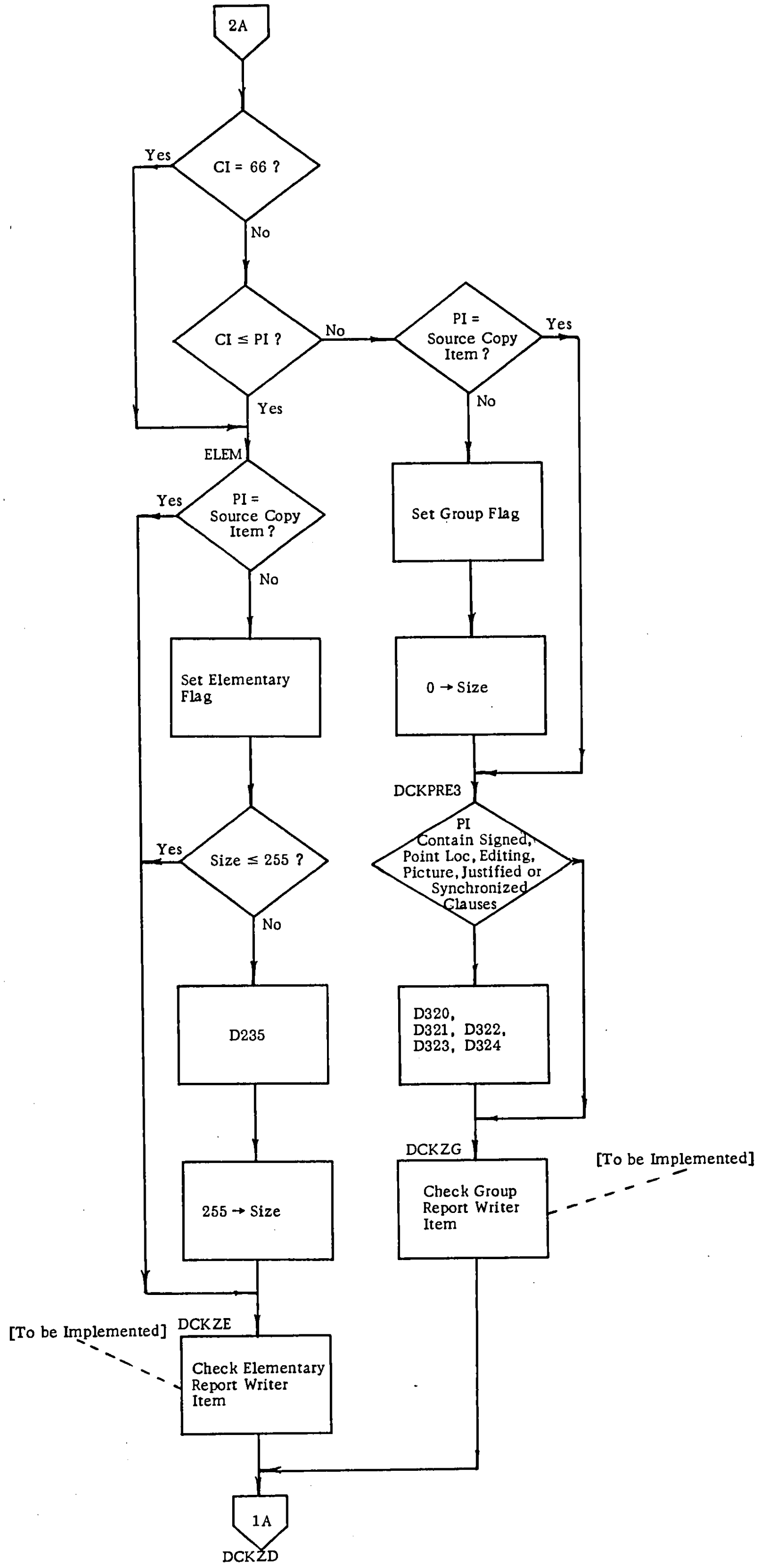


Figure 3-10. DCKPRE Flowchart (2 of 2)

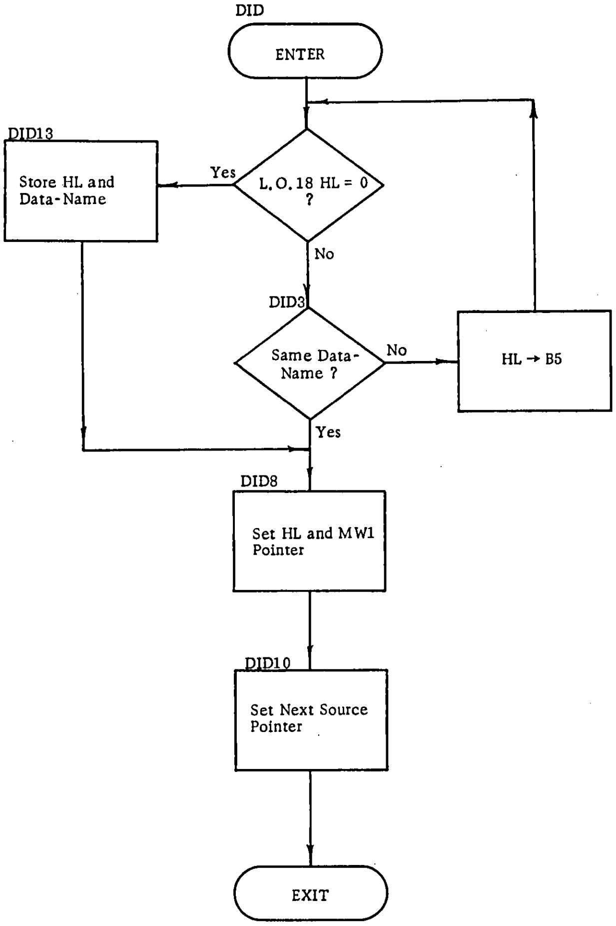


Figure 3-11. DID Flowchart

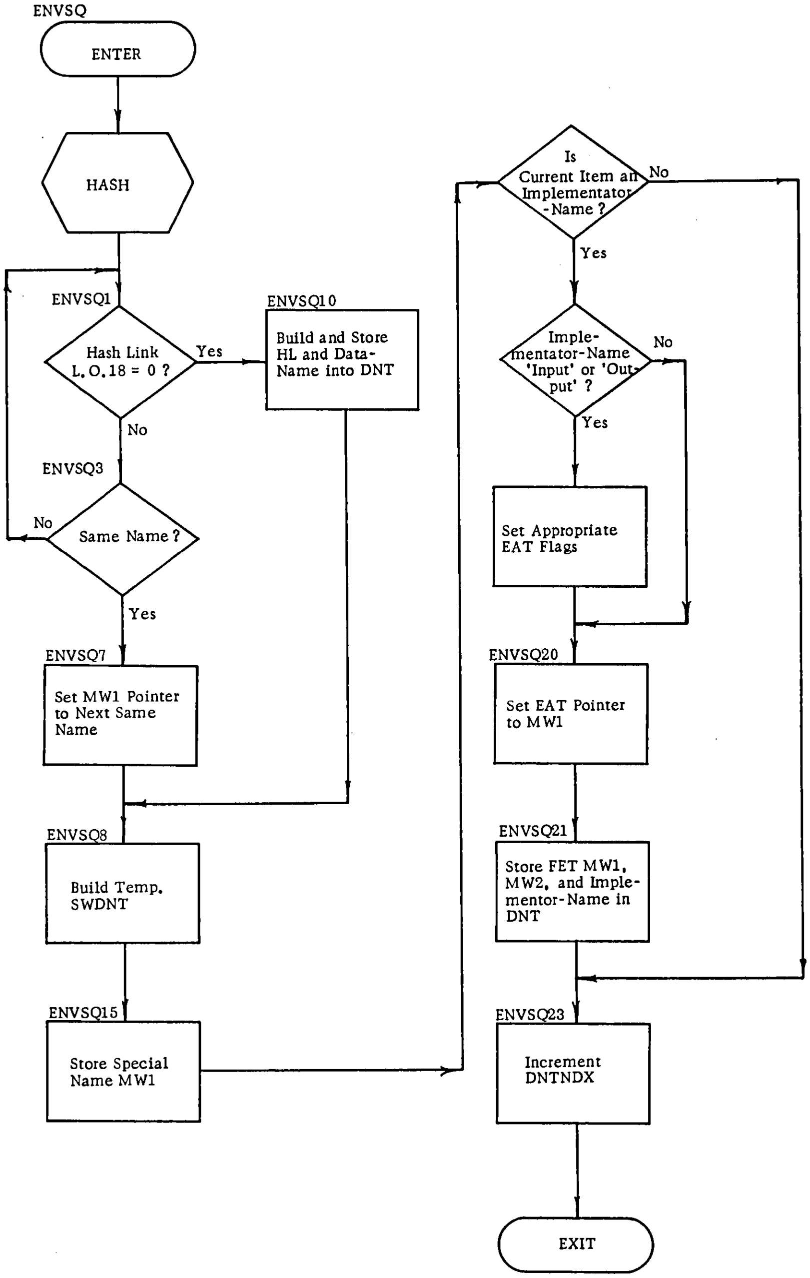


Figure 3-12. ENVSQ Flowchart

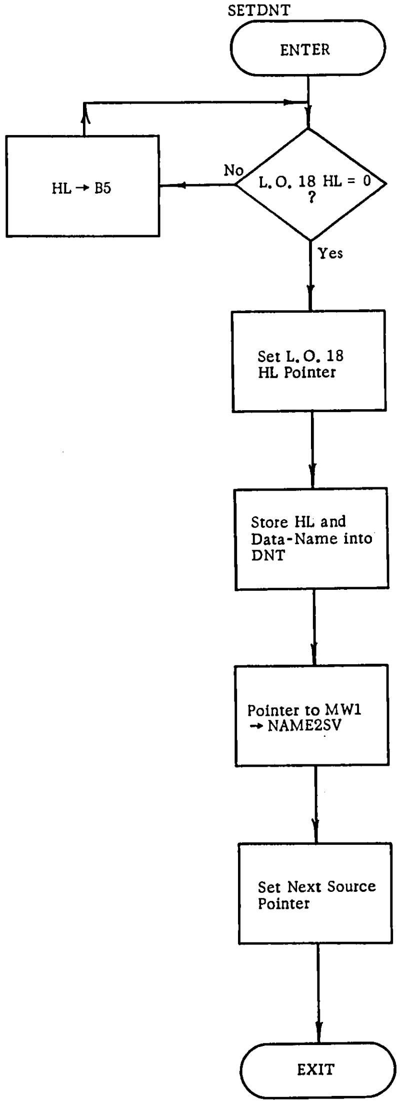


Figure 3-13. SETDNT Flowchart

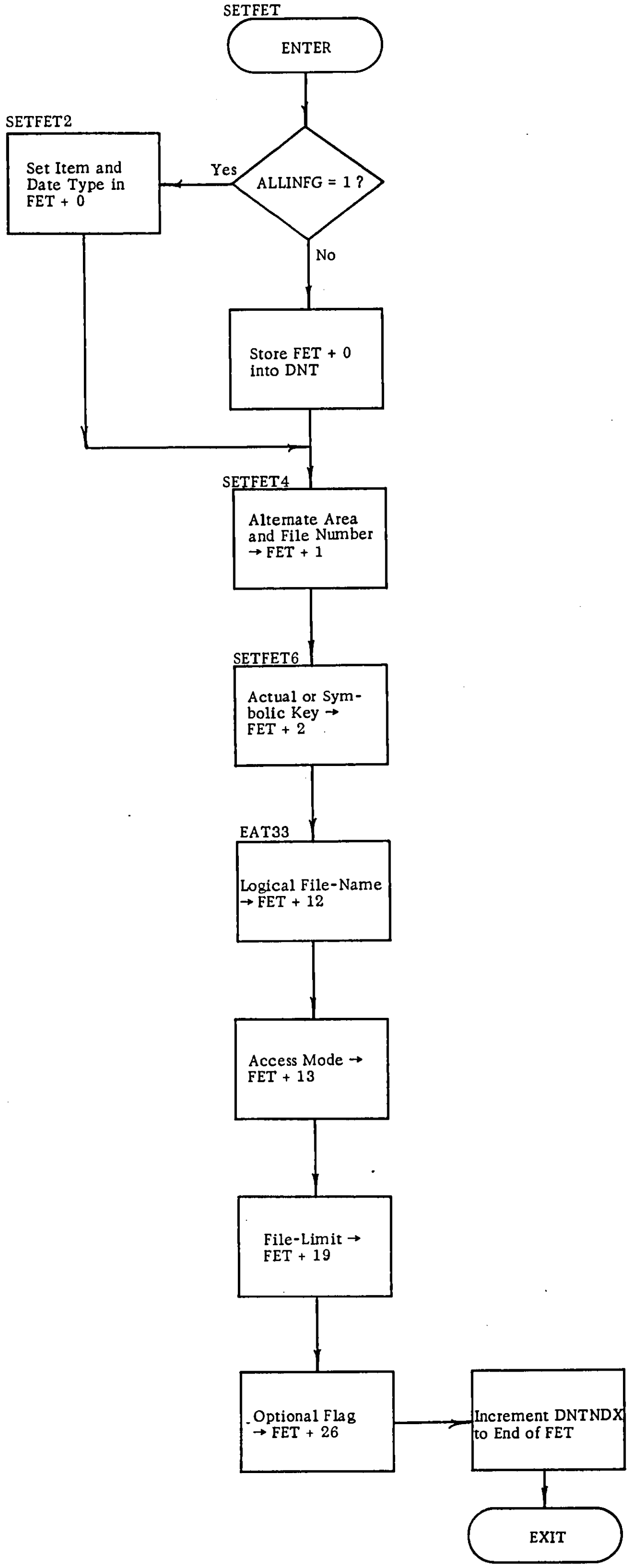


Figure 3-14. SETFET Flowchart

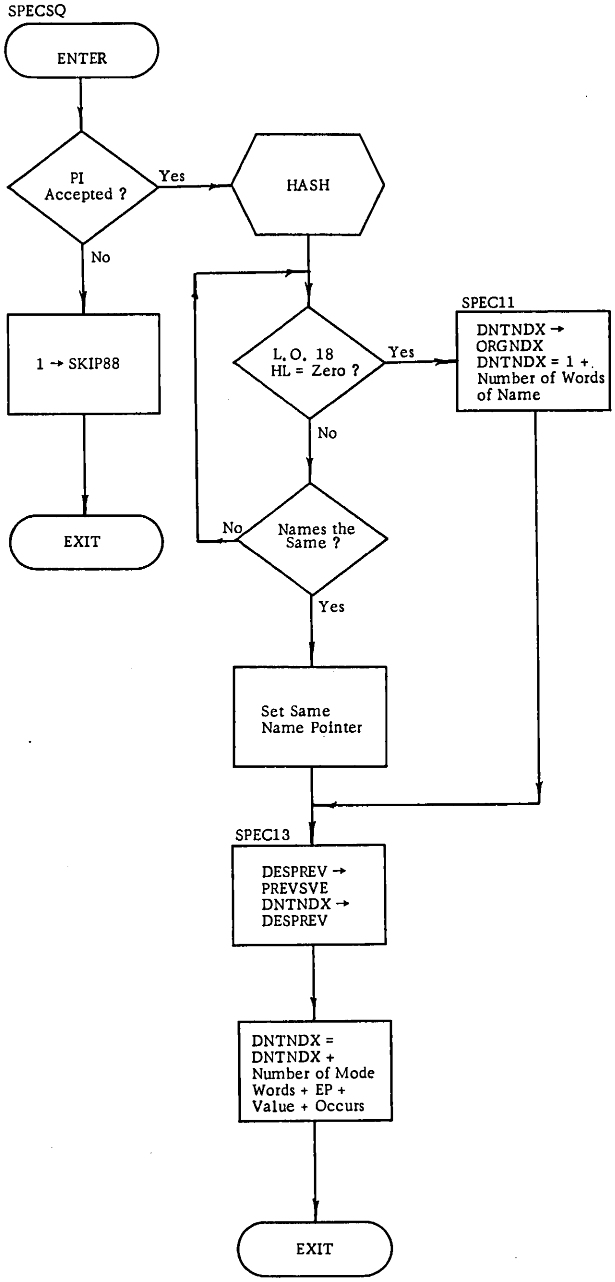


Figure 3-15. SPECSQ Flowchart

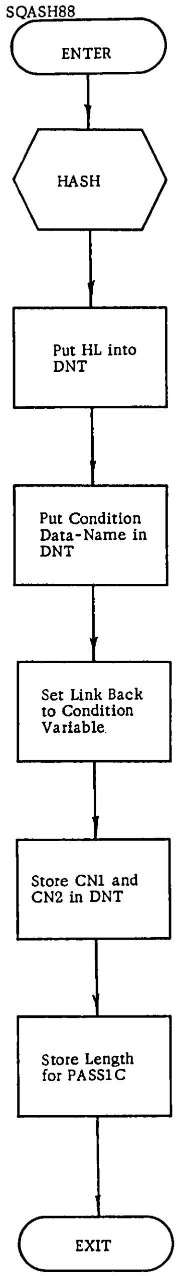


Figure 3-16. SQASH88 Flowchart

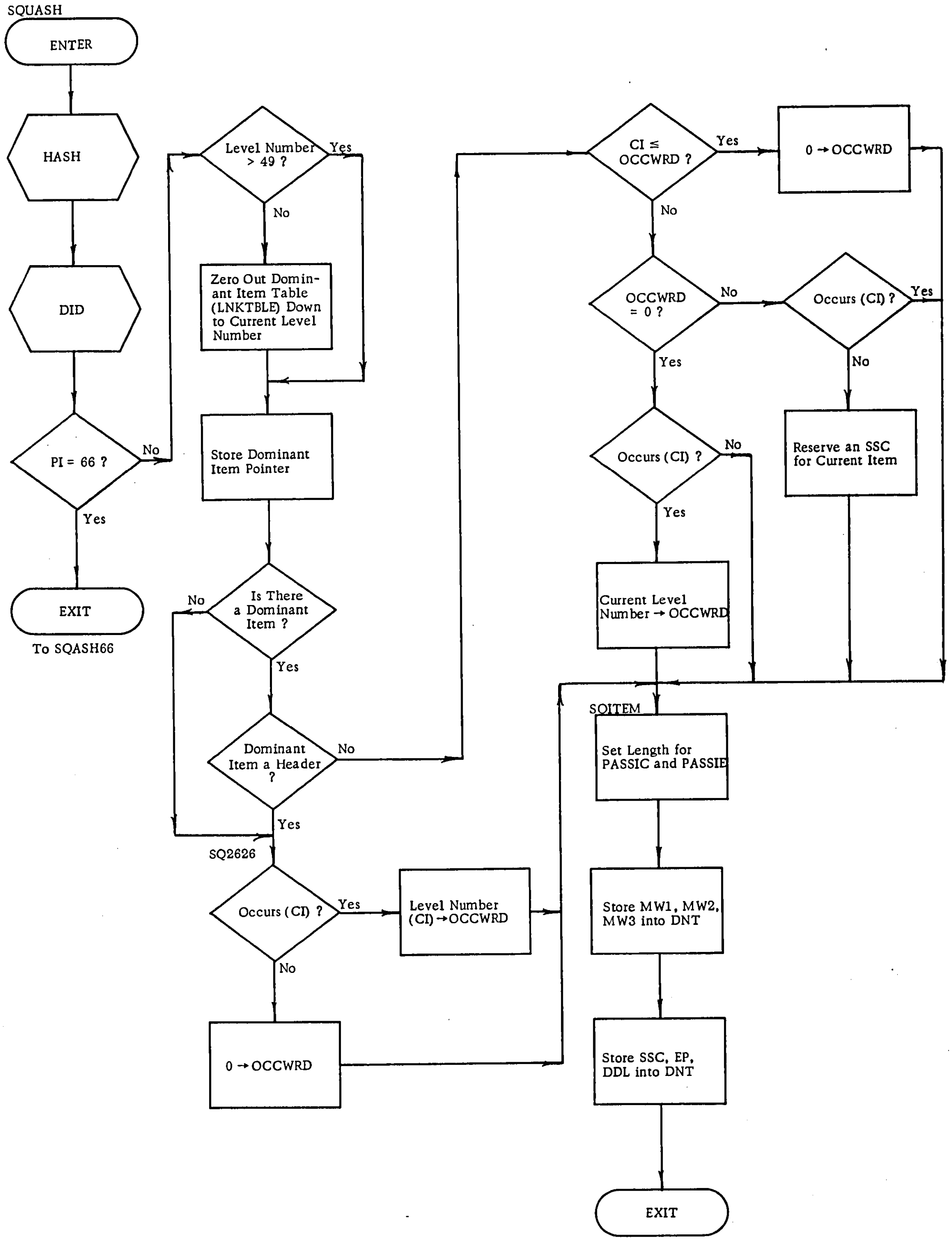


Figure 3-17. SQUASH Flowchart

Table 3-5. Subroutines Common to ID, Environment and Data Division Performed From SYNTBLE

Name	Description	Flowchart	Diagnostics Issued	Internal Subroutines Performed	Pertinent Compiler Internal Table Formats
A	Current source word must begin in column 8.		17		
AB	Current source word may begin in column 8.				
COMMA	Current source word may be preceded by a comma.				
ENDSOUR	Has 2 return exits, one to SADNO if current source word is not an EOF; and one to SADYES if current word is an EOF.				
INTEGER	Has 2 return exits, one to SADNO if current word is not an integer; one to SADYES if current word is an integer.				
KEYWORD	Has 2 return exits, one to SADNO if current word is not a keyword; one to SADYES if current word is a keyword.				
NAME	Has 2 return exits, one to SADNO if current word is not a data-name; one to SADYES if current word is a data-name.				
NONNLIT	Has 2 return exits, one to SADNO if current word is not a non-numeric literal; one to SADYES if current word is a non-numeric literal.				
NUMBER	Has 2 return exits, one to SADNO if current word is not a number; one to SADYES if current word is a number.				
SBW	"Scan Back Word"--Turn NOSCNFL flag on signifying to SCAN2 not to pick up next source word yet.				
SNC	"Scan Next Card"--Set SKIPOPS flag signifying to SCAN2 to skip to the next source card and position to the first word on it.				
SNP	"Skip Next Period"--Set SKIPOPS flag signifying to SCAN2 to skip to the source word following the next period.				
SNW	"Scan Next Word"--Set flag signifying to SCAN2 to pick up to next source word.				
XA	Has 2 return exits, one to SADNO if current word does not begin in column 8; one to SADYES if current word does begin in column 8.				

DCKPRE 7) Every elementary item must have a SUM, SOURCE or VALUE clause, except when within a SOURCE SELECTED group. These clauses may not appear at the group level.

XRECK 8) A COLUMN NUMBER clause can appear on an elementary item only and if it exists, a SIZE or PICTURE clause must also appear.

XRECK 9) The column number plus the field length must not be greater than end of the print line or column number on the next item.

XRECK 10) The column numbers within a report group (01) must be increasing. A column number that is out of order (position left of that on the preceding item) will cause the item to be discarded.

XRECK 11) The GROUP INDICATE clause can only be specified on an elementary item within a TYPE DETAIL report group.

XRECK 12) The RESET clause can only be specified on an elementary SUM item.

XRECK 13) The SUM clause can only be specified on an item within a TYPE CONTROL FOOTING report group.

XRECK 14) The USAGE can only be specified as DISPLAY.

15) No check is made on CLASS clause.

PICTURE Processor

The PICTURE processor interprets the PICTURE set aside or created by the item description processor. It examines the successive characters in a picture, determining the size of field, decimal point position, editing, class and sign. In addition, the legality of the combination of characters in the picture is determined by checking the Picture Precedence Table against the History Register Table. If any editing character is found in the PICTURE, a MURAL (condensed picture) is built as part of the elementary data description for later use by the editing. Finally, all the information determined by the PICTURE clause is placed in the SQASHBU Buffer table (see Table 3-1).

The following compile-time subroutine is included in the PICTURE processor to perform some of the major functions:

GETBYT - To get the next character in a picture and shift a "1" bit into the current bit word according to the bit position field of the PICTURE Character Table. If there is a number within the parentheses in the PICTURE, the previous character repeats the stated number of times, unless the number exceeds 255.

PAKMRL - To pack the successive elements that are needed in a mural. It checks to determine repeated mural instructions and also checks to see when it is near the end of a computer word.

SETMAT - To place information determined from the PICTURE clause with the clause checking matrix and give out diagnostics if syntax error or illegal characters are found in the PICTURE.

Picture Precedence Table

This table contains a one-word entry for each of the 64 Display Code characters and extra entries for repeated currency sign, - + and trailing P. (See Table 3-6.)

Bit 59-36	Not used.
Bit 39-15	Used to represent allowable precedence. One bit position is assigned to each character that may legally appear in a picture and also for trailing + - and P. The bit position is 1 for each character to the right of which the present character is illegal.
Bit 14	Decimal Count Indicator. This bit is 1 if the character being examined is to be counted as a numeric position in the elementary item. (Includes data and suppression characters and Ps.)
Bit 13	Exterior Count Indicator. This bit is 1 if the character being examined is to be counted as a character when used as a source field. (Includes data, suppression, insertion, and report sign characters.)
Bit 12	Interior Count Indicator. This bit is 1 if the character being examined is to be counted as a character when used in a receiving field. (Includes data, suppression, insertion and report sign characters.)

Table 3-6. Picture Precedence Table (Octal) (1 of 2)

0	0000	0000	0000	3762	1050
1	0000	0000	0000	0007	0470
2	0000	0000	0000	0007	0470
3	0000	0000	0000	0007	0470
4	0000	0000	0000	0007	0470
5	0000	0000	0000	0007	0470
6	0000	0000	0000	0007	0470
7	0000	0000	0000	0007	0470
8	0000	0000	0000	0007	0470
9	0000	0000	0000	3707	0436
A	0000	0000	3771	7777	0434
B	0000	0000	2000	3762	0747
C	0000	0003	32001	7762	1453
D	0000	0003	32001	7762	1554
E	0000	0000	0000	0007	0470
F	0000	0000	0000	0007	0470
G	0000	0000	0000	0007	0470
H	0000	0000	0000	0007	0470
I	0000	0000	0000	0007	0470
J	0000	0000	0000	0007	0470
K	0000	0000	0000	0007	0470
L	0000	0000	0000	0007	0470
M	0000	0000	0000	0007	0470
N	0000	0000	0000	0007	0470
O	0000	0000	0000	3762	1050
P	0000	0003	4330	3774	0041
Q	0000	0000	0000	0007	0470
R	0000	0003	2001	5762	1455
S	0000	0003	7777	7770	0037
T	0000	0000	0000	0007	0470
U	0000	0000	0000	0007	0470
V	0000	0003	1410	3700	0040
W	0000	0000	0000	0007	0470
X	0000	0000	3771	7777	0435
Y	0000	0000	0000	0007	0470

Table 3-6. Picture Precedence Table (Octal) (2 of 2)

Z	0000	0003	6106	3777	0142
Space	0000	0000	0000	0007	0470
+	0000	0003	2001	7762	1352
-	0000	0003	2001	7762	1251
*	0000	0003	6206	3777	0243
/	0000	0000	0000	0007	0470
(	0000	0000	0000	0007	0470
)	0000	0000	0000	0007	0470
=	0000	0000	0000	0007	0470
≠	0000	0000	0000	0007	0470
,	0000	0003	2000	3762	1145
.	0000	0003	3410	3762	0546
CS	0000	0003	7776	3762	0644
:	0000	0000	0000	0007	0470
≤	0000	0000	0000	0007	0470
%	0000	0000	0000	0007	0470
[	0000	0000	0000	0007	0470
]	0000	0000	0000	0007	0470
→	0000	0000	0000	0007	0470
≡	0000	0000	0000	0007	0470
∧	0000	0000	0000	0007	0470
∨	0000	0000	0000	0007	0470
↑	0000	0000	0000	0007	0470
↓	0000	0000	0000	0007	0470
>	0000	0000	0000	0007	0470
<	0000	0000	0000	0007	0470
≥	0000	0000	0000	0007	0470
¬	0000	0000	0000	0007	0470
;	0000	0000	0000	0007	0470
CS (Repeat)	0000	0003	6306	3767	0344
P (Trailing)	0000	0003	1410	0004	0060
- (Repeat)	0000	0003	6306	7777	0356
+ (Repeat)	0000	0003	6307	3777	0357
B (Trailing)	0000	0003	2001	6762	1561

Bit 11-6	Mural Code. This code is to be used in the output mural to represent the present character.
Bit 5-0	Amount of right shift necessary to shift a 1 from bit 59 to the bit position corresponding to the current character.

History Register

This register has a bit assigned for each different character allowed in a PICTURE. It contains a "1" if the bit has appeared before in a left-to-right scan. Bit assignments are shown in Table 3-7.

Mural (Condensed Picture)

The constant representing the condensed PICTURE is made of one or more words of coded bits to be interpreted by the EDIT subroutine. These PICTURE constants are left-justified and left-synchronized and contain their own termination indicator. (See Table 3-7.)

In scanning an encoded PICTURE from left to right, the following rules apply:

1. The first bit of a field is a "cue" bit. Depending on the value of this bit, the successive bits are interpreted to be a character field or repetition bit.
2. If the value of the cue bit is "0," then the next four bits is one of the 4-bit codes representing a legal PICTURE character (see Table 3-8). This may be the first character of the PICTURE or a new character and is not a repetition of the previous one.
3. If the value of the cue bit is "1," it means the previous character in the PICTURE is repeated once and the next bit becomes the cue bit. This process goes on till a cue bit of "0" is encountered. (The last field in a PICTURE must have a cue bit of "0" to indicate the end of the PICTURE.)

Examples:

00100001010010000000 ≡ 9.9
0010011001010010000000 ≡ 999.9
00011101001000111101001000111100101001001 ≡ $$,$$$,$$$.99

Table 3-7. History Register Table

Shift Amount (Octal)	Bit Position	Character
34	32	A
35	31	X
36	30	9
37	29	S
40	28	V
41	27	P (Leading)
42	26	Z
43	25	*
44	24	CS
45	23	,
46	22	.
47	21	B
50	20	O
51	19	- (Leading)
52	18	+ (Leading)
53	17	C
54	16	D
55	15	R
56	14	- (Trailing)
57	13	+ (Trailing)
60	12	P (Trailing)
67	5	Syntactical Error
70	4	Illegal Character

Table 3-8. Mural Code Table

$b_1b_2b_3b_4$	Meaning
0 0 0 0	End of picture
0 0 0 1	Z (repeat)
0 0 1 0	* (repeat)
0 0 1 1	Repeated currency sign , + or -
0 1 0 0	9, X, or A
0 1 0 1	.
0 1 1 0	Currency sign (first)
0 1 1 1	B
1 0 0 0	O
1 0 0 1	,
1 0 1 0	- (first)
1 0 1 1	+ (first)
1 1 0 0	CR
1 1 0 1	DB
1 1 1 0	End of computer word
1 1 1 1	Not used

REPORT WRITER - GENERAL DESCRIPTION

Purpose of the Report Writer

The report writer feature is introduced into COBOL in order to allow the specification of print page formats for a report without requiring the programmer to specify the procedures by which the lines are produced. The report writer thus is a nonprocedural feature of COBOL. The user describes the format of the report in the REPORT SECTION and utilizes data items which he as described elsewhere in the COBOL data DIVISION as the source of the data for the report.

Problems

Because the report function is nonprocedural, and because of the method by which certain of the features are specified, special problems appear when processing for the report writer.

References From Report Items (some problems which Phase 1B must solve)

Many of the items in the report writer serve as both definitions of items and as references to items defined outside the report writer. Sometimes more than one item is defined, and some of the references are to establish links rather than SOURCE items. The following three examples illustrate the problems:

EXAMPLE 1: 02 A SOURCE IS B OF C COLUMN 66

This example defines the item A as it appears on the print line in edited format. A cannot be referenced from the PROCEDURE DIVISION. It references the item B of C, which is outside the report writer.

EXAMPLE 2: CONTROL ON D OF E

This example defines an "old" D OF E which is to be referenced both in the comparisons the report writer uses to determine control breaks, as SOURCE of any items within a control footing, and in any USE statements specified for a report group that is a control footing. This example also references the item D OF E which is outside the report writer. Note that an unusual complication is imposed on the compiler when D OF E is allowed to be qualified or subscripted (it does not seem that DOD allows subscripting on CONTROL items) in that references to the "old" D OF E which is defined by the report writer are not truly references to a qualified or subscripted item. The qualification or subscripting serves as cross referencing only and is eventually replaced by a simple reference to a field generated by the report generator. For example, if a USE FOR REPORTING "report-group" appears referencing D OF E, it must first be determined that "report-group" refers to a control footing. Then somehow it must be ascertained that the reference D OF E, together with any subscripts, is the same item as referenced elsewhere in an RD by a CONTROL ON D OF E, in which case the reference D OF E is replaced by a reference to "old" D OF E, which in turn is replaced by a report generated field.

EXAMPLE 3: 03 F SUM B OF C

This example defines the item F as it appears on the edited print line and another item F as an arithmetic accumulator. The reference to B OF C does not mean that B OF C is a source. Instead it establishes a link to one or more DETAIL report groups which also reference B OF C. The code to sum B OF C into the sum-accumulator F must be provided at the DETAIL item (such as the item of EXAMPLE 1).

Thus, in some cases, the references made from the report writer serve a different purpose (i.e., cross reference) than references made from COBOL procedure division. When the reference is used this way, any items which happen to reference the same nonreport item must be linked together by Phase 1G.

General Discussion of the Processing of the Report Writer

The source cards describing the reports are first processed by Pass 1B. A check is made at that time to determine that the various clauses used in the description of report items comply with DOD rules. The report and item descriptions are placed in the data name table. But references to other data (such as to CONTROL items or SOURCE items) cannot be completed until after Pass 1C because of the possibility of COPYs. Furthermore, because it is possible to use difference source descriptions of the same data by items that must be linked because they use the same data (for example, a SOURCE item referenced in a TYPE DE line as A OF B with redundant qualifier B and the same item referenced in a SUM item as A), the references cannot be completed until after Pass 1D. Consequently, the references and all information about the reports except that necessary for defining item locations, are stored in the alphabetic form in which they are received from the SCAN routine on disk on the report reference file until the beginning of Pass 1G.

Pass 1G processes each report in the order the data items were stored by Pass 1B in the report reference file.

Pass 1G first reads the report references and report item descriptions from the report reference file. Each reference from the report is looked up in the data name table in the same manner as references from the procedure division.

Pass 1G must then analyze the interrelationships between data items in the report and store this analysis in table form. Syntax trees are generated for each compare, move, and add required in the report, and they are output as a special segment of the procedure division, handled like DECLARATIVE USE sections. A report module is prepared which contains a description of each line of the report and contains links to the procedure division code generated by the trees and which accomplish the work required for the line. The report module also includes those report-defined fields that can be referenced from the procedure division: LINE-COUNTER, PAGE-COUNTER, SUM items, and the old value of CONTROL items. The report table is passed to the assembler as an independent module with an entry point identified by the first seven characters of the report name.

Pass 1H assigns values to the report line images and other report items, and outputs these to the assembler as the last portion of WORKING-STORAGE.

Pass 2 generates code from the syntax trees for COMPARE, MOVE, and ADD, so that this code may be referenced from the report tables.

References from USE BEFORE REPORTING sections to CONTROL items are changed to reference the old value of these items. INITIATE, GENERATE, and TERMINATE verbs generate code linking via an external name consisting of seven characters of the report name to the report table a DETAIL line number, and to a report processor from the library.

The Report Processor routine will, at object time, utilize the information in the report table to prepare print-line images in the last portion of WORKING-STORAGE and write them on output files defined in the FILE SECTION.

These files are printed later using standard SCOPE 3.0 facilities.

Structure

The code necessary for the report will reside partly in a library program "REPORT." and partly in the generated output of the compiler. That part in the generated output is in five different areas of the main overlay of the PROCEDURE DIVISION and in one FET.

1. Each line image is set up, with initial VALUEs, in WORKING STORAGE.

2. A description of the conditions associated with each report group, with address pointers to the line image, to the MOVE code, etc., is in a separate report module. An entry point is provided at the beginning of this module consisting of seven characters of report name.

3. Strings of code to accomplish the comparisons, moves, adds, and clearing required by the report appear as the first part of the PROCEDURE DIVISION.

4. Any DECLARATIVE sections specifying USE BEFORE REPORTING appear in the DECLARATIVES SECTION of the generated PROCEDURE DIVISION.

5. Any GENERATE, INITIATE, or TERMINATE verbs appear in the main body of the PROCEDURE DIVISION. The calling sequence is approximately:

LOAD	report entry point
LOAD	DETAIL line number
RJ	REPORT.

6. The file upon which the report output is to be written is described separately in a FET.

Compiler Processing of the Report Function

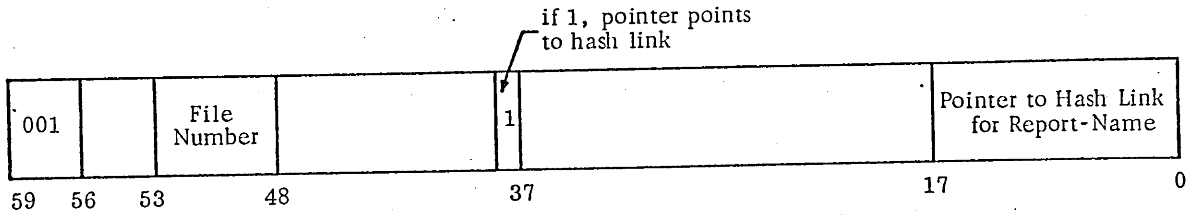
Phase 1A

Special Names. The CODE to be referenced by mnemonic name is saved in the Data Name Table as a SNL item.

Phase 1B - File Section

REPORTS ARE Clause. Each report named in a REPORTS ARE clause causes the report name to be placed in the DNT. A position is assigned in EAT. A pointer is placed in that position pointing to the hash link item of the report name in the DNT. Also, the file-number is placed in that EAT word.

The word in EAT is formatted as follows:



Pass 1B - Scanning of the Report Section

This subpass performs the initial scan, syntactic check, and encoding of the report section of the Data Division. It is not executed if Phase 1B did not encounter a report section header. It terminates and returns to CONTROL upon encountering the Procedure Division header (or the end of the source program).

Phase 1B accumulates the references made in the report reference file from the report sections, so that Pass 1G can retrieve the description and references, and form the object-time report table.

Those report group clauses that are concerned with data definition and location of report lines are encoded individually and placed in the DNT, just as are the same or similar clauses in other sections of the Data Division.

But, those clauses concerned with the data referenced for reporting and the control of the reporting levels require more complex handling. In addition, the preparation of a report requires the creation of totaling counters, although the report description does little more than imply their presence. Phase 1B creates definitions for these in the same manner as data entries displaying the same required characteristics would require elsewhere in the Data Division.

The report description implies the definition of fields containing the "old" value of control fields. For these, unfortunately, the size and usage may be unknown at Phase 1B time. The "old" value of control fields, however, can only be referenced in control footing lines as a SOURCE item and in USE BEFORE REPORTING of control footing lines. A mechanism in Phase 1G will define the "old" control field and substitutes its location for references made to the control field from control footing lines.

RD Processing

When the RD is encountered, a search through the EAT is performed looking for a report entry pointing to a matching name. This locates the proper hash link. If the report name is not found, a fatal diagnostic is issued. The hash link and the right 18 bits of EAT are set to point to the next available space in the DNT where the report description (RD1, RD2, RD3) will be stored. The report number (the position in the EAT), is placed in RD2. The file number is moved from EAT to RD3. The remainder of the report description is scanned. If CODE is encountered, the alphanumeric literal is located from a SNL item in the DNT and placed in RD3. Also, a one is ORed into bit number 48 of FD + 29 to indicate that a line format with a CODE must be chosen. When the CONTROLS clause is encountered, the referenced names are output onto the report reference file, and a count of the control items is kept in RD3.

For the benefit of Pass 1D, the data location assignment pass, RD2, bits 6-23, always contains the relative location of the next position in the report module which can be assigned. This is used by Pass 1D to assign locations and bypass 1G to determine where the remainder of the report module may be generated.

Next, for each RD, two items, one word in length, are placed in DNT with the names LINE-COUNTER and PAGE-COUNTER, with indication that they are to be assigned locations in the report module. The report items RED1, RED2, RED3, RED4 are never used.

Instead, for items defined in the report section, two bits are added to EDD3. Bit 44 = 1 if the item is to appear in a line image of 140 characters previously set aside for items with "COLUMN NUMBER." When bit 43 = 1, the area for a line image is reserved. If bit 44 is one, a fourth word EDD4 contains in it in bits 0-11, the ordinal number of the 01 report group in which this item is contained. Bits 12-23 contain the COLUMN NUMBER of the start of the field. When EDD4 is used, EDD2 contains the start location of the field. The start location of the line image is obtained by subtracting COLUMN NUMBER.

If bit 44 is zero, then bits 24-29 (the file or section number) are inspected; if it is in working storage, it is assigned to the next available location in working storage; if it is a report section, it is assigned the next available location in that report module.

Report Group and Item Processing

Each report group, subgroup, or elementary item must both describe an item on the print line image and must pass along information to Pass 1G to be used in setting up the tables for the report writer.

Pass 1D, when accomplishing data location assignment, must, in contrast to other Data Division processing, keep track of three location assignments simultaneously. First, all items to be printed (those having a column number clause) must be assigned a space in the current "line image" at the tail end of WORKING STORAGE. Other items must be assigned to a contiguous space either immediately following the "line image" or in the report module. For some items, such as GROUP INDICATE, initial values must be assigned a location in two areas and indicate which of the two location assignments is to be used on each item. Items for the "line image" will be recognized by Pass 1D because a COLUMN number is present for the item. Each time a new "line image" area is to be assigned, an indication is placed in the data name item. Report defined items, as LINE-COUNTER, PAGE-COUNTER, SUM, and the old value of CONTROL items are assigned a place in the report module.

The following special action is taken for items containing the following clauses:

GROUP-INDICATE	If the item contained a VALUE, the VALUE clause is first suppressed and an item is placed in the DNT. Thus, the location for the item can be defined in the line image area, and then the item is output again into the DNT without the COLUMN cluster but with the VALUE clause so that the VALUE itself is created by Pass 1H outside the line image area.

SUM	If the item contains a SUM clause it must be output twice into the DNT, first for the line image if there is a COLUMN clause, but without a name (so that Procedure Division references cannot be made to the edited sum), and second for the sum-counter in the report table area, in which case COLUMN is suppressed and the PICTURE is altered to show no editing. References (contributors to the sum) are output on the RRF in the same manner as control field references in the RD.
RESET	This clause causes references on the RRF in the same manner as control field references in the RD. If the control level is FINAL, this also is indicated in the RRF item.
SOURCE	This clause causes reference clusters on the RRF in the same manner as control field references in the RD.
SUM UPON	This clause causes reference clusters on the RRF in the same manner as control field references in the RD.
LINE NUMBER NEXT GROUP TYPE	Each of these clauses causes indication to be placed in the RRF item. The following cases are categorized: LINE LINE PLUS LINE NEXT PAGE NEXT GROUP NEXT GROUP PLUS NEXT GROUP NEXT PAGE TYPE
TYPE CONTROL	When a type CF or CH is output, it must be cross referenced to the control field. To do this, reference items in the same format as the control field references from CONTROLS will be output to RRF. The FINAL control level requires special indication.
SOURCE SELECTED	Is output on the RRF.
. (period)	At the conclusion of each item on the RRF a marker item is output with the contents of the History register.

Source Copies (Pass 1C)

Source copies are satisfied during Pass 1C by placing a reproduction of the encoded items which are encompassed by the COPY at the end of the DNT. (See Figure 3-18.)

The following processing takes place while reproducing the items:

1. "Level numbers" and "data type" codes are adjusted to reflect the context in which the entry appears on the completed Data File.

2. "Link to next item with same name" are moved from COPY item to copied item and replaced with pointer to copied item.

3. "Link to Dominant Item" are moved from COPY item to copied item.

4. "Link to next item in source sequence" are moved to last item encompassed by COPY and are replaced by link to first item encompassed by COPY.

5. Flags G_1 and E_1 are turned on in each item to indicate that the item is the result of a Source COPY.

6. "Line numbers" of copied items retain their original values.

7. The "File or Section Number" is moved from the COPY item to all of the copied items.

Storage Allocation (Pass 1D)

The four main functions of Pass 1D are:

1. Complete the diagnostic processing which could not be finished in Pass 1B because of the possibility of source copies having to be processed in Pass 1C.

2. Process REDEFINES and RENAMES.

3. Allocate storage to each item.

4. Fill in subscript coefficient for items having an OCCURS clause.

Figure 3-19 shows a flowchart for Pass 1D.

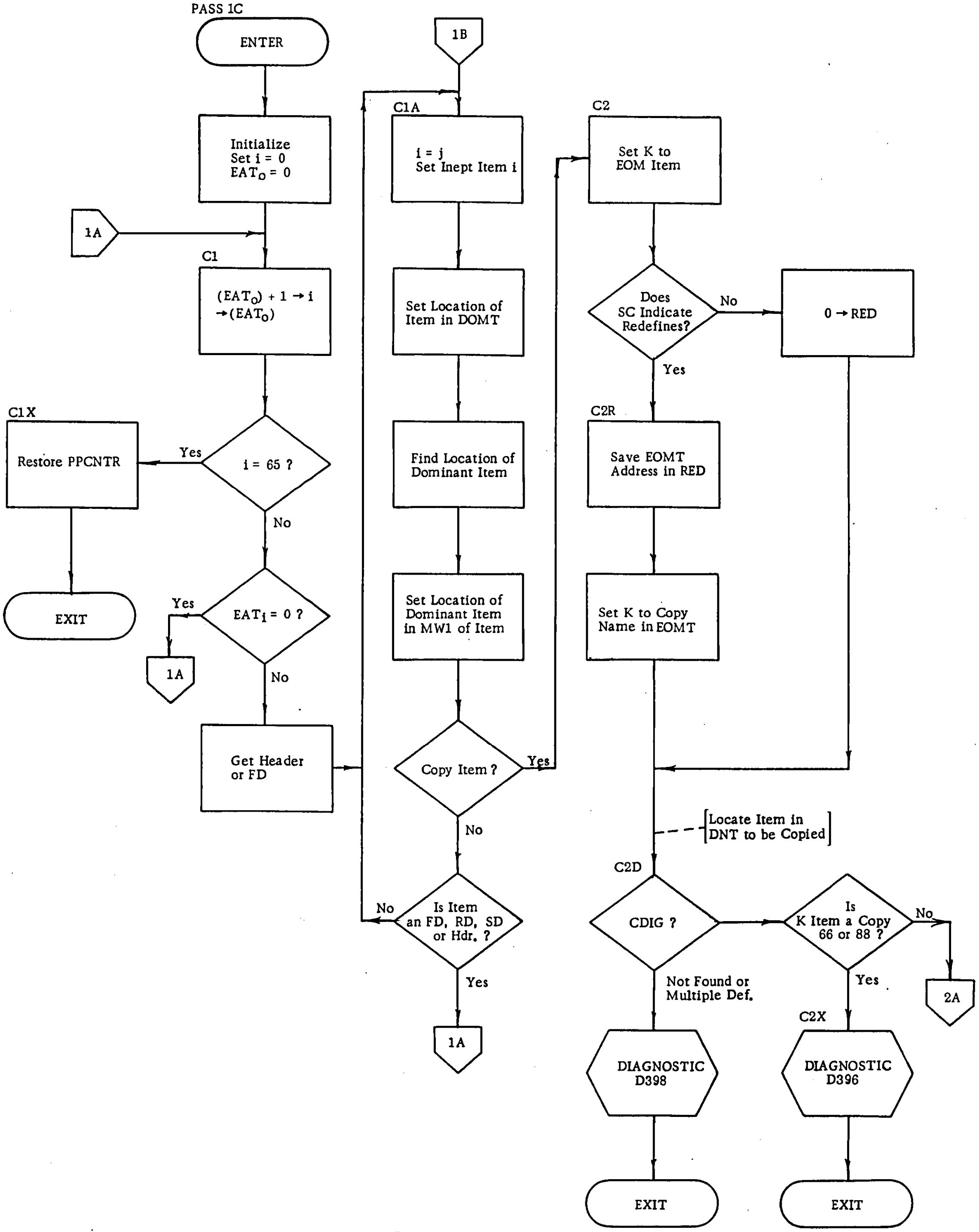


Figure 3-18. Pass 1C Flowchart (1 of 2)

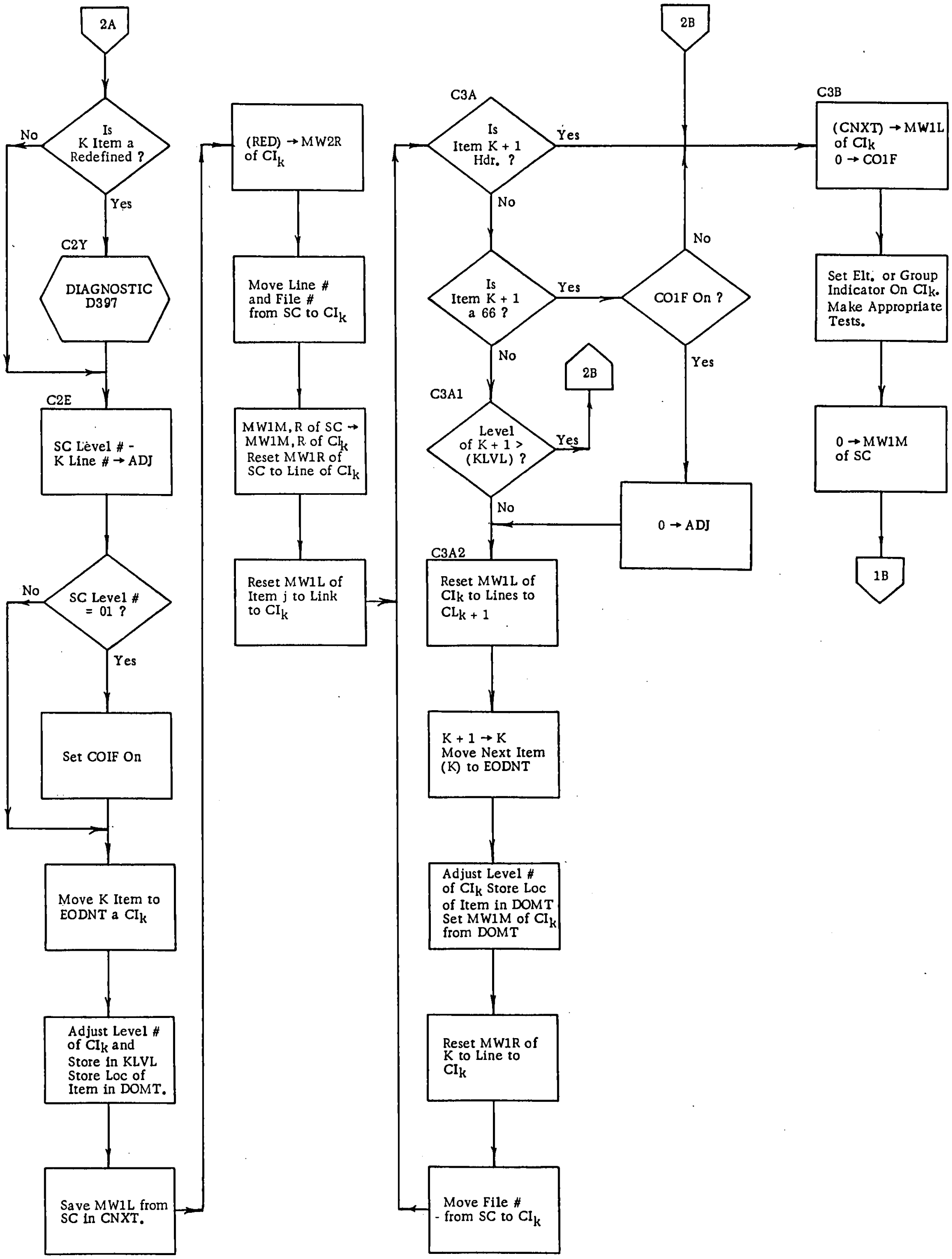
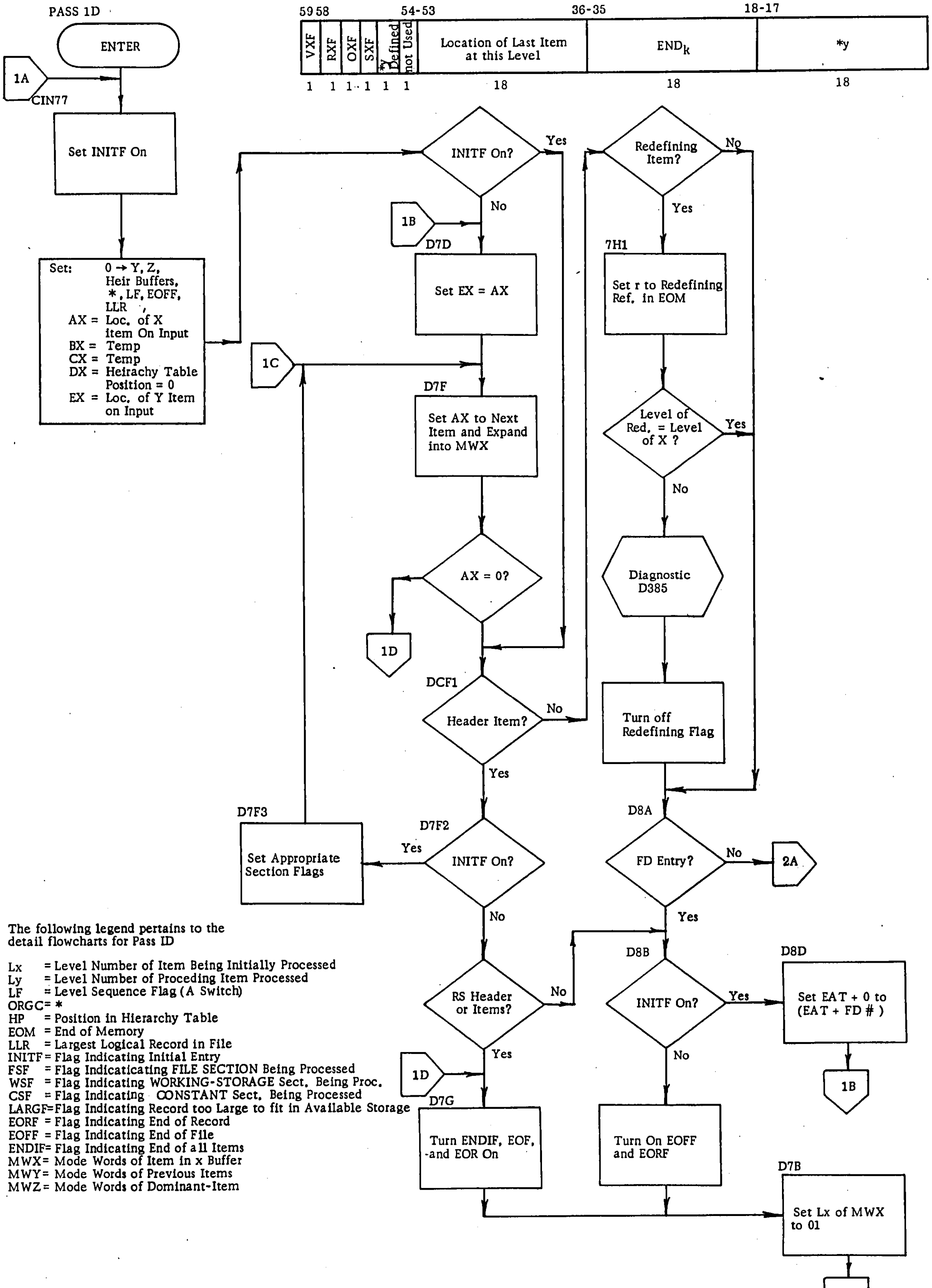


Figure 3-18. Pass 1C Flowchart (2 of 2)



The following legend pertains to the detail flowcharts for Pass 1D

Lx = Level Number of Item Being Initially Processed
Ly = Level Number of Proceding Item Processed
LF = Level Sequence Flag (A Switch)
ORGC = *
HP = Position in Hierarchy Table
EOM = End of Memory
LLR = Largest Logical Record in File
INITF = Flag Indicating Initial Entry
FSF = Flag Indicaticating FILE SECTION Being Processed
WSF = Flag Indicating WORKING-STORAGE Sect. Being Proc.
CSF = Flag Indicating CONSTANT Sect. Being Processed
LARGF = Flag Indicating Record too Large to fit in Available Storage
EORF = Flag Indicating End of Record
EOFF = Flag Indicating End of File
ENDIF = Flag Indicating End of all Items
MWX = Mode Words of Item in x Buffer
MWY = Mode Words of Previous Items
MWZ = Mode Words of Dominant-Item

Figure 3-19. Pass 1D Flowchart (1 of 14)

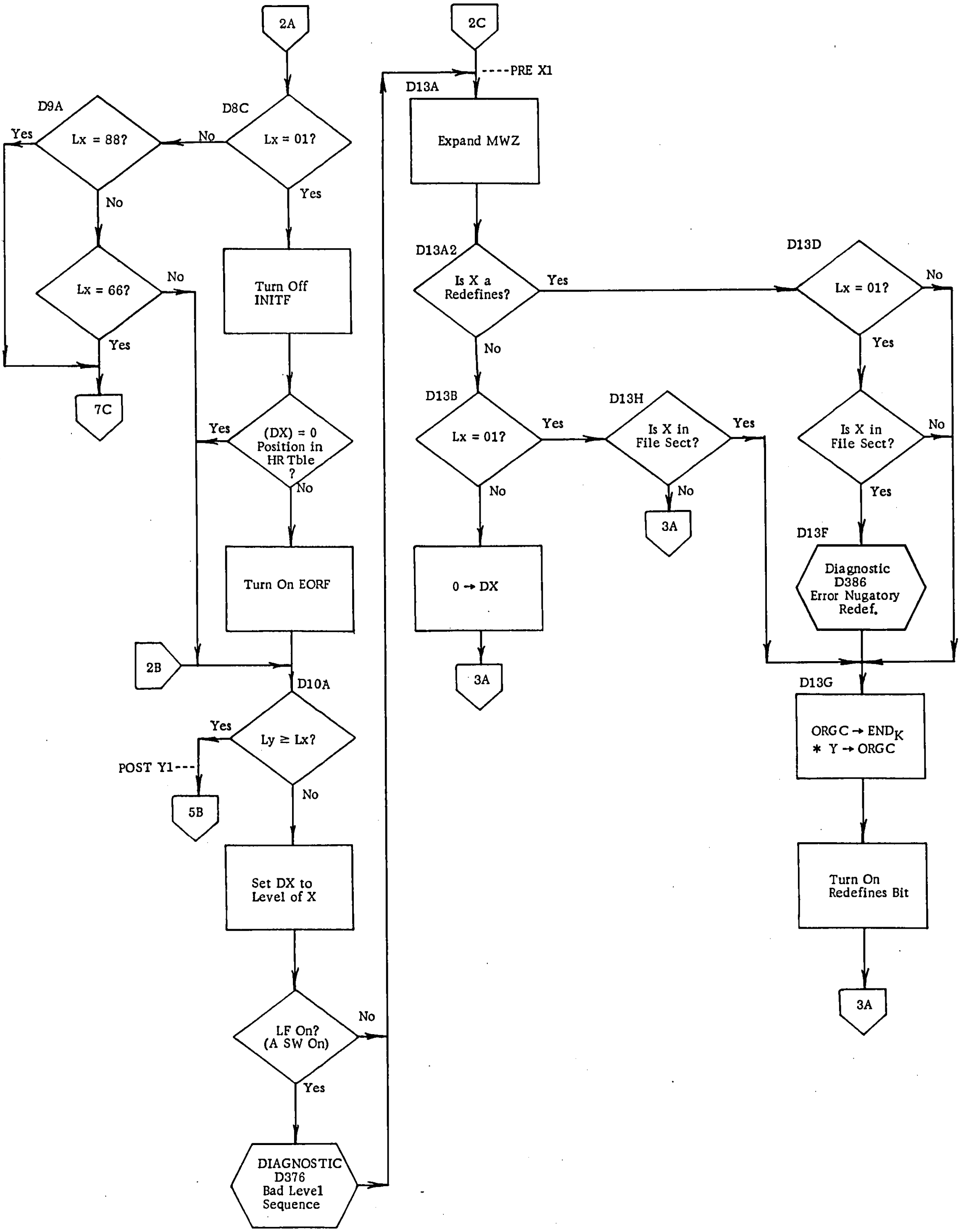


Figure 3-19. Pass 1D Flowchart (2 of 14)

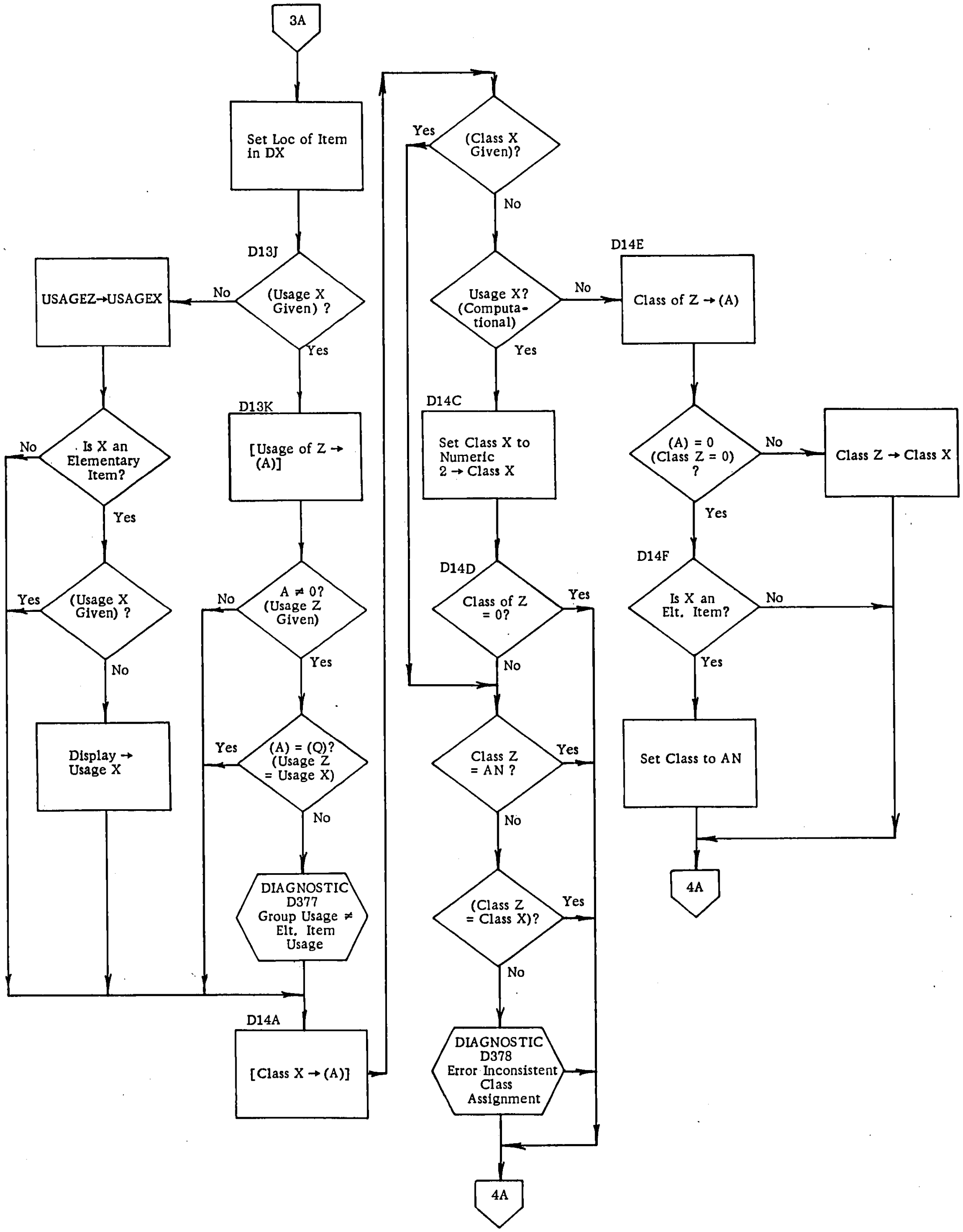


Figure 3-19. Pass 1D Flowchart (3 of 14)

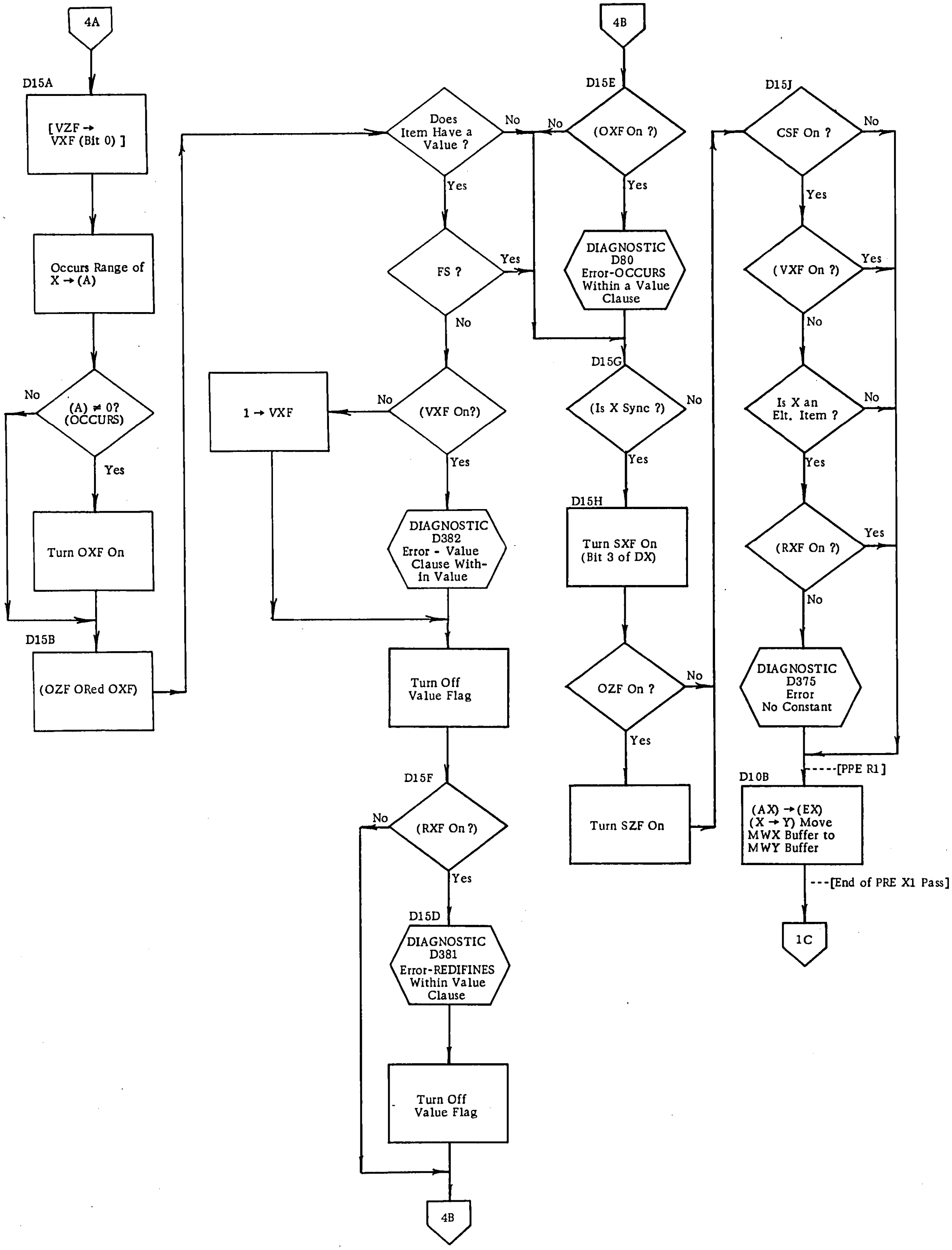


Figure 3-19. Pass 1D Flowchart (4 of 14)

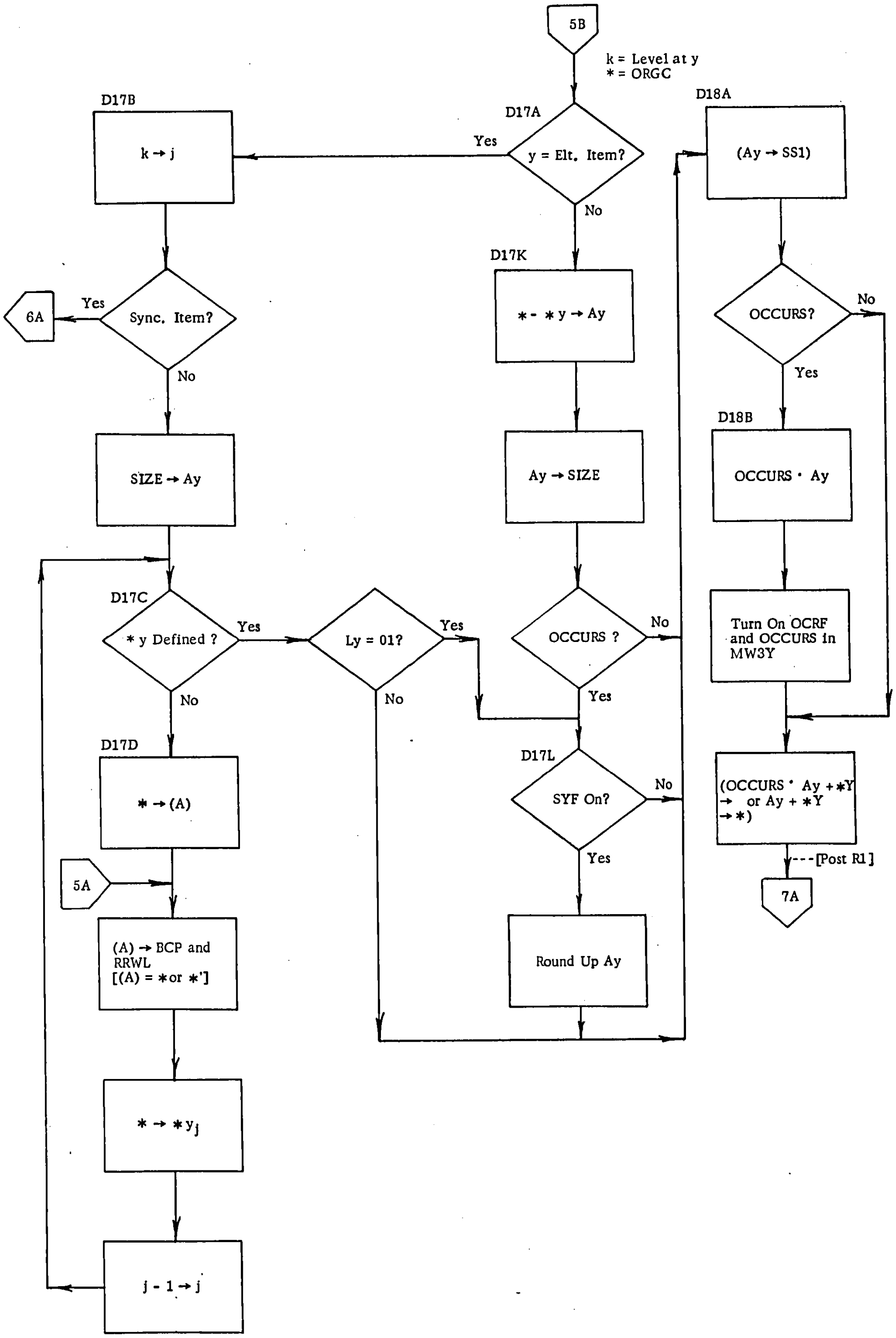


Figure 3-19. Pass 1D Flowchart (5 of 14)

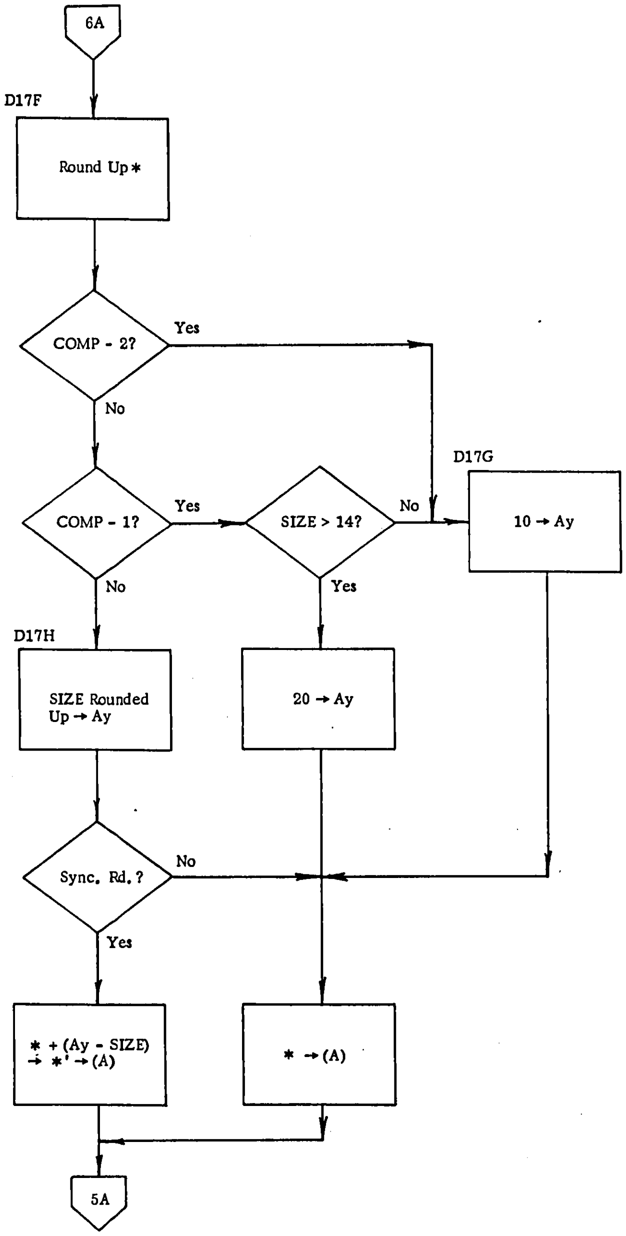


Figure 3-19. Pass 1D Flowchart (6 of 14)

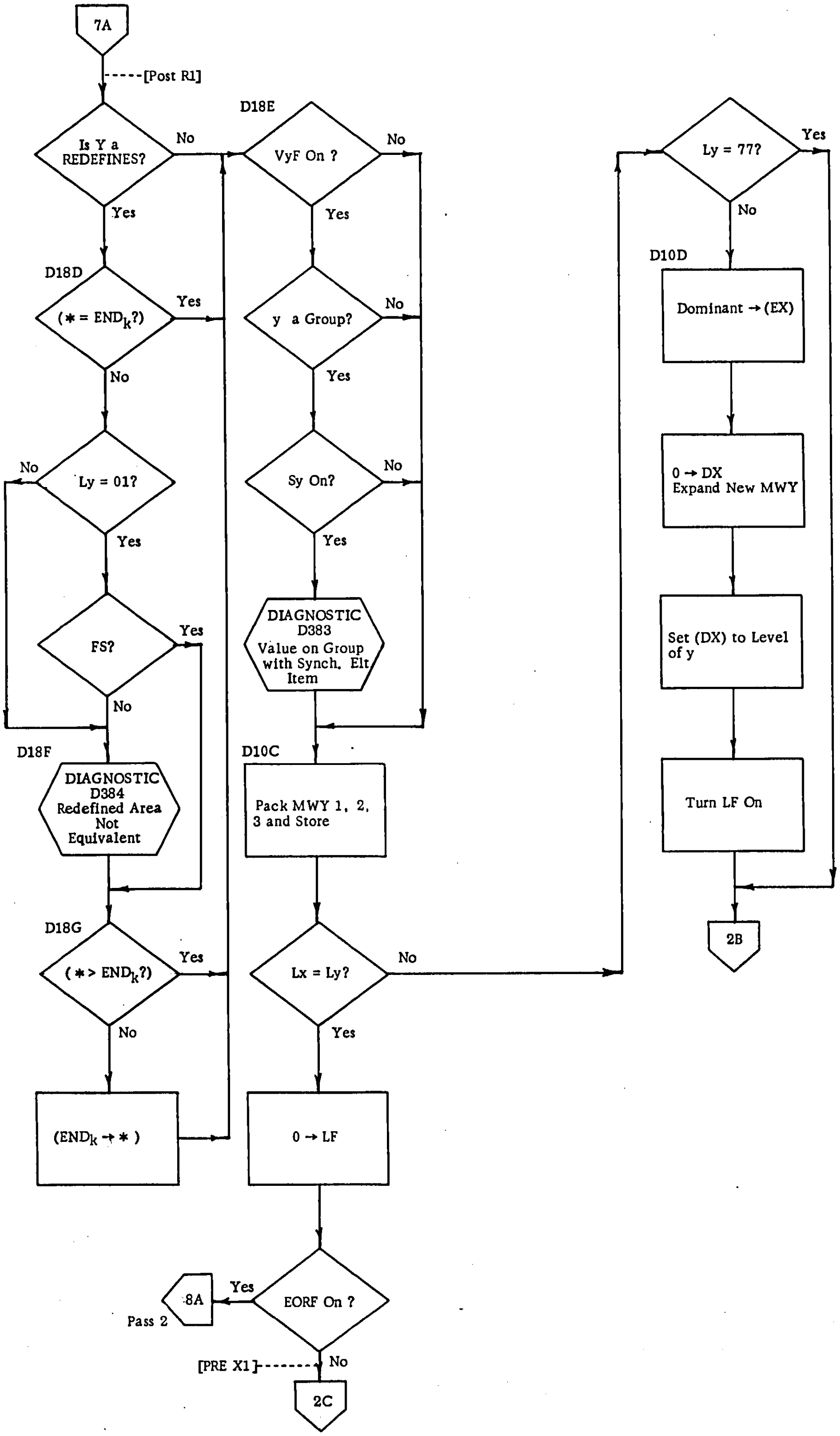


Figure 3-19. Pass 1D Flowchart (7 of 14)

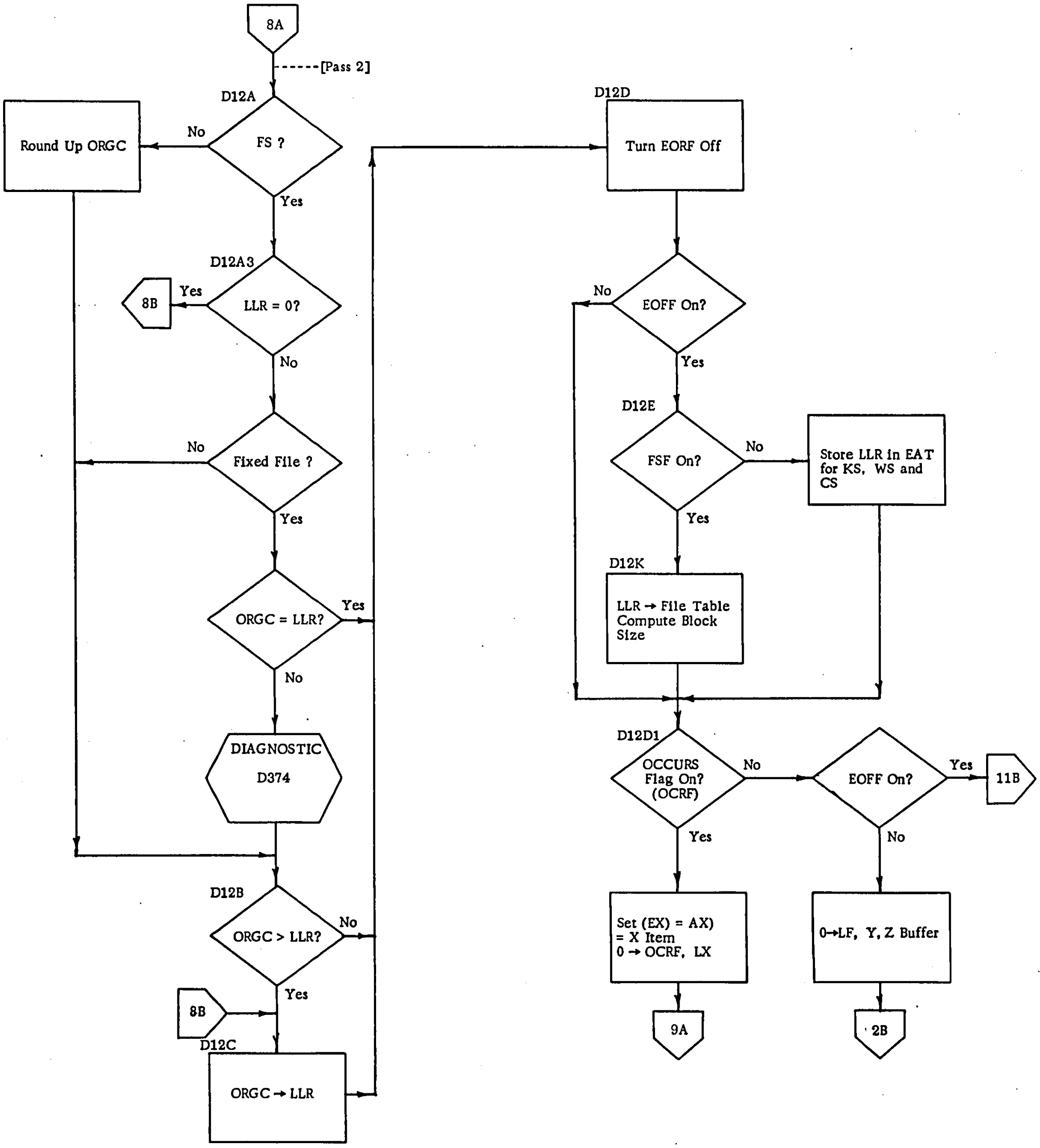


Figure 3-19. Pass 1D Flowchart (8 of 14)

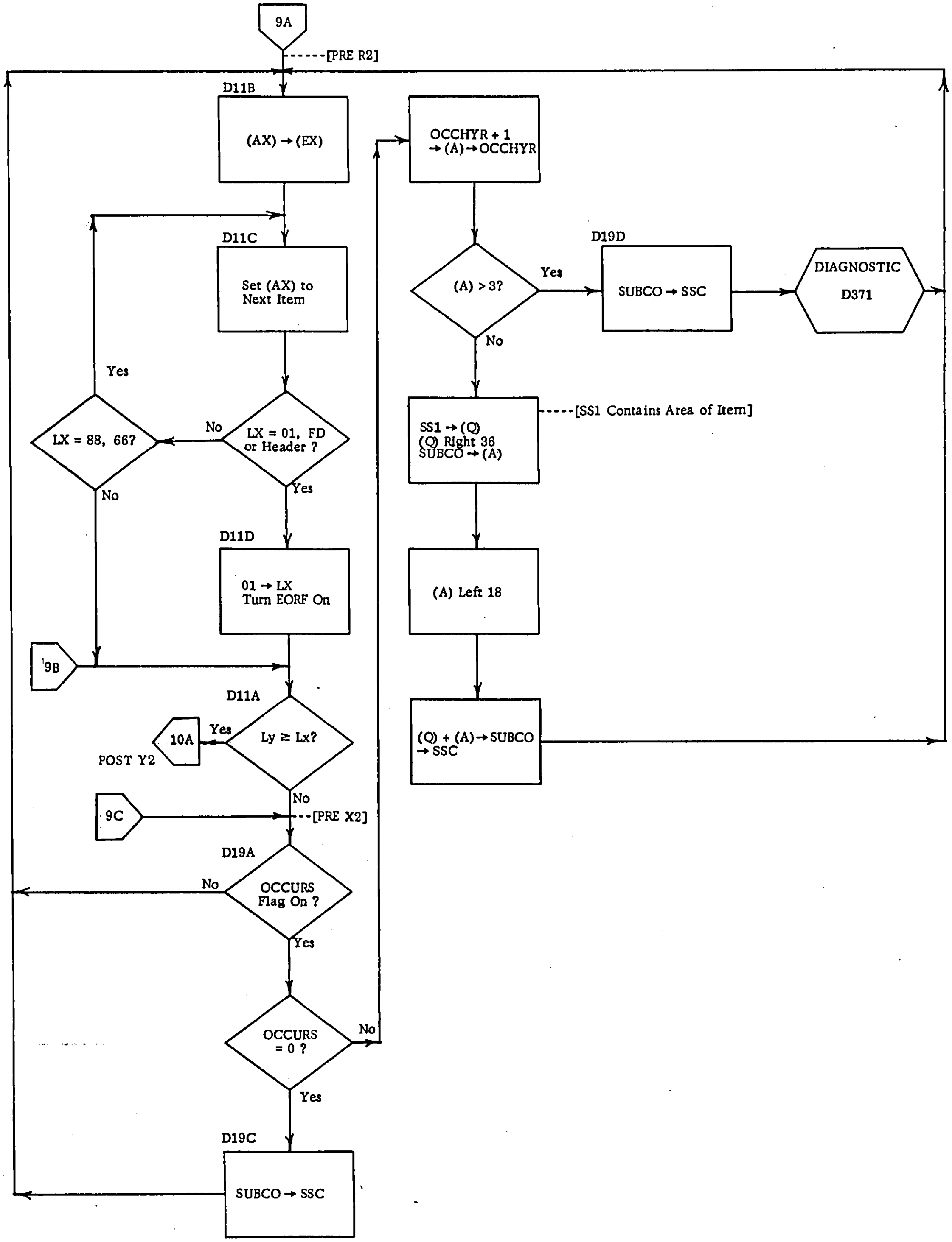


Figure 3-19. Pass 1D Flowchart (9 of 14)

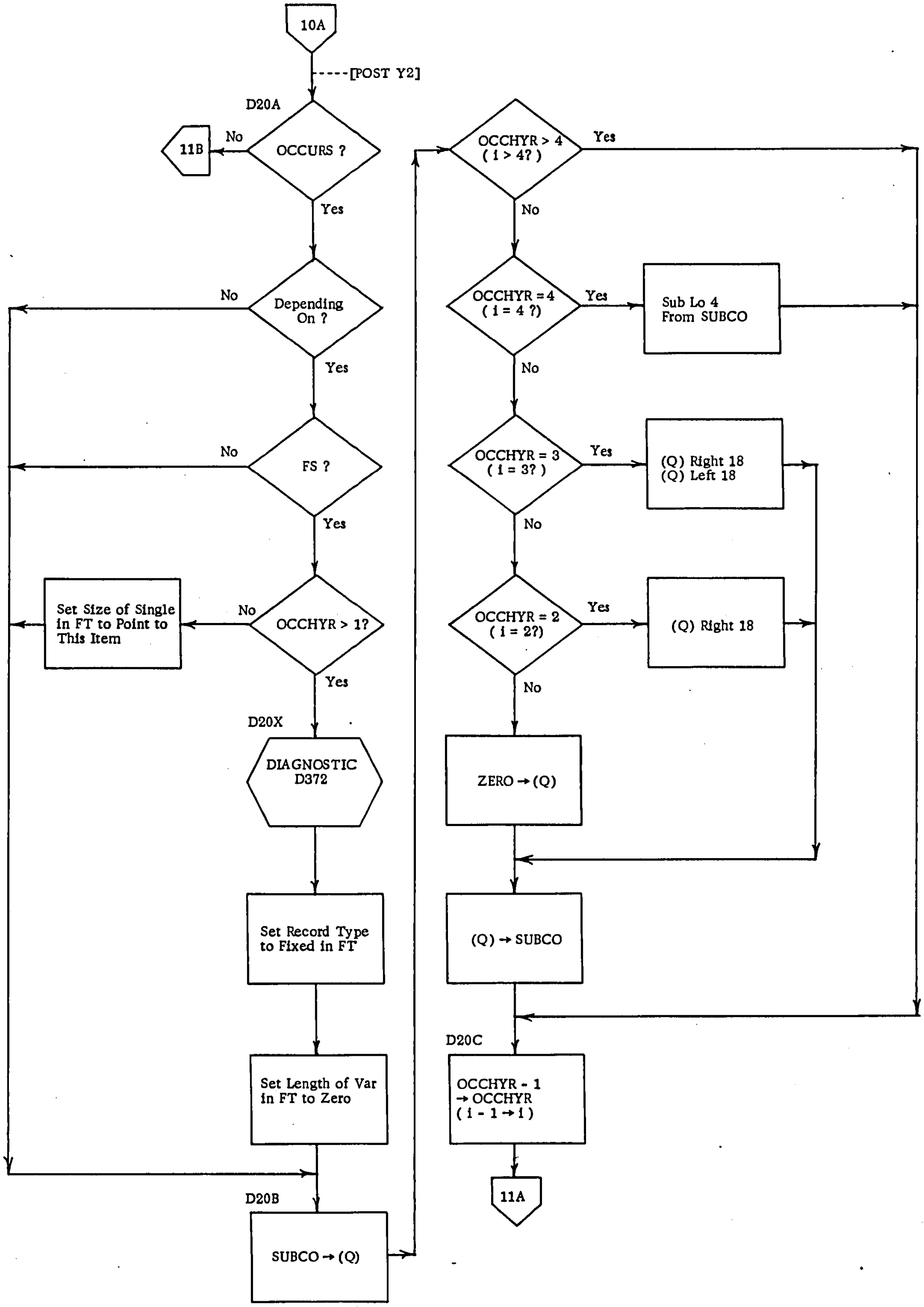


Figure 3-19. Pass 1D Flowchart (10 of 14)

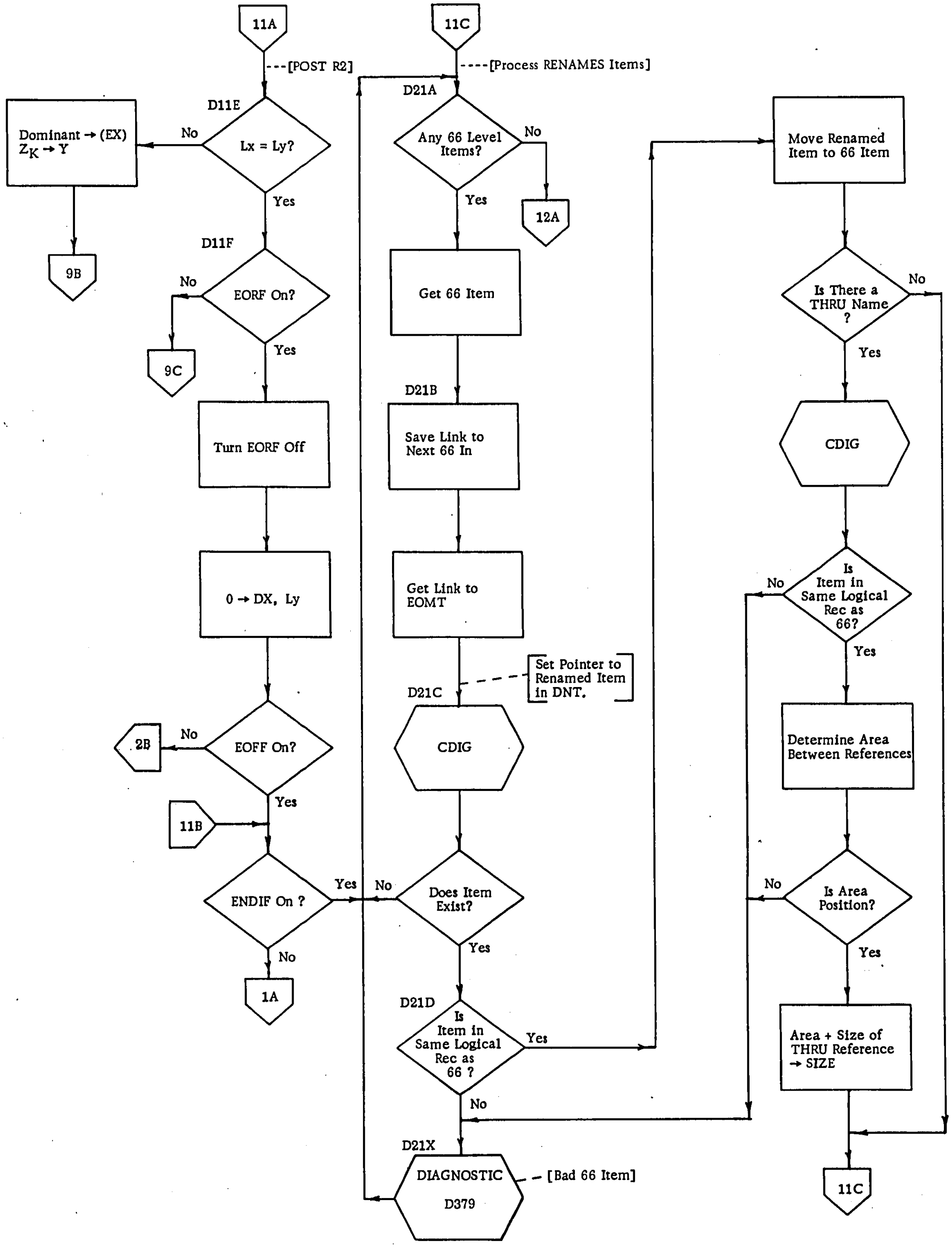


Figure 3-19. Pass 1D Flowchart (11 of 14)

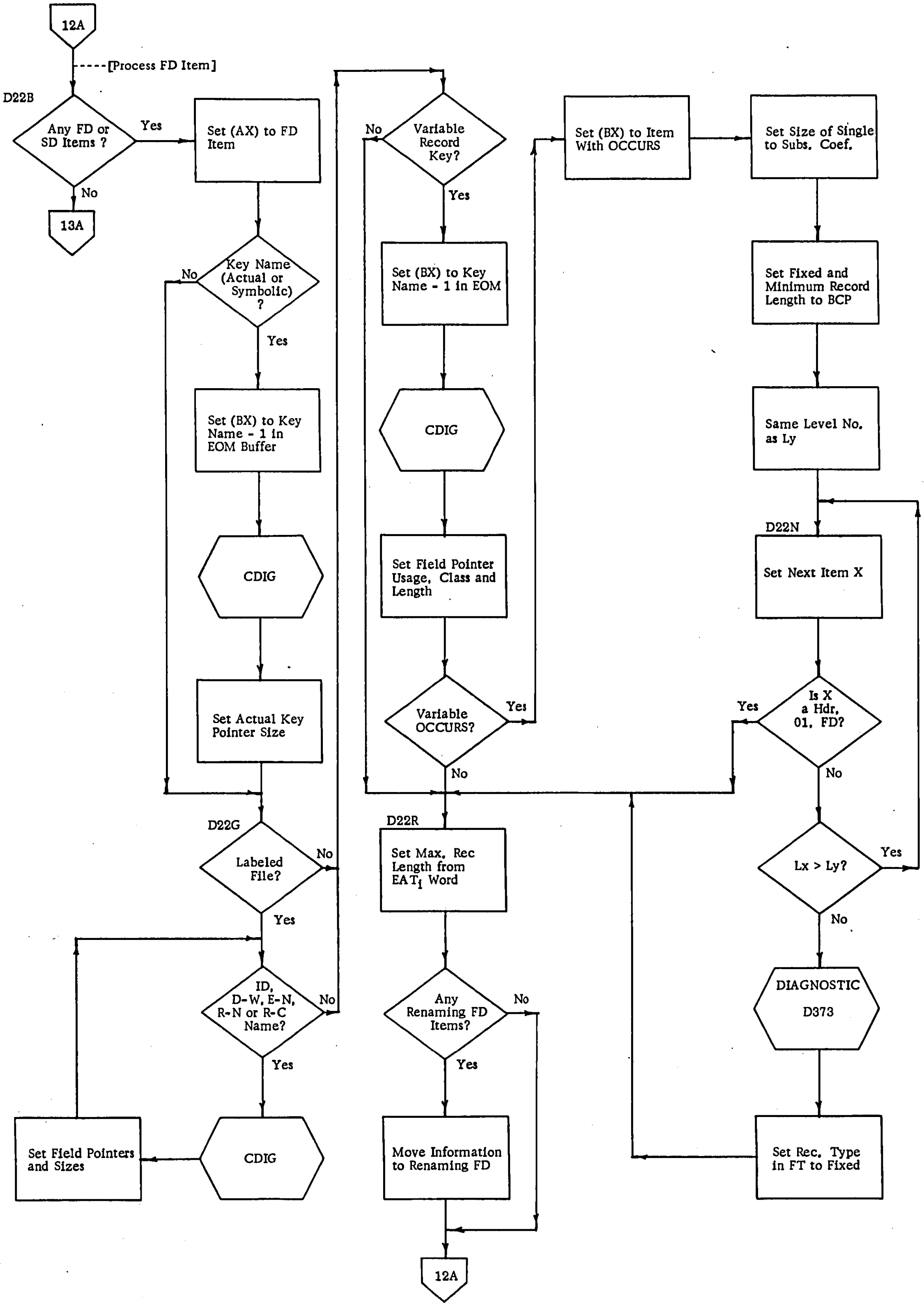


Figure 3-19. Pass 1D Flowchart (12 of 14)

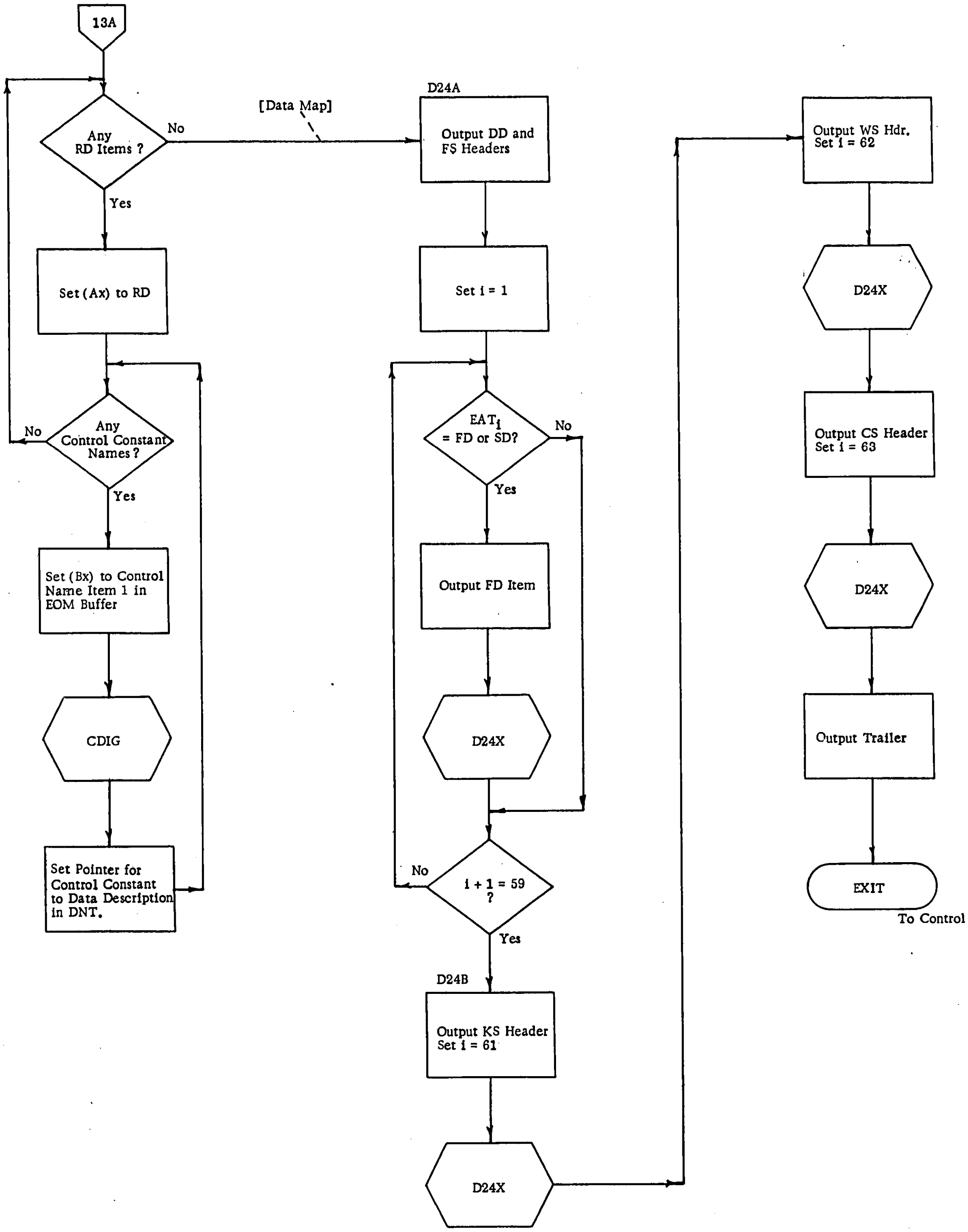


Figure 3-19. Pass 1D Flowchart (13 of 14)

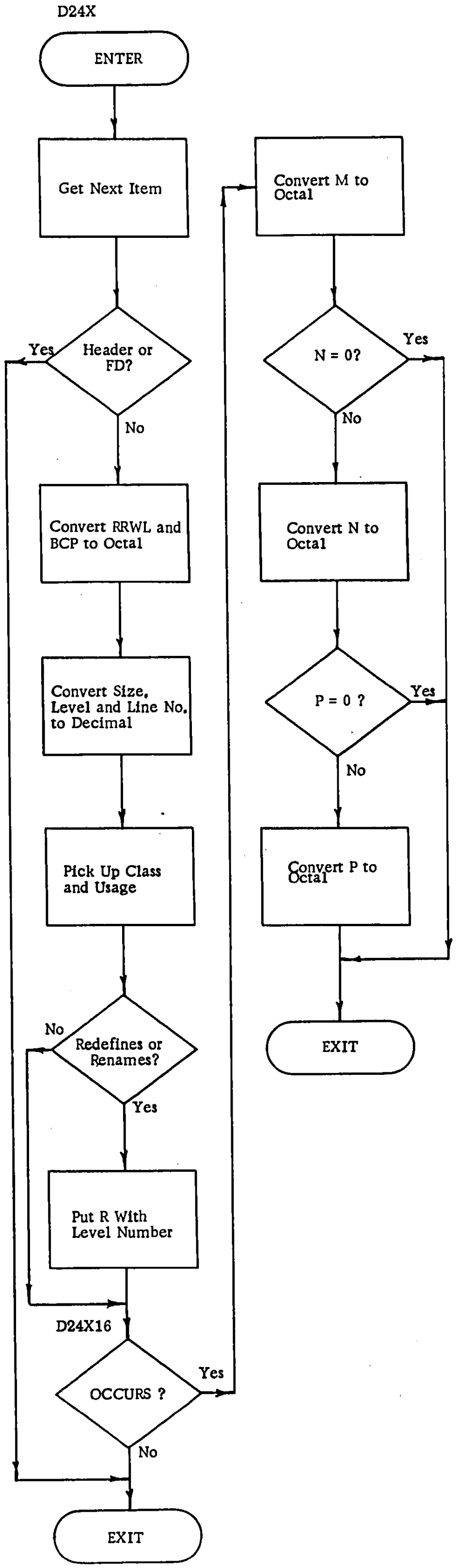


Figure 3-19. Pass 1D Flowchart (14 of 14)

Several scans are made up and down the hierarchy of each logical record. The following list comprises some items checked and determined during the various scans by the use of a supplementary hierarchy table.

1. Consistency of USAGE and CLASS.
2. Correct level number and area for redefining item.
3. Area required for each logical record, group and elementary item.
4. There is no VALUE clause:
   a. Within a VALUE clause.
   b. Within a REDEFINES range.
   c. Within an OCCURS range.
   d. On a report group.
   e. On a non-88 file item.
5. There is a VALUE clause on every elementary item in the CONSTANT SECTION.
6. There is a CLASS and SIZE for every elementary item.
7. Largest logical record determined for each file.
8. After all item areas are determined, fill in the appropriate subscript coefficient (in characters) position, relative beginning word position and beginning character position (BCP) for each.

   BCP has a value from 0 to 9, inclusive, and indicates the first position within the first word that the field occupies. If the item has an OCCURS clause, the BCP indicates the first character position for the first occurrence of the field. A noncontinuous item (level 77) and the first elementary item or the first group of a logical record (level 01) start in the zero character position of the word, unless synchronized right. All logical records of one start in the same word. Logical records (level 01) for all other sections begin in the first full word following the end of the preceding logical record.

TALLY will be a COMPUTATIONAL-1 item automatically generated so that it is available to all COBOL elements running together as a program.

Its size consists of five digits, but the arithmetic rules for COMPUTATIONAL-1 ignore overflow to allow faster object code. (This is not strict adherence to DOD 65 section 3.2.2.6.)

The following items from the report section are checked in Pass 1D or 1H and appropriate diagnostics given:

1. A CONTROL clause must appear in the RD item if any subordinate report group (01) is a TYPE CONTROL HEADING FOOTING or has a RESET clause.
2. The PAGE clause must appear in the RD if any of the TYPEs PH, PF or OH are specified as subordinate report groups.
3. DETAIL, CONTROL HEADING and CONTROL FOOTING are the only TYPE of report groups that can be specified more than once for a report.
4. Cannot have more than one control heading or more than one control footing associated with same CONTROL name.
5. Absolute LINE numbers within a report group (01) must be in ascending order. An absolute LINE number cannot be preceded by a relative LINE number.
6. A LINE number (relative or not) must be less than or equal to the PAGE limit.
7. A LINE number on a group item will be carried down to the elementary items subordinate to it and must not be contradicted by a LINE number on the associated elementary items. There must be a line number on a group or the first elementary item.

Syntax Checking

The Report Writer input is scanned and interpreted by the Report Writer Syntax Tables, which accomplish a certain amount of COBOL syntax rule checking. However, certain of the syntax rules must be checked by other means. To do this, three registers, History, Required, and Not Permitted, are set up to accumulate information obtained during the syntax scan.

The History register has one bit assigned to each clause that has so far appeared in the scan of the syntax. A zero (0) in the assigned bit indicates the clause has not appeared; a one (1) indicates it has appeared.

The Required register, with bit assignments corresponding to the History register has a "1" ORed into it for each clause that is required because of the presence of other clauses.

The Not Permitted register with bit assignments corresponding to the History register has a "1" ORed into it for each clause not permitted because of the presence of other clauses.

As each clause is encountered, its bit position in the History register is checked to make sure that clause has not already appeared. If a COPY clause appears in an RD, the bits in the History register for CONTROL, PAGE, and for COPY are checked, since a COPY clause is not permitted after any of the above clauses.

At the conclusion of the description of each item, the Required and the Not Permitted registers are checked against the History register, producing diagnostic messages if required and that portion of the History register corresponding to elementary items is cleared. Then, if by inspection of the next level number, the current item is a group item, that portion of the History register corresponding to group items is also cleared.

At the beginning of each item, the Required and Not Permitted registers are cleared, and that portion of the History register corresponding to the 01 level is cleared if an 01 item is encountered.

After the elementary portion of History is cleared, the LINE NUMBER bit of the 01 level portion is copied into the LINE NUMBER position of the elementary portion. If a LINE NUMBER appears in an 01 level item, the bit is set in the 01 level portion. Otherwise, it is set in the elementary portion. An additional test must be made after the item is finished to check for items subordinate to SOURCE SELECTED. If the item is elementary, then at least one of SOURCE SELECTED, SOURCE, SUM, VALUE must be ON.

The PICTURE, if present, is procured after the item is finished and must be numeric if BLANK ZERO is set.

Bit Assignment of the History Register

RD Portion

0	CODE CLAUSE	
1	COPY CLAUSE	
2	CONTROL CLAUSE	
3	PAGE LIMITS	
4	HEADING SUB-CLAUSE	
5	FIRST DETAIL SUB-CLAUSE	
6	LAST DETAIL SUB-CLAUSE	
7	FOOTING SUB-CLAUSE	
8	RH	
9	RF	
10	OH	Only one is permitted per report.
11	OV	
12	PH	
13	PF	

01 Level Portion

14	LINE NUMBER
15	NEXT GROUP
16	TYPE
17	DE
18	CF

Group Portion

19	01 level
20	SOURCE SELECTED

Elementary Portion

21	elementary item
22	COLUMN
23	GROUP INDICATE
24	PICTURE
25	BLANK ZERO
26	JUSTIFIED
27	RESET
28	SOURCE
29	SUM
30	VALUE
31	LINE NUMBER
32	EDITING
33	POINT
34	SIGNED
35	SIZE
36	CLASS
37	USAGE

Clauses Required or Not Permitted

Table 3-9 describes the clauses required when another clause appears and changes not permitted when another clause exists.

PASS 1D PROCESSING OF RENAMES

Names that are the object of the RENAMES clause (level 66) were stored in the DNT in the normal order followed by the qualifiers. Both sets of names are followed by the name of the logical record (01) with which it is associated, and these are followed by a word consisting of zeros. The size of the RENAMES item is determined by subtracting the relative beginning character position (BCP) of the first name from the relative BCP of the item following the THRU name. The class, level number, usage, and item type are taken from the first item named. No check (such as there being an item with a higher level than the first item named) will be made on the RENAMES range. The names defining the range is checked to assure that they are defined in the associated record.

Table 3-9. Clause-Change Table

If this clause appears in the current item,	these clauses are required,	and these clauses are not permitted.	Remarks
OH	LAST-DETAIL		These required clauses are tested when the clause is encountered rather than via the "REQUIRED" register.
OV	LAST-DETAIL		
PH	PAGE LIMITS		
PF	PAGE LIMITS		
LINE NUMBER integer	PAGE LIMITS		
LINE NUMBER NEXT PAGE	PAGE LIMITS		
NEXT GROUP PLUS	01 level		
NEXT GROUP integer	01 level, PAGE LIMITS		
NEXT GROUP NEXT PAGE	01 level, PAGE LIMITS		
DE	group-name		
CF	CONTROL		
CH	CONTROL		
01 level	TYPE		
SOURCE SELECTED		elementary item	
elementary item	PICTURE*		
COLUMN	elementary item PICTURE*, LINE at 01 or in hierarchy		
GROUP INDICATE	DE, elementary item, COLUMN, PICTURE*		One of CONTROL or PAGE is required. (Tested when the clause is encountered.)
PICTURE	elementary item	EDITING, POINT, SIGNED	
BLANK ZERO	elementary item, PICTURE*		
JUSTIFIED	elementary item, PICTURE*		
RESET	CONTROL, CF, SUM, elementary item, PICTURE*		
SOURCE	elementary item, PICTURE*	VALUE, SUM	
SUM	elementary item, CF, PICTURE*	VALUE, SOURCE	
VALUE	elementary item, PICTURE* if level is not 01	SUM, SOURCE TYPE, NEXT GROUP	
Editing	elementary item		
Point	elementary item		
Signed	elementary item		
Size			
Class			
Usage			

---

*In place of PICTURE, SIZE, POINT, CLASS, and USAGE may be substituted.

PROCEDURE DIVISION PROCESSING - PASS1E

PROCEDURE DIVISION SYNTAX

The Procedure Division is analyzed by means of a syntax table.

The parts of this table are equivalent to subroutines. Thus, the structure can be represented by flow networks.

The Procedure Division syntax can, for the most part, be represented by two major networks.

The first we well call the Procedure Network, because it links together the procedures (sections, paragraphs, sentences, and individual statements).

The second we will call the Reference Network, because it represents the data (operands) required by the individual statements.

The two networks cross connect at different places according to the requirements of the various statements (or verbs).

Figures 3-20 and 3-21 show this structure, omitting, of course, much detail.

Superimposed (and not shown) on this structure are error recovery routines. These routines operate by skipping forward after an error on the source cards to some point where a recognizable reserved word appears, and then returning control to the routine that called the routine that discovered the error. In this way, recovery is accomplished in an orderly manner.

Of particular interest are two loops shown, one in each network. The presence of these loops requires the syntax driver to be reentrant.

The loop shown in the Procedure Network is required because conditional statements may include additional imperative or conditional statements in their conditional branches. An example is nested IF statements.

The other loop, shown in the Reference Network, is required to simulate inclusion represented by parentheses. Because parentheses are allowed in both compound conditions and formulas, and because formulas are allowed in conditions, it is necessary to include them both in one grand loop. The part of the loop having to do with formulas is effective for COMPUTE statements only.

Thus, nested IFs are handled by the loop in the Procedure Network and may contain compound conditions and formulas with precedence of operations indicated by parentheses, which are handled by the loop in the Reference Network.

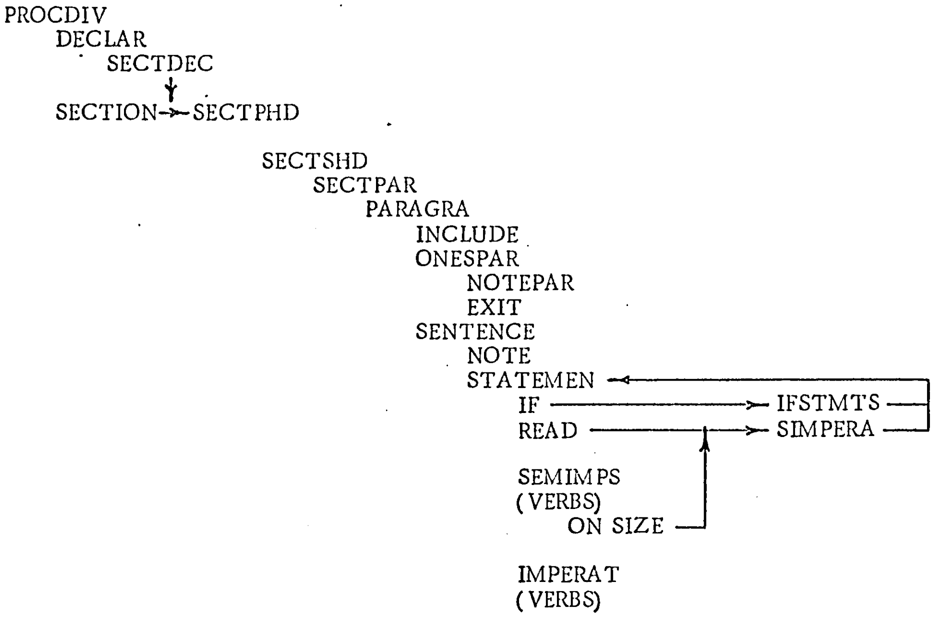


Figure 3-20. The Procedure Network

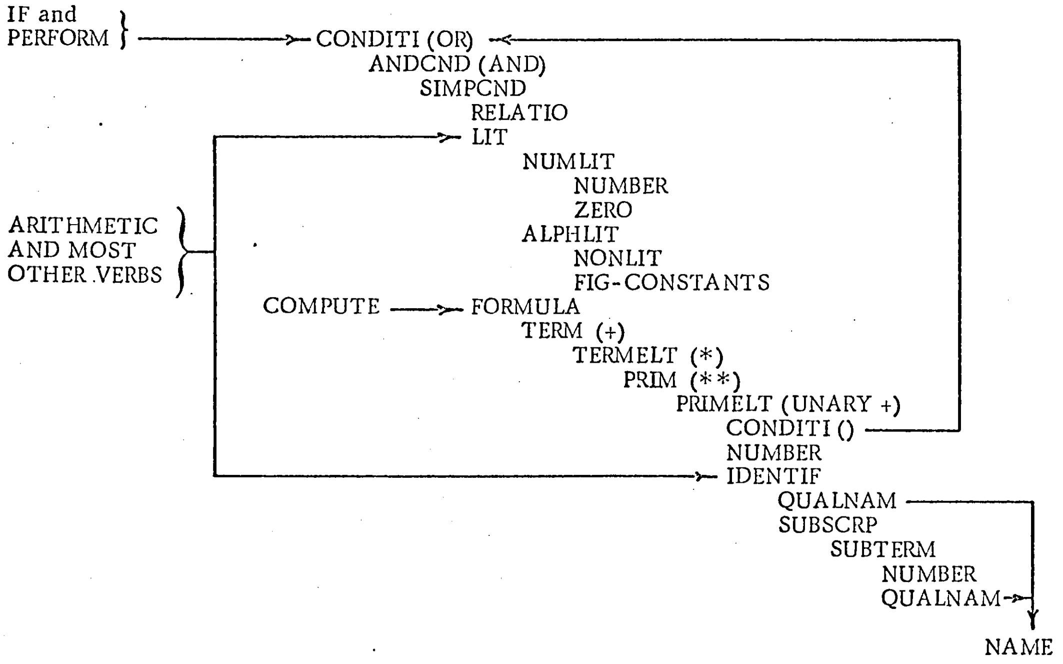


Figure 3-21. The Reference Network

Analysis of the Procedure Division begins at AAAAAA, which initializes all routines at the start and closes out all routines at the end of the Procedure Division. Then PROCDIV locates the DECLARATIVES SECTION or individual sections of the Procedure Division.

SECTPHD in turn attempts to locate and process one procedure header and the procedure.

SECTSHD determines whether the procedure is a section or a paragraph.

SECTPAR then processes the paragraphs belonging to a section one by one.

PARAGRA processes one paragraph at a time.

SENTENCE processes one sentence until a period if found.

STATMEN finds and processes the next verbs, selecting the appropriate verb routine for analysis of the verb syntax.

PASS 1E

Control during Pass 1E resides in a Syntax Analysis Driver (SAD). SAD makes decisions about which subroutines to enter and in which order to enter them based upon entries in the syntax analysis table.

The encoded syntax analysis table is contained in the element SYNTABL. All routines for building the trees are in the element Pass 1E. In addition the routines DIG, MIG, DIP and RIP are used.

HANDLING OF SEQUENCE CONTROL VERBS

For each reference or definition of a procedure name, the "Tree" routines in the first pass call one of two table routines to enter information into the Procedure Name Table (PNT). For each new procedure of the program, the routine DIP (Definition Indexing for Procedures) is called to enter the name in the PNT and to analyze previously encountered references to this name and to set their type. For each reference of the program to another procedure (ENTRY, ALTER, GOTO, or PERFORM statements), the routine RIP (Reference Indexing of Procedures) is called to enter the reference in the PNT and to determine the reference type. If the referenced name is not yet defined in the PNT, this determination is completed by DIP when the name is defined. A temporary flag is set in the PNT if a name has not been defined.

The following method is used to cross-reference procedures: If the procedure referenced is in the same overlay as the calling procedure or is in the main routine (i.e., always in core), a simple reference can be made. If the procedure referenced is in an overlay other than the one of the calling procedure, the overlay must be loaded before the reference can be made. The compiler generates a word ("index") in the main routine containing the overlay number and entry point location for each procedure name thus referenced in any overlay. The routine SOL exists in the COBOL Library, and is called by referencing routines. It interrogates "indexes," calls the system loader to load the overlay, and exits to the overlay's entry point designated by the "index." DIP and RIP assign these indexes. SOL is described in Section 6.

Code generated for ALTER and PERFORM statements changes the values of indexes in the index table at main level.

Procedures referenced by ENTRY statements may be objects of any number of types of references from other programs; therefore, indexes are generated for them regardless of their use within the program being compiled. Exits from these procedures are made via these indexes to provide return capabilities to other programs. A jump table of entry points is furnished in the base overlay for entry from other programs.

The main level Loading Index Table (LIT) contains all needed indexes and all entry points. The latter are in the table at the point that corresponds to the index number assigned to it. Entries from external routines are made to the location following the return index (which is the defined entry point).

Each overlay has a table of jumps at the beginning of the overlay. Each jump is to a particular procedure. Indexes contain the location in the overlay of the jump (and not the procedure itself). Jump table references in overlays are assigned by DIP and RIP.

TREE PROCESSING, TREE REPRESENTATION

During Pass 1E, the COBOL compiler arranges the information it finds into trees (or branches). Branches are represented by a left link (the right 24 bits of a word representing the left branch) and a right link (the next 24 bits of the word representing the right branch). These links point "up" the tree. Usually the branching indicates dependence of one operation upon the completion of some prior operation. For such cases, tree operations indicated at the tips of the branches must be completed first (left link before right link) in order to enable operations farther down the branch to be completed. For the true-false task generated by the IF and some other statements, the branches represent flow "up" the tree, with the left link being the true path and the right link being the false path. Also, a "master" operation is completed prior to its right link.

A tree is generated by every executable source code sentence and by special beginning and end signals.

Entries in the tree will be of the following format:

Node	▨	Type Right Link	Address Right Link	Type Left Link	Address Right Link
59 51	47	42	41 24	23 18	17 0

Types of Operations (Nodes)

A node represents a branch fork and may be:

1. A master verb: master verb nodes are left-linked to operation nodes and/or operands and right-linked to the next master verb node, a true-false fork, or end-of-sentence.

2. A true-false fork: true-false fork nodes are left-linked to the true path and right-linked to the false path. Both true and false paths may consist of master verbs and possibly subsequent true-false forks.

3. An operation to be performed: operation nodes are left- and right-linked to operands, other operation nodes or special link types.

4. Special beginning and end of declaratives, divisions, sections, and paragraphs: special nodes form a tree by themselves and have no right links. Left links are either nonexistent or, for sections and paragraphs, represent the current procedure--name PNT entry.

5. Special node designating line number of source statement: a line number node precedes each master verb node and is line-linked to the line number and right-linked to the master verb.

Types of Links

Link types are indicators that describe the type of data associated with the link address. These types may be:

1. A pointer to another operation node within the tree.

2. A pointer to operand information within the three such as data from the DNT or description of Procedure Division literals.

3. A pointer to an entry in the PNT.

4. A temporary cell number.

5. A special language indicator for end-of-sentence or NEXT SENTENCE.

6. An indicator that the link address describes a Procedure Division figurative constant or switch status indicator.

7. An indicator specifying an "AT END" diagnostic link.

Link Addresses

Link addresses to entries within the tree are relative to the first location of the tree.

Link addresses to entries within the PNT are absolute.

Descriptors of figurative constants or switch status indicators contained within link address are described under "Operand Information within the TREE."

Temporary cell numbers are integers indicating specific object time conditions. Five such conditions are recognized by Pass 1E:

1. An intermediate result that is used more than once.
2. A subscripted address that is needed more than once.
3. The subject of a conditional relation that is used as an implied subject.
4. The object of a conditional relation that is used as an implied object.
5. A condition-name variable that is compared to more than one value.

Link addresses of NEXT SENTENCE indicators on the eighth branch are tree addresses of the governing conditional fork. NEXT SENTENCE indicators on the sixth branch have no link addresses. Link address for EOS is the length of the entire tree.

Line number of the source statement for special line number node.

Field Definition

General Nodes

000	BR	Branch interpretively in tree.
001	HOLD	Save intermediate.
002	GRAB	Restore intermediate.
003	SUBSCR	Subscript.
004	COMMA*	Subscript, multiples as in assigned 00 to etc.
005	SECONDARY**	TO in move, TO PROCEED TO in ALTER.
006		
007		

Basic Operators

Arithmetic

010	+	(PLUS)
011	-	(MINUS)
012	x	(MULTIPLIED BY)
013	/	(DIVIDED BY)
014	**	(EXPONENTIATED BY)
015	ST	(REPLACE) (GIVING, implied, FROM) =
016	ST	ROUNDED
017	/	(DIVIDE INTO)
020	-	(SUBTRACT FROM)

(Unary minus is specified by type 020 with null right link.)

Relational	021	EQ	(=, EQUAL TO, EQUALS)
	022	NQ	(≠, NOT EQUAL TO, NOT EQ)
	023	GR	(>, GREATER THAN, EXCEEDS, NLQ)
	024	LS	(<, LESS THAN, NGQ)
	025	LQ	(LESS-EQUAL TO, ≤, NGR)
	026	GQ	(GREATER-EQUAL TO, ≥, NLS)

---

*COMMA includes:

REFERENCING ENTER DISPLAY
INITIATE

**SECONDARY includes:

TO PROCEED TO ALTER ACCEPT
BY EXAMINE
TO MOVE
THRU SORT
UPON DISPLAY

	027		NUMERIC
	030		ALPHABETIC
Conditional	031		POSITIVE
	032		NEGATIVE
	033		ZERO
	034		NOT POSITIVE
	035		NOT NEGATIVE
	036		NOT ZERO
	037		NOT NUMERIC
	040		NOT ALPHABETIC
Logical	041		NOT
	042		AND
	043		OR
Forks	050		IF FORK
	051		OSE FORK
	052		DEPENDING ON
	053		AT END FORK
	054		INVALID KEY FORK
Perform	110		THRU
	111		UNTIL
	112		FROM
	113		BY
	114		VARYING
	115		TIMES
	116		AFTER
Examine	120	S	REPLACING
	121	S	REPLACING ALL
	122	S	REPLACING LEADING
	123	S	REPLACING FIRST
	127		REPLACING UNTIL FIRST
	130		
	131		TALLYING ALL
	132		TALLYING LEADING
	137		TALLYING UNTIL FIRST
I/O SORT	142	S	OUTPUT
	143	S	INPUT
	144		ASCENDING
	145		DESCENDING
	146		BEFORE
	147		AFTER
	150		USING
	151		GIVING

MASTERS	400		LINE NUMBER
	401		BEGINNING OF NON-DECLARATIVE PROCEDURES
	402	END OF PROCEDURES	
	403	BEGINNING OF SECTION	
	404	END OF SECTION	
	405	BEGINNING OF PARAGRAPH	
	406	END OF PARAGRAPH	
	407	BEGINNING OF DECLARATIVES	
Arithmetic	410		ADD
	411		COMPUTE
	412		DIVIDE
	413		MULTIPLY
	414		SUBRACT
With Fork	430		ADD OSE
	431		COMPUTE OSE
	432		DIVIDE OSE
	433		MULTIPLY OSE
	434		SUBTRACT OSE
	435		IF
Other	440		EXAMINE
	441		PERFORM
	442		SORT
Masters that are also operators	610		INITIATE (Left link of zeros implies ALL)
	612		TERMINATE (Left link of zeros implies ALL)
	617		STOP RUN
	620	S	ACCEPT
	621	CS	DISPLAY
	622		ENTER
	623		EXIT
	624		GENERATE
	625		RELEASE
	626		SEEK
	627		STOP LITERAL
Close	630		CLOSE
	631		CLOSE REEL
	632		CLOSE WITH NO REWIND
	633		CLOSE REEL WITH NO REWIND
	634		CLOSE WITH LOCK
	635		CLOSE REEL WITH LOCK
Open	640		OPEN INPUT
	641		OPEN OUTPUT
	642		OPEN INPUT WITH NO REWIND
	643		OPEN OUTPUT WITH NO REWIND
	644		OPEN I/O
	646		OPEN INPUT REVERSED

Perhaps followed by FORK	650	READ
	651	RETURN
	652	WRITE
Perhaps followed by SECONDARY	660	ALTER
	661	GO TO (Left link of zeros implies alterable GO TO)
	662	MOVE
	663	IMPLIED GO TO

Abbreviations

S - may (or must) be followed by SECONDARY node
C - may be followed by arguments (i. e., comma)

Link Type	00	OPERATION OR LINE NUMBER NODE
	01	DNT INFORMATION
	02	PNT ENTRY
	03	PROCEDURE DIVISION LITERAL
	04	PROCEDURE DIVISION FIGURATIVE CONSTANT
	05	END-OF-SENTENCE
	06	NEXT SENTENCE
	07	SWITCH STATUS INDICATOR
	10	TEMPORARY CELL
	11	AT-END DIAGNOSTIC LINK
	12	NODE POINTED TO IS A SUBSCRIPTING NODE
	13	NULL LEAF (ADDRESS IS 000000)

Declarative Section Processing

Sections under the DECLARATIVE SECTION are analyzed under the assumption that object-time I/O subroutines will call them as subroutines using the initial JUMP table to obtain an address. DECLARATIVE routines have an entry point on a word preceding their first instruction and terminate by a jump to that entry point. Access to the initial JUMP table is made using a relative entry in a File Description Table.

In the File Description Table, two words of four 15-bit fields each are available to the object-time I/O routines to indicate the procedures to be performed under specific cases. The format of these fields is:

a	p	o	j	
2	1	4	8	

where

a - 0 if file-name.
1 if INPUT.
2 if OUTPUT.
3 if INPUT/OUTPUT or I/O.

p - 0 if the statement was USE BEFORE.
1 if the statement was USE AFTER.

o - 00 if ERROR PROCEDURE.
05 if ENDING FILE LABEL.
06 if ENDING REEL LABEL.
07 if ENDING (file and reel) LABEL.
11 if BEGINNING FILE LABEL.
12 if BEGINNING REEL LABEL.
13 if BEGINNING (file and reel) LABEL.
15 if (beginning and ending) FILE LABEL.
16 if (beginning and ending) REEL LABEL.
17 if (beginning and ending) (file and reel) LABEL.

j - location within JUMP table.

USE BEFORE REPORTING identifier-1 (identifier-2) is a special case. Location of the JUMP table entry is placed directly into the DNT identifier entry or entries. The statement "USE BEFORE REPORTING identifier-1 [,identifier-2]..." can be used following a section header in the DECLARATIVE section. A unique identifier that may represent any type of report group (01), except DETAIL, can appear in only one USE statement, although more than one report group (01 item) can appear in the same USE statement. This USE statement implies a PERFORM of the sentences between this statement and the next section or END for each report group listed in this USE statement. The implied PERFORM is executed immediately before the specified report group is produced. It can be used to suppress printing of the specified report group by the statement "MOVE 1 TO PRINT-SWITCH."

Report Writer statements must not be used in procedures associated with this USE statement. All logical paths within this declarative section must lead to a common exit point. Only PERFORM statements may refer to procedure names outside of the section and in the non-declarative part, only PERFORM statements may refer to procedure-names within the declarative section.

The item formats for the report group (RGD) will be modified to accommodate the address of the appropriate section of code.

Operand information within the TREE may be any of the following:

1. Information from the Data Name Table.

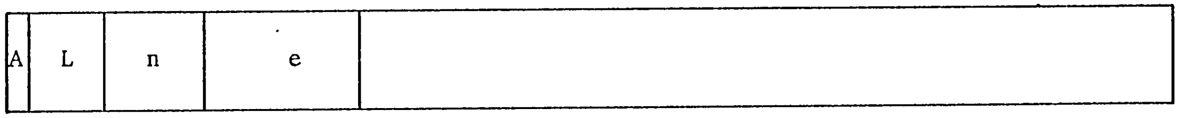
   a. File (FD or SD) information--words FD2, FD3, and FD29.

   b. Group item information--words GDD2 and GDD3. If item has an OCCURS clause, these two words are followed by word SCC, if OCCURS depending on the SCC word are followed by words EDD2 and EDD3 of the depending-on variable.

   c. Elementary item information--words EDD2 and EDD3. If item has an OCCURS clause, these two words will be followed by word SCC. If item has a picture clause, the SCC will be followed by the EP words. If OCCURS depending-on is present, the EP words will be followed by words EDD2 and EDD3 of the depending-on variable.

   d. Report description information--word RD2.

2. Procedure Division literals have the following format:



where

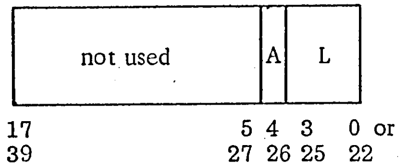
A - 1-bit indicator, 1 = ALL any literal, (non-numeric only).

L - 4-bit indicator, 10 = numeric, integer, unsigned.
11 = numeric, integer, signed.
12 = numeric, non-integer, unsigned.
13 = numeric, non-integer, signed.
17 = non-numeric.

n - 5-bit point location, n ≤ 31 left of assumed decimal point at right of literal.

e - 8-bit number of characters in literal, e ≤ 255
literal-begins in bit position 41 of first word and continues for $\frac{e + 3}{10}$ words.

3. Procedure Division figurative constants are contained within the link address and have the following format:

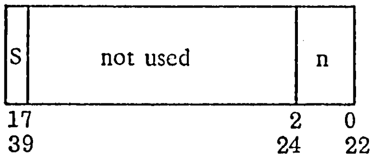


where

A - 1-bit indicator, 1 = ALL any figurative constant.

L - 4-bit indicator, 01 = figurative zero(s).
02 = figurative spaces(s) or low value(s).
03 = figurative quote.
04 = figurative record mark.
07 = figurative high value(s).

4. Switch status indicators are contained within the link address and have the following format:



where

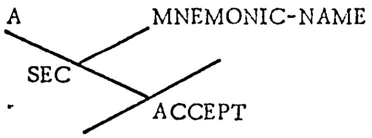
S - 1-bit indicator, 0 = OFF.
1 = ON.

n - the switch number, 1 ≤ n ≤ 6.

Tree Formats

A schematic representation of examples of the types of trees generated by Pass 1E for any given verb are presented in Figures 3-22 through 3-47.

ACCEPT A FROM MNEMONIC-NAME



ACCEPT B

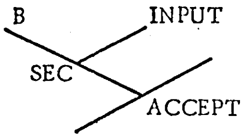
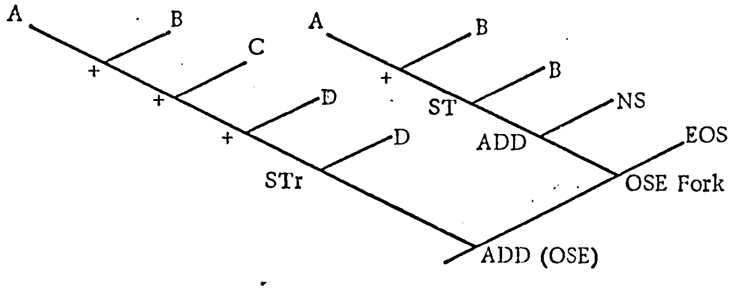


Figure 3-22. ACCEPT Tree Formats

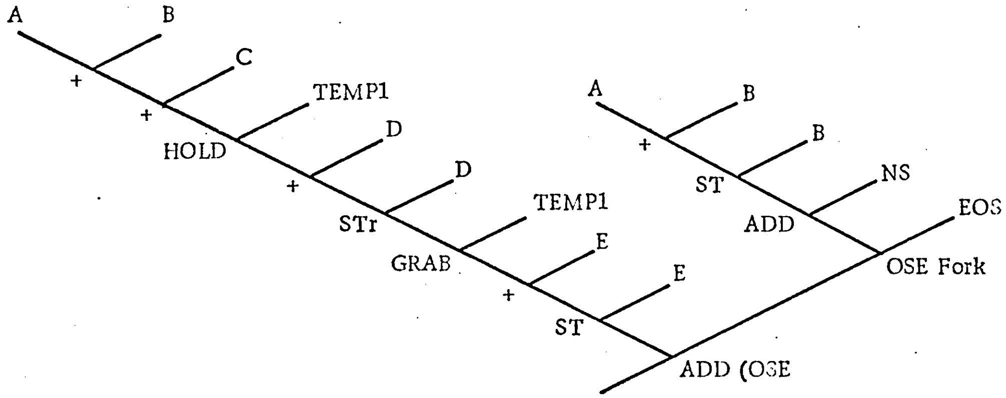
Format 1:

ADD A B C D ROUNDED ON SIZE ERROR ADD A TO B.



Format 2:

ADD A B C TO D ROUNDED E ON SIZE ERROR ADD A TO B.



ADD A B C GIVING D ROUNDED E ON SIZE ERROR ADD A B GIVING C.

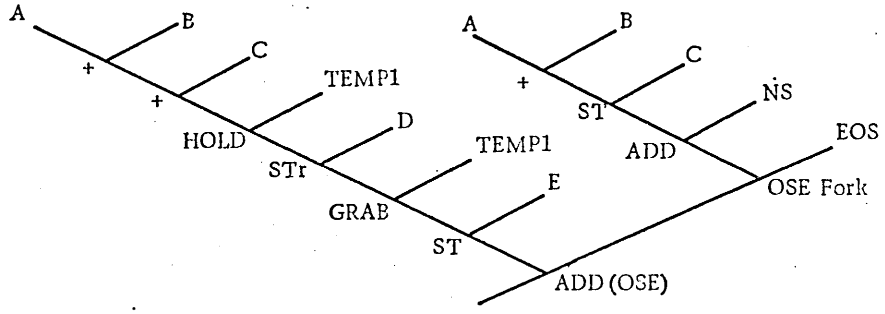
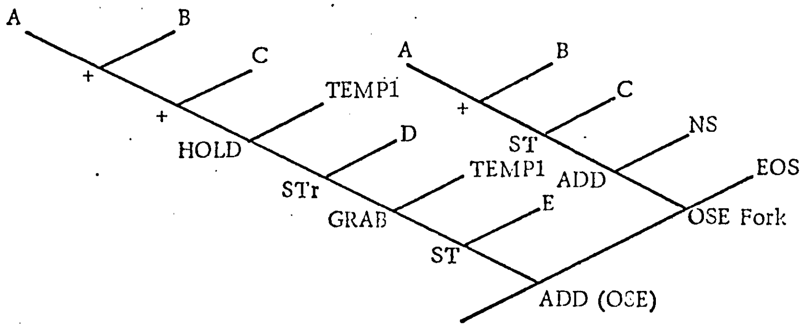


Figure 3-23. ADD Tree Formats (1 of 2)

Format 3:

ADD A B TO C GIVING D ROUNDED E ON SIZE ERROR ADD A TO B GIVING C.



Format 4:

ADD CORRESPONDING A TO B ON SIZE ERROR ADD F TO G, WHERE DATA DIVISION CONTAINS:

01 A
02C
02D
02E

01 B
02C
02D
02E

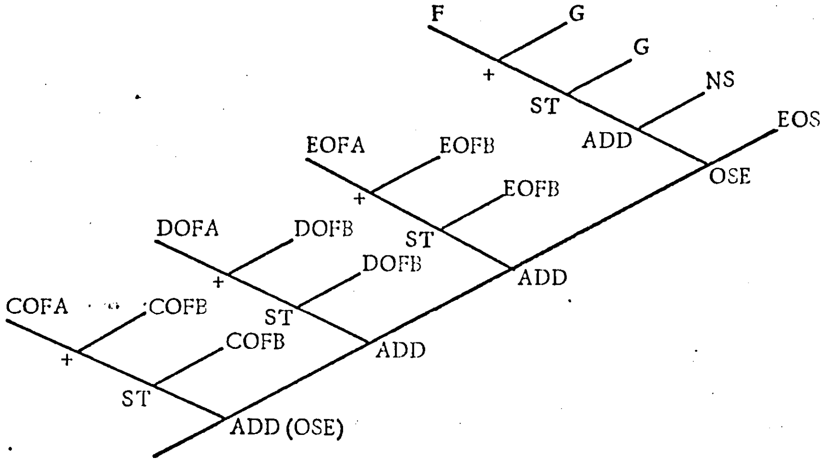


Figure 3-23. ADD Tree Formats (2 of 2)

ALTER PARA TO PROCEED TO PARB PARC TO PROCEED TO PARD.

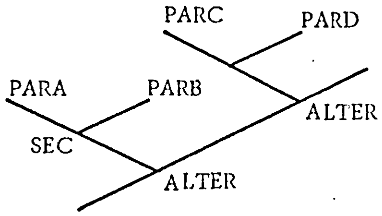


Figure 3-24. ALTER Tree Formats

CLOSE FILE-A REEL WITH LOCK FILE-B WITH NO REWIND.

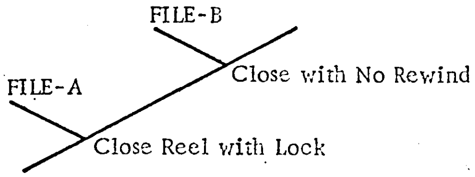


Figure 3-25. CLOSE Tree Formats

COMPUTE A ROUNDED B = C + (D* E)/(C* E)** 2 - D ON SIZE ERROR COMPUTE A = B.

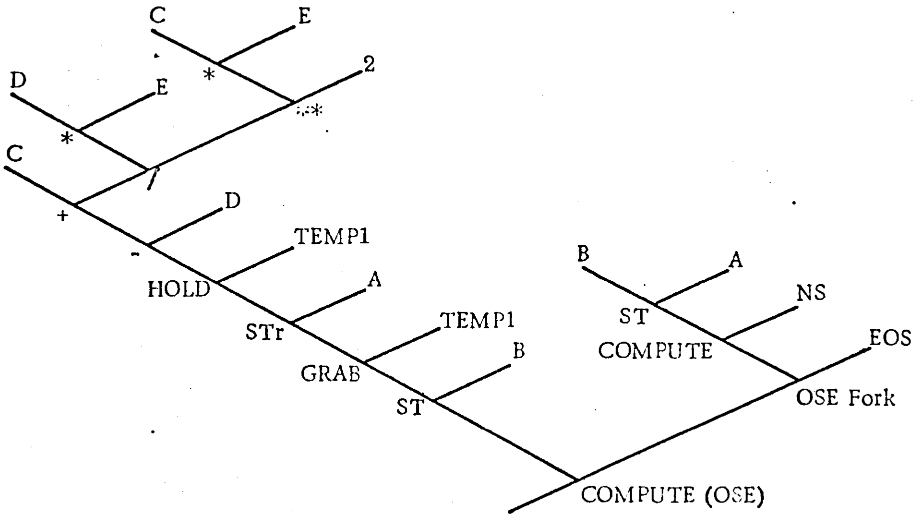
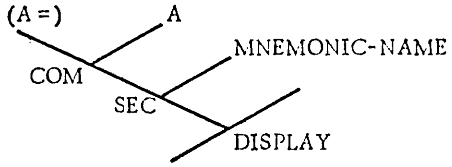


Figure 3-26. COMPUTE Tree Formats

DISPLAY 'A=' A UPON MNEMONIC-NAME.



DISPLAY.

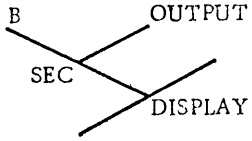
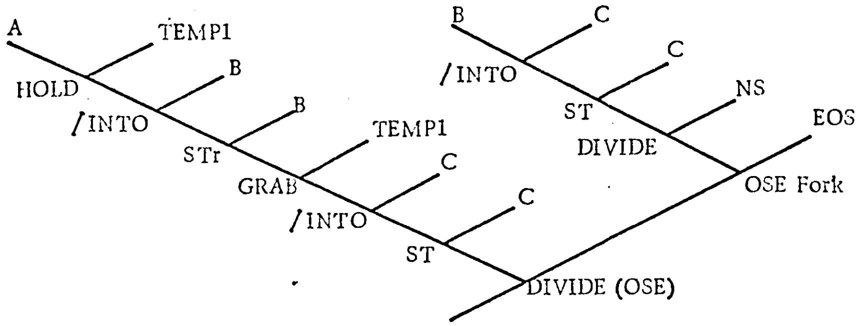


Figure 3-27. DISPLAY Tree Formats

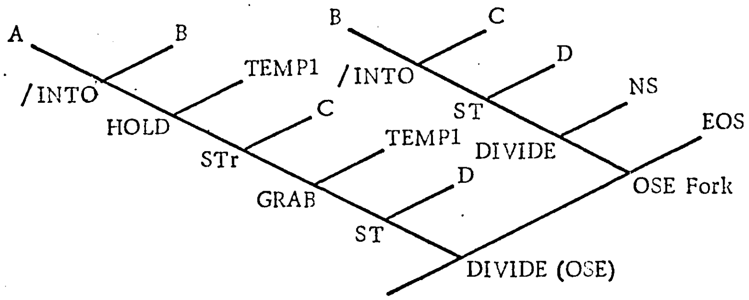
Format 1:

DIVIDE A INTO B ROUNDED C ON SIZE ERROR DIVIDE B INTO C.



Format 2:

DIVIDE A INTO B GIVING C ROUNDED D ON SIZE ERROR DIVIDE B INTO C GIVING D.



Format 3:

DIVIDE A BY B GIVING C ROUNDED D ON SIZE ERROR DIVIDE C BY B GIVING D.

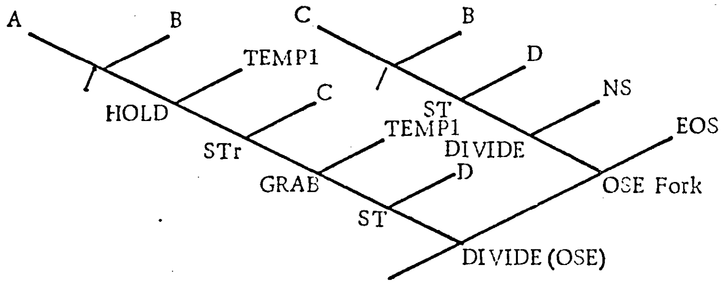
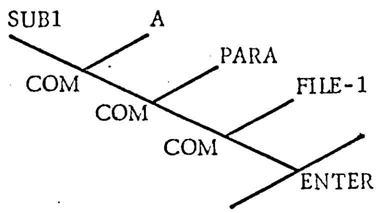


Figure 3-28. DIVIDE Tree Formats

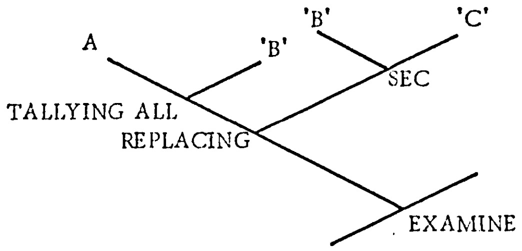
ENTER SUB1 A PARA FILE-1



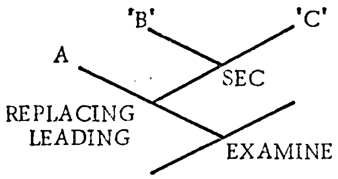
ENTRY generates no trees but marks procedure names as entry points as directed.

Figure 3-29. ENTER, ENTRY Tree Formats

EXAMINE A TALLYING ALL 'B' REPLACING BY 'C'.



EXAMINE A REPLACING LEADING 'B' BY 'C'.



EXIT.

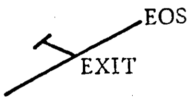


Figure 3-30. EXAMINE, EXIT Tree Formats

GENERATE REPORT-A.

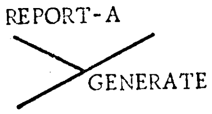
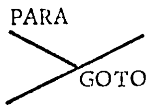


Figure 3-31. GENERATE Tree Formats

Option 1:

GO TO PARA.



Option 2:

GO TO PARA PARB PARC DEPENDING ON A.

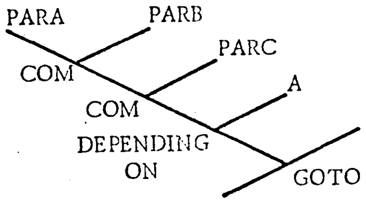
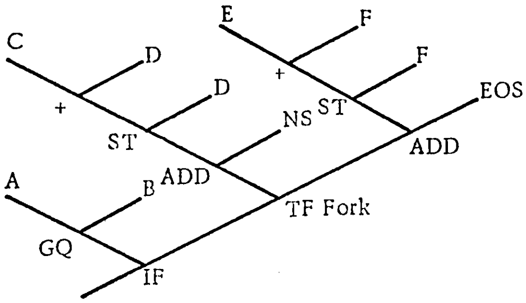
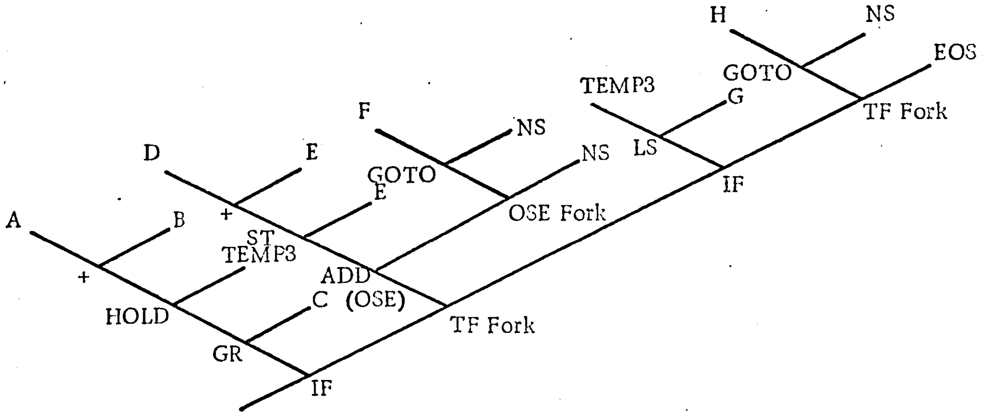


Figure 3-32. GO TO Tree Formats

IF A GQ B ADD C TO D ELSE ADD E TO F.



IF A + B GR C ADD D TO E ON SIZE ERROR GO TO F ELSE IF LS G GO TO H.



IF A NOT GREATER THAN C D OR E NEXT SENTENCE ELSE GO TO F.

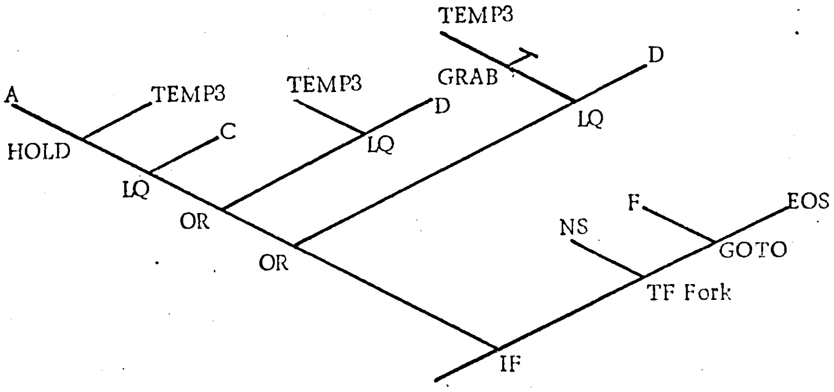
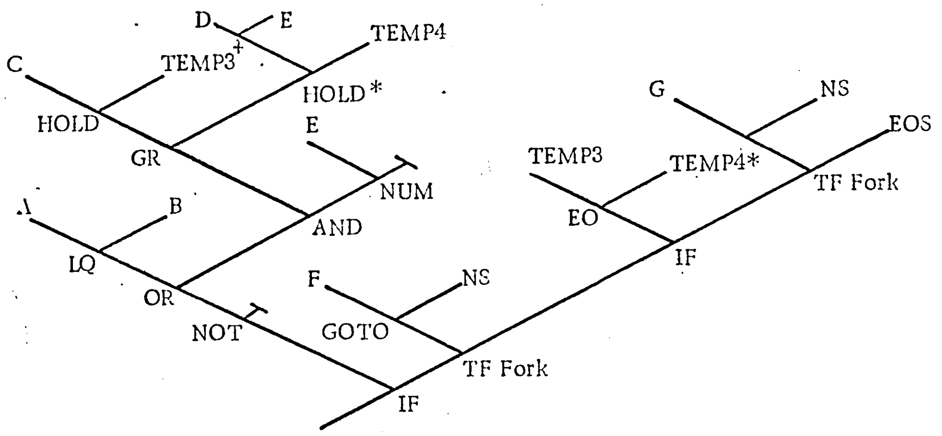


Figure 3-33. IF Tree Formats (1 of 2)

IF NOT (A LQ B OR C GR D+E AND E NUMERIC) GO TO F ELSE IF EQ GO TO G.



*Occur only if the object is an expression. Otherwise, the EQ node would point directly to the object.

IF COND-NAME-1 GO TO A, WHERE DATA DIVISION CONTAINS:

02 B.
88 COND-NAME-1 VALUE IS 1, 9 THROUGH 12.

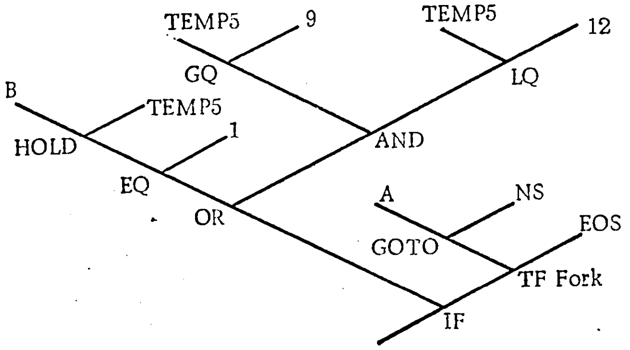
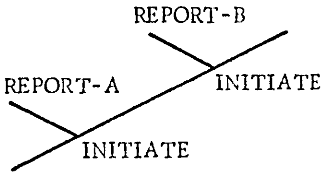


Figure 3-33. IF Tree Formats (2 of 2)

INCLUDE Tree Formats

INCLUDE generates no trees of its own accord, but trees for included statements are generated as directed.

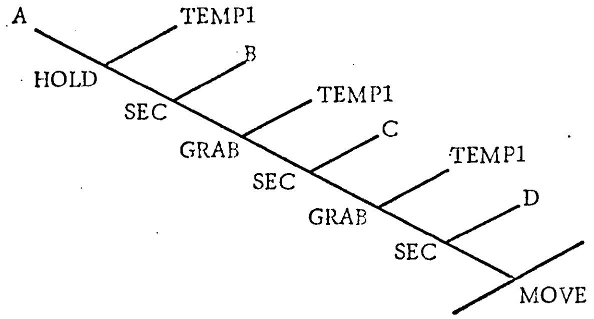
INITIATE REPORT-A REPORT-B.



INITIATE ALL.

Figure 3-34. INITIATE Tree Formats

MOVE A TO B C D.



MOVE CORRESPONDING A TO B, WHERE DATA DIVISION CONTAINS:

```
01  A
    02C
    02D
    02E

01  B
    02C
    02D
    02E
```

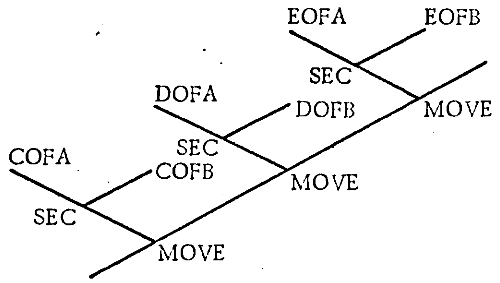
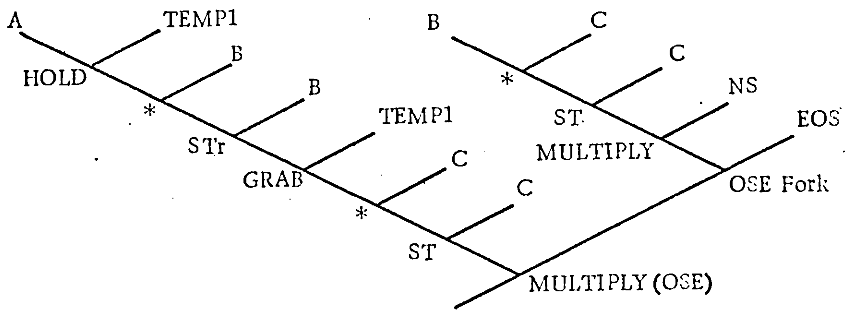


Figure 3-35. MOVE Tree Formats

Format 1:

MULTIPLY A BY B ROUNDED C ON SIZE ERROR MULTIPLY B BY C.



Format 2:

MULTIPLY A BY B GIVING C ROUNDED D ON SIZE ERROR MULTIPLY B BY C GIVING D.

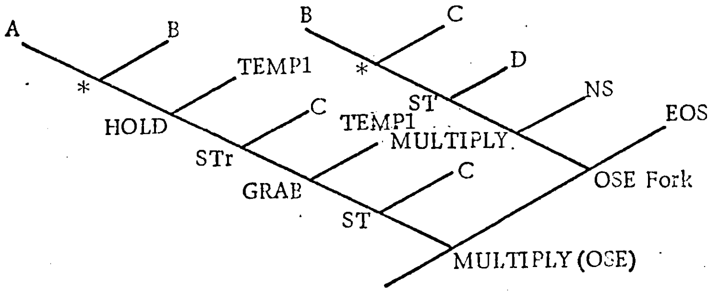
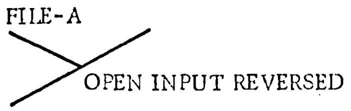
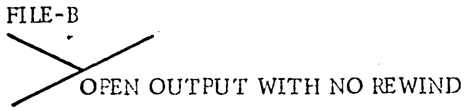


Figure 3-36. MULTIPLY Tree Formats

OPEN INPUT FILE-A REVERSED.



OPEN OUTPUT FILE-B WITH NO REWIND.



OPEN I-O FILE-C.

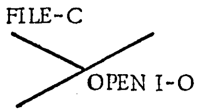
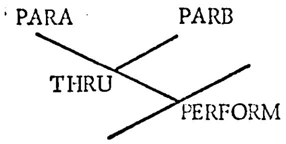


Figure 3-37. OPEN Tree Formats

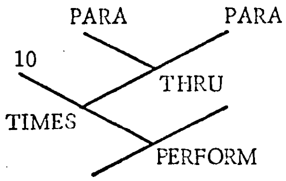
Option 1:

PERFORM PARA THRU PARB.



Option 2:

PER PARA 10 TIMES.



Option 3:

PERFORM PARA UNTIL A EQUAL B.

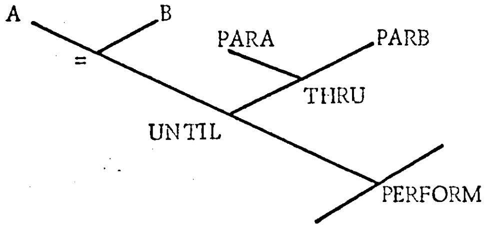


Figure 3-38. PERFORM Tree Formats (1 of 2)

Option 4:

PERFORM PARA VARYING A FROM 1 BY 1 UNTIL A EQ B AFTER C FROM 2 BY 2 UNTIL C GQ D AFTER E FROM 3 BY 3 UNTIL E EXCEEDS F.

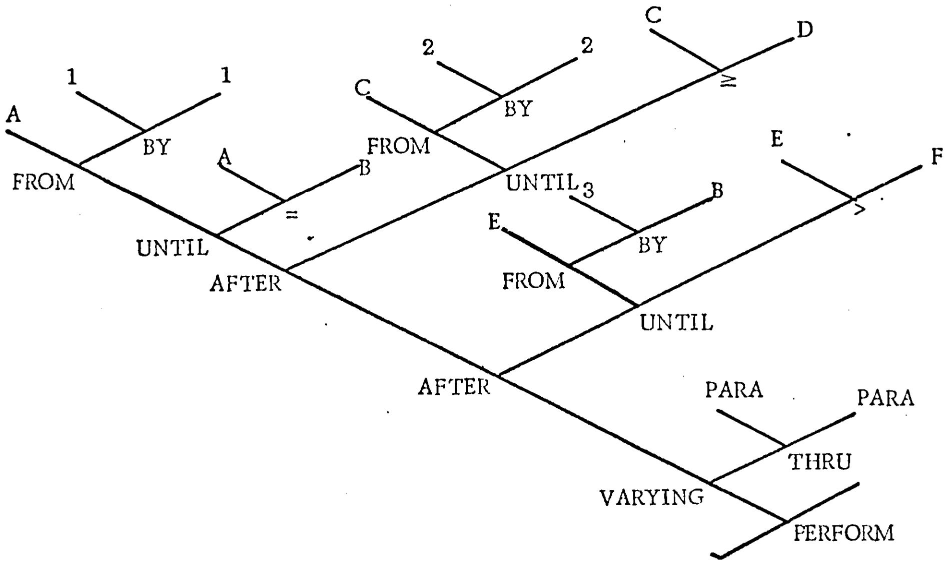
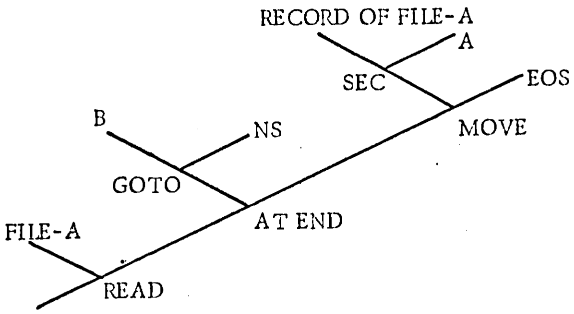


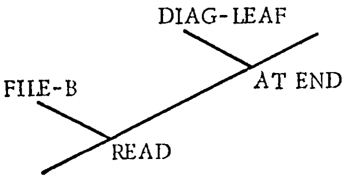
Figure 3-38. PERFORM Tree Formats (2 of 2)

Format 1:

READ FILE-A INTO A AT END GO TO B.



READ FILE-B.



Format 2:

READ FILE-C INVALID KEY GO TO D.

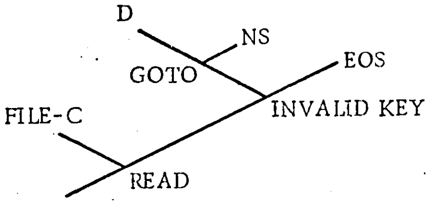


Figure 3-39. READ Tree Formats

RELEASE RECORD-1 FROM A.

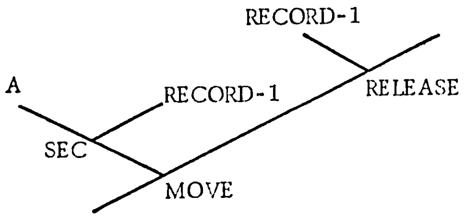
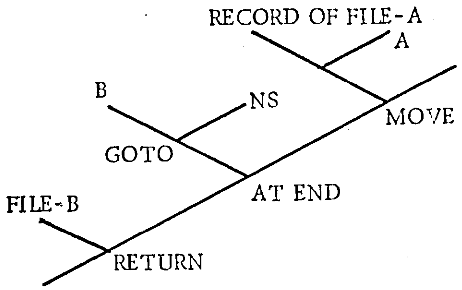


Figure 3-40. RELEASE Tree Formats

RETURN FILE-A INTO A AT END GO TO B.



RETURN FILE-B.

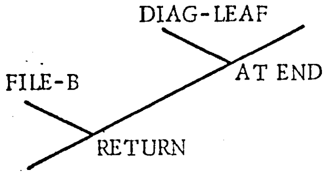
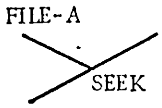
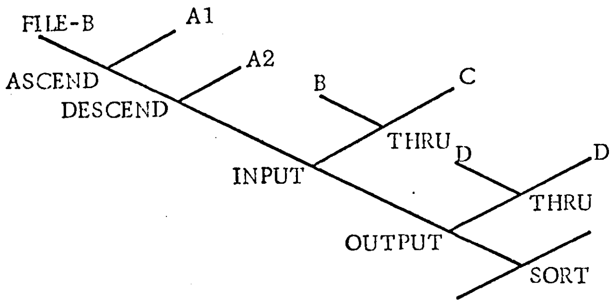


Figure 3-41. RETURN Tree Formats

SEEK FILE-A RECORD.



SORT FILE-B ON ASCENDING KEY A1 ON DESCENDING KEY A2. INPUT PROCEDURE IS B THRU C. OUTPUT PROCEDURE IS D.



SORT FILE-C ON ASCENDING KEY A3 A4 A5 USING FILE-D GIVING FILE-E.

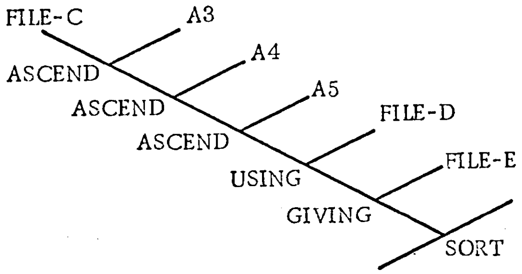


Figure 3-42. SEEK, SORT Tree Formats

STOP RUN.

STOP 'LITERAL'.

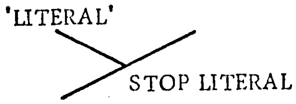


Figure 3-43. STOP Tree Formats

ADD A (B,C,D) TO E (F,G,H).

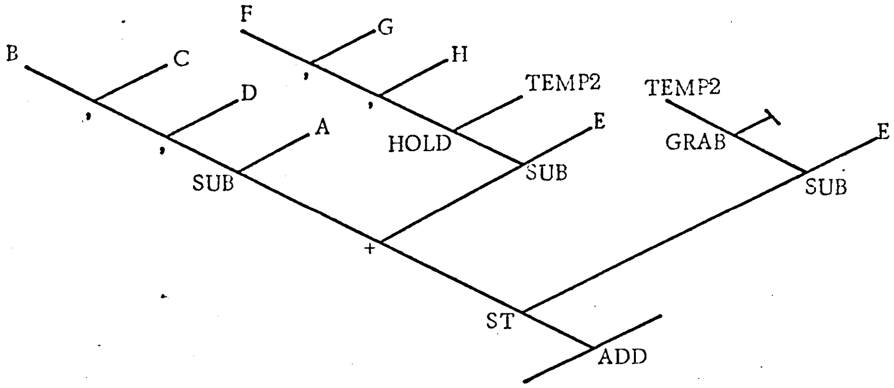
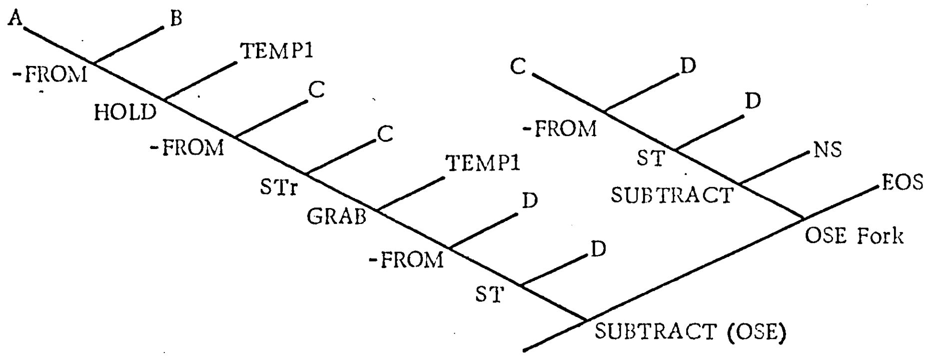


Figure 3-44. Subscripting Tree Formats

Format 1:

SUBTRACT A B FROM C ROUNDED D ON SIZE ERROR SUBTRACT C FROM D.



Format 2:

SUBTRACT A B FROM C GIVING D ROUNDED E ON SIZE ERROR SUBTRACT C FROM D GIVING E.

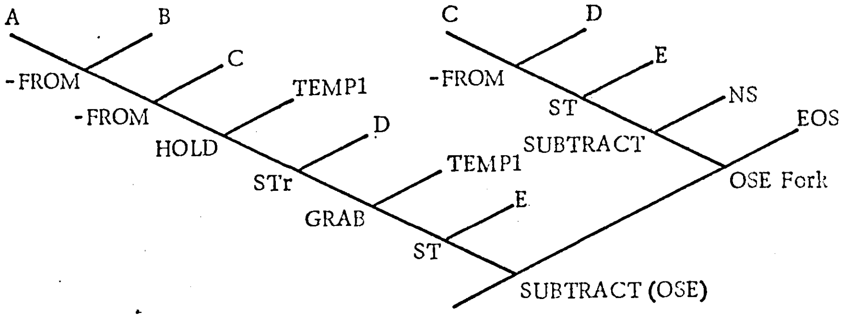


Figure 3-45. SUBTRACT Tree Formats (1 of 2)

Format 3:

SUBTRACT CORRESPONDING A FROM B ON SIZE ERROR SUBTRACT F FROM G WHERE DATA DIVISION CONTAINS:

01 A
02C
02D
02E

01 B
02C
02D
02E

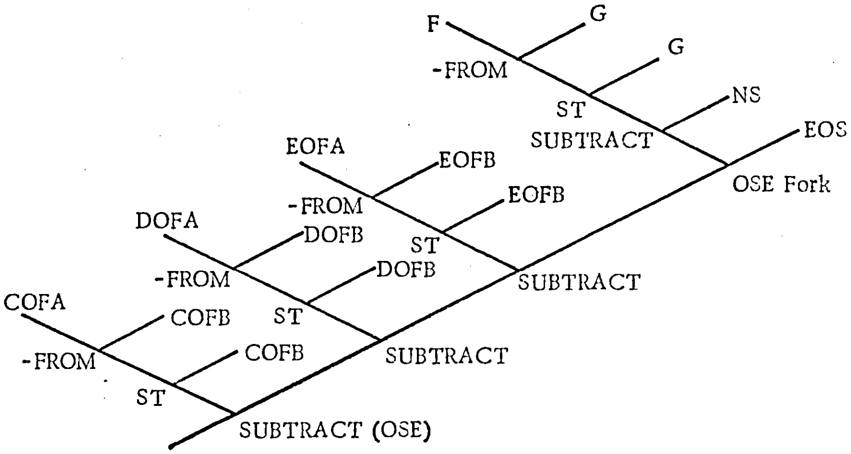
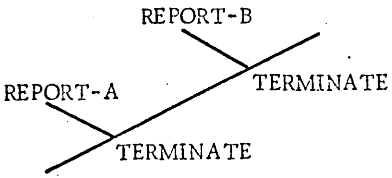


Figure 3-45. SUBTRACT Tree Formats (2 of 2)

TERMINATE REPORT-A REPORT-B.



TERMINATE ALL.

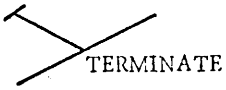
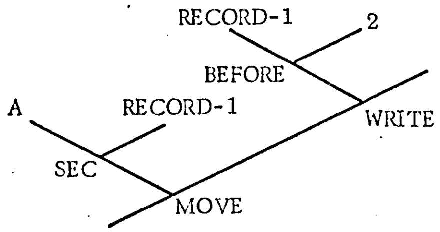


Figure 3-46. TERMINATE Tree Formats

Format 1:

WRITE RECORD-1 FROM A BEFORE ADVANCING 2 LINES.



Format 2:

WRITE RECORD-2 INVALID KEY GO TO B.

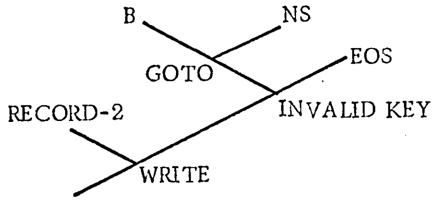


Figure 3-47. WRITE Tree Formats

PASS 1E SYNTABLE ROUTINES

Figure 3-48 on the following pages shows routines called by Pass 1E Syntable and/or used to generate the Trees.

A		IF COLUMST IS EQUAL TO 8, SET COL8FLG TO ZERO AND RETURN CONTROL TO SADNO. OTHERWISE, PRINT DIAGNOSTIC, SET COL8FLG TO ZERO AND RETURN CONTROL TO SADNO.
AB		SET COL8FLG TO ZERO. RETURN CONTROL TO SADNO.
ABRVAND	F	SET TYPE TO AND NODE. INSERT AND NODES LINKING ALL NODES IN PUSH DOWN FROM ABRVNOD TO CURRENT NODE LOCATION. RESET CURRENT NODE LOCATION TO ABRVNOD. RETURN CONTROL TO SADNO.
ABRVOR	F	SET TYPE TO OR NODE. INSERT OR NODES LINKING ALL NODES IN NODE PUSH DOWN FROM ABRVNOD TO CURRENT NODE LOCATION. RESET CURRENT NODE LOCATION TO ABRVNOD. RETURN CONTROL TO SADNO.
ABRVSET		SET ABRVNOD TO CURRENT NODE PUSH DOWN LOCATION. RETURN CONTROL TO SADNO.
ACCMNEM	F	IF FILE TYPE IS OUTPUT, TRANSFER CONTROL TO INPUTF. OTHERWISE, CALL MOVEDAT TO MOVE DNT INFORMATION TO TREE. ADD TREE LOCATION OF DNT INFORMATION TO NODE PUSH DOWN. SET DATAN TO ZERO. TRANSFER CONTROL TO SECNODE.
ACCT		SET CONCOMA TO NON-ZERO. RETURN CONTROL TO SADNO.
ACPTMAS		SET UP ACCEPT MASTER NODE. TRANSFER CONTROL TO MASTER.
ADDMAS		SET UP ADD MASTER NODE. SET TYPE TO PLUS. TRANSFER CONTROL TO MASTER.
AFTNODE		SET UP PERFORM AFTER NODE. TRANSFER CONTROL TO OUTNODE.
AFTTYPE		SET TYPE TO AFTER ADVANCING. RETURN CONTROL TO SADNO.
ALLDIAG		SET NOTDAG TO ZERO. RETURN CONTROL TO SADNO.
ALLOPT		OR EXAMINE ALL BIT INTO TYPE. RETURN CONTROL TO SADNO.
ALLSET		SET ALL BIT ON IN ALLLIT. RETURN CONTROL TO SADNO.
ALPNODE		IF RELATION NOT BIT IS NOT ON, SET UP ALPHABETIC NODE AND TRANSFER CONTROL TO PARNODE. OTHERWISE, TURN OFF RELATION NOT BIT. SET UP NOT ALPHABETIC NODE. TRANSFER CONTROL TO PARNODE.
ALRTOND	F	IF PQBUF IS NON-ZERO, SET UP RIP CONTROL WORD FOR PROCEDURE NAME WITH QUALIFIER. OTHERWISE, SET UP RIP CONTROL WORD FOR PROCEDURE NAME WITHOUT QUALIFIER. CALL RIP TO SET UP PNT ENTRY. ADD PNT LOCATION TO NODE PUSH DOWN. RETURN CONTROL TO SADNO.
ALTOTRE	F	SET UP LEAD INFORMATION FOR NON-NUMERIC LITERAL. MOVE LEAD INFORMATION AND ALL CHARACTERS OF LITERAL FROM CURNWD BUFFER TO TREE. SET DATANOW TO TREE LOCATION OF LITERAL. SET ALLLIT, DATAN, SUBFLAG TO ZERO. RETURN CONTROL TO SADNO.
ALTRMAS		SET UP ALTER MASTER NODE. TRANSFER CONTROL TO MASTER.
ANDNODE		SET UP AND NODE. TRANSFER CONTROL TO OUTNODE.
ASCEND		SET TYPE TO SORT ASCENDING NODE. RETURN CONTROL TO SADNO.

Figure 3-48. Routines Called by Pass 1E Syntable and/or Used to Generate the Trees (1 of 19)

BDECMAS	SET JUMPER, CURRNO TO ZERO. SET FTIMEF TO NON-ZERO. SET UP BEGINNING-OF-DECLARATIVES MASTER NODE. TRANSFER CONTROL TO MPLANT.
BEFTYPE	SET TYPE TO BEFORE ADVANCING. RETURN CONTROL TO SADNO.
BNDRMAS	SET FTIMEF TO ZERO. SET UP BEGINNING-OF-NON-DECLARATIVE-PROCEDURES MASTER NODE. TRANSFER CONTROL TO MPLANT.
BNPRSET	SET PNBUF TO ZERO. RETURN CONTROL TO SADNO.
BNSCMAS F	SET CURRNO TO ZERO. SET UP DIP CONTROL WORD FOR NO SECTION NAMED. CALL DIP TO SET UP PNT ENTRY. SET CURRSEC TO PNT LOCATION. ADD PNT LOCATION TO NODE PUSH DOWN. SET UP BEGINNING-OF-SECTION MASTER NODE. TRANSFER CONTROL TO MLPLANT.
BPARMAS F	IF PNBUF IS ZERO, SET UP DIP CONTROL WORD FOR UNNAMED PARAGRAPH. OTHERWISE, SET UP DIP CONTROL WORD FOR NAMED PARAGRAPH. CALL DIP TO SET UP PNT ENTRY. ADD PNT LOCATION TO NODE PUSH DOWN. SET CURRPAR TO PNT LOCATION. TRANSFER CONTROL TO MLPLANT.
BSECMAS F	IF PQBUF IS ZERO, SET UP DIP CONTROL WORD FOR SECTION NAMED BUT NO SECTION NUMBER. SET CURRNO TO ZERO. OTHERWISE, SET UP DIP CONTROL WORD FOR SECTION NAMED WITH SECTION NUMBER. SET CURRNO TO C(PQBUF). CALL DIP TO SET UP PNT ENTRY. SET CURRSEC TO PNT LOCATION. PICK UP PRIORITY NUMBER FROM PNT ENTRY. IF PRIORITY GREATER THAN HIGHSEC, SET HIGHSEC TO THIS PRIORITY NUMBER. OTHERWISE AND FOLLOWING STORE TO HIGHSEC, SET SECTOR TO THIS PRIORITY NUMBER. ADD PNT LOCATION TO NODE PUSH DOWN. SET UP BEGINNING-OF-SECTION MASTER NODE. TRANSFER CONTROL TO MLPLANT.
BYNODE	SET UP PERFORM BY NODE. TRANSFER CONTROL TO OUTNODE.
CANALT	SET UP RIP CONTROL WORD TO CANCEL LAST ENTRY. CALL RIP. RETURN CONTROL TO SADNO.
CANPER	SET UP RIP CONTROL WORD TO CANCEL LAST ENTRY. CALL RIP. RETURN CONTROL TO SADNO.
CCT	IF CONCOMA IS ZERO, RETURN CONTROL TO SADNO. OTHERWISE, SET CONCOMA TO ZERO AND TRANSFER CONTROL TO COMMA.
CLEROPT	SET OPTION, FORMAT TO ZERO. RETURN CONTROL TO SADNO.
CLOSMAS	SET UP CLOSE MASTER NODE. TRANSFER CONTROL TO MASTER.

Figure 3-48. Routines Called by Pass 1E Syntable and/or Used to Generate the Trees (2 of 19)

```
COBTOPN    IF PQBUF IS NON-ZERO, SET UP RIP CONTROL WORD FOR
           PROCEDURE NAME WITH MODIFIER.
           OTHERWISE, SET UP RIP CONTROL WORD FOR PROCEDURE NAME
           WITHOUT QUALIFIER. CALL RIP TO SET UP PNT ENTRY.
           TRANSFER CONTROL TO SADNO.
COMMA      SET COMAFLG TO ZERO.
           IF XTENDMD IS NON-ZERO, RETURN CONTROL TO SADNO.
           OTHERWISE, SET SEMIFLG TO ZERO AND RETURN CONTROL TO SADNO.
           OTHERWISE, RETURN CONTROL TO SADNO.
COMNODE    SET UP COMMA NODE. TRANSFER CONTROL TO OUTNODE.
COMPMAS    SET UP COMPUTE MASTER NODE. TRANSFER CONTROL TO MASTER.
COMPRIM F  IF ROUNDER IS ZERO, SET UP STORE NODE.
           OTHERWISE, SET ROUNDER TO ZERO AND SET UP STORE
           ROUNDED NODE.
           RIGHT LINK OF INPUT NODE IS CURRENT NODE.
           LEFT LINK IS OPEN. INSERT INPUT NODE IN TREE.
           SET GRABER TO TREE LOCATION.
           CANCEL CURRENT NODE FROM AND ADD TREE LOCATION TO
           NODE PUSH DOWN. SET POINTER TO ZERO.
           RETURN CONTROL TO SADNO.
COMPRS  F  IF ROUNDER IS ZERO, SET UP STORE NODE.
           OTHERWISE, SET UP STORE ROUNDED NODE AND SET
           ROUNDER TO ZERO.
           SET UP GRAB NODE. RIGHT LINK OF GRAB NODE IS TEMPORARY1.
           LEFT LINK OF GRAB NODE IS CURRENT NODE - 1.
           INSERT GRAB NODE IN TREE.
           RIGHT LINK OF STORE NODE IS CURRENT NODE.
           LEFT LINK OF STORE NODE IS TREE LOCATION OF GRAB NODE.
           SET POINTER TO NON-ZERO.
           CANCEL TWO MOST CURRENT NODES FROM AND ADD TREE LOCATION
           TO NODE PUSH DOWN. RETURN CONTROL TO SADNO.
COMPTIE F  IF POINTER IS NON-ZERO, SET UP HOLD NODE.
           RIGHT LINK IS TEMPORARY1. LEFT LINK IS CURRENT NODE.
           INSERT HOLD NODE IN TREE.
           CANCEL CURRENT NODE FROM AND ADD TREE LOCATION OF HOLD
           NODE TO NODE PUSH DOWN.
           OTHERWISE AND AFTER INSERTING HOLD NODE, INSERT CURRENT
           NODE AS LEFT LINK OF NODE POINTED TO BY GRABER.
           CANCEL CURRENT NODE FROM NODE PUSH DOWN.
           RETURN CONTROL TO SADNO.
CORRESP F  SET UP CONTROL WORD FOR MIG.
           THEN'
           1. CALL MIG TO MATCH ITEMS.
           2. IF NO MORE MATCHES, SET ROUNDER TO ZERO AND RETURN
              CONTROL TO SADNO.
              OTHERWISE, IF THIS IS FIRST MATCH, TRANSFER CONTROL
              TO STEP 3.
```

Figure 3-48. Routines Called by Pass 1E Syntable and/or Used to Generate the Trees (3 of 19)

```
       OTHERWISE, INSERT CURRENT NODE AS LEFT LINK OF
       CURRENT MASTER NODE.
       CANCEL CURRENT NODE FROM PUSH DOWN.
       INSERT NEW MASTER NODE AS RIGHT LINK OF CURRENT
       MASTER NODE. SET UP NEW MASTER NODE.
       INSERT MASTER NODE IN TREE.
       REPLACE CURRENT MASTER NODE IN PUSH DOWN WITH TREE
       LOCATION OF NEW MASTER NODE.
    3. CALL MOVEDAT TO MOVE SOURCE DNT INFORMATION TO TREE.
       IF SUBSOUR IS ZERO, SET DATANOW TO TREE LOCATION OF
       DNT INFORMATION AND TRANSFER CONTROL TO STEP 4.
       OTHERWISE, IF SUBSOUR IS NEGATIVE, SET RIGHT LINK OF
       NODE POINTED TO BY HLDSOUR TO TREE LOCATION OF DNT
       INFORMATION, SET DATANOW TO C(HLDSOUR), SET SUBSOUR
       TO +1 AND TRANSFER CONTROL TO STEP 4.
       OTHERWISE, SET UP SUBSCRIPTING NODE.
       RIGHT LINK IS TREE LOCATION OF DNT INFORMATION.
       LEFT LINK IS LEFT LINK OF NODE POINTED TO BY HLDSOUR.
       INSERT NODE IN TREE.
       SET DATANOW TO TREE LOCATION OF SUBSCRIPTING NODE.
    4. ADD C(DATANOW) TO NODE PUSH DOWN.
       CALL MOVEDAT TO MOVE SINK DNT INFORMATION TO TREE.
       IF SUBSINK IS ZERO, SET DATANOW TO TREE LOCATION OF
       DNT INFORMATION AND TRANSFER CONTROL TO STEP 5.
       OTHERWISE, IF SUBSINK IS NEGATIVE, SET DATANOW TO
       C(HLDSINK), SET RIGHT LINK OF NODE POINTED TO BY
       HLDSINK TO TREE LOCATION OF DNT, SET SUBSINK TO +1
       AND TRANSFER CONTROL TO STEP 5.
       OTHERWISE, SET UP SUBSCRIPTING NODE.
       RIGHT LINK IS TREE LOCATION OF DNT INFORMATION.
       LEFT LINK IS LEFT LINK OF NODE POINTED TO BY HLDSINK.
       INSERT NODE IN TREE.
       SET DATANOW TO TREE LOCATION OF SUBSCRIPTING NODE.
    5. SET UP NODE AS DIRECTED BY TYPE.
       RIGHT LINK IS C(DATANOW). LEFT LINK IS CURRENT NODE.
       INSERT NODE IN TREE.
       REPLACE CURRENT NODE IN PUSH DOWN WITH TREE LOCATION OF
       TYPE NODE.
       IF VERB TYPE IS NOT MOVE, TRANSFER CONTROL TO STEP 6.
       OTHERWISE, PICK UP MIG CONTROL WORD AND TRANSFER
       CONTROL TO STEP 1.
    6. IF SUBSINK IS ZERO ADD C(DATANOW) TO NODE PUSH DOWN
       AND TRANSFER CONTROL TO STEP 7.
       OTHERWISE, SET UP HOLD NODE. RIGHT LINK IS TEMPORARY2.
       LEFT LINK IS LEFT LINK OF NODE POINTED TO BY DATANOW.
       INSERT HOLD NODE IN TREE.
```

Figure 3-48. Routines Called by Pass 1E Syntable and/or Used to Generate the Trees (4 of 19)

```
              REPLACE LEFT LINK OF NODE POINTED TO BY DATANOW WITH
              TREE LOCATION OF HOLD NODE. SET UP GRAB NODE.
              RIGHT LINK IS NULL. LEFT LINK IS TEMPORARY.
              INSERT NODE IN TREE. SET UP SUBSCRIPTING NODE.
              RIGHT LINK IS RIGHT LINK OF NODE POINTED TO BY DATANOW.
              LEFT LINK IS TREE LOCATION OF GRAB NODE.
              INSERT NODE IN TREE.
              ADD TREE LOCATION OF SUBSCRIPTING NODE TO NODE
              PUSH DOWN.
           7. IF ROUNDER IS ZERO, SET UP STORE NODE.
              OTHERWISE, SET UP STORE ROUNDED NODE.
              RIGHT LINK IS CURRENT NODE.
              CANCEL CURRENT NODE FROM PUSH DOWN.
              LEFT LINK IS NEW CURRENT NODE. INSERT NODE IN TREE.
              REPLACE CURRENT NODE IN PUSH DOWN WITH TREE LOCATION
              OF THIS NODE.
              PICK UP MIG CONTROL WORD AND TRANSFER CONTROL TO
              STEP 1.
CORSINK    SET SINK TO C(DATAN). SET DATAN, SUBSINK TO ZERO.
           IF SUBFLAG IS ZERO, RETURN CONTROL TO SADNO
           OTHERWISE, SET SUBSINK TO -1.
           SET HLDSINK TO CURRENT NODE.
           CANCEL CURRENT NODE FROM PUSH DOWN. SET SUBFLAG TO ZERO.
           RETURN CONTROL TO SADNO.
CORSOUR    SET SOURCE TO C(DATAN). SET DATAN, SUBSOUR TO ZERO.
           SET OSENODE TO CURRENT MASTER NODE.
           IF SUBFLAG IS ZERO, SET CORRPNT TO ZERO AND RETURN
           CONTROL TO SADNO.
           OTHERWISE, SET SUBSOUR TO -1.
           SET HLDSOUR TO CURRENT NODE.
           CANCEL CURRENT NODE FROM PUSH DOWN.
           SET CORRPNT, SUBFLAG TO ZERO. RETURN CONTROL TO SADNO.
D          RETURN CONTROL TO SADNO.
DATTOND F  IF DATAN IS ZERO, THEN IF SUBFLAG IS ZERO, TRANSFER
           CONTROL TO SUBTOND.
           OTHERWISE, TRANSFER CONTROL TO SUBGRABR.
           OTHERWISE, CALL MOVEDAT TO MOVE DNT INFORMATION TO
           TREE. SET DATAN TO ZERO.
           IF SUBFLAG IS ZERO, SET DATANOW TO LOCATION OF DNT
           INFORMATION IN TREE. TRANSFER CONTROL TO SUBTOND.
           OTHERWISE, SET RIGHT LINK OF NODE POINTED TO BY SUBGRAB
           TO TREE LOCATION OF DNT INFORMATION.
           TRANSFER CONTROL TO SUBTOND.
DANDCND    TEST SAD TABLE FLAG WORD OF NEXT LOWER LEVEL FOR OR BIT.
           IF OR BIT IS SET AND PARENTHESIS BIT IS NOT SET, RETURN
           CONTROL TO SADNO.
           OTHERWISE, RETURN CONTROL TO SADYES.
```

Figure 3-48. Routines Called by Pass 1E Syntable and/or Used to Generate the Trees (5 of 19)

```
DCCND      TEST SAD TABLE FLAG WORD FOR RELATION BIT.
           IF RELATION BIT IS NOT SET, RETURN CONTROL TO SADNO.
           OTHERWISE, RETURN CONTROL TO SADYES.
DCCT       SET CONCOMA TO ZERO. RETURN CONTROL TO SADNO.
DCOND      TEST SAD TABLE FLAG WORD  OF NEXT LOWER LEVEL FOR
           CONNECTIVE BITS.
           IF CONNECTIVE BITS ARE SET AND THE PARENTHESIS BIT IS
           NOT SET, RETURN CONTROL TO SADNO.
           OTHERWISE, RETURN CONTROL TO SADYES.
DECCOND F  CALL MOVEDAT TO MOVE CONDITION VARIABLE TO TREE.
            IF SUBFLAG IS ZERO, ADD TREE LOCATION OF DNT
            INFORMATION TO NODE PUSH DOWN.
            OTHERWISE, INSERT TREE LOCATION OF DNT INFORMATION AS
            RIGHT LINK OF NODE POINTED TO BY SUBGRAB AND ADD
            C(DATANOW) TO NODE PUSH DOWN.
           MOVE VALUE OF CONDITION-NAME TO TREE AS LITERAL.
           SET DATANOW TO TREE LOCATION OF LITERAL.
           SET ANDER TO ZERO.
           IF ONLY ONE VALUE, SET UP EQUAL NODE.
           RIGHT LINK IS C(DATANOW). LEFT LINK IS CURRENT NODE.
           INSERT EQUAL NODE IN TREE.
           REPLACE CURRENT NODE IN NODE PUSH DOWN WITH TREE
           LOCATION OF EQUAL NODE. RETURN CONTROL TO SADNO.
           OTHERWISE, SET UP HOLD NODE. RIGHT LINK IS TEMPORARY5.
           LEFT LINK IS CURRENT NODE. INSERT HOLD NODE IN TREE.
           REPLACE CURRENT NODE IN PUSH DOWN WITH TREE LOCATION
           OF HOLD NODE.
           THEN, FOR EACH VALUE OF THE CONDITION-NAME'
           1. IF VALUE IS NOT SUBJECT OF THRU, SET UP EQUAL NODE.
              RIGHT LINK IS C(DATANOW). LEFT LINK IS CURRENT NODE.
              INSERT NODE IN TREE.
              REPLACE CURRENT NODE IN NODE PUSH DOWN WITH TREE
              LOCATION OF EQUAL NODE. TRANSFER CONTROL TO STEP 3.
           2. OTHERWISE, SET UP GREATER-EQUAL NODE.
              RIGHT LINK IS C(DATANOW). LEFT LINK IS CURRENT NODE.
              INSERT GREATER-EQUAL NODE IN TREE.
              REPLACE CURRENT NODE IN PUSH DOWN WITH TREE LOCATION
              OF GREATER EQUAL NODE.
              SET UP GRAB NODE.  RIGHT LINK IS NULL.
              LEFT LINK IS TEMPORARY5. INSERT NODE IN TREE.
              ADD TREE LOCATION OF GRAB NODE TO NODE PUSH DOWN.
              MOVE NEXT VALUE OF CONDITION-NAME TO TREE AS LITERAL.
              SET DATANOW TO TREE LOCATION OF LITERAL.
              SET UP LESS-EQUAL NODE. RIGHT LINK IS C(DATANOW).
              LEFT LINK IS CURRENT NODE. INSERT NODE IN TREE.
              REPLACE CURRENT NODE IN PUSH DOWN WITH TREE
              LOCATION OF LESS-EQUAL NODE.
              SET UP AND NODE. RIGHT LINK IS CURRENT NODE.
```

Figure 3-48. Routines Called by Pass 1E Syntable and/or Used to Generate the Trees (6 of 19)

	LEFT LINK IS CURRENT NODE - 1. INSERT NODE IN TREE. CANCEL CURRENT NODE FROM NODE PUSH DOWN. REPLACE NEW CURRENT NODE IN PUSH DOWN WITH TREE LOCATION OF AND NODE. 3. IF ANDER IS ZERO, SET ANDER TO NON-ZERO AND TRANSFER CONTROL TO STEP 4. OTHERWISE, SET UP OR NODE. RIGHT LINK IS CURRENT NODE. LEFT LINK IS CURRENT NODE - 1. INSERT NODE IN TREE. CANCEL CURRENT NODE FROM NODE PUSH DOWN. REPLACE NEW CURRENT NODE IN PUSH DOWN WITH TREE LOCATION OF OR NODE. 4. IF ALL VALUES OF CONDITION-NAME HAVE BEEN PROCESSED, SET DATAN TO ZERO AND RETURN CONTROL TO SADNO. OTHERWISE, MOVE VALUE OF CONDITION-NAME TO TREE AS LITERAL. SET DATANOW TO TREE LOCATION OF LITERAL. SET UP GRAB NODE. RIGHT LINK IS NULL. LEFT LINK IS TEMPORARY5. INSERT NODE IN TREE. ADD TREE LOCATION OF GRAB NODE TO NODE PUSH DOWN. REPEAT STEPS 1 THRU 4.
DEPNODE	SET UP DEPENDING ON NODE. TRANSFER CONTROL TO OUTNODE.
DESCEND	SET TYPE TO SORT DESCENDING NODE. RETURN CONTROL TO SADNO.
DFORM	TEST SAD TABLE FLAG WORD FOR CONNECTIVE AND/OR RELATION BITS. IF NONE ARE SET, RETURN CONTROL TO SADYES. OTHERWISE, RETURN CONTROL TO SADNO.
DIAGLF	SET LEFT LINK OF CURRENT MASTER NODE TO DIAGNOSTIC LEAF TYPE. RETURN CONTROL TO SADNO.
DISMNEM F	IF FILE TYPE IS INPUT, TRANSFER CONTROL TO OUTPUTF. OTHERWISE, CALL MOVEDAT TO MOVE DNT INFORMATION TO TREE. ADD TREE LOCATION OF DNT INFORMATION TO NODE PUSH DOWN. SET DATAN TO ZERO. TRANSFER CONTROL TO SECNODE.
DISPMAS	SET UP DISPLAY MASTER NODE. TRANSFER CONTROL TO MASTER.
DIVDMAS	SET UP DIVIDE MASTER NODE. SET TYPE TO DIVIDE INTO. TRANSFER CONTROL TO MASTER.
ENDFORK	SET UP AT END FORK. TRANSFER CONTROL TO MASFORK.
ENDSOUR	IF SOREND IS ZERO, RETURN CONTROL TO SADNO. OTHERWISE, RETURN CONTROL TO SADYES.
ENTRMAS	SET UP ENTER MASTER NODE. TRANSFER CONTROL TO MASTER.
EPARMAS	ADD C(CURRPAR) TO NODE PUSH DOWN AS PNT LOCATION. SET UP END-OF-PARAGRAPH MASTER NODE. TRANSFER CONTROL TO MLPLANT.
EPROMAS	SET UP END-OF-PROCEDURES MASTER NODE. SET SECTOR TO C(HIGHSEC). SET ENDFLAG TO NON-ZERO. TRANSFER CONTROL TO MPLANT.
EQTYPE	SET TYPE TO EQUAL NODE. RETURN CONTROL TO SADNO.
ESECMAS F	IF FTIMEF IS ZERO, SET FTIMEF TO NON-ZERO AND RETURN CONTROL TO SADNO.

Figure 3-48. Routines Called by Pass 1E Syntable and/or Used to Generate the Trees (7 of 19)

```
            OTHERWISE, ADD C(CURRSEC) TO NODE PUSH DOWN AS PNT
            LOCATION. SET UP END-OF-SECTION MASTER NODE.
            TRANSFER CONTROL TO MLPLANT.
EXAMMAS     SET UP EXAMINE MASTER NODE. TRANSFER CONTROL TO MASTER.
EXITMAS     SET UP EXIT MASTER NODE. TRANSFER CONTROL TO MASTER.
EXPNODE     SET UP EXPONENTIATE NODE. TRANSFER CONTROL TO OUTNODE.
FILENAM     TRANSFER CONTROL TO FILNAM.
FILNAM    F CALL SCAN TO COLLECT NEXT CHARACTER STRING.
            IF CHARACTER STRING IS NOT NAME, RETURN CONTROL TO SADNOSN.
            OTHERWISE, MOVE NAME TO DATANAM BUFFER.
            CALL DIG TO DEFINE NAME.
            IF NAME IS UNDEFINED, RETURN CONTROL TO SADNOSN.
            OTHERWISE, IF DATA TYPE IS NOT FD OR SD RETURN CONTROL TO
            SADNOSN.
            OTHERWISE, SET DATAN TO DNT LOCATION OF FILE INFORMATION,
            SET SUBFLAG TO ZERO AND RETURN CONTROL TO SADYES.
FILSRCH   F SET FORMAT TO INPUT OPTION + C(OPTION) + C(JUMPER).
            FOR EACH ENTRY IN EAT TABLE, IF ENTRY IS FD AND OPTION IS
            ERROR PROCEDURE, CALL NSERTR.
            OTHERWISE, IF ENTRY IS FD AND LABELS ARE NOT OMITTED,
            CALL NSERTR.
            OTHERWISE, CHECK NEXT ENTRY IN EAT.
            WHEN ALL ENTRIES IN EAT HAVE BEEN CHECKED, RETURN
            CONTROL TO SADNO.
FINDDAT   F SET UP DIG CONTROL WORD. CALL DIG TO FIND DNT ENTRY.
            IF NOT FOUND, RETURN CONTROL TO SADNO.
            OTHERWISE, SET DATAN TO DNT LOCATION.
            SET SUBFLAG TO ZERO. RETURN CONTROL TO SADYES.
FMOBJ     F IF OBJHELD IS ZERO, SET UP HOLD NODE.
            LEFT LINK IS RIGHT LINK OF NODE POINTED TO BY COMPREL.
            RIGHT LINK IS TEMPORARY4. INSERT NODE IN TREE.
            REPLACE RIGHT LINK OF NODE POINTED TO BY COMPREL WITH
            TREE LOCATION OF HOLD NODE. SET OBJHELD TO NON-ZERO.
            OTHERWISE, AND FOLLOWING INSERTION OF HOLD NODE,
            ADD TEMPORARY4 TO NODE PUSH DOWN. SET UP GRAB NODE.
            TRANSFER CONTROL TO PARNODE.
FMSUB     F IF SUBHELD IS ZERO, SET UP HOLD NODE.
            LEFT LINK IS LEFT LINK OF NODE POINTED TO BY COMPREL.
            RIGHT LINK IS TEMPORARY3. INSERT NODE IN TREE.
            REPLACE LEFT LINK OF NODE POINTED TO BY COMPREL WITH
            TREE LOCATION OF HOLD NODE. SET SUBHELD TO NON-ZERO.
            OTHERWISE, AND FOLLOWING INSERTION OF HOLD NODE,
            ADD TEMPORARY3 TO NODE PUSH DOWN. SET UP GRAB NODE.
            TRANSFER CONTROL TO PARNODE.
FMSUBRL     SET TYPE TO RELATION OF NODE POINTED TO BY COMPREL.
            TRANSFER CONTROL TO FMSUB.
FRONODE     SET UP PERFORM FROM NODE. TRANSFER CONTROL TO OUTNODE.
```

Figure 3-48. Routines Called by Pass 1E Syntable and/or Used to Generate the Trees (8 of 19)

FRSTOPT		OR EXAMINE (UNTIL) FIRST BIT INTO TYPE. RETURN CONTROL TO SADNO.
GENRMAS		SET UP GENERATE MASTER NODE. TRANSFER CONTROL TO MASTER.
GIVNODE		SET UP SORT GIVING NODE. TRANSFER CONTROL TO OUTNODE.
GOTOMAS		SET UP GO TO MASTER NODE. TRANSFER CONTROL TO MASTER.
GOTOND	F	IF PQBUF NOT EQUAL TO ZERO, SET UP RIP CONTROL WORD FOR PROCEDURE NAME WITH QUALIFIER. OTHERWISE, SET UP RIP CONTROL WORD FOR PROCEDURE NAME WITHOUT QUALIFIER. CALL RIP TO SET UP PNT ENTRY. ADD PNT LOCATION TO NODE PUSH DOWN. RETURN CONTROL TO SADNO.
GQTYPE		SET TYPE TO GREATER-EQUAL NODE. RETURN CONTROL TO SADNO.
GRAB1	F	IF POINTER . 0 SET UP HOLD NODE. RIGHT LINK IS TEMPORARY1. LEFT LINK IS C(HOLDER). INSERT HOLD NODE IN TREE. REPLACE LEFT LINK OF NODE POINTED TO BY GRABER WITH TREE LOCATION. INCREASE POINTER BY 2. OTHERWISE AND FOLLOWING INSERTION OF HOLD NODE, ADD TEMPORARY1 TO NODE PUSH DOWN. SET UP GRAB NODE. IF TYPE IS SECONDARY NODE, TRANSFER CONTROL TO PARNODE. OTHERWISE, TRANSFER CONTROL TO OUTNODE.
GRTYPE		SET TYPE TO GREATER THAN NODE. RETURN CONTROL TO SADNO.
GVMODE		SET POINTER TO -1. RETURN CONTROL TO SADNO.
HITOTRE		SET DATANOW TO FIGURATIVE HIGH-VALUE(S). SET ALLLIT, DATAN, SUBFLAG TO ZERO. RETURN CONTROL TO SADNO.
IFMAS		SET UP IF MASTER NODE. TRANSFER CONTROL TO MASTER.
IMPGOTO	F	IF FTIMEF IS ZERO, RETURN CONTROL TO SADNO. OTHERWISE, IF PRIORITY NUMBER OF THE NEXT SECTION IS THE SAME AS THIS SECTION, RETURN CONTROL TO SADNO. OTHERWISE, SET UP CONTROL WORD FOR RIP WITH NO QUALIFIER PRESENT. CALL RIP TO SET UP PNT ENTRY. ADD PNT LOCATION TO NODE PUSH DOWN. SET UP IMPLIED GO TO MASTER. TRANSFER CONTROL TO MLPLANT.
IMPKEYW		CALL SCAN TO COLLECT CHARACTER STRING. IF CHARACTER IS AN IMPERATIVE KEY WORD, SET NOSCNFL TO NON-ZERO AND RETURN CONTROL TO SADYES. OTHERWISE, RETURN CONTROL TO SADNOSN.
INITMAS		SET UP INITIATE MASTER NODE. TRANSFER CONTROL TO MASTER.
INNODE		SET UP SORT INPUT NODE. TRANSFER CONTROL TO OUTNODE.
INPUTF	F	SEARCH EAT TABLE FOR INPUT FILE. CALL MOVEDAT TO MOVE DNT INFORMATION TO TREE. ADD TREE LOCATION OF DNT INFORMATION TO NODE PUSH DOWN. SET DATAN TO ZERO. TRANSFER CONTROL TO SECNODE.
INTEGER		CALL SCAN TO COLLECT NEXT CHARACTER STRING. IF CHARACTER STRING IS NOT NUMERIC, RETURN CONTROL TO SADNO. OTHERWISE, IF PNTLOC IS NON-ZERO, RETURN CONTROL TO SADNO. OTHERWISE, RETURN CONTROL TO SADYES.

Figure 3-48. Routines Called by Pass 1E Syntable and/or Used to Generate the Trees (9 of 19)

INTOBUF		SET PQBUF TO ZERO. MOVE FOUR WORDS FOR CURNWD BUFFER TO PNBUF BUFFER. TRANSFER CONTROL TO SADNO.
INTODAT	F	MOVE NAME FROM CURNWD BUFFER TO DATANAM BUFFER BEGINNING AT START OF BUFFER. SET FOLLOW-UP WORD IN DATANAM BUFFER TO ZERO. RETURN CONTROL TO SADNO.
INTOQAT	F	MOVE NAME FROM CURNWD BUFFER TO DATANAM BUFFER BEGINNING WHERE PREVIOUS NAME STOPPED. SET FOLLOW-UP WORD IN DATANAM BUFFER TO ZERO. RETURN CONTROL TO SADNO.
INTOQUF		MOVE FOUR WORDS FROM CURNWD BUFFER TO PQBUF BUFFER. RETURN CONTROL TO SADNO.
IOMAS		SET UP OPEN I-O MASTER NODE. TRANSFER CONTROL TO MASTER.
IOOPN		OR INPUT-OUTPUT BIT INTO CURRENT MASTER NODE. RETURN CONTROL TO SADNO.
KEYWORD		CALL SCAN TO COLLECT NEXT CHARACTER STRING. IF CHARACTER STRING IS NOT PROCEDURE DIVISION KEY WORD, RETURN CONTROL TO SADNOSN. OTHERWISE, SET NOSCNFL TO NON-ZERO AND RETURN CONTROL TO SADYES.
LEADOPT		OR EXAMINE LEADING BIT INTO TYPE. RETURN CONTROL TO SADNO.
LITOTRE	F	SET UP LEAD INFORMATION FOR NUMERIC LITERAL, INCLUDING WHETHER SIGNED OR UNSIGNED, INTEGER OR NON-INTEGER. MOVE LEAD INFORMATION, SIGN IF PRESENT, AND ALL CHARACTERS OF LITERAL FROM CURNWD BUFFER TO TREE. SET DATANOW TO TREE LOCATION OF LITERAL. SET DATAN, SUBFLAG TO ZERO. RETURN CONTROL TO SADNO.
LLMASTN	F	IF M IS EQUAL TO NPDL, TRANSFER CONTROL TO RSNEXT. OTHERWISE, INSERT CURRENT NODE AS LEFT LINK OF CURRENT MASTER NODE. CANCEL CURRENT NODE FROM PUSH DOWN. RETURN CONTROL TO SADNO.
LQTYPE		SET TYPE TO LESS-EQUAL NODE. RETURN CONTROL TO SADNO.
LSTOTRE	F	SET UP LEAD INFORMATION FOR UNSIGNED NUMERIC LITERAL. MOVE LEAD INFORMATION AND ALL CHARACTERS OF LITERAL FROM CURNWD BUFFER TO TREE. SET DATANOW TO TREE LOCATION OF LITERAL. RETURN CONTROL TO SADNO.
LSTYPE		SET TYPE TO LESS THAN NODE. RETURN CONTROL TO SADNO.
MASFORK	F	INSERT FORK INTO TREE. ADD TREE LOCATION TO MASTER NODE PUSH DOWN. SET LASTMAS TO TREE LOCATION OF FORK. RETURN CONTROL TO SADNO.
MASTER	F	SET FVERB TO CURRENT VERB TYPE. INSERT MASTER NODE INTO TREE. ADD TREE LOCATION TO MASTER NODE PUSH DOWN.

Figure 3-48. Routines Called by Pass 1E Syntable and/or Used to Generate the Trees (10 of 19)

```
            RETURN CONTROL TO SADNO.
MASTX       MASTER NODE PUSH DOWN TABLE EXCEEDED.
            TRANSFER CONTROL TO CONEX2.
MLPLANT F   INSERT CURRENT NODE AS LEFT LINK OF INPUT MASTER NODE.
            CANCEL CURRENT NODE FROM PUSH DOWN.
            INSERT INPUT MASTER NODE IN TREE.
            ADD TREE LOCATION TO MASTER NODE PUSH DOWN.
            TRANSFER CONTROL TO PLANTER.
MNEMONF F   CALL SCAN TO COLLECT NEXT CHARACTER STRING.
            IF CHARACTER STRING IS NOT NAME, RETURN CONTROL TO
            SADNOSN.
            OTHERWISE, MOVE NAME TO DATANAM BUFFER.
            CALL DIG TO DEFINE NAME.
            IF NAME IS UNDEFINED, RETURN CONTROL TO SADNOSN.
            OTHERWISE, IF NAME IS NOT MNEMONIC FOR FILE, RETURN CONTROL
            TO SADNOSN.
            OTHERWISE, SET DATAN TO DNT LOCATION OF FILE INFORMATION.
            SET SUBFLAG TO ZERO. RETURN CONTROL TO SADYES.
MNEMONI F   CALL SCAN TO COLLECT NEXT CHARACTER STRING.
            IF CHARACTER STRING IS NOT A NAME, RETURN CONTROL TO
            SADNOSN.
            OTHERWISE, MOVE NAME TO DATANAM BUFFER.
            CALL DIG TO DEFINE NAME.
            IF NAME UNDEFINED, RETURN CONTROL TO SADNOSN.
            OTHERWISE, IF NAME IS NOT MNEMONIC FOR LITERAL, RETURN
            CONTROL TO SADNOSN.
            OTHERWISE, SET DATAN TO DNT LOCATION OF MNEMONIC NAME.
            SET SUBFLAG TO ZERO. RETURN CONTROL TO SADYES.
MNETOND F   SET UP LEAD INFORMATION FOR NON-NUMERIC LITERAL.
            INSERT LEAD INFORMATION AND LITERAL TO TREE.
            SET DATANOW TO TREE LOCATION OF LITERAL.
            SET DATAN TO ZERO. TRANSFER CONTROL TO SUBTOND.
MNSNODE     SET UP MINUS NODE. TRANSFER CONTROL TO OUTNODE.
MOVEDAT F   IF DATA TYPE IS FD OR SD, MOVE 10 WORDS FROM DNT TO TREE
            AND RETURN CONTROL TO CALLING ROUTINE.
            OTHERWISE, IF DATA TYPE IS RD, MOVE ONE WORD FROM DNT TO
            TREE AND RETURN CONTROL TO CALLING ROUTINE.
            OTHERWISE, MOVE NUMBER OF WORDS AS DIRECTED BY ENTRY FROM
            DNT TO TREE.
            IF ENTRY HAS NO OCCURS DEPENDING ON CLAUSE, RETURN CONTROL
            TO CALLING ROUTINE.
            OTHERWISE, MOVE EDD2 AND EDD3 OF DEPENDING ON VARIABLE TO
            TREE. RETURN CONTROL TO CALLING ROUTINE.
MOVEMAS     SET UP MOVE MASTER NODE.  SET TYPE TO SECONDARY.
            TRANSFER CONTROL TO MASTER.
MOVEOUT F   IF MOVFLAG IS ZERO RETURN CONTROL TO SADNO.
            OTHERWISE, SET RIGHT LINK CURRENT MASTER TO C(MOVRCRD).
            REPLACE CURRENT MASTER IN PUSH DOWN WITH C(MOVRCRD).
```

Figure 3-48. Routines Called by Pass 1E Syntable and/or Used to Generate the Trees (11 of 19)

```
           SET LASTMAS TO C(MOVRCRD).
           RETURN CONTROL TO SADNO.
MOVFROM  F SET UP SECONDARY NODE. RIGHT LINK IS CURRENT NODE - 1.
           LEFT LINK IS CURRENT NODE. INSERT NODE IN TREE.
           CANCEL CURRENT NODE FROM PUSH DOWN.
           LEFT LINK OF CURRENT MASTER NODE IS TREE LOCATION OF
           SECONDARY NODE.
           RIGHT LINK OF CURRENT MASTER NODE IS NEXT TREE LOCATION.
           LASTMAS IS CURRENT MASTER NODE.
           CANCEL CURRENT MASTER NODE FROM PUSH DOWN.
           RETURN CONTROL TO SADNO.
MOVTRIG  F CALL MOVEDAT TO MOVE DATA POINTED TO BY MOVRCRD TO TREE.
           SET DATANOW TO TREE LOCATION OF DNT INFORMATION.
           SET UP SECONDARY NODE. RIGHT LINK IS C(DATANOW).
           LEFT LINK IS CURRENT NODE. INSERT NODE IN TREE.
           LEFT LINK OF CURRENT MASTER NODE IS TREE LOCATION OF
           SECONDARY NODE.
           SET MOVRCRD TO TREE LOCATION OF CURRENT MASTER NODE.
           SET MOVFLAG TO NON-ZERO.
           CANCEL CURRENT NODE FROM PUSH DOWN.
           CANCEL CURRENT MASTER NODE FROM PUSH DOWN.
           RETURN CONTROL TO SADNO.
MPLANT   F INSERT INPUT MASTER NODE IN TREE.
           ADD INPUT MASTER NODE TO PUSH DOWN.
           TRANSFER CONTROL TO PLANTER
MULNODE    SET UP MULTIPLY NODE. TRANSFER CONTROL TO OUTNODE.
MULTMAS    SET UP MULTIPLY MASTER NODE. SET TYPE TO TIMES,
           TRANSFER CONTROL TO MASTER.
NAME       CALL SCAN TO COLLECT NEXT CHARACTER STRING.
           IF CHARACTER STRING IS NAME, RETURN CONTROL TO SADYES.
           OTHERWISE, RETURN CONTROL TO SADNOSN.
NEGNODE    IF RELATION NOT BIT IS NOT ON, SET UP NEGATIVE NODE
           AND TRANSFER CONTROL TO PARNODE.
           OTHERWISE, TURN OFF RELATION NOT BIT.
           SET UP NOT NEGATIVE NODE. TRANSFER CONTROL TO PARNODE.
NGMODE     SET POINTER TO -2. RETURN CONTROL TO SADNO.
NONNLIT    CALL SCAN TO COLLECT NEXT CHARACTER STRING.
           IF CHARACTER STRING IS NOT A NON-NUMERIC LITERAL, RETURN
           CONTROL TO SADNOSN.
           OTHERWISE, RETURN CONTROL TO SADNO.
NOTHING    TEST SAD TABLE FLAG WORD OF NEXT LOWER LEVEL.
           IF FLAG WORD IS NON-ZERO, RETURN CONTROL TO SADNO.
           OTHERWISE, RETURN CONTROL TO SADYES.
NQTYPE     SET TYPE TO NOT EQUAL NODE. RETURN CONTROL TO SADNO.
NREWIND    OR NO REWIND BIT INTO CURRENT MASTER NODE.
           RETURN CONTROL TO SADNO.
NSERTI     SET UP INPUT FILE OPTION. TRANSFER CONTROL TO FILSRCH.
NSERTIO    SET UP I-O FILE OPTION. TRANSFER CONTROL TO FILSRCH.
```

Figure 3-48. Routines Called by Pass 1E Syntable and/or Used to Generate the Trees (12 of 19)

NSERTK	INSERT C(JUMPER) IN REPORT GROUP ITEM. SET DATAN TO ZERO. RETURN CONTROL TO SADNO.
NSERTO	SET UP OUTPUT FILE OPTION. TRANSFER CONTROL TO FILSRCH.
NSERTR F	INSERT C(FORMAT) INTO NEXT AVAILABLE QUARTER OF FD LABEL PROCESSING WORDS. RETURN CONTROL TO CALLING ROUTINE.
NUMBER	CALL SCAN TO COLLECT NEXT CHARACTER STRING. IF CHARACTER STRING IS NOT A NUMERIC LITERAL, RETURN CONTROL TO SADNOSN. OTHERWISE, RETURN CONTROL TO SADYES.
NUMNODE	IF RELATION NOT BIT IS NOT ON, SET UP NUMERIC NODE AND TRANSFER CONTROL TO PARNODE. OTHERWISE, TURN OFF RELATION NOT BIT. SET UP NOT NUMERIC NODE. TRANSFER CONTROL TO PARNODE.
NVKFORK	SET UP INVALID KEY FORK. TRANSFER CONTROL TO MASFORK.
NXTLEFT	REPLACE LEFT LINK OF CURRENT MASTER NODE WITH NEXT SENTENCE LINK TYPE. RETURN CONTROL TO SADNO.
NXTRGHT	IF FKFLAG IS NON-ZERO, SET FKFLAG TO ZERO AND RETURN CONTROL TO SADNO. OTHERWISE, SET RIGHT LINK OF CURRENT MASTER NODE TO NEXT SENTENCE LINK TYPE. RETURN CONTROL TO SADNO.
OPENMAS	SET UP OPEN MASTER NODE. TRANSFER CONTROL TO MASTER.
OPTNODE	SET UP SORT OUTPUT NODE. TRANSFER CONTROL TO OUTNODE.
OR001	SET IDENTIFIER BIT IN SAD TABLE FLAG WORD. RETURN CONTROL TO SADNO.
OR002	SET NUMBER BIT IN SAD TABLE FLAG WORD. RETURN CONTROL TO SADNO.
OR004	SET OPERATOR BIT IN SAD TABLE FLAG WORD. RETURN CONTROL TO SADNO.
OR010	SET RELATION BIT IN SAD TABLE FLAG WORD. RETURN CONTROL TO SADNO.
OR020	SET RELATION NOT BIT IN SAD TABLE FLAG WORD. RETURN CONTROL TO SADNO.
OR040	SET IS BIT IN SAD TABLE FLAG WORD. RETURN CONTROL TO SADNO.
OR100	SET AND BIT IN SAD TABLE FLAG WORD. RETURN CONTROL TO SADNO.
OR200	SET CONDITIONAL NOT BIT IN SAD TABLE FLAG WORD. RETURN CONTROL TO SADNO.
OR400	SET PARENTHESIS BIT IN SAD TABLE FLAG WORD. RETURN CONTROL TO SADNO.
OR1000	SET OR BIT IN SAD TABLE FLAG WORD. RETURN CONTROL TO SADNO.
ORNODE	SET UP OR NODE. TRANSFER CONTROL TO OUTNODE.
ORVNODE	INSERT CURRENT NODE AS LEFT LINK OF INPUT NODE. CANCEL CURRENT NODE FROM PUSH DOWN. INSERT NEW CURRENT NODE AS RIGHT LINK OF INPUT NODE.

Figure 3-48. Routines Called by Pass 1E Syntable and/or Used to Generate the Trees (13 of 19)

	INSERT INPUT NODE IN TREE REPLACE CURRENT NODE IN PUSH DOWN WITH TREE LOCATION OF INPUT NODE. RETURN CONTROL TO SADNO.
OSEFORK	SET UP ON SIZE ERROR FORK. TRANSFER CONTROL TO MASFORK.
OSESET	SET OSENODE TO CURRENT NODE. TRANSFER CONTROL TO SADNO.
OUTNODE F	INSERT CURRENT NODE AS RIGHT LINK OF INPUT NODE. CANCEL CURRENT NODE FROM PUSH DOWN. INSERT NEW CURRENT NODE AS LEFT LINK OF INPUT NODE. INSERT INPUT NODE IN TREE. REPLACE CURRENT NODE IN NODE PUSH DOWN WITH TREE LOCATION OF INPUT NODE. RETURN CONTROL TO SADNO.
OUTOPN	OR OUTPUT BIT INTO CURRENT MASTER NODE. RETURN CONTROL TO SADNO.
OUTPUTF F	SEARCH EAT TABLE FOR OUTPUT FILE. CALL MOVEDAT TO MOVE DNT INFORMATION TO TREE. ADD TREE LOCATION OF DNT INFORMATION TO NODE PUSH DOWN. SET DATAN TO ZERO. TRANSFER CONTROL TO SECNODE.
PARNODE F	INSERT CURRENT NODE AS LEFT LINK OF INPUT NODE. INSERT INPUT NODE IN TREE. REPLACE CURRENT NODE IN NODE PUSH DOWN WITH TREE LOCATION OF INPUT NODE. RETURN CONTROL TO SADNO.
PERFMAS	SET UP PERFORM MASTER NODE. TRANSFER CONTROL TO MASTER.
PERTOND F	IF PQBUF IS ZERO, SET UP RIP CONTROL WORD FOR PROCEDURE NAME WITHOUT QUALIFIER. OTHERWISE, SET UP RIP CONTROL WORD FOR PROCEDURE NAME WITH QUALIFIER. CALL RIP TO SET UP PNT LOCATION. ADD PNT LOCATION TO NODE PUSH DOWN. RETURN CONTROL TO SADNO.
PLANTER F	IF NO ENTRIES IN TREE, RETURN CONTROL TO SADNO. OTHERWISE, INSERT NEXT SENTENCE LINK TYPE AS RIGHT LINK OF ALL MASTER NODES IN THE MASTER PUSH DOWN UNTIL ONLY ONE REMAINS. INSERT END-OF-SENTENCE LINK TYPE AND COUNT OF WORDS IN TREE AS RIGHT LINK OF REMAINING MASTER NODE. CALL TREEOUT TO OUTPUT TREE TO DISK. IF ENDFLAG IS NON-ZERO, CALL PHASEND TO CLOSE DISK FILE. OTHERWISE AND FOLLOWING CALL TO PHASEND, RESET PUSH DOWN AND TREE COUNTERS. RETURN CONTROL TO SADNO.
PLSNODE	SET UP PLUS NODE. TRANSFER CONTROL TO OUTNODE.
POSNODE	IF RELATION NOT BIT IS NOT ON, SET UP POSITIVE NODE AND TRANSFER CONTROL TO PARNODE. OTHERWISE, TURN OFF RELATION NOT BIT. SET UP NOT POSITIVE NODE. TRANSFER CONTROL TO PARNODE.
PROCSET	PRESET PUSH DOWN AND TREE COUNTERS. RETURN CONTROL TO SADNO.
PUSHX	NODE PUSH DOWN TABLE EXCEEDED. TRANSFER CONTROL TO CONEX2.

Figure 3-48. Routines Called by Pass 1E Syntable and/or Used to Generate the Trees (14 of 19)

QCONDNM	IF DATA TYPE IS A GROUP CONDITION NAME OR AN ELEMENTARY CONDITION NAME, RETURN CONTROL TO SADYES. OTHERWISE, RETURN CONTROL TO SADNO.
QDECL	IF DECFLAG IS ZERO, TRANSFER CONTROL TO SADNO. OTHERWISE, PRINT DIAGNOSTIC AND TRANSFER CONTROL TO SADNO.
QFIDENT	TEST SAD TABLE FLAG WORD FOR IDENTIFIER BIT. IF THE IDENTIFIER BIT AND ONLY THE IDENTIFIER BIT IS SET, RETURN CONTROL TO SADYES. OTHERWISE, RETURN CONTROL TO SADNO.
QFNUMB	TEST SAD TABLE FLAG WORD FOR NUMBER BIT. IF THE NUMBER BIT AND ONLY THE NUMBER BIT IS SET, RETURN CONTROL TO SADYES. OTHERWISE, RETURN CONTROL TO SADNO.
QGOTOND F	MOVE FIRST NAME FROM DATANAM BUFFER TO PNBUF BUFFER. IF NEXT WORD IN DATANAM BUFFER IS ZERO, SET PQBUF, QUALIFICATION FLAG TO ZERO AND RETURN CONTROL TO SADNO. OTHERWISE, MOVE NEXT NAME FROM DATANAM BUFFER TO PQBUF BUFFER. SET QUALIFICATION FLAG TO NEXT WORD IN DATANAM BUFFER. RETURN CONTROL TO SADNO.
QIF	IF FVERB IS COMPUTE, RETURN CONTROL TO SADNO. OTHERWISE, RETURN CONTROL TO SADYES.
QSUBJ	TRANSFER CONTROL TO QSUBRL.
QSUBRL	IF COMPREL IS ZERO, RETURN CONTROL TO SADNO. OTHERWISE, RETURN CONTROL TO SADYES.
QUTOTRE	SET DATANOW TO FIGURATIVE QUOTE. SET ALLLIT, DATAN, SUBFLAG TO ZERO. RETURN CONTROL TO SADNO.
RDVNODE	SET UP DIVIDE BY NODE. RETURN CONTROL TO SADNO.
READMAS	SET UP READ MASTER NODE. TRANSFER CONTROL TO MASTER.
RECORDN F	CALL DIG TO DEFINE NAME IN DATANAM BUFFER. IF NAME IS UNDEFINED, RETURN CONTROL TO SADNOSN. OTHERWISE, IF LEVEL NUMBER OF DATA ITEM IS NOT 01, RETURN CONTROL TO SADNOSN. OTHERWISE IF DATA ITEM IS NOT RECORD OF FD OR SD, RETURN CONTROL TO SADNOSN. OTHERWISE, SET DATAN TO DNT LOCATION OF RECORD ITEM, SET SUBFLAG TO ZERO AND RETURN CONTROL TO SADYES.
REEL	OR REEL BIT INTO CURRENT MASTER NODE. RETURN CONTROL TO SADNO.
RELNODE	IF RELATION NOT BIT IS OFF, TRANSFER CONTROL TO OUTNODE. OTHERWISE, TURN OFF RELATION NOT BIT. SET TYPE TO REVERSE RELATION. TRANSFER CONTROL TO OUTNODE.
RELSMAS	SET UP RELEASE MASTER NODE. TRANSFER CONTROL TO MASTER.
REPORTD	IF DATA TYPE IS RD, RETURN CONTROL TO SADYES. OTHERWISE, IF DATA TYPE IS A REPORT GROUP, RETURN CONTROL TO SADYES. OTHERWISE, IF DATA TYPE IS A REPORT DESCRIPTION, RETURN

Figure 3-48. Routines Called by Pass 1E Syntable and/or Used to Generate the Trees (15 of 19)

	CONTROL TO SADYES. OTHERWISE, RETURN CONTROL TO SADNO.
REPORTN F	CALL SCAN TO COLLECT NEXT CHARACTER STRING. IF CHARACTER STRING IS NOT NAME, RETURN CONTROL TO SADNOSN. OTHERWISE, MOVE NAME TO DATANAM BUFFER. CALL DIG TO DEFINE NAME. IF NAME IS UNDEFINED, RETURN CONTROL TO SADNOSN. OTHERWISE, IF DATA TYPE IS NOT RD, RETURN CONTROL TO SADNOSN. OTHERWISE, SET DATAN TO THE DNT LOCATION OF RD INFORMATION. SET SUBFLAG TO ZERO. RETURN CONTROL TO SADYES.
RETRMAS	SET UP RETURN MASTER NODE. TRANSFER CONTROL TO MASTER.
RLMASTX F	IF NO ENTRIES IN TREE, SET FKFLAG, LASTMAS TO ZERO AND RETURN CONTROL TO SADNO. OTHERWISE, IF MASTER NODE IS A FORK, SET FKFLAG TO ZERO AND RETURN CONTROL TO SADNO. OTHERWISE, INSERT THE NEXT AVAILABLE LOCATION IN THE TREE AS THE RIGHT LINK OF THE CURRENT MASTER NODE. SET LASTMAS TO CURRENT MASTER NODE. CANCEL CURRENT MASTER NODE FROM PUSH DOWN. RETURN CONTROL TO SADNO.
RMTOTRE	SET DATANOW TO FIGURATIVE RECORD-MARK. SET ALLLIT, DATAN, SUBFLAG TO ZERO. RETURN CONTROL TO SADNO.
ROUND	SET ROUNDER TO NON-ZERO. RETURN CONTROL TO SADNO.
ROUTINA	TRANSFER CONTROL TO NAME.
RPLACNG	SET TYPE TO EXAMINE REPLACING. RETURN CONTROL TO SADNO.
RPTSRCH	ADD NULL NODE TO NODE PUSH DOWN. RETURN CONTROL TO SADNO.
RRLNODE	EXCHANGE CURRENT NODE AND CURRENT NODE - 1. TRANSFER CONTROL TO RELNODE.
RSBNODE	SET UP SUBTRACT FROM NODE. TRANSFER CONTROL TO OUTNODE.
RSNEXT	RESET TREE POINTER TO BEGINNING OF CURRENT MASTER NODE. IF INITIAL ENTRY, RESET MASTER NODE PUSH DOWN POINTER AND RETURN CONTROL TO SADNO. OTHERWISE, REPLACE CURRENT MASTER NODE IN PUSH DOWN WITH THE C(LASTMAS). CANCEL RIGHT LINK OF THE NODE POINTED TO BY LASTMAS. RETURN CONTROL TO SADNO.
RVERSED	OR REVERSED BIT INTO CURRENT MASTER NODE. RETURN CONTROL TO SADNO.
SAVESRO	SET COMPREL TO POINT TO CURRENT NODE IN THE NODE PUSH DOWN. SET SUBHELD, OBJHELD TO ZERO. RETURN CONTROL TO SADNO.
SBW	SET NOSCNFL TO NON-ZERO. RETURN CONTROL TO SADNO.
SCNANW	SET SCNOTNW TO NON-ZERO. RETURN CONTROL TO SADNO.
SECNODE	SET UP SECONDARY NODE TYPE.

Figure 3-48. Routines Called by Pass 1E Syntable and/or Used to Generate the Trees (16 of 19)

	TRANSFER CONTROL TO OUTNODE.
SEEKMAS	SET UP SEEK MASTER NODE. TRANSFER CONTROL TO MASTER.
SEMICOL	IF SCNOTNW IS NON-ZERO RETURN CONTROL TO SADNO. OTHERWISE, SET SEMIFLG TO ZERO. THEN IF XTENDMD IS NON-ZERO, RETURN CONTROL TO SADNO. OTHERWISE, SET COMAFLG TO ZERO AND RETURN CONTROL TO SADNO.
SETAFT	OR (USE) AFTER BIT INTO OPTION. RETURN CONTROL TO SADNO.
SETBEG	OR (USE) BEGINNING BIT INTO OPTION. RETURN CONTROL TO SADNO.
SETDECL	SET DECFLAG TO NON-ZERO. RETURN CONTROL TO SADNO.
SETEND	OR (USE) ENDING BIT INTO OPTION. RETURN CONTROL TO SADNO.
SETFILE	OR (USE) FILE BIT INTO OPTION. RETURN CONTROL TO SADNO.
SETHOLD	SET HOLDER TO CURRENT NODE - 1. SET GRABER TO THE NEXT TREE LOCATION. TRANSFER CONTROL TO SADNO.
SETJTAB	INSERT C(CURRSEC) IN JUMP TABLE + C(JUMPER). SET JUMPER TO JUMPER + 1. RETURN CONTROL TO SADNO.
SETRCRD	SET MOVRCRD TO NEXT SOURCE ITEM. SET MOVFLAG TO ZERO. RETURN CONTROL TO SADNO.
SETREEL	OR (USE) REEL BIT INTO OPTION. RETURN CONTROL TO SADNO.
SIMPKEY	CALL SCAN TO COLLECT CHARACTER STRING. IF CHARACTER STRING IS A SEMI-IMPERATIVE KEY WORD, SET NOSCNFL TO NON-ZERO AND RETURN CONTROL TO SADYES. OTHERWISE, RETURN CONTROL TO SADNOSN.
SIZERR	SET ON SIZE ERROR BIT IN MASTER NODE POINTED TO BY OSENODE. TRANSFER CONTROL TO LLMASTN.
SNC	SET SKIPOPS TO 2. RETURN CONTROL TO SADNO.
SNP	SET SKIPOPS TO 4. RETURN CONTROL TO SADNO.
SNW	SET NOSCNFL TO ZERO. RETURN CONTROL TO SADNO.
SORTMAS	SET UP SORT MASTER NODE. TRANSFER CONTROL TO MASTER.
SPTOTRE	SET DATANOW TO FIGURATIVE SPACE(S). SET ALLLIT, DATAN, SUBFLAG TO ZERO. RETURN CONTROL TO SADNO.
STONODE	IF ROUNDER IS ZERO, SET UP STORE NODE AND TRANSFER CONTROL TO OUTNODE. OTHERWISE, SET ROUNDER TO ZERO, SET UP STORE ROUNDED NODE AND TRANSFER CONTROL TO OUTNODE.
STPLMAS	SET UP STOP LITERAL MASTER NODE. TRANSFER CONTROL TO MASTER.
STPRMAS	SET UP STOP RUN MASTER NODE. TRANSFER CONTROL TO MASTER.
SUBGRBR F	IF SUBPNT IS ZERO, SET UP HOLD NODE. RIGHT LINK IS TEMPORARY2. LEFT LINK IS C(SUBHOLD). INSERT HOLD NODE IN TREE. SET SUBPNT TO NON-ZERO. LEFT LINK OF NODE POINTED TO BY DATANOW IS TREE LOCATION OF HOLD NODE.

Figure 3-48. Routines Called by Pass 1E Syntable and/or Used to Generate the Trees (17 of 19)

	OTHERWISE AND FOLLOWING INSERTION OF HOLD NODE, SET UP GRAB NODE. RIGHT LINK IS NULL. LEFT LINK IS TEMPORARY2. INSERT GRAB NODE IN TREE. ADD TREE LOCATION OF GRAB NODE TO NODE PUSH DOWN. ADD LEFT LINK OF NODE POINTED BY DATANOW TO NODE PUSH DOWN. TRANSFER CONTROL TO OUTNODE.
SUBNODE F	SET UP SUBSCRIPTING NODE. RIGHT LINK IS OPEN. INSERT CURRENT NODE AS LEFT LINK OF INPUT NODE. INSERT INPUT NODE IN TREE. SET SUBGRAB TO TREE LOCATION. SET DATANOW TO TREE LOCATION. SET SUBHOLD TO LOCATION OF CURRENT NODE. SET SUBPNT TO ZERO. SET SUBFLAG TO NON-ZERO. CANCEL CURRENT NODE FROM PUSH DOWN. RETURN CONTROL TO SADNO.
SUBRNAM	TRANSFER CONTROL TO NAME.
SUBTMAS	SET UP SUBTRACT MASTER NODE. SET TYPE TO SUBTRACT FROM. TRANSFER CONTROL TO MASTER.
SUBTOND	ADD C(DATANOW) TO NODE PUSH DOWN. RETURN CONTROL TO SADNO.
SUPDIAG	SET NOTDAG TO NON-ZERO RETURN CONTROL TO SADNO.
SWCHDIV	SET TYPE TO DIVIDE BY NODE. RETURN CONTROL TO SADNO.
SWITCHN F	CALL SCAN TO COLLECT NEXT CHARACTER STRING. IF CHARACTER STRING IS NOT NAME, RETURN CONTROL TO SADNOSN. OTHERWISE, MOVE NAME TO DATANAM BUFFER. CALL DIG TO DEFINE NAME. IF NAME IS UNDEFINED, RETURN CONTROL TO SADNOSN. OTHERWISE, IF NAME IS NOT A SPECIAL NAME, RETURN CONTROL TO SADNOSN. OTHERWISE, SET DATAN TO DNT LOCATION OF SPECIAL NAME. SET SUBFLAG TO ZERO. RETURN CONTROL TO SADYES.
SWITCHR	SET UP SWITCH STATUS NODE. ADD NODE TO NODE PUSH DOWN. RETURN CONTROL TO SADNO.
TALLYNG	SET TYPE TO EXAMINE TALLYING. RETURN CONTROL TO SADNO.
TERMMAS	SET UP TERMINATE MASTER NODE. TRANSFER CONTROL TO MASTER.
TESTPNT	IF POINTER IS NON-ZERO, RETURN CONTROL TO SADNO. OTHERWISE, RETURN CONTROL TO SADYES.
TFFORK	SET UP TRUE-FALSE FORK. SET FKFLAG TO NON-ZERO TRANSFER CONTROL TO MASFORK.
THRNODE	SET UP PERFORM THRU NODE. TRANSFER CONTROL TO OUTNODE.
TIMNODE	SET UP TIMES NODE. TRANSFER CONTROL TO ORV-NODE.
TIS	TEST SAD TABLE FLAG WORD FOR IS BIT. IF IS BIT IS SET, RETURN CONTROL TO SADYES. OTHERWISE, RETURN CONTROL TO SADNO.
TNOT	TEST SAD TABLE FLAG WORD FOR RELATION NOT BIT. IF RELATION NOT BIT IS SET, RETURN CONTROL TO SADYES. OTHERWISE, RETURN CONTROL TO SADNO.
TREEX	TREE TABLE EXCEEDED. TRANSFER CONTROL TO CONEX2.

Figure 3-48. Routines Called by Pass 1E Syntable and/or Used to Generate the Trees (18 of 19)

TYPNODE	SET UP NODE AS DIRECTED BY C(TYPE). TRANSFER CONTROL TO OUTNODE.
UMNNODE	SET UP UNARY MINUS NODE. TRANSFER CONTROL TO PARNODE.
UNDECL	SET DECFLAG TO ZERO. RETURN CONTROL TO SADNO.
UNRNODE	SET UP PERFORM UNTIL NODE. TRANSFER CONTROL TO ORVNODE.
UNTLOPT	OR EXAMINE UNTIL (FIRST) BIT INTO TYPE. RETURN CONTROL TO SADNO.
UNTNODE	SET UP PERFORM UNTIL NODE. TRANSFER CONTROL TO OUTNODE.
USENODE	SET UP SORT USING NODE. TRANSFER CONTROL TO OUTNODE.
VARNODE	SET UP PERFORM VARYING NODE. TRANSFER CONTROL TO ORVNODE.
VSTOTRE F	SET UP DIG CONTROL WORD. CALL DIG TO FIND ENTRY IN DNT. IF ENTRY IS AN ILLEGAL SUBSCRIPT, ISSUE A DIAGNOSTIC AND RETURN CONTROL TO SADNO. OTHERWISE, CALL MOVEDAT TO MOVE DNT INFORMATION TO TREE. SET DATANOW TO TREE LOCATION OF DNT INFORMATION. RETURN CONTROL TO SADYES.
WRITMAS	SET UP WRITE MASTER NODE. TRANSFER CONTROL TO MASTER.
X	RETURN CONTROL TO SADNO.
XX	RETURN CONTROL TO SADNO.
ZERNODE	IF RELATION NOT BIT IS NOT ON, SET UP ZERO NODE AND TRANSFER CONTROL TO PARNODE. OTHERWISE, TURN OFF RELATION NOT BIT. SET UP NOT ZERO NODE. TRANSFER CONTROL TO PARNODE.
ZRTOTRE	SET DATANOW TO FIGURATIVE ZERO(S). SET ALLLIT, DATAN, SUBFLAG TO ZERO. RETURN CONTROL TO SADNO.

Figure 3-48. Routines Called by Pass 1E Syntable and/or Used to Generate the Trees (19 of 19)

PASS 1E FLOWCHARTS

Figures 3-49 through 3-51 on the following pages shows the flowcharts for Pass 1E, INCLIB and INC.

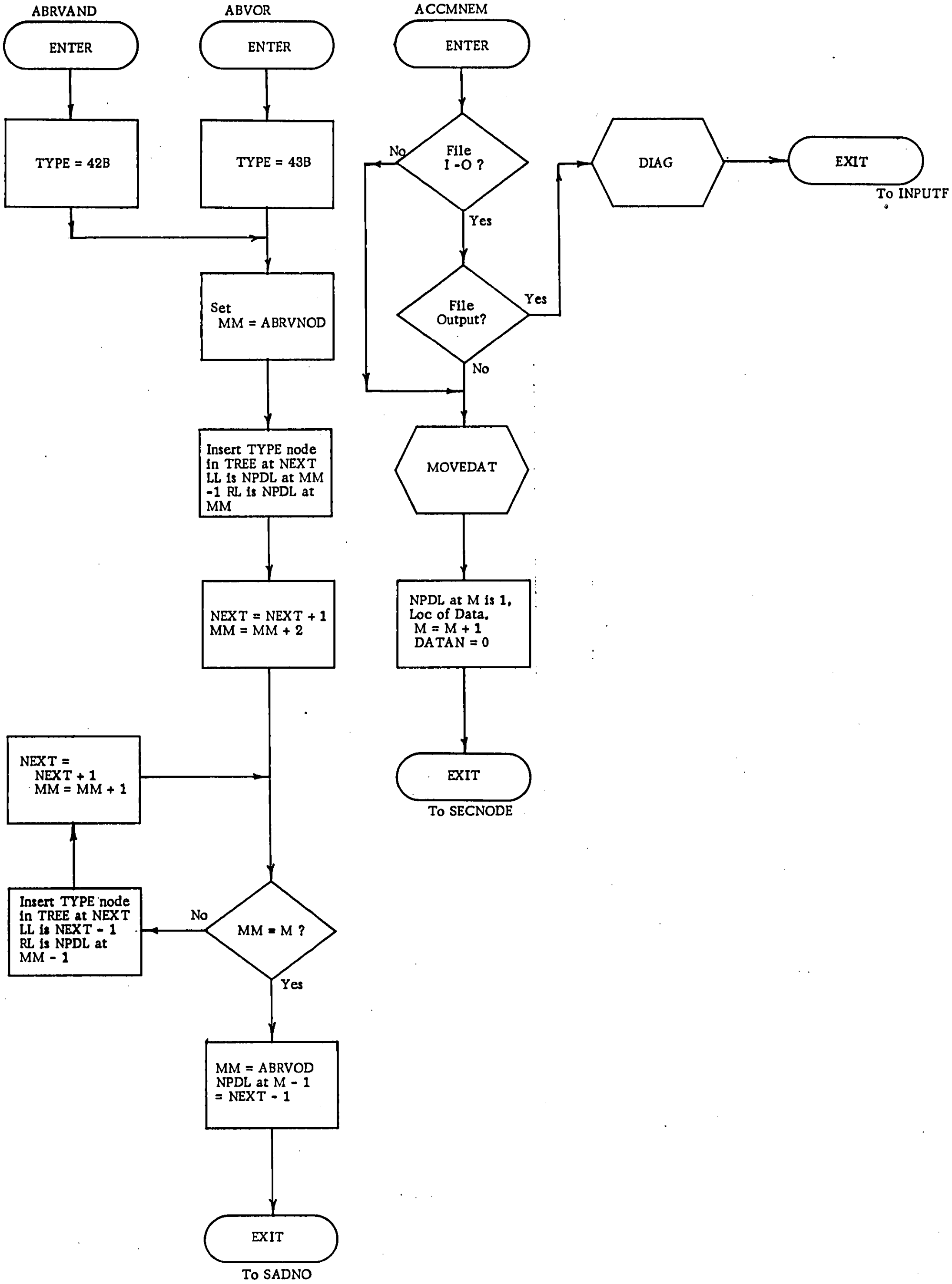
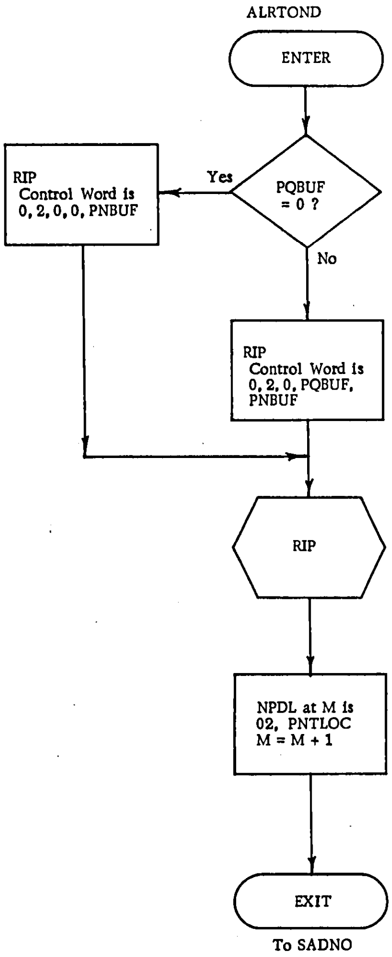


Figure 3-49. Pass 1E Flowchart (1 of 22)



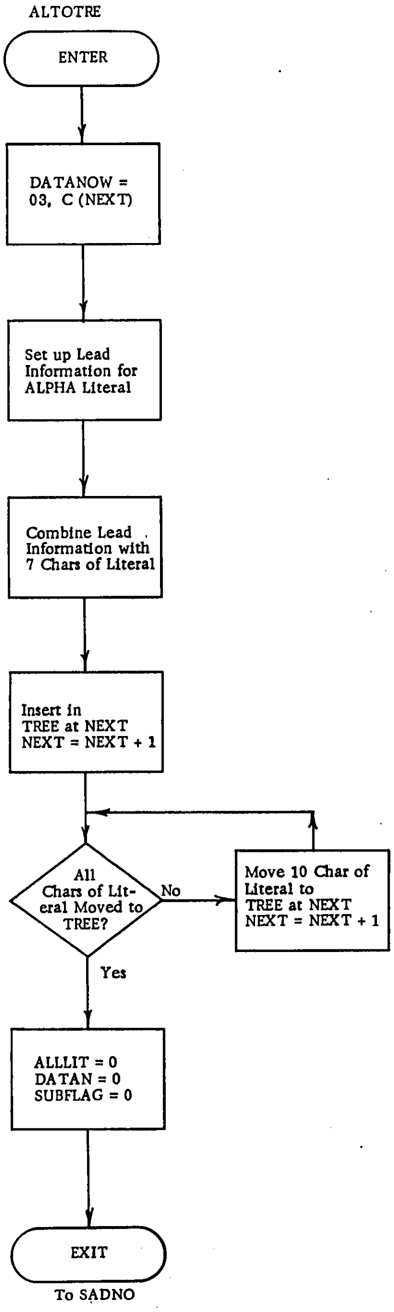


Figure 3-49. Pass 1E Flowchart (2 of 22)

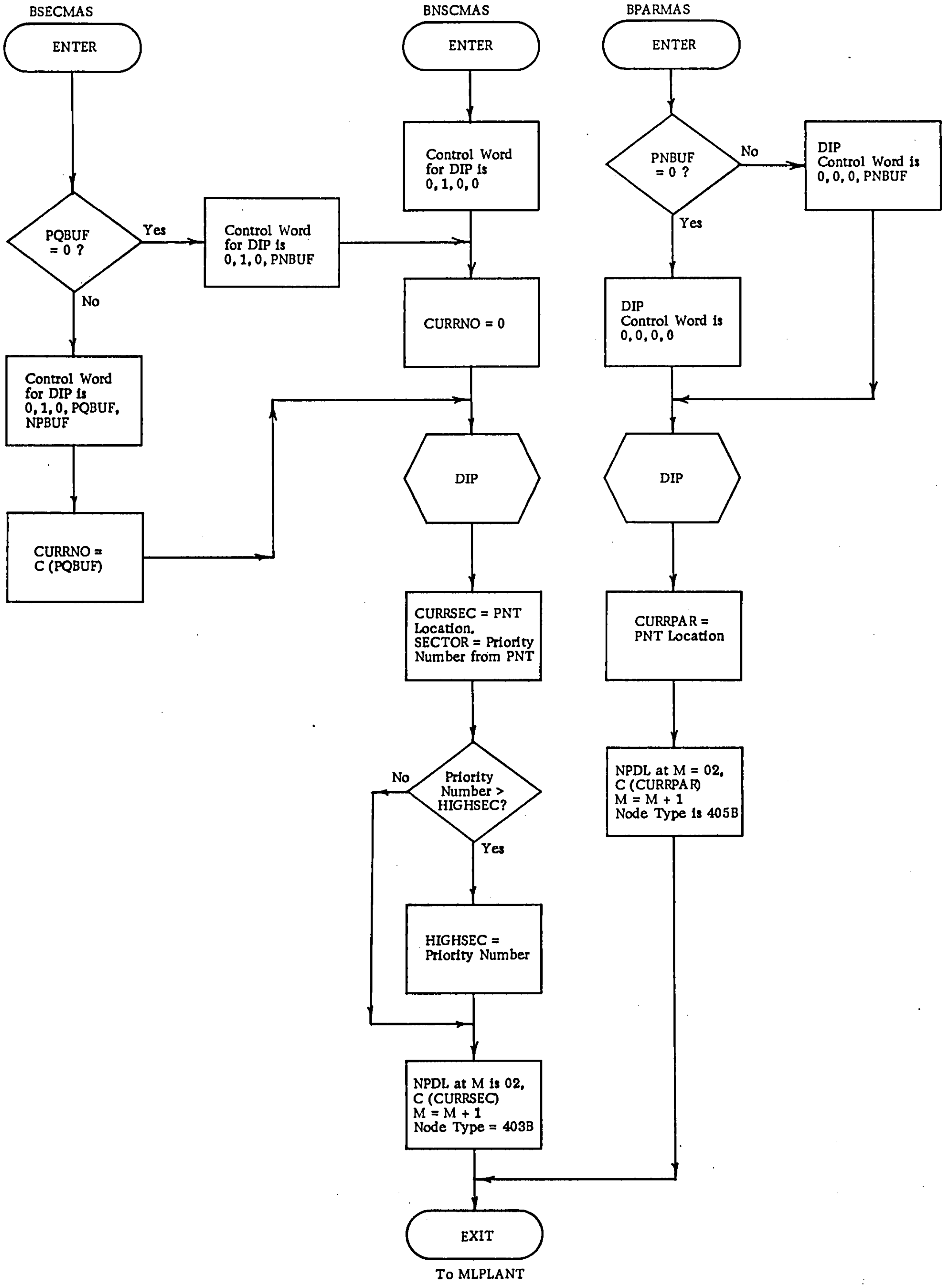


Figure 3-49. Pass 1E Flowchart (3 of 22)

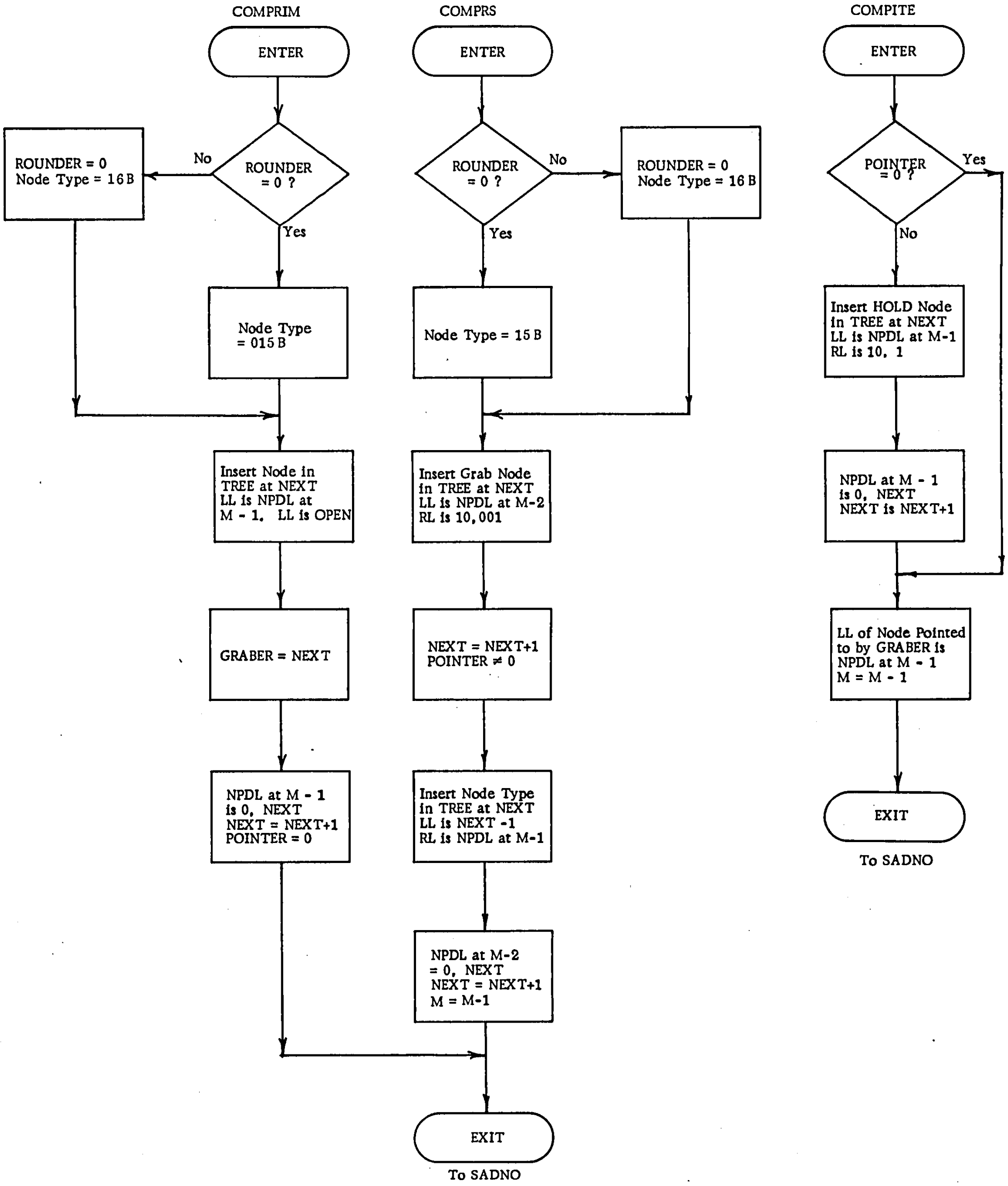


Figure 3-49. Pass 1E Flowchart (4 of 22)

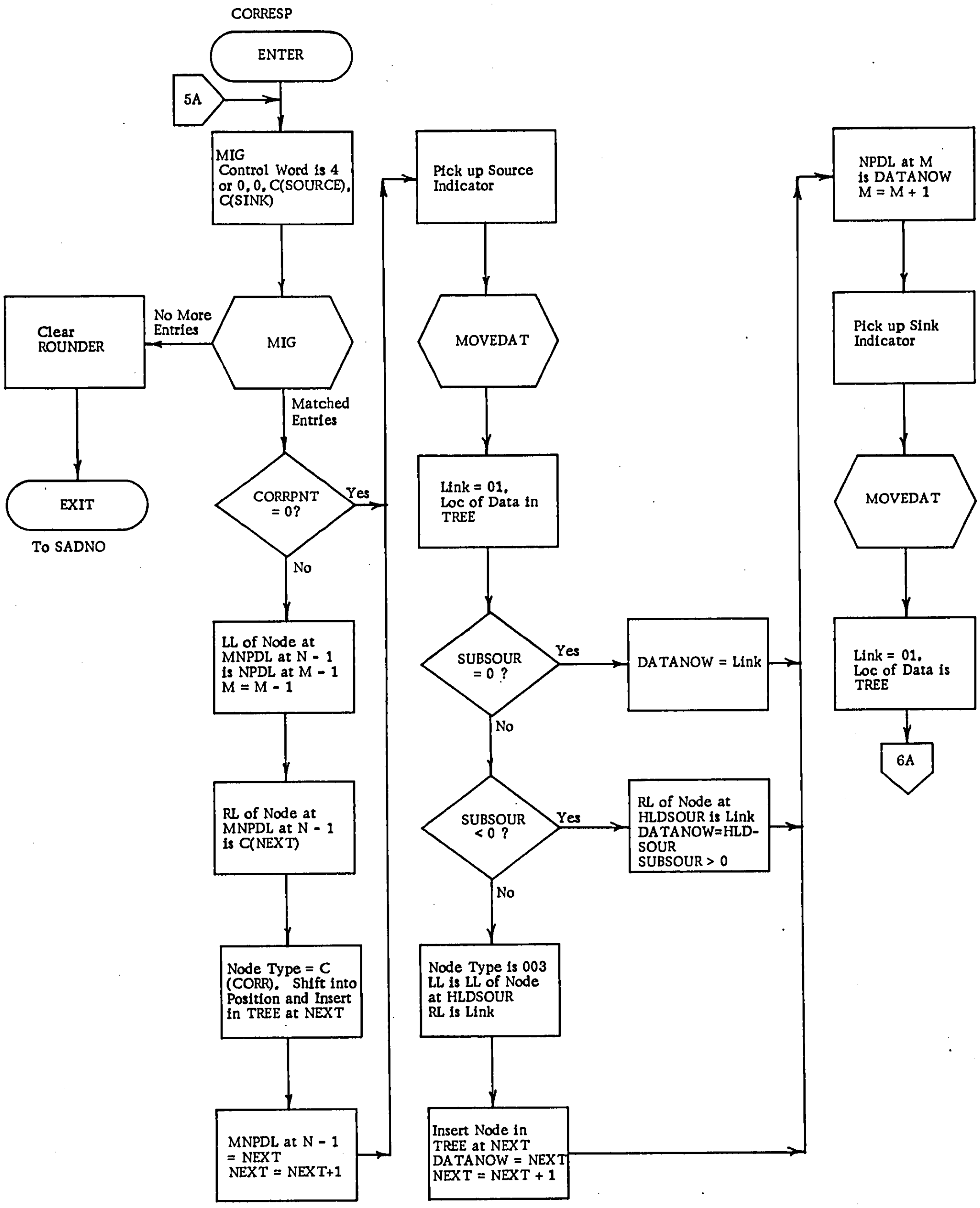


Figure 3-49. Pass 1E Flowchart (5 of 22)

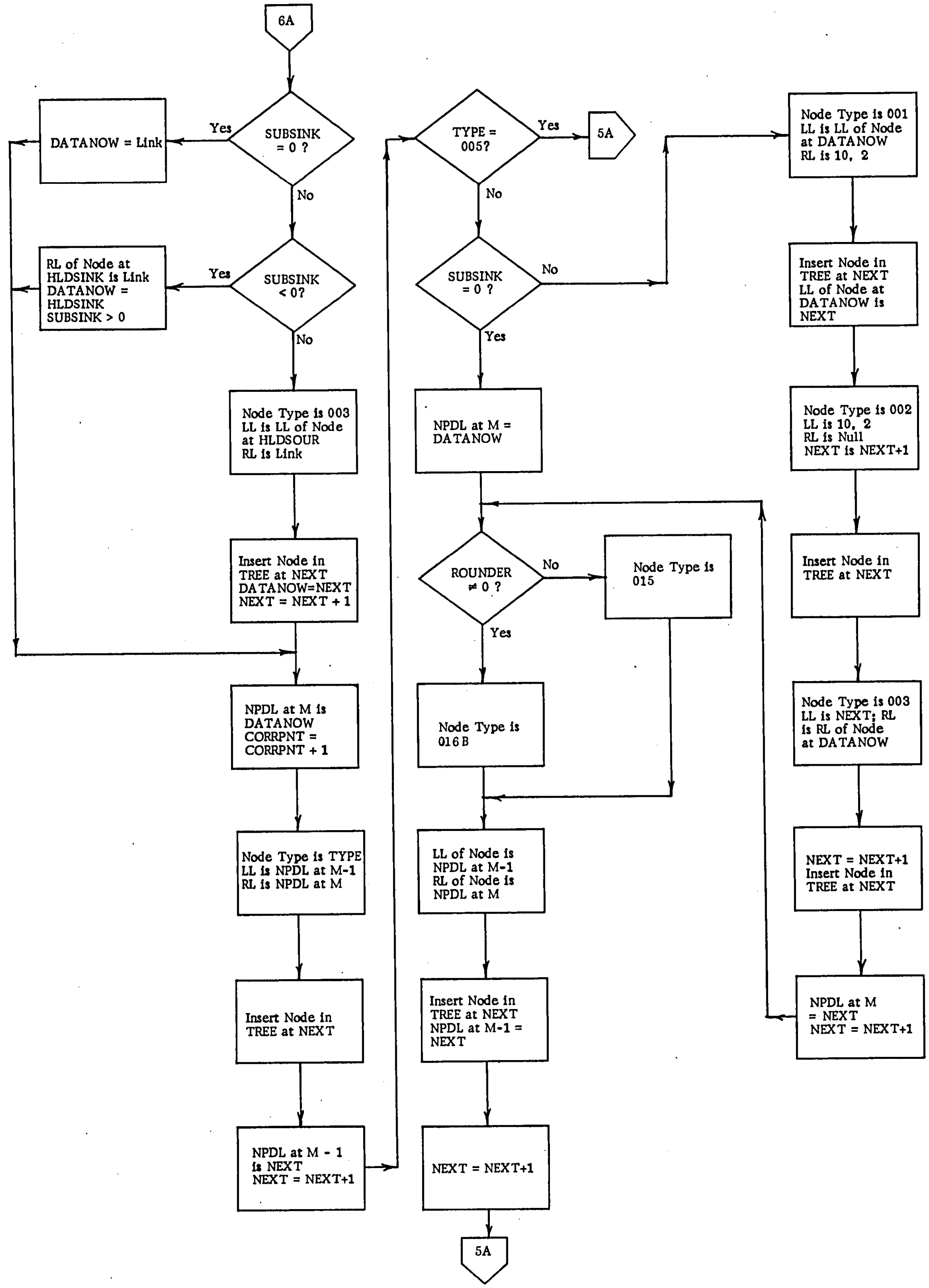


Figure 3-49. Pass 1E Flowchart (6 of 22)

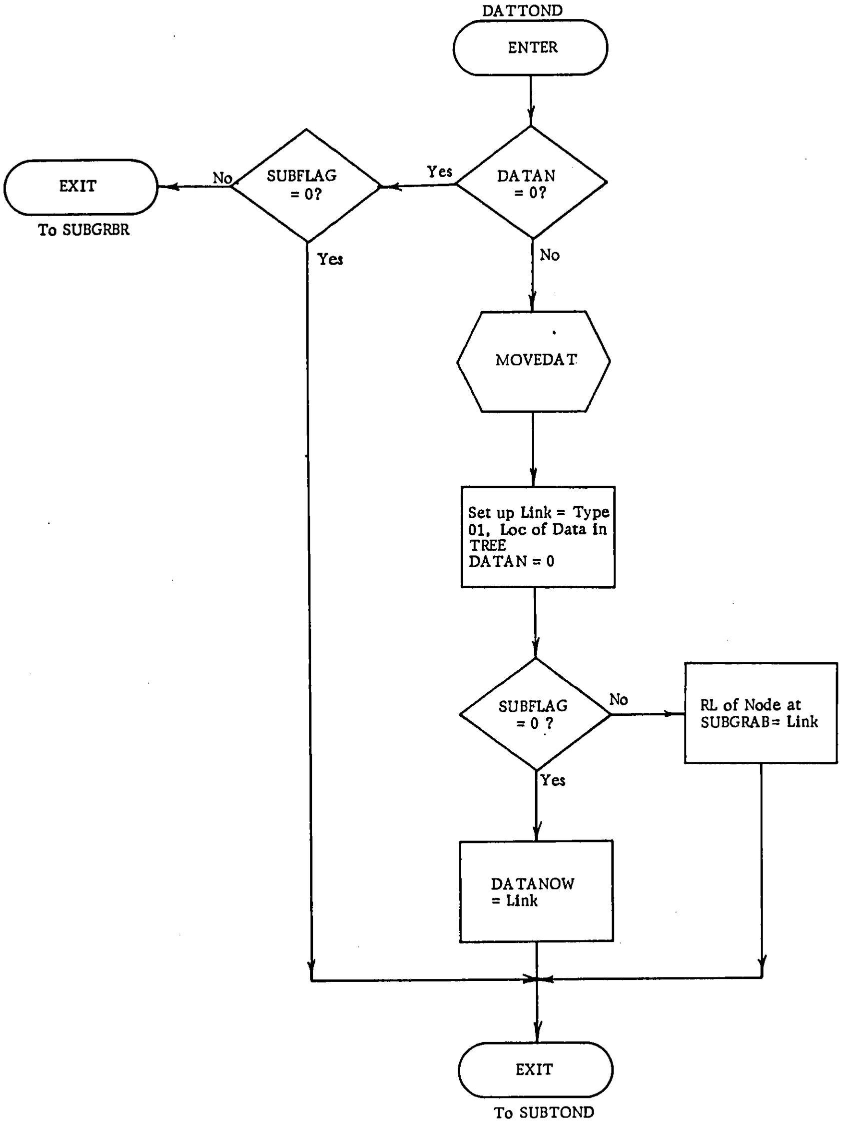


Figure 3-49. Pass 1E Flowchart (7 of 22)

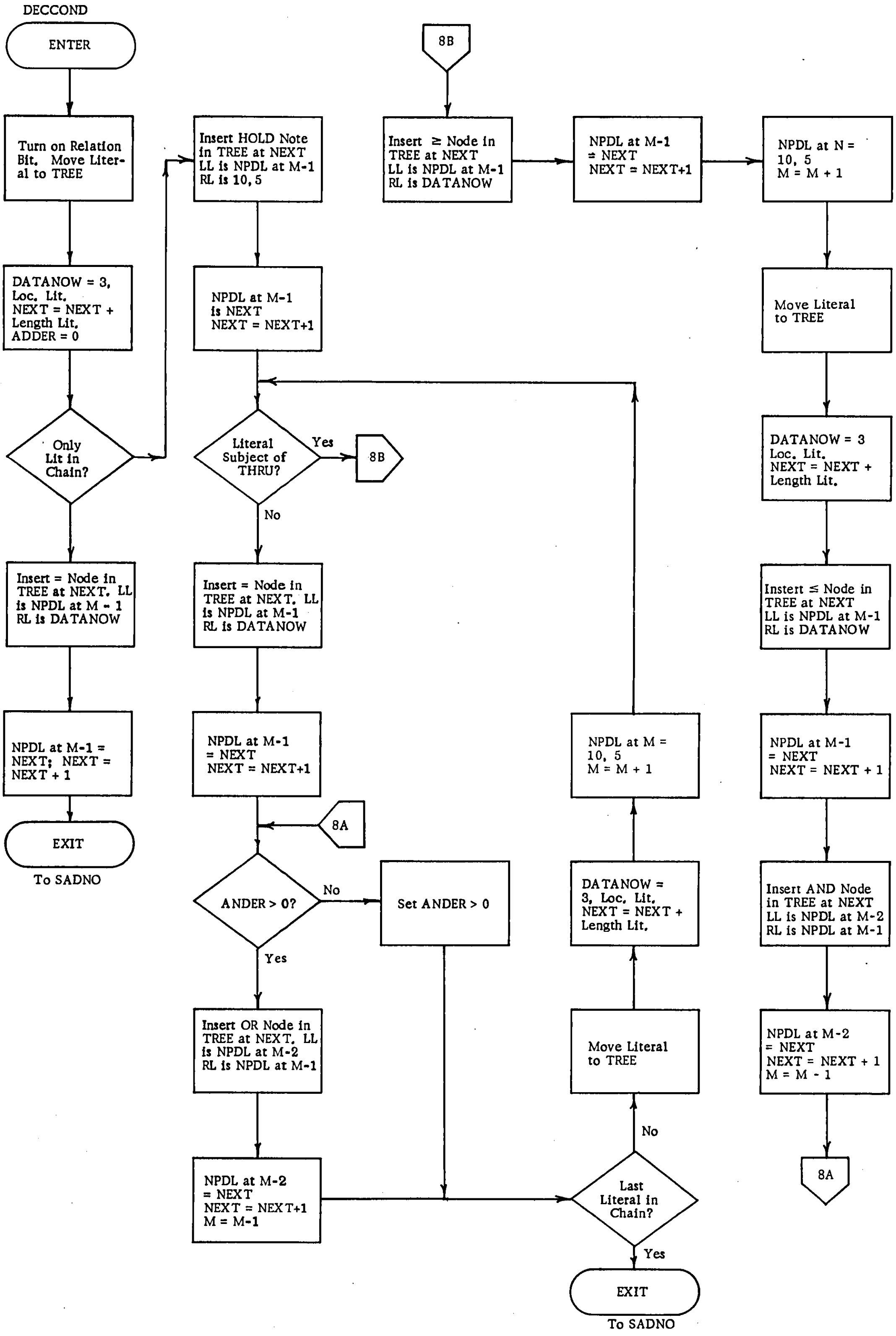


Figure 3-49. Pass 1E Flowchart (8 of 22)

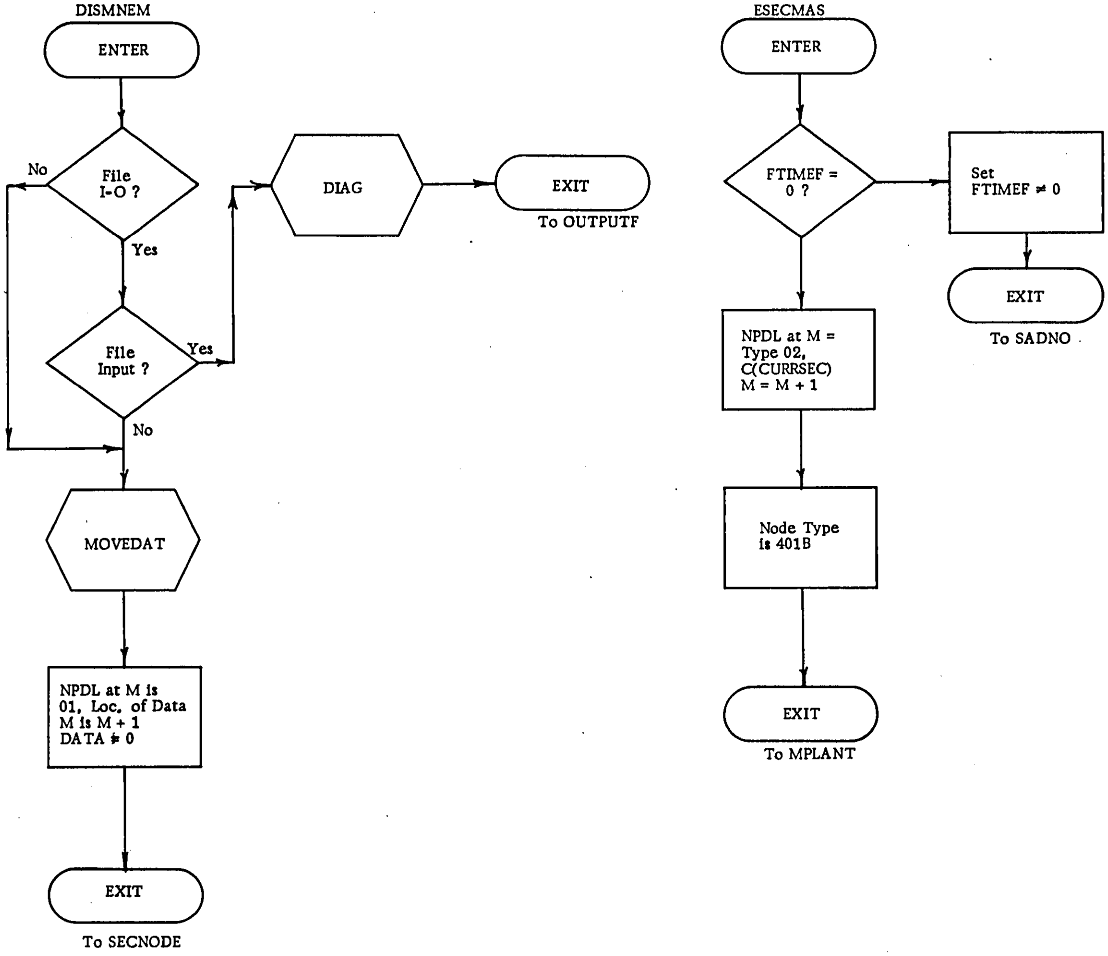


Figure 3-49. Pass 1E Flowchart (9 of 22)

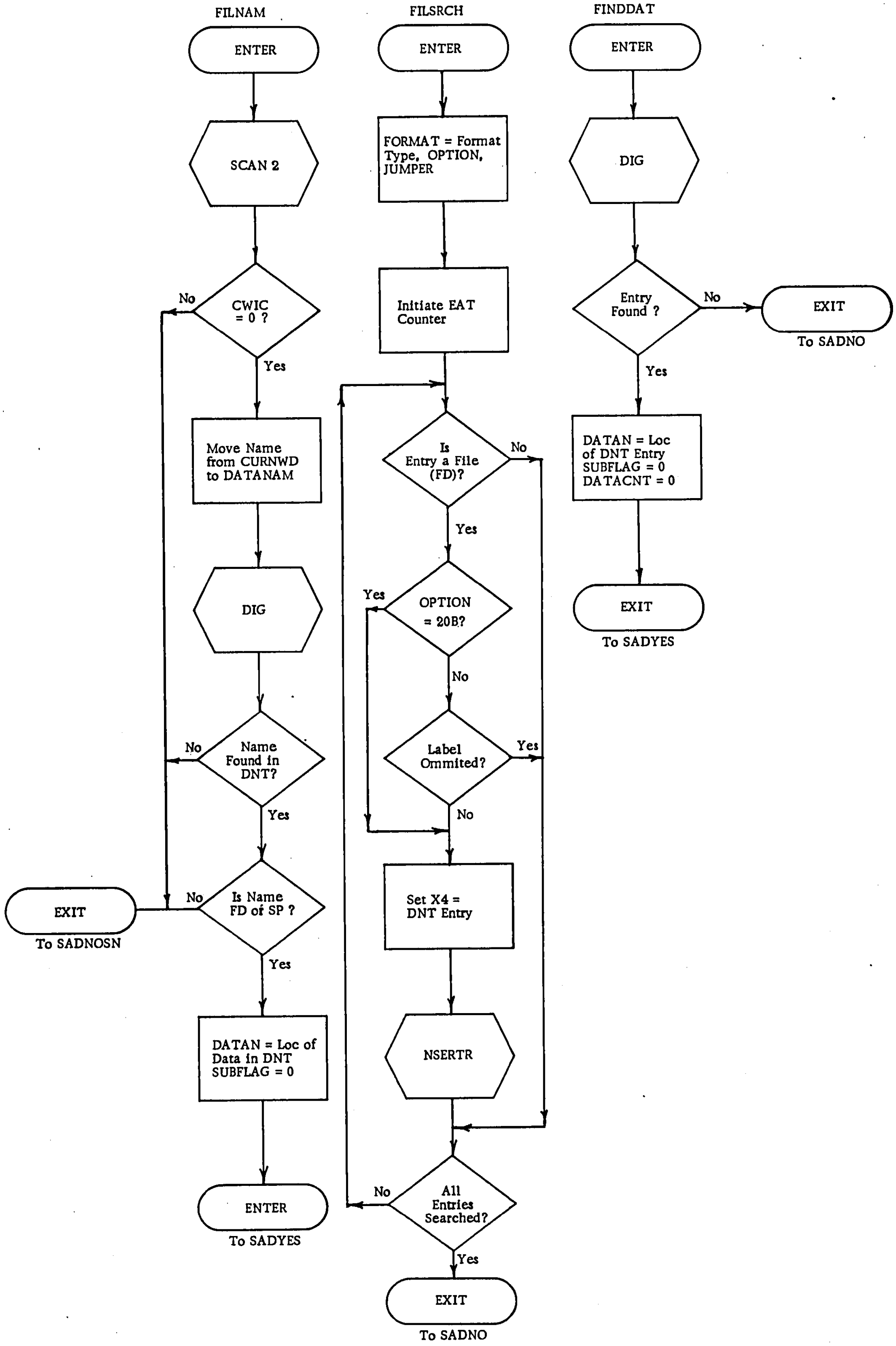


Figure 3-49. Pass 1E Flowchart (10 of 22)

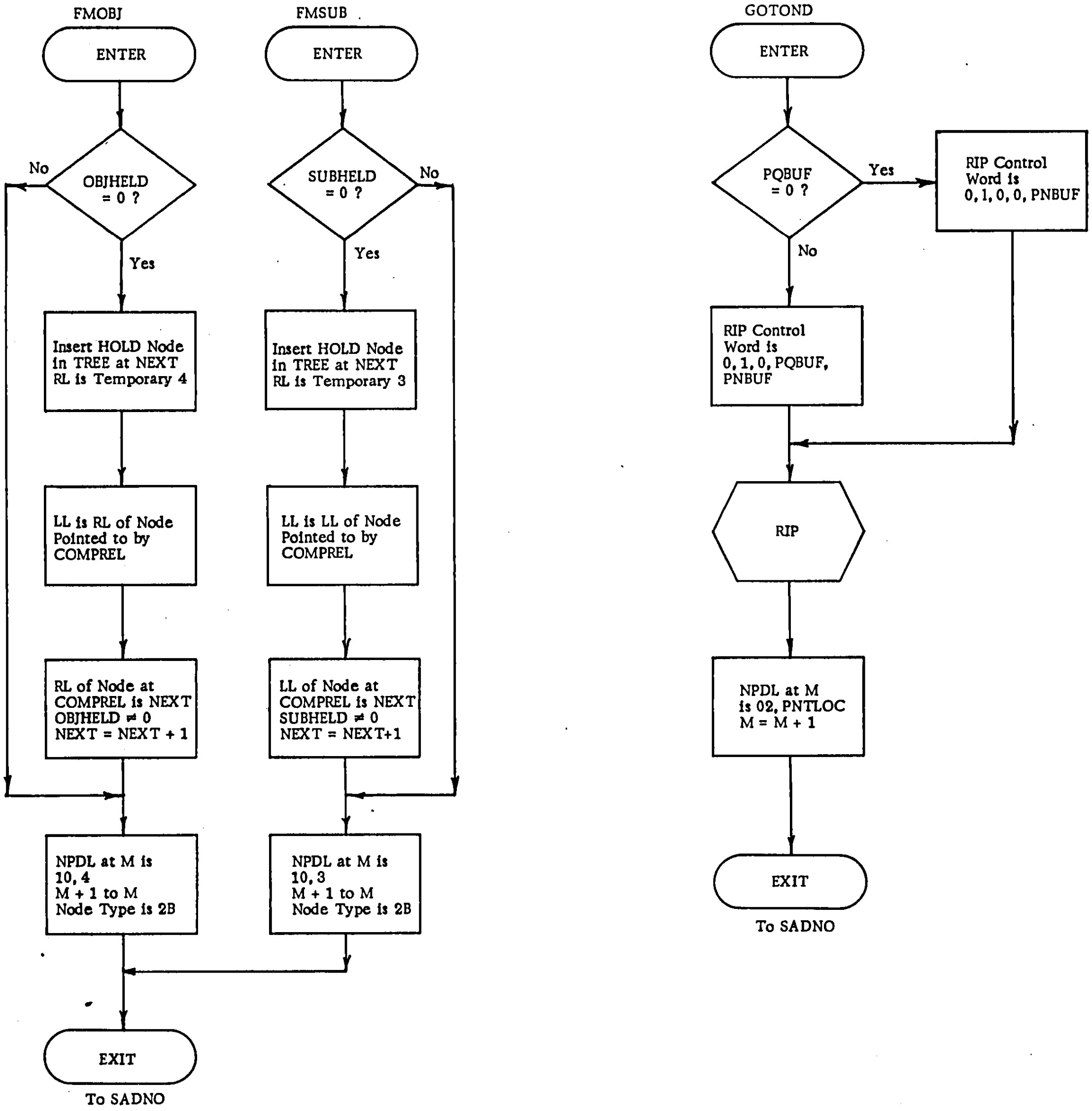


Figure 3-49. Pass 1E Flowchart (11 of 22)

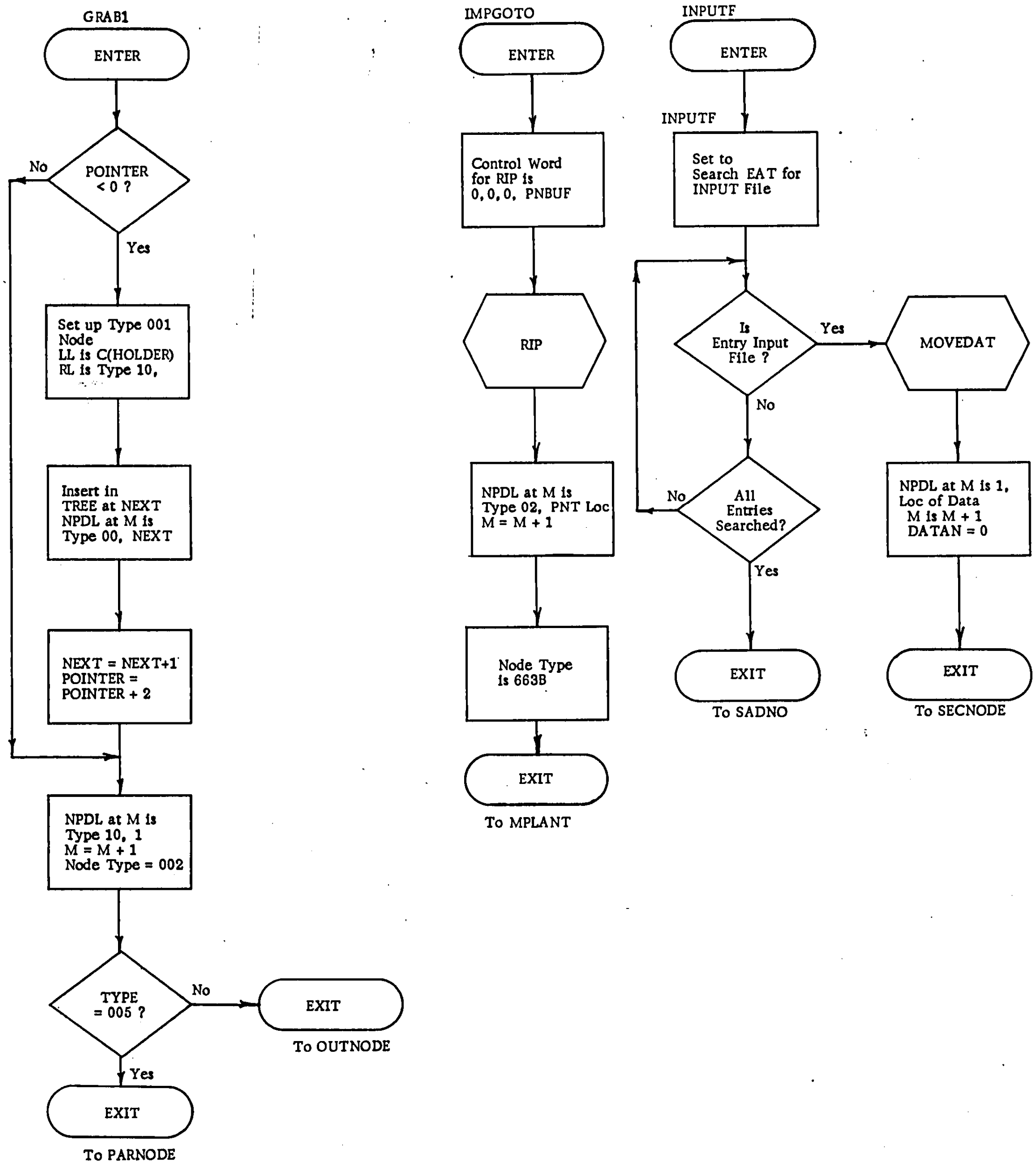


Figure 3-49. Pass 1E Flowchart (12 of 22)

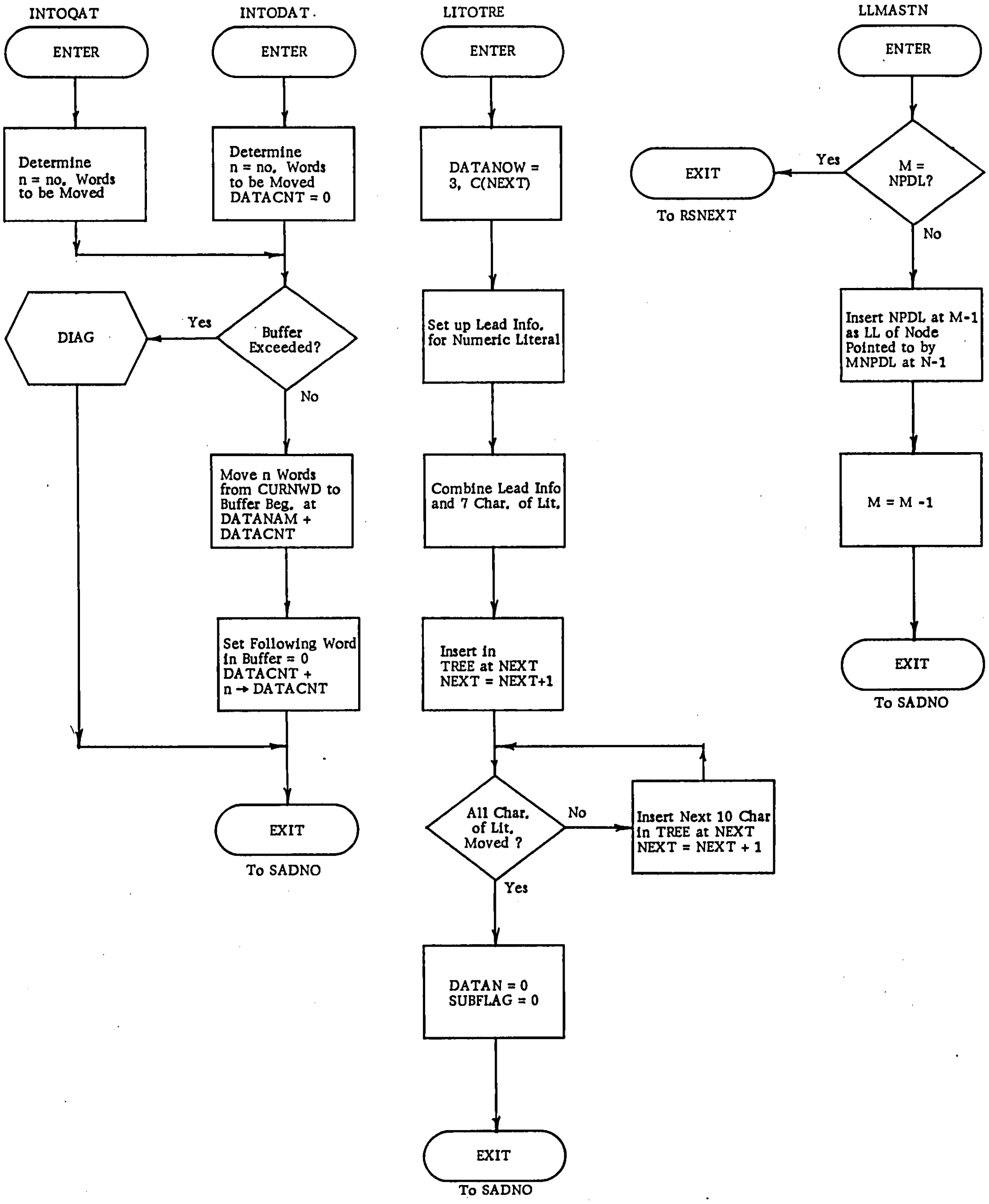


Figure 3-49. Pass 1E Flowchart (13 of 22)

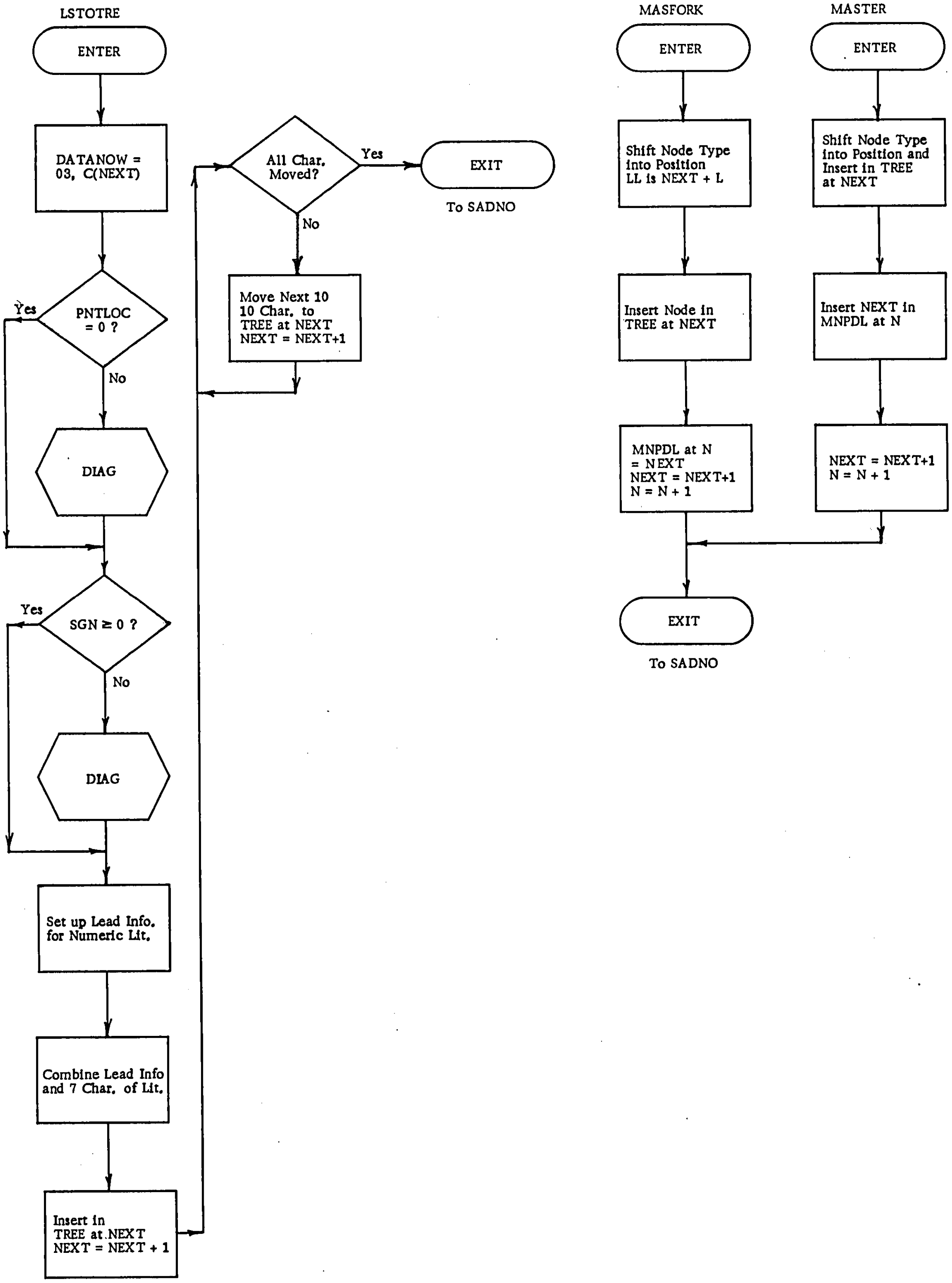


Figure 3-49. Pass 1E Flowchart (14 of 22)

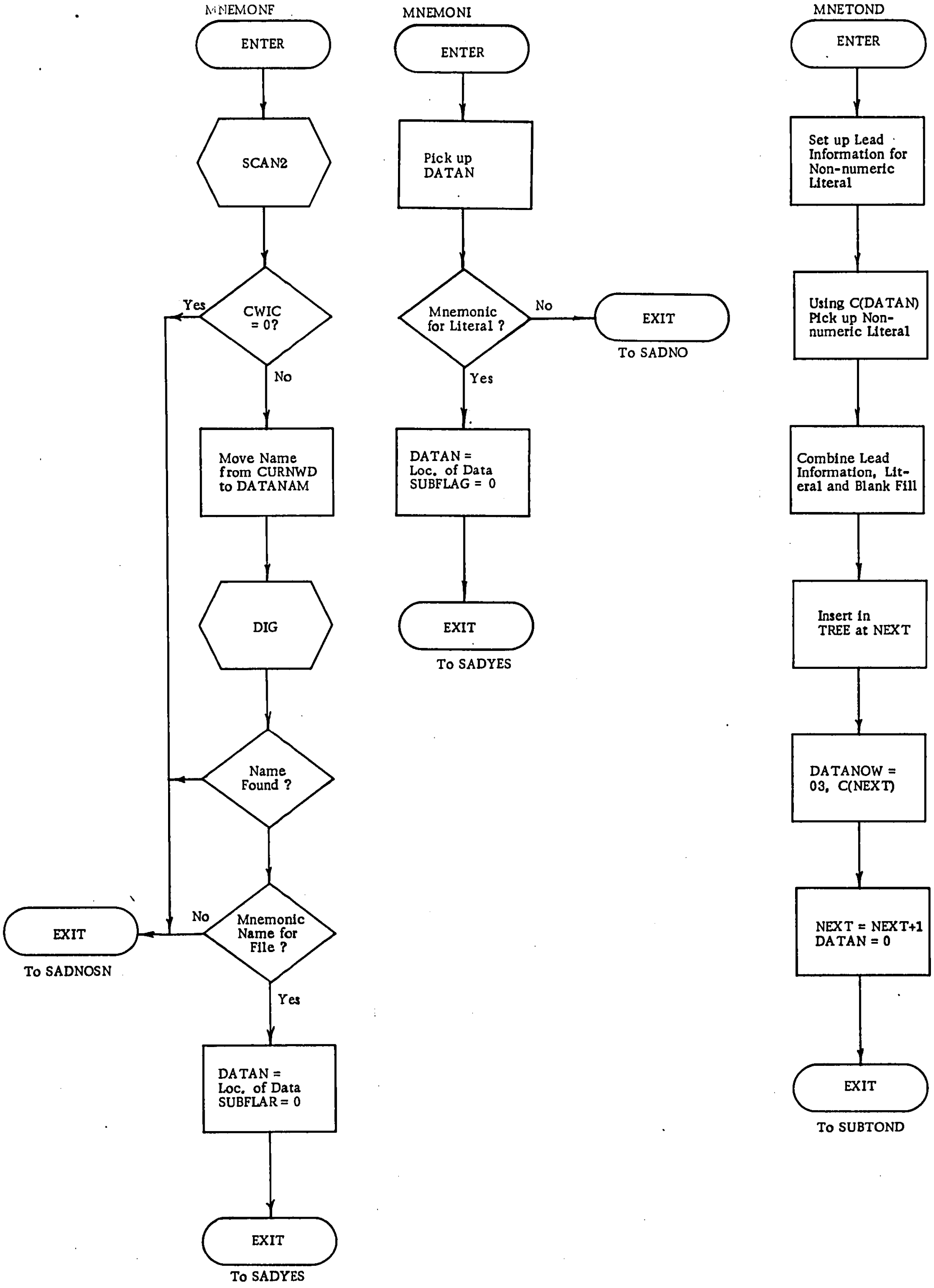


Figure 3-49. Pass 1E Flowchart (15 of 22)

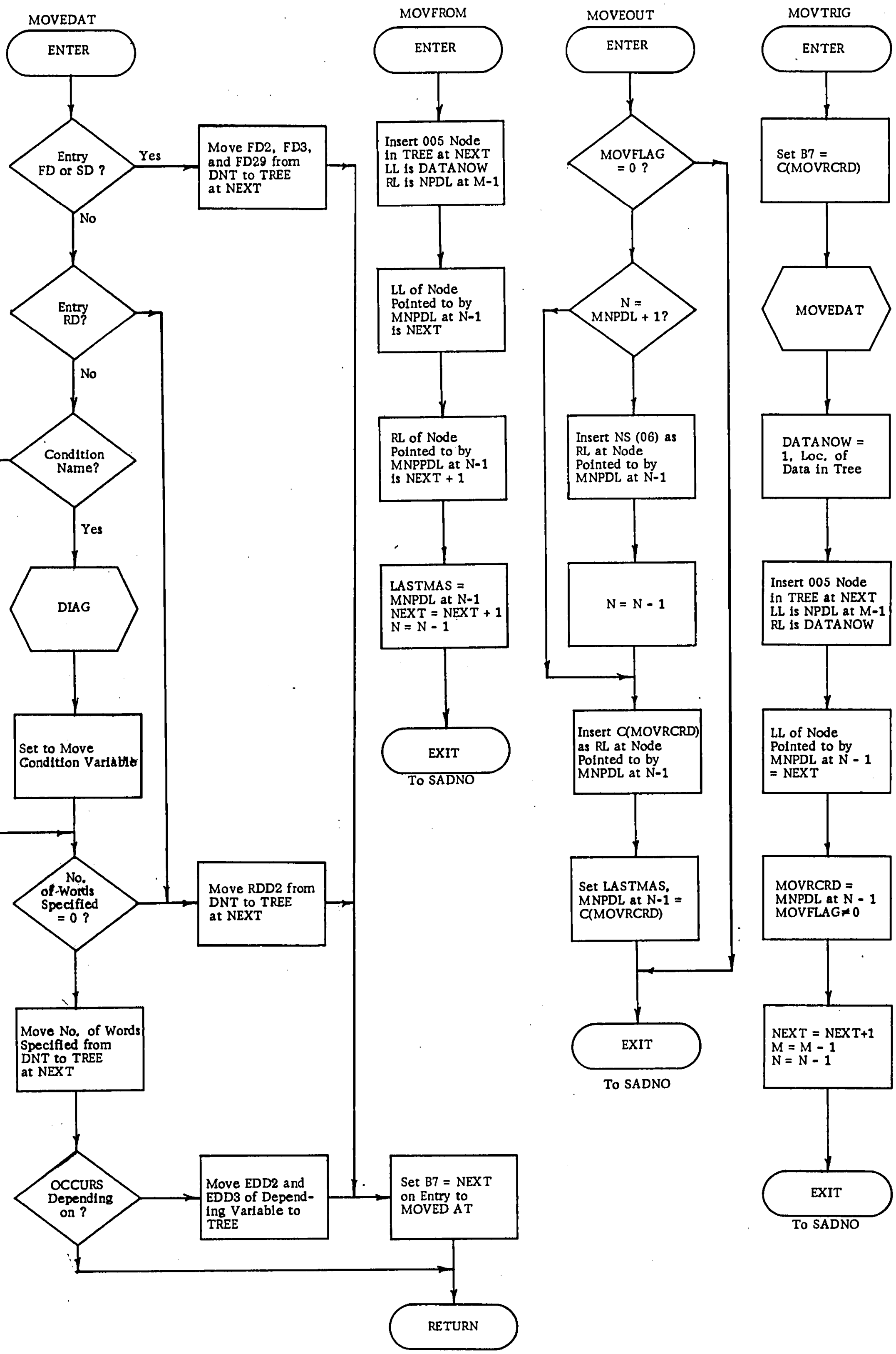


Figure 3-49. Pass 1E Flowchart (16 of 22)

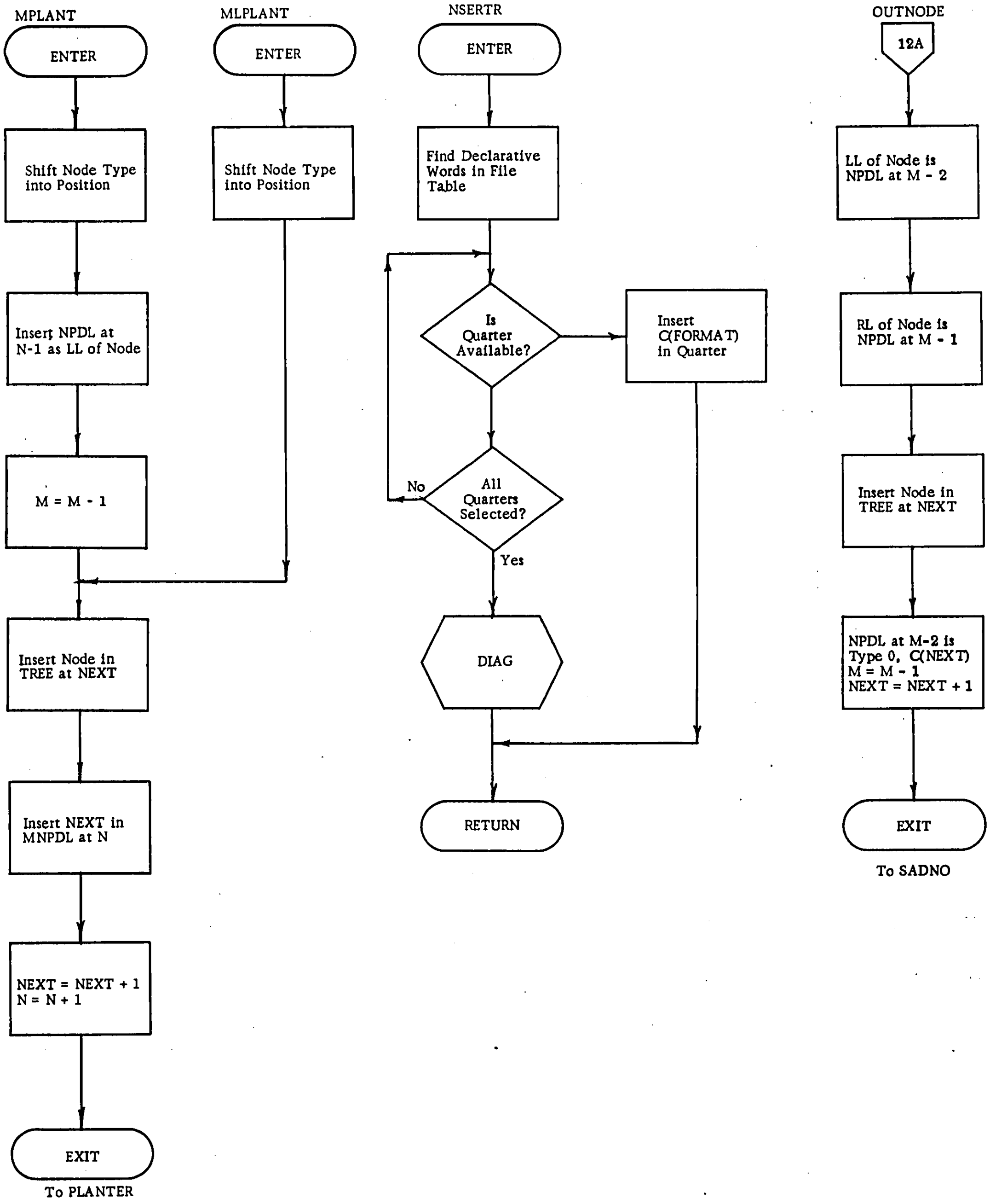


Figure 3-49. Pass 1E Flowchart (17 of 22)

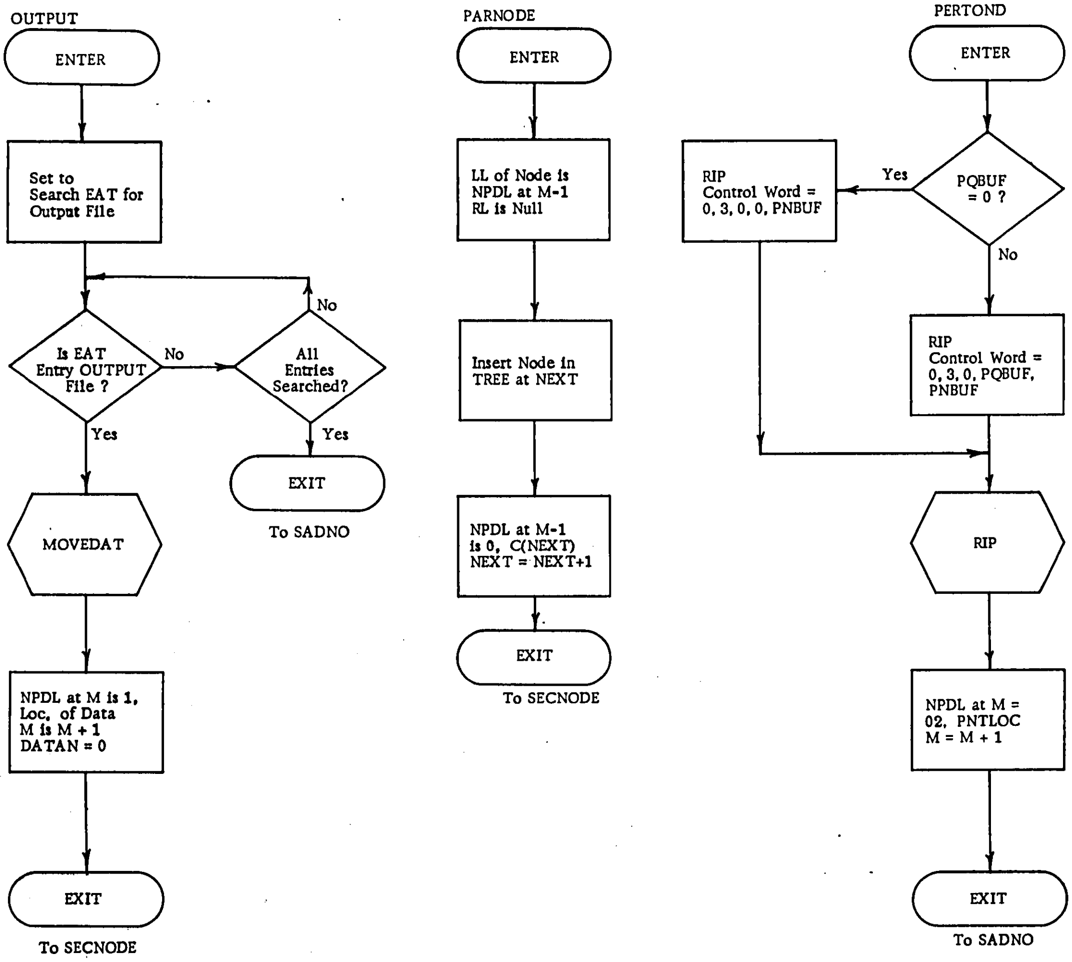


Figure 3-49. Pass 1E Flowchart (18 of 22)

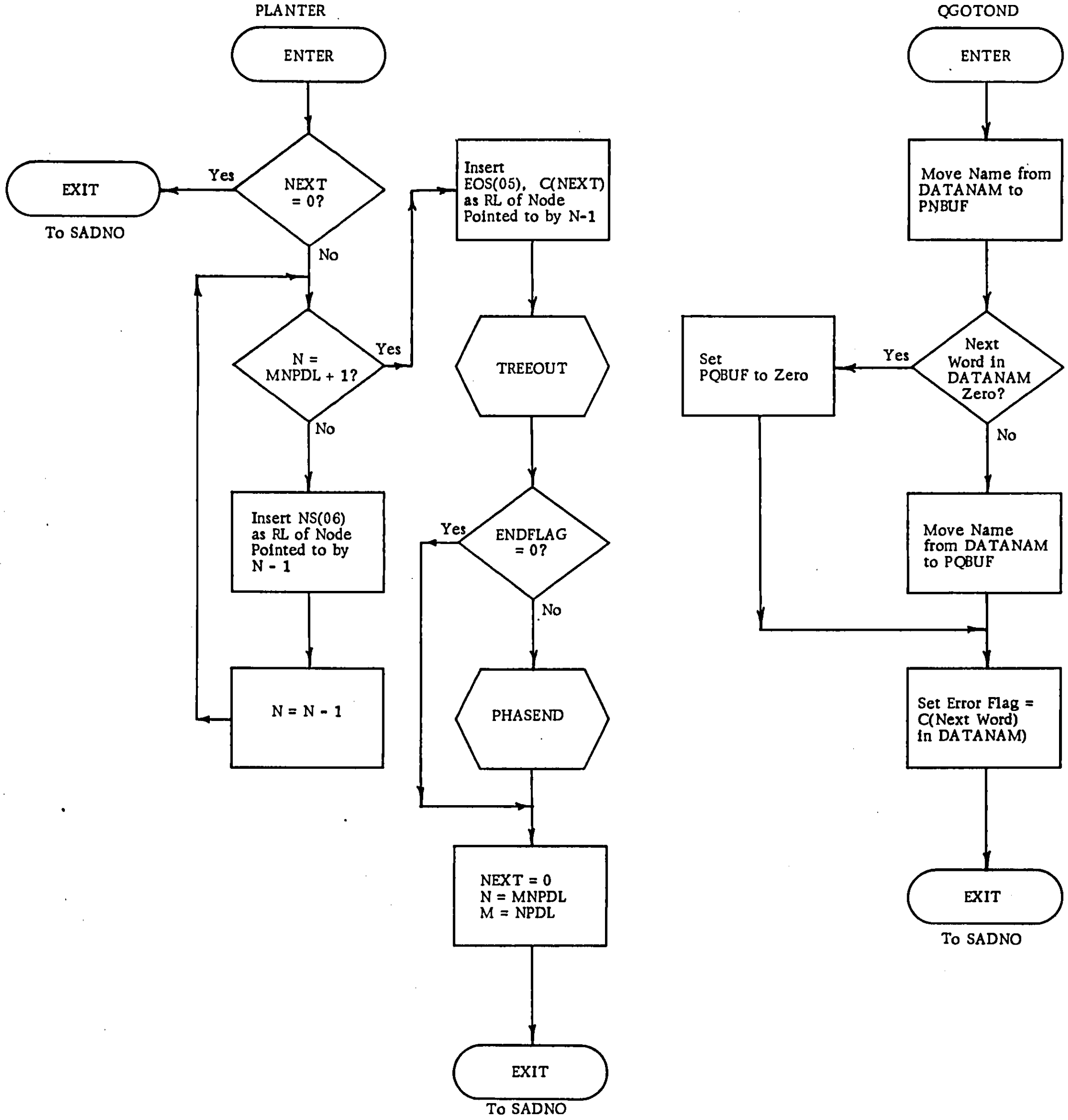


Figure 3-49. Pass 1E Flowchart (19 of 22)

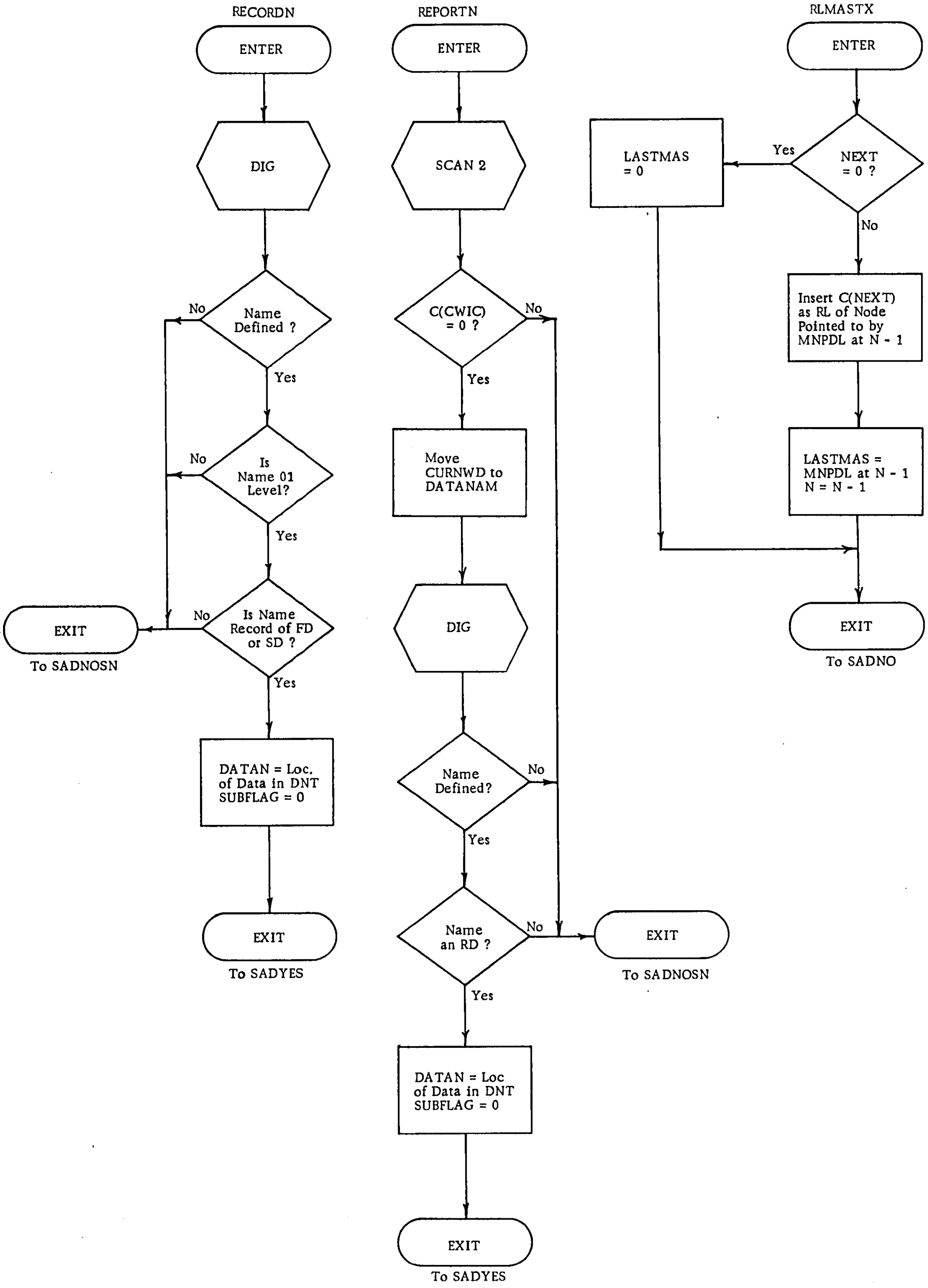


Figure 3-49. Pass 1E Flowchart (20 of 22)

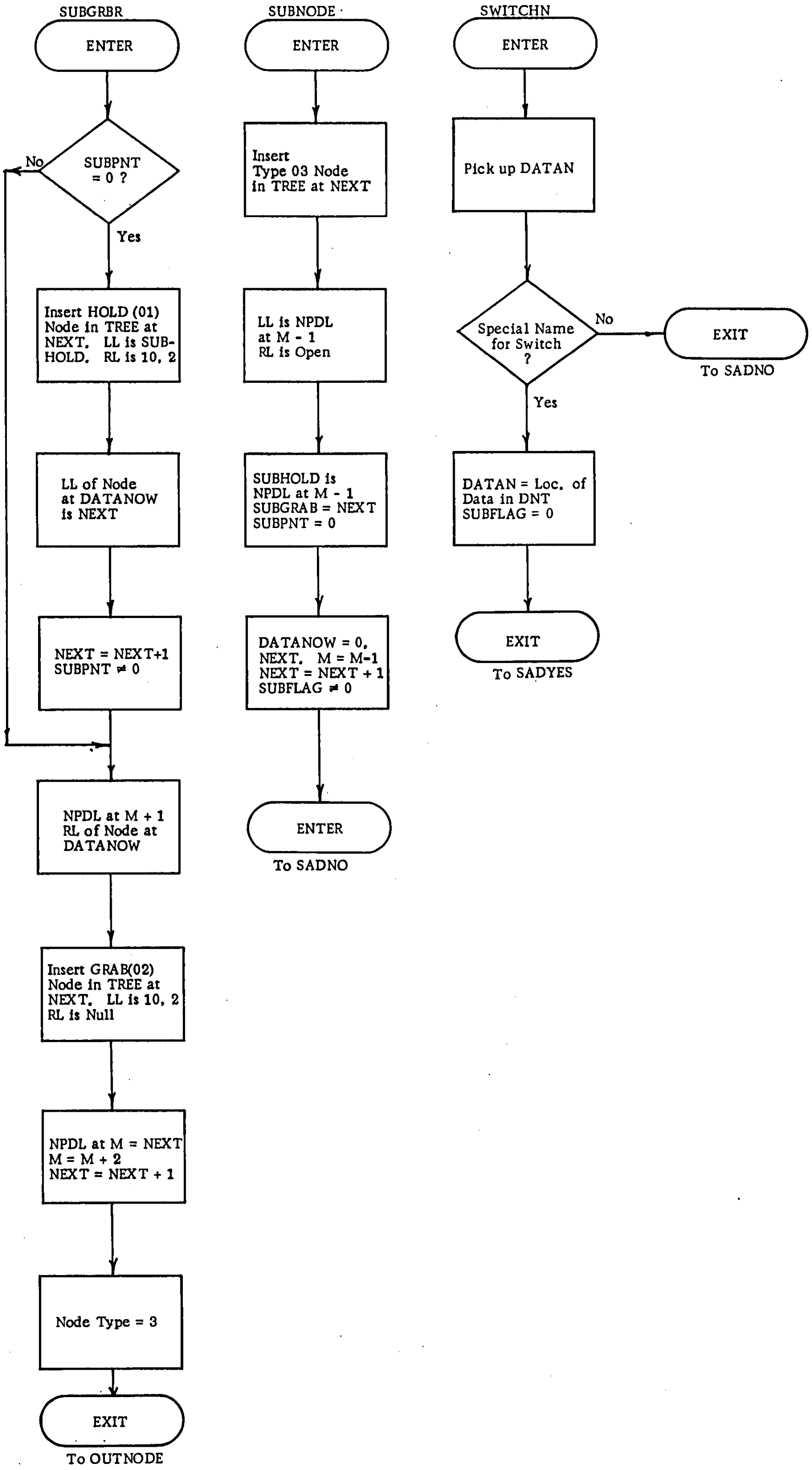


Figure 3-49. Pass 1E Flowchart (21 of 22)

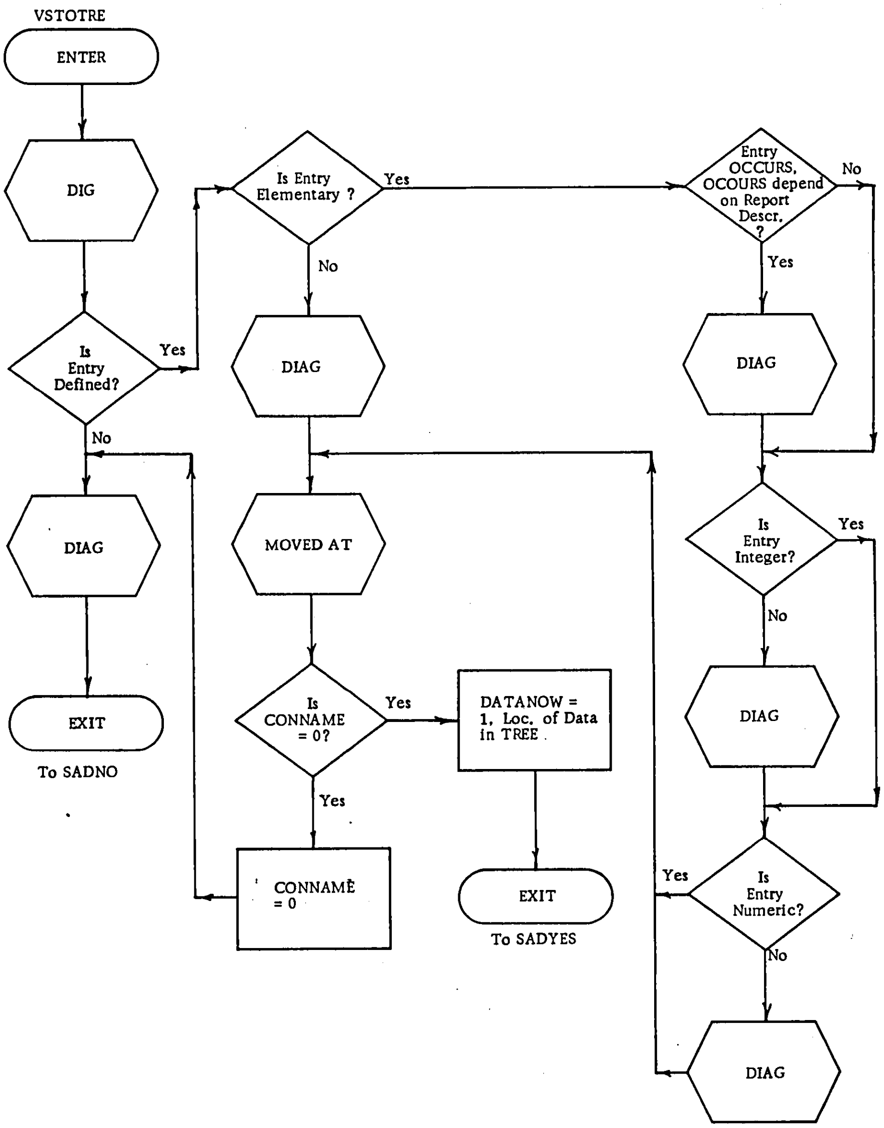


Figure 3-49. Pass 1E Flowchart (22 of 22)

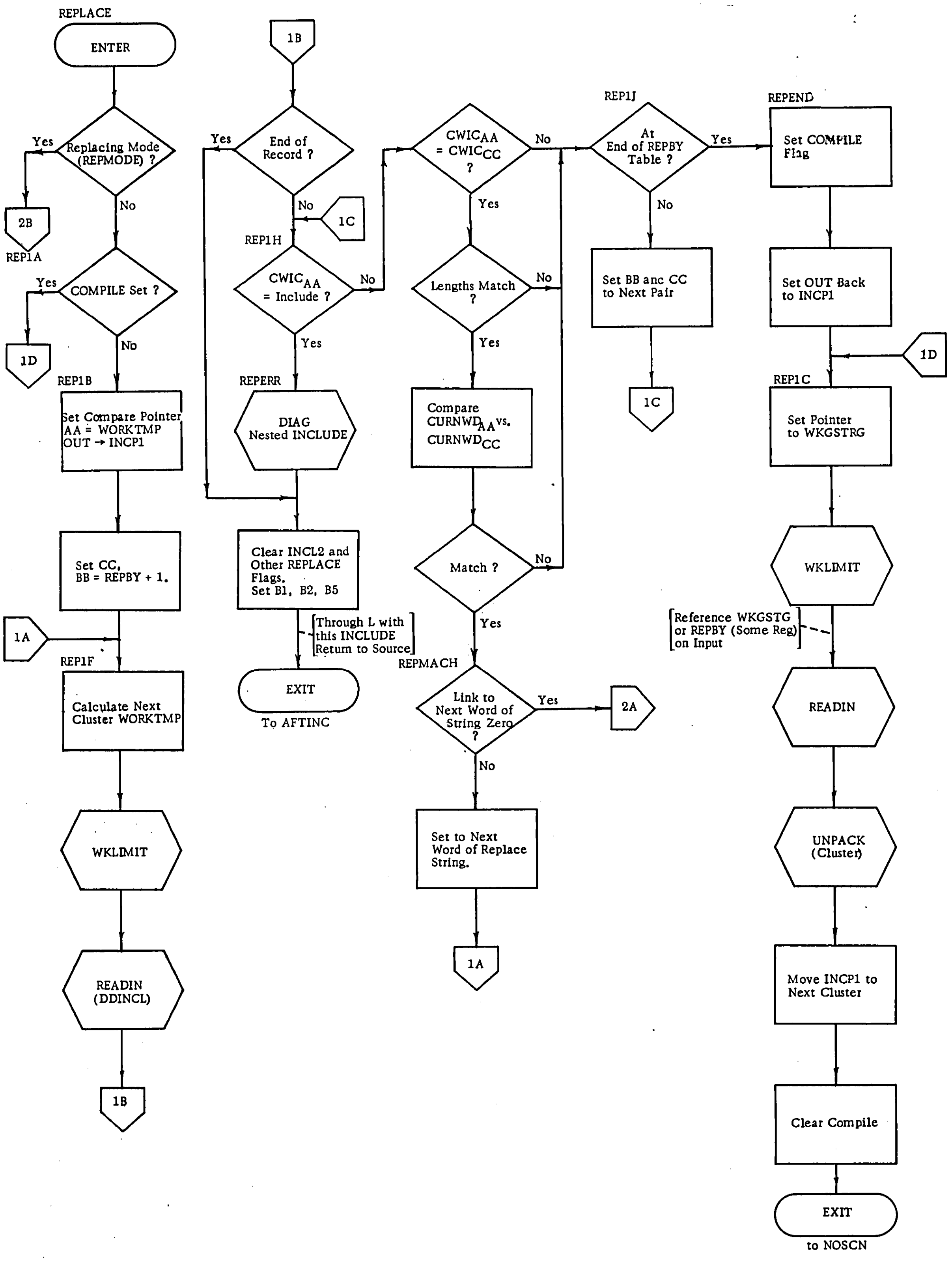
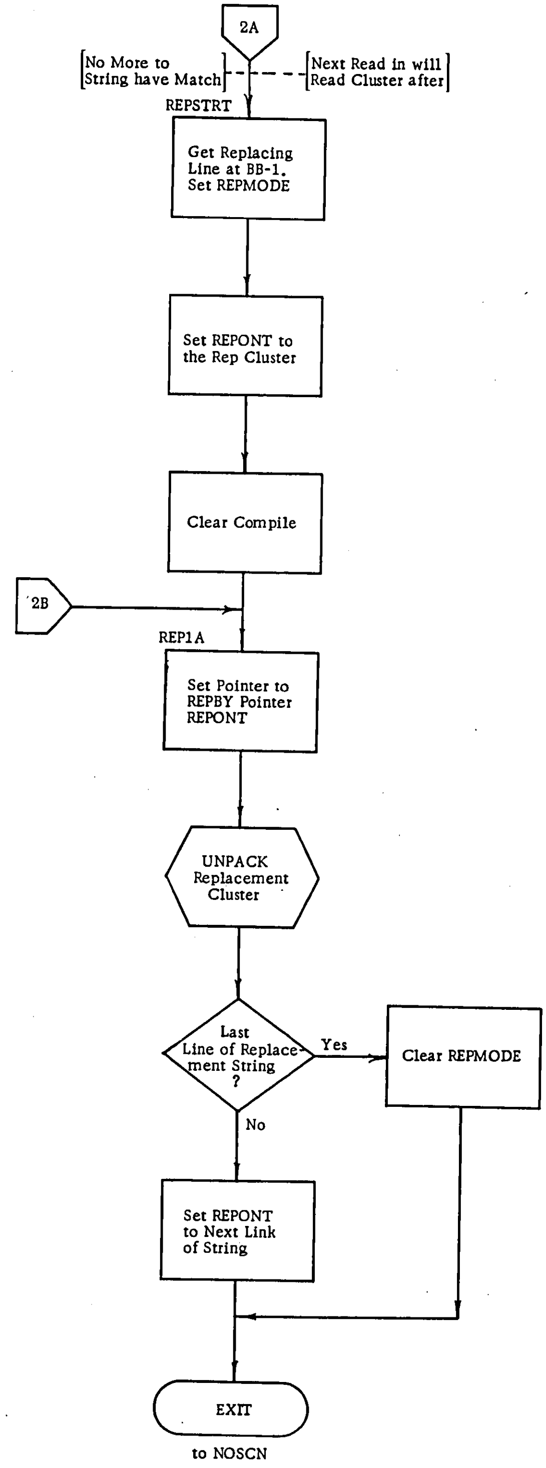


Figure 3-50. INCLIB (Pass 1E) Flowchart (1 of 2)



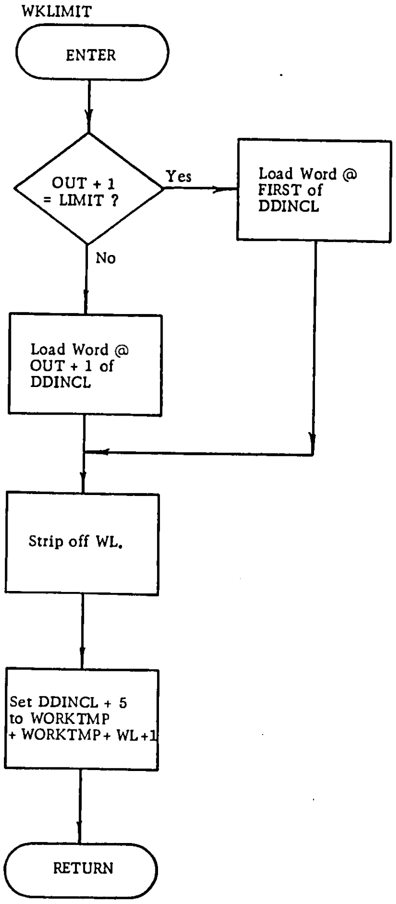


Figure 3-50. INCLIB (Pass 1E) Flowchart (2 of 2)

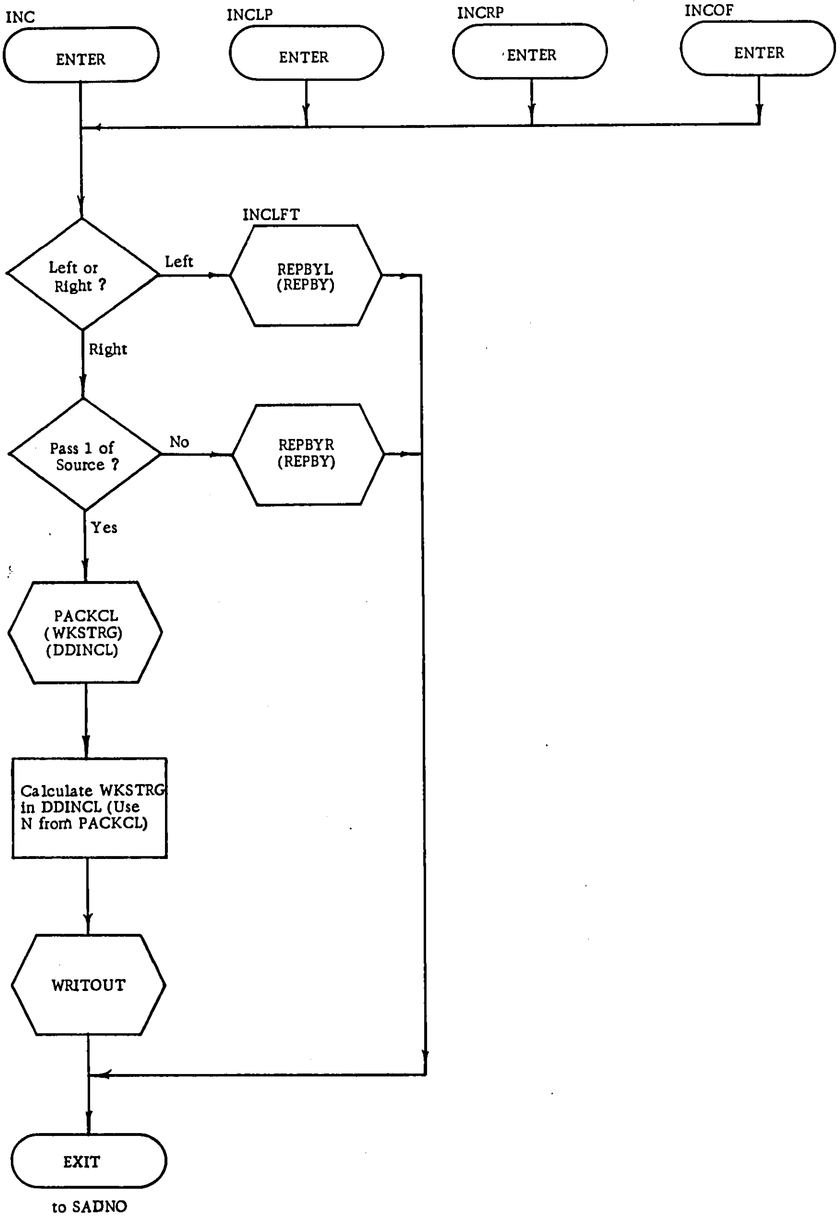


Figure 3-51. INC (Pass 1E) Flowchart (1 of 3)

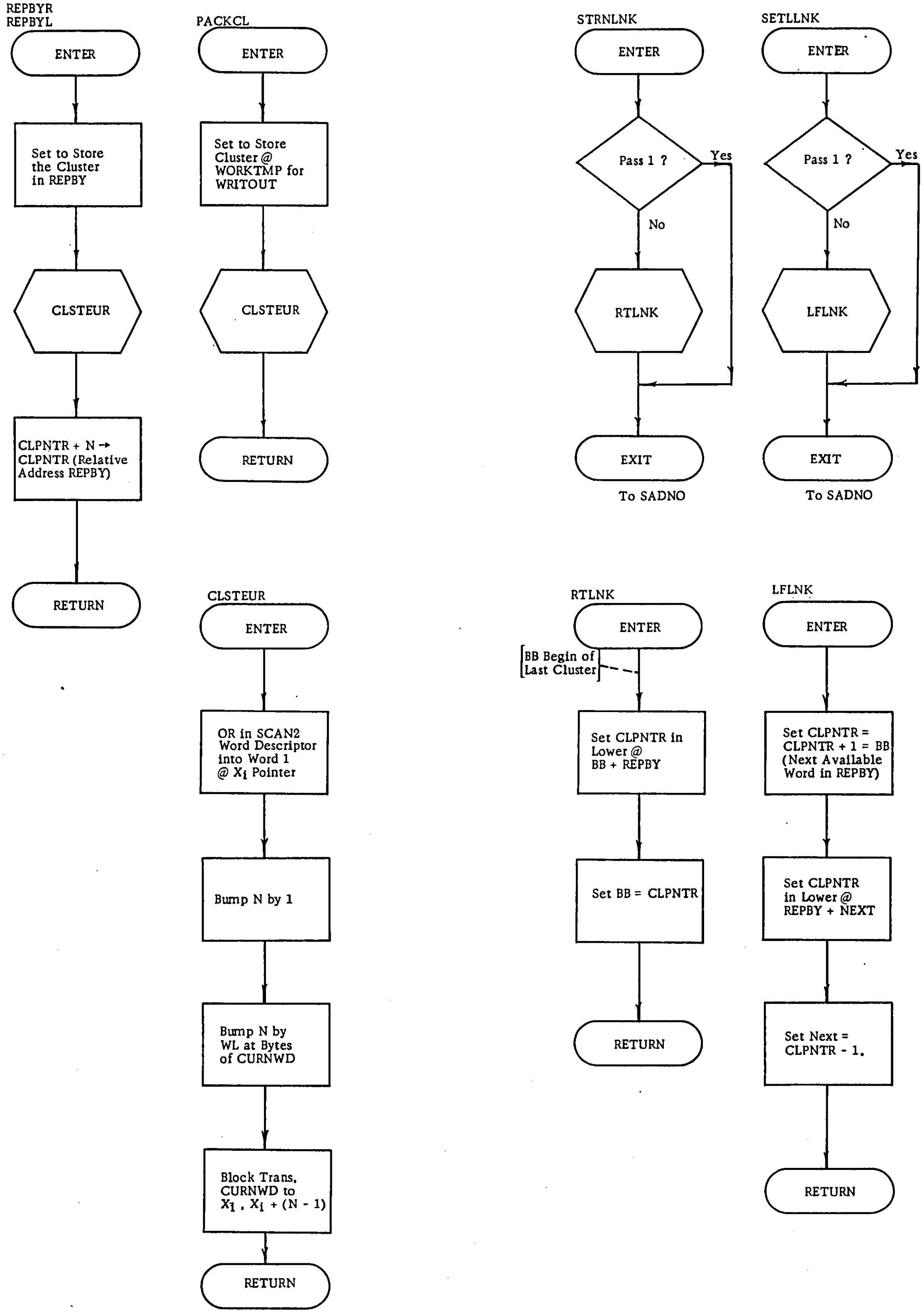


Figure 3-51. INC (Pass 1E) Flowchart (2 of 3)

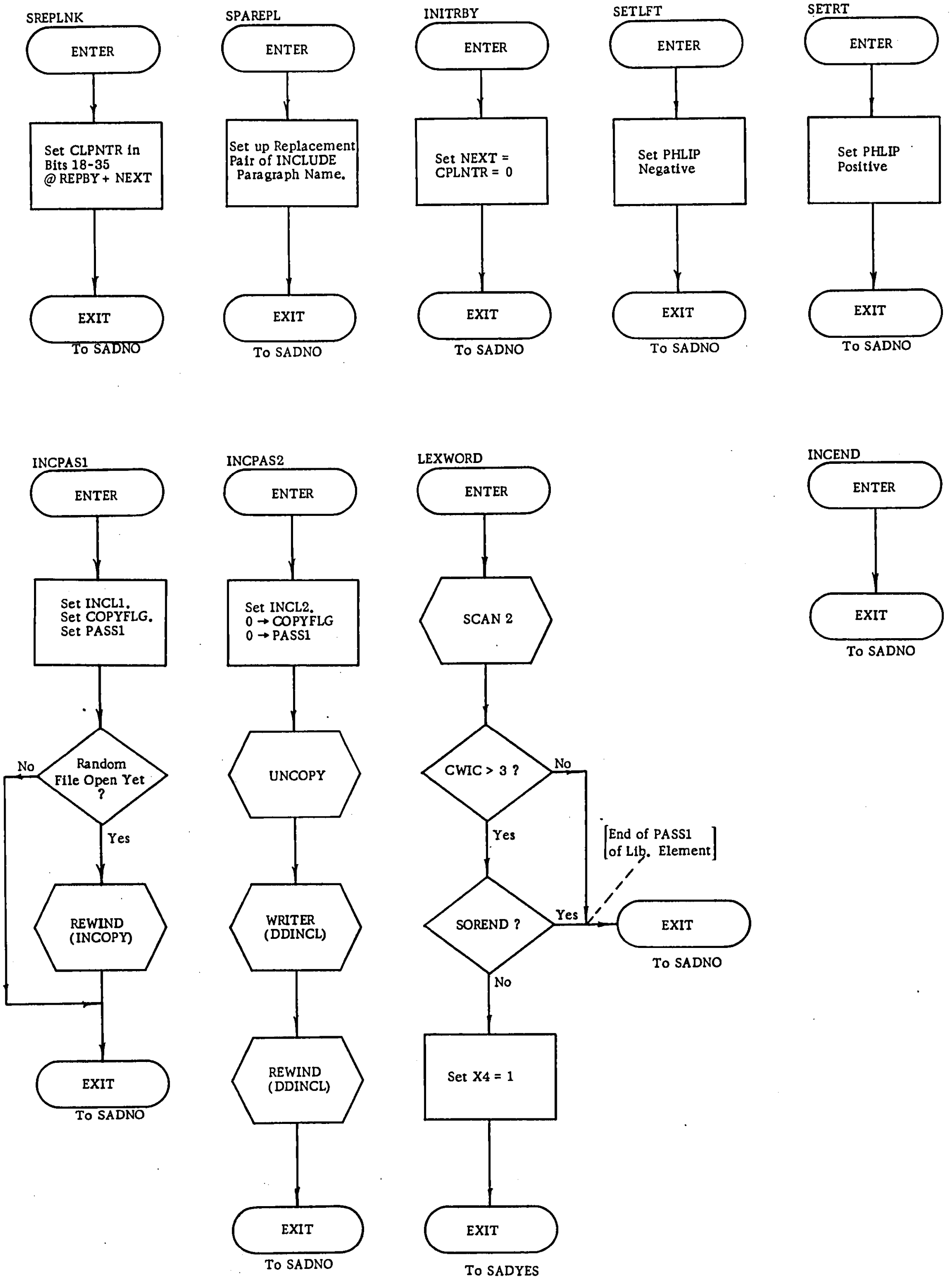


Figure 3-51. INC (Pass 1E) Flowchart (3 of 3)

Segmentation

The overlay method is used to load a COBOL program with priority sections (see Table 3-9). Primary overlay cards are generated by the COBOL compiler and appropriately placed in the binary relocatable output file as this file is generated. Secondary overlays are not generated. An overlay consists of an element for the code of a priority section. Each overlay card reflects the priority section number defining it by using a primary overlay number equal to 49 less than the defining section number. The main overlay includes sections 0-49. If no overlays are needed, no overlay cards will exist. If more than one compilation is to be loaded, section numbers must be chosen to given unique overlay numbers and the elements must be merged either by hand (cards) or using SCOPE file manipulation routines.

Accessing Overlays

The Overlay Method used by the system SCOPE loader does not allow external references from nonzero-level overlays referring to entry points in other nonzero-level (primary) overlays. The compiled program keeps information in the main overlay concerning its primary overlays so that the primary overlays can be loaded and properly entered when needed. A COBOL library subroutine (DDSOL) is furnished for this purpose. DDSOL is loaded in the main overlay, and maintains the status of the load (i.e., remembers which overlay is loaded), builds loading parameters, calls the system loader when a new overlay is needed, and enters the code at the appropriate location in the overlay when the system loader returns after loading the overlay. To accomplish this, the compiled COBOL program must furnish the library subroutine with the proper overlay number for each call, and also furnish the relative location in the overlay to which control is to be given.

The compiler builds a pointer word in the first-level main element containing these two items of information for each entry point in each priority section. An entry point is a procedure which is referenced by a GO TO, ALTER, SORT, ENTER subroutine, or PERFORM statement from the main or another overlay. Also, each entry point defined by an ENTER COBOL procedure causes at least one pointer word to be generated (see section on entry points below).

All pointer words including exits from entry point procedures are located in a table together at first level. Indexes to this table may be used to locate any particular pointer word. The actual location in the overlay of the procedure referenced by a pointer word is not defined when the pointer word table must be output to disk. Thus a jump table is placed at the beginning of each overlay with a jump to each externally-referenced procedure in that overlay. This is output last when the overlay is processed. The compiler specifies jump table locations in the overlay before and during overlay processing.

Each time a GO TO or PERFORM is made to an overlay, the pointer word is loaded in a register before calling the special loading library routine. PERFORMed or ALTERed exits use the same method. The place to which the control transfers is in another nonzero level overlay. Such ALTERs are made by simply changing the value of the exit's first-level pointer word.

The library subroutine adds a base address returned to it after loading is completed by the system loader to the relative address in the furnished pointer word to obtain the location in an overlay jump table to which control is to be given.

Whenever code from one section drops through to code of the next section, and the next section is a priority section, this is also such an external jump.

Entry Points and External References

Entry points are defined by an ENTER COBOL statement. The first instruction of a program is also an entry point.

For each entry point defined in a compilation, a pointer word (see above) is reserved for it, for possible use by a PERFORM or is ALTERed from another compilation. The entry point is externally defined to be the location of the pointer word of the exit. The call words to access the entry point's generated code immediately follow the pointer used. The call words for an entry point consists of either a direct jump to the code if the code is at first level also, or the loading of a pointer word and a call to the library loading routine if the code is in an overlay.

In any case, the exit from any procedure that is named as an entry point in the compilation, is made by using the pointer word at the entry point's external definition location and calling the library load routine as if from an overlay, as defined above. The pointer at this location will be initialized to "point" to the normal exit from the procedure (usually the next procedure).

When a PERFORM, or ALTER of an external reference is found, the compiler generates the code to ALTER a pointer word assumed to be at the location of the external reference. The entrance generated to such an assumed entry point (using PERFORM or GO TO statements) is to the location plus one of the external symbols.

Generated Code in Main Overlay

PERFORM statements using no load return index result in a sequence of code that does the following:

1. Saves the present exit code of the last indicated procedure of the code being PERFORMed by saving a jump instruction word. (Exit jumps are placed at the beginning of a whole word).

   The current exit is saved in the instruction described in Step 4. Nested PERFORMs with common exits can be allowed with this technique.

2. ALTER the exit of the last procedure being PERFORMed. This is done by replacing the saved jump instruction with a jump to the instructions described in Step 4.

3. If the first procedure to be PERFORMed is in the main overlay, a jump is generated directly to the procedure. If not, its pointer word is referenced and a jump is made to the library loading routine described above. (GO TO statements always have these two possible ways of being generated.)

4. After return from the PERFORMed code, the previously saved exit (Step 1) is restored.

Generated Code in Priority Section Overlays

PERFORM statements result in a sequence of code which will:

1. Pick up the pointer word used for the exit of the procedure to be performed.

2. Create a pointer word to be used instead of the one in Step 1 above so that return will be up to the code following this sequence of code. This pointer word is flagged as a temporary pointer.

3. Pick up the pointer word to enter the perform code.

4. Go to SOL which will in turn save the old exit pointer index, place the new one in the old one's spot, load the first procedure to be performed, and go the the perform code.

5. No code is needed at the return. SOL is used to exit from the perform. It detects the flagged pointer word and restores the exit prior to return.

The overlay number and defining line number of referenced names is inserted when the defining procedure is found. If, in second pass, the line number is found still undefined, the procedure name is treated as an external reference.

The section number is used in first pass to determine if a reference to the procedure required loading of any overlays and thus causing one of two types of pointer words to be assigned.

When a reference to a procedure is made prior to its definition, the need for loading is unknown. It could be that the procedure referenced will be undefined (external), or in another overlay or in the same overlay. Each possibility could cause a different sequence of code generated. When the loading status is unknown like this, flags will be set in the PNT to indicate which overlays referenced the procedure. Thus, at the time the procedure is defined, these flags can be evaluated to determine if pointer words are needed. PERFORM and GO TO references will cause possible use of a pointer word for entering a procedure. PERFORM and ALTER references will cause possible use of a pointer word for exiting from a procedure.

Table 3-10. Load File Layout With Overlays

Overlay Card (0, 0)
Element in main overlay, including File Tables, Common Storage, Working Storage, Sections 0-49
Overlay Card (X, 0)
Element for Section (X + 49)
Overlay Card (Y, 0)
Element for Section (Y + 49)
Etc.

A COBOL programmer can scatter parts of one overlay through the procedure division. This requires that all the pieces of one overlay be gathered together as the relocatable binary object code is produced.

The compiler uses random-access files on disk with an Index Buffer, as described in the ERS for SCOPE 3.0. For details of Tree I/O, see Section 2, Tree Output (TREEOUT) Subroutine.

Miscellaneous Other Subroutines for Pass 1E (DIG, MIG, DIP, and RIP)

A subroutine to find data items in the Data Name Table is used by the Tree routines in Pass 1G.

The call is as follows:

```
        RJ          DIG "Data Item Getter"
        Return
```

where Register B7 contains a pointer to a buffer containing the name in a format similar to the format of the name in the Data Name Table. If the name has modifiers, each modifier immediately follows the previous item in the buffer, the first modifier following the elementary item, etc., using the same format rules for each name. A word of all zeros terminates the buffer.

The return is to the word following the call with the address of the entry in the DNT in B7. If the address is 0, no entry was found. If two or more entries were found by the routine to satisfy the call, one is selected and the DIG routine issues a diagnostic (see Figure 3-52). It should be noted that if many calls to DIG are made regarding the same reference (i.e., one from 1G, one from 1F, and one from Pass 2), the error message will be restated for each call. The DIG routine follows links of names and modifiers as required. It calls the Hash routine and any other routines necessary.

For ADD, SUBTRACT, or MOVE CORRESPONDING statements, two groups of items must be matched, item for item. The MIG (Matched Item Getter) routine is furnished to match these data items.

The calling sequence is:

RJ MIG
Return if no more matches
Return with a match

where the name of the two group items in the statement along with the type of statement are furnished in a control word in register X4.

N	0	Location of D1	Location of D2
3	21	18	18

where

N - 4 if this is a MOVE statement.
0 if this is an Arithmetic statement.

and

D1 - the first Data Group Name.
D2 the second Data Group Name.

Each time the routine is called a new matched pair will be returned until the groups are exhausted. Register X4 will contain the Control Word upon return with the location fields replaced by pointers to respective corresponding items in the DNT.

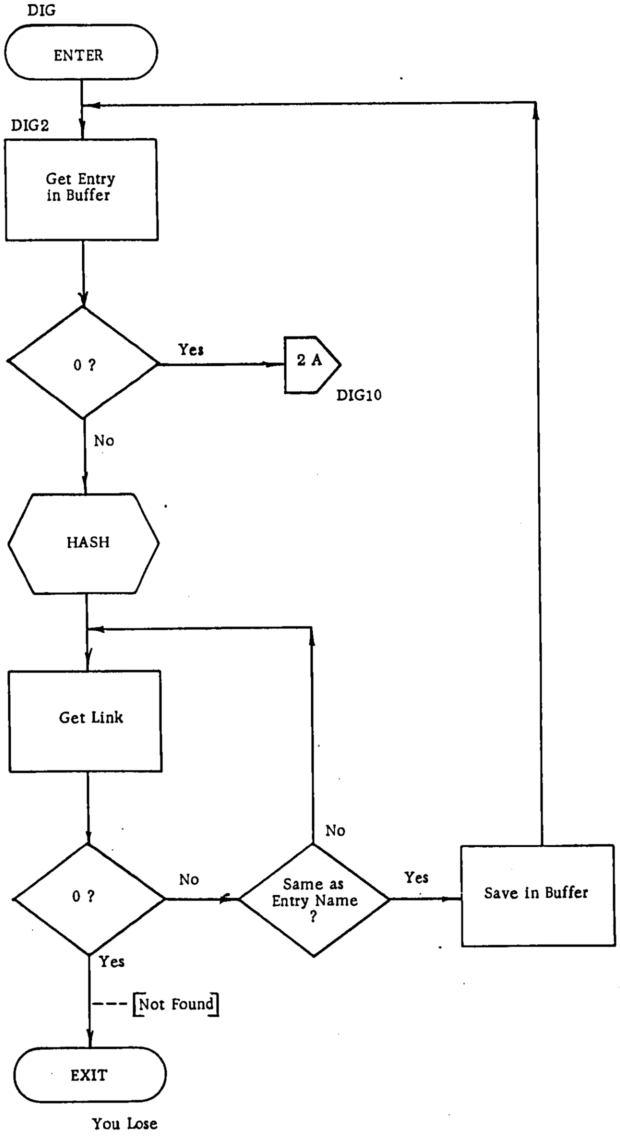


Figure 3-52. DIG Flowchart (1 of 2)

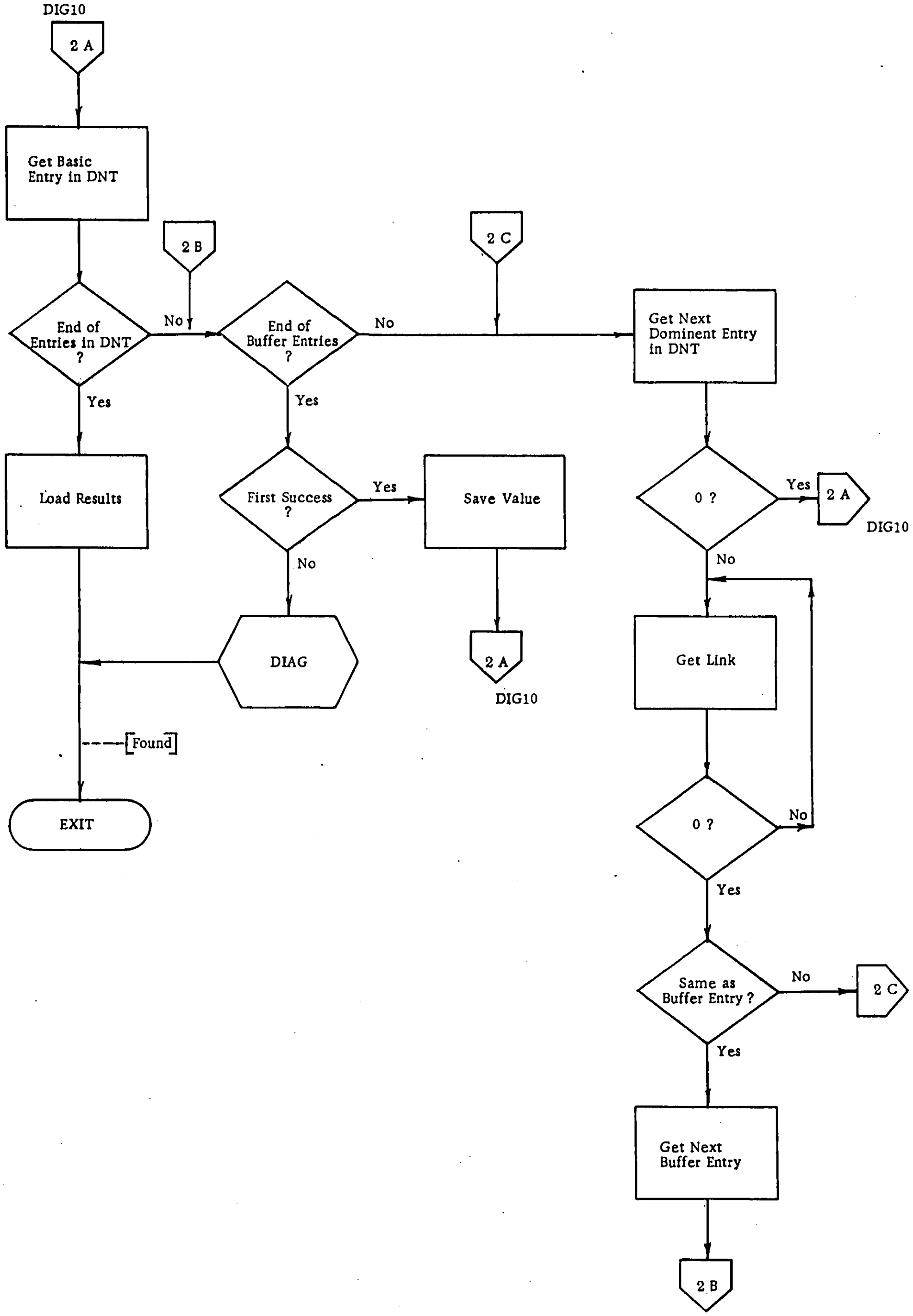


Figure 3-52. DIG Flowchart (2 of 2)

DIP and RIP Subroutines

Information is required in the Procedure Name Table for references between procedures in different priority sections because object code generated in the second pass is loaded by the overlay method which does not handle referencing between overlays. The object code itself handles the referencing.

For each defined procedure of the program, the routine DIP (Definition Indexing for Procedures, see Figure 3-53) is called to enter the name in the PNT as a four-word entry (see Section 4). For each reference of the program to a procedure, (ENTER COBOL, ALTER, GO TO, ENTER subroutine or PERFORM statements), the routine RIP (Reference Indexing of Procedures, see Figure 3-41) is called to enter the reference in the PNT as a one-word entry.

The Procedure Name Table is built initially by the cross-referencing routines DIP and RIP although other routines contribute information to the table from time to time.

DIP and RIP are called by and require the following input from the tree-building routines during Pass 1E: the input word will be placed in X4; return will include the PNT location of the entry in X4.

Definition Indexing for Procedures DIP - Pass 1E

DIP is called by the tree-building routines in Pass 1E when a procedure is to be defined. DIP constructs a four-word PNT entry. The following items are filled in (if available):

- Item type
- Procedure type
- Link to next item in source sequence
- Link to dominant section
- Link to next item with same name
- Priority section number
- Line number

Linkage to the HASH table and hash links are created if needed.

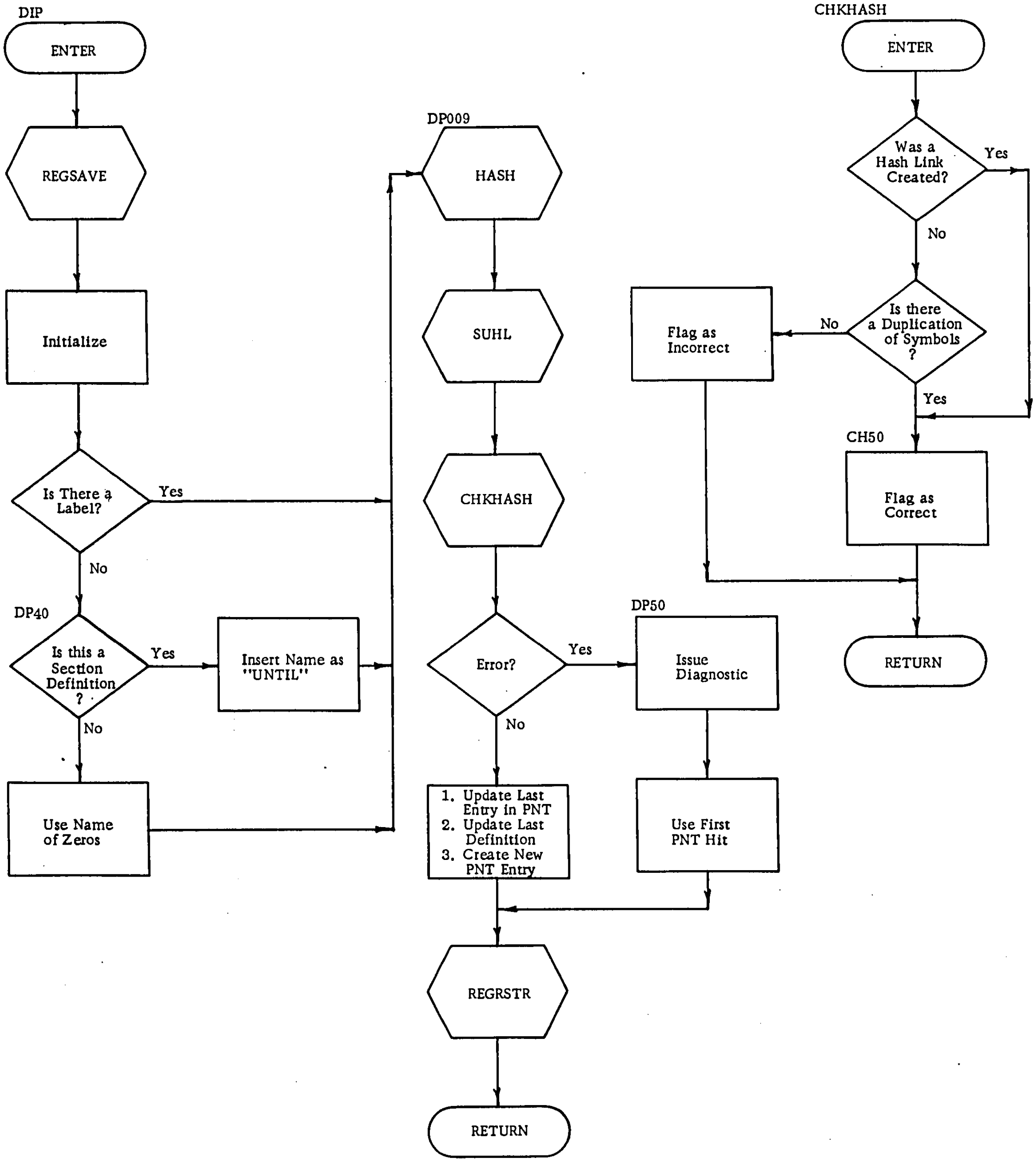


Figure 3-53. DIP Flowchart

DIP is called by the tree-building routines in Pass 1E by:

```
RJ          DIP
Return
```

where X4 contains:

0	Flags	Location of Modifier	Location of Label
18	6	18	18

Flags - 0 if a paragraph name.
1 if Label is a section name.

Modifier - Buffer address of section number in BCD if Label is a section name.

Location of Label - Buffer address of section of paragraph name. It may be left zeros for paragraph names.

RIP - Reference Index Procedure

RIP is called by the tree-building routines in Pass 1E when a procedure is referenced. RIP constructs a one-word PNT entry. (See Figure 3-54.)

Linkage to the HASH table and hash links are created if needed.

The calling sequence is:

```
        RJ        RIP
        Return
```

where X4 contains:

0	Type	Flags	Location of Modifier	Location of Label
12	6	6	18	18

Type - 1 if a GO TO reference.
2 if an ALTER reference.
3 if a PERFORM reference.
4 if an ENTER COBOL reference.

Flags - 0 normally.

32 if processing for this reference is to be cancelled. This will happen on second calls for PERFORMs or ALTERs when a code error was detected by the tree-building routine and the statement is to be ignored.

Location of Modifier - Buffer address of section name modifying paragraph. Zero is no modifier for Label or if Flags is 32.

Location of Label - Buffer address of section or paragraph name. This cannot be zero unless Flags is 32.

All Procedure Names in an "ENTER subroutine REFERENCING" statement including the subroutine name cause a GO TO call to the RIP routine.

Section names in the SORT statement cause PERFORM calls to the RIP routine. One call to RIP is made for each procedure referenced in the Procedure Division code. Statements like

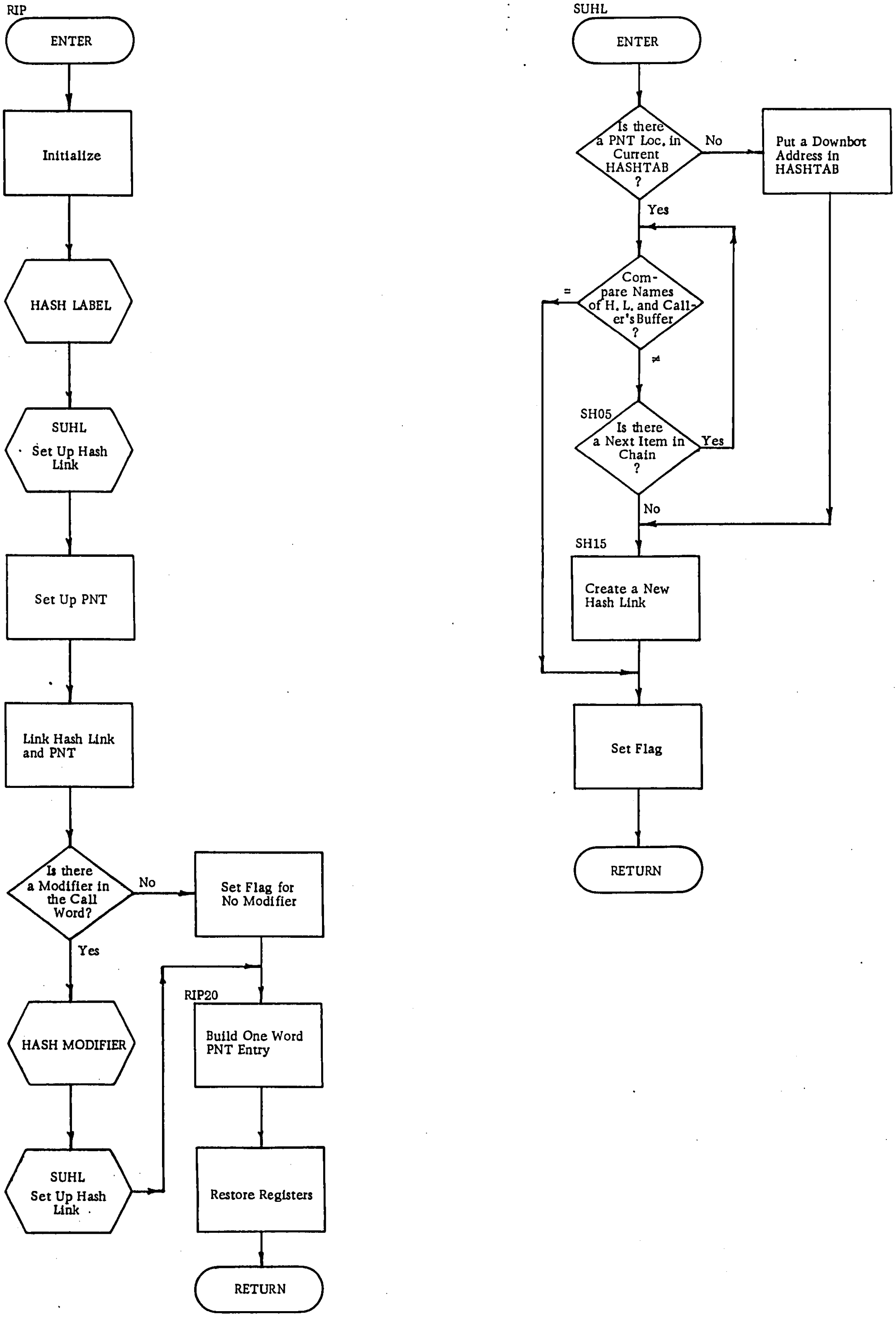


Figure 3-54. RIP Flowchart

PERFORM and ALTER make two calls to the RIP routine. ENTER subroutine REFERENCING and GO TO DEPENDING ON statements may make many calls depending on the number of procedures named in the statement.

PASS 1F AND ELEMENTS

Pass 1F Subroutine to control processing during Pass 1F.

Pass 1F has two purposes:

1. Complete the Procedure Name Table (PNT) by comparing reference and definition entries in the PNT and appropriately filling in the definition entry.

2. Determine assignments for the index table (to precede generated procedure code) and jump tables (to precede each overlay except the main overlay). Both of the aforementioned tables are to be generated during Pass 2. Pass 1F also builds the section jump tables (JMPTBL) and sets the index counter (INDXCNT), both of which are in CONTROL.

Interfacing is as follows:

1. External subroutine referenced:
    a. REGSAVE, REGRSTR
    b. COREDMP, SNAP
    c. CONEXF
    d. REF
    e. ASJE

2. External tables and pointers referenced:
    a. PNTBASE, DOWNBOT
    b. OVNO
    c. TSPASS
    d. INDXCNT
    e. HASHTAB

Pass 1F is the entry point for overlay 1,6. Thus, Pass 1F is entered via CONTROL calling the loader and transferring to Pass 1F.

Control of execution during Pass 1F is divided into two parts according to function.

1. SCAN1F completes the PNT by sifting through the PNT from PNTBASE to DOWNBOT. Hash links and definition entries are skipped over by the sift and only reference entries are processed. When a reference entry is encountered, REF is called to compare the reference and definition entries and fill in the definition entry.

2. P1F searches the PNT via the jump table (JMPTBL) and assigns index positions. It also fills in the JUMP TABLE INDEX and LINK TO NEXT JUMP TABLE ENTRY in PDR2 of the definition entry.

The following method is used by Pass 1F to cross-reference procedures; if the procedure referenced is in the same overlay as the calling procedure or in the main routine (i.e., always in core), normal references are made from the calling procedure. If the procedure referenced is in an overlay other than the one of the calling procedure, the overlay must be loaded (GO TO or PERFORMs) before entrance is made to the overlay. The compiler generates a word ("pointer index") in the main routine containing the overlay number and entry point location of each procedure externally referenced in each overlay. A routine, SOL, will be furnished in the COBOL Project Library, which is called by referencing routines to interrogate "indexes," to call the system loader to load the overlay, and to exit to the overlay's entry point designated by the "index." Pass 1F assigns these indexes. SOL is described elsewhere in this document.

Codes generated for ALTER and PERFORM statements change values of indexes in the index table at main level.

Procedures referenced by ENTER COBOL statements always have indexes generated for them. Exits for these procedures are made via these indexes to provide return capabilities to other programs. A jump table of entry points is included at first level of object code for the entry from other programs.

The main level object code Loading Index Table (LIT) consists of all indexes needed and all entry points, the latter being defined at the point in the table that corresponds to the index assigned to it. Entries from external routines are made to the location following the return index (and defined entry point). This location consists of an appropriate jump to the code required.

Each overlay, for which indexes are generated for procedures within the overlay, has a table of jumps to procedures at the beginning of the overlay. Indexes contain the location in the overlay of the jump and not the procedure location itself. That is defined when the code is generated for it, and this takes place after the main routine, containing the LIT, is generated. Jump table references in overlays are also assigned by DIP and RIP.

Flags in the Procedure Name Table entries will be assigned by the following routines:

F6	By the RIP routine in 1E for an ENTER COBOL statement.
F5	By the RIP routine in 1E for a PERFORM, or by the Tree Building routines before entering RIP for an implied GO TO when a new procedure has also a new priority section (drop-through automatic jump).

F4 By the RIP routine in 1E for entry points not yet defined and for other forced alter indexes.

Pass 1F also processes data placed in the Procedure Name Table (PNT) by the DIP and RIP routines (procedure cross-referencing) in Pass 1E. Processing in this pass consists of passing over the PNT, entry by entry, looking for jump table, entry point, and index pointer flags, assigning them sequentially as encountered. The Index Table (LIT) which eventually will be part of the main routine of the object code, is laid out in this pass. This consists of assigning addresses to each index and entry point indicated in the PNT by DIP and RIP. The second pass code output routines build the LIT table at the end of the second-pass processing of the main routine. Its location is assigned early in the main routine (main overlay). Entry points to code located in overlays use "indexed" jumps from the jump table in the overlay and jump table locations have already been assigned. Each jump table is at the beginning of its overlay. Jump tables in overlays are not coded until the rest of the overlay has been completed for the same reason that the LIT table is generated at the end of the main level processing.

REF - Process PNT References

REF is given a reference entry of the PNT (see Figure 3-55). REF finds the object of the reference (always a definition entry) and compares the section in which the reference appears and the section in which the definition appears. If they are different, REF sets the appropriate flags in the definition entry to enable PASSAF to successfully hook-up linkage.

REF is called by Pass 1F with the following register containing:

X4 - reference PNT entry
B1 - 1
B3 - SCAN1F PNT pointer

External subroutines referenced:

1. REGSAVE, REGRSTR
2. DIAG
3. HASH
4. SNAP

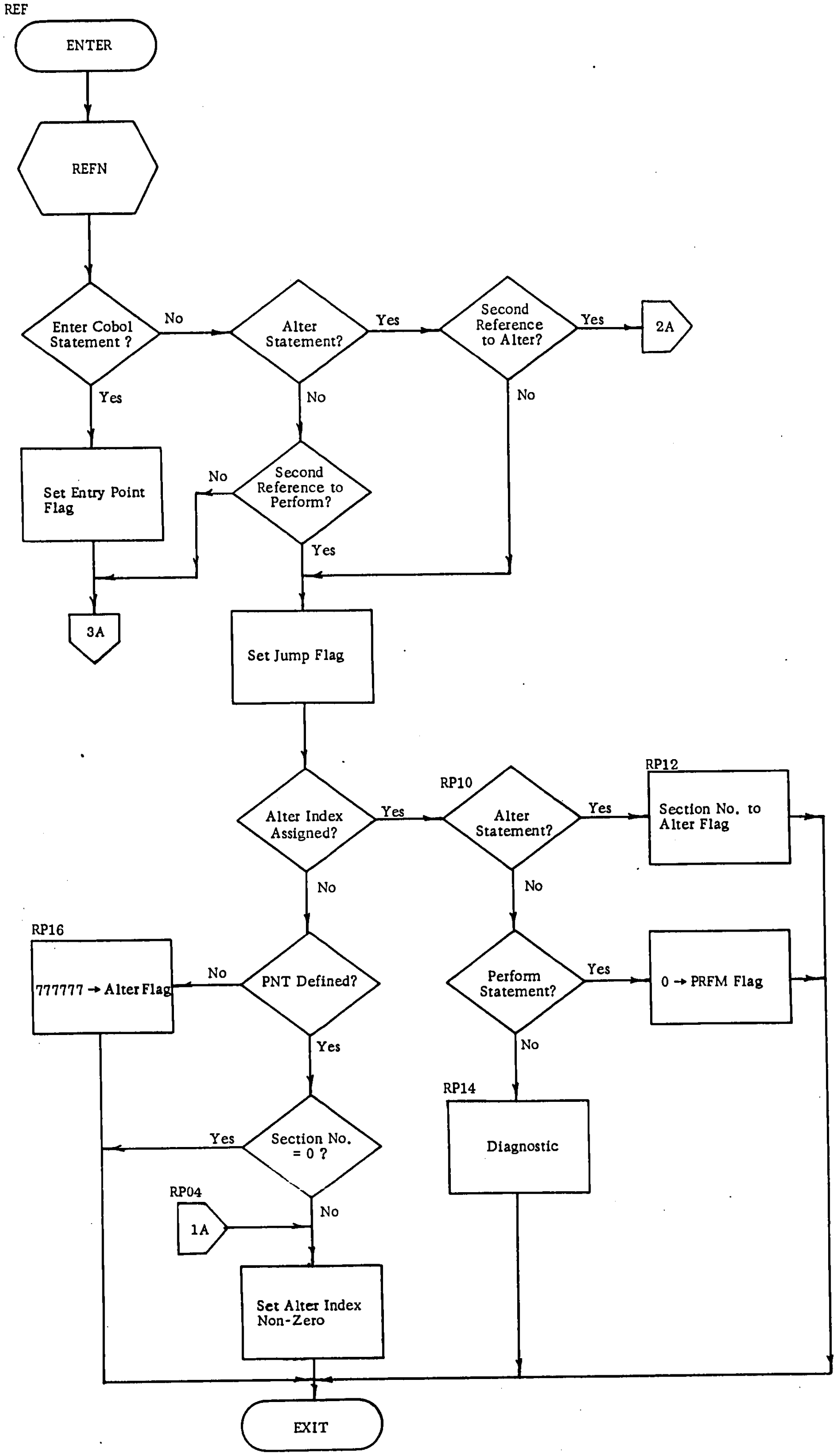


Figure 3-55. REF Flowchart (1 of 3)

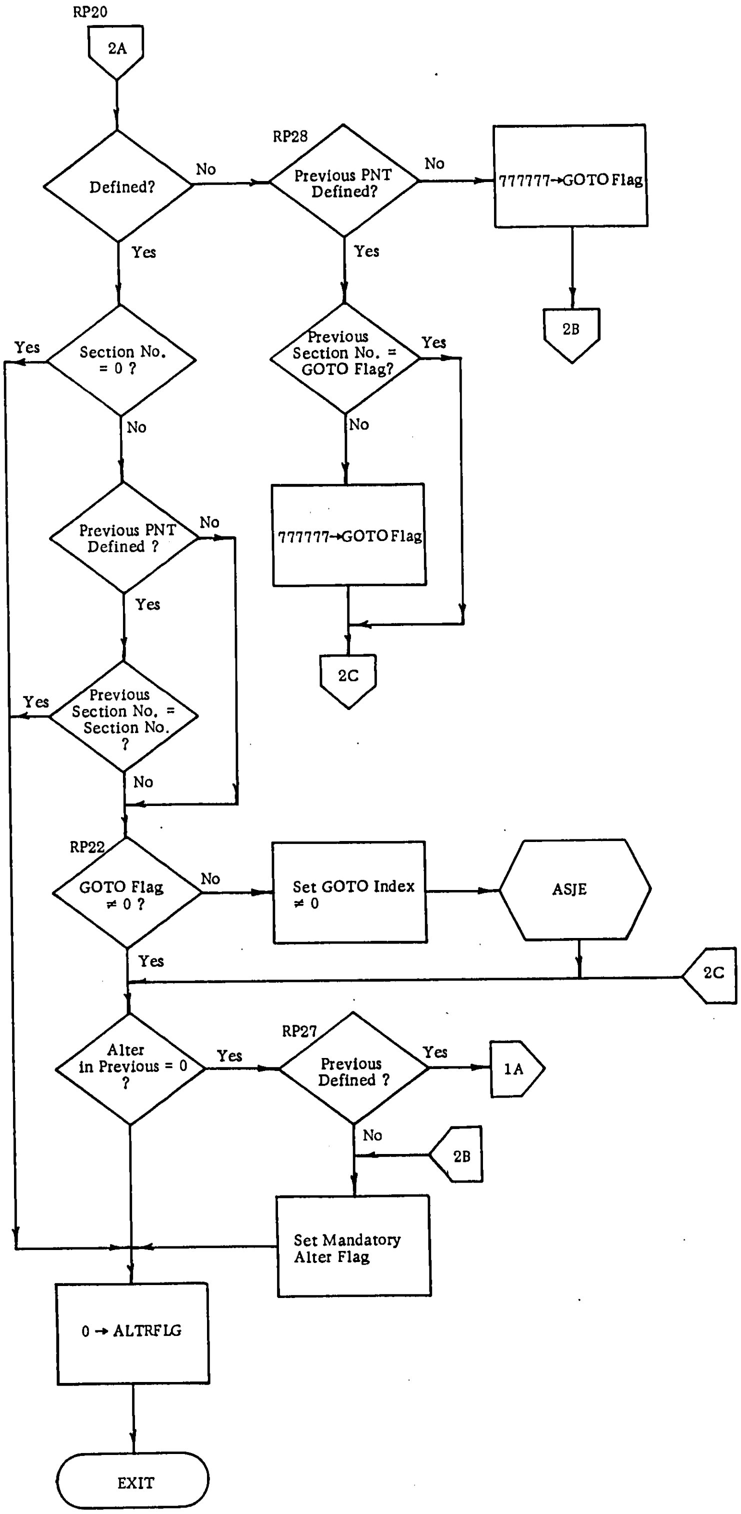


Figure 3-55. REF Flowchart (2 of 3)

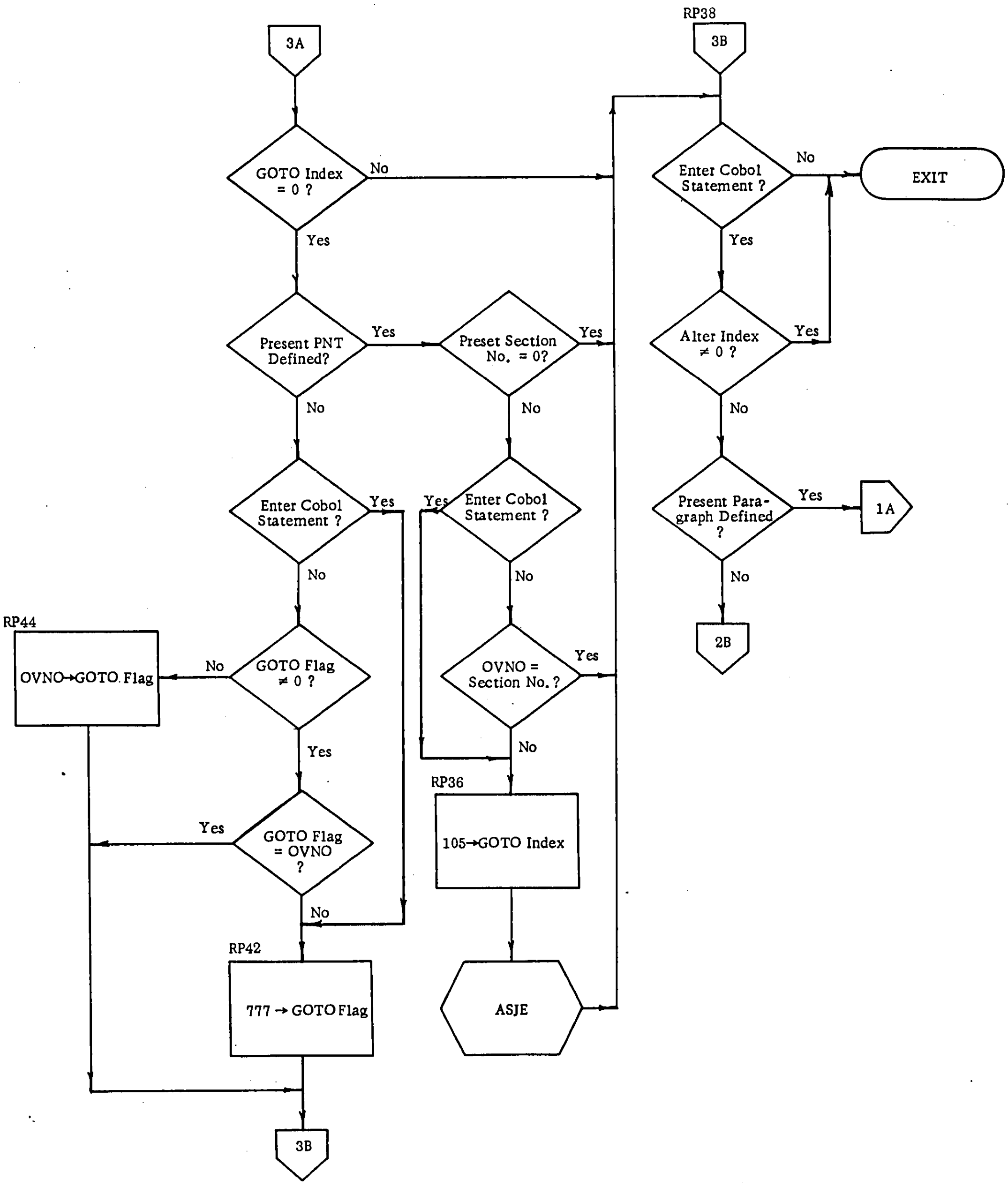


Figure 3-55. REF Flowchart (3 of 3)

External tables and pointers referenced:

1. HASHTAB
2. DOWNBOT
3. DAGLOC3
4. TSPASS
5. JMPTBC
6. PPCNTR
7. SCLINE

REF saves and restores all registers. Output is a pointer to the correct definition PNT in X4.

REF issues a diagnostic if a unique definition PNT cannot be found.

OBJECT CODE GENERATION

Figures 3-56 through 3-67 show the flowcharts used in Object Code Generation.

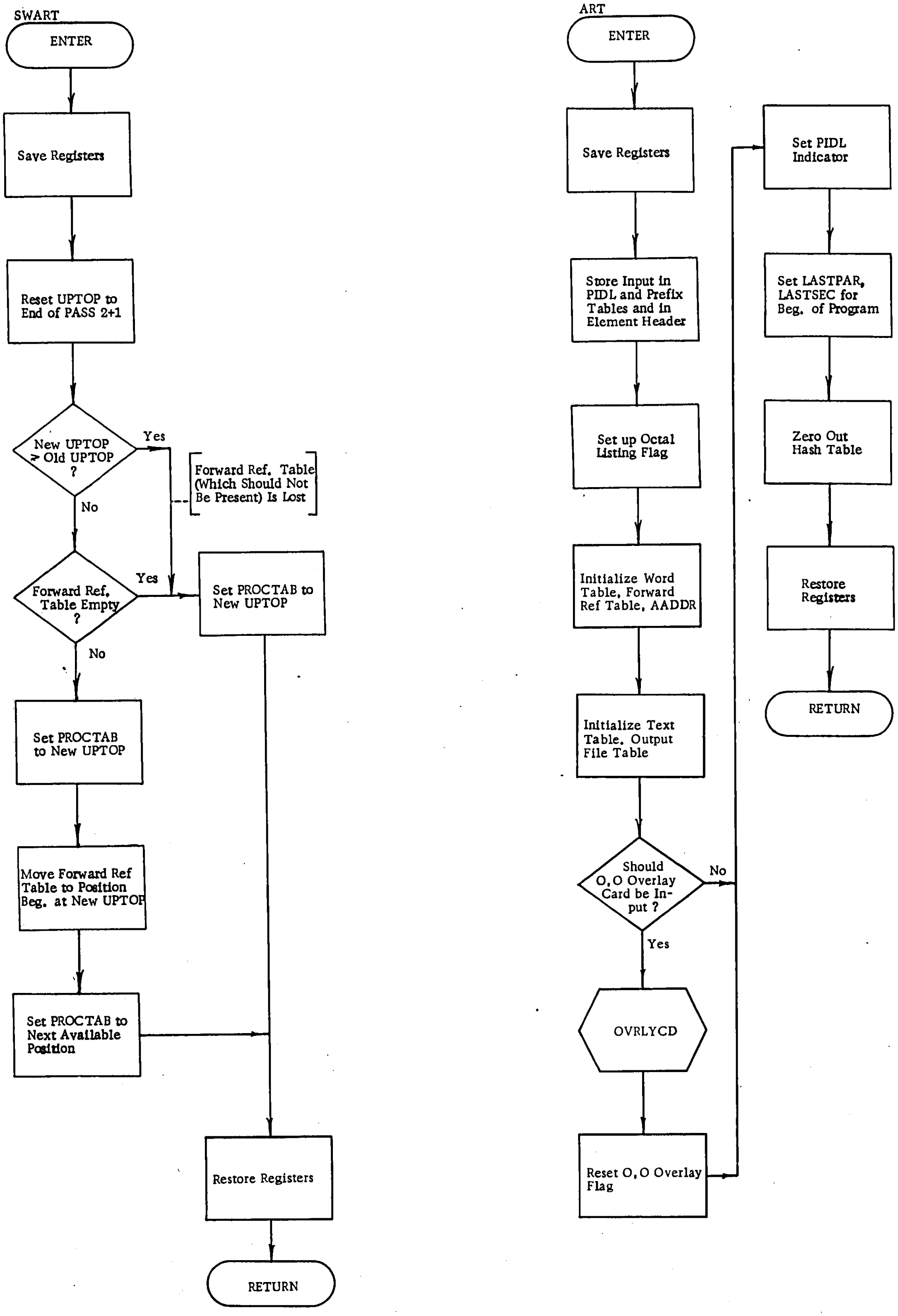


Figure 3-56. SWART and ART Flowcharts

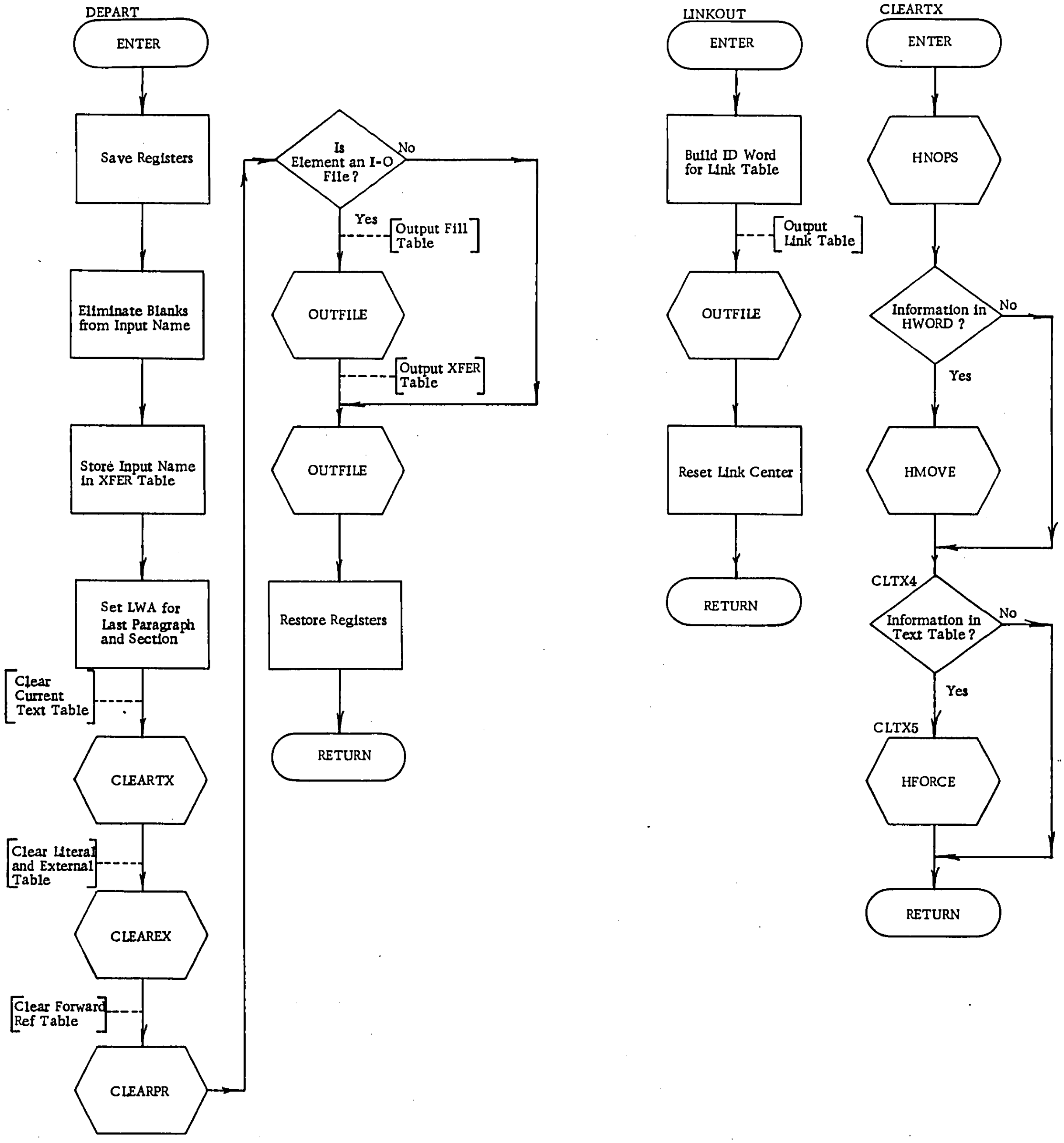


Figure 3-57. DEPART, LINKOUT, and CLEARTX Flowcharts

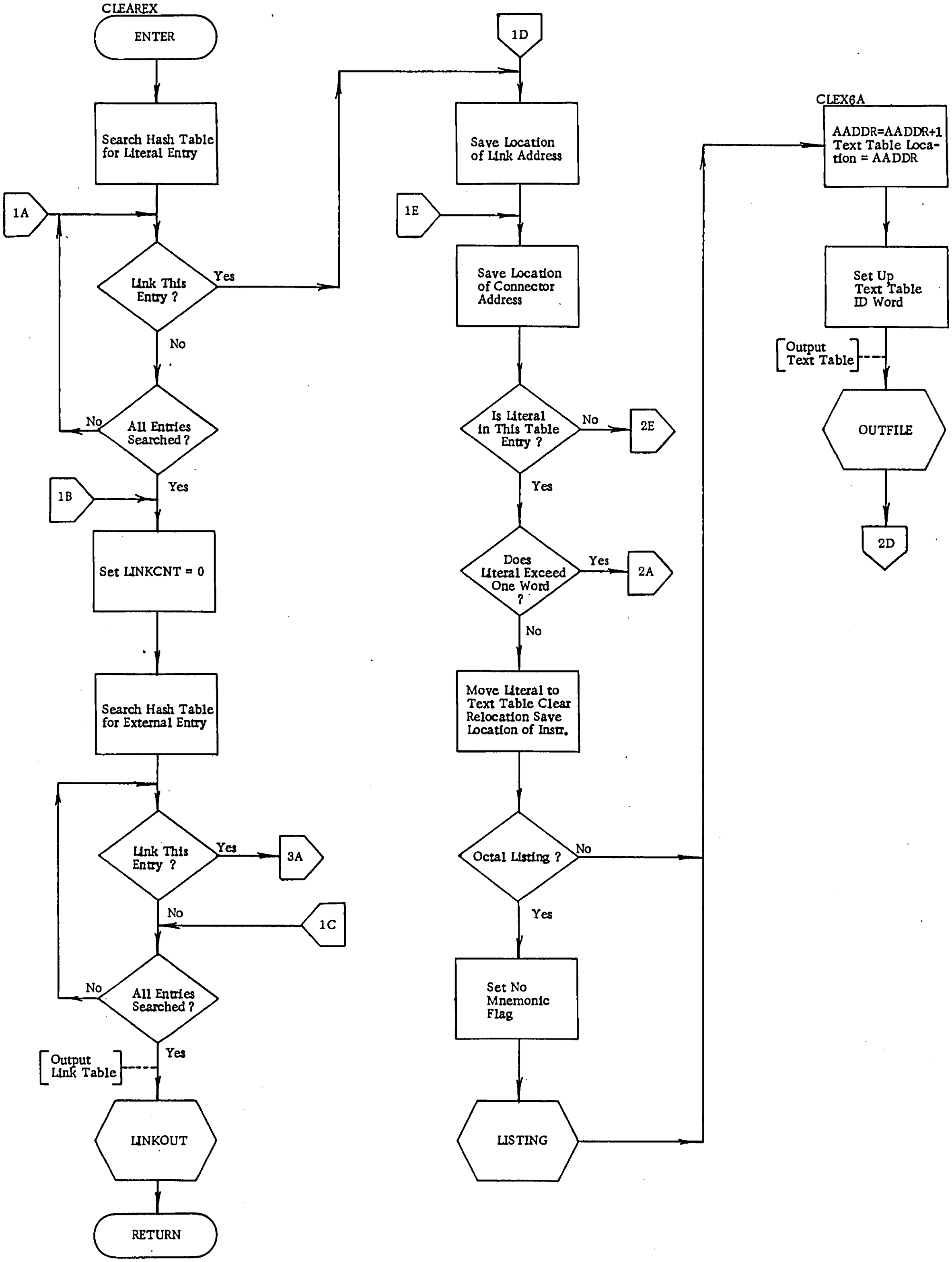


Figure 3-58. CLEAREX Flowchart (1 of 3)

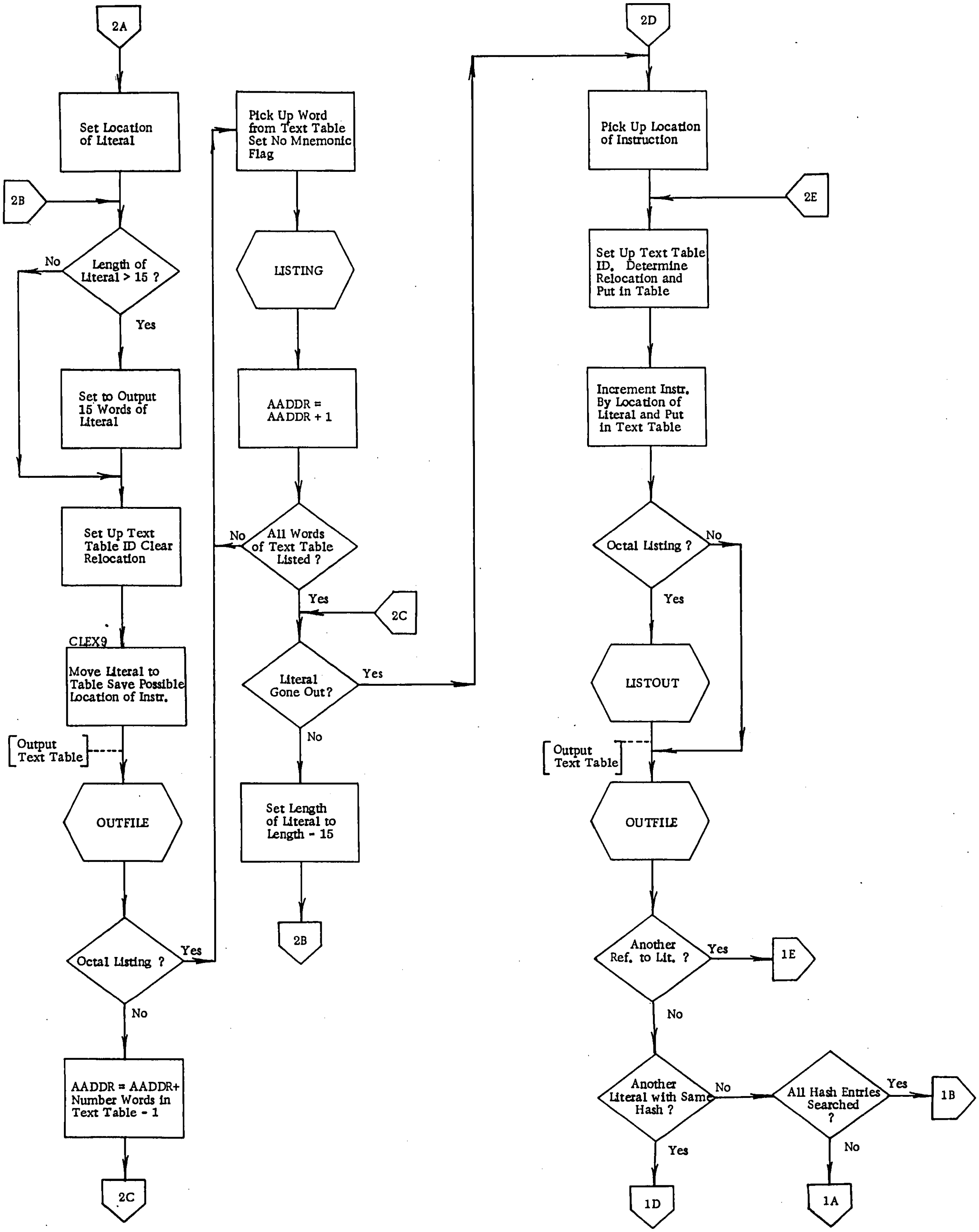


Figure 3-58. CLEAREX Flowchart (2 of 3)

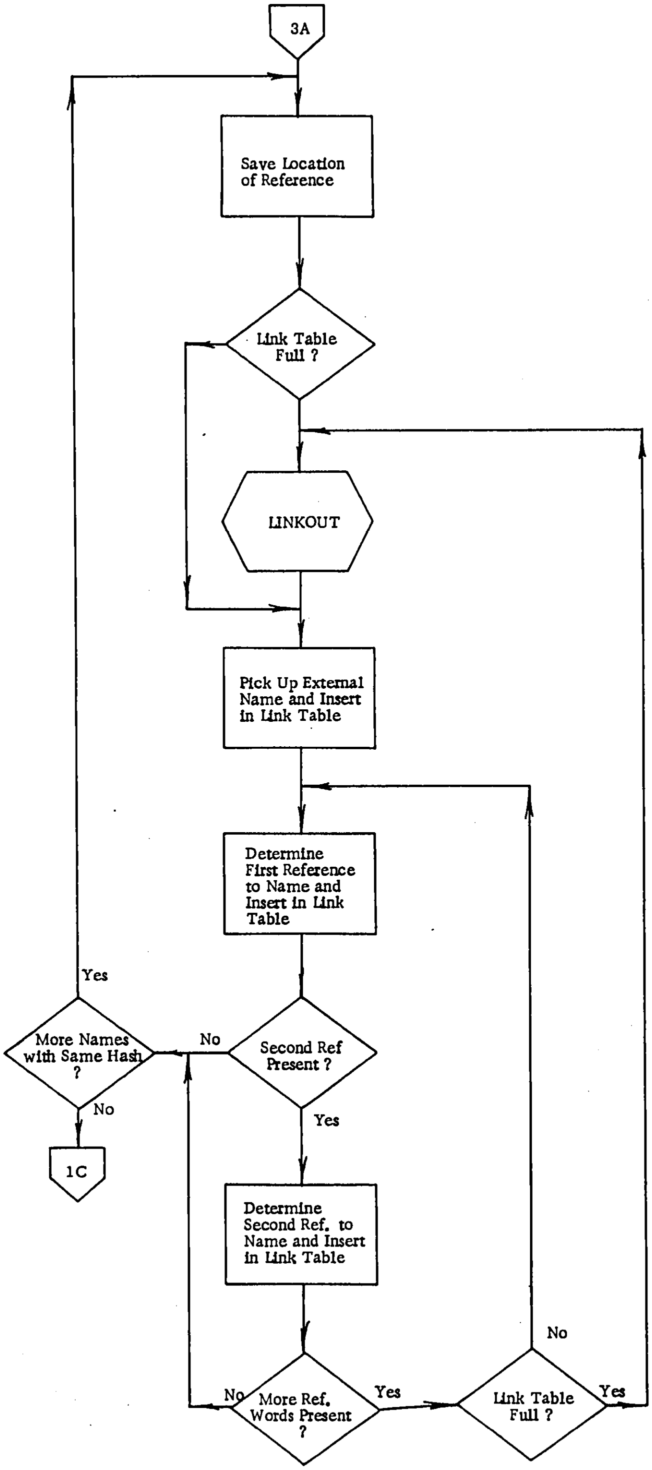


Figure 3-58. CLEAREX Flowchart (3 of 3)

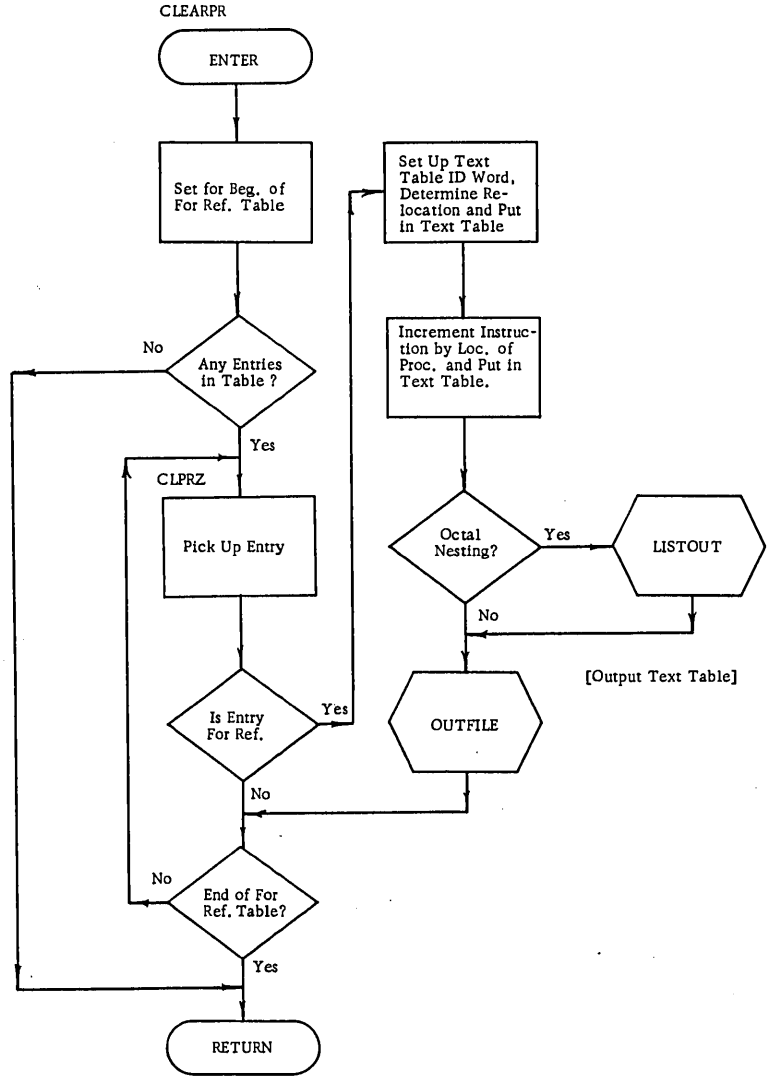


Figure 3-59. CLEARPR Flowchart

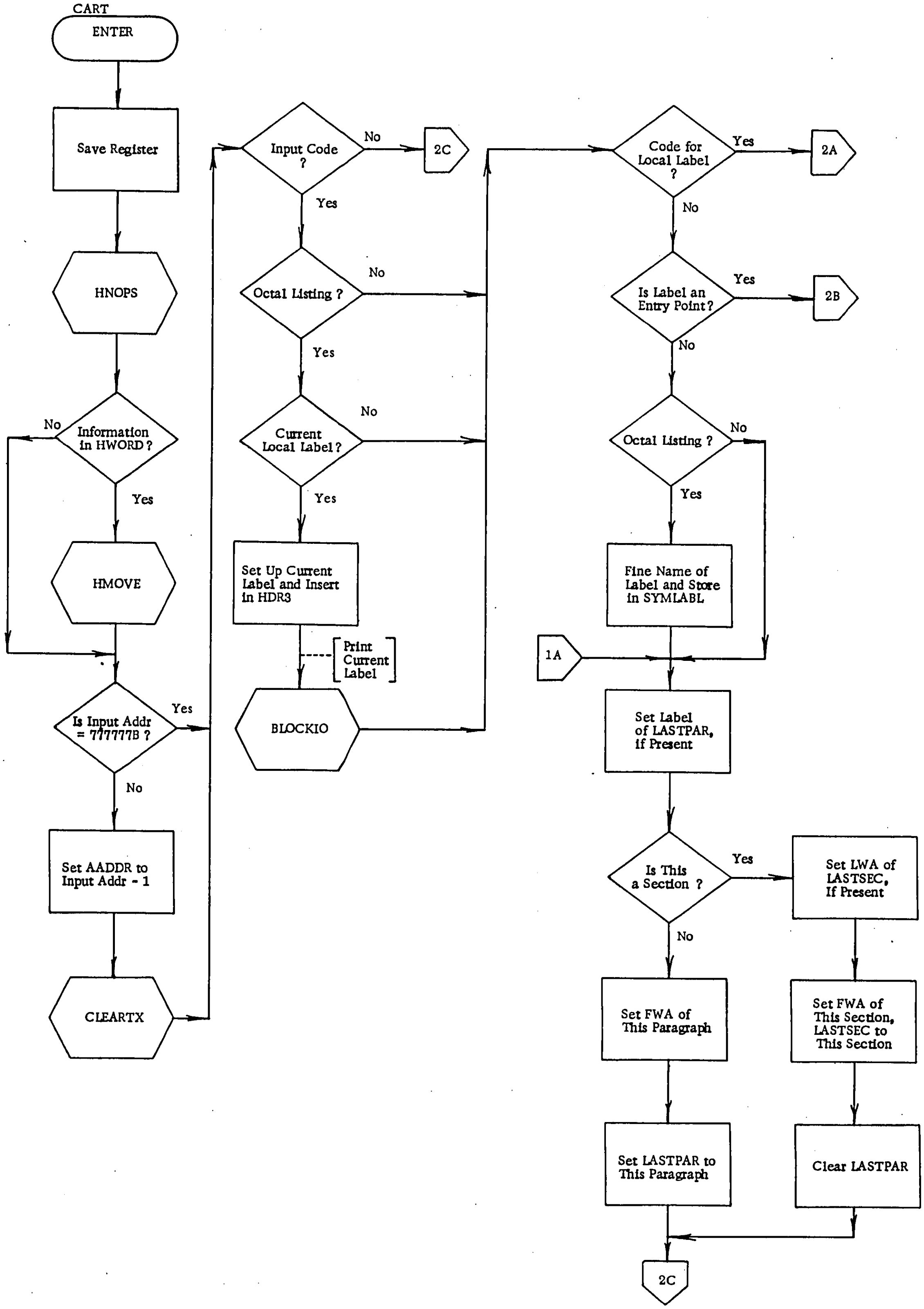


Figure 3-60. CART Flowchart (1 of 2)

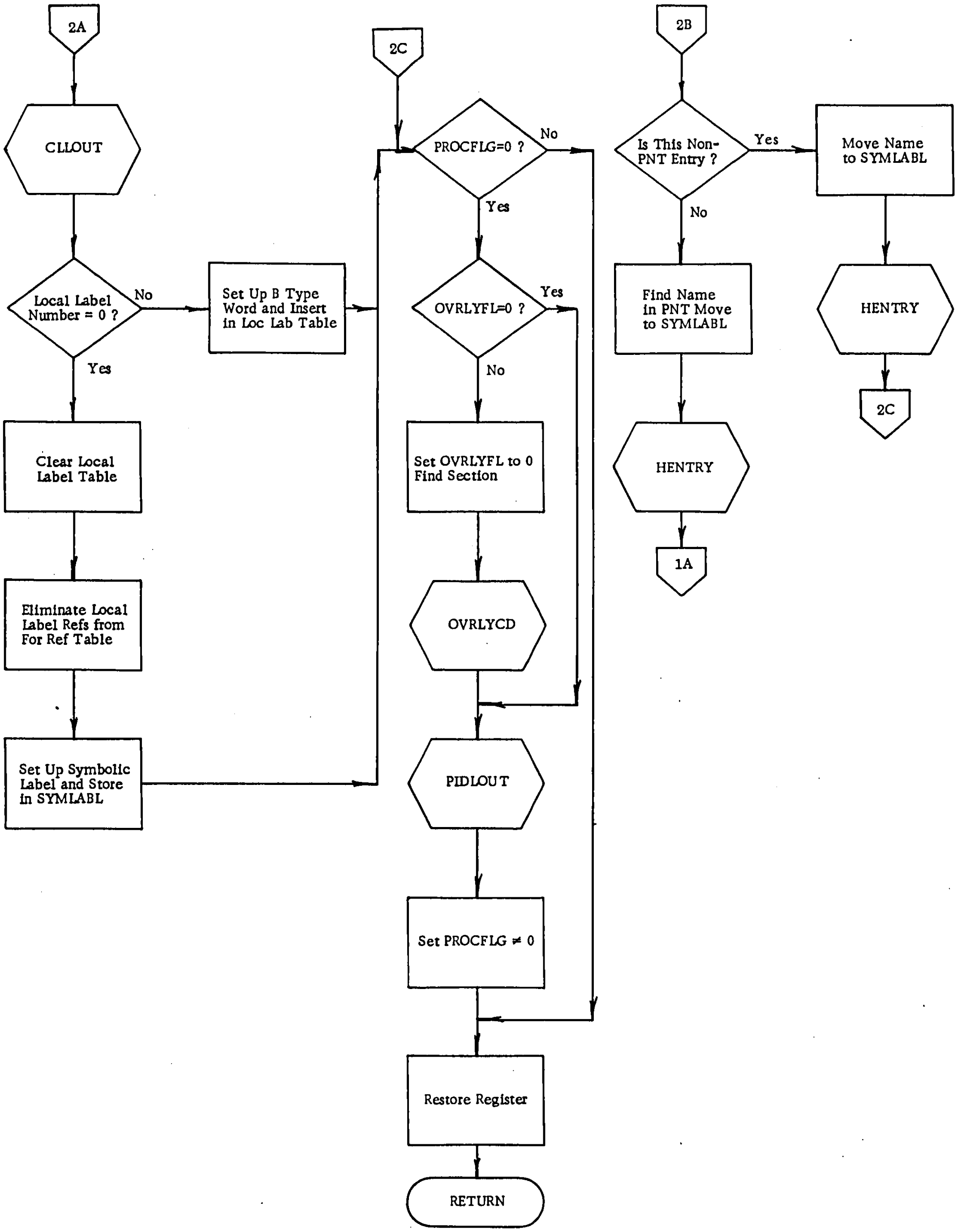


Figure 3-60. CART Flowchart (2 of 2)

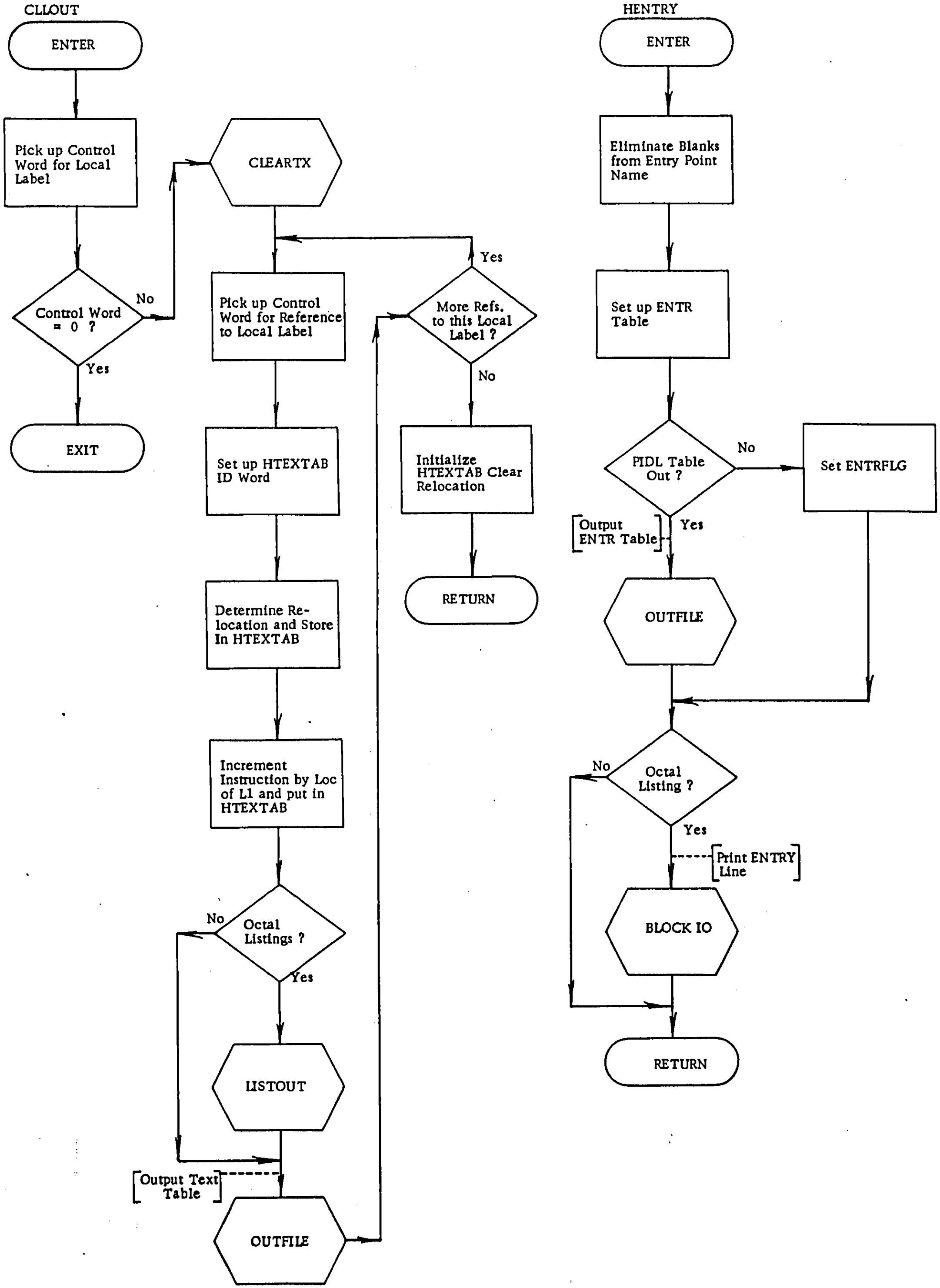


Figure 3-61. CLLOUT and HENTRY Flowcharts

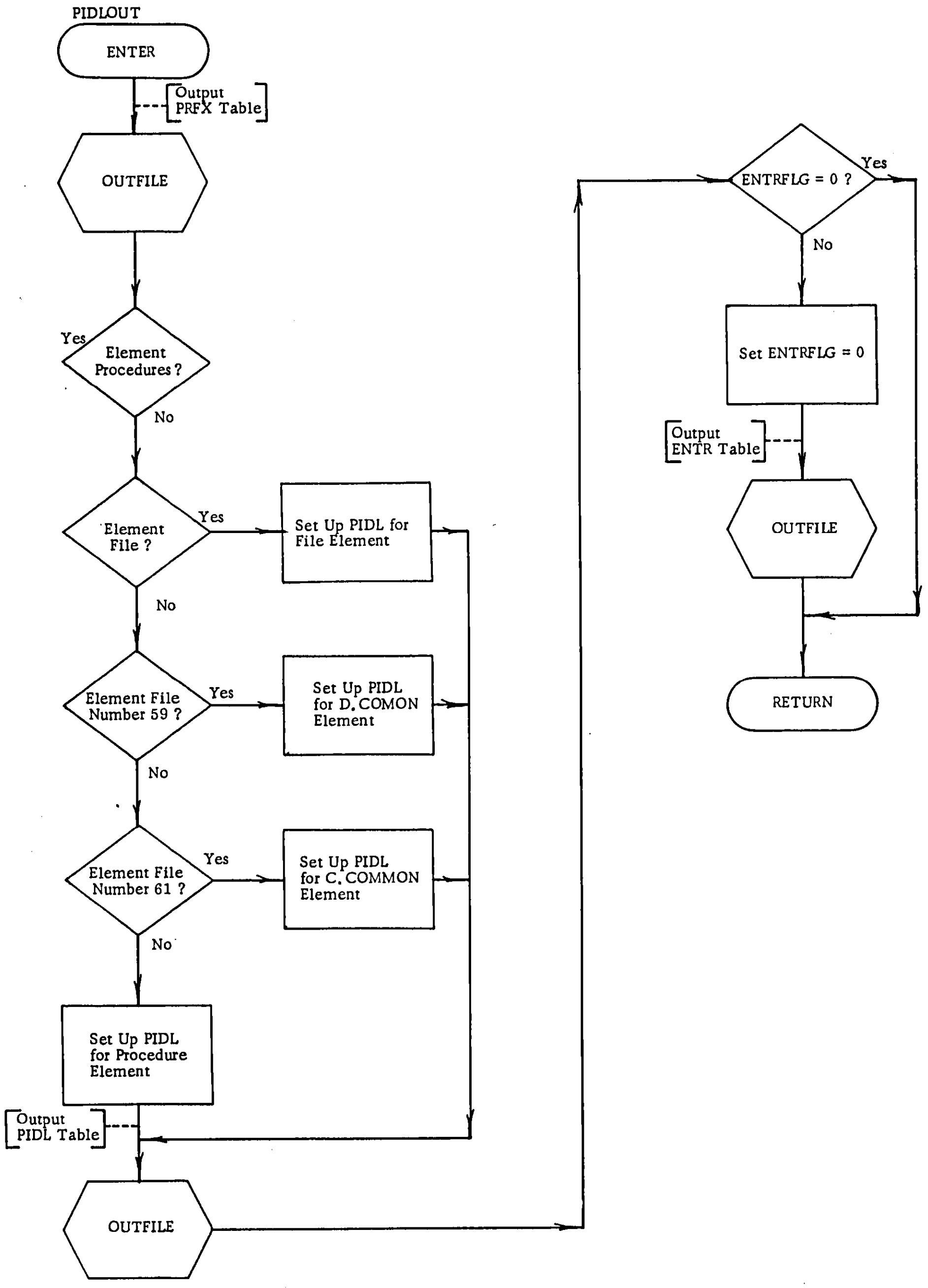


Figure 3-62. PIDLOUT Flowchart

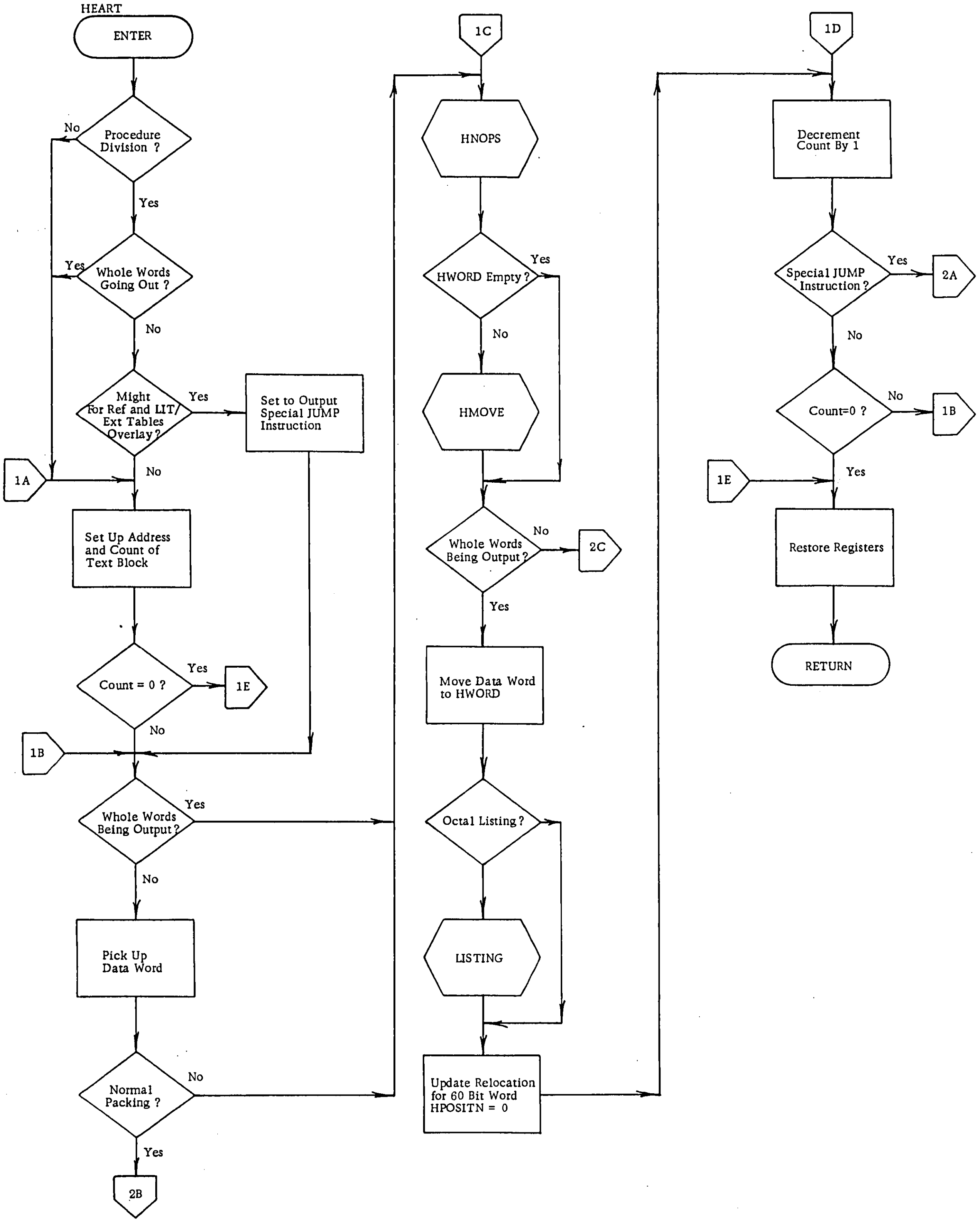


Figure 3-63. HEART Flowchart (1 of 5)

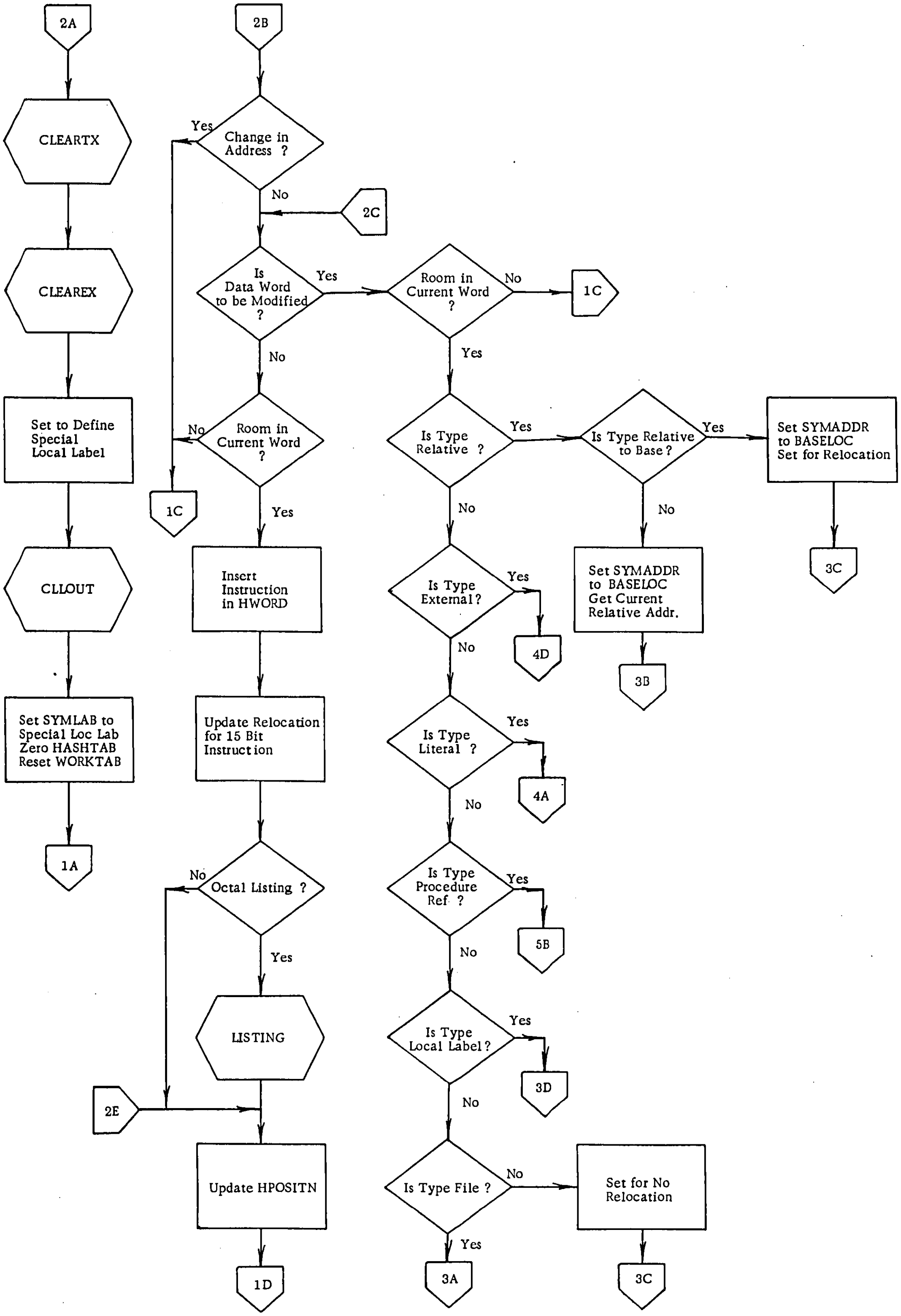


Figure 3-63. HEART Flowchart (2 of 5)

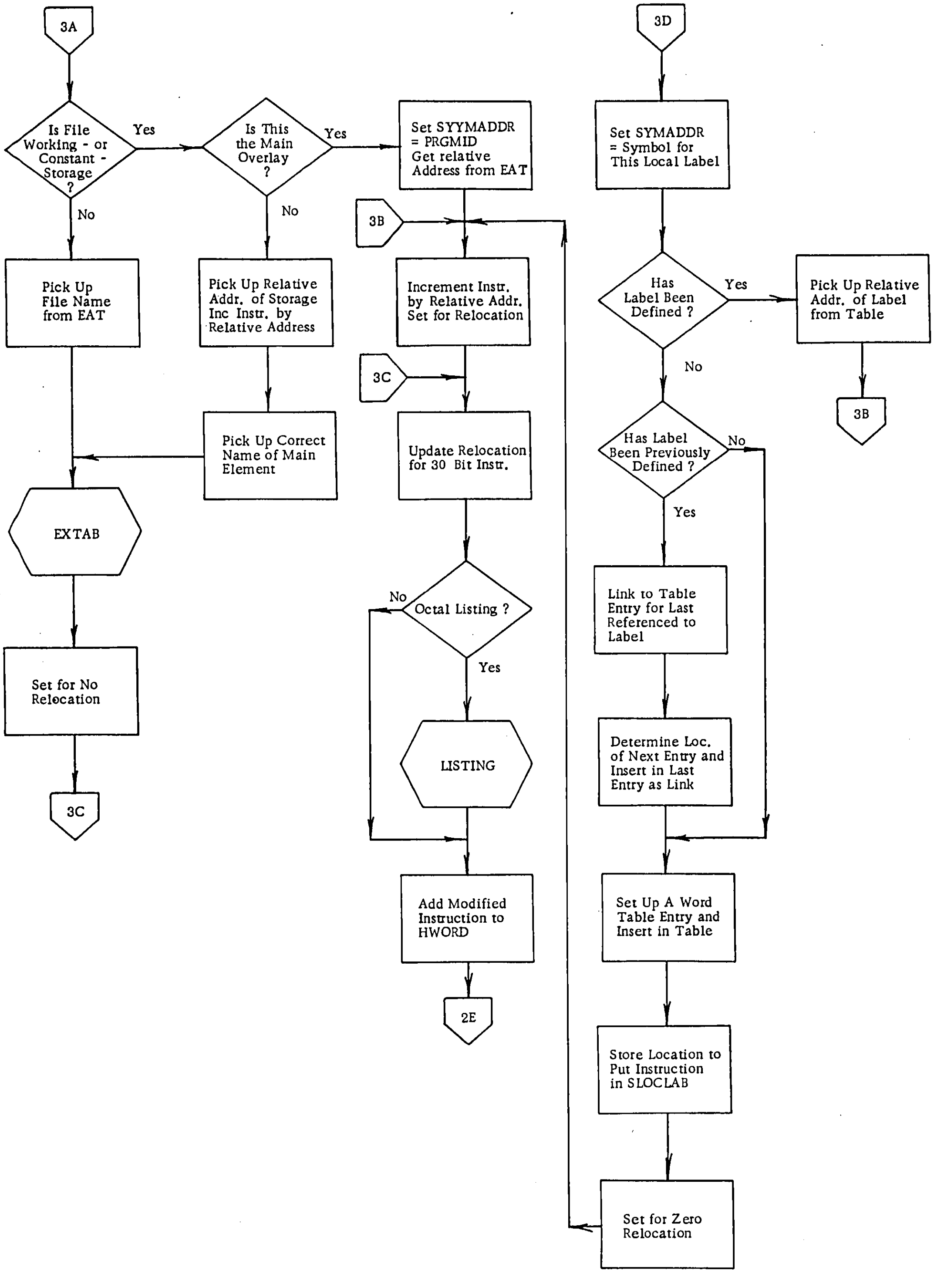


Figure 3-63. HEART Flowchart (3 of 5)

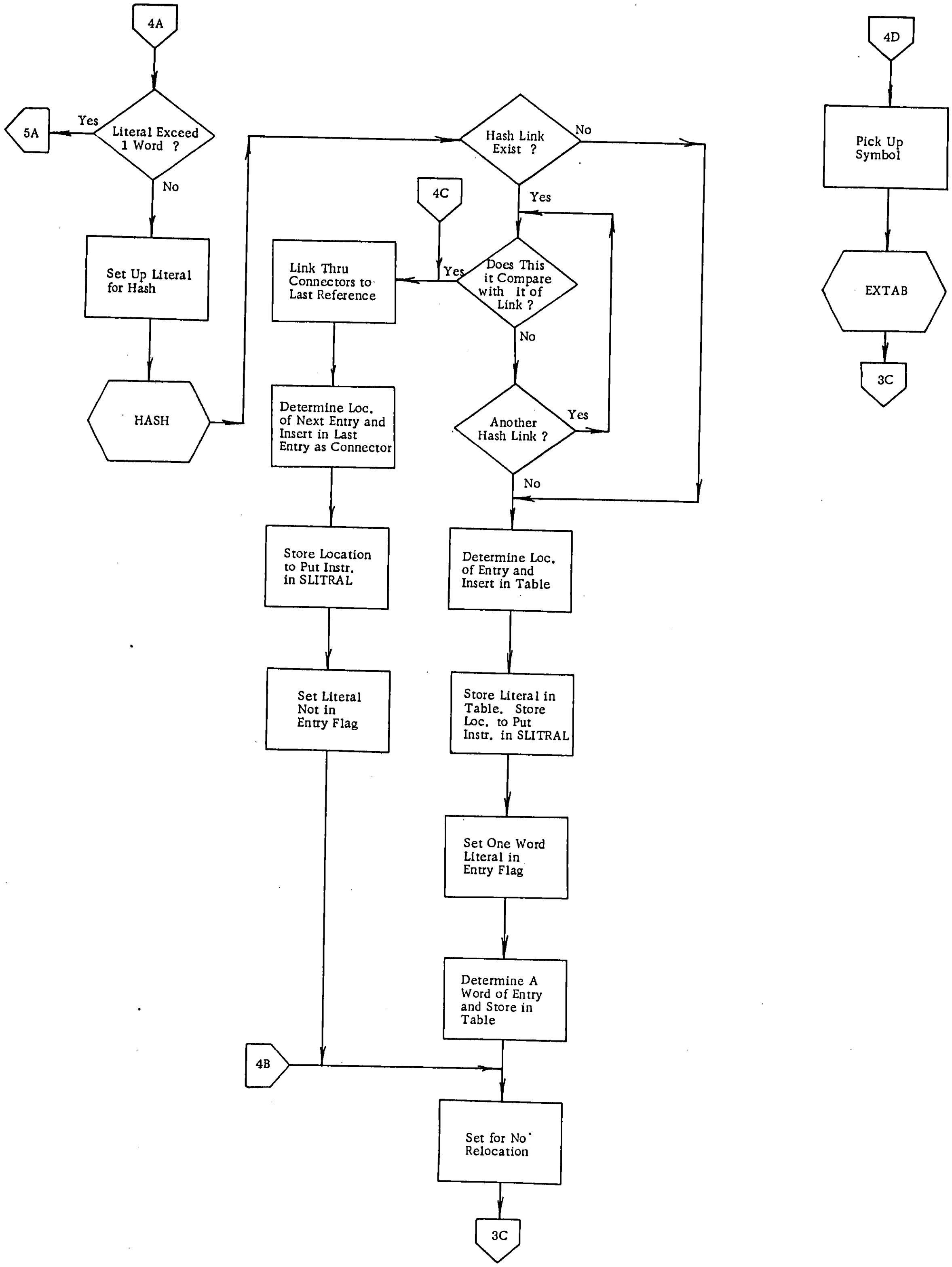


Figure 3-63. HEART Flowchart (4 of 5)

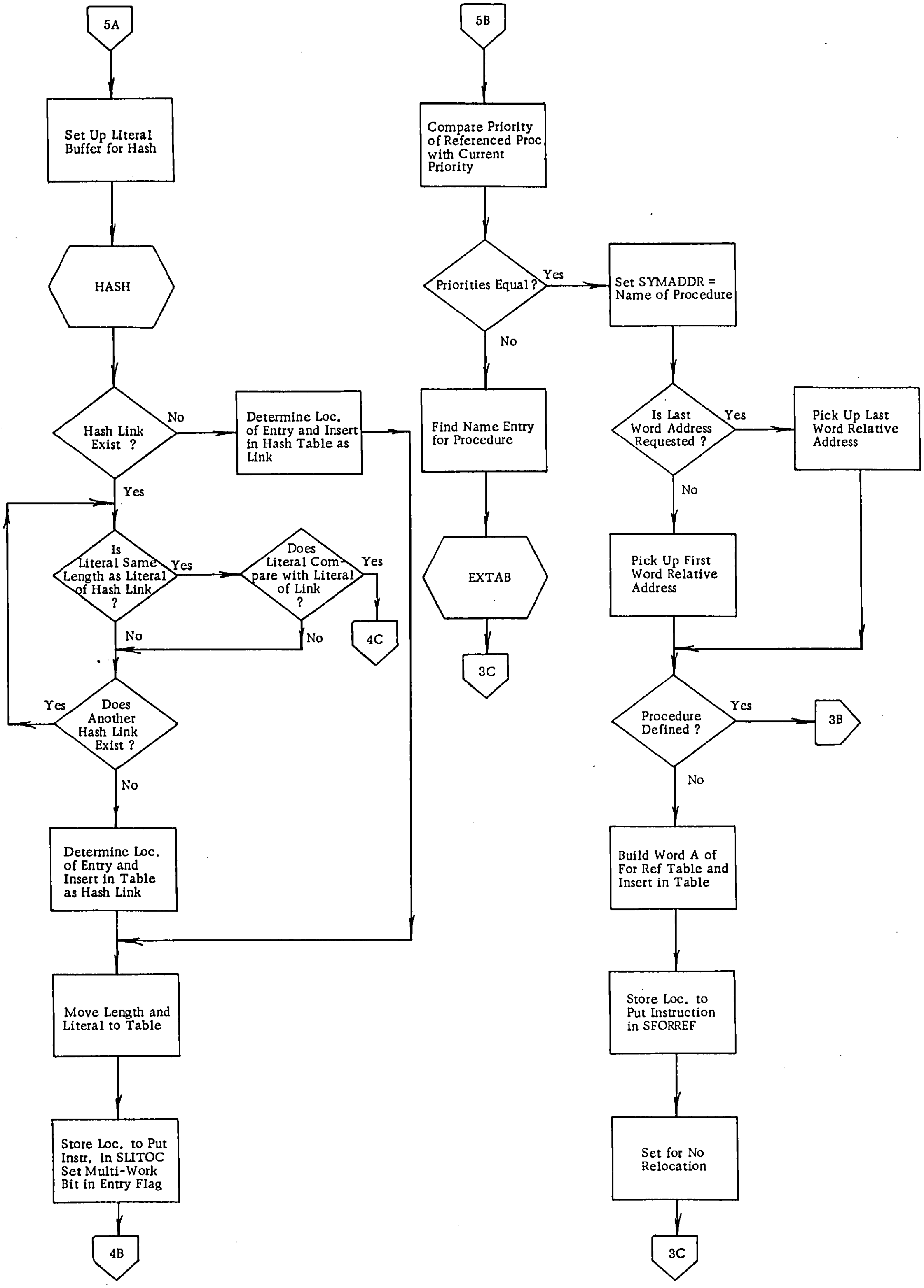


Figure 3-63. HEART Flowchart (5 of 5)

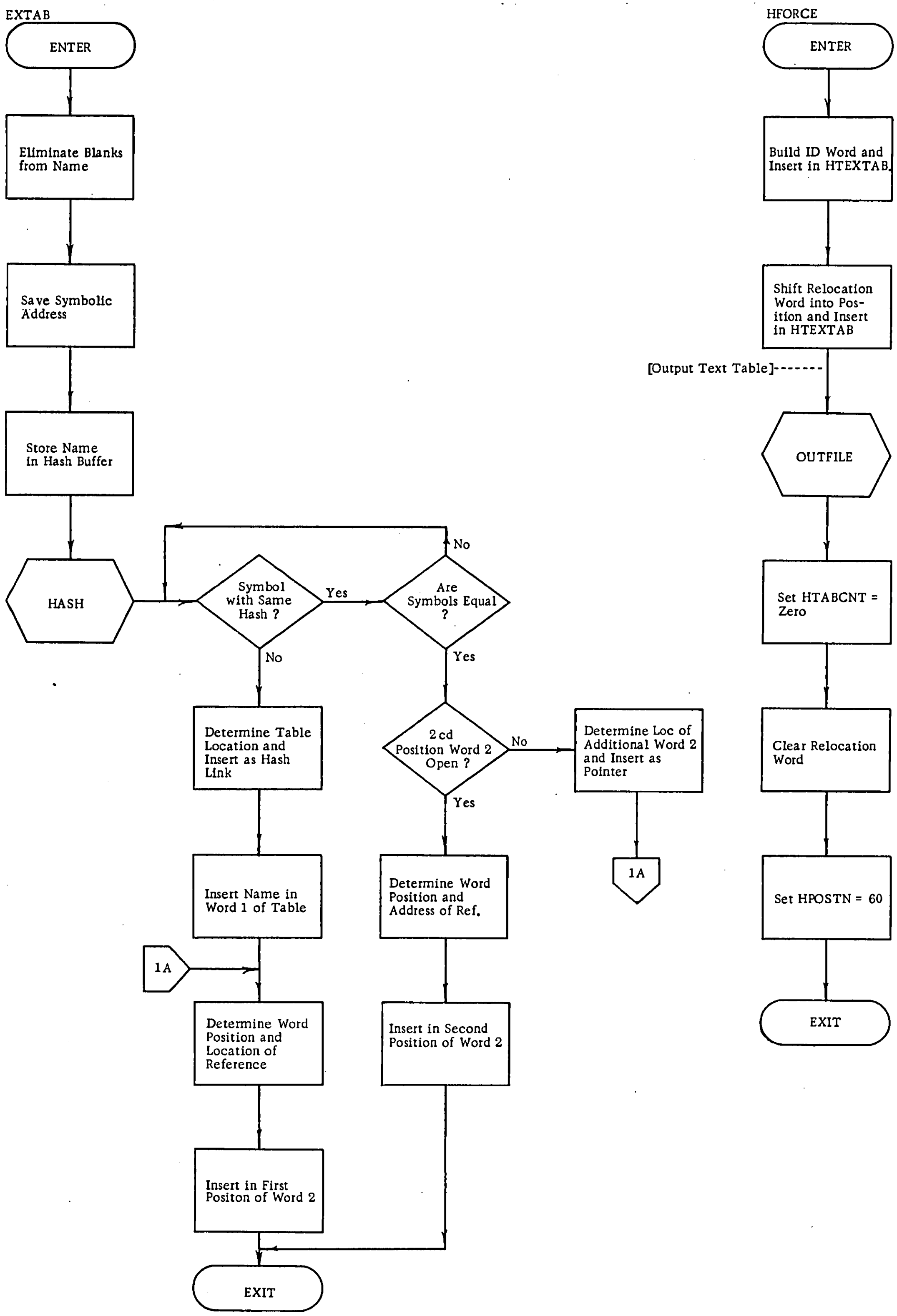


Figure 3-64. EXTAB and HFORCE Flowcharts

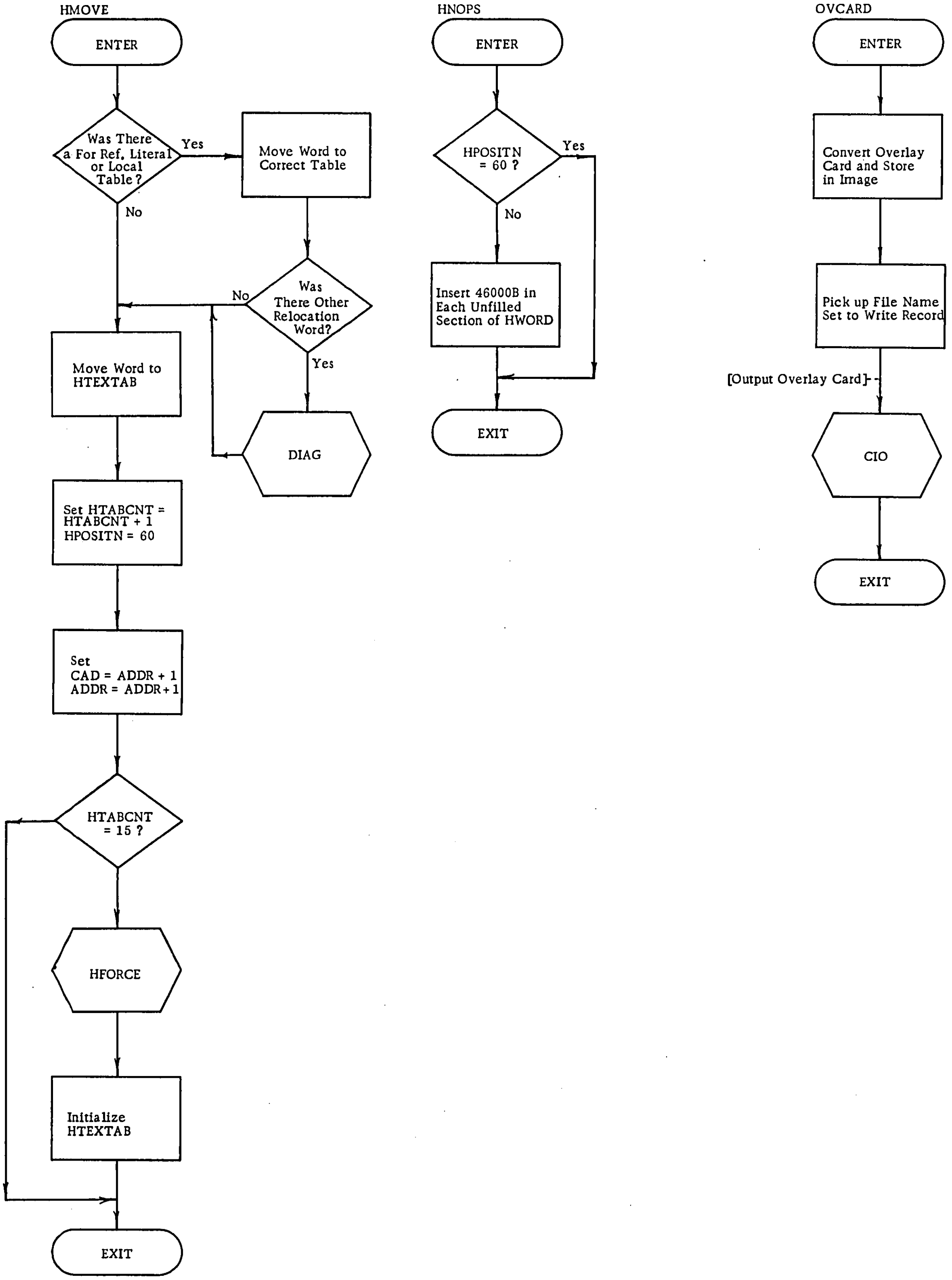


Figure 3-65. HMOVE, HNOPS, and OVCARD Flowcharts

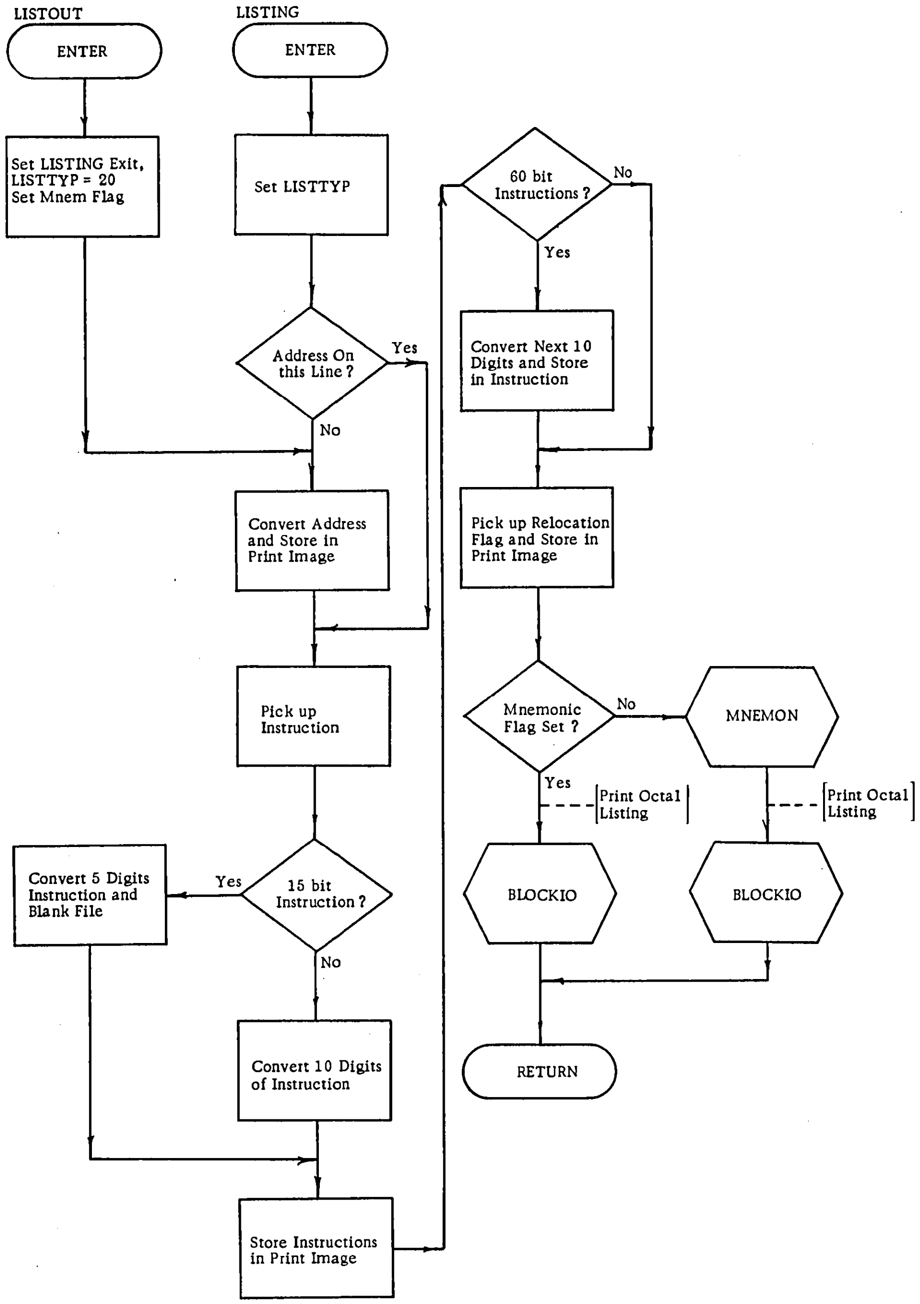


Figure 3-66. LISTOUT Flowchart

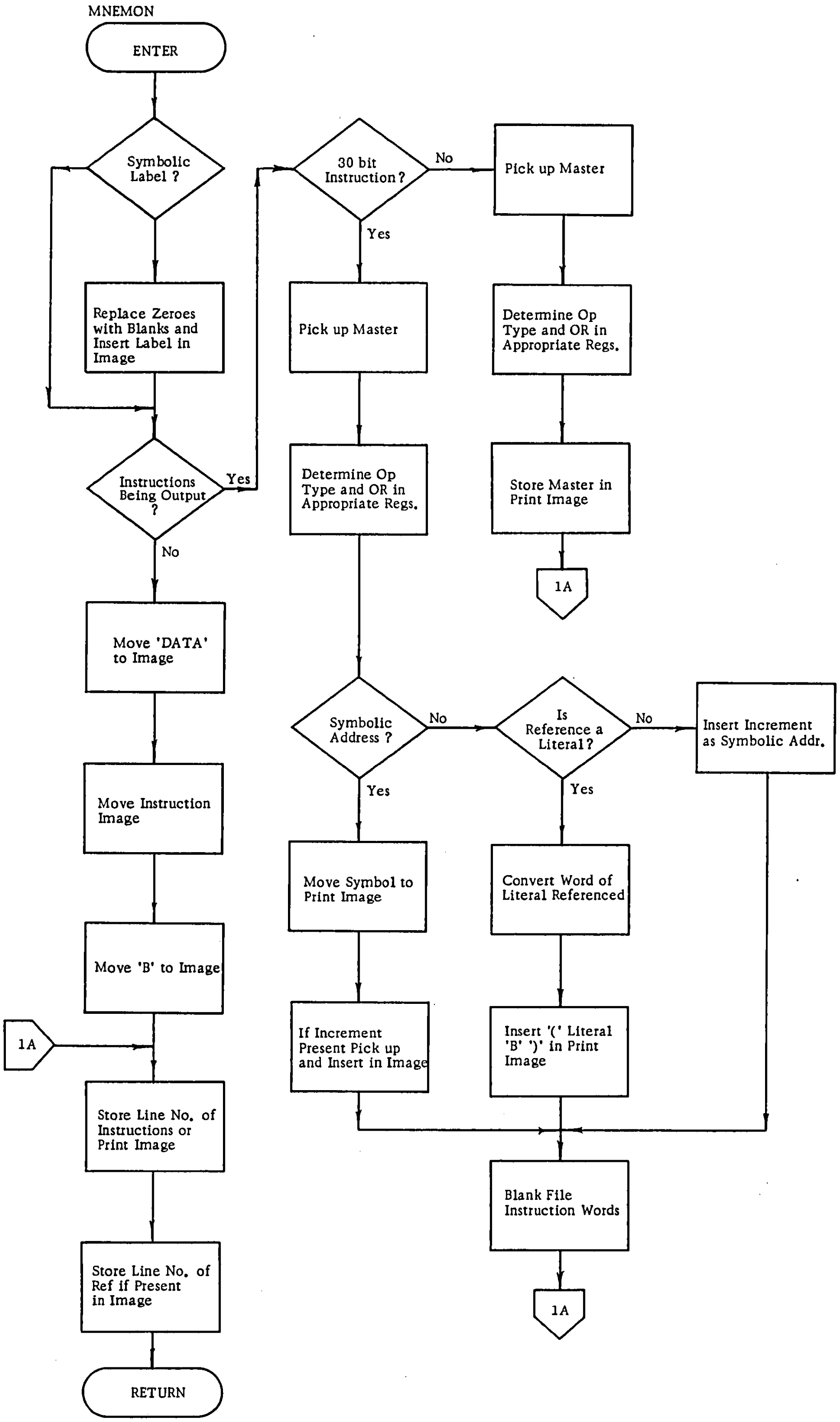


Figure 3-67. MNEMON Flowchart

ASSEMBLER (ART)

ART is called to output all object code to the loader

General call:

RJ Entry Point in ART with the location of an input word in Register X4.

Begin Assembly

(RJ ART)

Input word:

Name of Subprogram	0
42	18

where

Length is not furnished (thus 0) for elements (subprogram) containing Procedure Division code.

This call causes the building of the PIDL Table. If overlays exist, it also causes an overlay card to be output when appropriate. The overlay number is obtained from the PNT, by subtracting 49 from the section number of the first procedure encountered in the assembly. This procedure is "named" on the first subsequent call to the assembler following this call. A (0,0) overlay card may also be generated if there are overlays but no subcompiled programs. The name of the assembled program automatically becomes an entry point at location 0 of that element. Subsequently, the entry point CENT.00 is also attached to location 0 for the main overlay (not in subcompiled) and CENT.nn, where nn is the overlay number for overlays.

Text Control Routine (CART)

CART processes text control and label information. Another subroutine HEART processes data.

(RJ CART)

Input word:

Type	Code	0	Label	Address
3	9	12	18	18

where

Type - 0 if subsequent calls to HEART furnish instructions.
4 if subsequent calls to HEART furnish whole words.

Code - 000 if no label is being assigned with this call.
400 if a local label is to be assigned to this address.
440 is a nonlocal label is to be assigned to this address.
444 if the label is also an Entry Point (defined in PNT as such).
446 if the label is an Entry Point (not defined in PNT).

Label - 0 if code is 000.

PNT location of entry if code is 440 or 444.

Number of local label if code is 400. If number is zero and code is 400, all previously encountered local labels will be initialized as undefined.

Location of name word if code is 446.

Address - Beginning relative address of label and/or the text furnished in subsequent calls to HEART.

777777 if the address is to continue sequentially from the location of the last HEART call.

This call always forces code upper (left-justified) upon the next call to HEART. It can be used to assign a label to the designated address. It can be used to change the type of text being furnished through HEART. It can be used to change the address of the text. It can be used to clear the local label references. More than one consecutive CART call can be used to satisfy multiple label definitions if needed (i.e., one local and one nonlocal reference were made to this location).

CART and DEPART also require that EAT contains the file number and length of blank common storage, while the file's common storage (working or common storage areas) are being output and when instructions are to be output.

All previous and subsequent references to the label as assigned in this call (if any) are satisfied by the assembler. When a local label is defined by this call, all unsatisfied references to this label are satisfied and output. The local label maintains its value until it is removed by a subsequent CART call removing the label. The current line number is maintained by the calling routine and is available to the assembler for use in the octal listings. The current address is maintained by the assembler available to the calling routine so that the calling routine can generate out-of-line code and return.

Entry points cause entries to be built in an ENTR table. PNT entries with EP flags have two defined locations. The call indicates which is being defined.

The assembler places the first and last address of each defined procedure in the PNT entry when the procedure is defined.

Text Data

(RJ HEART)

Input word:

0	Count	Location of Block of Text Data
36	6	18

where

Count is the number of words furnished in this block of text data.

Text data formats are:

A

DATA
60

B

0	U		Instruction
2	1	42	15

or

C

2	U	Type	Reference	Instruction
2	1	9	18	30

where

Text data format is determined by the last CART call.

Reference in C - 0 if no further modification is needed.

Local label referenced X2.

PNT entry location of label referenced.

Buffer address of literal. (First word of buffer contains literal's length if type is 441.)

Buffer address of 7-character external name (left-justified).

Line number of Data Division reference.

Type - 000 if no reference is needed.

100 if this is a local label reference.

200 if this is a PNT FWA reference.

204 if this is a PNT LWA reference.

300 if this is a PNT entry point reference.

400 if this is to be relocated relative to the subprogram base.

404 if this is to be relocated relative to the current instruction position.

440 if this reference causes a literal to be generated.

441 if this reference causes a multiword literal to be generated.

444 if this reference is to an external subprogram.

File number if this is a Data Division reference.

U - 0 if normal instruction packing should be used.

1 if this instruction should begin a new word.

No relocation is allowed for word types A and B above. PNT entries that are external are detected with type 200 for 300 words. All external references are placed in the LINK table automatically by the assembler. Instructions are packed unless otherwise indicated.

When code is to be relocated by the base of the current element and not absolute, the reference must be type 400. Type 400 will not have an entry in the 18-bit reference field.

For type 404, the instruction address field contains the count of compiled words from the current word (positive or 1's compliment arithmetic). Type 404 does not have an entry in the 18-bit reference field.

End of Assembly

(RJ DEPART)

Input word:

Name of Entry Point	0
42	18

Name may be left zero. It is the name placed in the XFER table of binary output for the element. No more calls for this assembly are allowed. The next call must be to ART to begin a new assembly. This call also causes the outputting of all remaining information for this subprogram including the forward references and LINK table data stored in save tables.

Assembler Restriction

The following restriction applies to users of the compiler assembler.

No word of instructions that contains a reference to

1. A procedure name other than an external reference, or
2. A local label, or
3. A procedure division literal generated by the assembler,

may contain another relocatable reference such as

1. Any of the above,
2. Working-Storage or Constant-Storage data items (applies to main overlay only), or
3. Current location counter plus or minus an increment.

The XFER name furnished in the DEPART call is the name of each overlay (priority sections) or the starting point of the main overlay code and zero for all other elements. The compiler must previously have generated an entry point definition for the name of the starting point in the main overlay if the user did not.

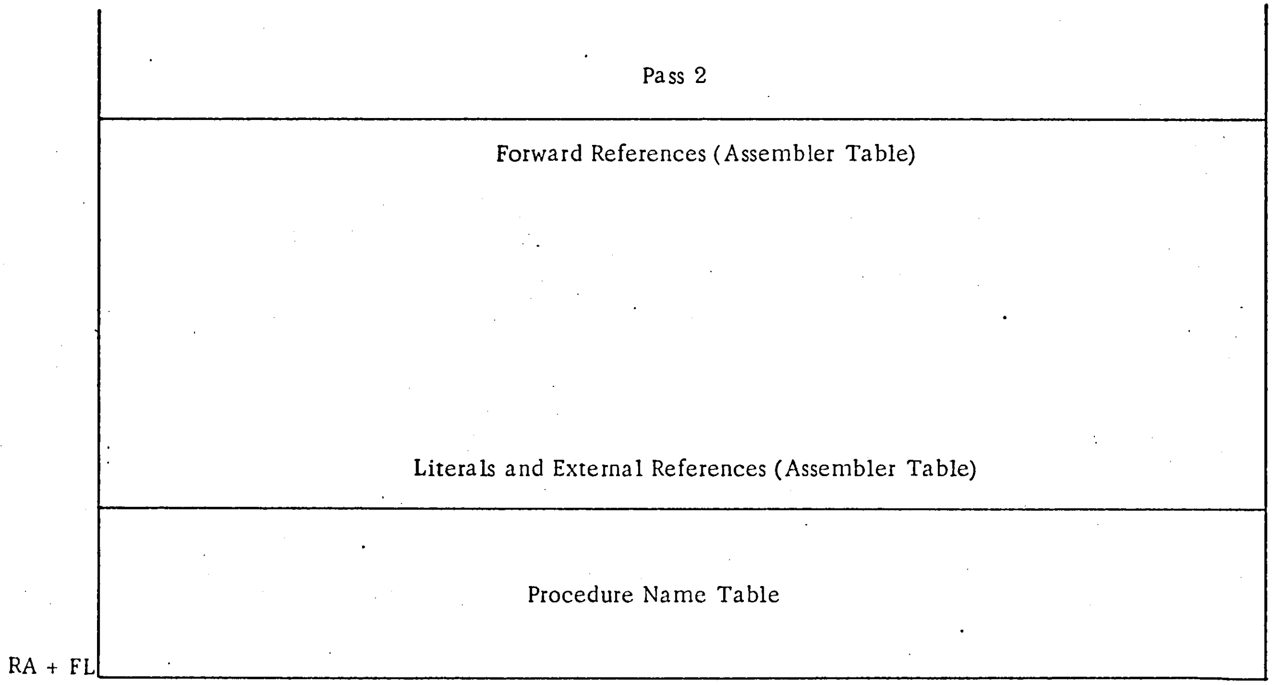
At the beginning of Pass 2, the location of the FORWARD REFERENCE table is moved by use of SWART.

(R(J SWART)

Input word:

0	Location of Top of Pass 2
42	18

1. The assembler will use the area of core between the top location of Pass 2 code and the low end of the Procedure Name Table (PNT) for internal tables.



For Passes 1G and 1H the low limit is the top of the Data Name Table (DNT) if that table extends above the top of Pass 2.

a. The table growing up from Pass 2 contains information about forward references to Procedure names in the compiler program. It also is used to contain information regarding local labels (see Step 5).

b. The table growing down from the PNT contains information about Literal References and External References.

c. If the two tables meet, the table growing down is cleared by the assembler by generating and outputting all of the current saved literals at the current location and completing and outputting the references to them (saved in the same table). A jump around them is generated to connect the instructions being generated. LINK tables are built and output at this time for all of the saved external references. Subsequent entries in this table again start from the PNT. This process is repeated, as necessary, until the assembly ends or the Forward Reference Table reaches the PNT. If the latter occurs, compilations is aborted.

2. The Forward Reference Table entry is as follows:

O	R	WA	PNT Location	Address
3	3	18	18	18

Instruction Word Referencing PNT Entrie (s)

where

R - 1 if only the lower address is to be modified.
2 if only the middle address is to be modified.
4 if only the upper address is to be modified.

WA - 0 FWA.
0 LWA.

PNT location - Points to the procedure whose location is used to modify the upper address if R is 4 or 5.

Address - Is the relative address of the instruction word situated in the next word of this table.

Entries are placed in this table whenever a reference is made to a procedure name as yet undefined to be defined later in the code. This table is cleared at the conclusion of the assembly by completing each reference and outputting the instruction word.

3. The External Reference Table (ERT) entry is as follows:

Symbol	Hash Link
42	18

R	Address	R	Address	Pointer
3	18	3	18	18

where

R - 1 if the lower instruction of the word at the corresponding address is to be modified by the value of Symbol.

2 if the middle instruction of the word at the corresponding address is to be modified by the value of Symbol.

4 if the upper instruction of the word at the corresponding address is to be modified by the value of Symbol.

Address - Is the relative address of the word to be modified by Symbol.

Pointer - Is a link to the next word in this table containing references to this symbol (zero if none).

Hash link - Is the link to the next external Symbol with the same hash (zero if none).

Symbol - Is the 7-character name of an external Symbol.

Further references to Symbol are indicated using the same entry format. However, the symbol word is not repeated.

Each time a new symbol is referenced, causing an entry in the table of both a reference word and a symbol word, the symbol is hashed by calling HASH and the table location of the symbol is placed in the left 20 bits of a general HASH table word. The HASH table entries are zeroed when the Table of External Symbols and References is cleared. Clearing will be done by taking each nonzero link from the HASH table and following that set of hash links (from symbol to symbol) building link table entries for each reference to each symbol.

Entries are kept in this table rather than output when reference is defined because all references to an external symbol are placed with the symbol in the link table.

4. References to literals are entered as follows:

A

Code	Address	Connector	Link
6	18	18	18

B

0	Length
42	18

C

Literal

D

Instruction Word

where

Code is subdivided as follows:

L	M	0	0	R
1	1	1	1	2

L - 0 if this reference does not contain the literal.
1 if this reference also defines the literal.

M - 0 if the literal is one word long.
1 if the literal is more than one word long.

Unused if L is 0.

R - 0 if this reference to the literal is in the lower address of the instruction word.

1 if the reference to the literal is in the middle address of the instruction word.

2 if the reference to the literal is in the upper address of the instruction word.

Address - The address of the instruction word.

Connector - The location in this table of the next reference to the same literal. 0 if none.

Link - A hash link to another A word whose entry contains a literal with identical hash if L is 1.

A pointer to the literal's original A word if L = 0.

Length - The number of words for this literal if M is 1. Nonexistent if M is 0.

An "A" type word exists for each reference to a literal entry.
A "B" type word exists if M is 1 (and L is 1).
A "C" type word exists if L is 1.
A "D" type word exists for each reference to a literal entry.

As an entry in the table is placed, the literal is hashed and the table location of the entry ("A" type word position) is placed in the right 20 bits of the general HASH table word (also used for the DNT hash, but zeroed by ART) whose table position was determined by the hash. The HASH table entries are zeroed if the table of references to literals is cleared.

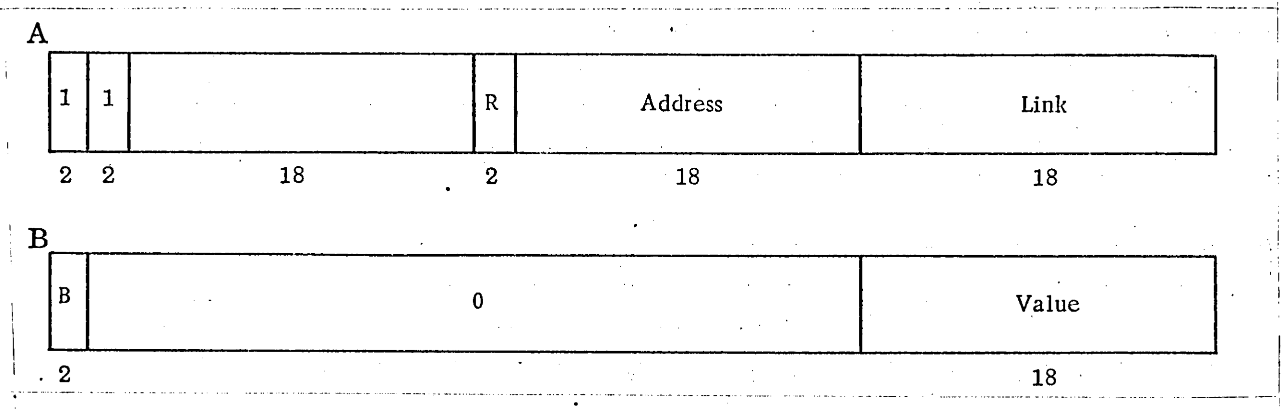
Clearing is done by taking each nonzero link from the HASH table and following that set of links (from literal to literal) outputting the literals from the current object address assignment and completing and outputting the instruction words referencing each literal.

5. Local labels are labels that are internally defined by the code generator or other calling routines to the assembler and are used for referencing within a given sentence or related group of codes. They may be reinitialized for use in each new sentence or logical group of code.

There is a maximum number of allowable local labels. They are sequentially numbered from the 1 to the maximum number and may be used in any order.

The assembler remembers all references and definitions of local labels in the following manner:

> Three tables of the same size, each of which is equal to the maximum number of local labels, are used. One of the tables has the following entry format:



Word type A is used as an entry for references to a local label when the local label is not yet defined. Word type B is used after it has been defined. The local label is used as an index to find the table position of the entry pertaining to it. If no reference or definition of the label has been made, the entry contains all zeros.

Value	-	The address assigned to the label.
Address	-	The text address of a word containing a forward reference to the local label.
R	-	The position in the word to be loaded at address that referenced the local label. 0 - lower address 1 - middle address 2 - upper address
Link	-	The link to an entry elsewhere containing more references to the same local label.

A second word of the table is situated immediately behind the first. It contains the reference to the local label and whose address is specified in word type A.

When more than one word is required to describe all the references to a local label, its respective word type "A" in the local label tables links to another word type "A" located in the Table of Literals and External References. The word that referenced the local label is placed in the following word of the same table. If necessary, the link points to further references in the same table.

When a local label is defined and forward references were made to it prior to definition, each prior reference is completed and its word output as the word type "B" is entered in the table. Subsequent references to the local label whose entry is a word type "B" are satisfied and output as they are encountered.

Each element produced by the COBOL compiler by one "assembly" contains a PIDL table, ENTR table(s), TEXT tables, and an XFER table. LINK tables appear for the Procedure Division. FILL tables (to satisfy blank COMMON references) also appear for file table elements. LINK tables are used for any unsatisfied reference. XFER tables are blank except for Procedure Division elements.

a. When no length is furnished for a PIDL table, the loader uses the highest relative address to determine space allocation. Table space in such elements are followed by code to properly define the highest relative address.

b. When dealing with a procedure element and procedure entry point with the same name, the first and last address is placed in the PNT only for the procedure element.

c. Each overlay has one entry point, the first location of the overlay which is the first location of the jump table for the overlay.

d. NO's are furnished, where necessary, by the assembler to space instructions.

PASS 1H AND ELEMENTS

All data sections (COMMON-STORAGE, WORKING-STORAGE, and CONSTANT) are output to the CDC loader by Pass 1H. Flow charts for Pass 1H are shown in Figures 3-68 through 3-75.

Each literal is converted and positioned as described by its data descriptor. The initial content of any field that does not have an initial value designated by the source program is not initialized by the compiler. Values described as COMPUTATIONAL-1 are output as full-word synchronized, un-normalized floating-point numbers. Decimal scale factors are reflected in the use of these fields but not the representation. COMPUTATIONAL-2 items are output as full-word synchronized, normalized floating-point numbers, adjusted properly for any decimal scale factor.

Noncontiguous (Level 77) literals are output as synchronized items positioned within the word as described by the appropriate data descriptor.

Text data format for data references in file tables is as follows:

2	0	Type	Reference	BCP	0 0	RRWL
2	1	9	18	6	6	18

1. File Table references to items in or attached to File Tables (including the same file) such as "Address of OP Word," "First Word Address," and "Key Position (variable record)," require the following parameter values:

   TYPE = 444_8.

   REFERENCE = Location in indicated DNT file of FD + 12. FD + 12 contains the seven-character SCOPE file name that was ASSIGNED left justified and zero filled.

   BCP = 0 for "Address of OP Word" and "First Word Address."
   = BCP from referenced data item for "Key Position."

   RRWL = Location of "OP Word" (20) and "First Word" (22) relative to FET1 (FD + 12).
   = Location of "Key Position" (if in record of same file) field. Location of "Key Position" = RRWL from referenced Data Item + 22.

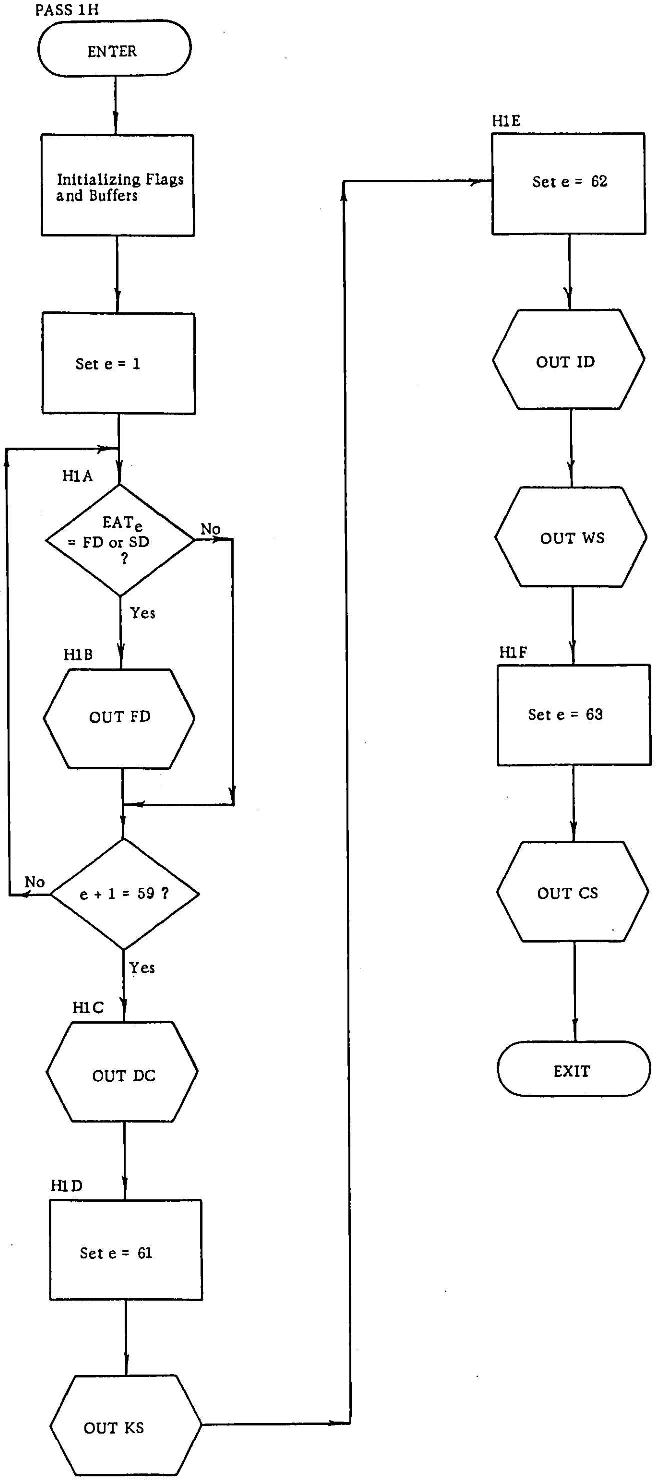


Figure 3-68. Pass 1H Flowchart

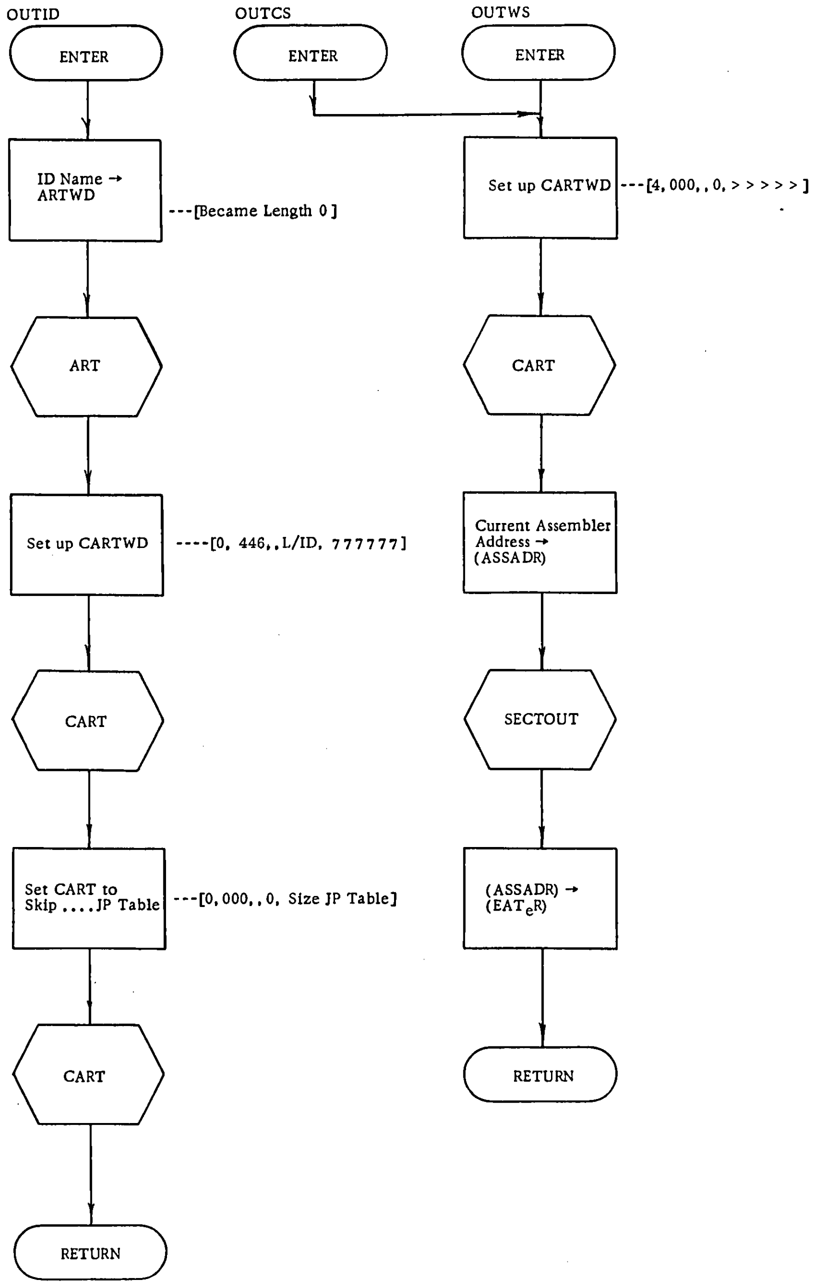


Figure 3-69. OUTID and OUTCS Flowcharts

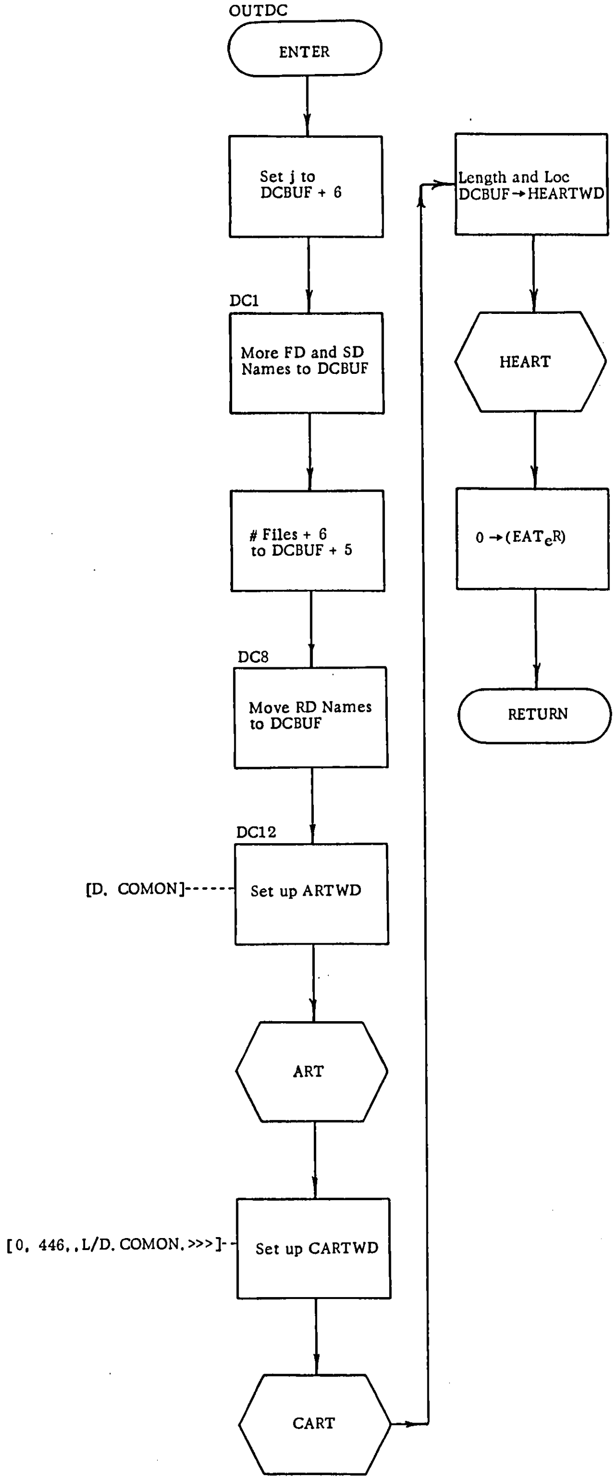


Figure 3-70. OUTDC Flowchart

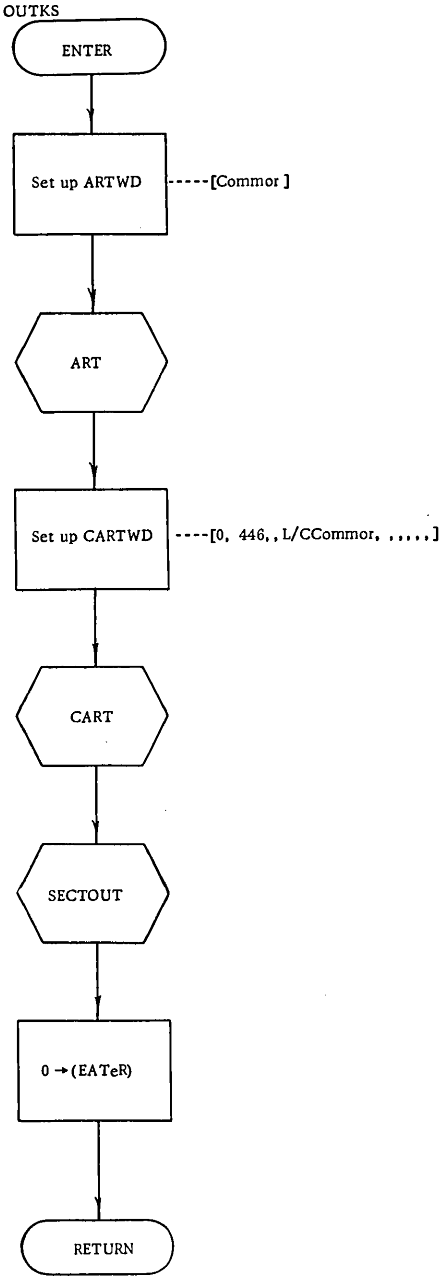


Figure 3-71. OUTKS Flowchart

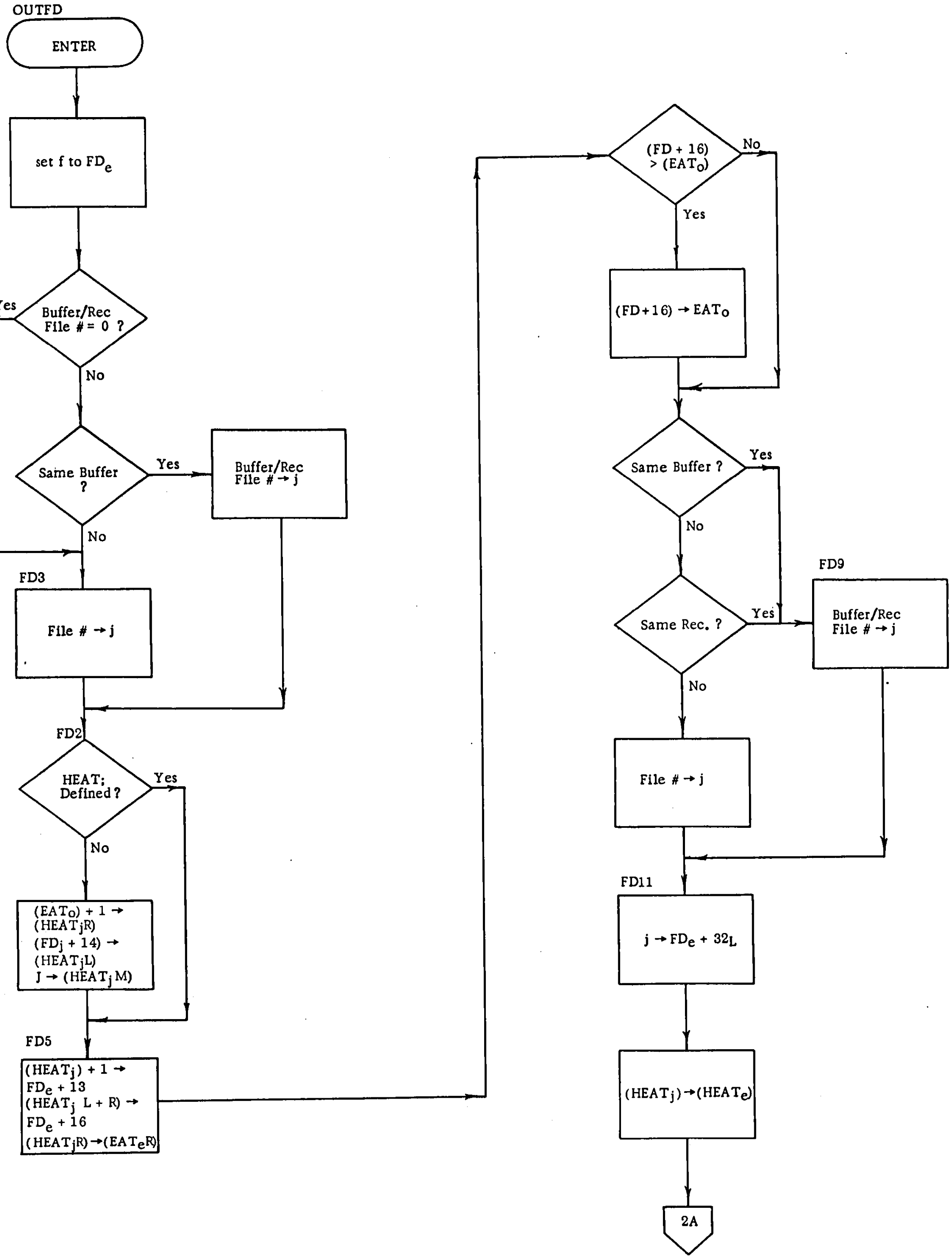


Figure 3-72. OUTFD Flowchart (1 of 3)

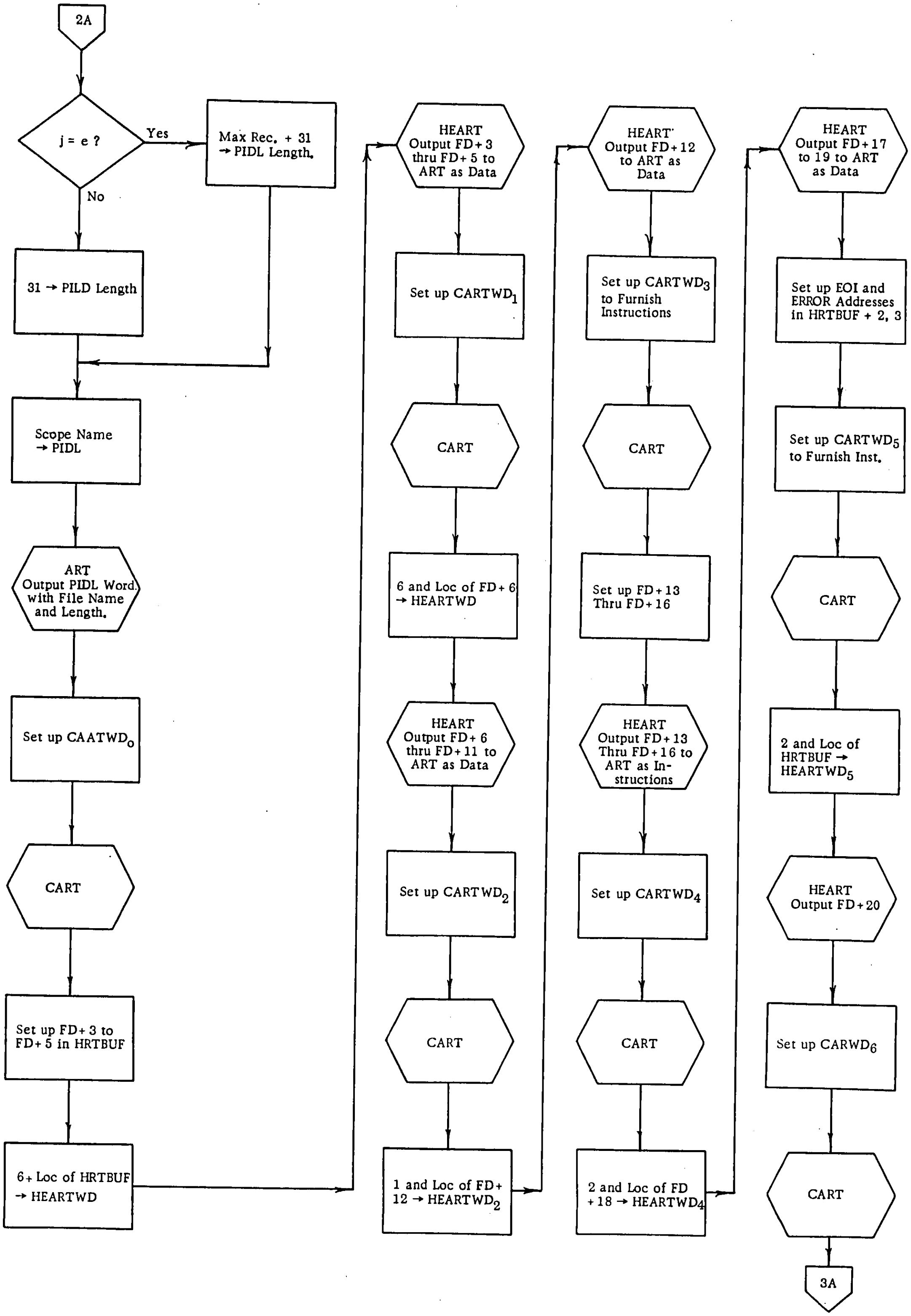


Figure 3-72. OUTFD Flowchart (2 of 3)

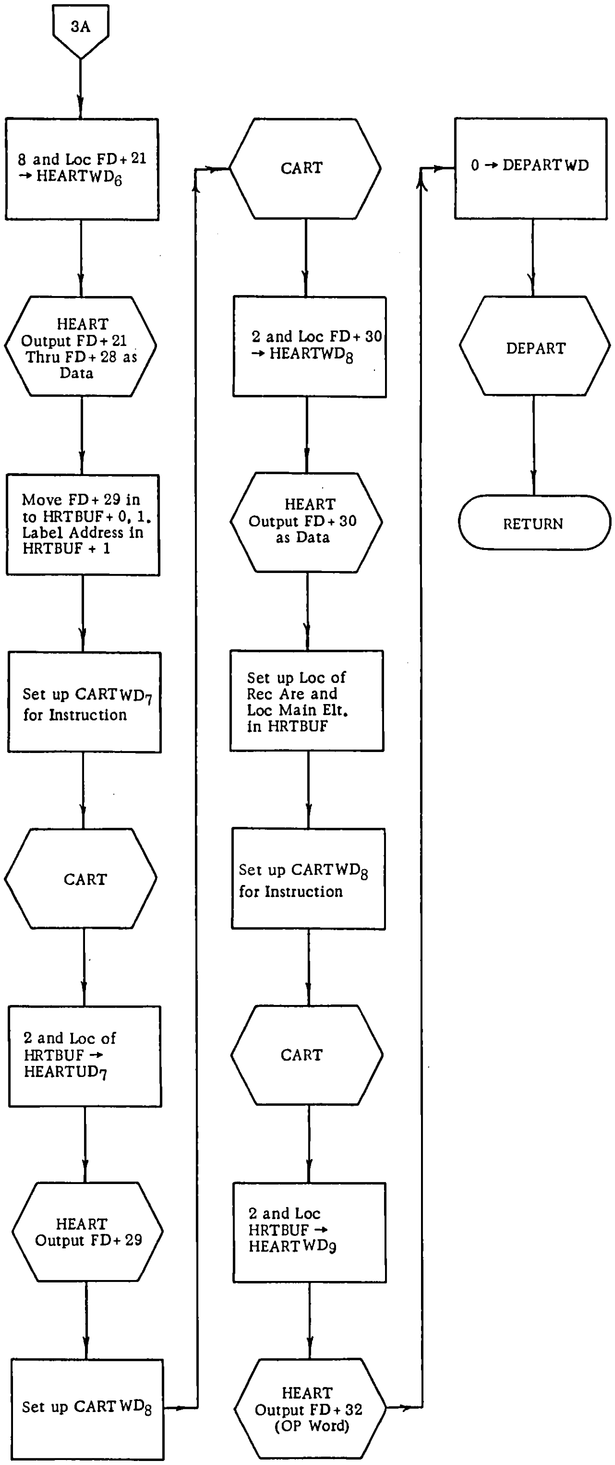


Figure 3-72. OUTFD Flowchart (3 of 3)

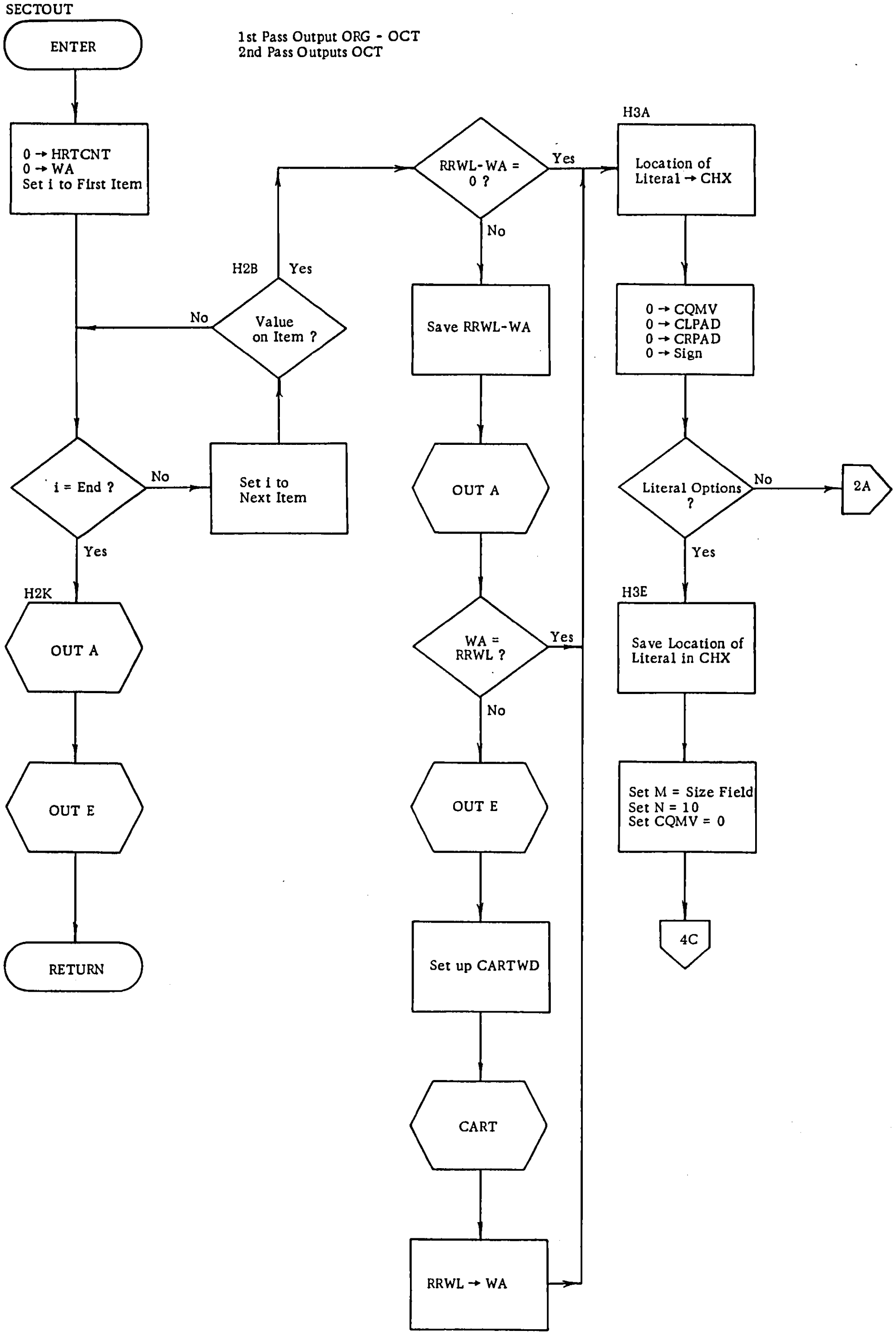


Figure 3-73. SECTOUT Flowchart (1 of 4)

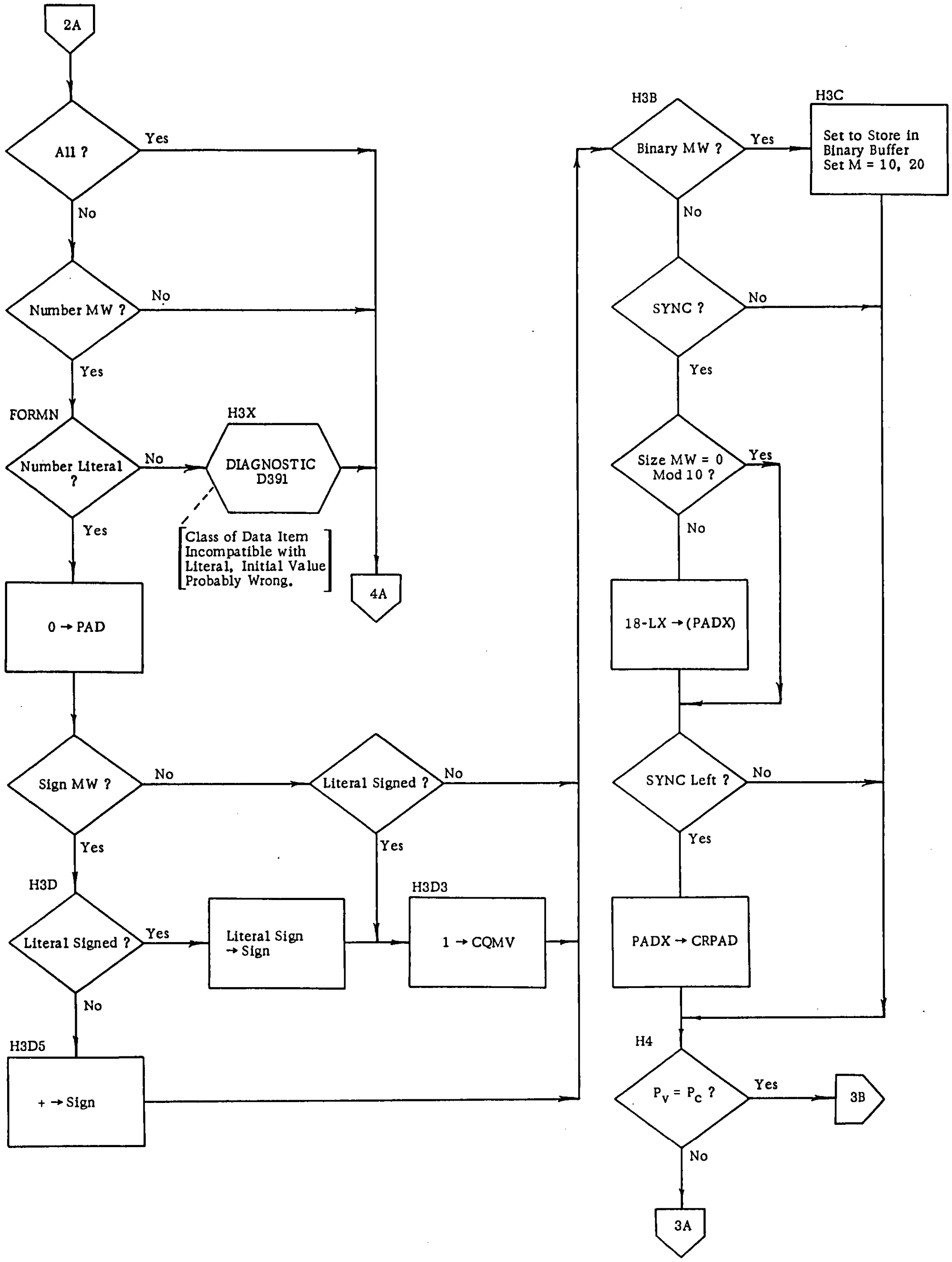


Figure 3-73. SECTOUT Flowchart (2 of 4)

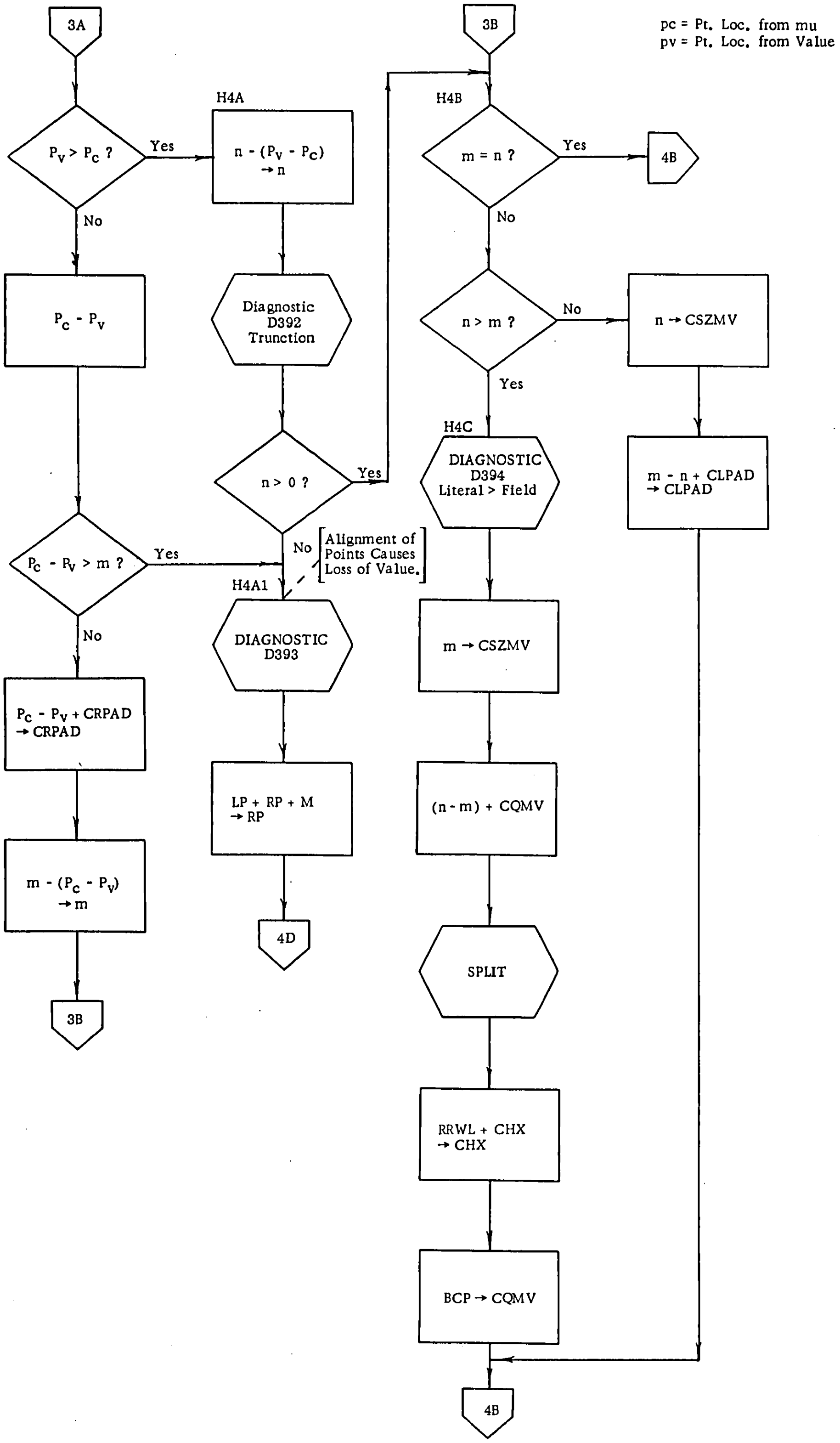


Figure 3-73. SECTOUT Flowchart (3 of 4)

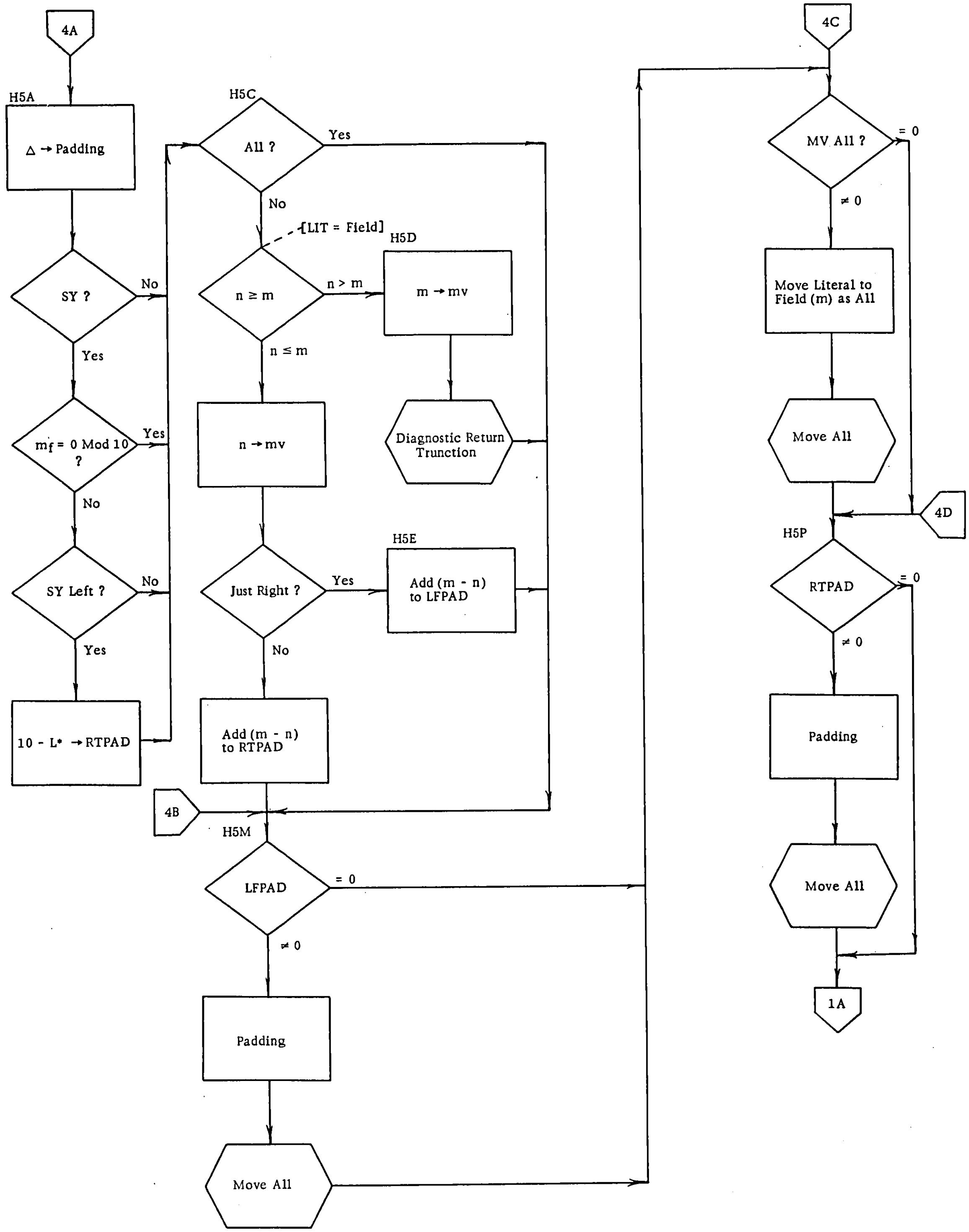


Figure 3-73. SECTOUT Flowchart (4 of 4)

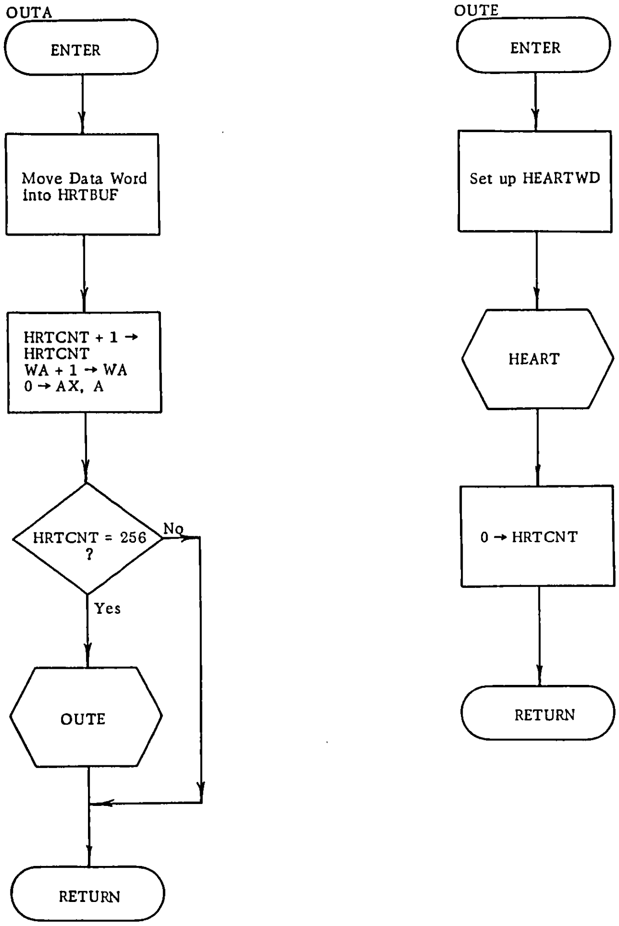


Figure 3-74. OUTA and OUTE Flowcharts

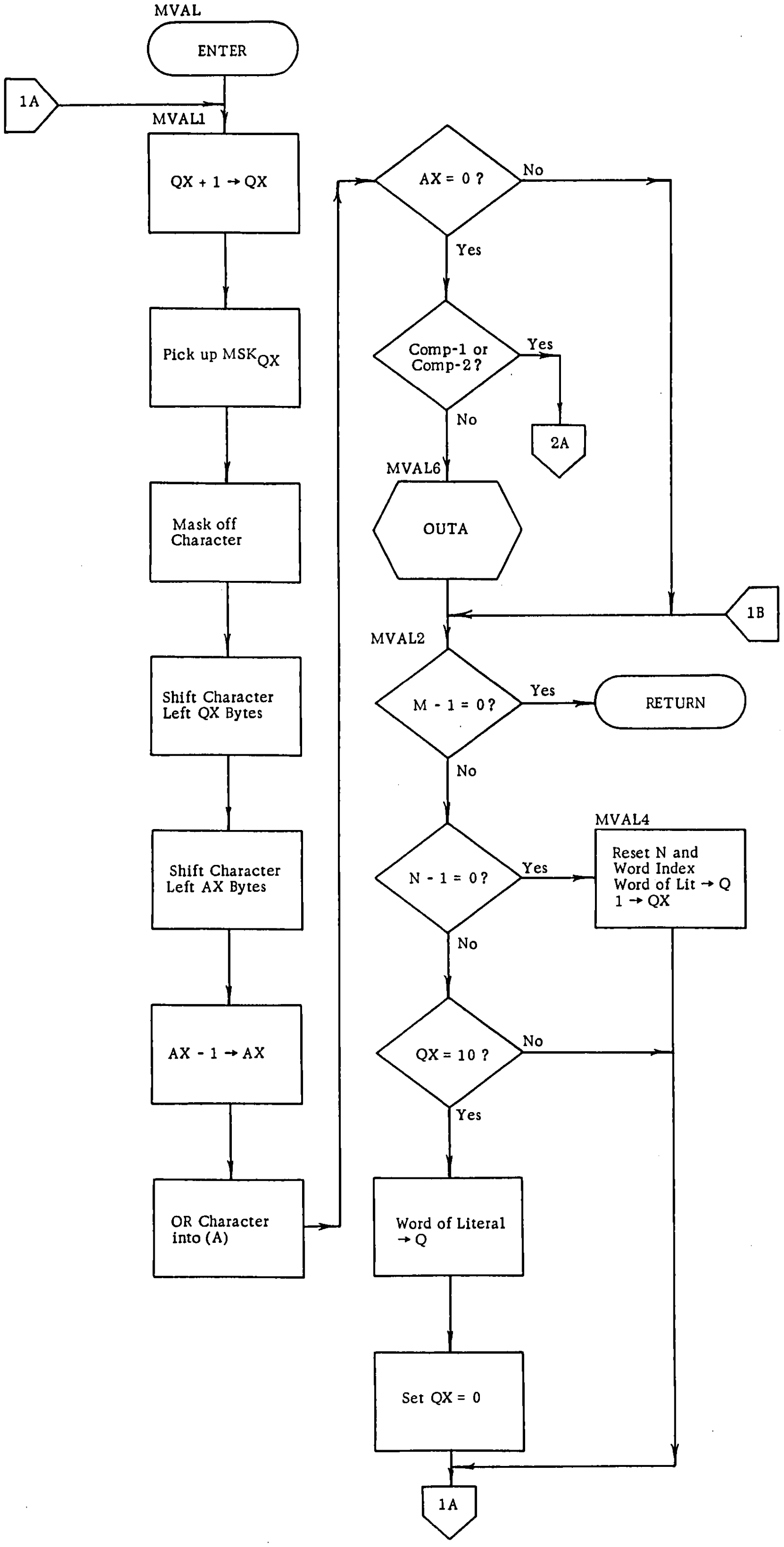


Figure 3-75. MVAL Flowchart (1 of 2)

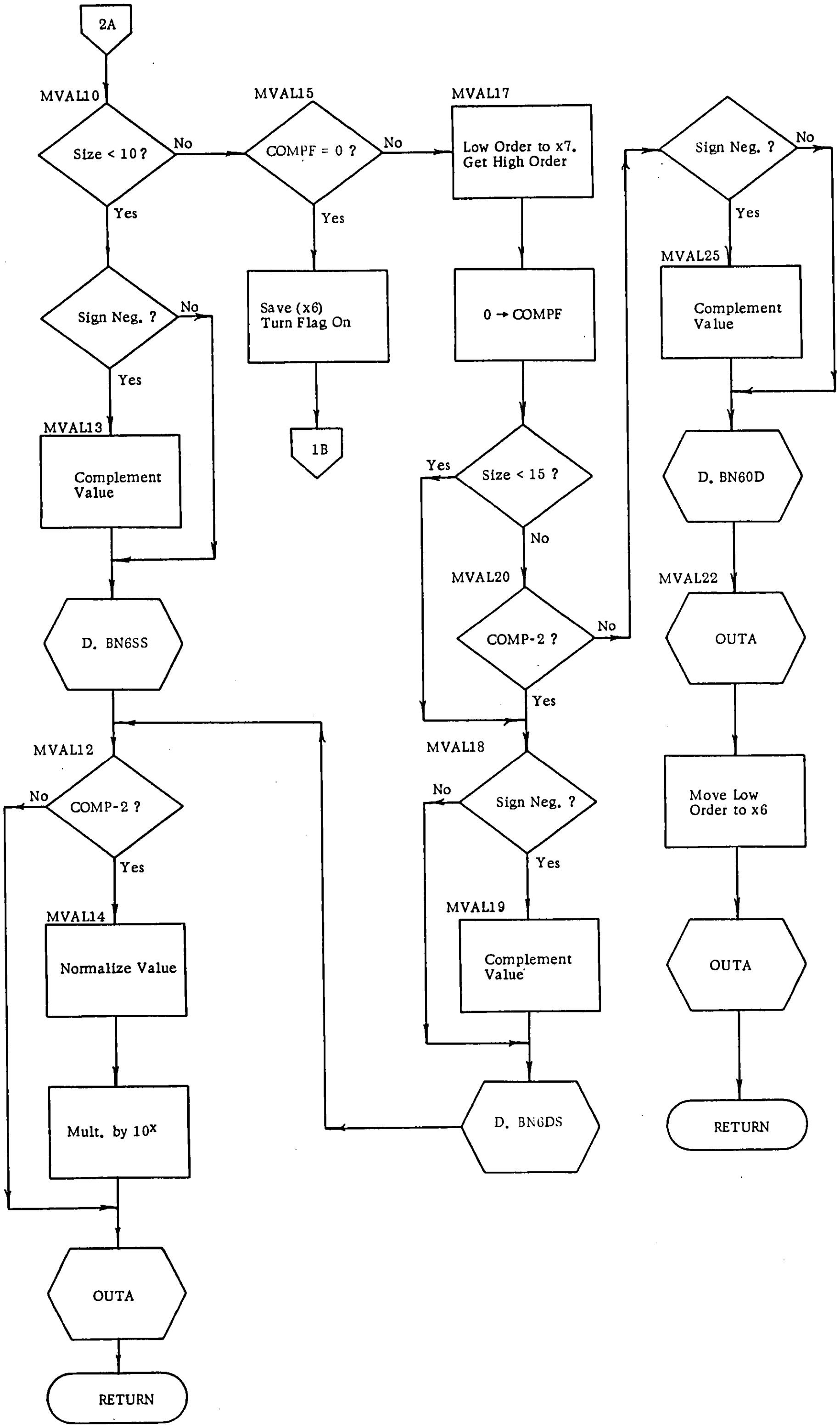


Figure 3-75. MVAL Flowchart (2 of 2)

2. File Table references to items in COMMON-STORAGE such as "LABEL" names (5) and "Key Position (variable record)" require the following parameter values:

   TYPE = 444_8.

   REFERENCE = Location of word containing the word "CCOMMON" (left justified).

   BCP = BCP from referenced data item in COMMON-STORAGE.

   RRWL = RRWL from referenced data item in COMMON-STORAGE.

3. File Table reference to items in WORKING-STORAGE and CONSTANT section such as "LABEL" names (5) and "Key Position" require the following parameter values:

   TYPE = 444_8.

   REFERENCE = Location of word containing the Program ID or CENT.00 if not subcompile (SUBSUB = 0).

   BCP = BCP from referenced data item in WS or CS.

   RRWL = RRWL + Assembler address of WS or CS.

Before outputting anything to the assembler, Pass 1H must determine where the assembler address will be for the start of WORKING-STORAGE and CONSTANT sections relative to the beginning of the main element. The label on the first word of the main element is that given by the first seven characters of the PROGRAM-ID. The label given to FET1 (FD + 12) is the first seven characters of the name given in the "ASSIGNED TO" option of the SELECT clause, the name given in the SPECIAL-NAMES paragraph, or the name OUTPUT as assigned by the Pass 1B. The label for COMMON-STORAGE is "CCOMMON" and for D-COMMON is "DCOMMON." The corresponding CART calls present the same names as the label given in the HEART call.

PASS 2 AND ELEMENTS

Pass 2 is the code generation phase of the compiler (i.e., the second pass over the procedure division source code). It receives as input encoded procedure division source (trees) and it outputs the binary compiled program to the assembler.

Purpose

This routine serves as a master controller for code generation. Generally it climbs the trees, invoking code generators as needed.

Calling Sequence

Pass 2 is called by compiler CONTROL following Pass 1H, and returns to CONTROL upon completion

Routines Called

Pass 2 calls all code generators, which return control to it upon completion

Entry Points

P2 START initially
SCALE08 to continue down the tree
SCALE01 to continue up the tree
SCALE 06 to get the next tree

Communication and Switches

Pass 2 Communication and Switches are set and used as follows:

HOLDSW (P2)	NZ	HOLD node
	Z	OSE fork, SCALLOC, when result is stored in TEMP
	Use	1) If on, GENSTO, GENLOD save in X3/5 or TEMP the contents of X6, 7 2) If off at GRAB time, result is in TEMP, if on in X3/5
GRABSW (P2)	NZ	GRAB node
	Z	SCALLOC, when interrogated
	Use	Retrieve saved result from X3, 5 or TEMP
OSESW (P2)	NZ	430-434 Master node
	Z	OSE fork node, SCALLOC
	Use	1) In GENSTO, to generate on-size coding if SR< SC 2) Check stores climbing tree, for multiple receiving fields

OSETRTH (P2)	NZ	More than one store
	Z	SCALLOC
		Used to set up and interrogate object-time truth switch
OSEIST (P2)	NZ	2nd store in GENSTO, climb
	Z	At use, SCALLOC
		Used first-time switch
		1) Initialize truth switch
		2) Identify OSETRTH
MOVESW (P2)	NZ	MOVE master node
	Z	SCALLOC
		Used to cause GENLOD to accept alpha, on S >18
FLUSHSW (P2)	NZ	Items entered into ASSMALT (STIT999)
	Z	SCALLOC
		Used to dump ASSMALT at SCALLOC
LABDFSW (P2)	NZ	Item entered into ASSMALT (OSE, M12F)
	Z	At use, SCALLOC
		Used in GENLOD, GENSTO to define local label (LABARG) for return from OSE truth switch instructions
LABZRDF (P2)	NZ	Local labels have been defined
	Z	SCALLOC
		Used to cause definition of LOCAL (O) and resultant redefinition of LOCALS
ZEROFLG (P2)	NZ	At object time X4 = (33---3)
	Z	X4 has been destroyed (I" DPC store, blanks loaded, etc.)
		Used to restore X4 at GENLOD, etc.
BLNKFLG (P2)	NZ	SCALLOC, when X4 loaded with zeros, etc.
	Z	Short move, JUSTIFIED RIGHT, alpha
		Used to load blanks (55---5) in X4
SIXFLG (P2)	NZ	when 6060--is in X5
	Z	when X5 used in other ways (e.g., d.p. HOLD)
		Used to prevent reloading, at object time of 6060--
TEMPSW (P2)	NZ	COMPUTE, left operand loaded from PEMPC
	Z	SCALLOC, on use
		Used to prevent LOAD in GENARTH

NOLLAB	Initiallized to 1 at SCALLOC Contains the next unused local label number (for ART, this must be multiplied by 2)
BASELBL	The local label number first assigned in a series of OSE tests
LABARG	The local label (and pseudo-instruction) assigned to OSE code with only one store
LCLOSE	Local label assigned to OSE code
COMPSZE	NZ when situation (below) occurs Z SCALLOC Used first-time switch for SIZE GREATER than 18 message
CUMULAT	Count of successive adds, subtracts. Used to increase size for carry Z SCALLOC, MST
BNDCFLG	NZ when D. BNDCx is called at object time Z SCALLOC Used to interrogate B7 for OSE
TEMPCTR	Tally for assigned temporaries, COMPUTE
LASTMX (X3)	contains the last MXO instruction issued Altered when XO is destroyed in object code, or flow
LODSW (P2)	NZ MST Z Arithmetic operator, MOVE, IF, PERFORM Used to suppress loading in GENLOD LFTLEAF is returned
LFTLEAF	The left link when GENLOD is suppressed
SHRTLOD	NZ A load meets the restrictions in MOVE ZR MSTR Used to cause abbreviated load
STORCOR	NZ CNV subroutines, etc. Z MSTR Used to force TEMP (HOLD) to be core

FLOWSW	NZ	Branches
	Z	SCALLOC
	Used to prevent a load of X4, say, in OSE code to hold for main-line flow	
RTLODSW	NZ	Right conditional is an expression
	ZR	MST
	Used by IFGEN	
RTTRUNC	NZ	Right truncation in MOVE
	Z	MST
	Used to get SF from SC rather than EDD2	
RTBLNKS	NZ	Alpha move ST SF
	Z	MST
	Used to cause right padding of blanks	

Operation

Any starting point in the climb of a tree is a master node. At this point master initialization is accomplished. If the master is also an operator, control passes to the appropriate generator. Switches are set, and the tree is climbed up left links till a leaf is reached. As nodes are traversed, some switches are set or generators are invoked. A trail is retained in the left link position (complement) so that the climb can be retraced. Upon reaching a leaf on the left path, in arithmetic, the item found is loaded. When the right link is a leaf also, the operator is accomplished and descent is commenced. If a node on the right, the right branch is sealed. The end of sentence leaf brings in the next tree.

Pass 2 subelements are explained in the following sections.

INIT

Purpose

To initialize Pass 2 and the assembler. (See Figure 3-76.)

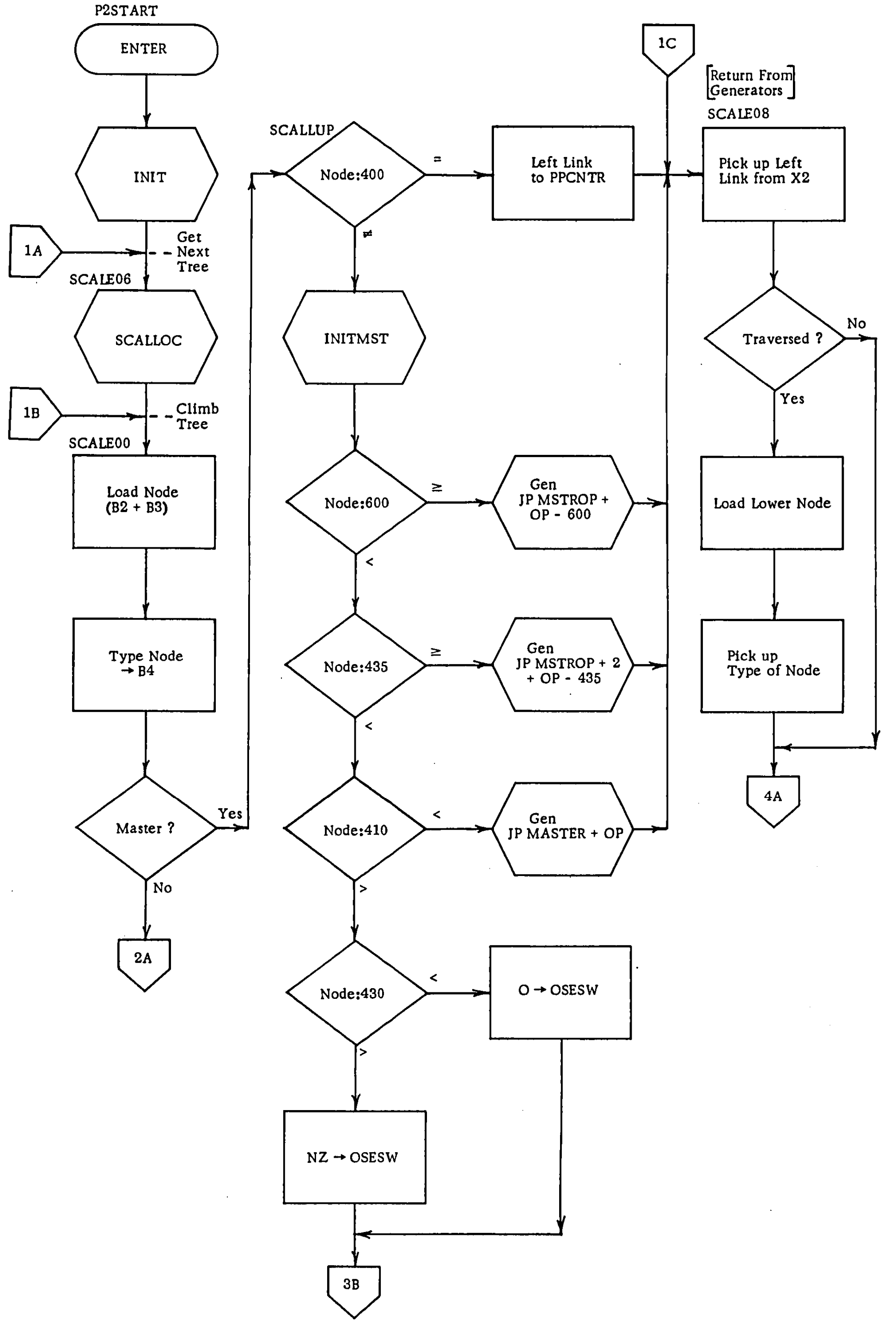


Figure 3-76. P2START Flowchart (1 of 5)

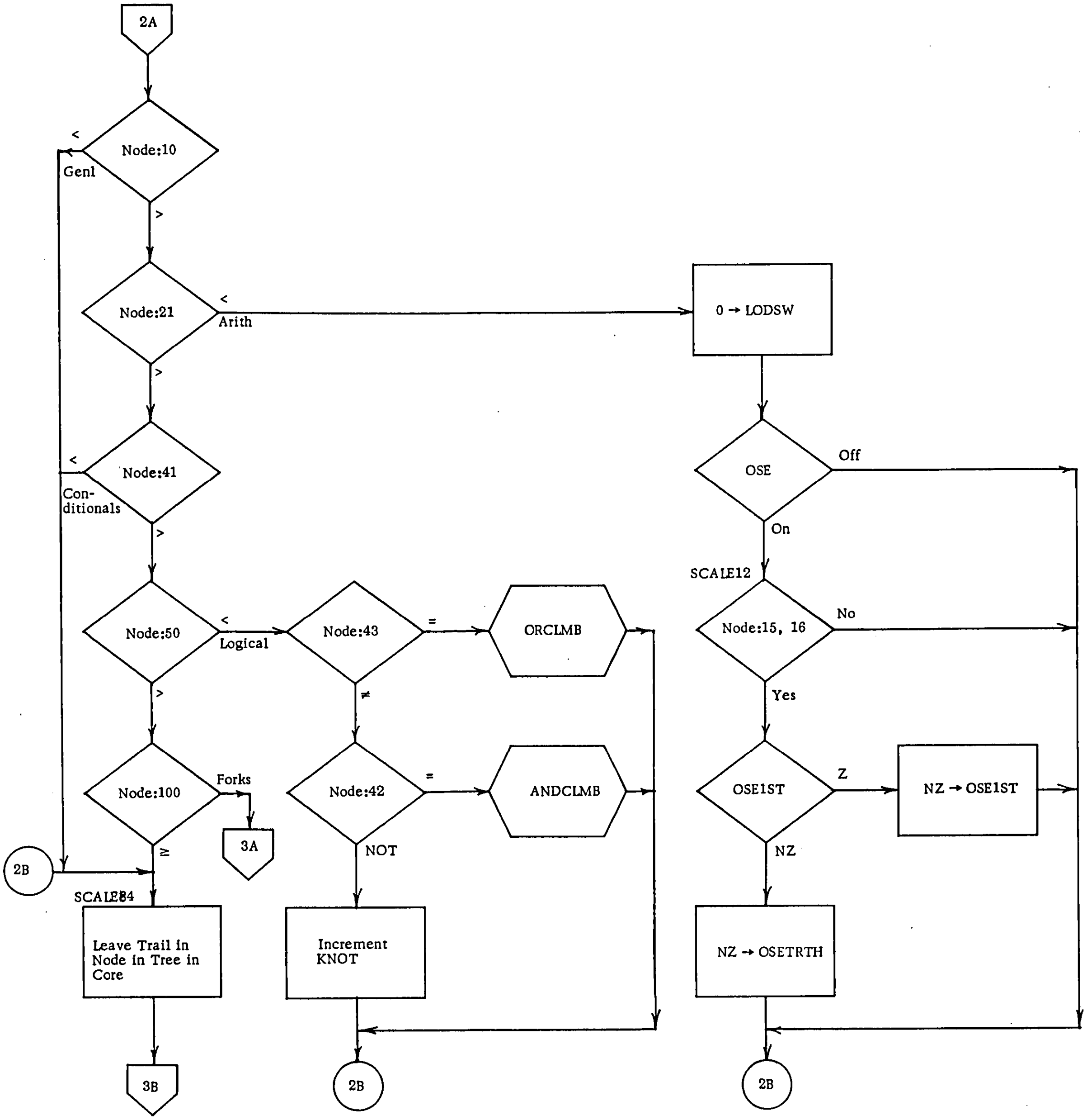


Figure 3-76. P2START Flowchart (2 of 5)

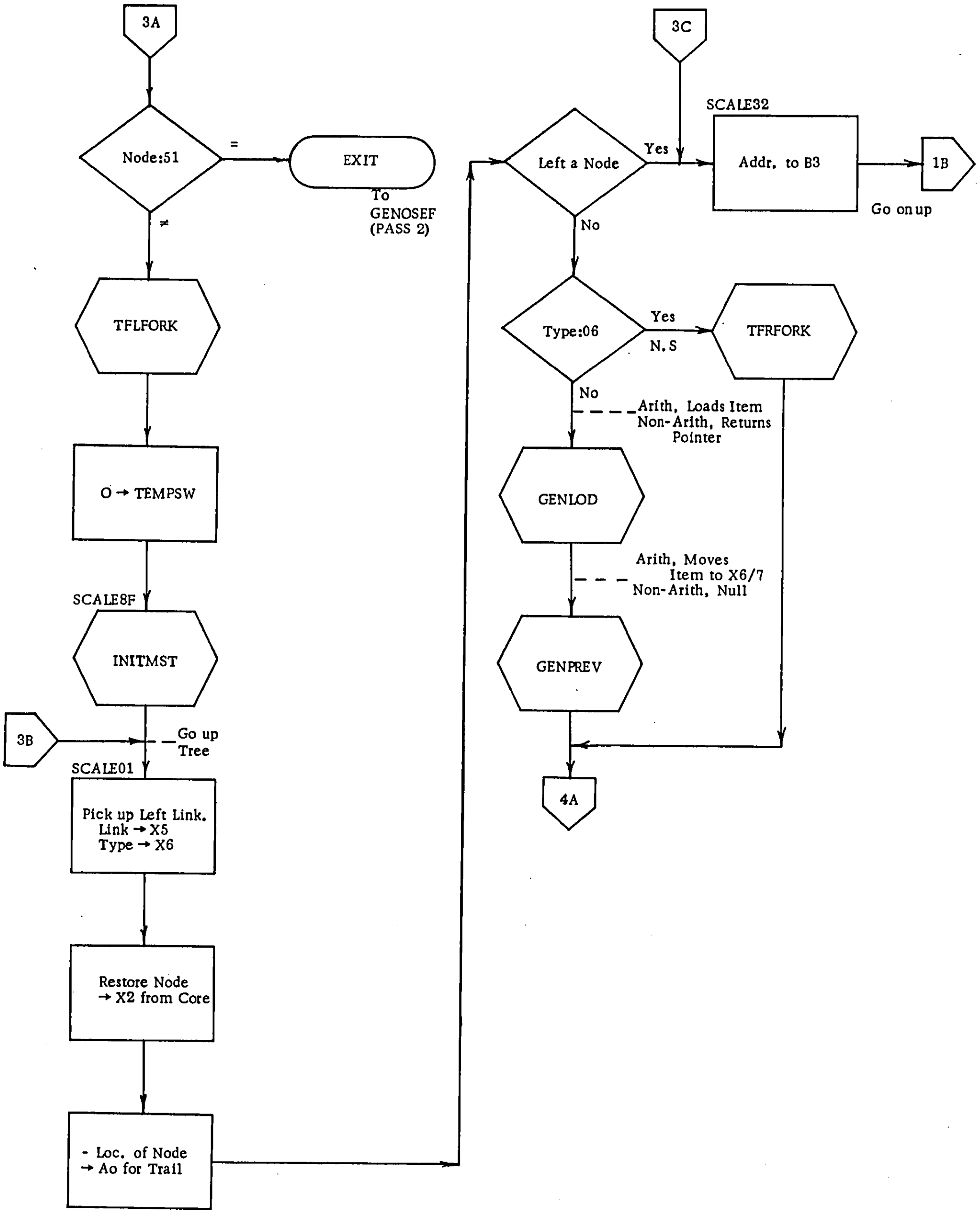


Figure 3-76. P2START Flowchart (3 of 5)

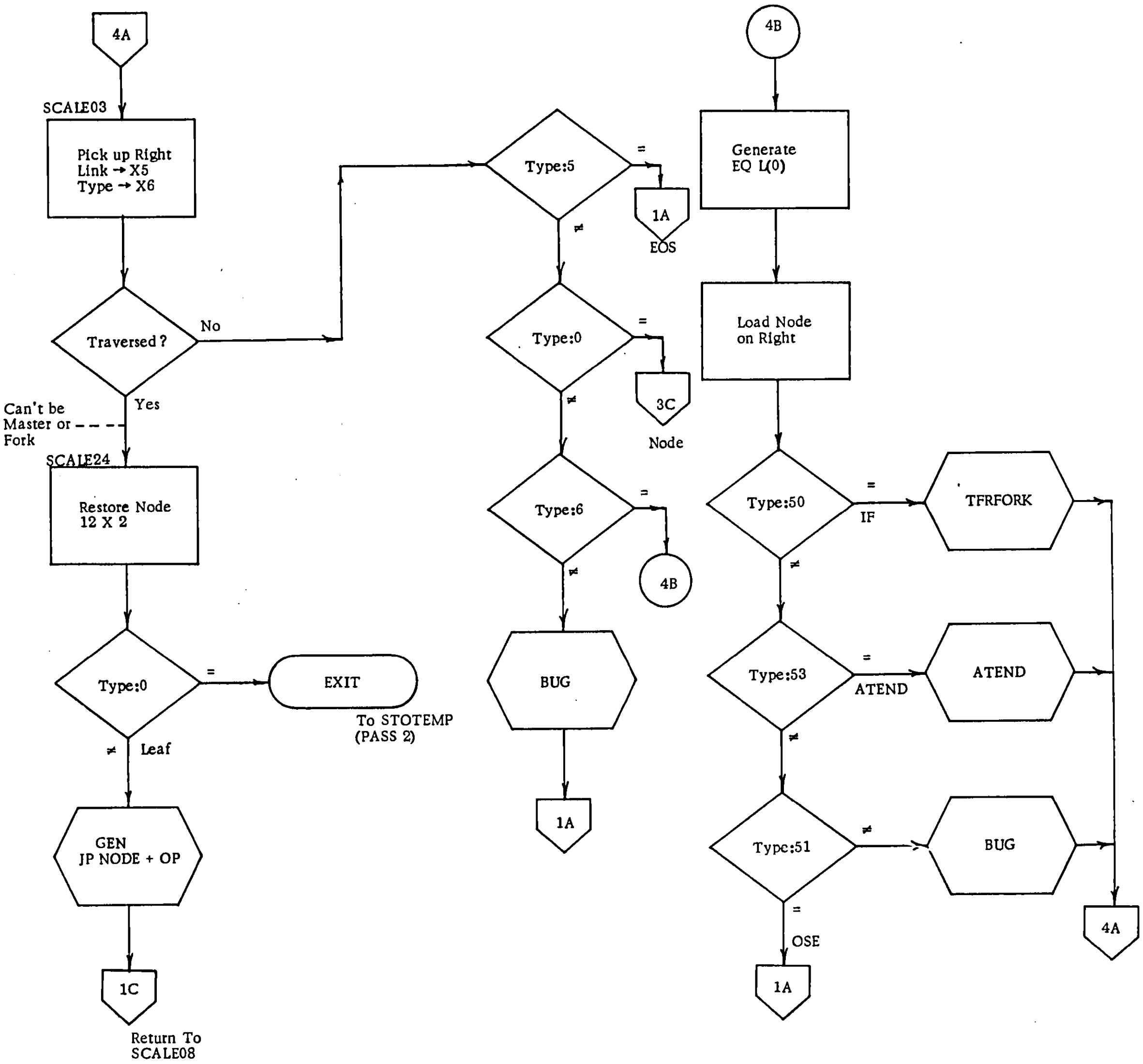


Figure 3-76. P2START Flowchart (4 of 5)

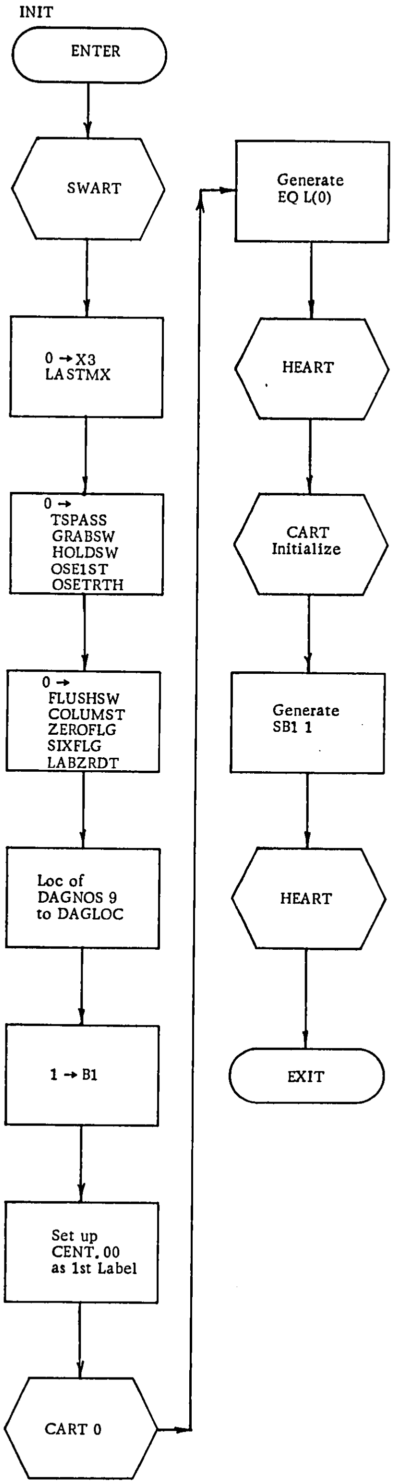


Figure 3-76. P2START Flowchart (5 of 5)

EDINIT

Purpose

To generate a call to DDEDITI if required by entries in the SPECIAL-NAMES section of the ENVIRONMENT DIVISION.

SCALLOC

Purpose

To bring in and locate the next tree (i.e., sentence).

Calling Sequence

RJ SCALLOC

Routines Called

CART
TREEIN

Operation

ASMALT, the alternate assembly buffer, is flashed if the preceding sentence contained ON-SIZE ERROR with multiple receiving fields. LOCAL label zero is defined if needed, then both statement and sentence initialization are accomplished. The next tree is then brought in.

LOC 20 is invoked at the end of an overlay to generate drop-through coding.

GENMOLD sets HOLDSW on a HOLD node or, if the leaf is a literal, generates the literal.

GENGRAB sets GRABSW on a GRAB node.

GENOSEF is invoked by an OSE for a while climbing up the tree (i.e., prior to generating OSE statements). If there are not multiple receiving fields, a jump around the OSE statements is generated, and the OSE label is defined.

In the multiple receiving field case, a local label is defined if necessary. The test of truth is generated, and then a local label for the OSE code is defined if needed.

STOTEMP

Purpose

To generate the store or load from temporary cells, where intermediate results are required by the computer verb.

Called

When a right link is a node.

Operation

On entry, if the retrieve switch (in the node in core) is off, it is set. The node is examined, and if it is AND or OR, calls to ANDDOWN or ORDOWN in GENIF are made. If the expression has not yet been loaded, the load is performed. Code is then generated to store the intermediate result in temporary core. Climb of the tree is resumed.

The second time a node with a node on the right is reached (i.e., retrieve switch is on, and the node is not AND or OR), code is generated to load the item from core. The appropriate generator is then invoked.

MOVES

1. Moving information from one place to another, with possible transformations, is the heart of the problem of causing a computer like the CDC 6400/6600 to process effectively data processing problems (i.e., the problem of COBOL). We will consider first the numeric case though short alpha moves use the same routines.

   The basic scheme is to load from core a numeric item into X1 or, if necessary, X1 and X2. In arithmetic it may then be combined with a current result in X6, or X6 and X7, to form a new current result in X6 or X6/7. Or it may be merely transferred to X6/7. At some point in the statement, the result will be stored in core from X6 or X6/7. X0 is used largely as a mask register. X4 normally contains the frequently used DPC zero. X3 and X5 are, where possible, used to hold intermediate results for successive stores, or they may be used to reload intermediate results that have been stored in core. Note that the above is quite simplified, that many exceptions occur (e.g., where move registers are needed for an operation such as double precision add) and many local optimizations are made. It is the basic scheme of register assignment.

2. In memory (i.e., core), numeric items are represented as follows:

   a. Numeric (COMPUTATIONAL or DISPLAY). BCD.

      (1) Items are in CDC "display" (as defined on p. B-1, ERS) code, without regarding word orientation. Numbers are 33-44.

      (2) If the item is signed, the right position of the item contains the sign as an underpunch (i.e., the display code representation of the character that is the number with a 12 or 11 zone punch in card code).

      Thus, +1 = A = 01, -3 = L = 14, +0 = 72, -0 = 66.

   b. COMPUTATION-1. Binary representation of the decimal item.

      (1) SIZE - 14 or less.

         (a) Item is one computer word. The number is a binary integer of 48 bits, or less. It is un-normalized, with a zero exponent, but formally a floating-point number, i.e., biased by 2000_8.

         (b) Sign is indicated by one's-complement notation.

         (c) Decimal point location is specified by the user and is known to the compiler. It is not explicitly carried in the data.

(2) SIZE - greater than 14 (but by DOD definition less than or equal to 18).

(a) Item is two computer words. The integer item is stored as a 6600 double-precision word pair. It is normalized and biased by 2000_8. The low-order word has an exponent 48 less than the high-order word.

(b) Sign is carried by one's complement.

(c) Decimal point is known to the compiler (same as for 14 or less).

c. COMPUTATIONAL-2. FORTRAN floating point.

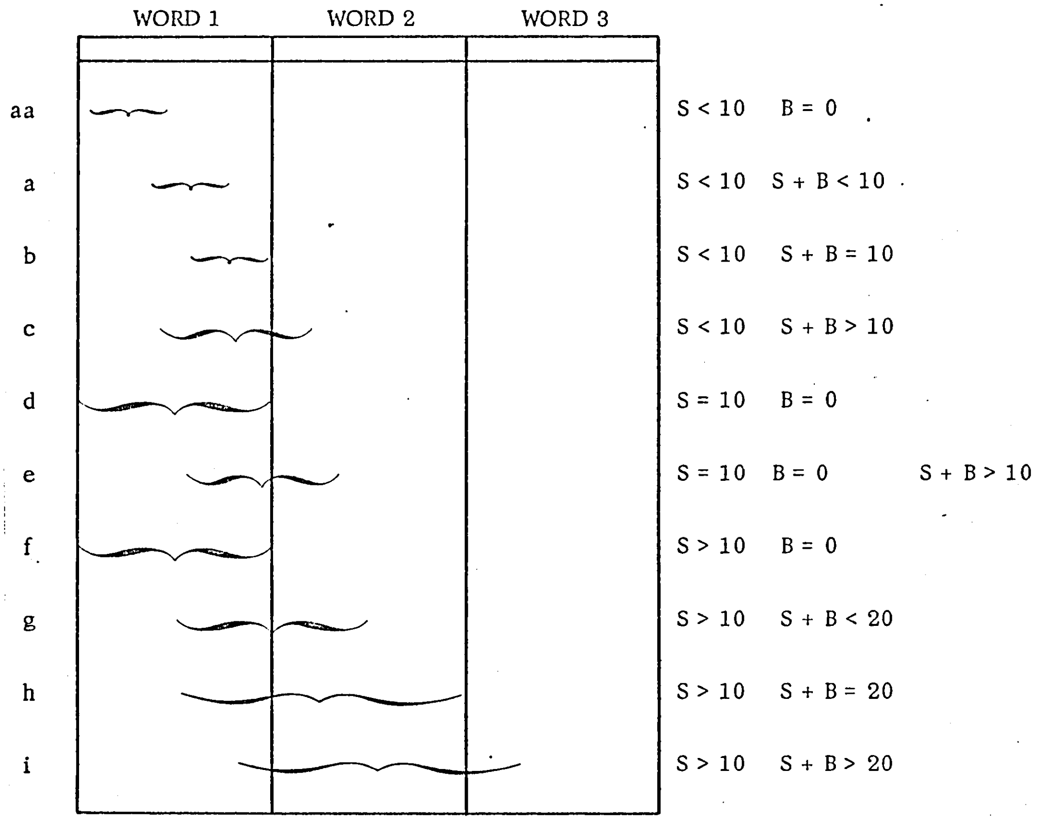
(1) The item is one computer word. No double precision is provided. The word is normalized floating point. Sign is one's complement.

(2) The properties of the item are carried internal to it. Thus no PICTURE or equivalent specification statements apply.

3. Internally binary, (i.e., COMPUTATIONAL-1 or COMPUTATIONAL-2) items are held in registers as they exist in CORE. Decimal (DPC) numbers are right adjusted with leading DPC zeros inserted if needed (e.g., when the item is to be used for arithmetic.) Negative numbers are converted to one's complement, which is also nine's complement using the DPC character set. Thus the leading byte in a register must be reserved as a sign indication. Single precision is 9 decimal digits and double precision 19.

4. In fetching and storing short DPC items, memory must be considered to be character oriented. Thus, on a word machine, shifting and masking are required

to accomplish the fetch of an item and to preserve the background into which an item must be inserted. On the 6400 the following word orientations are recognized:



where

S = Size
B = Beginning character position

The properties of the loaded item are placed in the load PROPERTIES area.

Associated in the compiler, with items in X1, X1/2, X6/7, X3/5 and temporary storage, is a property word. The following is a description of an OPERAND DESCRIPTOR (property word).

Module		0 0	Decimal Point Location	Size in Characters
59 54	52	36-35	18-17	0
6	1	17	18	18

Mode (octal)

00 = BCD (CDC display character set)
17 = Non-numeric
40 = COMPUTATIONAL-1 (binary = un-normalized floating point) single or double precision
60 = COMPUTATIONAL-2 (normalized floating point, single precision)
27 = justified

Signed (binary)

0 = not signed
1 = signed

Decimal Point Location (binary)

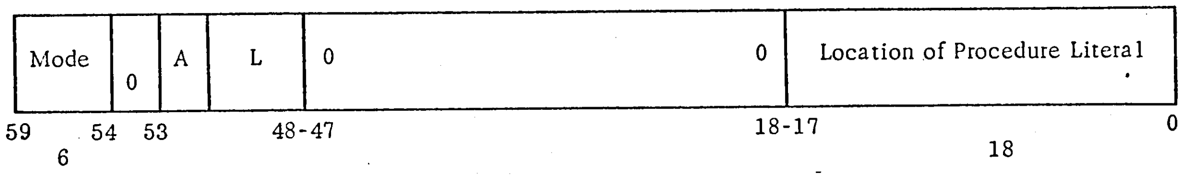
Binary number indicating the power (X) of $\left(\frac{1}{10}\right)^*$ by which the integral operand must be multiplied to produce the appropriate value. A "point right" will appear as a complemented 18-bit number.

Size

Indicates length of operand field in characters.
If mode is COMP-1 and size is >14, the operand is double precision.
If mode is BCD and size is >9, the operand is double precision.
If mode is COMP-2, the operand is single precision.

Literal descriptor (properties word)

The following is a description of the literal descriptor:



Mode (octal)

- 01 = Literal properties word
- 02 = Figurative constants. (Location at literal = 0)
  - A = 1 ALL any non-numeric literal
  - L (Hex)
    - 1 = Zero(s)
    - 2 = Space(s) or low value(s)
    - 4 = Record mark (62_8)
    - 7 = High value(s)

GENLOD

Purpose

GENLOD is invoked to fetch from core any NUMERIC item and short ALPHABETIC or AN items. The code generated loads the item into X1 or X1 and X2, inserts leading zeros, or leading or trailing blanks as needed, strips the sign underpunch and complements the item if negative. (See Figure 3-77.)

Call

RJ	GENLOD with leaf type in X6 leaf in X5 tree pointer in B2

Called by

Pass 2 when a leaf is the left link on climb, and LODSW on

GENARTH
PRFOPS
GENMOVE
GENIF
GENDISP
SUBSCR

Routines Called

CART, HEART
SUBSCRL

Operation

If LODSW is non-zero (i.e., a generator needs to call GENLOD itself rather than allowing the tree climb to do it, the leaf link is saved and GENLOD exits. Registers are saved, governed by a possible one-level recursion where GENLOD calls SUBSCRL, which then calls GENLOD to lead a variable subscript.

When the leaf that GENLOD is asked to operate on is DNT (i.e., this is a fetch from memory), the DNT entry is decomposed. Code is then generated to load the item.

If the switch SHRTLOD is on (set in GENMOVE if there are no conversions in a MOVE, and the size of the receiving field is not greater than the source field), abbreviated code is

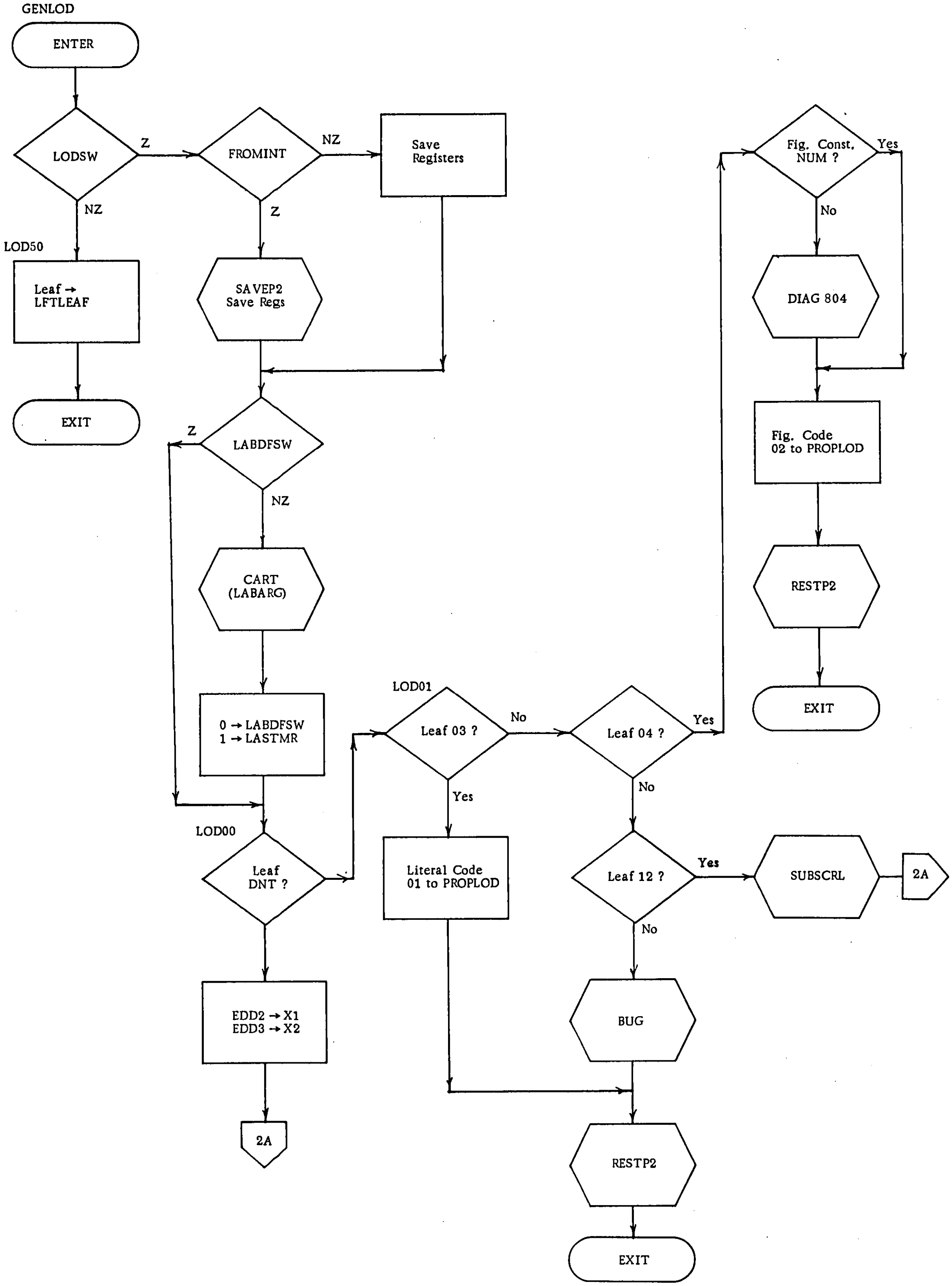


Figure 3-77. GENLOD Flowchart (1 of 8)

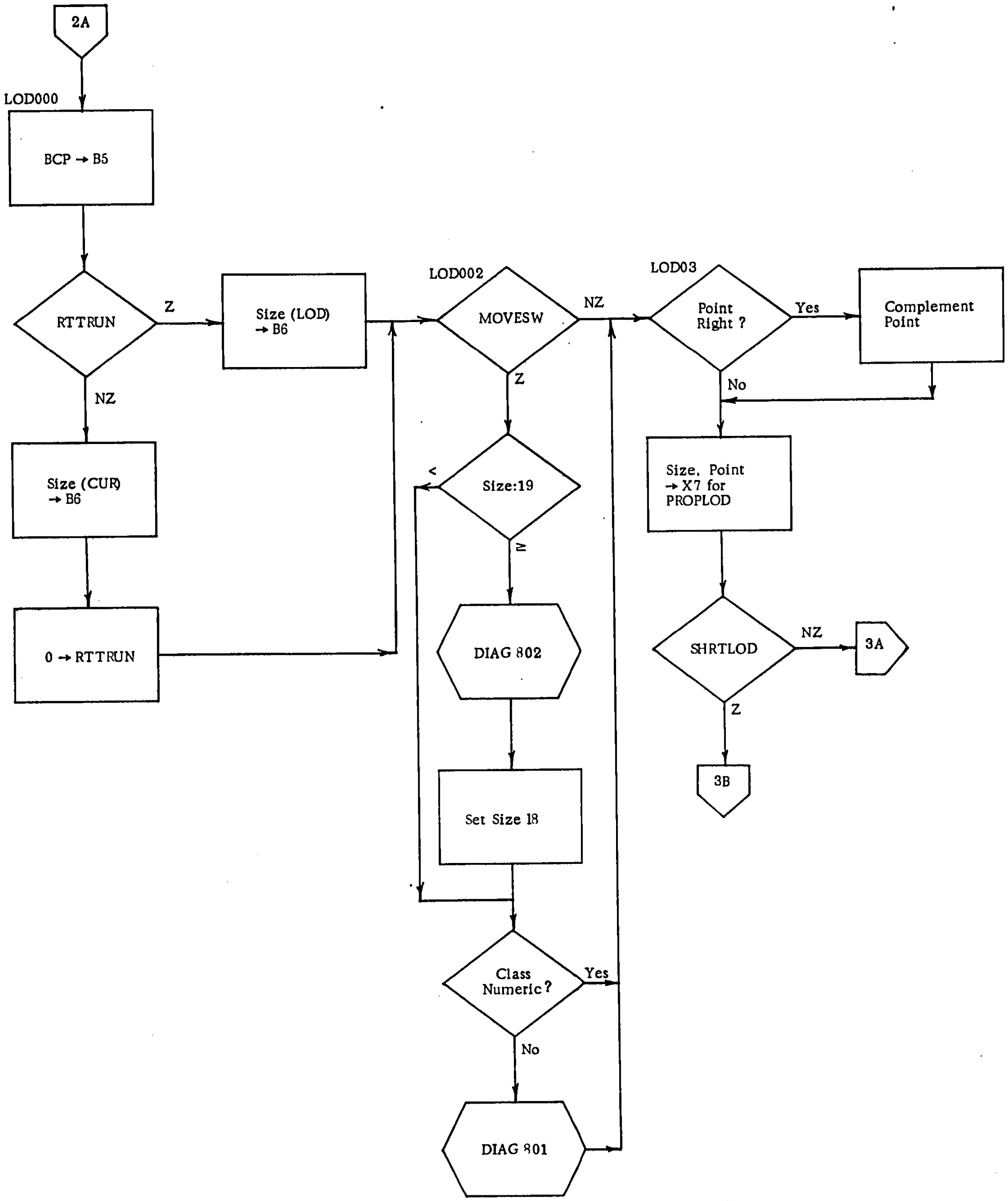


Figure 3-77. GENLOD Flowchart (2 of 8)

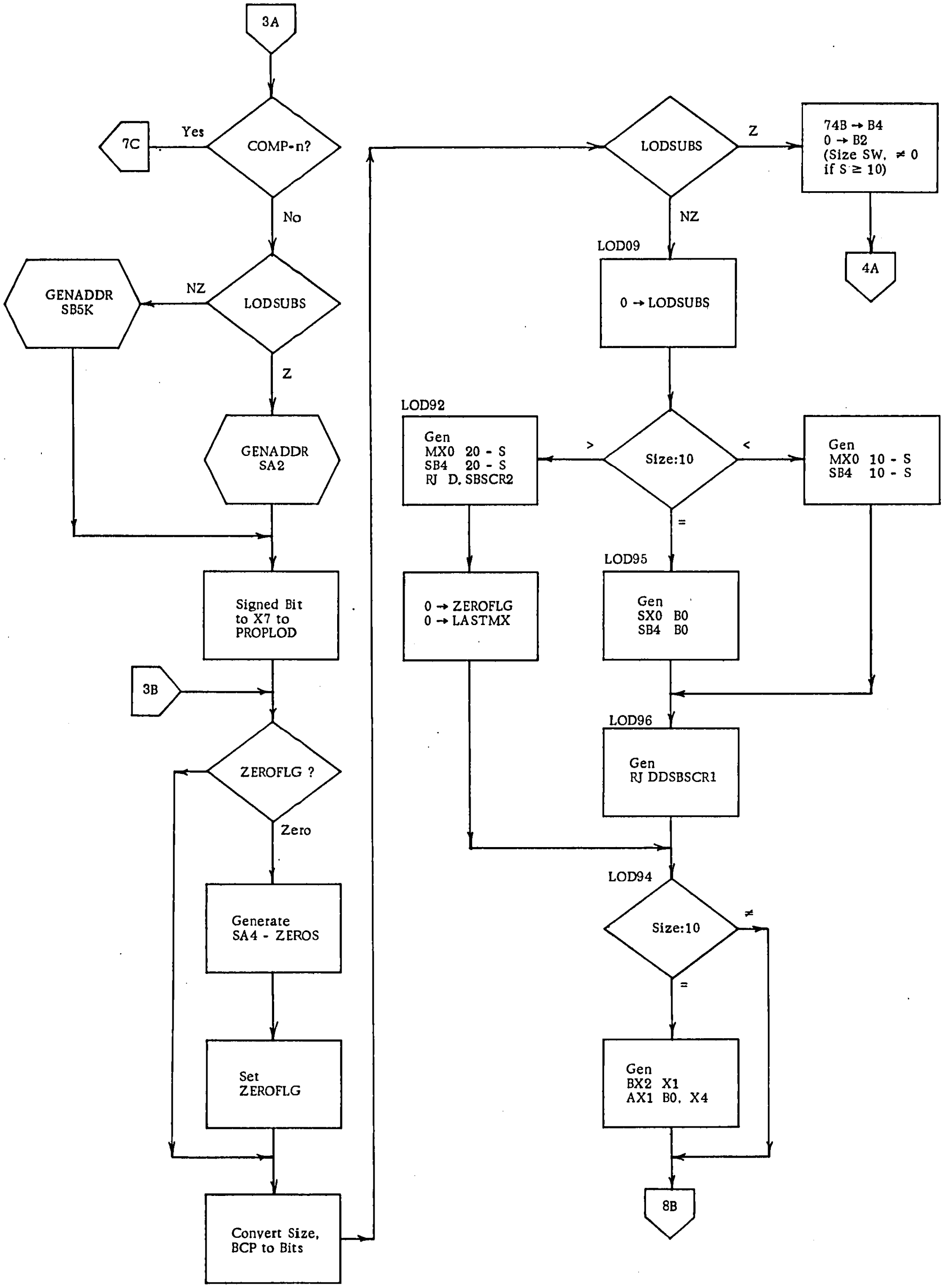


Figure 3-77. GENLOD Flowchart (3 of 8)

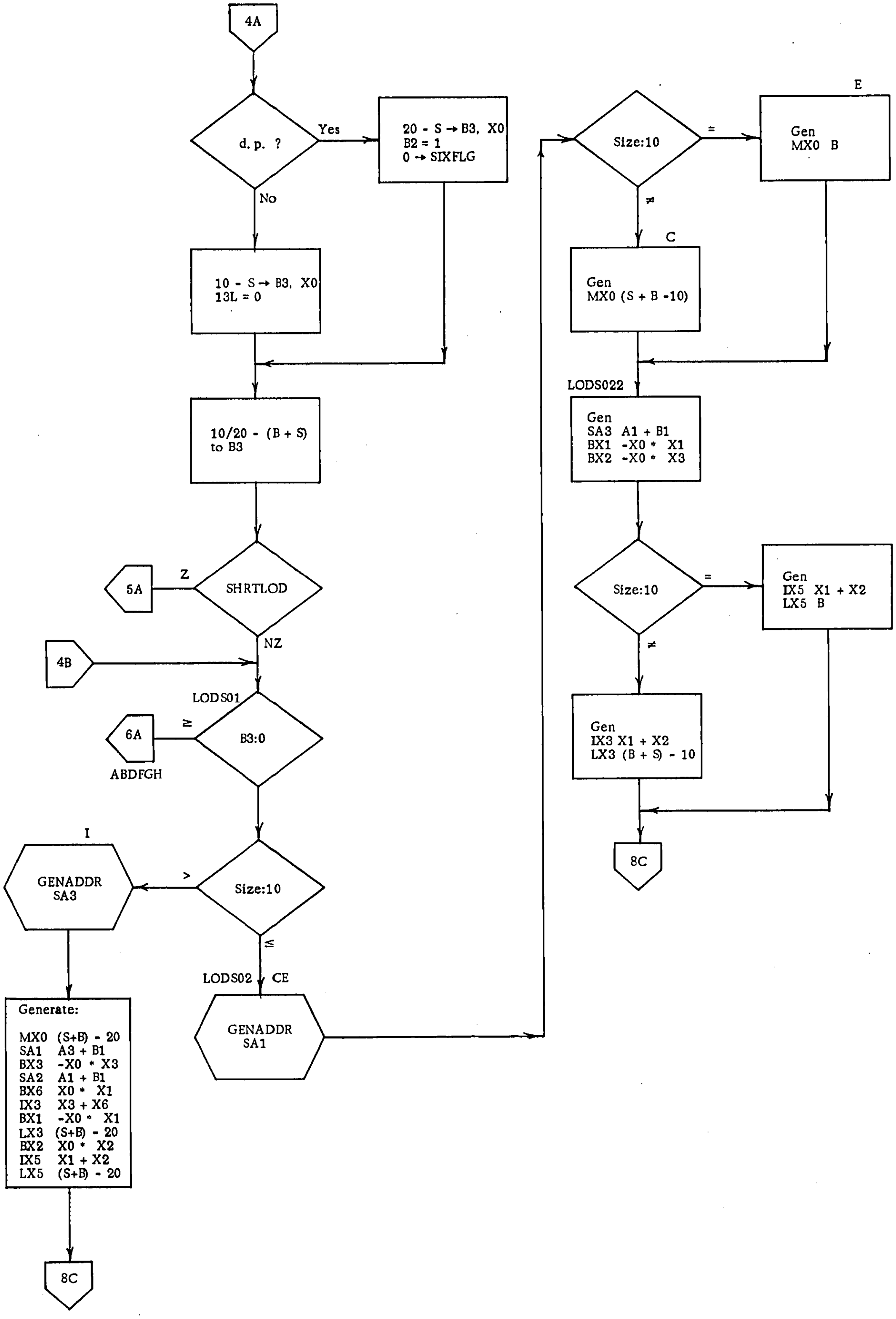


Figure 3-77. GENLOD Flowchart (4 of 8)

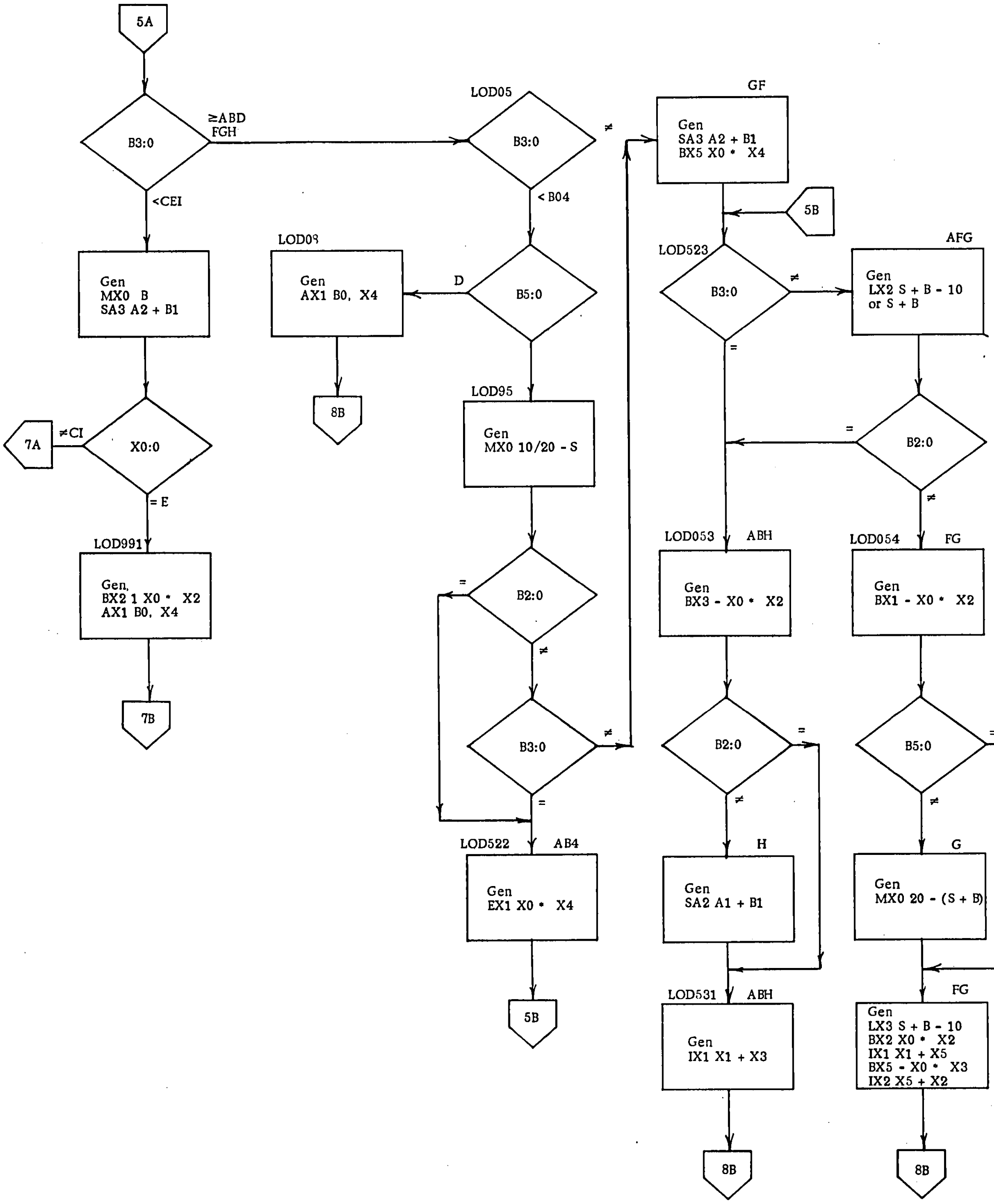


Figure 3-77. GENLOD Flowchart (5 of 8)

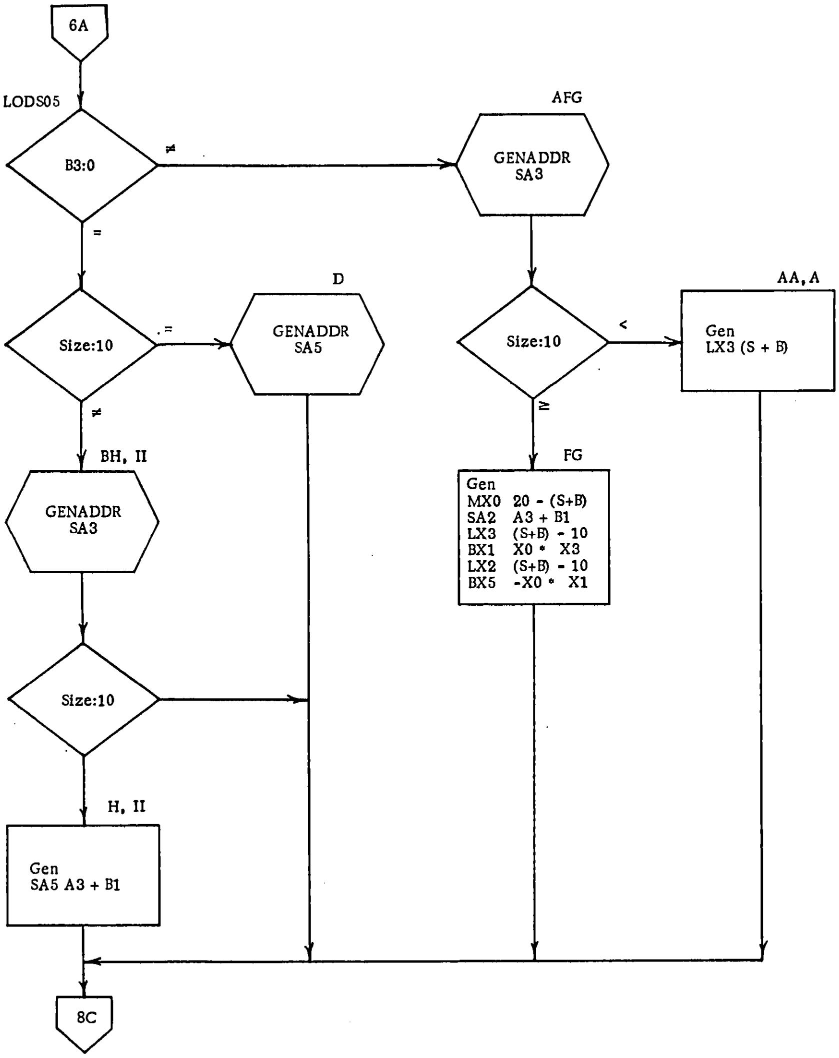


Figure 3-77. GENLOD Flowchart (6 of 8)

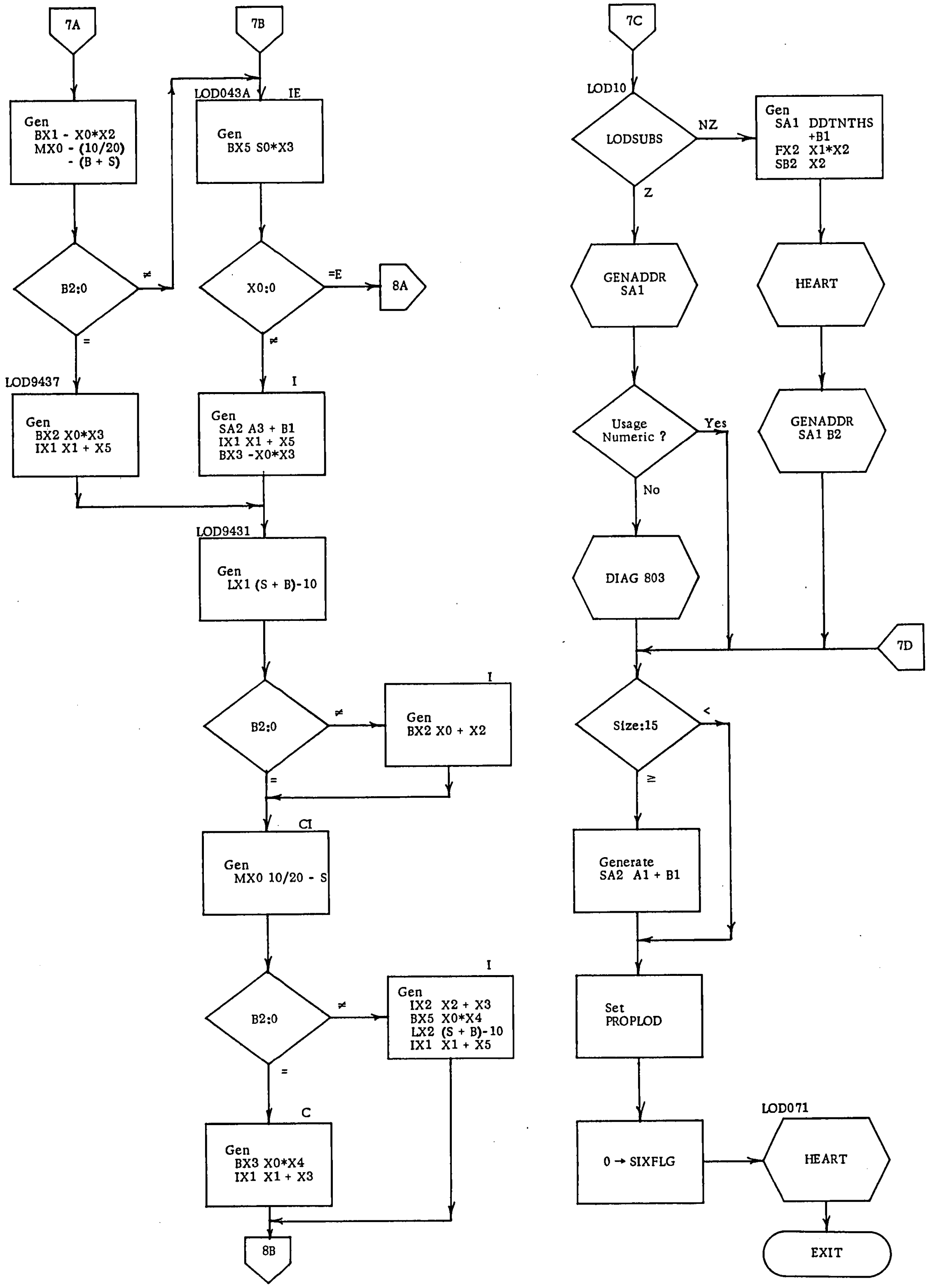


Figure 3-77. GENLOD Flowchart (7 of 8)

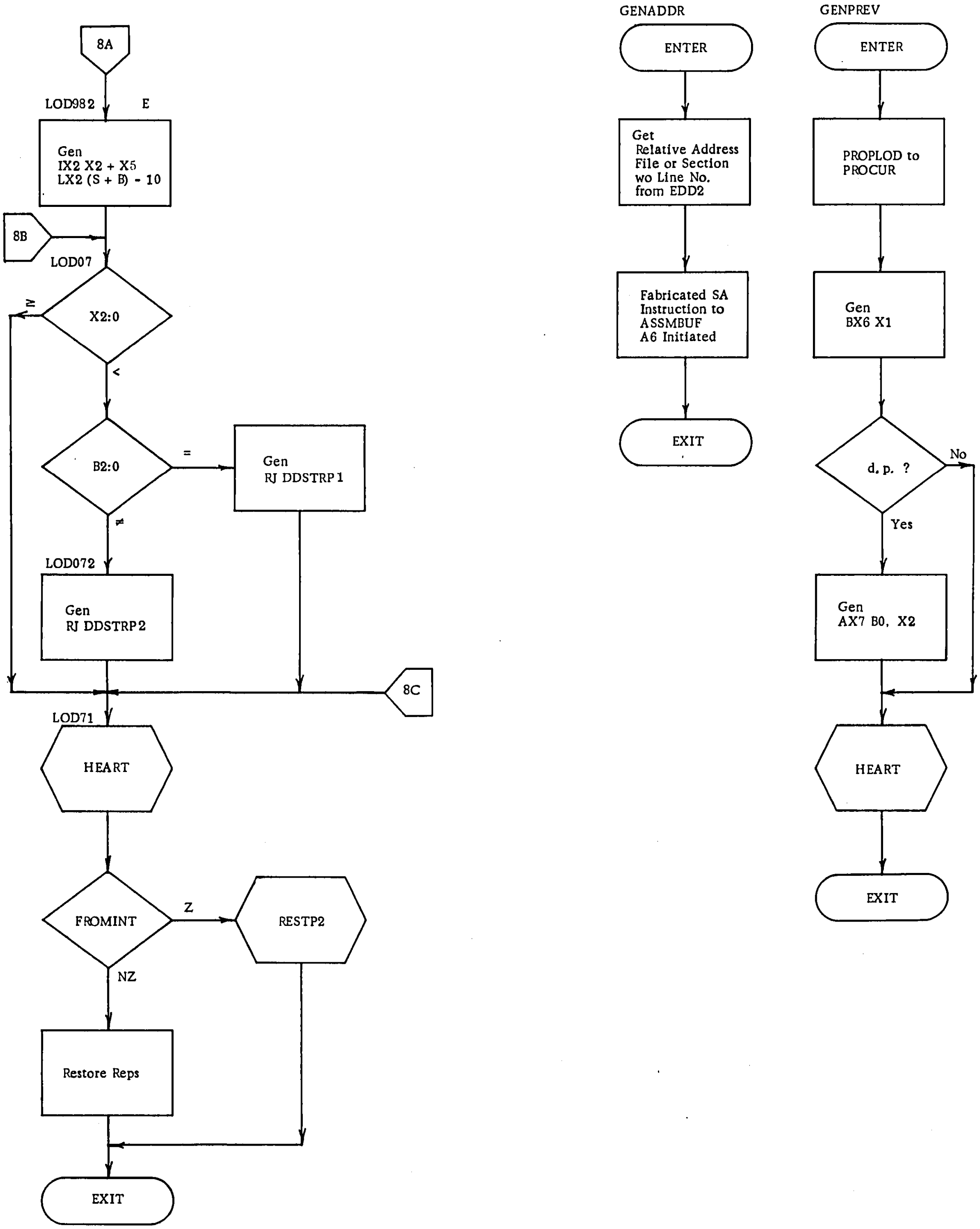


Figure 3-77. GENLOD Flowchart (8 of 8)

generated, loading the item into X3 and eliminating the insertion of leading zeros, etc. The properties of the loaded item are placed in PROPLOD.

If the leaf is not DNT, but is subscript, SUBSCRL is called.

The loading of literals or figurative constants is not accomplished in GENLOD, but is deferred until the properties of the other operand are known.

GENADDR

Purpose

To generate a load (SA) instruction addressing core.

Call

RJ GENADDR

Called by

GENLOD
GENSTO
GENMOVE

Operation

Decomposes EDD2 to construct the pseudo instruction needed by ART. Initializes the assembly buffer ASMBUF and its pointer, A6.

GENPREV

Purpose

To generate the transfer from LOD (X1) to CUR (X6) and the associated transfer of PROP.

Call

RJ GENPREV

Called by

PASS2 after GENLOD
GENMOVE
GENIF

GENSTO

Purpose

GENSTO generates code to store numeric or short alpha items in core. This code includes any conversions required, the insertion of sign if needed, and testing for ON-SIZE ERROR. Provision is made for holding and retrieving current results for many receiving fields. (See Figure 3-78.)

Called

from PASS 2 on a store node, with return to SCALE08
by RJ STORE
    from GENMOVE
        PRFOPS
    X5 contains the leaf
    X6 contains the leaf type
    B2 contains the tree pointer

Routines Used

HEART, CART
GENADDR
SUBSCR
LIT02

Operation

If the leaf is subscript, SUBSCRS is invoked. If it is not DNT, a literal, a message is written. If HOLDSW is on (i.e., the current result will be needed for additional stores, code is generated to move it to X3 or X3, 5. If later manipulations require the use of these registers, this code will be changed to save the current result in temporary cells in core. When GRABSW is on and the hold result is in temporary core, it is loaded into X3 and X5. X3/5 SW is then set, and, where possible manipulations, are performed on X3/5 rather than X6/7. In both cases above, the PROPERTY is transferred in the compiler as code is generated to transfer the item.

The DNT information of the receiving field and PROPCUR are extracted next. If the receiving field is DPC and CUR is binary, code is generated to convert the result to DPC. Then, on a DPC receiving field, code, consisting of masks and shifts, is generated to align decimal points as needed, and to round by adding decimally an appropriate 5 if source code requires this. If the source requires BLANK WHEN ZERO, code is generated to accomplish this. Next, if the receiving field is signed, code is generated to call D. ZONE to complement a negative result and to insert a zone underpunch. If the receiving field is

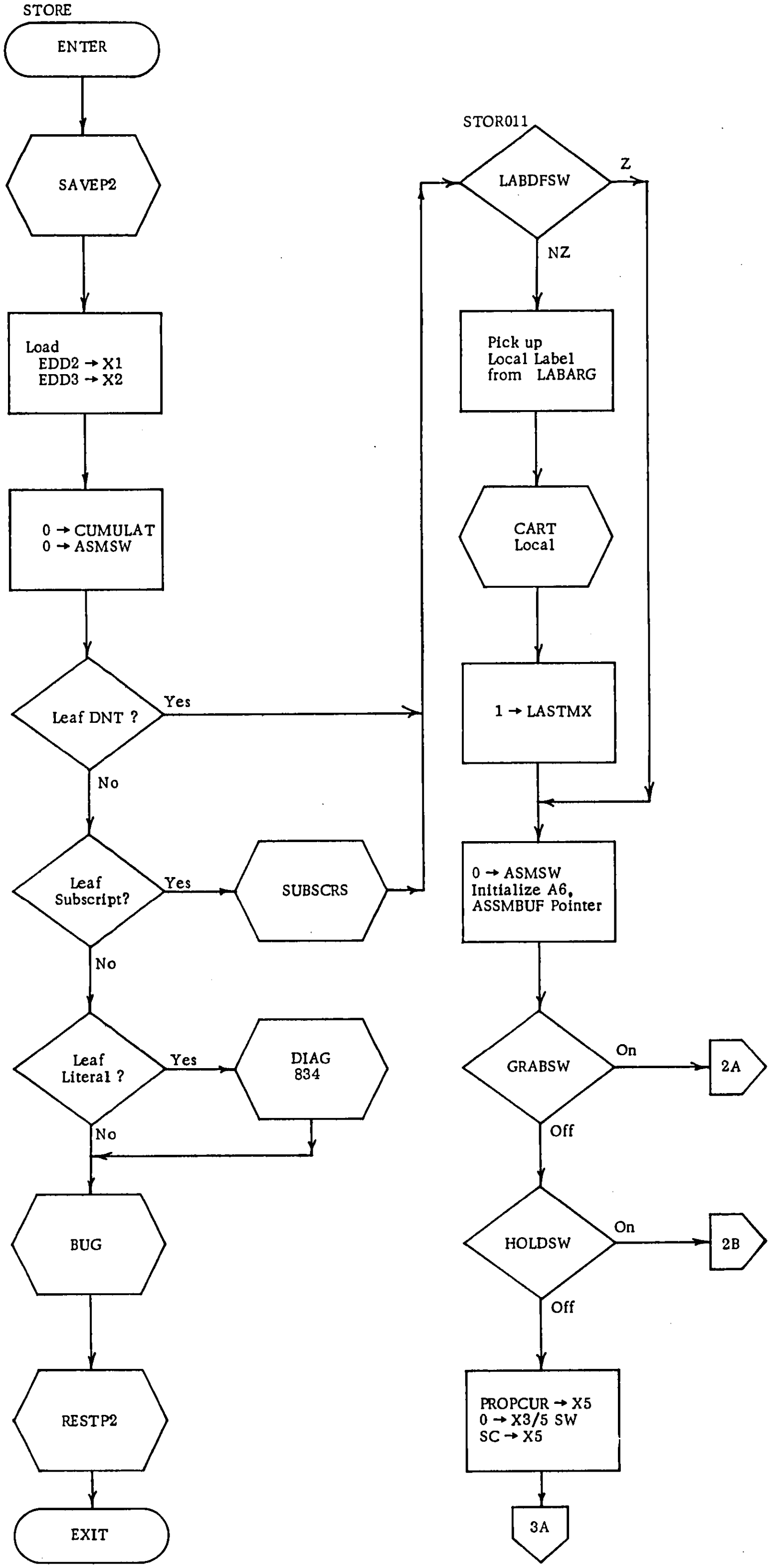


Figure 3-78. GENSTO Flowchart (1 of 44)

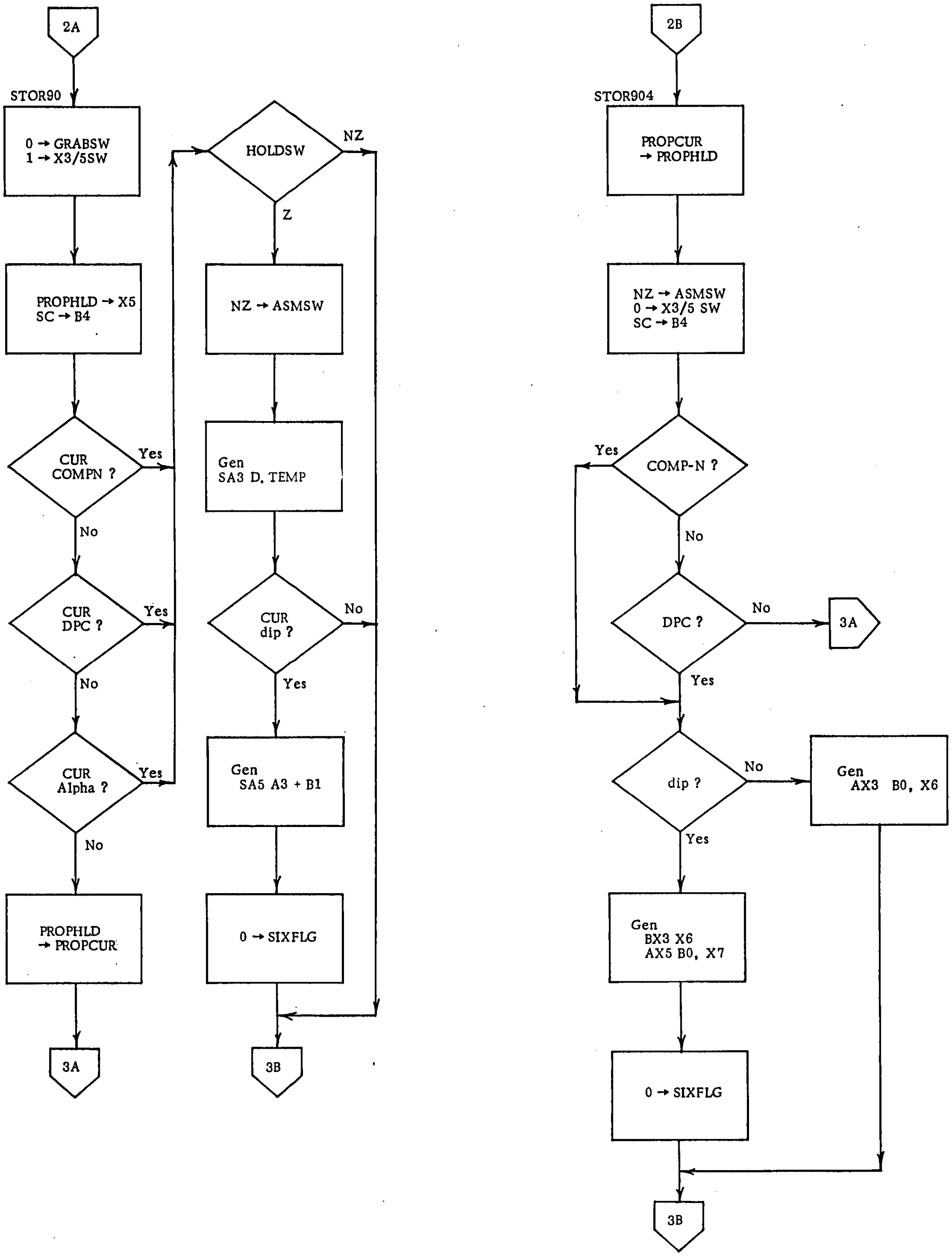


Figure 3-78. GENSTO Flowchart (2 of 44)

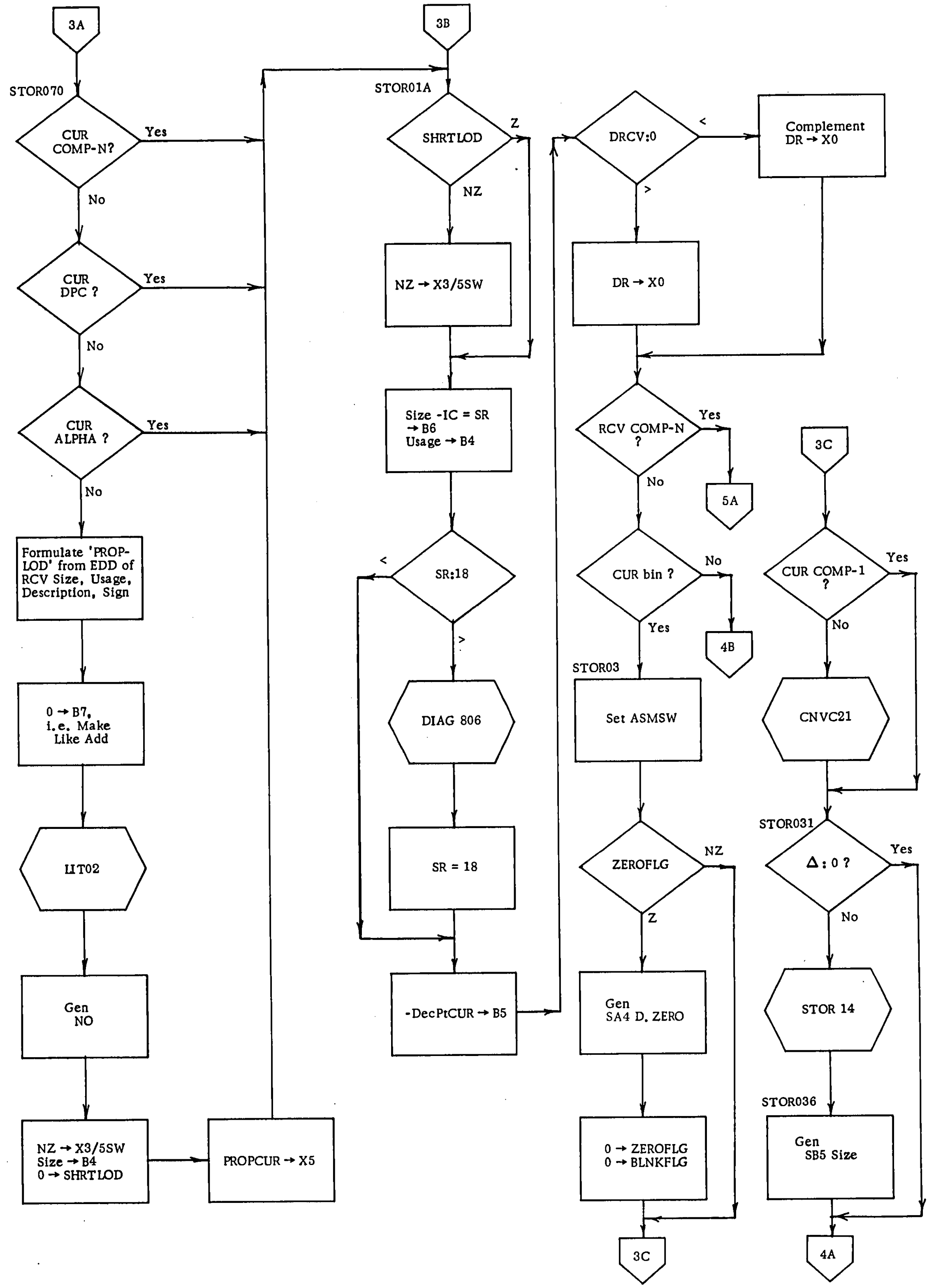


Figure 3-78. GENSTO Flowchart (3 of 44)

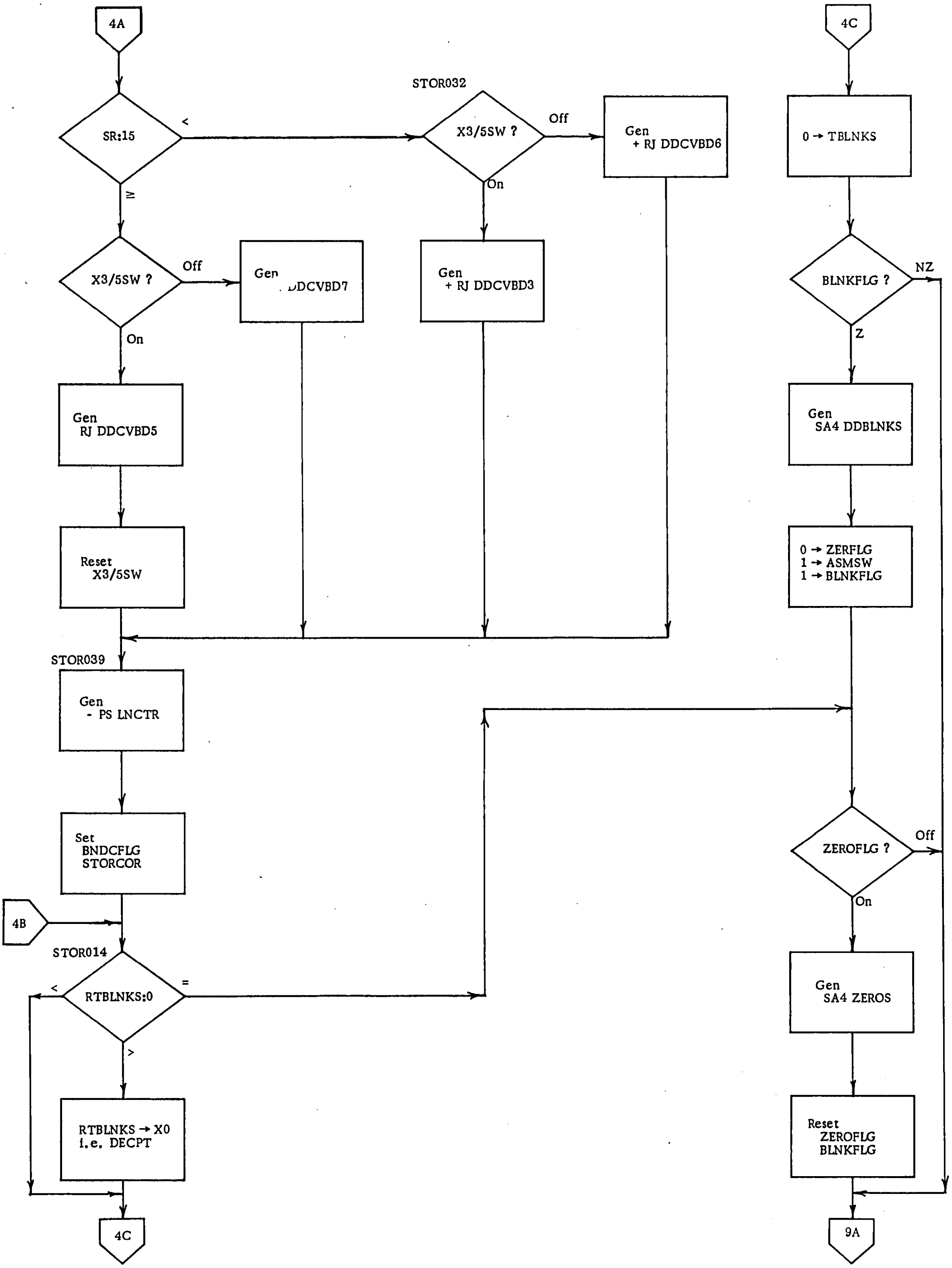


Figure 3-78. GENSTO Flowchart (4 of 44)

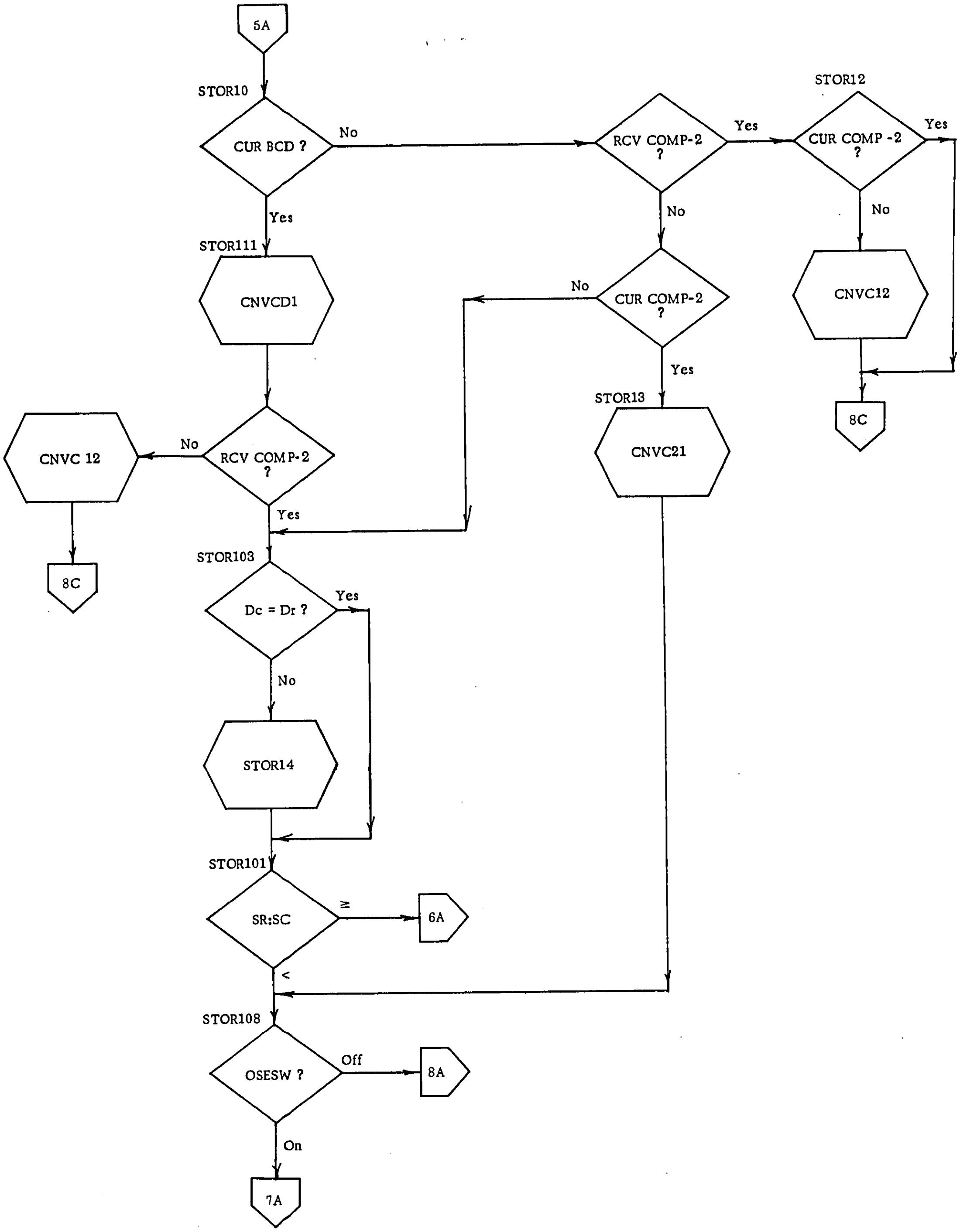


Figure 3-78. GENSTO Flowchart (5 of 44)

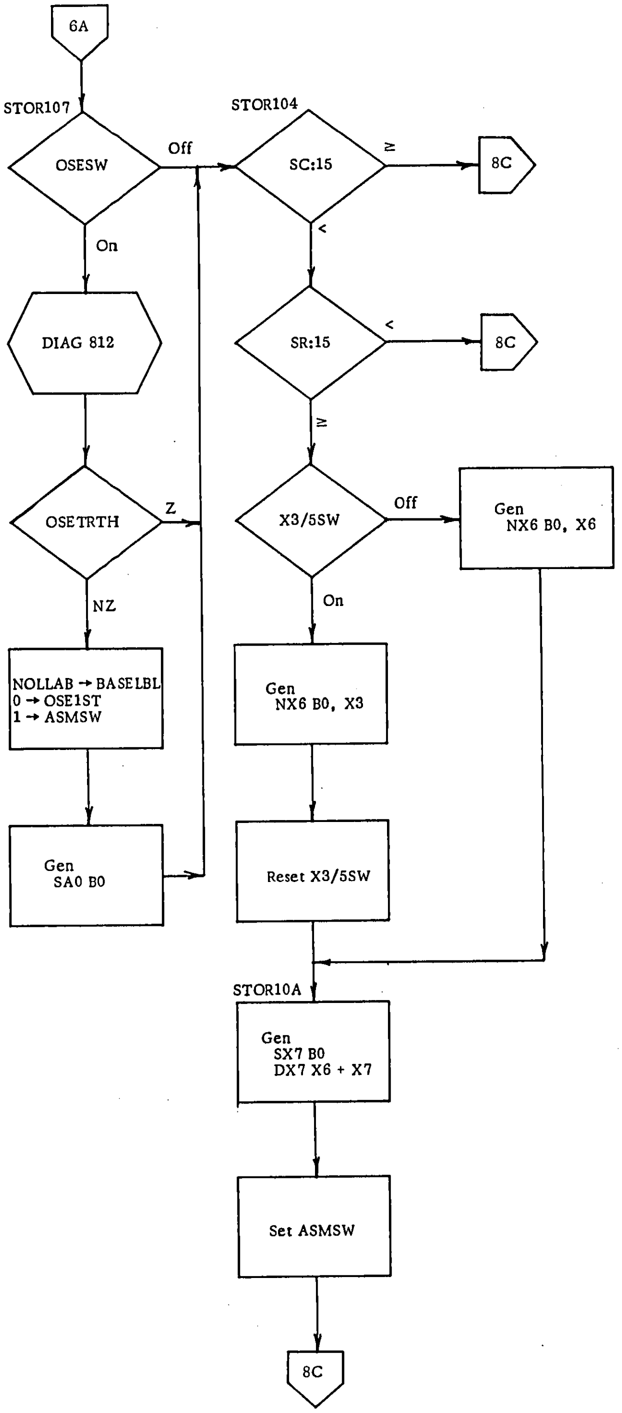


Figure 3-78. GENSTO Flowchart (6 of 44)

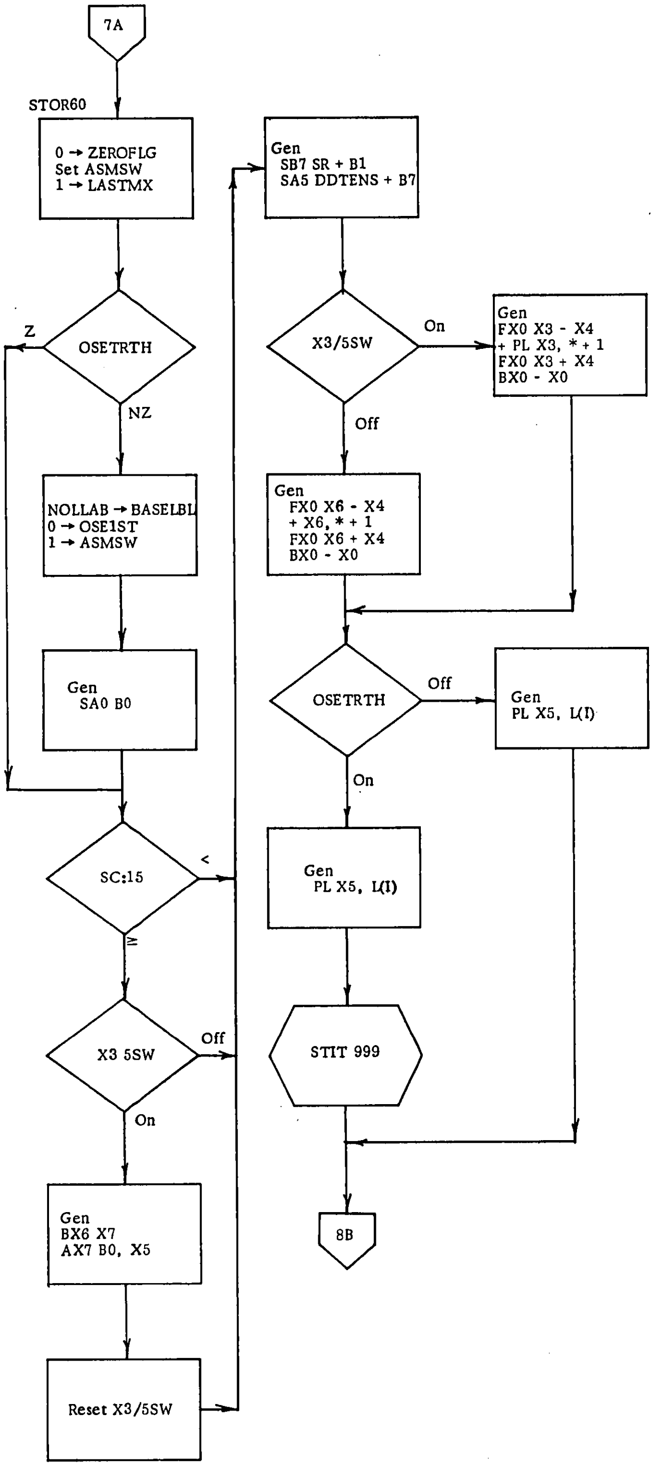


Figure 3-78. GENSTO Flowchart(7 of 44)

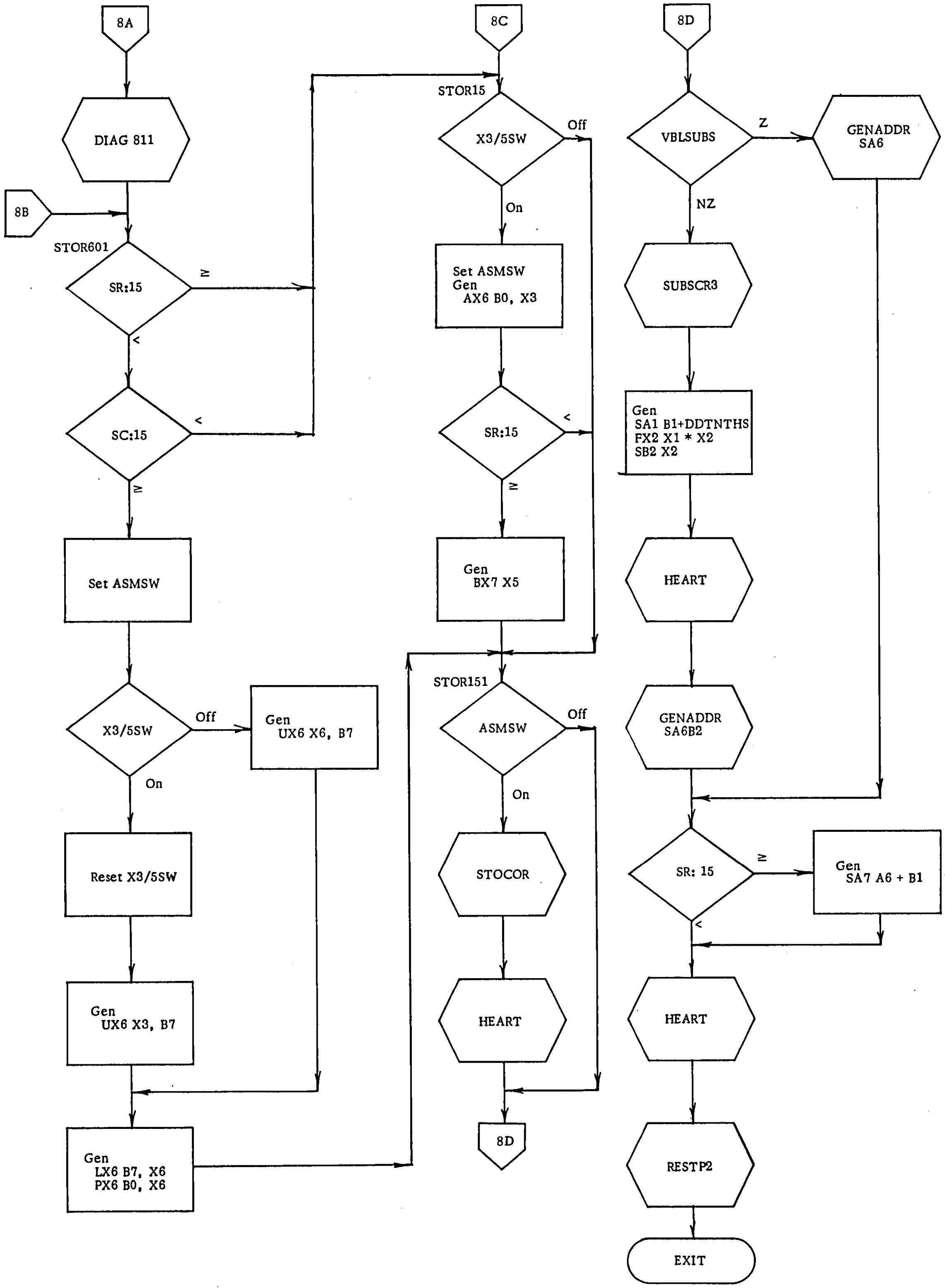


Figure 3-78. GENSTO Flowchart (8 of 44)

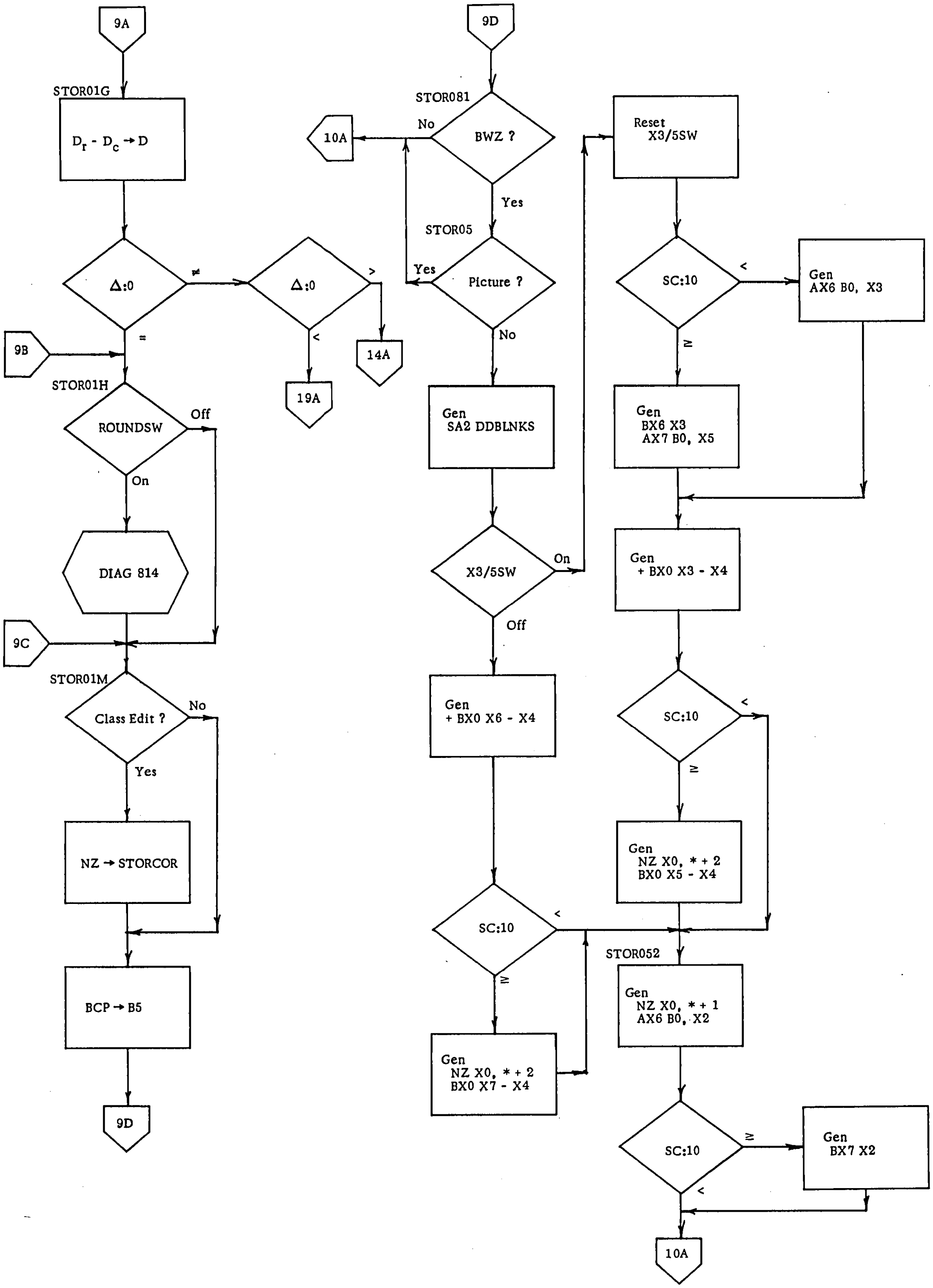


Figure 3-78. GENSTO Flowchart (9 of 44)

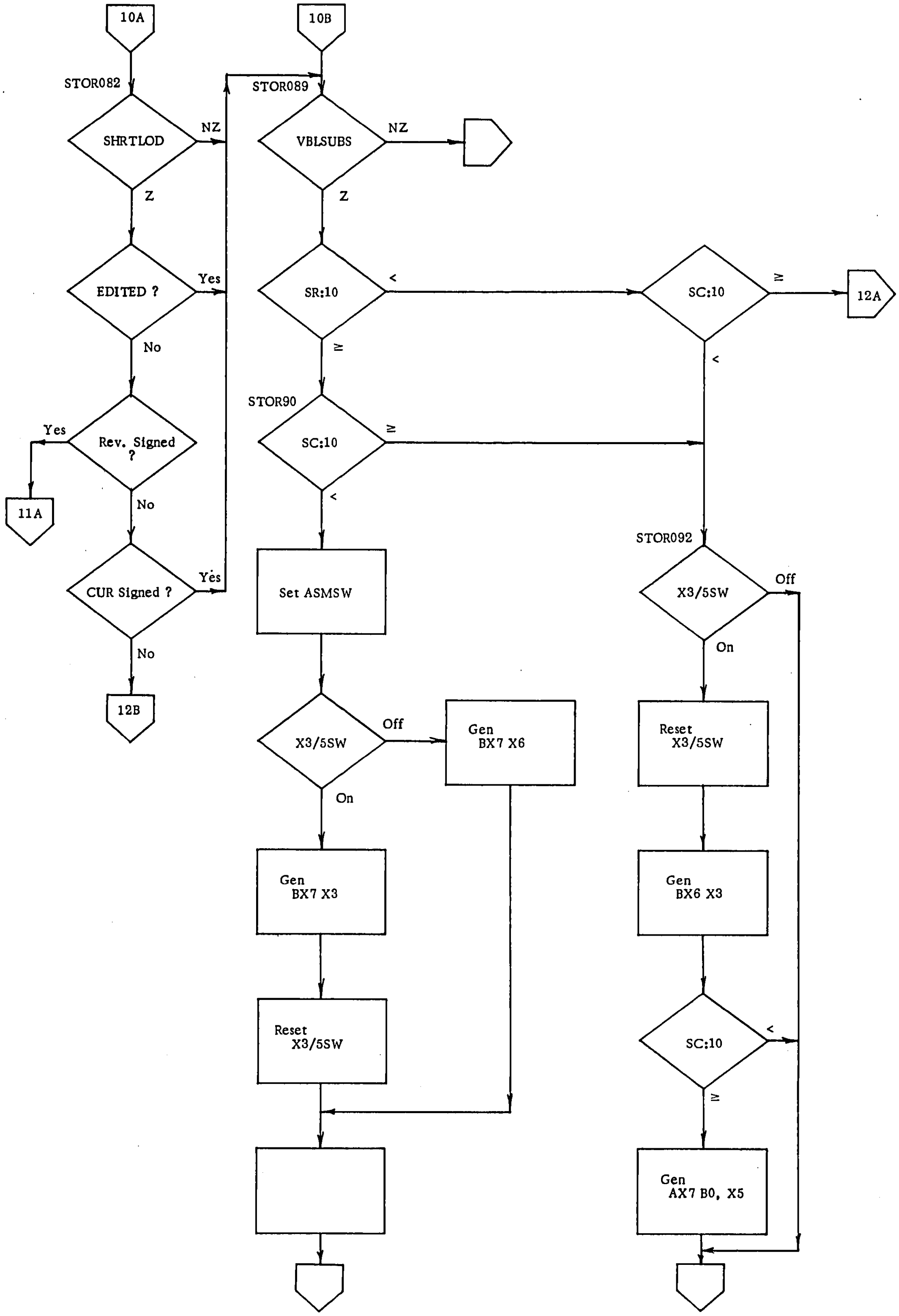


Figure 3-78. GENSTO Flowchart (10 of 44)

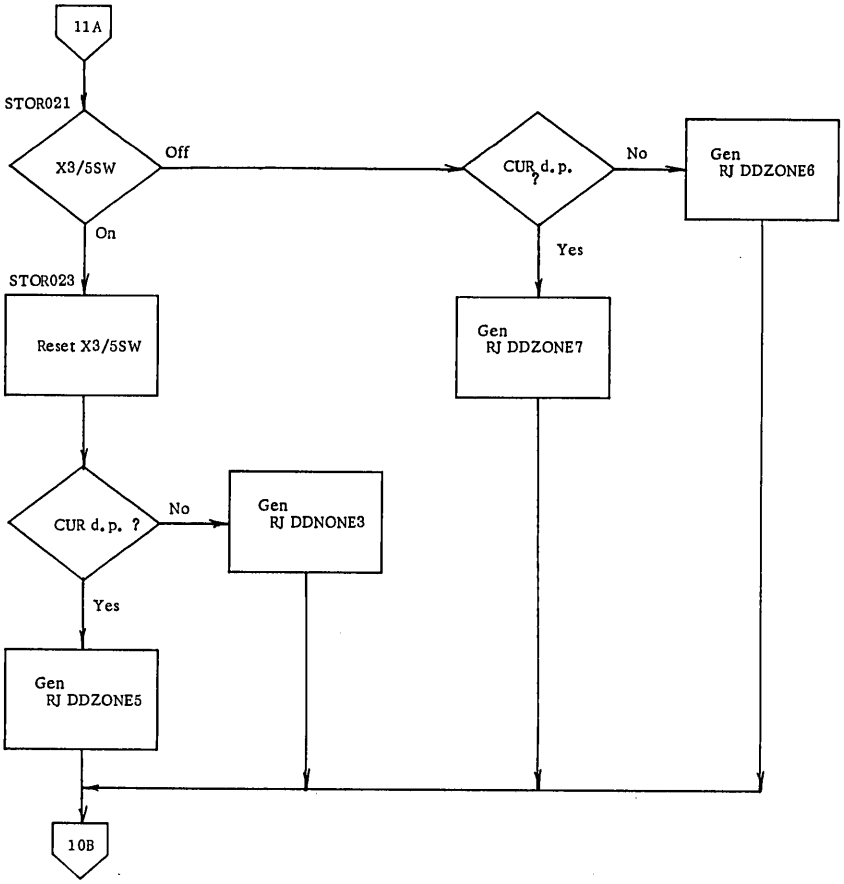


Figure 3-78. GENSTO Flowchart (11 of 44)

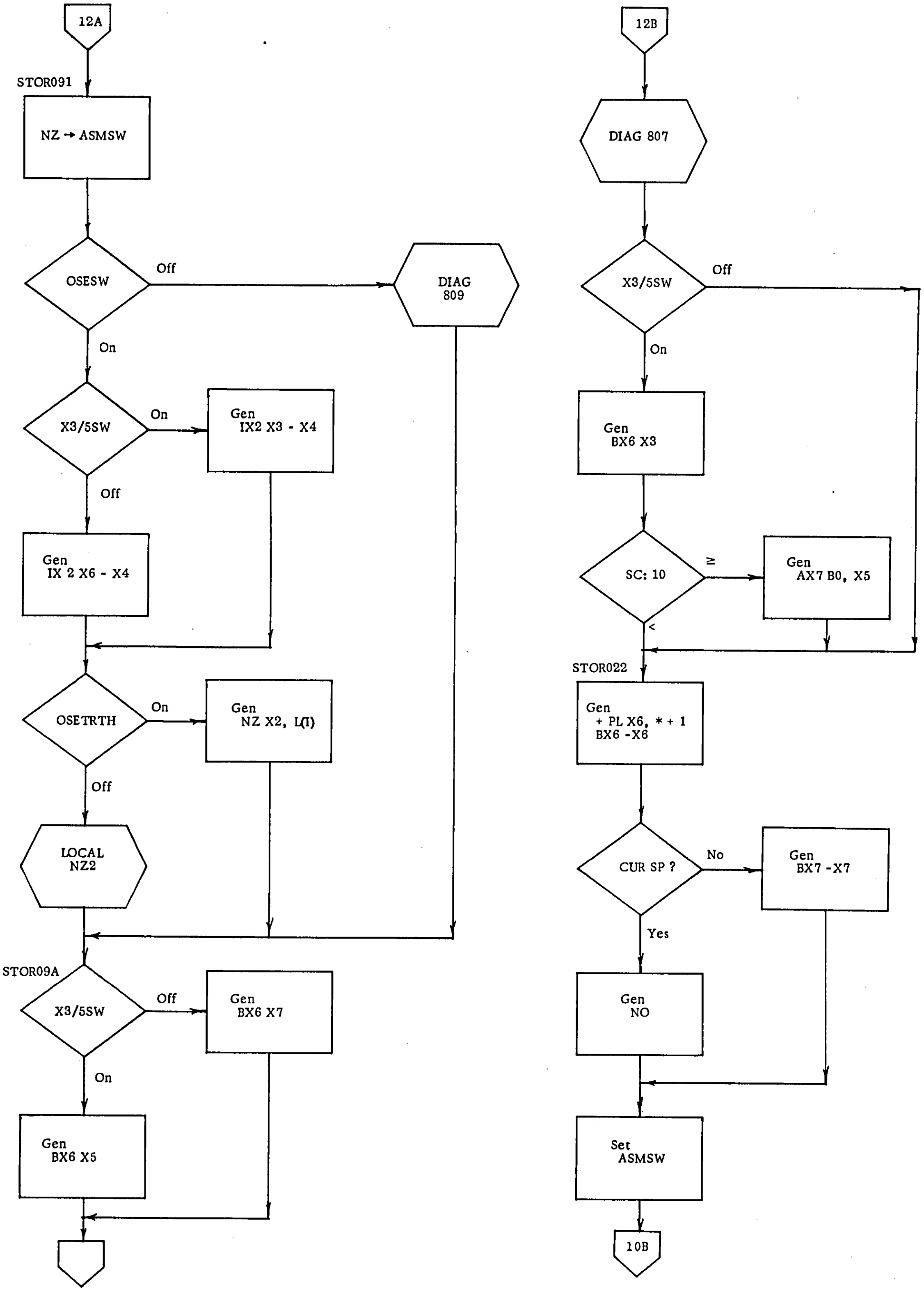


Figure 3-78. GENSTO Flowchart (12 of 44)

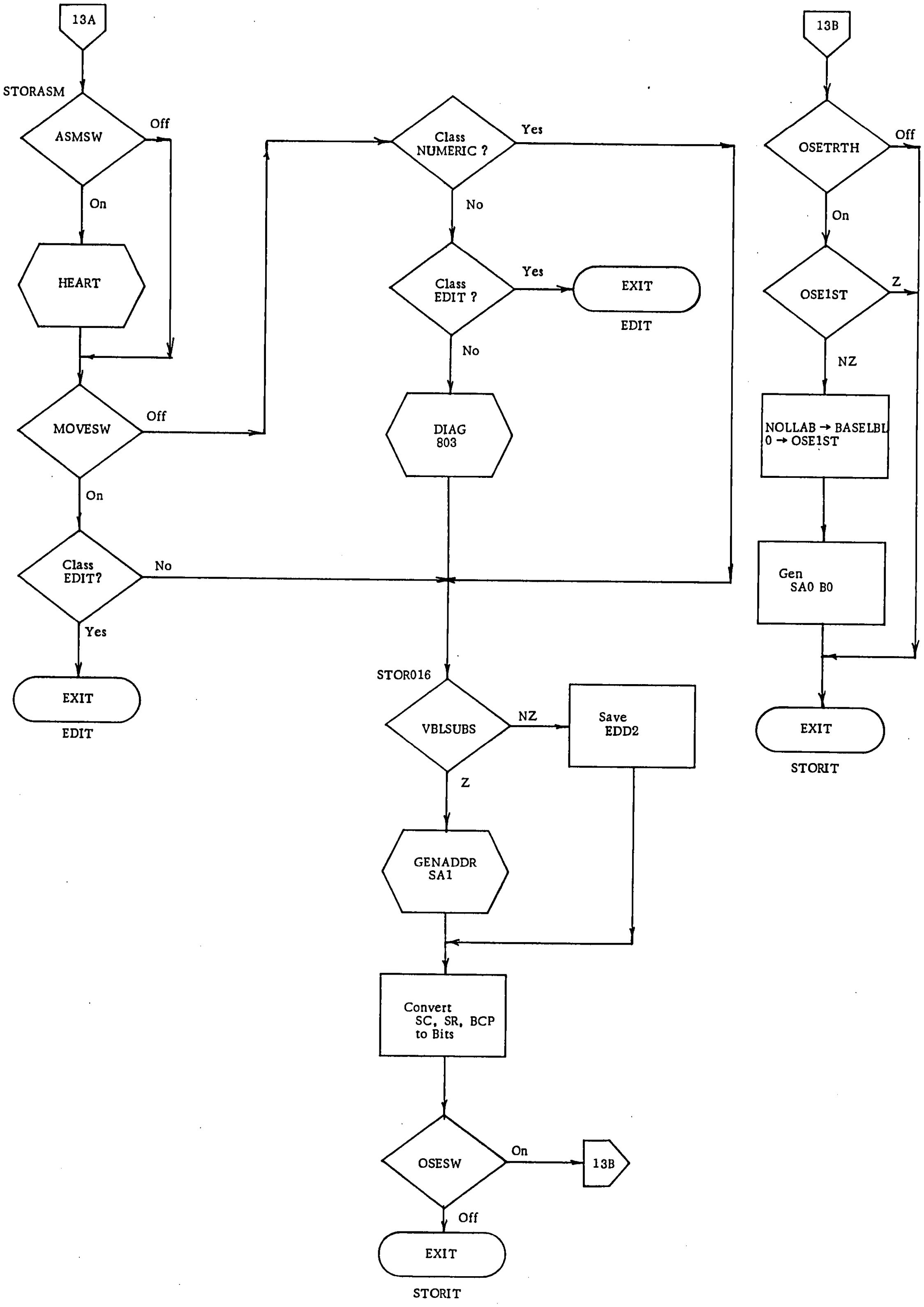


Figure 3-78. GENSTO Flowchart (13 of 44)

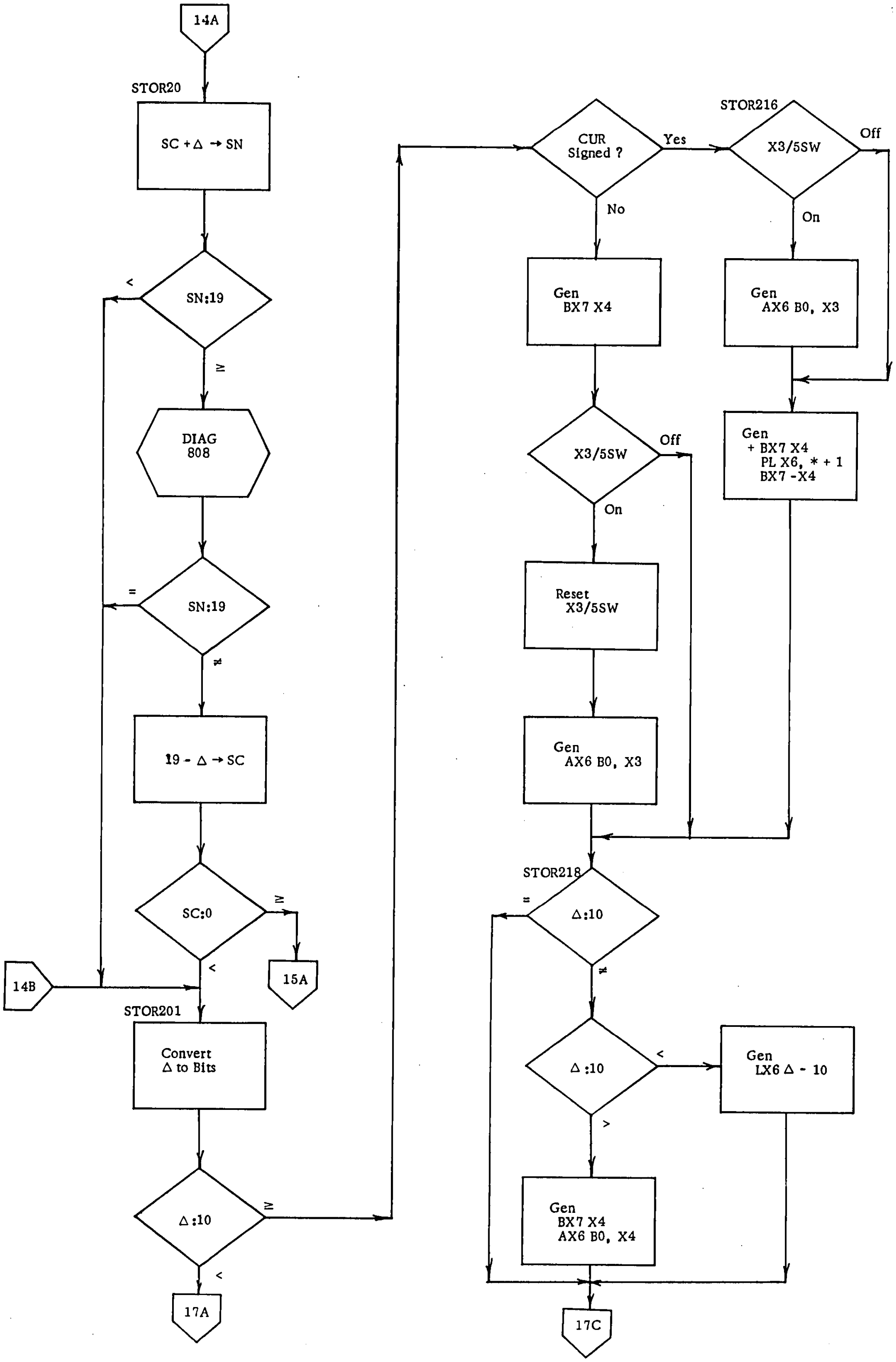


Figure 3-78. GENSTO Flowchart (14 of 44)

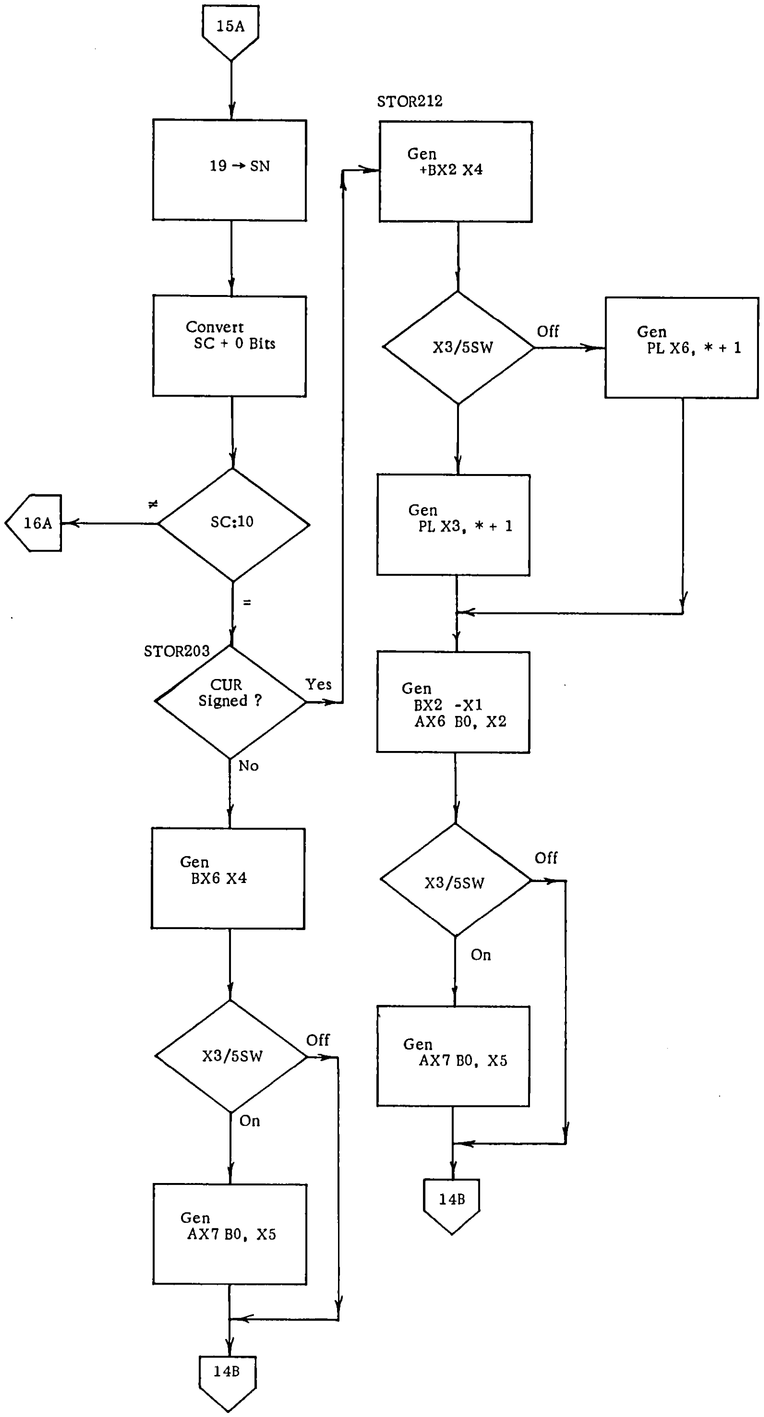


Figure 3-78. GENSTO Flowchart (15 of 44)

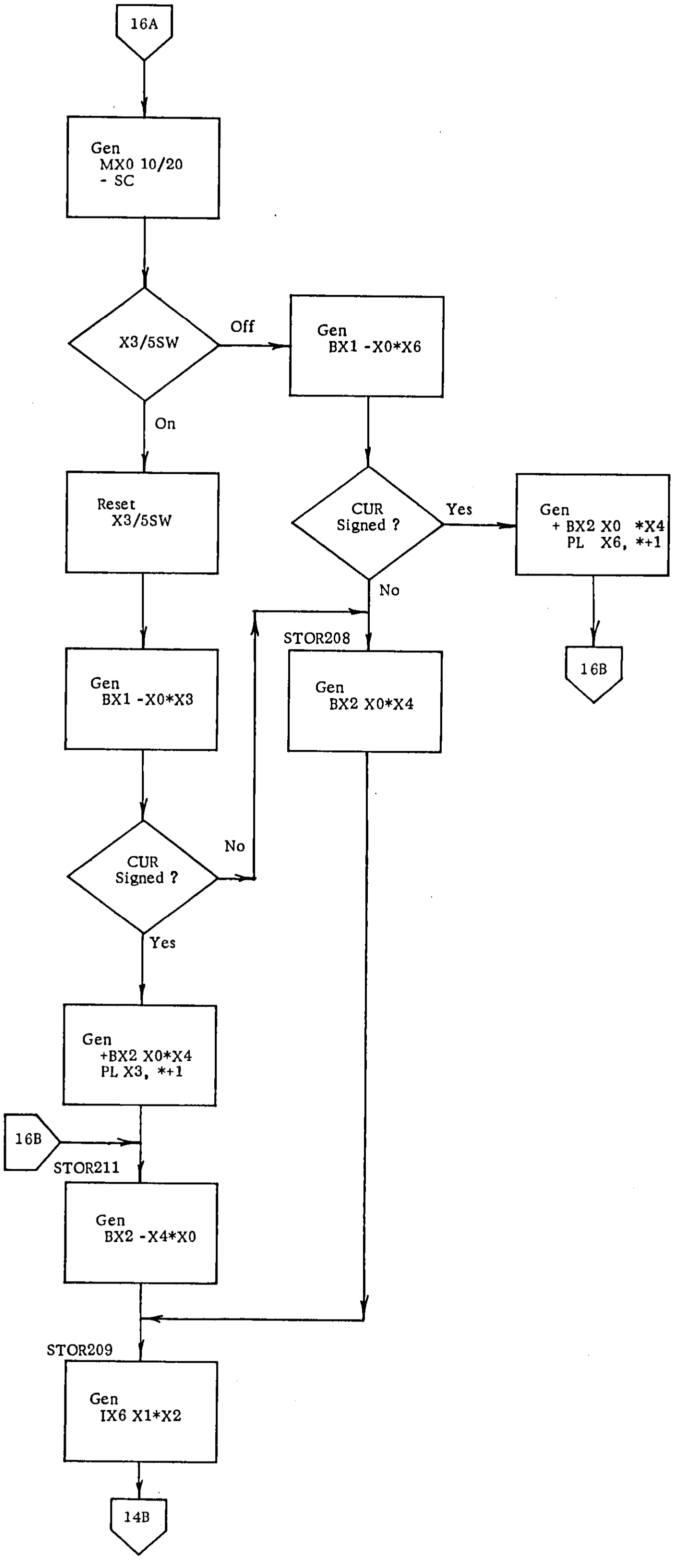


Figure 3-78. GENSTO Flowchart (16 of 44)

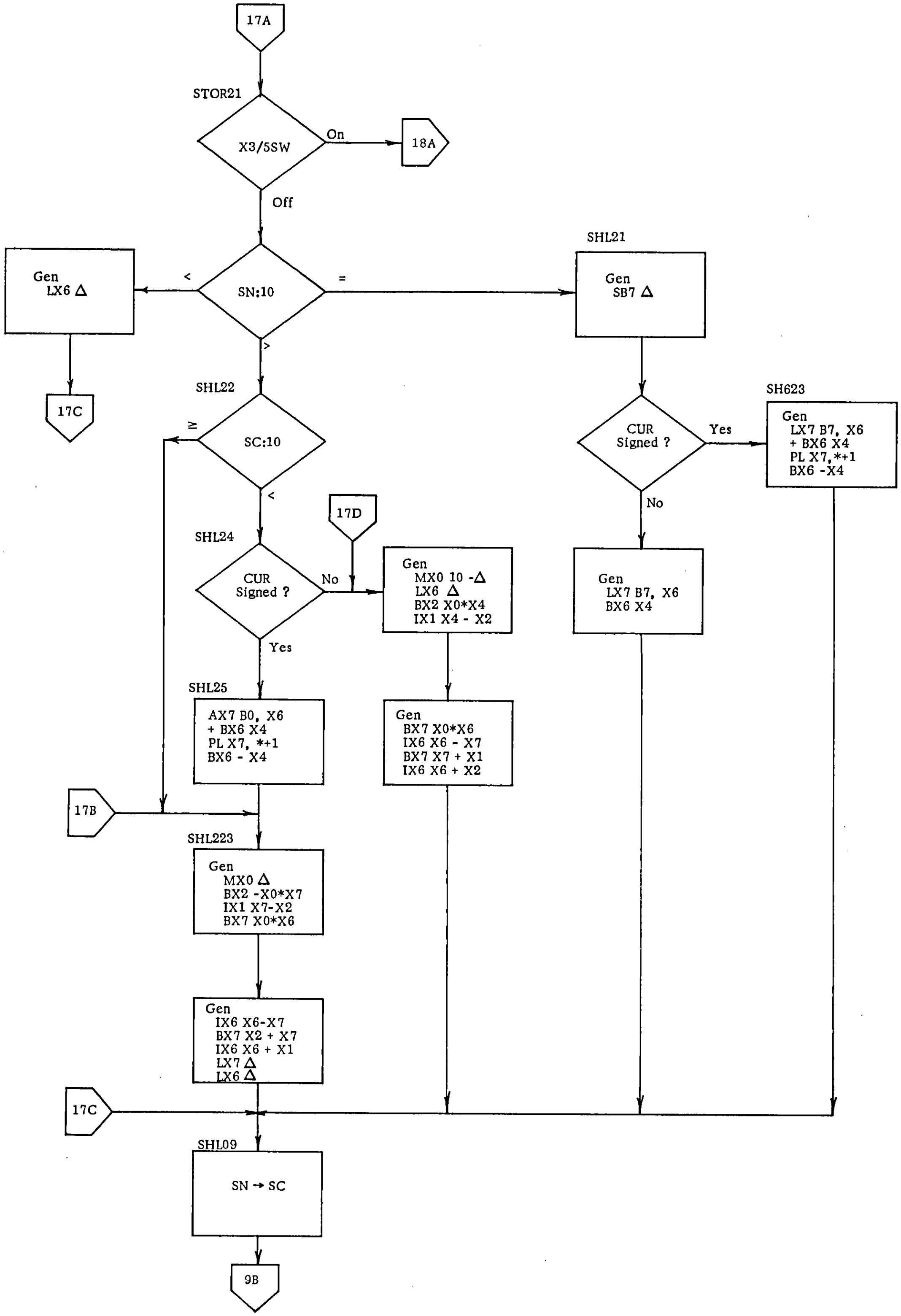


Figure 3-78. GENSTO Flowchart (17 of 44)

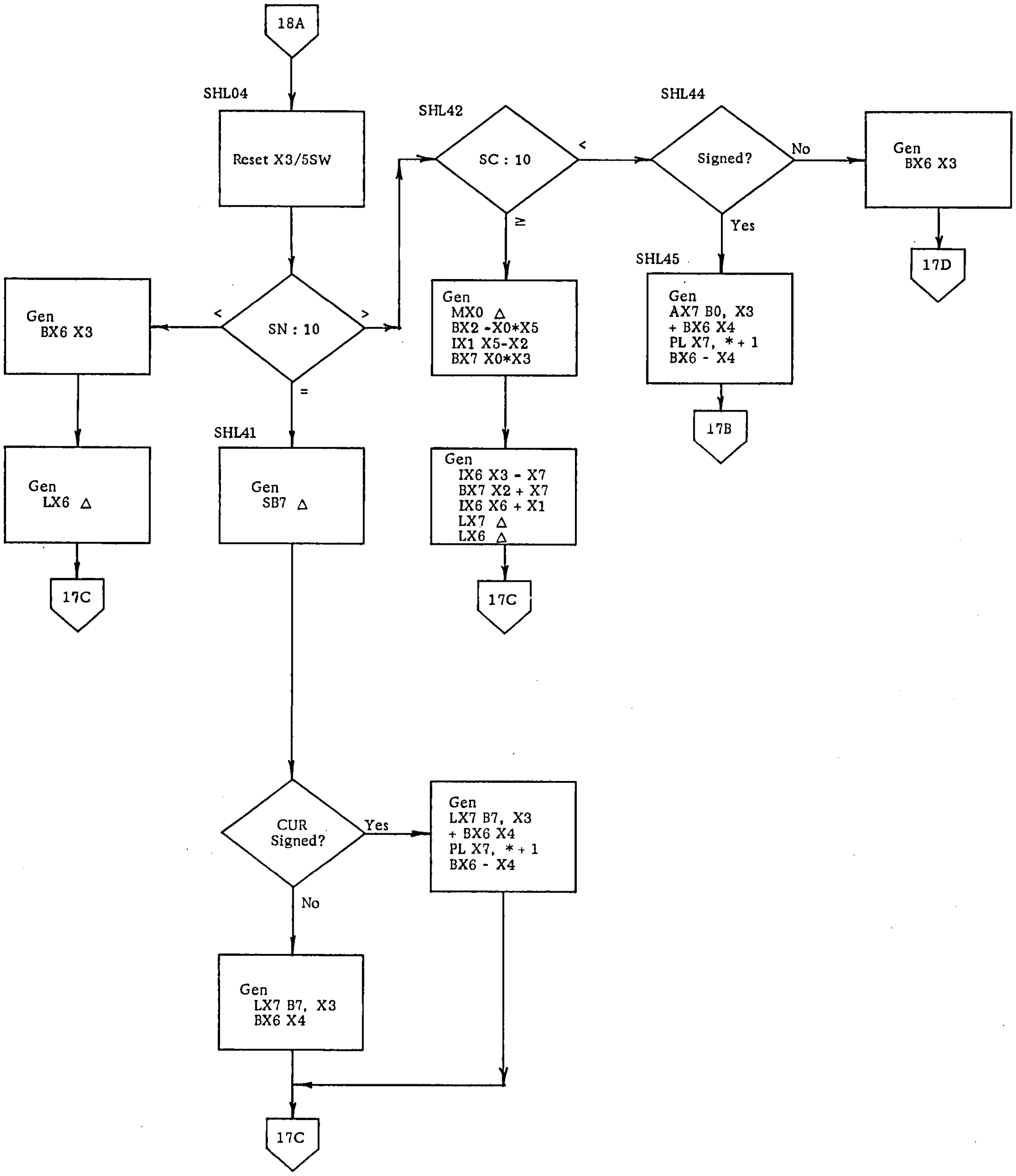


Figure 3-78. GENSTO Flowchart (18 of 44)

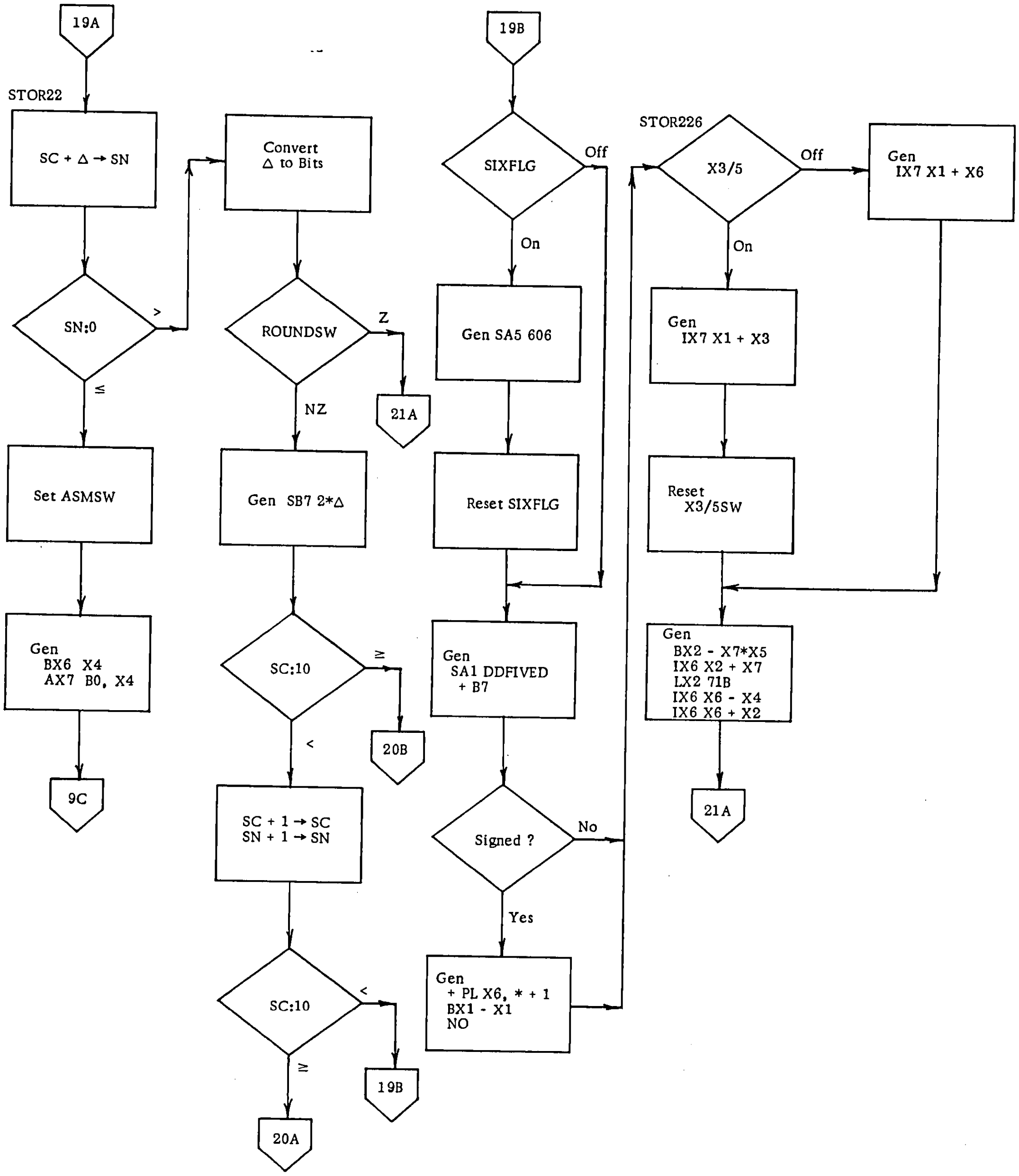


Figure 3-78. GENSTO Flowchart (19 of 44)

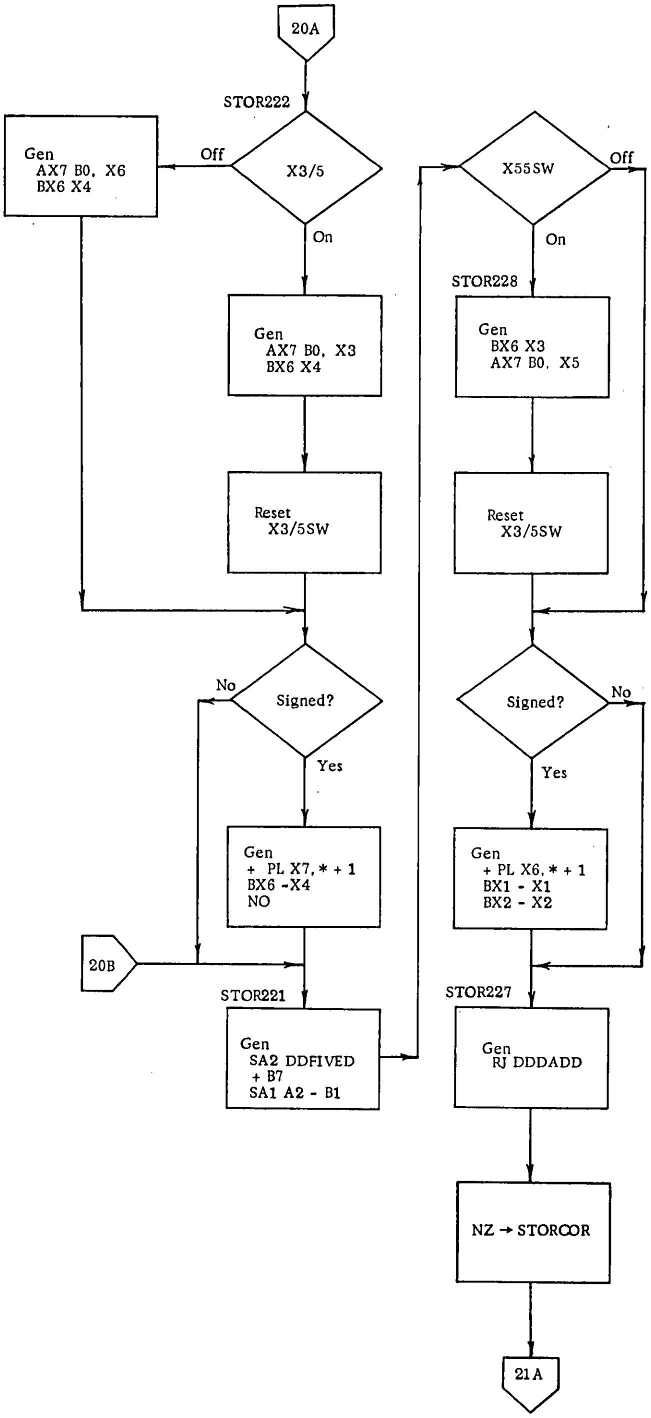


Figure 3-78. GENSTO Flowchart (20 of 44)

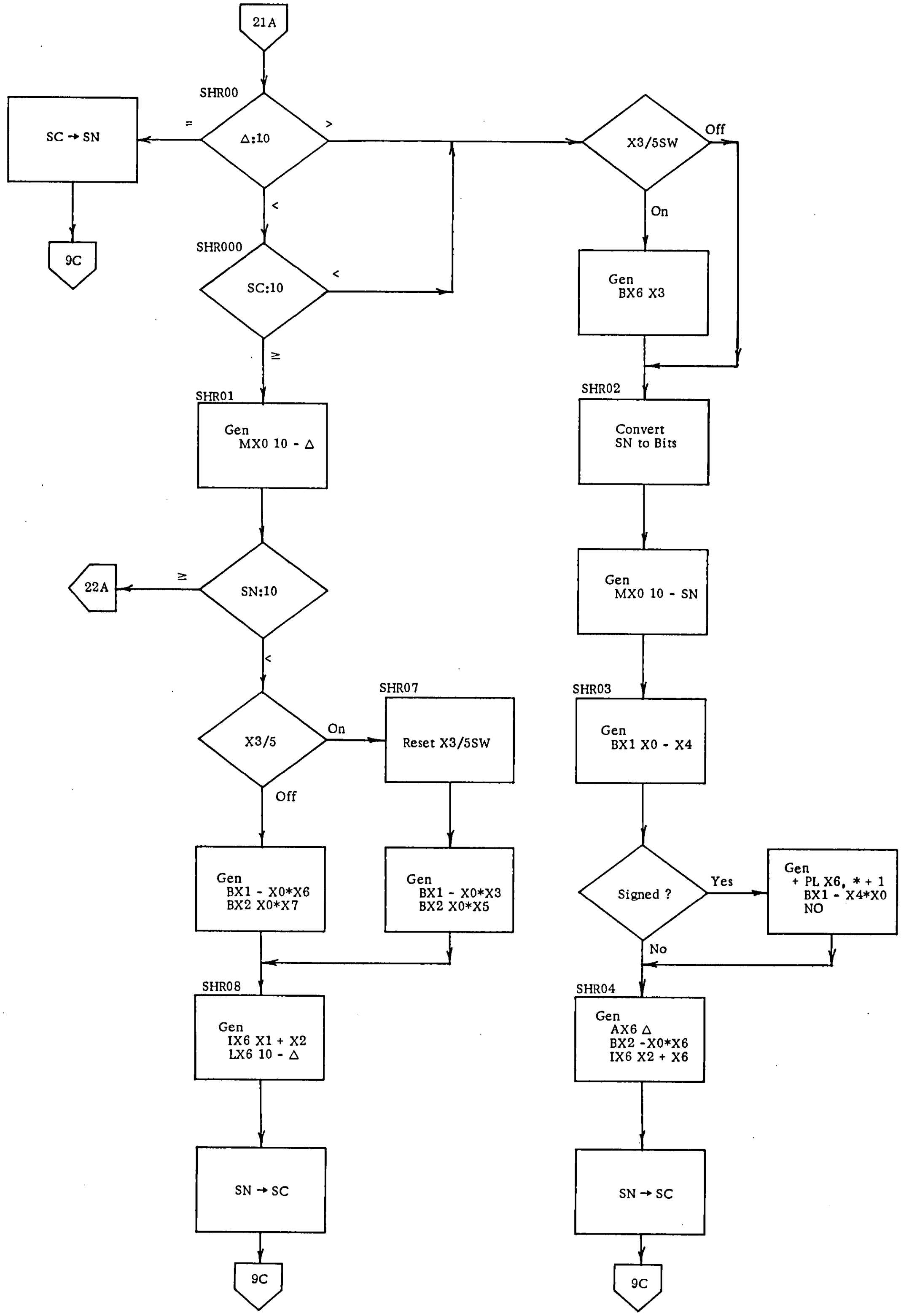


Figure 3-78. GENSTO Flowchart (21 of 44)

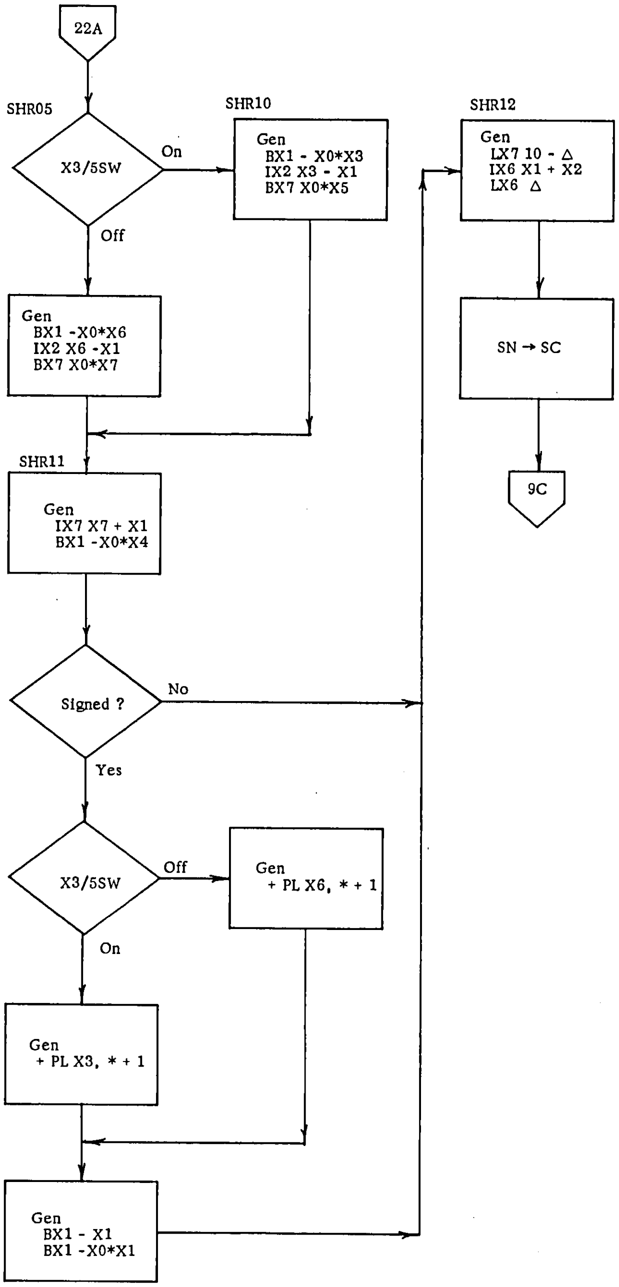


Figure 3-78. GENSTO Flowchart (22 of 44)

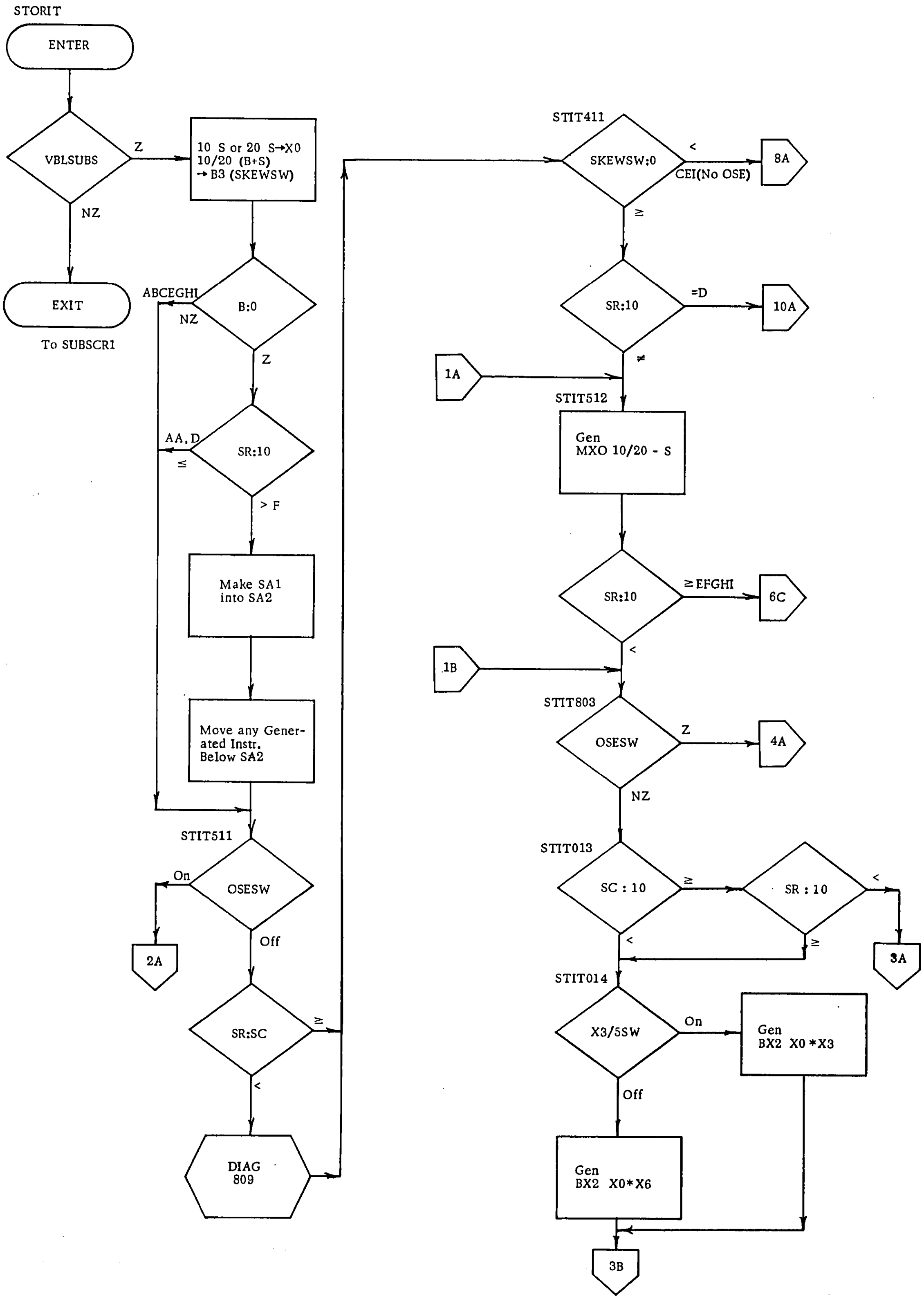


Figure 3-78. GENSTO Flowchart (23 of 44)

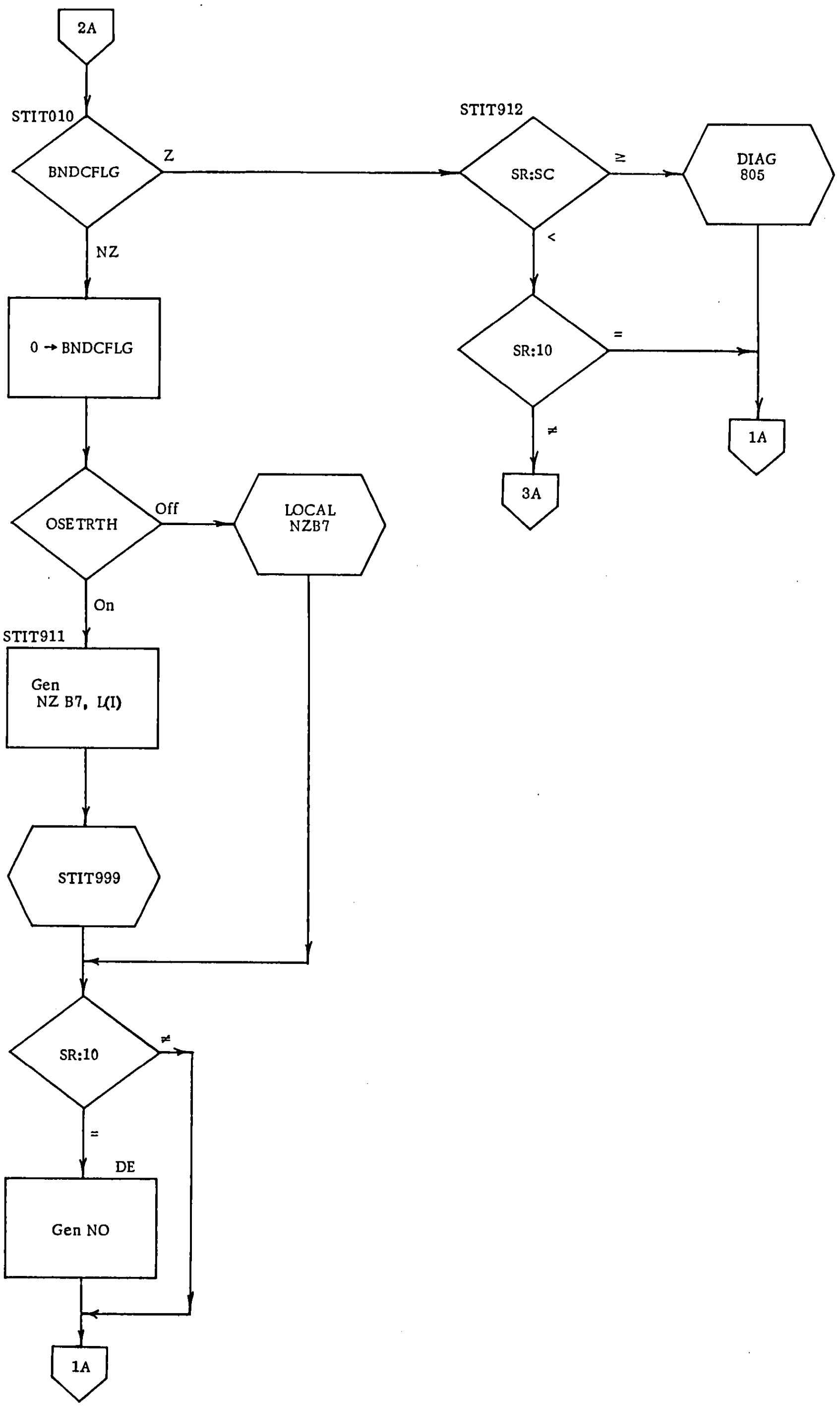


Figure 3-78. GENSTO Flowchart (24 of 44)

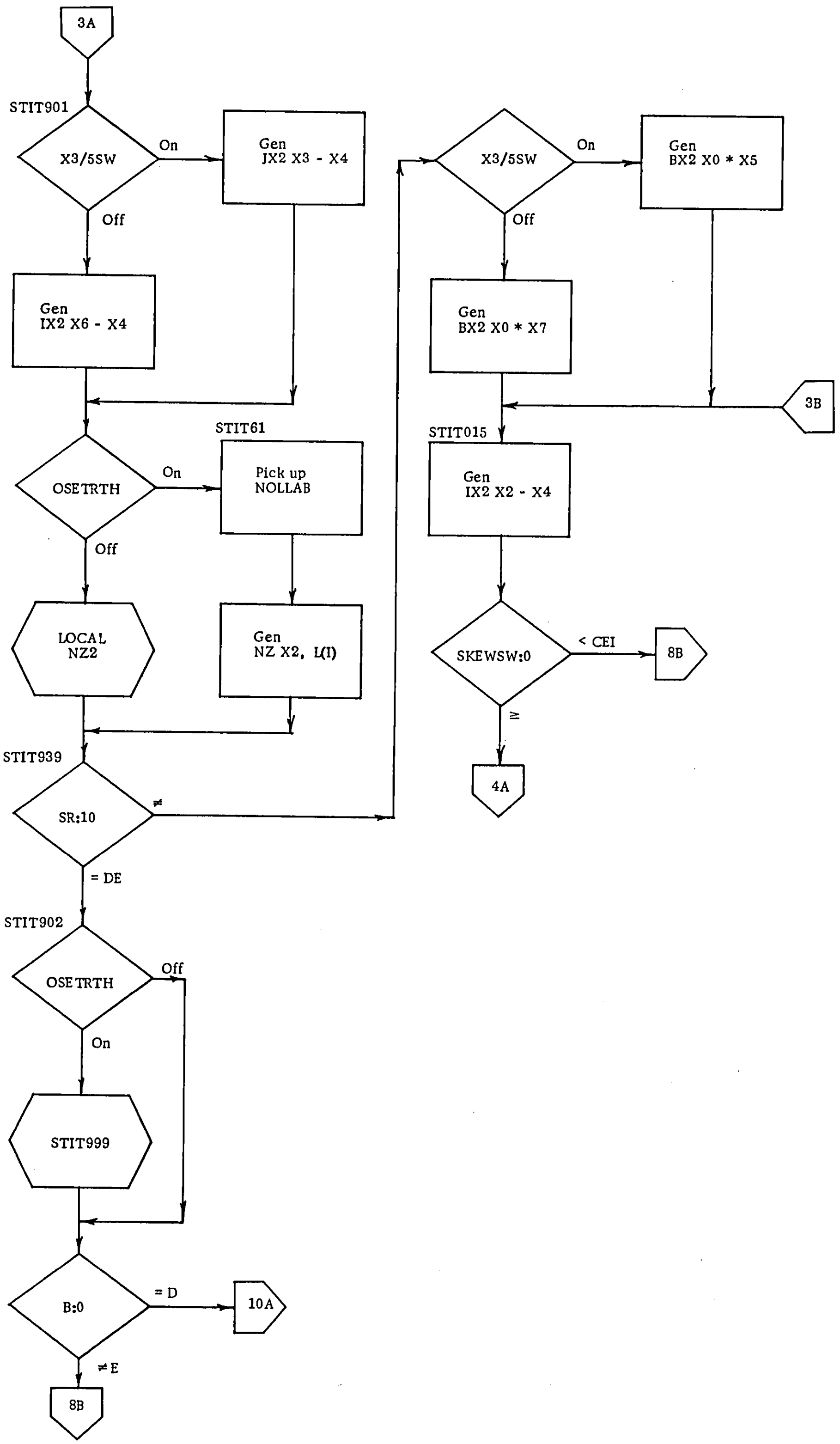


Figure 3-78. GENSTO Flowchart (25 of 44)

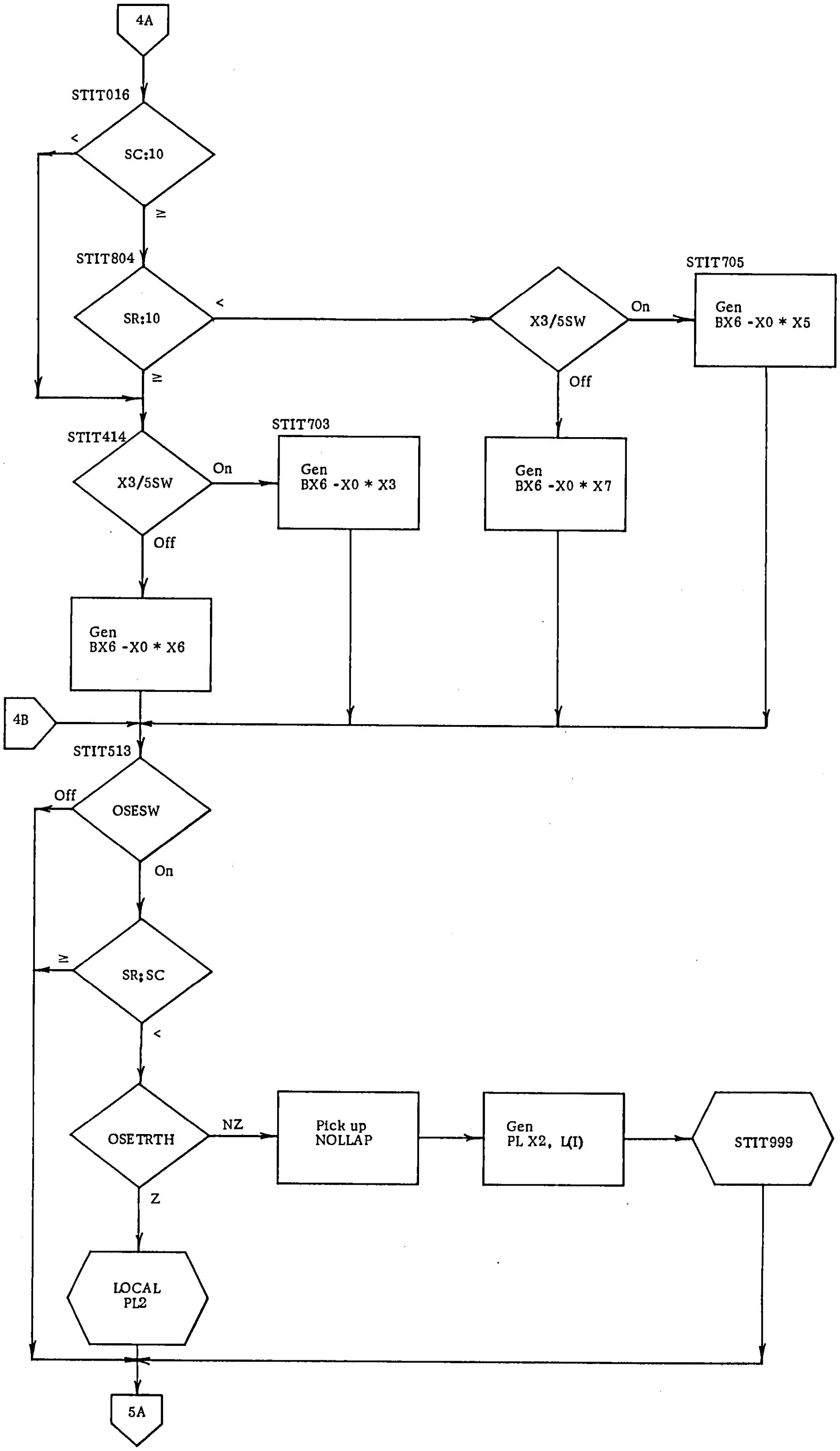


Figure 3-78. GENSTO Flowchart (26 of 44)

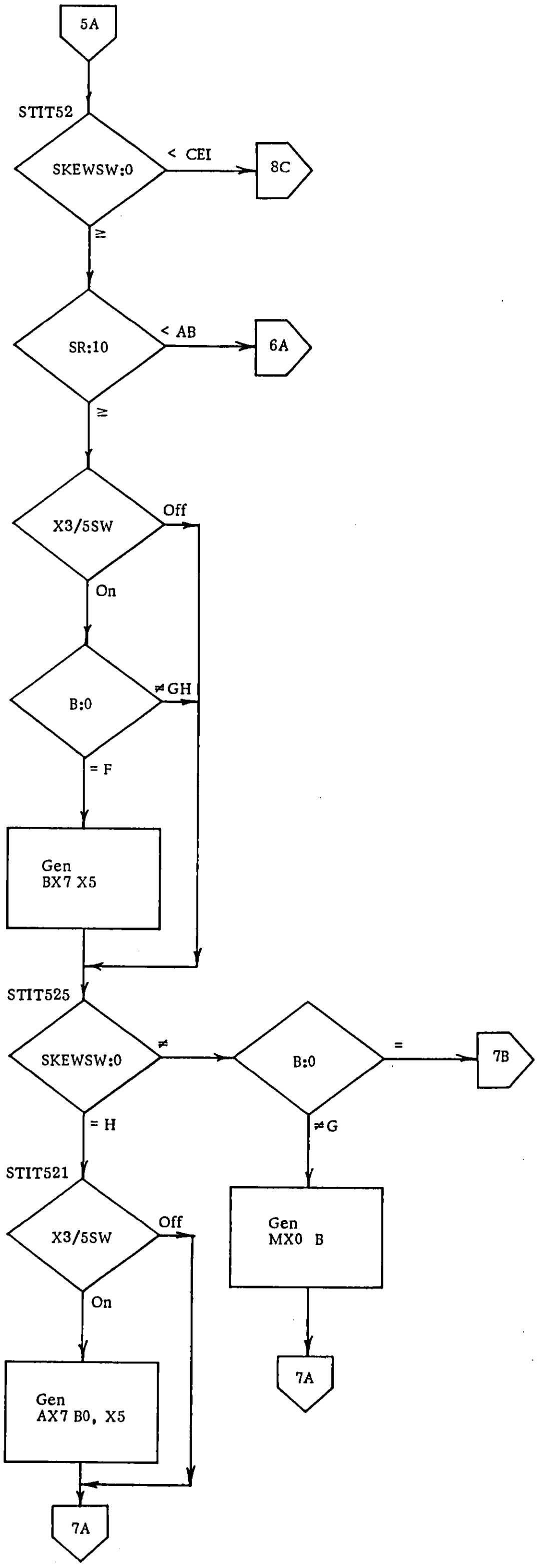


Figure 3-78. GENSTO Flowchart (27 of 44)

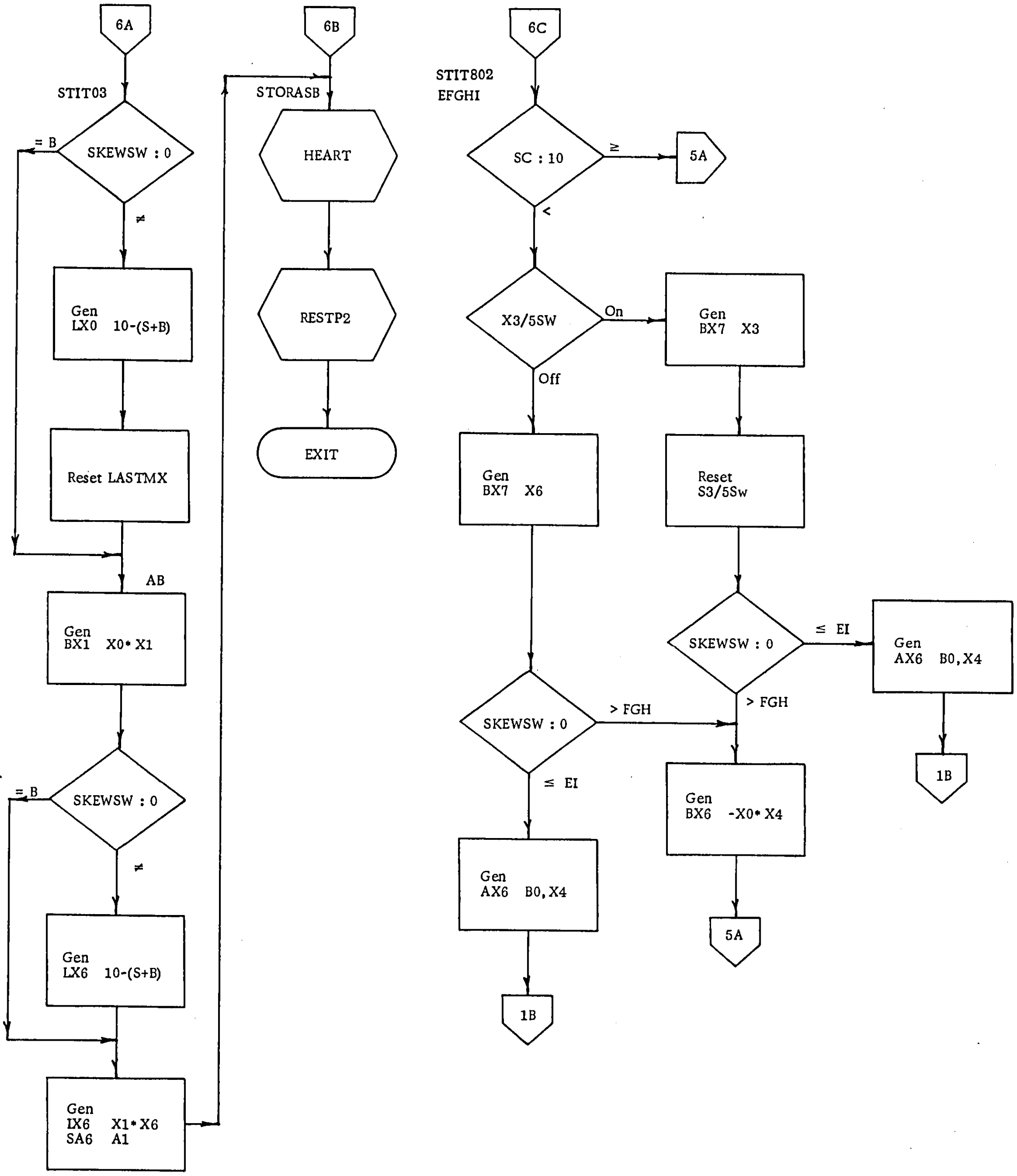


Figure 3-78. GENSTO Flowchart (28 of 44)

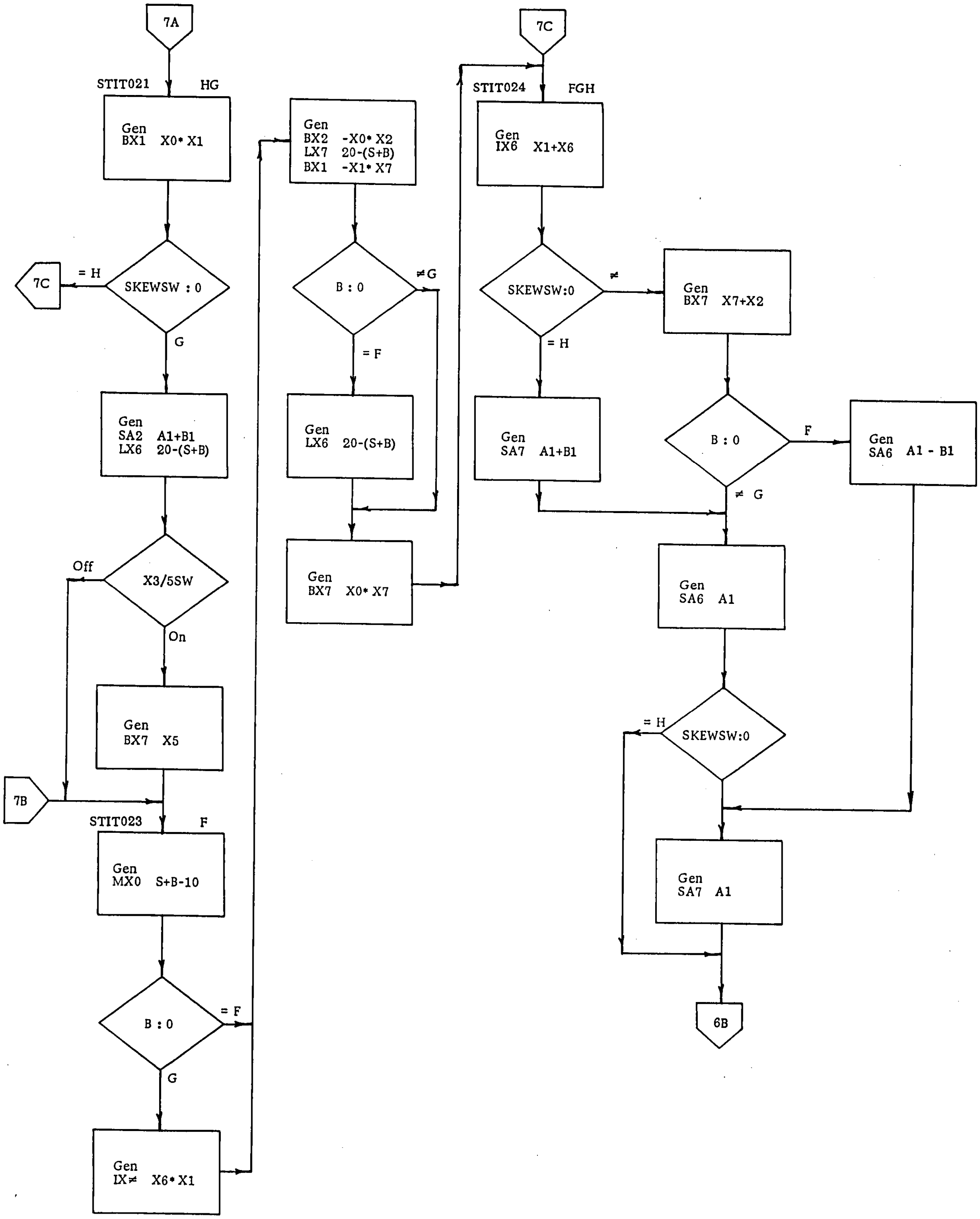


Figure 3-78. GENSTO Flowchart (29 of 44)

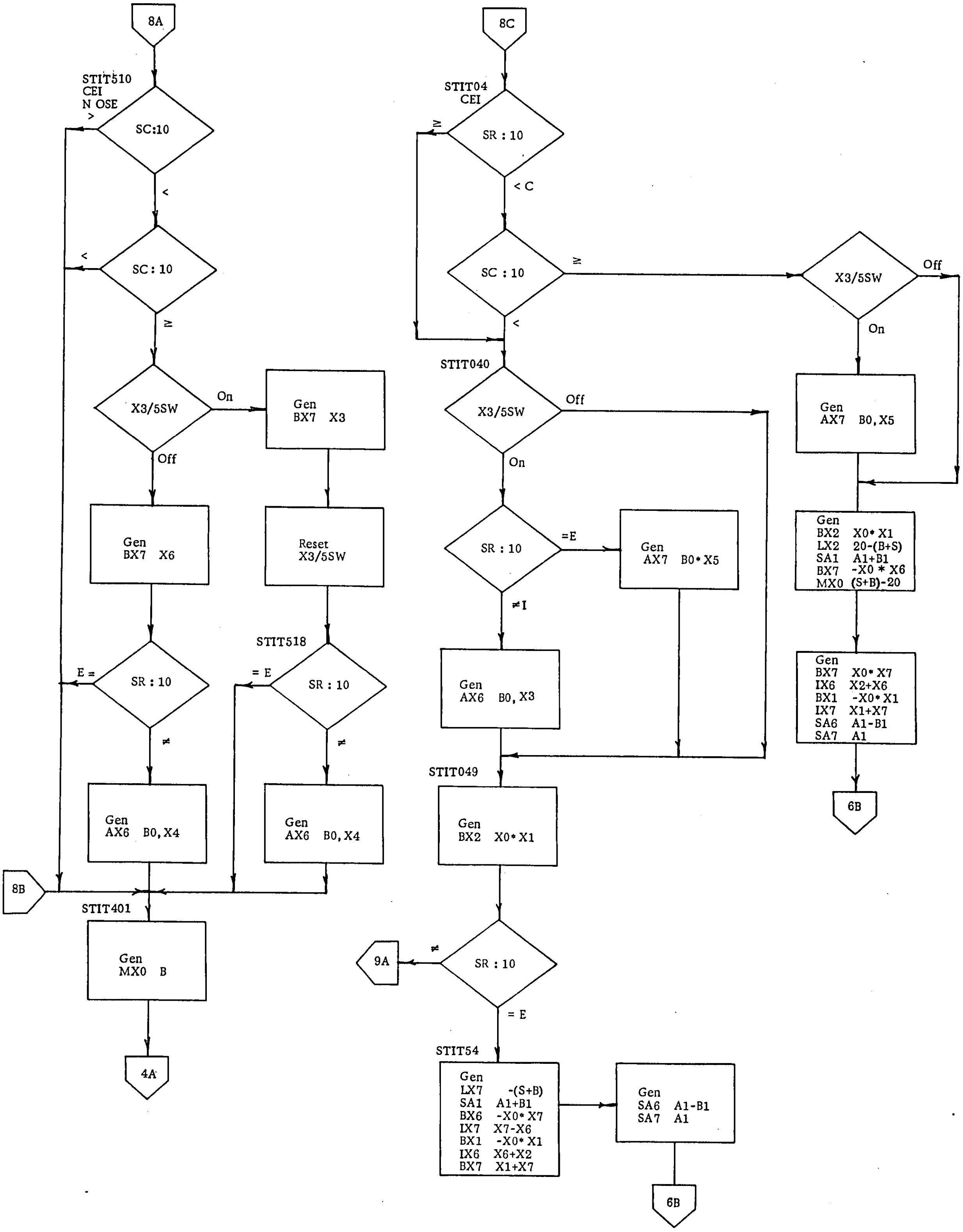


Figure 3-78. GENSTO Flowchart (30 of 44)

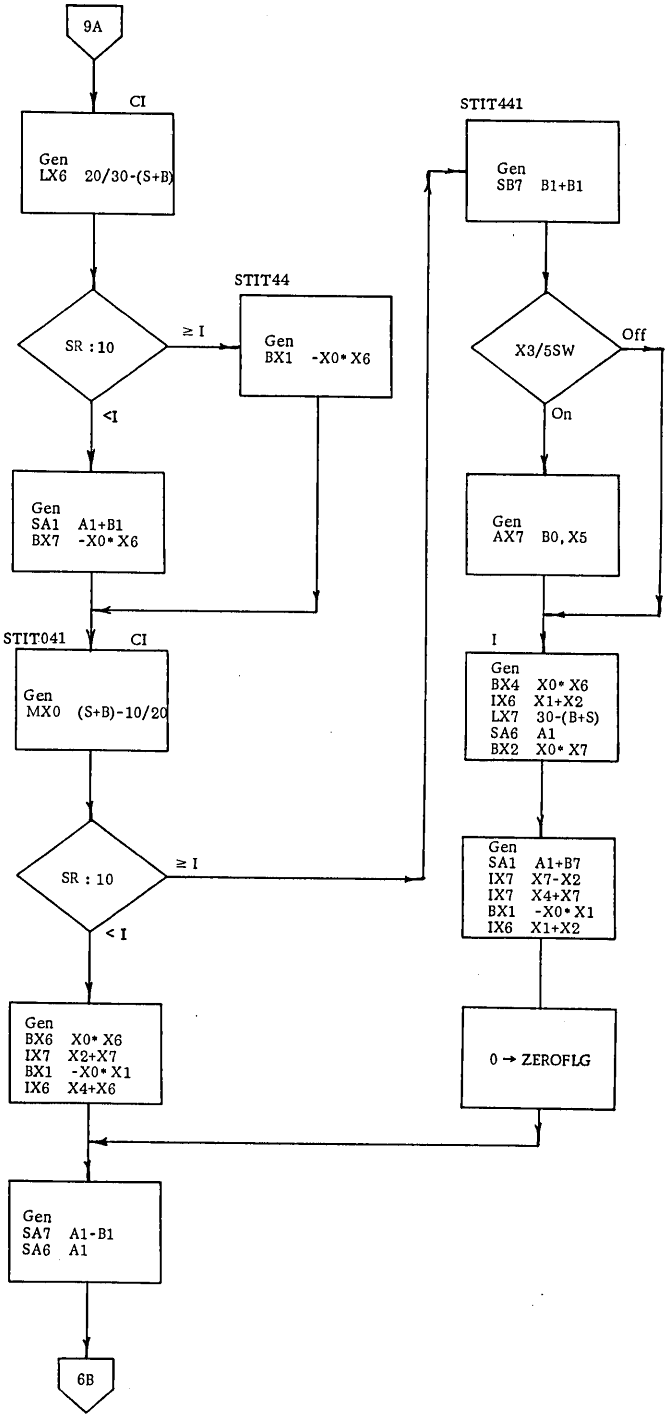


Figure 3-78. GENSTO Flowchart (31 of 44)

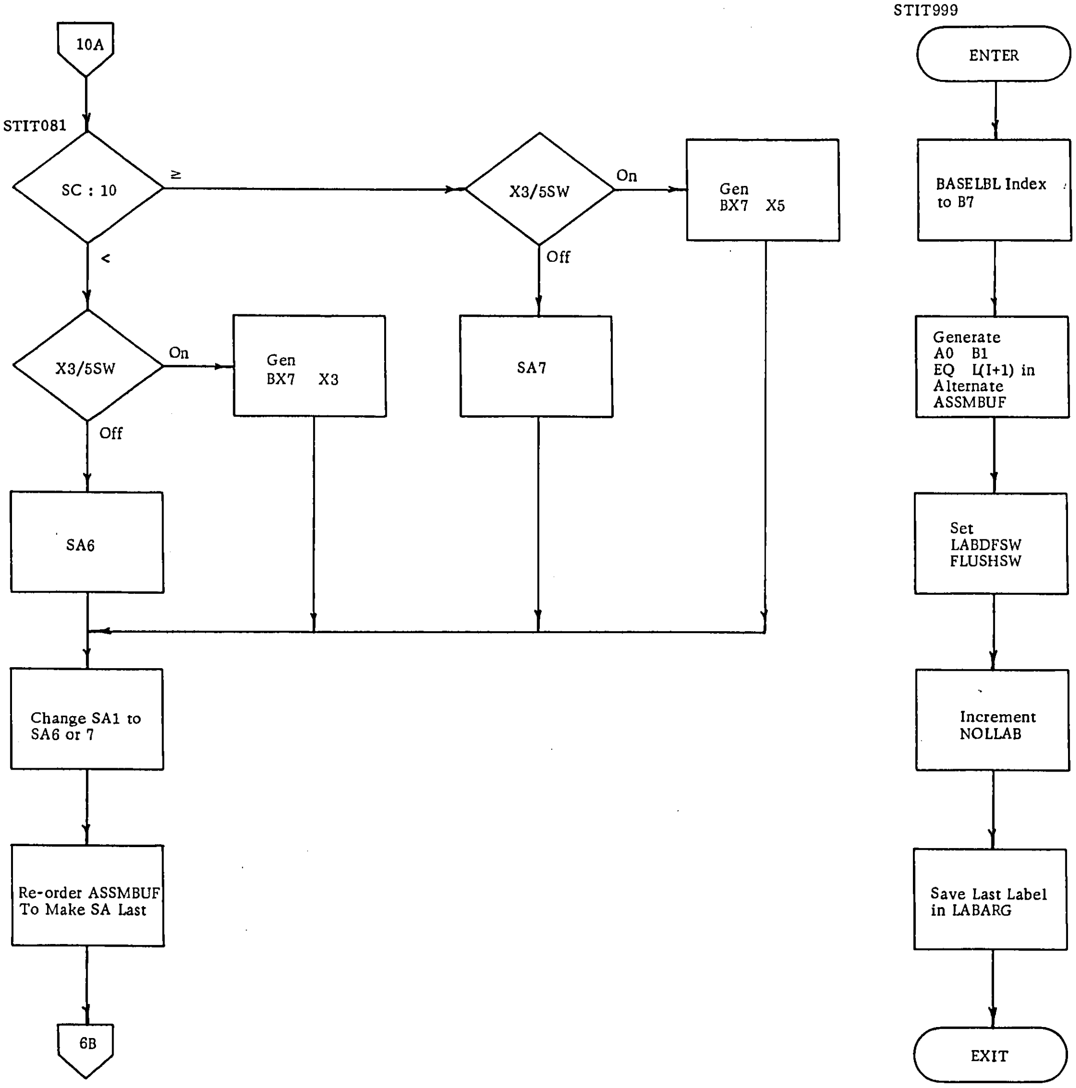


Figure 3-78. GENSTO Flowchart (32 of 44)

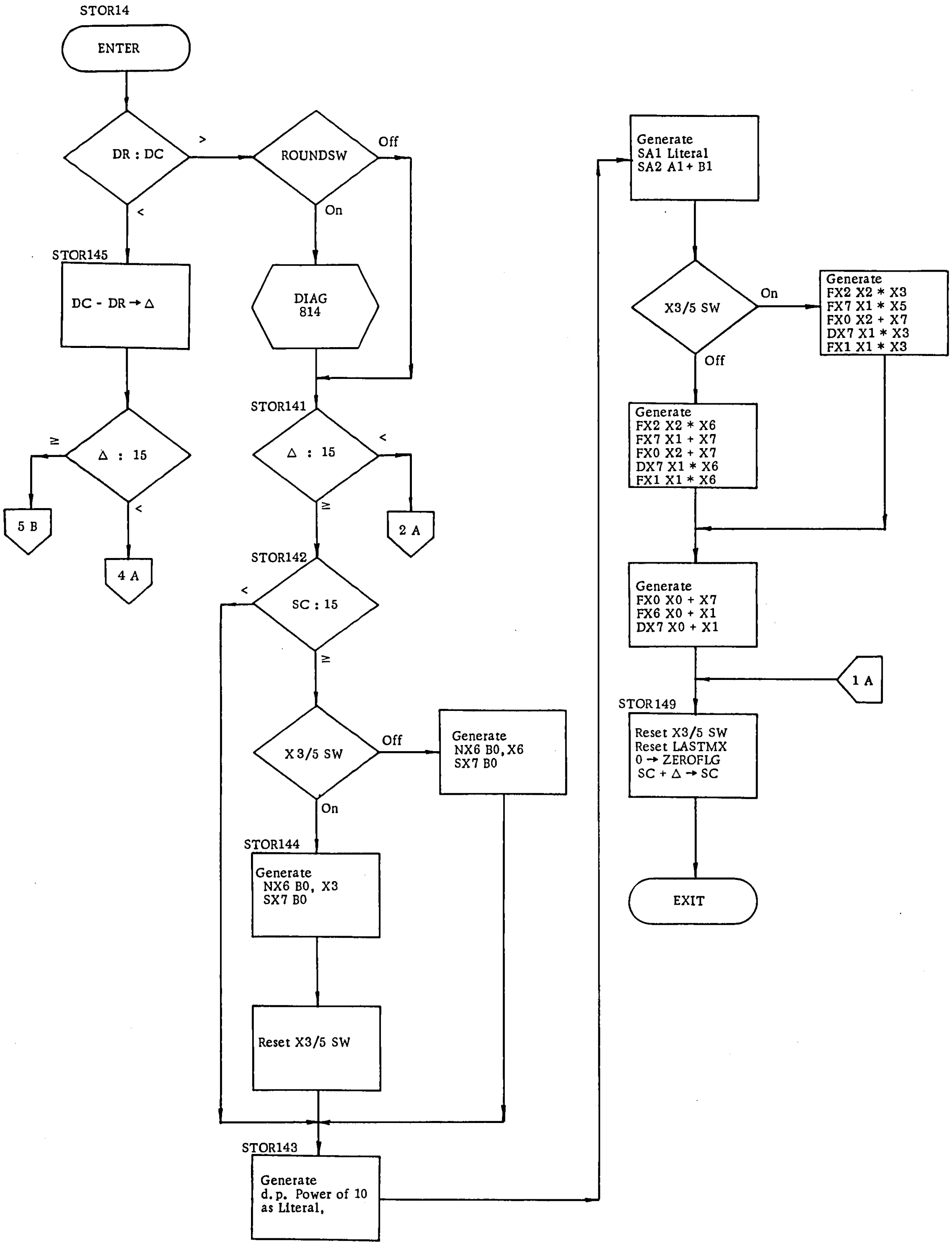


Figure 3-78. GENSTO Flowchart (33 of 44)

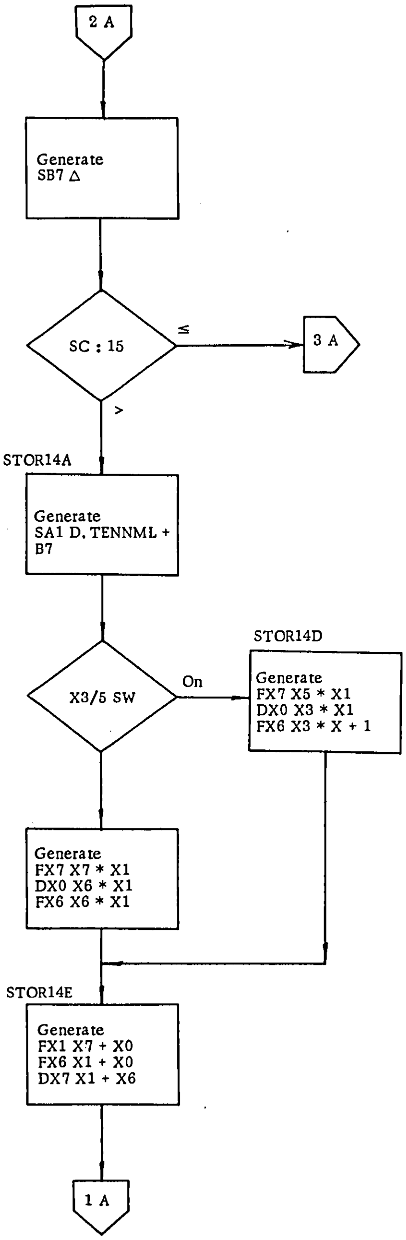


Figure 3-78. GENSTO Flowchart (34 of 44)

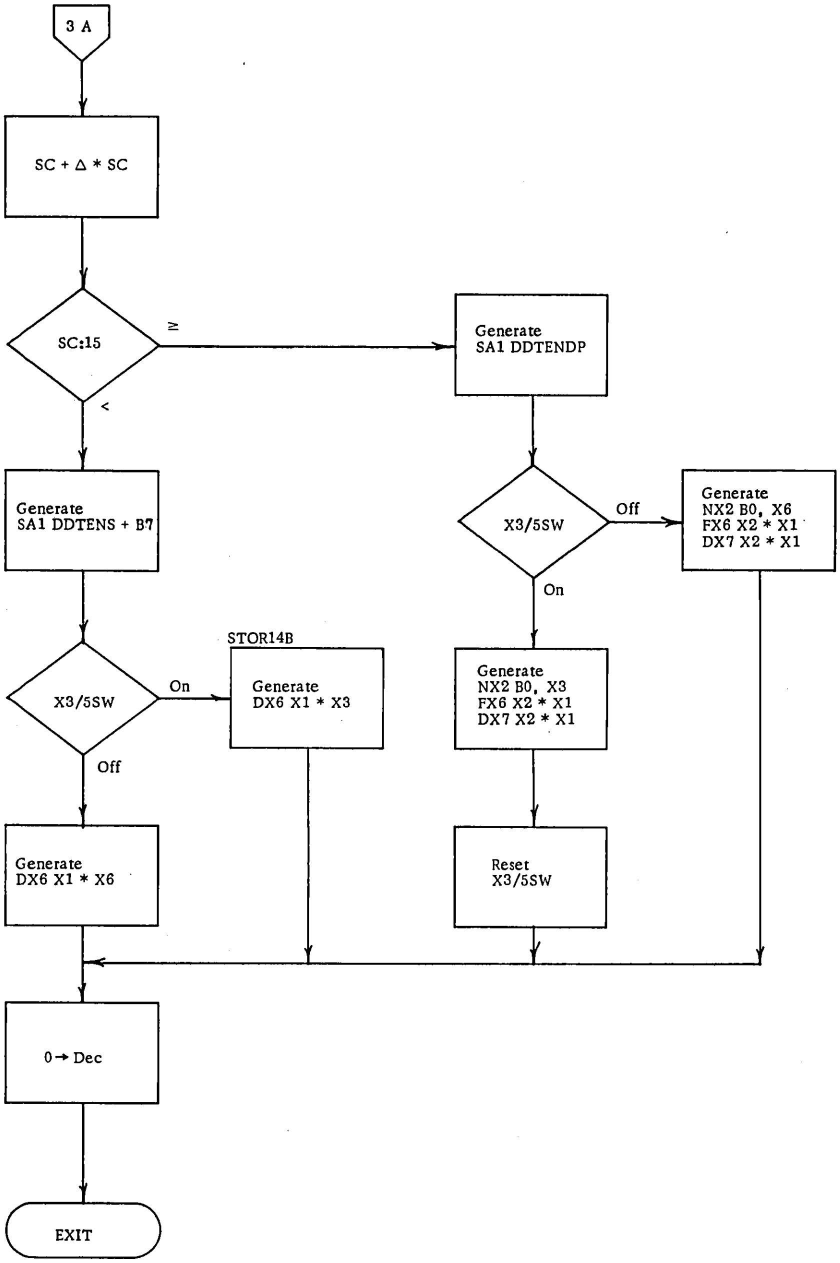


Figure 3-78. GENSTO Flowchart (35 of 44)

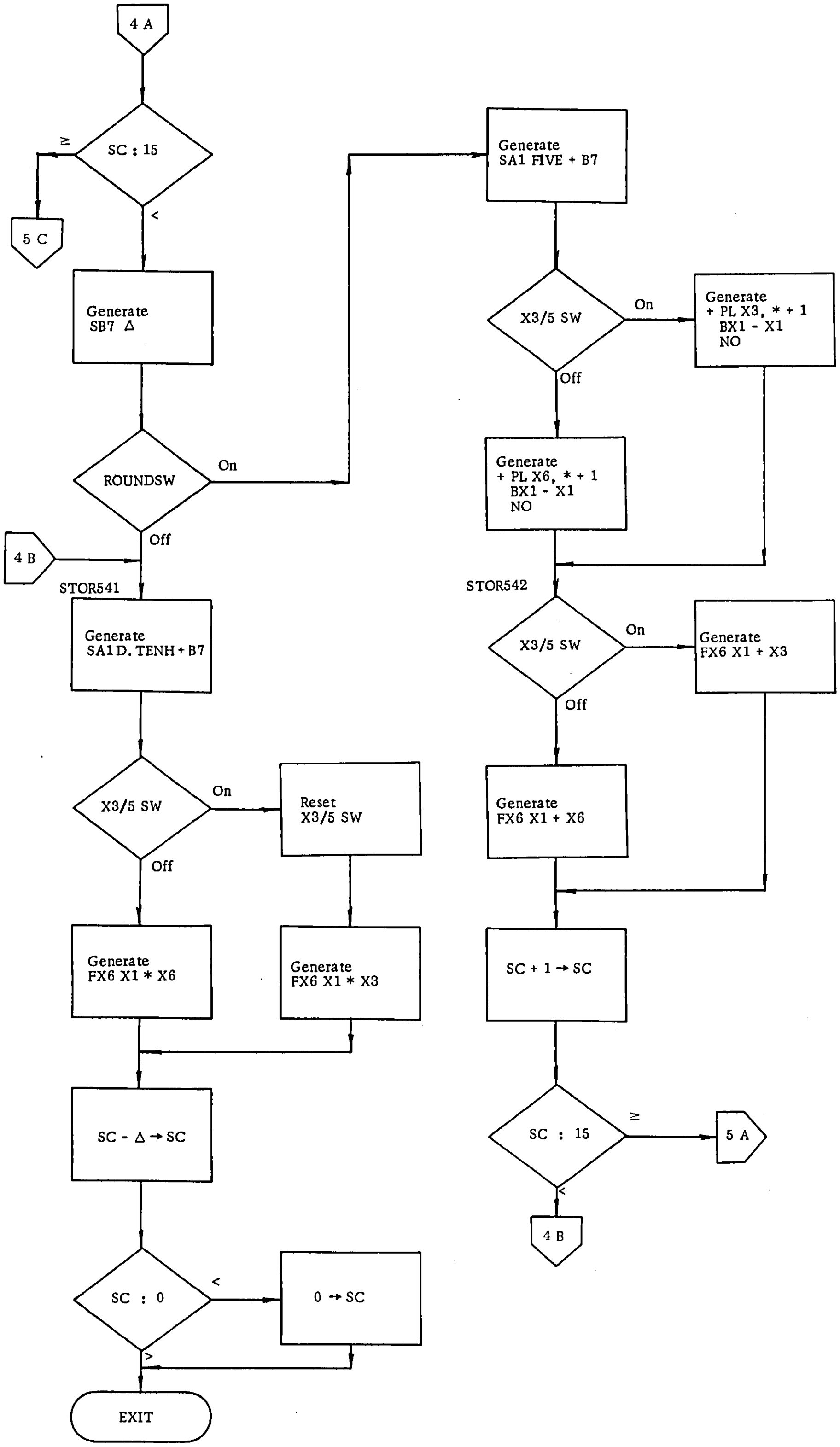


Figure 3-78. GENSTO Flowchart (36 of 44)

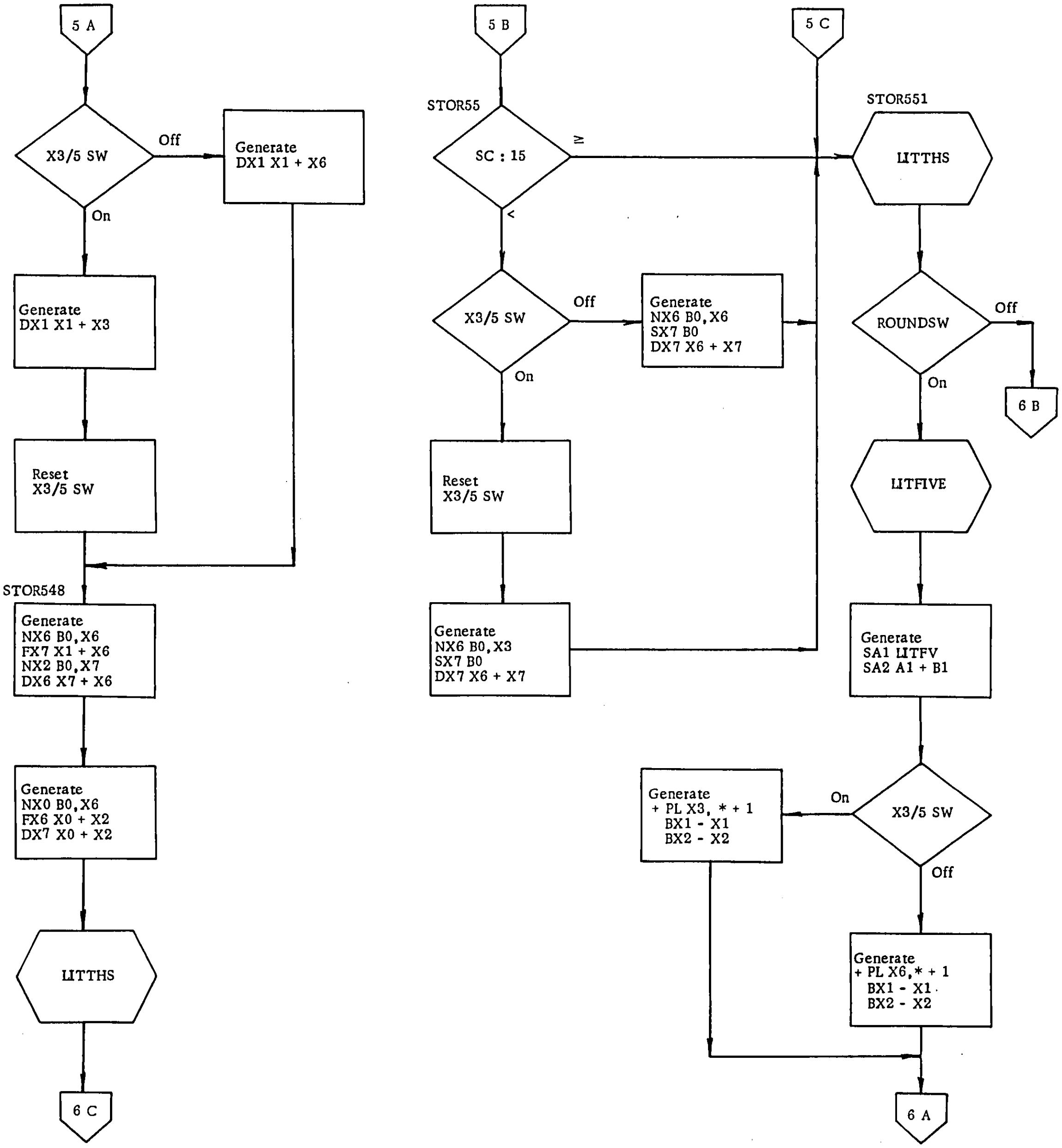


Figure 3-78. GENSTO Flowchart (37 of 44)

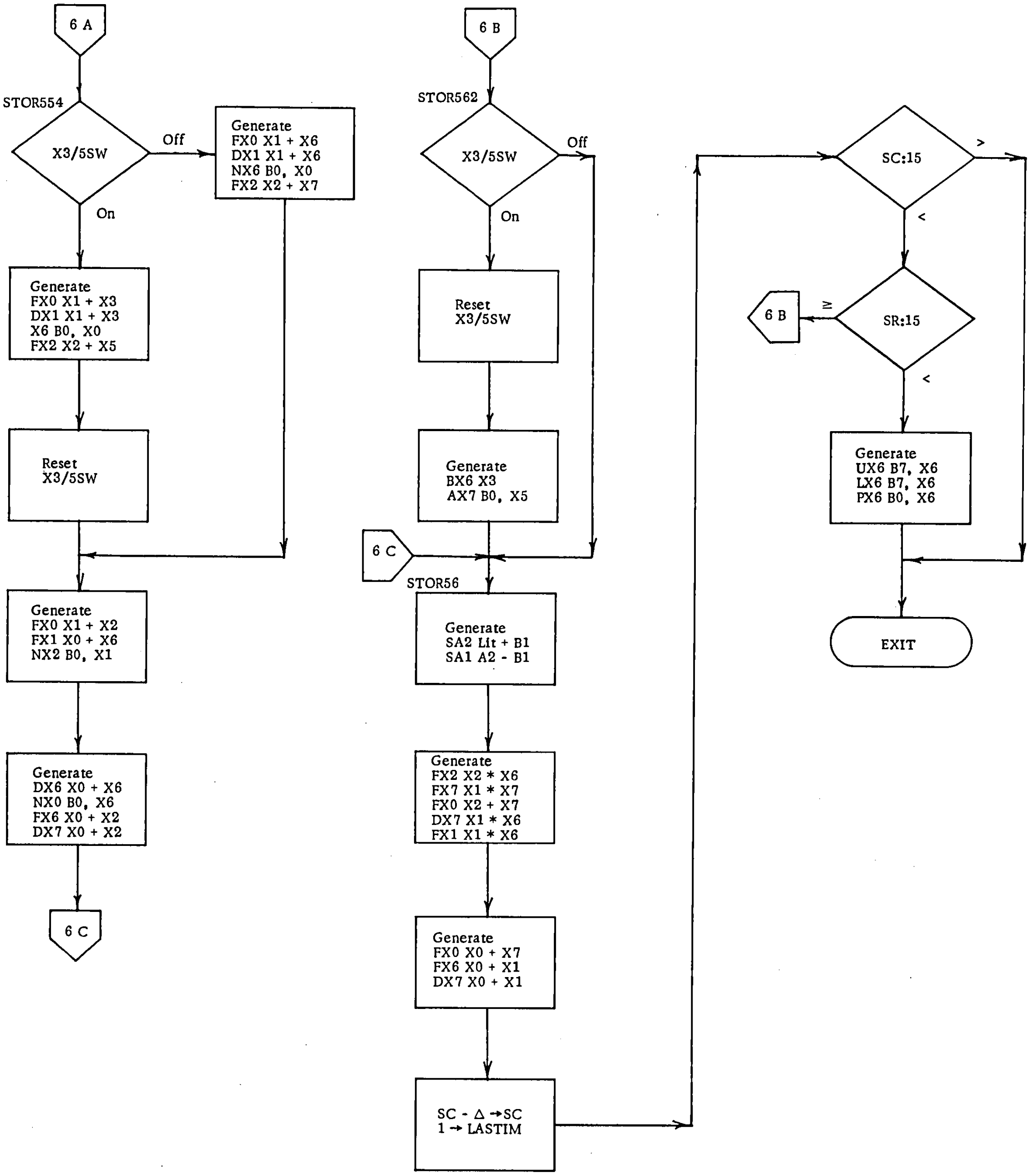


Figure 3-78. GENSTO Flowchart (38 of 44)

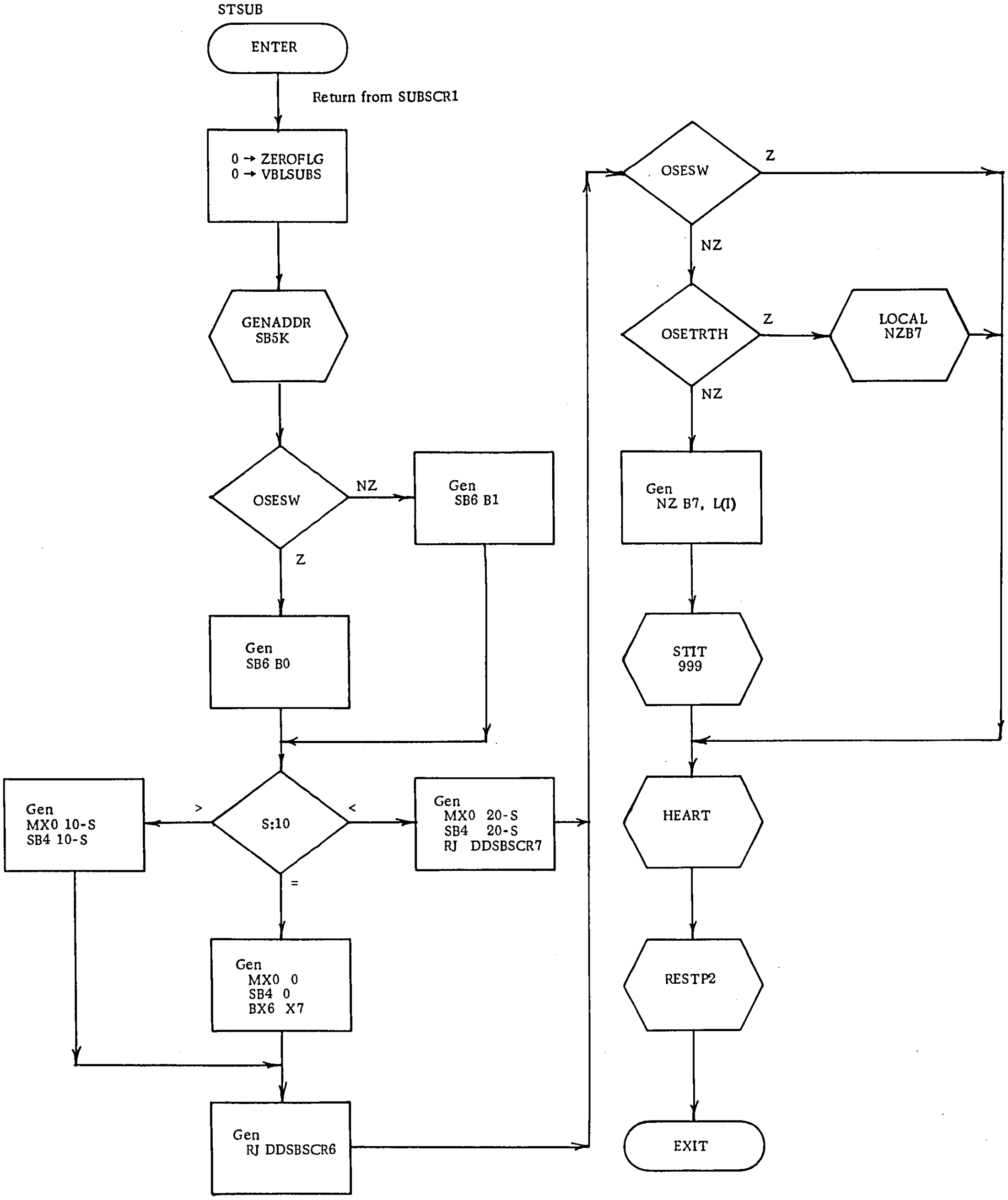


Figure 3-78. GENSTO Flowchart (39 of 44)

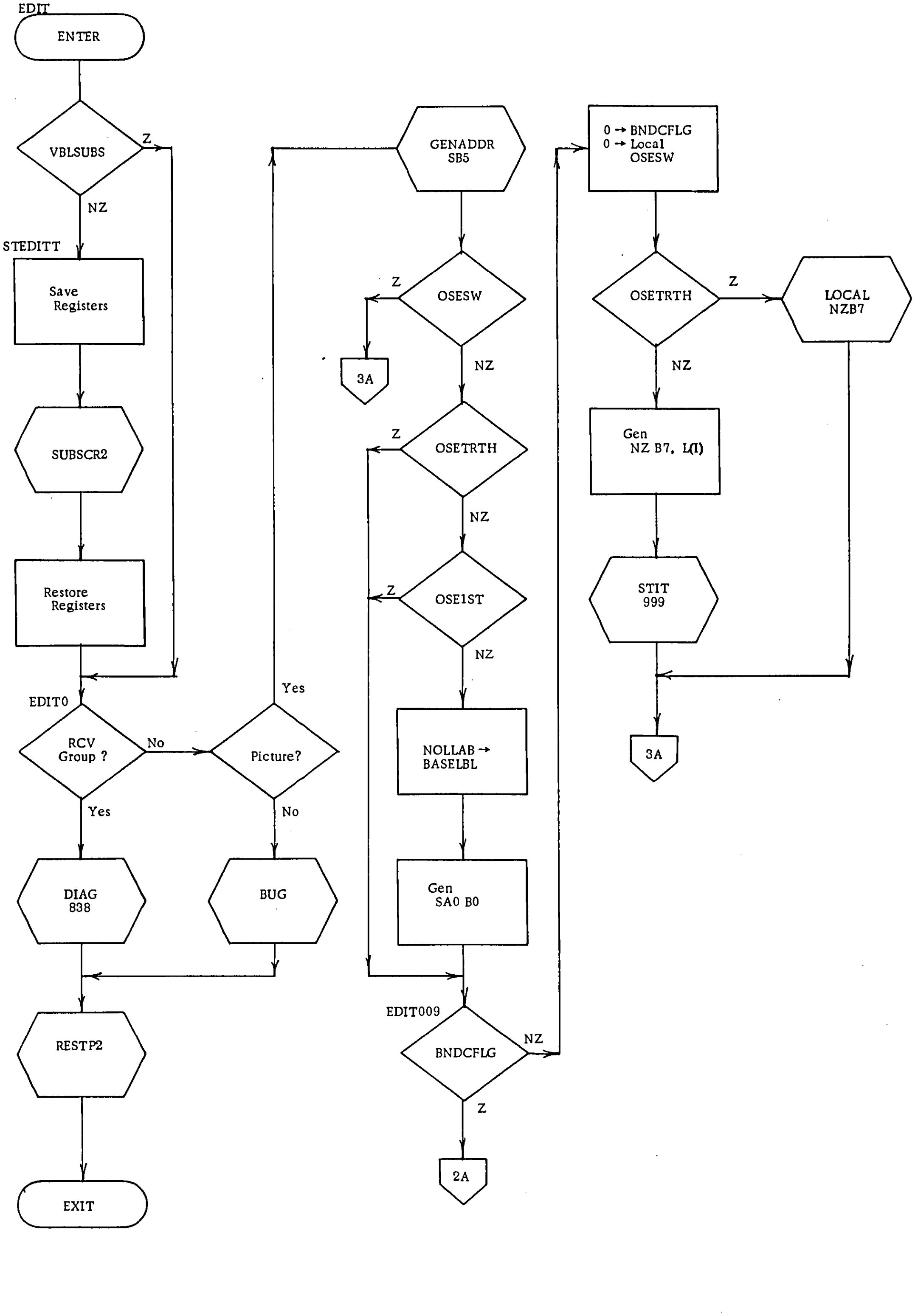


Figure 3-78. GENSTO Flowchart (40 of 44)

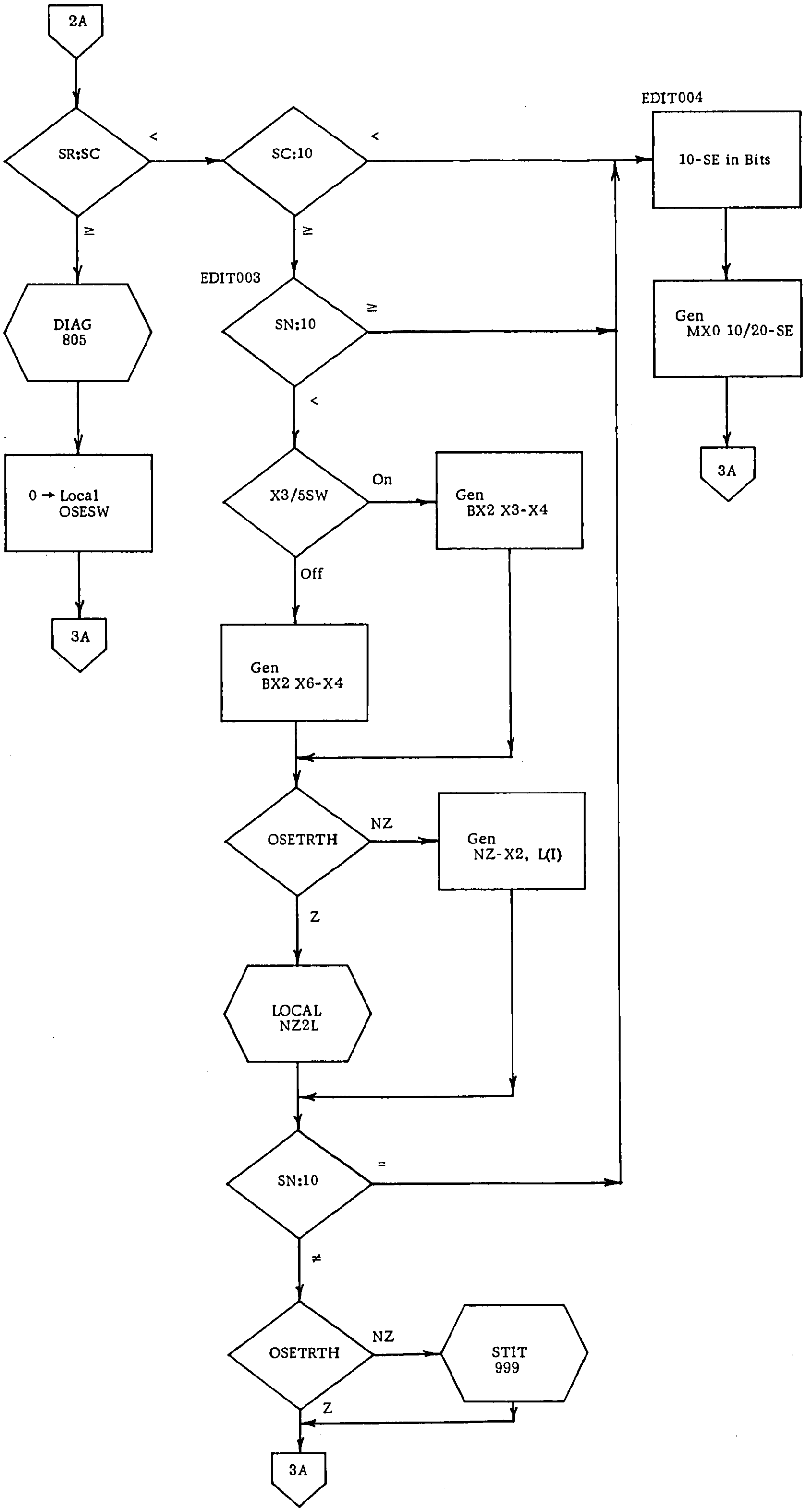


Figure 3-78. GENSTO Flowchart (41 of 44)

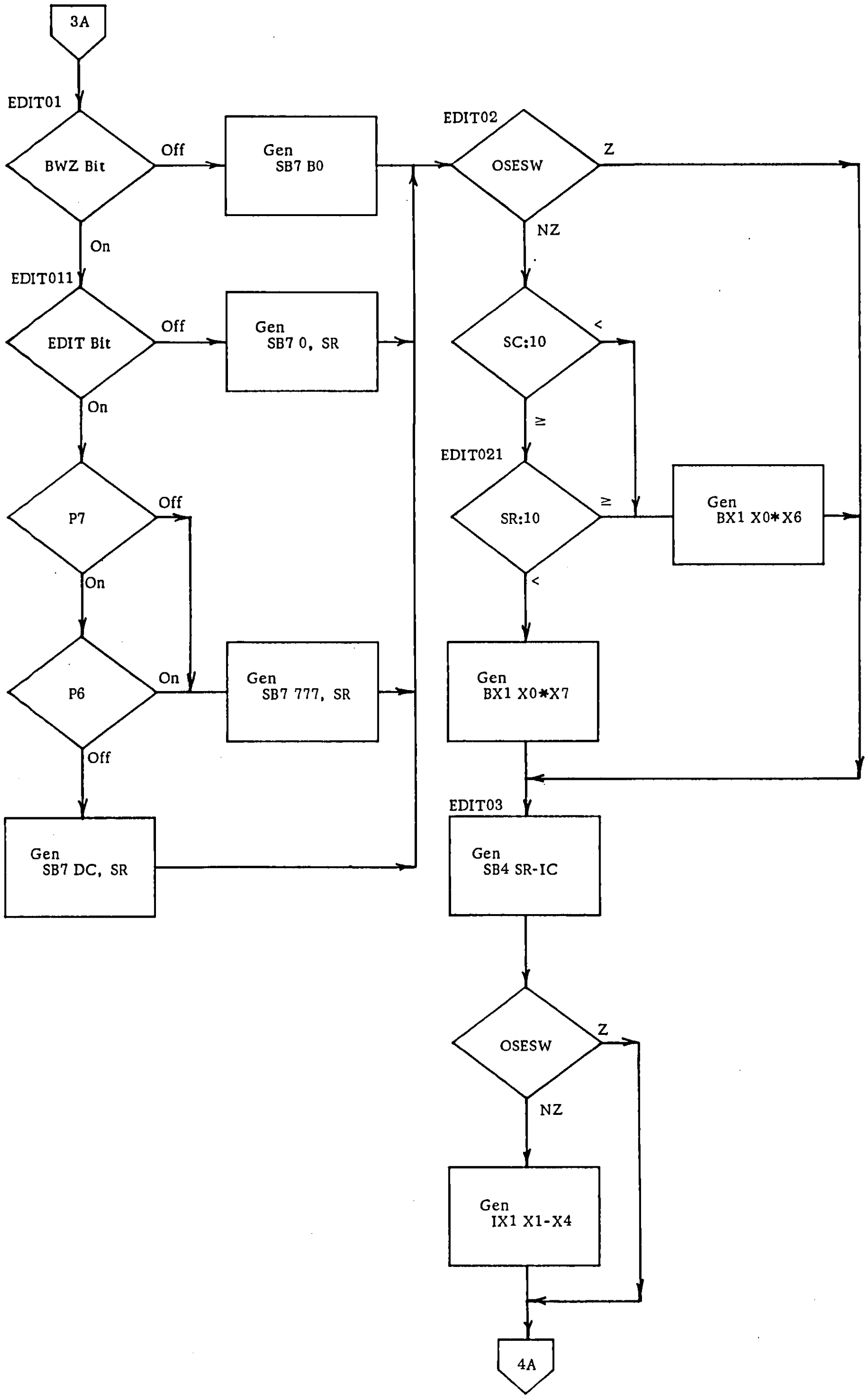


Figure 3-78. GENSTO Flowchart (42 of 44)

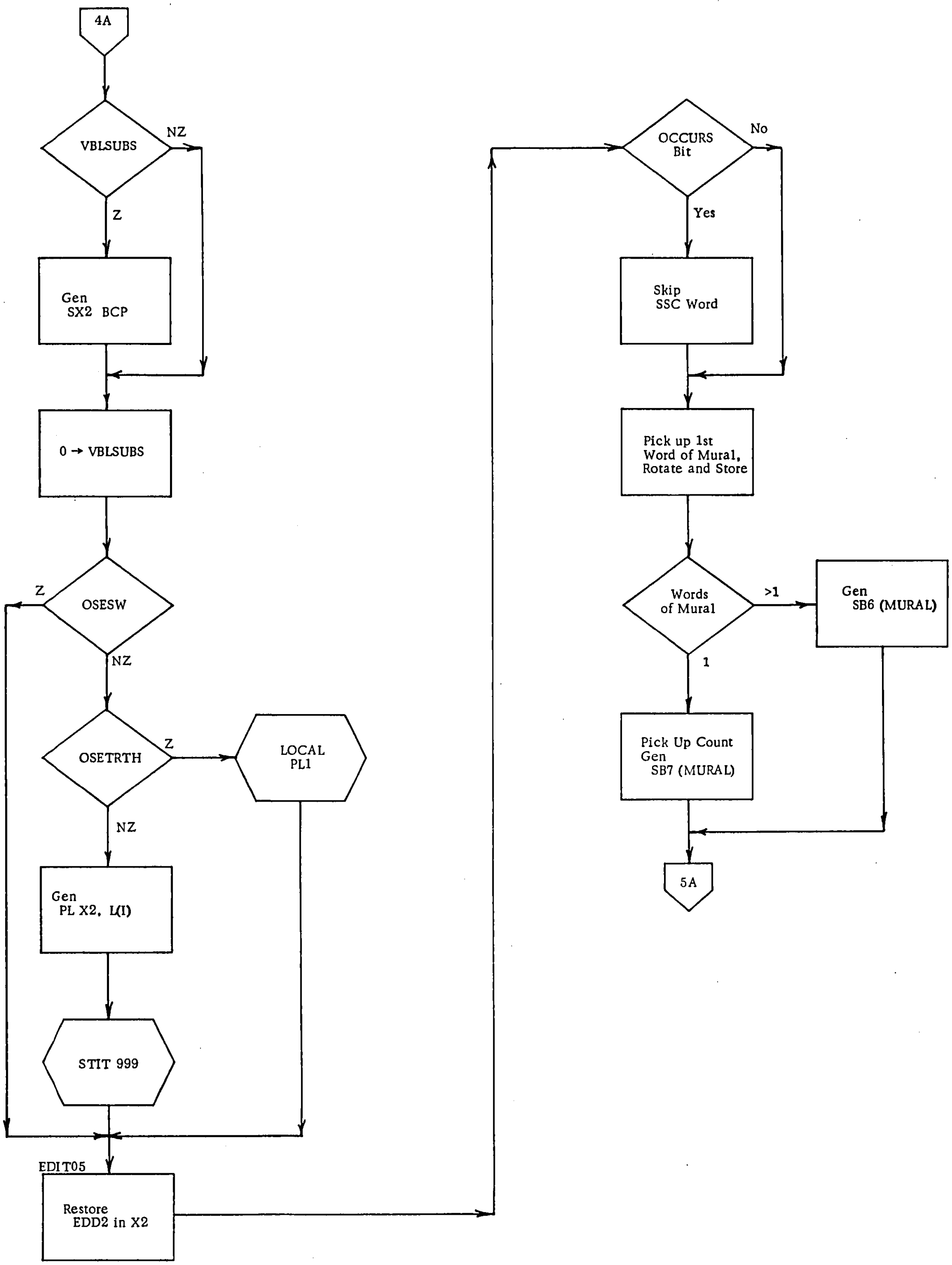


Figure 3-78. GENSTO Flowchart (43 of 44)

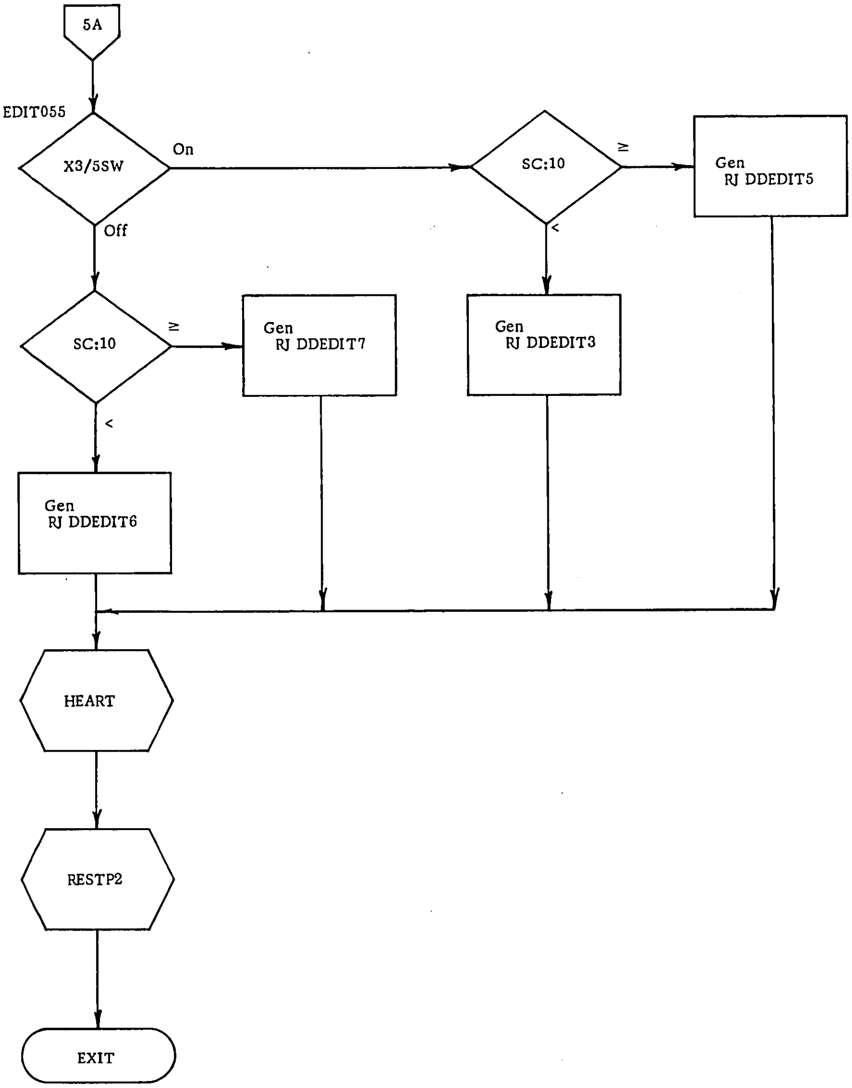


Figure 3-78. GENSTO Flowchart (44 of 44)

unsigned but the current result is signed (when any operand is signed or COMPUTATIONAL-1 or -2, or when the operator is subtract), code is generated to give the absolute value and a message is written. Any code generated thus far is now assembled. If the receiving field is EDITED, flow goes to EDIT. If not, GENADDR is invoked to generate code to load the first word of core that contains the receiving field and control passes to STORIT.

If the receiving field is COMPUTATIONAL-1, the current result is converted to COMPUTATIONAL-1 if necessary. Decimals are then aligned by multiplying by the appropriate power of 10 taken from an object time table; this follows, if necessary, a floating add of an appropriate 5 to round. Any specified and germane ON-SIZE ERROR test is made by comparing the current result against the appropriate power of 10 table entry by subtraction. Code is then generated to store the item. If the multiplication by a power of 10 requires a double-precision operand, the power is generated as a literal in a simple-minded way.

When the receiving field is COMPUTATIONAL-2, the conversion occurs, if necessary, of the current result is conversion to binary, followed by a multiplication by the appropriate power of 10 taken from the object time tables. Conversions from COMPUTATIONAL-2 provide point alignment inherently.

STORIT

Purpose

To generate code to insert into a DPC receiving field a DPC item. Invoked on a DPC store.

Operation

If the receiving item is subscripted, SUBSCRI is invoked. STORIT then generates code to accomplish the store depending upon the word orientation of the field. ON-SIZE ERROR testing is also accomplished.

STSUB

Purpose

To generate a call to the object-time subscripted store routines.

Invoked

By SUBSCR when a variable subscript appears on a receiving field.

Operation

Code is generated to call D. SUBCC6 or D. SUBSCC7. Code is then generated to test B7 if ON-SIZE ERROR is specified.

EDIT

Purpose

To generate a call to the object time editing routines

Invoked

In GENSTO when the receiving field is edited.

Operation

If the receiving field is subscripted, SUBSCR2 is called to generate the subscript offset, etc. Code to load the parameters D. EDIT requires is generated, interspersed with ON-SIZE ERROR testing, if called for. A call to D. EDIT is generated.

STOCOR

Purpose

To generate HOLD in temporary core (D. TEMP) rather than in registers.

Invoked

Prior to the first call to ART in GENSTO.

Operation

The front of the buffer (ASSMBUF) is searched for BX3 etc., and the instructions are replaced with SA6, etc.

ARITHMETIC

Consider a triad of an operator operating on a previous result and operand, i.e., X1, X2 ⊙ X6, X7. The store operator is not considered here.

1. BCD vs. BCD

   a. SIZE constraints must be met (see below).

   b. If the operator is multiply, divide, or exponentiate, code is generated to call, at object time, the BCD to binary subroutine, and both modes are set accordingly.

2. BDC vs. non-BCD

   a. Code is generated to convert the BCD item to binary, and the mode is set to binary.

3. Binary vs. binary (single precision)

   a. If the operator is divide, code is generated to normalize both operands, and mode is set accordingly. The result is then unnormalized with decimal set per ERS.

   b. If the operator is exponentiate, code is generated to normalize both operands and to multiply both by the appropriate powers of 10 to convert them to true floating point. The result mode is set to floating point (i.e., COMPUTATIONAL-2).

4. Floating point vs. BCD, binary

   a. The result mode is floating point.

   b. Code is generated to convert the non-floating-point operand to floating point, i.e., convert to binary, normalize, multiply by the appropriate power of 10.

5. Floating point vs. double precision

   a. The result is single precision, true floating point.

   b. Code is generated to scale the double-precision operand. The high-order part of the result is then used in the computation.

Size Constraints

1. Composite size is defined, by DOD, as "that data item resulting from the superimposition of all operands ... aligned on their decimal points ..." Thus, for ADD, SUBTRACT, receiving field:

$$c_i = \max \left[C_{i-1}, (S_i - d_i) \max (d_i, d_{i-1}) \right]$$

   a. A composite decimal (max d_i) is computed for receiving fields, a COMPUTE statement, on the climb of the tree. This is used on divide to fix a normalized result.

   b. During descent of the tree, on ADD and SUBTRACT, the composite size is compared to 18, and a friendly message is written when it is exceeded.

   c. During stores following a MULTIPLY or DIVIDE, the composite size of receiving fields is computed and checked against 18.

2. A pseudo-accumulator of sensibly infinite size is required by the COBOL language. Mode changes are generated to accomplish this.

   a. Size constraints are not germane when the mode is floating point, i.e., any operand is COMPUTATIONAL-2, or any operator is exponentiate, or a divide appears in an IF conditional.

   b. The 6600 COBOL implementation provides a pseudo-accumulator of 28 decimal positions. If the composite size of a computation exceeds this, a friendly message is written.

   c. During computation, the current composite size is maintained in the property word of the result.

      (1) Initially, and on every 10 succeeding ADD or SUBTRACT modes, 1 is added to the composite size.

      (2) On multiplication, the composite is the sum of the sizes of the operands.

      (3) On divide, the composite size depends also on the decimal positions retained because of the composite of the result fields.

   d. When the composite size resulting from an operation where the operands are BCD exceeds 10, code is generated to convert both operands to binary, and the result mode is so set.

e. When the composite size of a binary operation exceeds 14, the result mode is set to double precision, and double-precision arithmetic is invoked.

GENMOVE

Purpose

GENMOVE analizes the MOVE statement and either calls GENLOD and STORE to accomplish the move or generates the necessary code itself. (See Figure 3-79.)

Call

From PASS2 on the MOVE secondary node. Exits to SCALE08.

Routines Called

GENLOD
GENPREV
STORE
HEART
LIT02

Operation

The FROM leaf and the TO leaf are compared as to class. When both are numeric elementary items, and the size of FROM is less than or equal to the size of TO (i.e., no padding is required), and the decimals are equal (i.e., no shifting), and both are COMPUTATIONAL (i.e., no conversion), and HOLDSW is off (we knew all of the TOs), and TO is not BLANK WHEN ZERO, and TO is not EDITED, SHRTLOD is set to generate an abbreviated load. GENLOD and STORE are then called to accomplish the MOVE.

An AN move that is short is also accomplished by GENLOD-STORE. Short is less than 20 characters. Blanks are loaded into X4 if padding is necessary. The decimal alignment mechanism is used to provide left justification if needed.

Long moves (unsubscripted) cause to be generated in-line a prefix, loop, and suffix. There are 11 possible prefixes, 5 loops, and 8 suffixes, designed to accomplish the moves in an optimum manner considering possible word orientations. The criteria in choosing the code skeletons include:

SF, ST size of the FROM and TO items
BF, BT beginning character positions
NF, NT = INT ((S + B - 10)/10)
RF, RT = REM ((S + B - 10)/10)

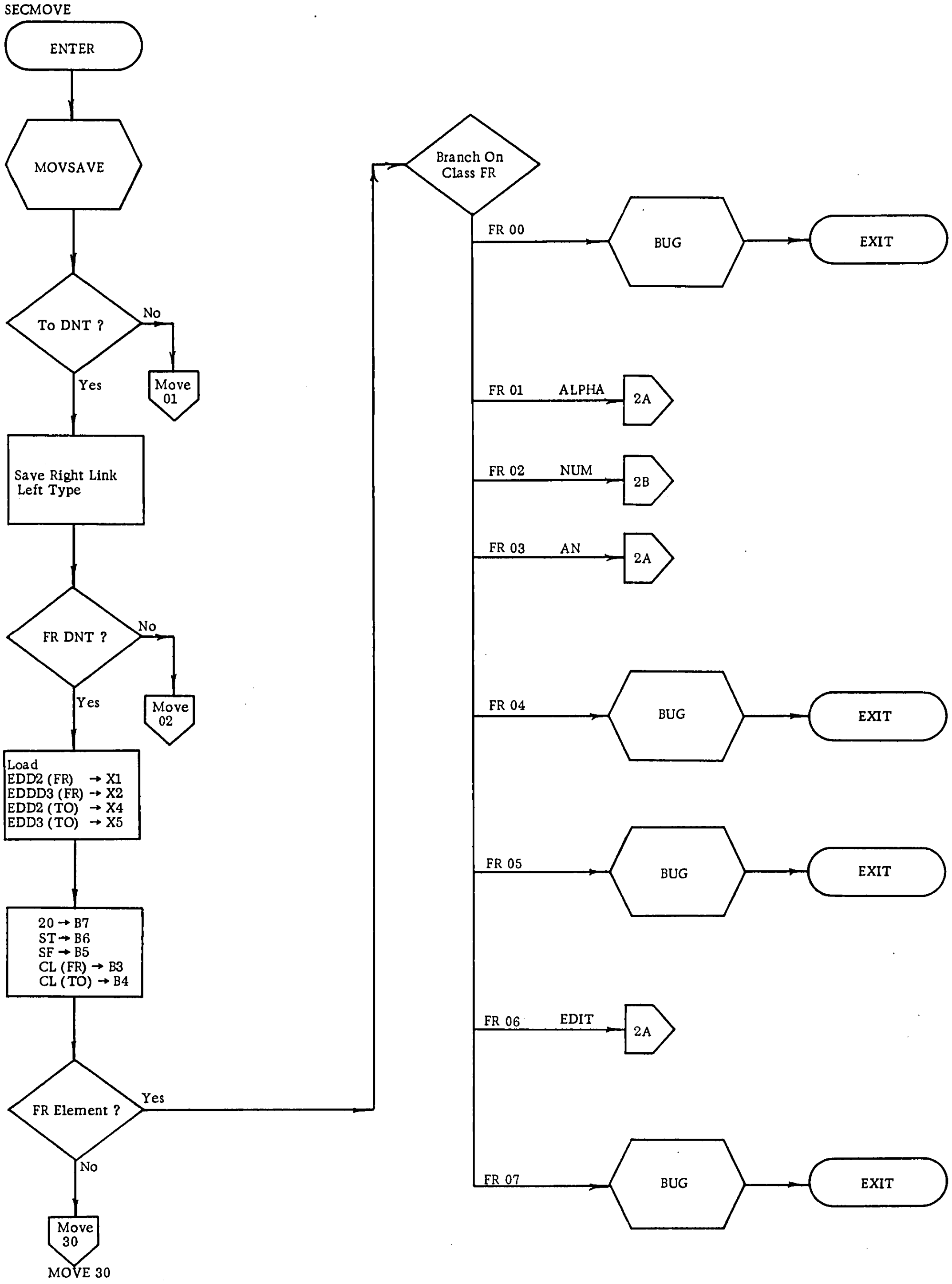


Figure 3-79. GENMOVE Flowchart (1 of 5)

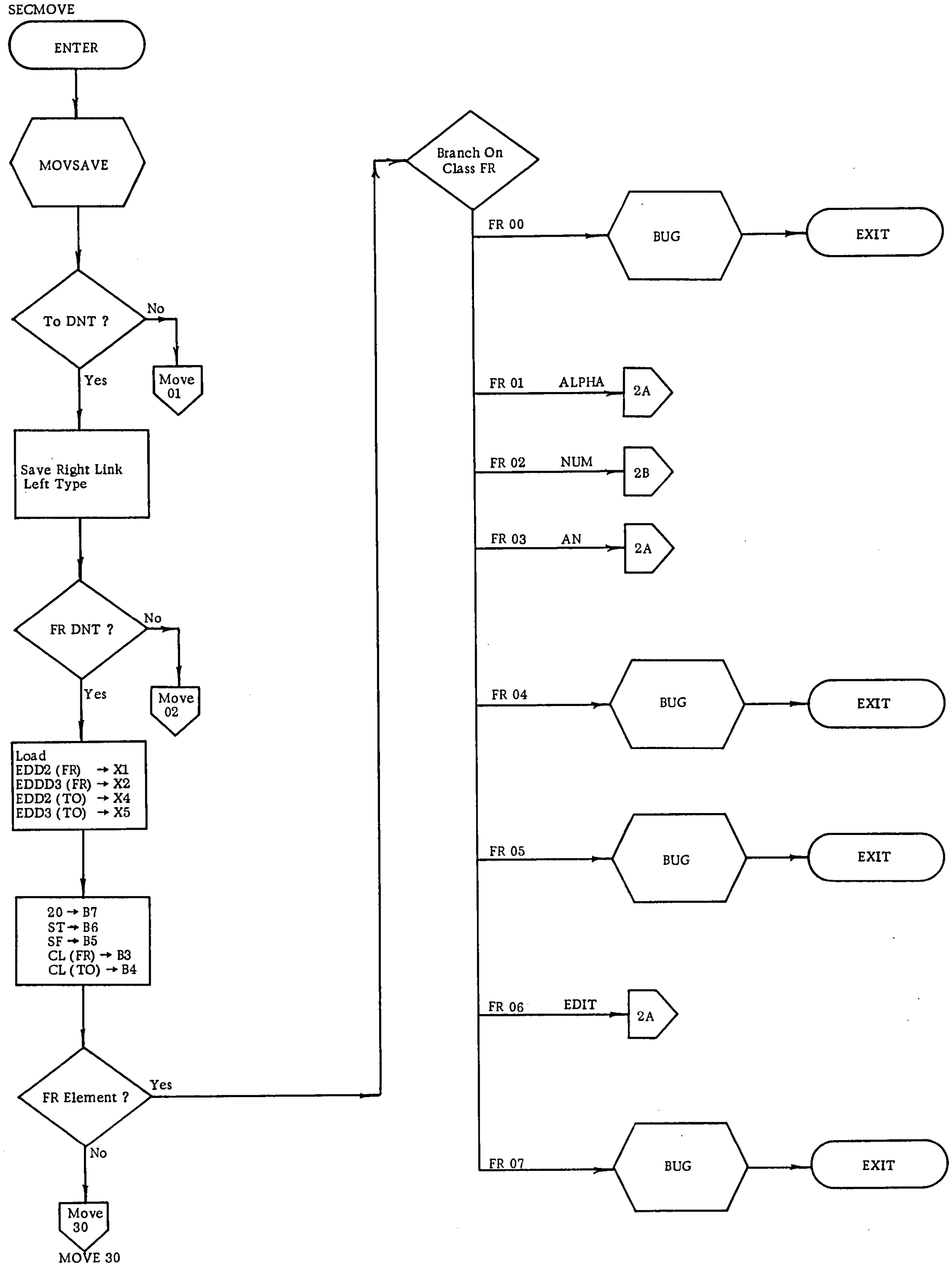


Figure 3-79. GENMOVE Flowchart (1 of 5)

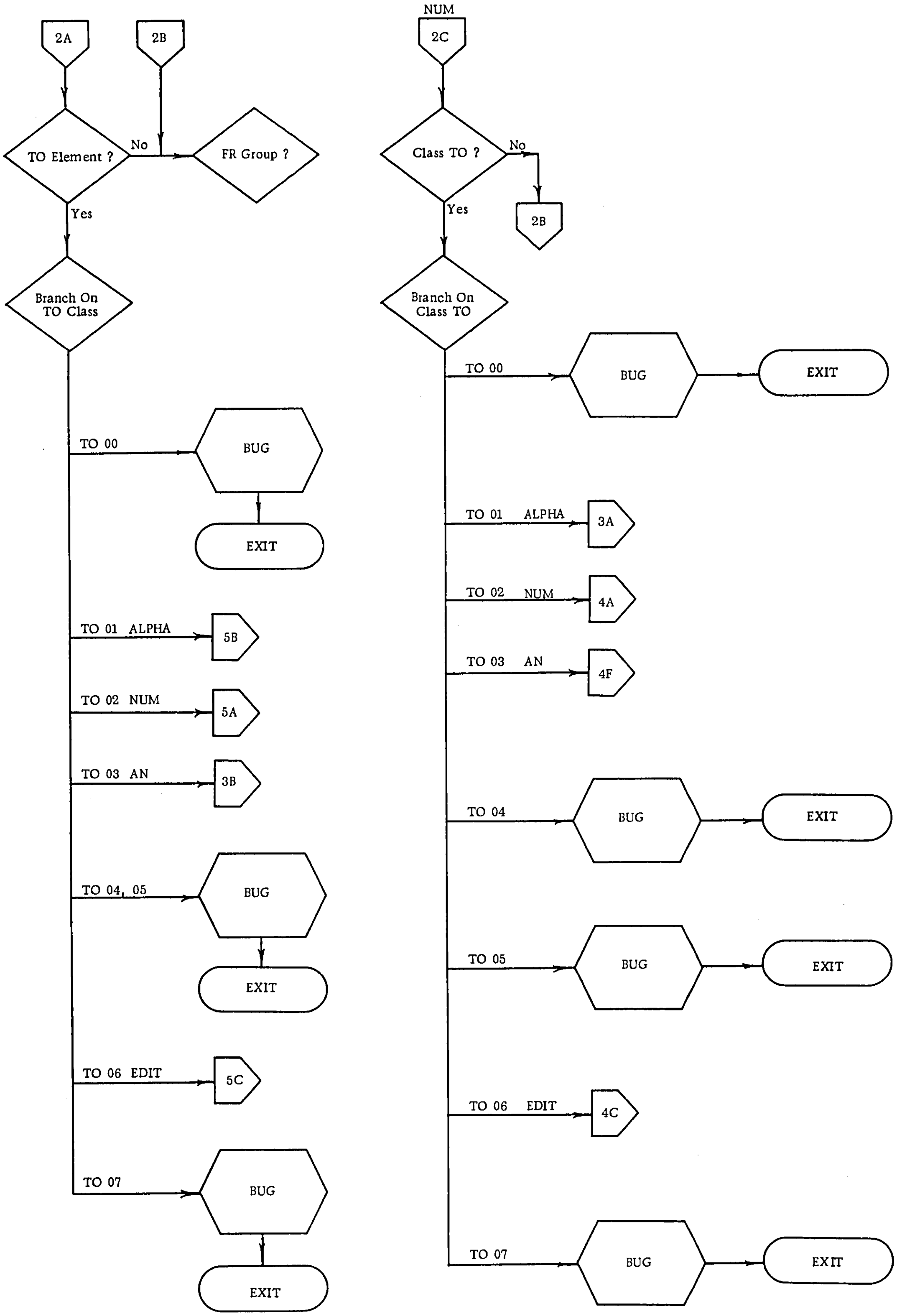


Figure 3-79. GENMOVE Flowchart (2 of 5)

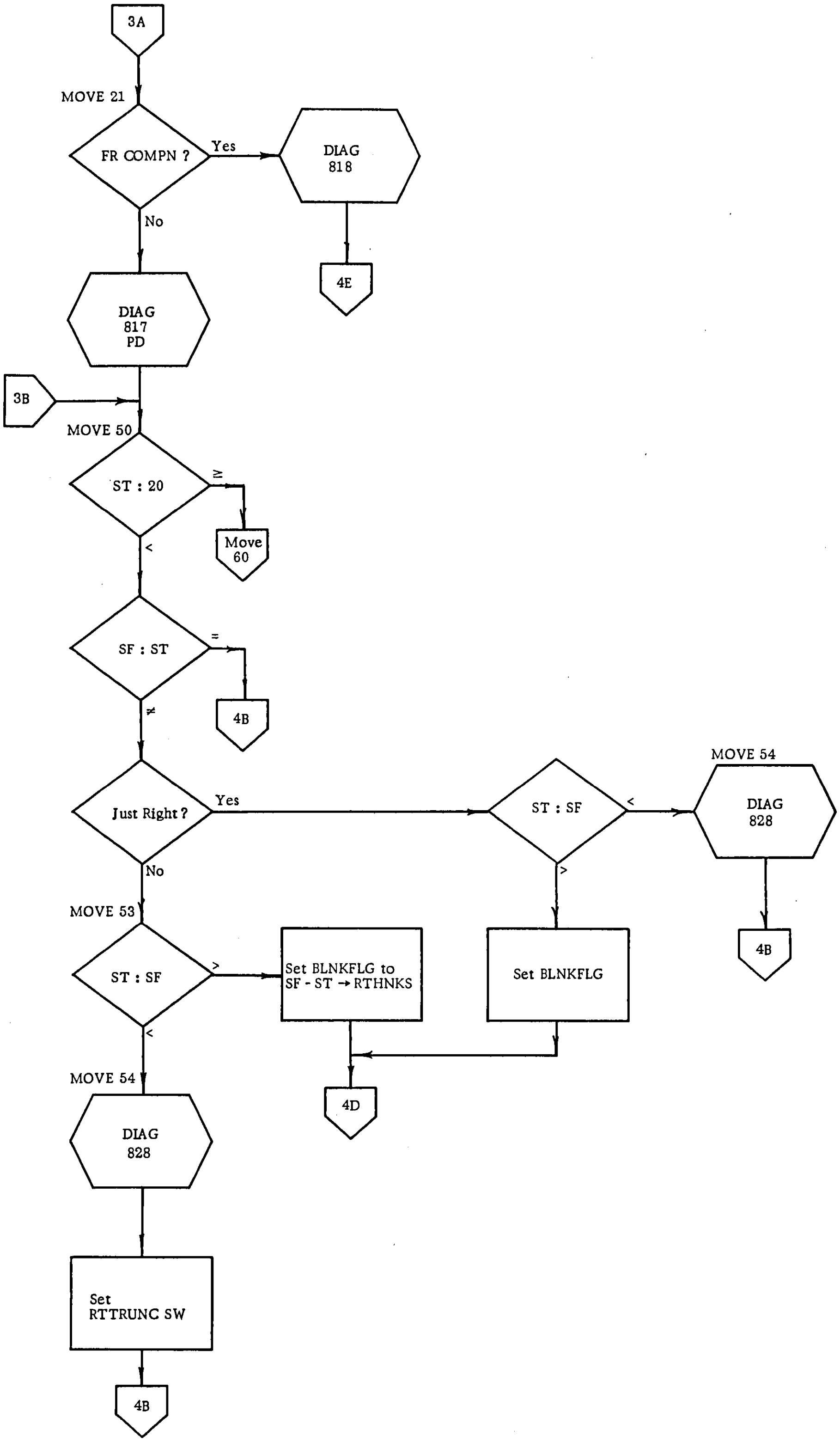


Figure 3-79. GENMOVE Flowchart (3 of 5)

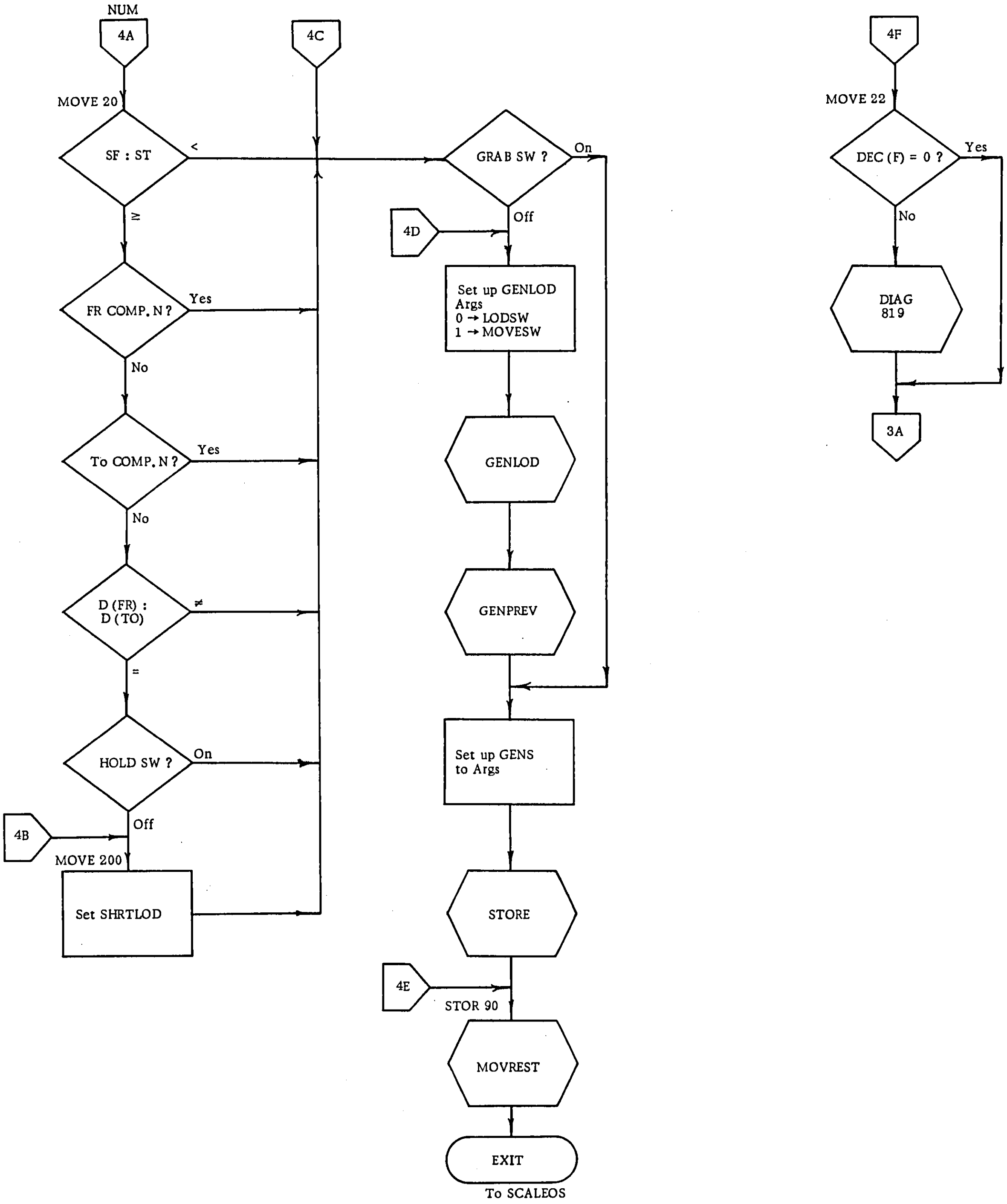


Figure 3-79. GENMOVE Flowchart (4 of 5)

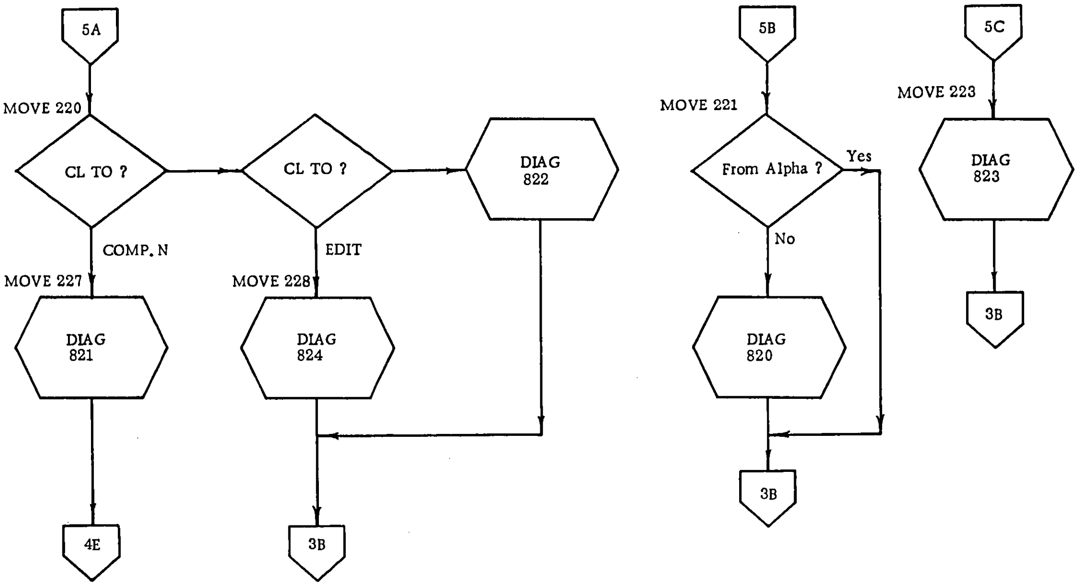


Figure 3-79. GENMOVE Flowchart (5 of 5)

Where the sizes of fields are mismatched and:

1. SF > ST, and TO is not JUSTIFIED RIGHT, ST is set to SF, and a message is written by the compiler.

2. SF > ST and TO is JUSTIFIED RIGHT. SF - ST (bytes) is added to FROM, Location and the MOVE is generated.

3. SF < ST. A second move of ST - SF blanks is generated after the MOVE of the fields.

4. SF < ST, JUSTIFIED RIGHT. A move of ST - SF blanks is generated before the move, and TO (bytes) is incremented by ST - SF.

Where a long move contains an operand with a variable subscript, the library routine DDMOVIO is invoked via the library routine DDSUBMV.

If the FROM field is a procedure division literal and the move is short, LIT02 is called to generate the literal. Unless TO is EDITED or a right shift is indicated, LIT02, given the MOVE operator, accomplishes any conversions, zoning, shifting. On EDITED, etc., a fake add is presented to LIT02, and the manipulations of GENLOD, STORE come into play.

Long literals cause the generation of the literal followed by a long MOVE. LIT02 takes care of blank padding, etc.

Long moves of figurative constants cause generation of a load followed by a store loop.

GENARTH

Purpose

To generate code to accomplish arithmetic. (See Figure 3-80.)

Called

1. By PASS 2 on an arithmetic operator when descending a tree. Return is to SCALE08.

2. By PERFORM and conditionals as a subroutine.

Routines Called

GENLOD
HEART

Operation

GENARTH operates upon a trial of an arithmetic operator and two operands, one of which is a current (or accumulator) result. The nature of the operands is given in the respective properties word, PROPLOD, PROPCUR. If either of these is a literal, LIT02 is asked to generate a literal with properties matching the other.

Where the operator is add or subtract, and both operands are COMPUTATIONAL, SHFD is used to align the decimals, (i.e., to generate code to accomplish any shifting necessary). Where shifting OCCURS, complementation is used in transfers, when possible, to accomplish subtraction. Size is maintained to provide for carries. The add is then generated, in-line if single precision, or by subroutine (D. DADD) call. The two numeric fields consist of DPC characters, and the operation is performed directly in decimal in a nine's-complement manner. Single precision is done in-line, double precision by use of the DDDADD routine. The technique is to use a 60-bit binary addition and then to use masking and boolean operations to do both intra-word and inter-word carries. (This process is a well-known one in the literature.)

The operators other than add or subtract cause conversion, as needed, to COMPUTATIONAL-1, as does the presence of a COMPUTATIONAL-1 operand. Any point alignment for ADD, SUBTRACT is accomplished at SHFT by generation of multiplication by tabled powers of 10, or a generated literal if double precision is required.

With either operand COMPUTATIONAL-2 or the exponentiation, conversion to true floating point is generated.

Code is then generated to accomplish the operation, and PROPCUR is updated.

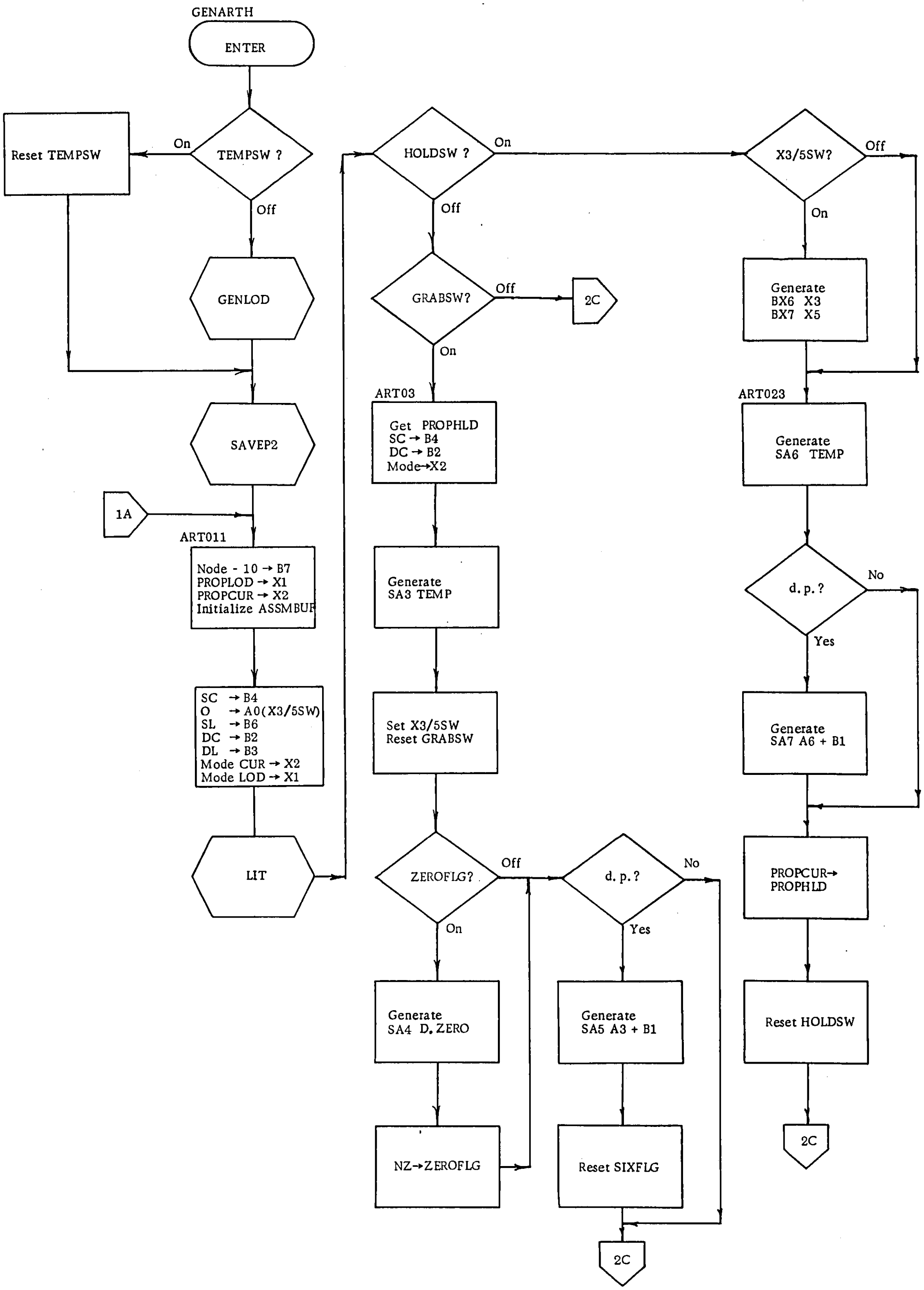


Figure 3-80. GENARTH Flowchart (1 of 33)

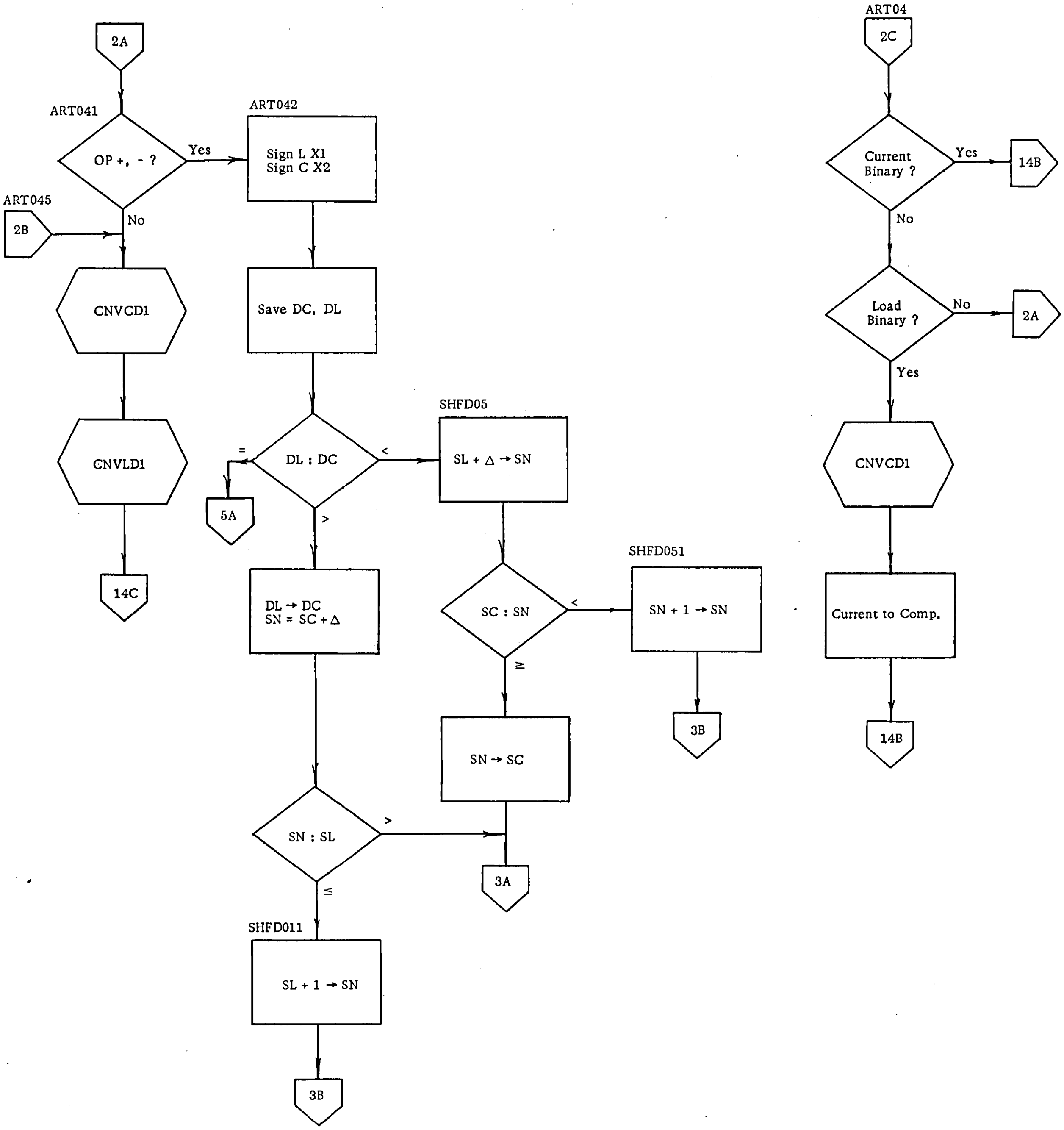


Figure 3-80. GENARTH Flowchart (2 of 33)

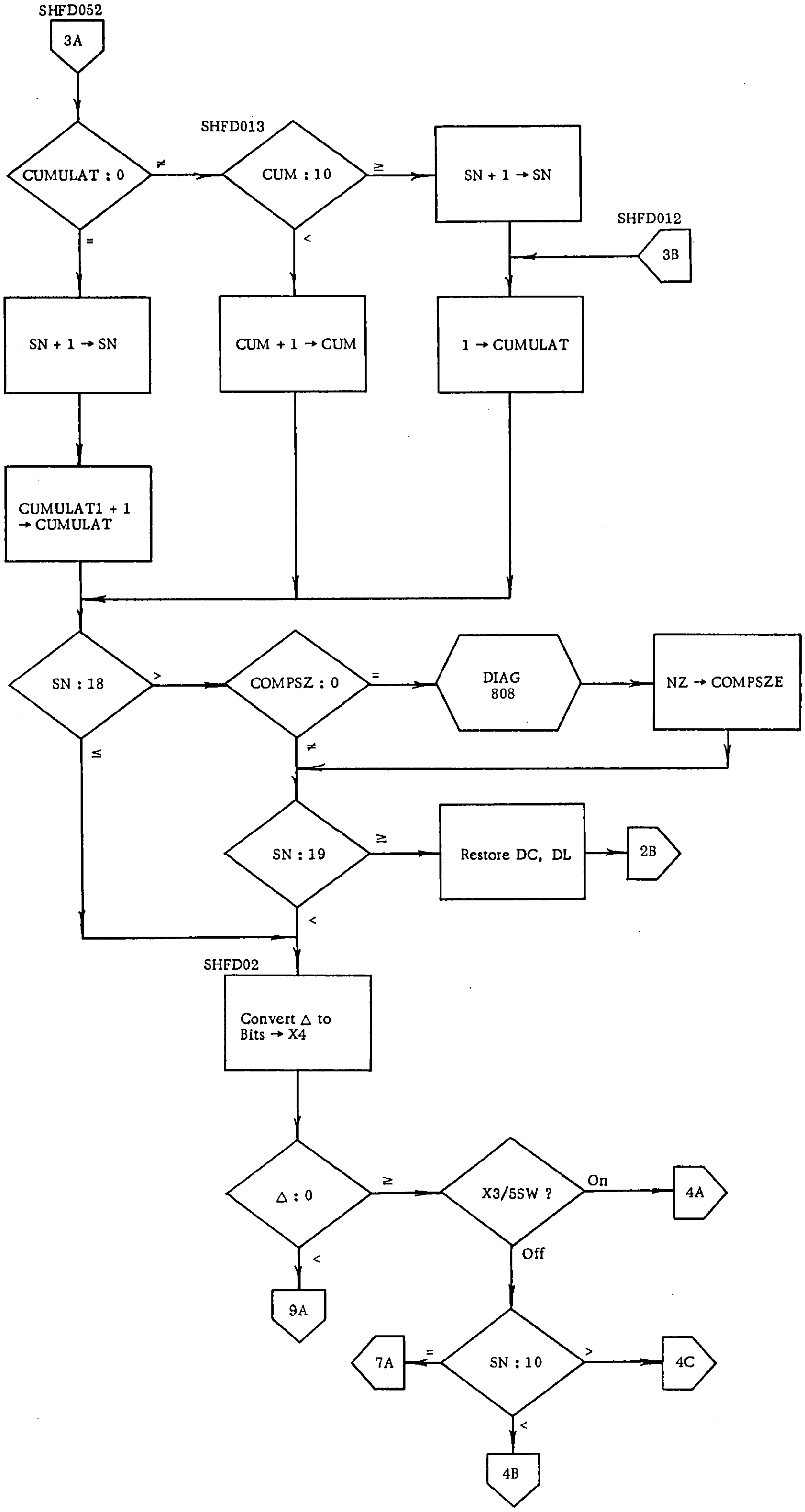


Figure 3-80. GENARTH Flowchart (3 of 33)

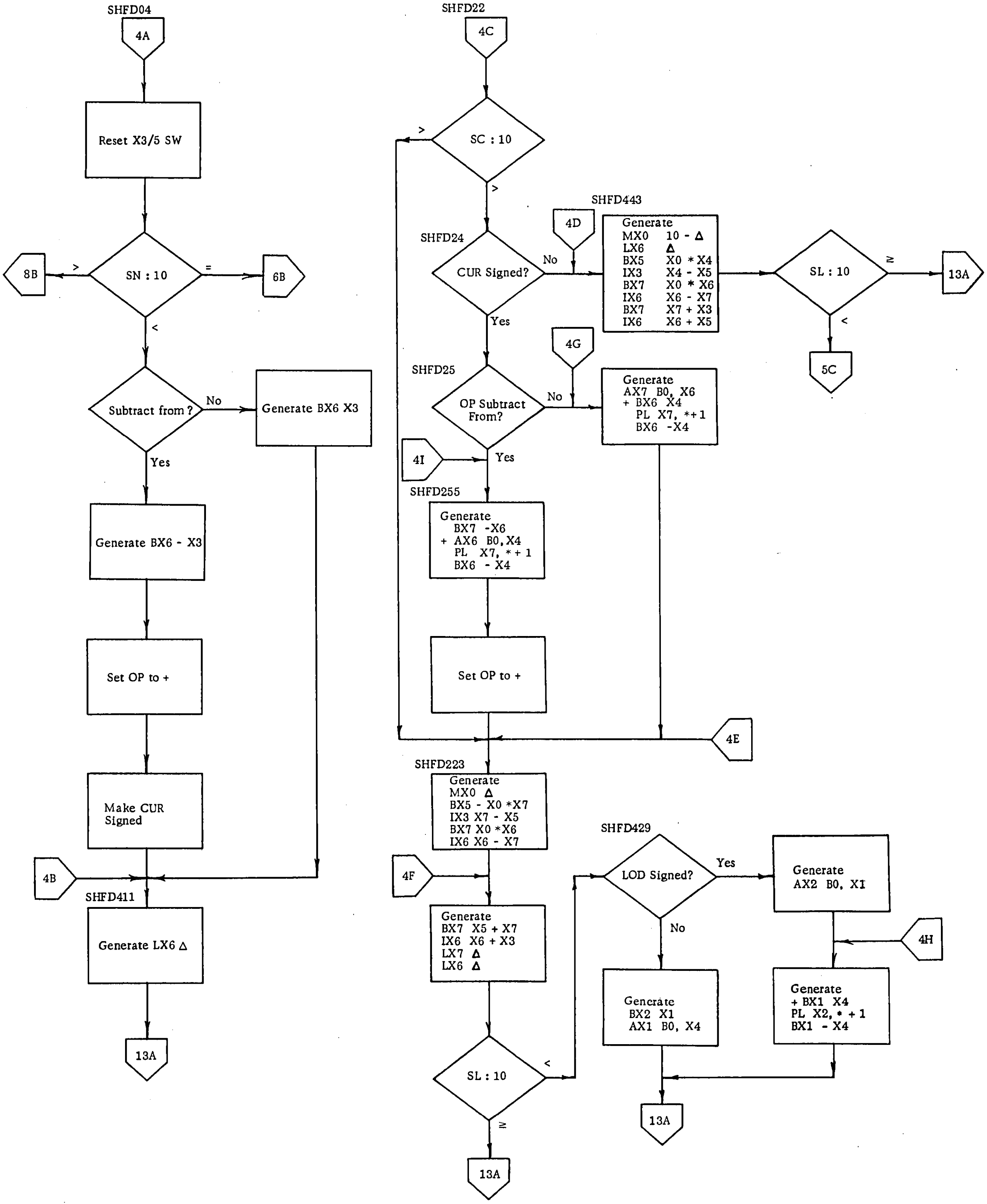


Figure 3-80. GENARTH Flowchart (4 of 33)

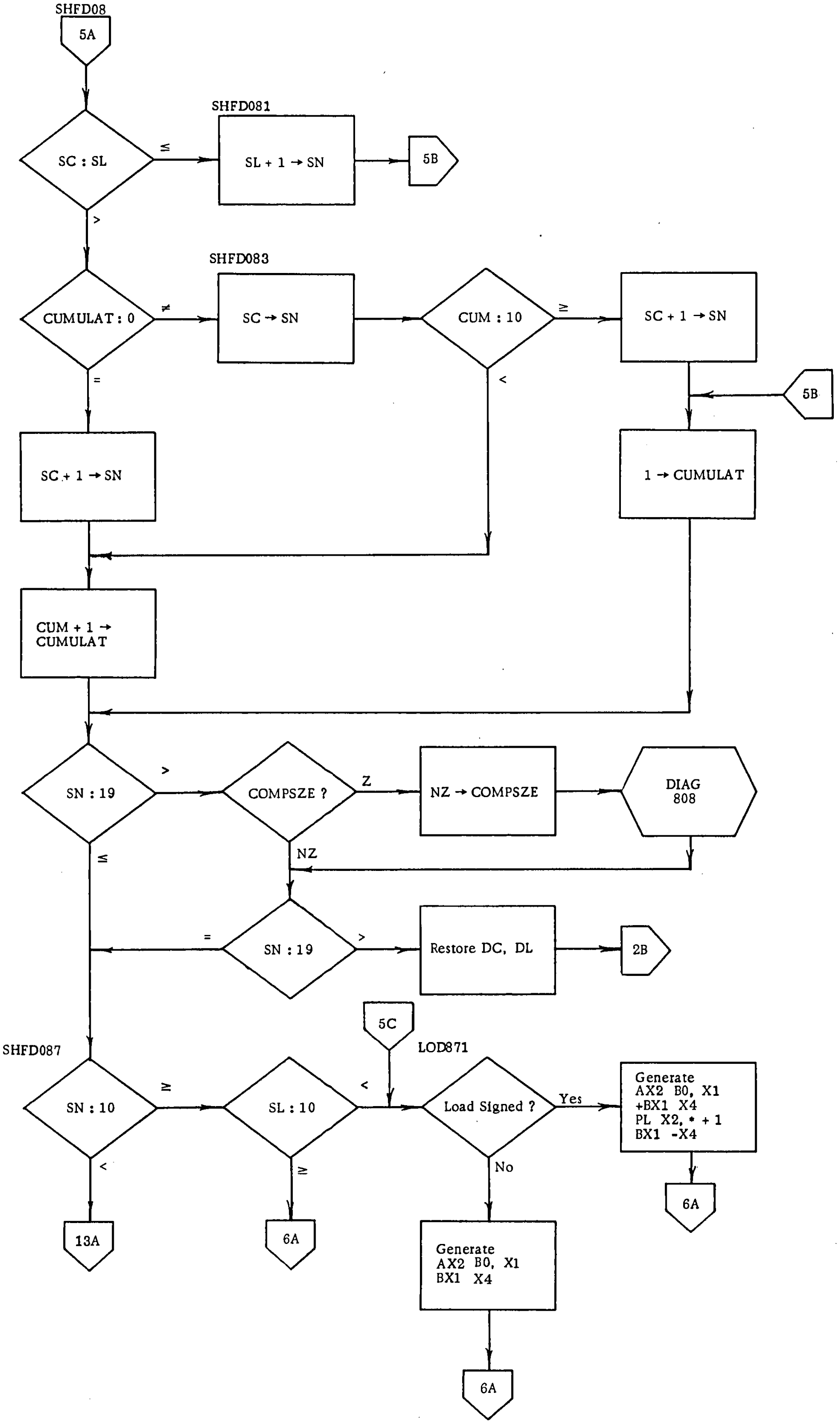


Figure 3-80. GENARTH Flowchart (5 of 33)

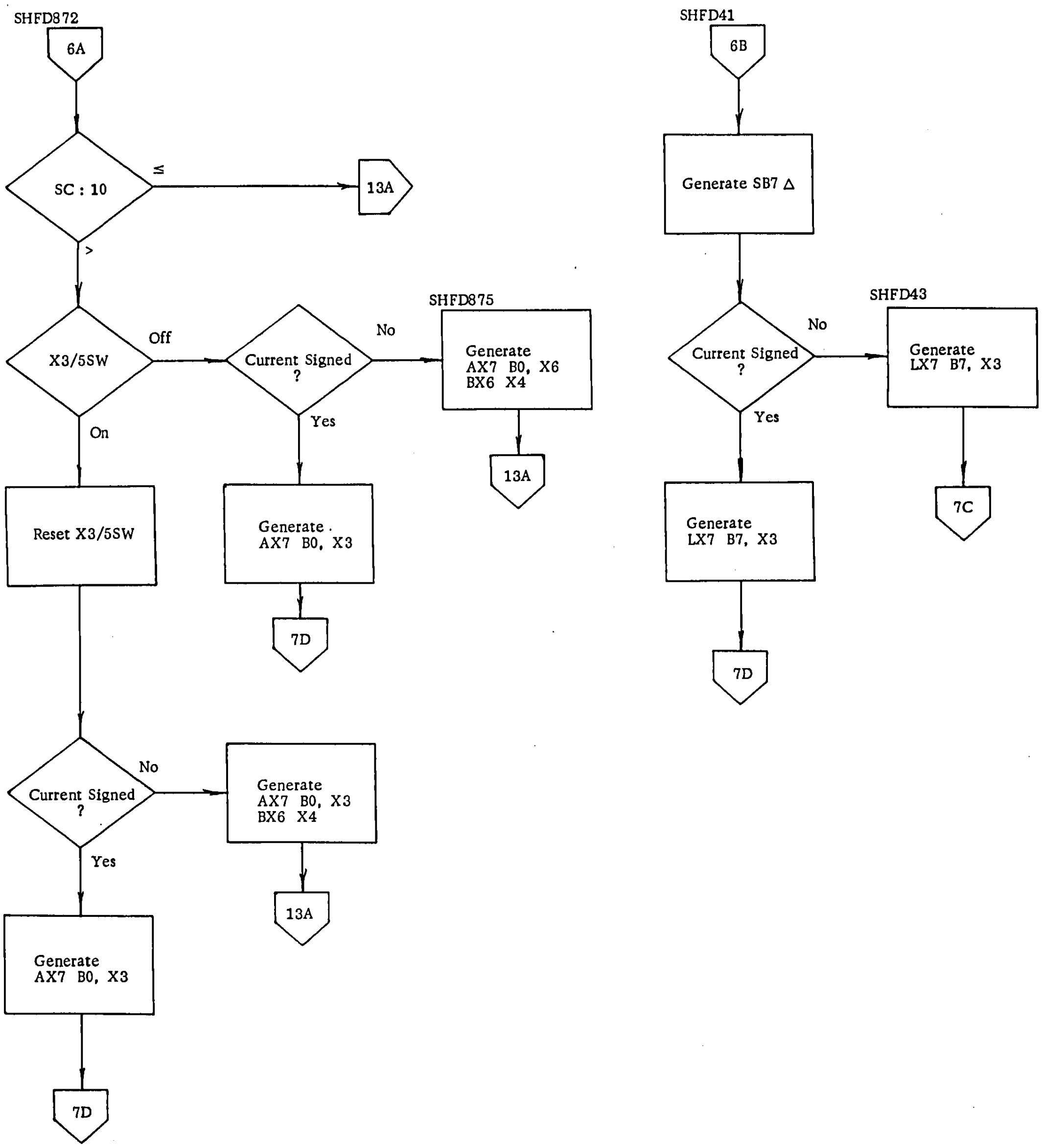


Figure 3-80. GENARTH Flowchart (6 of 33)

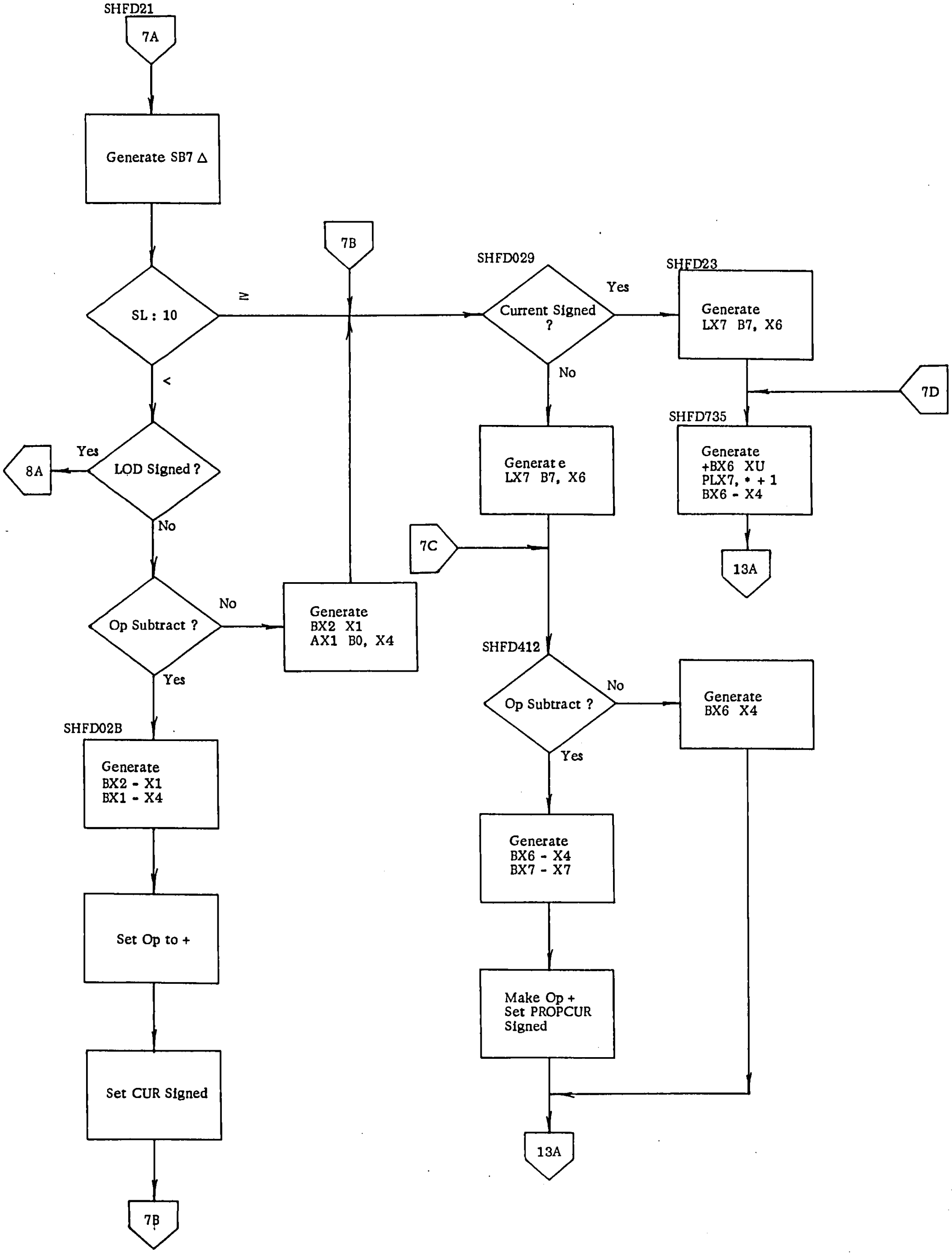


Figure 3-80. GENARTH Flowchart (7 of 33)

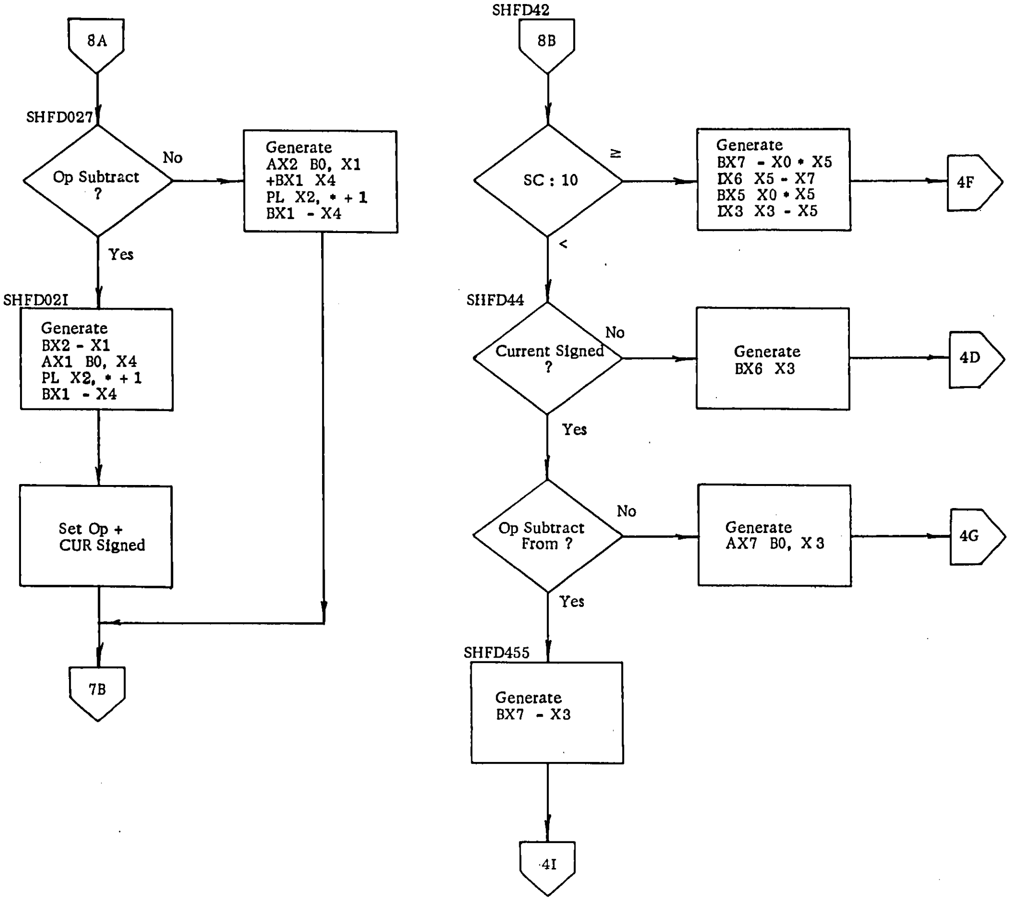


Figure 3-80. GENARTH Flowchart (8 of 33)

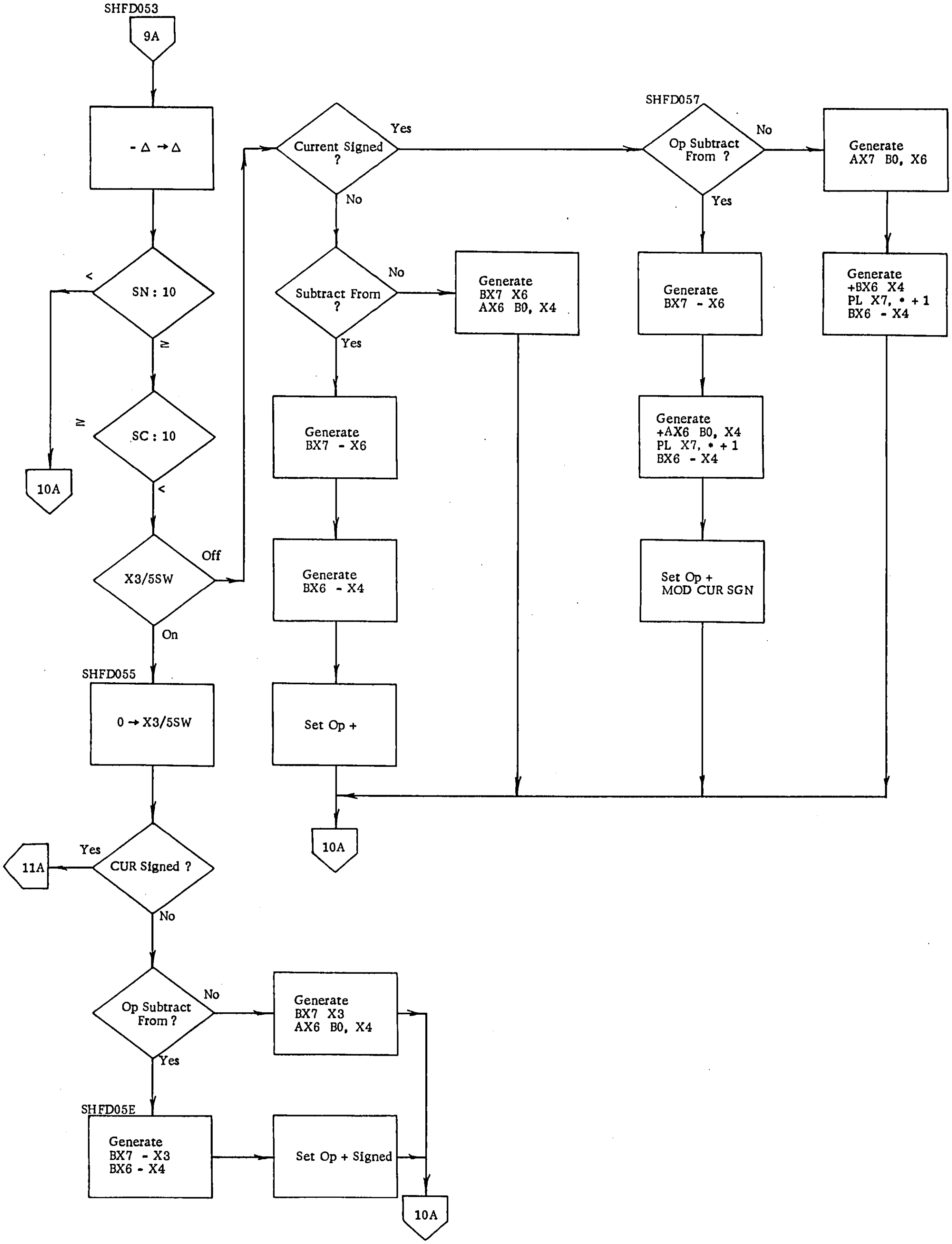


Figure 3-80. GENARTH Flowchart (9 of 33)

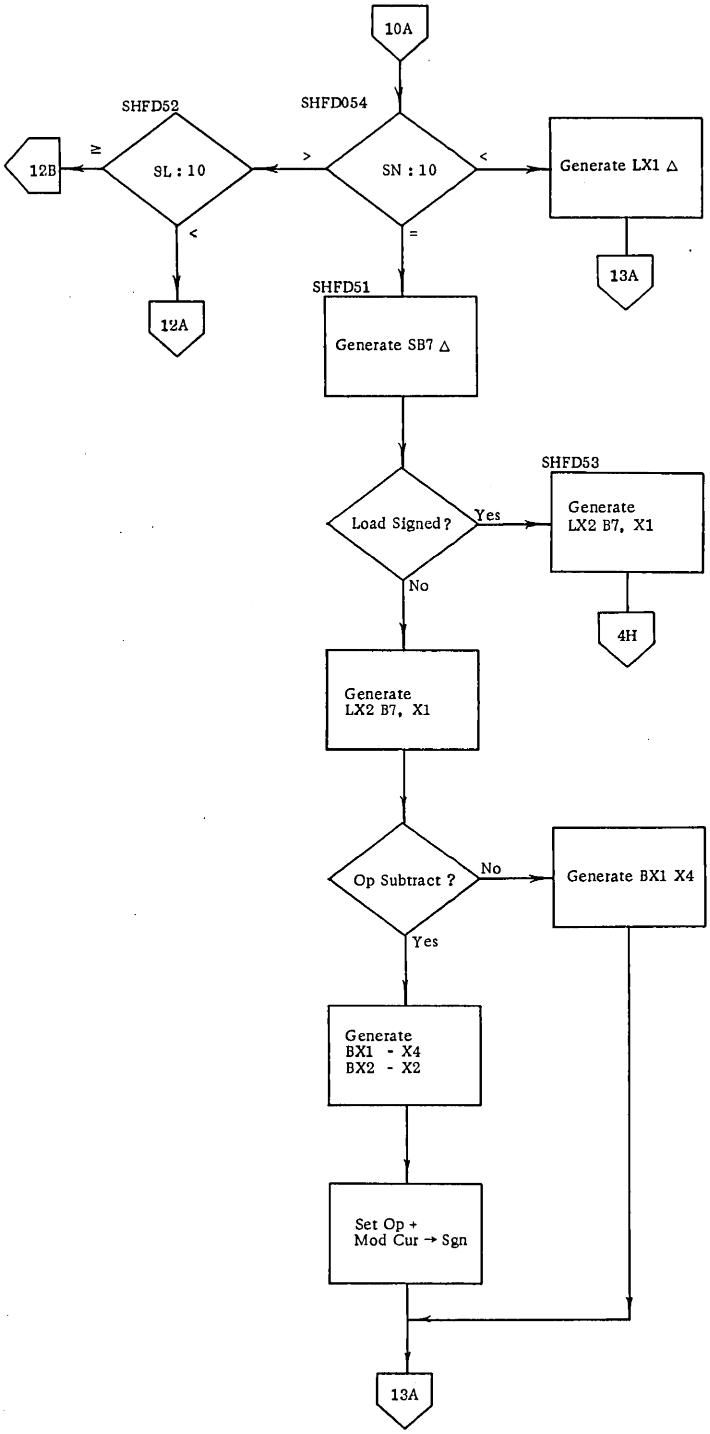


Figure 3-80. GENARTH Flowchart (10 of 33)

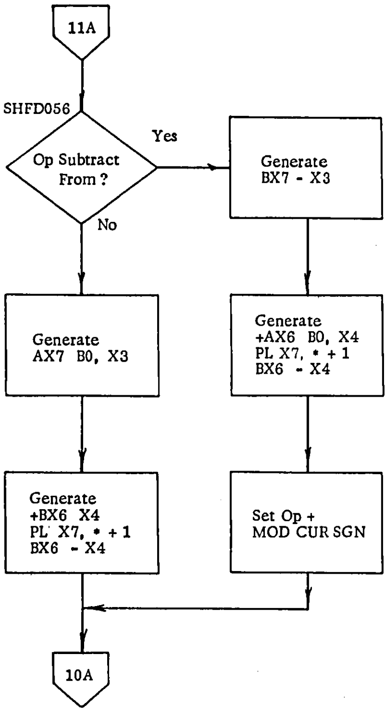


Figure 3-80. GENARTH Flowchart (11 of 33)

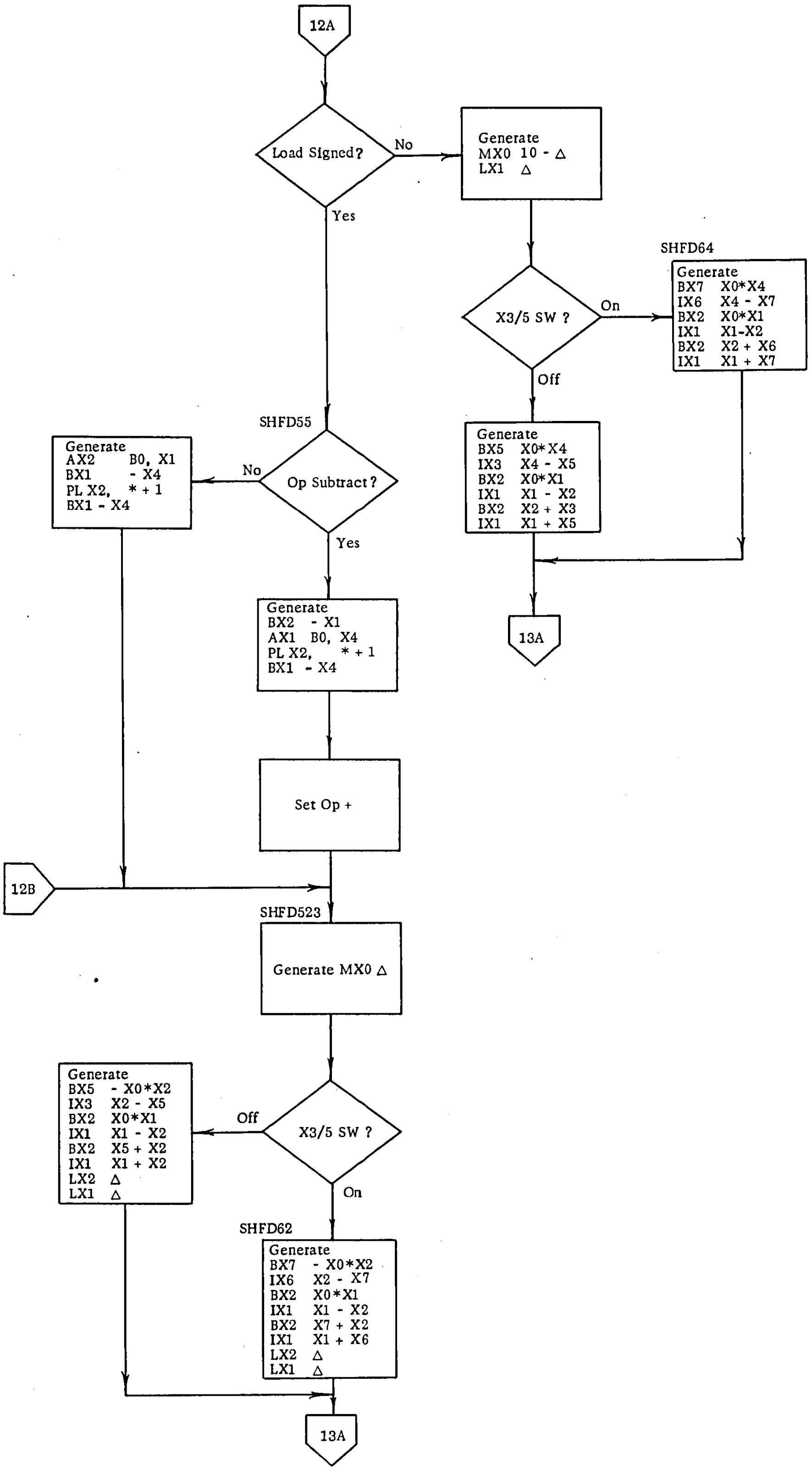


Figure 3-80. GENARTH Flowchart (12 of 33)

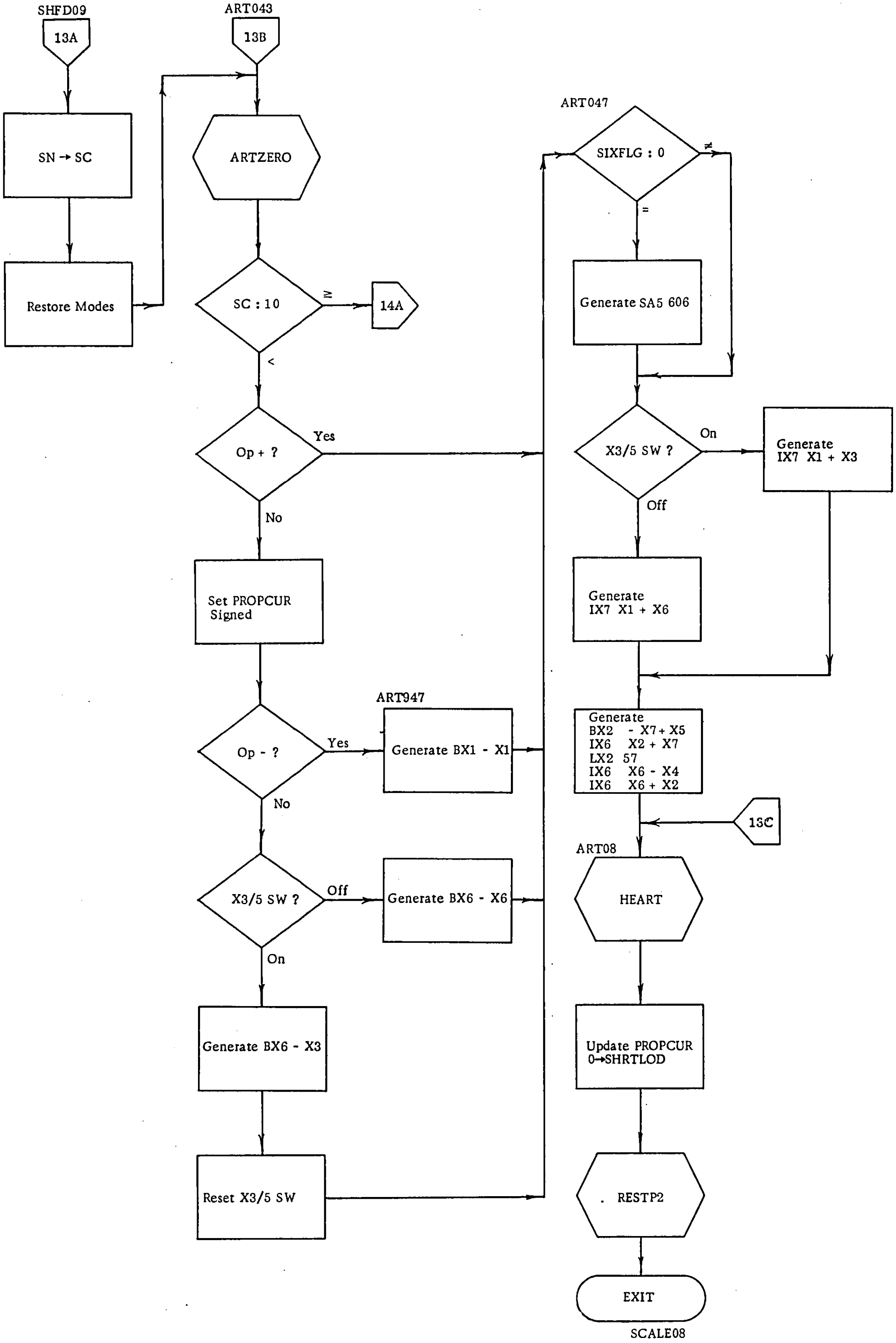


Figure 3-80. GENARTH Flowchart (13 of 33)

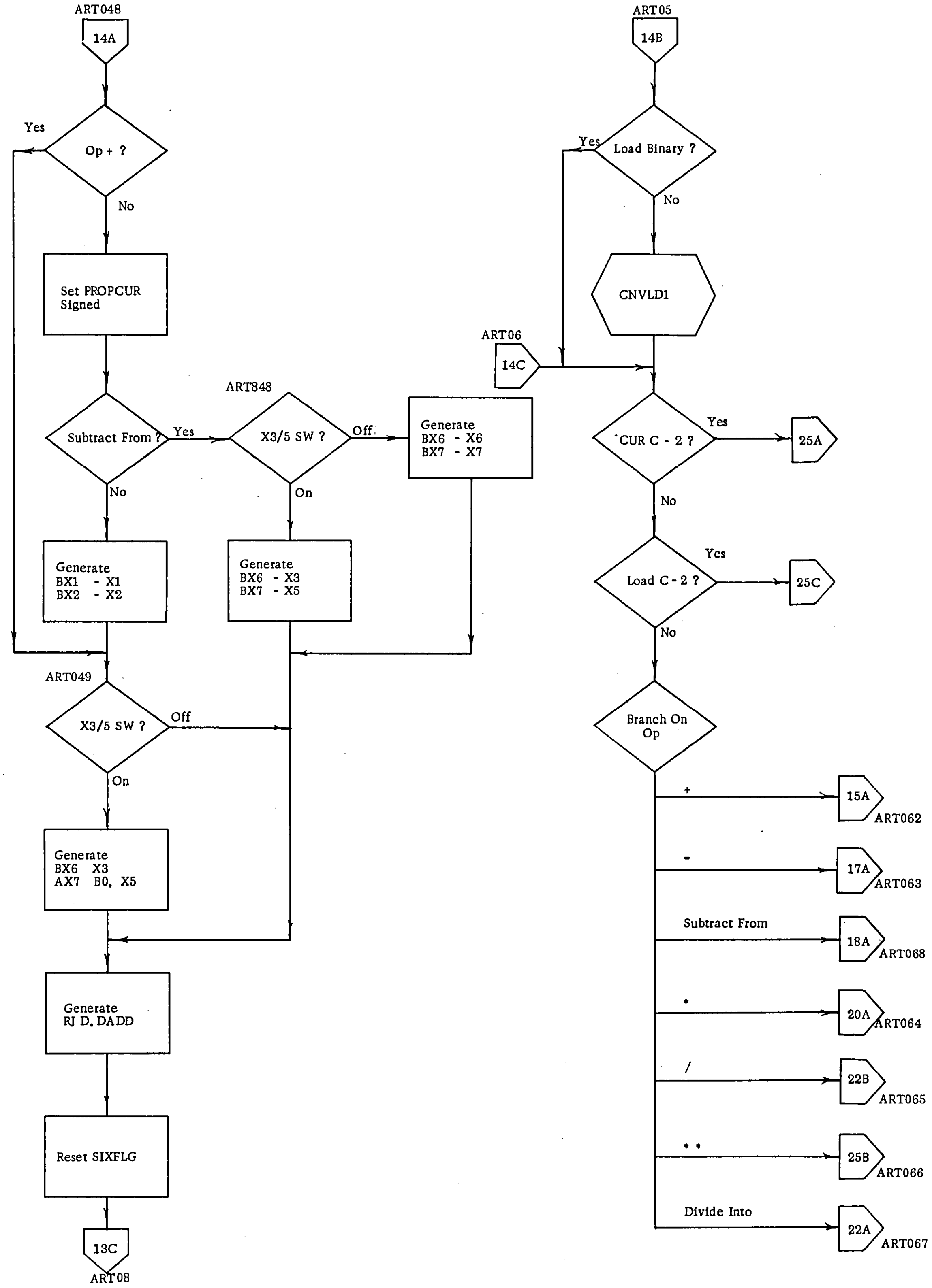


Figure 3-80. GENARTH Flowchart (14 of 33)

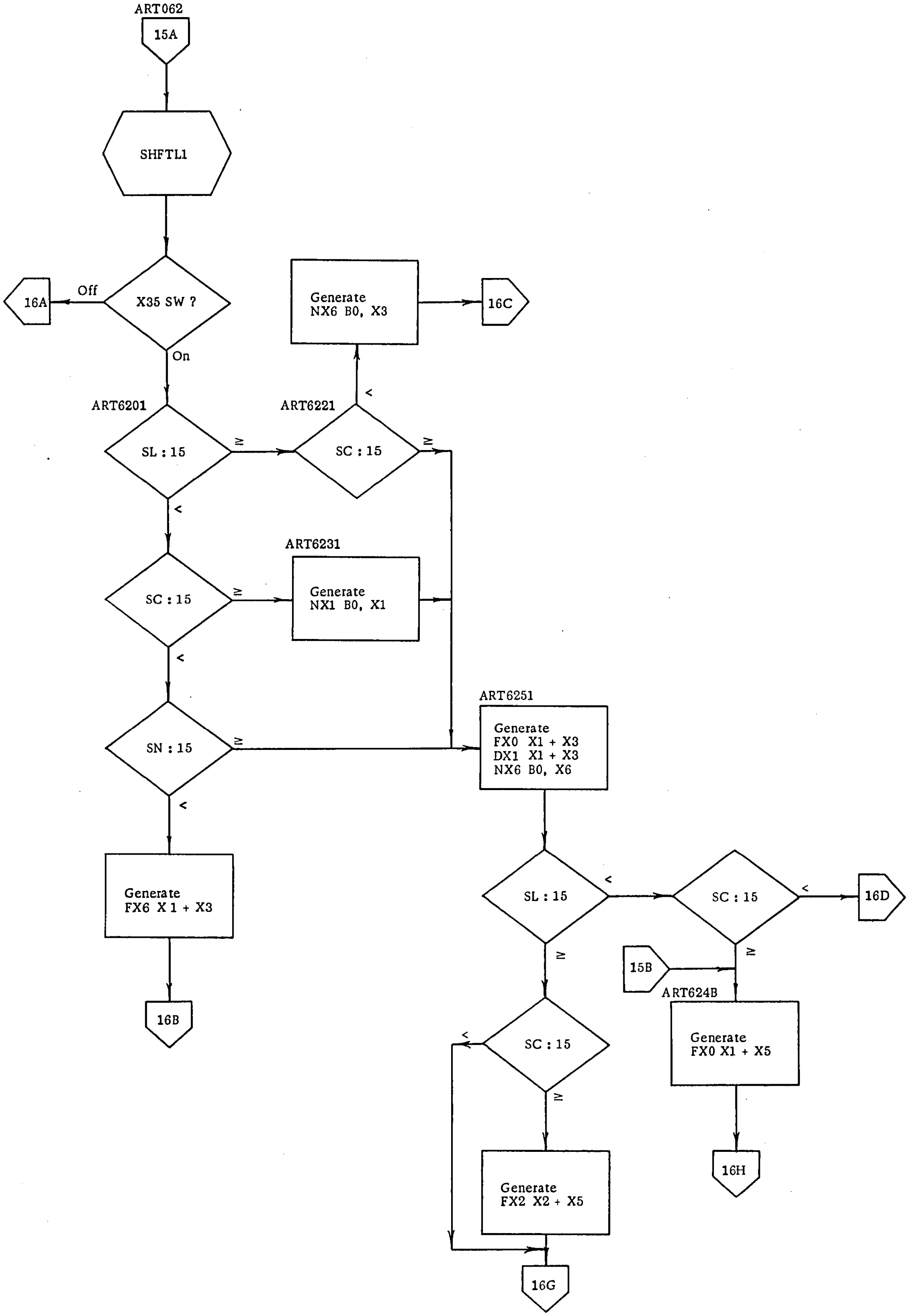


Figure 3-80. GENARTH Flowchart (15 of 33)

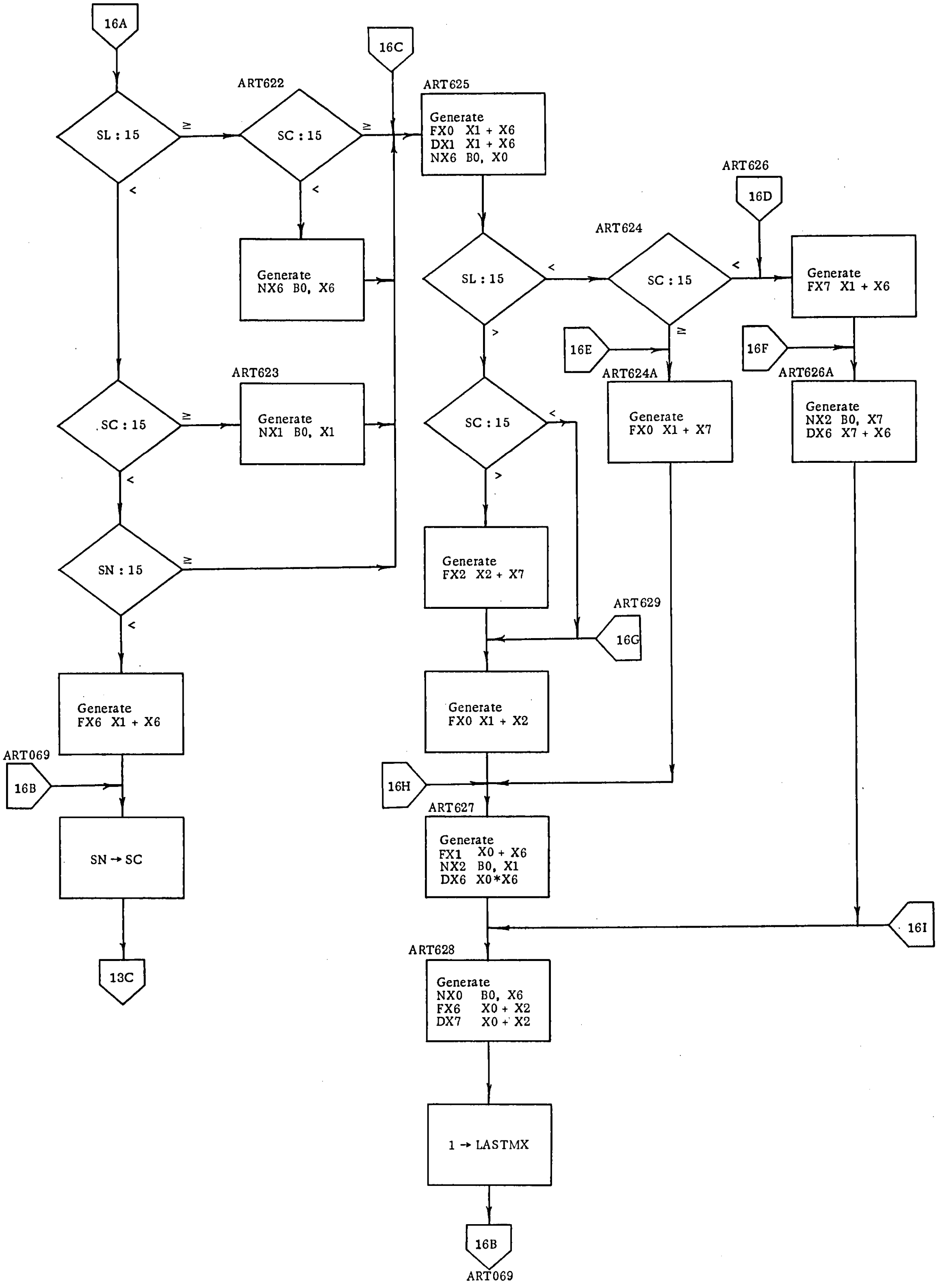


Figure 3-80. GENARTH Flowchart (16 of 33)

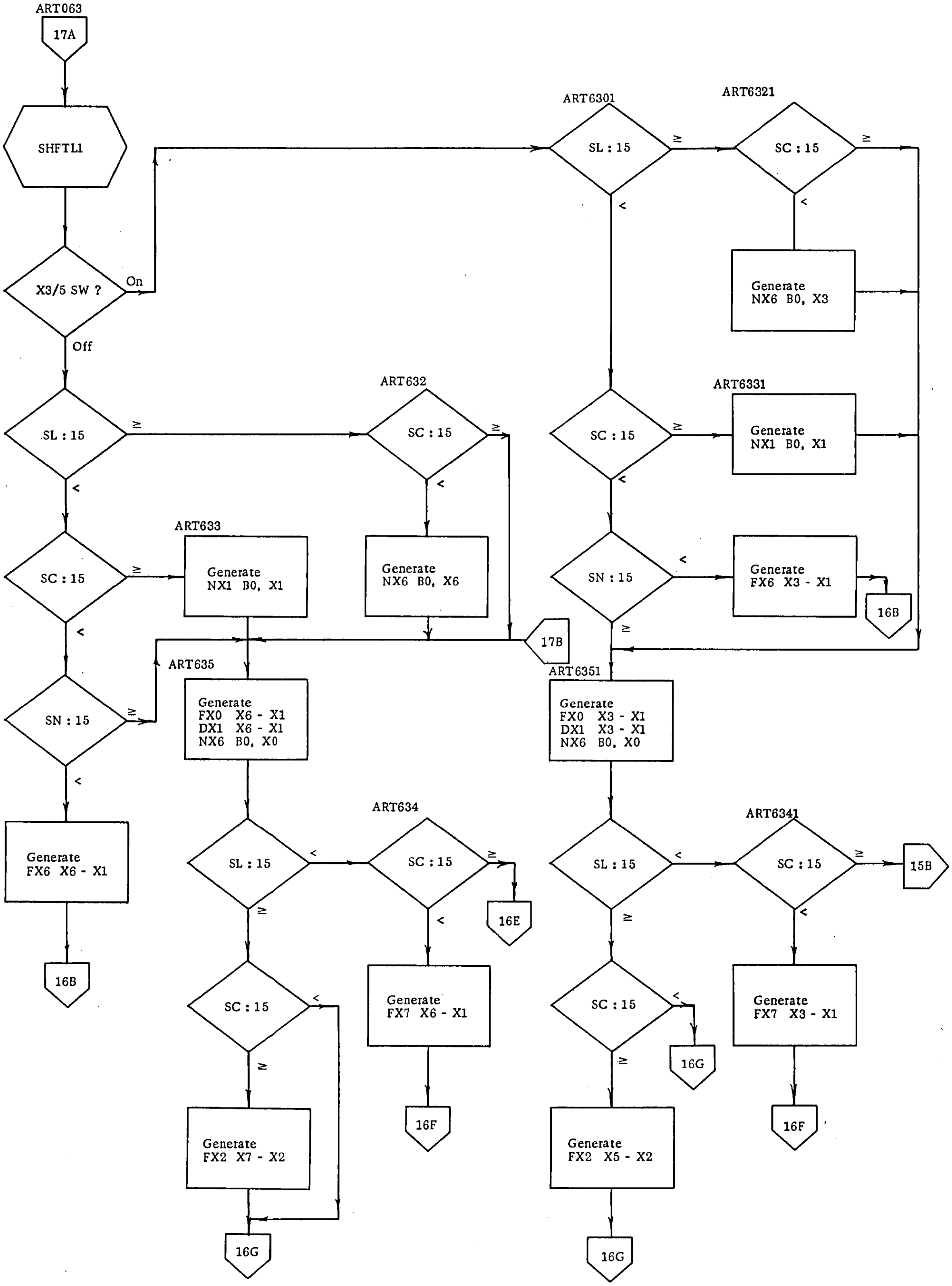


Figure 3-80. GENARTH Flowchart (17 of 33)

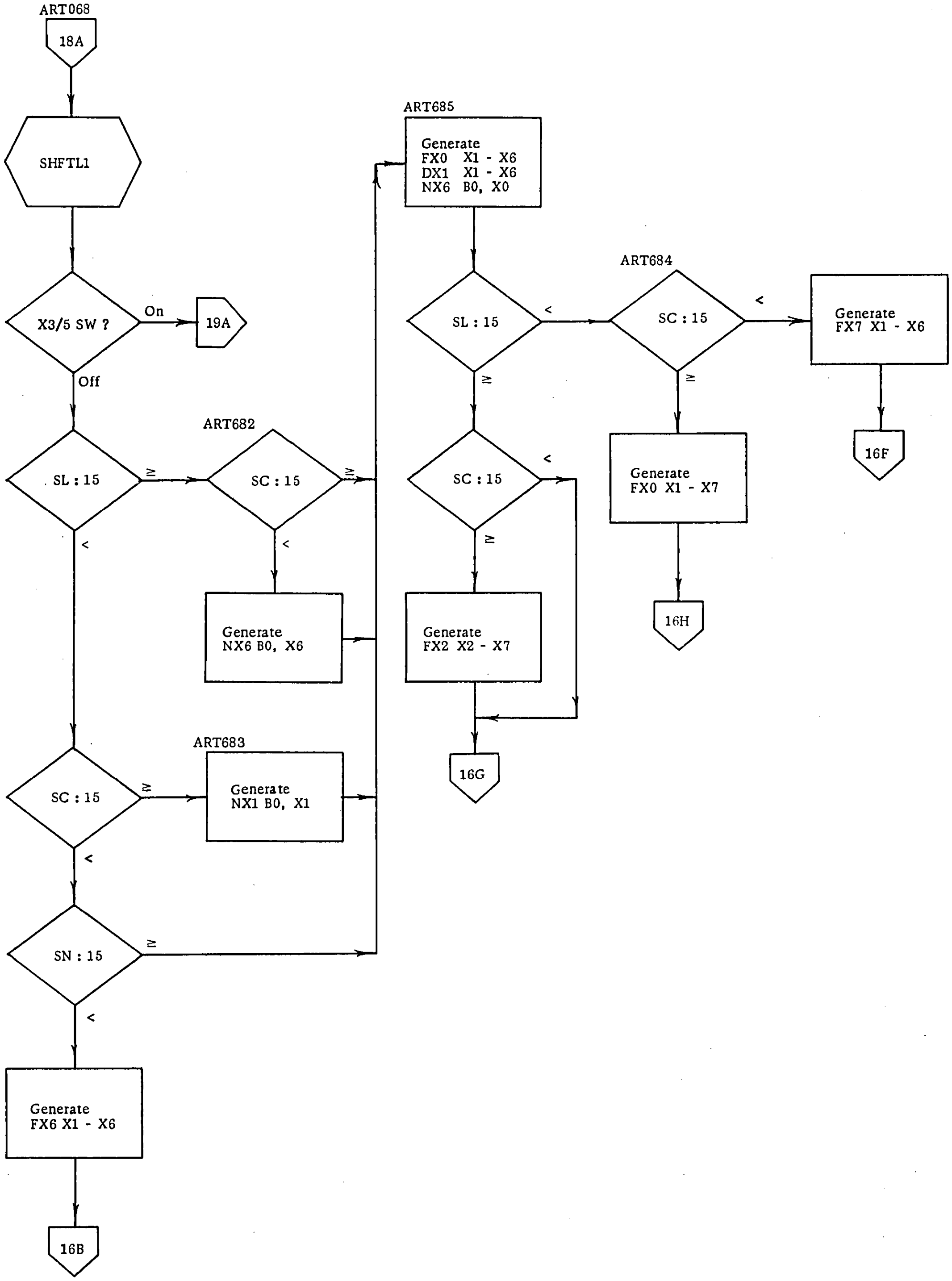


Figure 3-80. GENARTH Flowchart (18 of 33)

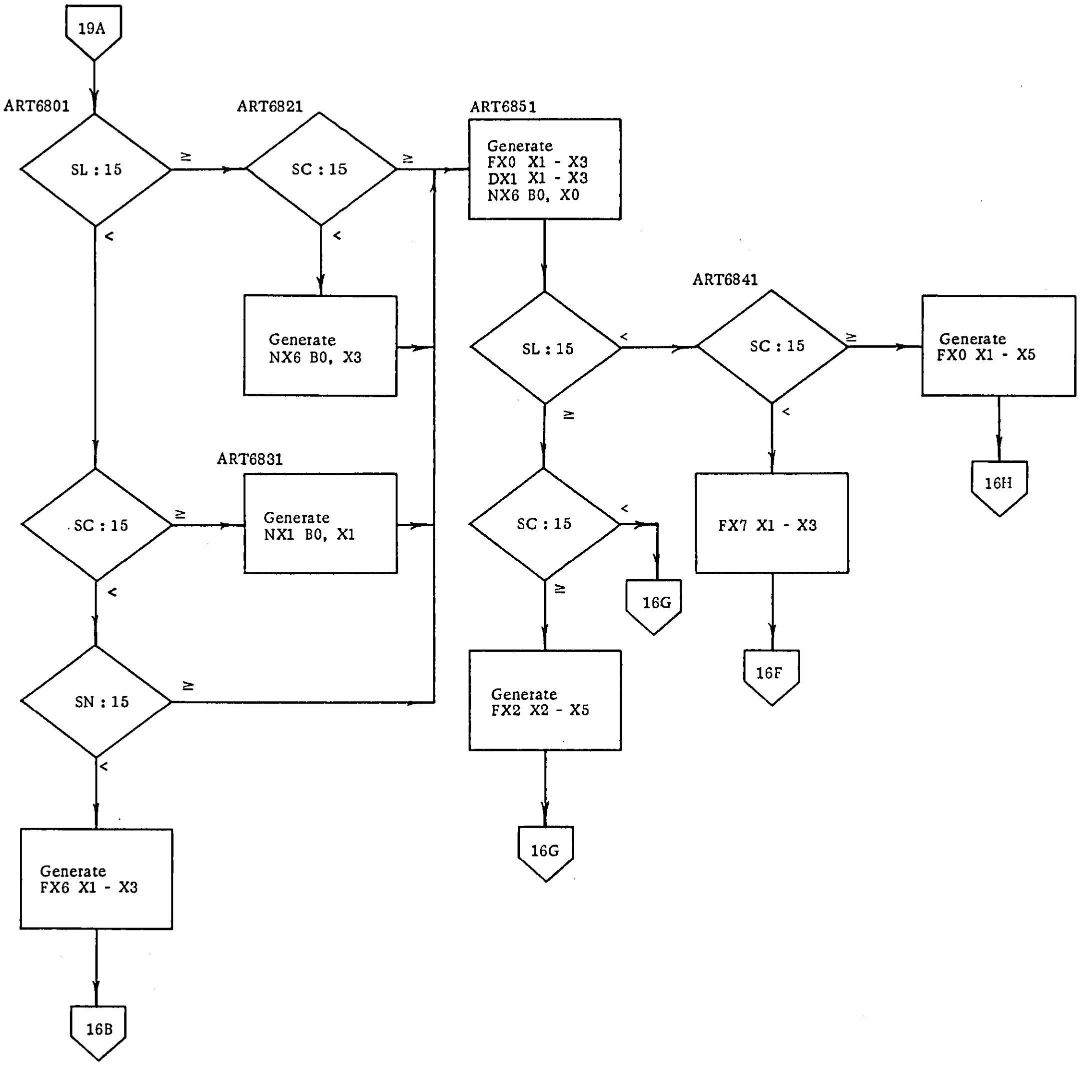


Figure 3-80. GENARTH Flowchart (19 of 33)

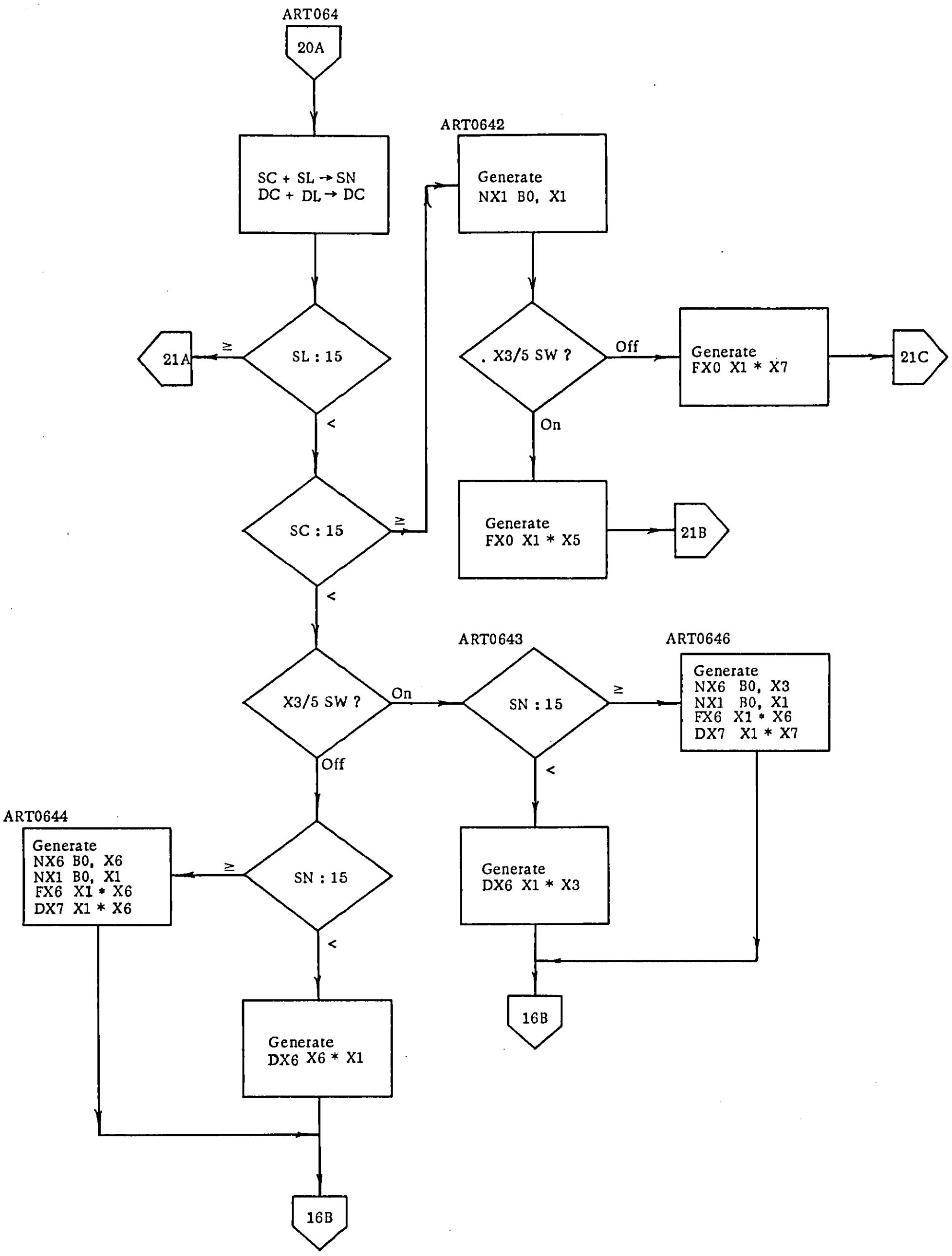


Figure 3-80. GENARTH Flowchart (20 of 33)

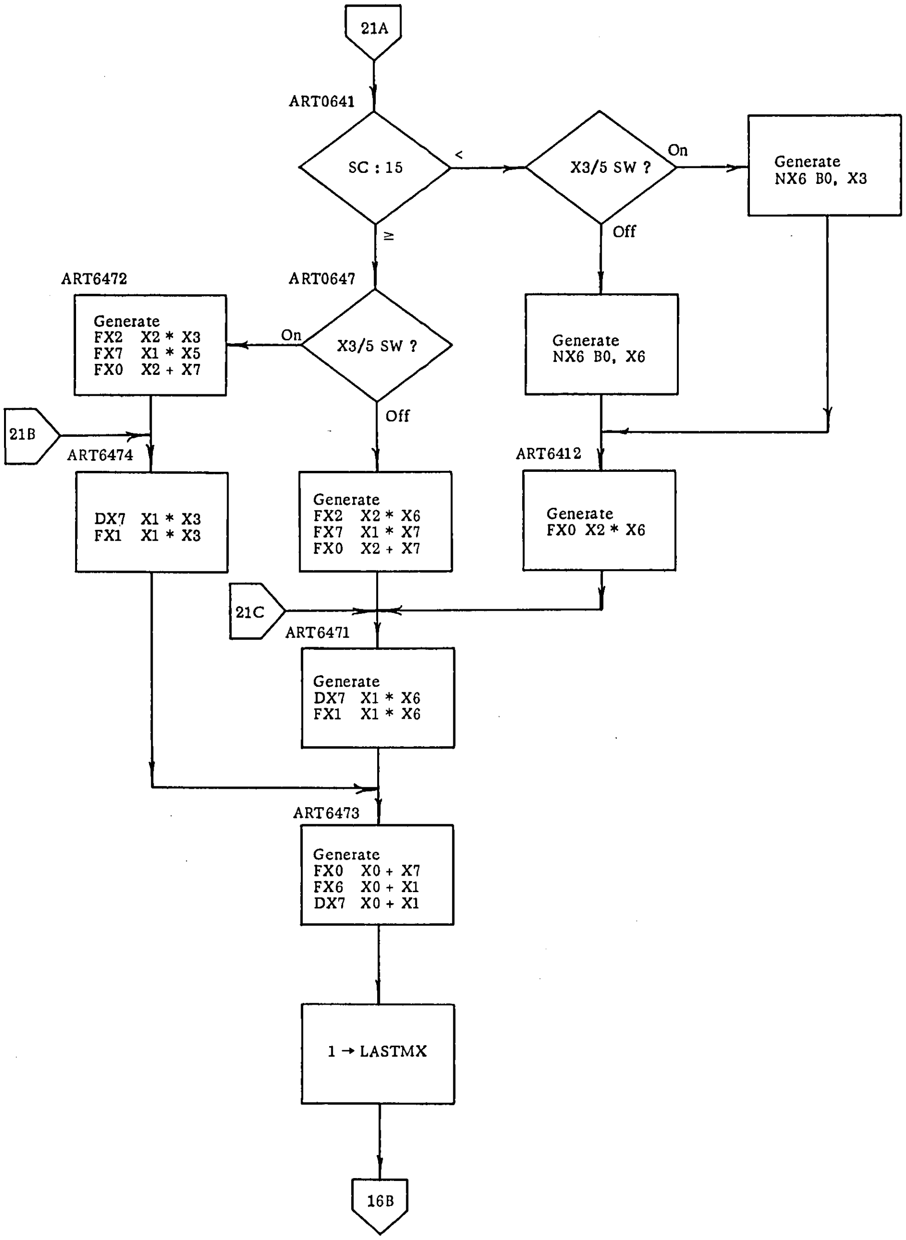


Figure 3-80. GENARTH Flowchart (21 of 33)

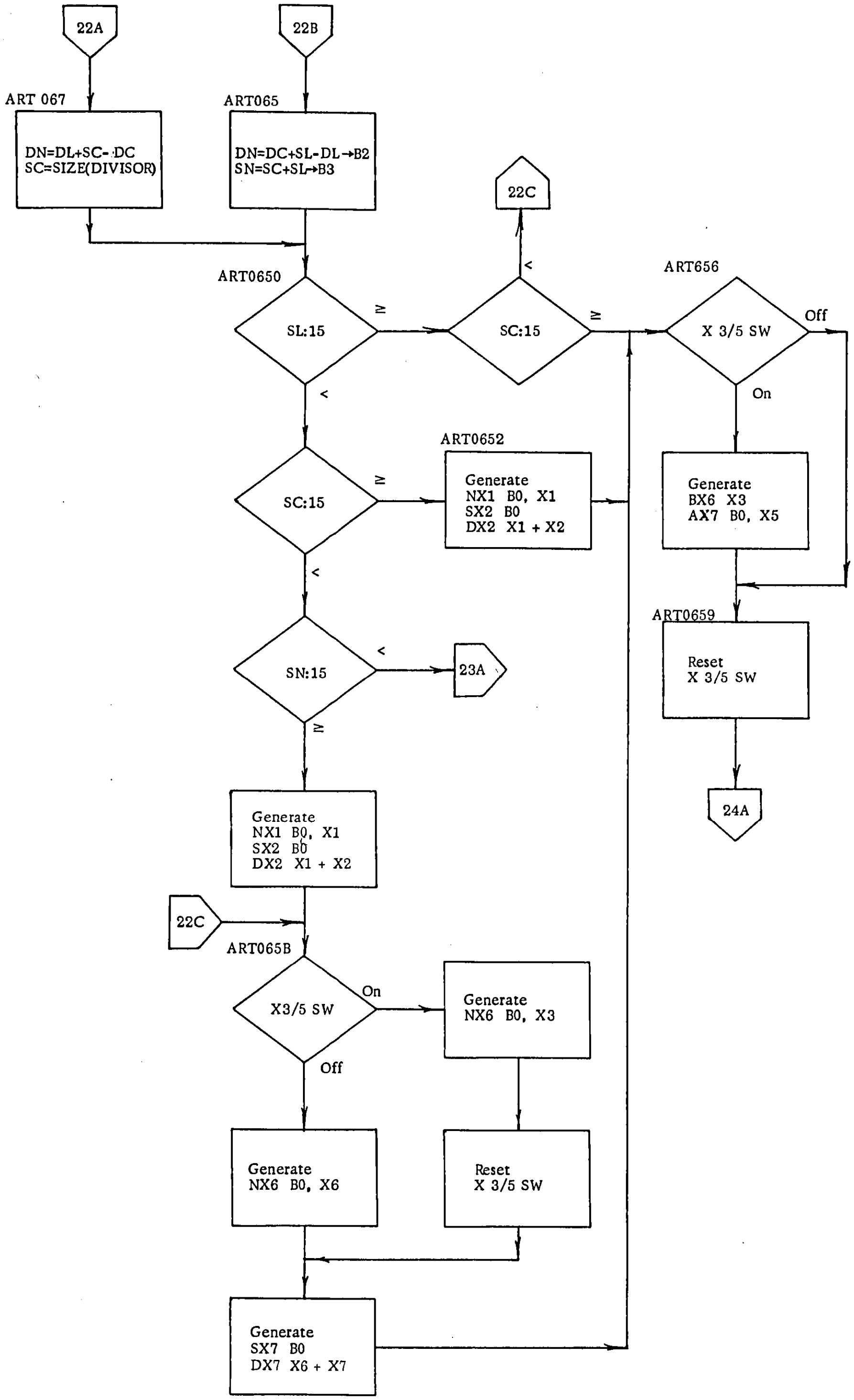


Figure 3-80. GENARTH Flowchart (22 of 33)

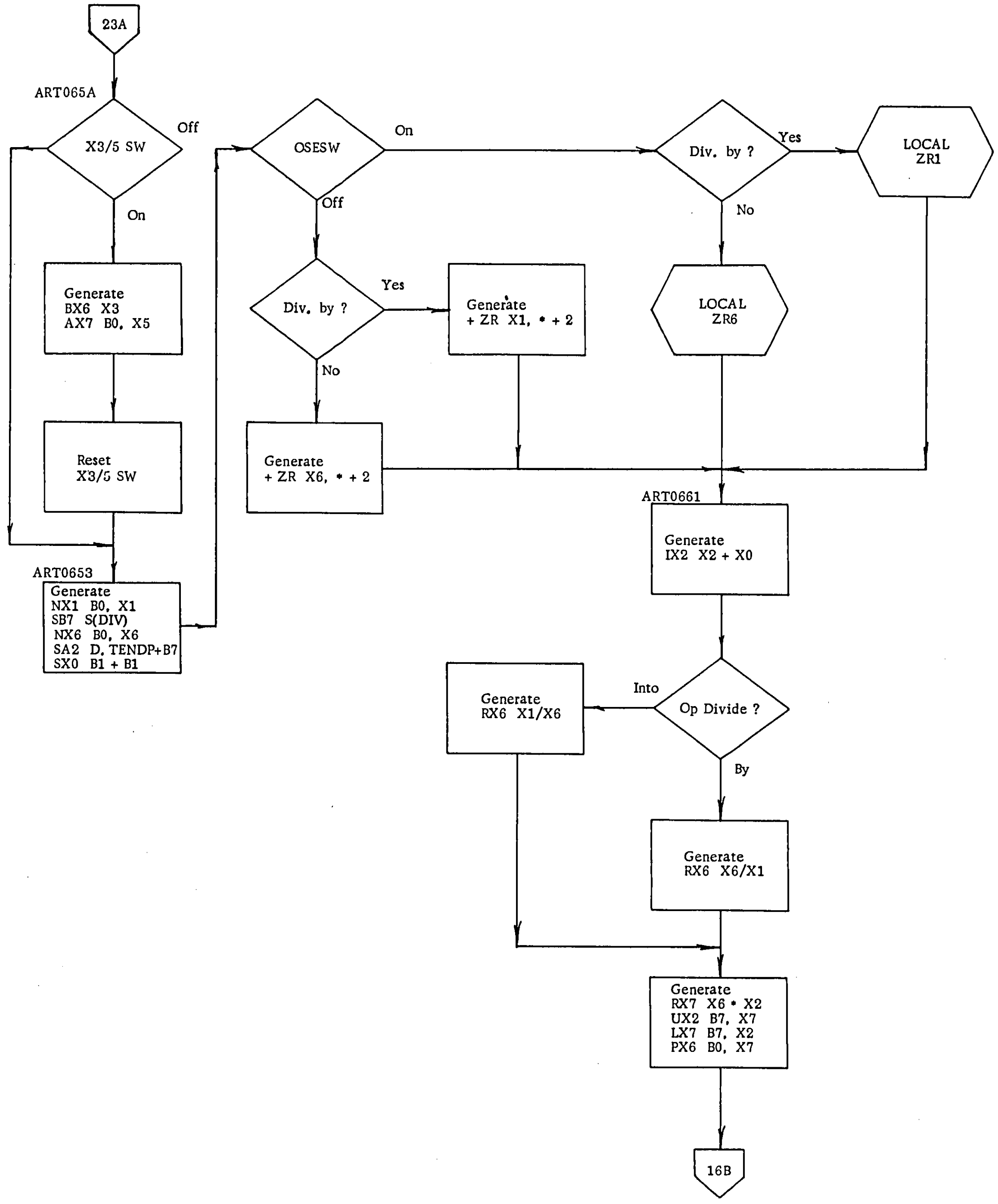


Figure 3-80. GENARTH Flowchart (23 of 33)

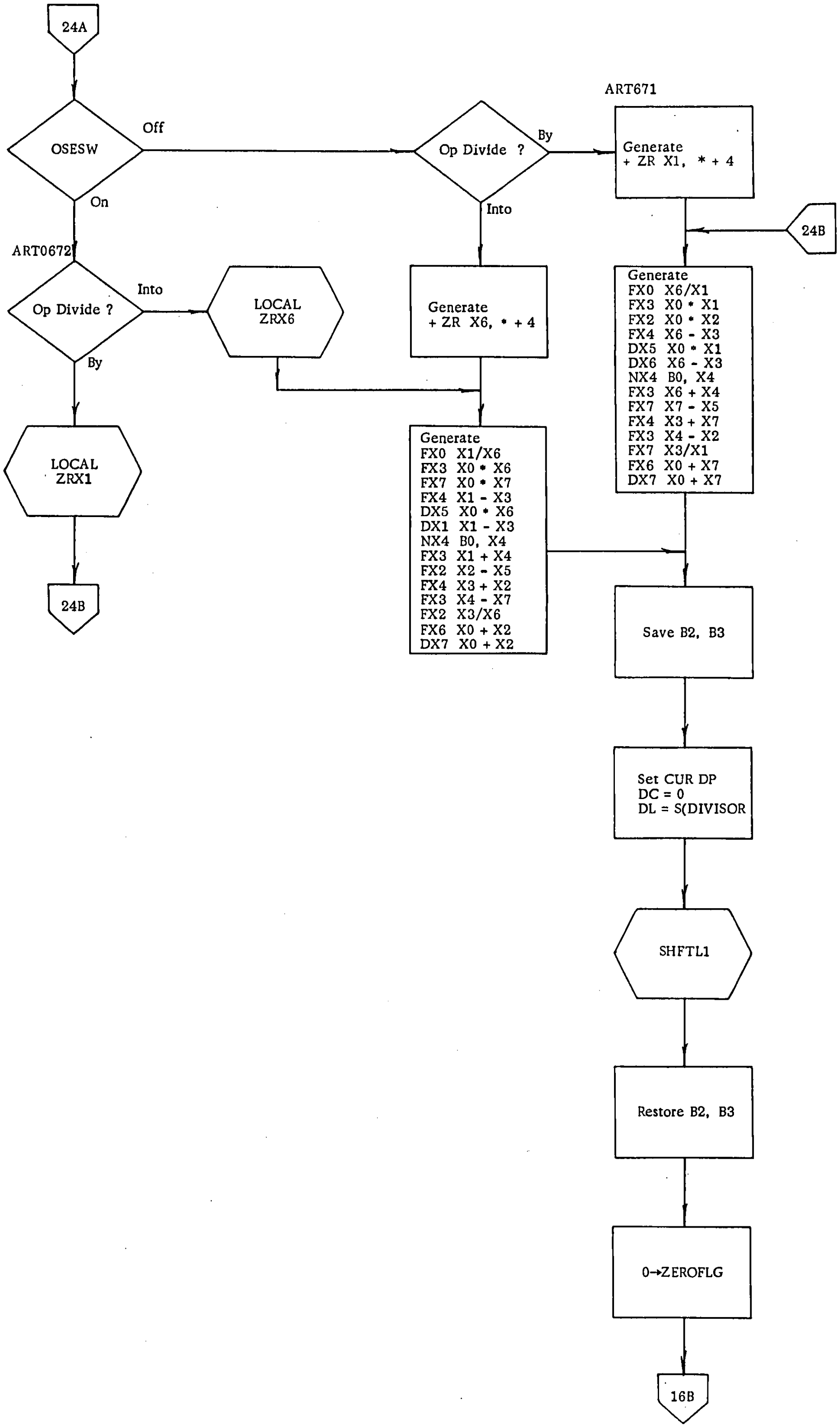


Figure 3-80. GENARTH Flowchart (24 of 33)

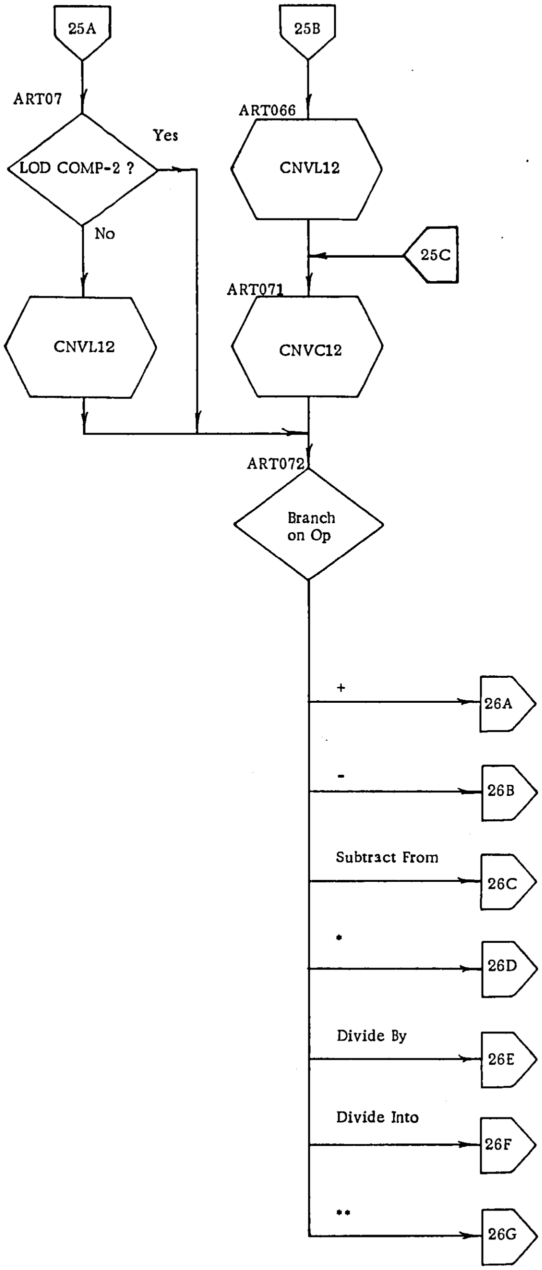


Figure 3-80. GENARTH Flowchart (25 of 33)

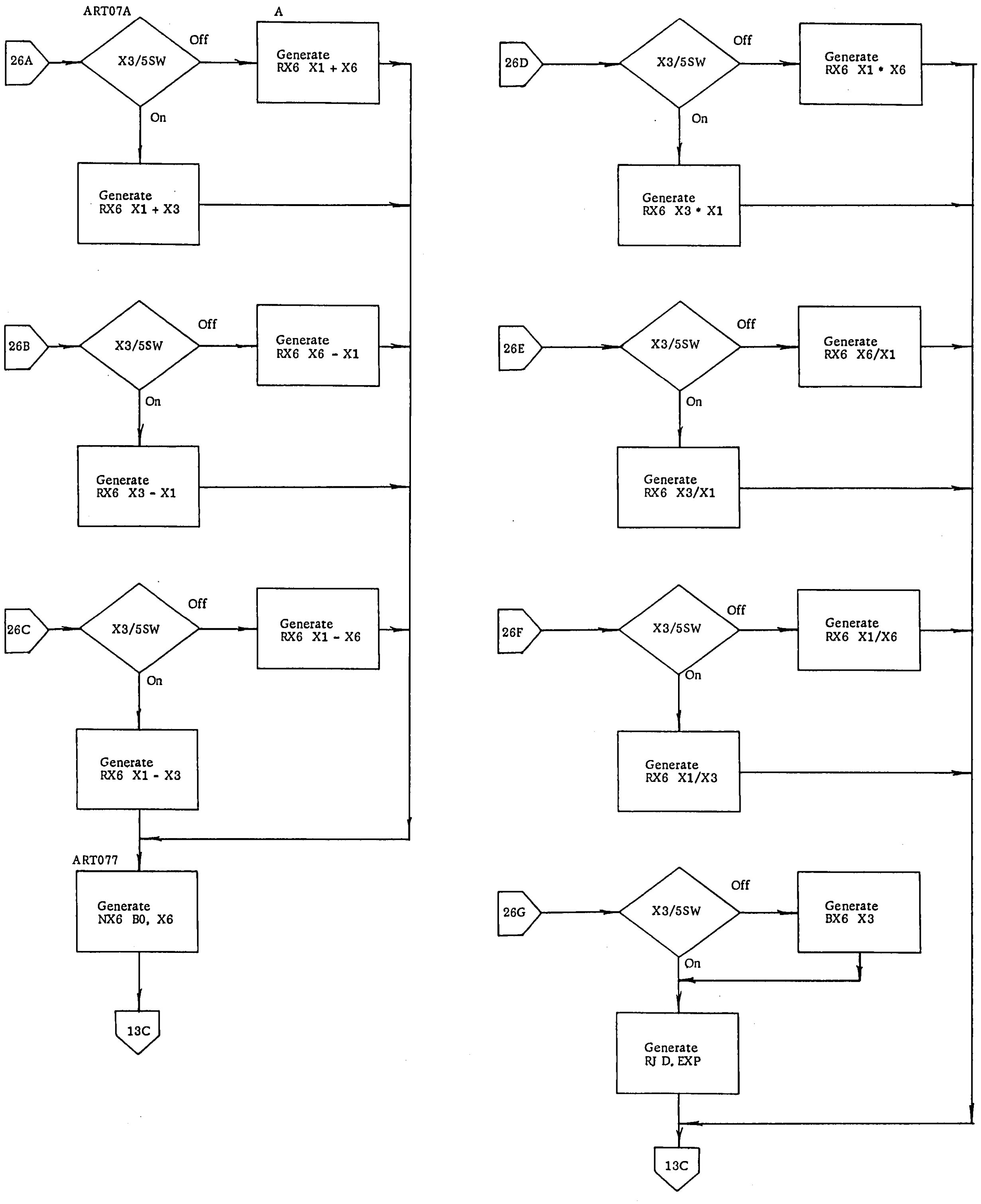


Figure 3-80. GENARTH Flowchart (26 of 33)

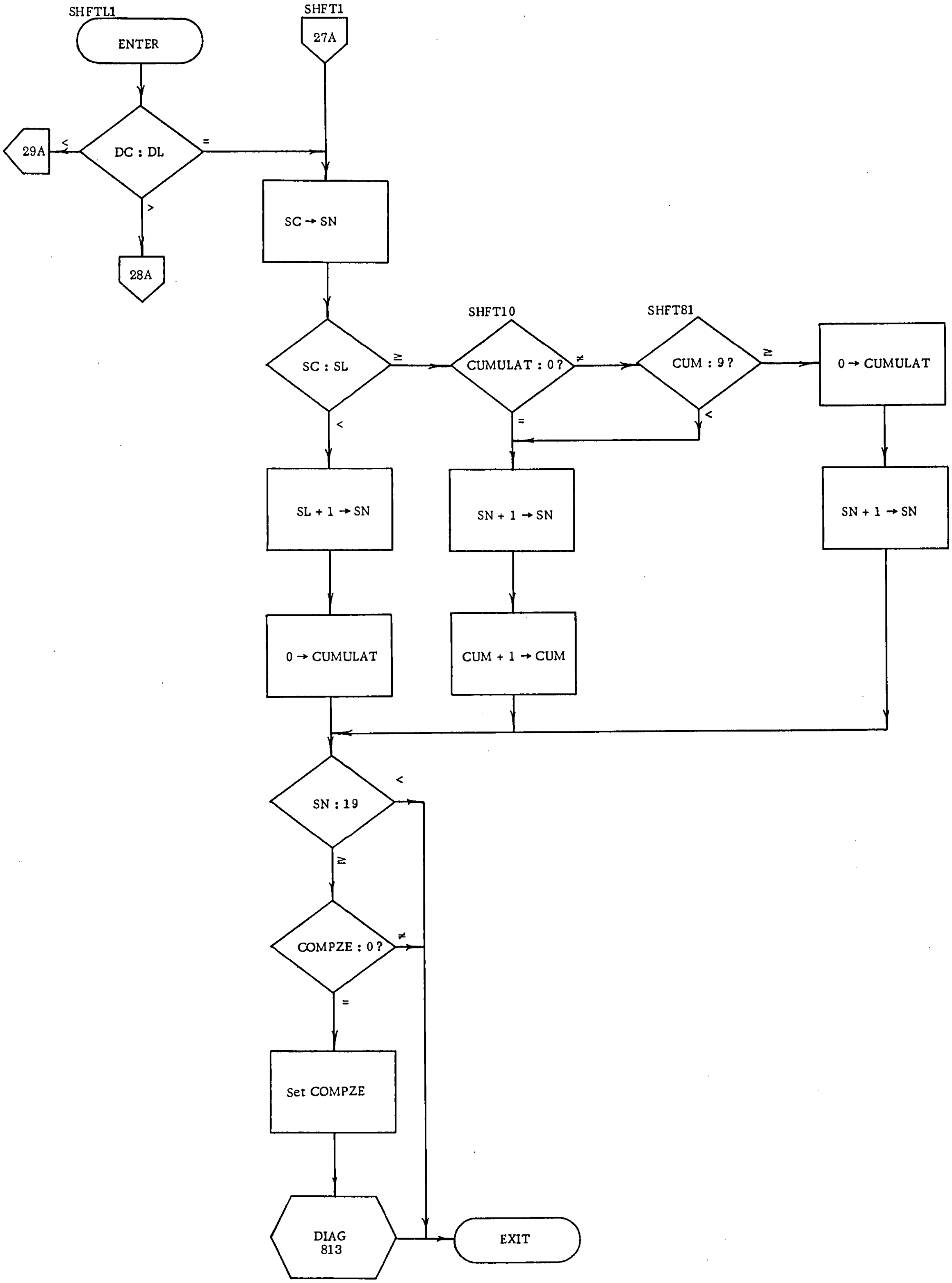


Figure 3-80. GENARTH Flowchart (27 of 33)

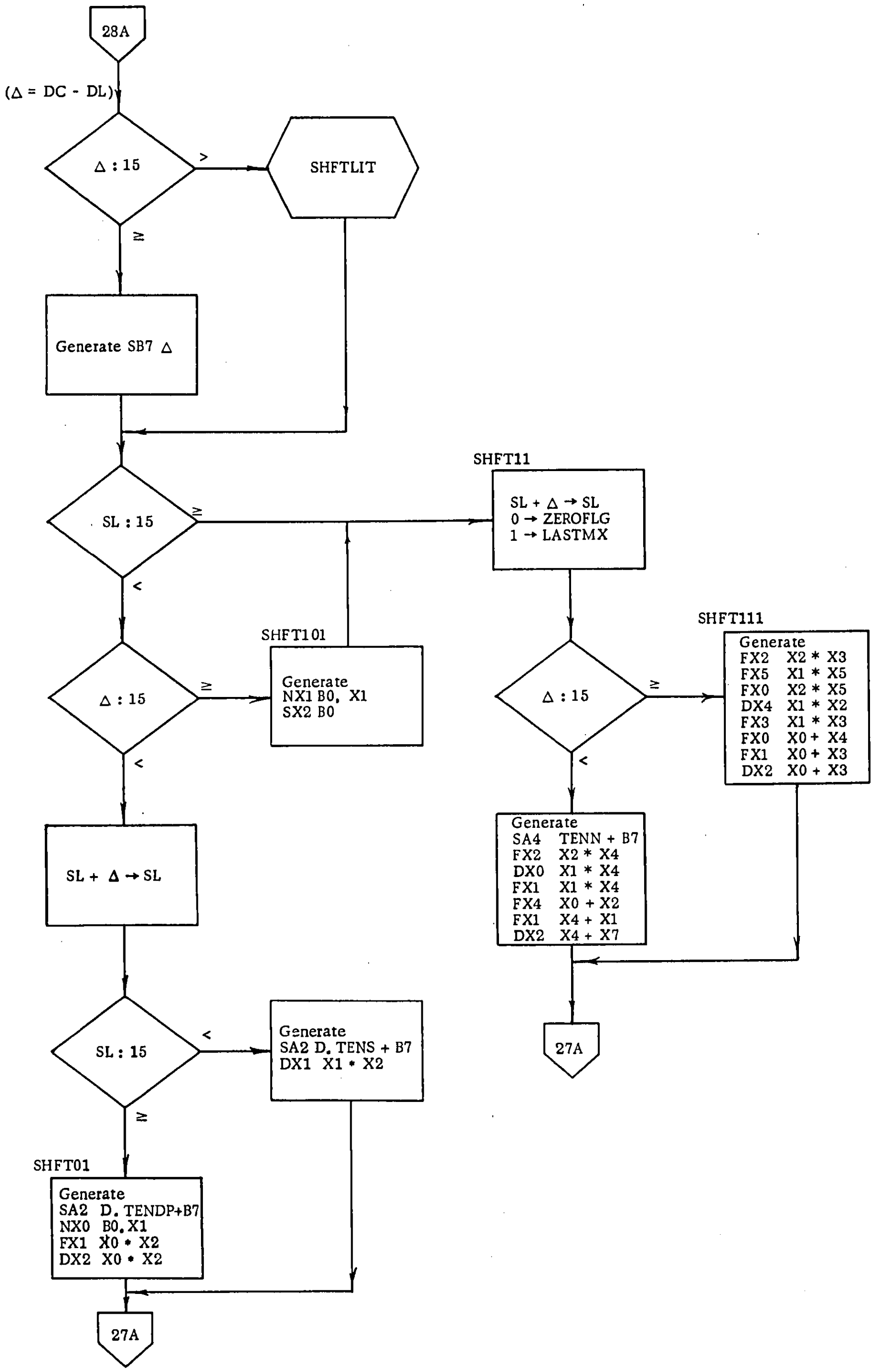


Figure 3-80. GENARTH Flowchart (28 of 33)

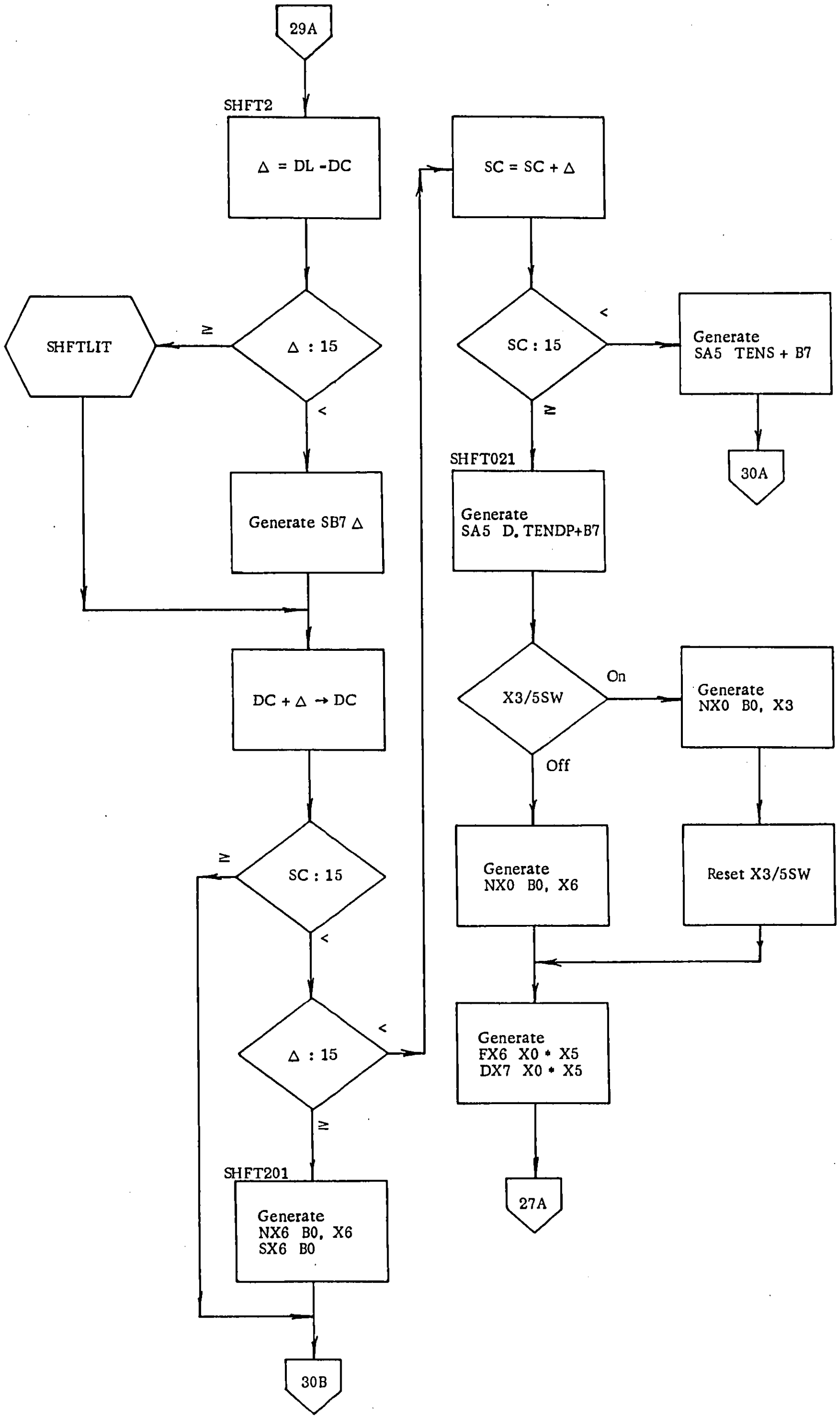


Figure 3-80. GENARTH Flowchart (29 of 33)

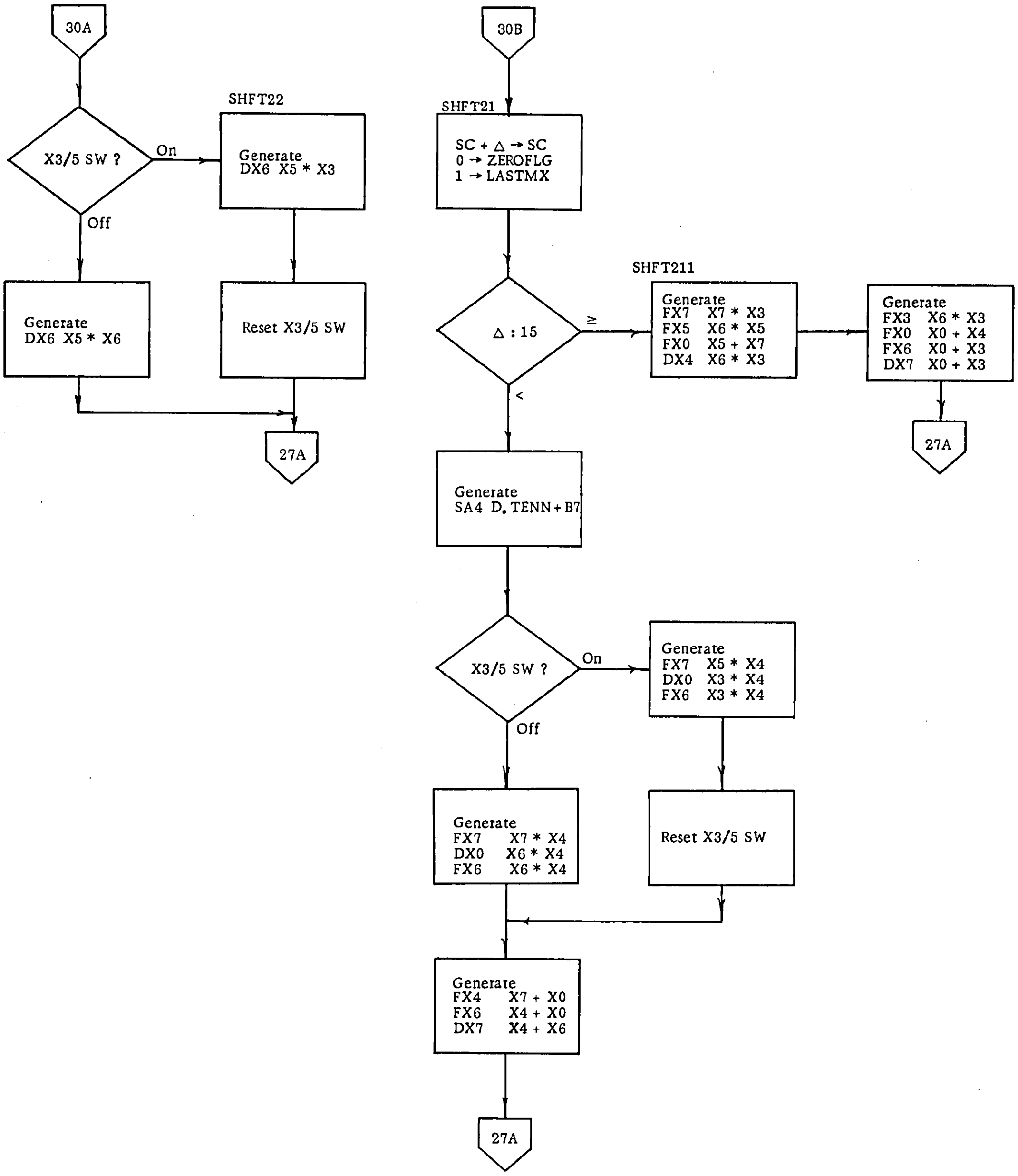


Figure 3-80. GENARTH Flowchart (30 of 33)

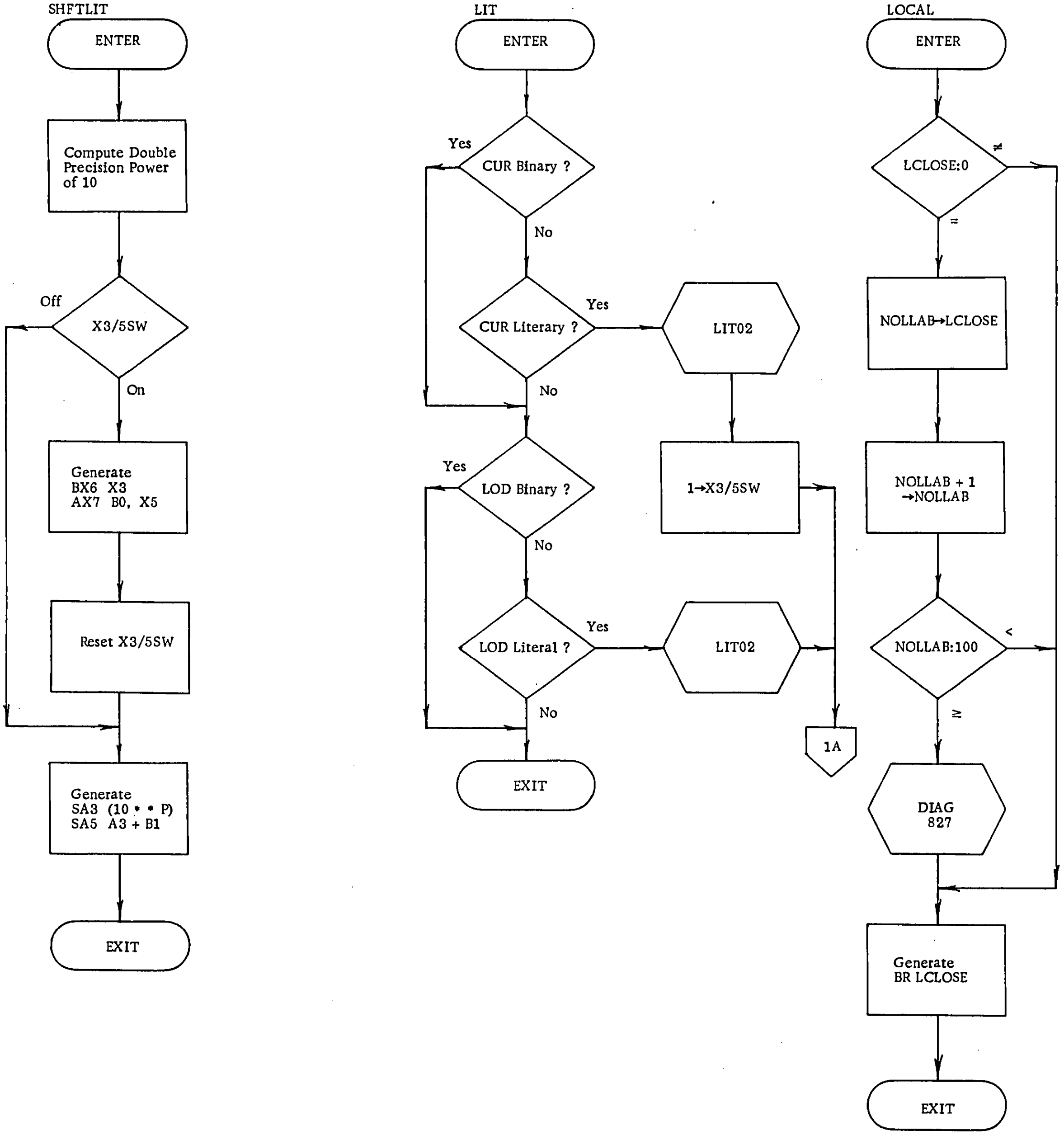


Figure 3-80. GENARTH Flowchart (31 of 33)

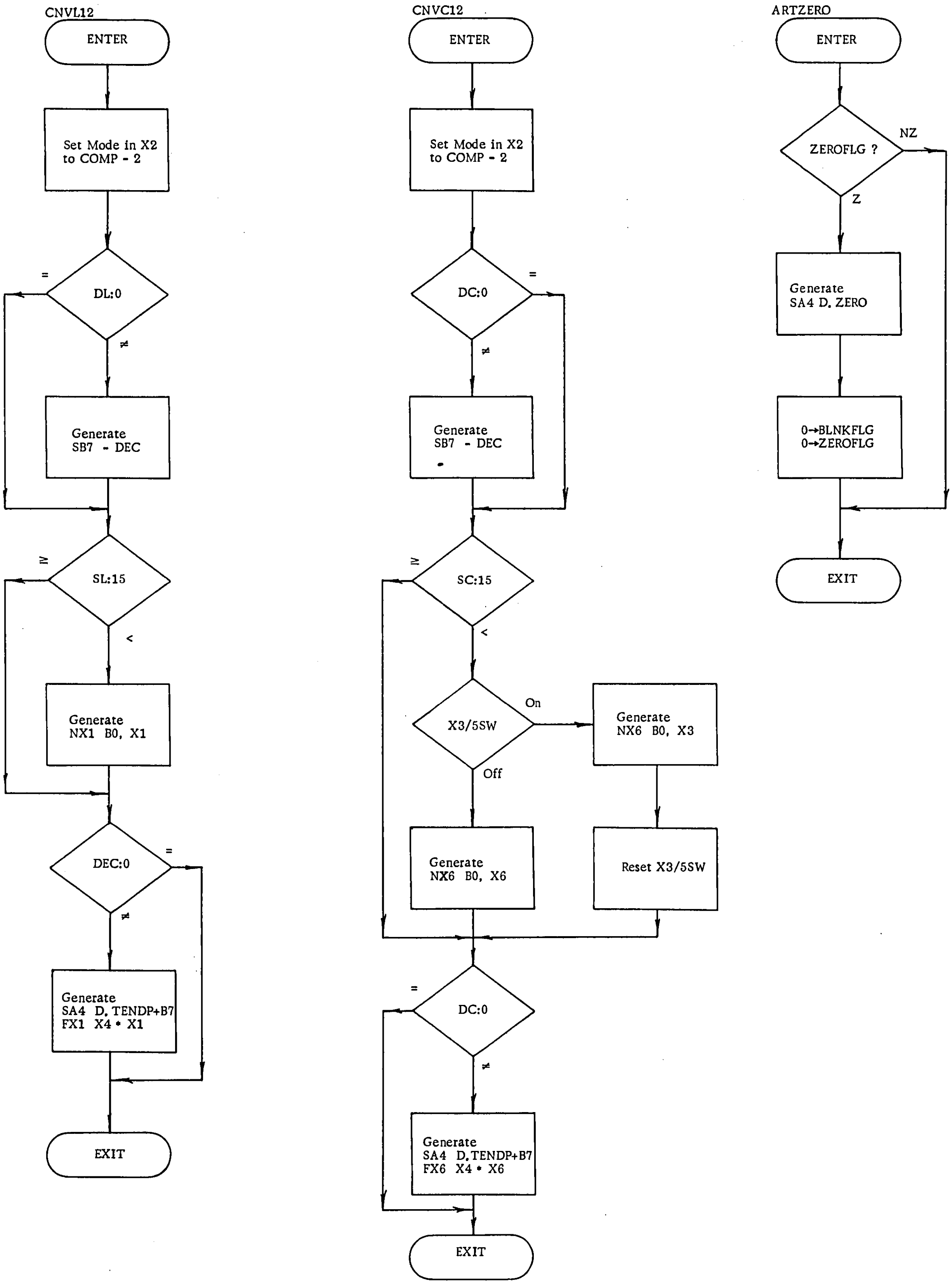


Figure 3-80. GENARTH Flowchart (32 of 33)

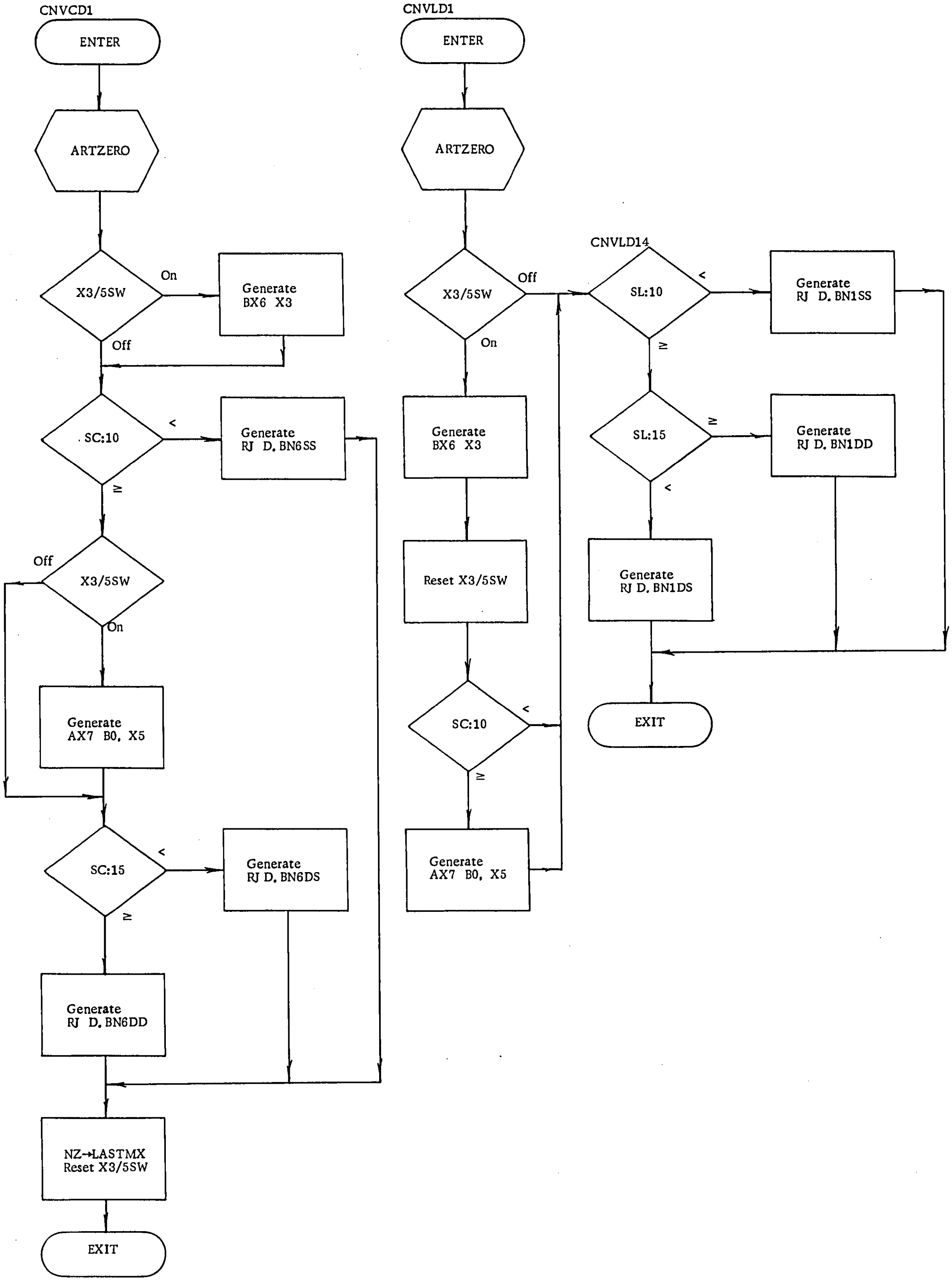


Figure 3-80. GENARTH Flowchart (33 of 33)

SUBSCR

Purpose

To field the subscript leaf, either generating code to compute a subscript or modifying the data pointers in EDD 2, 3 to point to the item addressed by a subscript.

Call

SUBSCR is called by RJ to an entry particular to the calling program. Return is:

1. On the RJ if the subscript is all constant.
2. To the stated point particular to the calling program if the subscript is variable.

Routines Called

GENADDR
LODINT

Operation

The tree is climbed to pick up the EDD (description) of the variable being subscripted. The dimensionality is determined by presence of comma modes.

Where the subscript item is a literal, its contribution to the byte offset is accumulated in SUBSCM. If all elements of the subscript are literals, the BCP of the base item is added to the offset, and a new BCP and origin are computed in EDD format in X1 and X2, and SUBSCR exits on the RJ.

When the first variable subscript is encountered, if the call was from STORE switches are set and STORE is reentered to provide code generation for any shifting, conversion, etc., required. Code is then generated to load the variable subscript by LODINT and to multiply the variable by the dimension information.

Code is generated to add the constant part of the subscript expression, loading it as a literal. The code generated depends upon the requirements of the calling program.

LODINT

Purpose

To generate code to load the item specified by a leaf pointer as a binary (18-bit) integer.

Call

RJ LODINT
    from SUBCR, GO TO, Perform, GENDISP

Calls

GENLOD (perhaps recursively).

Operation

GENLOD is called, saving and restoring its entry point and with switches set to load it directly. Code is then generated to convert the item to a binary integer if necessary.

GENIF

Purpose

GENIF is used during PASS2 to generate all code and labels necessary to do a comparison between two fields or to perform a class test upon one field. (See Figure 3-81.)

Interface

The entry points in GENIF are referenced by the element PASS2. All subroutines except ANDCLMB, ORCLMB, ANDDOWN, ORDOWN, and HOLDCON are called by a direct jump. These five subroutines are called by a return jump. Entry at GENREL preserves all registers except A5 and A7. Entry at other entry points preserves all registers required by PASS2. GENREL requires left leaf information in LFTLEAF and right leaf information in X4.

Elements Called

For numeric comparisons, GENREL calls GENLOD and/or GARTH to load parameters and perform a subtraction for comparison.

Operation

GENREL generates all instructions necessary to perform a comparison. If the comparison is numeric, the numeric fields are loaded and subtracted and tests are made on the result depending upon its type. If one or both of the fields are non-numeric (this includes figurative constants other than zero), instructions for loading the parameters and calling the object-time subroutine D. BCDCM are generated.

For class tests ZERO, NOT ZERO, POSITIVE, NOT POSITIVE, NEGATIVE, and NOT NEGATIVE, one field is loaded and tests are made on the field depending on its type. No code is generated for non-numeric fields.

For class tests NUMERIC, NOT NUMERIC, ALPHABETIC, NOT ALPHABETIC, instructions for loading the parameters and calling either the object-time routine D_1 NUMCM or the routine D. ALPCM are generated. No code is generated for COMPUTATIONAL-1 or COMPUTATIONAL-2 fields.

For switch tests, instructions to test the correct bit of RA + 0 are generated.

IFMSTR presets local labels and fields moded by GENREL.

TFLFORK defines a true point if one is present.

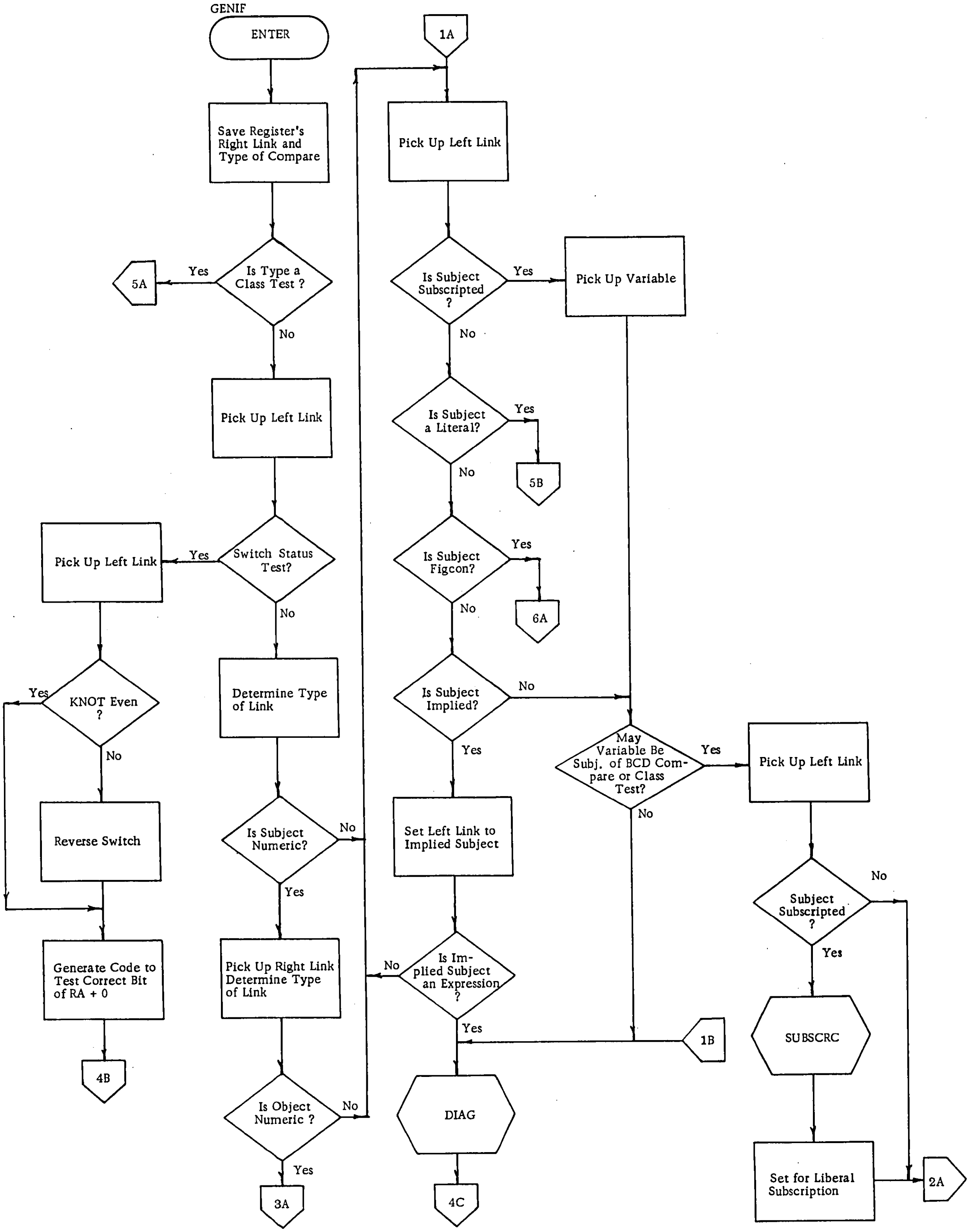


Figure 3-81. GENIF Flowchart (1 of 9)

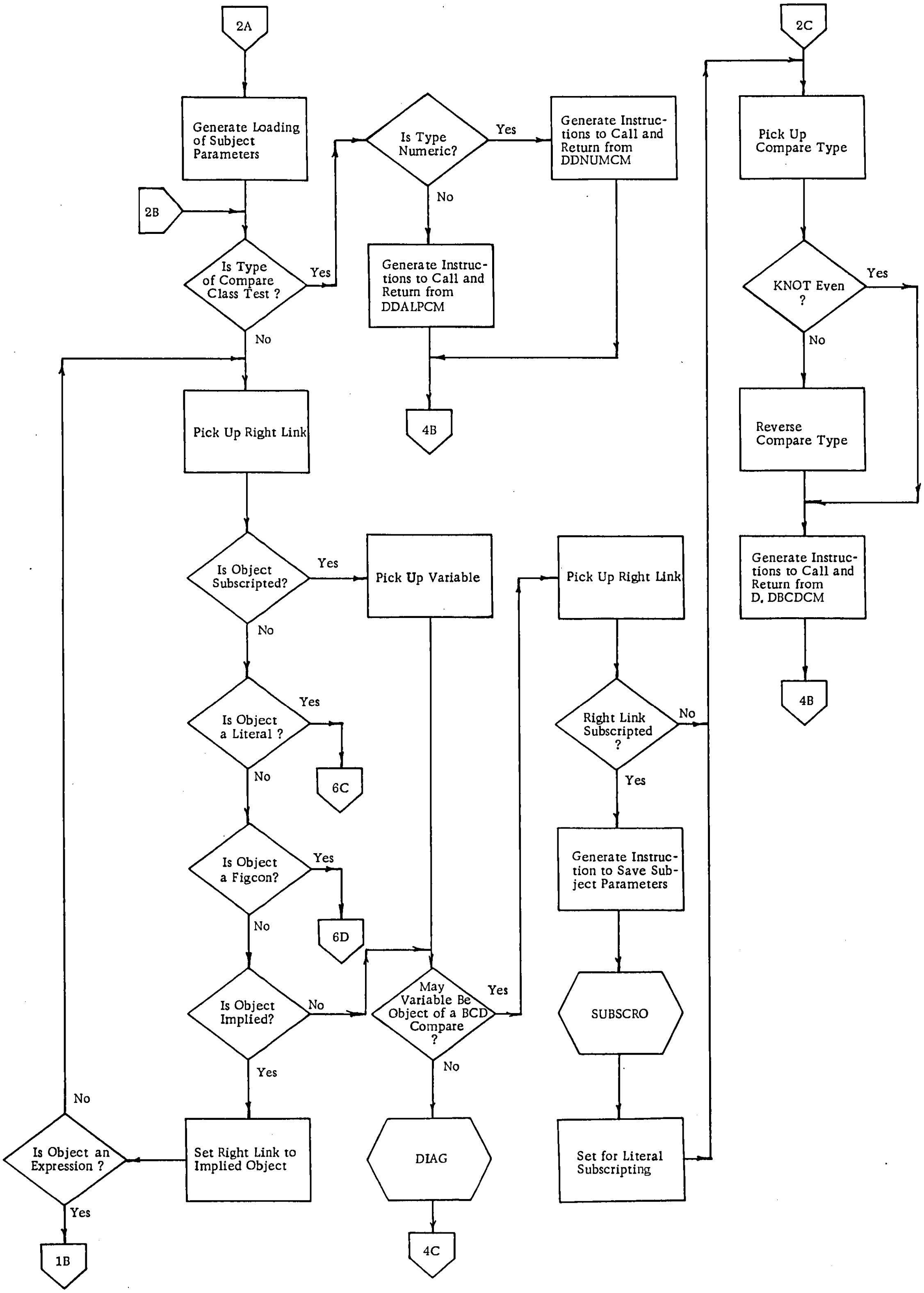


Figure 3-81. GENIF Flowchart (2 of 9)

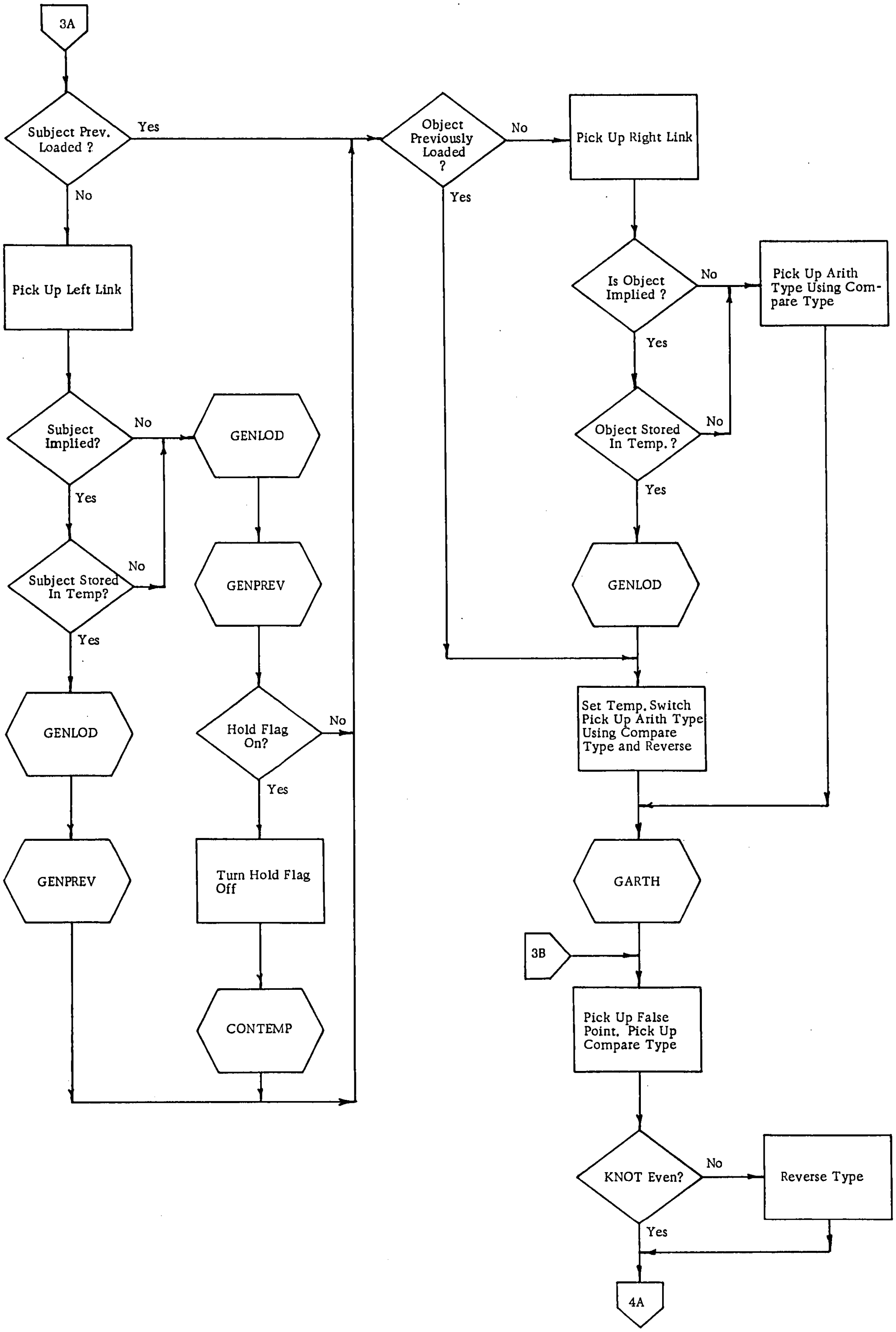


Figure 3-81. GENIF Flowchart (3 of 9)

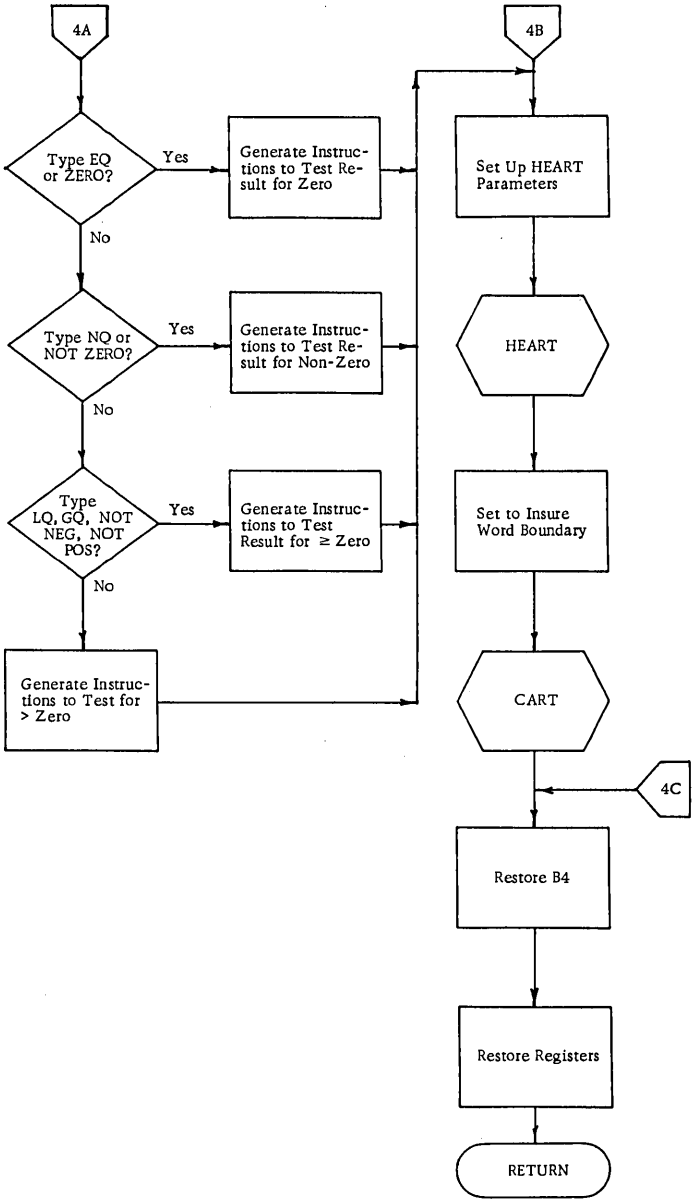


Figure 3-81. GENIF Flowchart (4 of 9)

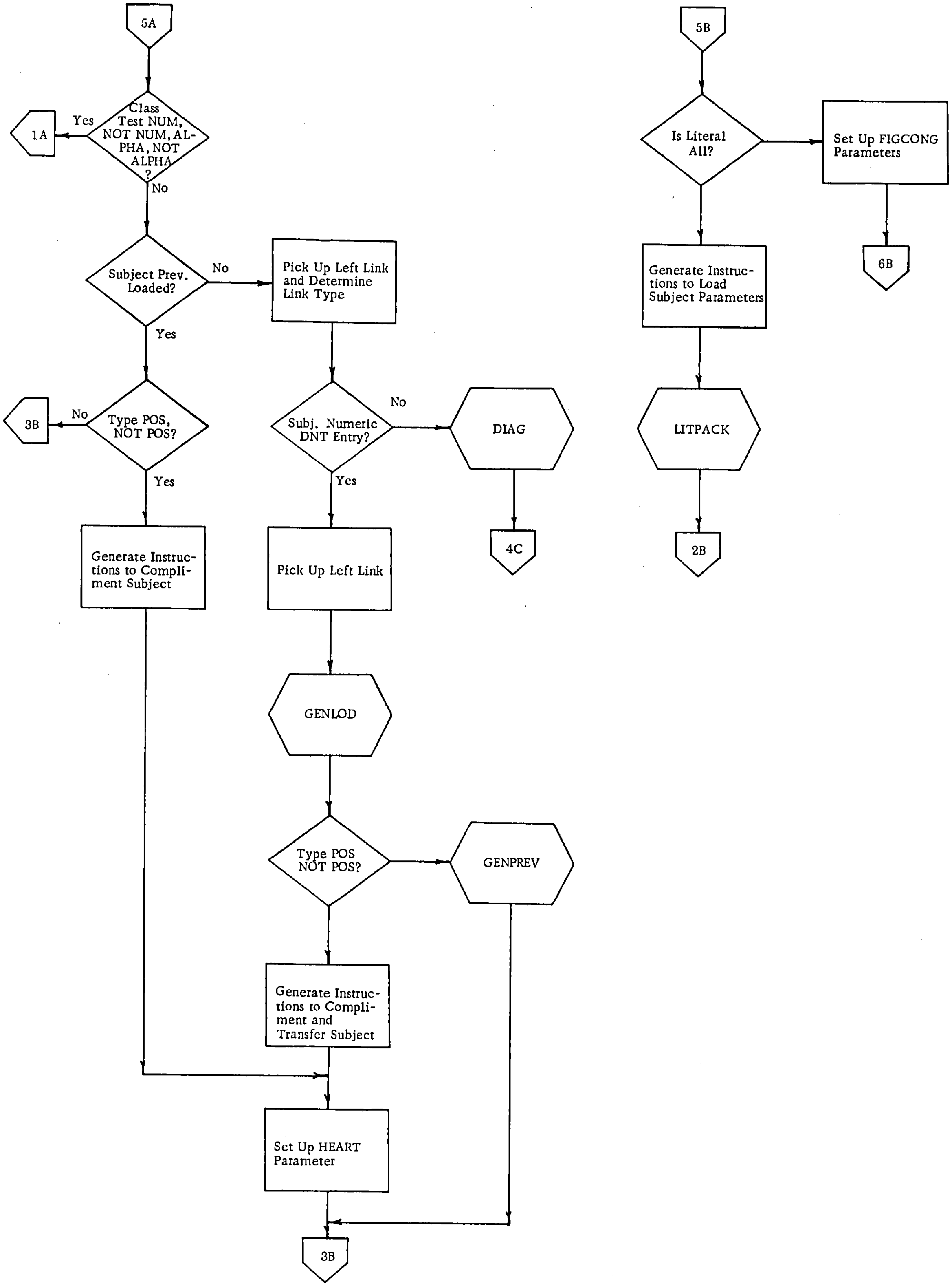


Figure 3-81. GENIF Flowchart (5 of 9)

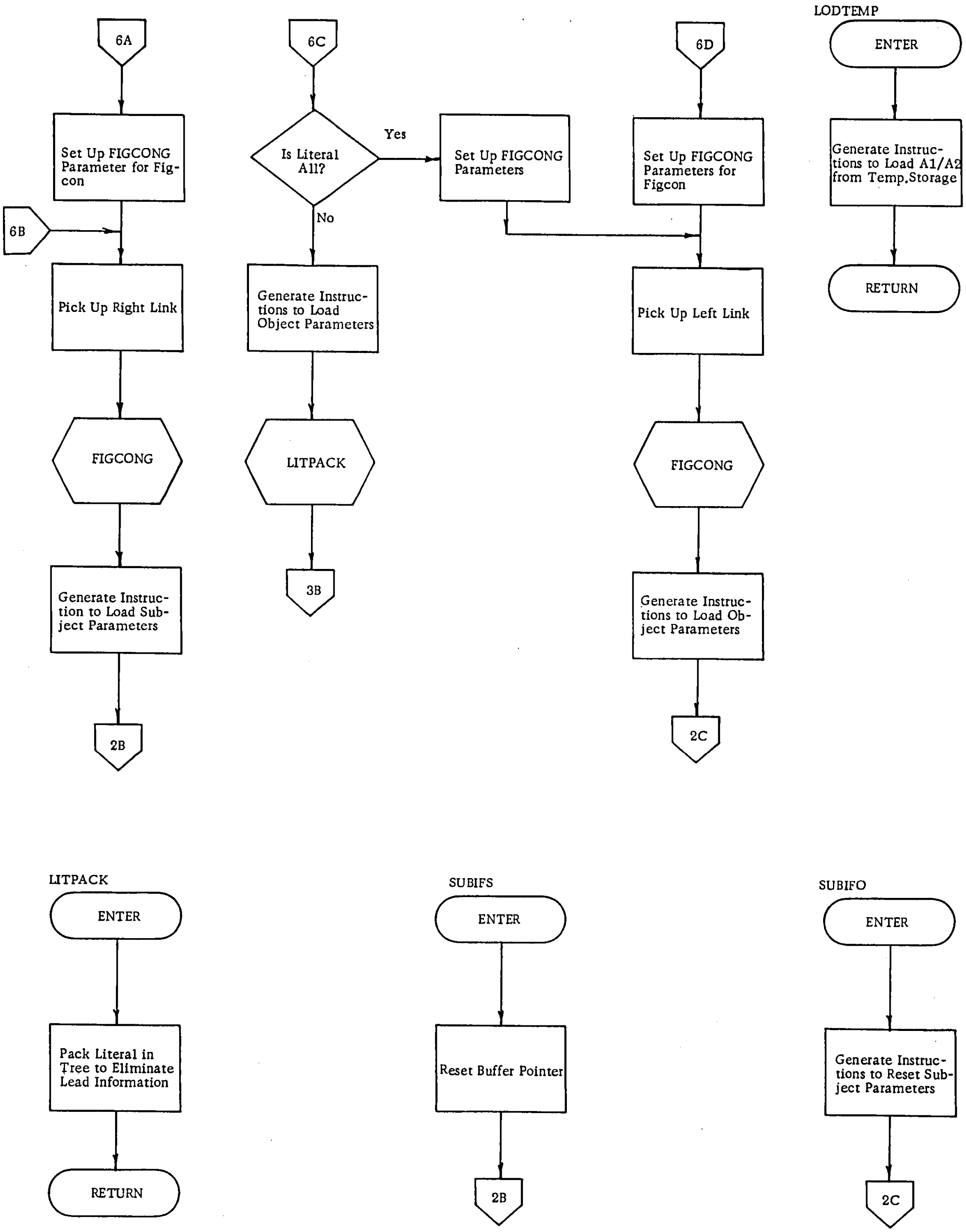


Figure 3-81. GENIF Flowchart (6 of 9)

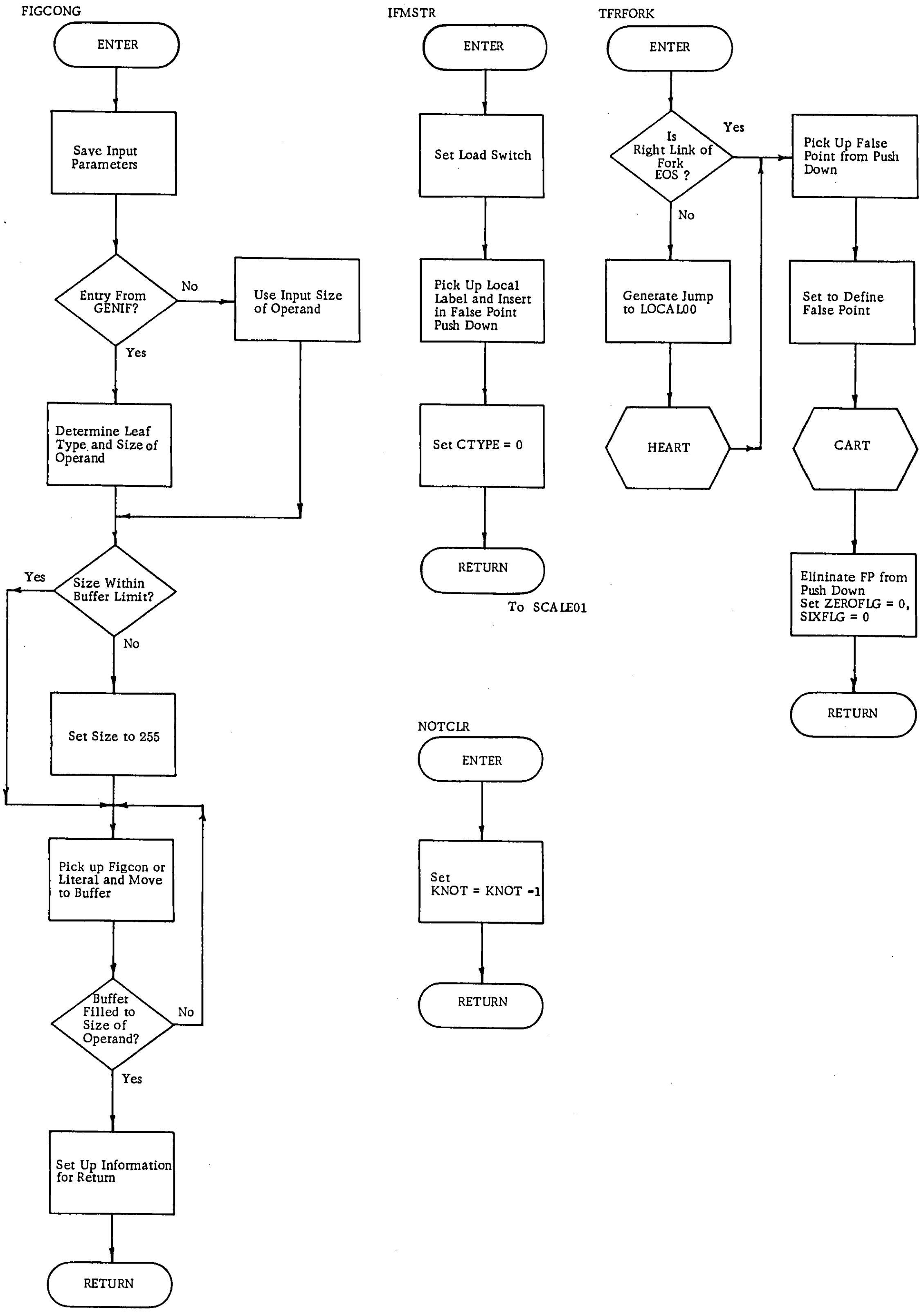


Figure 3-81. GENIF Flowchart (7 of 9)

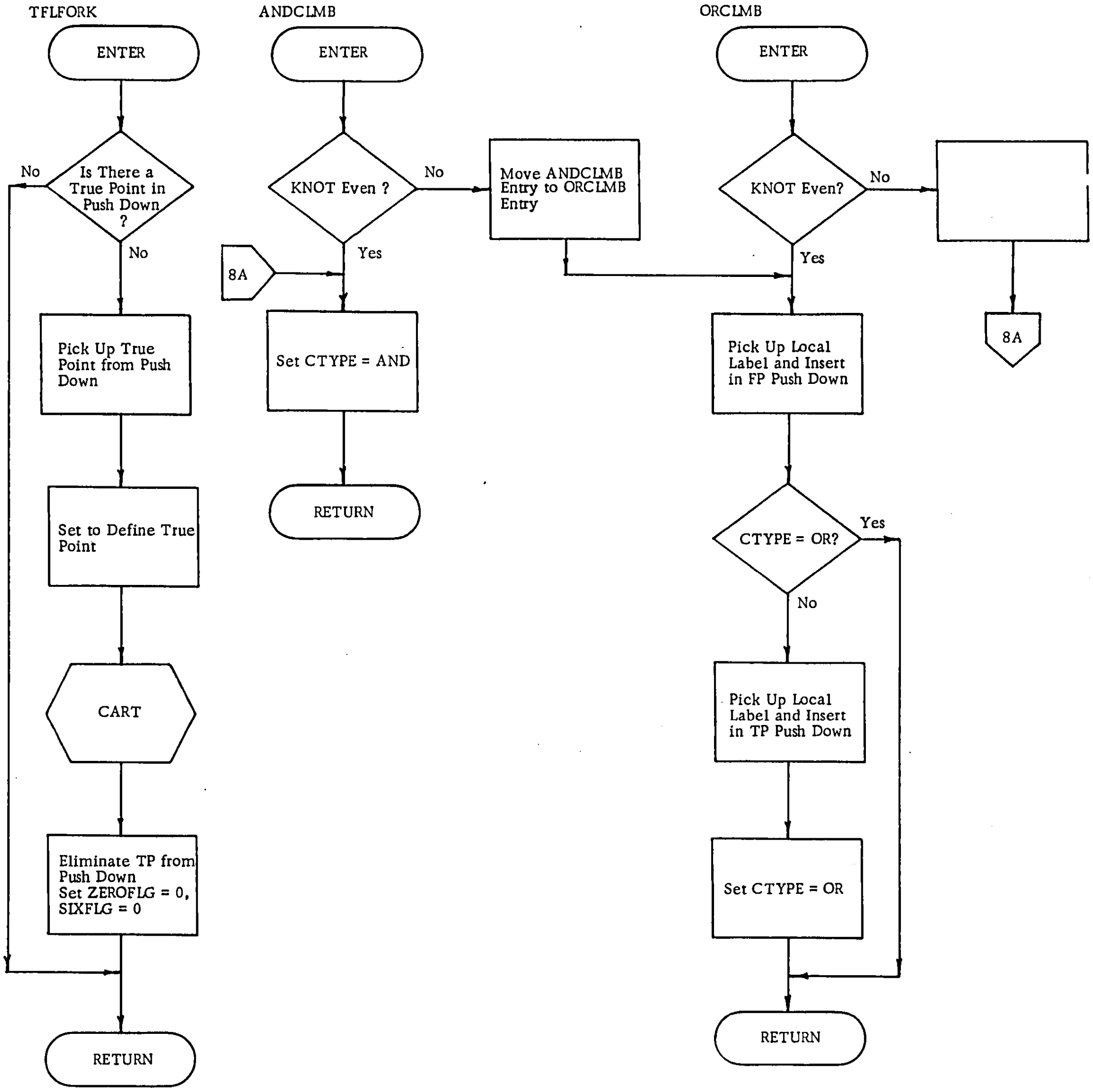


Figure 3-81. GENIF Flowchart (8 of 9)

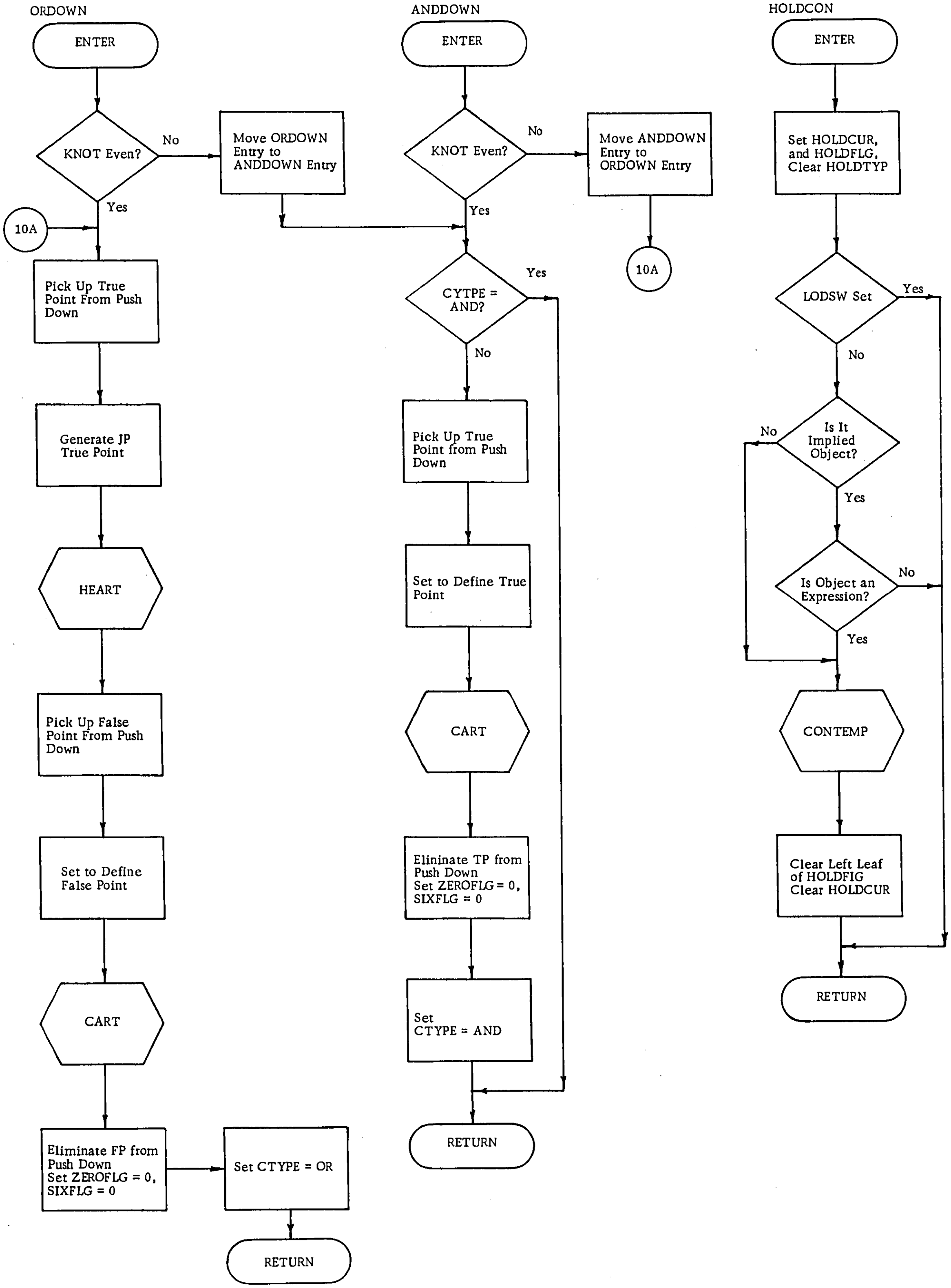


Figure 3-81. GENIF Flowchart (9 of 9)

TFRFORK generates a jump to LOCAL00 unless the right link of the fork is an EOS. It always defines a false point.

NOTCLR reduces the not flag KNOT by 2.

ANDCLMB tests KNOT and jumps to ORCLMB if KNOT is odd. Otherwise it sets CTYPE to AND.

ORCLMB tests KNOT and jumps to ANDCLMB if KNOT is odd. Otherwise it inserts a false point in the push down and, if CTYPE is not OR, inserts a true point in the push down and sets CTYPE to OR.

ORDOWN tests KNOT and jumps to ANDDOWN if KNOT is odd. Otherwise it generates a jump to the last true point in the push down, defines the last false point in the push down and set CTYPE = OR.

ANDDOWN tests KNOT and jump to ORDOWN if KNOT is odd. Otherwise, if CTYPE is not AND, it defines the last true point in the push down and sets CTYPE = AND.

HOLDCON presets flags to store the fields in temporary to be used as an implied subject or object.

LIT02 SUBROUTINE (COMPILER)

Purpose

This compiler subroutine formats a designated PROCEDURE literal to conform to the requirements of the operand and the operation to be performed jointly on the operand and literal. Code is generated in HEART format and stored in the Pass 2 assembler buffer. The code generated will cause a load of the resulting literal into registers X1, X2, or X6, X7, as indicated by the input parameters. The HEART word for the first generated load instruction will contain information which will cause the assembler to store the resulting literal as one of the object code literals and supply the loading address for the load instruction. (See Figure 3-82.)

Subroutine entered by an

```
RJ      LIT02
```

Input Parameters

PROPCUR	contains operand descriptor or a literal descriptor for literal which is to be loaded into X6 and X7
PROPLOD	contains operand descriptor or a literal descriptor for literal which is to be loaded into X1 and X2.
Register B7	contains operation code.
Register A6	contains location of last item stored in assembler buffer.

Output

PROPCUR	unchanged if it contained operand descriptor. New literal descriptor if it contained the literal descriptor.
PROPLOD	unchanged if it contained operand descriptor. New literal descriptor if it contained the literal descriptor.
Register B7	unchanged
Register A6	incremented by one for each HEART instruction stored in the assembler buffer via an SA6 A6+B1 instruction.

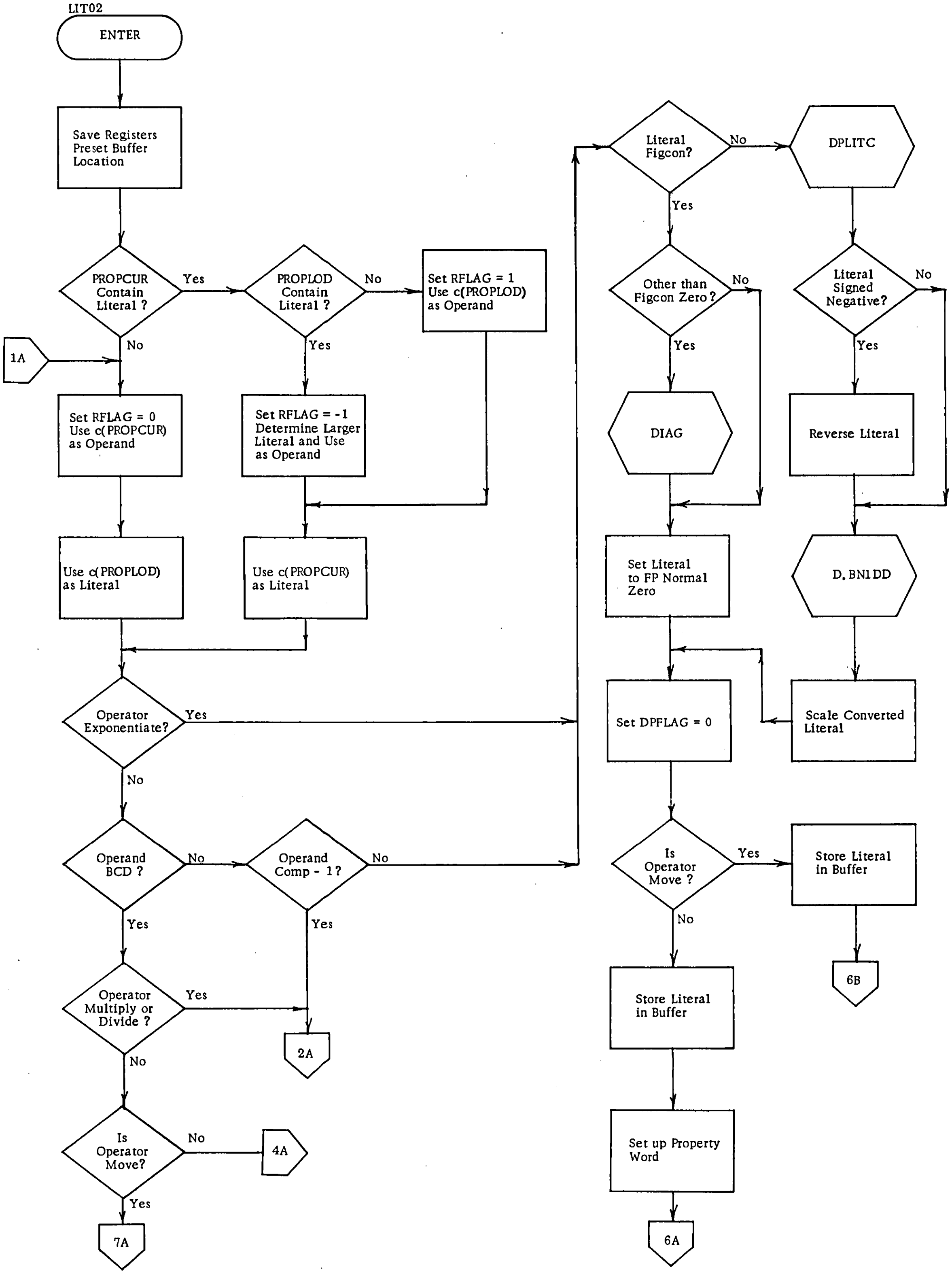


Figure 3-82. LIT02 Flowchart (1 of 8)

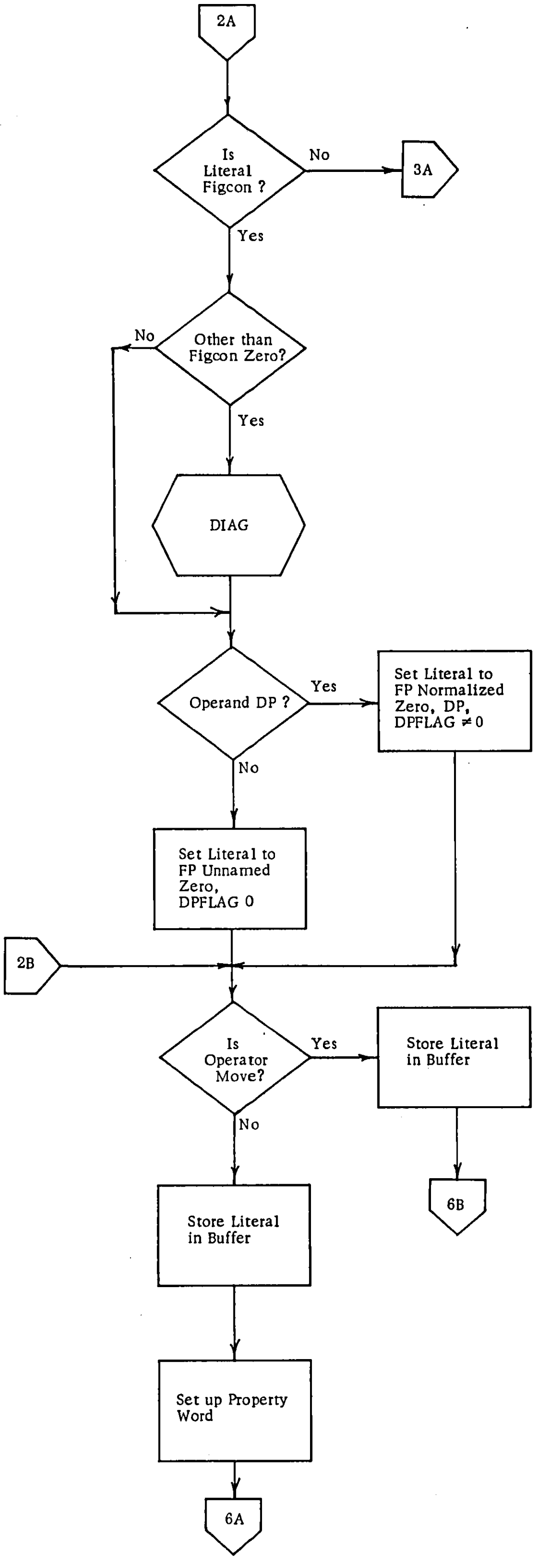


Figure 3-82. LIT02 Flowchart (2 of 8)

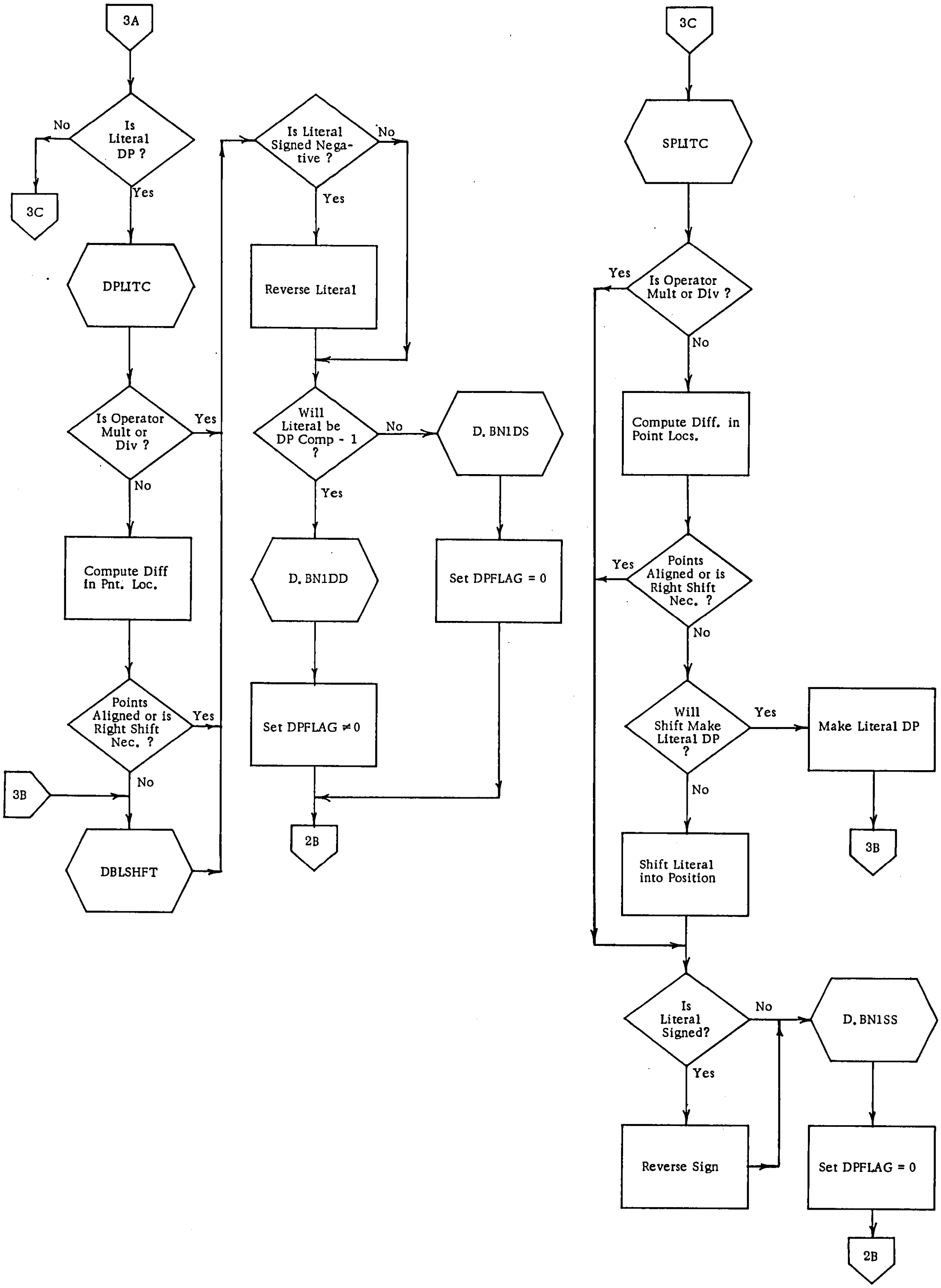


Figure 3-82. LIT02 Flowchart (3 of 8)

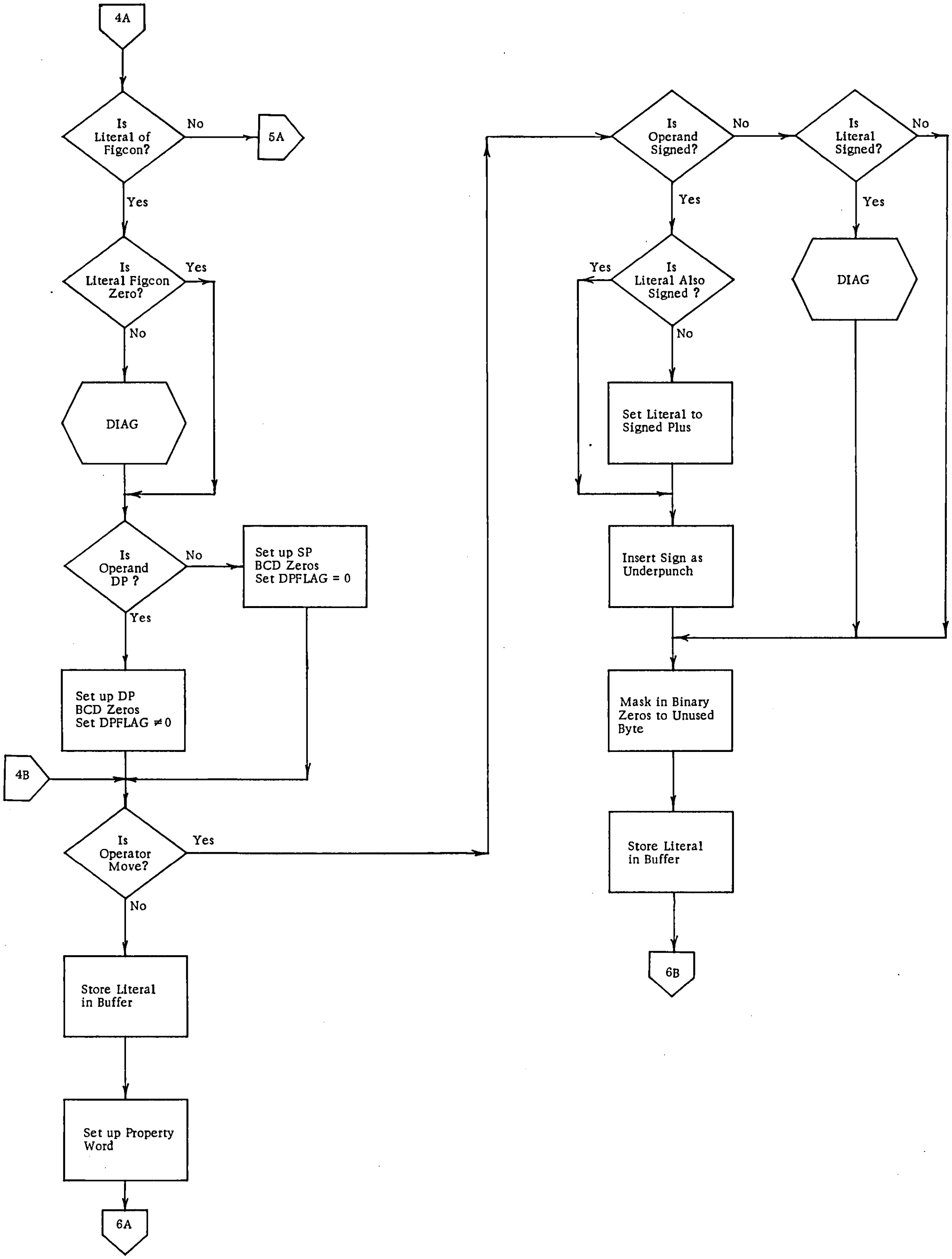


Figure 3-82. LIT02 Flowchart (4 of 8)

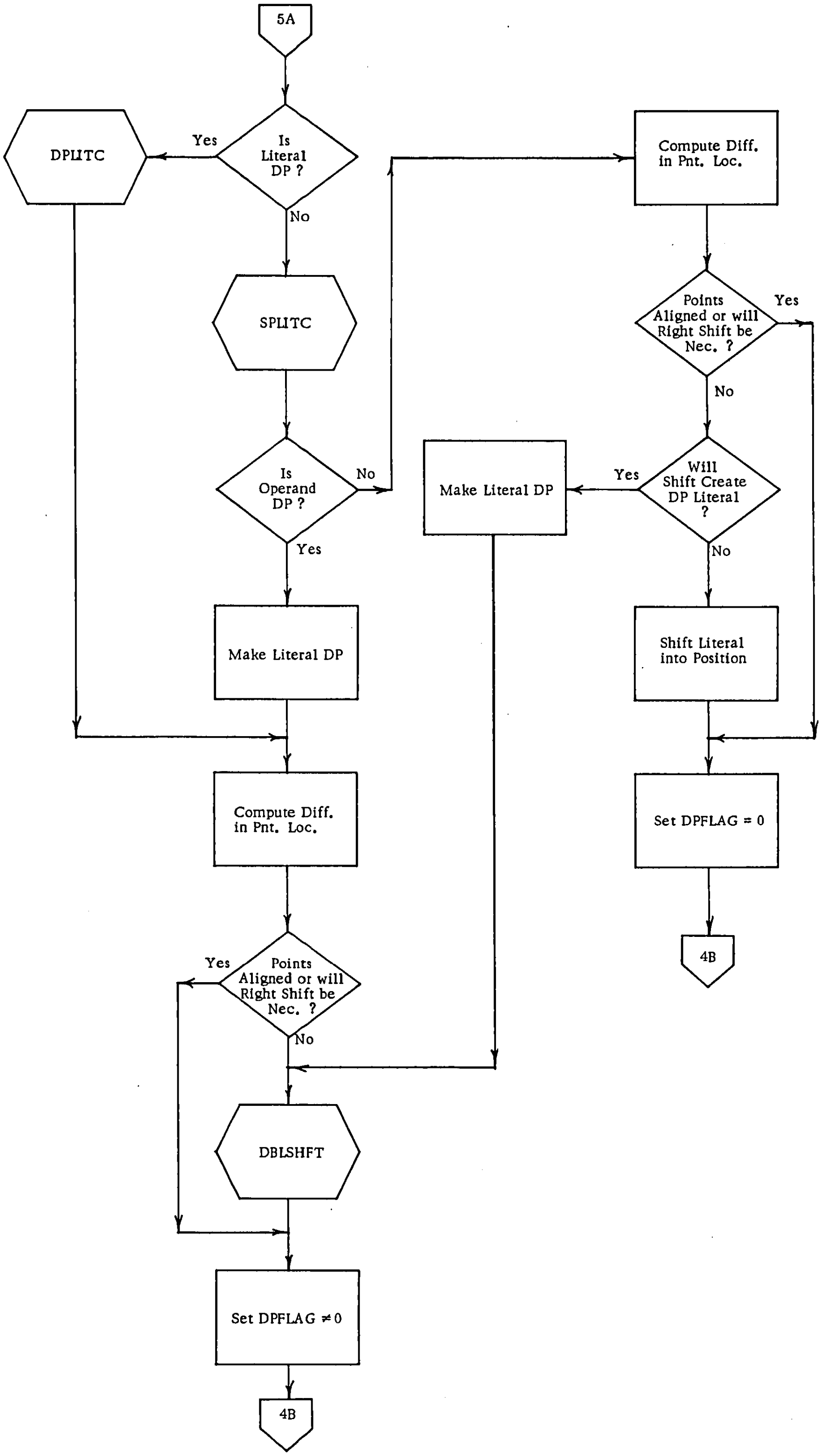


Figure 3-82. LIT02 Flowchart (5 of 8)

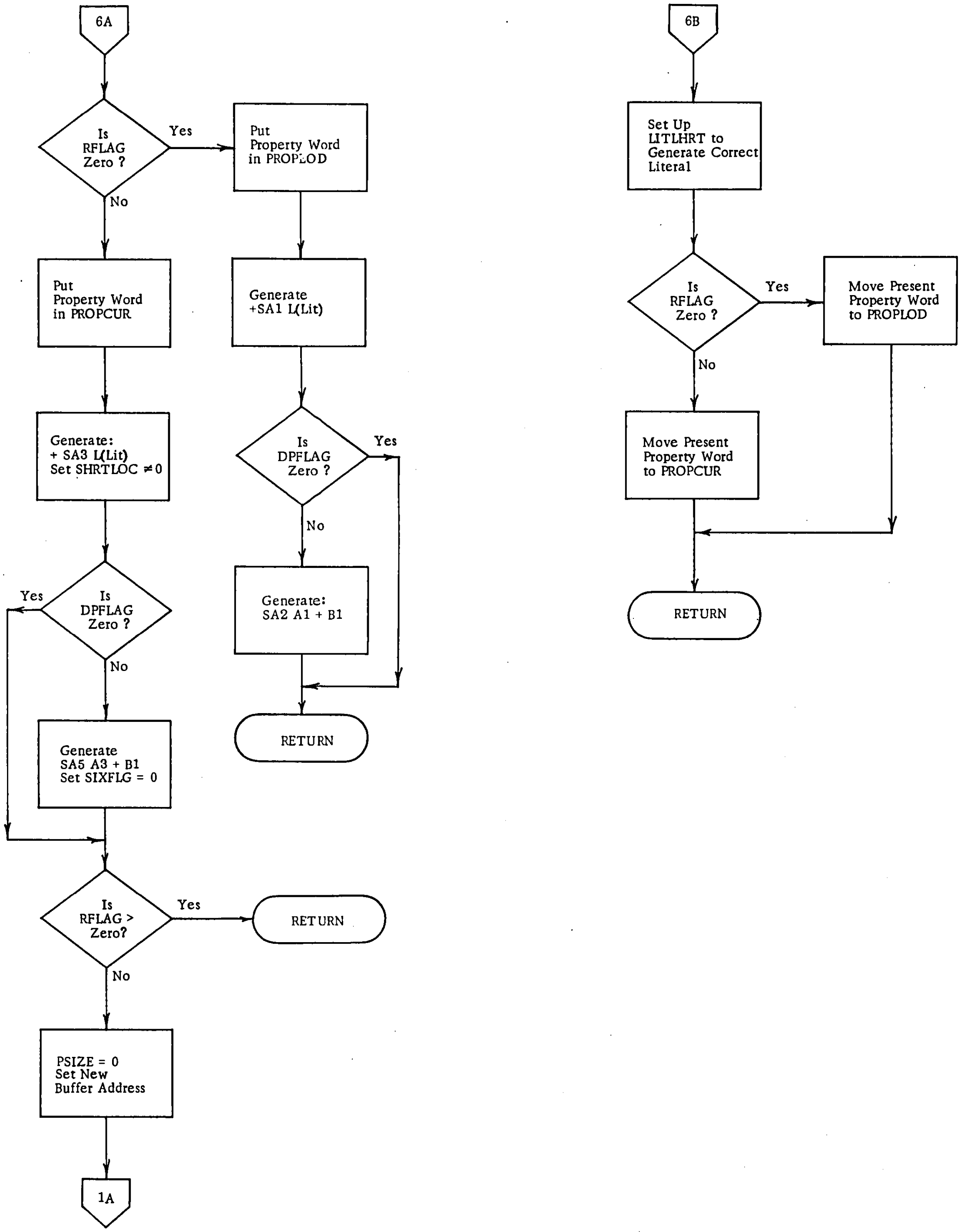


Figure 3-82. LIT02 Flowchart (6 of 8)

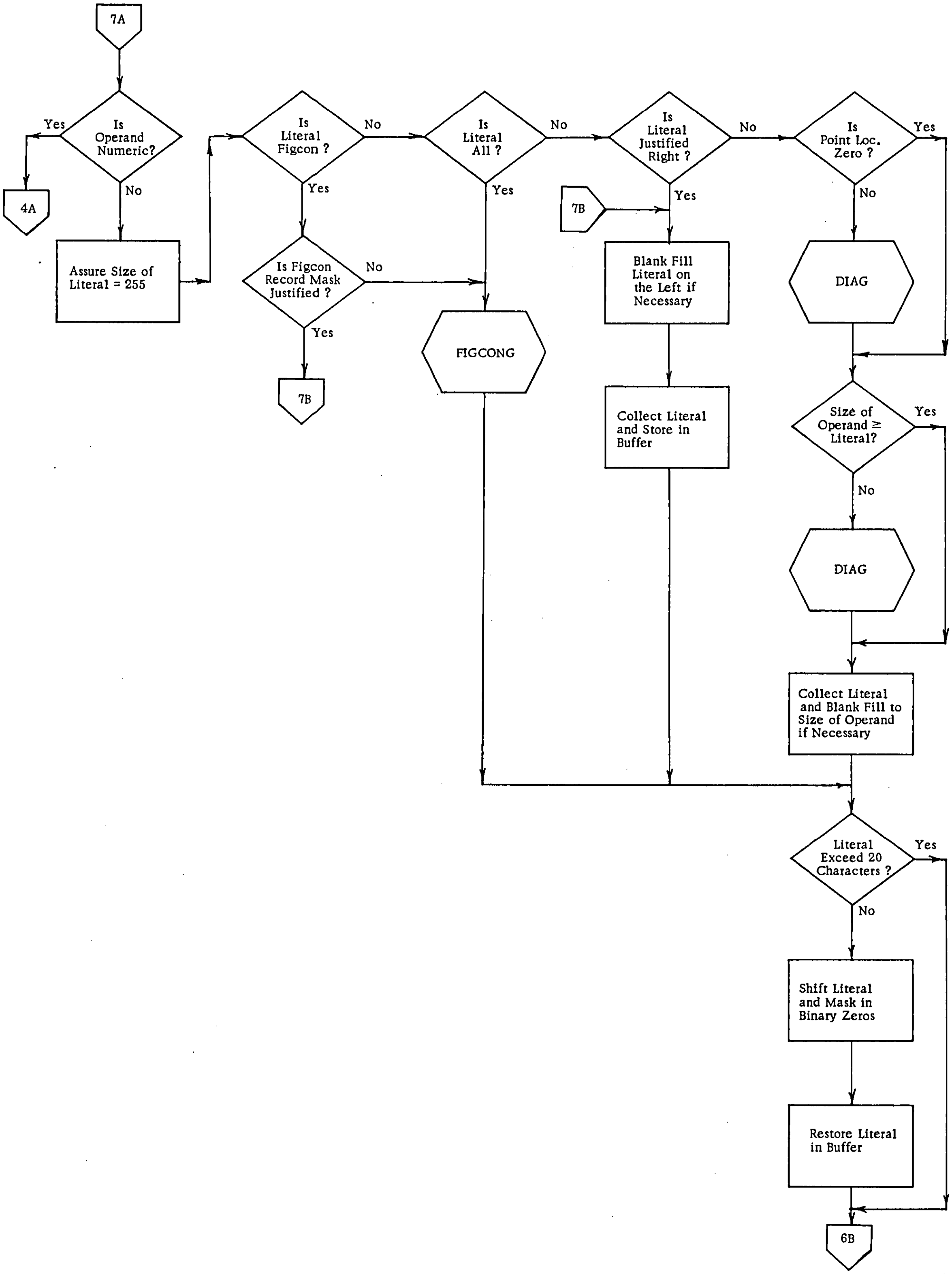


Figure 3-82. LIT02 Flowchart (7 of 8)

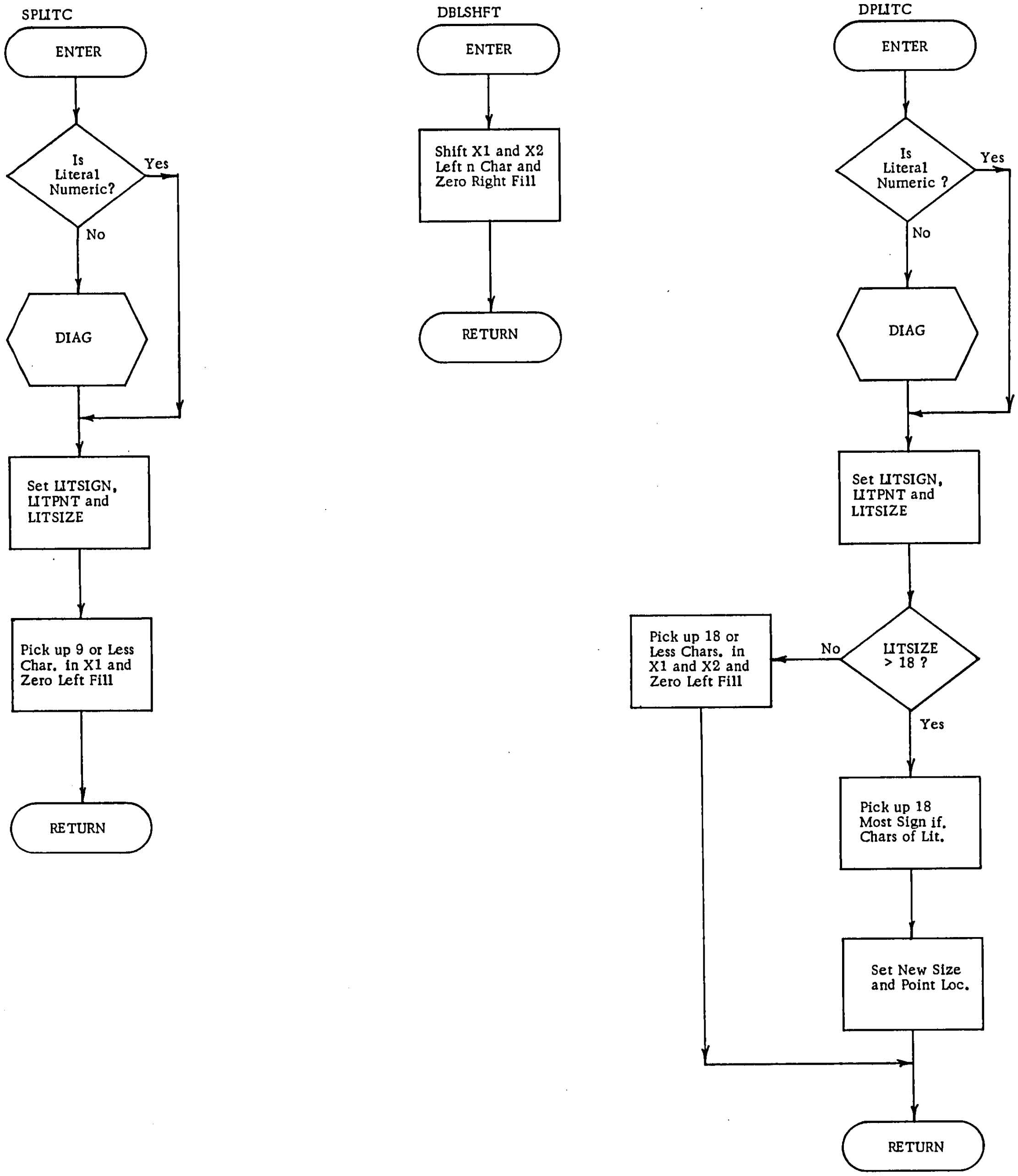


Figure 3-82. LIT02 Flowchart (8 of 8)

Generated Code

The following code will be generated for object time execution when the literal descriptor is in PROPCUR:

Single-precision load

```
SA3 (assembler-supplied address at literal) - 30-bit instruction
5130000000
BX6     X3                                  - 15-bit instruction
10633
```

Double-precision load

```
SA3 (assembler-supplied address of literal) - 30-bit instruction
5130000000
SA5     A3+B1                               - 15-bit instruction
54531                                       - 15-bit instruction
```

The following code will be generated for object time execution when the literal description is in PROPLOD:

Single-precision load

```
SA1 (assembler-supplied address of literal) - 30-bit instruction
5110000000
```

Double-precision load

```
SA1 (assembler-supplied address of literal) - 30-bit instruction
5110000000
SA2     A1+B1                               - 15-bit instruction
54211
```

At object time, register B1 is assumed to contain the integer 1, and the X4 register has all BCD blanks or zeros as appropriate for the operand.

An operator type of $11)_8$ (MOVE) is considered a special case. Input and output are the same as for other operators except that A6 need not point to an instruction buffer since no code will be generated. (A6 will not be restored, however). Instead a HEART word is set up in the cell LITCHRT. All fields will be filled in except the instruction field itself. The calling routine is responsible for insuring that no assembler restrictions are violated.

The literals setup fall into five categories.

1. Numeric Display - Single or double precision (size 19 or less), right justified in the field, sign if present appears as an underpunch in the rightmost digit, left fill is BCD zeros to the size of the receiving field, binary zeros to 10 (for single precision) or 20 (for double precision) characters.

2. COMPUTATIONAL-1 - As for other operators.

3. COMPUTATIONAL-2 - As for other operators.

4. Non-numieric - Single precision (size 9 or less) or double precision (size 19 or less), blank filled on the right to size of the receiving field, left fill is binary zeros to 10 or 20 characters, right justified in the field.

5. Non-numeric literals (or numeric literals moved to fields ≥ 20 characters) - Blank filled on the right to size of the receiving field, left justified in the field.

Registers Used (Compile Time)

A6 is used to store in Pass 2 assembler buffer. All other A registers may be destroyed and are not restored on exit from subroutine.

All X and B registers used in this subroutine are saved on entrance to this subroutine and are restored on exit from same.

B1 is assumed to contain the integer 1 on entrance to this subroutine.

Operation Code (contents of register B7 at compile time) in octal

00 = + (PLUS)
01 - (MINUS)
02 X (Multiplied by)
03 /(Divided by)
04 **(Exponentiated by)
07 /(Divided into)
10 -(Subtracted from)
11 MOVE

General Processing Rules

1. Non-numeric literals can only occur with a move operation 11_8.

2. No truncation of a literal is to occur regardless of point locations and relative sizes.

3. Align point location of literal is left to match that of the operand if operand is not COMP-2.
4. If operand is COMP-1, then literal is COMP-1 and may be double precision even though the operand is single precision.
5. If operand is double precision COMP-1, the resulting literal may be single precision.
6. If operation is add, subtract, or move, and operand is not COMP-2, point location of literal is aligned to match that of the operand.
7. No alignment of point location is needed when operation is not add, sub or move.
8. If operation is not add, subtract, or move, the resulting literal will be COMP-1, unless the operand is COMP-2.
9. If operand is double precision BLD the resulting literal must be double precision.
10. No right shifts of the literal will be made except in the case of a move operation.

Special Cases

If both PROPCUR and PROPLOD contain literals, the one with the larger size is assumed to be the operand, the descriptor (properties) word is remade, and the other literal is aligned to match the larger.

CONTROL TRANSFERS

The simplest control transfer used is a jump instruction. A more complex mechanism is needed to handle interoverlay and intercompilation transfer of control (and return transfer for PERFORM).

Every overlay starts with a jump table. Any control transfer into the overlay jumps to one of these jumps. Every compilation has a table in its base section consisting of "index" words each describing an entry point for a control transfer.

A zero overlay number indicates the location is absolute. These words are used as parameters to a routine, SOL, which controls overlay loading, interoverlay transfers, and intercompilation transfers.

Every procedure name declared by an ENTRY statement to be external to this compilation is used as a symbol on the first of a pair of indexes. The first index of this pair is a variable and holds the exit pointer (initially to the following procedure). The second word of the pair is a constant and is the entry point index for this procedure.

Code Generated for GO

If this is an intraoverlay transfer only, a jump is used. If this is ALTERed but still only within the same overlay, it is left-justified in a full computer word.

If this is an interoverlay or intercompilation GO or is ALTERed from another overlay, an index word is reserved in the index table for it, and SOL must be called to execute the jump.

Code Generated for GO TO ... DEPENDING ON ...

After the switch variable is loaded in binary form, there is a test to determine whether the switch variable is between 1 and n inclusive. If it is not, a transfer to the next statement occurs. If so, an indexed jump occurs to one of n branch points. Each full word branch is either a jump or an index-type GO TO.

Code Generated by ALTER

If the GO generates a jump (see above) ALTER replaces that jump. Otherwise, ALTER replaces an index in the index table with a new index.

Code Generated by PERFORM

Table 3-11 below illustrates the functions of the PERFORM statement before control is transferred and after it is restored.

Table 3-11. PERFORM Code

Before Control is Transferred	1. Saves the previous exit (normally a drop-through).	Refers to an index (if an index is used to effect a return). If not, refers to a JUMP instruction.
	2. Sets the exit to return control.	Processed as an ALTER statement.
	3. Sets up a transfer of control to the routine.	Processed as a GO TO statement.
After Control is Restored	4. Restores the exit.	Refers to an index (if an index is used to effect a return). If not, refers to a JUMP instruction.

GOTOGEN SUBROUTINE

Purpose

Code generator for GO TO (node: 661) and IMPLIED GO TO (node: 663). (See Figure 3-83.)

Both the GO TO and IMPLIED GO TO nodes are handled in the same manner.

GOTOGEN determines which indexes (entry and/or exit) both the origin and object of the jump have.

Depending on which of these are set, different sets of code are generated.

Calling Sequence

Called by Pass 2 with the following registers containing:

X2	GO TO node
B2	Base of tree

Routines Called

BUG
HEART
CART
REGSAVE, REGRSTR
SCALE08
GENLOD

External Tables and Pointers Referenced

SECPNTR
BEGXTAB
PROPLOD

Output

Output is generated code to the assembler. Registers are returned with B2 intact and X2 the core image.

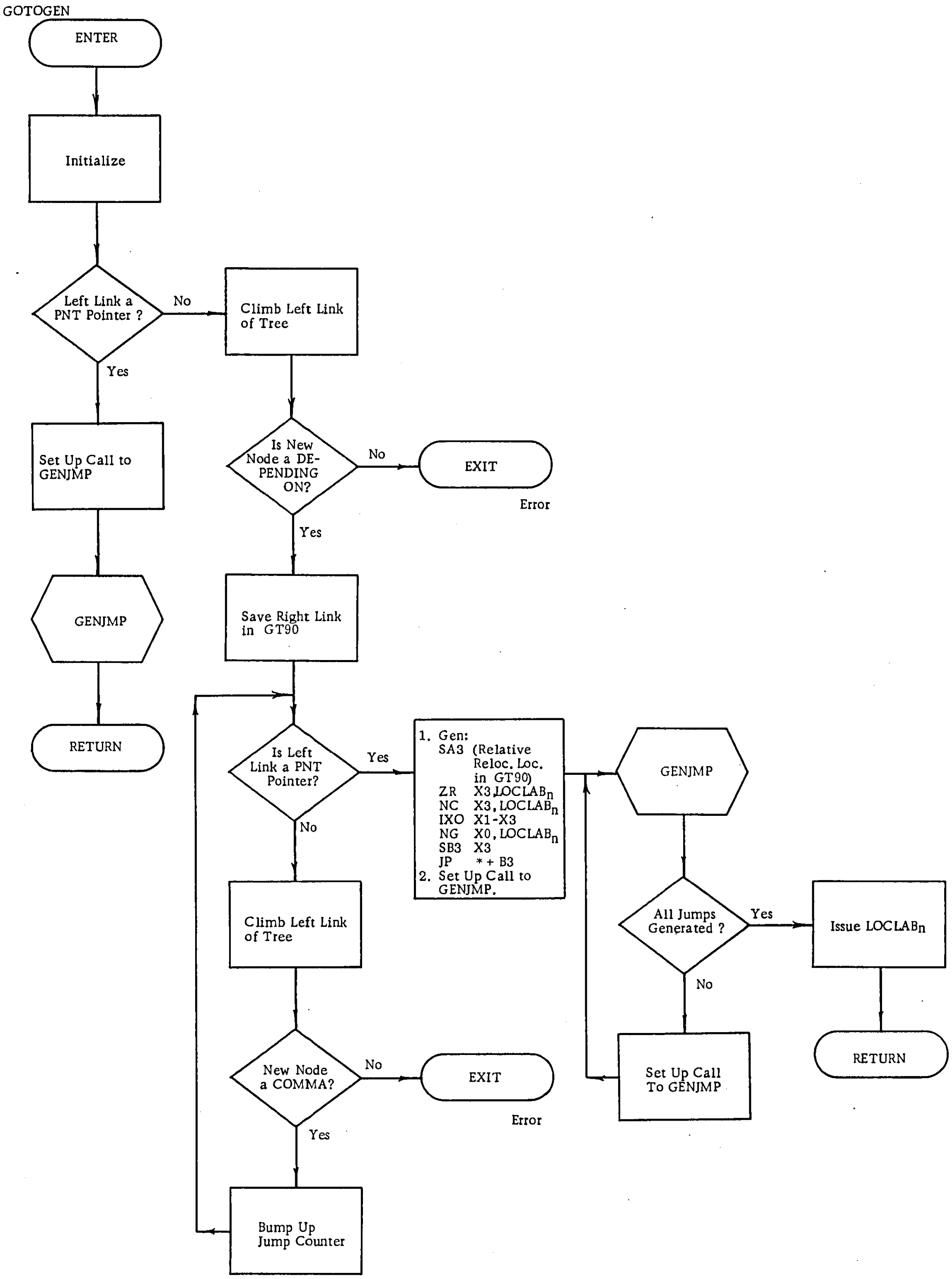


Figure 3-83. GOTOGEN Flowchart (1 of 2)

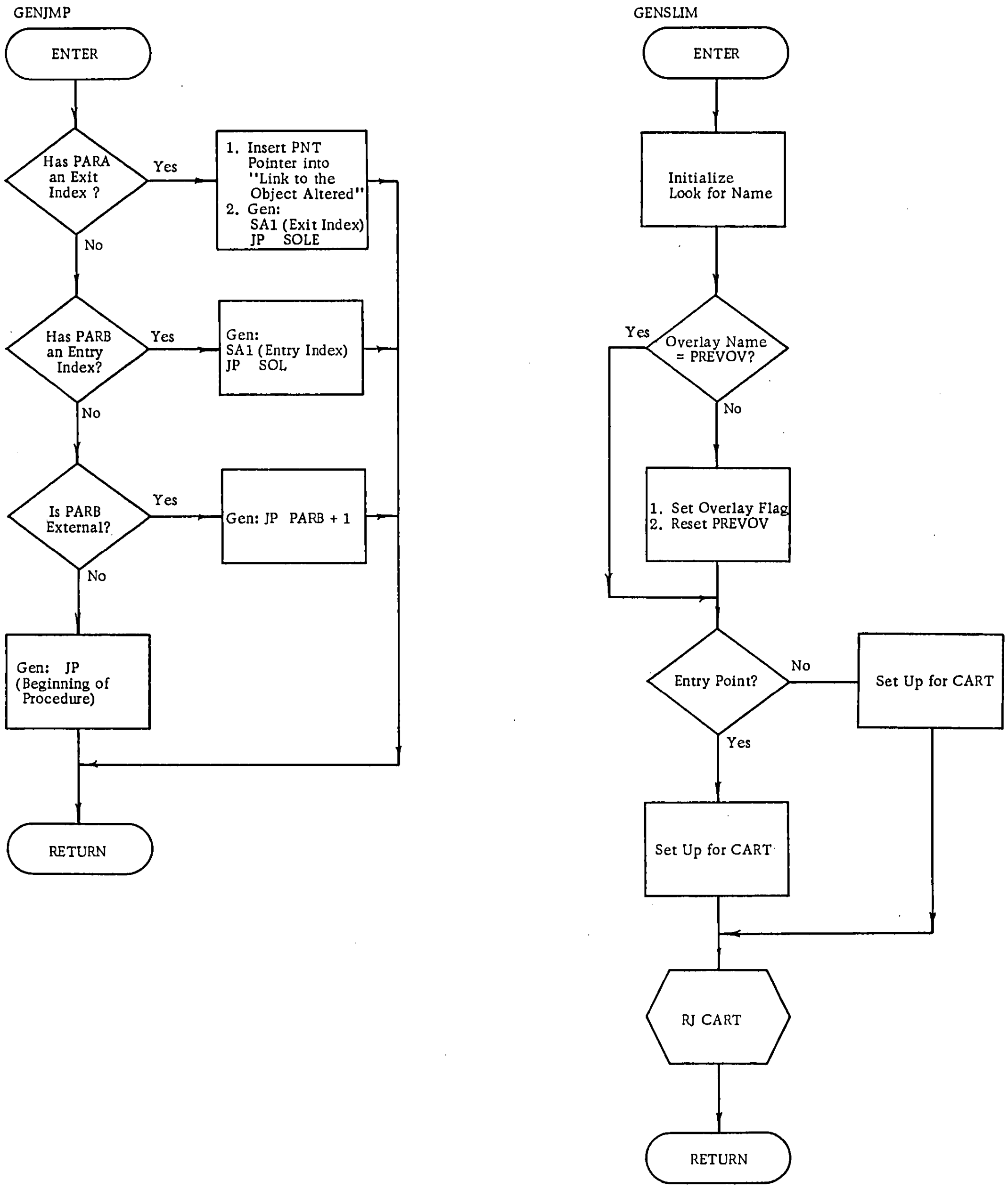


Figure 3-83. GOTOGEN Flowchart (2 of 2)

GOTOGEN issues a diagnostic (via BUG) when an unexpected node is found in the trees.

Example: PARA.

GO TO PARB.

Case 1: PARA has exit index

Generate: SA1 (beginning, of index table and exit index)
JP SOLE

Case 2: PARB has entry index

Generate: SA1 (beginning of index table and entry index)
JP SOL

Case 3: PARB is external (undefined in this compilation)

Generate: JP PARB+1

Case 4: If none of the above

Generate: JP PARB

The DEPENDING ON clause of GO TO evokes the generation of the following code:

```
- load variable and convert to a binary integer -
SA3     literal = count of number of jumps in source
ZR      X1, LOCLAB2
NG      X1, LOCLAB2
IXO     X3-X1
NG      X0, LOCLAB2
SB3     X1
JP      B3+*
- jumps to variables specified in source code -
```

<u>Restrictions</u>

The DEPENDING ON clause may not have more than 200 procedure names specified.

ALTRGEN - CODE GENERATOR FOR ALTER VERBS (NODE 660)

Purpose

ALTRGEN examines the object being altered and the new procedure to which the new jump must proceed. Depending on the entry and/or exit indexes associated with these two, code is produced. (See Figure 3-84.)

ALTRGEN is called by Pass 2 with the following registers containing:

X2	Alter node
B2	Base of tree

Routines Called

CART
HEART
SCALE08
BUG
REGSAVE, REGRSTR

External Pointers and Tables Referenced

SECPNTR
BEGXTAB

Output

Output from ALTRGEN is generated code and upon return to SCALE08, B2 is reset and X2 contains the core image of the X2 input.

A diagnostic is output (via BUG) when an unexpected node is encountered.

Example: ALTER PARA TO PROCEED TO PARB.

Case 1: PARA external (not defined in this compilation)
PARB entry index

```
Generator:  SA1    (PARB's entry index + BEGXTAB)
            BX7    X1
            SA7    C(AG90)
```

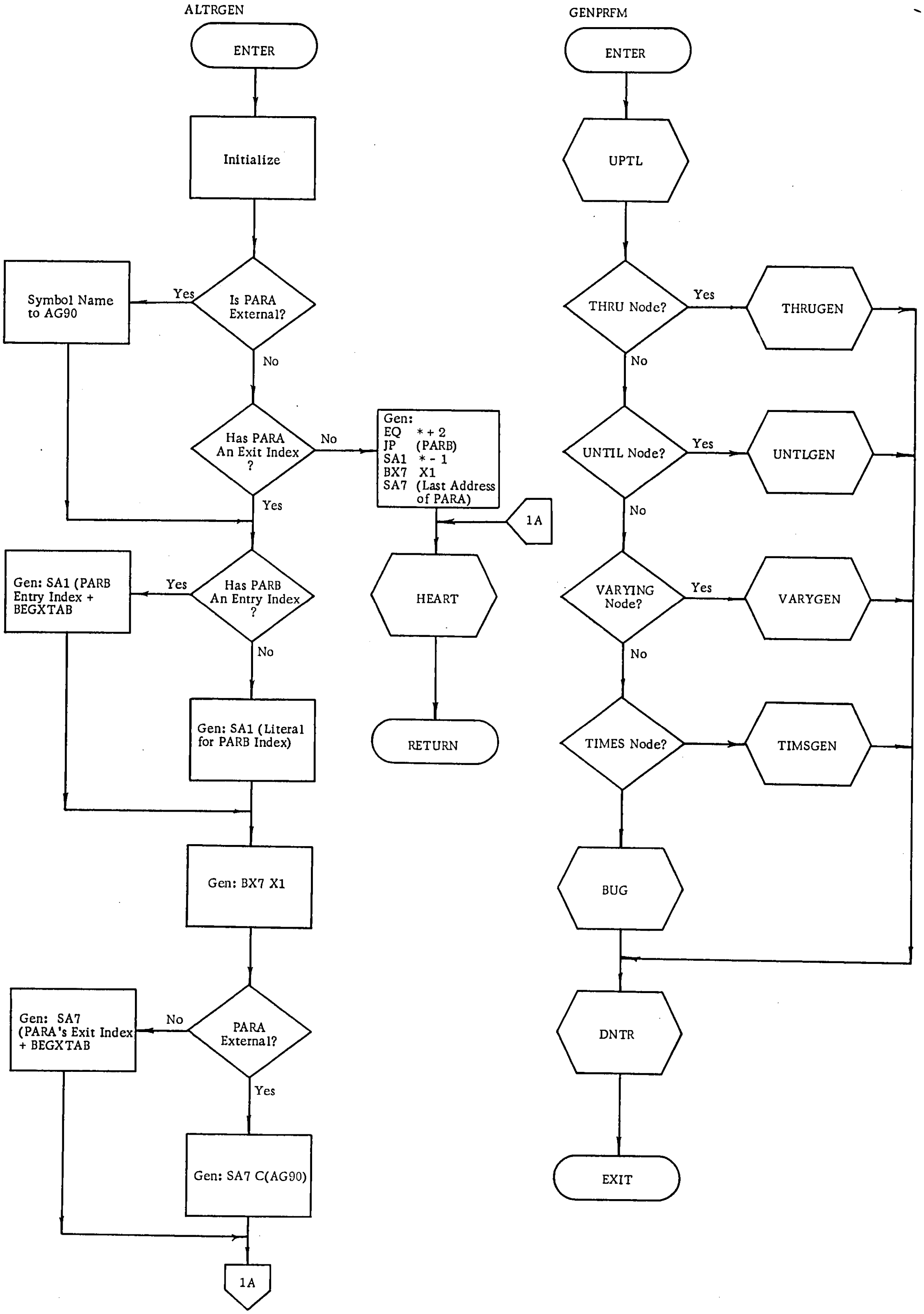


Figure 3-84. ALTRGEN and GENPRFM Flowcharts

Case 2: PARA external
PARB no entry index

```
Generator:  SA1    (literal for PARB index)
            BX7    X1
            SA7    C(AG90)
```

Case 3: PARA no exit index

```
Generator:  EQ     * + 2
            JP     (PARB)
            SA1    * - 1
            BX7    X1
            SA7    (last address of PARA)
```

Case 4: PARA exit, index
PARB no entry index

```
Generator:  SA1    (literal for PARB index)
            BX7    X1
            SA7    (PARA's exit index + BEGXTAB)
```

Case 5: PARA exit index
PARB entry index

```
Generator:  SA1    (PARB entry index + BEGXTAB)
            BX7    X1
            SA7    (PARA's exit index + BEGXTAB)
```

TCLIMB - TREE CLIMBER

Purpose

TCLIMB processes trees up and down depending upon various entry points used. It also has entry points for a FILO stack processor.

The following are the tree manipulating entry points and the functions that each does:

Entry Point	Function
UPTL	Climb tree to next left node.
UPTR	Climb tree to next right node.
DNTR	Climb down tree to previous node.

Register Usage

The following registers are destroyed by UPTL, UPTR, and DNTR:

A0, A2, A5, A7, X2, X5, X7

GENPLIM - CODE GENERATION FOR LIMITS OF PARAGRAPHS

Purpose

GENPLIM generates the code necessary to process the BEGINNING OF PARAGRAPH (405) and END OF PARAGRAPH (406) nodes. There are two entry points: GENBOP for the 405 node, and GENEOP for the 406.

Both GENBOP and GENEOP are called by Pass 2 via the jump table. X2 must contain a 405 or 406 node and B2 the base of the tree.

Routines Called

REGSAVE, REGRSTR
CART
SCALE08

GENBOP SUBROUTINE

Purpose

Calls CART to define the procedure name and returns to SCALE08.

GENEOP SUBROUTINE

Purpose

Merely returns to SCALE08.

GENSLIM - CODE GENERATION FOR LIMITS OF SECTIONS

Purpose

Generates code necessary for the BEGINNING OF SECTION (403) and END OF SECTION (404) nodes. There are six entry points: GENBOS, GENEOP, SECPNTR, BEGXTAB, BEGSEC, LASTPNT.

Both GENBOS and GENEOS are called by Pass 2 via the jump table. X2 must contain a 403 or 404 node and B2 the base of the tree.

The other entry points are pointers:

SECPNTR	Last section defined.
BEGXTAB	Address of beginning of index table.
BEGSEC	Beginning section.
LASTPNT	Last procedure defined (either a section or a paragraph).

Routines Called

REGSAVE, REGRSTR
CART
SCALE08

GENBOS SUBROUTINE

Purpose

Calls CART to define the procedure name and returns to SCALE08.

GENEOS SUBROUTINE

Purpose

Merely returns to SCALE08.

GENPRFM - GENERATE CODE FOR PERFORM VERBS

Purpose

GENPRFM controls the generation of all code connected with the PERFORM (441) verb. There are two entry points to this routine. GENPRFM is called by Pass 2 with X2 containing a 441 node and B2 containing the address of the base of the tree. (See Figure 3-84.)

Routines Called

SCALE08
REFDEF
UNTLGEN, VARYGEN, TIMSGEN
UPTL, DNTR, UPTR
BUG, DIAG
HEART
REGSAVE, REGRSTR

GENPRFM climbs the left branch of the tree and decides what type of node is encountered. Legal nodes are THRU (110), UNTIL (111), VARYING (114), and TIMES (115). GENPRFM then calls THRUGEN, UNTLGEN, VARYGEN, TIMSGEN depending on the node encountered. A return from the previously mentioned subroutines goes to an exit where DNTR is called (backing downward to the PERFORM node) and control is given to SCALE08.

THRUGEN - AN ENTRY POINT IN SUBROUTINE GENPRFM

Purpose

Generates code for the THRU (110) node of the PERFORM verb. (See Figure 3-85.)

Calling Sequence

Called by GENPRFM, UNTLGEN, VARYGEN, and TIMSGEN. X2 must contain a 110 node, B2 the base of the tree.

Routines Called

REGSAVE, REGRSTR
REFDEF
HEART

Code is generated dependent upon the indexes (or lack thereof) for the beginning and ending procedures.

Example: PERFORM PA THRU PB.

Case 1: PA has no entry index
PB has no exit index

```
Generator:  +SA1    (ending address of PB)
             BX7    X1
             SA7    * + 4
             SA1    * + 4
             BX7    X1
             SA7    (ending address of PB)
             JP     (beginning address of PA)
             BSSZ   1
             JP     * + 1
             SA1    * - 2
             SA1    * - 2
             BX7    X1
             SA7    (ending address of PB)
```

THRUGEN — subroutine of GENPRFM

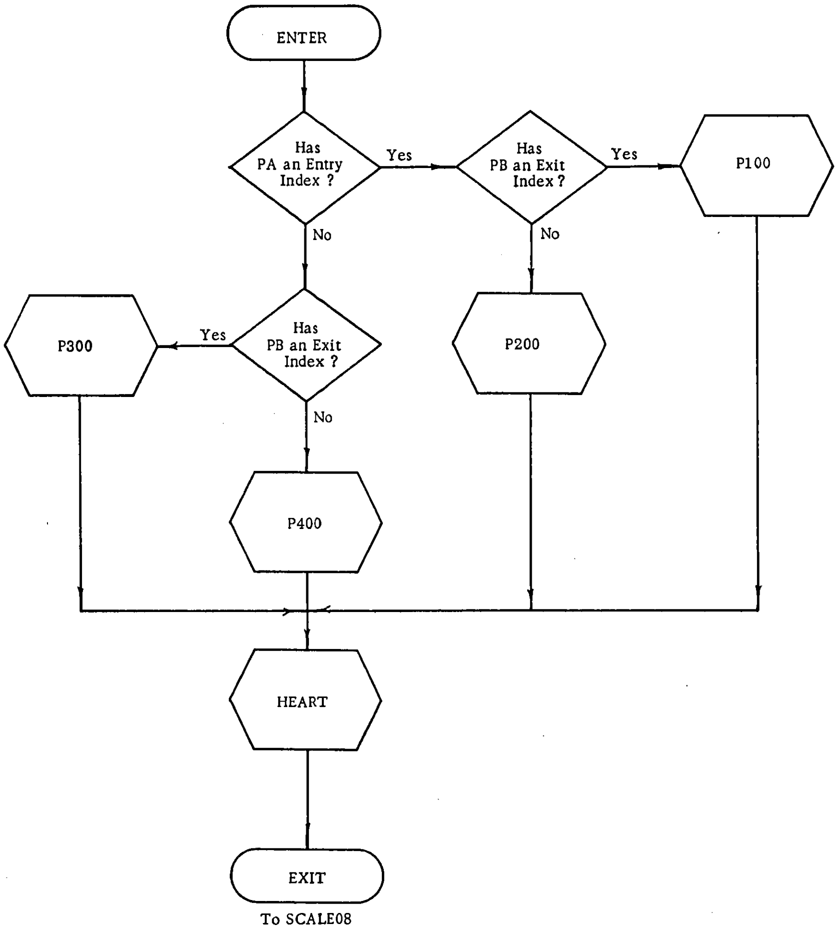


Figure 3-85. THRUGEN Flowchart (1 of 3)

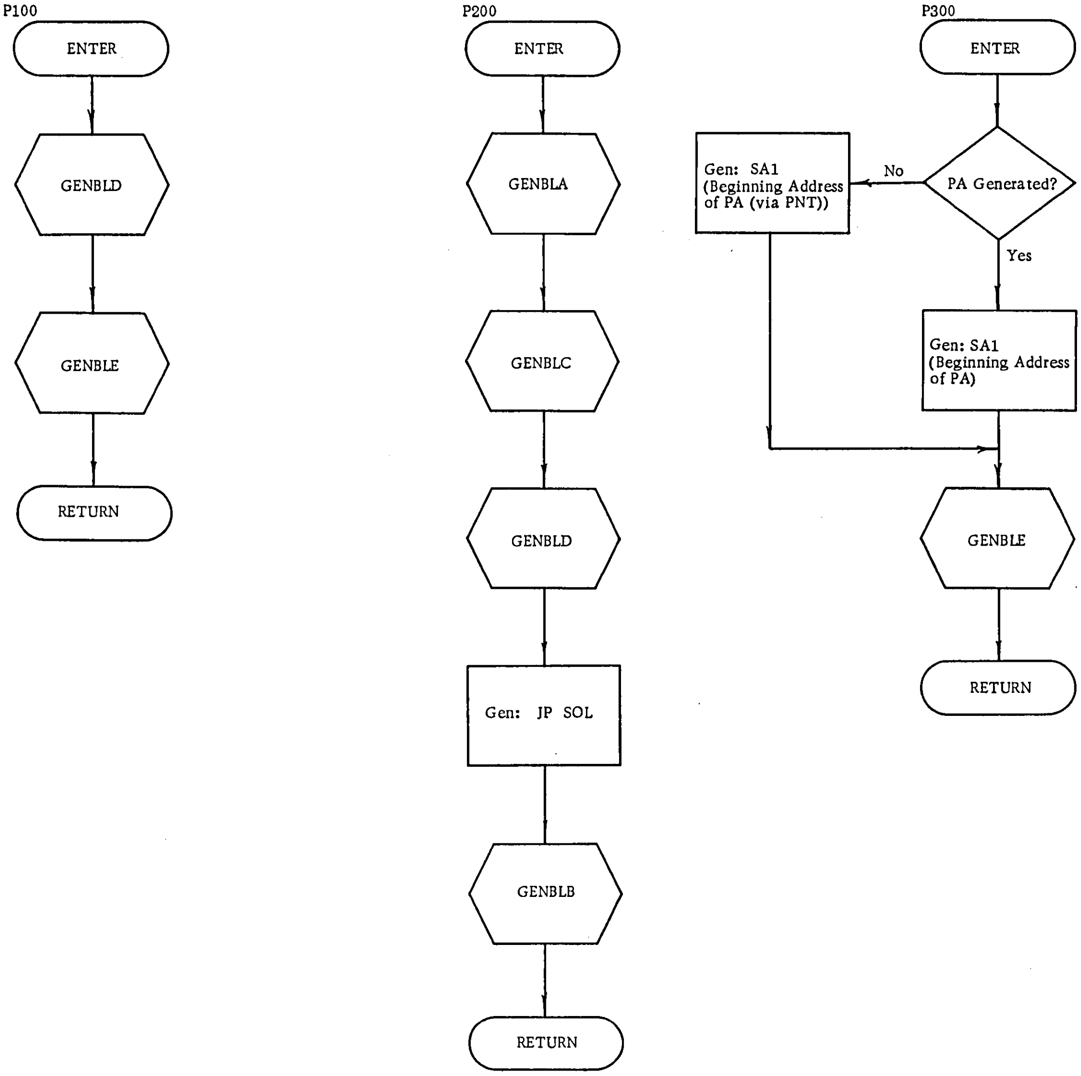


Figure 3-85. THRUGEN Flowchart (2 of 3)

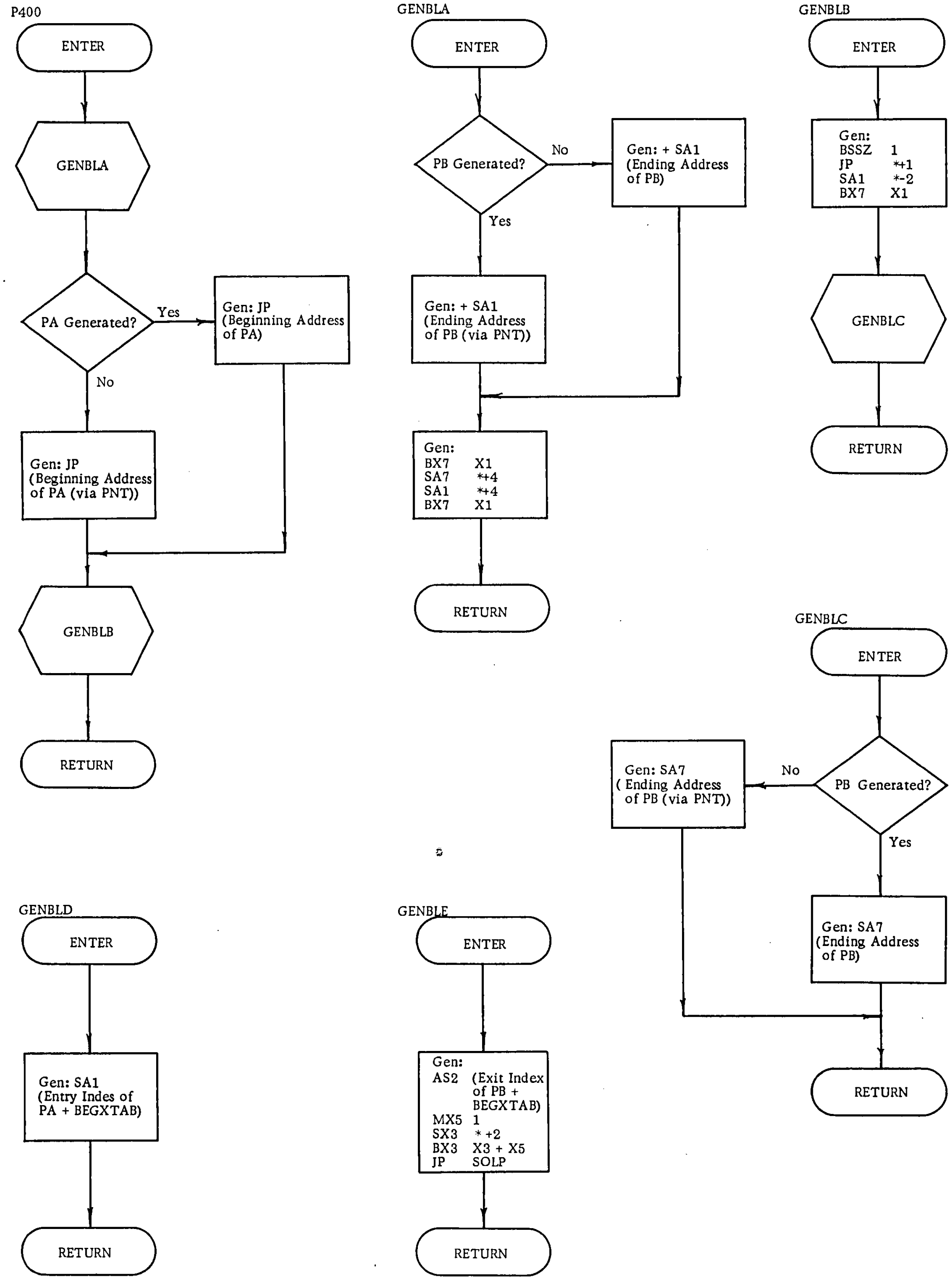


Figure 3-85. THRUGEN Flowchart (3 of 3)

Case 2: PA has an entry index
PB has no exit index

```
Generator:  +SA1    (ending address of PB)
             BX7    X1
             SA7    * + 4
             SA1    * + 4
             BX7    X1
             SA7    (ending address of PB)
             SA1    (entry index of PA + BEGXTAB)
             JP     SOL
             BSSZ   1
             JP     * + 1
             SA1    * - 2
             BX7    X1
             SA7    (ending address of PB)
```

Case 3: PA has an entry index
PB has an exit index

```
Generator:  SA1     (entry index of PA + BEGXTAB)
            SA2     (exit index of PB + BEGXTAB)
            MX5     1
            SX3     * + 2
            BX3     X3 + X4
            JP      SOLP
```

Case 4: PA has no entry index
PB has an exit index

```
Generator:  +SA1    (beginning address of PA)
             SA2    (exit index of PB + BEGXTAB)
             MX5    1
             SX3    * + 2
             BX3    X3 + X4
             JP     SOLP
```

PRFOPS - PERFORM OPTIONS

Purpose

PRFOPS controls the generation of code for the options of the PERFORM verb. The legal options are TIMES (115), UNTIL (111), and VARYING (114), with PRFOPS having three entry points: TIMSGEN, UNTLGEN, and VARYGEN, respectively.

All three entry points are called by PRFMGEN and themselves depending on the node that needs to be processed. In each case, X2 must contain the appropriate node and B2 the address of the base of the tree.

Routines Called

BUG, DIAG
HEART, CART
UPTR, UPTL, DNTR
THRUGEN
PUT, PICK
REGSAVE, REGRSTR
GENLOD, STORE, GARTH

External Pointers Referenced

LODSW
LABZRDF, NOLLAB
PROPCUR, PROPLOD

UNTLGEN - PRFOPS SUBROUTINE

Purpose

UNTLGEN controls the generation of code stemming from an UNTIL (111) node. When called, X2 must have a 111 node and B2 the base of the tree. (See Figure 3-86.)

Different code is generated depending on whether this is a simple UNTIL option or part of the VARYING option.

If a simple UNTIL option, e.g., PERFORM PA THRU PB UNTIL A EQ B, UNTLGEN calls the conditional generator, climbs the right branch, and calls THRUGEN. UNTLGEN then generates a jump back to the local label for the conditional test.

If the UNTIL node is part of the VARYING option, the left node will be a FROM. In this case, SCFRM is called, a climb down the tree is executed back to the UNTIL node, a climb up the right branch is executed, and the conditional generator is called. The tree is then descended and control returned to the calling routine.

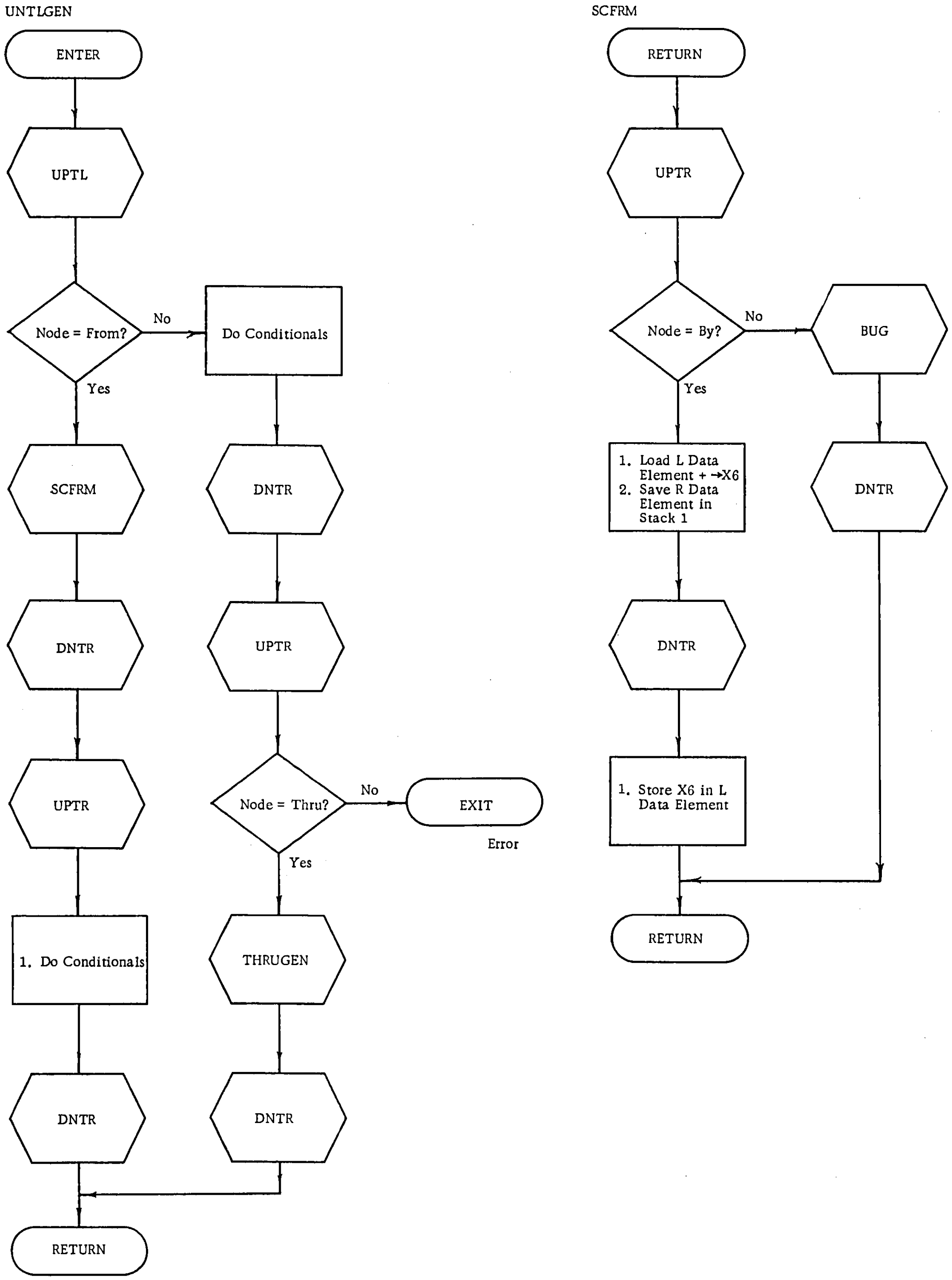


Figure 3-86. UNTLGEN and SCFRM Flowcharts

TIMSGEN - PRFOPS SUBROUTINE

Purpose

TIMSGEN controls the generation of code stemming from a TIMES node (115). When called, X2 must have a 115 node and B2 the base of the tree.

GENLOD is called to load a number into register X1 or X1 and X2. The number is converted to a COMP-1 and the following code generated:

```
                SX7     X1 + B1
                SA7     * + 2
                EQ      LOCLABn
                BSSZ    1
    LOCLABn     SA5     * - 1
                SX7     X5 - 1
                ZR      X7, LOCLABn+1
                SA7     * - 2
```

The tree is then climbed to the right and THRUGEN called. The tree is descended and the local label $LOCLAB_{n+1}$ is defined. Control is then returned to the calling subroutine.

SCFRM - PRFOPS SUBROUTINE

Purpose

SCFRM controls the operation of code for the FROM (112) node of the PERFORM verb. When called, X2 must have a 112 node and B2 the base of the tree. (See Figure 3-86.)

The FROM node is saved in a FILO stack called NODSTAK, and the right branch of the tree is climbed. Load, add, and store are executed via GENLOD, GARTH, and STORE respectively. Control is then returned to the calling routine.

AFTRGEN - PRFOPS SUBROUTINE

Purpose

AFTRGEN controls the generation of code from an AFTER (116) node of the PERFORM verb. When called, X2 must have a 116 node and B2 the base of the tree. (See Figure 3-87.)

AFTRGEN is a serially recursive subroutine and, therefore, saves the calling address in a FILO stack called RETSTAK. The left branch is climbed and the code executed is dependent on the type of node found:

1. If the new node is an AFTER node, AFTRGEN is called, the tree descended, the right link followed, UNTLGEN called, the tree descended, and control is given back to the calling subroutine.

2. If the new node is an UNTIL node, UNTLGEN is called, the tree descended, the right branch climbed, UNTLGEN called again, the tree descended, and control given back to the calling subroutine.

3. If any other node is encountered, a diagnostic, via BUG is issued.

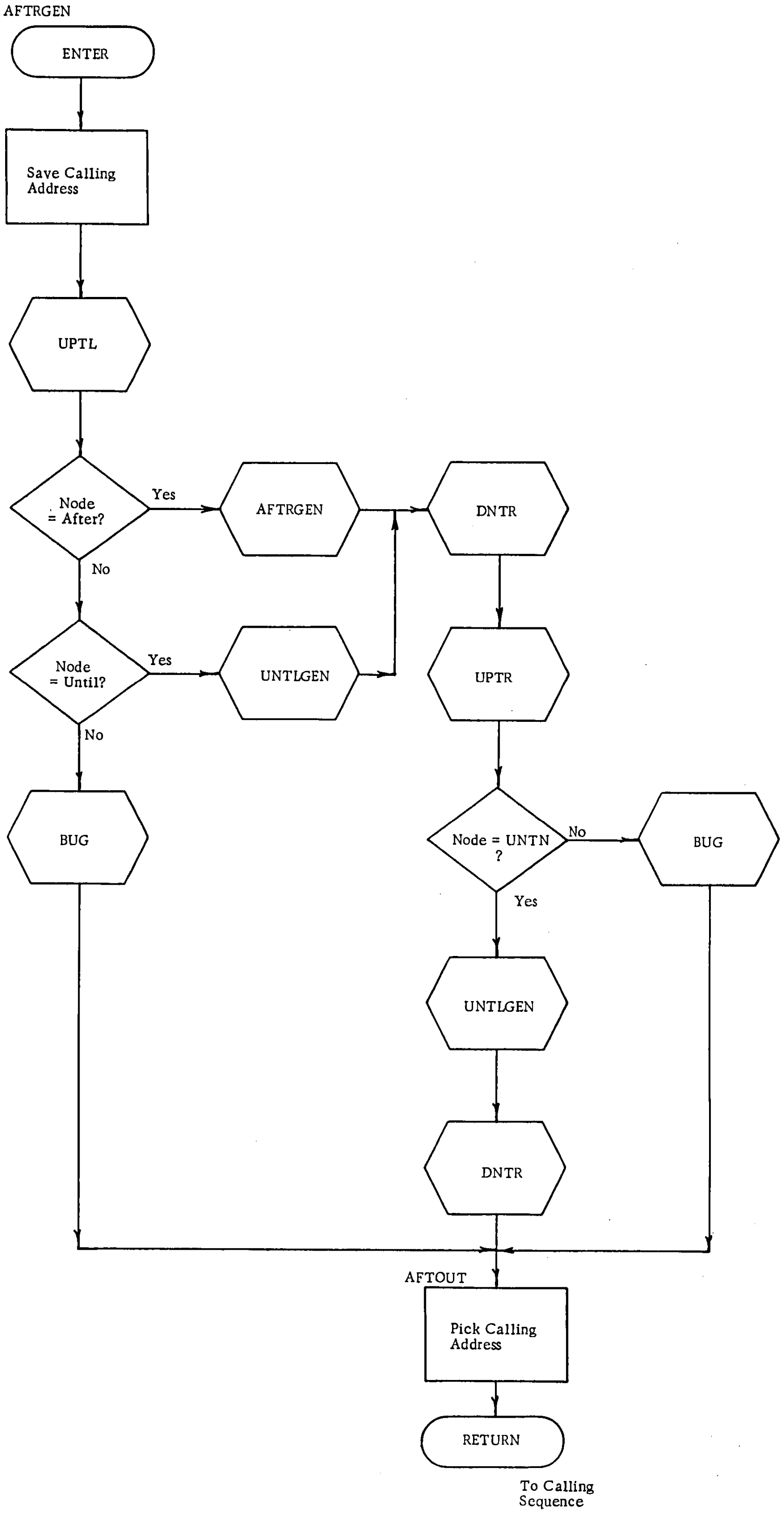


Figure 3-87. AFTRGEN Flowchart

VARYGEN - PRFOPS SUBROUTINE

Purpose

VARYGEN controls the generation of code from a VARYING (114) node. When called, X2 must have a 114 node and B2 the base of the tree. (See Figure 3-88.)

VARYGEN follows the left link and examines the next node:

1. If the node is AFTER, AFTRGEN is called, the tree descended, the right branch climbed, THRUGEN called, the tree descended, and clean-up code generated (see below).
2. If the node is UNTIL, UNTLGEN is called, the tree descended, the right branch climbed, THRUGEN called, the tree descended, and clean-up code generated (see below).
3. If any other code is found, a diagnostic is issued, via BUG, and control returned to the calling subroutine.

A set of instructions is generated for each time the conditional generator is called. The sets differ only in the local labels used. The actual code generated is dependent on the type of variables used.

```
LOCLABn     - GENLOD instructions -
            BX6     X1
            BX7     X2
            - GARTH instructions -
            - STORE instructions -
            EQ      LOCLABn+1
```

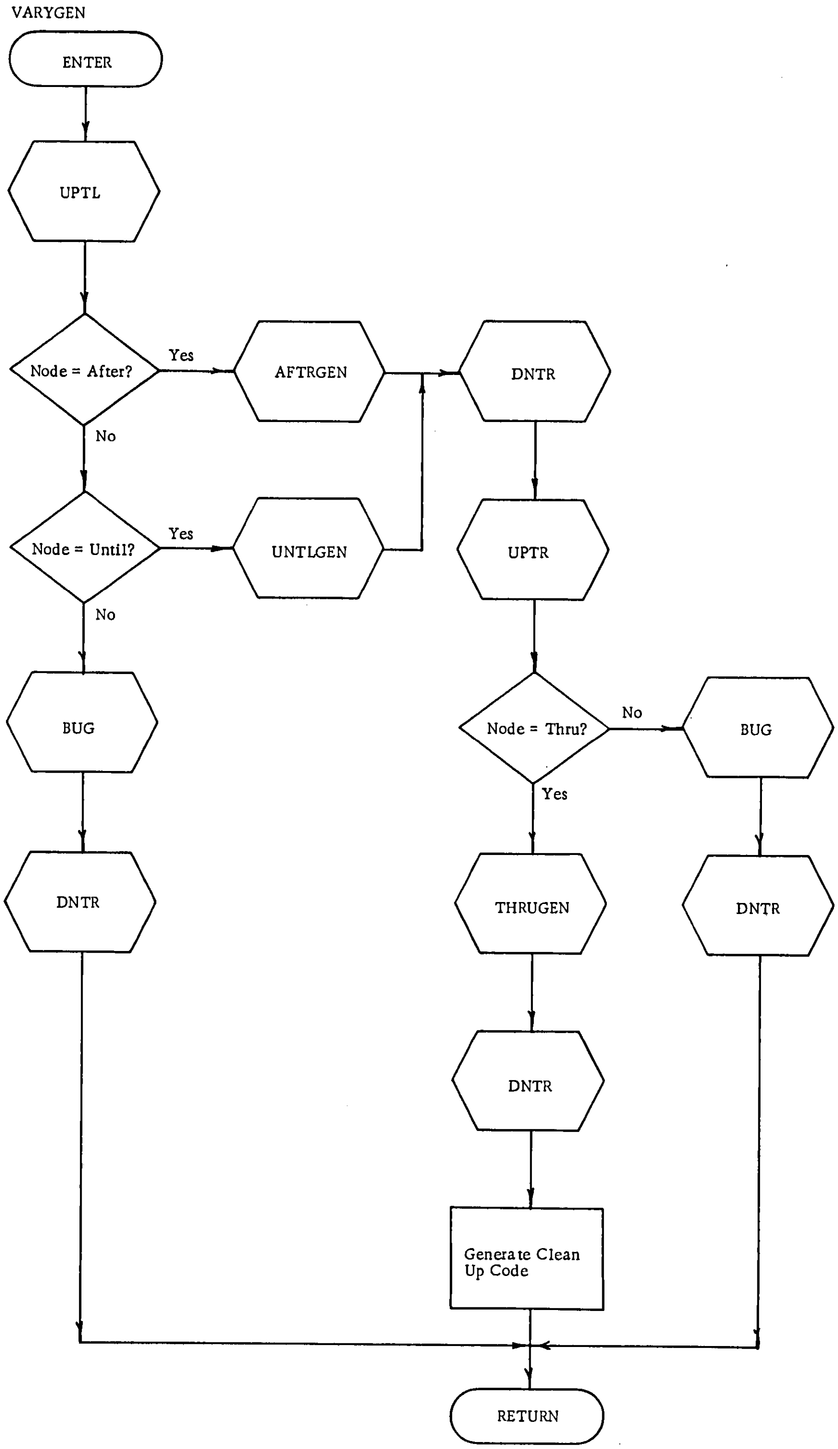


Figure 3-88. VARYGEN Flowchart

GENDISP - OBJECT SUBROUTINE CALLING SEQUENCE CODE GENERATION ROUTINE

Purpose

Assembled in GENDISP are the subroutines which generate calling sequences to their object-time subroutines for the miscellaneous verbs. (See Figure 3-89 through 3-99.)

Calling Sequence

All subroutines are entered from Pass 2 by a JP.

Routines Called

DIAG
GENLOD
LODINT
HEART
CART
SCALE08

Operation

For any of the verbs, the tree pointed to by B2 is climbed by the UPTL and UPTR subroutines. Descent from any branch, left or right, is made by DNTR. All prototype HEART instructions are stored in CYMSTRG. FLUSHRT is called to send HEART the accumulated instructions for assembly.

For details of instructions generated for any of the particular verbs, see their respective descriptions in the Object Library section.

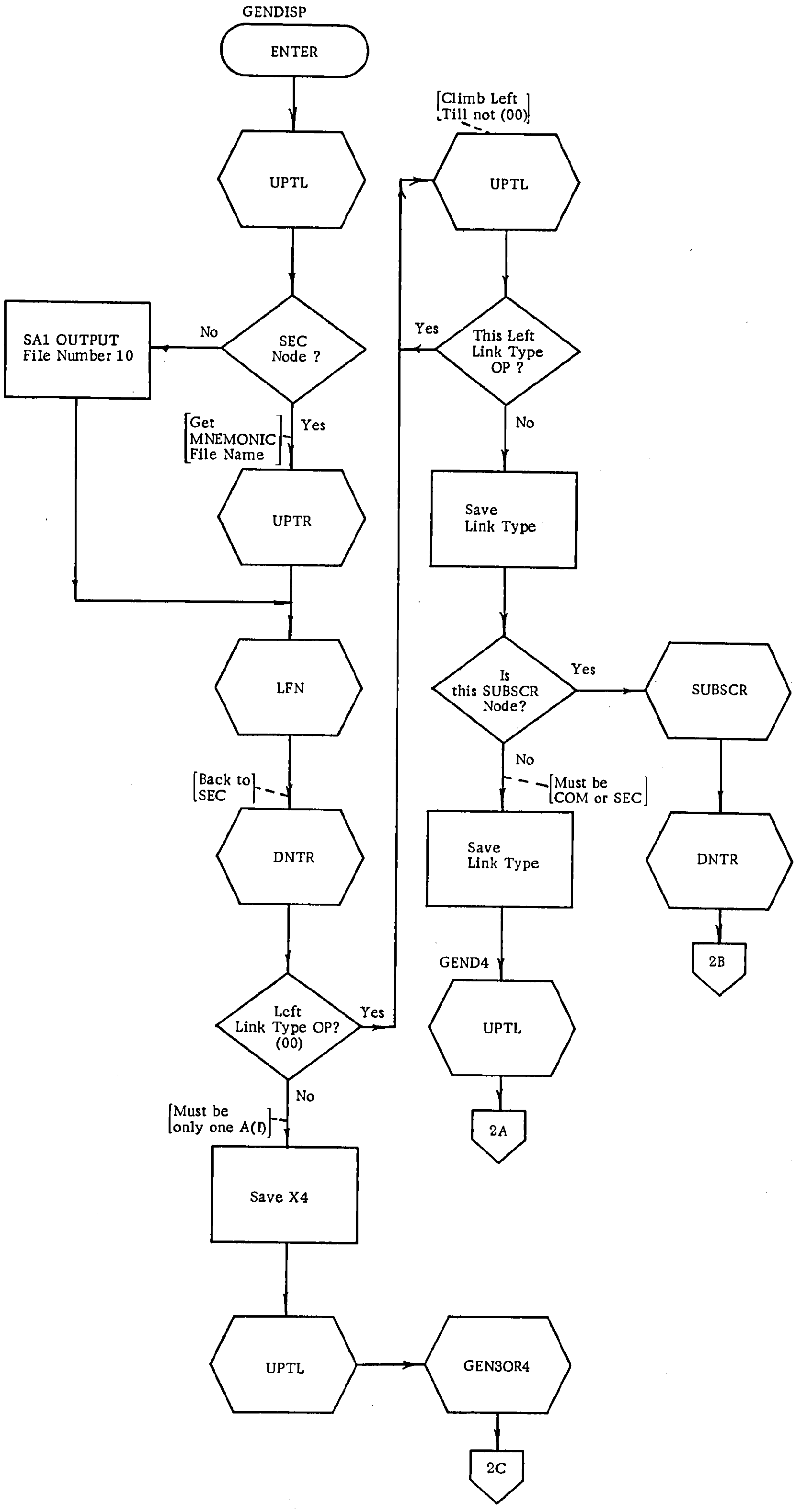


Figure 3-89. GENDISP Flowchart (1 of 2)

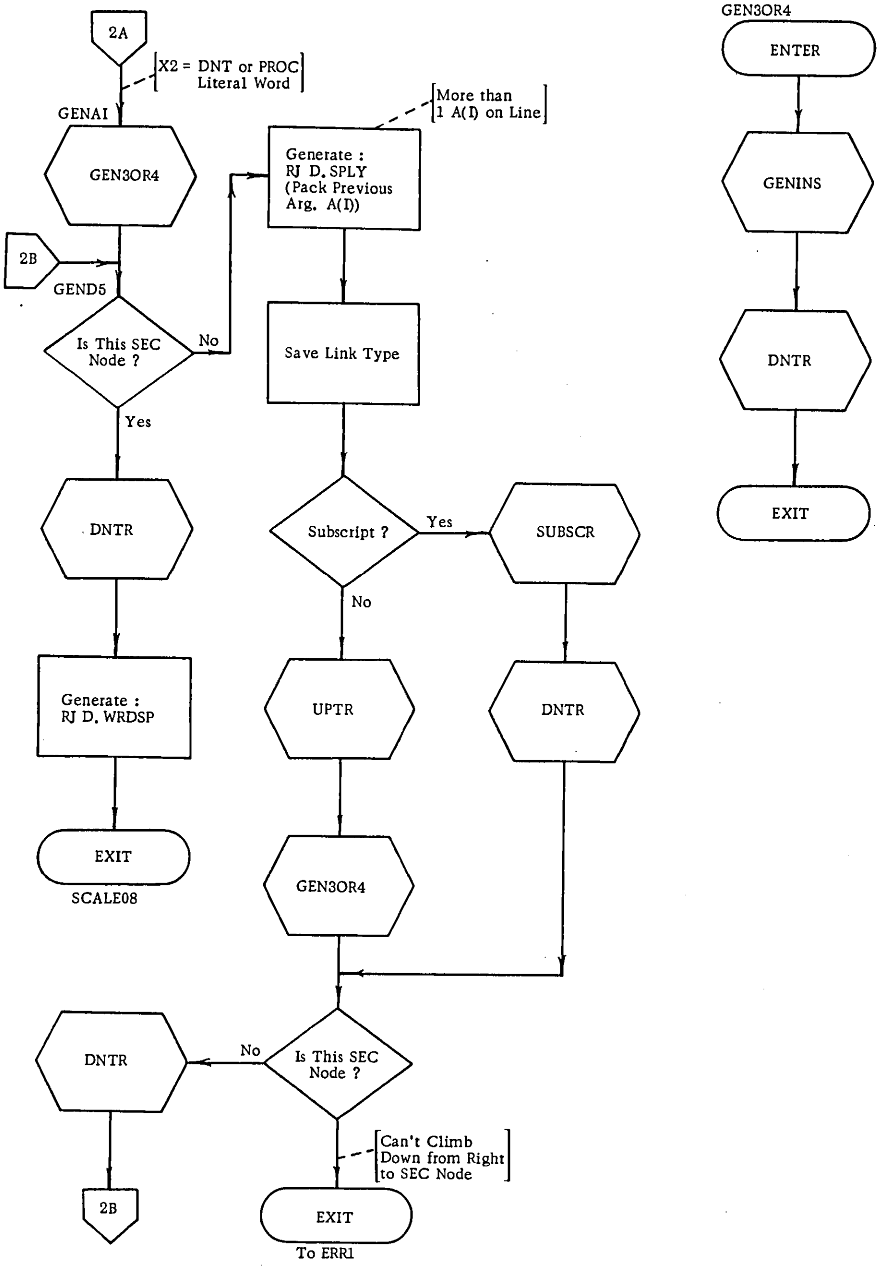


Figure 3-89. GENDISP Flowchart (2 of 2)

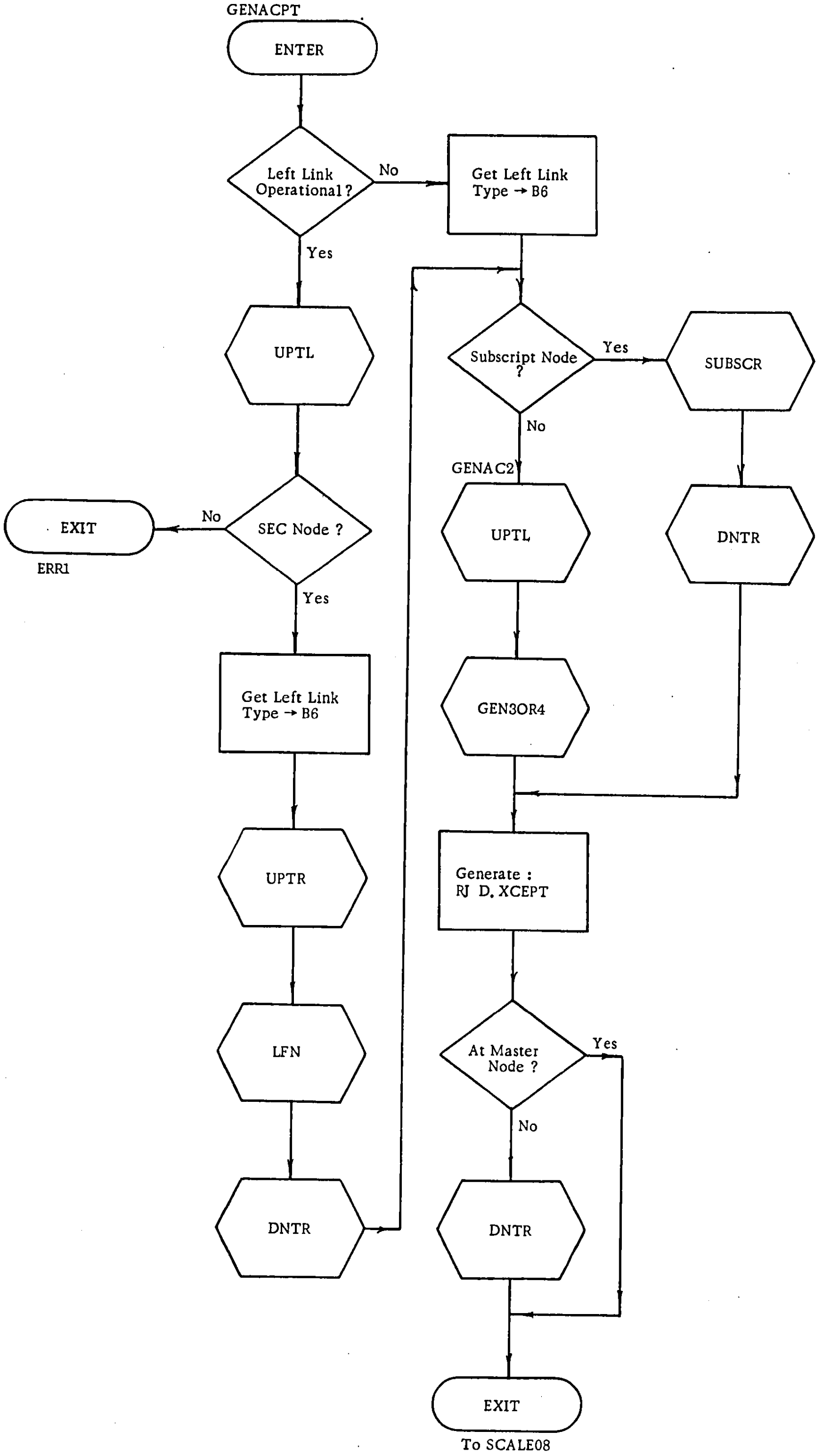


Figure 3-90. GENACPT Flowchart

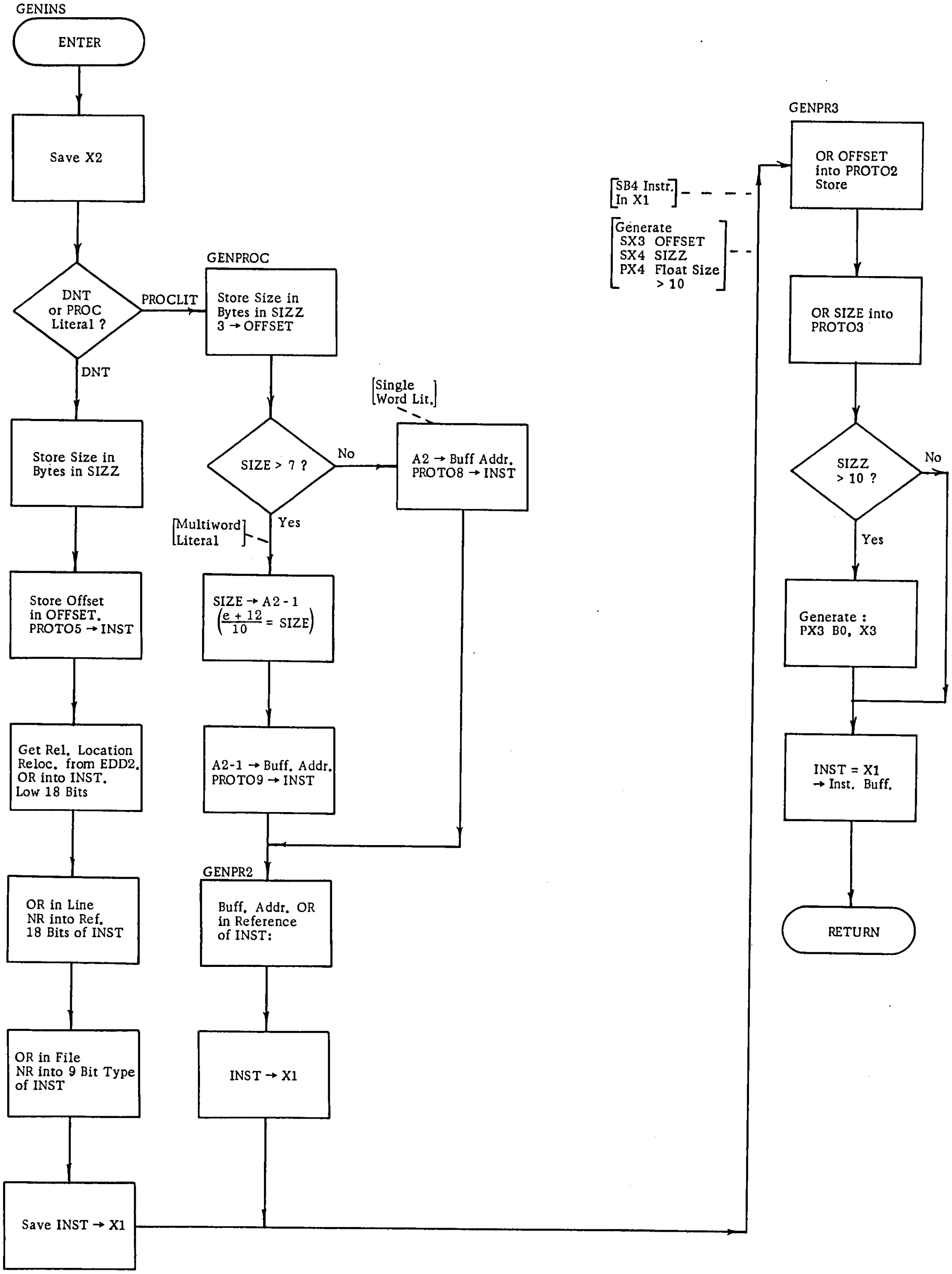


Figure 3-91. GENINS Flowchart

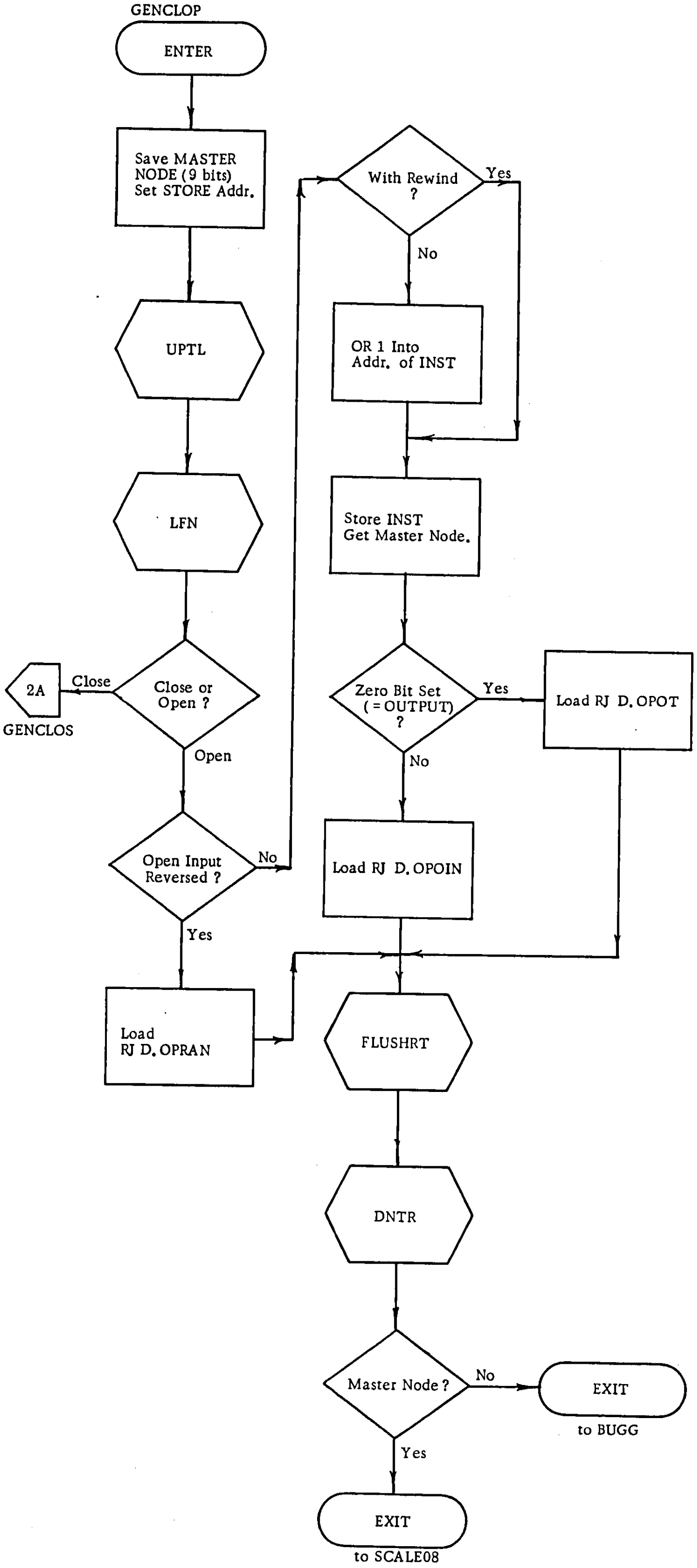


Figure 3-92. GENCLOP Flowchart (1 of 2)

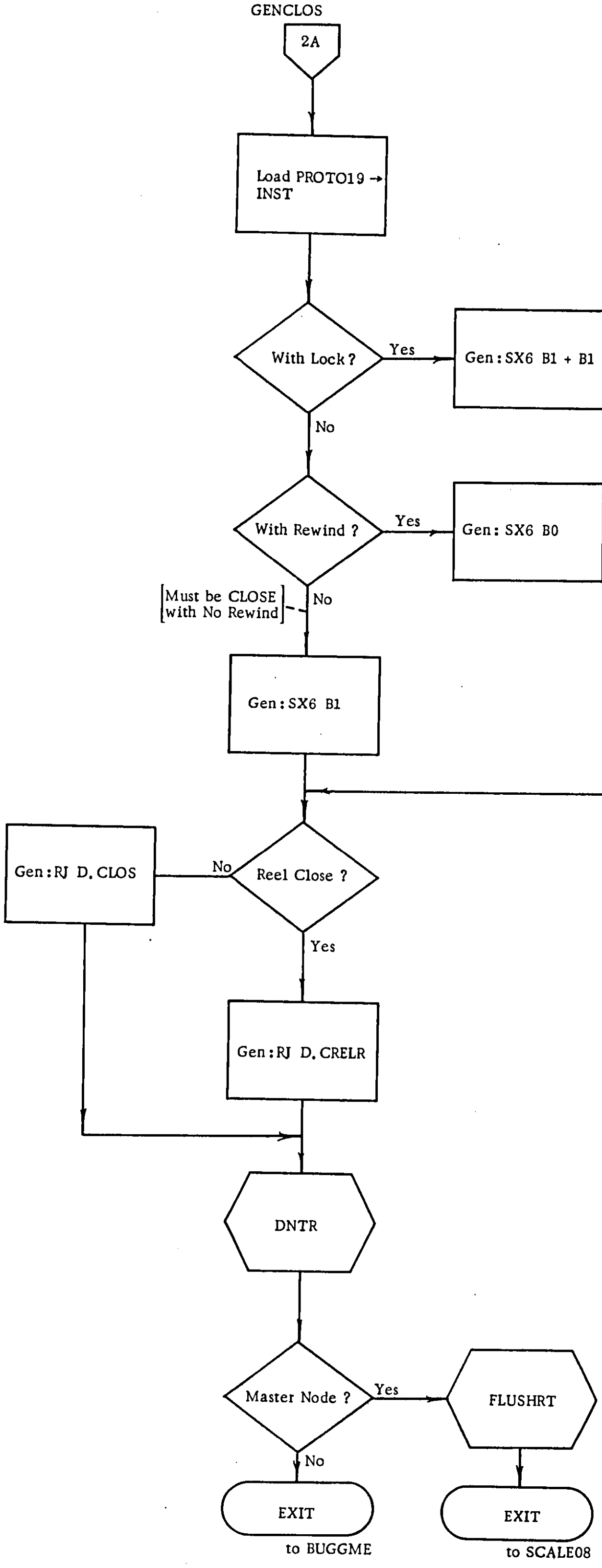


Figure 3-92. GENCLOP Flowchart (2 of 2)

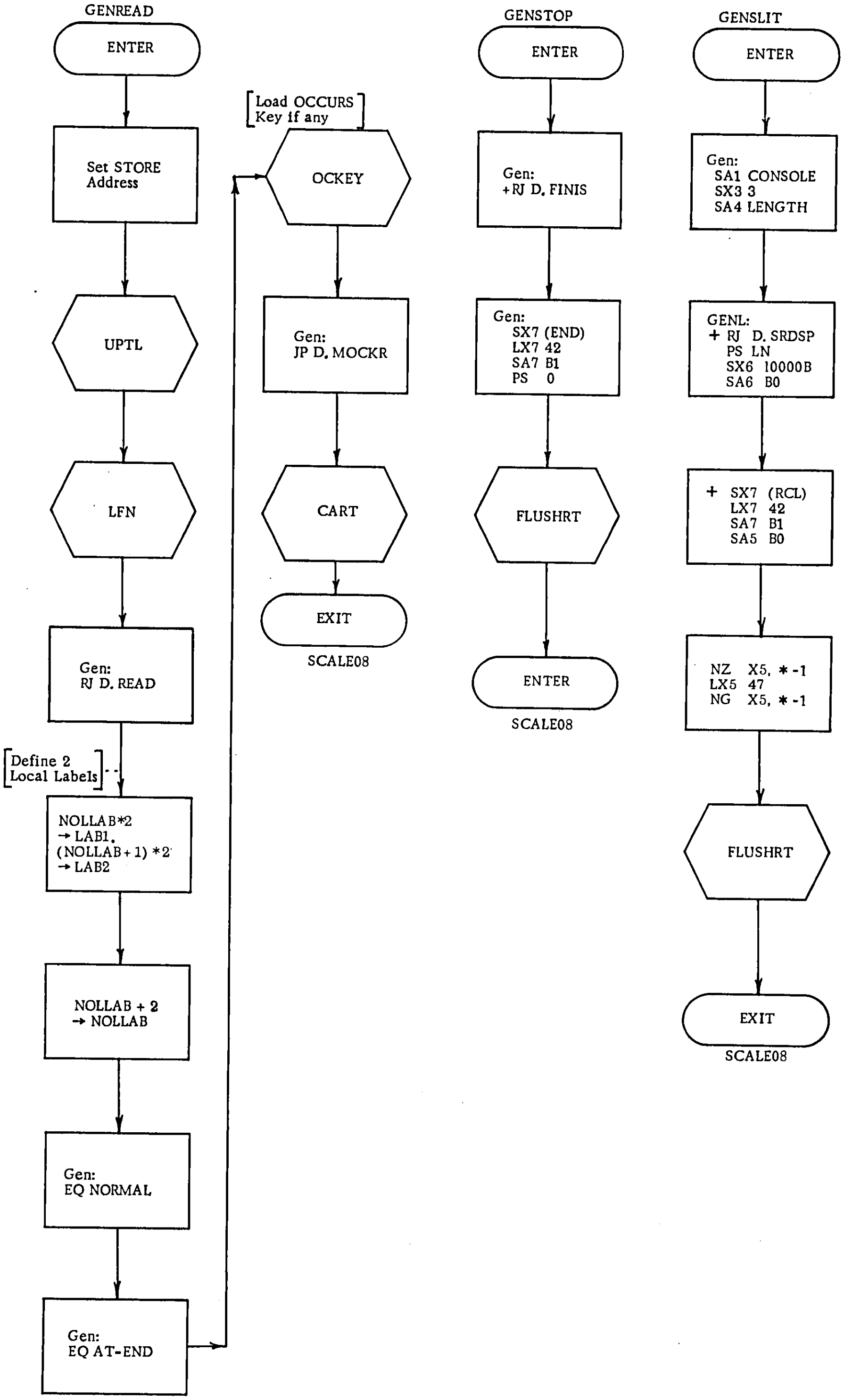


Figure 3-93. GENREAD, GENSTOP, and GENSLIT Flowcharts

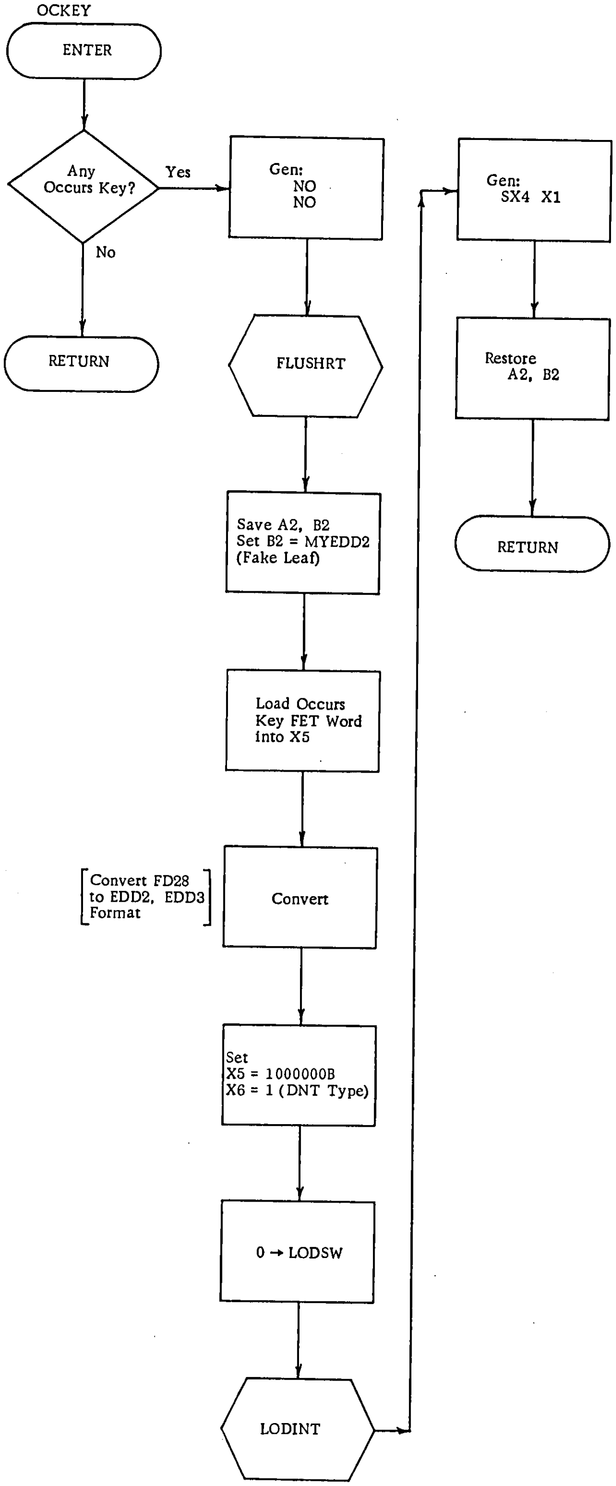


Figure 3-94. OCKEY Flowchart

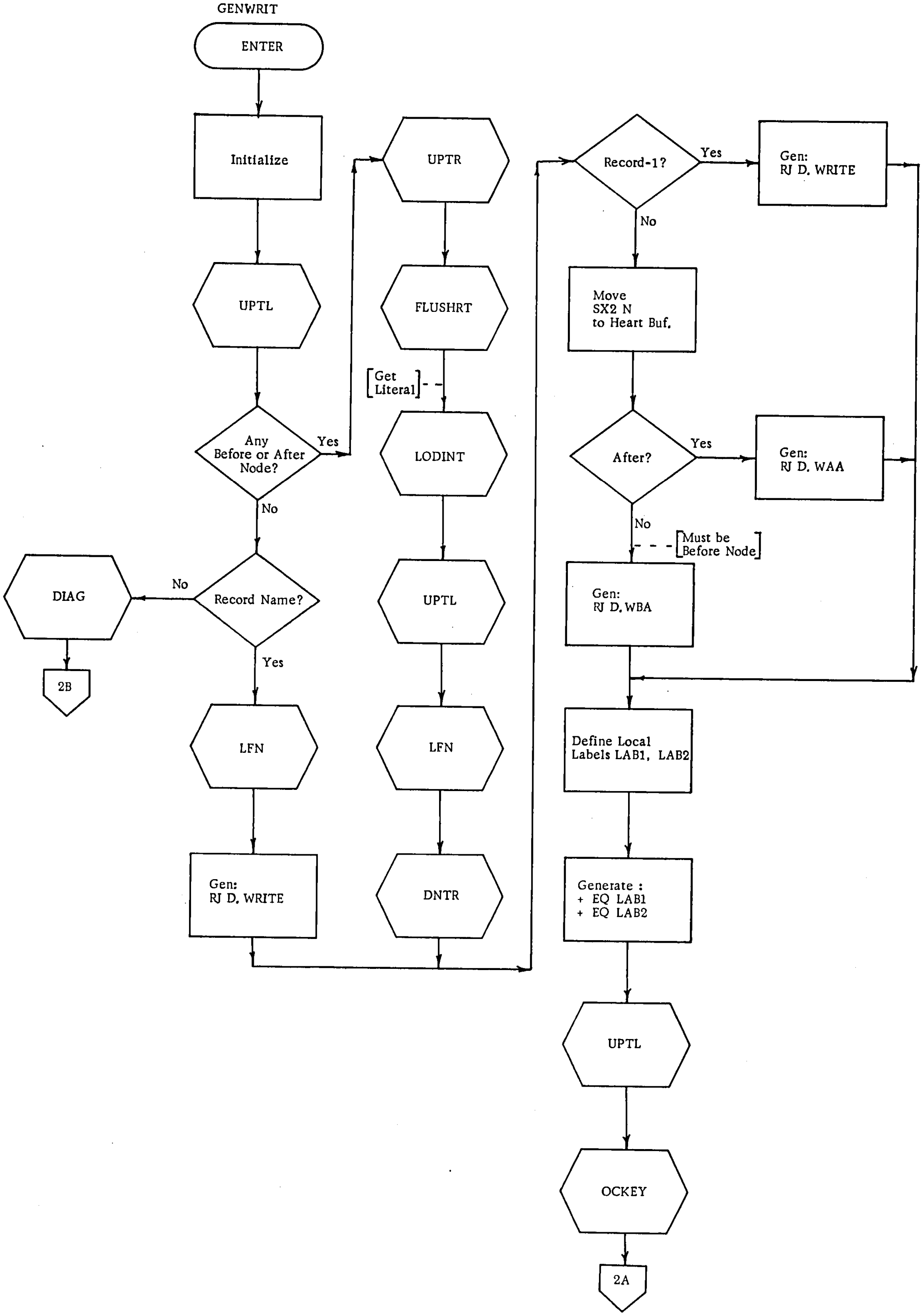


Figure 3-95. GENWRIT Flowchart (1 of 2)

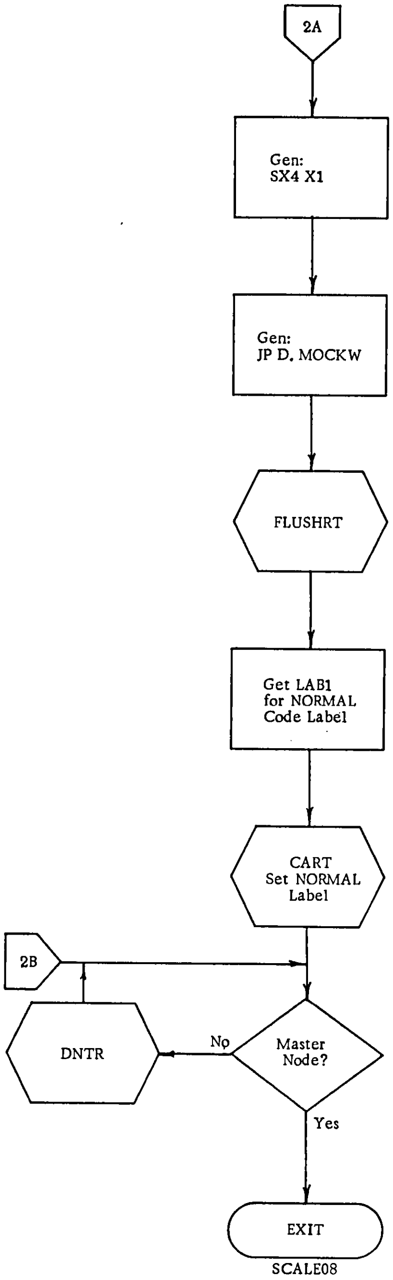


Figure 3-95. GENWRIT Flowchart (2 of 2)

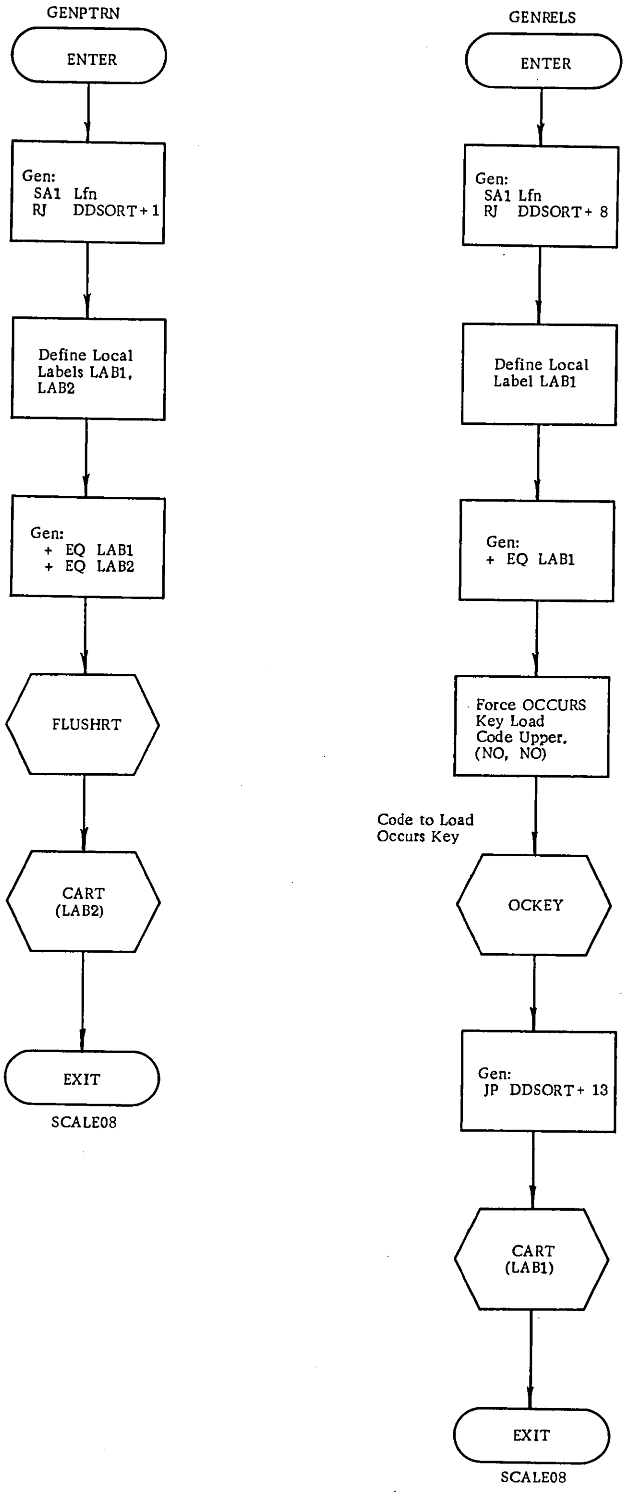


Figure 3-96. GENPTRN and GENRELS Flowcharts

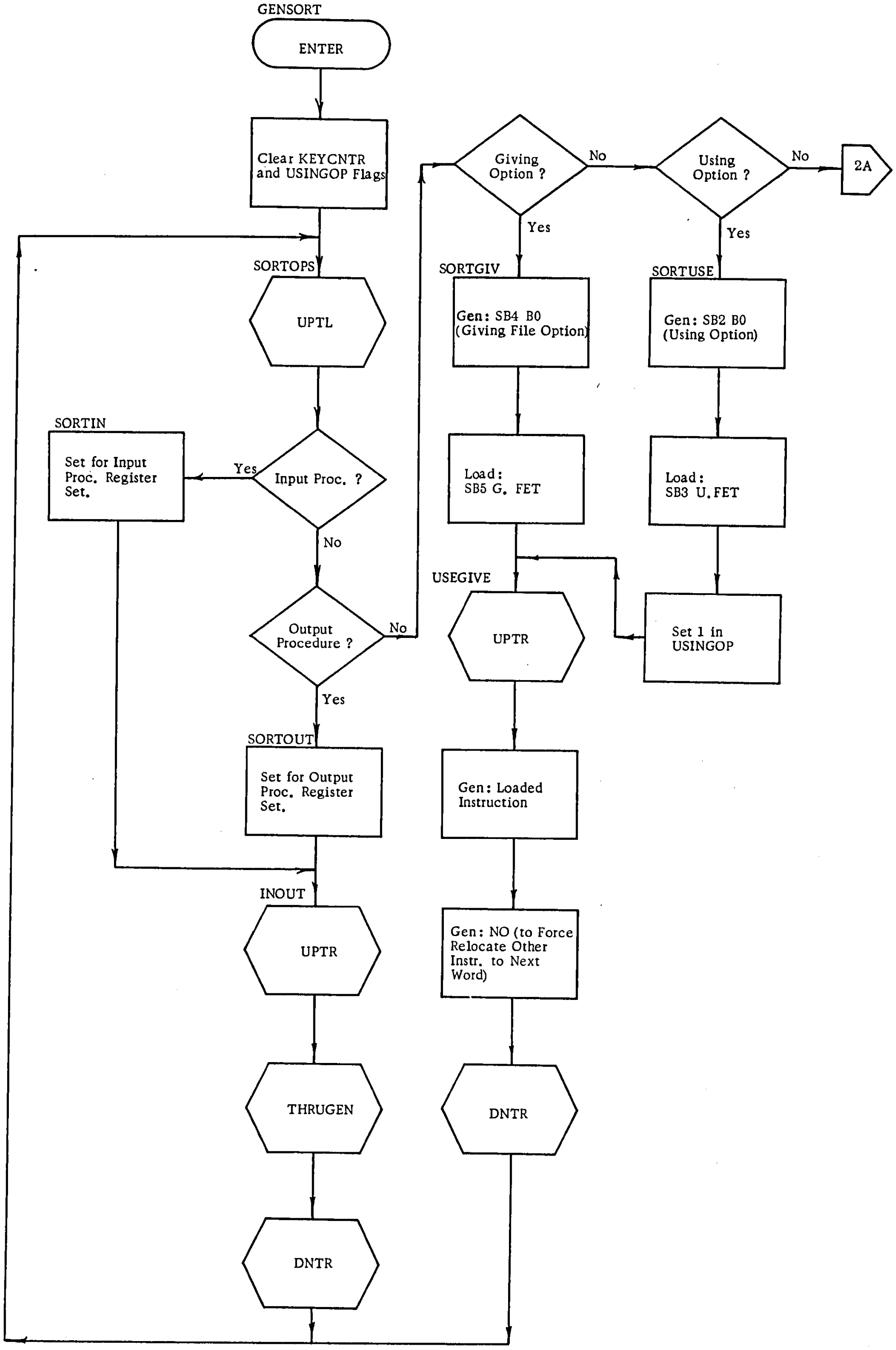


Figure 3-97. GENSORT Flowchart (1 of 3)

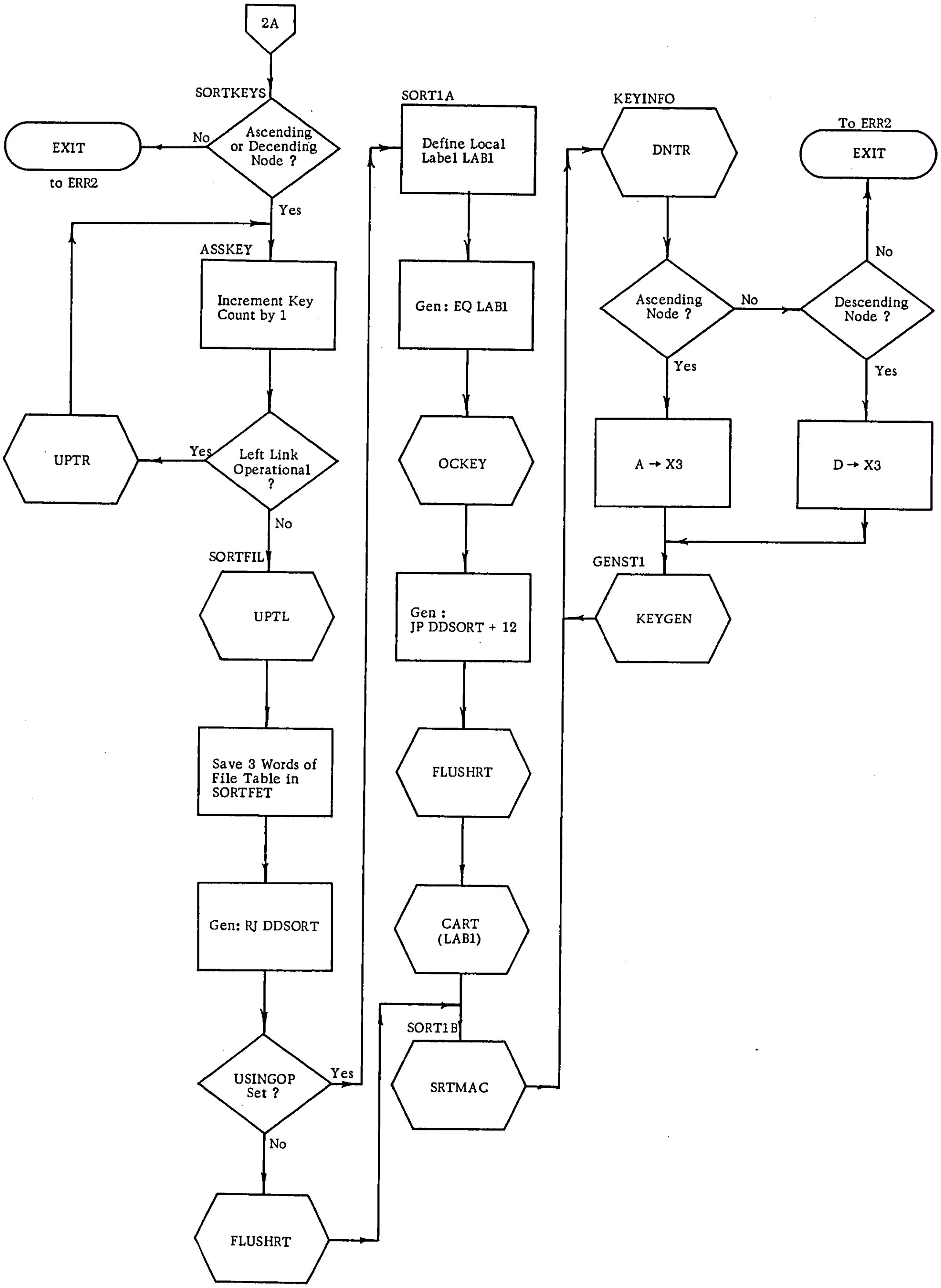


Figure 3-97. GENSORT Flowchart (2 of 3)

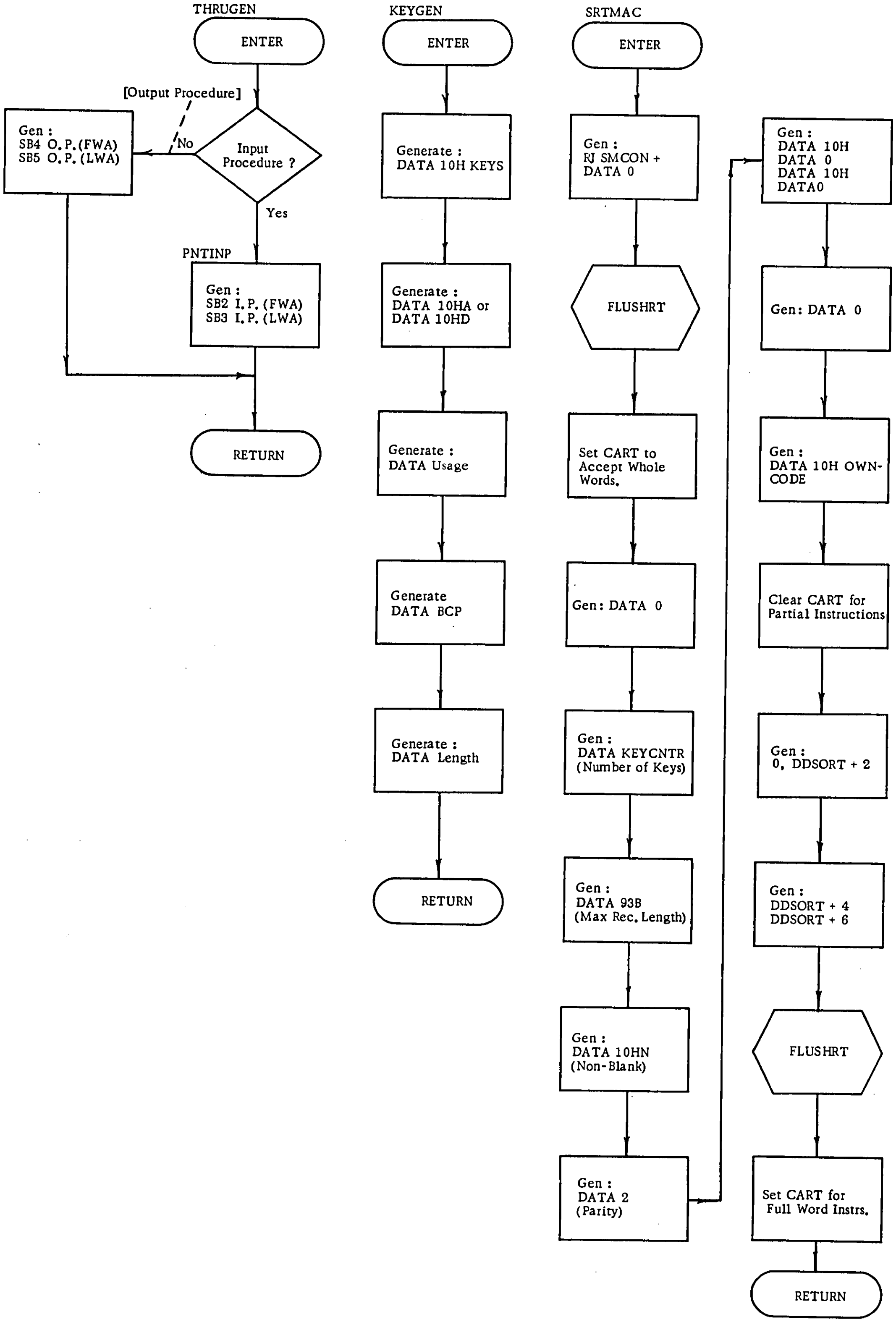


Figure 3-97. GENSORT Flowchart (3 of 3)

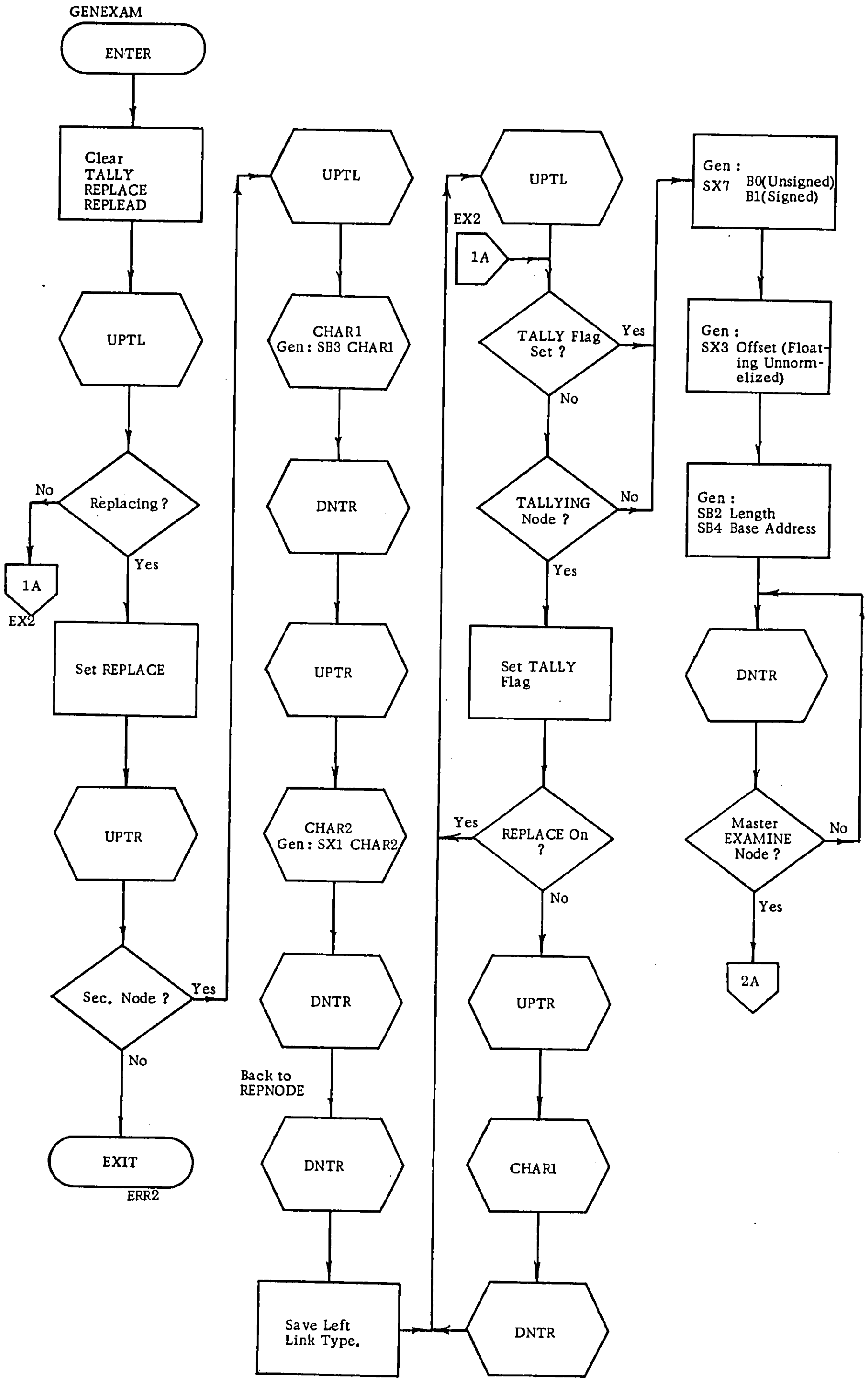


Figure 3-98. GENEXAM Flowchart (1 of 2)

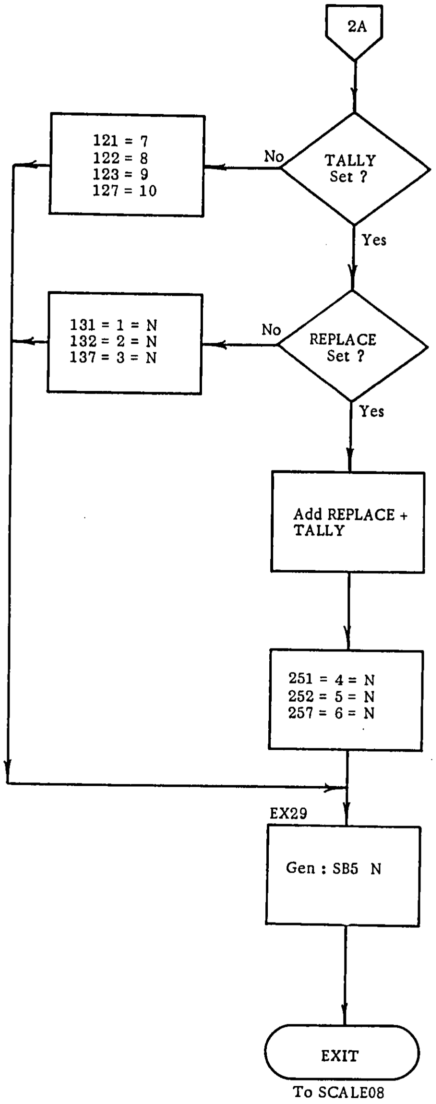


Figure 3-98. GENEXAM Flowchart (2 of 2)

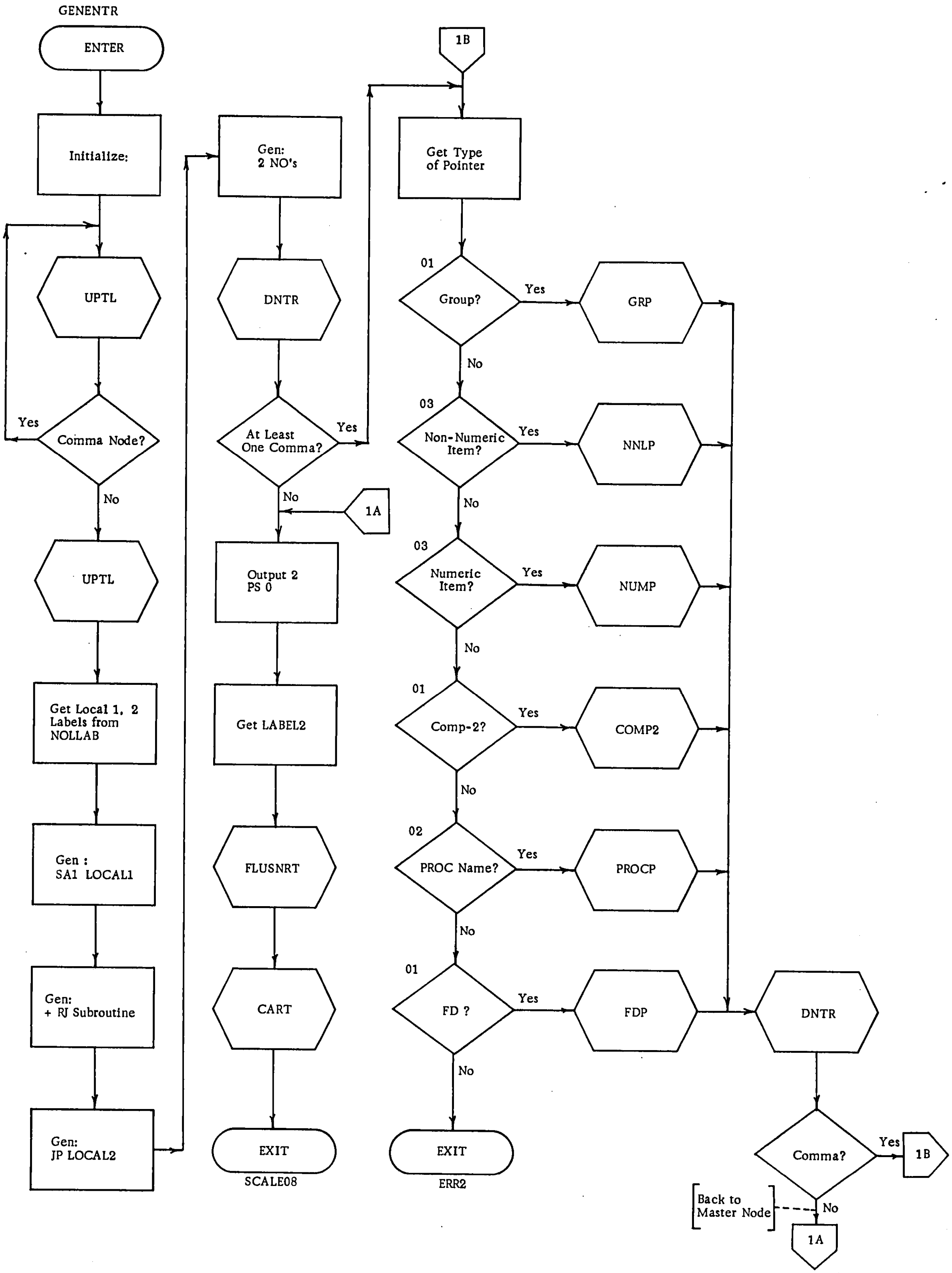


Figure 3-99. GENENTR Flowchart

SECTION 4

SECTION 4 - COMPILER INTERNAL TABLE FORMATS

DATA NAME AND PROCEDURE NAME TABLES

Entries in these tables are identified by three bits in the first word ($T_3T_2T_1$) with further identification provided in most cases by the next three bits. Hash link entries are chained together from the HASH table, one link for each unique name. Other entries are chained from the hash link entry for that name, one entry for each use of the name, this latter chain also linking back to the hash link. Entry formats follow with their legend.

Data and Procedure Name tables are shown in Tables 4-1 and 4-2.

Table 4-1. Data and Procedure Name Tables - Legend for T and D Fields

$T_{3\,2\,1}$	$D_3D_2D_1$
000	Hash link (HL) followed by Name (NM)
	001 Special Name (SN)
	010 Procedure Name (PN)
	100 Data Name (DN)
011	Data Division Section Headings
	000 File Section
	001 Common-Storage
	010 Working-Storage
	100 Constant
	101 Report
	111 DCOMON
100	Group Data Description (GDD)
	100 FD
	000 DD item (GDD1)
	010 Redefining DD or Renaming DD
	001 Condition Name (88)-(CN)
	110 RD
	101 SD
101	Elementary Data Description (EDD)
	000 DD item (EDD1) no Redefines or Renames
	010 Redefining DD or Renaming DD
	001 Condition Name (88)-(CN)
	100 Picture (EP) following this EDD3 or SSC
001	Literal
	001 Data Division Literal (DDL)
	000 Procedure Division Literal (PDL)
111	Special Name Item (SNI)
	000 Mnemonic File Name (SNF)
	001 Mnemonic Name for Literal (SNL)
	100 Switch Status (SNS)
110	Copy (SC)
	010 Redefining Item
	000 Not a Redefining Item
	100 Renames (Level 66)-(RN)
010	Procedure Item
	000 Procedure Definition (PD)
	100 Procedure Reference (PR)

Table 4-2. Data Name Table - Legend for Entry Fields

Level Numbers

Levels 01 through 49 are given their binary equivalents.
Special level numbers are carried as follows:
RD=0
SD=0
FD=0
66=62_8
77=63_8
88=64_8

Class

$C_3C_2C_1$	
000	Unspecified
001	Alphabetic
010	Numeric
011	Alphanumeric (AN)
100	Not used
101	Not used
110	Edit (numeric)
111	Mixed (group)

Usage

$U_3U_2U_1$	
000	Unspecified
001	Display
010	Computational
011	Not used
100	Computational-1
101	Not used
110	Computational-2
111	Mixed (group)

Synchronized

S_2 0 = Not Synchronized
1 = Synchronized
S_1 0 = Synchronized left
1 = Synchronized right
LR 0 = Not a label record
1 = Label record

Group Flags

G_6		Not used
G_5	0	No occurs
	1	Occurs--SSC follows GDD3
G_4	0	Not a Report Description
	1	Report Description
G_3	0	Not occurs Depending
	1	Occurs Depending
G_2	0	Not a FILLER item
	1	FILLER item
G_1	0	Not an 01 level
	1	01 level

Point Location

P_7 0 = Assumed Point
1 = Actual Point
P_6 0 = Point Left
1 = Point Right
P_5-P_1 0 to 31

Elementary Flags

E_6	0	No Value
	1	Value follows EDDX and SSC
E_5	0	No occurs
	1	Occurs--SSC follows EDDX
E_4	0	Not a Report Description
	1	Report Description
E_3	0	Not occurs Depending
	1	Occurs Depending
E_2	0	Not a FILLER item
	1	FILLER item
E_1	0	Not an 01 level
	1	01 level

sign - SIGNED
BWZ - Blank when Zero

Occurs

0 = Item did not contain an occurs clause
1 = Item contained an occurs clause

Edit

0 = Not Check Protect (*)
1 = Check Protect (*)

Notes to Table 4-2:

Size

$n \leq 2,142$ Characters

1. Number of 6-bit Data Format Characters.
2. Elementary Items:
   a. Does not include extra character positions for synchronization.
   b. For signed numeric items, size does not include a character position for the sign character or bit.
3. Group and Record Items:
   Size is the actual size (i.e., the actual number of character positions occupied by the item in memory).

File or Section Number

File Number = 1 - 72_8
D. COMON = 73_8
Report Section = 74_8
Working Storage = 76_8
Constant Section = 77
Procedure Division Literal = 77_8

COBOL FILE TABLE LEGEND

(See Table 4-3.)

Compiler Use Only

Block size: 1B maximum integer from Block Contains clause

Rec: = 0 if not records in Block Contains clause
= 1 if records in Block Contains clause

File number: position in External.Access Table (EAT)

Alternate areas assigned: integer from Reserve Alternate Area clause

SM: = 1 if same buffer and record area as another file.
SR: = 1 if same record area as another file but not the same buffer area.

Buffer/Record file number:

Not equal to file number of CIO buffer or record area if directly attached to another file.

Used in conjunction with "beginning of record area," and "end of record area," "first" and "limit" by the assembler to produce appropriate relocatable addresses for the record area pointers and CIO buffer area pointers respectively.

Line number: line number of current File Description

Words FET1 through FET13 are determined by the SCOPE I/O system and are described in the SCOPE 3.0 manual, which constitutes the basic reference for this part of the File Table.

Object Code Use

Size of ID, Data Written, Edition Number, Reel Number, and Retention Cycle label fields are assumed to be DISPLAY usage and are set by Pass 1D.

RRWL and associated File/Section number for the above listed label fields are also set by Pass 1D.

Pointer to Actual Key or Symbolic Key field and associated length usage may be used to store disk location for reading from and writing to the disk in random access.

SB: = 1 if Symbolic Key specified

The a, p, o and j fields (set by Pass 1E) are used by the object subroutine to point to a position in the JUMP table for execution of the appropriate DECLARATIVE section subroutine.

Table 4-3. COBOL File Table Legend

59 56 53 50 47 44 41 38 35 32 29 26 23 20 17 14 11 8 5 2 0

COBOL Compiler		Item Type 1 0 0	Data Type 1 0 0	Link to next item in Source Sequence	Link to Renaming FD or Zero	Link to next item with same name												FD
		Level Number 0 0 0 0 0 0	SM	SR	Rec.	Alt. Areas Assigned	Line Number	File Number	Buffer/Record File No.	Block Size								FD + 1
		Length of Actual Key Field	Pointer to Actual Key Field File or Section Number	Beginning Char. Pos'n	Usage	SB												+ 2
COBOL Subroutine		Size ID Field	Size Date-Written Field	Size Edition-Number Fld.	Size Reel-Number Field	Size Retention-Cycle Fld.	BCP	File/Sect. Number	RRWL for ID Field									+ 3
		BCP	File/Sect. Number	RRWL for Date-Written Field	BCP	File/Sect. Number	RRWL for Edition-Number Field											+ 4
		BCP	File/Sect. Number	RRWL for Reel-Number Field	BCP	File/Sect. Number	RRWL for Retention-Cycle Field											+ 5
		a	p	o	j	a	p	o	j	a	p	o	j	a	p	o	j	+ 6
		a	p	o	j	a	p	o	j	a	p	o	j	a	p	o	j	+ 7
		a	p	o	j	a	p	o	j	a	p	o	j	a	p	o	j	+ 8
		a	p	o	j	a	p	o	j	a	p	o	j	a	p	o	j	+ 9
		a	p	o	j	a	p	o	j	a	p	o	j	a	p	o	j	+ 10
		a	p	o	j	a	p	o	j	a	p	o	j	a	p	o	j	+ 11
SCOPE-FILE MANAGER	FET 1	Scope File Name - /fn - (7 characters)	Code and Status															+ 12
	2	Device Type	r	n	u p	E p		Disposition Code	L	FIRST								+ 13
	3	←——→ 0	IN															+ 14
	4	0 ←——→ 0	OUT															+ 15
	5	FNT Pointer	Record Block Size	Physical Record Unit Size	LIMIT													+ 16
	6		Working Storage fwa		Working Storage lwa+1													+ 17
	7		Record Request/Return Information															+ 18
	8	Record Number		Index Length	Index Address													+ 19
	9		EOI Address		Error Address													+ 20
Label Storage	10	Catalogue File Name (First 10 Characters)																+ 21
	11		Catalogue File Name (Second 10 Characters)															+ 22
	12	Edition Number	Retention Cycle	Creation Date (Julian)														+ 23
	13	Position Number	Multi-file Name	Reel Number														+ 24
User FET Area	14		(REC) Blocker Address		(REC) Deblocker Address													+ 25
	15	Disposal Code		Minimum Record Length (Logical)		Logical Status		Use Code	Record Type									+ 26
	16	Record Mark Value		Maximum Record Length (Logical)	Blocker Byte (REC)	Blocker Byte (CIO)	Size of Single Occurance of Trailer											+ 27
	17	BCP	File/Sect. Number	Key Position (Relocatable Address)	Deblocker Byte (REC)	Deblocker Byte (CIO)	M-Usage	Length of Variable Record Key Field										+ 28
	18	t	EOR	IN	OUT		Spacing Control	Size of Non-standard Label Record	Address of OP Word									+ 29
	19		Record Count															+ 30
	20		Fixed Record Length (Logical)	Record Count Rerun Period														+ 31
	21	OP		First Word Address (Beginning of Record Area)		Address of Main Element												+ 32
	22	Symbolic or Actual Key	Not Used															+ 33
		MAXIMUM RECORD AREA																

59 56 53 50 47 44 41 38 35 32 29 26 23 20 17 14 11 8 5 2 0

Beginning of record area is set by Pass 1D. The blocking and deblocking routines get the address of the record area from this field at object time.

Device type: not set at compile time, but will be set by the SCOPE system at object time. Check must be made at object time to assure that the CIO buffer set up by the compiler is adequate for the fixed block size associated with the device type. If CIO buffer is too small, abort job.

r: Random Access specified for file (set by compiler Pass 1A).

Code status: used to communicate status between central processor program and peripheral processor I/O routines.

FIRST: first word of CIO buffer (set by Pass 1D) relative to beginning of BLANK COMMON.

IN: during reading, SCOPE varies IN as it fills the buffer. During writing, the user varies IN as he fills the buffer.

OUT: during reading, the user varies OUT as he removed data from the buffer. During writing, SCOPE varies OUT as it removes data from the buffer.

LIMIT: last word of CIO buffer (set by Pass 1D).

Symbolic or Actual Key: see SCOPE manual for discussion.

EOI address: address of routine that is executed when EOF, BOF, EOR, or BOR is encountered by the File Manager (FM). Set at object time.

Error address: address of routine that is executed by the FM when one of the following conditions are encountered as indicated in bits 9-13 of Status field (FD9). Set at object time:

04 - unreceivable parity error
10 - device capacity exceeded
20 - OPEN function redundant
21 - CLOSE function redundant
22 - illegal function

t: 0 - Unlabeled file
1 - Nonstandard label (LABEL RECORDS ARE data-name)
2 - Standard label

Logical Status (6 bits)

XX10 XX: Count records for restart dump (set by Pass 1B)
XX01 XX: End of reel controls restart dump (set by Pass 1B)

Record type (set Pass 1D)

000001 = Fixed-length record.

000010 = Variable-length records--RECORD CONTAINS...DEPENDING on data-name clause.

000100 = Variable length records--OCCURS...DEPENDING clause.

Record mark: contains 62_8 when end of record is indicated by RECORD-MARK.

Spacing control: count of the number of lines to advance the pointer when the WRITE BEFORE ADVANCING or the WRITE AFTER ADVANCING option is used.

Record count: count of the number of records read or written for the file (set by object subroutine).

Record count rerun period: when record count equals the contents of this field, a restart dump is taken and the record count is set back to zero (set by Pass 1B).

Blocker address: address of subroutines that move data from the record area to the CIO buffer.

Deblocker address: address of subroutines that moves data to the record area from the CIO buffer.

Fixed Record Length (logical): contains the number of characters in the fixed-length records or zero length for variable length records (RECORD CONTAINS...DEPENDING) set by Pass 1D.

Size of a single occurrence of a trailer: field is set by Pass 1D.

M-Usage:	001 Display	100 Computational-1
	010 Computational	110 Computational-2

Maximum record length: contains the maximum length of the variable length records associated with the file (set by Pass 1B or 1D).

OP: = 1 - Buffer open and in use.
0 - Buffer not open and not in use.

UP: = 1 - Execute routine indicated in EOI address. Set at OPEN or CLOSE time.
0 - No extra label handling routines.

Key position: position of the first byte of the key field.

Key length: length of the key field.

Sort/Merge Routine Use Only

Disposal Code

Use Code

001 - if Sort file
000 - if not Sort file

File Manager Use Only

n = 1 Release record after reading. Not used by COBOL.

Disposition code: indicates disposition of file at job termination. Not used by COBOL.

FNT pointer: set at OPEN time by SCOPE to the location of the file in the SCOPE File Name Table.

Record block size: set at OPEN Time by SCOPE.

Physical record unit size: set at OPEN time by SCOPE. Set to the size of the physical record size of the indicated device.

Index length

Index Address

Logical Status (6 bits)

1XXXXX = File is OPEN (set by FM)
X1XXXX = Optional file (set by Pass 1B)

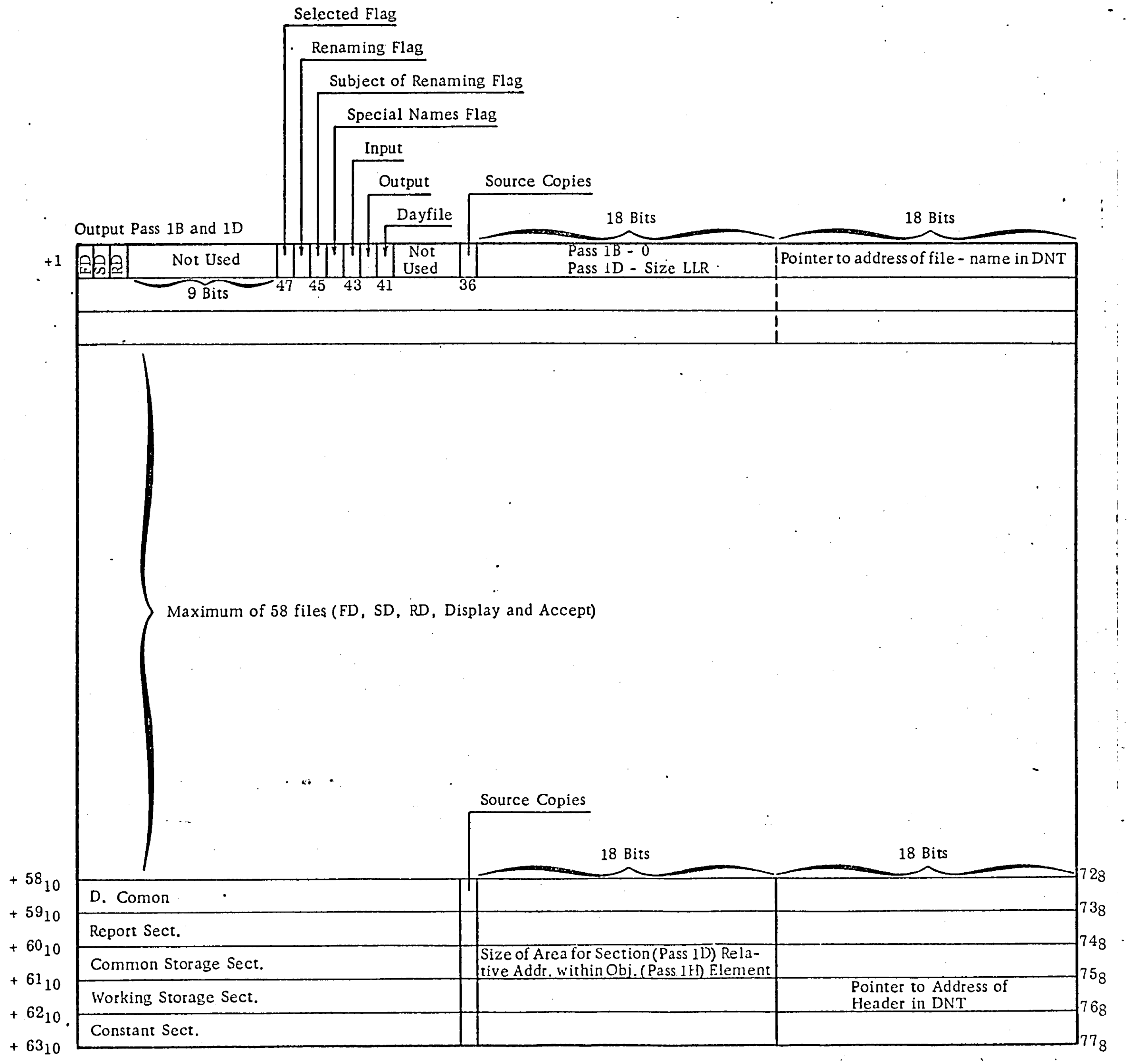
Position number: set by compiler to indicate position of file on multifile pool.

Multifile name: set by compiler to "COB" when file is assigned to a multifile pool.

EXTERNAL ACCESS TABLE

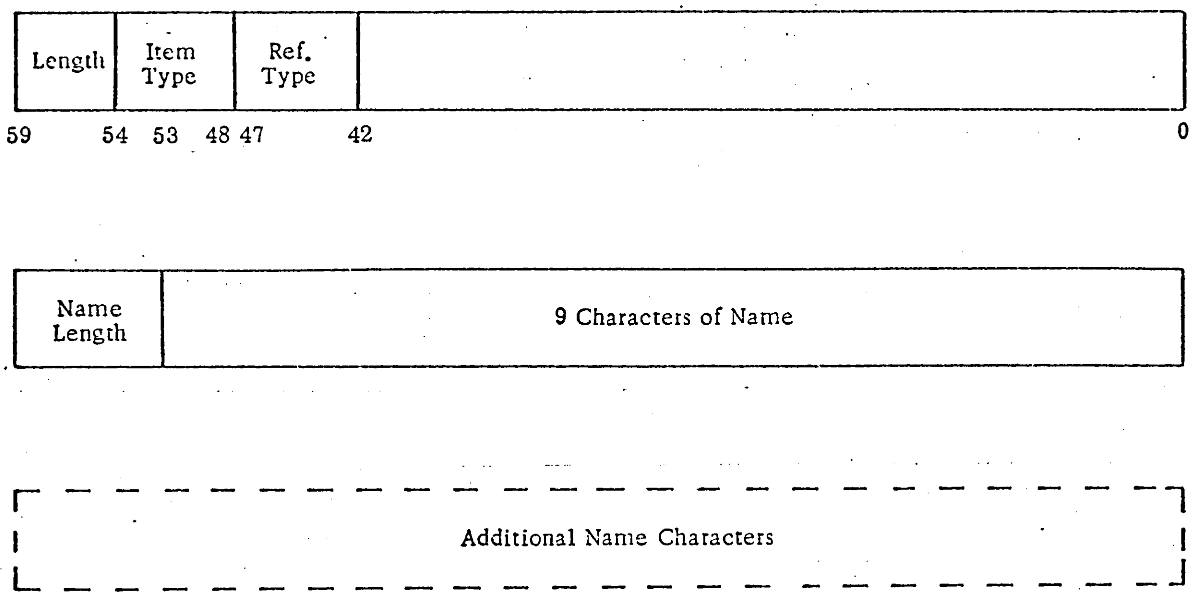
The External Access Table is shown in Table 4-4.

Table 4-4. External Access Table

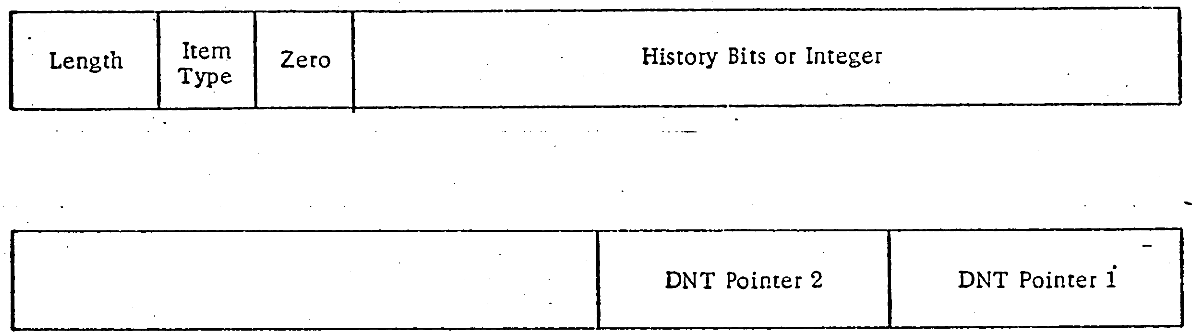


FORMAT OF ITEMS ON THE REPORT REFERENCE FILE (ON DISK)

Reference Item



Information Item



CODING OF REPORT REFERENCE ITEMS

If the reference type field is zero, there is not reference item. Otherwise each bit has a special meaning.

bit 42 = 0	integer_1 = data-name
bit 43 = 1	qualifier
bit 44 = 1	subscript
bit 45 = 1	FINAL

The item types are numbered in octal:

01	Control fields
02	Type CH reference to control field
03	Type CF reference to control field
04	Reset reference to control field
05	SOURCE reference
06	SOURCE SELECTED
07	SUM operand
10	SUM UPON
11	Beginning of RD (references report name)
12	End of RD (contains history)
13	End of report item (contains history)
14	LINE integer if bit 41 = 1
	LINE PLUS integer if bit 41 = 0
	LINE NEXT PAGE if bit 40 = 1
15	NEXT GROUP integer if bit 41 = 1
	NEXT GROUP plus integer if bit 41 = 0
	NEXT GROUP NEXT PAGE if bit 40 = 1
16	TYPE in which the types are expressed by the following integers:

01 = RH
02 = PH
03 = OH
04 = CH
05 = RF
06 = PF
07 = OV
10 = CF

REPORT TABLES

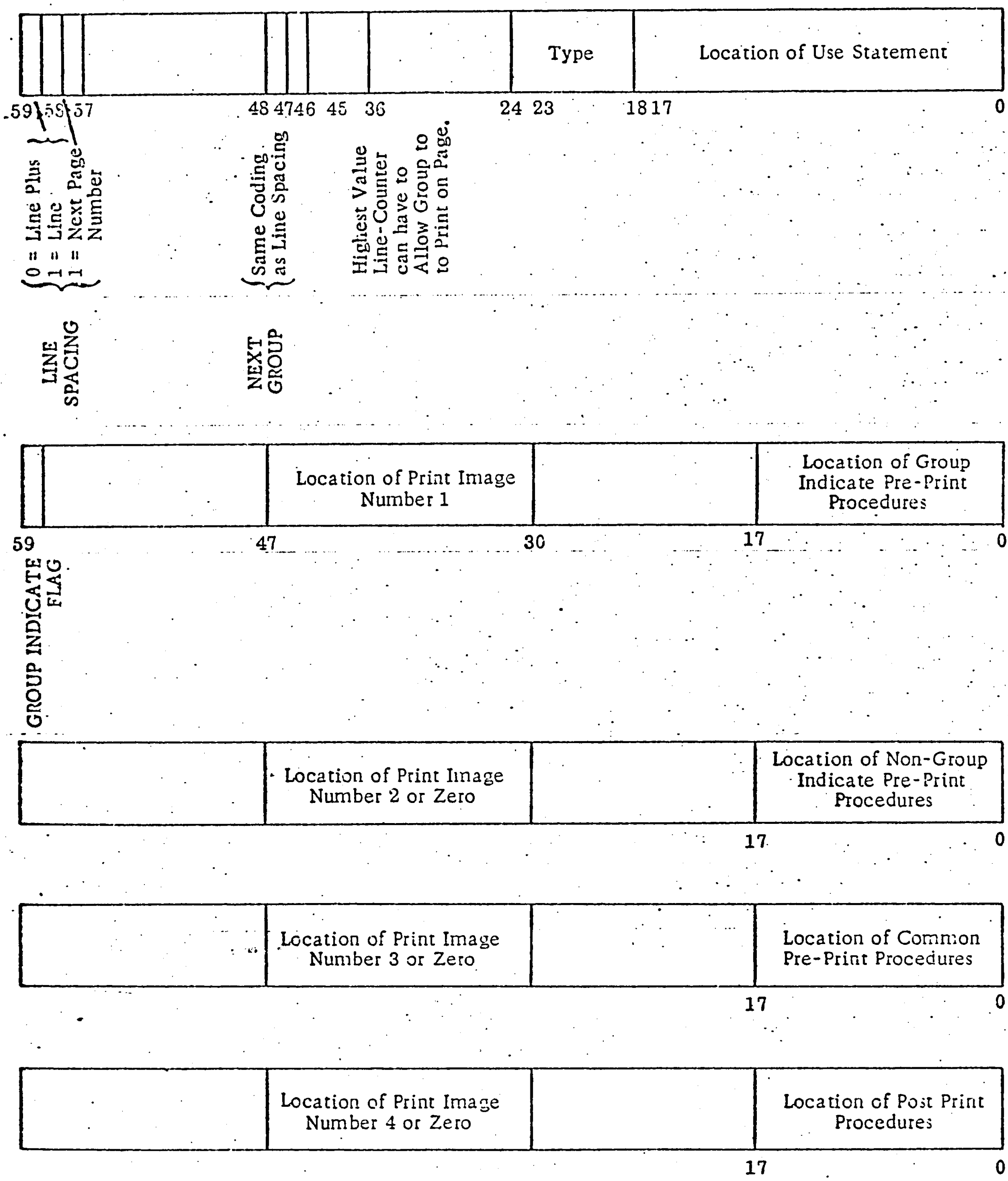
DETAILED DESCRIPTION OF THE REPORT MODULE

A single module contains the tables necessary for one report. This module cannot be produced by a COBOL program that is a subprogram. The least inclusive information in the report is produced first and the most inclusive last, in order that most references be backward references. The entry point to the most inclusive portion is the first seven characters of the report name. The structure of the module is:

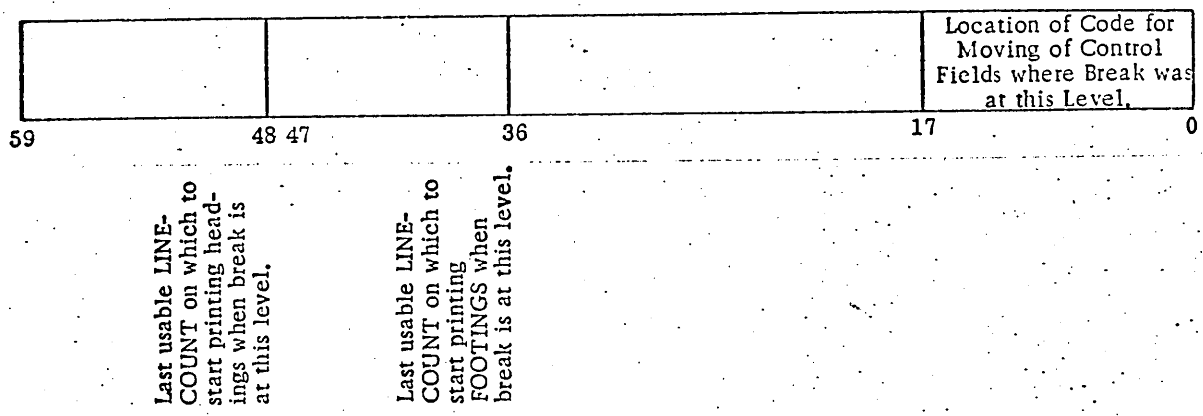
1. The entry point and a pointer to the table for the report as a whole.
2. Tables describing each Report Group.
3. Tables for each control level.
4. Tables for the report as a whole.

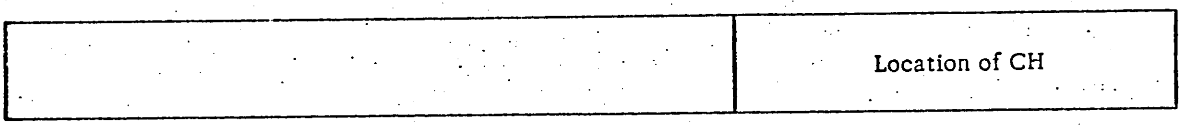
The data fields needed are output by pass 1H at the end of working storage. Their location relative to the beginning of working storage is known before pass 1G starts. Sequences of instructions needed are output as instruction trees and are generated by pass 2 as part of the Procedure Division. They are referenced using numbered tags as identification of the first instruction in each set.

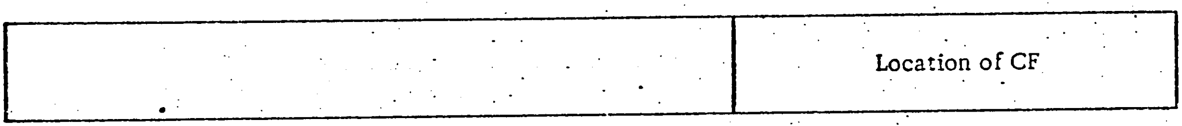
The report-groups are each generated with the following format:



For each control level the following information is contained in the table.

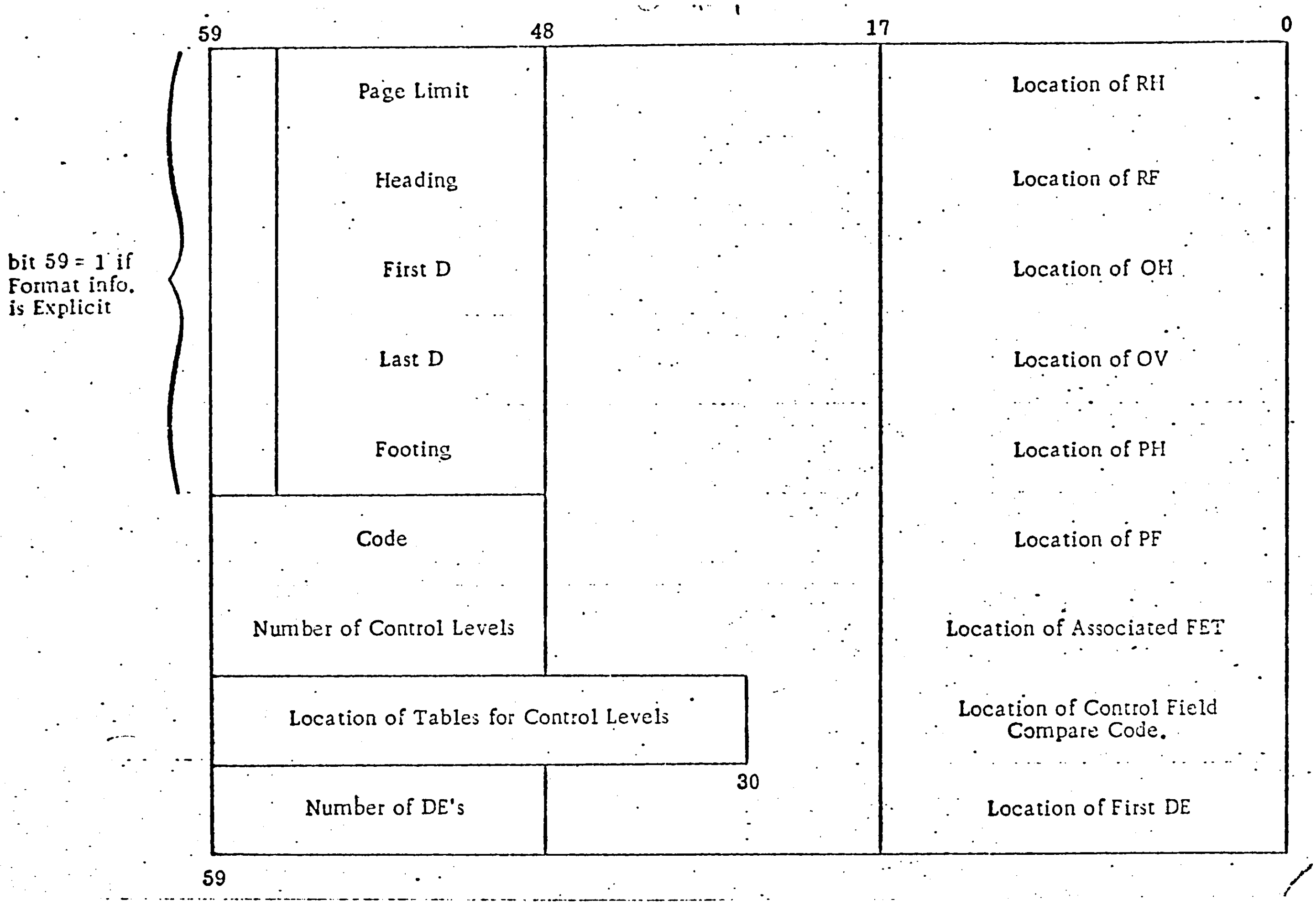






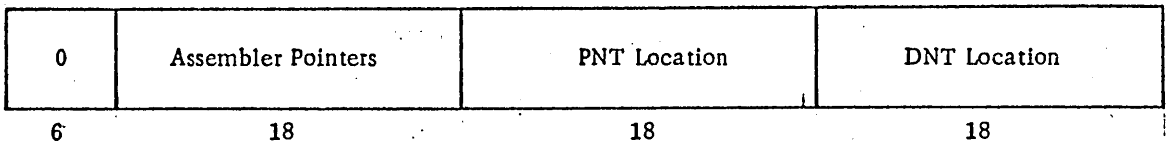
For the whole report the following information is contained in Table 4-5.

Table 4-5. Report Table



HASH TABLE

The HASH table entry format is:



The DNT field will also be used by the assembler during Pass 2 assemblies.

DIAGNOSTIC TABLE

The Diagnostic Table is shown in Table 4-6.

Table 4-6. Diagnostic Table

<table>
<tr><td>DAGNOS1</td><td colspan="5">DL C000</td><td colspan="5">DL C001</td></tr>
<tr><td></td><td colspan="5">DL C002</td><td colspan="5">DL C003</td></tr>
<tr><td></td><td colspan="5">DL C004</td><td colspan="5">DL C005</td></tr>
<tr><td></td><td colspan="10">⋮</td></tr>
<tr><td></td><td colspan="5">DL C096</td><td colspan="5">DL C097</td></tr>
<tr><td></td><td colspan="5">DL C098</td><td colspan="5">DL C099</td></tr>
<tr><td></td><td colspan="10"></td></tr>
<tr><td>C000</td><td colspan="10">Δ ←→ Δ</td></tr>
<tr><td></td><td>X</td><td>Δ</td><td>I</td><td>N</td><td>C</td><td>O</td><td>R</td><td>R</td><td>E</td><td>C</td></tr>
<tr><td></td><td>T</td><td>Δ</td><td>S</td><td>T</td><td>A</td><td>R</td><td>T</td><td>I</td><td>N</td><td>G</td></tr>
<tr><td></td><td>Δ</td><td>C</td><td>O</td><td>L</td><td>U</td><td>M</td><td>N</td><td>Δ</td><td>B</td><td>E</td></tr>
<tr><td></td><td>F</td><td>O</td><td>R</td><td>E</td><td>Δ</td><td>C</td><td>O</td><td>L</td><td>U</td><td>M</td></tr>
<tr><td></td><td>N</td><td>1</td><td>2</td><td>Δ</td><td>Δ</td><td>Δ</td><td>Δ</td><td>Δ</td><td>Δ</td><td>Δ</td></tr>
<tr><td>C002
⋮</td><td colspan="10">⋮</td></tr>
<tr><td>C099</td><td colspan="10">Δ ←→ Δ</td></tr>
<tr><td></td><td colspan="10">Δ ←→ Δ</td></tr>
</table>

LEXICON TABLE

The Lexicon Table is shown in Table 4-7.

Table 4-7. Lexicon Table (1 of 2)

ONEA	1	A Δ ——— Δ		
	1			
	1			
		⋮		
ONEZ	1	Z Δ ——— Δ		
SPEC	1	. Δ ——— Δ		
		⋮		
	1	/ Δ ——— Δ		
ENDEX	1	** Δ ——— Δ		
TWUWDS	2	T W O Δ W O R D Δ		
	E	N T R I E S Δ Δ Δ		
	2	etc.		
	2	E N D Δ O F Δ T Δ		
	O	Δ W O R D S Δ Δ Δ		
LEXNRS	00005	01725	00002	01602
	00003	01601	00001	00006
LEXPROC		ONEA		
+1	0	ONEB		

Table 4-7. Lexicon Table (2 of 2)

+25		ONEZ
+26		SPEC
+27		ENDEX
+28		ONEA
+29		LEXNRS
+30		LEXPROC
+31		TWUWDS

Note: The last four cells are needed by the LEXSRCH routine to establish limits for its searches.

LEXDATA and LEXPROC are lists of the same reserved words. Nonzero lexicon numbers are assigned to the words of each list if the words are legal in the appropriate division. Otherwise a zero lexicon number is assigned.

SYNTAX ANALYSIS TABLES

Each table entry consists of two words; each word is divided into four quarters; each part consists of a 3-bit clue, and a 12-bit parameter. The quarters are executed in order: 1st quarter, 2nd quarter, 3rd quarter, 4th quarter.

3	12	3	12	3	12	3	12

See Section 2, Syntax Analysis, for a description of these words.

SECTION 5

SECTION 5 - COMPILER OUTPUT

STRUCTURE OF LOAD

File tables, report tables, and common storage areas are generated in elements (subprograms) separate from the Procedure Division code elements, and are placed in the main overlay if overlays exist. When the user has more than one compilation, he must ensure that elements with duplicate names, such as names for file tables, are removed. (See Tables 5-1 and 5-2.)

OBJECT CODE FORMATS

Binary output table formats for each subprogram (or element) are specified by the SCOPE Loader. The formats used by the COBOL compiler output are described here briefly:

1. Each table within the subprogram output begins with an identification word

ID	CN	Not Used	WC	Not Used	LR	L
	59 54	53 48	47 36	35 27	26 18	17 0
	6	6	12	9	9	18

CN Code Number - to identify the table (represented using octal values in this writeup).

WC Word Count - number of words in table excluding ID word.

LR Relocation Indicator - not used in some tables:

0 - L absolute, relative to RA.
1 - L relative to beginning of subprogram.
3 - L relative to 1st entry in Local COMMON Table (LCT)

L Location - not used in some tables: Initial address of data described in this table as relocated by LR above.

2. Each subprogram (element) has a PIDL (program identification and length) table. The ID word for PIDL is:

ID	34	0	WC	0	0	0

Table 5-1. Binary Output from COBOL Compiler

Entity	Name	Entry Point	Content	Conditions for Presence
CARD	OVERLAY	N/A	OVERLAY (COBCODE, 0, 0)	See 9.2.6, page 9-3 (COBOL ERS)
Relocatable Deck	D. COMON	D. COMON	TALLY FILE-LABEL POINTERS TO LISTS FET LIST REPORT LIST	Not present in subcompilations.
Labeled COMMON Relocatable Deck	CCOMMON	CCOMMON	COMMON STORAGE FIELDS	Not present in subcompilations.
Relocatable Deck	Implementor-name	Implementor-name	FET Record Storage Area CIO buffer open indicator	Not present in subcompilations. (One for each file)
Relocatable Deck	Report-name	Report-name	Encoded Report Sum, line, page counters Report group output areas	Not present in subcompilations. (One for each report)
Blank Common		N/A	CIO buffer areas	Not present in subcompilations.
Relocatable Deck	PROG-ID (7 characters only)	As defined by ENTRY's.	See next table.	Always present for any compilation (Base Section).
CARD	OVERLAY	N/A	OVERLAY (1, n)	Section with priority n + 50.
Relocatable Deck	C. section-name	As defined by ENTRY's.	See next table.	Section with priority n + 50.
EOF Mark	N/A	N/A	Tape mark.	Will be written over and deleted by any ensuing compilations.

Table 5-2. Structural Details – Relocatable COBOL Output Decks
(Except Files and Common)

Deck Type					
Main Compilation			Subcompilation		
	Base	Overlay	Base	Overlay	DECK PART (In Order of Occurrence)
1.	x				INITIAL JUMP - start of execution
2.	x				JUMP TABLE - entries for declarative procedures
3.		x		x	JUMP TABLE - entry points for inter-overlay jumps
4.			x		WORKING STORAGE AND CONSTANT STORAGE when present in source
5.	x	x	x	x	"INDEX" TABLE when needed for inter-overlay and inter-compilation jumps
6.	x				DECLARATIVE PROCEDURES when present in source
7.	x	x	x	x	REMAINDER PROC DIVISION for base or this overlay
8.	x	x	x	x	INTER-OVERLAY JUMP provided as required
					ENTRY POINTS
1.	x				CENT. 00 on first word of deck
2.		x		x	CENT. nn (nn is overlay number) on first word of deck
3.			x		PROG-ID (7 characters only) on first word of deck
4.	x	x	x	x	Data-name in ENTRY statement. Occurs on the procedure so named.

Notes: If there are no non-overlay sections in a Main compilation, items 1. and 8. will be present, and item 7. will be missing in the base deck. OVERLAY decks are produced only when 7. is present.

At least one other word must be present and consists of the subprogram name and length (length entry optional):

Subprogram Name	Subprogram Length or 0
59 18	17 0
42	18

This may be followed by a word of the same format having a name for Named COMMON and 7 blanks for Blank COMMON. This word appears only in file elements and Named COMMON elements and is the LCT.

3. Object code or data is output by means of TEXT tables. Any number of TEXT tables in any order may be used to create a subprogram. The length of any TEXT table cannot exceed 16 words in addition to the first (ID) word. One word is for relocation and there are up to 15 words of text.

The ID word for TEXT is:

ID	40	0	WC	0	LR	L

The first word of the TEXT table is relocation information for the text in this table:

R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
59 56	55 52	51 48	47 44	43 40	39 36	35 32	31 28	27 24	23 20	19 16	15 12	11 8	7 4	3 0
4	4	4	4	4	4	4	4	4	4	4	4	4	4	4

There is one R field for each text word. Unused R's are 0. Relocation is relative to the subprogram origin. Possible values of R are:

0	No relocation
2	Lower address
4	Middle address
8	Upper address
10	Lower and upper addresses.

The text or data words follow the relocation word. There are WC-1 text words and WC-1 R's in the table. The R's start in the high-order portion of the relocation word. The first text word is loaded at the address defined in the ID word by LR and L. The second text word (when there is one) is loaded in the next sequential address, etc. LR is 1 for all elements except named COMMON elements where it is 3.

4. External references are indicated by means of a LINK Table. The ID word for LINK is:

ID	44	0	WC	0	0	0

Entries in the table consist of an external symbol in a name word:

Name of External Symbol	0
59 42 18	17 18 0

followed by any number of 30-bit data bytes packed two to a word, where each byte describes a reference to the external symbol described in the name word. Any number of such combination of entries can exist in the LINK table.

I	P	RL	LOC
29 1	2	9	18 0

LOC Address of the word containing the reference to the external symbol

RL Relocation of LOC
0 - If absolute (relative to RA)
1 - If relative to beginning of subprogram

P Position in word at LOC of external reference
0 - Lower
1 - Middle
2 - Upper

5. Entry points to the subprogram are indicated in the ENTR (Entry Point) table. (S) One table is generated for every entry point in the program. The ID word for ENTR is:

ID	36	0	2	0	0	0

The entry in the table is two words long. The first of the two is the name of the entry point.

Name	0
59 42 18	17 18 0

The second word of the entry pair is the location of the entry point.

0	RL	LOC
59 27	26 9 18	17 18 0

<u>LOC</u>	Address of entry point
<u>RL</u>	Relocation of LOC 0 - If absolute 1 - If relative to the beginning of subprogram 3 - If relative to 1st entry in Local COMMON Table (LCT)

6. Binary output tables for each subprogram must terminate with a XFER (transfer) table. This table may optionally contain the name of an entry point which can be used by the loader as a pointer to a spot in the code. The name of the first instruction of the subprogram is used as the XFER table entry point for each overlayed priority section. The XFER table of the main routine must contain the entry point of the first executed code. Data Division tables do not contain a XFER table.

The format of the ID word of XFER is:

ID	46	0	1	0	0	0

The format of the entry point word (may be zero) is:

	Name	0
	42	18
bits	59 — 18	17 — 0

7. For file elements, FILL Table is also generated. The format of the ID word of FILL is:

ID	42	1	WC	0	0	0

All remaining words are partitioned into sets of 30-bit bytes; each set is headed by one control byte and followed by an indefinite number of data bytes. All bytes will be contiguous. The last byte may be zero. The control byte contains information concerning each of the subsequent data bytes until another control byte is encountered.

A zero byte is treated as a control byte.

The format of the control byte is:

0		AR
1	20	9
29		0

where AR is the relocation of the value in the address position of a word specified in the succeeding data bytes.

AR has the values:

0 - absolute, relative to RA (no relocation)
1 - program relocation
3 - relative to COMMON block in position AR-2 of LCT.

One control byte suffices for several data bytes.

The format of the data byte is:

1 1	P 2	RL 9	LOC 18
29			0

where

P is the position within the word of the address specified by RL and LOC

10 - upper
01 - middle
00 - lower

RL is the relocation of the address specified by LOC.

LOC is the address of the data word to be modified. The contents of the address field position (P) at location LOC relative to RL will be added to the origin as specified by AR in the control byte.

8. Overlays output by the code assembly are preceded by an overlay card generated before the ID word of the PIDL table for the overlay subprogram. The format of the card is:

OVERLAY (Fn, L1, O)

Fn	Output file name.
L1	Primary overlay number; 0 if the main routine, and section number less 49 for an overlay priority section.

LISTINGS

See Figures 5-1 and 5-2.

Source Listing

CC	Carriage control character.

From Compiler:

Line Nr.	Sequential number produced by the compiler.
Def. Nr.	Number taken from a referenced item. Used as a debugging aid by the source programmer.
Lib. Line Nr.	Number assigned by COBOL Library Update program during previous update.
New or Δ	This field contains the word "NEW" if the copied item was added or changed on the previous update. Blank if not added or changed.

From Update:

New Line Nr.	Sequential number produced and assigned by the COBOL Library Update program during the update.
Old Line Nr.	Number assigned during previous update.

Octal Listing

An octal listing of the compiler's relocatable binary output is available upon request by the O listing option on the COBOL control card. Fields on the print line are as follows:

1. Octal location: instruction location relative beginning of element.
2. Octal instruction (or data word): the 15-, 30-, or 60-bit instruction or data.
3. Flag: type of address

   E = External reference
   F = Forward procedure reference
   L = Reference to Forward local label
   N = Reference to literal
   R = Relocatable reference
   Δ = Absolute address

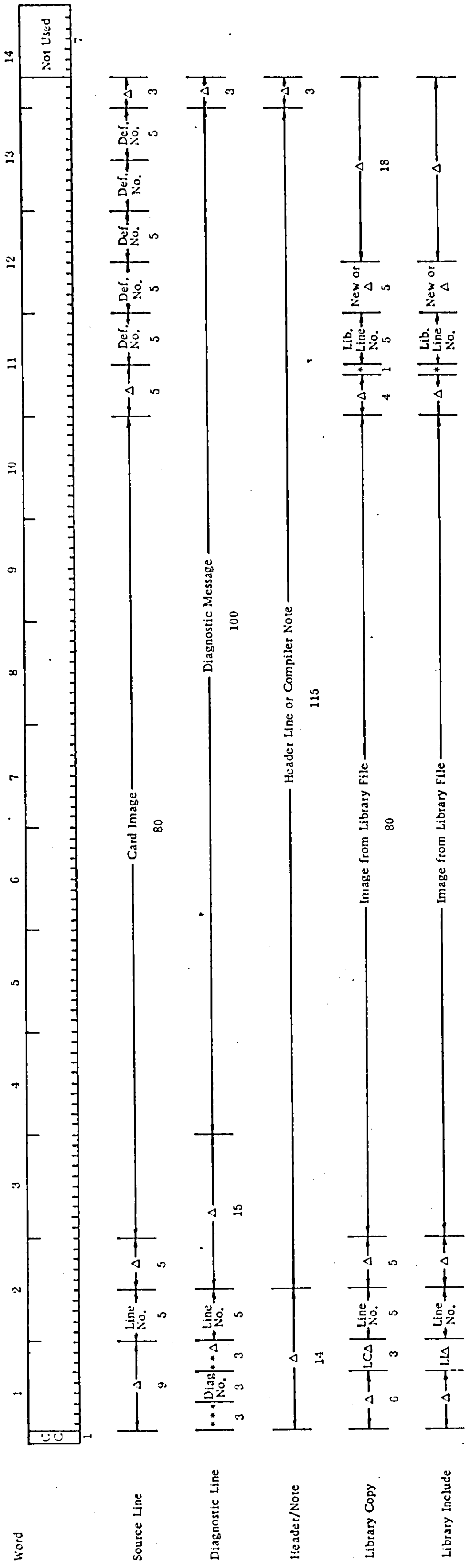


Figure 5-1. Compiler Print Lines

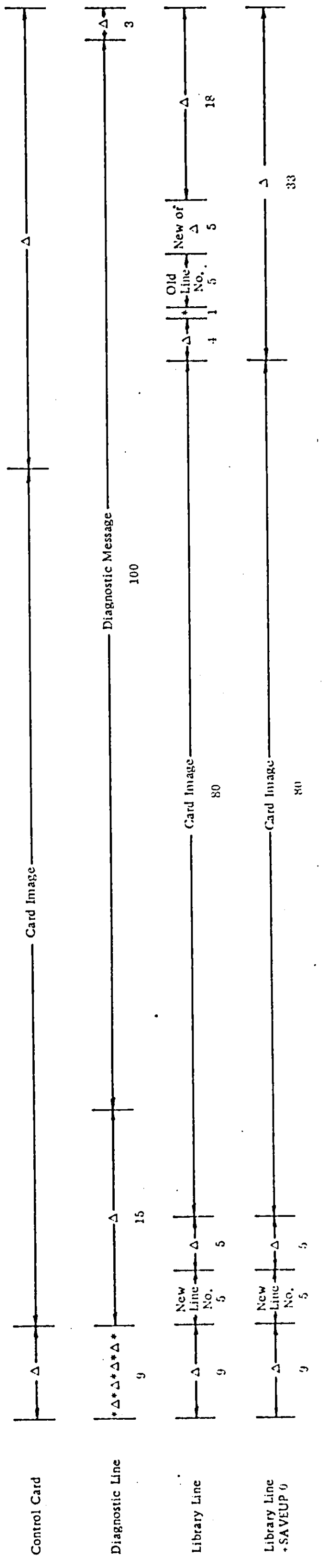


Figure 5-2. COBOL Update Print Lines

4. Symbolic location.

5. Mnemonic instruction code.

6. Symbolic address or octal literal.

7. Source line number: approximate line number within the source program of procedure statements for which the octal code is listed.

8. Source line number of reference: the approximate line number of the procedure-name or data referenced.

Since the assembly process is completed in a single pass, certain addresses cannot be completed at the time the line is printed. In this case, a symbolic address is given.

Four situations are handled this way:

1. References to external symbols. In this case the name of the external symbol appears in the symbolic address.

2. Forward references to "local" compiler generated symbolic locations (such as NEXT SENTENCE, the ELSE branch of an IF, or the VARYING code for a PERFORM) causes LOCAL 99 (where 99 is the local symbol number) to appear in the symbolic address. Later in the listing the local symbol should appear in the symbolic location field. Since a local symbol may be used over many times, the reference is to the first following appearance of the local symbol.

3. Forward references to procedure names cause seven characters of name to appear in the symbolic address. The beginning of each procedure is indicated also by the procedure name appearing in the symbolic location field. Completed references to procedure names also cause the procedure name to be printed, but the octal instruction contains the correct address.

4. References to literals cause the 60-bit octal literal to appear on the print line, starting in the symbolic address field. The literal also appears again (when the literal table fills up) in its exact location.

Sample Line:

	Octal Loc		Octal Instruction		FIG. 9		Symbolic Location		Mnemonic Code		Symbolic Address or Literal	Source Line of Procedure	Ref. Source Line
ΔΔ	← 6 →	ΔΔ	← 20 max →	Δ		Δ	← 7 →	ΔΔΔ	← 4 max →	ΔΔΔ	← 30 max →	← 5 →	← 4 →

SECTION 6

SECTION 6 - OBJECT TIME ROUTINES

INTRODUCTION

COBOL compiled programs may call other programs explicitly or implicitly. Explicit calls are done by the ENTER verb (calling FORTRAN or COMPASS compiled programs) or GO TO and PERFORM verbs (calling COBOL compiled programs). Implicit calls are generated for library programs specifically designed to effect COBOL operations. There are two kinds of such programs: tables of constants and operating routines. Both may be differentiated into those that are called by one specific COBOL statement and those that perform a more universal function and may be called by different COBOL statements.

All the operating routines are entered, at least initially, by an RJ (return jump) instruction. In order to allow object time diagnostics to "trace-back" to a source code line number, every RJ of this sort generated in the main object code occurs in the left half of a word, with a source code line number in the right half (in binary form). Every RJ of this sort within a subroutine that calls another subroutine is in the left half of a word and has the address of its enter/return word in the right half of that word.

In general, subroutines do not save and restore most registers. However, most COBOL subroutines will restore X4 to all DPC zeroes, and B1 to a one. Also, all general purpose routines which can occur between presetting and testing A0 as the on-size-error-switch must restore A0.

OBJECT TIME REGISTER USAGE

The following register assignment is current for object time code. Note that these assignments hold only within statements, and pertain only to those statements that generate significant in-line code, i.e., arithmetic statements: IFs, MOVEs.

X0	Mask
X1	Current operand
X2	Low-order part of current operand if double precision
X3	Volatile
X4	33333333333333333333B
X5	Volatile
X6	Previous result
X7	Low-order part of previous result if double precision
B0	0
B1	1
B2	Sign of current operand
B3	Volatile, used in SUBSCR
B4	Volatile, used in SUBSCR
B5	Less volatile--variable index
B6	Less volatile--variable index
A0	Truth or falsity on size error

TABLES OF BINARY CONSTANTS

COMPUTATIONAL-1 describes a binary representation of decimal numeric values. Any indicated shifting, right or left, and rounding is accomplished by indexing into the tables described here and adding, subtracting, or multiplying by the item found. These tables include:

D. TNTHS	A table of single-precision fractional powers of ten constructed so that an F multiply of an item by a COMPUTATIONAL-1 single-precision number (i.e., binary integer with zero exponent and a bias) will result in a decimally accurate truncated value.
D. TENS	Unnormalized integer powers of ten.
D. FIVE	Unnormalized binary integer table of 5, 50, etc. When ROUNDED is specified in COMPUTATIONAL-1, the indicated 5 is added (F+) prior to right-shifting via D. TNTHS.

SUBROUTINES

BLOCKIO - OUTPUT CIO BUFFERING SUBROUTINE

Purpose

BLOCKIO buffers all snapshots and core dump printer images to be written onto the standard OUTPUT file.

Calling Sequence

```
RJ      BLOCKIO (WRITOUT)
VFD     30/IMAGE, 30/LENGTH
I/O     Interface
```

(See Compiler I/O Interface for detailed description of BLOCKIO.)

Routines Called

I/O write, BNDC5 CPC

Register Usage

All registers are preserved except Registers A5 and A7.

Operation

A special entry to BLOCKIO named FLUSH can be called at any time to dynamically write a logical record of buffer output in BLOCKIO's CIO buffer. The calling sequence to FLUSH is:

```
RJ      FLUSH (WRITER)
```

DDDATCN SUBROUTINE

Purpose

A subroutine to convert a BCD date of the form YYMMDD to the form YYDDD.

Calling Sequence

X6 = YYMMDD RIGHT-JUSTIFIED WITH BINARY ZERO LEFT FILL.
RJ = D.DATCN

Upon return X6 contains:

YYDDD = RIGHT-JUSTIFIED WITH BINARY ZERO LEFT FILL.

Routines Called

None

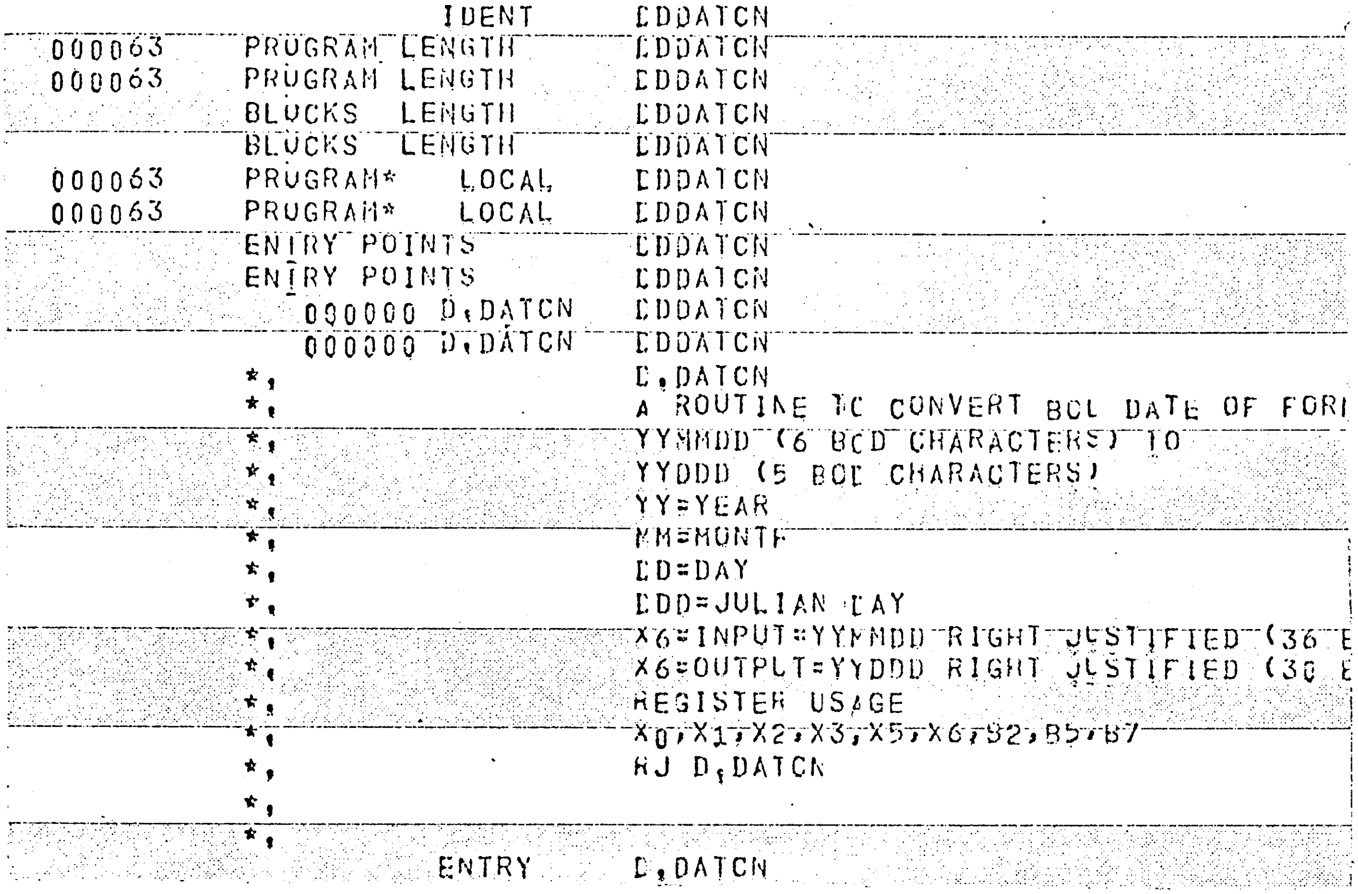
Register Usage

Registers X0, X1, X2, X3, X4, X5, X6, X7, B2, B5, and B6 are volatile.

All other registers are undisturbed.

Interface

See Figure 6-1.



```
                      IDENT      DDDATCN
 000063    PROGRAM LENGTH        DDDATCN
 000063    PROGRAM LENGTH        DDDATCN
           BLOCKS  LENGTH        DDDATCN
           BLOCKS  LENGTH        DDDATCN
 000063    PROGRAM*    LOCAL     DDDATCN
 000063    PROGRAM*    LOCAL     DDDATCN
           ENTRY POINTS          DDDATCN
           ENTRY POINTS          DDDATCN
              000000 D,DATCN     DDDATCN
              000000 D,DATCN     DDDATCN
           *,                    D,DATCN
           *,                    A ROUTINE TO CONVERT BCD DATE OF FOR
           *,                    YYMMDD (6 BCD CHARACTERS) TO
           *,                    YYDDD (5 BCD CHARACTERS)
           *,                    YY=YEAR
           *,                    MM=MONTH
           *,                    DD=DAY
           *,                    DDD=JULIAN DAY
           *,                    X6=INPUT=YYMMDD RIGHT JUSTIFIED (36
           *,                    X6=OUTPUT=YYDDD RIGHT JUSTIFIED (30
           *,                    REGISTER USAGE
           *,                    X0,X1,X2,X3,X5,X6,B2,B5,B7
           *,                    RJ D,DATCN
           *,
           *,
                      ENTRY      D,DATCN
```

Figure 6-1. DDDATCN Interface Printout

DDBCDCM - BCD COMPARE SUBROUTINE

Purpose

This subroutine compares two BCD items P and Q of length m and n. (See Figure 6-2.) When m and n are unequal, the shorter item is considered blank filled for the remainder of the comparison. When an inequality is established during the process of comparison, a table backup of the two different characters is made and one of two exits is taken--depending on the value of the characters in the collecting sequence.

Calling Sequence

```
COMPARE     PVS.Q
SX2         (Byte offset of P unnormalized floating point)
SB2         Base address of P
SB5         Length of P in bytes
SX3         (Byte offset of Q unnormalized floating point)
SB3         Base address of Q
SB6         Length of Q in bytes
SB7         Type of compare
RJ          D. BCDCM
TRUE        Return
FALSE       Return
Type 0      P = Q
Type 1      P > Q
Type 2      P < Q
Type 3      P ≥ Q
Type 4      P ≤ Q
```

Routines Called

None

Register Usage

Save B1.

Interface

See Figure 6-3.

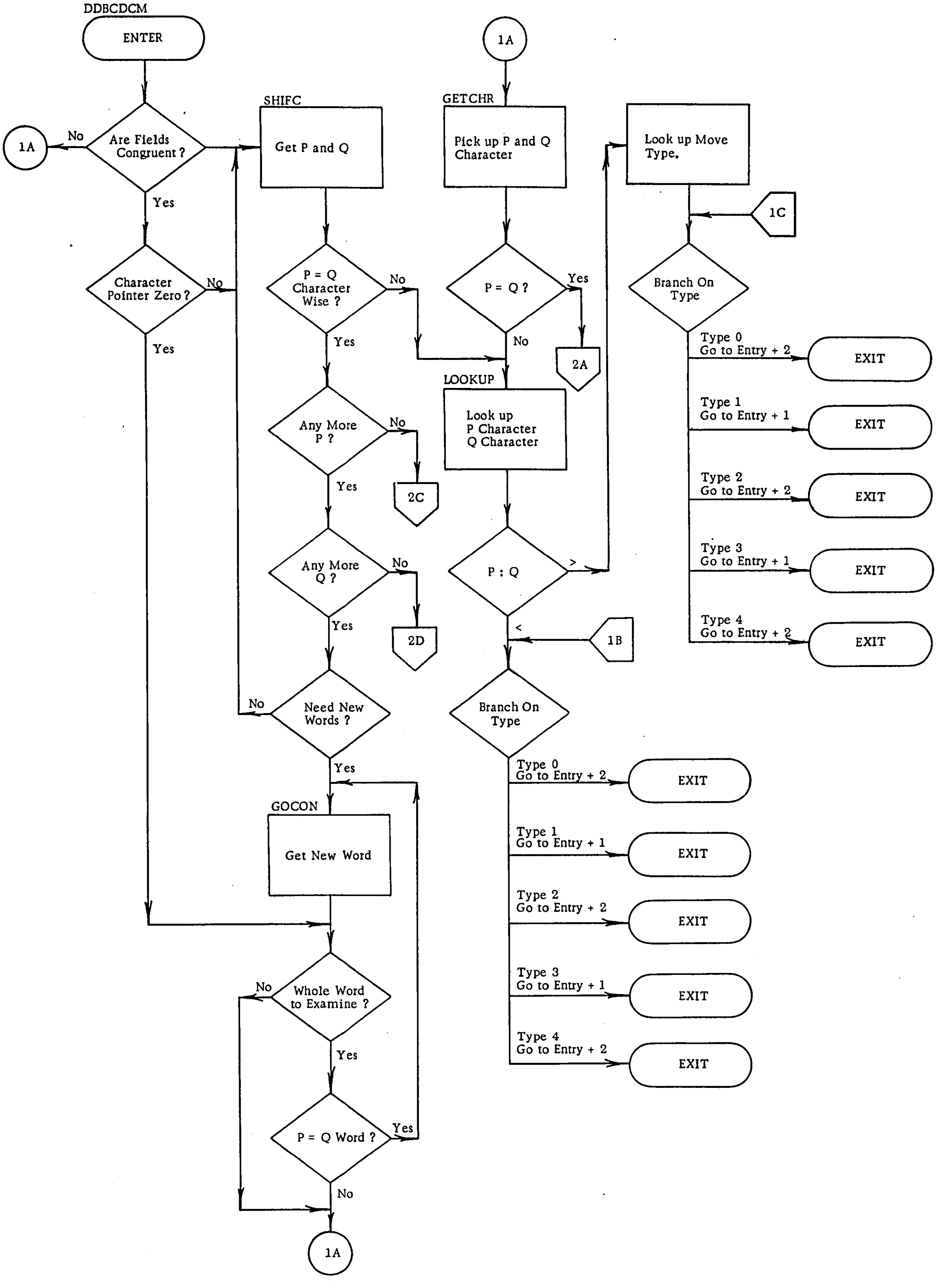


Figure 6-2. DDBCDCM Flowchart (1 of 2)

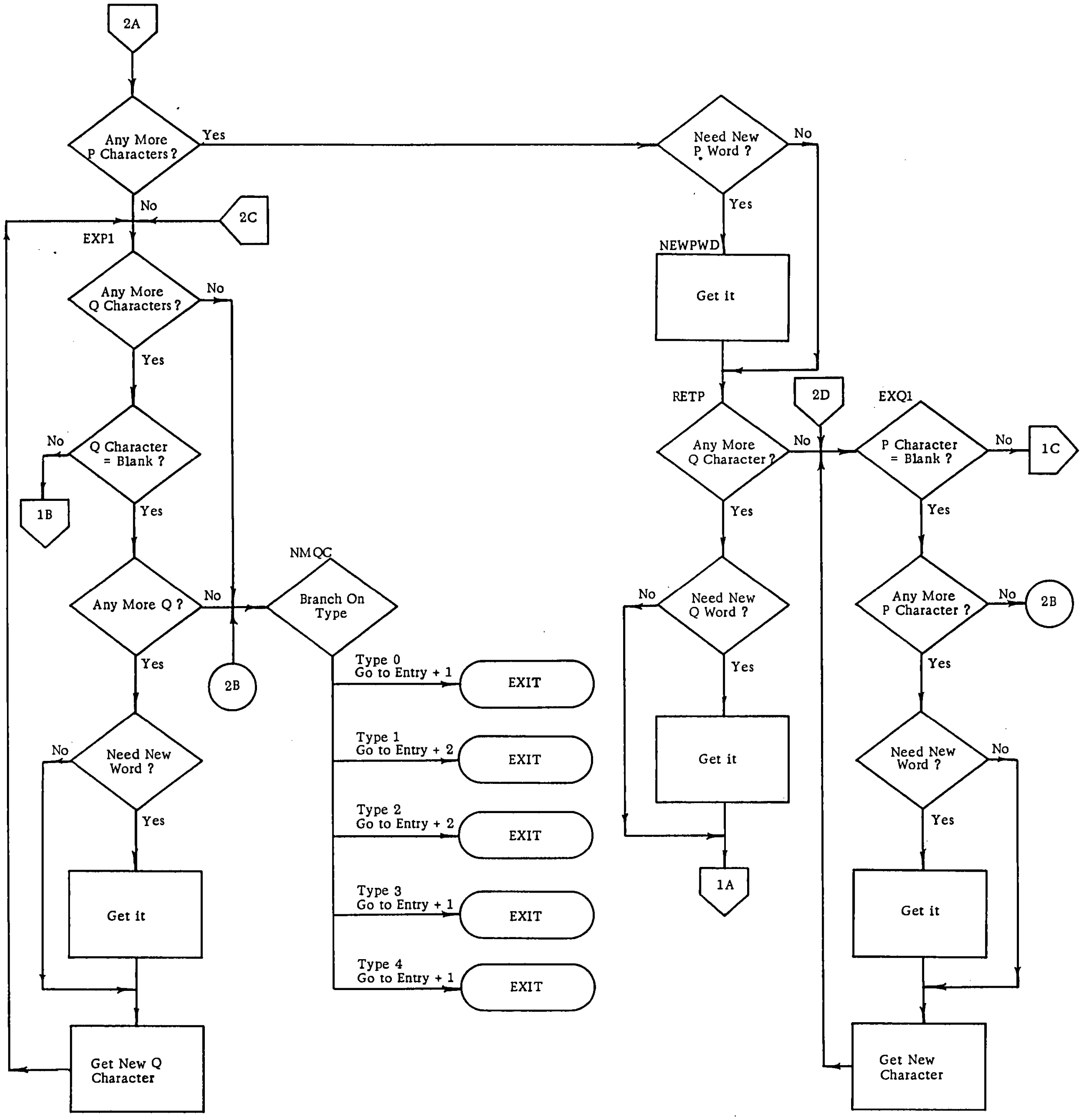
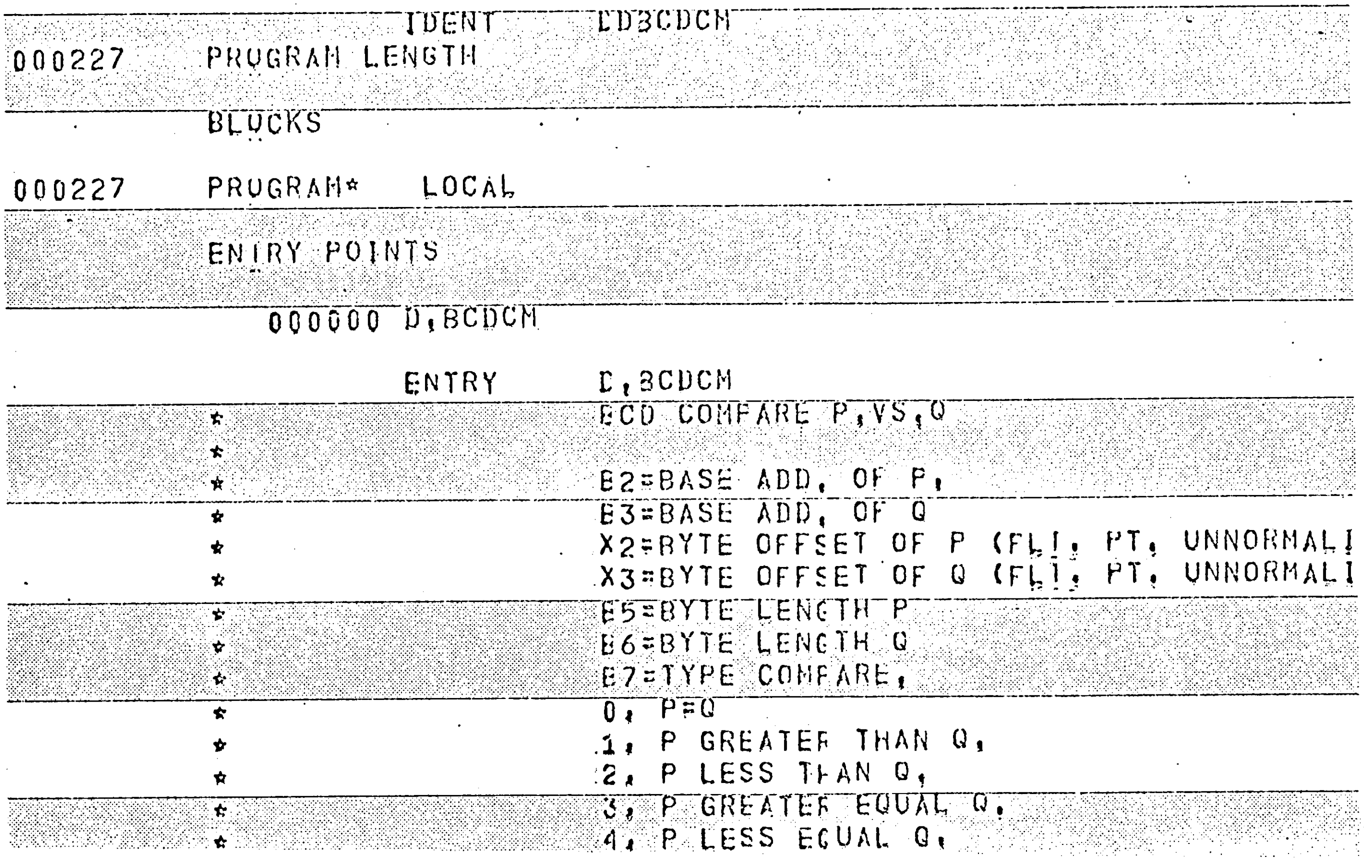


Figure 6-2. DDBCDCM Flowchart (2 of 2)



```
                 IDENT      DDBCDCM
000227    PROGRAM LENGTH

          BLOCKS

000227    PROGRAM*   LOCAL

          ENTRY POINTS

             000000 D,BCDCM

                     ENTRY      D,BCDCM
          *                     BCD COMPARE P,VS,Q
          *
          *                     B2=BASE ADD, OF P,
          *                     B3=BASE ADD, OF Q
          *                     X2=BYTE OFFSET OF P (FLI, PT, UNNORMALI
          *                     X3=BYTE OFFSET OF Q (FLI, PT, UNNORMALI
          *                     B5=BYTE LENGTH P
          *                     B6=BYTE LENGTH Q
          *                     B7=TYPE COMPARE,
          *                     0, P=Q
          *                     1, P GREATER THAN Q,
          *                     2, P LESS THAN Q,
          *                     3, P GREATER EQUAL Q,
          *                     4, P LESS EQUAL Q,
```

Figure 6-3. DDBCDCM Interface Printout

DDBN - BINARY CONVERSION SUBROUTINE

Purpose

The BCD to Binary Conversion (DBN) subroutine converts information from binary coded decimal (BCD) to pure binary. The inputs, referred to as arguments, must be in register X1 or X6 if the information to be converted is less than ten BCD digits. If the information to be converted is more than nine BCD digits and less than nineteen digits, the arguments must be in registers X1 and X2 or registers X6 and X7.

The register(s) containing the argument must be BCD zero filled and the argument itself must be right-justified. If the argument is negative, it will be carried in its 9's complement form.

Calling Sequence

There are six entry points or possible calling sequences. These six points are tabulated below:

Entry Point Name	Argument Register(s)	Result Register(s)	Registers Preserved
D. BN1SS	X1	X1	X6, X7
D. BN6SS	X6	X6	X1, X2
D. BN1DS	(X1, X2)	X1	X6, X7
D. BN6DS	(X6, X7)	X6	X1, X2
D. BN1DD	(X1, X2)	(X1, X2)	(X6, X7)
D. BN6DD	(X6, X7)	(X6, X7)	(X1, X2)

In general, the acronym used is DDBN1JK, where:

I = 1	X1 principle input register; X6 and X7 saved
I = 6	X6 principle input register; X1 and X2 saved
J = S	Single-register input
J = D	Double-register input
K = S	Single-register output
K = D	Double-register output

Routines Called

None

Register Usage

Save B1 and X4.

Operation

The following tabulation shows which entry point is entered to convert certain ranges of digits.

Number of Digits	Entry Point
0 through 9	D. BN1SS or D. BN6SS
10 through 14	D. BN1DS or D. BN6DS
15 through 18	D. BN1DD or D. BN6DD

The result from the conversion of digits in the range 0 through 14 is a single-precision, unnormalized, floating-point number. The result from the conversion digits in the range of 15 through 18 is a double-precision, normalized, floating-point number.

The general calling sequence is:

```
SA
SA
RJ      D. BNIJK
```

DDDSPLY - COBOL STATEMENT SUBROUTINE

Purpose

Low volume output.

Calling Sequence

DISPLAY A_1, A_2, ... A_{n-1}, A_n .

where the A_i are in display code and the total length of all $A_i \leq 120$. For display on the DAYFILE, the total length of all the $A_i \leq 40$.

SA1 LFN (before each new line to be displayed)

X3 = Byte offset of A_1
X4 = Fixed-point length of A_1
B4 = Base address of A_1

RJ D.DSPLY
X3 = Byte offset of A_2
X4 = Fixed-point length of A_2
B4 = Base address of A_2

RJ D.DSPLY
X3 = Byte offset of A_{n-1}
X4 = Fixed-point length of A_{n-1}
B4 = Base address of A_{n-1}

RJ D.DSPLY
X3 = Byte offset of A_n
X4 = Fixed-point length of A_n
B4 = Base address of A_n

RJ D.WRDSP

D.DSPLY FUNCTION

Stack A_i (i = 1, 2, ... n-1) consecutively for one output line on the output device.

D.WRDSP FUNCTION

Stack A_n immediately following last A_i and output the display line on the output device.

If there is only <u>one</u> item to be displayed, the calling sequence will be:

```
SA1     LFN (logical file name)
X3 =    Byte offset of item
X4 =    Fixed-point length of item
B4 =    Base address of item
RJ      D. WRDSP
```

<u>Routines Called</u>

DDMOVIO, CPC, IOWRITE.

<u>Register Usage</u>

B1 is preserved.

DDCOBIO SUBROUTINE

Purpose

The Input/Output (COBIO) subroutine is an object time subroutine residing in the object library. The COBOL compiler generates calls to COBIO as needed for the following statements. The following information describes various calling sequences to the Input/Output (COBIO) subroutine.

Calling Sequence

```
            Source Statement                         Generated Code

                                                      X6 = 0 = rewind
                                                      X6 ≠ 0 = no rewind
      OPEN INPUT LFN.                      SA1        LFN
      (Logical file name)                  RJ         D. OPIN (Open input LFN)

      OPEN OUTPUT LFN.                     SA1        LFN
                                           RJ         D. OPOT (Open output LFN)


                  I/O                      SA1        LFN
      OPEN (INPUT/OUTPUT) LFN              RJ         D. OPRAN


      READ LFN AT END                              SA1            LFN
      ANY IMPERATIVE STATEMENT                     RJ             D. READ
                                        ┌──── +JP            NORMAL
                                        │  ┌─ +JP            AT-END-CODE
      FOR ALL WORDS OF OCCURS KEY       │  │  [MOVE          OCCURS KEY]
                                        │  │  |[Key will be in X4 in ]|
                                        │  │  |[unnormalized float- ]|
                                        │  │  |[ing point           ]|
                                        │  │  [JP            D. MOCKR  ]
                                        │  └─▷[AT-END-CODE             ]
                                        └────▷

                                                      X6 = 0 = rewind
                                                      X6 = 2 = lock
                                                      X6 = 1 = no rewind
                                           SA1        LFN
                                           RJ         D. CRELR (Close LFN Reel)
```

Source Statement	Generated Code	
WRITE RECORD-NAME	SA1	LFN
	RJ	D. WRITE
	JP	NORMAL
	JP	INVALID-KEY
	MOVE	OCCURS KEY
	⋮	
	JP	D. MOCKW
WRITE RECORD-NAME BEFORE ADVANCING N LINES	SAI	LFN
	SX2 or SA2	X2 contains N
	RJ	D. WBA
	JP	NORMAL
	JP	INVALID-KEY
	MOVE	OCCURS KEY
	⋮	
	JP	D. MOCKW
WRITE RECORD-NAME AFTER ADVANCING N LINES	SAI	LFN
	SX2 or SA2	X2 contains N
	RJ	D. WAA
	JP	NORMAL
	JP	INVALID-KEY
	MOVE	OCCURS KEY
	⋮	
	JP	D. MOCKW
CLOSE LFN	SA1	LFN
		X6 = 0 = rewind
		X6 = 2 = lock
		X6 = 1 = no rewind
	RJ	D. CLOS (Close LFN)

Routines Called

DDMOVIO, CPC, I/O READ, I/O WRITE, I/O RA, I/O RW, DDDATCN.

Register Usage

Save B1.

Operation

The following subroutines constitute the DDCOBIO subroutine:

DDOPIN - Open Input
DDOPOT - Open Output
DDOPRAN - Open Input/Output
DDREAD - Read file name
DDRDNCH - Read N characters
DDWRITE - Write record name
DDWBA - Write before advancing
DDWAA - Write after advancing
DDWRNCH - Write N characters
DDCLOS - Close file name
DDCRELR - Close reel name
Use Declarative Section

These subroutines are described in detail on the following pages.

DDOPIN SUBROUTINE

Purpose

This subroutine is called when the COBOL statement "OPEN INPUT FILE-NAME" is encountered by the compiler. (See Figure 6-4.)

Calling Sequence

```
SA1        File-name
SX6        0 = rewind
SX6        ≠ 0 = no rewind
RJ         D. OPIN
Normal return
```

Routines Called

CPC

Register Usage

Register B1 preserved.

Operation

Its function is to ready the named file to accept input data from some specific device depending on the statements in the input/output section and the presence or absence of a SCOPE request card. If there are any USE declaratives that are applicable, they will be executed at the appropriate time.

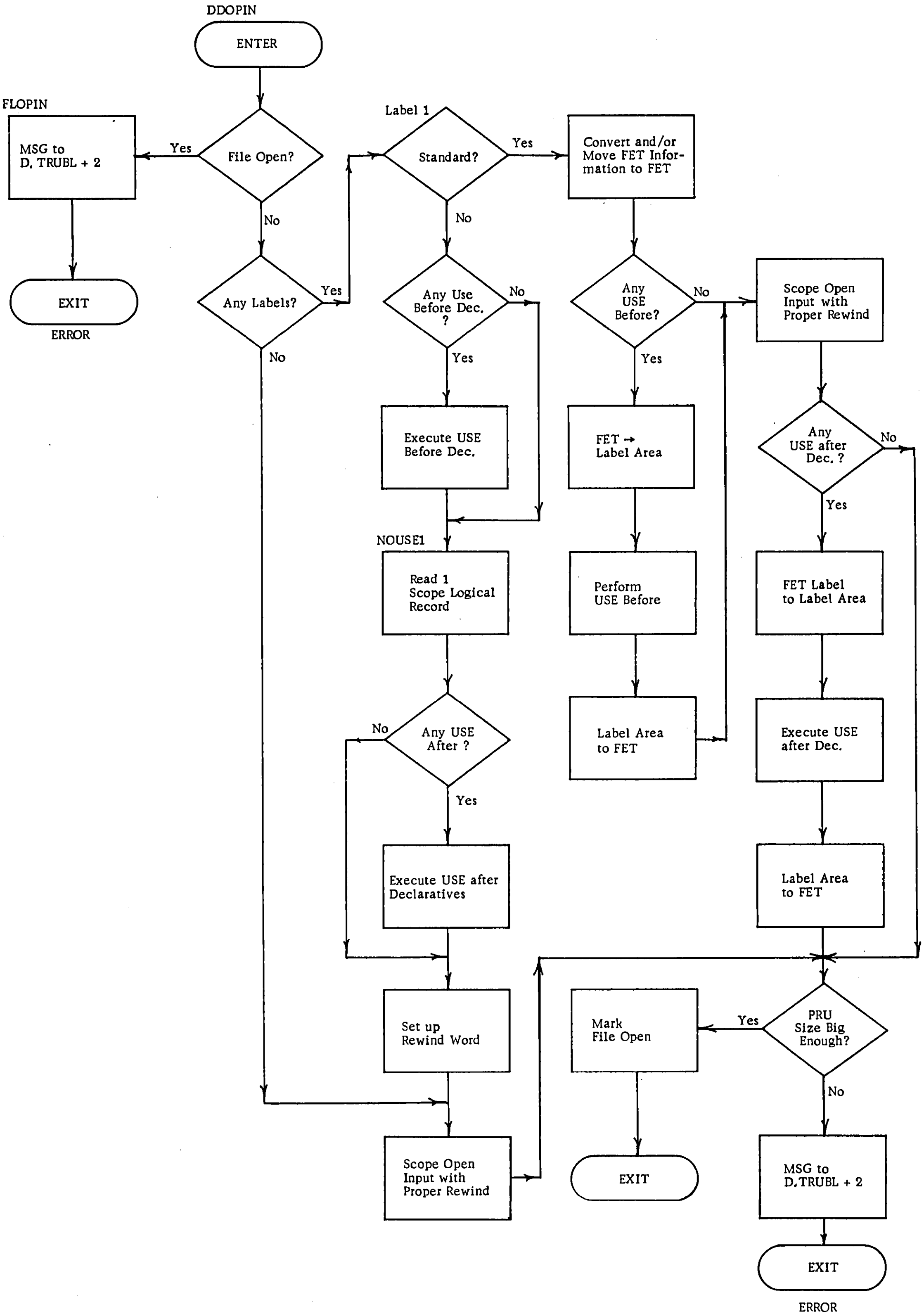


Figure 6-4. DDOPIN Flowchart

USE Declaratives

The USE Declaratives table associated with each FET has the following format:

- The table length is six words.
- Each word contains four 15-bit sectors. Each sector describes one USE declarative.
- A zero sector indicates the end of the table.

Each sector is composed as shown in Table 6-1.

Table 6-1. USE Declarative Sector Composition

bits 0, 1	Value	0 = File-name 1 = Input 2 = Output 3 = I/O
bits 2 - 6	Value	0 = Error procedure 5 = Before ending file 6 = Before ending reel 7 = Before ending reel and file 11_8 = Before beginning file 12_8 = Before beginning reel 13_8 = Before beginning reel and file 15_8 = Before beginning and ending file 16_8 = Before beginning and ending reel 17_8 = Before beginning and ending file and reel 105_8 = After ending file 106_8 = After ending reel 107_8 = After ending reel and file 111_8 = After beginning file 112_8 = After beginning reel 113_8 = After beginning reel and file 115_8 = After beginning and ending file 116_8 = After beginning and ending reel 117_8 = After beginning and ending file and reel

To facilitate the detection of certain combinations of these codes the following groupings were made and tables were generated for use by the detection and/or execution routines as shown in Table 6-2.

Table 6-2. Detection/Execution Group Table Guide

Table	Usage	Name of Table	Values (Octal)
1	Before beginning reel file	BBRF	11,12,13,15,16,17
2	After beginning reel file	ABRF	111,112,113,115,116,117
3	Before ending reel file	BERF	5,6,7,15,16,17
4	After ending reel file	AERF	105,106,107,115,116,117
5	After beginning reel	ABR	112,113,116,117
6	Before ending reel	BER	6,7,16,17
7	After ending before beginning reel	AERBBR	106,107,116,117,12,13,16,17

The usage of these tables is shown in Table 6-3.

Table 6-3. Table Usage Summary

Table	Used by These Subroutines
1, 2	Open input (D.OPIN), Open output (D.OPOT), Open I/O (D.OPRAN)
3, 4	Close file (D.CLOS)
5, 6, 7	Close reel (D.CRELR)

DDOPOT SUBROUTINE

Purpose

This subroutine is called when the COBOL statement "OPEN OUTPUT FILE-NAME" is encountered by the compiler. (See Figure 6-5.)

Calling Sequence

```
SA1        File-name
SX6        0 = rewind
SX6        ≠ 0 = no rewind
RJ         DDOPOT
Normal return
```

Routines Called

CPC

Register Usage

Register B1 preserved.

Operation

DDOPOT readies the named file to send output data to some specific device depending on the statements in the input/output section and the presence or absence of a SCOPE request card. If there are any USE declaratives that are applicable, they will be executed at the appropriate time. A description of the USE tables associated with each File Environment Table (FET) is shown in Tables 6-1, 6-2, and 6-3.

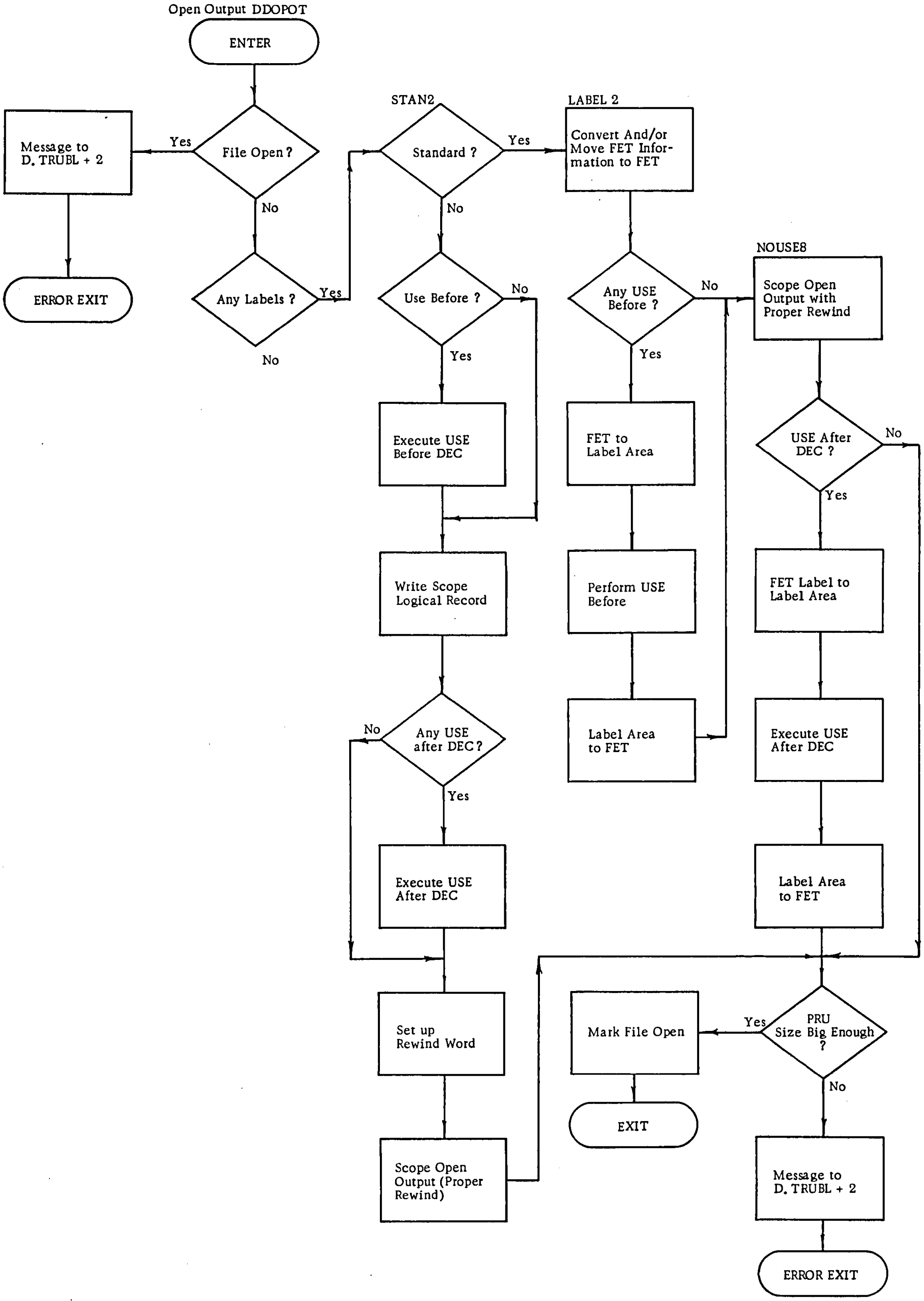


Figure 6-5. DDOPOT Flowchart

DDOPRAN - OPEN INPUT/OUTPUT SUBROUTINE

Purpose

This subroutine is called when the COBOL statement "OPEN INPUT/OUTPUT FILE-NAME" or "OPEN I/O FILE-NAME" is encountered by the compiler. (See Figure 6-6.)

Calling Sequence

```
SA1         File-name
RJ          D. OPRAN
Normal return
```

Routines Called

CDC

Register Usage

Register B1 preserved.

Operation

DDOPRAN readies a file to either accept input data or output data. Information going to or coming from the I/O device (always disk) will be processed according to the SCOPE definition of a random file. Each COBOL record will be a SCOPE logical record and will be word bounded (character length will be a multiple of ten). If the user does not describe the file in such a manner, the I/O routine will round the size of the record to a full-word boundary. If there are any USE declaratives that are applicable they will be executed at the appropriate time. A description of the USE tables associated with each File Environment Table is shown in Tables 6-1, 6-2, and 6-3.

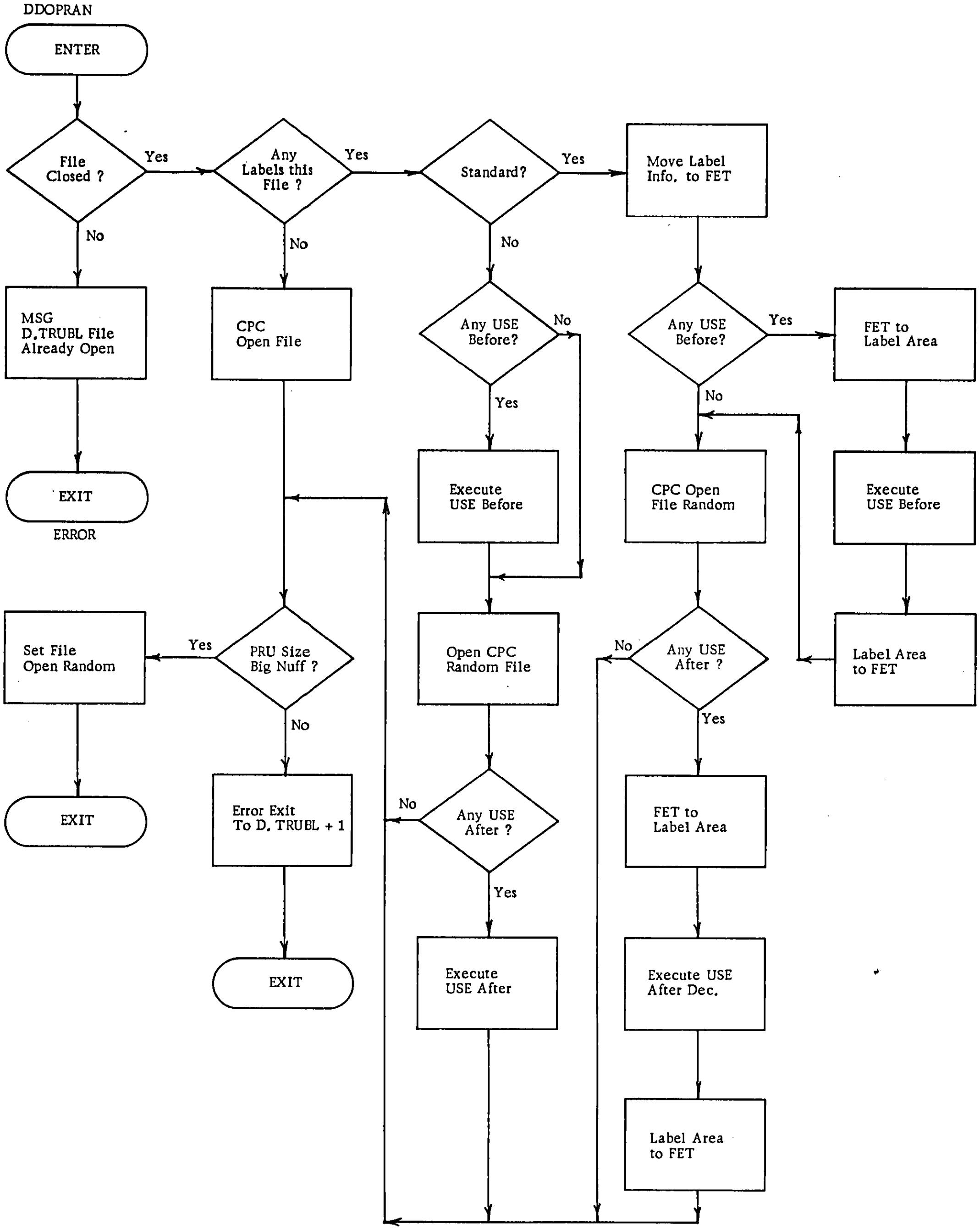


Figure 6-6. DDOPRAN Flowchart

DDREAD - READ FILE-NAME SUBROUTINE

Purpose

This routine is called for the source program word "READ." (See Figure 6-7.)

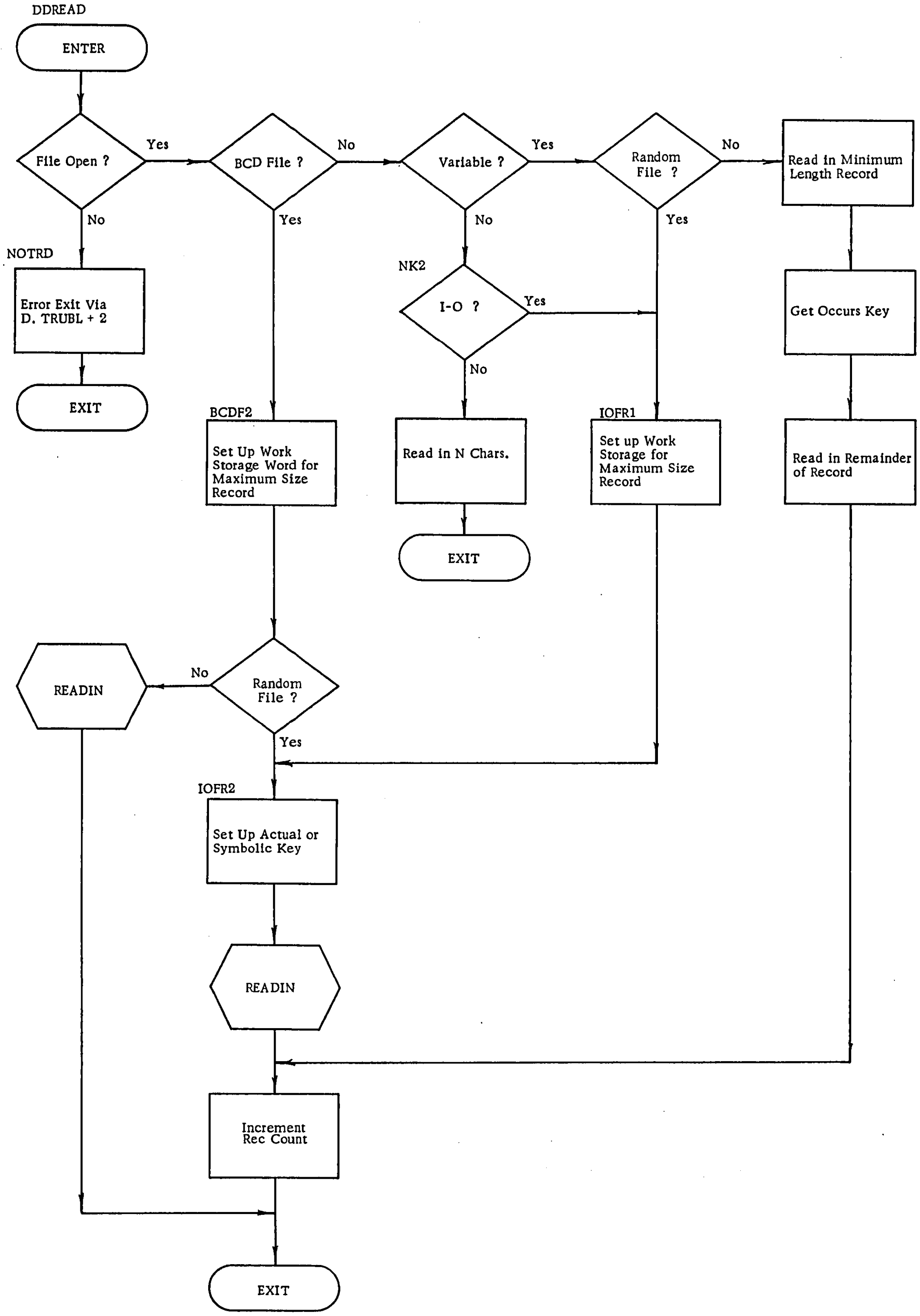


Figure 6-7. DDREAD Flowchart

DDRDNCH - READ N CHARACTERS SUBROUTINE

Purpose

This subroutine is called by DDREAD when the reading of a continuous binary file is indicated. (See Figure 6-8.)

Calling Sequence

```
SX6      N (number of characters to read)
SA1      File-name
RJ       DDRDNCH
Normal return (X3 ≥ 0 normally, X3 < 0 if end of file encountered)
```

Routines Called

CPC, DDMOVIO

Register Usage

Register B1 preserved.

Operation

DDRDNCH transfers information from the CIO area (SCOPE) to the user record area in a characterwise fashion. If a read cannot be satisfied with the information remaining in the buffer, DDRDNCH issues a read and go into recall status. When control is again returned to DDRDNCH, it continues the transfer of data to the user area until the read is satisfied, repeating the read with recall as often as is necessary. Upon completion of the read, a test is made and if the EOF bit is ON, it exits normally. Otherwise, a test is made to see if the file is busy. If not, a read is issued and the normal exit is taken. Upon entry if there are zero characters to transfer and the EOF bit is ON the AT END exit is taken.

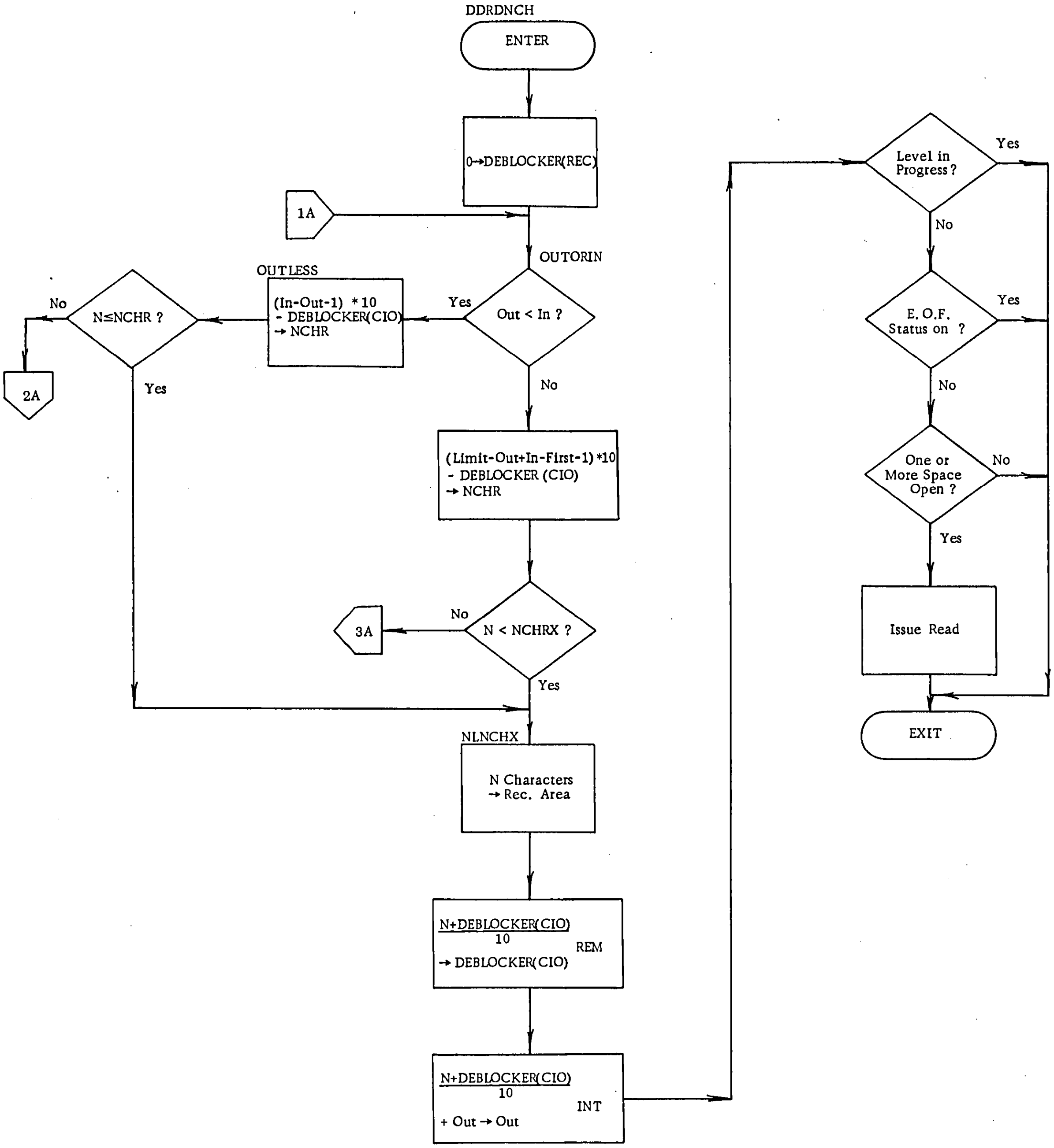


Figure 6-8. DDRDNCH Flowchart (1 of 3)

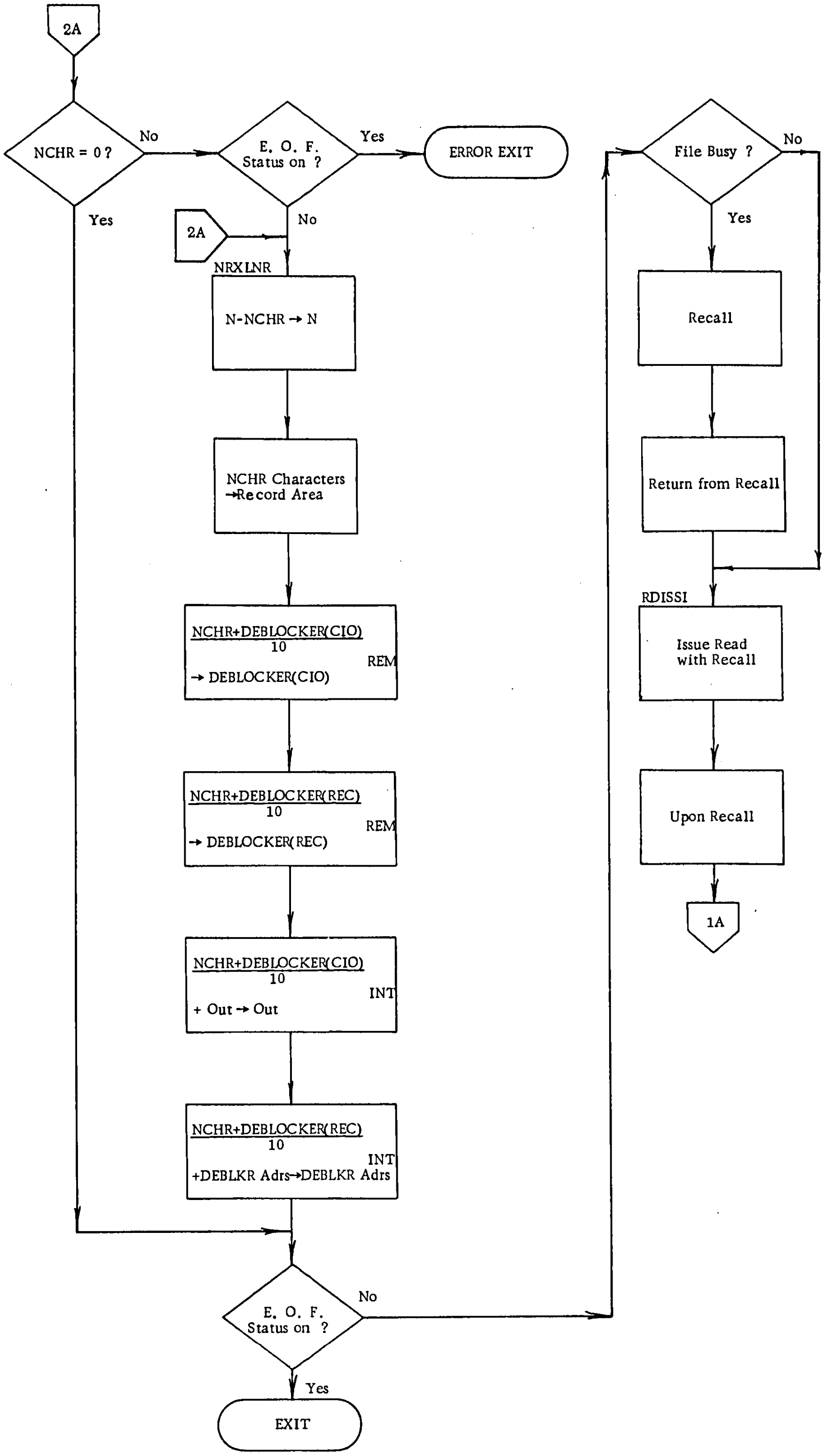


Figure 6-8. DDRDNCH Flowchart (2 of 3)

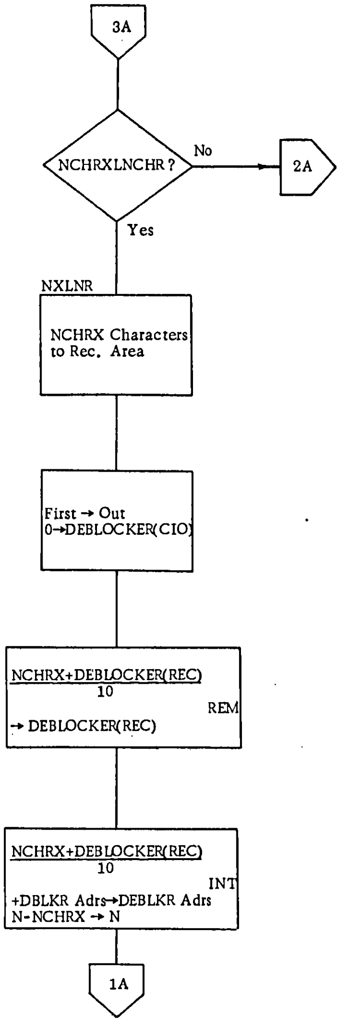


Figure 6-8. DDRDNCH Flowchart (3 of 3)

DDWRITE - WRITE RECORD-NAME SUBROUTINE

Purpose

This subroutine is called when the COBOL statements, WRITE record-name or Write record-name, invalid-key any imperative statement is encountered by the compiler. (See Figure 6-9.)

Calling Sequence

```
SA1          File-name
RJ           DDWRITE
┌─     JP    Normal Return
│  ┌─  JP    Invalid Key
│  │  [Move  occurs key  ]
│  │  [                  ]
│  │  [                  ]
│  │  [JP    D.MOCKW     ]
│  └▷[Invalid key code  ]
└▷Normal Code
```

{This code is generated only if the file contains variable-length records, (i.e., records are described with a record contains clause or an occurs clause with a dependency depending on data-name).}

Routines Called

CPC, IORW, IOWRITE

Register Usage

Register B1 preserved.

Operation

DDWRITE transfers information from the user record area to the appropriate output device.

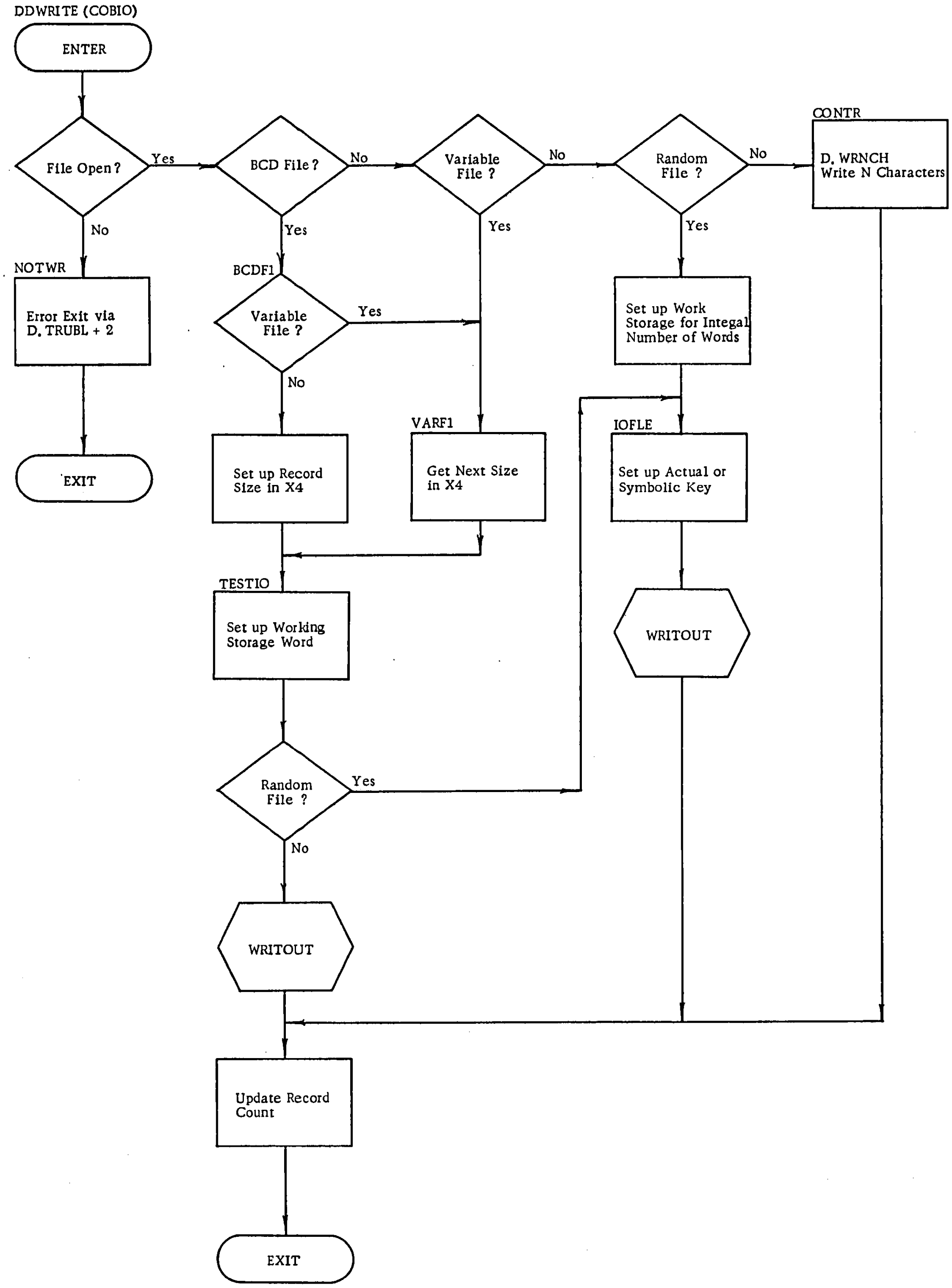


Figure 6-9. DDWRITE Flowchart

DDWBA - WRITE RECORD-NAME BEFORE ADVANCING SUBROUTINE

Purpose

This subroutine is called when the COBOL statement, "Write Record Name Before Advancing $\left\{\begin{array}{l}n\\ \text{Data Name}\end{array}\right\}$ Lines" is encountered by the compiler. (See Figure 6-10.)

Calling Sequence

```
SA1          File-name
RJ           D.WBA
 ┌─    JP    Normal Return
 │  ┌─ JP    Invalid Key
 │  │ ┌Move occurs key code┐
 │  │ │                    │
 │  │ │                    │
 │  │ │                    │
 │  │ └JP    D.MOCKW      ─┘
 │  └▷┌Invalid Key code   ┐
 │    └                  ─┘
 └▷Normal code
```

This code is generated only if the file contains variable-length records (i.e., records are described with a record contains clause or an occurs clause with a depending on data name.

Routines Called

CPC, DDMOVIO, IOWRITE

Register Usage

Register B1 preserved.

Operation

DDWBA behaves exactly like the "Write Record Name" if the file is not BCD or if the file is random. However, the first six bits will be altered in each record. Its function is the same as DDWRITE with the additional task of line spacing control.

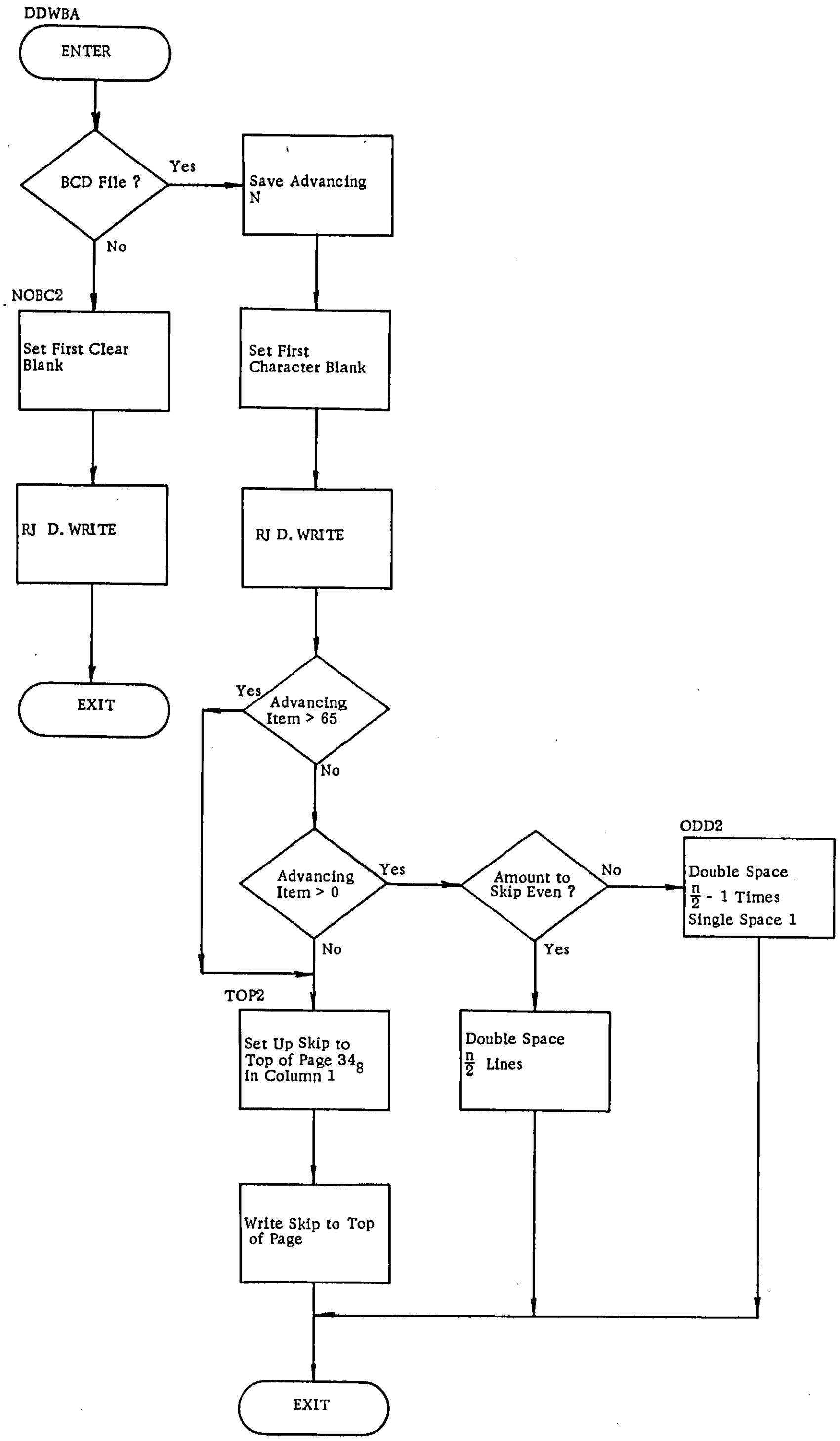


Figure 6-10. DDWBA Flowchart

DDWAA - WRITE RECORD-NAME AFTER ADVANCING SUBROUTINE

Purpose

This subroutine is called when the COBOL statement, "Write Record Name After Advancing $\left\{\frac{n}{\text{Data Name}}\right\}$ Lines" is encountered by the compiler. (See Figure 6-11.)

Calling Sequence

```
SA1            File-name
RJ             D.WAA
|--       JP   Normal Return
|   |--   JP   Invalid Key
|   |   [Move occurs key code ]
|   |   [                     ]
|   |   [                     ]
|   |   [JP    D.MOCKW        ]
|   |-->[Invalid Key code     ]
|-->Normal code
```

{ This code is generated only if the file contains variable-length records (i.e., records are described with a record contains clause or an occurs clause with a depending on data name). }

Routines Called

CPC, DDMOVIO, IOWRITE

Register Usage

Register B1 preserved.

Operation

DDWAA behaves exactly like the "Write Record Name" if the file is not BCD or if the file is random. However, the first six bits will be altered in each record. Its function is the same as DDWRITE with the additional task of line spacing control.

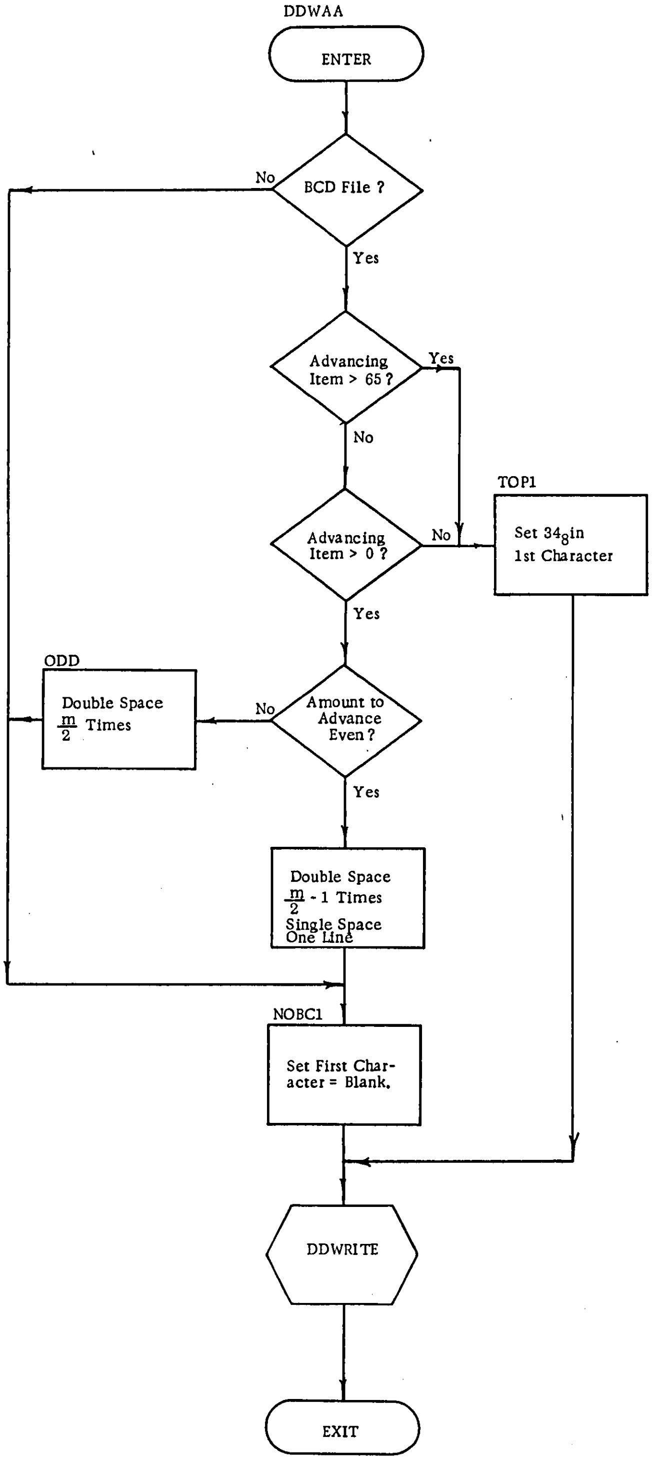


Figure 6-11. DDWAA Flowchart

DDWRNCH - WRITE N CHARACTERS SUBROUTINE

Purpose

This subroutine is called by DDWRITE when the writing of a continuous binary file is indicated. (See Figure 6-12.)

Calling Sequence

```
SX6         N (number of characters to write)
SA1         File-name
RJ          D.WRNCH
Normal return
```

Routines Called

CPC, DDMOVIO

Register Usage

Register B1 is preserved.

Operation

DDWRNCH transfers information from the user record area to the SCOPE CIO buffer in a characterwise fashion. If a write cannot be satisfied with the space remaining in the CIO area, the CIO area is filled and a SCOPE write with recall is issued. When control is again returned to DDWRNCH it attempts to transfer the remaining characters to the CIO area. It continues the process of writing with recall and transferring characters until the write is satisfied. After the write is satisfied, and if the file is not busy, a test is made to see if one or more PRVs are available for writing. If so a write is issued and then D.WRNCH exits to the calling program.

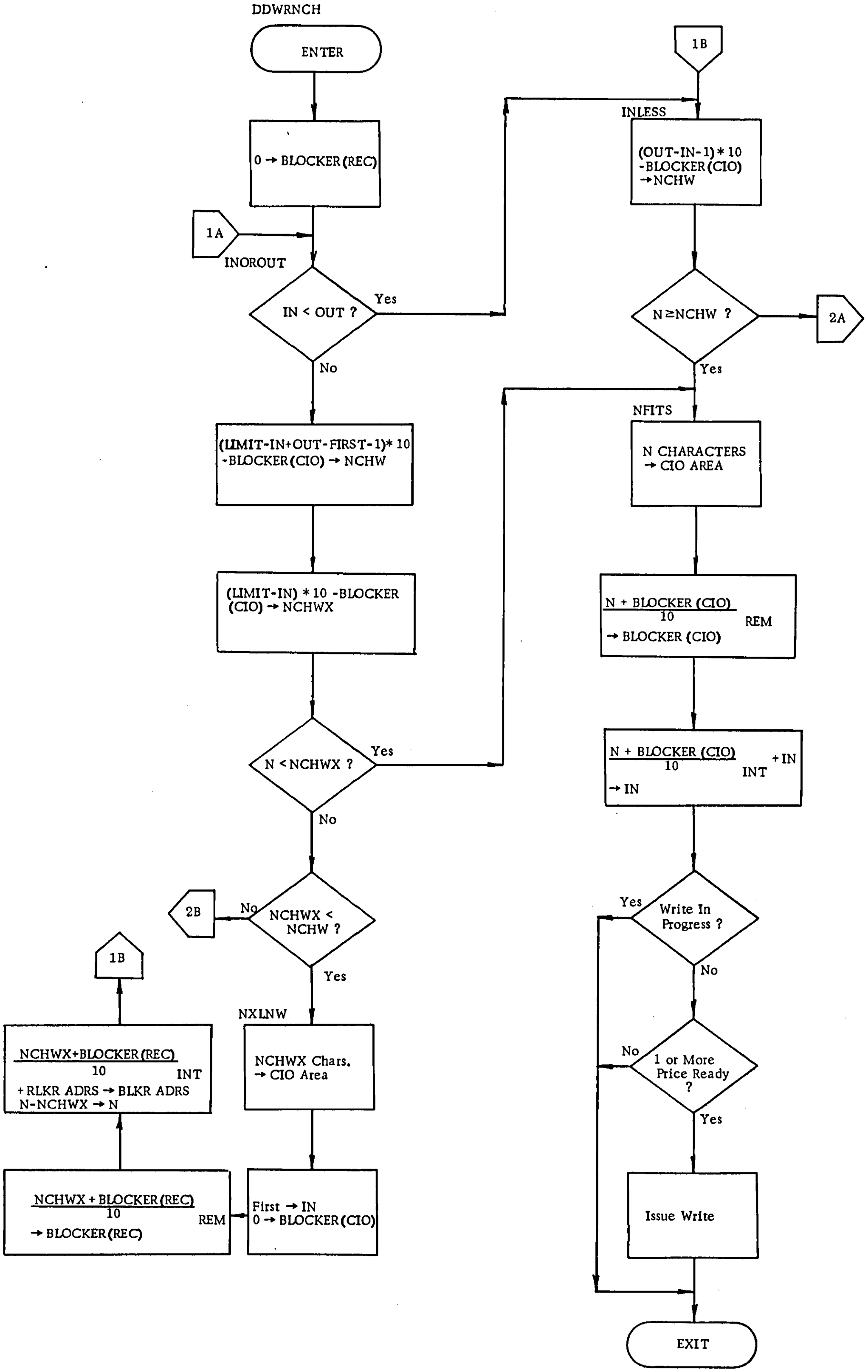


Figure 6-12. DDWRNCH Flowchart (1 of 2)

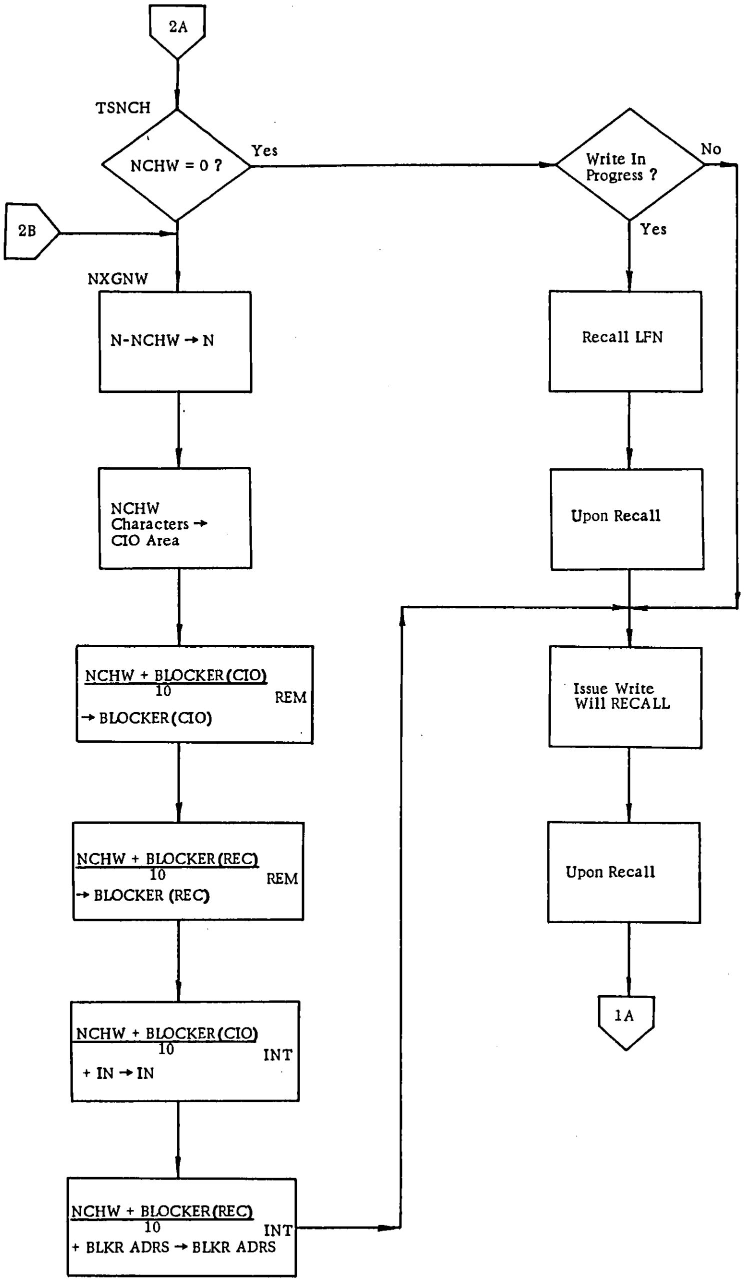


Figure 6-12. DDWRNCH Flowchart (2 of 2)

DDCLOS - CLOSE FILE-NAME SUBROUTINE

Purpose

DDCLOS is called when the COBOL statement CLOSE file-name is encountered by the compiler. (See Figure 6-13.)

Calling Sequence

```
SA1        File-name
SX6        0 = rewind
SX6        1 = no rewind
SX6        2 = lock
RJ         D. CLOS
Normal return
```

Routines Called

CPC

Register Usage

Register B1 preserved.

Operation

DDCLOS terminates the action on the file name. If there are any USE declaratives that are applicable, they will be executed at the appropriate time. A description of the USE tables associated with each File Environment Table (FET) is shown in Tables 6-1, 6-2, and 6-3.

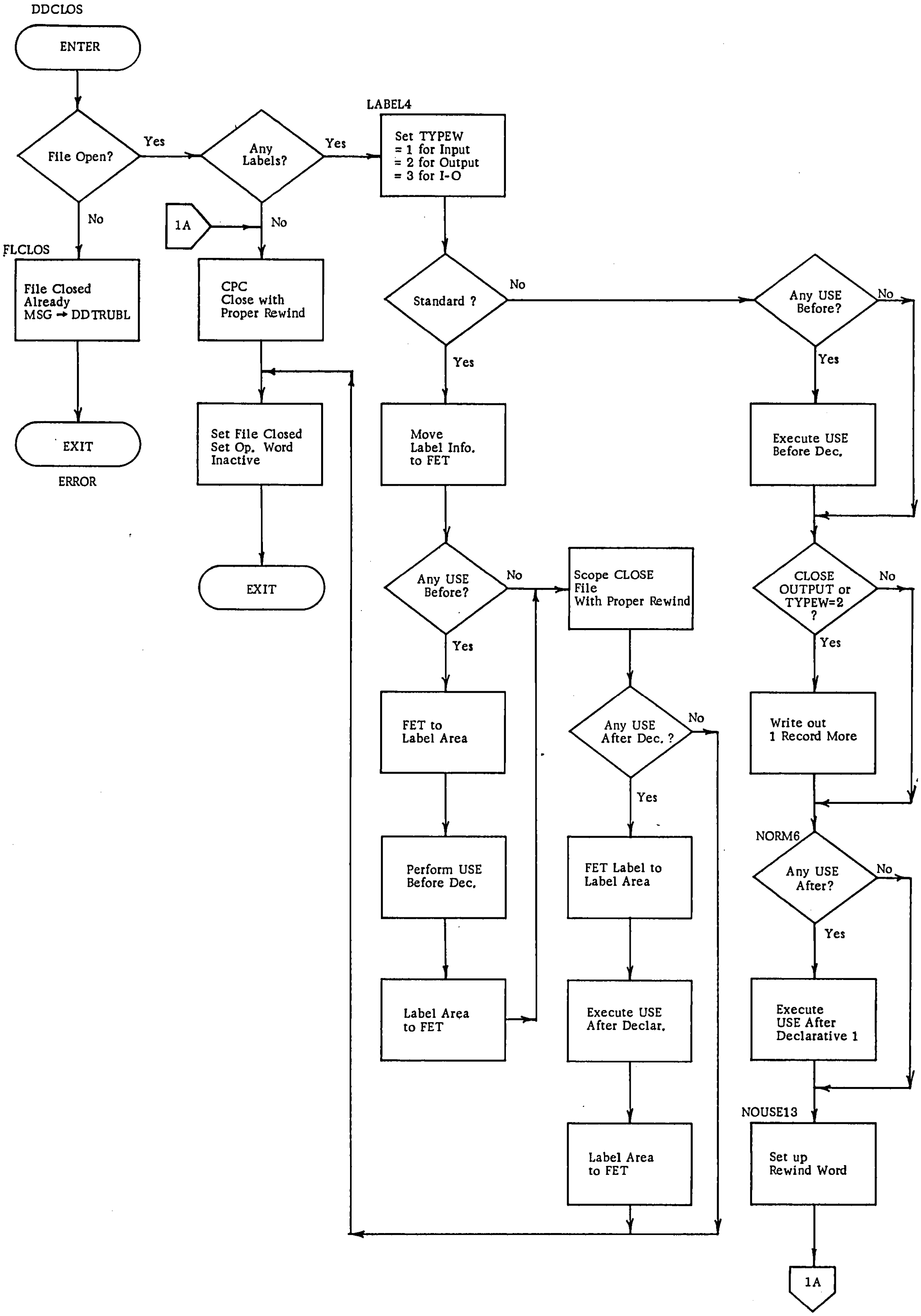


Figure 6-13. DDCLOS Flowchart

DDCRELR - CLOSE REEL-NAME SUBROUTINE

Purpose

This subroutine is called when the COBOL statement CLOSE REEL file-name is encountered by the compiler or when an end of reel is encountered during the processing of a tape file. (See Figure 6-14.)

Calling Sequence

The calling sequence to D. CRELR is given below:

```
SA1      File-name
SX6      0 = rewind
SX6      1 = no rewind
SX6      2 = lock
RJ       D.CRELR
Normal return
```

Routines Called

CPC

Register Usage

Register B1 preserved.

Operation

DDCRELR initiates reel closing procedures and new reels are opened.

Any USE declaratives that are applicable are executed at the appropriate time. A description of the USE tables associated with each File Environment Table (FET) is shown in Tables 6-1, 6-2, and 6-3.

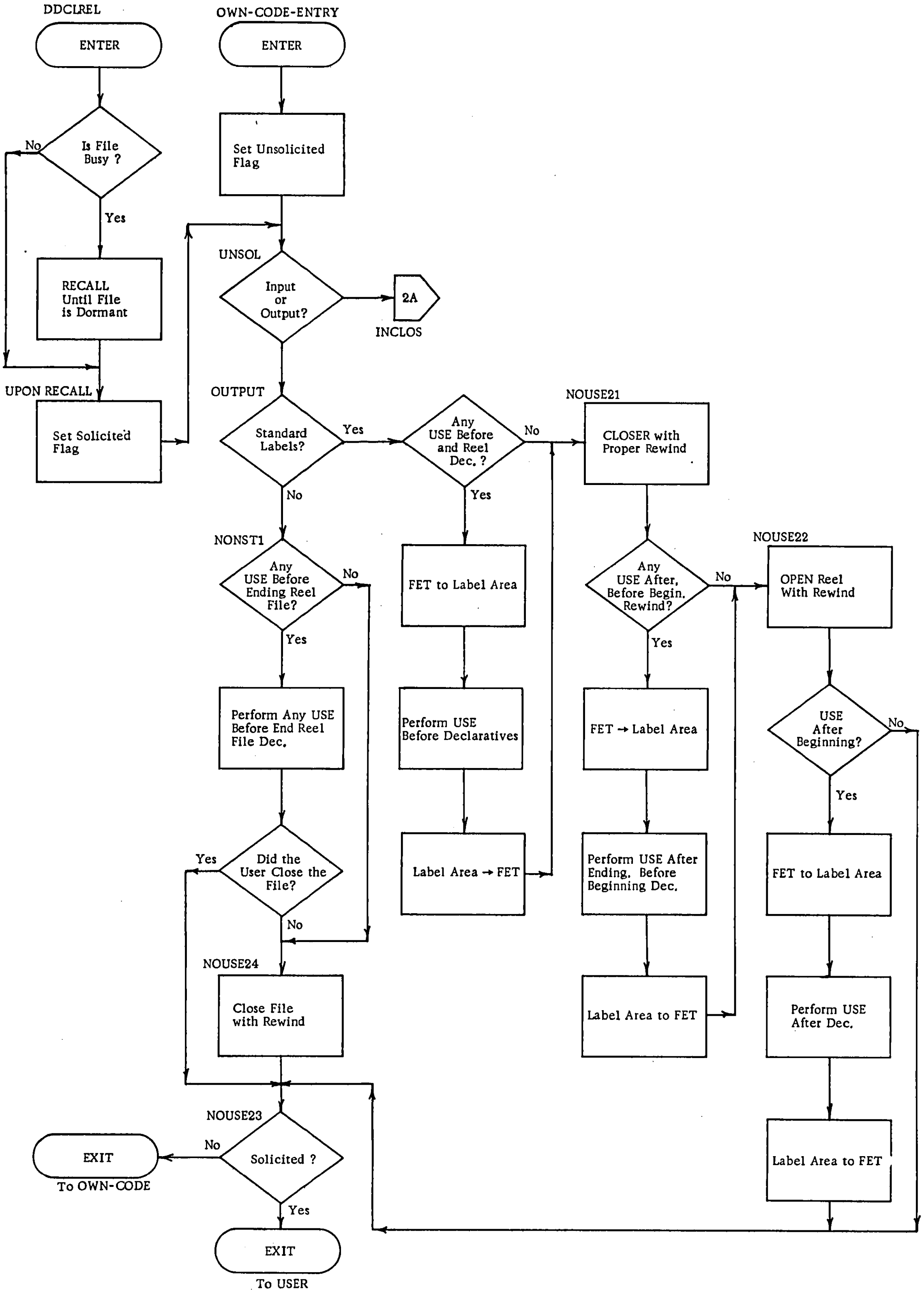


Figure 6-14. DDCLREL Flowchart (1 of 3)

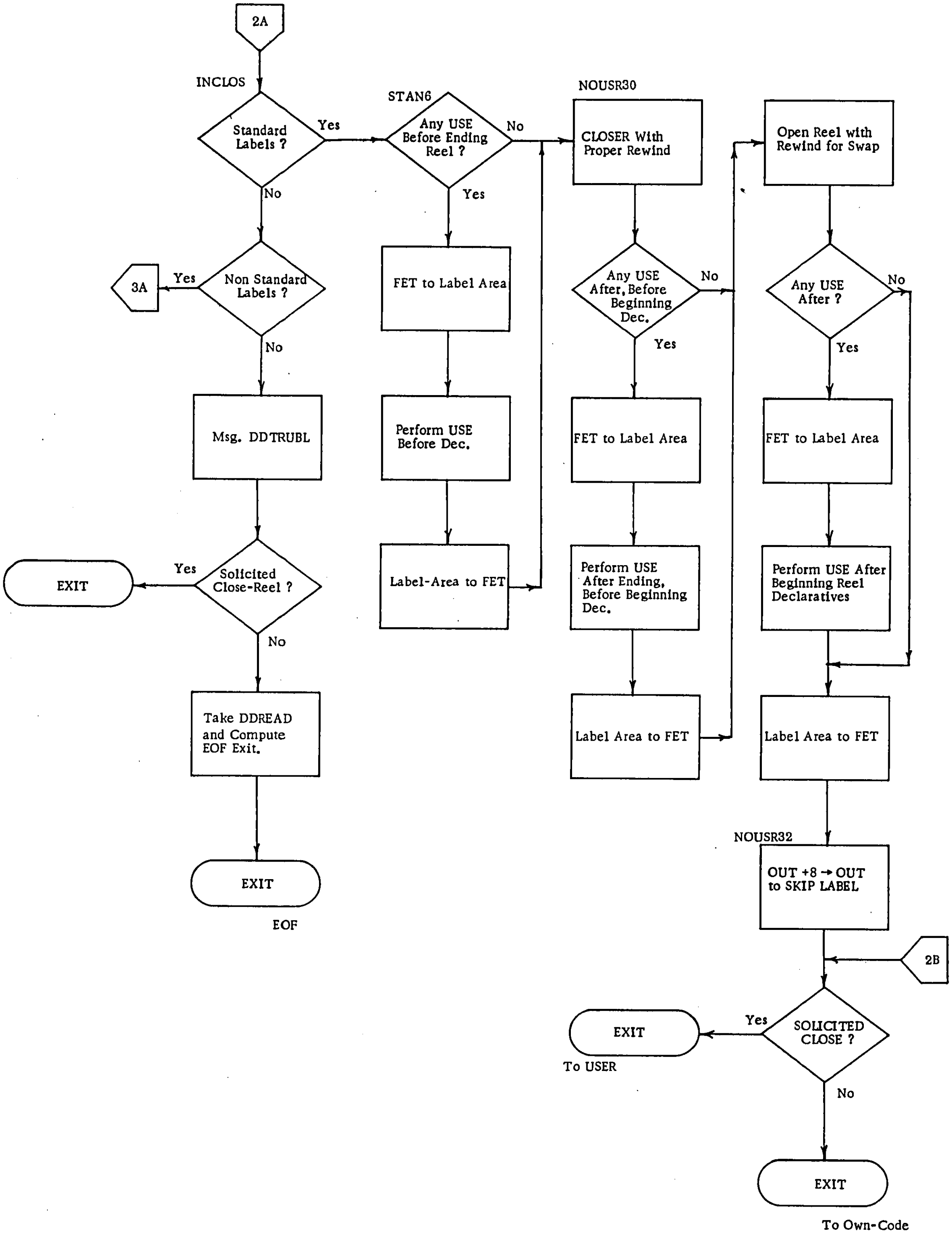


Figure 6-14. DDCLREL Flowchart (2 of 3)

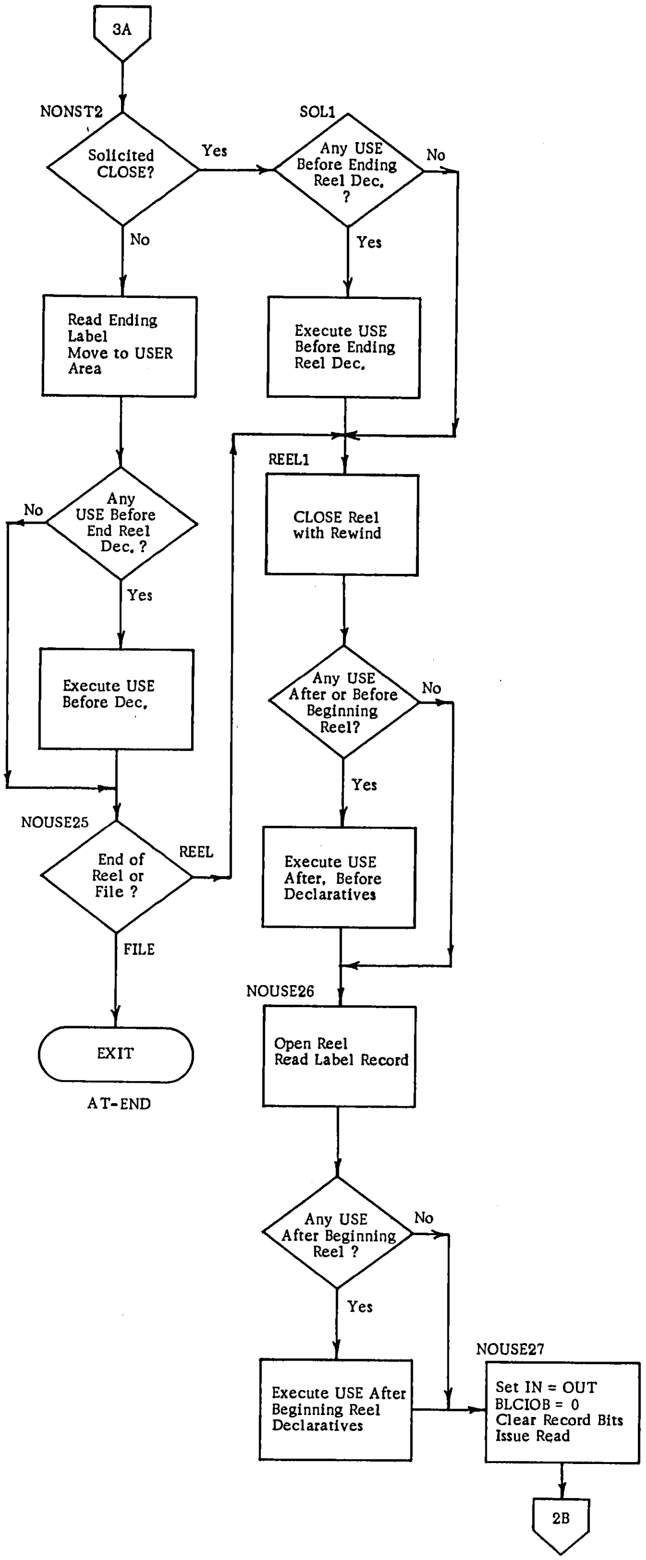


Figure 6-14. DDCLREL Flowchart (3 of 3)

DDDADD - DOUBLE-PRECISION DECIMAL ADDITION SUBROUTINE

Purpose

This routine adds two display code numbers in double-precision form, resulting in a double-precision display code. The form of number representation is 9s complement.

Interface

The interface is shown in Figure 6-15.

Restrictions

The numbers are assumed to be 9's complement for the full two registers, but have only 18 significant digits at the most. The two left digits in the first register consist of either nines (9's) or zeros (0's), depending upon the sign of the quantity being added.

Routines Called

None

Register Usage

All other registers volatile except X4. All A registers volatile except A0. No B registers used.

Operation

This routine makes use of the fact that the display code on the 6400 and 6600 gives a carry from each 6-bit digit when a CARRY should occur for the decimal addition. The CARRIES within the separate words are handled by a technique of extracting and adding bits at the rate of a whole word at a time. Since a CARRY from a high-order digit of one number should be added into the low-order digit of the other word (whereas it actually carries into the low-order digit of its own word), a test is made to find instances when a CARRY occurred from one word but not from the other. In this case the CARRY is corrected. One other situation arises: an addition of two words without any CARRIES can sometimes result in a word of all binary ones (1's); the hardware converts this immediately from binary minus zero to binary plus zero. This instance must be tested for and corrected if it occurs.

```
                        IDENT        DDDADD
000026      PROGRAM LENGTH

            BLOCKS
000026      PROGRAM*    LOCAL

            ENTRY POINTS
               000004 D.DADD
                        ENTRY       D.DADD
            * D.DDADD     DOUBLE PRECISION DISLAY CODE ADD
            *INPUT AND OUTPUT ARE NINES COMPLEMENT DOUBLE WORDS
            * INPUT IS IN X1,X2   AND    X6,X7
            *  OUTPUT IS IN X6,X7
            *
```

Figure 6-15. DDDADD Interface Printout

DDCVBD - BINARY TO DECIMAL CONVERSIONS SUBROUTINE

Purpose

This routine converts numbers from binary representation to decimal representation during execution of the object program.

Interface

The interface is shown in Figure 6-16.

Restrictions

All numbers coming into this routine as single-precision numbers are converted to the proper representation and are decimally left-truncated if they are larger than 10^{14}. If the size is larger than the indicated size, the ON-SIZE error flag is turned ON. For those double-precision numbers that are too large to convert by the technique used in this routine, the ON-SIZE error flag is turned ON and the routine exits without returning any reasonable information.

Elements Called

This routine makes use of the two external tables, DDTENS and DDTENDP.

Operation

This routine is divided into two routines, one of which, BINDEC1, converts single-precision numbers to either single- or double-precision output (since the decimal representation sometimes occupies two registers for a single register binary number). The other routine, BINDEC2 is used when the input consists of a double-precision binary number. BINDEC2 calls BINDEC1 to convert two halves of the number. The basic technique used in this conversion is:

1. To convert the binary number to a fraction by multiplying by a suitable negative power of ten.

2. Then multiplying by ten time and time again, each time shifting out another decimal digit to the left of the decimal place in order to make the answers always come out correctly.

The original input is rounded by the addition of one-half in the next place prior to the conversion.

```
                      IDENT      DDCVBD
 000106     PROGRAM LENGTH

            BLOCKS

 000104     PROGRAM*   LOCAL
 000002     LITERALS*  LOCAL

            ENTRY POINTS

               000046 D.CVBD3      000000 D.CVBD5      000052 D.CVBD6

            EXTERNAL SYMBOLS

            D.TENS     D.TENDP

                      ENTRY      D.CVBD3,D.CVBD5,D.CVBD6,D.CVBD7
                      EXT        D.TENS,D.TENDP
0000000 X    TENS     EQU        D.TENS
0000000 X    TENTHS   EQU        D.TENDP
            *         B I N D E C 2   DOUBLE PRECISION BINARY TO DECI
            *
            *         INPUT REGISTERS ARE
            *         B1         1
            *         B5         SIZE
            *         X4         DISPLAY ZEROES
            *         X6         HI ORDER INPUT
            *         X7         LO ORDER INPUT
            *
            *         OUTPUT REGISTERS ARE
            *
            *         A0         RESTORED
            *         B1         1
            *         B5         NONZERO INDICATES SIZE ERROR
            *         X4         DISPLAY ZEROES
            *         X6         HI ORDER OF DECIMAL OUTPUT
            *         X7         LO ORDER OF DECIMAL OUTPUT
            *         OUTPUT IS 9S COMPLEMENT DISPLAY CHARACTERS
            *
```

Figure 6-16. DDCVBD Interface Printout (1 of 2)

```
*         B I N D E C 1   SINGLE PRECISION BINARY TO SINGLE OR
*                         DOUBLE PRECISION DECIMAL.
*
*         INPUT REGISTERS
*
*         A0        (NOT USED IN ROUTINE)
*         B1        1
*         B5        SIZE
*         X0        (NOT USED IN ROUTINE)
*         X4        DECIMAL ZEROES
*         X6        INPUT
*         ALL OTHERS ARE ASSUMED VOLATILE
*
*         OUTPUT REGISTERS
*
*         B1        1
*         B5        NON ZERO INDICATES SIZE ERROR
*         X4        DECIMAL ZEROES
*         X6        OUTPUT IF SINGLE, HIGH OUTPUT IF DOUBLE
*         X7        LO OUTPUT IF DOUBLE
*         OUTPUT IS 9S COMPLEMENT DISPLAY CODE
*
*         B I N D E C N   ALTERNATE ENTRY
*                   SAME AS BINDEC1 EXCEPT INPUT IN X3
*
*         INTERNAL REGISTER USE
*
*         A0        (NOT USED)
*         B1        1
*         B2        NORMALIZING SHIFT,I
*         B3        SHIFT COUNT
*         B4        SECOND WORD COUNT
*         B5        SIGHS                 DECIMAL DIGITS/FIELD
*         B6        SIGHN                 SIGN INDICATOR, ZERO IS PLUS
*         B7        AYRER                 ON SIZE ERROR
*         X0        (NOT USED)
*         X2        1,SHIFTER,PRODUCT
*         X3        FACTOR SCALE, 10
*         X4        DECIMAL
*         X5        COMPARATOR,FACTOR
*         X6        ACCUMULATE RESULT
*         X7        LOW ORDER OF DOUBLE RESULT
```

Figure 6-16. DDCVBD Interface Printout (2 of 2)

DDEDIT - COBOL EDITING SUBROUTINE

Purpose

The purpose of this routine is to effect an edited move as described in COBOL language. It can be called into play by either a MOVE or by an arithmetic statement, the result of which is stored into a field which has editing characteristics.

Interface

The interface is shown in Figure 6-17.

Restrictions

This routine expects to have legal input. That is, the input is to be in the format expected by the routine. There are no error checks and the usual result of having illegal input is that the output is also garbled. Special actions to be carried out when the field is numerically zero are also part of this routine. When a field is to be blanked, the entire receiving field is replaced by blank. When the field is to be filled with the check-protect symbol, the entire output field is filled with the check protect, except that a decimal point is inserted in the check-protect symbols at a distance from the right end of the field to indicate the number of numeric positions past the decimal point if a decimal point was indicated in the receiving field.

Routines Called

None

Register Usage

See Table 6-4.

Operation

This routine expects its input to be in numeric form with 9's complement representation of negative numbers in one or two registers. The picture of the receiving field (which determines the editing) is contained in a coded form called a mural. (See Table 6-4.)

One section of this program initializes it with a special character for a currency sign or interchanges the dot and comma. Another special section of the program takes care of the situation where the input is zero and a special action has been requested. The rest of the program consists of a section that initializes the registers and a loop. The main feature of the loop is an indexed jump, depending upon the current character in the mural, (indicating the picture character in the original picture). In order to take care of floating dollar signs, the suppression of zeros and other features, characters are

```
                    IDENT     DDEDIT
000262    PROGRAM LENGTH

          BLOCKS

000261    PROGRAM*   LOCAL
000001    LITERALS*  LOCAL

          ENTRY POINTS

            000000 D.EDIT3       000003 D.EDIT5       000006 D.EDIT6
            000015 D.EDITI

          EXTERNAL SYMBOLS

          D.TRUBL

          *         C O B O L   E D I T I N G   R O U T I N E
          *
          *         INPUT PARAMETERS ARE
          *
          *         B4  BYTES/FIELD (EXCLUDING ALL INSERTIONS)
          *         B5  BASE ADDRESS OF FIELD
          *         B6  BASE ADDRESS OF MURAL
          *         B7  BYTES/FIELD (INCLUDING INSERTIONS), ZERO=NOT
          *             IF ASTERISKS REPLACE TOTAL ZERO, -N2 IN THE F
          *               +N1 IN THE LAST 6 BITS WHERE N1= CHARACTERS
          *                LEFT OF THE POINT. N2= CHARACTERS TO THE R
          *                THE POINT PLUS ONE FOR THE POINT ITSELF IF
          *                E.G. **** GIVES 777704,  **.** GIVES 7774
          *
          *         X2  OFFSET OF FIELD IN BYTES
          *         X6  HI ORDER OF NUMERIC INPUT  (9*S COMPLEMENT
          *         X7  LO ORDER OF NUMERIC INPUT     DISPLAY CODE)
          *
          *         RESTORED REGISTERS ARE
          *
          *         A0
          *         B1  1
          *         X4  ALL OCTAL 3*S
```

Figure 6-17. DDEDIT Interface Printout (1 of 2)

```
       *
       *
       *               INTERNAL REGISTER USAGE
       *               B1  1
       *               B2  CHARACTERS/WORD AS ACCUMULATED IN X6
       *               B3  DELAYED CHARACTER
       *               B4  CURRENT CHARACTER
       *               B5  TWO FILLER CHARACTERS (-1 MEANS NO FLOAT YET)
       *               B6. SIGNIFICANCE INDICATOR (1=NO FLOAT YET, 0=FLO
       *               B7  MURCODE (CURRENT 4 BIT CODE FOR EDIT OF ONE C
       *               X0  6 BIT MASK, RIGHT ADJUSTED
       *               X1  4 BIT MASK, RIGHT ADJUSTED
       *               X2  MURAL CODES (ONE WORD, CURRENT CODE ROTATED T
       *               X3  HI ORDER OF NUMERIC INPUT (UNCOMPLEMENTED, SH
       *               X4  LO ORDER OF NUMERIC INPUT (UNCOMPLEMENTED, SH
       *               X5  VOLATILE
       *               X6. CURRENT FIELD WORD
       *               X7  VOLATILE
*
*    M U R A L   C O D E S
*   00  01  02  03  04  05  06  07  10  11  12  13  14  15  16  17
*   END Z   *   R   9   ,   $   B   0   .   -   +   CR  DB  WRD NIL
*
*
*      ENTRY AT D.EDITI INITIALIZES FOR CURRENCY SIGN AND RADIX PO
*      AT THIS ENTRY B2 CONTAINS CURRENCY SIGN-RADIX PT=COMMA,
*
       EXT         D.TRUBL
       ENTRY       D.EDIT3,D.EDIT5,D.EDIT6,D.EDIT7,D.EDITI
```

Figure 6-17. DDEDIT Interface Printout (2 of 2)

Table 6-4. Murcode Processor Input/Output

Input						Output			
NEXT	B4	B5	B6	MURCODE	B3	B4	B5	B6	X4
⑨			-1	A, 9, X		⑨			
⑨		ab	1,⓪	A, 9, X	a	⑨		- 1	
⑨		- 1	1,⓪	A, 9, X		⑨		- 1	
				Ⓑ		^			
				$		^	^,$		
	0			-		^	^Δ		
	0			+		^	^±		
⓪		- 1	- 1	Z		^	^^	⓪	
⓪		- 1	1	*		*	**	0	
①		- 1	1	Z		①	^^	- 1	
①		- 1	1	*		①	**	- 1	
⓪		ab	1	Z	a	a	^^	⓪	
⓪		ab	1	*	b	*	**	⓪	
⓪		ab	1	r		a		⓪	
①		ab	1	Z	a	①	^^	-1	
①		ab	1	*		①	**	-1	

Input						Output				
OTHER	NEXT	B4	B5	B6	MURCODE	B3	B4	B5	B6	X4
B3 ≠ 0	①		ab	1	r	a	①		- 1	
B3 = 0	①			1	r		①		- 1	
				⓪			^			
	⓪		ab	⓪	r, Z, *		a			
			ab	⓪		b	.		- 1	
			ab	⓪	0	b	⓪		- 1	
	①		ab	⓪	r, Z, *	b	①		- 1	
				≠⓪	.		.			
				≠⓪	0		⓪			
				≠⓪	,		,			
	⑨			- 1	Z, *		⑨			
		≠0			-		Δ			
		≠0			+		±			
X4 ≠ 0					CR	^Ⓒ	^Ⓡ	6	0	
X4 ≠ 0					DB	^Ⓓ	^Ⓑ	18	0	
X4 = 0					CR	^Ⓡ		6		
X4 = 0					DB	^Ⓑ				

LEDGEND

NEXT = next character of source (if blank, not fetched)
B3 = delayed character
B4 = current character
B5 = Filler-blanker characters
B6 = Significance Indicator

⓪ = display "0"
① = display "1" thru "9"
⑨ = display "0" thru "9"
ab = two display characters
r = repeated +, -, $

Initial Condition is :
B4 = 0 X4 = nonzero
B5 = - 1
B6 = 1

Significance Indicator (B6)
1 initialized
⓪ suppressing zeroes
-1 suppression terminated

Blank for Input - Immaterial

Blank for Output - Unchanged

accumulated with a delay of one character--one character being kept aside before it is placed in the receiving register. In order to describe the action during the main cycle of the operation of the EDIT routine, looping around for each character in the mural, a decision table called the Murcode table is appended. The Murcode table indicates the situation in registers prior to processing each Murcode and the situation in the registers following processing of the Murcodes.

DDEXAMO - EXAMINE SUBROUTINE

Purpose

DDEXAMO is a subroutine that handles the ten different varieties of the EXAMINE verb. The EXAMINE verb is used to replace a character(s) in a data item and/or to count the number of times a character appears in the item. (See Figures 6-18 through 6-27.)

The types of EXAMINE statements are listed below:

Type	
1	Examine identifier falling all literal-1
2	Examine identifier falling leading literal-1
3	Examine identifier falling until first literal-1
4	Examine identifier falling all literal-1 replacing by literal-2
5	Examine identifier falling leading literal-1 replacing by literal-2
6	Examine identifier falling until first literal-1 replacing by literal-2
7	Examine identifier replacing all literal-1 by literal-2
8	Examine identifier replacing leading literal-1 by literal-2
9	Examine identifier replacing first literal-1 by literal-2
10	Examine identifier replacing until literal-1 by literal-2

Calling Sequence

X7	Zero if the identifier is unsigned
X7	Nonzero if the identifier is signed
X3	The floating-point unnormalized offset
X1	Literal-2 (if needed) right-adjusted zero filled (binary)
B2	Length of the identifier in characters
B3	Identifier 1, right-adjusted (binary zero filled)
B4	Base address of the identifier
B5	Type of Examine (binary right-adjusted)

Routines Called

None

Register Usage

Register B1 preserved.

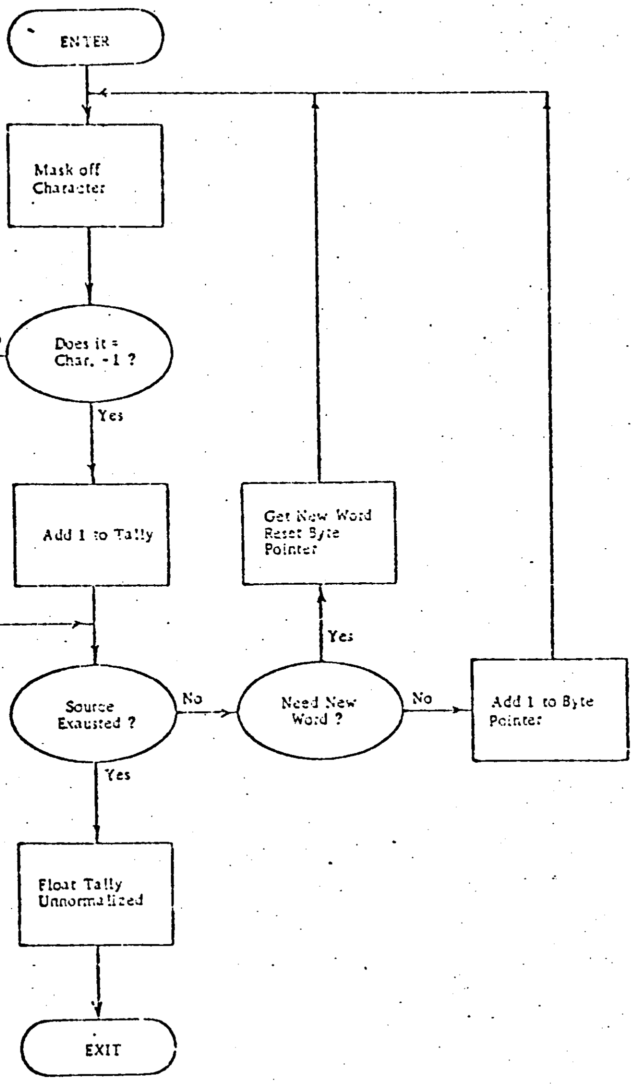


Figure 6-18. Examine Tallying All CHAR-1 Flowchart

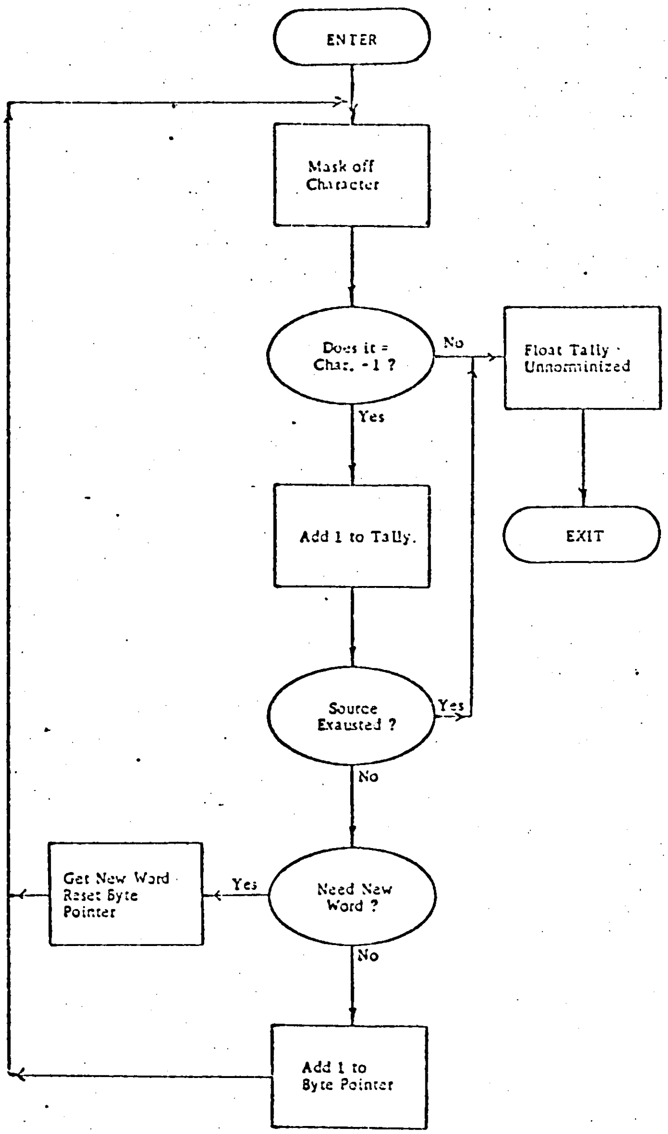


Figure 6-19. Examine Tallying Leading CHAR-1 Flowchart

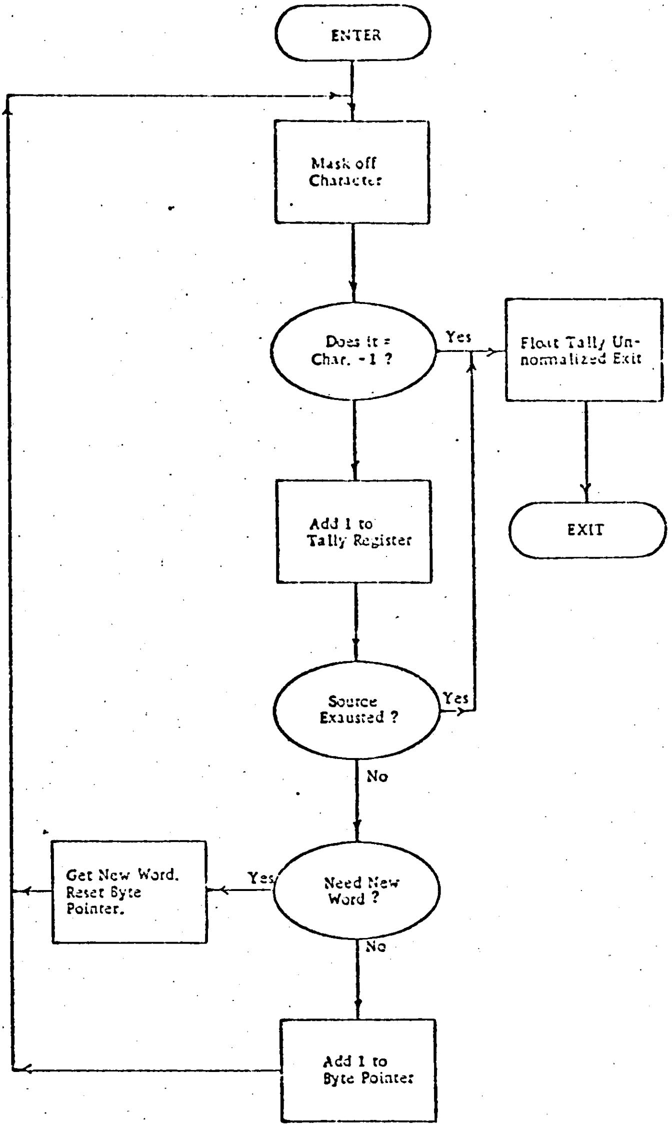


Figure 6-20. Examine Tallying Until First CHAR-1 Flowchart

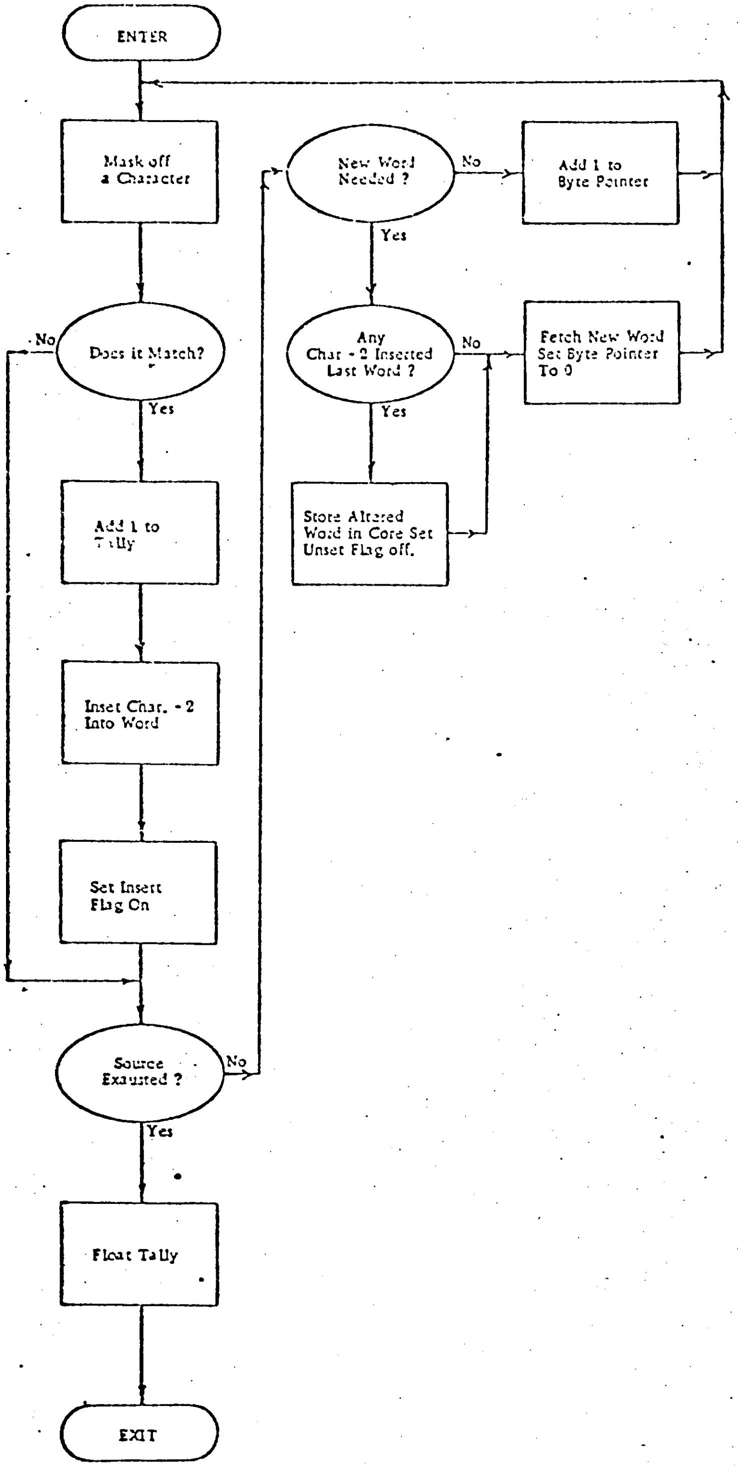


Figure 6-21. Examine Tallying All CHAR-1
Replacing by CHAR-2 Flowchart

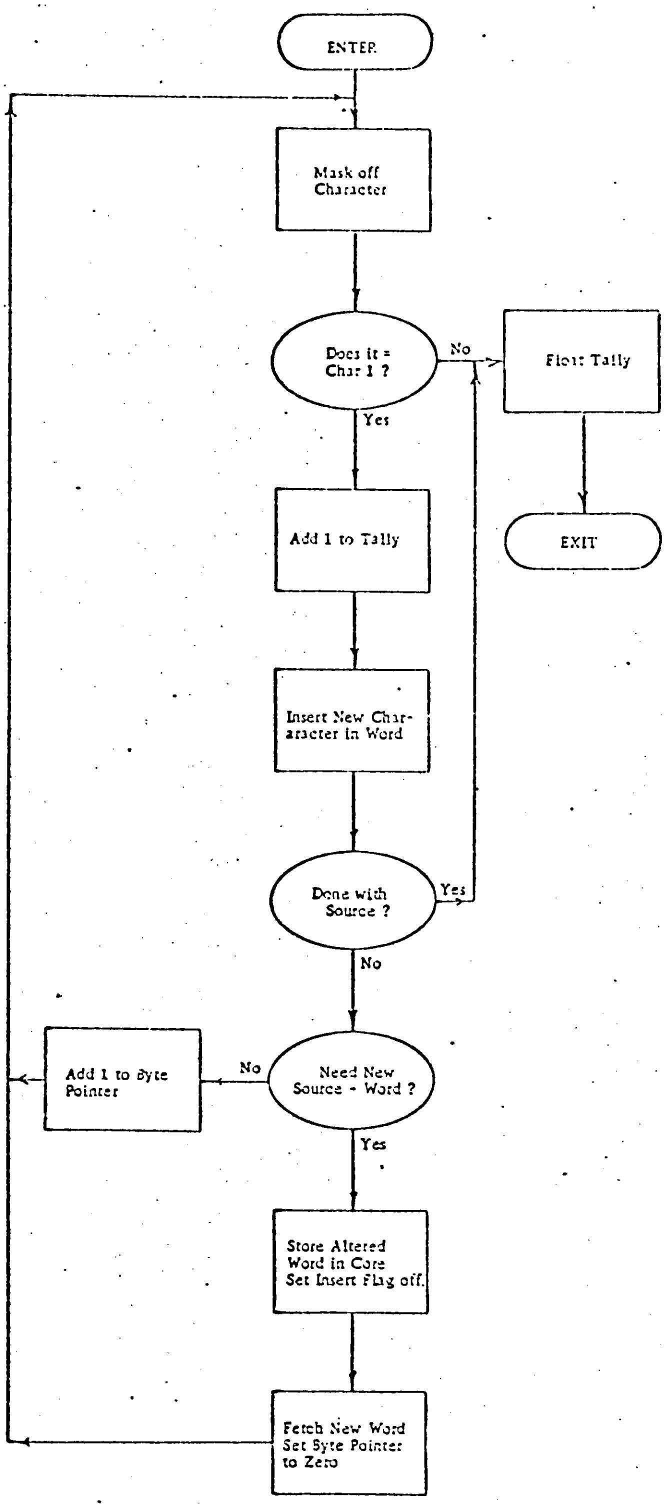


Figure 6-22. Examine Tallying Leading CHAR-1 Replacing by CHAR-2 Flowchart

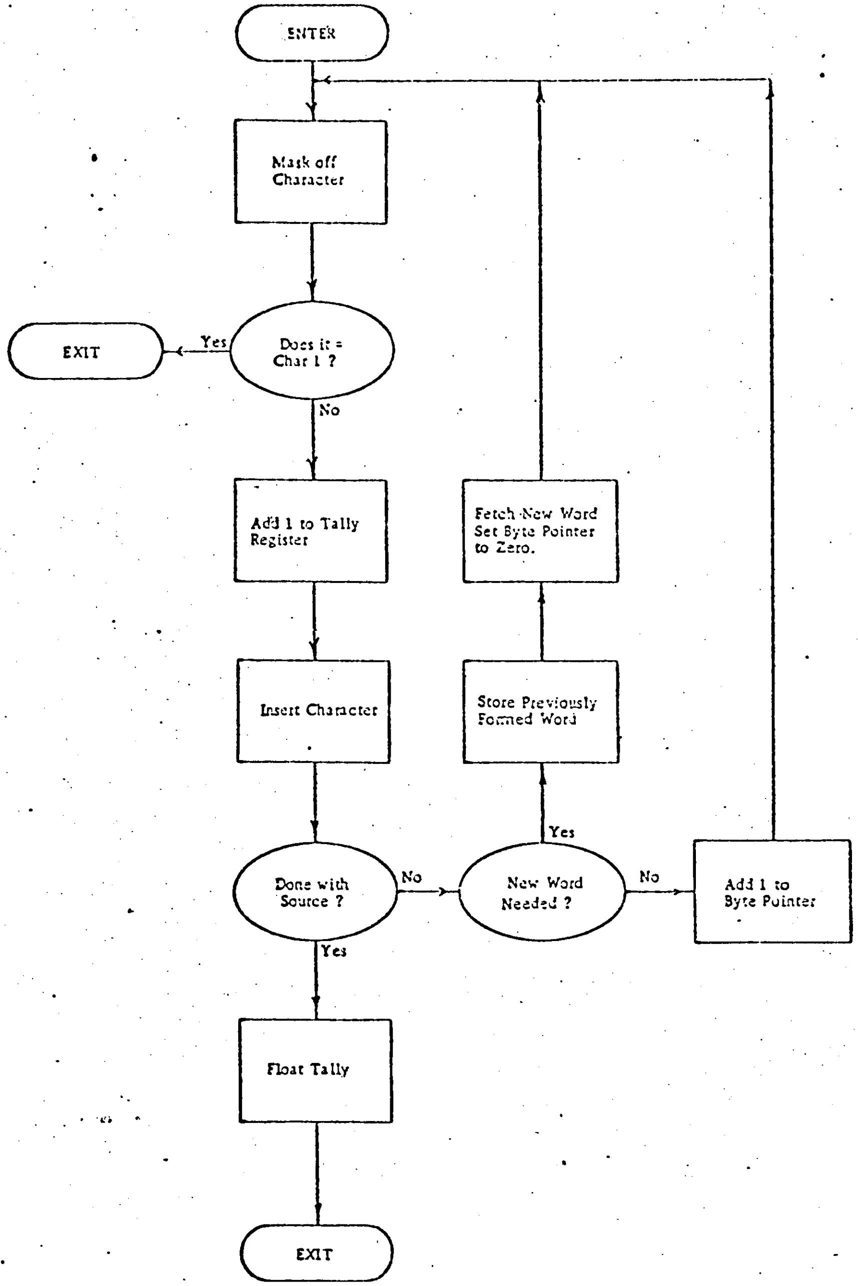


Figure 6-23. Examine Tallying Until First CHAR-1 Replacing by CHAR-2 Flowchart

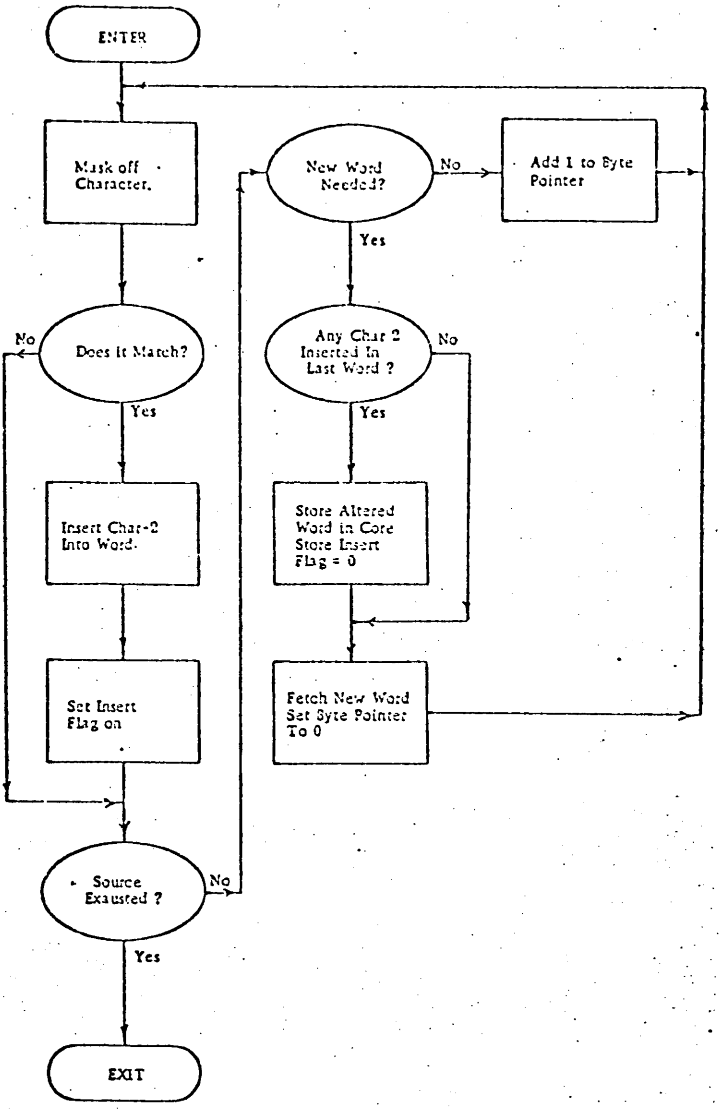


Figure 6-24. Examine Replacing All CHAR-1 by CHAR-2 Flowchart

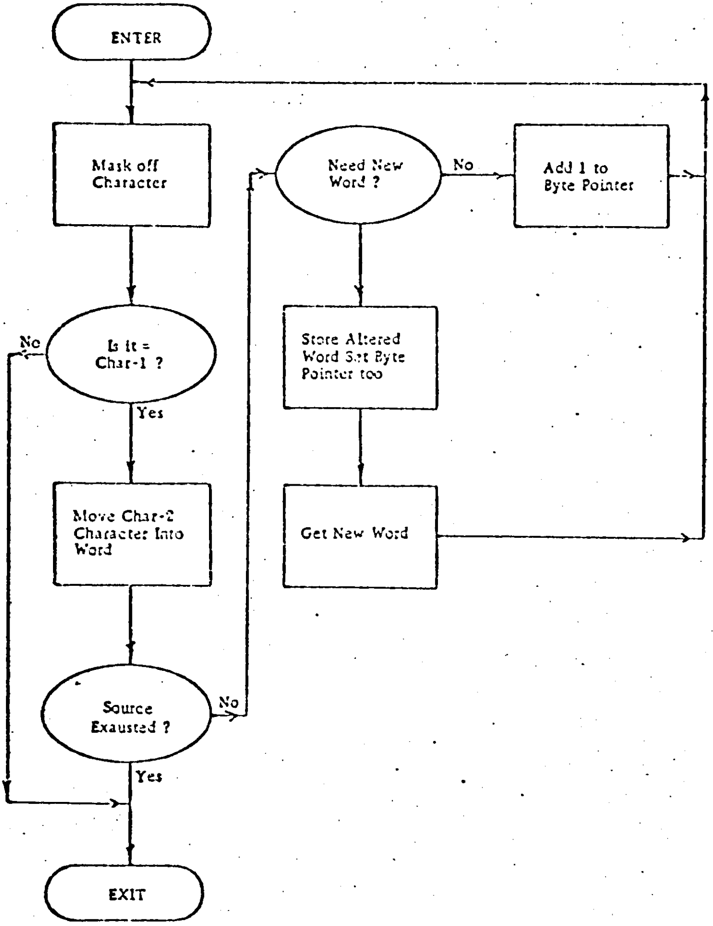


Figure 6-25. Examine Replacing Leading CHAR-1 by CHAR-2 Flowchart

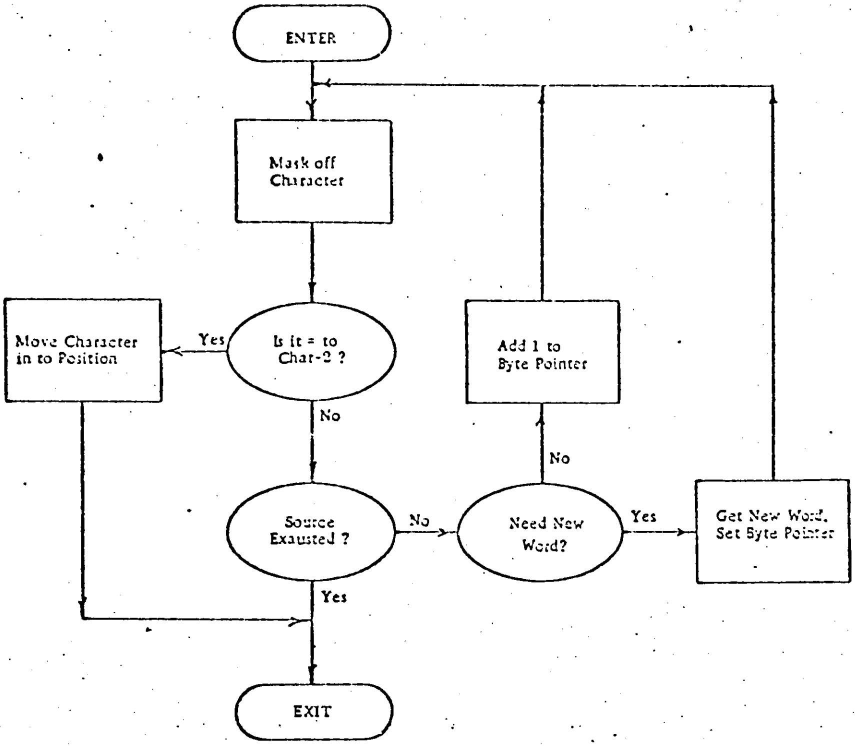


Figure 6-26. Examine Replacing First CHAR-1 by CHAR-2 Flowchart

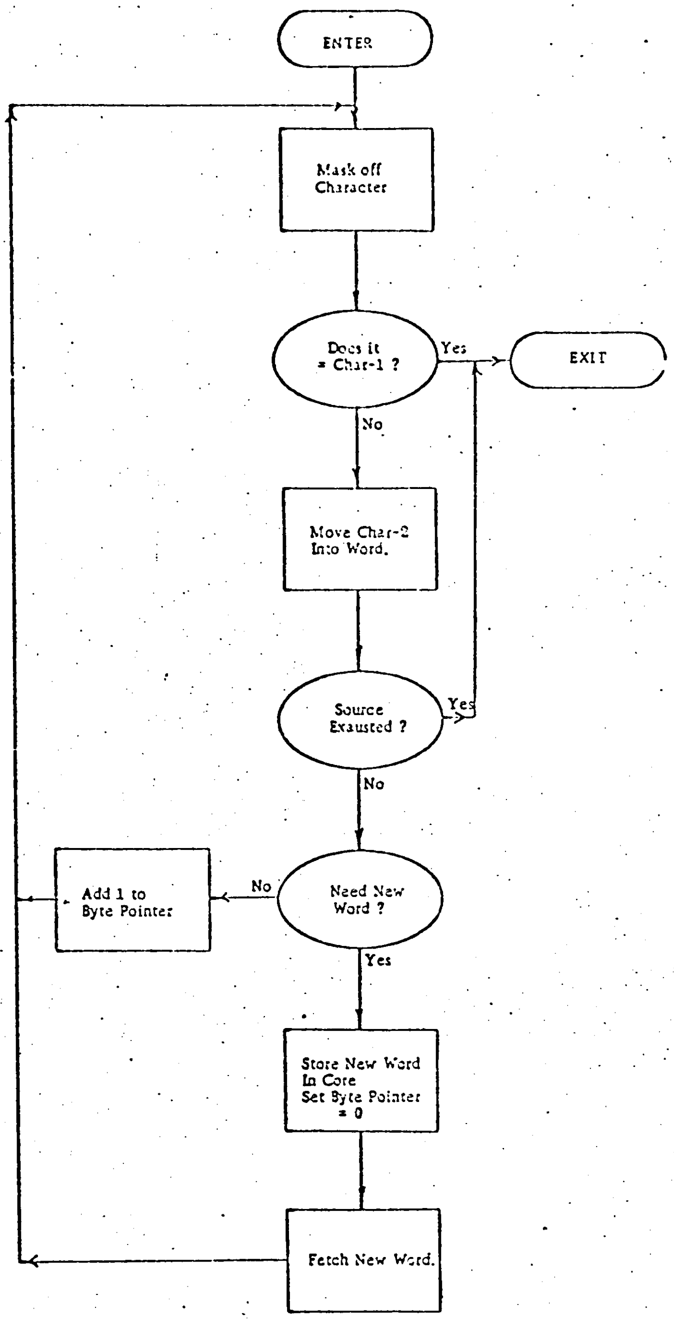


Figure 6-27. Examine Replacing Until CHAR-1 by CHAR-2 Flowchart

DDFINIS - TERMINATE ALL ACTION FOR INPUT/OUTPUT AT OBJECT TIME SUBROUTINE

Purpose

This routine has the purpose of containing any necessary terminating code. In Release 1 it jumps to FLUSH to write out the information that is retained in the buffers. It is mainly provided as a convenient means of making possible future changes. (See Figure 6-28.)

```
                    IDENT     FINIS
000003    PROGRAM LENGTH

          BLOCKS

000003    PROGRAM*   LOCAL

          ENTRY POINTS

             000000 D.FINIS

          EXTERNAL SYMBOLS

          FLUSH

                    ENTRY     D.FINIS
          **********SUBROUTINE FINIS
          *
          *         CLOSE ALL FILES AND FLUSH FOR POSSIBLE SNAPS
          *         ACCOMPLISH ALL REQUIRED END-OF-EXECUTION TASKS
          *
          ******
```

Figure 6-28. DDFINIS Interface Printout

DDFIVES - MISCELLANEOUS OBJECT TIME CONSTANTS SUBROUTINE

Purpose

The purpose of this routine is to provide constants for rounding purposes and other miscellaneous constants and temporary storage to be used in the object time.

Interface

There is no interface since this routine is not executed.

DDMOVIO - I/O MOVE SUBROUTINE

Purpose

DDMOVIO provides character moves. (See Figure 6-29.)

Calling Sequence

```
Load B2 with source byte address (0-9)
Load B3 with sink byte address (0-9)
Load B4 with the byte length of the move
Load A5 with source word address
Load A4 with sink word address
RJ      D.MOVIO
```

Routines Called

None

Register Usage

Register B1 is preserved. (See Figure 6-30.)

Operation

Its principal use is the transmission of data items between the CIO buffer and the user record area.

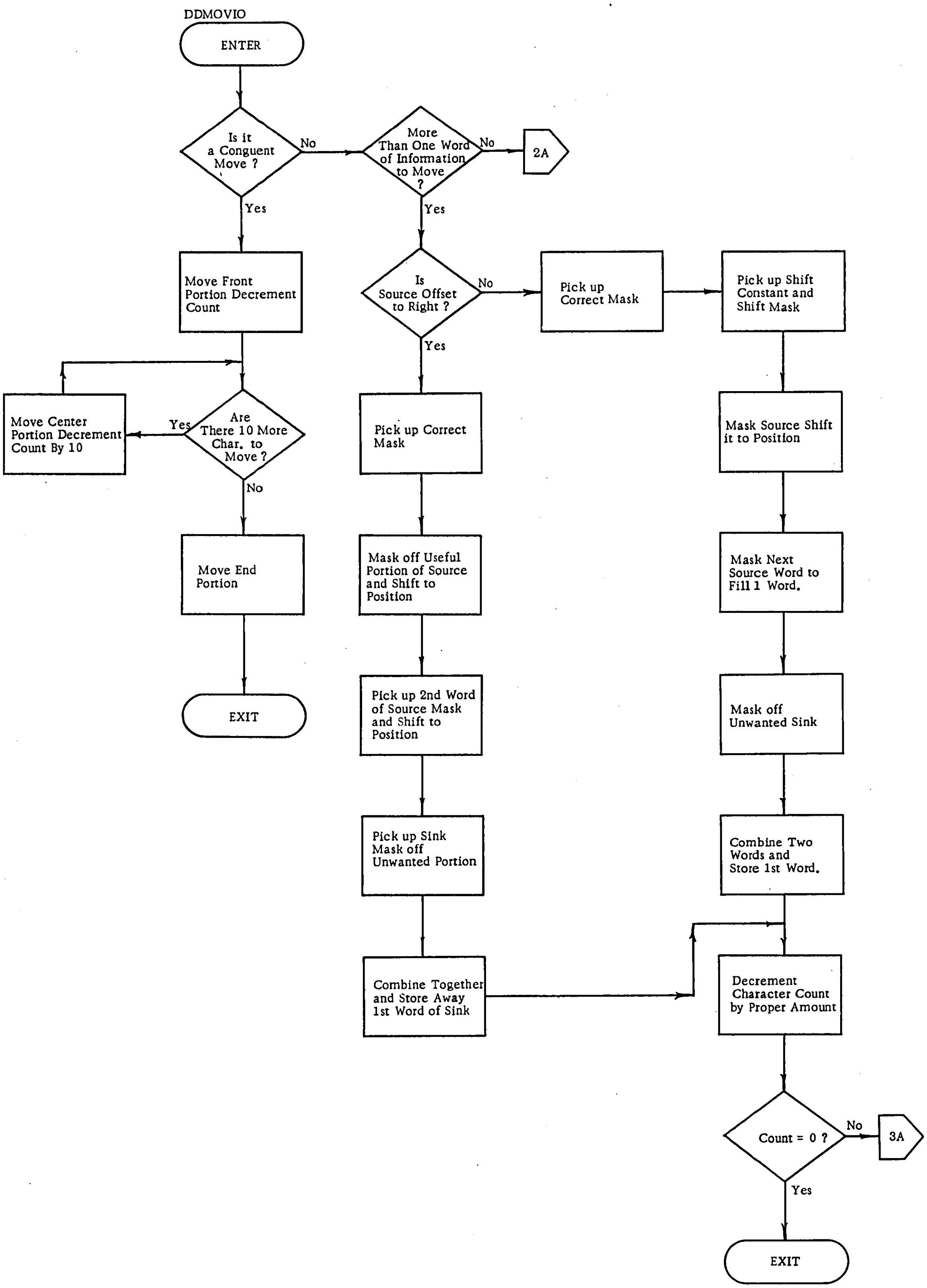


Figure 6-29. DDMOVIO Flowchart (1 of 3)

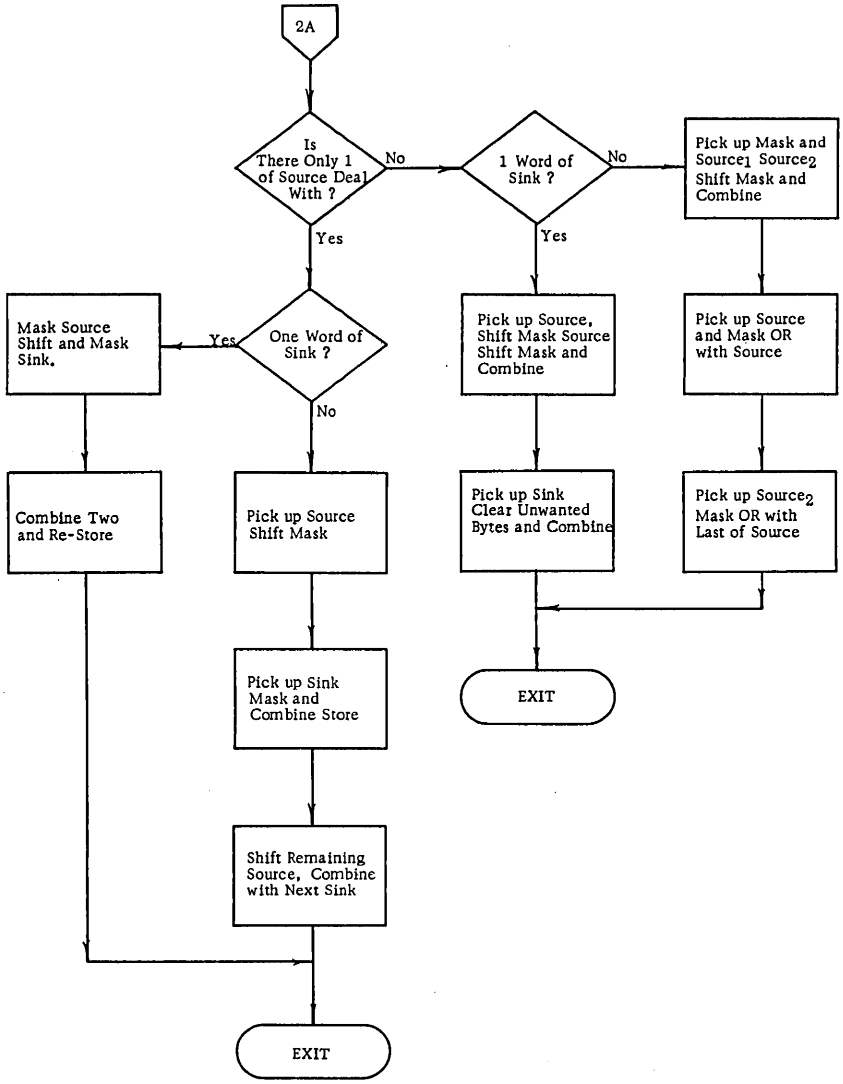


Figure 6-29. DDMOVIO Flowchart (2 of 3)

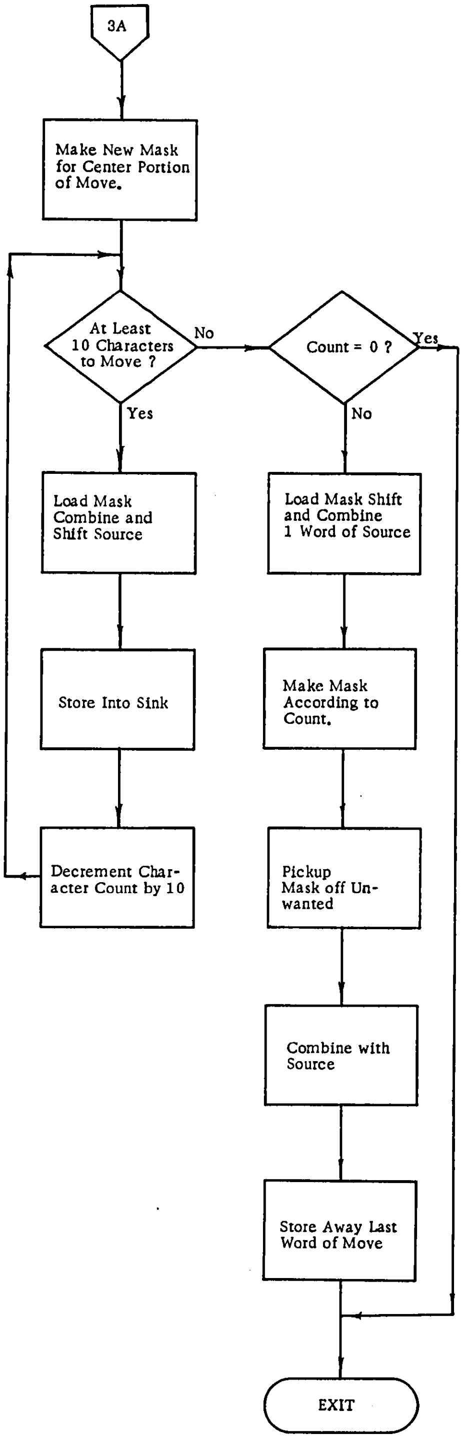


Figure 6-29. DDMOVIO Flowchart (3 of 3)

```
                     IDENT     DDMOVIO
 000154    PROGRAM LENGTH

           BLOCKS
 000154    PROGRAM*   LOCAL

           ENTRY POINTS

              000000 D,MOVIO

                     ENTRY     D.MOVIO
00000000    MOVEIO   DATA      0                      , I=0 MOVE ROUTINE
           *                                            B2=SOURCE BYTE AD
           *                                            B3=SINK BYTE ADD.
           *                                            B4=BYTE LENGTH AD
           *                                            A5=SOURCE ADDRESS
           *                                            A6=SINK ADDRESS.
           *                                            X1=MASK IN USE.
           *                                            X2=TEMP REG.
           *                                            X3=TEMP REG.
           *                                            B5=SHIFT REG.
           *                                            B6=TEMP REG,
           *                                            B0=0
           *                                            B1=1
```

Figure 6-30. DDMOVIO Interface Printout

DDSOL - SEGMENT OVERLAY SUBROUTINE

Purpose

DDSOL loads the overlays of the COBOL object program and accomplishes the necessary linking when this is done. (See Figure 6-31.)

Interface

The interface is shown in Figure 6-32.

Restrictions

The number of nested PERFORMS that can occur at any one time is limited to seven. In the event that there are too many PERFORMs called into play, this program displays a message and aborts.

Elements Called

The program calls the basic machine loader and by using it calls the different overlays into operation.

Calling Sequences

SOL	X1 contains an index. Control is transferred to the point represented by the index in X1 after the overlay is located or loaded.
SOL P	X1 contains "index" of the entry point X2 contains "index" of the drop-through location A2 contains the location to store return index X3 contains the return index (flagged) SOL stores the return index using A2, puts the drop-through index in its pushdown list, and then proceeds as for SOL entry.
SOL E	X1 contains an index, possibly flagged A1 contains location to store drop-through index. If X1 is flagged, the drop-through index is retrieved from the pushdown list and stored using A1. SOL then proceeds as for SOL entry.

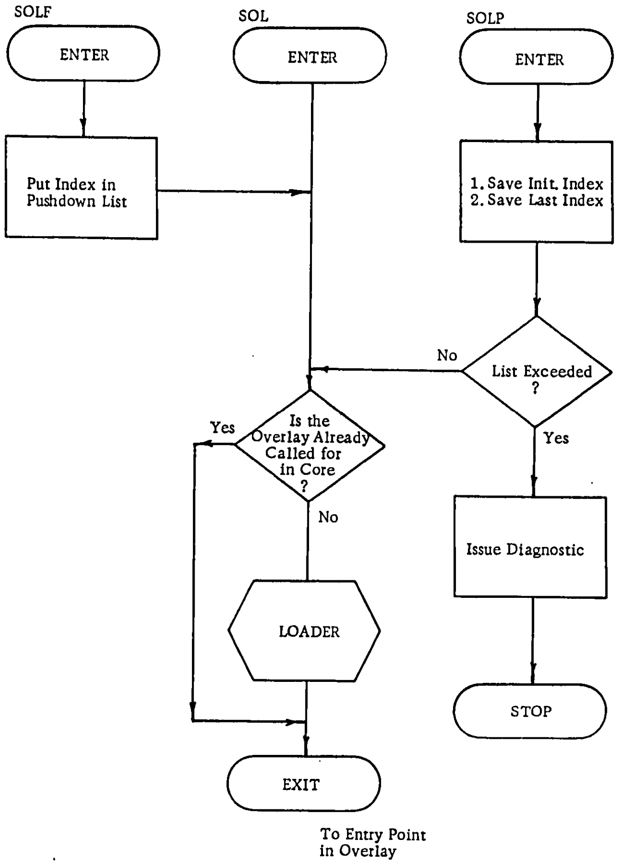
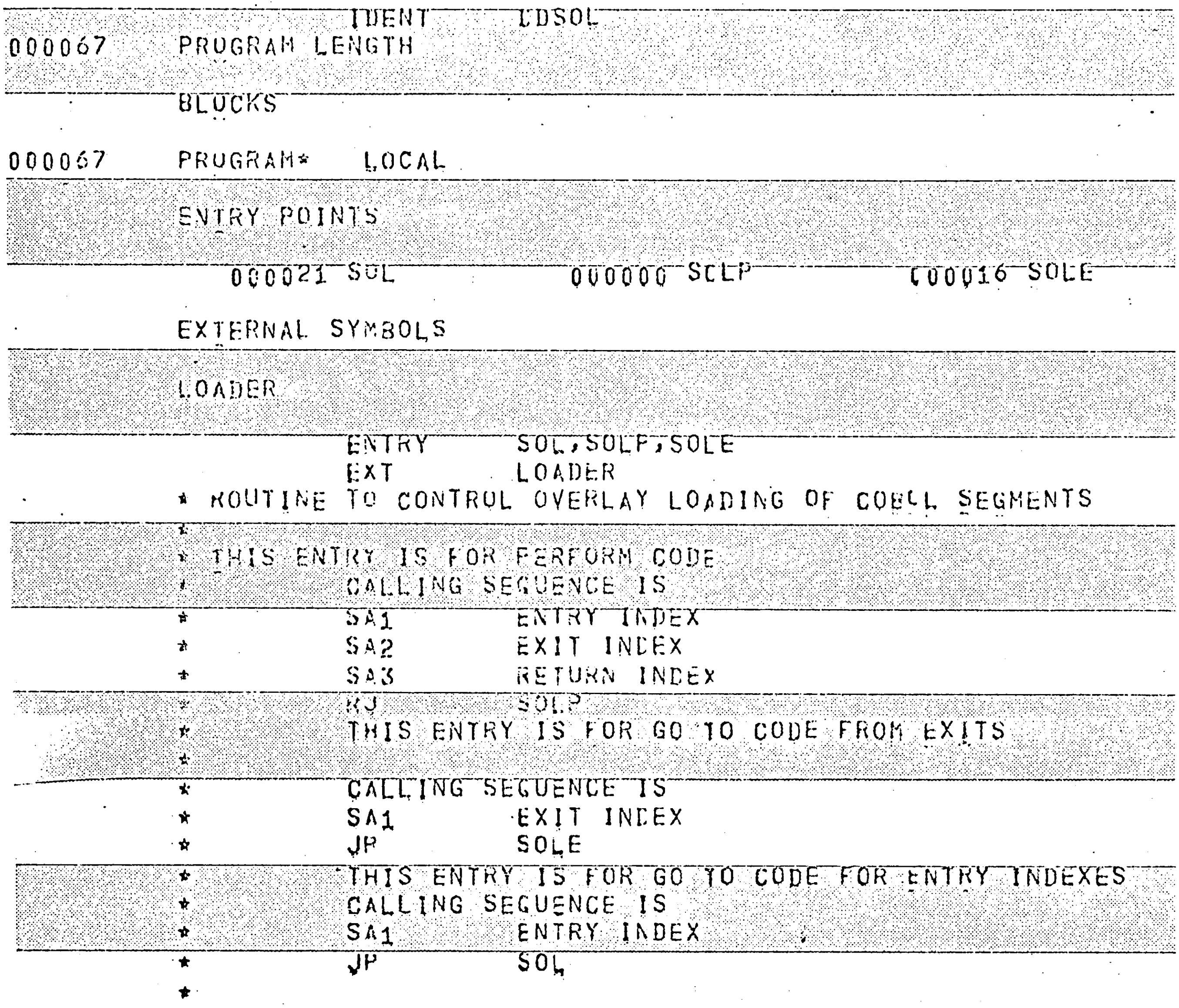


Figure 6-31. DDSOL Flowchart



```
                    IDENT     DDSOL
000067    PROGRAM LENGTH

          BLOCKS

000067    PROGRAM*   LOCAL

          ENTRY POINTS
             000021 SOL          000000 SOLP          000016 SOLE

          EXTERNAL SYMBOLS

          LOADER
                    ENTRY     SOL,SOLP,SOLE
                    EXT       LOADER
          * ROUTINE TO CONTROL OVERLAY LOADING OF COBOL SEGMENTS
          *
          * THIS ENTRY IS FOR PERFORM CODE
          *         CALLING SEQUENCE IS
          *         SA1       ENTRY INDEX
          *         SA2       EXIT INDEX
          *         SA3       RETURN INDEX
          *         RJ        SOLP
          *         THIS ENTRY IS FOR GO TO CODE FROM EXITS
          *
          *         CALLING SEQUENCE IS
          *         SA1       EXIT INDEX
          *         JP        SOLE
          *         THIS ENTRY IS FOR GO TO CODE FOR ENTRY INDEXES
          *         CALLING SEQUENCE IS
          *         SA1       ENTRY INDEX
          *         JP        SOL
          *
```

Figure 6-32. DDSOL Interface Printout

Routines Called

The Loader routine.

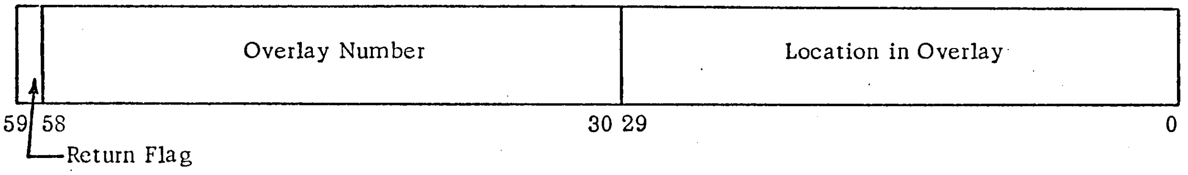
Register Usage

Destroys all registers except B1 and X4.

Operation

This routine keeps track of which overlay is in memory at any one time. When a reference is made to an overlay that is not in memory, this routine automatically calls it in and sets up the proper link to that reference.

Information about all locations is stored in words ("INDICES") with this format:



A zero overlay number indicates that the location is absolute.

SOL retains a pushdown list of indices. During the time that a section of code is being performed, the memory location which normally contains the drop-through index (specifying a normal control transfer at the exit point) will contain the return index (flagged). The drop-through index will be saved by SOL in the pushdown list.

DDSORT - COBOL SORT INTERFACE SUBROUTINE

Purpose

DDSORT forms all necessary interfaces between the COBOL object program and the general purpose SORT routine written by CDC. (See Figure 6-33.)

Restrictions

Since this routine is an interface to the main SORT routine, the writeup for that routine should be used to indicate all restrictions and possibilities that can occur in the sorting process. (See SORT-MERGE Manual.)

Calling Sequence

Shown in Figure 6-34.

Routines Called

DDTRUBL, DDOPIN, DDREAD, DDMOCKR, DDCLOS, DDOPOT, DDWRITE, DDMOCKW.

Routine SMCON, the main SORT routine, is not called by this routine but calls this routine and is intimately connected to it.

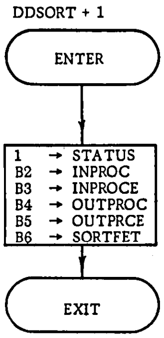
Register Usage

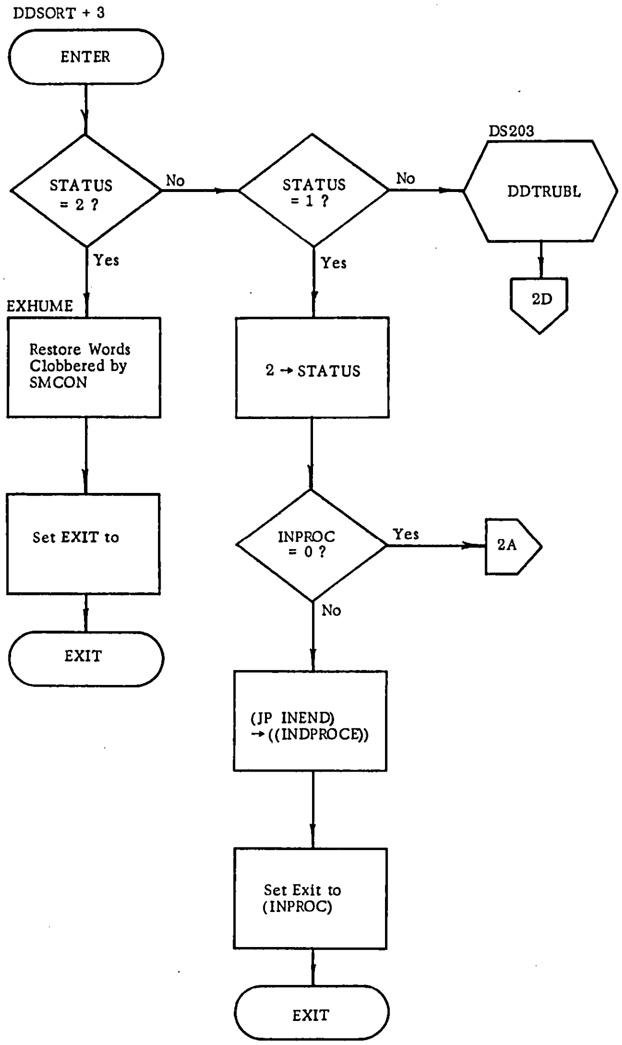
All registers volatile.

Operation

The flowchart shown in Figure 6-32 indicates the actual operation of this routine. In general, this routine is a collection of interconnecting routines to provide the proper linkage and switching between:

1. The COBOL object code.
2. The main SORT routine.
3. The input procedure provided in the COBOL object code (if such exists).
4. The output procedure provided in the COBOL object code (if such exists).





DDSORT + 1	Initialize
3	EXIT 1
5	EXIT 3
7	EXIT 4
9	Release
11	Return
12	Move Completed (+ 0) : Using Rec Length Key
13	Move Completed (+ 8) : Released Rec Length Key

Figure 6-33. DDSORT Flowchart (1 of 6)

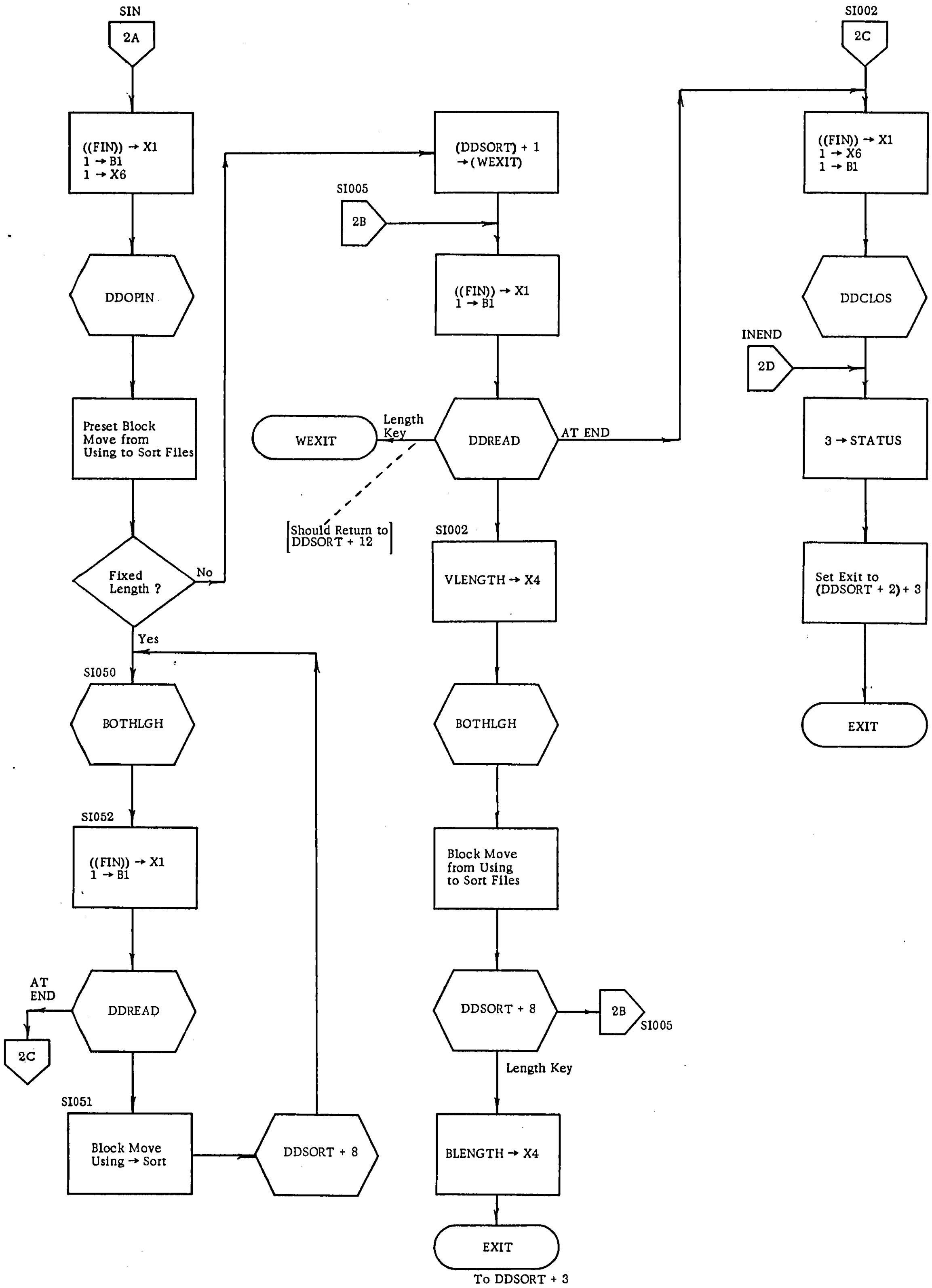


Figure 6-33. DDSORT Flowchart (2 of 6)

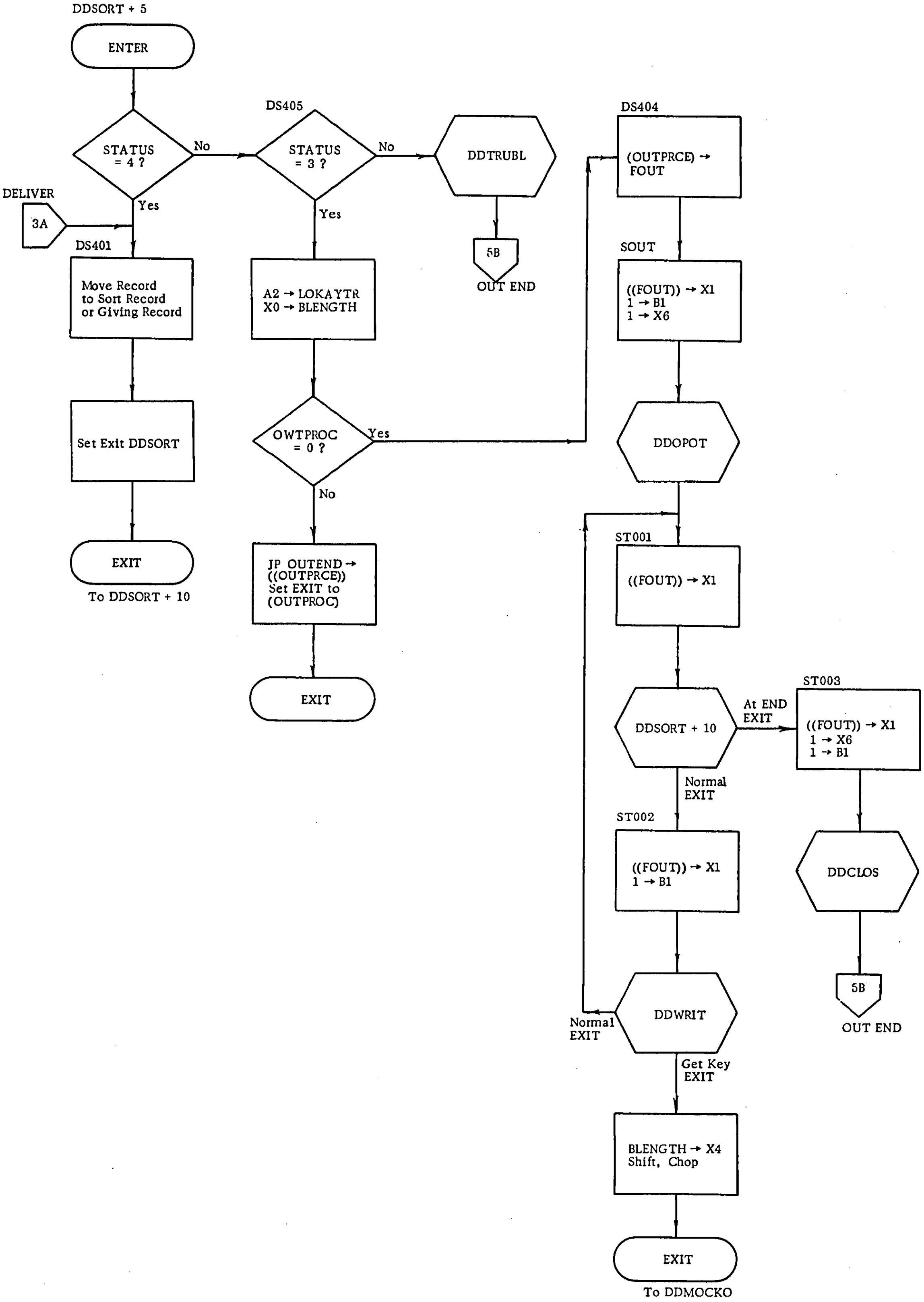


Figure 6-33. DDSORT Flowchart (3 of 6)

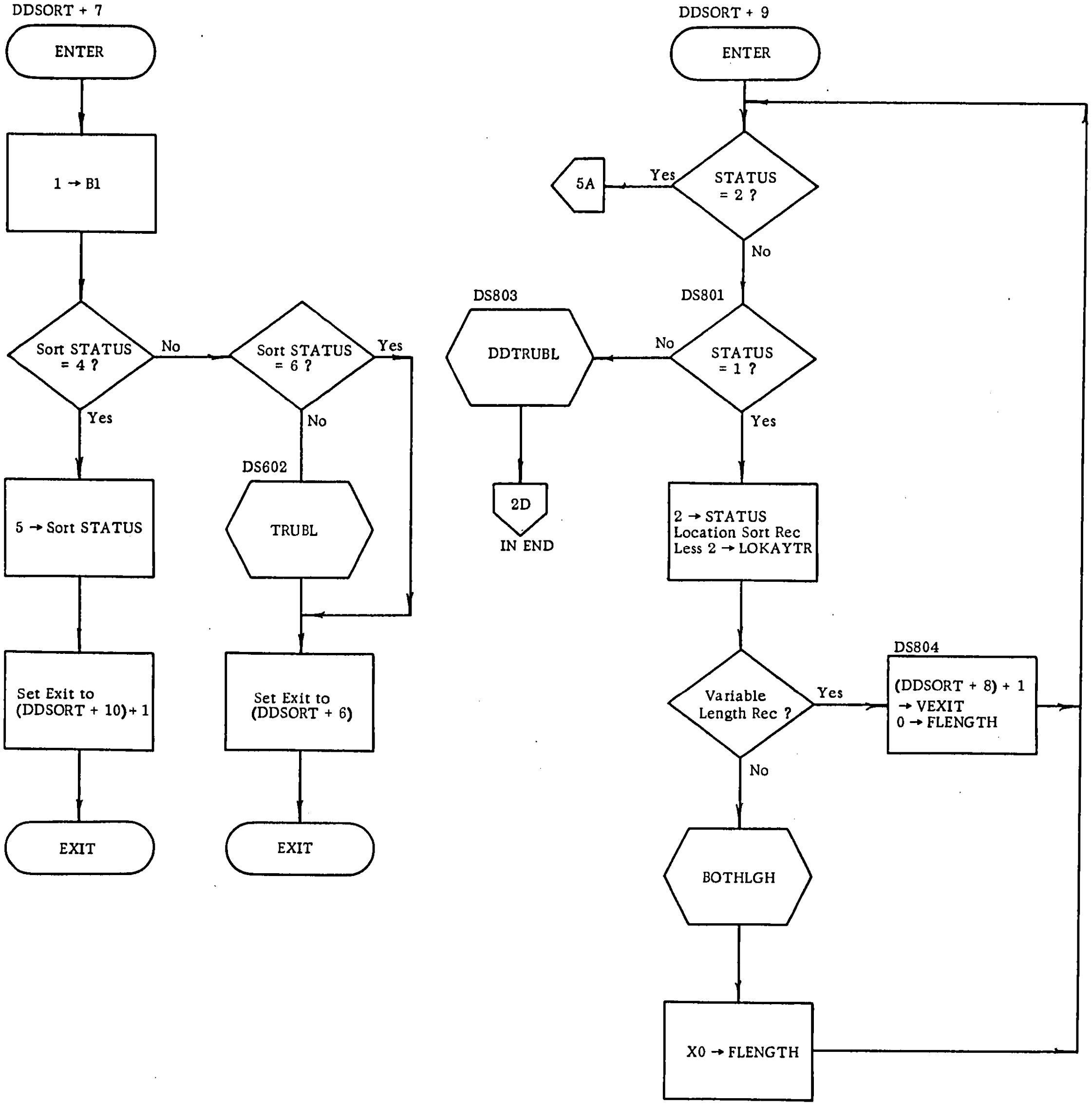


Figure 6-33. DDSORT Flowchart (4 of 6)

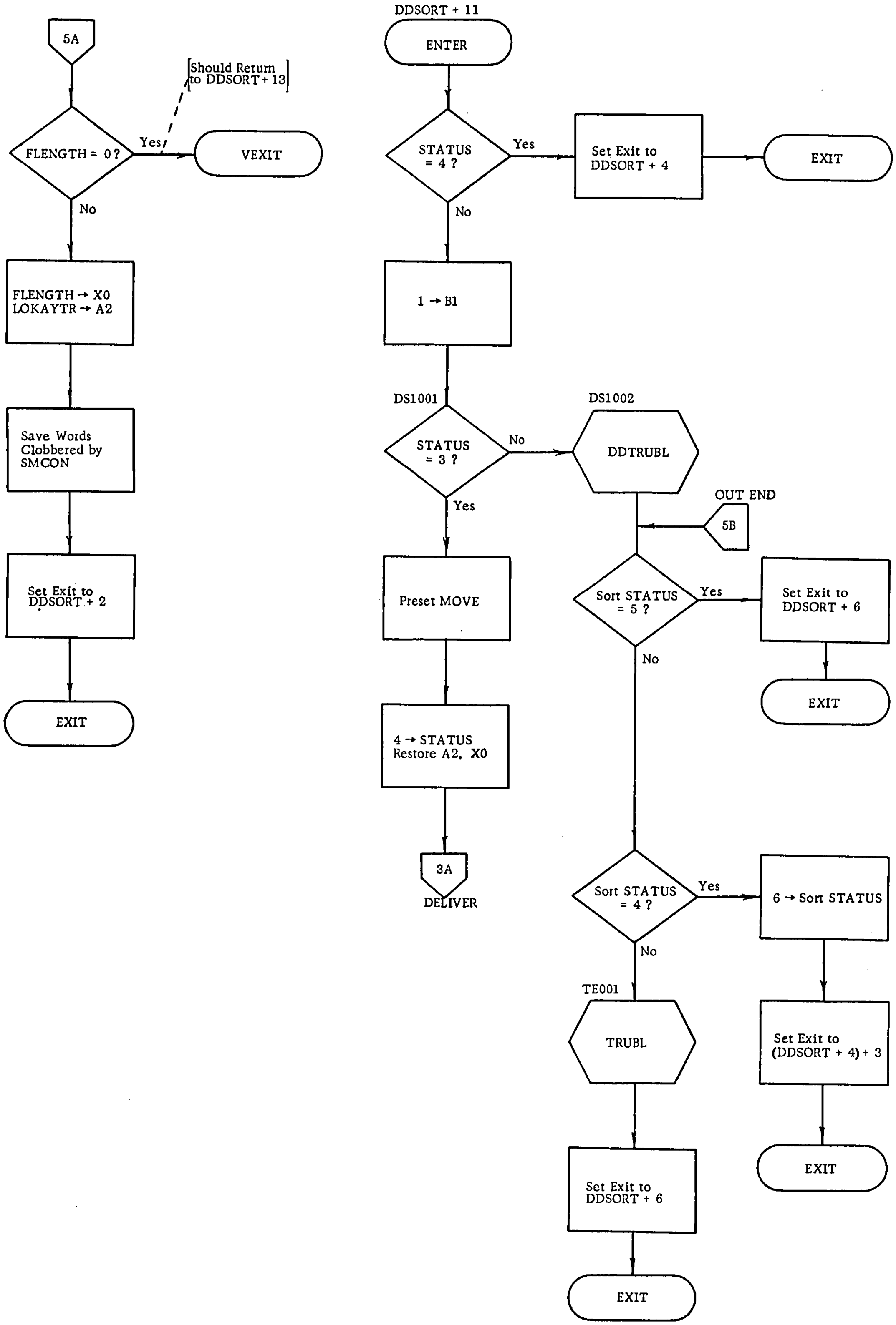


Figure 6-33. DDSORT Flowchart (5 of 6)

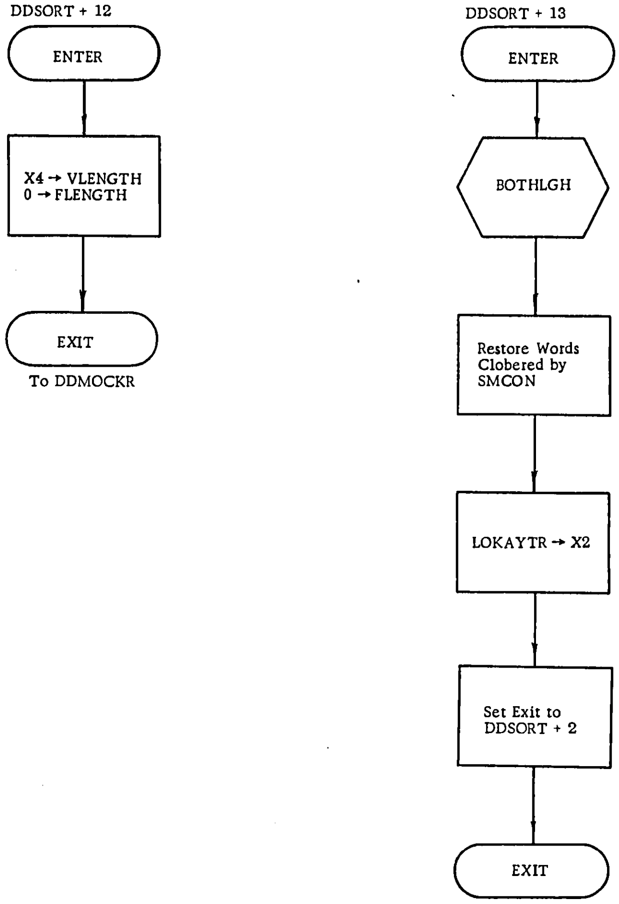


Figure 6-33 DDSORT Flowchart (6 of 6)

```
                      IDENT      DDSORT
000355    PROGRAM LENGTH

          BLOCKS

000355    PROGRAM*    LOCAL

          ENTRY POINTS

             000000 D.SORT

          EXTERNAL SYMBOLS

          D,TRUBL    D,OPIN     C,READ     D.MOCKR    D.CLOS     D,OPUT
          CPC

                      ENTRY     D.SORT
                      EXT       C.TRUBL,D.OFIN,D.READ,D.MOCKR,D,CLOS
                      EXT       C.OPUT,C.WRITE,D,MOCKW
          *  *  *     CALLS TO D,SORT
          *
          *           FROM SMCON
          *              RJ     EXIT1=D,SORT+2        TO GET A RECORD TO
          *              RJ     EXIT3=D,SORT+4        TO LELIVER A SORTEC
          *              RJ     EXIT4=D.SORT+6        TO LO END-OF-FILE F
          *
          *           FROM MAIN CODE
          *              RJ     INITIALIZE=C.SORT+0
          *              JP     RETURN FROM INLINE LOAD = D.SORT+12
          *
          *           FROM INPUT PROCEDURE
          *              RJ     RELEASE=D,SCRT+8
          *              JP     RETURN FROM IN LINE LOAC = D.SORT+13
          *
          *           FROM OUTPUT PROCEDURE
          *              RJ     RETURN=C,SORT+10
          *
          *  *  *     RETURNS FRCM D,SORT
          *                     VIA(D,SCRT+0)+0       INITIALIZED, (CALL
          *                     VIA(D,SCRT+2)+0       DELIVER IN RECORD T
          *                     VIA(D,SCRT+2)+3       INDICATE END OF FIL
          *                     VIA(D,SCRT+4)+0       INDICATE OUT RECORC
          *                     VIA(D,SCRT+4)+3       FORCE EARLY END-OF-
          *                     VIA(D,SCRT+6)+0       END-OF-FILE PROCESS
          *                     VIA(D.SCRT+8)+0       INRECORD ACCEPTED A
          *                     VIA(D.SCRT+8)+1       TO GET LENGTH INTO
          *                     VIA(D.SCRT+10)+0      DELIVER OUTRECORD T
          *                     VIA(D.SCRT+10)+1      INDICATE END-OF-FIL
          *
```

Figure 6-34. DDSORT Interface Printout (1 of 2)

```
*  *  *   MAIN CODE CALLING SEQUENCE (INITIALIZE)
*
*         INPUT REGISTERS
*                   B2 =0                      (USING OPTION)
*                      =INPUT PROCEDURE ENTRANCE
*                   B3 =USING FILE FET (IF B2=0)
*                      =INPUT PROCEDURE EXIT (IF B2 NOT 0)
*                   B4 =0                      (GIVING OPTION)
*                      =OUTPUT PROCEDURE ENTRANCE
*                   B5 =GIVING FILE FET (IF B4=0)
*                      =OUTPUT PROCEDURE EXIT (IF B4 NOT 0)
*                   B6 =SORT FILE FET
*
*         RJ        D.SORT
*         JP        NORMAL
*         (CODE TO LOAD LENGTH KEY OF USING FILE INTO X4)
*         JP        D.SORT+12
* NORMAL  BSS       0
*         (CALL TO SMCON - SEE SORT MERGERS, EXIT1=D.SORT+2,
*                   EXIT3=D.SORT+4, EXIT4=D.SORT+6,
*
*  *  *   RELEASE CALLING SEQUENCE
*         SA1       SORT-FILE-FET
*         RJ        D.SORT+8
*+        JP        NORMAL
*         (CODE TO LOAD LENGTH KEY OF SORT FILE INTO X4)
*         JP        D.SORT+13
* NORMAL  BSS       0
*
*  *  *   RETURN    CALLING SEQUENCE
*         SA1       SORT-FILE-FET
*         RJ        D.SORT+10
*+        JP        NORMAL
*+        (CODE FOR AT-END)
* NORMAL  BSS       0
*
*  *  *   OUTPUT REGISTERS
*         ON ALL RETURNS FROM INITIALIZE, RELEASE, RETURN
*         B1=1
*         X4=10H0000000000
*
*         ON RETURN TO (D.SORT+2)+0 AND CALL
*         X0=VFD 30/BYTE-SIZE,30/WORD-SIZE
*         A2=RECORD LOCATION LESS TWO
*
```

Figure 6-34. DDSORT Interface Printout (2 of 2)

This routine also handles an input file if the USING option exists or an output file in the case of the GIVING option. These routines will be enumerated and described separately in the following paragraphs.

The first routine is the INITIALIZE routine, which stores information passed to it regarding the options and the location of files that are going to be used during the sorting process.

The second routine is called by the SMCON routine at the place it calls EXIT1. The SORT routine is calling for an input record. The first time through this routine there is some initialization completed. After that and for all subsequent times, control is immediately passed to the input procedure. In case an input procedure is not provided, the standard input procedure that is part of this packaged routine is used.

The third routine is the routine that is called by SMCON at its EXIT3. At this point, the first time this is called, it initializes the entrance and exit to the output procedure. When the output procedure has executed the first RELEASE verb, sending control back to another routine in this package of routines, this routine is used to store the record transmitted into the output file. On subsequent calls to this routine from EXIT3, the record is transferred immediately into the output file.

The fourth routine is the routine entered from SMCON when it comes to the end of all records. Control may come here because it has discovered the end of records or control may come here because the output procedure dropped through its own exit and a return was made to SMCON by INEND, in which case SMCON comes here directly.

The fifth routine in the packaged routines is the RELEASE routine. Control comes here when the RELEASE verb is executed in an input procedure. The first time, the record location and its length is preset. If the SORT file is a variable-length record type, it will compute this length and properly pass it to the SORT routine each time through.

The sixth routine is the RETURN routine. This routine is called when the RETURN verb is executed in an output procedure. This routine presets the loop in the EXIT3 code mentioned above to transfer the record delivered by the SORT into the SORT file for the customer. This transfer loop is somewhat different if the record has a variable length. The length of the record was stored when it was given to the SORT routine and it is returned by the SORT routine. There is no computation here from the contents of the record to give its length.

The seventh routine in this package of routines is the standard input procedure that is provided when the source programmer uses the USING option in the SORT variable. It takes care of opening and reading the input file, delivering the records to the SORT, and taking proper action at the end of the input file.

The eighth routine in this package of routines is the standard output procedure. This, in a similar manner, opens and writes the file specified by the GIVING option when this is used with the SORT verb..

DDSTRP - SIGN STRIPPING SUBROUTINE

Purpose

DDSTRP converts a field from one in which the sign is given by an overpunch on a low digit to 9's complement format.

Interface

The interface is described in Figure 6-35.

Restrictions

There is no check for the validity of the input. Certain illegal input will probably be interpreted as $\overset{+}{0}$ or $\overset{-}{0}$ by default and the output cannot be easily predicted. The operation of this routine can easily be understood from the code.

Calling Sequence

Return jump.

Routines Called

None

Register Usage

The register usage is shown in Figure 6-35.

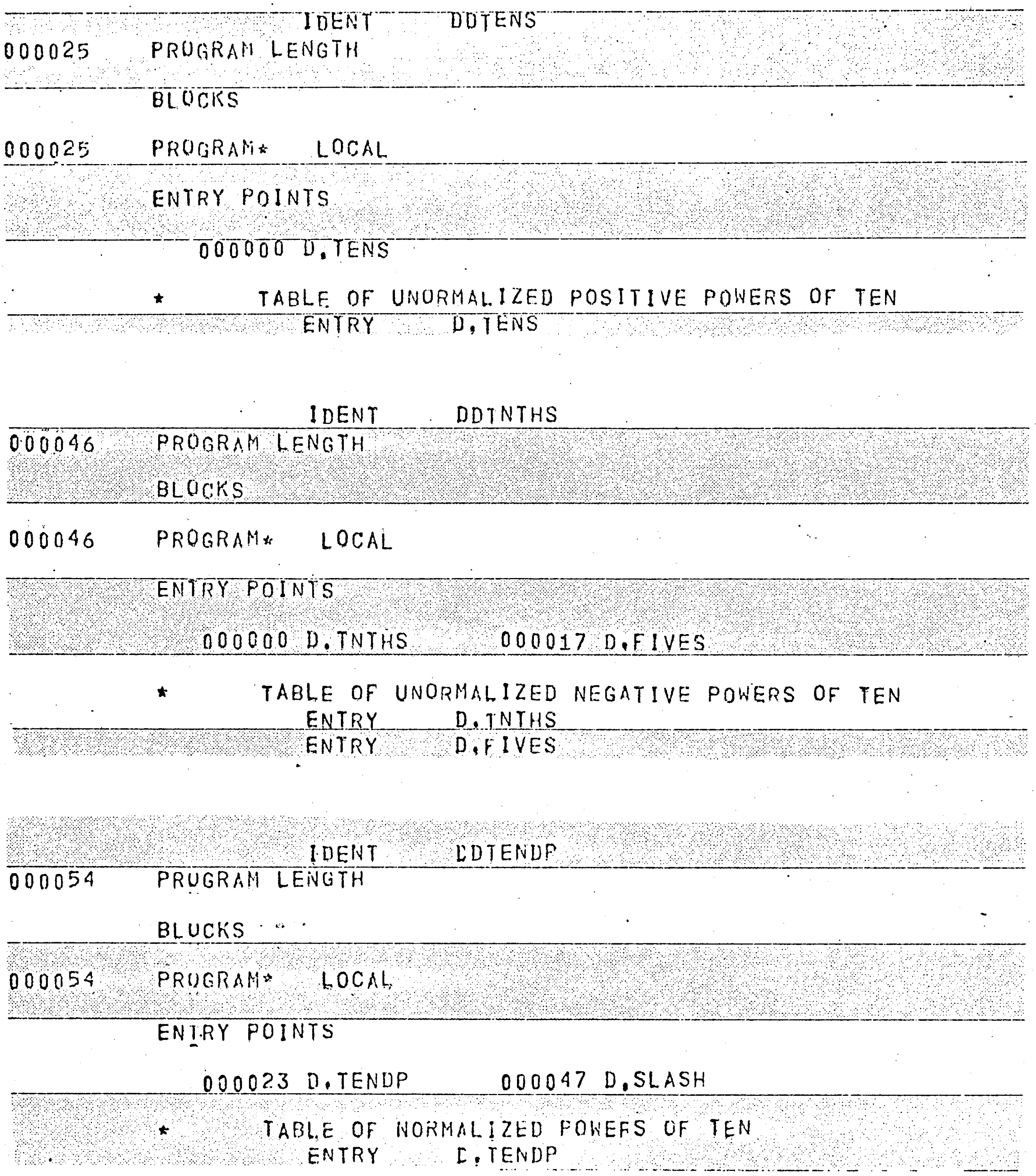
```
                         IDENT DDSTRP
000031    PROGRAM LENGTH

          BLOCKS

000031    PROGRAM*   LOCAL

          ENTRY POINTS
             000000 D,STRP1        000014 D,STRP2

          *
          *              CONVERT FROM SIGN GIVEN BY OVERPUNCH CF LO DIGIT
          *                        NINES COMPLEMENT,  ALL ADJUST RIGHT,
          *
          *              REGISTER USAGE
          *
          *              X1        INPUT AND OUTPUT FOR SINGLE PRECISION
          *              X1,X2     INPUT AND OUTPUT FOR DOUBLE PRECISION,
          *
          *              X3,X5     VOLATILE
          *
          *              ALL OTHER REGISTERS ARE UNTOUCHED
                         ENTRY     D,STRP1,D,STRP2
```

Figure 6-35. DDSTRP Interface Printout

DDTENS, DDTNTHS, AND DDTENDP SUBROUTINES

Purpose

The purpose of these three routines is to provide powers of ten for various operations requiring them throughout the object code. For further information see Figure 6-36.



```
                         IDENT      DDTENS
000025    PROGRAM LENGTH

          BLOCKS

000025    PROGRAM*   LOCAL

          ENTRY POINTS

             000000 D.TENS

          *        TABLE OF UNORMALIZED POSITIVE POWERS OF TEN
                         ENTRY      D.TENS


                         IDENT      DDTNTHS
000046    PROGRAM LENGTH

          BLOCKS

000046    PROGRAM*   LOCAL

          ENTRY POINTS

             000000 D.TNTHS      000017 D.FIVES

          *        TABLE OF UNORMALIZED NEGATIVE POWERS OF TEN
                         ENTRY      D.TNTHS
                         ENTRY      D.FIVES


                         IDENT      DDTENDP
000054    PROGRAM LENGTH

          BLOCKS

000054    PROGRAM*   LOCAL

          ENTRY POINTS

             000023 D.TENDP      000047 D.SLASH

          *        TABLE OF NORMALIZED POWERS OF TEN
                         ENTRY      D.TENDP
```

Figure 6-36. DDTENS, DDTNTHS, and DDTENDP Interface Printouts

DDTRUBL - COBOL ERROR ROUTINE FOR OBJECT RUNNING SUBROUTINE

Purpose

DDTRUBL furnishes a means for the object program and subroutines running at object time to:

1. Inform the user that abnormal situations have arisen.
2. Allow the user to make a decision at this point.
3. In certain cases, to give diagnostic information about the situation.

The routine delivered with the compiler does not perform all these functions completely. It is expected that this routine will be rewritten by using installations to suit the particular mode of operation for those installations.

Interface

The interface is shown in Figure 6-37.

Restrictions

This routine does not save and restore registers. In the event that the message from this routine is displayed on the console, the operating system as presently operating will abort the job if the message is in any way garbled.

Calling Sequence

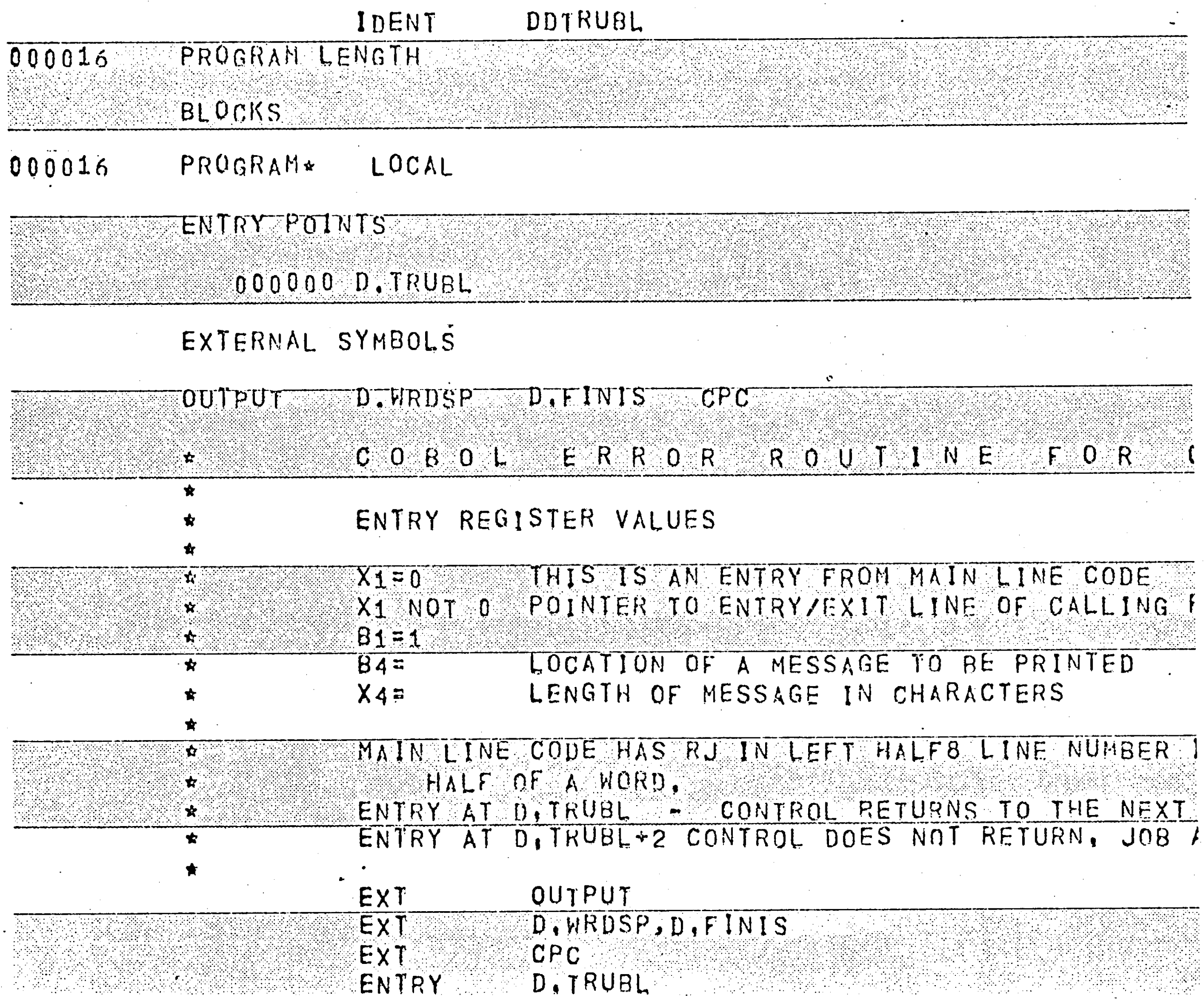
See Routines Called.

Routines Called

DDDSPLY, DDFINIS

Register Usage

All registers used.



```
                          IDENT      DDTRUBL
000016     PROGRAM LENGTH

           BLOCKS

000016     PROGRAM*   LOCAL

           ENTRY POINTS

              000000 D.TRUBL

           EXTERNAL SYMBOLS

           OUTPUT     D.WRDSP    D.FINIS    CPC

           *          C O B O L   E R R O R   R O U T I N E   F O R
           *
           *          ENTRY REGISTER VALUES
           *
           *          X1=0       THIS IS AN ENTRY FROM MAIN LINE CODE
           *          X1 NOT 0   POINTER TO ENTRY/EXIT LINE OF CALLING
           *          B1=1
           *          B4=        LOCATION OF A MESSAGE TO BE PRINTED
           *          X4=        LENGTH OF MESSAGE IN CHARACTERS
           *
           *          MAIN LINE CODE HAS RJ IN LEFT HALF8 LINE NUMBER
           *              HALF OF A WORD.
           *          ENTRY AT D,TRUBL  -  CONTROL RETURNS TO THE NEXT
           *          ENTRY AT D,TRUBL+2 CONTROL DOES NOT RETURN, JOB
           *
                      EXT        OUTPUT
                      EXT        D,WRDSP,D,FINIS
                      EXT        CPC
                      ENTRY      D,TRUBL
```

Figure 6-37. DDTRUBL Interface Printout

Operation

This routine has the information coming to it to enable it to do a backward trace through levels of subroutines to the main program to determine where it is in the main program and to display the line number of the source code. It has the ability to return to the main program or not to return, depending upon which entry is used. After entry, this routine calls the DDDSPLY routine to display the message. As delivered in Version 1, this routine will display the message on the final output.

DDZONE - COBOL FORMAT SUBROUTINE

Purpose

The purpose of this routine is to convert a number from the 9's complement format to one in which the sign is given by an "overpunch" over the low digit.

Interface

The interface is shown in Figure 6-38.

Restrictions

This routine makes no checks of any kind and in the event of getting garbage in the program will deliver garbage out.

Operation

DDZONE is a simple routine; the operational details can be easily understood from the code.

```
                    IDENT      DDZONE
000050     PROGRAM LENGTH

           BLOCKS

000050     PROGRAM*    LOCAL

           ENTRY POINTS

              000000 D.ZONE3       000012 D.ZONE5       000024 D.ZONE6
           *           CONVERT FROM NINES COMPLEMENT FORMAT TO SIGN GIVE
           *                      OVERPUNCH OVER LO DIGIT.  ALL ADJUSTED
           *
           *           REGISTER USAGE
           *
           *           D.ZONE3    INPUT AND OUTPUT IN X3
           *           D.ZONE5    INPUT AND OUTPUT IN X3,X5
           *           D.ZONE6    INPUT AND OUTPUT IN X6
           *           D.ZONE7    INPUT AND OUTPUT IN X6,X7
           *
           *           REGISTERS X1,X2,B6 ARE VOLATILE, NO OTHER TOUCHED
                       ENTRY      D.ZONE3,D.ZONE5,D.ZONE6,D.ZONE7
```

Figure 6-38. DDZONE Interface Printout

SNAP - SNAPSHOT OF REGISTER STATUS SUBROUTINE

Purpose

This routine converts all the 6400/6500/6600 hardware registers to display code and prints them on the output device.

Calling Sequence

```
        RJ      SNAP
```

Routines Called

BLOCKIO

Register Usage

Registers A5 and A7 are the only two 18-bit registers which are destroyed by a call to SNAP. All other registers are preserved.

Operation

SNAP also has the facility to dump, or take a snapshot, specified portion of core. These features can be utilized by ENTER subroutines, which set up the proper calling sequences to either CORE or COREDMP (see compiler description of debugging aids).

SNAP uses BLOCKIO to buffer to the output device its snapshots.

DDXCEPT - ACCEPT SUBROUTINE

Purpose

To be used by the "ACCEPT" verb. (See Figure 6-39.)

Interface

Input registers:

A1	Source file (input)
B4	Base address for field
X3	BYTE offset of field
X4	Field size in bytes

All other registers volatile, none restored. (See Figure 6-40.)

Calling Sequence

```
RJ      DDXCEPT
```

Restrictions

Calls D. TRUBL if file is not open

Routines Used

DDMOVIO
CPC
IOWRITE
IOREAD
DDTRUBL

Operation

This routine reads a unit record into working storage from the input files named and then uses DDMOVIO to move it into the field named.

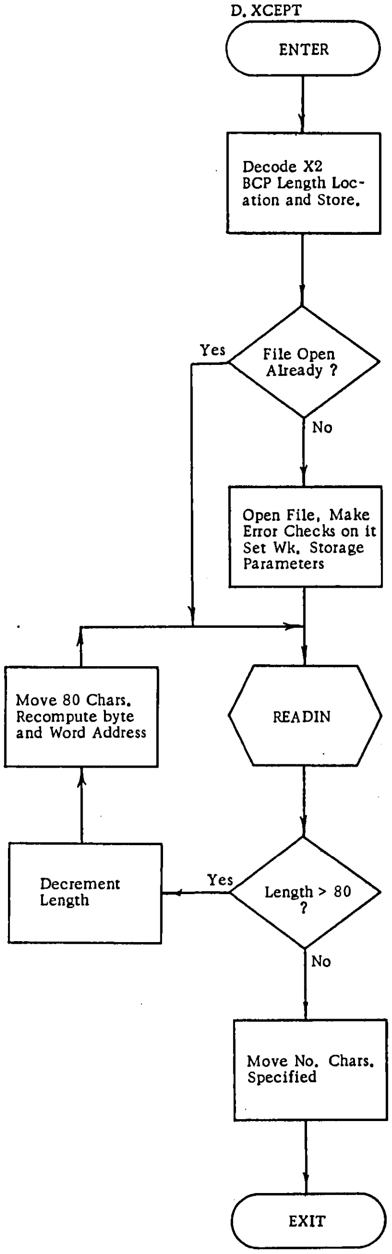


Figure 6-39. DDXCEPT Flowchart

```
                    IDENT     DDXCEPT
000102.   PROGRAM LENGTH

          BLOCKS

000102    PROGRAM*   LOCAL

          ENTRY POINTS
             000000 D,XCEPT

          EXTERNAL SYMBOLS

          D.MOVIO   CPC       IOWRITE   IOREAD

                    ENTRY     D,XCEPT
                    EXT       D,MOVIO
                    EXT       CPC,IOWRITE,IOREAD
```

Figure 6-40. DDXCEPT Interface Printout

DDSUBSC - SHORT FIELD SUBSCRIPTED LOAD - STORE SUBROUTINE

Purpose

To fetch or store short fields from or to subscripted fields.

Interface

See Figure 6-41.

Routines Called

DDTNTHS
DDTENS

Operation

For loading, the field is picked up and shifted properly to right adjust it in one or two registers which are then filled out with zeros or blanks.

For storing, the field is shifted to position, combined with the proper right and left background, and then stored. The mask in X0 is used in this case as a preliminary test for ON-SIZE ERROR.

```
                     IDENT         DDSUBSC
000106      PROGRAM LENGTH

            BLOCKS

000106      PROGRAM*    LOCAL

            ENTRY POINTS

               000003 D.SBSC1          000023 D.SBSC2          000040 D.SBSC6

            EXTERNAL SYMBOLS

            D.TNTHS    D.TENS      SNAP        FLUSH

            *
            *          D.SBSC1     LOAD S,P, DPC SUBSCRIPTED ITEM
            *          D.SBSC2     LOAD D,P, DPC SUBSCRIPTED ITEM
            *          D.SBSC6     STORE S,P, DPC SUBSCRIPTED ITEM
            *          D.SBSC7     STORE D,P, DPC SUBSCRIPTED ITEM
            *          D.SBSCG     STORE S,P, ITEM IN FIELD JUST LOADED
            *                      (NOT YET IMPLEMENTED)
            *          ON ENTRY
            *                      X2 CONTAINS BYTES OF OFFSET
            *                      X0 IS A MASK - 10/20- SIZE
            *                      B5 IS THE BASE ADDRESS OF THE FIELD
            *                      B4 IS 10-SIZE, IN BITS
            *                      X4 CONTAINS ZEROS OR BLANKS
            *                      B1 IS 1
            *
            *          ON EXIT
            *                      LOADED ITEM IS IN X1 OR X1,X2
            *                      STORED ITEM, FROM X6 OR X6,X7 IS IN COR
            *                      IF ON SIZE ERROR TRUNCATION OCCURS, B7
            *
                       ENTRY       D.SBSC1
            *
                       EXT         D.TNTHS,D.TENS
                       EXT         SNAP,FLUSH
                       ENTRY       D.SBSC2,D.SBSC6,D.SBSC7
```

Figure 6-41. DDSUBSC Interface Printout

SUBMV - SUBSCRIPTED LONG MOVE SUBROUTINE

Purpose

To effect moves where either the source or the subfield is subscripted by a variable. (See Figure 6-42.)

Interface

See Figure 6-43.

Restrictions

None.

Routines Called

D. MOVIO

Operation

This routine converts byte offsets, which may equal or exceed 10, to equivalent word offsets and fractional word offsets as required to call DDMOVI0.

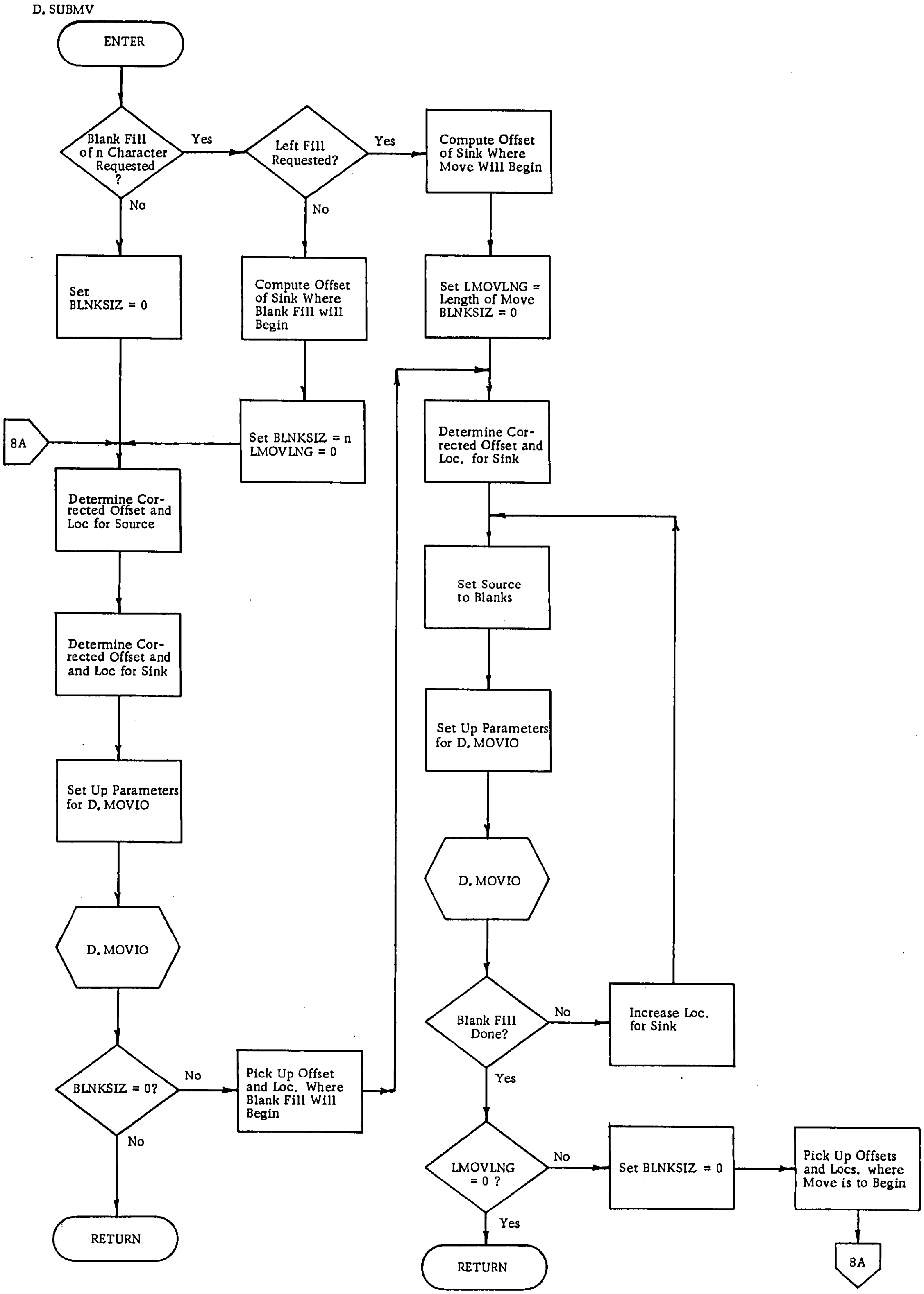


Figure 6-42. DDSUBMV Flowchart (1 of 2)

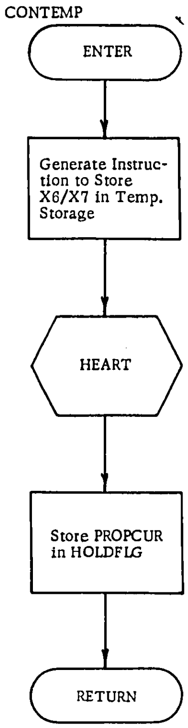
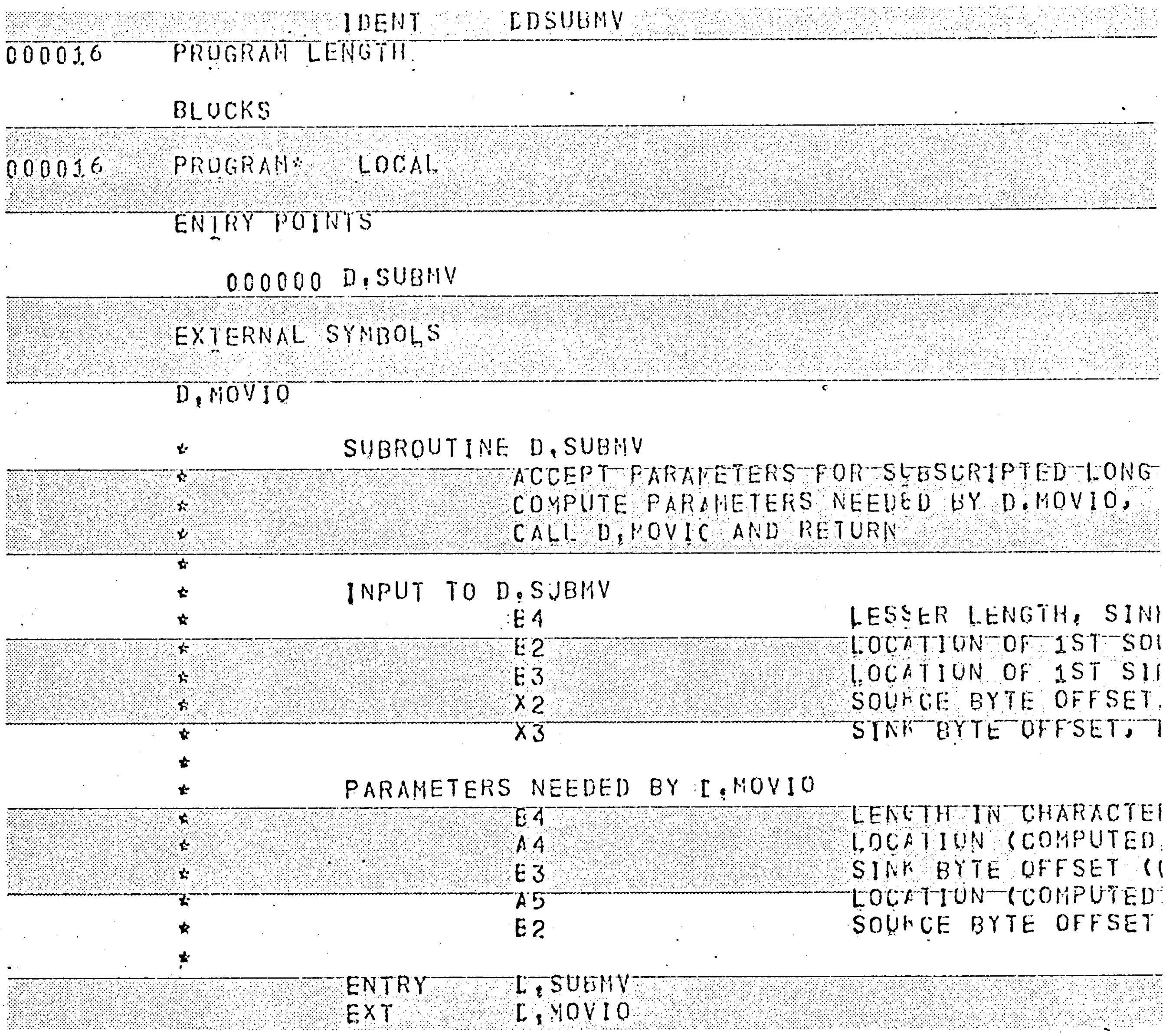


Figure 6-42. DDSUBMV Flowchart (2 of 2)



```
                          IDENT     DDSUBMV
000016        PROGRAM LENGTH

              BLOCKS

000016        PROGRAM*    LOCAL

              ENTRY POINTS

                 000000 D,SUBMV

              EXTERNAL SYMBOLS

              D,MOVIO

              *        SUBROUTINE D,SUBMV
              *                  ACCEPT PARAMETERS FOR SUBSCRIPTED LONG
              *                  COMPUTE PARAMETERS NEEDED BY D.MOVIO,
              *                  CALL D,MOVIO AND RETURN
              *
              *        INPUT TO D,SUBMV
              *                  E4                 LESSER LENGTH, SIN
              *                  E2                 LOCATION OF 1ST SO
              *                  E3                 LOCATION OF 1ST SI
              *                  X2                 SOURCE BYTE OFFSET.
              *                  X3                 SINK BYTE OFFSET,
              *
              *        PARAMETERS NEEDED BY D,MOVIO
              *                  E4                 LENGTH IN CHARACTE
              *                  A4                 LOCATION (COMPUTED
              *                  E3                 SINK BYTE OFFSET (
              *                  A5                 LOCATION (COMPUTED
              *                  E2                 SOURCE BYTE OFFSET
              *
                       ENTRY     D,SUBMV
                       EXT       D,MOVIO
```

Figure 6-43. DDSUBMV Interface Printout

DDEXP - EXPONENTIAL INTERFACE SUBROUTINE

Purpose

This routine is called for the exponential operation in source code COMPUTE statements.

Interface

See Figure 6-44.

Restrictions

Even for integers where the results are exact, this routine may introduce a relative error of up to 2^{-45}.

Routines Called

ALOG. and EXP. (These routines are described with other FORTRAN subroutines in other CDC literature).

Operation

This routine uses ALOG. to compute the logarithm of the base, multiplies that by the exponent, and then uses EXP. to get the result.

```
                         IDENT      DDEXP
000014    PROGRAM LENGTH

          BLOCKS

000012    PROGRAM*   LOCAL
000002    LITERALS*  LOCAL

          ENTRY POINTS

           000000 D.EXP

          EXTERNAL SYMBOLS

          ALOG.      EXP.

                         ENTRY      D.EXP
                         EXT        ALOG.,EXP.
        *
        *                C O B O L   E X P O N E N T I A L   R O U T I N
        *
        *                USES THE SAME LOGARITHM AND EXONENTIAL ROUTINES ,
        *                INPUT      B1=1, X6=BASE, X1=EXPONENT
        *                OUTPUT  B1=1, X6=RESULT
        *                           A0    RESTORED
        *                ALL OTHER REGISTERS ARE VOLATILE
```

Figure 6-44. DDEXP Interface Printout

DDANCM - ALPHANUMERIC STATUS TEST SUBROUTINE

Purpose

This routine is called for the source code statements "IF NUMERIC" and "IF ALPHABETIC" to determine the current status of a given field.

Interface

Shown in Figure 6-45.

Restrictions

If any characters are neither alphabetic nor numeric, their effect on this test is not predictable in general.

Routines Called

None

Operation

This routine does a character-by-character examination of the field.

```
                IDENT     DDANCM
000051    PROGRAM LENGTH

          BLOCKS
000051    PROGRAM*    LOCAL
          ENTRY POINTS

            000026 D.ALPCM       000000 D.NUMCM

                ENTRY     D.ALPCM,D.NUMCM
*               IF ALPHABETIC, IF NUMERIC ROUTINES
*
*               A L L   R E G I S T E R S   A R E   U S E D
*
*               B1=1 ON EXIT, NO OTHER REGISTERS SAVED.
*
*
*               B2 = BASE ADDRESS
*               X2 = BYTE OFFSET
*               B5 = BYTE LENGTH
*               B7 = 0 (UNSIGNED), 1(SIGNED) FOR NUMERIC ONLY
*
*               IF NUMERIC ENTERS AT D.NUMCM AND RETURNS AT
*                     NEXT LINE FOR TRUE, THE FOLLOWING FOR F
*               IF ALPHABETIC ENTERS AT D.ALPCM AND RETURNS AT
*                     NEXT LINE FOR TRUE, THE FOLLOWING FOR F
*               B1 =1 ON EXIT
```

Figure 6-45. DDANCM Interface Printout

DDDSPLY

Purpose

To effectuate the DISPLAY statement. (See Figure 6-46.)

Calling Sequence

RJ	D. DSPLY This entry adds a field into the output line but does not output it.
RJ.	D. WRDSP This entry adds a field to the output line, outputs the line and clears it.

Registers on input

B4	base address for field
X3	character offset of field
X4	character size of field
A1	output file location

Restrictions

The total number of characters is limited to 136.

Routines Called

CPC
IOREAD
IOWRITE
DDMOVIO

Operation

DDMOVIO is called to move fields into an 136 word buffer. This routine keeps track of a current location in this buffer.

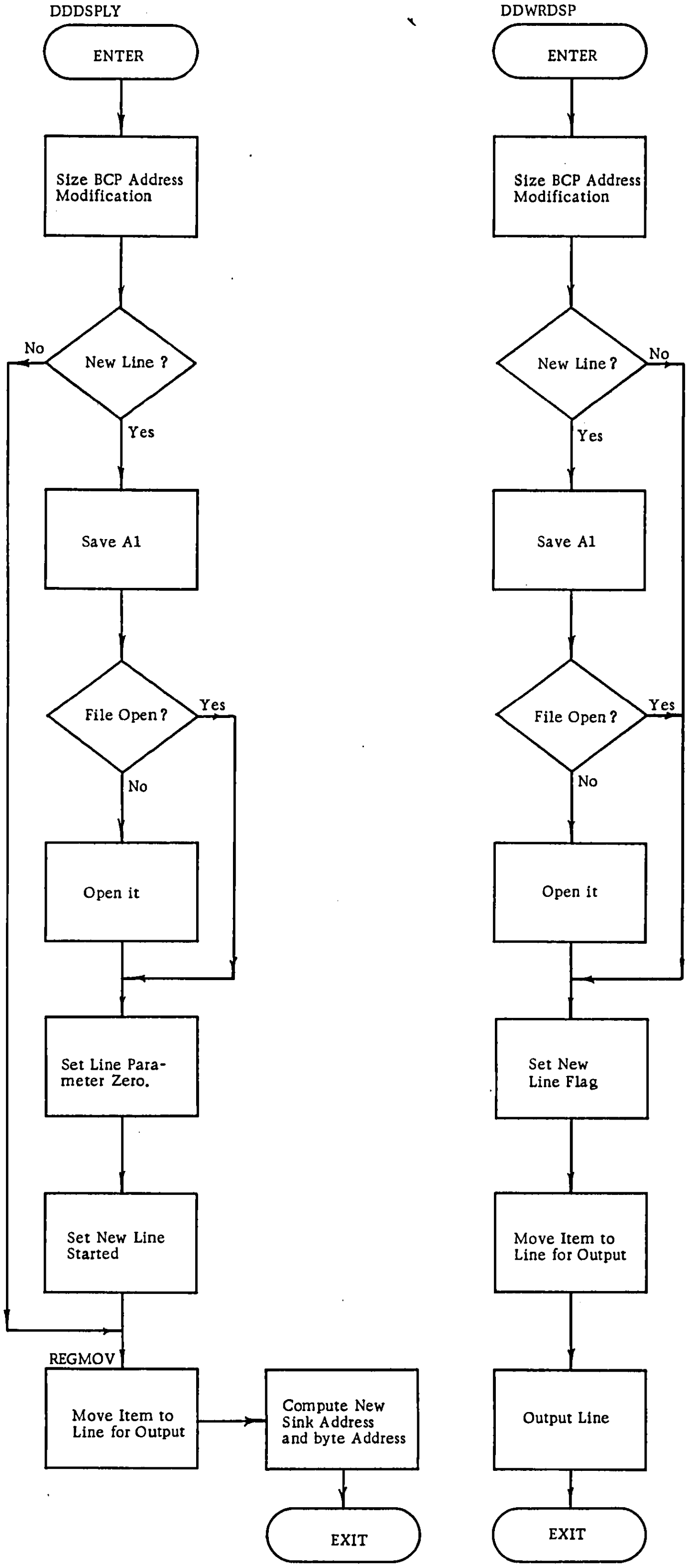


Figure 6-46. DDDSPLY Flowchart

APPENDICES

APPENDIX A – PROCEDURE DIVISION LEXICON LIST

.	1041
,	0125
&	0121
*	0126
**	0127
>	0141
-	0122
/	0123
(	0124
=	0120
<	0140
ACCEPT	1601
ADD	1724
ADVANCING	0134
AFTER	0135
ALL	0136
ALPHABETIC	0137
ALTER	1602
AND	0005
ASCENDING	0006
AT	0007
BEFORE	0011
BEGINNING	0010
BY	0012
CLOSE	1603
COBOL	0013
COMPUTE	1725
CONSOLE	0132
CORRESPONDING	0014
DECLARATIVES	1034
DEPENDING	0015
DESCENDING	0016
DISPLAY	1604
DIVIDE	1726
DIVISION	0017
ELSE	1035
END	1036
ENDING	0020
ENTER	1605
ENTRY	1623
EQ	4003
EQUAL	0021
EQUALS	0130
ERROR	0022
EXAMINE	1606
EXCEEDS	0023
EXIT	1607
FILE	0024
FIRST	0025
FROM	0026
GENERATE	1610
GIVING	0027
GO	1611
GQ	4005
GR	4001
GREATER	0030
GREATER-EQUAL	4012
HIGH-VALUE	0031
I-O	0036
IF	1531
IN	0032
INCLUDE	1442
INITIATE	1612
INPUT	0033
INPUT-OUTPUT	0036
INTO	0034
INVALID	0035
IS	0037
KEY	0040
KEYS	0040
LABEL	0041
LEADING	0042
LESS	0043
LESS-EQUAL	4011
LINE	0044
LINE-COUNTER	0045
LINES	0044
LOCK	0046
LOW-VALUE	0047
LQ	4004
LS	4002
MOVE	1613
MULTIPLY	1727
NE	0110
NEGATIVE	0050
NEXT	0051
NGR	4007
NLS	4010
NO	0052
NOT	0053
NOTE	1443
NQ	4006
NUMERIC	0054
OF	0032
ON	0056
OPEN	1614

OR	0057
OTHERWISE	1035
OUTPUT	0060
PAGE-COUNTER	0061
PERFORM	1615
POSITIVE	0062
PROCEDURE	1040
PROCEED	0063
QUOTE	0064
QUOTES	0064
READ	1532
RECORD	0065
RECORD-MARK	0133
RECORDS	0131
REEL	0066
RELEASE	1616
REPLACING	0067
REPORTING	0070
RETURN	1533
REVERSED	0071
REWIND	0072
ROUNDED	0073
RUN	0074
SECTION	0075
SEEK	1617
SENTENCE	0076
SIZE	0077
SORT	1620
SPACE	0100
SPACES	0100
STANDARD	0101
STOP	1621
SUBTRACT	1730
TALLYING	0102
TERMINATE	1622
THAN	0103
THEN	0104
THROUGH	0105
THRU	0105
TIMES	0106
TO	0107
UNEQUAL	0110
UNITS	0111
UNTIL	0112
UPON	0113
USE	1444
USING	0114
VARYING	0115
WITH	0116
WRITE	1723
ZERO	0117
ZEROES	0117
ZEROS	0117

APPENDIX B – SYNTAX ANALYSIS TABLE PROGRAM (SYNTBLE)

GENERAL

The Syntax Analysis Table Program (SYNTBLE) is written in COBOL language. See Section 2 for a description of the syntax language.

SYNTBLE accepts card input from the following three formats, which occur in prescribed order (not intermixed):

1 7	8 9	10 19	20	28	36	44	52	57	65	73 80
xxxxxxx	99	$xxxxxxxxx xxxxxxxxxx	D999 xxxxxxxx	D999 xxxxxxxx	Yes No 99 R2	D999 xxxxxxxx	blank	D999 xxxxxxxx	D999 xxxxxxxx	Yes No 99 R2

cc 1-7 The name of a syntax table item. These names must be submitted to SYNTBLE in 6600 collating sequence.

cc 8-9 The numeric ordinal within the alphabetic syntax table item starting with 1. These ordinals must be in ascending numeric sequence with no numbers skipped.

cc 10-19 One of

1. A lexicon which must be found in the lexicon table in working storage of the SYNTBLE program. A lexicon is preceded by a $ in cc 10.
2. A syntax table item name.
3. An external subroutine name.
4. A blank.

cc 20-27
cc 28-35
cc 44-51
cc 57-64
cc 65-72

The "no" action fields and "yes" action fields, respectively. They may be comprised of one of four options.

1. A three-digit diagnostic number preceded by a "D".
2. A syntax table item name.
3. An external subroutine name.
4. A blank.

cc 36-43
cc 73-80 The "no" GO TO field and the "yes" GO TO field, respectively. They may be comprised of one of five options:

1. The word YES.
2. The word NO.
3. A two-digit number which must be an ordinal of the syntax table item name subset.
4. The two character field R2.
5. A blank.

Cards of the above type are separated by one blank card from the second type, which is as follows:

```
cc
1                                   20
xxxxxxxxx                           9999
```

cc 1-9	The name of a lexicon to be read into a lexicon dictionary in core.
cc 20-23	The four-digit lexicon attribute code.

The only characters significant to SYNTBLE are cc 1-9 and cc 20-23. All other information on the card is ignored.

This lexicon deck is required input. cc 1-9 must be in ascending alphabetical sequence. Cards of the above type are separated by one blank card from the third type, which is as follows:

```
cc
1
xxxxxxxxxx
```

cc 1-10	The name of an external subroutine.

The only characters significant to SYNTBLE are cc 1-10. All other information on the card is ignored.

This external subroutine master deck is optional input and, if submitted to SYNTBLE, must immediately follow the lexicon deck separated by one blank card.

cc 1-10 must be in ascending collating sequence.

Whenever SYNTBLE enters an external subroutine in the SUBJUMP TABLE, the name of the external subroutine is checked to ensure that it is in the master external subroutine deck.

SYNTBLE creates one punch file (SYNTAX ANALYSIS TABLE) of 6600 card output in the following format:

		Quarter 1		Quarter 2		Quarter 3		Quarter 4					
cc 1	11	20	21	25	26	30	31	35	36	40	41	63	72 80
blank *	CON	9	9999	9	9999	9	9999	9	9999	B	blank	xxxxxxx 99 blank	blank

*cc 1-10	Blank with the exception of the first card which is labeled "SYNTBLE."
cc 11-19	Contains the 6600 operation "DATA."
cc 20 cc 25 cc 30 cc 35	One-digit clue fields which contain one of the values 0, 2, 4, 5, 6, 7.
cc 21-24 cc 26-29 cc 31-34 cc 36-39	Four-digit parameters which represent one of the following seven options: 1. The syntax table octal address of a syntax table item name. 2. The diagnostic table octal address of a diagnostic message. 3. The attributes of a lexicon. 4. The subjump table octal address of an external subroutine. 5. The parameter "0000" with a 7 in corresponding clue field. 6. The parameter "0001" with a 7 in corresponding clue field. 7. The parameter "7777" with a 4 in corresponding clue field. 8. The parameter "0000" with a 0 in corresponding clue field.
cc 40	A "B".
cc 42-50	A COMMENT which is the name and ordinal of the syntax table item being outputted. SYNTBLE outputs two cards for every one input card, only the first output card contains the syntax table item and ordinal.

SYNTBLE creates a second punch file (SUBJUMP TABLE) of 6600 card output in the following format:

cc 1	11	20	30 80
blank *	JP	xxxxxxxxxx	blank

*cc 1-10	Blank with the exception of the first card which is labeled "SUBJUMP."
cc 11-19	Contains the 6600 operation "JP."
cc 20-29	The name of the external subroutine.
cc 30-80	Always blank.

GENERAL DESCRIPTION

SYNTBLE is a two-pass program. The first pass reads the card input and creates in core the syntax analysis name table, each entry of which is the name of a syntax table subset, and its table address. A maximum of 1000 subsets can be included in the table. If the need arises for more subsets, the table can be enlarged. The lexicon list is then read and a lexicon dictionary is created in core. A maximum of 175 lexicons can be included. If need arises, the table can be enlarged. The master external subroutine deck (if present) is then read and a dictionary is created in core. A maximum of 500 external subroutines can be included. This can also be enlarged, if necessary.

The syntax card input must precede the lexicon card input, separated by one blank card. The lexicon card input must precede the master external subroutine deck (if present), separated by one blank card.

Pass 1 copies the input onto tape in 80/80 cards image format which is read as input to Pass 2.

Pass 2 reads the tape input, scans the syntax input card images, creates two punch files as output, creates six or eight print files as output, and types on-line messages on the console.

Each input record is scanned as it is read from tape. Various computations and searches are performed and two 6600 format cards are outputted for each input record. As each input record is read, Pass 2 computes the syntax analysis table address of each syntax table item. The card is then scanned. If the syntax option is:

LEXICON	A 4 is moved into the first quarter clue field. A binary search is performed if the lexicon attributes are moved into the first quarter parameter field. If the lexicon is not found, 7777 is moved into the first quarter parameter field.*
SYNTAX TABLE ITEM NAME	A 6 is moved into the first quarter clue field. The syntax table octal address of the syntax table item name is moved into the first quarter parameter field.
EXTERNAL SUBROUTINE NAME	A 0 is moved into the first quarter clue field. The subjump table (table comprised of external subroutine names) octal address of the external subroutine is moved into the first quarter parameter field.
BLANK	Scan proceeds to the next input field. A diagnostic is issued.

The "no" action and "yes" action fields are scanned similarly. If any of these fields is:

DIAGNOSTIC	A 2 is moved to the next available quarter clue field. The 3-digit diagnostic number is converted to octal and this 4-digit diagnostic table octal address is moved into the next available quarter parameter field.
SYNTAX TABLE ITEM NAME	A 6 is moved into the next available quarter clue field. The syntax table octal address of the syntax table item name is moved into the next available quarter parameter field.

---

*This is a fatal error. A listing of unfound lexicons is printed. The person using SYNTBLE program should ensure that all lexicons submitted as input are entered in the lexicon table in Working Storage of the SYNTBLE program. This lexicon table must be in 6600 collating sequence.

EXTERNAL SUBROUTINE NAME	A 0 is moved into the next available quarter clue field. The subjump table octal address of the external subroutine is moved into the next available quarter parameter field.
BLANK	Scan proceeds to the next input field.

The "no" GO TO and "yes" GO TO fields are scanned similarly. If either of these fields is:

YES	A 7 is moved into the next available quarter clue field, and a 0001 is moved into the next available quarter parameter field.
NO	A 7 is moved into the next available quarter clue field, and a 0000 is moved into the next available quarter parameter field.
TWO-DIGIT NUMBER	A 5 is moved into the next available quarter clue field. This 2-digit ordinal represents the syntax table item name of the current subset. The octal address of this syntax table item name is computed and is moved into the next available quarter parameter field.
R2	A 7 is moved into the next available quarter clue field, and a 0002 is moved into the next available quarter parameter field.
BLANK	A diagnostic is issued. One of the two required output cards is punched with zero fill. Scan proceeds either to the next input field or to the next input card.

OUTPUT LISTINGS

SYNTBLE creates the following six or eight print files as output:

1. Listing of card syntax input.
2. Listing of card lexicon input.
3. Listing of card external subroutine master deck (if submitted).
4. Listing of punched SYNTAX ANALYSIS TABLE.

5. Listing of syntax option external subroutines which were not found in the syntax analysis name table.

6. Listing of lexicons which were not found in the lexicon table.

7. Listing of external subroutines which were entered in SUBJUMP TABLE but were not found in external subroutine master deck (if submitted).

8. Listing of punched SUBJUMP TABLE.

ERROR DIAGNOSTICS

1. "xxxxxxx 99 OUT OF ALPHABETIC SEQUENCE--CARD IGNORED"

   where:

   xxxxxxx 99 is the syntax table item name and its ordinal.

   This message is typed during Pass 1 if a syntax table item name is not in 6600 collating sequence. The card is ignored.

   Note that the ordinals must not only be in ascending numeric sequence, but also no ordinal may be omitted within a syntax table subset. This condition is not checked by the SYNTBLE program. If these ordinal specifications are not met, results are unpredictable.

2. "LEXICON xxxxxxxxx OUT OF ALPHABETIC SEQUENCE--CARD IGNORED."

   where:

   xxxxxxxxx is the name of the lexicon.

   This message is typed during Pass 1 if a lexicon name is not in 6600 collating sequence. The card is ignored. This will most likely result in a fatal unfound lexicon condition.

3. "SUBROUTINE xxxxxxxxxx OUT OF ALPHABETIC SEQUENCE--CARD IGNORED."

   where:

   xxxxxxxxxx is the name of the external subroutine.

This message is typed during Pass 1 if an external subroutine name on the master input deck is not in 6600 collating sequence. The card is ignored.

4. "xxxxxxx 99 BLANK cc 10-19--FIELD IGNORED."

   where:

   xxxxxxx 99 is the syntax table item name and its ordinal.

   This message is typed during Pass 2 if the syntax option field is found blank. The field is ignored and scan proceeds to the next field.

5. "xxxxxxx 99 UNRECOVERABLE PROGRAM ERROR--TERMINATE RUN"

   where:

   xxxxxxx 99 is the syntax table item name and its ordinal.

   This message is typed during Pass 2 if the syntax table item name cannot be found in the syntax analysis name table. This means that Pass 1 did not create the tape properly. Tape on Unit 184 should be dumped.

6. "xxxxxxx 99 FAULTY cc 36 or cc 73--FIELD IGNORED."

   where:

   xxxxxxx 99 is the syntax table item name and its ordinal.

   This message is typed during Pass 2 if cc 36 ("no" GO TO field) or cc 73 ("yes" GO TO field) consists of a blank or something other than the allowable options. The field is ignored.

END OF RUN

"End of Run" is typed at the end of execution of the SYNTBLE program.

APPENDIX C - COPY FROM COBOL SOURCE LIBRARY PROGRAM (COPYCL)

COPYCL is a modified version of the SCOPE 3.0 EDITSYM utility routine. COPYCL produces a random file whose logical records are those elements from the COBOL source library which may be referenced by COPY FROM LIBRARY or INCLUDE FROM LIBRARY and another file which is the index to the random file.

CALLING SEQUENCE

1. If the sixth option is missing or is other than the words EDIT or SELECT, COPYCL will enter none of the code added for COBOL libraries, but will function exactly as EDITSYM.
2. If the sixth option is the word EDIT, the following rules apply:
    a. The parameter OPL is required.
    b. The parameter NPL is optional.
    c. The parameter LIST is optional.
    d. The parameter COMPILE = file-name or C = file-name is required, and file-name becomes the name of the COBOL library file.
    e. Decks are placed in the COBOL library file when called for by *EDIT, dn cards.
    f. The only EDITSYM control cards allowed are *CATALOG, *DELETE, *INSERT, and *EDIT.
3. If the sixth option is the word SELECT, the following rules apply:
    a. The parameter OPL is required.
    b. The parameter NPL is not used.
    c. The parameter LIST is optional.
    d. The parameter COMPILE = file-name or C = file-name is required, and file-name becomes the name of the COBOL library file.

e. All decks from OPL with n = 4 are placed in the COBOL library file.

f. No EDITSYM control cards are allowed.

ELEMENTS CALLED

RANDMAK

OPERATION

COPYCL checks for the xith option. If not specified (by EDIT or SELECT) COPYCL bypasses all code added for COBOL libraries. Otherwise, all EDITSYM text decks (with index n = 4) on the old program library are placed on an internal compile file. Then RANDMAK is called to place all decks on the compile file, on a random file named by the C-parameter on the COPYCL control card. The flowchart for RANDMAK is given in Figure C-1.

An index file containing the disk address of the logical records is constructed by RANDMAK while writing the random file. The addresses are placed in the index relative to the hash indexes of the record names. The name of the index file is DDNxxxx, where xxxx is the first four characters of the random file name (whose default name is COLIB).

COBOL links to the random file and its index by the S-parameter.

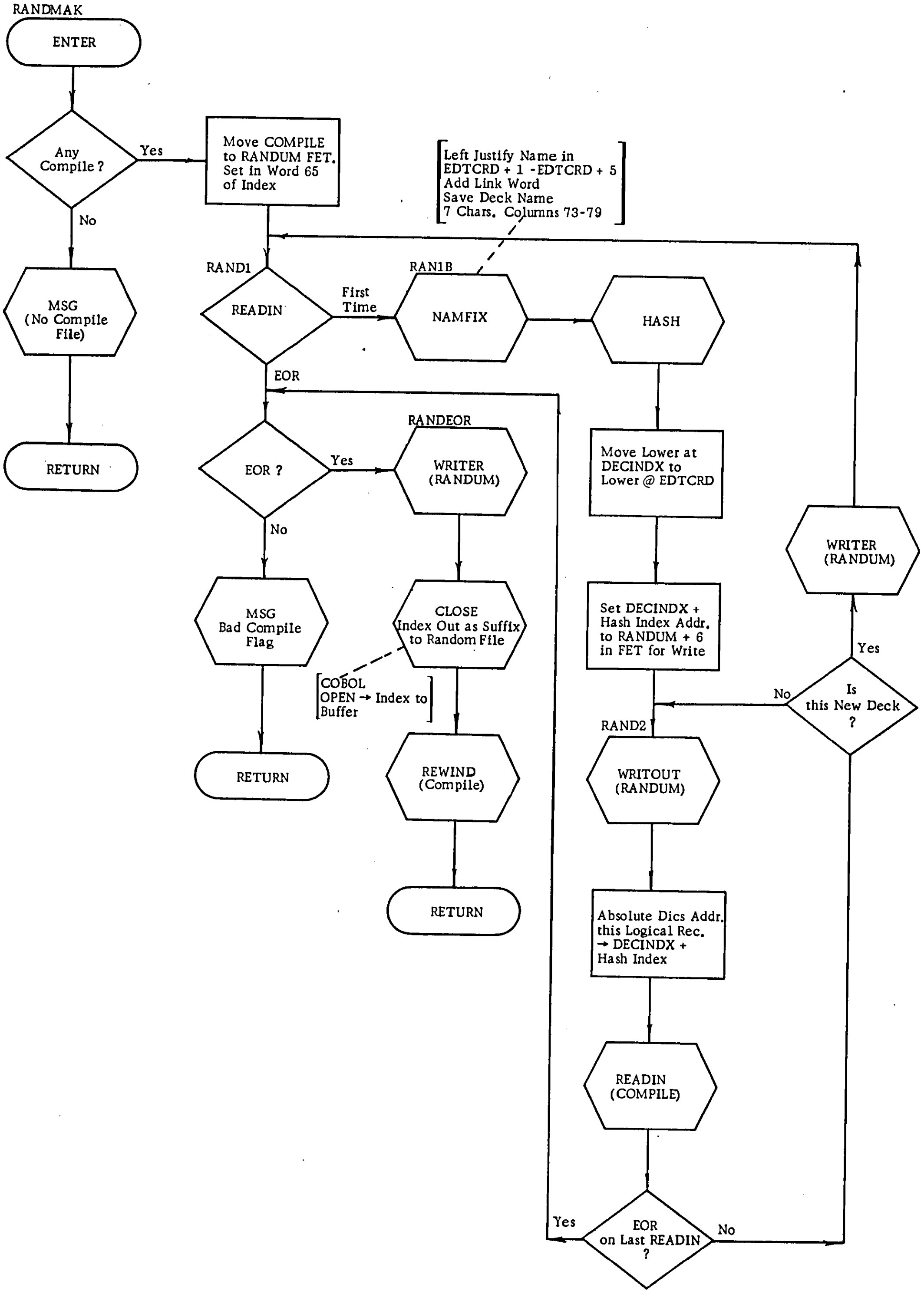


Figure C-1. RANDMAK Flowchart

APPENDIX D - LIST OF COMPILER DIAGNOSTICS

Diagnostic Number	Type*	Message
001	T	INCORRECT STARTING COLUMN BEFORE COLUMN 12
002	T	INCORRECT PUNCTUATION PRIOR TO WORD
003	T	ILLEGAL USE OF RESERSEL WORD-TREATED AS NAME
004	T	NO SPACE BEFORE LEFT PARENTHESIS
005	T	ILLEGAL SPACE BEFORE RIGHT PARENTHESIS
006	T	NO SPACE AFTER DECIMAL INDICATOR-ASSUMED PERIOD
007	E	INVALID CHARACTER(S) IN WORD-TREATED AS DATA-NAME
008	E	DATA-NAME, NUMERIC LITERAL OR PICTURE GREATER THAN MAXIMUM CF 30 CHARACTERS
009	E	NON-NUMERIC LITERAL GREATER THAN 255 CHARACTERS
010	E	) IS FIRST CHARACTER OF WORD
011	T	) NOT FOLLOWED BY SPACE
012	T	NO SPACE FOLLOWING PUNCTUATION
013	T	CHARACTER RESIDES HYPHEN IN COLUMN 7-ACCEPTED
014	E	MISSING QUOTE ON CONTINUATION CARD
015	T	CONTINUATION OF (OR) NOT ALLOWED
016	T	MISSING QUOTE AT END OF NON-NUMERIC LITERAL
017	T	FIRST WORD ON NEXT CARD SHOULD BEGIN IN COLUMN 8
018	T	SEQUENCE NUMBER IN COLUMNS 1-6 BREAKS
019	T	COMPILER NOTE -
020	T	COMPILER NOTE - BAD SKIPOPS
021	E	BAD CARD TERMINATOR (00)
022	T	RESERVED WORD USED IN UNAPPLICABLE DIVISION-TREATED AS NAME
023	T	COMPILER NOTE -
024	E	COMPILER NOTE-END CARD TERMINATION BAD NEXT CARD
025	E	BAD CALL TO DIAG - NUMBER BREATER THAN 1000
030	T	DECIMAL POINT SHOULD BE COMMA - SEE SPECIAL-NAMES
031	E	NO RECORD ON LIBRARY FOR NAME ITEM
032	E	REQUESTED ITEM NOT PART OF NAME RECORD
033	E	EQUIVALENT OR SMALLER LEVEL NUMBER FOUND BEFORE NAMED ITEM
034	E	LEVEL NUMBER NOT FIRST WORD OF ITEM DESCRIPTION
035	E	NESTED INCLUDES NOT PERMITTED

---

*C - Fatal error with no execution possible.
E - Serious error with probable execution of part of the object code.
T - Trivial error (precautionary message) (extended).
U - Non-DOD error condition (extended).
Z - Dummy.

Diagnostic Number	Type	Message
101	T	IDENTIFICATION HEADER NOT FIRST CARD OF PROGRAM
102	E	BAD WORD OR COMPILER OUT OF SYNCHRONIZATION
103	E	PROGRAM-ID NOT FIRST PARAGRAPH IN ID DIVISION
104	T	BAD OR MISSING PROGRAM NAME - ASSIGNED IDCOBOL
105	T	WORD -IS- IS MISSING
106	E	BAD SPECIAL NAMES CLAUSE - CLAUSE IGNORED
107	E	ILLEGAL USE OF RESERVED WORD FOR NAME
108	E	NO FILES SELECTED
109	E	NO OBJECT OR RENAMING CLAUSE
110	C	BAD OR MISSING ASSIGN OPTION - FATAL
111	T	WORD -REEL- OR -UNIT- IS MISSING - ASSUME REEL
112	T	WORD -SEQUENTIAL- IS MISSING BUT IS ASSUMED
113	T	BAD ACCESS CLAUSE - ASSUME SEQUENTIAL
114	E	BAD FILE-LIMIT CLAUSE - CLAUSE IGNORED
115	T	BAD RESERVE CLAUSE - ASSUME NO ALTERNATE AREA
116	E	BAD FILE-CONTROL PARAGRAPH
117	E	BAD ITEM ON LIBRARY
118	E	BAD RERUN CLAUSE - CLAUSE IGNORED
119	E	BAD SAME CLAUSE - CLAUSE IGNORED
120	T	WORD -FILE- IS MISSING BUT IS ASSUMED
121	E	BAD MULTIPLE FILE CLAUSE - CLAUSE IGNORED
122	E	ILLEGAL OBJECT OF ASSIGN OPTION - TRUNCATED TO 7 CHARACTERS
123	E	MISSING DIVISION HEADER - APPROPRIATE HEADER ASSUMED
124	E	FILE-NAME ALREADY SELECTED - CLAUSE IGNORED
125	T	MISSING DATA RECORD OR REPORT CLAUSE IN FILE DESCRIPTION
126	C	NOT ALL RENAMED FILES WERE SELECTED - FATAL
127	E	FILE-CONTROL PARAGRAPH IS MISSING
128	C	FILE-CONTROL PARAGRAPH OUT OF ORDER OR DUPLICATED - FATAL
129	T	SORT FILES ALWAYS SHARE THE SAME AREA
130	C	MISSING, MISPLACED OR DUPLICATED DIVISION - FATAL
131	E	ACTUAL AND SYMBOLIC KEY IN SAME FILE-CONTROL PARAGRAPH - SYMBOLIC KEY IGNORED
132	E	POINT LOCATION, LEAVING AND SIZE CLAUSES CONFLICT - EDITING IGNORED
133	C	OBJECT OR SUBJECT OF RENAMING IS ALSO SUBJECT OR OBJECT OF ANOTHER REMANING - FATAL
134	T	FILENAME IN MULTIPLE FILE CLAUSE NOT SELECTED
135	C	FILENAMES IN MULTIPLE FILE CLAUSE NOT ASSIGNED TO THE SAME IMPLEMENTOR-NAME - FATAL
136	Z	DUMMY
137	Z	DUMMY
138	C	LIMIT OF 53 FILES HAS BEEN EXCEEDED - FATAL
139	E	OCCURS DEPENDING ON CLAUSE ILLEGAL WITH RECORD.... DEPENDING ON OPTION IN THE FD - CLAUSE IGNORED

Diagnostic Number	Type	Message
140	T	VALUE OF CLAUSE IS MEANINGLESS WHEN LABEL RECORDS ARE OMITTED - VALUE OF CLAUSE IGNORED
141	E	ILLEGAL TO HAVE FILE DESCRIPTION FOR A FILENAME USED AS SUBJECT OF A RENAMING
142	T	A SELECTED FILENAME WHICH IS NOT THE SUBJECT OF A RENAMING DOES NOT HAVE AN FD, SD OR RD
143	E	MISSING OR BAD SECTION HEADER - FILE SECTION ASSUMED
144	E	MISSING OR BAD SECTION HEADER - WORKING-STORAGE SECTION ASSUMED
145	E	MISSING DATA RECORD CLAUSE IN SORT DESCRIPTION
146	E	CONFLICTING SIGN AND VALUE CHARACTERISTICS
200	E	FEATURE NOT IMPLEMENTED
201	T	WORD -DIVISION- MISSING ON HEADER
202	E	RENAMES WITH THRU OPTION ONLY LEGAL CLAUSE ON A 66 ITEM - ALL OTHER CLAUSES IGNORED
203	T	WORD -SECTION- MISSING ON HEADER
204	E	NUMBER OF 01 RECORD DESCRIPTIONS DOES NOT AGREE WITH DATA RECORDS CLAUSE
205	E	REPORT DESCRIPTION FOUND IN NON-REPORT SECTION
206	E	BAD LEVEL NUMBER - ITEM IGNORED
207	T	MISSING OR MISPLACED PERIOD
208	E	MISSING SWITCH STATUS CLAUSE - SWITCH NUMBER IGNORED
209	E	MISSING DIVISION HEADER
210	E	BAD RECORDING MODE CLAUSE - CLAUSE IGNORED
211	E	NO FILE DESCRIPTION PRECEDING RECORD
212	E	-THRU- OPTION IN VALUE CLAUSE IS ILLEGAL - OPTION IGNORED
213	E	PROCEDURE-NAME NOT PRECEDED BY HEADER
214	C	NO SELECT FOR FILE - FATAL
215	E	NO NAME FOLLOWING LEVEL NUMBER, FD, SD OR RD
216	E	BAD OR MISSING LABEL CLAUSE - ASSUME OMITTED
217	T	WORD -RECORD- IS MISSING FROM LABEL CLAUSE
218	T	NUMERIC LITERAL -ALL- IS ILLEGAL - ASSUME ALPHANUMERIC
219	E	BAD OR MISSING VALUE OF CLAUSE FOR STANDARD LABEL
220	E	BAD VALUE OF CLAUSE
221	T	WORD -OF- IS MISSING ON VALUE CLAUSE
222	T	WORD -RECORD- IS MISSING ON DATA RECORDS CLAUSE
223	E	BAD RECORD CONTAINS CLAUSE
224	T	WORD -TO- IS MISSING
225	T	WORD -RECORDS- IS MISSING FROM FILE CONTAINS CLAUSE
226	E	BAD FILE CONTAINS CLAUSE
227	T	BAD BLOCK CONTAINS CLAUSE
228	E	BAD NAME IN REDEFINES CLAUSE - CLAUSE IGNORED
229	T	CLAUSE NOT APPROPRIATE ON SORT FILE
230	E	NO NAME GIVEN IN COPY CLAUSE - ITEM IGNORED

Diagnostic Number	Type	Message
231	E	WORD FROM NOT FOLLOWED BY WORD LIBRARY - LIBRARY ASSUMED
232	E	WORD -OF- NOT FOLLOWED BY AN ACCEPTABLE NAME
233	E	NO CLAUSE ALLOWED FOLLOWING COPY CLAUSE
234	E	FIRST RECORD DESCRIPTION IS NOT NON-STANDARD DATA-NAME - ASSUME LABEL RECORDS OMITTED
235	E	INDICATED SIZE TOO LARGE - SET TO MAXIMUM OF 255
236	E	SIZE CLAUSE DOES NOT CONTAIN INTEGER
237	E	WORD LEFT OR RIGHT MISSING FROM POINT CLAUSE - LEFT ASSUMED
238	E	INDICATED POINT LOCATION TOO LARGE - SET TO MAXIMUM OF 31
239	E	POINT CLAUSE DOES NOT CONTAIN ACCEPTABLE INTEGER - CLAUSE IGNORED
240	T	WORD -RIGHT- OR -LEFT- MISPLACED
241	E	LEFT OR RIGHT MISSING FROM SYNCHRONIZED CLAUSE - LEFT ASSUMED
242	E	BAD USAGE CLAUSE - ASSUME DISPLAY
243	E	BAD CLASS CLAUSE - ASSUME ALPHANUMERIC
244	T	ITEM CAN NOT BE DESIGNATED AS JUSTIFIED LEFT - CLAUSE IGNORED
245	E	BAD JUSTIFIED CLAUSE
246	U	BWZ ACCEPTED AS BLANK WHEN ZERO
247	T	WORD -ZERO- ASSUMED FOR BLANK WHEN ZERO
248	E	BAD ZERO SUPPRESS CLAUSE OR MISPLACED WORD ZERO- - CLAUSE IGNORED
249	T	WORD -SIGN- IS MISSING FROM FLOAT CLAUSE BUT IS ASSUMED
250	E	WORD -LEAVING- NOT FOLLOWED BY ACCEPTABLE INTEGER - OPTION IGNORED
251	E	LEAVING INTEGER TOO LARGE - PRECEDING EDITING CLAUSE IGNORED
252	E	BAD DEPENDING ON NAME
253	E	WORD -OCCURS- NOT FOLLOWED BY ACCEPTABLE INTEGER - CLAUSE IGNORED
254	E	WORD -TO- NOT FOLLOWED BY ACCEPTABLE INTEGER - TO OPTION IGNORED
255	E	VALUE CLAUSE DOES NOT HAVE AN ACCEPTABLE LITERAL - CLAUSE IGNORED
256	E	WORD -THRU- NOT FOLLOWED BY A LITERAL
257	E	RENAMES ON A NON-66 LEVEL ITEM - ITEM IGNORED
258	E	NO NAME FOLLOWING WORD THRU OR RENAMES
259	E	MISPLACED COPY CLAUSE - ITEM IGNORED
260	E	WORD -THRU- NOT FOLLOWED BY ACCEPTABLF INTEGER - THRU OPTION IGNORED
261	T	BAD RANGE CLAUSE

Diagnostic Number	Type	Message
262	E	LEVEL NUMBER NOT APPROPRIATE IN THIS SECTION
263	E	SECTION DUPLICATED OR OUT OF ORDER
264	E	REPORT ELEMENT DESCRIPTION DOES NOT FOLLOW RD
265	C	REPORT NOT NAMED IN ANY FD DESCRIPTION - FATAL
266	E	INDICATED CODE CHARACTERS TRUNCATED TO THREE POSITIONS
267	E	NO ACCEPTABLE NAME FOLLOWING WORD CODE - CLAUSE IGNORED
268	E	BAD CONTROL CLAUSE
269	E	BAD PAGE CLAUSE
270	E	BAD HEADING CLAUSE
271	E	BAD FIRST DETAIL CLAUSE
272	E	BAD LAST DETAIL CLAUSE
273	E	BAD FOOTING CLAUSE
274	E	CLAUSE NOT APPROPRIATE IN REPORT SECTION
275	E	BAD TYPE CONTROL CLAUSE
276	E	BAD TYPE CLAUSE
277	T	-OF- ACCEPTED AS -OV- IN TYPE CLAUSE
278	E	BAD TYPE PAGE CLAUSE
279	E	BAD TYPE OVERFLOW CLAUSE
280	E	BAD TYPE REPORT CLAUSE
281	E	BAD SOURCE CLAUSE - ITEM IGNORED
282	E	BAD SUM CLAUSE - ITEM IGNORED
283	E	BAD NEXT GROUP CLAUSE
284	E	BAD LINE CLAUSE
285	E	BAD COLUMN CLAUSE
286	E	BAD RESET CLAUSE
287	E	USAGE IS ALWAYS DISPLAY IN REPORT DESCRIPTION
288	C	REPORT NAME NOT UNIQUE
289	E	AN ACCEPTABLE NAME DOES NOT FOLLOW LEVEL - ASSUME FILLER
290	T	RECORD NAME NOT LISTED IN DATA RECORDS CLAUSE
291	E	CONTROL NAME IN REPORT GROUP ITEM NOT LISTED IN RD
292	E	BAD SUBSCRIPT - ITEM IGNORED
293	E	LEVEL 77 NOT FIRST IN SECTION - ITEM IGNORED
294	E	LEVEL 77 IN FILE SECTION - ITEM IGNORED
295	E	CONDITION NAME UNDER ILLEGAL CONDITIONAL VARIABLE IS IGNORED
296	E	SECTION HEADER NOT FOLLOWED BY ACCEPTABLE LEVEL NUMBER
297	T	SIZE, CLASS, POINT LOCATION, SIGNED OR EDITING CLAUSES NUGATORY WITH PICTURE - PICTURE ACCEPTED
298	E	SIGNED CLAUSE ILLEGAL WITH EDITING - SIGNED CLAUSE IGNORED
299	E	CONFLICTING EDITING AND USAGE CLAUSES - ASSUME USAGE DISPLAY
300	T	EDITING ILLEGAL ON FILLER ITEM - CLAUSE IGNORED
301	T	JUSTIFIED CLAUSE ILLEGAL ON NUMERIC ITEM - CLAUSE IGNORED
302	E	USAGE COMPUTATIONAL REQUIRES NUMERIC CLASS - ASSUME NUMERIC

Diagnostic Number	Type	Message
303	Z	DUMMY
304	E	VALUE ON A REDEFINES OR OCCURS CLAUSE - VALUE IGNORED
305	E	REDEFINES CLAUSE ILLEGAL ON FILLER ITEM - REDEFINES IGNORED
306	E	OCCURS CLAUSE ILLEGAL ON FILLER ITEM - OCCURS IGNORED
307	E	FILLER ILLEGAL ON 77 AND 88 - ITEM IGNORED
308	E	SIZE CLAUSE ILLEGAL ON 88 ITEM - SIZE IGNORED
309	E	OCCURS CLAUSE ILLEGAL ON 88 ITEM - OCCURS IGNORED
310	E	USAGE CLAUSE ILLEGAL ON 88 ITEM - USAGE IGNORED
311	E	REDEFINES CLAUSE ILLEGAL ON 88 ITEM - REDEFINES IGNORED
312	E	EDITING CLAUSE ILLEGAL ON 88 ITEM - EDITING IGNORED
313	E	VALUE CLAUSE ON 88 ITEM IS MISSING - ITEM IGNORED
314	E	PICTURE CLAUSE ILLEGAL ON 88 ITEM - PICTURE IGNORED
315	E	OCCURS CLAUSE ILLEGAL ON 77 ITEM - OCCURS IGNORED
316	E	VALUE CLAUSE ON 77 ITEM IN CONSTANT SECTION IS MISSING - ITEM IGNORED
317	E	LEVEL 66 IS NOT A RENAMES ITEM - ITEM IGNORED
318	Z	DUMMY
319	E	AN ACCEPTABLE NAME DOES NOT FOLLOW LEVEL 77 OR 88 - ITEM IGNORED
320	E	SIGNED CLAUSE ILLEGAL ON GROUP ITEM - SIGNED CLAUSE IGNORED
321	E	POINT LOCATION CLAUSE ILLEGAL ON GROUP ITEM - POINT LOCATION IGNORED
322	E	EDITING AND PICTURE CLAUSES ILLEGAL ON GROUP ITEM - CLAUSE IGNORED
323	E	JUSTIFIED CLAUSE ILLEGAL ON GROUP ITEM - CLAUSE IGNORED
324	E	SYNCHRONIZED CLAUSE ILLEGAL ON GROUP ITEM - CLAUSE IGNORED
325	E	LEVEL 66 NOT LAST ITEM IN RECORD - ITEM IGNORED
326	E	FIRST ITEM IN FILE NOT AN 01 - ASSUME 01
327	T	QUALIFYING NAME NOT PRECEDED BY -OF- OR -IN-
328	E	RENAMES CANNOT BE ASSOCIATED WITH OTHER CLAUSES - ITEM IGNORED
329	E	CONFLICTING CLASS CLAUSE AND EDITING CLAUSE - ASSUME CLASS NUMERIC
330	E	RENAMES ON NON-66 - ITEM IGNORED
331	T	REDEFINES MEANINGLESS ON AN 01 IN FILE SECTION
332	E	DEPENDING ON CLAUSE NOT ALLOWED ON SORT FILE - CLAUSE IGNORED
333	E	BOTH REPORT CLAUSE AND DATA RECORDS CLAUSE ON SAME FD
334	T	COPY CLAUSE DOES NOT END WITH PERIOD - SKIP TO NEXT CARD
335	T	REQUIRED COMMA IS MISSING BUT IS ASSUMED
336	E	SELECTED OPTION CAN ONLY BE ON GROUP ITEM - OPTION IGNORED
337	E	NO TYPE CLAUSE ON REPORT GROUP - ASSUME DETAIL
338	T	DETAIL REPORT GROUP NOT NAMED

Diagnostic Number	Type	Message
339	E	TYPE CLAUSE ON NON-REPORT GROUP IGNORED
340	E	SUM CLAUSE CAN ONLY APPEAR WITHIN CONTROL FOOTING - ITEM IGNORED
341	E	RESET CAN ONLY APPEAR ON A SUM ITEM - CLAUSE IGNORED
342	E	NEXT GROUP CLAUSE CAN ONLY APPEAR ON A REPORT GROUP - CLAUSE IGNORED
343	E	ITEM WITH COLUMN NUMBER MUST HAVE PICTURE OR SIZE - CLAUSE IGNORED
344	E	COLUMN NUMBER OVERLAPS PREVIOUS ITEM - ITEM IGNORED
345	E	ITEM WILL PRINT BEYOND END OF LINE
346	E	GROUP INDICATE MUST ONLY APPEAR ON ELEMENTARY ITEM WITHIN DETAIL REPORT GROUP
347	E	SOURCE, SUM, OR VALUE CLAUSE NOT APPLICABLE ON GROUP ITEM - IGNORED
348	E	COLUMN NUMBER NOT APPROPRIATE ON GROUP ITEM - CLAUSE IGNORED
349	E	SOURCE, SUM OR VALUE CLAUSE WITHIN SOURCE SELECTED GROUP
350	E	SOURCE, SUM OR VALUE CLAUSE NOT IMMEDIATELY PRECEDED BY SELECTED GROUP - ITEM IGNORED
351	E	FD NOT FOLLOWED BY AN 01 OR AN FD - ITEM IGNORED
352	E	88 ILLEGAL AS A GROUP ITEM - ITEM IGNORED
353	E	88 DOES NOT FOLLOW A LEGAL LEVEL NO - ITEM IGNORED
354	E	66 ILLEGAL AS A GROUP ITEM - ITEM IGNORED
355	E	LEVEL NO DOES NOT FOLLOW LOGICAL RECORD
356	T	SIZE CLAUSE CONTAINS NEGATIVE INTEGER - SIGN IGNORED
357	E	REPEAT COUNT IN PARENTHESES EXCEEDS MAXIMUM - SET TO MAXIMUM OF 255
358	E	ILLEGAL CHARACTER IN PICTURE - FILLED WITH DUMMY 9, A OR X
359	E	SYNTACTICAL ERROR IN PICTURE - PICTURE DROPPED
360	E	NON-NUMERIC WITHIN PARENTHESES - PICTURE DROPPED
361	E	ILLEGAL IMPLEMENTOR NAME - TRUNCATED TO 7 CHARACTERS
362	E	66 DOES NOT FOLLOW LEGAL LEVEL NUMBER - ITEM IGNORED
363	E	NO SYMBOLIC OR ACTUAL KEY SPECIFIED FOR RANDOM ACCESS - CLAUSE IGNORED
364	E	MUTUALLY EXCLUSIVE EDITING CLAUSES ON SAME ITEM - FIRST ONE ACCEPTED

Diagnostic Number	Type	Message
500	E	IMPROPER DIVISION HEADER
501	E	IMPROPER DIVISION HEADER
502	T	PROCEDURE NAME MISSING
503	T	SECTION NAME MISSING
504	E	PREMATURE END OF PROGRAM
505	E	MISPLACED END
506	C	THIS ERROR SHOULD NOT OCCUR IN FINISHED COMPILER
507	E	IMPROPER SECTION HEADER
508	E	SECTION NAME MISSING
509	E	SECTION NAME MISSING
510	C	PREMATURE END OF PROGRAM
511	E	IMPROPER END DECLARATIVES
512	E	IMPROPER END DECLARATIVES
513	E	PERIOD MISSING
515	E	PERIOD MISSING
516	T	NULL SECTION
517	T	NULL SECTION
518	T	NULL PARAGRAPH
519	E	PERIOD MISSING
520	E	PERIOD MISSING
521	E	PROCEDURE NAME MISSING
522	E	REPLACEMENT PAIR MISSING
523	E	PERIOD MISSING
524	E	SYNTAX ERROR, KEYWORD EXPECTED AND NOT FOUND OR PERIOD MISSING
525	E	PREMATURE END OF PROGRAM
526	E	PROCEDURE NAME MISSING
527	E	PROCEDURE NAME MISSING
528	E	CONDITIONAL STATEMENT NOT ALLOWED WITH AT END, ON SIZE OR INVALID KEY
529	E	PERIOD OR ELSE MISSING
530	E	VARYING CLAUSE BAD
531	E	VARYING CLAUSE BAD
532	E	VARYING CLAUSE BAD
533	E	VARYING CLAUSE BAD
534	E	VARYING CLAUSE BAD
535	E	VARYING CLAUSE BAD
536	E	PROCEED CLAUSE BAD
537	E	PROCEED CLAUSE BAD
538	E	PROCEED CLAUSE BAD
539	E	PROCEED CLAUSE BAD
540	E	WITH NO REWIND BAD
541	E	WITH NO REWIND BAD
542	E	INPUT CLAUSE BAD
543	E	OUTPUT CLAUSE BAD

Diagnostic Number	Type	Message
544	E	FILE NAME MISSING
545	E	WITH NO REWIND BAD
546	E	WITH NO REWIND BAD
547	E	IDENTIFIER MISSING
548	E	ASCENDING OR DESCENDING MISSING
549	E	IDENTIFIER MISSING
550	E	FROM OR EQUALS MISSING
551	E	IDENTIFIER MISSING
552	E	TO MISSING
553	E	RESULT MISSING
554	E	IDENTIFIER MISSING
555	E	MNEMONIC NAME UNRECOGNIZABLE
556	E	PROCEED TO CLAUSE MISSING
557	E	FILE NAME MISSING
558	E	MNEMONIC NAME UNRECOGNIZABLE
559	E	CONDITION UNRECOGNIZABLE
560	E	TRUE STATEMENTS MISSING
561	E	SENTENCE MISSING FROM NEXT SENTENCE
562	E	STATEMENTS APPEAR AFTER NEXT SENTENCE
563	E	FALSE STATEMENTS MISSING
564	E	SENTENCE MISSING FROM NEXT SENTENCE
565	E	PARAGRAPH NAME MISSING AFTER PERFORM
566	E	PARAGRAPH NAME MISSING AFTER THRU
567	E	TIMES MISSING
568	E	CONDITION MISSING AFTER UNTIL
569	E	BAD VARYING CLAUSE
570	E	BAD VARYING CLAUSE
571	E	BAD VARYING CLAUSE
572	E	CLAUSE MISSING AFTER WITH
573	E	CLAUSE MISSING AFTER WITH
574	E	NO REWIND MISSING
575	E	INFORMATION AFTER OPERATIONAL SIGN UNINTELLIGIBLE
576	E	BADLY POSITIONED SIGN
577	E	BAD QUALIFIER
578	E	ALPHA LITERAL MISSING AFTER ALL
579	E	OBJECT OF CONDITION MISSING
580	E	SUBSCRIPT TERM MISSING
581	E	SECOND SUBSCRIPT TERM MISSING
582	E	THIRD SUBSCRIPT TERM MISSING
583	E	RIGHT) MISSING
584	E	BAD STATEMENTS IN BRANCH OF IF VERBS
585	E	BAD TERM AFTER * OR /
586	E	BAD EXPONENT
587	E	EXTRA RIGHT)
588	E	MISSING RIGHT)

Diagnostic Number	Type	Message
589	E	BAD ABBREVIATION TYPE 3
590	E	BAD ABBREVIATION TYPE 3
591	E	CONDITION MISSING
592	E	RIGHT) MISSING
593	E	REPORT GROUP MISSING FROM GENERATE STATEMENT
594	E	ONE STATEMENT PARAGRAPH CONTAINS MORE THAN ONE SENTENCE
595	E	AND NOT FOLLOWED BY OBJECT
596	E	PROCEDURE NAME EXPECTED HERE OR KEYWORD IS MISSPELLED
597	E	AT END OR INVALID KEY CLAUSE IN READ IS MISSING
598	E	AT END CLAUSE IN RETURN IS MISSING
599	E	APPARENT ILLEGAL ABBREVIATION
600	E	IDENTIFIER MISSING
602	E	INCORRECT SYNTAX IN MULTIPLY
604	E	IDENTIFIER MISSING
605	E	GIVING MISSING
606	U	NON DOD COBOL
607	E	ARITHMETIC EXPRESSION IS BAD
611	E	INTO OR BY MISSING
613	E	RESULT IDENTIFIER MISSING
616	E	PROCEDURE NAME MISSING
617	E	SUBROUTINE NAME MISSING
619	E	INCORRECT SYNTAX IN EXAMINE
620	E	IMPROPER LITERAL
621	E	INCORRECT SYNTAX IN GO TO
623	E	INCORRECT -REPLACING BY- CLAUSE
625	E	INCORRECT SYNTAX IN MOVE
626	E	INPUT OR OUTPUT MISSING IN OPEN
627	E	STATEMENTS IN AT END, INVALID KEY OR ON SIZE ERROR CLAUSE ARE MISSING OR CONDITIONAL
628	E	FILE NAME MISSING
631	E	SORT KEY MISSING
632	E	INCORRECT SYNTAX IN SORT
633	E	INCORRECT SYNTAX IN STOP STATEMENT
634	E	INCORRECT SYNTAX IN SUBTRACT
635	U	SUBTRACT CORRESPONDING FROM DATA NAME SERIES IS NOT DOD
636	E	INCORRECT SYNTAX IN USE
637	E	RECORD NAME MISSING
640	E	INCORRECT SYNTAX IN WRITE
641	E	INCORRECT SYNTAX IN TERMINATE
642	E	WORDS SKIPPED BECAUSE OF INCORRECT SYNTAX
643	E	CONDITION NAME USED IMPROPERLY IN A FORMULA
650	E	ILLEGAL SUBSCRIPT WITHIN FORMULA
651	E	ILLEGAL VARIABLE SUBSCRIPT
656	E	ILLEGAL NUMERIC SUBSCRIPT
658	E	QUALIFICATION EXCEEDS BUFFER LIMITATIONS

Diagnostic Number	Type	
659	E	ILLEGAL ENTER PARAMETER
660	C	PROCEDURE DIVISION TABLES EXCEEDED -- FATAL
663	E	ACCEPT FROM OUTPUT FILE
664	E	DISPLAY UPON INPUT FILE
665	E	MORE THAN 24 USE STATEMENTS APPLICABLE TO FILE
666	E	MORE THAN 255 USE STATEMENTS
667	E	I-O VERB WITHIN DECLARATIVE SECTION
668	E	ILLEGAL CONDITIONAL STATEMENT
669	E	DATA ITEM WITHIN GENERATE IS NOT A REPORT GROUP
670	E	CONDITION-NAME ILLEGALLY USED
671	E	DATA ITEM IN USE BEFORE REPORTING IS NOT REPORT GROUP
674	E	IDENTIFIER REFERENCED MULTIPLY DEFINED OR IMPROPERLY QUALIFIED
675	E	IDENTIFIER REFERENCED UNDEFINED OR IMPROPERLY QUALIFIED
676	E	IMPROPER IDENTIFIER USED WITH CORRESPONDING
678	E	NO CORRESPONDING ITEMS FOUND BETWEEN IDENTIFIERS
679	E	NO PROCEDURE NAME AFTER OF - INCLUDE FROM LIBRARY
680	E	PERIOD MISSING
681	E	MULTIPLY DEFINED PROCEDURE NAME
682	E	IDENTIFIER REQUIRES SUBSCRIPTS - FIRST OCCURRENCE ASSUMED
683	E	INVALID KEY CLAUSE NOT APPLICABLE TO SEQUENTIAL FILE
684	E	UNQUALIFIED REFERENCE TO ITEM IN RENAMED FILE
685	E	DIG ERROR2
686	E	FILE NAME ASSOCIATED WITH SEEK HAS NO KEY INFORMATION
687	C	-FATAL- OVERFLOW OF PNT INTO DNT- INCREASE FIELD LENGTH TO ENLARGE TABLE AREA
801	E	ALPHABETIC OR AN CLASS IN A NUMERIC FETCH
802	E	SIZE OF A NUMERIC ITEM GREATER THAN 18 USAGE NOT NUMERIC
803		IN ARITHMETIC
804	E	ALPHA FIGURATIVE CONSTANT IN ARITHMETIC ON SIZE ERROR
805		CANNOT OCCUR, TEST OMITTED
807	T	ABSOLUTE VALUE OF SIGNED RESULT STORED IN UNSIGNED FIELD
808	T	COMPOSITE SIZE EXCEEDS 18
809	T	LEFT TRUNCATION POSSIBLE IN COMPUTATIONAL STORE
810	E	TEMPORARY STORAGE AREA EXCEEDED IN COMPUTE
811	T	SIZE OF COMPUTATIONAL-1 RESULT CAN BE GREATER THAN THE SIZE OF A RECEIVING FIELD
812	T	ON SIZE ERROR CANNOT OCCUR, TEST OMITTED
813	T	COMPOSITE SIZE EXCEEDS 18
814	T	SPECIFIED ROUNDING CANNOT OCCUR
815	E	COMPILER ILL, OR FEATURE NOT YET IMPLEMENTED
816	E	MORE INSERTION CHARACTERS THAN FIELD SIZE
817	T	INVALID ITEM MOVED TO AN ALPHA FIELD
818	E	COMPUTATIONAL-N ITEM MOVED TO AN ALPHA FIELD
819	T	NON-INTEGER NUMERIC ITEM MOVED TO AN AN FIELD

Diagnostic Number	Type	
821	E	AN ETC MOVED TO COMPUTATIONAL-N FIELD
822	T	AN ETC MOVED TO NUMERIC FIELD
823	E	AN ETC MOVED TO AN EDIT(NUM)FIELD
824	E	EDIT(NUM) MOVED TO A NUMERIC FIELD
825	T	ITEM MOVED TO OR FROM MIXED OR BINARY GROUP
827	C	TOO MANY LOCAL LABELS
828	T	TRUNCATION IN MOVE
829	E	ILLEGAL COMPARISON - CLASSES NOT COMPATABLE OR COMPARISON IS PREDETERMINED
830	T	NON-NUMERIC TO NUMERIC COMPARISON
831	E	ILLEGAL CLASS TEST - INAPPLICABLE TO NON-NUMERIC FIELD
832	E	ILLEGAL CLASS TEST - INAPPLICABLE TO COMPUTATIONAL-1 OR COMPUTATIONAL-2
833	C	COMPLEXITY OF CONDITIONALS EXCEEDS COMPILER LIMITATIONS
834	E	STORE INTO LITERAL, OR FIGURATIVE CONSTANT
835	T	ITEM TRUNCATED TO INTEGER
836	E	INTEGER ITEM LARGER THAN 14 CHARACTERS
837	E	ALPHABETIC FIGURATIVE CONSTANT MOVED TO A NUMERIC FIELD
838	E	MOVE TO AN EDITED FIELD WHICH IS A GROUP IS ILLEGAL AND IS SUPPRESSED
839	E	SUBSCRIPTED ITEM HAS INSUFFICIENT OR MISSING OCCURS
840	T	LITERAL EXCEEDS 18 DIGITS - MOST SIGNIFICANT DIGITS USED
841	T	LITERAL EXCEEDS SIZE OF NON-NUMERIC RECEIVING FIELD - TRUNCATED ON RIGHT

CSC

COMPUTER SCIENCES CORPORATION

- LOS ANGELES
- SAN FRANCISCO
- HOUSTON
- WASHINGTON, D.C.
- RICHLAND, WASHINGTON
- SAN DIEGO
- NEW YORK
- HUNTSVILLE