AD-783 407

MACHINE INDEPENDENT DATA MANAGEMENT SYSTEM (MIDMS) SYSTEM SPECIFICATIONS. CHAPTER 3 - FILE MAINTENANCE

Defense Intelligence Agency
Washington, D. C.

1 July 1974

AD783407

1. REPORT NUMBER | 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
Machine Independent Data Management System (MIDMS) System Specifications
CHAPTER 3 - File Maintenance

5. TYPE OF REPORT & PERIOD COVERED
User Documentation

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
Defense Intelligence Agency (DIA)

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Defense Intelligence Agency
ATTN: Information Processing Division (DS-5)
Washington, D.C. 20301

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS
Same as 9

12. REPORT DATE
1 July 1974

13. NUMBER OF PAGES
584

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

15. SECURITY CLASS. (of this report)
UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)
Approved for public release; unlimited distribution

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

MIDMS	Subsystem	Fixed Set
File Structuring	Subroutine	Variable Set
Librarian	Retrieval	Module
File Maintenance	Output	
Language Processor	Periodic Set	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
This document describes the MIDMS system specifications. Each module of the system is described in detail to include its subsystem structure, subprogram identification and description, flow charts, and error messages. Differences between the IBM and Honeywell versions of MIDMS are documented.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

MACHINE INDEPENDENT DATA MANAGEMENT SYSTEM

(MIDMS)

SYSTEM SPECIFICATIONS

FOREWORD

This manual has been developed by the Defense Intelligence Agency (DIA) for technical purposes. It does not reflect either explicitly or implicitly official DIA policy or intelligence matters. Its purpose is to provide a detailed description of the Machine Independent Data Management System (MIDMS) programs. DIA assumes no system installation, program maintenance, or system operation responsibility, nor is DIA responsible for the data upon which the system operates.

TABLE OF CONTENTS

PAGE

Enclosures A, B, C + D - AD-783409

CHAPTER 3

Overview

File Maintenance

1. GENERAL. File Maintenance (FM) is that portion of MIDMS that handles the validation of input data and the production and modification of MIDMS data file records. Figure 3-1 illustrates the general structure of the FM subsystem which consists of ten separately produced load modules. Eight of the load modules are dynamically overlaid during execution of the FM process.

a. The FM load module acts as File Maintenance Supervisor and causes the loading of the FMIO load module and the dynamic loading and execution of the FMLP, FMIP, FMIPSRT, FMMP, and FMMPSRT load modules.

b. The FMIO load module performs all of the card reading and line printing activities required by the other load modules and punches cards upon request from a logic package. The entry point address of FMIO is passed to the other load modules by the FM Supervisor.

c. The FMIP load module comprises the Language Processor portion of FM. It transforms user-written Logic Packages into COBOL programs for Logical Maintenance and Data Source Descriptions (DSD's) into Data Source Tables (DST's) for Ordinary Maintenance.

d. The FMIP load module is the Input Processor portion of Ordinary Maintenance which validates source data in accordance with previously producted Data Source Tables and generates a transaction file of validated data.

e. The FMIPSRT load module sorts the transaction file produced by FMIP so that the transactions can be merged with the MIDMS data file during Maintenance Proper. FMIPSRT is not executed when FMIP determines that the transaction file is already in the proper sort.

f. The FMMP load module is the supervisor to the Maintenance Proper portion of File Maintenance which controls the actual update of a MIDMS file based on the data and control information in the transaction file (for Ordinary Maintenance) and/or the single file Logic Package programs (for Logical Maintenance). Records with changed ID's and out of sort records are placed in a separate output file. FMMP causes the loading of FMMPCD, OMMP, and LMMP load modules and executes them, as necessary.

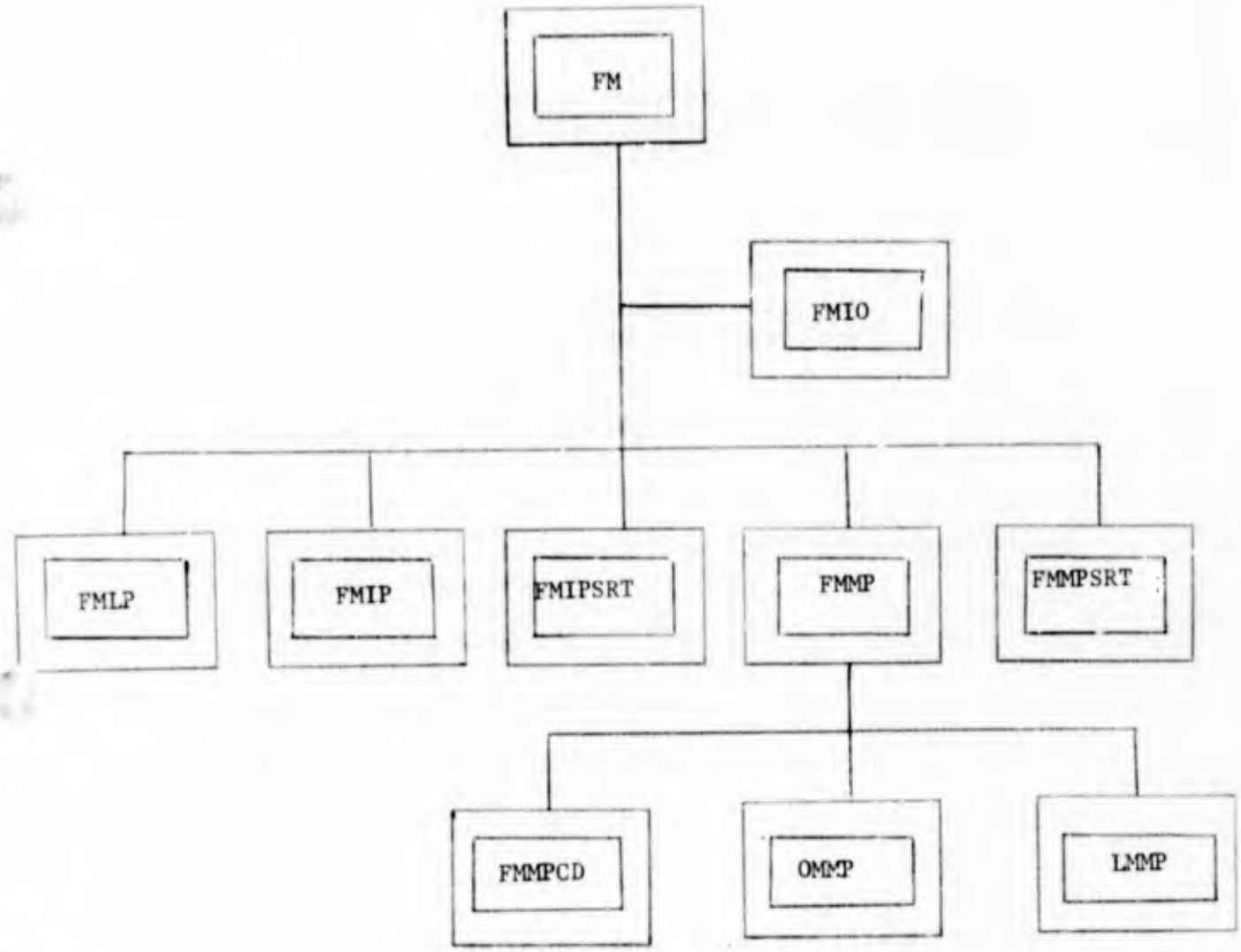


Figure 3-1. File Maintenance Structure (General)

g. The FMMPSRT load module sorts out of sort records and merges the sorted file with the balance of the updated MIDMS data file. FMMPSRT is not executed if FMMP determines that no records are out of sort.

h. Figure 3-2 shows the detailed structure of the FM load module as well as a first level breakdown of the FMLP, FMIP and FMMP load modules indicating subroutines which appear repeatedly in File Maintenance. The detailed structure of FMLP, FMIP, and FMMP will be described separately.

2. FILE MAINTENANCE (FM) SUBPROGRAMS.

a. FM

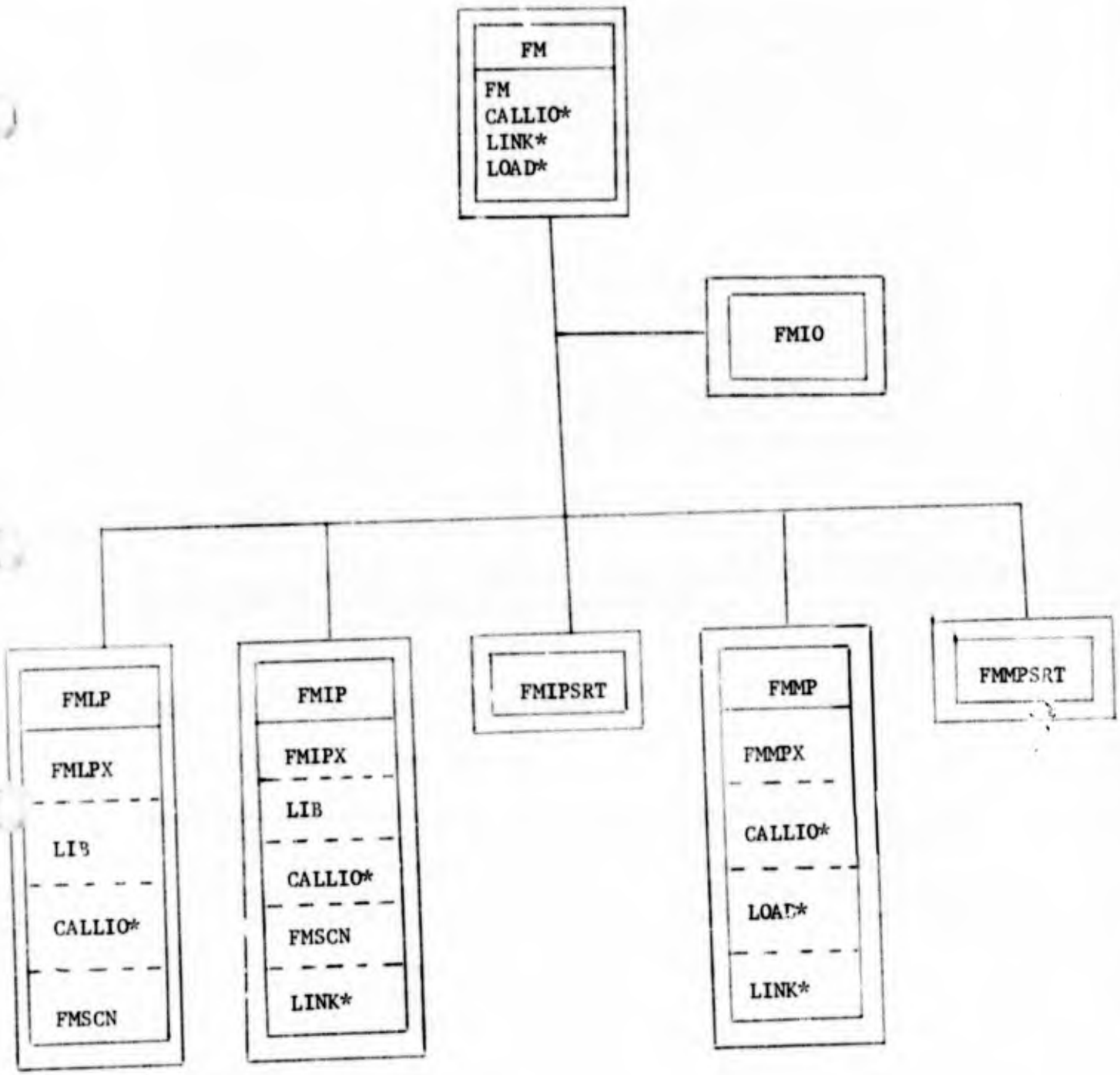
(1) Function. To control the overall processing of the File Maintenance subsystem.

(2) Calling Sequence.

```
CALL 'LOAD'   USING ENTRY-NAME FMIO.
CALL 'FMSET'  USING FMIO.
CALL 'FMCRD'  USING FM-INPUT END-SW.
CALL 'FMPRT'  USING PRT-LINE CC.
CALL 'LINK'   USING ENTRY-NAME FM-DD.
```

(3) Program Description.

FM is the first program to be executed for any File Maintenance process. This routine, the FM Supervisor, causes the loading of the FMIO load module, the reading of the first FM control card, and the dynamic loading and execution of the Language Processor (FMLP), Input Processor (FMIP), Input Processor Sort (FMIPSRT), Maintenance Proper (FMMP), and Maintenance Proper Sort (FMMPSRT) load modules. To load the FMIO load module, FM uses the LOAD subroutine supplying the entry point name (FMIO) and an area to hold the entry point address of the FMIO routine. This address is provided to the CALLIO subroutine through its FMSET entry point so that future calls to CALLIO via its FMCRD and FMPRT entry points will result in branching to FMIO to read a card or print a line. After setting the FMIO address in CALLIO, FMCRD is called to read a card and FMPRT is called to print it. If the card contains the proper identification for a File Maintenance control card (FMLP, FMIP, or FMMP in columns 1-4), the appropriate load module entry point name is set up and LINK is called to load and execute the programs necessary to perform the detailed processing of the user-specified actions. A comment card (* in column 1) causes the reading of another card. Any other cards are errors which



* ALC SUBROUTINE

Figure 3-2. File Maintenance Structure (First Level)

result in the printing of an ILLEGAL CARD TYPE message through FMPRT. Upon completion of FMIP and FMMP processing, the setting of FM-LINK-SW determines if a sort is required. If so, the sort's load module entry point name (FMIPSRT or XMPSRT) is set up and LINK is called to load and execute the sort activities. Upon return to FM after completion of a module, any additional FM control card is read and acted upon.

The Working-Storage Section of the FM program is stored in the COBOL COPY library under the name of FMDDV and contains a collection of fields widely used throughout File Maintenance as well as those used by the FM program itself. FMDD on the COPY library is a name by name, character by character duplicate of FMDDV used in the Linkage Section of almost every subroutine within File Maintenance. The only difference between FMDDV and FMDD is that FMDDV contains VALUE clauses to initialize certain fields for FM processing. FM-DD is the gorup name for both FMDDV and FMdd fields.

b. FMIOX

(1) Function. To read and print lines for all File Maintenance programs and to punch cards for Logic Packages.

(2) Calling sequence.

```
ENTRY 'FMIO' USING FMIO-SW LINE-IMAGE CC.
```

(3) Program Description.

The FMIOX subroutine is contained within a load module with a name of FMIOX and an alias for its entry point of FMIO. It is loaded under the direction of the File Maintenance Supervisor, FM, through the use of the ALC subroutine LOAD. Execution is invoked through the CALLIO subroutine which accepts calls in the form of

```
        CALL 'FMCRD' USING FM-INPUT END-SW
        CALL 'FMPRT' USING PRT-LINE CC
or      CALL 'FMPUN' USING PRT-LINE
```

and transforms them to the ALC equivalent of

```
        CALL 'FMIO' USING FMIO-SW LINE-IMAGE CC
```

after setting FMIO-SW to 1 for card reading, 2 for printing, or 3 for punching. FMIOX tests FMIO-SW to determine whether card reading, printing, or punching is required. After reading a card into CARD-IMAGE (the first 80 positions of LINE-IMAGE), FMIOX checks the first four characters of the card and sets END-SW

(which is the same field as CC from the calling sequence) to one of five possible values:

0 = ordinary card, not an FM control card
1 = end of file, no more cards available
2 = FMLP control card
3 = FMIP control card
4 = FMMP control card

Before printing a line, CC is checked for one of five characters, three of which are for carriage control:

b = one line
0 = two lines
1 = next page
* = close print file
C = classification line

The carriage control characters b, 0 and 1 are the usual ones for IBM 360 printing and are used directly by the 360 COBOL WRITE statement (they cannot generally be used directly by other machine's COBOL compilers). When the number of lines reaches 52, a new page is started without regard to the carriage control character. When 'C' is specified for CC, a classification header and trailer is printed with a date on every page. Page numbering is always printed.

c. FMSCN

(1) Function. To examine the characters in a control statement and break the character string into words which can be easily analyzed by appropriate routines.

(2) Calling Sequence.

```
ENTRY 'FMSCAN'   USING FM-DD.
ENTRY 'FMSCAN2'  USING FM-DD.
```

(3) Program Description.

FMSCN examines the characters in an FM control statement or an Ordinary Maintenance Language Processor (OMLP) Data Source Description (DSD) statement and breaks the characters into words of equal length. These words can easily be analyzed by the calling subroutine because the words are placed in a table (FM-WORD-AREA). A "word," as the term is used here, is a series of consecutive characters terminated by a blank (b), comma (,), hyphen (-), or column 78 of a card image. Following the blank or comma that terminates a word may be an arbitrary number of additional blanks not

considered to be part of any word. The word after a hyphen always begins with the next character. A word can have "zero length" when two consecutive terminating characters are specified in the statement (excluding blank-blank and comma-blank pairs). A literal may be found after a blank or a comma. It is initiated by a 4-8 or 5-8 punch character (* or '). The literal continues until a 4-8 or 5-8 character is followed by a word terminator character or column 78 of a card.

The word table (FM-WORD-AREA) is described in the COBOL Data Division (FMDD) as follows:

```
02  FM-WORD-AREA.
  03  FM-WORD-COUNT          PICTURE 999 COMPUTATIONAL.
  03  FM-DATA-WORDS.
    04  FM-DATA-ITEM OCCURS 50 TIMES.
      05  FM-CHAR-COUNT      PICTURE 99 COMPUTATIONAL.
      05  FM-LIT-POINT       PICTURE 999 COMPUTATIONAL.
      05  FM-LIT-LEN         PICTURE 999 COMPUTATIONAL.
      05  FM-TERMINATOR      PICTURE X.
      05  FM-DATA-WORD.
        06  FM-DATA-CHAR OCCURS 20 TIME PICTURE X.
02  FM-LIT-CHAR-COUNT        PICTURE 999 COMPUTATIONAL.
02  FM-LIT-DATA.
  03  LM-LIT-CHAR OCCURS 999 TIMES PICTURE X.
```

FM-WORD-COUNT.

Contains the number of words in the statement examined by FMSCN. As indicated by the OCCURS clause for FM-DATA-ITEM, the maximum number of words a statement may have is 50.

FM-DATA-ITEM.

Has a fixed number of occurrences (50). However, only the items matching words contain significant information (that is, item 1 to the value of FM-WORD-COUNT).

FM-CHAR-COUNT.

Contains the number of characters in the word. It is zero when the word is a literal or when two consecutive terminating characters appeared in the statement.

FM-LIT-POINT.

Contains the subscript of the first character of a literal in the FM-LIT-DATA area. For words exceeding a size of 20 characters, FM-LIT-POINT indicates the position in FM-LIT-AREA of the twenty-first character of the word. In all other cases, FM-LIT-POINT is zero.

FM-LIT-LEN.

Contains the number of characters in a literal placed in FM-LIT-DATA area. The 4-8 characters surrounding the literal in the statement is not included in this size. FM-LIT-LEN is zero when the word is not a literal.

FM-TERMINATOR.

Contains the character which follows the word or literal. It is always a blank, comma, or hyphen. When two terminators are placed in succession (except blank-blank and comma-blank) the second terminator is given a separate FM-DATA-ITEM with zero in FM-CHAR-COUNT, FM-LIT-POINT, and FM-LIT-LEN; blank in FM-DATA-WORD; and the second of the delimiters is placed in FM-TERMINATOR.

FM-DATA-WORD.

Contains a non-literal word from the statement. The word is left justified in the 20 character area and the remaining characters are blanks. When FM-CHAR-COUNT is zero, FM-DATA-WORD is completely blank. When FM-CHAR-COUNT exceeds 20 characters, FM-DATA-WORD contains the first 20 characters of the word with additional characters placed in FM-LIT-DATA area. (In this case, FM-CHAR-COUNT contains the size of the full word and FM-LIT-POINT points to the twenty-first character of the word.)

FM-LIT-COUNT.

Contains the total number of characters placed in the FM-LIT-DATA area.

FM-LIT-DATA.

Contains all literals specified on the specified on the statement. It also has the characters above 20 when a word exceeds 20 characters. In either case FM-LIT-POINT specifies the first position for an item. The number of characters for a literal is FM-LIT-LEN. When a word exceeds 20 characters, the number of characters in FM-LIT-DATA is FM-CHAR-COUNT minus 20.

LM-LIT-CHAR.

Is one character of an item in FM-LIT-DATA. FM-LIT-POINT is used as the subscript of LM-LIT-CHAR when locating a literal. (NOTE: LM-LIT-CHAR received its name when it was used for Logical Maintenance. Due to its extensive use in LM, the name was not changed when the balance of File Maintenance programs began to reference LM-LIT-CHAR.)

FMSCN has two entry points, FMSCAN and FMSCAN2. FMSCAN is used for examining a new statement. The first word is assumed to begin in column 1 of the card image, FM-WORD-AREA is cleared, and FM-WORD-COUNT and FM-LIT-CHAR-COUNT are initialized. FMSCAN2 is used for examining a continuation card. In this case, scanning begins in column 2 of the card to look for the beginning of the first word on the card. Since FM-WORD-COUNT and FM-LIT-CHAR-COUNT are not initialized, the first word on the continuation card follows the words on the previous card(s) for the statement.

FMDD from the COBOL COPY library is specified in the Linkage Section of FMSCN. No Working-Storage Section is present.

CHAPTER 3

Section I

File Maintenance Language Processor

1. OVERVIEW. File Maintenance Language Processor (FMLP) is made up of two basic elements -- Ordinary Maintenance Language Processor (OMLP) and Logical Maintenance Language Processor (LMLP). In order to support these two elements there are also certain common routines which handle the processing and validation of control cards and determination of run type (OMLP or LMLP).

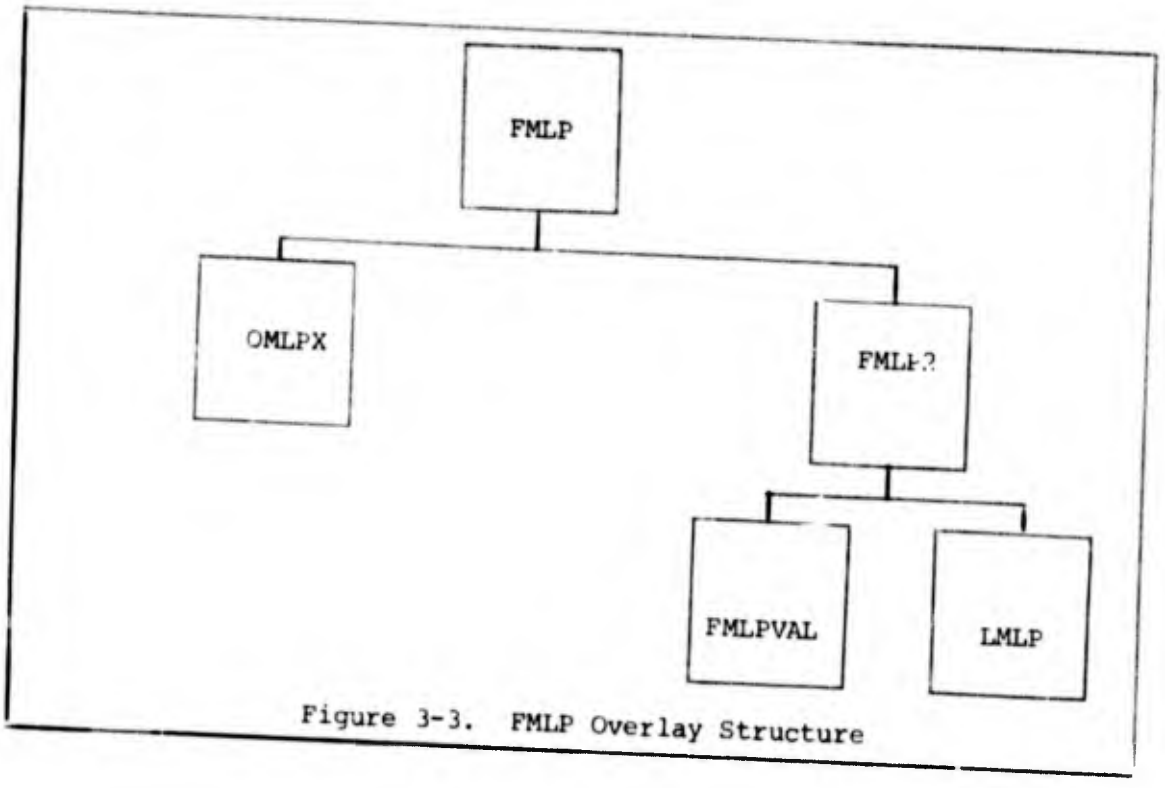


Figure 3-3. FMLP Overlay Structure

a. FMLP. This program controls the process of both OM and LM runs. It passes control to have the control card verified and upon return to it, passes control to OMLP or returns to FM since at this point all LM processing would have been completed.

b. FMLP2. This program is used by both OM and LM. Its function is to pass control to FMLPVAL which validates the control card statements. FMLP2, based on that analysis, then determines whether to pass control to LMLP (in the case of a Logical Maintenance run) or return back to FMLP which will pass control to OMLP (in the case of an Ordinary Maintenance run).

c. FMLPVAL. This program validates the control card (FMLP card) by examination of the words built by FMSCN. It further determines if an LM or OM run is indicated. It also reads in those logical records which are needed for the indicated run.

d. OMLPX. This program is the supervisor for all of Ordinary Maintenance Language Processor. It controls the processing and building of DSD's.

e. LMLP. This program is the supervisor for all of the Logical Maintenance Language Processor. It controls the processing of the user written logical package and generation of appropriate COBOL code to accomplish the desired user specified functions.

2. File Maintenance Language Processor (FMLP) Subprograms.

a. FMLP

(1) Function. This subprogram provides the function of controlling the processing of all File Maintenance language processor runs.

(2) Calling Sequence.

```
ENTRY 'FMLP'   USING FM-DD
CALL  'FMLP2'  USING FM-DD  LP-DD
CALL  'OMLP'   USING FM-DD  LP-DD
```

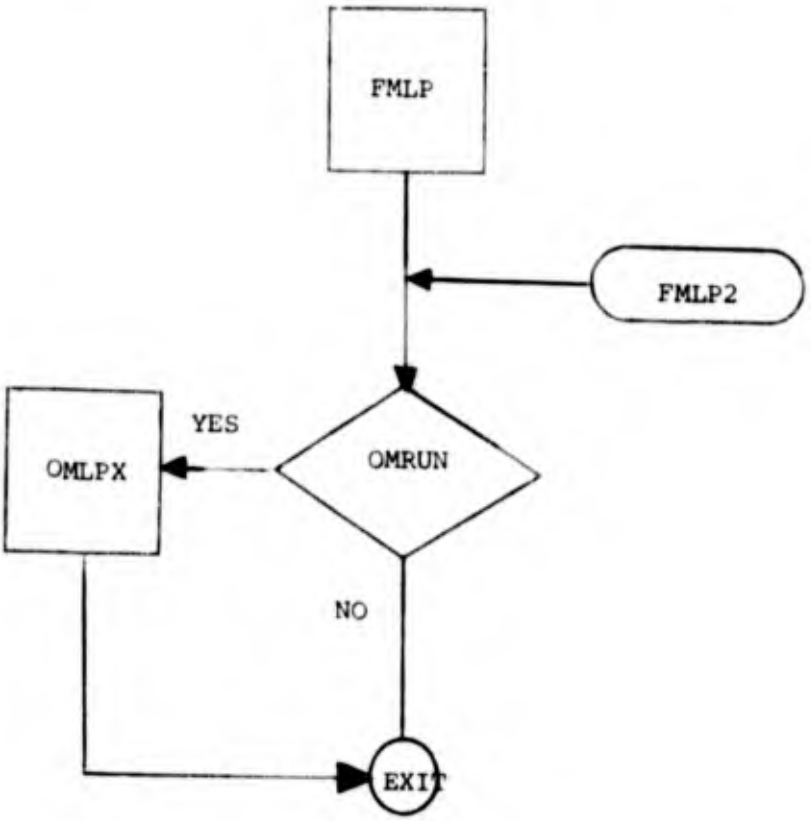
(3) Program Description.

FMLP.

Calls FMLP2.

CHK-RUN-TYPE.

Upon return from FMLP2 determines if it is an Ordinary Maintenance run type; if it is, a call is issued to OMLP. Otherwise, control is returned to FM.



b. FMLP2

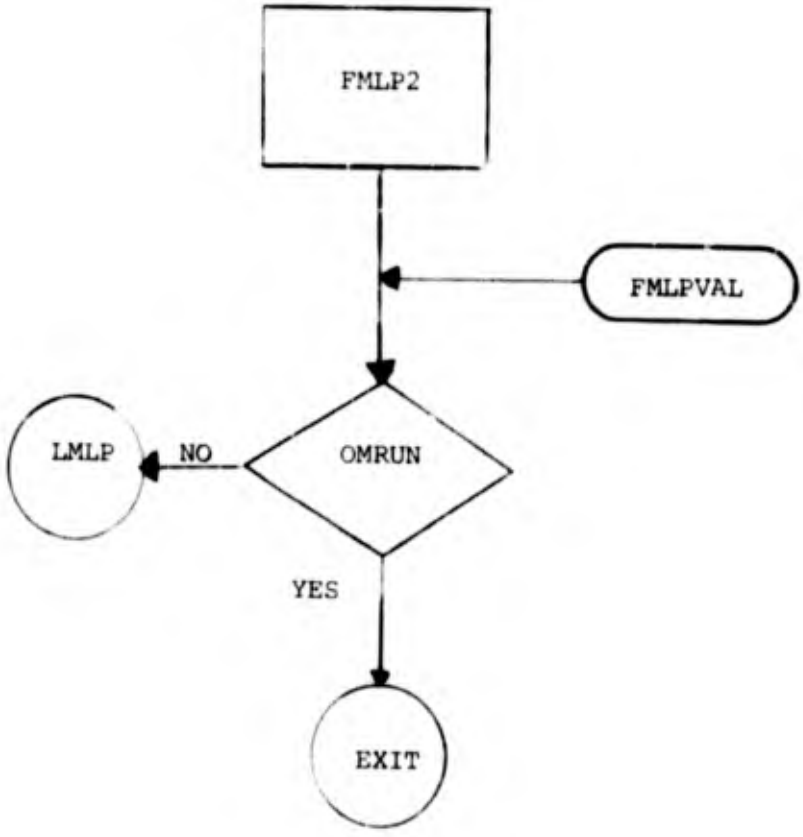
(1) Function. Acts as a go-between OMLP and FMLPVAL which validates the control card. Its main function is to house LMDDV which has the data division for the minor file logical records. For OM runs there is no reference to the minor file so FMLP2 can be overlayed.

(2) Calling Sequence.

```
ENTRY 'FMLP2'  USING FM-DD  LP-DD
CALL  'FMVAL'  USING FM-DD  LP-DD  LM-DD
CALL  'LMLP'   USING FM-DD  LP-DD  LM-DD
```

(3) Program Description.

Calls FMLPVAL to validate the control card. Upon return if an Ordinjry Maintenance run is indicated, control is returned to FMLP. Otherwise, LMLP is called to process the logic package. Upon return from LMLP control is returned to FMLP.



c. FMLPVAL

(1) Function. This program validates the FMLP control card. It determines if an LM or OM run is indicated and in the case of LM if it is file-to-file or single file run type and program or subroutine mode. It also reads in those logical records from the library which are needed for the indicated run-type.

(2) Calling Sequence.

```
ENTRY 'FMVAL'   USING FM-DD  LP-DD  LM-DD
CALL  'FMSET'   USING FMIO
CALL  'FMSCN'   USING FM-DD
CALL  'LB'      USING CALLING-SEQ (desired logical record)
```

(3) Program Description.

Calls FMSET to initialize the entry point address of the FMIO subroutine.
Calls FMSCN to build for control card.
FMLP.

Vertifies second word (DST, DSD, LOG).
CHK-3RD-WRD.
SCAN-3RD-WRD.

Validates third word (logic package name or DSD name).
CHK-4TH-WRD.

Validate major file (5 characters ending in A or T).
CHK-5TH-WRD.

Validates run-type (O for OM, S for Single File LM, F for File-to-File LM).
CHK-RUN-TYPE.

If file-to-file LM run validates minor file name.
CHK-RUN-SPCL.

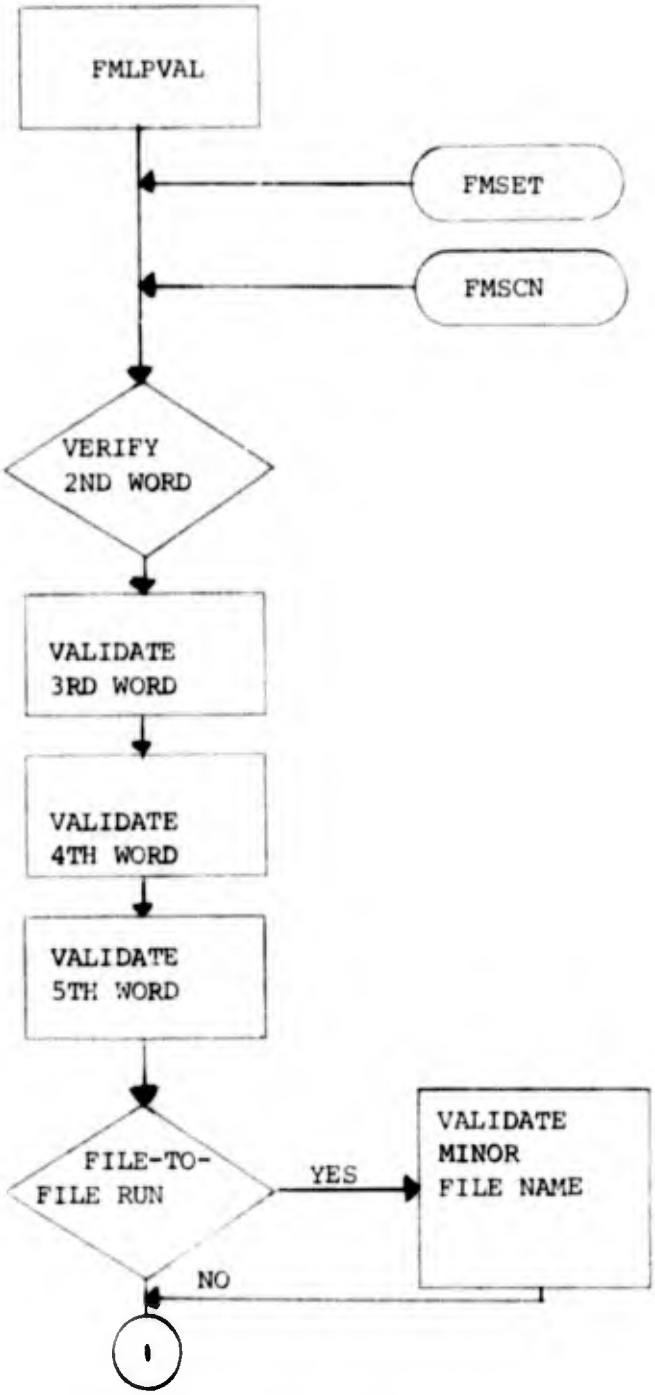
Validates run-special (S=subroutine, D=delete, P=program).
CALL-LIB-NOTE1.

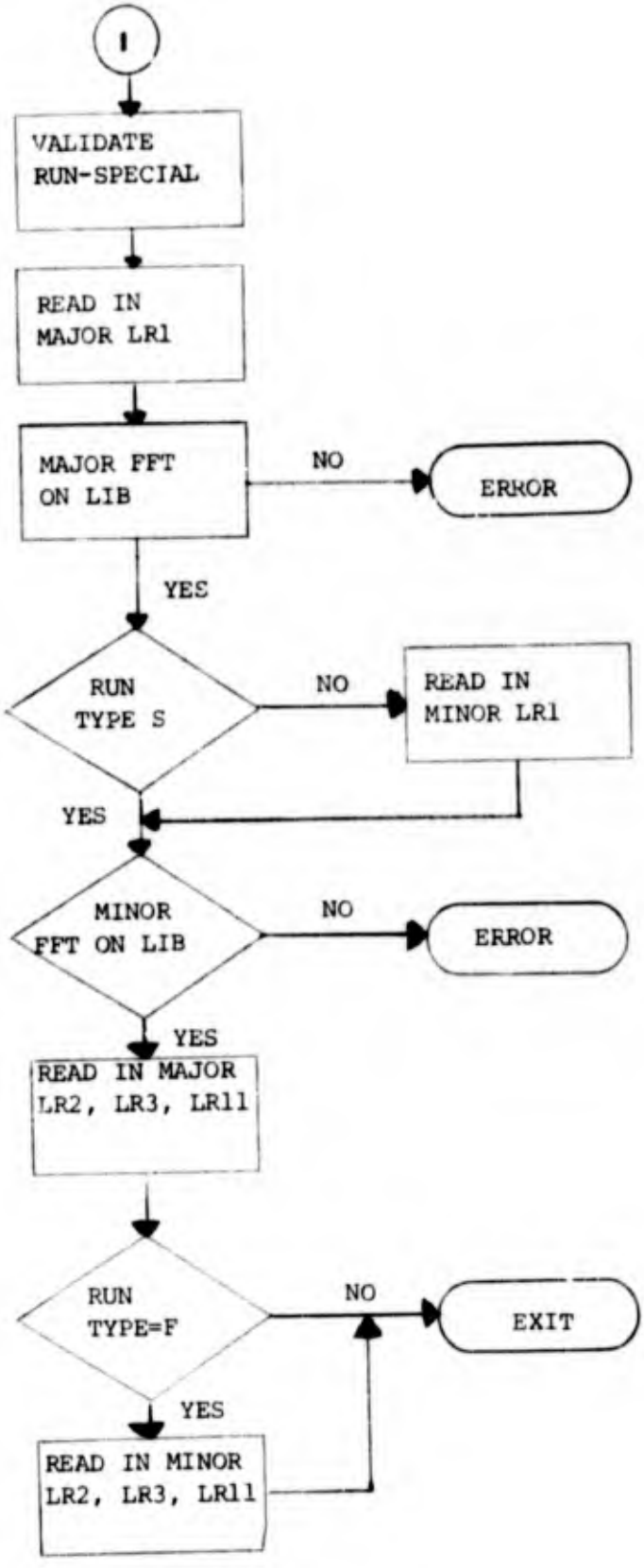
Read in MAJOR-LR1 from Library verify that MAJOR-FFT on library.
CHK-OP-CODE1.

If run-type is file-to-file, read in MINOR-LR1, verify that minor FFT on LIB.
CALL-LIB-NOTE2.

Read in major LR2, LR3, LR11.
CHK-RUN-TYPE1.

If file-to-file run, read in minor LR2, LR3, LR11.
FMLPVAL-EXIT. Returns to FMLP2.


FMLPVAL
FMSET
FMSCN
VERIFY
2ND WORD
VALIDATE
3RD WORD
VALIDATE
4TH WORD
VALIDATE
5TH WORD
FILE-TO-
FILE RUN
YES
VALIDATE
MINOR
FILE NAME
NO
1


1
VALIDATE
RUN-SPECIAL
READ IN
MAJOR LR1
MAJOR FFT
ON LIB
NO
ERROR
YES
RUN
TYPE S
NO
READ IN
MINOR LR1
YES
MINOR
FFT ON LIB
NO
ERROR
YES
READ IN MAJOR
LR2, LR3, LR11
RUN
TYPE=F
NO
EXIT
YES
READ IN MINOR
LR2, LR3, LR11

CHAPTER 3

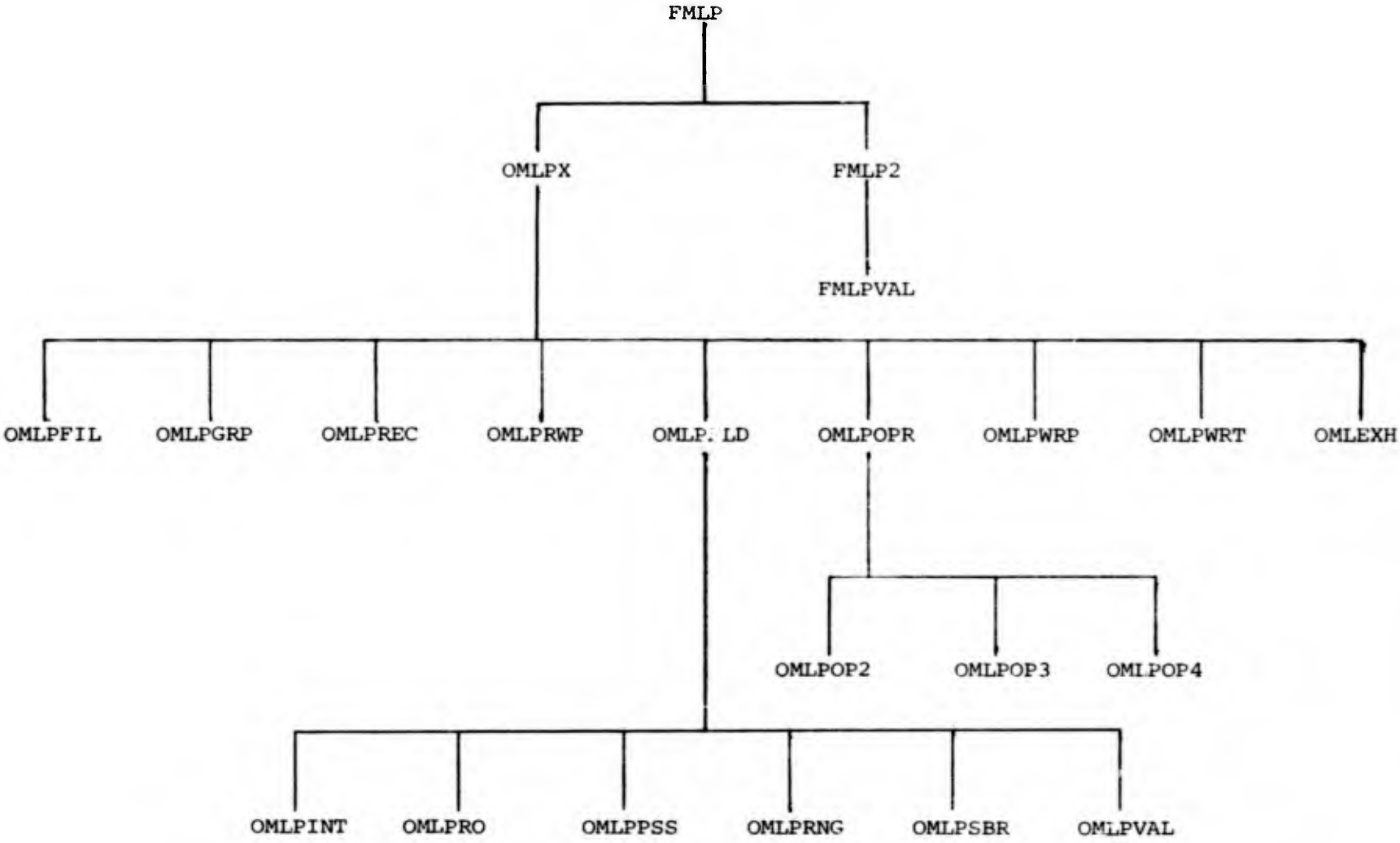
Section II

Ordinary Maintenance Language Processor (OMLP)

1. SUBSYSTEM STRUCTURE.

a. OMLPX is the supervisor for the Ordinary Maintenance Language Processor. It is called by FMLP. The function of OMLPX is to scan the incoming card images and thru the use of keywords, call the appropriate subprograms to validate the card and assimilate the information presented on it. In this way OMLPX calls OMLPFIL for the FILE card, OMLPGRP for the GROUP card, OMLPREC for RECORD cards, and OMLPFLD for FIELD cards. When the final field card has been read and verified, OMLPX will call OMLPRWP to do final processing on the last RECORD. OMLPX then calls OMLPWRP which completes final processing on the DSD. If the DUMPDSD option has been specified on the FMLP control card, OMLPX calls OMLEXH to print a dump of the DST. If no error conditions have been found, OMLPWRT is called to write the DST onto the MIDMS Library.

b. Because of core considerations, the OMLPFLD and OMLPOPR subprograms have been further subdivided. OMLPFLD calls OMLPINT to do basic initialization of the FIELD card. OMLPPSS is called if the field specified is defined as a Periodic field. The remaining programs called by OMLPFLD verify and assimilate the information concerning the optional Validity Checks available to the FIELD card. OMLPPRO processes the PROFILE check; OMLPRNG processes the RANGE check; OMLPVAL processed the VALUE check, and OMLPSBR processes the SUBCON and SUBCHK checks. The breakdown of OMLPOPR was not designed by specific function but by logic flow groupings.



OMLP Overlay Structure

2. COBOL ENTRIES & SUBROUTINES. The following description of entries in DSD01 is in two parts. Part 1 is an itemized list of the items in the DST tables and a brief explanation of what they are used for. Part 2 describes the entries in the VAL-LIT-AREA for OPR's and the validity checks.

a. DSD01 Entries. The DST tables are a part of the total OM-DD. Many of the tables in the DST have counterparts with table names ending in TEMP. Item entries in the TEMP tables are used by the total group of programs comprising OMLP. They are temporary tables that are kept only as long as the DSD is being processed. Once the DST is completed these temporary tables are destroyed. The tables comprising the DST begin with the 02 level DSD01 and end with the 02 level VAL-LIT-AREA.

FILE-REC-HOP	High Order Position (HOP) of the FILE RECORD CODE
FIL-REC-LEN	Length (LEN) of the FILE RECORD CODE
VAL-LIT-CNT	Number of positions used in the VAL-LIT-AREA
FIL-OPR-CT	Number of OPR entries on the FILE level
FIL-OPR-POINT	Subscript of the first OPR in the VAL-LIST
FIL-ID-CNT	Number of items in the FIL-ID-LIST
GRP-ID-CNT	Number of items in the GRP-ID-LIST
GRP-TYPES-CNT	Number of items in the GRP-TYPES-LIST
MAX-FLDS	The maximum number of fields in an SIR taken over all SIR's
MAX-TOT-FLD-LEN	Maximum size of the largest SIR
REC-IDS-CNT	Maximum number of record ID's in this DSD
REC-TYPE-CNT	Number of items in the REC-TYPE-LIST
FIELD-CNT	Number of items in the FIELD-LIST
FIL-OPR-NAME-CNT	Number of items in the FIL-OPR-NAME-LIST
REC-FIL-OPR-CNT	Number of items in the REC-FIL-OPR-LIST
VAL-CNT	Number of items in the VAL-LIST
OPR-SUP-SW	Indicates if the DST can process blanks
FIL-REC-ERR	Error action code for FILE RECORD CODE parameter
FIL-SEQ-ORD-ERR	Error action code for FILE SEQ/ORD field
GRP-SEQ-ORD-ERR	Error action code for GROUP SEQ/ORD field
GRP-FREQ-ERR	Error action code following the GROUP TYPES
FIL-NONE-SW	Set at 1 if FILE RECORD CODE NONE, otherwise 0
FIL-SEQ-ORD-SW	Set at 0 if no FILE SEQ/ORD field specified
GRP-SEQ-ORD-SW	Set at 0 if no GROUP SEQ/ORD field specified
FIL-ID-HOP	HOP of field within FFS ID specifications
GRP-ID-HOP	HOP of field within GRP ID specifications
GRP-FREQ	Frequency code for the record type, default 0
REC-SIZE	Size of the record
REC-TYPE-CODE	Record name specified in the RECORD TYPE parameter
REC-FIELD-CT	Number of fields in the record

REC-FIELD-POINT	Subscript of the first field in the FIELD LIST per record
REC-OPR-CT	Number of OPR entries per record
REC-OPR-POINT	Subscript of first OPR in the VAL-LIST
REC-GRP-POINT	Subscript of the GRP-TYPE entry in the GRP-TYPES-LIST
REC-CDMS-CT	Number of FFS-ID fields in the REC-IDS-LIST
REC-CDMS-ID	Subscript of the first FFS ID field in REC-IDS-LIST
REC-GRP-ID	Subscript of the first GRP-ID field in REC-IDS-LIST
REC-TOT-FLD-LEN	Sum of the lengths of the fields of an SIR
REC-FIL-SEQ-ORD	Subscript of FILE SEQ/ORD field in FIELD-LIST
REC-GRP-SEQ-ORD	Subscript of GROUP SEQ/ORD field in FIELD-LIST
REC-FIL-OPR	Subscript of FILE OPR field in REC-FIL-OPR-LIST
REC-IDS-NO	Subscript of item in FIL-ID-LIST, GRP-ID-LIST, SUB-ID-LIST
REC-IDS-FLD	Subscript of field in FIELD-LIST
FLD-HOP	HOP of FIELD in record
FLD-LEN	LEN of field
FLD-PSS-HOP	HOP of field's PSSQ in record
FLD-PSS-LEN	LEN of fields PSSQ
FLS-LR2-POINT	Subscript of FFT LR2 entry or 0
FLD-VAL-CT	Number of validity checks on field
FLD-VAL-POINT	Subscript of first validity check in VAL-LIST
FLD-SKIP	Indicates if field can be blank
FLD-NAME	Name of field
FIL-OPR-LIT-POINT	HOP of C or D OPR subscript in VAL-LIT-AREA
REC-FIL-OPR-FLD	Subscript of FIL-OPR field in FIELD-LIST
VAL-LIT-HOP	Subscript of first character in VAL-LIT-AREA
VAL-LIT-LEN	Number of characters in VAL-LIT-AREA for the test
VAL-TYPE	Numberic designation of the OPR or Validity Check
VAL-ERR	Error action code for validity check or OPR
VAL-LIT-AREA	Data string storage for validity checks

b. The entries in the VAL-LIT-AREA for OPR's and validity checks are as follows:

DESCRIPTION OF ENTRIES IN VAL-LIT-AREA

VAL-TYPE	ENTRY IN VAL-LIT-AREA	TYPE OF PHRASE	REMARKS
1	nn...nn	VALIDITY CHECK	Low value of range (numeric).
9	nn...nn		High value of range (numeric).
2	nn nn ...		Two digit pair describing each PROFILE character.
5	nnnn value		nnnn relative HOP within SIR of field to be tested when partial field addressing is used.
7	nnnn values...		nnnn number of repeating values to be checked against the data (used with full field and literal values).
3	literal aaaaS	SUBCON	aaaaS is subroutine name ending in S.
4	literal aaaaS	SUBCHK	Identical to SUBCON.
6	1 KK	OPR	Type A OPR. 1 indicates type, kk is numeric opcode.
	2 kk llll mmmm		2 indicates Type E, kk is numeric opcode, llll and mmmm are HOP indicators for VSET OPR's.
	3 nnnn l literal kk		3 indicates Type B OPR, nnnn is HOP in the record to be tested. l specifies the length of the literal. kk is a numeric opcode.
	4 ssss l literal kk		4 indicates Type C OPR. ssss is subscript of the field name entry in the FIELD-LIST. l specifies the literal length. kk is a numeric opcode.

VAL-TYPE	ENTRY IN VAL-LIT-AREA	TYPE OF PHRASE	REMARKS
	5 ssss tttt kk		5 indicates Type D OPR. ssss and tttt are subscripts of the fields in the FIELD-LIST. kk is a numeric opcode.
	6 l literal kk		6 indicates a continuation of a type 3 or 4 OPR. l is the length of the literal. kk is a numeric opcode.

c. Subroutines.

(1) OMLPX

(a) Function. The function of OMLPX is to call the appropriate subroutines which process the Data Source Description (DSD) and store the Data Source Table (DST) on the MIDMS Library.

(b) Calling Sequence.

```
ENTRY   'OMLP'      USING FM-DD  LP-DD
CALL    'FMCRD'     USING FM-INPUT  END-SW
CALL    'FMPRT'     USING PRT-LINE  CC
CALL    'FMSCAN'    USING FM-DD
CALL    'FMSCAN2'   USING FM-DD
CALL    'OMFIL'     USING FM-DD  OM-DD
CALL    'OMGRP'     USING FM-DD  OM-DD
CALL    'OMSUB'     USING FM-DD  OM-DD
CALL    'OMREC'     USING FM-DD  OM-DD
CALL    'OMRWP'     USING FM-DD  OM-DD
CALL    'OMFLD'     USING FM-DD  LP-DD  OM-DD
CALL    'OMWRP'     USING FM-DD  LP-DD  OM-DD
CALL    'OMWRT'     USING FM-DD  OM-DD  LP-DD
CALL    'OMOPR'     USING FM-DD  LP-DD  OM-DD
```

(c) Program Description.

OMLP-1A and OMLP-3.
These paragraphs print each card and perform the basic card scanning functions, allowing for comment cards, continuation cards as well as the first and image of a given card type.

OMLP-SCAN1 and OMLP-SCAN2.
These paragraphs scan the first card image of a given card type and its continuation cards building word tables for each card type.

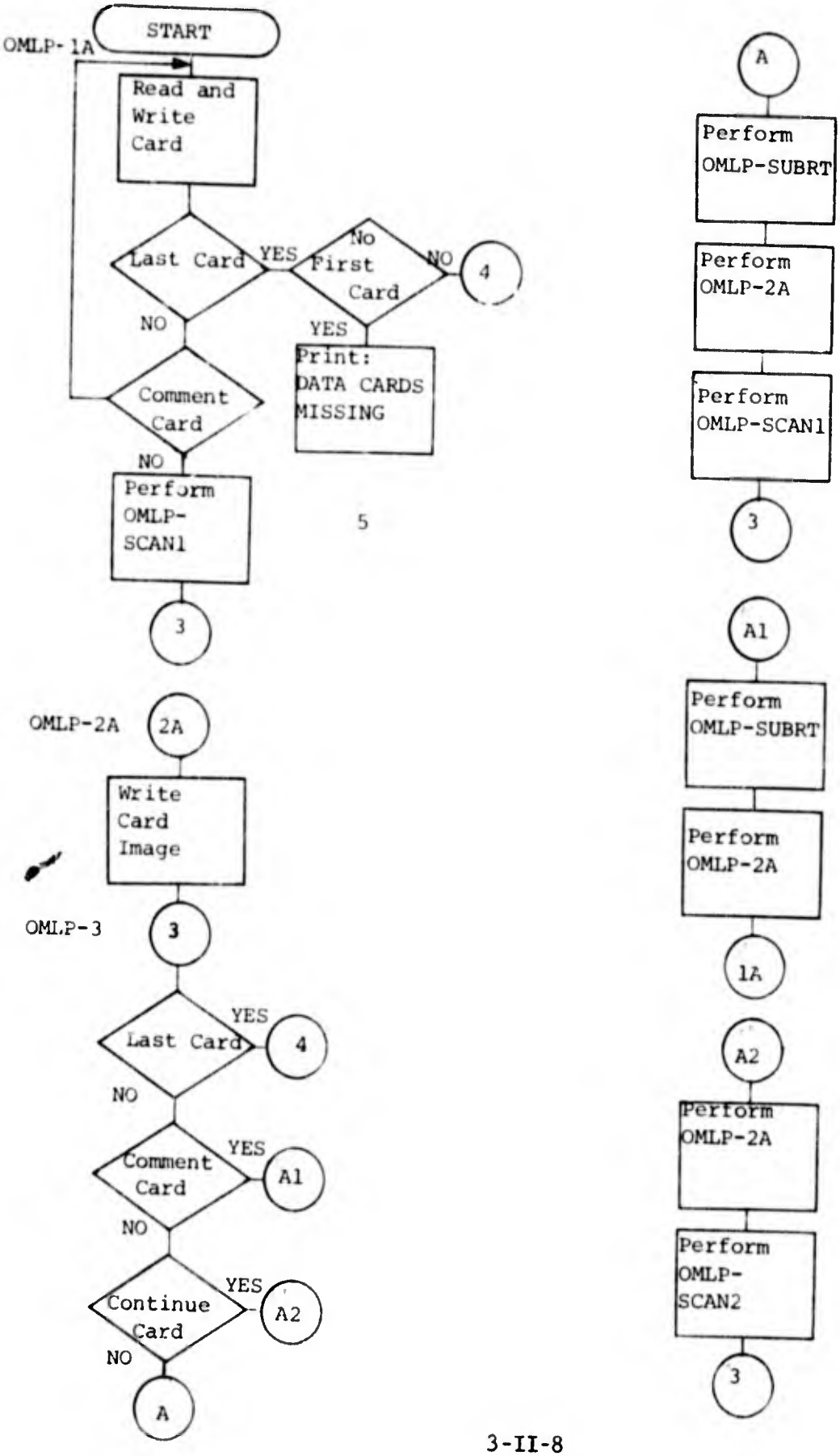
OMLP-SUBRT.
Calls the appropriate subroutine to process the DSD card type.

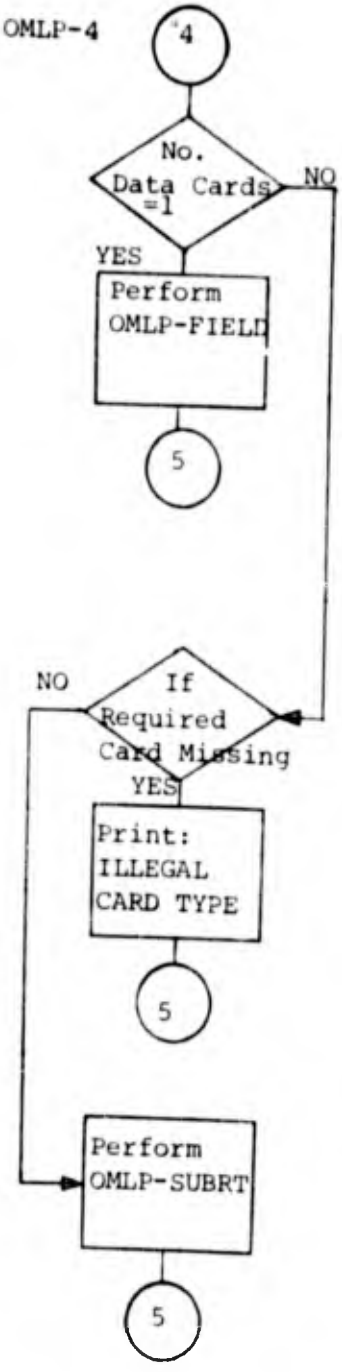
OMLP-5.
Wraps up the final record and providing no errors were encountered during processing calls program OMLWRT to write the DSD on the MIDMS Table Library.

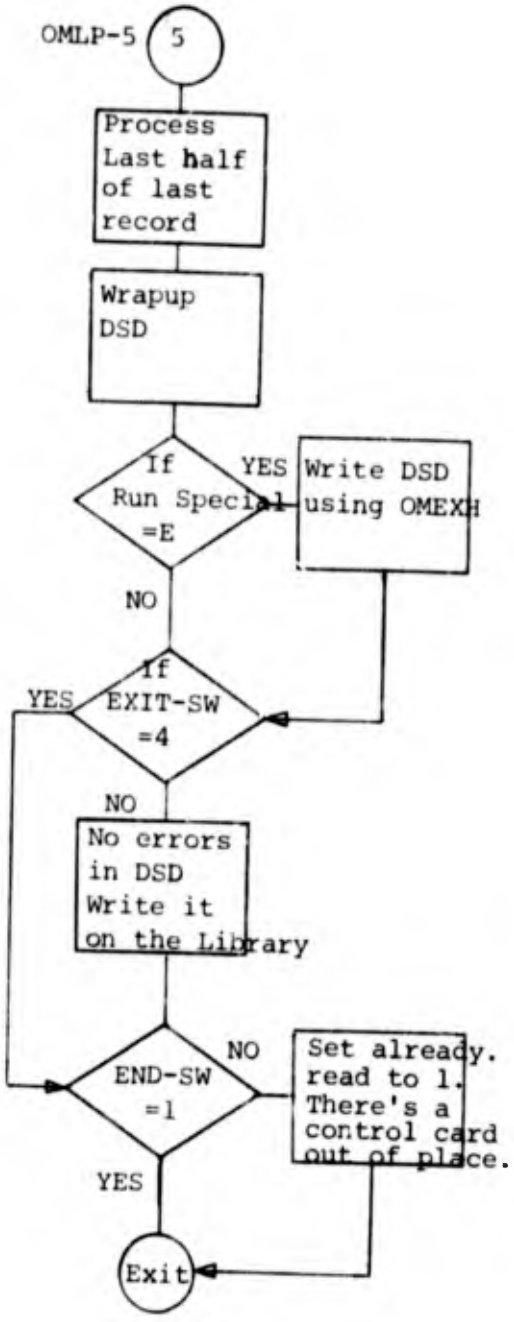
OMLP-WRITE.
Calls the write program to place the DSD in the table library.

OMLP-CALL-OMEXH.
Gives a dump of the DSD if the DUMPOSD option was specified in the FMLP control card.

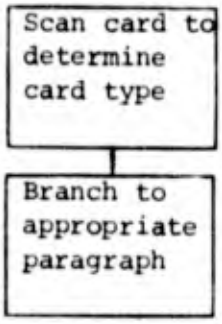
(d) Limitations. None.


OMLP-1A
START
Read and Write Card
Last Card
YES
No First Card
NO
4
NO
YES
Print: DATA CARDS MISSING
Comment Card
NO
Perform OMLP-SCAN1
5
3
OMLP-2A
2A
Write Card Image
OMLP-3
3
Last Card
YES
4
NO
Comment Card
YES
A1
NO
Continue Card
YES
A2
NO
A
A
Perform OMLP-SUBRT
Perform OMLP-2A
Perform OMLP-SCAN1
3
A1
Perform OMLP-SUBRT
Perform OMLP-2A
1A
A2
Perform OMLP-2A
Perform OMLP-SCAN2
3

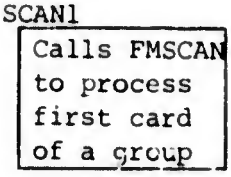

OMLP-4
4
No. Data Cards =1
NO
YES
Perform OMLP-FIELD
5
NO
If Required Card Missing
YES
Print: ILLEGAL CARD TYPE
5
Perform OMLP-SUBRT
5


OMLP-5
5
Process Last half of last record
Wrapup DSD
If Run Special =E
YES
Write DSD using OMEXH
NO
If EXIT-SW =4
YES
NO
No errors in DSD Write it on the Library
END-SW =1
NO
Set already. read to 1. There's a control card out of place.
YES
Exit

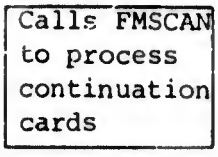
OMLP-SUBRT



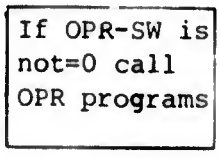
OMLP-SCAN1



OMLP-SCAN2



OMLP-FILE-EXIT



(2) OMLPFIL

(a) Function. The function of OMLPFIL is to process the FILE card. Keywords break up the FILE card parameters into phrases. These phrases are edited. If errors are found in the specifications, diagnostics are provided. The information on the FILE card is incorporated into the DST.

(b) Calling Sequence.

```
ENTRY  'OMFIL'  USING FM-DD  OM-DD
CALL   'FMPRT'  USING PRT-LINE  CC
```

(c) Program Description.

OMLPFIL.

Determines if a file card has been processed. If so an error message is printed and control is passed to OMLPX. If not, it initializes counters, switches and hold areas.

FIL-1.

Tests to see if all words have been processed.

FIL-2.

Scans card for keywords and branches to appropriate paragraph when one is found.

FIL-3, FIL-3E, FIL-3F, FIL-4A.

These paragraphs edit the RECORD CODE phrase and make entries in the DST as follows:

FIL-3E sets FIL-NON-SW.

FIL-4C sets FIL-REC-HOP and FIL-REC-LEN.

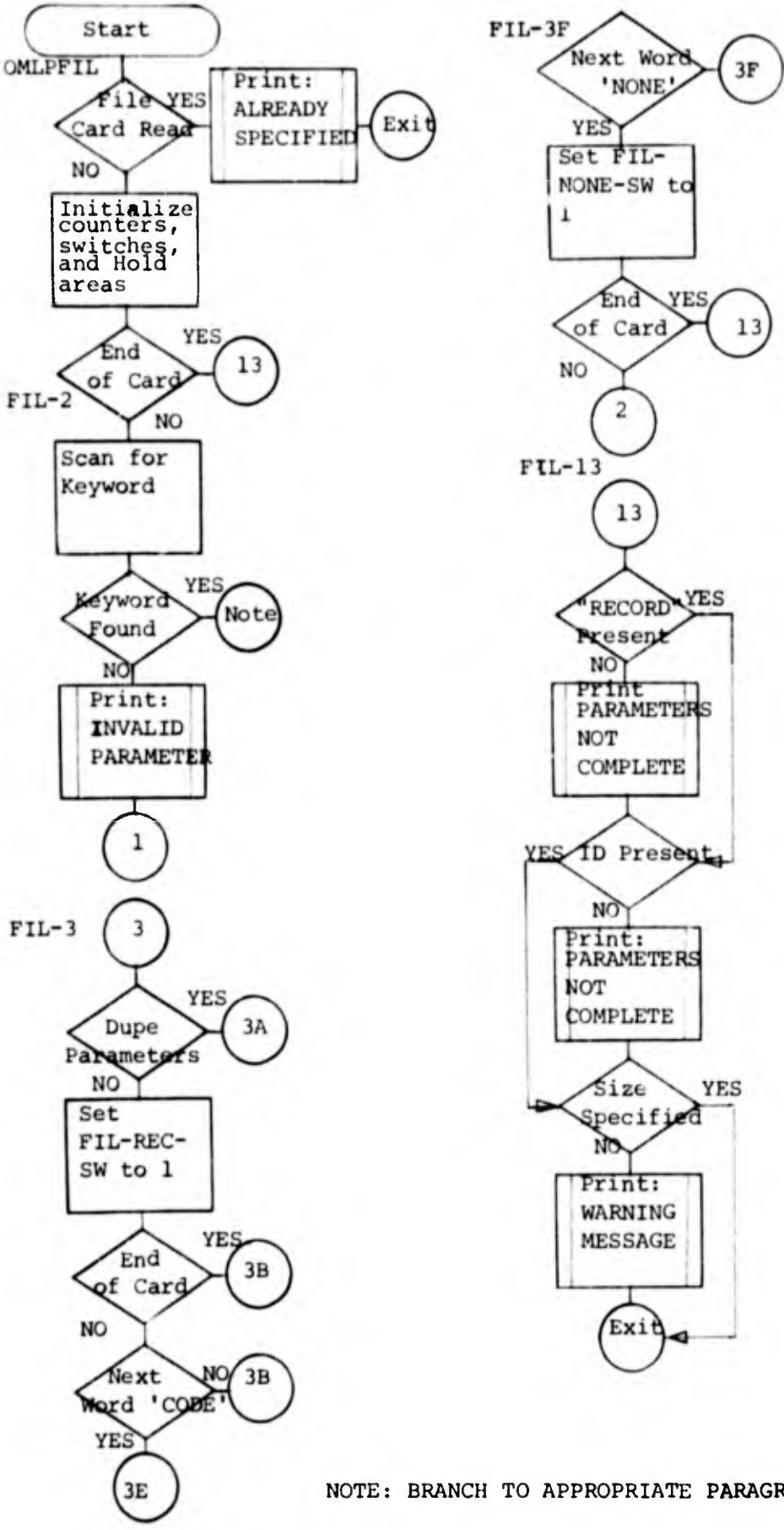
FIL-4 enters the error action code for this phrase in FIL-REC-ERR.

FIL-6C edits the FFS-ID phrase and enters the ID in the FIL-ID-LIST-TEMP. FIL-ID-LEN is initialized to 0 for each FIL-ID-NAME to be used as a check to insure that all names appear in the DSD.

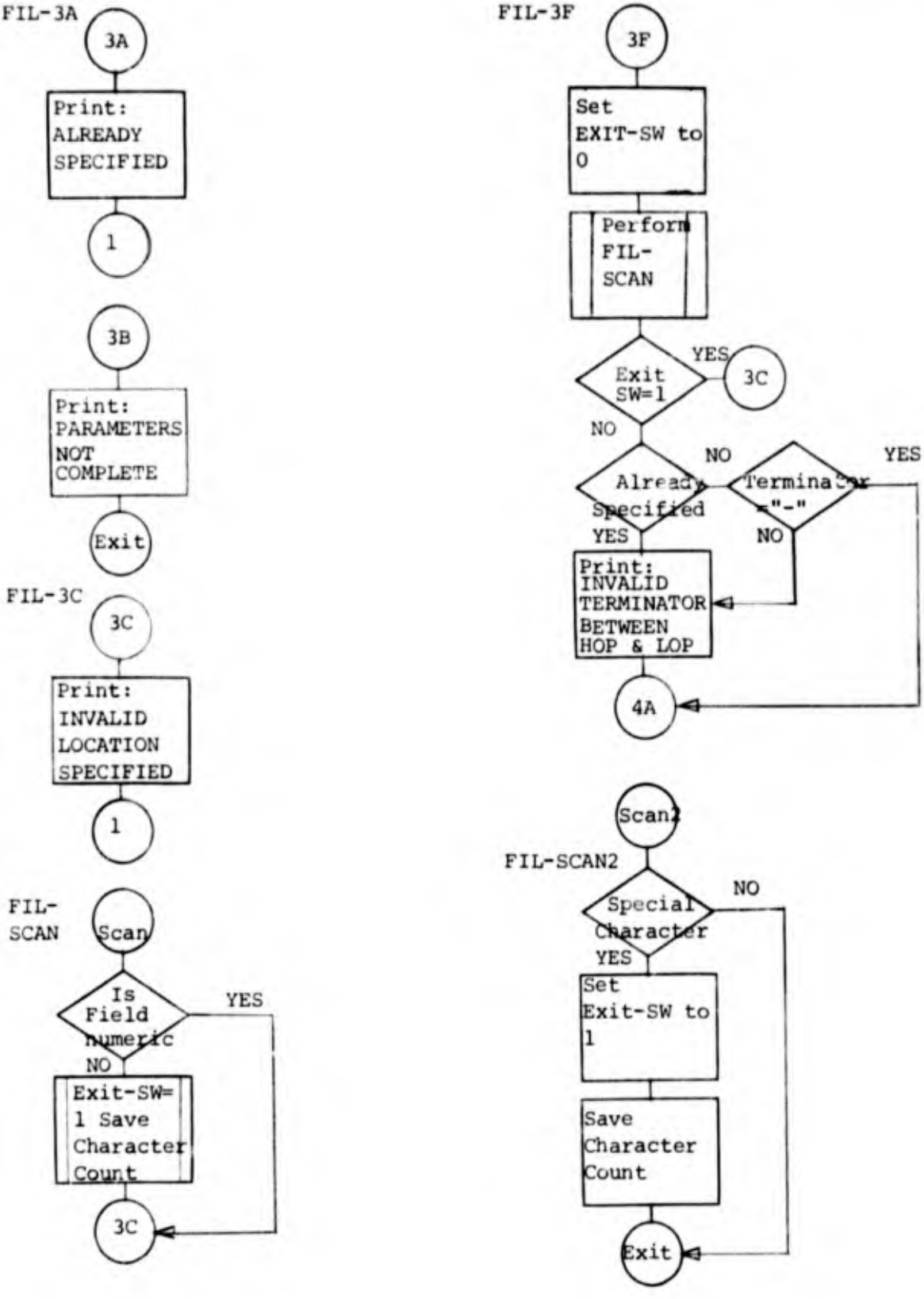
FIL-8 edits the SIZE parameter and stores it in FILE-REC-SIZE. FIL-NUM-FIELD is performed to right justify the SIZE in a BCD field which is then moved to FIL-REC-SIZE. This is initialized as 80.

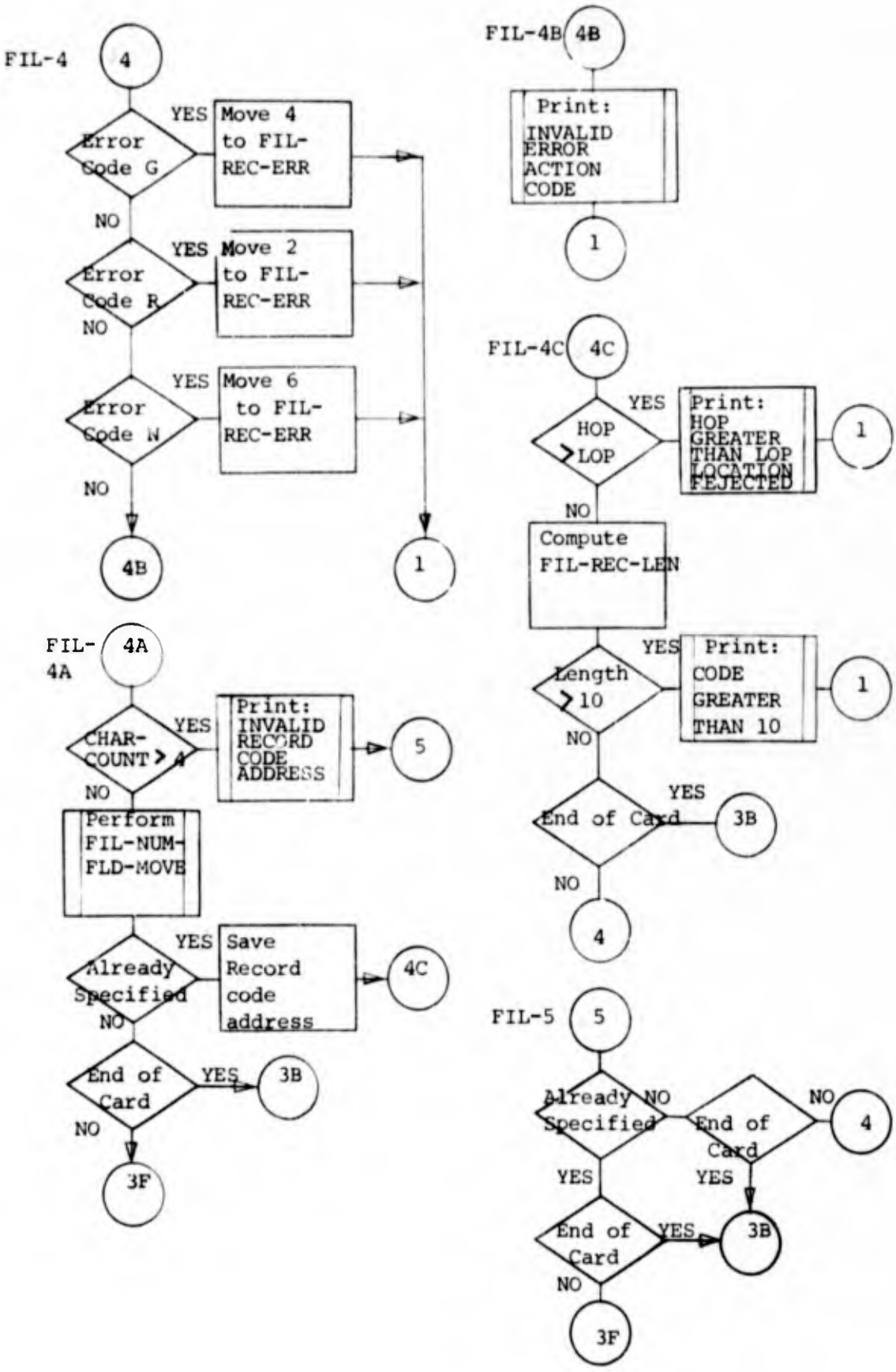
FIL-9B edits the Sequence/Order Test Field names and stores it in FILE-SEQ-ORD. FILE-SEQ-ORD-SW indicates whether it is a Sequence or Order field (FIL-9/FIL-11). FIL-10 enters the error action code in FIL-SEQ-ORD-ERR. FIL-12 sets FM-OPR-SW if an OPR phrase is encountered. OMLPX will call the OPR program.

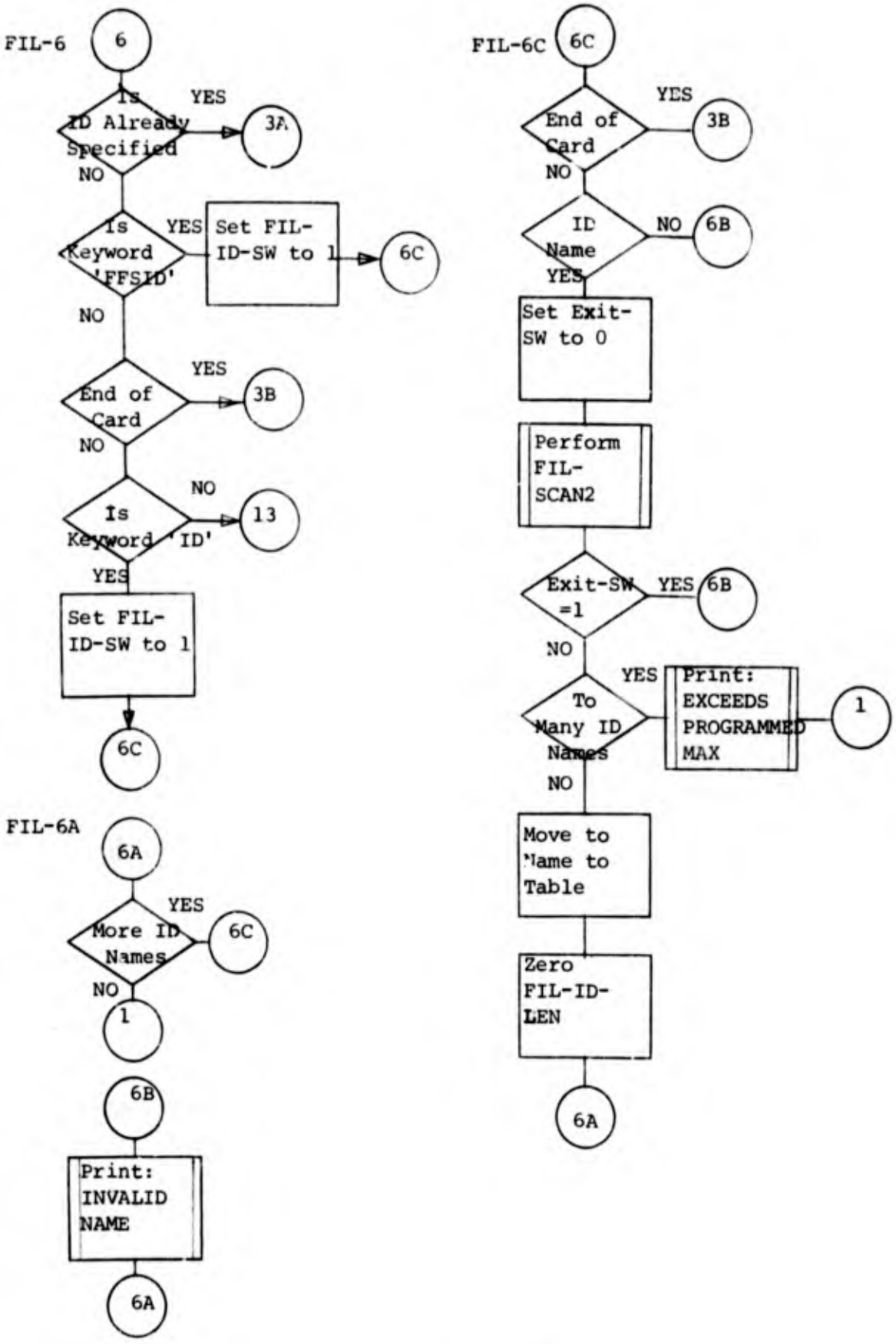
(d) Limitations. If a file record code is specified, its length cannot exceed 10 characters. The size of the input records default to 80 characters, and cannot exceed 9999 characters.

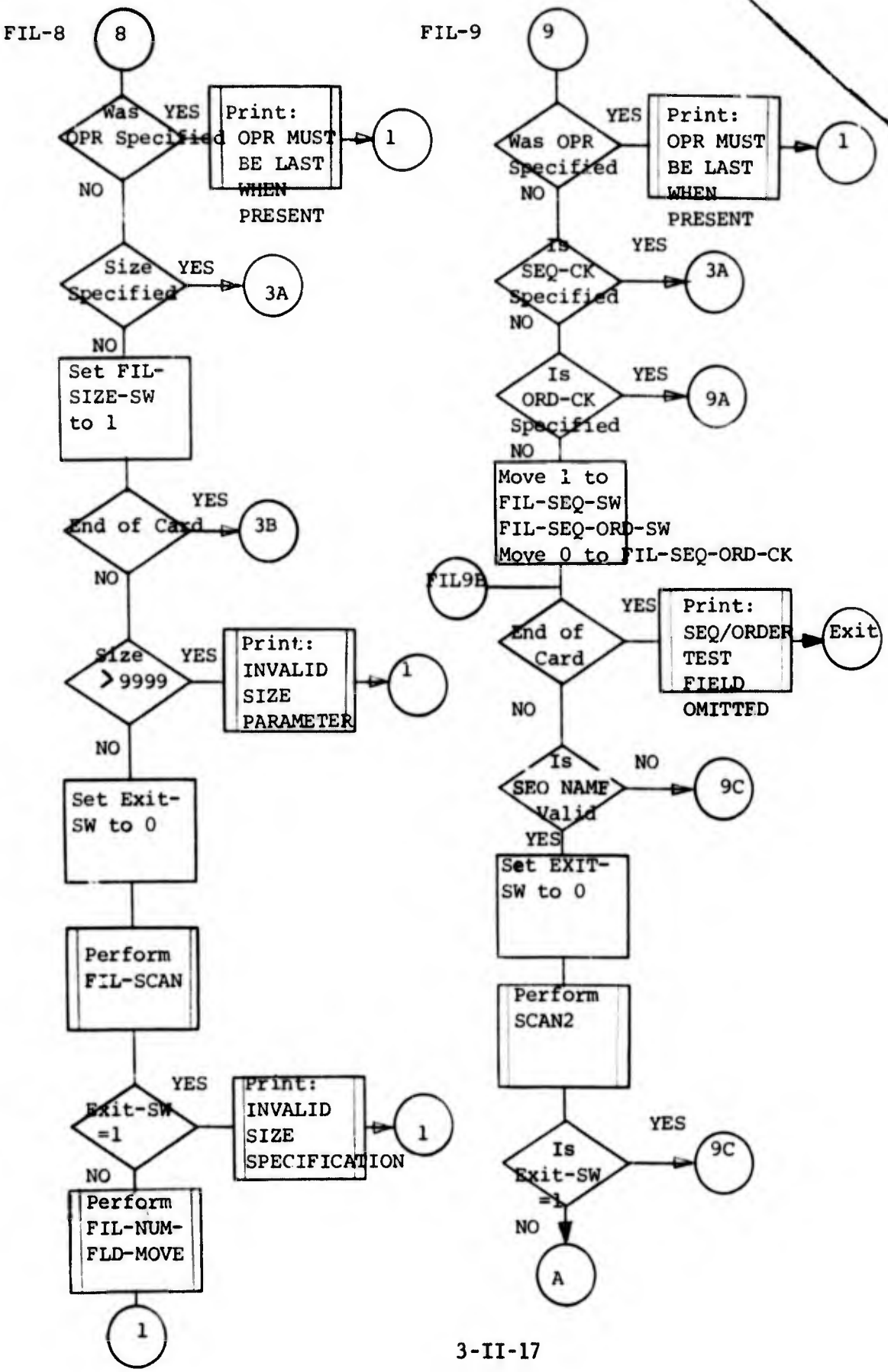


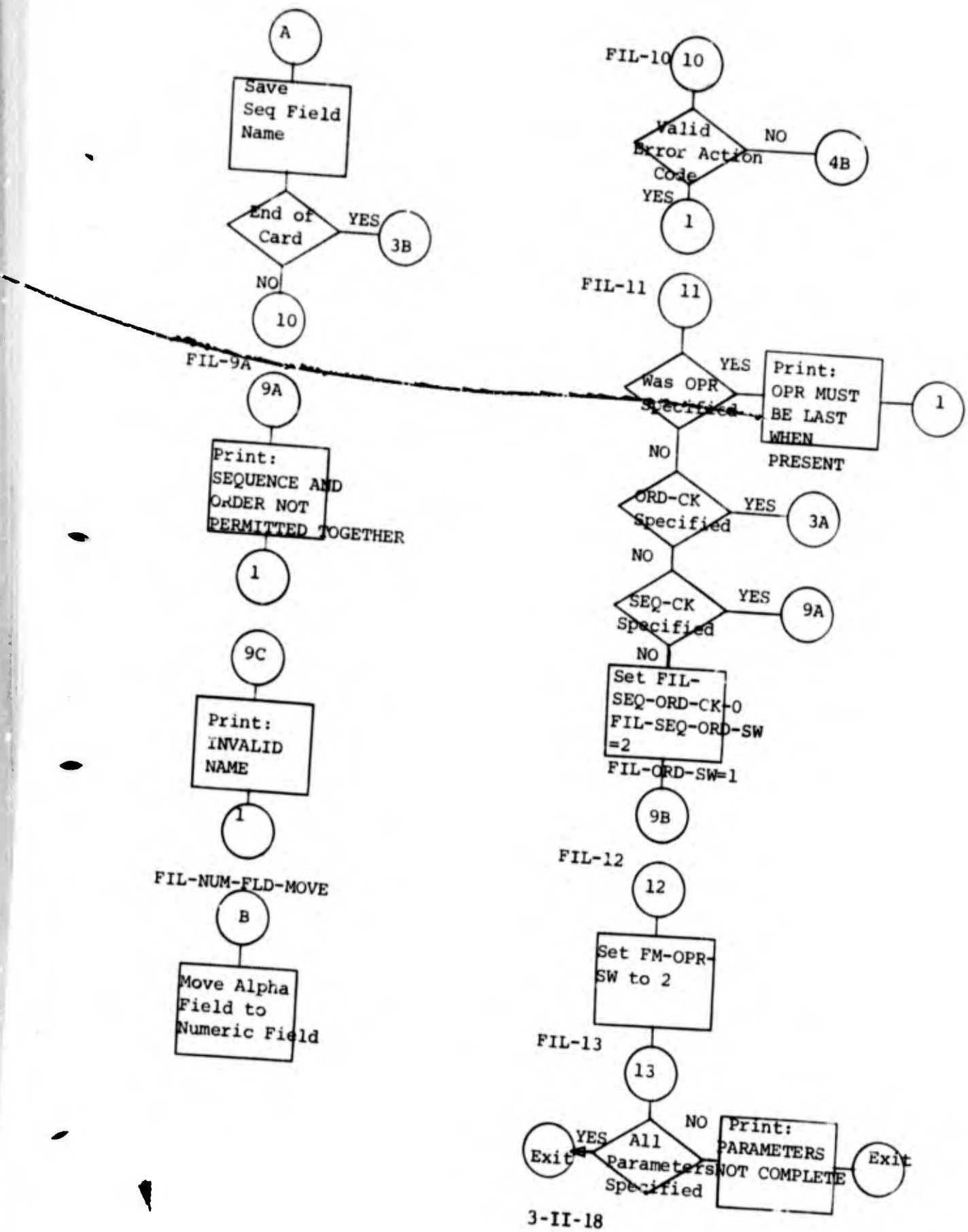
NOTE: BRANCH TO APPROPRIATE PARAGRAPH


FIL-3A
3A
Print: ALREADY SPECIFIED
1
3B
Print: PARAMETERS NOT COMPLETE
Exit
FIL-3C
3C
Print: INVALID LOCATION SPECIFIED
1
FIL-SCAN
Scan
Is Field numeric
YES
NO
Exit-SW= 1 Save Character Count
3C
FIL-3F
3F
Set EXIT-SW to 0
Perform FIL-SCAN
Exit SW=1
YES
3C
NO
Already Specified
NO
Terminator "-"
YES
NO
YES
Print: INVALID TERMINATOR BETWEEN HOP & LOP
4A
Scan2
FIL-SCAN2
Special Character
NO
YES
Set Exit-SW to 1
Save Character Count
Exit


FIL-4
4
Error Code G
YES
Move 4 to FIL-REC-ERR
NO
Error Code R
YES
Move 2 to FIL-REC-ERR
NO
Error Code N
YES
Move 6 to FIL-REC-ERR
NO
4B
1
FIL-4B
4B
Print: INVALID ERROR ACTION CODE
1
FIL-4C
4C
HOP > LOP
YES
Print: HOP GREATER THAN LOP LOCATION REJECTED
1
NO
Compute FIL-REC-LEN
Length > 10
YES
Print: CODE GREATER THAN 10
1
NO
End of Card
YES
3B
NO
4
FIL-4A
4A
CHAR-COUNT > 4
YES
Print: INVALID RECORD CODE ADDRESS
5
NO
Perform FIL-NUM-FLD-MOVE
Already Specified
YES
Save Record code address
4C
NO
End of Card
YES
3B
NO
3F
FIL-5
5
Already Specified
NO
End of Card
NO
4
YES
YES
End of Card
YES
3B
NO
3F


FIL-6
6
Is ID Already Specified
YES
3A
NO
Is Keyword 'FFSID'
YES
Set FIL-ID-SW to 1
6C
NO
End of Card
YES
3B
NO
Is Keyword 'ID'
NO
13
YES
Set FIL-ID-SW to 1
6C
FIL-6A
6A
More ID Names
YES
6C
NO
1
6B
Print: INVALID NAME
6A
FIL-6C
6C
End of Card
YES
3B
NO
ID Name
NO
6B
YES
Set Exit-SW to 0
Perform FIL-SCAN2
Exit-SW =1
YES
6B
NO
To Many ID Names
YES
Print: EXCEEDS PROGRAMMED MAX
1
NO
Move to Name to Table
Zero FIL-ID-LEN
6A


FIL-8
8
Was OPR Specified
YES
Print: OPR MUST BE LAST WHEN PRESENT
1
NO
Size Specified
YES
3A
NO
Set FIL-SIZE-SW to 1
End of Card
YES
3B
NO
Size > 9999
YES
Print: INVALID SIZE PARAMETER
1
NO
Set Exit-SW to 0
Perform FIL-SCAN
Exit-SW =1
YES
Print: INVALID SIZE SPECIFICATION
1
NO
Perform FIL-NUM-FLD-MOVE
1
FIL-9
9
Was OPR Specified
YES
Print: OPR MUST BE LAST WHEN PRESENT
1
NO
Is SEQ-CK Specified
YES
3A
NO
Is ORD-CK Specified
YES
9A
NO
Move 1 to FIL-SEQ-SW FIL-SEQ-ORD-SW
Move 0 to FIL-SEQ-ORD-CK
FIL9B
End of Card
YES
Print: SEQ/ORDER TEST FIELD OMITTED
Exit
NO
Is SEQ NAME Valid
NO
9C
YES
Set EXIT-SW to 0
Perform SCAN2
Is Exit-SW =1
YES
9C
NO
A


A
Save Seq Field Name
End of Card
YES
3B
NO
10
FIL-9A
9A
Print: SEQUENCE AND ORDER NOT PERMITTED TOGETHER
1
9C
Print: INVALID NAME
1
FIL-NUM-FLD-MOVE
B
Move Alpha Field to Numeric Field
FIL-10
10
Valid Error Action Code
NO
4B
YES
1
FIL-11
11
Was OPR Specified
YES
Print: OPR MUST BE LAST WHEN PRESENT
1
NO
ORD-CK Specified
YES
3A
NO
SEQ-CK Specified
YES
9A
NO
Set FIL-SEQ-ORD-CK-0 FIL-SEQ-ORD-SW =2
FIL-ORD-SW=1
9B
FIL-12
12
Set FM-OPR-SW to 2
FIL-13
13
All Parameters Specified
YES
Exit
NO
Print: PARAMETERS NOT COMPLETE
Exit

(3) OMLPGRP

(a) Function. The function of OMLPGRP is to process the GROUP card. Keywords break up the GROUP card parameters into phrases. These phrases are edited. If errors are found in the specifications, diagnostics are provided. The information on the GROUP card is incorporated into the DST.

(b) Calling Sequence.

```
ENTRY  'OMGRP'  USING FM-DD  CM-DD
CALLS  'FMPRT'  USING PRT-LINE  CC
```

(c) Program Description.

OMLPGRP.
Initializes counters, switches, and hold areas. Tests to see if group card is valid at this point.

GRP-1A.
Identifies keywords and branches to appropriate sections of coding which process the various specified phrases.

GRP-3, GRP-3A and PERFORM-GRP-SCAN.
These paragraphs edit the GROUP-ID phrase.

GRP-4.
Enters GRP-ID-NAME, GRP-ID-LEN and GRP-NAME-CK into GRP-ID-LIST-TEMP for later use in providing GRP-ID-HOP for each ID field in the GRP-ID-LIST of the DST.

GRP-5, GRP7.
Edits the sequence and order test fields when either is present. Branches to appropriate error print routine upon encountering an error.

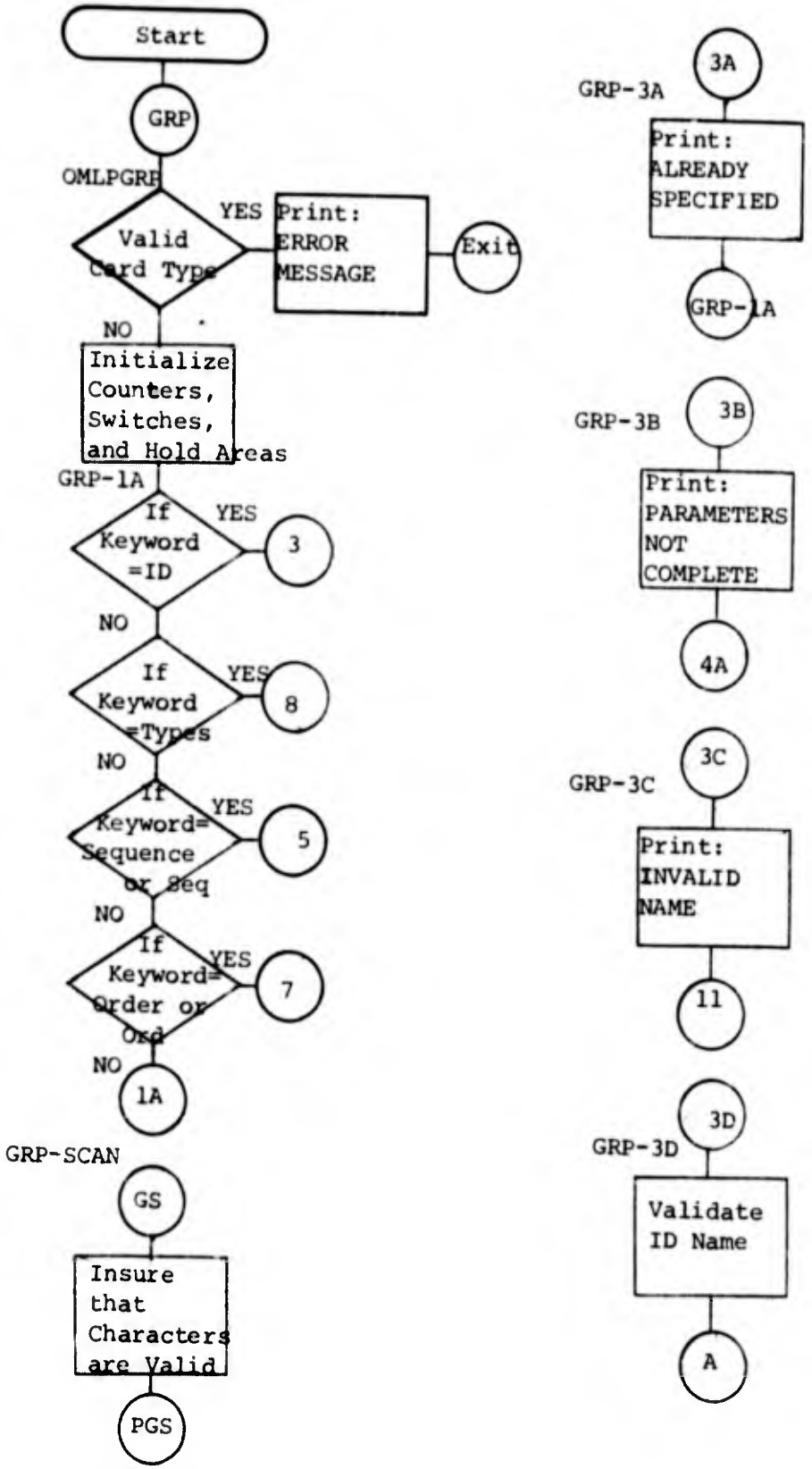
GRP-6.
Determines whether error action code is valid. Acceptable codes are G, C, or R. Any other code will cause a branch to GRP-6A where an error message is printed.

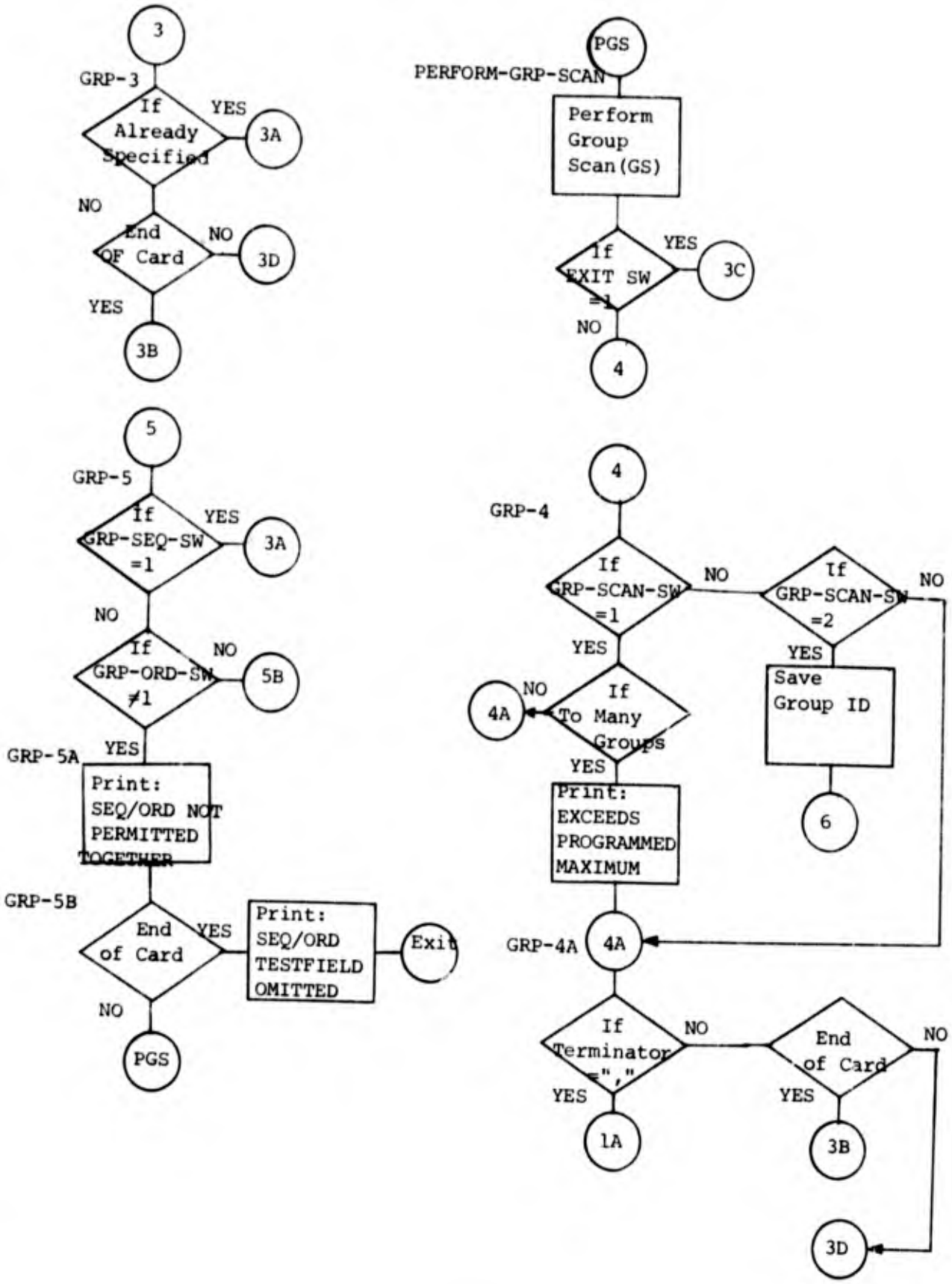
GRP-8, GRP-8A.
Edits the group types phrase. Enters valid group types into GRP-TYPES-LIST-TEMP.

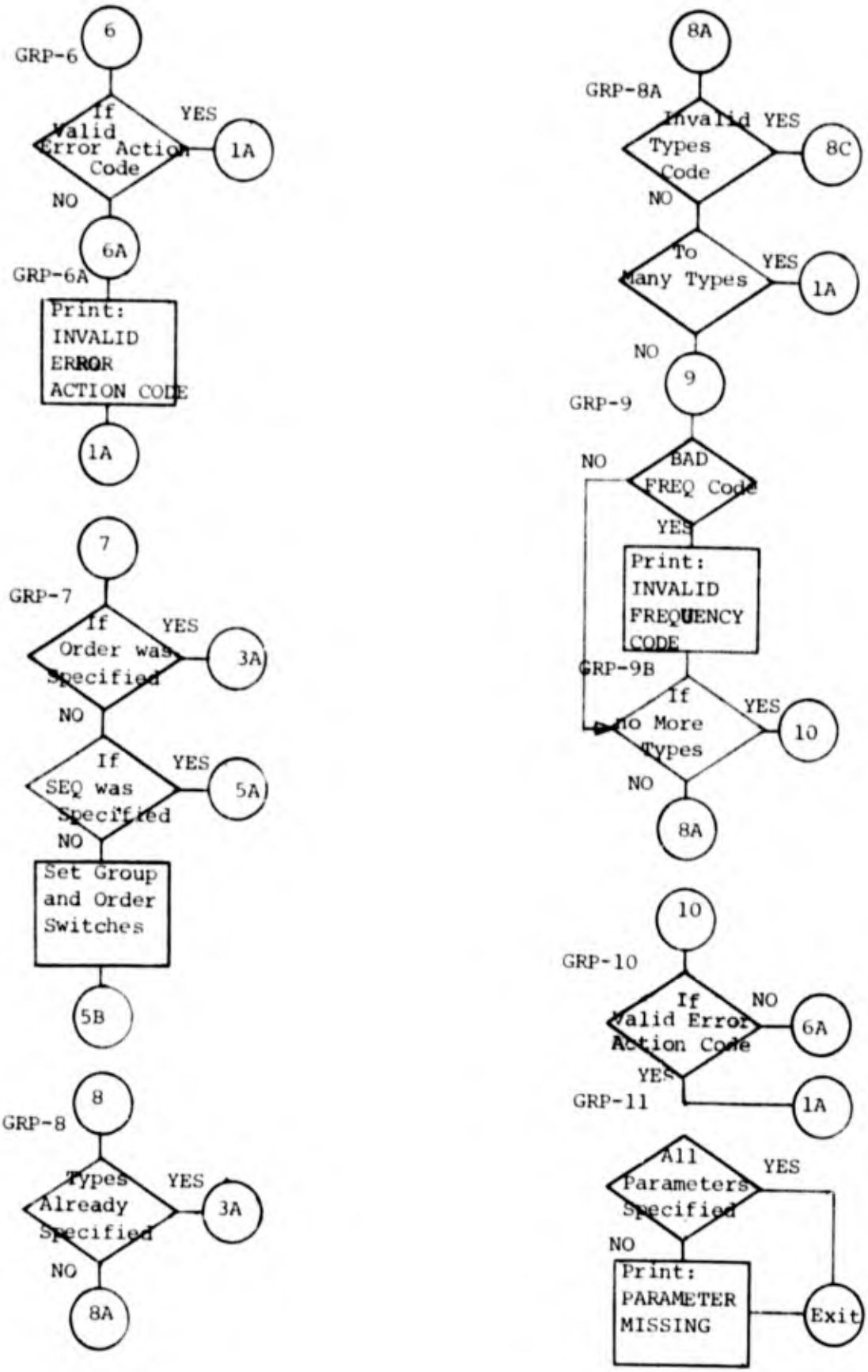
GRP-9.
Stores the frequency code in GRP-FREQ in the GRP-TYPES-LIST.

GRP-10.
Stores the error action code in GRP-FREQ-ERROR.

(d) Limitations. There is a maximum of 15 field names that can be specified in the GRP-ID phrase. There is a maximum of 50 GROUP TYPE names that can be specified. This card is invalid when file code none is specified.


Start
GRP
OMLPGRP
Valid Card Type
YES
Print: ERROR MESSAGE
Exit
NO
Initialize Counters, Switches, and Hold Areas
GRP-1A
If Keyword =ID
YES
3
NO
If Keyword =Types
YES
8
NO
If Keyword= Sequence or Seq
YES
5
NO
If Keyword= Order or Ord
YES
7
NO
1A
GRP-SCAN
GS
Insure that Characters are Valid
PGS
GRP-3A
3A
Print: ALREADY SPECIFIED
GRP-1A
GRP-3B
3B
Print: PARAMETERS NOT COMPLETE
4A
GRP-3C
3C
Print: INVALID NAME
11
GRP-3D
3D
Validate ID Name
A


3
GRP-3
If Already Specified
YES
3A
NO
End OF Card
NO
3D
YES
3B
PGS
PERFORM-GRP-SCAN
Perform Group Scan(GS)
If EXIT SW =1
YES
3C
NO
4
5
GRP-5
If GRP-SEQ-SW =1
YES
3A
NO
If GRP-ORD-SW ≠1
NO
5B
YES
GRP-5A
Print: SEQ/ORD NOT PERMITTED TOGETHER
GRP-5B
End of Card
YES
Print: SEQ/ORD TESTFIELD OMITTED
Exit
NO
PGS
4
GRP-4
If GRP-SCAN-SW =1
NO
If GRP-SCAN-SW =2
NO
YES
Save Group ID
6
If To Many Groups
NO
4A
YES
Print: EXCEEDS PROGRAMMED MAXIMUM
GRP-4A
4A
If Terminator =","
NO
End of Card
NO
YES
YES
1A
3B
3D


6
GRP-6
If Valid Error Action Code
YES
1A
NO
6A
GRP-6A
Print: INVALID ERROR ACTION CODE
1A
7
GRP-7
If Order was Specified
YES
3A
NO
If SEQ was Specified
YES
5A
NO
Set Group and Order Switches
5B
8
GRP-8
Types Already Specified
YES
3A
NO
8A
8A
GRP-8A
Invalid Types Code
YES
8C
NO
To Many Types
YES
1A
NO
9
GRP-9
NO
BAD FREQ Code
YES
Print: INVALID FREQUENCY CODE
GRP-9B
If no More Types
YES
10
NO
8A
10
GRP-10
If Valid Error Action Code
NO
6A
YES
GRP-11
1A
All Parameters Specified
YES
NO
Print: PARAMETER MISSING
Exit

(4) OMLPREC

(a) Function. The function of OMLPREC is to process the RECORD card. Keywords break up the RECORD card into phrases. These phrases are edited. If errors are found in the specifications, diagnostics are provided. The information on the RECORD card is incorporated into the DST.

(b) Calling Sequence.

```
ENTRY 'OMREC'  USING FM-DD  OM-DD
CALL  'FMPRT'  USING PRT-LINE  CC
```

(c) Program Description.

OMLPREC.

Determines if card is valid at this point. If file code none was specified, an error message is printed and control is passed back to OMLPX. Prints warning message if record has no fields.

REC-1.

Initializes counters, switches, and hold areas.

REC-1A.

Identifies keywords and branches to appropriate section to process specified phase.

REC-2.

Edits and processes the record type phrase. If the record type was previously specified on the group level it has been stored as a GRP-TYPE in the GRP-TYPES-LIST. If not it will be added to the GRP-TYPES-LIST and given a default of 0 in GRP-FREQ. In either case the location of the record type in the GRP-TYPES-LIST will be stored in REC-GRP-POINT in the REC-TYPE-LIST.

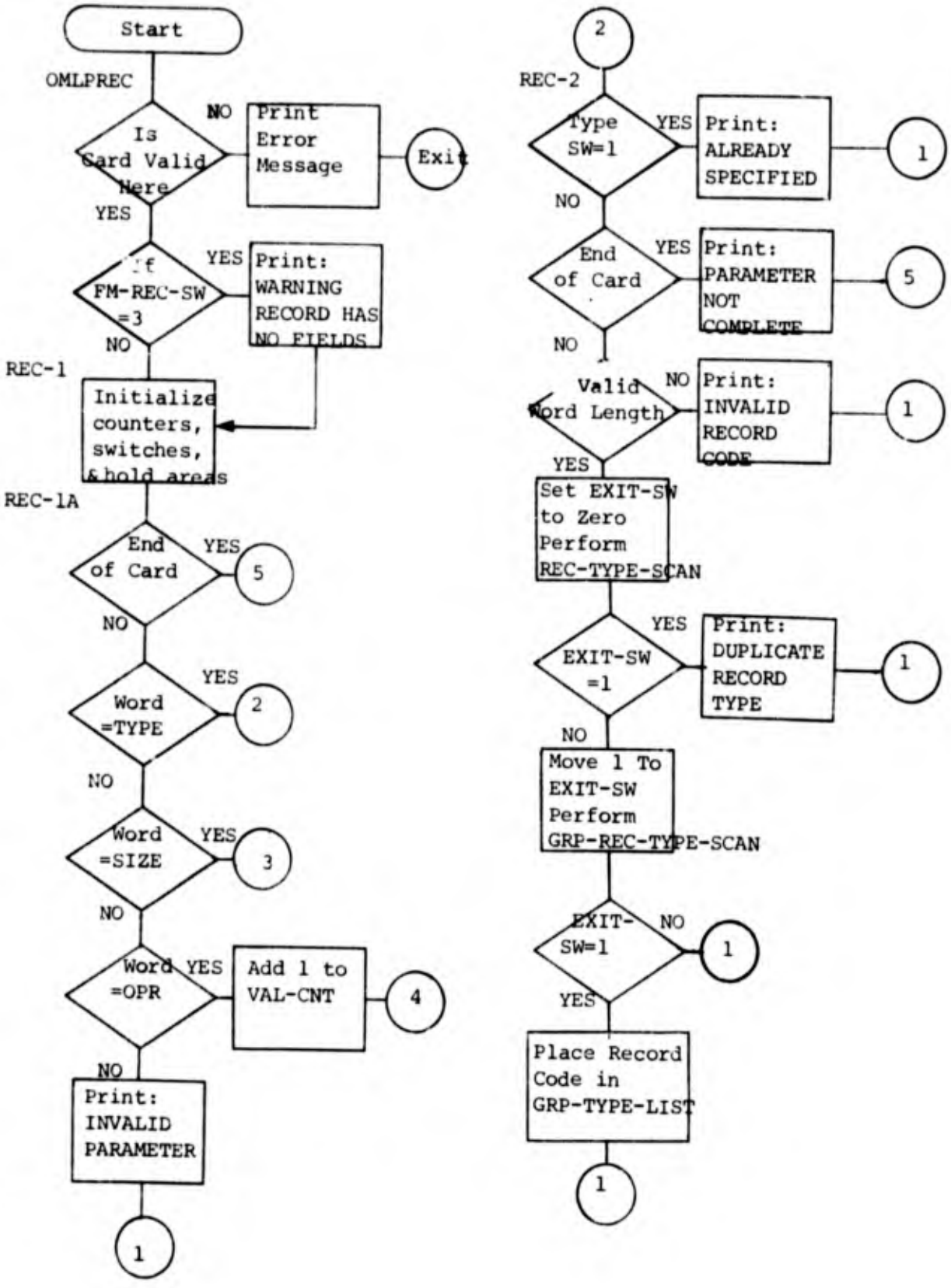
REC-3.

Edits the size parameter and stores it in REC-SIZE in the REC-TYPE-LIST.

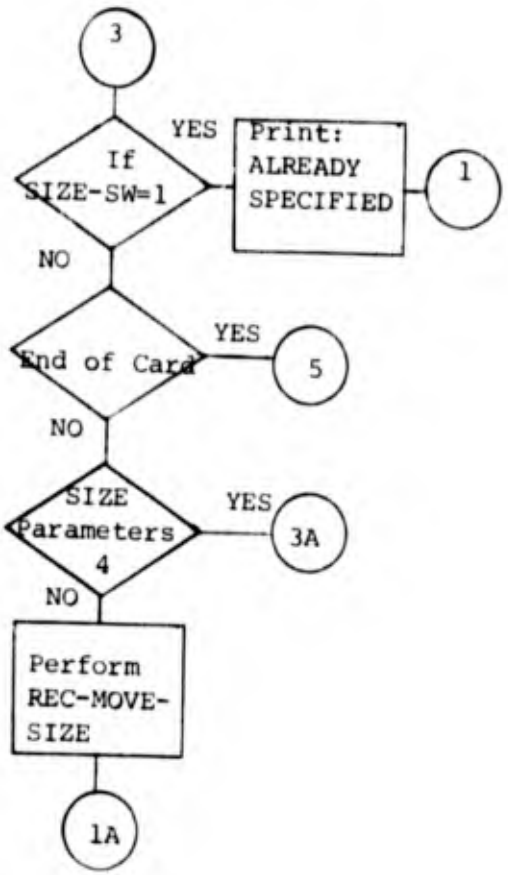
REC-4.

Sets FM-OPR-SW if an operator is present.

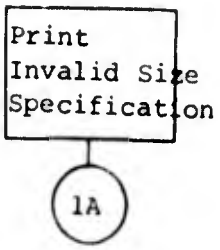
(d) Limitations. There is a maximum of 50 records in the REC-TYPE-LIST. The SIZE defaults to 80 when no size is specified. When specified size cannot exceed 9999.


Start
OMLPREC
Is Card Valid Here
NO
Print Error Message
Exit
YES
If FM-REC-SW =3
YES
Print: WARNING RECORD HAS NO FIELDS
NO
REC-1
Initialize counters, switches, & hold areas
REC-1A
End of Card
YES
5
NO
Word =TYPE
YES
2
NO
Word =SIZE
YES
3
NO
Word =OPR
YES
Add 1 to VAL-CNT
4
NO
Print: INVALID PARAMETER
1
2
REC-2
Type SW=1
YES
Print: ALREADY SPECIFIED
1
NO
End of Card
YES
Print: PARAMETER NOT COMPLETE
5
NO
Valid Word Length
NO
Print: INVALID RECORD CODE
1
YES
Set EXIT-SW to Zero
Perform REC-TYPE-SCAN
EXIT-SW =1
YES
Print: DUPLICATE RECORD TYPE
1
NO
Move 1 To EXIT-SW
Perform GRP-REC-TYPE-SCAN
EXIT-SW=1
NO
1
YES
Place Record Code in GRP-TYPE-LIST
1

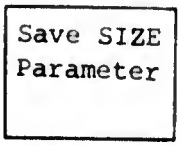
REC-3



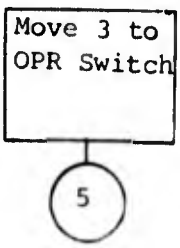
REC-3A



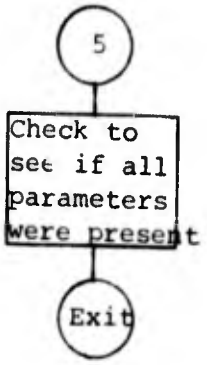
REC-MOVE-SIZE



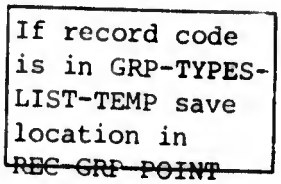
REC-4



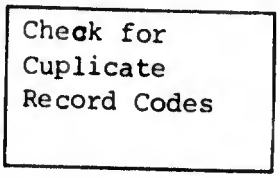
REC-5



REC-GRP-TYPE-SCAN



REC-TYPE-SCAN



(5) OMLPFLD

(a) Function. The function of OMLPFLD is to identify keywords on the FIELD card and call appropriate subprograms to process each phrase.

(b) Calling Sequence.

```
ENTRY 'OMFLD'  USING FM-DD  LP-DD  OM-DD
CALL  'OMINT'  USING FM-DD  OM-DD  LP-DD
CALL  'OMPSS'  USING FM-DD  OM-DD
CALL  'OMRNG'  USING FM-DD  OM-DD  LP-DD
CALL  'OMPRO'  USING FM-DD  OM-DD
CALL  'OMVAL'  USING FM-DD  OM-DD  LP-DD
CALL  'OMSBR'  USING FM-DD  OM-DD
CALL  'FMPRT'  USING PRT-LINE  CC
```

(c) Program Description.

OMLPFLD.

Tests the legality of a FIELD card and calls OMINT to do basic initialization for the FIELD card.

FLD-5.

Recognizes keywords and branches to appropriate sections of coding. In all cases except for FLD-23, these sections perform calls to subprograms, which edit the phrase, make entries in the DST, and return.

FLD-23.

Sets FM-OPR-SW if an OPR phrase was identified. OMLPX will call OMLPOPR.

FLD-6.

Calls OMLPPSS when the field is periodic.

FLD-8.

Calls OMLPRNG to process the RANGE phrase.

FLD-12.

Calls OMLPPRO to process the PROFILE phrase.

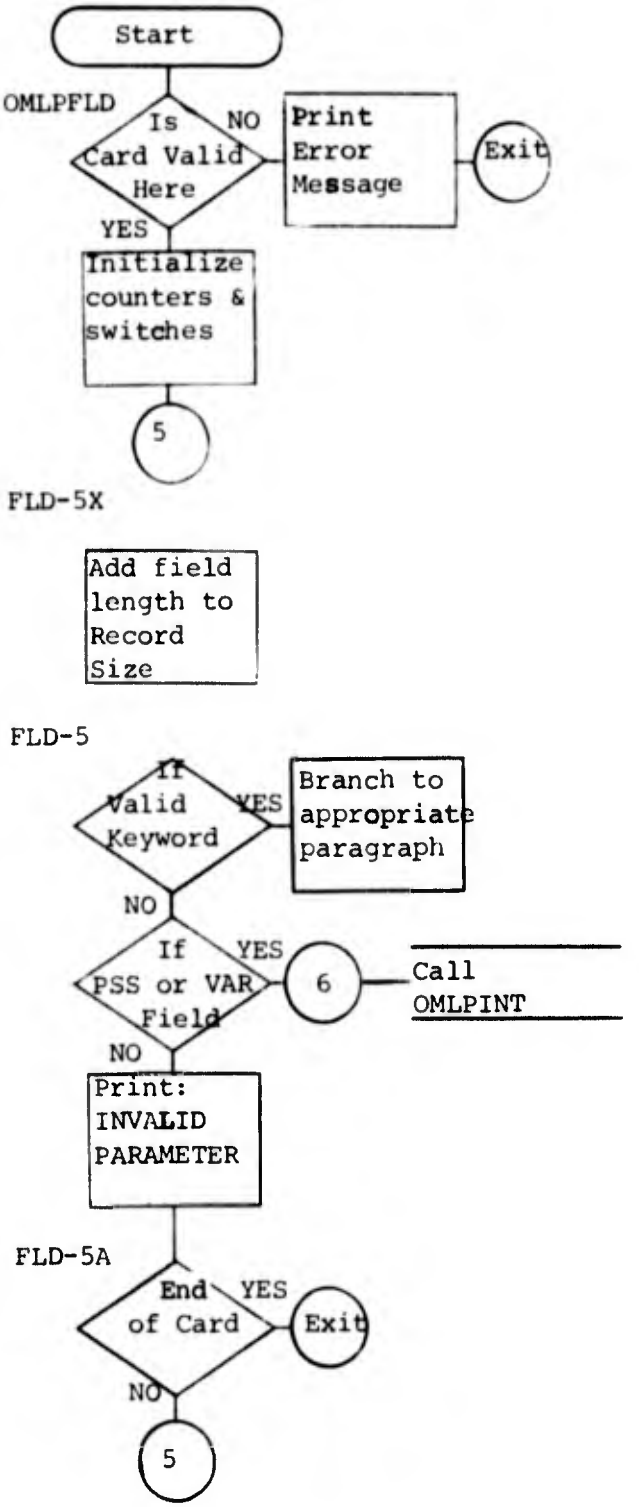
FLD-14.

Calls OMLPVAL to process the VALUE phrase.

FLD-21.

Calls OMLPSBR to process the SUBCON or SUBCHK phrases.

(d) Limitations. None.


Start
OMLPFLD
Is Card Valid Here
NO
Print Error Message
Exit
YES
Initialize counters & switches
5
FLD-5X
Add field length to Record Size
FLD-5
If Valid Keyword
YES
Branch to appropriate paragraph
NO
If PSS or VAR Field
YES
6
Call OMLPINT
NO
Print: INVALID PARAMETER
FLD-5A
End of Card
YES
Exit
NO
5

(6) OMLPINT

(a) Function. OMLPINT performs basic initialization prior to processing the field card. This program examines only the first two or three words on the card. It determines if the field is in the FFT, validates the field name and saves its relative HOP and length. If the VAR option is specified, field HOP and length are processed in OMPSS.

(b) Calling Sequence.

```
ENTRY 'OMINT'    USING FM-DD  OM-DD  LP-DD
CALL  'FMPRT'    USING PRT-LINE  CC
CALL  'MOVCMP'   USING FM-DATA-WORD  CNT10  LENGTHA
                       FLD-LEN       CNT11  LENGTHB
                       CONTINUE-SAVE
```

(c) Program Description.

FLD-1.

Initializes the first record if record code none was specified on the FILE card.

FLD-1A.

Initializes counters, switches, and hold areas. Edits field name and saves it if it's valid.

FLD-2.

Searchs LR11 to determine if field is defined in the FFT. The position in LR11 is saved in FLD-LR2-POINT in the field-list of the DST. FLD-HOP and FLD-LEN are initialized at zero.

FLD-2A.

Stores the FLD-HOP and FLD-LEN from the given HOP and LOP.

FLD-3.

Determines the location of the field in the FFT and stores the location in FLD-LR2-POINT in the field-list.

FLD-4.

Compares DSD FIELD-SIZE to FFT FIELD-SIZE. If the DSD FIELD is the smaller, a warning message is printed; if it's larger, FLD-SUBCON-CK is set to 1. In the latter case a SUBCON must be specified.

FLD-4A.

Compares the field name with the ID lists on the file and group levels.

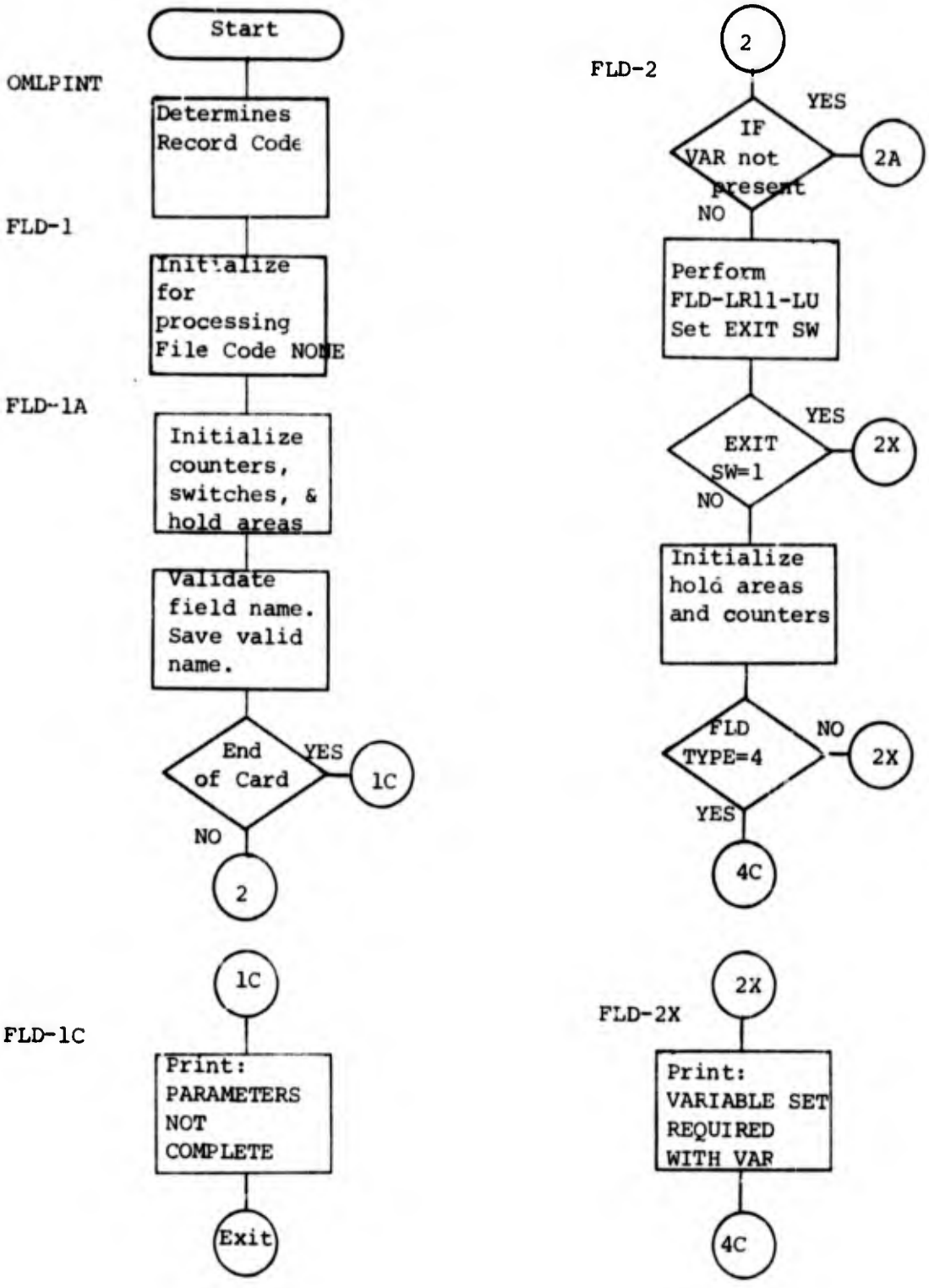
FIL-REC-IDS.

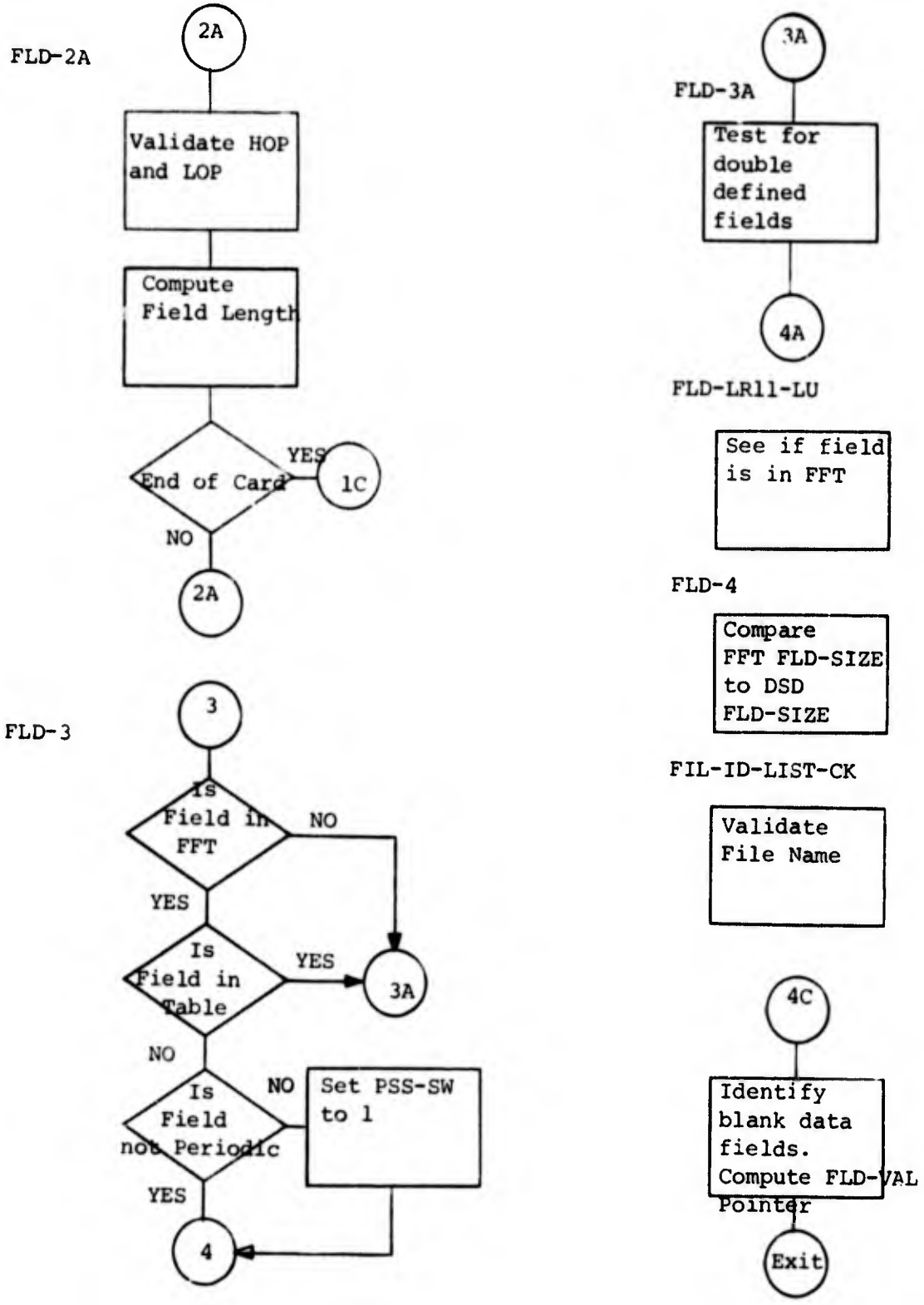
Computes the subscript that points to the first field of a record type.

FLD-4C.

Identifies those fields that may have blank input during the update phase. If blanks are allowed, FLD-SKIP is set to 1; otherwise, it is set to 2.

(d) Limitations. The REC-IDS-LIST has a limit of 100 entries. The FIELD-LIST has a limit of 300 entries.


Start
OMLPINT
Determines Record Code
FLD-1
Initialize for processing File Code NONE
FLD-1A
Initialize counters, switches, & hold areas
Validate field name. Save valid name.
End of Card
YES
1C
NO
2
1C
FLD-1C
Print: PARAMETERS NOT COMPLETE
Exit
2
FLD-2
IF VAR not present
YES
2A
NO
Perform FLD-LR11-LU Set EXIT SW
EXIT SW=1
YES
2X
NO
Initialize hold areas and counters
FLD TYPE=4
NO
2X
YES
4C
2X
FLD-2X
Print: VARIABLE SET REQUIRED WITH VAR
4C


FLD-2A
2A
Validate HOP and LOP
Compute Field Length
End of Card
YES
1C
NO
2A
FLD-3
3
Is Field in FFT
NO
YES
Is Field in Table
YES
3A
NO
Is Field not Periodic
NO
Set PSS-SW to 1
YES
4
FLD-3A
3A
Test for double defined fields
4A
FLD-LR11-LU
See if field is in FFT
FLD-4
Compare FFT FLD-SIZE to DSD FLD-SIZE
FIL-ID-LIST-CK
Validate File Name
4C
Identify blank data fields. Compute FLD-VAL Pointer
Exit

(7) OMLPOPR

(1) The purpose of OMLPOPR is to process a part of the OPR phrase and call other subroutines to process remaining elements of the phrase.

(2) Calling Sequence.

```
ENTRY 'OMOPR'    USING FM-DD  LP-DD  OM-DD
CALL  'FMPRT'    USING PRT-LINE  CC
CALL  'OMCPR2'   USING FM-DD  OM-DD  LP-DD
CALL  'OMOPR3'   USING FM-DD  OM-DD  LP-DD
CALL  'OMOPR4'   USING FM-DD  OM-DD  LP-DD
```

(3) Program Description.

OMLPOPR and OPR-1.
Initializes switches.

OPR-1A.
Checks the value set in FM-OPR-SW and branches to initilize the file, record, or field level as appropriate.

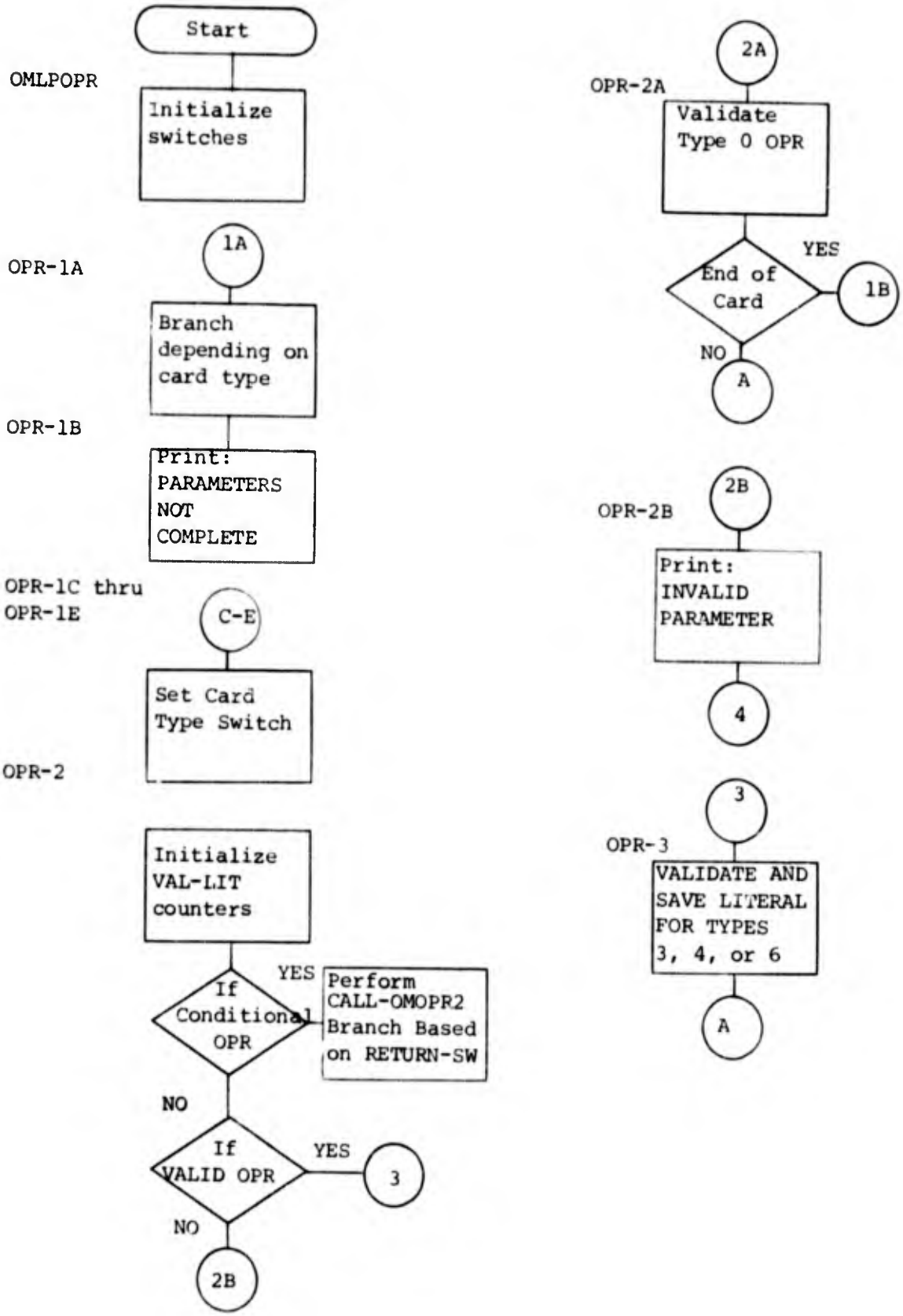
OPR-2.
Sets VAL-TYPE to 6 in the VAL-LIST for an OPR and sets the HOP for the OPR literal in the data string VAL-LIT-AREA, in VAL-LIT-HOP in the VAL-LIST. VAL-LIT-LEN is incremented throughout the program as the different formats have varying literal lengths. If the second word in the phrase is followed by a comma, the OPR is conditional and FM-OPR-COND-SW is set to 1 and OMLPOP2 is called. Upon return from OMLPOP2 the OPR-RETURN-SW is a 1, 2, 3, or 4. A branch is taken either to OPR-1B which prints an OPR-phrase in complete diagnostic, or CALL-OMOPR3 which calls OMOPR3 and OMOPR4, or OPR-4 which completes processing of the parameter, or OPR-3 which processes a literal, depending on OPR-RETURN-SW.

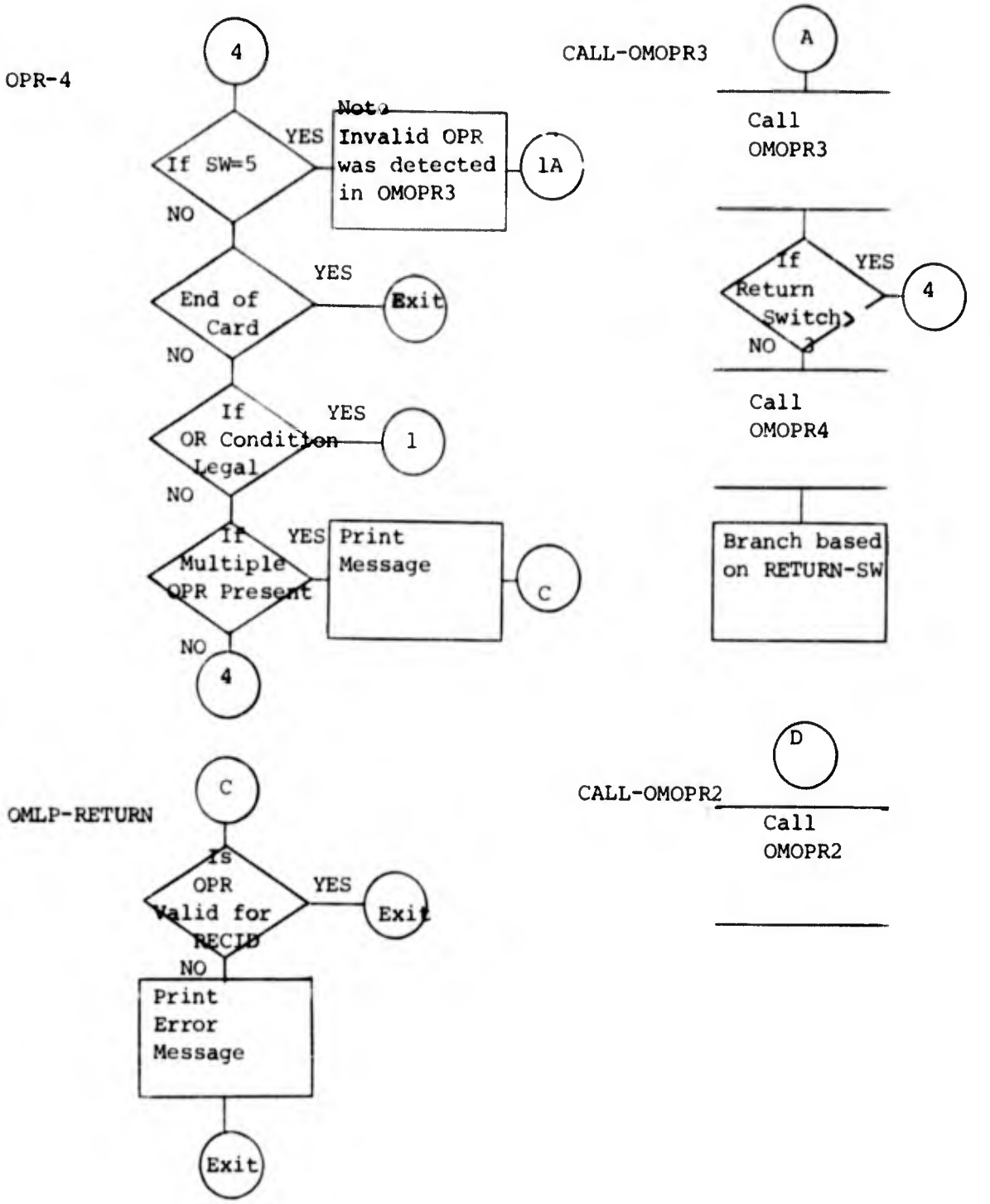
CALL-OMOPR3.
If the format is type 1, OMOPR3 is called followed by a call to OMOPR4. The return codes from this double call are 1, 2, and 3 indicating a multiple OPR specification, incomplete OPR phrase, and complete OPR phrase respectively. When multiple OPR specifications are given in a single phrase, those specifications following the first one are coded as OPR-TYPE=6.

OPR-3.
Edits the literal in the OPR specification and moves it into the VAL-LIT-AREA and branches to paragraph CALL-OMOPR3.

OPR-4.
Initiates processing of another OPR phrase if the error action code of the proceeding OPR phrase was 0 for an OR condition. OMLPOPR verifies that conditional and unconditional OPR's have not been mixed for the same field and record using the VAR option.

(d) Limitations. OMLPOPR provides entries primarily to the VAL-LIST and VAL-LIT-AREA. The number of OPR specifications is limited by the 50 words per card in the DSD. There may be 600 entries in the VAL-LIST including OPR's and validity checks. The VAL-LIT-AREA cannot exceed 5000 characters.


Start
OMLPOPR
Initialize switches
1A
OPR-1A
Branch depending on card type
OPR-1B
Print: PARAMETERS NOT COMPLETE
OPR-1C thru OPR-1E
C-E
Set Card Type Switch
OPR-2
Initialize VAL-LIT counters
If Conditional OPR
YES
Perform CALL-OMOPR2 Branch Based on RETURN-SW
NO
If VALID OPR
YES
3
NO
2B
2A
OPR-2A
Validate Type 0 OPR
End of Card
YES
1B
NO
A
2B
OPR-2B
Print: INVALID PARAMETER
4
3
OPR-3
VALIDATE AND SAVE LITERAL FOR TYPES 3, 4, or 6
A


OPR-4
4
If SW=5
YES
Not a Invalid OPR was detected in OMOPR3
1A
NO
End of Card
YES
Exit
NO
If OR Condition Legal
YES
1
NO
If Multiple OPR Present
YES
Print Message
C
NO
4
CALL-OMOPR3
A
Call OMOPR3
If Return Switch > 3
YES
4
NO
Call OMOPR4
Branch based on RETURN-SW
CALL-OMOPR2
D
Call OMOPR2
OMLP-RETURN
C
Is OPR Valid for RECID
YES
Exit
NO
Print Error Message
Exit

(8) OMLPOP2

(a) Function. The function of OMLPOP2 is to process the second parameter of a conditional operator phrase. If the operator is a Type D, the third parameter is also processed.

(b) Calling Sequence.

```
ENTRY 'OMOPR2'  USING FM-DD  OM-DD  LP-DD
CALL  'FMPRT'   USING PRT-LINE  CC
```

(c) Program Description.

OMLPOPR2.
Checks to see if OPR Type B is present.

OPR-SCAN.
Checks for alpha field.

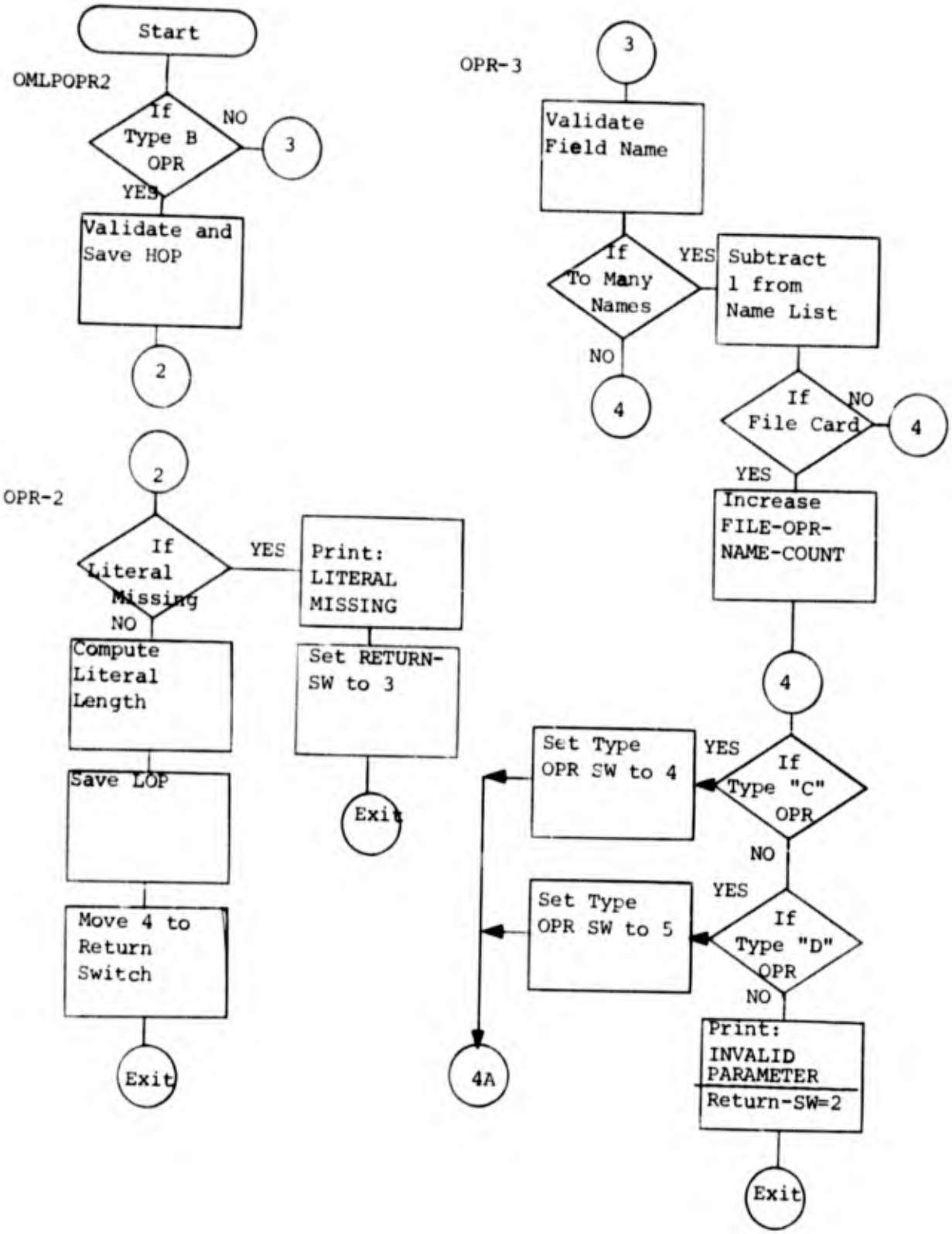
OPR-2.
Determines the length and low order position of the literal. Saves the length in VAL-LIT-LEN and puts the literal in the VAL-LIT-AREA.

OPR-3.
If the field is alphabetic edits the field name and stores it in the OPR-NAME-LIST-TEMP. Increments VAL-LIT-LEN. If the operator is a file operator phrase, the next position in VAL-LIT-AREA is stored in FIL-OPR-LIT-POINT.

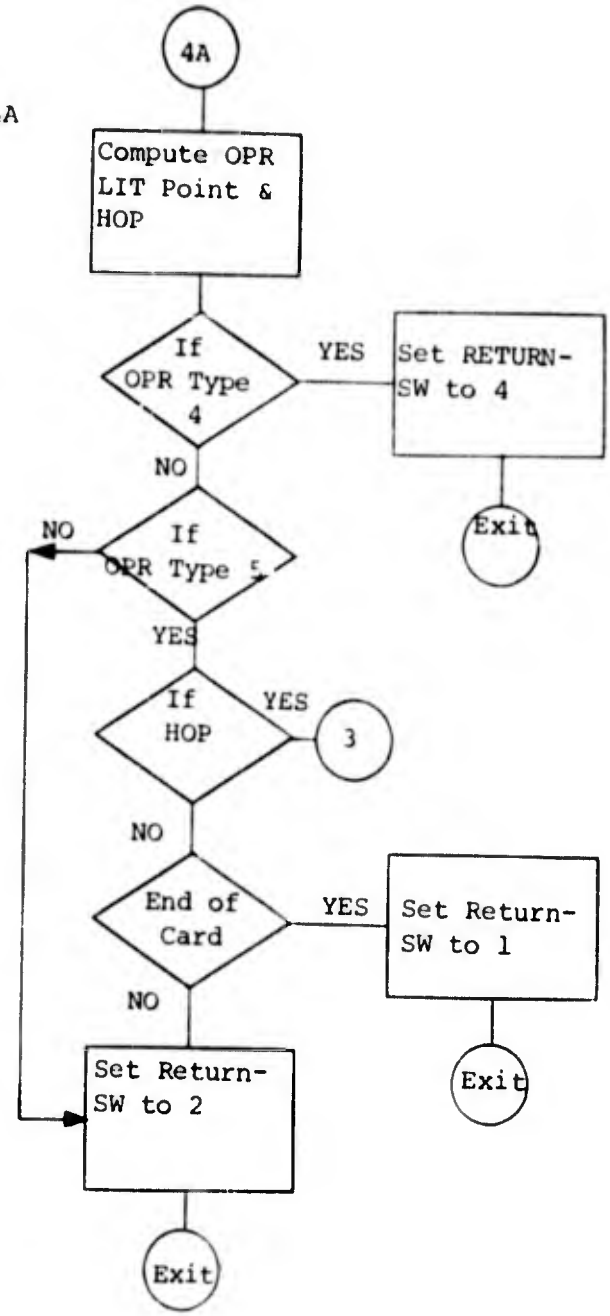
OPR-4A.
Stores the next available location in VAL-LIT-AREA in OPR-VAL-LIT-HOP in the OPR-NAME-LIST-TEMP.

OPR-4.
Validates the operator for Type C or D. If invalid, an error message is printed.

(d) Limitations. The OPR-NAME-LIST-TEMP and FIL-OPR-NAME-LIST can have no more than 40 entries in them.


Start
OMLPOPR2
If Type B OPR
NO
3
YES
Validate and Save HOP
2
2
OPR-2
If Literal Missing
YES
Print: LITERAL MISSING
NO
Compute Literal Length
Set RETURN-SW to 3
Save LOP
Exit
Move 4 to Return Switch
Exit
3
OPR-3
Validate Field Name
If To Many Names
YES
Subtract 1 from Name List
NO
4
If File Card
NO
4
YES
Increase FILE-OPR-NAME-COUNT
4
Set Type OPR SW to 4
YES
If Type "C" OPR
NO
Set Type OPR SW to 5
YES
If Type "D" OPR
NO
4A
Print: INVALID PARAMETER
Return-SW=2
Exit

OPR-4A


4A
Compute OPR LIT Point & HOP
If OPR Type 4
YES
Set RETURN-SW to 4
Exit
NO
If OPR Type 5
NO
YES
If HOP
YES
3
NO
End of Card
YES
Set Return-SW to 1
Exit
NO
Set Return-SW to 2
Exit

(9) OMLPOP3

(a) Function. The function OMLPOP3 is to continue processing of the OPR phrase. Specifically, this subprogram converts the operation in the OPR phrase to a two digit numeric code, and insures that the operation specified meets required logical prerequisites.

(b) Calling Sequence.

```
ENTRY 'OMOPR3'  USING FM-DD  OM-DD  LP-DD
CALL  'FMPRT'   USING PRT-LINE  CC
```

(c) Program Description.

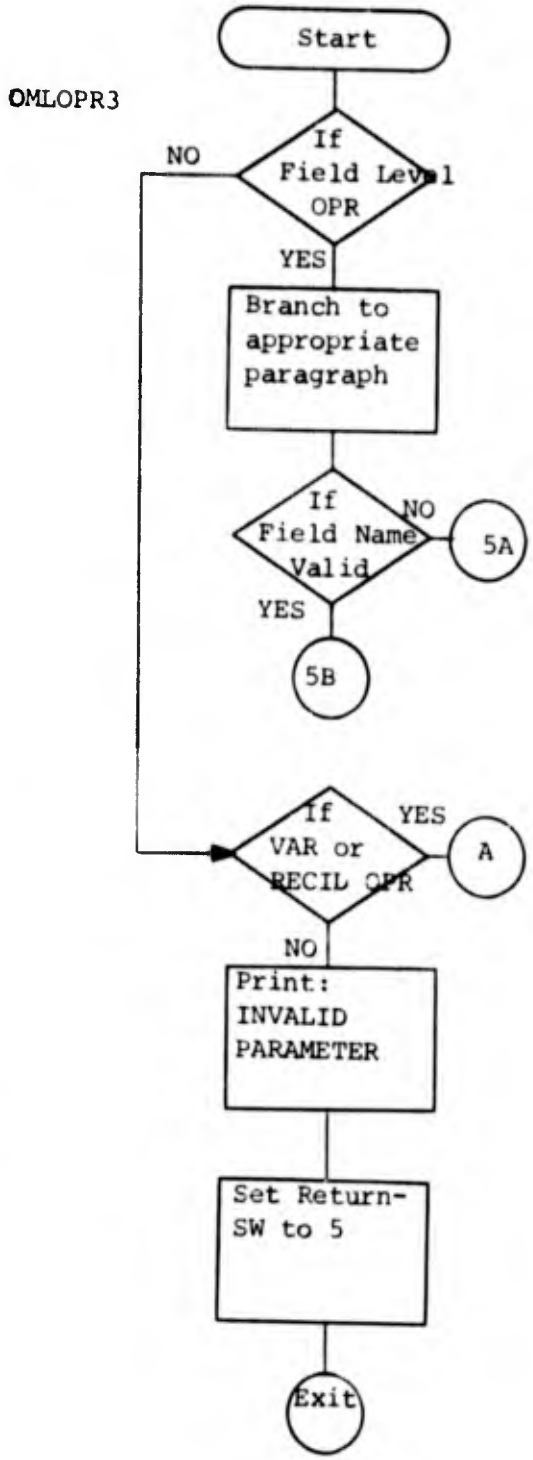
OMLPOPR3.
Converts the opcode to a two digit numeric code. If opcode is invalid for the field, print an error message.

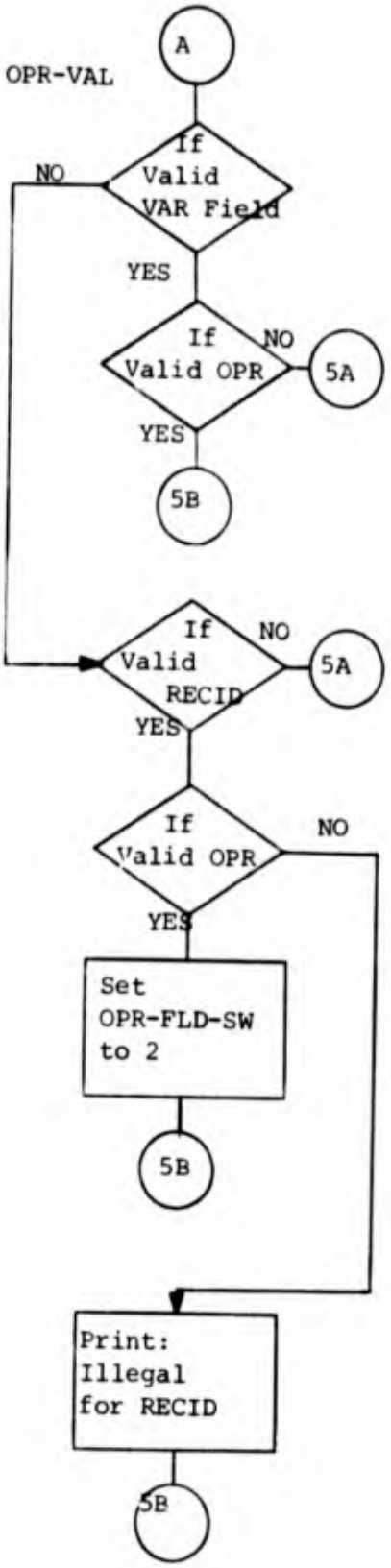
OPR-VAL.
For the special operations, OPR-VAL insures that logic prerequisites are met (i.e., GER cannot be use when VAR option is present).

OPR-13 thru OPR-13E.
When conventional operation codes are used these paragraphs process logic prerequisites.

OPR-5, OPR-5A, OPR-5B.
Sets OPR-RETURN-SW to 1, 2, or 3 respectively and returns control to the calling program.

(d) Limitations. None.


OMLOPR3
Start
If Field Level OPR
NO
YES
Branch to appropriate paragraph
If Field Name Valid
NO
5A
YES
5B
If VAR or RECIL OPR
YES
A
NO
Print: INVALID PARAMETER
Set Return-SW to 5
Exit


OPR-VAL
A
If Valid VAR Field
NO
YES
If Valid OPR
NO
5A
YES
5B
If Valid RECID
NO
5A
YES
If Valid OPR
NO
YES
Set OPR-FLD-SW to 2
5B
Print: Illegal for RECID
5B

(10) OMLPOP4

(a) Function. The function of OMLPOP4 is to continue processing the OPR phrase. More specifically, it further validates the OPR and edits the error action code. If the OPR is good after this final edit it is placed in the DST.

(b) Calling Sequence.

```
ENTRY  'OMOPR4'  USING FM-DD  OM-DD  LP-DD
CALL   'FMPRT'   USING PRT-LINE  CC
CALL   'MOVNUM'  USING FM-DATA-WORD  CNT10
                       LENGTHA VAL-LIT-AREA
                       CNT11   LENGTHB
                       EXIT-SW2
```

(c) Program Description.

OMLPOPR4.

Upon entering OMLPOPR4 further OPR processing is determined by a branch to OPR-5, OPR-5A or OPR-5B depending on the value of OPR-RETURN-SW.

OPR-5.

If the OPR is on the field card level and is present in the FFT and is not a VSC or PSC and OPR is not a control field with an opcode of GENU or GECU then it is legal.

OPR-5B.

This paragraph validates the OPR for the VAR, FIELD, and control field levels. It identifies OPR types 1 and 2 and saves the internal numeric operation code.

OPR-5D.

This paragraph edits the error action code and stores a numeric code in VAL-ERR in VAL-LIST. If code is invalid, an error message is printed.

OPR-5E.

If allowable blank fields have been specified, set appropriate level switches. When OPR-SUP-SW is 0, no allowable blank fields were specified for the DSD.

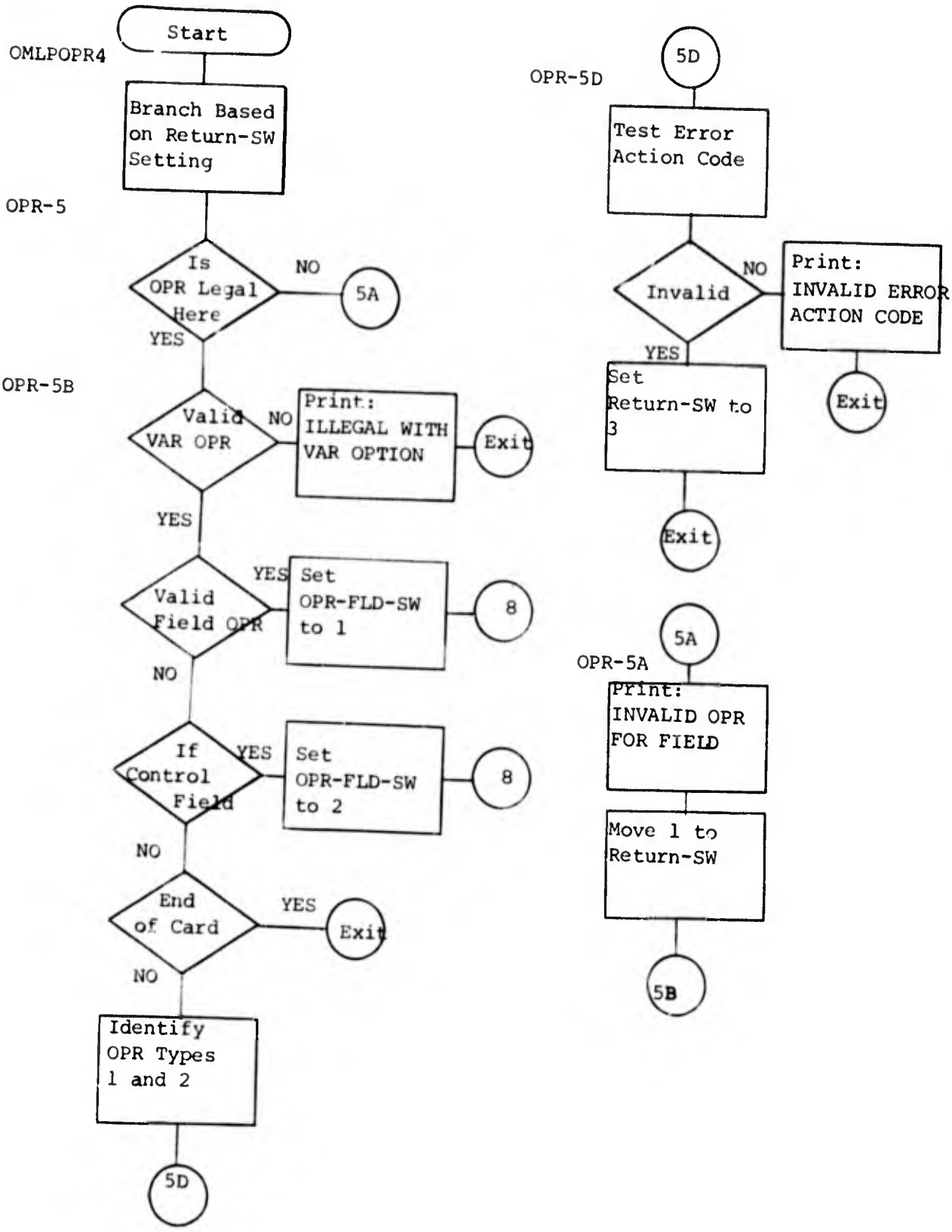
OPR-6B.

This paragraph validates the hOP and LOP and moves them into the VAL-LIT-AREA.

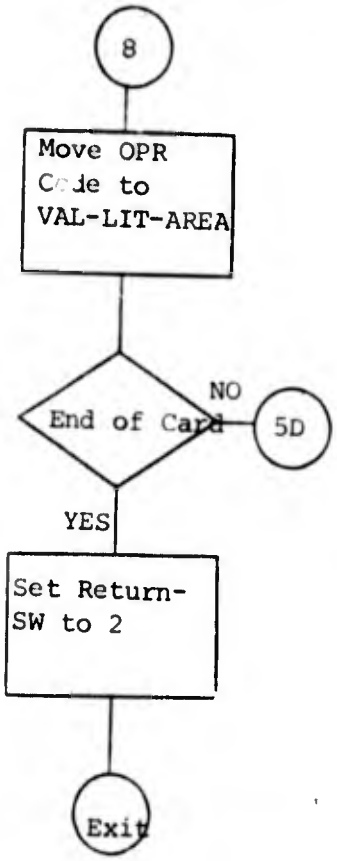
OPR-8.

Once an OPR on the FIELD card has passed the edit thru paragraph OPR-5B, a branch is made to OPR-8 where the interal numeric code is moved to the VAL-LIT-AREA.

(d) Limitations. OMLPOP4 makes entries in VAL-LIST and VAL-LIT-AREA. There may be 600 entries in VAL-LIST and VAL-LIT-AREA cannot exceed 5000 characters.


OMLPOPR4
Start
Branch Based on Return-SW Setting
OPR-5
Is OPR Legal Here
NO
5A
YES
OPR-5B
Valid VAR OPR
NO
Print: ILLEGAL WITH VAR OPTION
Exit
YES
Valid Field OPR
YES
Set OPR-FLD-SW to 1
8
NO
If Control Field
YES
Set OPR-FLD-SW to 2
8
NO
End of Card
YES
Exit
NO
Identify OPR Types 1 and 2
5D
OPR-5D
5D
Test Error Action Code
Invalid
NO
Print: INVALID ERROR ACTION CODE
Exit
YES
Set Return-SW to 3
Exit
5A
OPR-5A
Print: INVALID OPR FOR FIELD
Move 1 to Return-SW
5B

OPR-8


8
Move OPR
Code to
VAL-LIT-AREA
End of Card
NO
5D
YES
Set Return-
SW to 2
Exit

(11) OMLPRNG

(a) Function. OMLPRNG edits the RANGE check phrase insuring that the user format is correct. If errors are detected in this format, error messages are given. Information from this phrase is validated, interpreted and stored in the DST.

(b) Calling Sequence.

```
ENTRY 'OMRNG'   USING FM-DD  OM-DD
CALL  'FMPRT'   USING PRT-LINE  CC
CALL  'MOVNUM'  USING FM-DATA-LINE  CNT10
                      LENGTHA
                      VAL-LIT-AREA  CNT11
                      LENGTHB     EXIT-SW2
```

(c) Program Description.

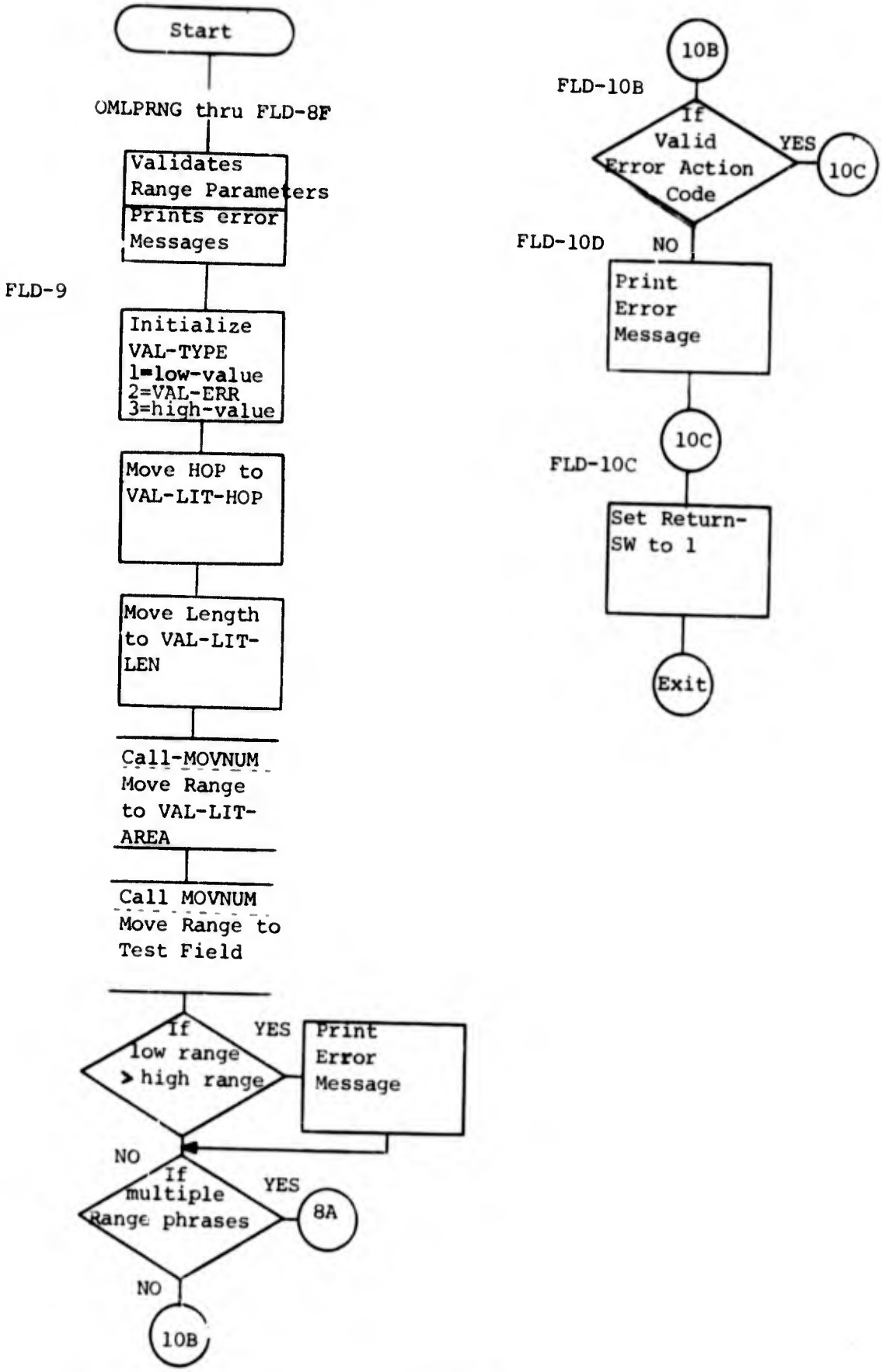
FLD-9.

Initializes VAL-TYPE ("1" for low range and "9" for high range) and VAL-ERR ("8") in the VAL-LIST. FLD-VAL-CT in the FIELD-LIST is incremented for each validity check. MOVNUM is called to move the Range Literal to the VAL-LIT-AREA. Its HOP is set in the VAL-LIT-HOP and its length in VAL-LIT-LEN in the VAL-LIST.

FLD-10B.

Moves the error action code to VAL-ERR in the VAL-LIST.

(d) Limitations. There is no limit to the number of range checks on a field, but only 50 words may be specified on a field. The VAL-LIT-AREA cannot exceed 5000 characters and the VAL-LIST has a maximum of 500 entries. This includes all validity checks and special operators.


Start
OMLPRNG thru FLD-8F
Validates Range Parameters
Prints error Messages
FLD-9
Initialize VAL-TYPE 1=low-value 2=VAL-ERR 3=high-value
Move HOP to VAL-LIT-HOP
Move Length to VAL-LIT-LEN
Call-MOVNUM
Move Range to VAL-LIT-AREA
Call MOVNUM
Move Range to Test Field
If low range > high range
YES
Print Error Message
NO
If multiple Range phrases
YES
8A
NO
10B
10B
FLD-10B
If Valid Error Action Code
YES
10C
FLD-10D
NO
Print Error Message
10C
FLD-10C
Set Return-SW to 1
Exit

(12) OMLPVAL

(a) Function. The function of OMLPVAL is to edit the value phrase on the field card insuring that the user format is correct. If errors are detected in this format, appropriate error messages are given. Information from this phrase is validated, interpreted, and stored in the DST. This program insures that table sizes are not exceeded.

(b) Calling Sequence.

```
ENTRY 'OMVAL'   USING FM-DD  OM-DD  LP-DD
CALL  'FMPRT'   USING PRT-LINE  CC
CALL  'MOVNUM'  USING FM-DATA-WORD  CNT10  LENGTHA
                      VAL-LIT-AREA  CNT11  LENGTHB
                      EXIT-SW2
CALL  'MOVALF'  USING FM-DATA-WORD (OR FM-LIT-DATA)
                      CNT10  LENGTHA
                      VAL-LIT-AREA
                      CNT11  LENGTHB
```

(c) Program Description.

FLD-14A.
Determines whether the validity check is a full field value or if it uses partial field addressing and branches to appropriate paragraph.

FLD-15.
If the value check does not use partial field addressing determine if the value is a literal or a nonliteral.

FLD-16.
If the value is a literal, see if it is numeric; if it is, see if the value is blank.

FLD-16B.
If the value is a nonliteral, determine whether the field is defined as numeric and if so move the data to the VAL-LIT-AREA by branching to a paragraph which performs FLD-MOVE-NUM. If the field is alpha, a branch is made to a paragraph which performs FLD-MOVE-ALPHA.

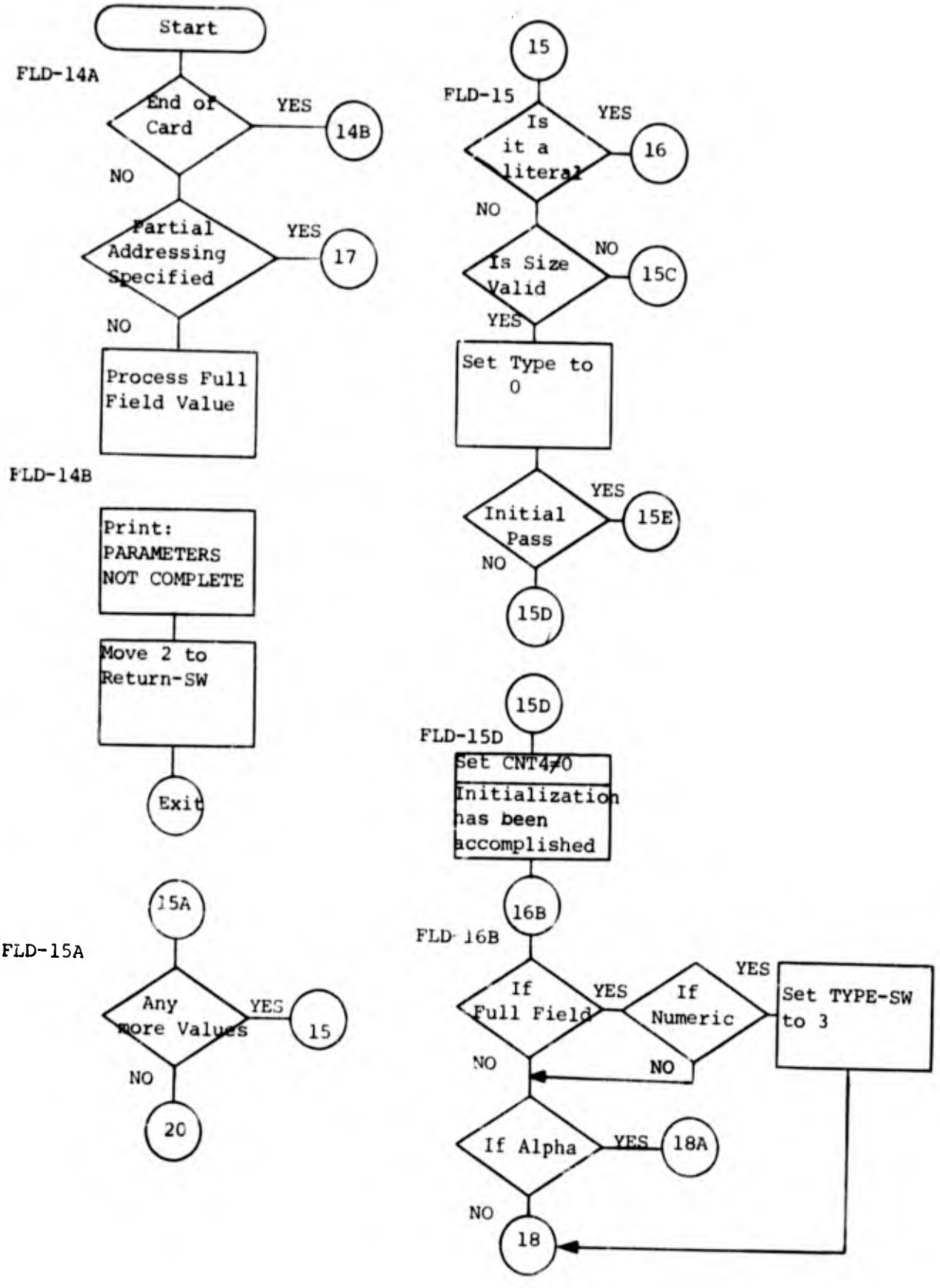
FLD-17.
If partial field addressing is used, move the relative HOP of the field to VAL-LIT-AREA and store the value using MOVALF.

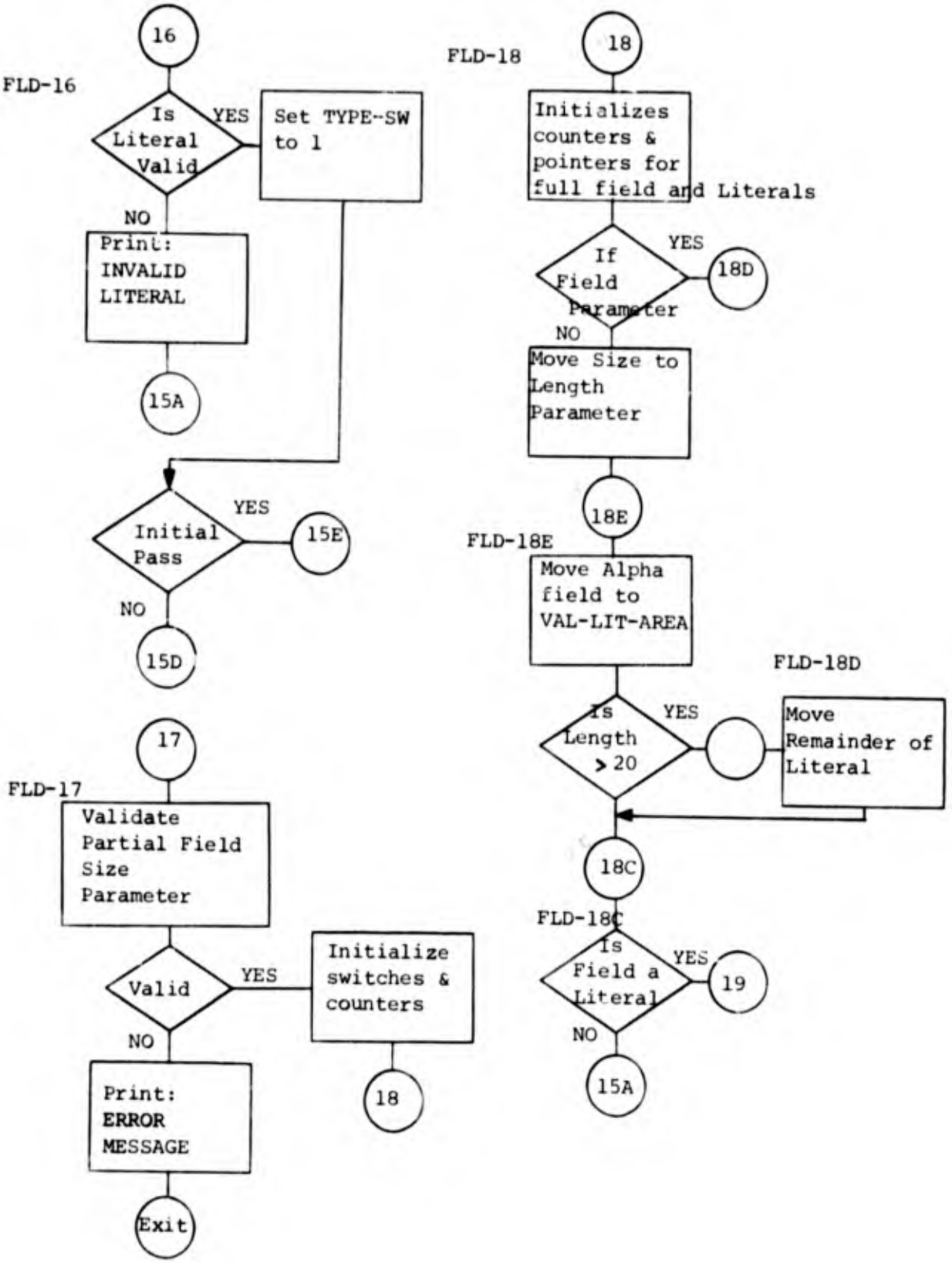
FLD-18 thru FLD-19.
These paragraphs perform further processing of the literal and full field parameters. They compute VAL-LIT-LEN and VAL-LIT-CNT, determines if the field is alpha or numeric and branch to the appropriate paragraph to move the field to the VAL-LIT-AREA.

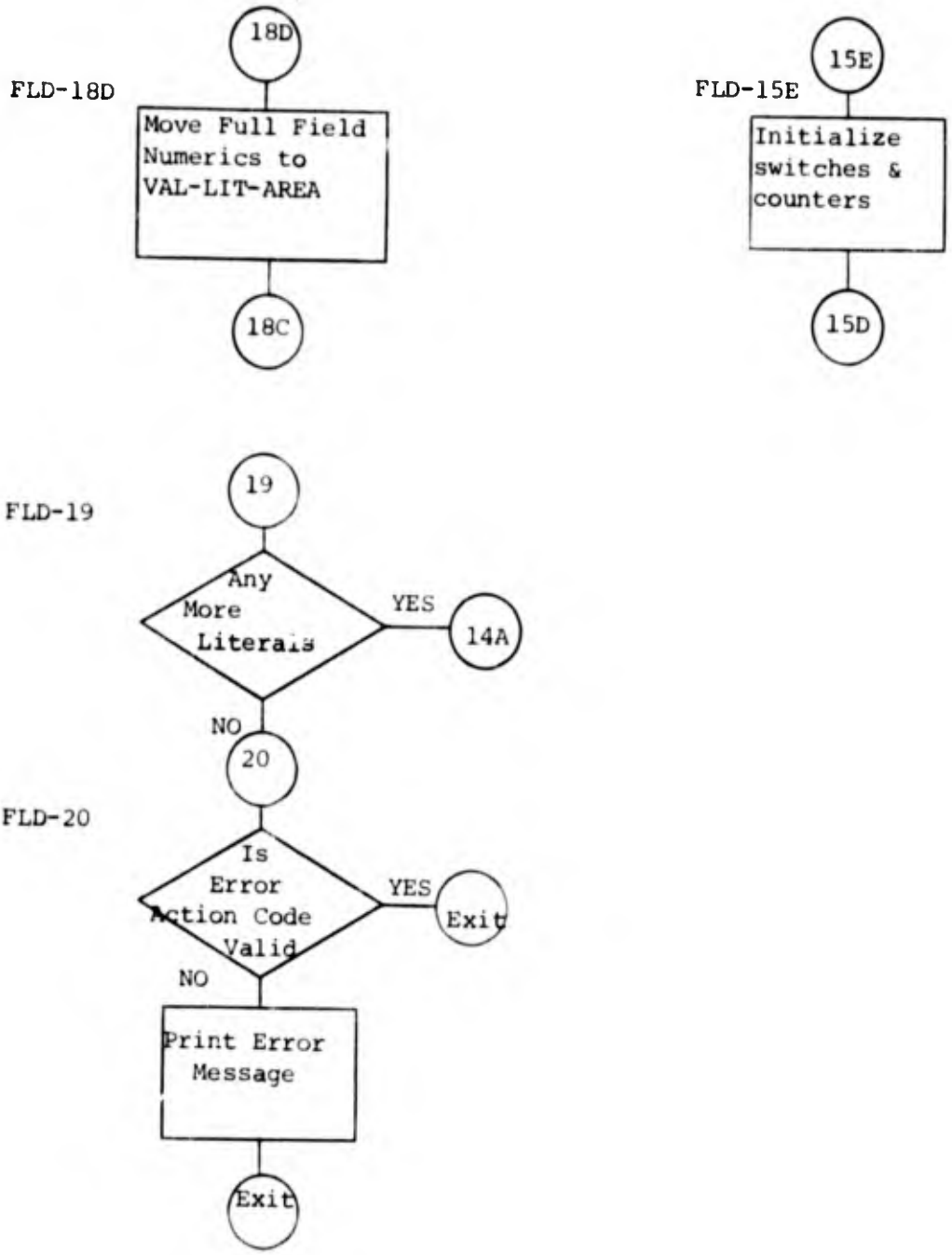
FLD-20.

Processes the error action code and if it is good it stores a numeric code in VAL-ERR in the VAL-LIST.

(d) Limitations. Although there is no limit to the number of value checks used against a field by convention, only 50 words may be specified on a field card. VAL-LIST can have no more than 600 entries and the VAL-LIT-AREA cannot exceed 5000 characters.


Start
FLD-14A
End of Card
YES
14B
NO
Partial Addressing Specified
YES
17
NO
Process Full Field Value
FLD-14B
Print: PARAMETERS NOT COMPLETE
Move 2 to Return-SW
Exit
15A
FLD-15A
Any more Values
YES
15
NO
20
15
FLD-15
Is it a literal
YES
16
NO
Is Size Valid
NO
15C
YES
Set Type to 0
Initial Pass
YES
15E
NO
15D
15D
FLD-15D
Set CNT4≠0
Initialization has been accomplished
16B
FLD-16B
If Full Field
YES
If Numeric
YES
Set TYPE-SW to 3
NO
NO
If Alpha
YES
18A
NO
18


16
FLD-16
Is Literal Valid
YES
Set TYPE-SW to 1
NO
Print: INVALID LITERAL
15A
Initial Pass
YES
15E
NO
15D
17
FLD-17
Validate Partial Field Size Parameter
Valid
YES
Initialize switches & counters
NO
Print: ERROR MESSAGE
18
Exit
18
FLD-18
Initializes counters & pointers for full field and Literals
If Field Parameter
YES
18D
NO
Move Size to Length Parameter
18E
FLD-18E
Move Alpha field to VAL-LIT-AREA
FLD-18D
Is Length > 20
YES
Move Remainder of Literal
18C
FLD-18C
Is Field a Literal
YES
19
NO
15A


FLD-18D
18D
Move Full Field Numerics to VAL-LIT-AREA
18C
FLD-15E
15E
Initialize switches & counters
15D
FLD-19
19
Any More Literals
YES
14A
NO
FLD-20
20
Is Error Action Code Valid
YES
Exit
NO
Print Error Message
Exit

(13) OMLPPRO

(1) Function. Edits the PROFILE validity check phrase on the field card. If errors are detected appropriate error messages are printed. Information from this phrase is validated, interpreted and stored in the DST.

(2) Calling Sequence.

```
ENTRY   'OMPRO'   USING FM-DD  OM-DD
CALL    'FMPRT'   USING PRT-LINE  CC
CALL    'MOVALF'  USING GROUP-HOLD  CNT10 LENGTHA
                        VAL-LIT-AREA CNT11 LENGTHB
                        EXIT-SW
```

(3) Program Description.

FLD-12A.

FLD-12A increments FLD-VAL-CT in the FIELD-LIST and sets the HOP for the profile literal in the data string VAL-LIT-AREA in VAL-LIT-HOP in the VAL-LIST.

FLD-12C.

Converts each character of the profile to a two digit numeric code.

FLD-13D.

Stores this converted string in the VAL-LIT-AREA, enters VAL-LIT-LEN in the VAL-LIST, and enters a 2 for PROFILE in VAL-TYPE in the VAL-LIST. FLD-14 stores the error action code in VAL-ERR.

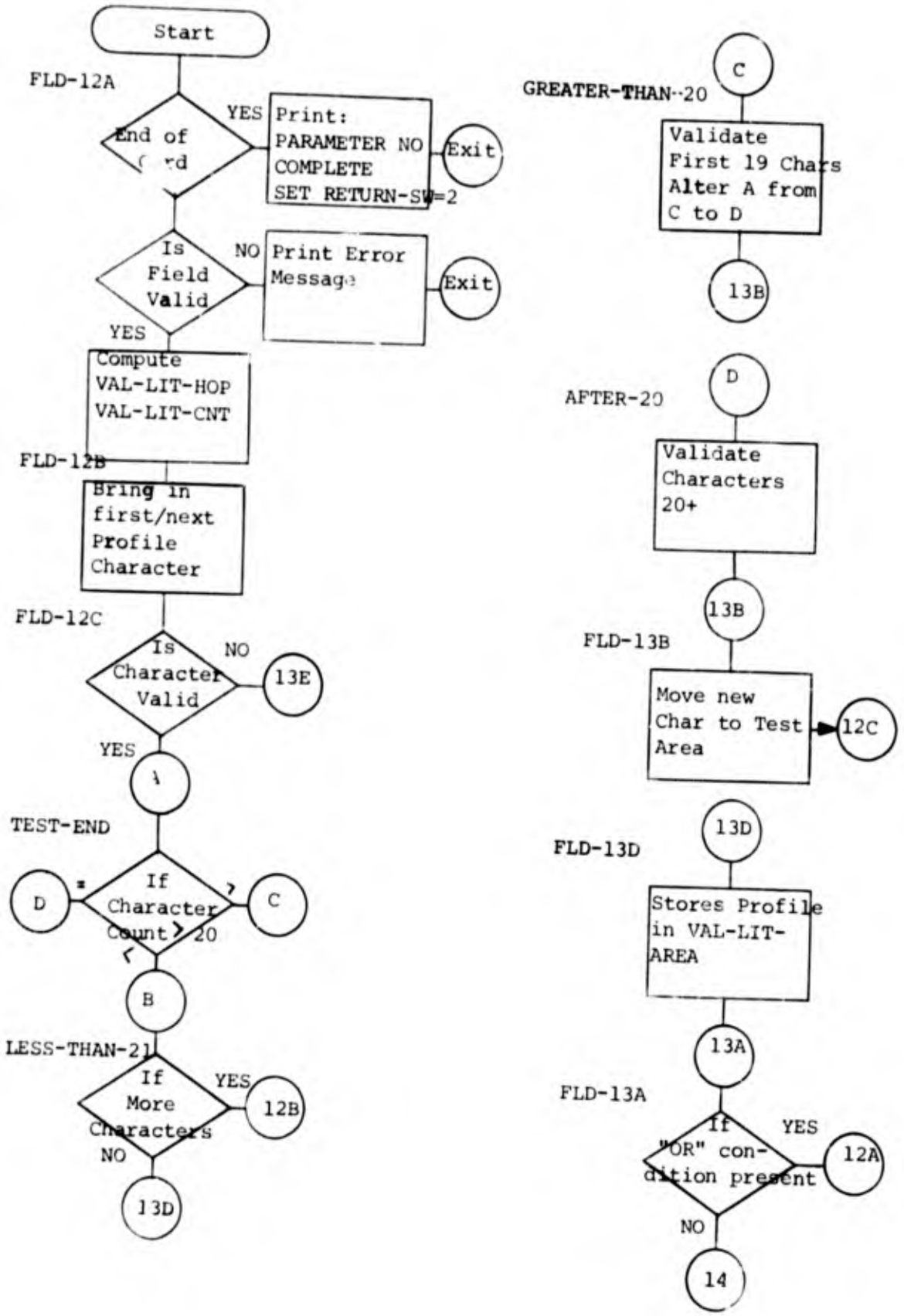
FLD-13A.

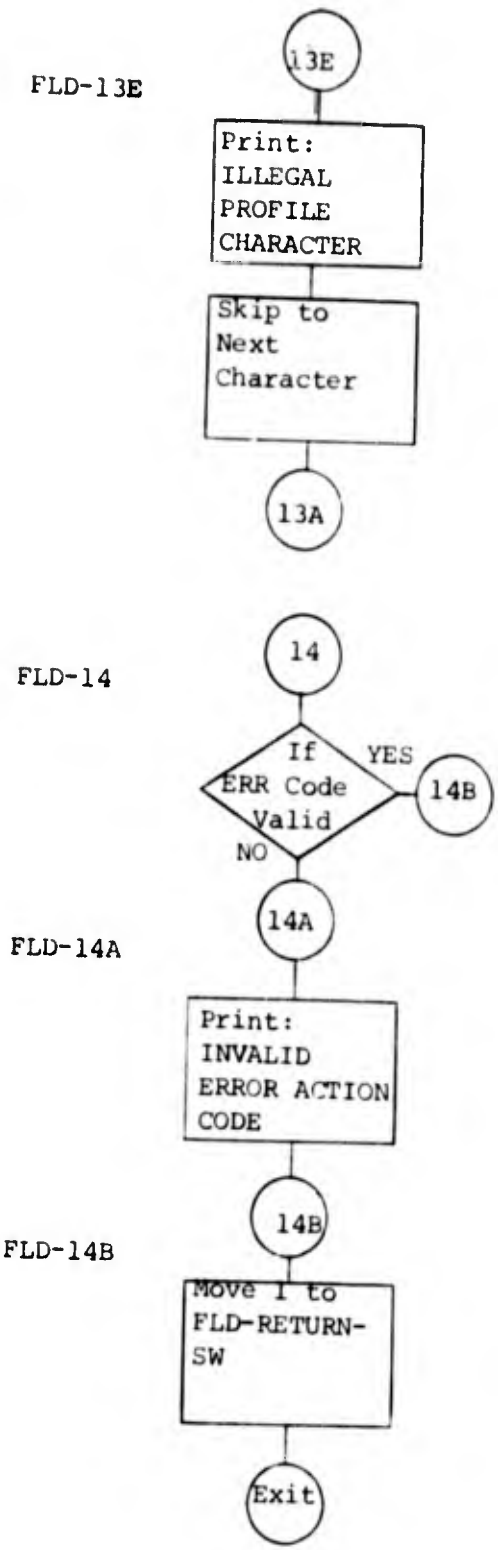
Will branch to FLD-12A when multiple profiles are specified.

FLD-14.

Edits and stores the error action code in VAL-ERR in the VAL-LIST.

(4) Limitations. A maximum of 52 profile characters may describe one field. There is no limit of profile checks on a field, but only 50 words may be specified on the field card. The VAL-LIT-AREA cannot exceed 5000 characters. There is a maximum of 600 entries in the VAL-LIST. This includes all validity checks and operations.


Start
FLD-12A
End of
YES
Print:
PARAMETER NO
COMPLETE
SET RETURN-SW=2
Exit
Is
Field
Valid
NO
Print Error
Message
Exit
YES
Compute
VAL-LIT-HOP
VAL-LIT-CNT
FLD-12B
Bring in
first/next
Profile
Character
FLD-12C
Is
Character
Valid
NO
13E
YES
A
TEST-END
D
=
If
Character
Count
>
20
<
C
B
LESS-THAN-21
If
More
Characters
YES
12B
NO
13D
GREATER-THAN-20
C
Validate
First 19 Chars
Alter A from
C to D
13B
AFTER-20
D
Validate
Characters
20+
13B
FLD-13B
Move new
Char to Test
Area
12C
13D
FLD-13D
Stores Profile
in VAL-LIT-
AREA
13A
FLD-13A
If
"OR" con-
dition present
YES
12A
NO
14


FLD-13E
13E
Print:
ILLEGAL
PROFILE
CHARACTER
Skip to
Next
Character
13A
FLD-14
14
If
ERR Code
Valid
YES
14B
NO
FLD-14A
14A
Print:
INVALID
ERROR ACTION
CODE
FLD-14B
14B
Move 1 to
FLD-RETURN-
SW
Exit

(14) OMLPSBR

(a) Function. OMLPSBR edits the SUBCON and SUBCHK phrases insuring that the user format is correct. This format is documented in the MIDMS User's Reference Manual, Chapter 3. If errors are detected in this format, appropriate error messages are given. Information from these phrases are validated, interpreted, and stored in the DST.

(b) Calling Sequence.

```
ENTRY   'OMSBR'   USING FM-DD  OM-DD
CALL    'FMPRT'   USING PRT-LINE  CC
CALL    'MOVALF'  USING FM-LIT-DATA   CNT10 LENGTHA
                        VAL-LIT-AREA  CNT11 LENGTHB
```

(c) Program Description.

OMLPSBR.

There are multiple formats of the SUBCON/SUBCHK phrase. The key to these formats is a hyphen following the second word of the phrase. OMLPSBR checks for this hyphen and if it is found appropriate coding is executed. If a hyphen is present, it may or may not be preceeded by a literal, but it must be followed by a subroutine name. If there is no hyphen, the word must be a subroutine name.

FLD-21.

Validates the subroutine name and moves it to the VAL-LIT-AREA.

FLD-MOVE2.

Places the subroutine name in the VAL-LIT-AREA.

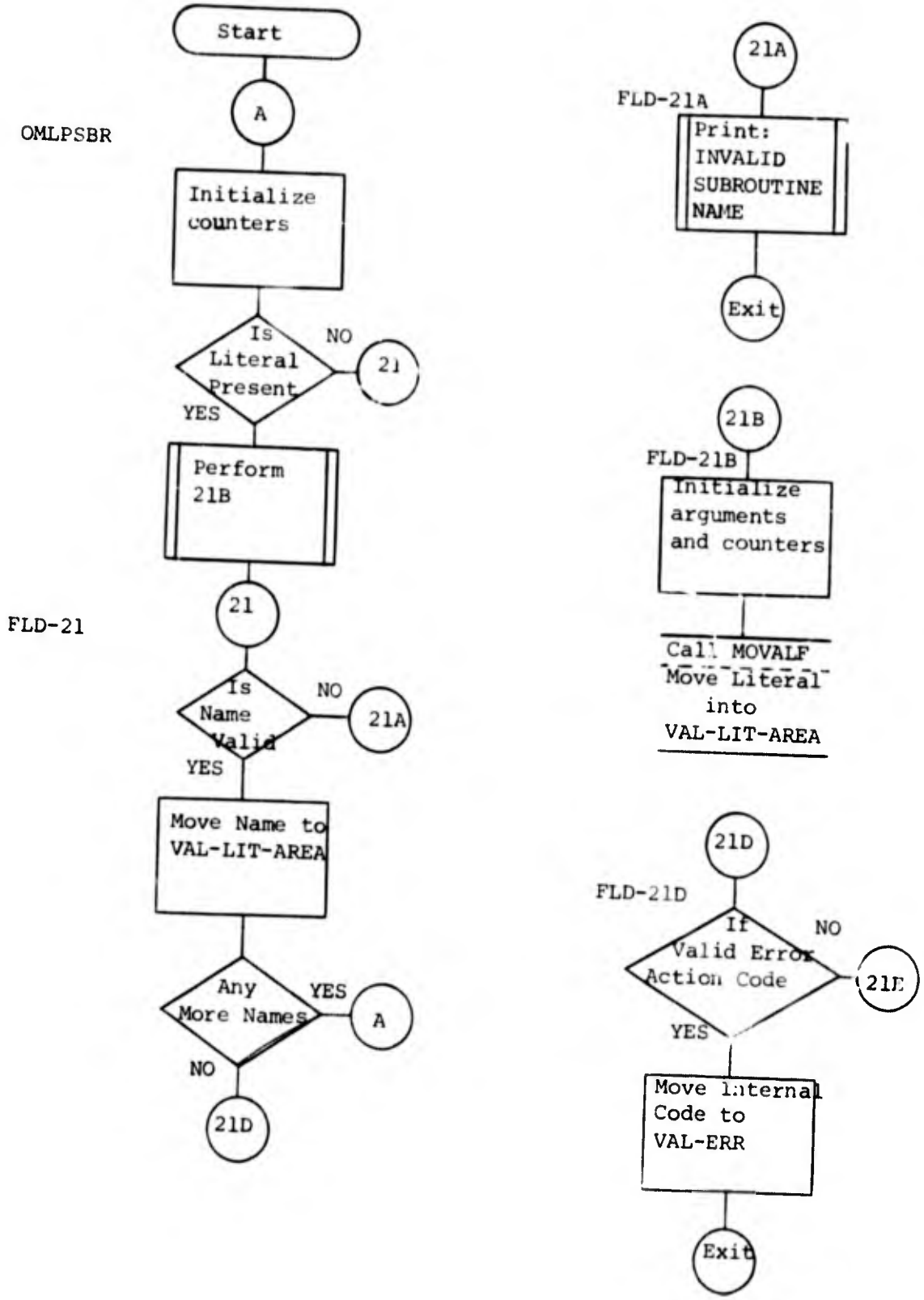
FLD-MOVE-LIT.

Moves the literal preceding the hyphen to the VAL-LIT-AREA.

FLD-21D.

Stores the numeric error action code in VAL-ERR in the VAL-LIST.

(d) Limitations. Multiple subroutine names may be specified; however, the interlinkage must be set up by the user. Only 50 words may be specified on a field card. The VAL-LIT-AREA cannot exceed 5000 characters. There is a maximum of 600 entries in the VAL-LIST which includes validity checks and operators. Each subroutine name is considered a separate validity check.


OMLPSBR
Start
A
Initialize counters
Is Literal Present
NO
21
YES
Perform 21B
FLD-21
21
Is Name Valid
NO
21A
YES
Move Name to VAL-LIT-AREA
Any More Names
YES
A
NO
21D
FLD-21A
21A
Print: INVALID SUBROUTINE NAME
Exit
FLD-21B
21B
Initialize arguments and counters
Call MOVALF
Move Literal into VAL-LIT-AREA
FLD-21D
21D
If Valid Error Action Code
NO
21E
YES
Move Internal Code to VAL-ERR
Exit

(15) OMLPPSS

(a) Function. Processes the PSSQ entry if the field is defined as periodic. If errors are detected in format, appropriate error messages are printed. Information from this phrase is validated and stored in the DST.

(b) Calling Sequence.

```
ENTRY 'OMPSS'    USING FM-DD  OM-DD
CALL  'FMPRT'    USING PRT-LINE  CC
CALL  'MOVCMP'   USING FM-DATA-WORD  CNT10
                       LENGTHA
                       FLD-PSS-LEN   CNT11
                       LENGTHB   CONTINUE-SAVE
```

(c) Program Description.

OMLPPSS.

Scans the first word to determine if it is alpha or numeric. Alpha field are handled by FLD-6B and numeric fields by FLD-6D.

FLD-6B.

Edits the name and stores it in FLD-PSS in the FIELD-LIST. The subscript of the name in the FIELD-LIST is stored in PSS-FLD-POINT in the PSS-LIST-TEMP.

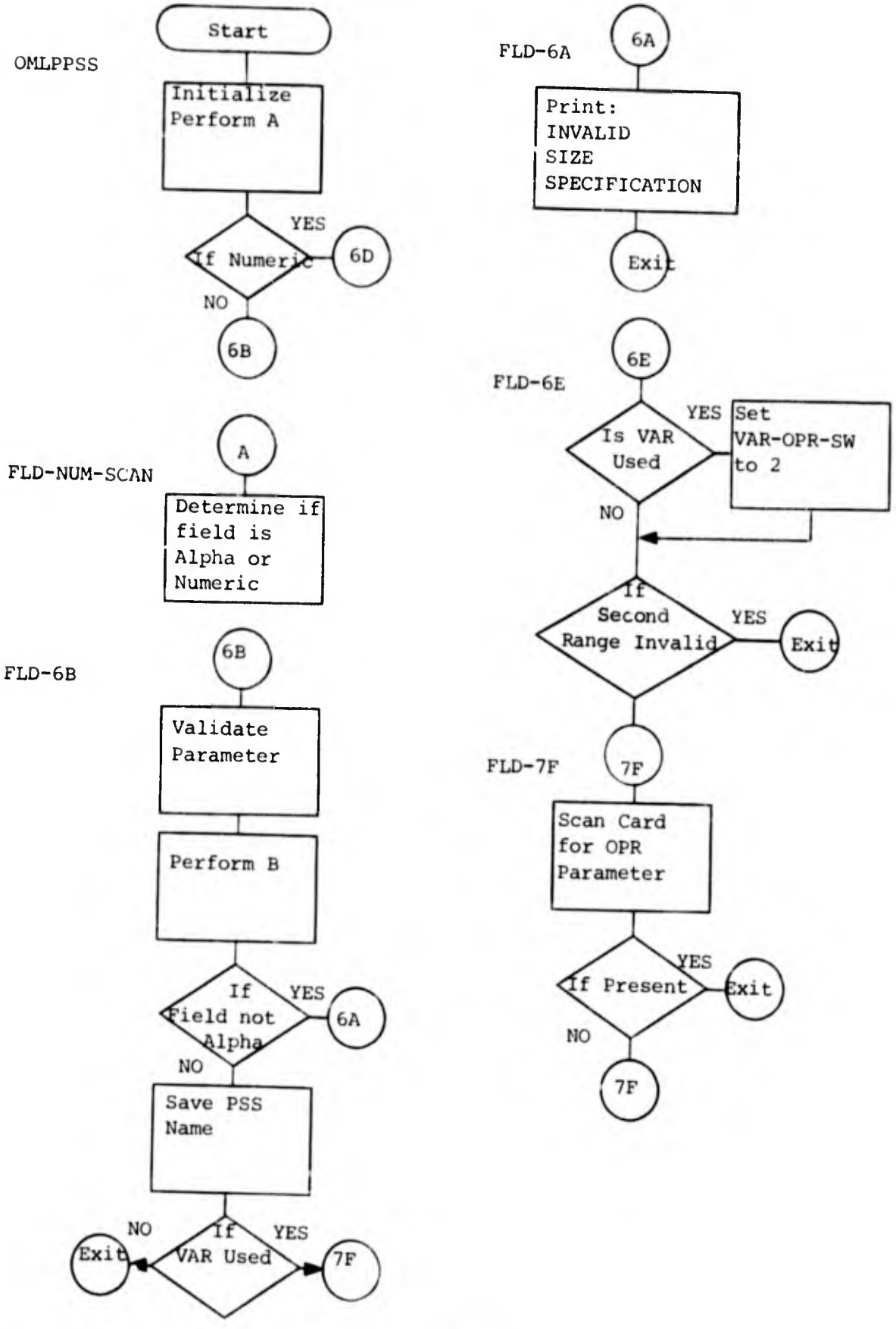
FLD-6D.

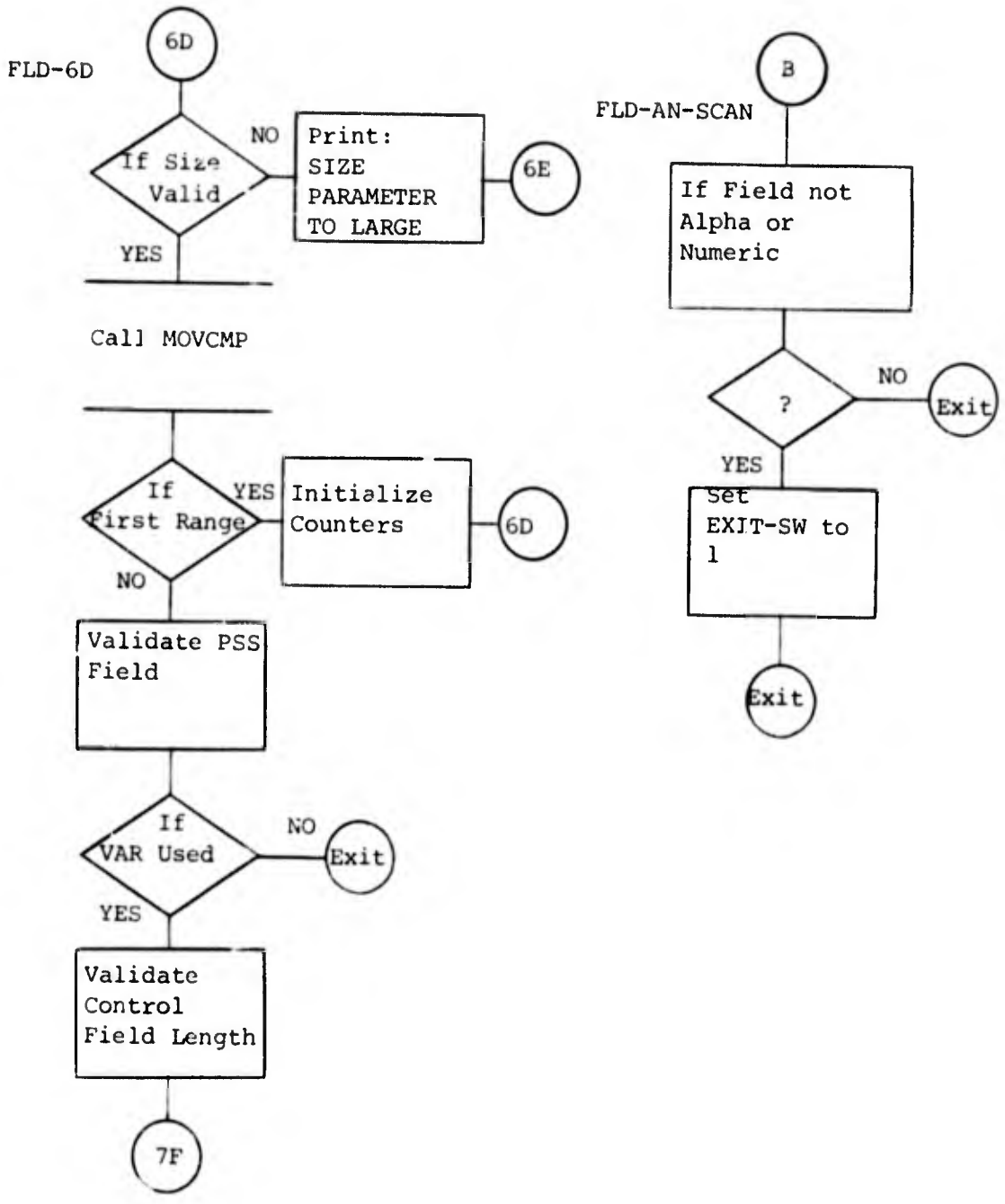
Stores the HOP-LOP pair as a HOP and LEN in FLD-PSS-HOP and FLD-PSS-LEN in the FIELD-LIST.

FLD-7F.

Checks for the OPR phrase which should be present immediately following the VAR option if it is used.

(d) Limitations. Maximum entries in the PSS-LIST-TEMP is 200.


OMLPPSS
Start
Initialize
Perform A
If Numeric
YES
6D
NO
6B
FLD-NUM-SCAN
A
Determine if field is Alpha or Numeric
FLD-6B
6B
Validate Parameter
Perform B
If Field not Alpha
YES
6A
NO
Save PSS Name
If VAR Used
NO
Exit
YES
7F
FLD-6A
6A
Print: INVALID SIZE SPECIFICATION
Exit
FLD-6E
6E
Is VAR Used
YES
Set VAR-OPR-SW to 2
NO
If Second Range Invalid
YES
Exit
FLD-7F
7F
Scan Card for OPR Parameter
If Present
YES
Exit
NO
7F


FLD-6D
6D
If Size Valid
NO
Print: SIZE PARAMETER TO LARGE
6E
YES
Call MOVCMP
If First Range
YES
Initialize Counters
6D
NO
Validate PSS Field
If VAR Used
NO
Exit
YES
Validate Control Field Length
7F
FLD-AN-SCAN
B
If Field not Alpha or Numeric
?
NO
Exit
YES
Set EXIT-SW to 1
Exit

(16) OMLPWRP

(a) Function. OMLPWRP completes final processing of the DST checking basic information that could not be checked logically before the entire DSD was read.

(b) Calling Sequence.

```
ENTRY 'OMWRP'  USING FM-DD  LP-DD  OM-DD
CALL  'FMPRT'  USING PRT-LINE  CC
```

(c) Program Description.

OMLPWRP.

Paragraph OMLPWRP initiates the following processing. If a FILE level SEQ/ORD field is specified, the fields in the DSD are searched for a match (FIND-SEQ-ORD-FLD). If a GROUP level SEQ/ORD field is specified, the fields in the DSD are searched for a match (FIND-SEQ-ORD-FLD2). The GRP-ID fields must be specified in at least one record of the DSD (GRP-ID-CK not equal to 0). All fields specified in the FILE card should be present in the FFT and their size should equal the FFT control field size.

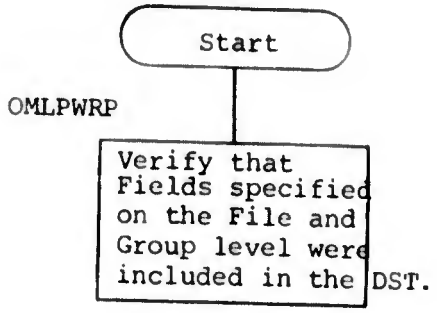
FIL-ID-VALIDATE.

Stores the relative HOP of the FIL-ID fields in FIL-ID-HOP in the FIL-ID-LIST.

SET-GRP-ID-HOP.

Stores the relative HOP of the Group ID field in GRP-ID-HOP.

(d) Limitations. None.



FIND-SEQ-ORD-FLD
and
FIND-SEQ-ORD-FLD2

Examine the Record to see if the SEQ/ORD field name specified was present.

FIL-ID-VALIDATE

Check control field names and compute field length.

SET-GRP-ID-HOP

Compute GRP ID length for comparison with GRP-ID-CNT

(17) OMLPWRT

(a) Function. To write the DSD tables to the MIDMS Table Library.

(b) Calling Sequence.

```
ENTRY   'OMWRT'     USING FM-DD  OM-DD  LP-DD
CALL    'FMPRT'     USING PRT-LINE  CC
CALL    'DATESUB'   USING ITEM-DATE
CALL    'LB'        USING DSD01
                          GRP-TYPES-LIST
(Separate Calls)          REC-IDS-LIST
                          FIL-OPR-NAME-LIST
                          REC-FIL-OPR-LIST
                          REC-TYPE-ITEM
                          FIELD-ITEM
                          VAL-ITEM
                          VAL-LIT-CHAR
```

(c) Program Description.

OMLPWRT.

Consists of separate calls of LB writing out tables whose sizes are relatively small. The REC-TYPE-LIST, FIELD-LIST, VAL-LIST and VAL-LIT-AREA are written onto the MIDMS table library by repetitions of paragraphs WRITE-MORE-RECS, WRITE-MORE-FIELDS, WRITE-MORE-VALS and WRITE-MORE-LITS espectively.

(d) Liritations. None.

(18) OMLPRWP

(a) Function. The function of OMLPRWP is to complete processing of a record after all field cards for that record have been processed.

(b) Calling Sequence.

```
ENTRY 'OMRWP' USING FM-DD  OM-DD
CALL  'FMPRT' USING PRT-LINE  CC
```

(c) Program Description.

OMLPRWP.

Performs conditioned searches of the field names that occur within the record for the following information. If a FILE SEQ/ORD field was specified and the field is in the record, the subscript of the field in FIELD-LIST is stored in REC-FILE-SEQ-ORD in the REC-TYPE-LIST. If a GROUP SEQ/ORD field was specified and it occurs in the record, the subscript of the field is stored in REC-GRP-SEQ-ORD in the REC-TYPE-LIST. If any of the specified GRP-ID fields were present in the record (compared on the field level) then all must be present. The GRP-ID-LIST is checked to verify this. The subscript of the field name matching the GRP-ID is stored in REC-IDS-FLD in the REC-IDS-LIST, and the subscript of the GRP-ID in the GRP-ID-LIST is stored in REC-IDS-NO in the REC-IDS-LIST. The subscript of the first REC-IDS-LIST entry for the GRP-ID found in the record is stored in REC-GRP-ID in the REC-TYPE-LIST.

OPR-NAME-VALIDATION.

Checks OPR-NAME-LIST-TEMP to verify that those fields named on the record level are present on the field level for that record. If the OPR field name occurred on the file level, the subscript of the field name in the FIELD-LIST is stored in REC-FIL-OPR-FLD in the REC-FIL-OPR-LIST.

CK-TYPE5-OPRS.

Checks to see that the OPR field lengths are equal.

OPR-PHASE4.

Moves the HOP of the name in VAL-LIT-AREA (OPR-VAL-LIT-HOP in the OPR-NAME-LIST-TEMP) to the VAL-LIT-AREA.

FIND-PSS-SUBSCRIPT.

If periodic fields are described in the record and these fields use a field name to give the location of the PSSQ compare the fiel name with FLD-PSS subscripted by PSS-FLD-POINT in the FIELD-LIST. The name in FLD-PSS must occur in a field card for that record.

(d) Limitations. The only table used by the program is REC-FIL-OPR-LIST. It has a maximum of 40 entries.

3. ERROR MESSAGES.

a. ALREADY SPECIFIED

(1) OMLPFIL, OMLPGRP, OMLPREC

(2) Parameter was previously specified on the card. Remove the dupe and rerun.

b. CARD ILLEGAL WITH FILE CODE NONE

(1) OMLPREC

(2) The RECORD card should not be present when "NONE" is specified on the FILE card.

c. DOUBLY DEFINED LMCON

(1) OMLPINT

(2) Indicates that a non FFT defined field (LM CONSTANT) has been defined twice.

d. DUPLICATE RECORD TYPE

(1) OMLPREC

(2) Indicated record type name has been used previously. Record types must be unique.

e. DSD FIELD LARGER THAN FFT FIELD, SUBCON MISSING

(1) OMLPFLD

(2) User must specify a conversion program.

f. DST, DSD or LOG REQUIRED

(1) FMLPVAL

(2) The second word of the FMLP control card has to be DST, DSD or LOG.

g. DSD NAME MUST END IN D

(1) FMLPVAL

(2) The last character of the DSD name has to be the letter D.

h. DST SIZE GREATER THAN PROGRAMMED MAXIMUM

(1) OMLPWRT 360 only

(2) The total DST size exceeds 20000 characters.

i. ERROR CODE 0 ILLEGAL PRECEEDING FIRS OPR

(1) OMLPFLD

(2) 0 must be used after the first OPR when multiple OPR's are being used but it should not appear before the first one.

j. EXCEEDS PROGRAMMED MAXIMUM

(1) OMLPFIL, OMLPGRP, OMLPREC, OMLPINT

(2) A maximum value as set in OMDOV has been exceeded. Check FLD-LIST-MAX, GRP-TYPES-MAX, GRP-ID-MAX, REC-TYPE-MAX, FIL-ID-MAX.

k. FFS ID FIELDS EXCEED PROGRAMMED MAXIMUM

(1) OMLPINI

(2) There is a maximum of 100 record ID fields allowed. This figure has been exceeded.

l. FIELD LOCATION EXCEEDS RECORD DEFINITION

(1) OMLPINT

(2) Location specified extends beyond the area assigned to the record.

m. FIELD MUST BE A LITERAL

(1) OMLPSBR

(2) Value test is expecting a literal.

n. FILE CARD OUT OF PLACE

(1) OMLPFIL, OMLPGRP, OMLPFLD

(2) File card should be the first card in the DSD.

o. FILE ID SIZE MUST EQUAL FFT CONTROL SIZE

(1) OMLPWRP

(2) File ID must be maintained throughout all cards in the DSD, and must be equal to the FFT ID size.

P. FILE RECORD CODE LENGTH GREATER THAN 10

(1) OMLPFIL

(2) Record code length (difference between HOP and LOP) cannot exceed 10 characters.

q. FREQUENCY CODE OMITTED

(1) OMLPGRP

(2) Frequency code could not be found. Program looks for next valid group type. Check for hyphen.

r. GROUP ID FIELDS EXCEED PROGRAMMED MAXIMUM

(1) OMLPRWP

(2) There can be a maximum of 100 record ID fields. This figure has been exceeded.

s. GRP-ID FIELDS OMITTED

(1) OMLPWRP

(2) The GROUP ID fields specified in the GROUP card did not exist in any record in the DSD as field cards.

t. HOP GREATER THAN LOP, LOCATION REJECTED

(1) OMLPINT, OMLPPSS, OMLPFIL

(2) Check record code, PSSQ location field. LOP must always be greater.

u. ID OMITTED FROM PREVIOUS RECORD

(1) OMLPRWP

(2) A GROUP ID field was omitted from the previous record.

v. ILLEGAL FOR RECID

(1) OMLPOP3

(2) The OPR code specified cannot be used on a Record ID field.

w. ILLEGAL PROFILE CHARACTERS

(1) OMLPPRO

(2) Character(s) used is not a valid test character. See MIDMS User's Reference Manual for list of profile test characters.

x. ILLEGAL RUN SPECIAL...CODE IGNORED

(1) FMLPVAL

(2) Not one of the valid run special keywords. Should be PROGRAM, SUBROUTINE, DELETE, or DUMPDSD. Check your FMLP control card for spelling or possible omission of keyword.

y. ILLEGAL RUN TYPE

(1) FMLPVAL

(2) Acceptable run types are O for Ordinary Maintenance, S for Single File LM and F for File to File LM or File Retrieval.

z. ILLEGAL VAR CONTROL FIELD LENGTH

(1) OMLPRWP, OMLPWRP, OMLPPSS

(2) Field length must be eight characters long.

aa. ILLEGAL WITH OTHER UNCONDITIONAL OPR

(1) OMLPOPR

(2) Only one unconditional OPR can occur following the OPR keyword.

ab. ILLEGAL WITH UNCONDITIONAL RECID REP REG GER OPRS

(1) OMLPOPR

(2) The OPR codes specified are mutually exclusive.

ac. ILLEGAL WITH VAR OPTION

(1) OMLPOP4

(2) Only OPRS legal with VAR are: REP, GEC, REG, ATV, CHV.

ad. INVALID CHANGE LENGTH HOP

(1) OMLPOP4

(2) The high order position indicated was either too long or not numeric.

ae. INVALID CHARACTER BETWEEN TEST VALUE AND OPCODE

(1) OMLPOPR

(2) A hyphen should come between test value and the opcode it represents.

af. INVALID DSD or LOG NAME

(1) FMLPVAL

(2) Name should be five characters long and must begin with an alpha character.

ag. INVALID ERROR ACTION CODE

(1) OMLPFIL, OMLPGRP, OMLPRNG, OMLPVAL, OMLPPRO, OMLPSBR, OMLPOP4

(2) This one character code must be one of these specified in the MIDMS User's Reference Manual and must be an allowable action for the particular card and operation being performed.

ah. INVALID FREQUENCY CODE

(1) OMLPGRP

(2) Frequency code must be an 'A' or 'O' or any number 0 to 9. Check for invalid characters.

ai. INVALID GROUP TYPE CODE

(1) OMLPGRP

(2) Check the length of the code.

aj. INVALID INPUT RECORD HOP

(1) OMLPOP4

(2) The high order position indicated was either too long or not numeric.

ak. INVALID LITERAL

(1) OMLPVAL

(2) Length of literal must equal field size. Check that literal is not equal to zero, or too long.

al. INVALID LOCATION SPECIFICATION

(1) OMLPFIL, OMLPINT, OMLPPSS

(2) Location must be all numeric.

am. INVALID LOP

(1) OMLPOP2

(2) The LOP field is too large or contains a number greater than the source input record size.

an. INVALID MAJOR FILE NAME

(1) FMLPVAL

(2) Last character of name must be an A or T.

ao. INVALID MINOR FILE NAME

(1) FMLPVAL

(2) Minor file name must be five characters long, the first character must be alpha, and the last character must be either A or T.

ap. INVALID NAME

(1) OMLPFIL, OMLPINT, OMLPGRP, OMLPOP2

(2) A field name is invalid. Check for special characters or improper length.

aq. INVALID OPR FOR FIELD

(1) OMLPOP4

(2) The OPR code indicated cannot be used on the specified field.

ar. INVALID PARAMETER

(1) OMLPOPR, OMLPFIL, OMLPREC, OMLPFLD, OMLPGRP, OMLPOP2, OMLPOP3, OMLPOP4

(2) Parameters on card either missing or not applicable to card type.

as. INVALID PARTIAL FIELD

(1) OMLPVAL

(2) Partial field must begin with numeric entry indicating field position number. Check for zero alphabetic or special characters.

at. INVALID PSS HOP

(1) OMLPPSS

(2) HOP cannot be zero.

au. INVALID RECORD CODE

(1) OMLPREC

(2) Address length cannot exceed 9999 or 4 characters.

av. INVALID RECORD CODE ADDRESS

(1) OMLPFIL

(2) Address length should not exceed four characters.

aw. INVALID SIZE PARAMETER

(1) OMLPFIL

(2) Size parameter is probably too large. Must be four characters or less.

ax. INVALID SIZE SPECIFICATION

(1) OMLPFIL, OMLPREC, OMLPPSS

(2) Make sure size is numeric and four characters or less.

ay. INVALID SUBROUTINE NAME

(1) OMLPSBR

(2) Subroutine names must be five characters, alphabetic, and must end in s.

az. INVALID TEERMINATOR BETWEEN HOP AND LOP

(1) OMLPFIL

(2) Record code location must be separated by a hyphen.

ba. LEGAL ONLY ON FIELD CARD

(1) OMLPOP3

(2) The whole record opcodes and VAR option can only be used on a FIELD card.

bb. LEGAL ONLY WITH FFT DEFINED FIELD

(1) OMLPOP3

(2) The field to be operated on has not been found in the FFT.

bc. LITERAL INVALID WITH NUMERIC FIELD

(1) OMLPVAL

(2) With a numeric field the only literal acceptilbe by the value test must be all blanks. In this case a non-blank was found.

bd. LITERAL MISSING

(1) OMLPOP2

(2) A literal is missing.

be. LOCATION OUTSIDE RECORD

(1) OMLPPSS

(2) If the length is greater than the record size, the location will be outside the record.

bf. LOG NAME MUST END IN P

(1) FMLPVAL 360 only

(2) Change the last character of the logic package name to a P.

bg. LOW-RANGE GREATER THAN HI-RANGE

(1) OMLPRNG

(2) HI-RANGE must be higher than LOW-RANGE.

bh. MAJOR FFT NOT ON LIBRARY

(1) FMLPVAL 360 only

(2) The file named to be updated cannot be found. Make sure the name is correct and the file has been placed on the library.

bi. MINOR FFT NOT ON LIBRARY

(1) FMLPVAL

(2) During a file-to-file run the data source file or the old file named cannot be found on the library. See if the name is correct or if the file is present.

bj. MULTIPLE OPRS LEGAL ONLY WITH ERROR CODE 0

(1) OMLPOPR

(2) Multiple OPRS when used are connected with error code 0.

bk. MULTIPLE SUBCONS LEGAL ONLY WITH ERROR CODE 0

(1) OMLPFLD

(2) Multiple SUBCONS when used are connected with error code 0.

bd. LITERAL MISSING

(1) OMLPOP2

(2) A literal is missing.

be. LOCATION OUTSIDE RECORD

(1) OMLPPSS

(2) If the length is greater than the record size, the location will be outside the record.

bf. LOG NAME MUST END IN P

(1) FMLPVAL 360 only

(2) Change the last character of the logic package name to a P.

bg. LOW-RANGE GREATER THAN HI-RANGE

(1) OMLPRNG

(2) HI-RANGE must be higher than LOW-RANGE.

bh. MAJOR FFT NOT ON LIBRARY

(1) FMLPVAL 360 only

(2) The file named to be updated cannot be found. Make sure the name is correct and the file has been placed on the library.

bi. MINOR FFT NOT ON LIBRARY

(1) FMLPVAL

(2) During a file-to-file run the data source file or the old file named cannot be found on the library. See if the name is correct or if the file is present.

bj. MULTIPLE OPRS LEGAL ONLY WITH ERROR CODE 0

(1) OMLPOPR

(2) Multiple OPRS when used are connected with error code 0.

bk. MULTIPLE SUBCONS LEGAL ONLY WITH ERROR CODE 0

(1) OMLPFLD

(2) Multiple SUBCONS when used are connected with error code 0.

bl. MUST BE NUMERIC

(1) OMLPVAL, OMLPRNG

(2) Field in question must be numeric for a valid test.

bm. OPR MUST FOLLOW HOP AND LOP

(1) OMLPPSS

(2) When the PSSQ location is described on a field card that field card must contain an OPR immediately after the high order position and low order position specification.

bn. OPR OPTION MUST BE LAST WHEN IT IS USED

(1) OMLPFIL

(2) The operational parameter is optional but when used must be the last entry on the card.

bo. OPR TESTFIELD OMITTED FROM PREVIOUS RECORD

(1) OMLPRWP

(2) No test fields specified for the operator on the previous record.

bp. OPR TEST FIELDS OMITTED FROM RECORD

(1) OMLPRWP

(2) No test fields specified for operator.

bq. PARAMETER ALREADY SPECIFIED

(1) OMLPREC

(2) Parameter was previously specified on the card.

br. PARAMETER MAY NOT FOLLOW OPR

(1) OMLPOPR

(2) OPR must be last parameter on field card.

bs. PARAMETER MAY NOT FOLLOW SUBCON

(1) OMLPFLD

(2) SUBCON cannot be followed by a validity check; can be followed only by an OPR.

bt. PARAMETER MISSING

(1) OMLPGRP

(2) Check for missing parameters.

bu. PARAMETERS NOT COMPLETED

(1) OMLPGRP, OMLPRNG, OMLPPSS, OMLPVAL, OMLPPRO, OMLPINT, OMLPOPR, OMLPFIL, OMLPSBR, OMLPREC

(2) Check for incomplete or missing parameters.

bv. PARTIAL AND FULL FIELD VALUE NOT PERMITTED TOGETHER

(1) OMLPVAL

(2) Either full field or partial addressed values can be tested, but not both. Check value validity tests.

bw. PARTIAL LOP GREATER THAN 4 DIGITS

(1) OMLPVAL

(2) Largest number is 9999. Check partial LOP to insure that it does not exceed this figure.

bx. PARTIAL SPECIFICATION EXCEEDS FIELD SIZE

(1) OMLPVAL

(2) Check that partial field is not larger than FFT field length.

by. PROGRAM MODE REQUIRES RUN TYPE F

(1) FMLPVAL

(2) Program mode can only be specified for file to file processing. If this is what is desired, change run type to F.

bz. PSS SIZE EXCEEDS 3 CHARACTERS

(1) OMLPRWP, OMLPPSS

(2) PSSQ field must be one to three characters long.

ca. PSSQ COUNT EXCEEDS PROGRAMMED MAXIMUM

(1) OMLPPSS

(2) PSSQ count as set in OMDDV is exceeded by program.

cb. RUN TYPE MUST BE O

(1) FMLPVAL

(2) Change run type to O if OM run is desired.

cc. RUN TYPE MUST BE S OR F

(1) FMLPVAL

(2) Change run type to either S for Single File LM or F for File to File.

cd. SEQ/ORD FIELD OMITTED FROM RECORD

(1) OMLPWRP

(2) The sequence or order test field definition was omitted from the DSD.

ce. SEQ/ORD TESTFIELD OMITTED

(1) OMLPGRP

(2) The test field has been omitted from the Sequence or Order test phrase.

cf. SEQUENCE/ORDER TEST FIELD OMITTED

(1) OMLPFIL

(2) The sequence or order test field name has been omitted from the FILE card.

cg. SEQUENCE AND ORDER NOT PERMITTED TOGETHER

(1) OMLPGRP, OMLPFIL

(2) Both parameters have been specified--must eliminate one.

ch. SIZE GREATER THAN FIELD SIZE

(1) OMLPPRO

(2) Profile specifications larger than field size.

ci. SIZE MUST EQUAL FIELD SIZE

(1) OMLPVAL

(2) The test field must be equal in size to the FFT defined field.

cj. SIZE MUST EQUAL FIELD SIZE-PROCESSING ABORTED

(1) OMLPRNG

(2) The test field size exceeds the FFT defined field size.

ck. SIZE PARAMETER TOO LARGE

(1) OMLPPSS, OMLPINT

(2) Size cannot exceed four characters.

cl. SKIPPING FOR FILE CARD

(1) OMLPFLD, OMLPREC, OMLPGRP

(2) FILE card must be first card in DSD; check sequence of your deck.

cm. SPLIT FIELDS ILLEGAL

(1) OMLPINT

(2) When defining a field the user must use a contiguous area. There cannot be spaces within a field or intervening fields within a group of fields.

cn. TYPE C OR D TEST FIELDS EXCEED PROGRAMMED MAXIMUM

(1) OMLPOP2

(2) The number of test fields is greater than the predefined maximum.

co. TYPE 5 FILE OPR FLD-LENGTHS MUST BE EQUAL

(1) OMLPRWP

(2) Test fields used in Type D OPR format are compared and therefore, must have equal lengths.

cp. UNDEFINED NAME

(1) OMLPWRP, OMLPRWP

(2) No location specified for name.

cq. VALIDATION CHECK EXCEEDS PROGRAMMED MAXIMUM

(1) OMLPPRO, OMLPVAL, OMLPOPR, OMLPSBR, OMLPRNG

(2) Validation check maximum as set in OMDDV is exceeded by program.

cr. VARIABLE SET REQUIRED WITH VAR

(1) OMLPINT

(2) The VAR option is legal only with variable sets. Check the type field to see how it has been defined in the FFT.

cs. VARIABLE SET UPDATE ILLEGAL

(1) OMLPOP4

(2) The Type E OPR format can be used only on a variable field.

4. WARNING MESSAGES. The following messages are given as indicators of conditions that could possibly result in problems if the user were left unaware of them. These messages do not interfere with the run in any way and are for your information only.

a. CARD IGNORED IF-FILE RECORD CODE NONE-SPECIFIED

(1) OMLPGRP

(2) A "NONE" specified on the file card indicates there should not be a GROUP card. The group card will be passed over.

b. DSD FIELD LARGER THAN FFT FIELD, SUBCON PRESENT

(1) OMLPFLD

(2) A DSD DEFINED FIELD is larger than the corresponding FFT defined field. A conversion subroutine has been specified.

c. DSD SIZE SMALLER THAN FFT SIZE

(1) OMLPINT

(2) Sizes differ but no conversion subroutine is required.

d. NO OPR FOR FIELD-NOP ASSUMED

(1) OMLPFLD

(2) No operation specified on any level. Opcode NOP is assumed.

e. PSSQ MISSING, BLANK ASSUMED

(1) OMLPFLD

(2) In a periodic field a PSSQ must be used to indicate the subset to be updated. Not found in this instance.

f. RECORD CARD MISSING, FILE RECORD CODE USED

(1) OMLPINT

(2) Record card should be present unless 'NONE' has been specified on file card.

g. RECORD HAS NO FIELDS-WILL BE IGNORED BY IP

(1) OMLPREC

(2) IP will not process a record when it is input unless FIELD cards are specified.

h. SIZE LESS THAN FIELD SIZE

(1) OMLPPRO

(2) The size of the profile being used is smaller than the field it is being used against. Some part of the field is not being validated.

i. SIZE MISSING, 80 ASSUMED

(1) OMLPREC, OMLPFIL

(2) If size parameter not used, IP will assume 80.

j. SIZE MISSING, FILE SIZE ASSUMED

(1) OMLPREC

(2) Size indicated on FILE card is used if no further size specifications are listed.

CHAPTER 3

Section III

File Maintenance (Ordinary) Input Processor (FMIP)

1. OVERVIEW.

a. File Maintenance (Ordinary) Input Processor (FMIP) validates source data in accordance with previously produced Data Source Tables (DST's) and generates a transaction file of validated data. Input Processor consists of two load modules: FMIP and FMIPSRT. Both are dynamically loaded and executed by the File Maintenance Supervisor (the FM load module). In addition, the FMIO load module is used for reading cards (control cards and data cards) as well as printing FMIP messages. Figure 3-3-1 shows the detailed structure of the FMIP load module and its relationship to FM, FMIO, and FMIPSRT. The primary functions of the major FMIP routines are as follows:

b. FMIPX is the Input Processor Supervisor and calls most of the other subroutines. In addition, it initializes pseudo subset sequence numbers and reads tape source files when requested by FMIPREC and FMIPUN.

c. FMIPCD analyzes the parameters on FMIP control cards, sets appropriate switches and reads File Format and Data Source Tables. A table of fields to be confirmed by Maintenance Proper is constructed by FMIPCD.

d. FMIPREC performs the high level processing of source data records including validation of DSD-specified record codes, file and group ID's, record frequencies, etc.

e. FMIPVAL performs the detailed field validation as specified in the DSD, converts the data by a user subroutine (when required), determines the type of maintenance operation, and builds entries in a table describing the transactions to update the MIDMS file.

f. FMIPTRN sorts an input group of transaction table entries built by FMIPVAL and writes out transaction records.

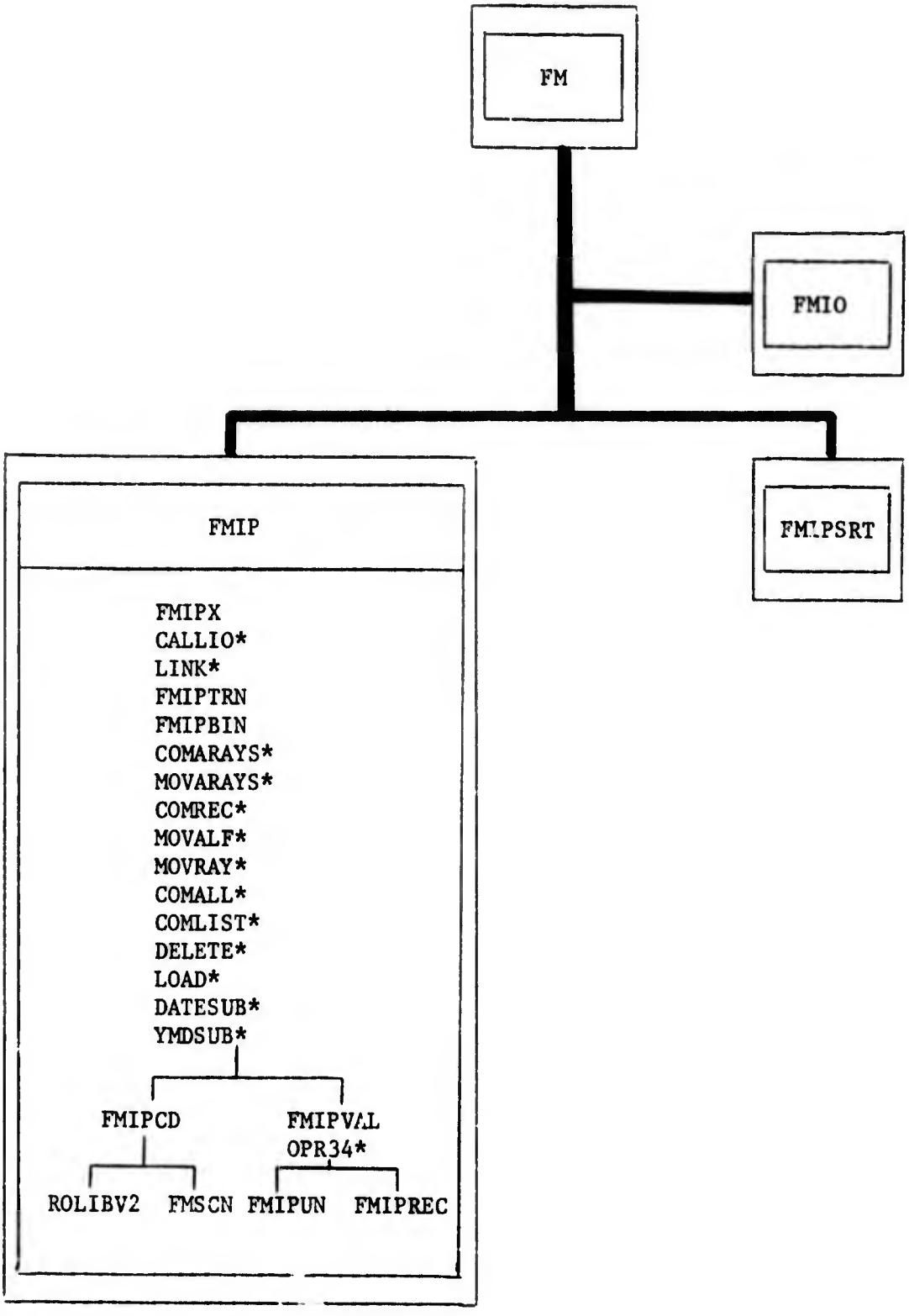
g. FMIPUN handles the preliminary processing of Unit Update data replacing FMIPREC. It validates field names, determines the type of maintenance operations that are valid, and builds a small series of data source tables so that FMIPVAL can check

the operation and build an entry in the transaction table. Record ID's on the Unit Update card are treated as group ID's for transaction record purposes.

h. FMIPBIN is a binary search program used to find a field in FFT Logical Record 11 after being called by FMIPCD or FMIPUN.

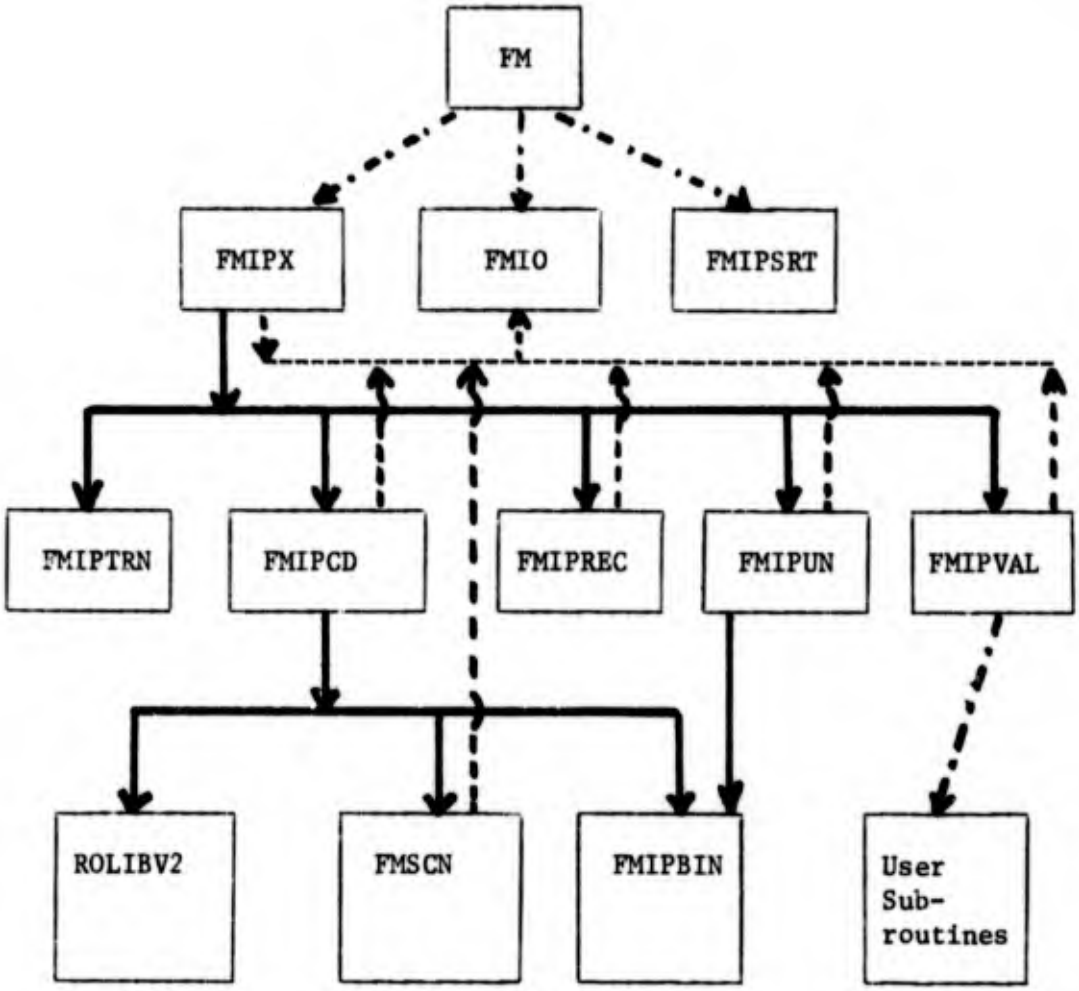
i. FMIPSRT sorts the transaction file produced by FMIPTRN so that the transactions can be merged with the MIDMS data file during Maintenance Proper. FMIPSRT is not executed when FMIPTRN determines that the transaction file is already in the proper sort.

j. The relationships between the FMIP subprograms are illustrated by figure 3-3-2.



*ALC Subroutines

Figure 3-3-1. FM Input Processor (FMIP) Structure for IBM S/360



-.-.-.-.- Link
Call
--------- I/O Call (thourgh CALLIO

Figure 3-3-2. FM Input Processor (FMIP) Subprogram Relationship for IBM S/360

2. FILE MAINTENANCE (ORDINARY) INPUT PROCESSOR (FMIP) SUBPROGRAMS.

a. FMIPX

(1) Function. To control the execution of the Input Processor subprograms, initialize pseudo subset sequence numbers, and read tape source data.

(2) Calling Sequence.

```
ENTRY 'FMIP' USING FM-DD.
CALL 'FMSET' USING FMIO.
CALL 'FMSCAN' USING FM-DD.
CALL 'IPCD' USING FM-DD IP-DD DSD-AREA DSD-CHAR (LR11-HOP).
CALL 'IPREC' USING FM-DD IP-DD
   DSD-AREA
   DSD-CHAR (IP-REC-TYPE-HOP)     DSD-CHAR (IP-REC-IDS-HOP)
   DSD-CHAR (IP-FIELD-HOP)        DSD-CHAR (IP-FIL-OPR-HOP)
   DSD-CHAR (IP-REC-FIL-OPR-HOP) DSD-CHAR (IP-VAL-HOP)
   DSD-CHAR (IP-VAL-LIT-HOP)
   input record
CALL 'IPVAL' USING FM-DD IP-DD
   DSD-CHAR (LR2-HOP)             DSD-AREA
   DSD-CHAR (IP-REC-TYPE-HOP)     DSD-CHAR (IP-REC-IDS-HOP)
   DSD-CHAR (IP-FIELD-HOP)        DSD-CHAR (IP-FIL-OPR-HOP)
   DSD-CHAR (IP-REC-FIL-OPR-HOP) DSD-CHAR (IP-VAL-HOP)
   DSD-CHAR (IP-VAL-LIT-HOP)      input record
   DSD-CHAR (TRANS-ITEM-HOP)      DSD-CHAR (TRANS-TEXT-HOP).
CALL 'IPTRN' USING IP-DD UNSORTED-TRANS-TAB.
   DSD-CHAR (TRANS-TEXT-HOP).
CALL 'IPUN' USING FM-DD IP-DD
   DSD-CHAR (LR2-HOP)             DSD-CHAR (LR11-HOP)
   DSD-CHAR (IP-REC-TYPE-HOP)     DSD-CHAR (IP-FIELD-HOP)
   DSD-CHAR (IP-VAL-HOP)          DSD-CHAR (IP-VAL-LIT-HOP)
   DSD-CHAR (IP-UNIT-HOP)         input record.
CALL 'FMCRD' USING FM-INPUT END-SW.
CALL 'FMPRT' USING PRT-LINE CC.
```

(3) Capabilities.

FMIPX is the first program to be executed when input processing functions are required. It is entered by the FMIP entry point after dynamic loading of the FMIP load module is requested by the FM program through the LINK subroutine. FMIPX is the Input Processor Supervisor and determines the execution sequenc of most of the subprograms of the IP phase of FM.

Upon entry to FMIPX, FMSET is called to supply the entry point address of the FMIO routine to the CALLIO subroutine so that future calls to CALLIO via its FMCRD and FMPRT entry point will result in branching to FMIO to read a card or print a line. FMIPTRN is called to open the transaction file.

The CALL-OPCD paragraph calls the FMSCAN entry point to the FMSCN subroutine to divide the characters on the FMIP control card into words and calls IPCD to analyze the FMIP card, determine the sizes of each table in the File Format Table (FFT) and the Data Source Table (DST), assign relative locations (high order position, HOP, values) to these tables and to transaction item and text areas, and compute the total amount of memory required to hold all of this information. FMIPX uses the MOVARAYS subroutine to insert the total size value into the block size (BLKSIZE) field of the Data Control Block (DCB) of a dummy output file (AREA1-DD) whose buffer becomes a large work area. After FMIPX opens this artificial file, IPCD is called again to read all of the FFT and DST tables and process subsequent FMIP cards applying to the input source. The CALL-IPCD paragraph may be executed more than once if multiple sources of input data are to be processed in a single run. If an error is detected by FMIPCD, the File Maintenance run is terminated immediately.

The GET-NEXT-DATA paragraph determines if the tape source data file has been opened. It is not open for card data or before the first tape record is processed. The GET-NEXT-DATA paragraph is used every time a source data record is to be passed to FMIPREC or FMIPUN and divides the calls into two pairs: card calls and tape calls.

The CARD-CALLS and TAPE-CALLS paragraphs have the function of distinguishing Batch Update runs from Unit Update runs. For card input, the CALL-IPREC paragraph or the CALL-IPUNIT paragraph supplies the data and Data Source Table information to the FMIPREC or FMIPUN subprograms. For tape input, CALL-IPREC-TAPE or CALL-IPUNIT-TAPE performs the same functions. The only difference in the calls to IPREC and IPUN is whether CARD-DATA or TAPE-RECORD is specified in the calling sequence. (The sample calling sequence indicates "input record".) The IPREC calling sequence indicates the beginning of every table in the series of Data Source Tables by specifying its first character position in DSD-AREA. (The HOP field values are inserted by FMIPCD while reading the tables.) IPUN is passed only those tables which it modifies.

Upon completion of either IPREC or IPUN, the PROCESS-DATA paragraph determines what further action on a data record is required based on switch settings from the FMIPREC or FMIPUN subprograms. If TAPE-SW has been set to 2, a new tape record is needed

and control proceeds to the READ-TAPE paragraph. If the record processed is within the previous input group, any blank pseudo subset sequence numbers are changed to low-value and the CALL-IPVAL paragraph is used to further validate the record. If an input group is complete, pseudos are reinitialized for the next input group. If no transactions have been generated (possibly due to errors), the CHECK-EOF paragraph is executed to check for an end of data condition. Otherwise, IPTRN is called to sort the transaction table entries for an input group and write out transaction file records.

The CHECK-EOF paragraph tests INPUT-EOF-SW. Upon end of file, it goes to CALL-FMCRD to look for another input source specification. If there is more data, control is passed to the GET-NEXT-DATA paragraph.

The CALL-IPVAL paragraph calls the FMIPVAL subprogram to handle the detailed field validation of the input record and build transactions from the validated data. FMIPVAL is supplied with all tables of the Data Source Table collection, the input record to be validated, and the areas into which to place the transaction entries and data. Since all records have previously been handled by FMIPREC or FMIPUN, all distinctions between card versus tape data or Unit versus Batch updates are lost by the time FMIPVAL is supplied the input record. After completion of FMIPVAL, control is given to GET-NEXT-DATA.

The CALL-FMCRD paragraph reads a card through FMCRD if tape data has been processed. For either card or tape data, the TEST-END paragraph decides if another FMIP card is present indicating that more source data is to be processed in the same run. If so, certain switches are cleared (so that later actions may be determined from the new packet of FMIP cards) and control is given to CALL-IPCD.

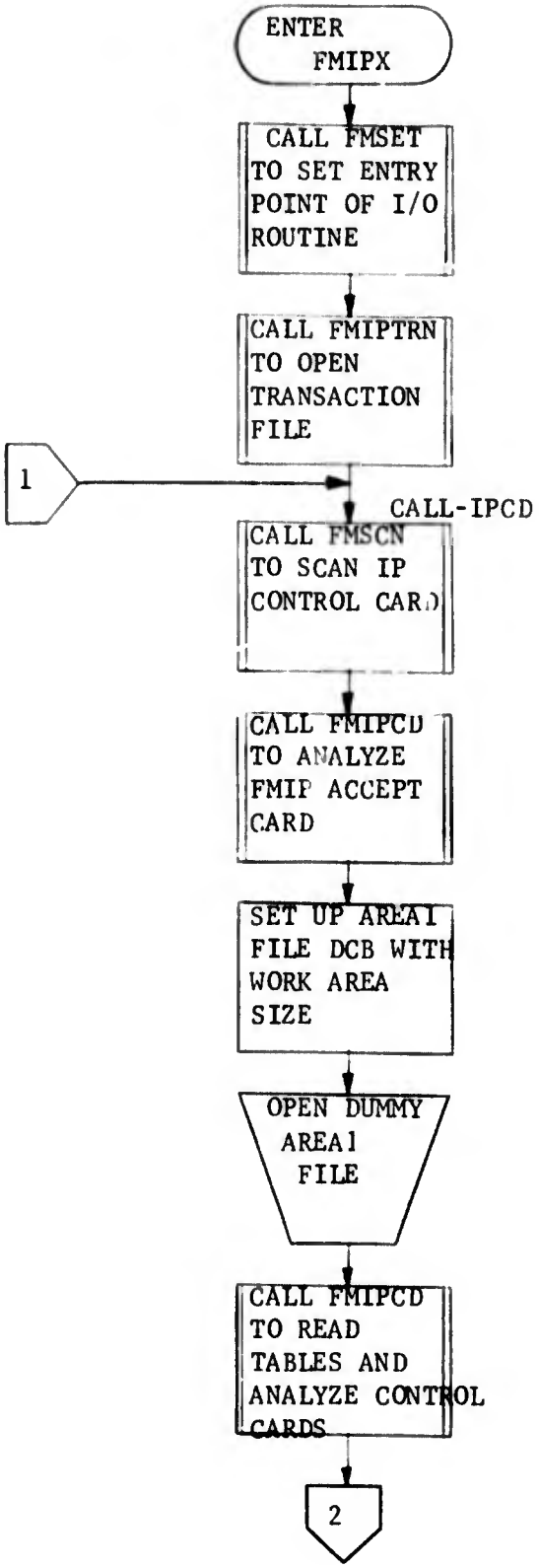
The CALL-IPSRT paragraph is executed when all sources of input data have been processed. If the transaction file was never written on, a "NO TRANSACTIONS GENERATED" message is printed through FMPRT and the File Maintenance run is aborted. If transactions have been generated, IPTRN is called after a 2 is placed in WRITE-SW. This causes IPTRN to close the transaction file and inform FMIPX whether a sort of the transaction file is required. If WRITE-SW contains the value 2 upon completion of FMIPTRN, a sort is required and FMIPX provides this information to the File Maintenance Supervisor by setting FM-LINK-SW to 1. In either case, FMIPX returns to FM at this time.

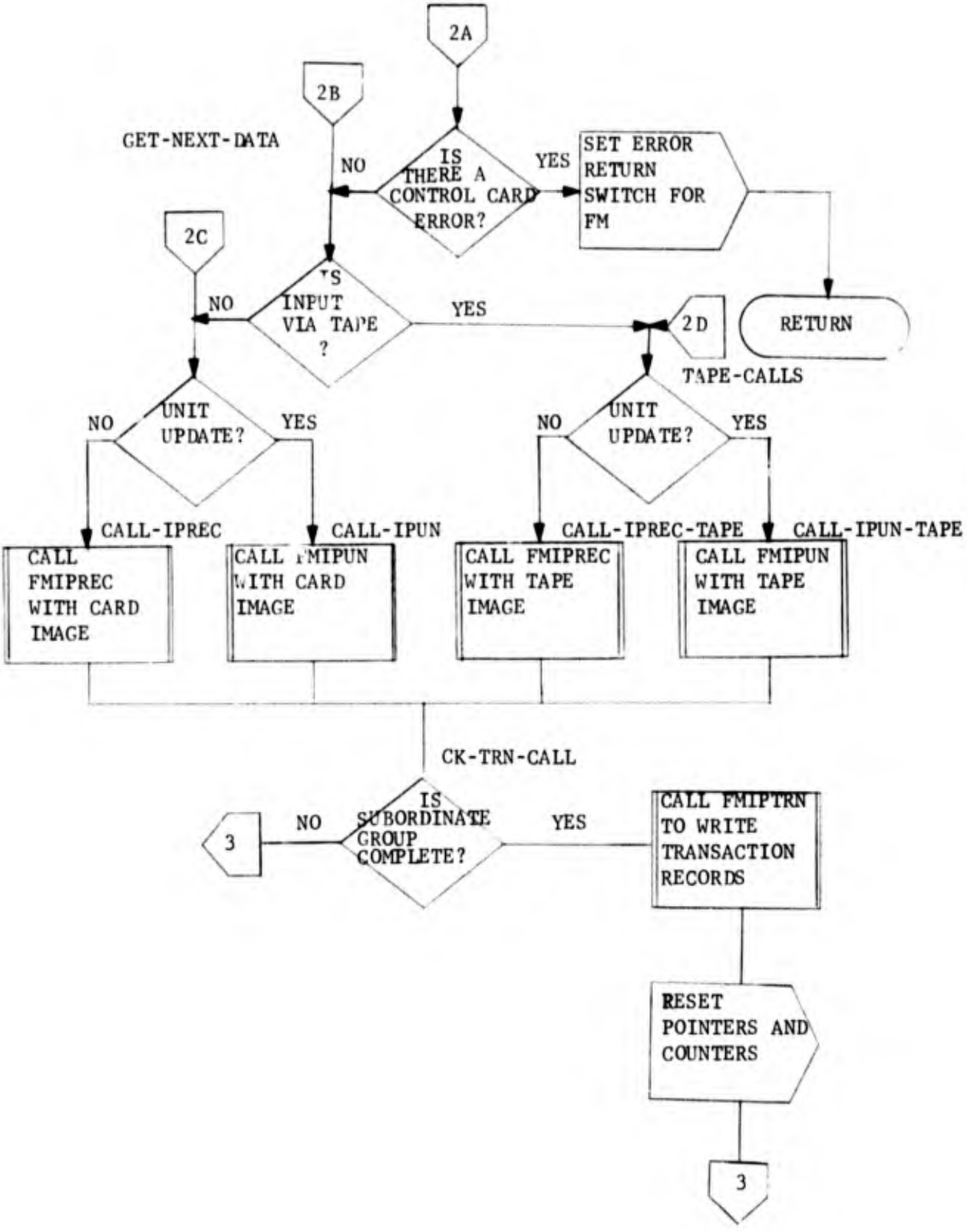
The READ-TAPE paragraph is used to read a single tape source data record. If the tape file is not open, the file is opened. (NOTE: Before the file is opened in the IBM 360, the block size (BLKSIZE), logical record length (LRECL), and record format (RECFM) fields of the COBOL-generated DCB are changed to zero so that this information can be determined from a

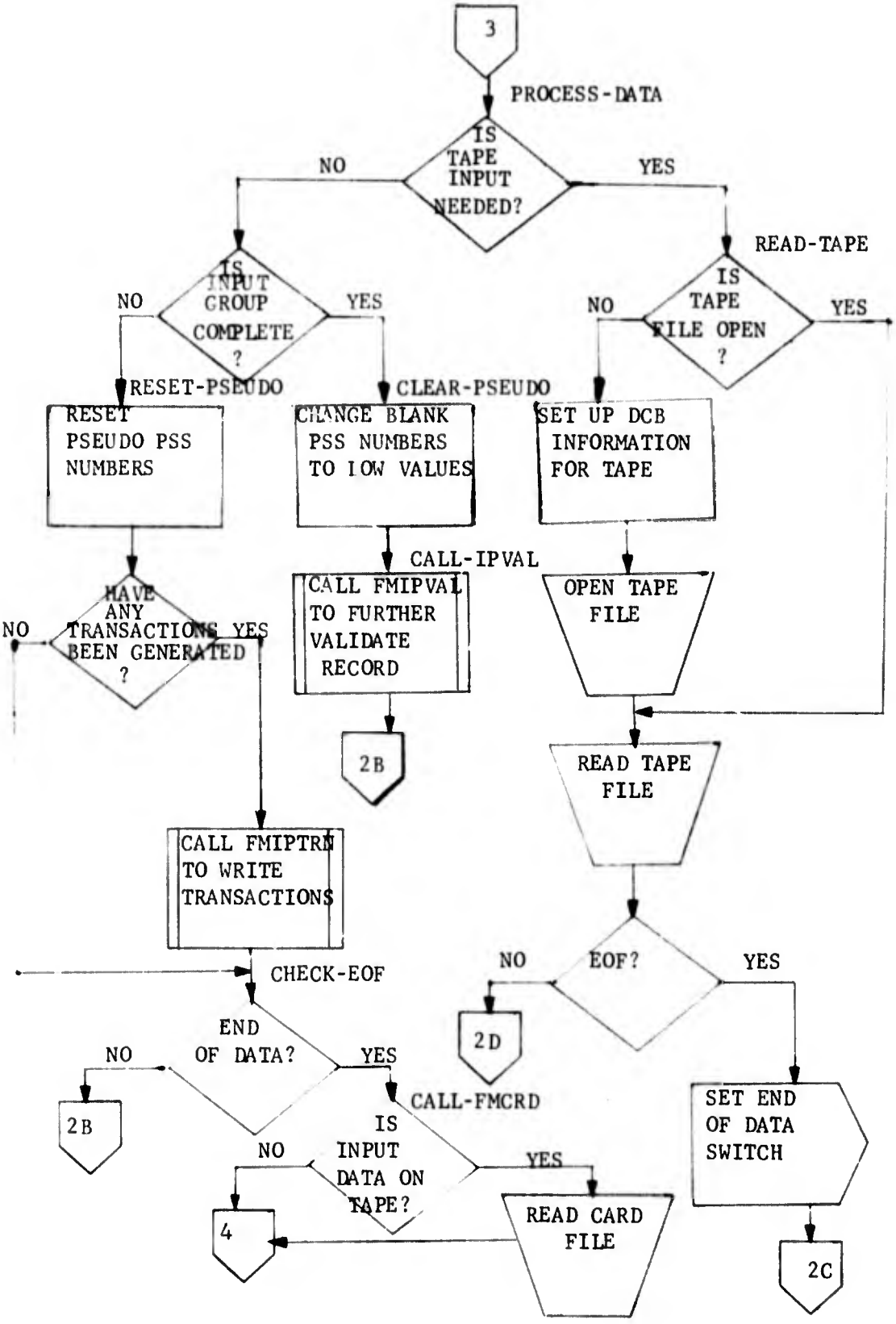
Job Control Language statement for the TAPEIN file. The MOVARAYS subroutine is used for this purpose. This permits the IBM 360 user to specify detailed format information at execution time and allows the program to allocate a buffer size in accordance with actual requirements so that core memory will not be wasted.) When an end of file is reached on the tape file, the file is closed, EOF-SW is set to 1, and control is given to the CARD-CALLS paragraph due to the lack of an actual input record to be passed to FMIPREC or FMIPUN. After a tape record has been read, control is given to the TAPE-CALLS paragraph.

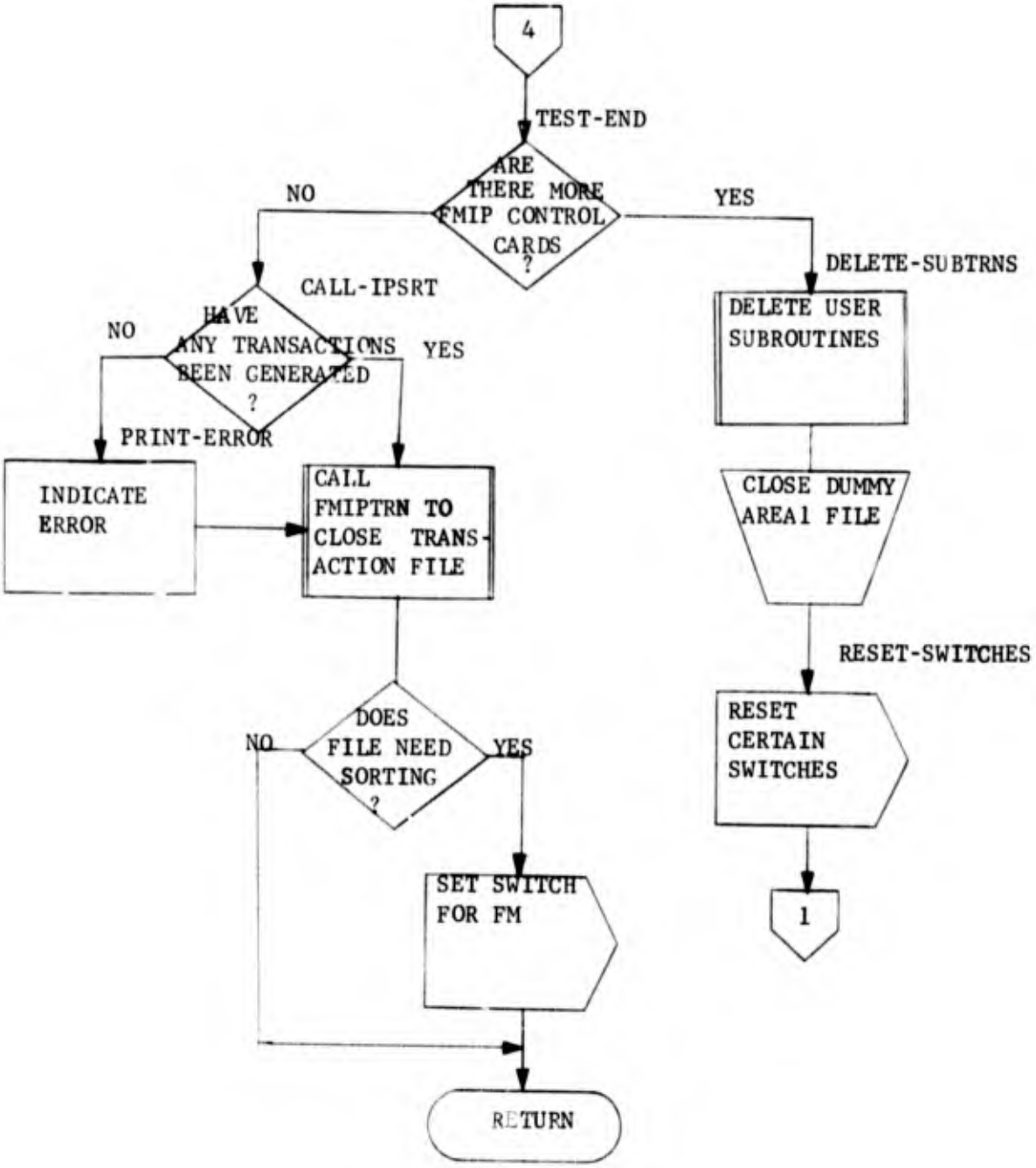
The Working-Storage Section of the FMIPX program is stored in the COBOL COPY library under the name of IPDDV and contains a collection of fields widely used throughout FMIP as well as those used by the FMIPX program itself. IPDDV contains two 01 level groups: IP-DD and CARD-DATA. Both are passed to the IP subprograms requiring them. IPDD on the COPY library is used in the Linkage Section of almost every subprogram within FMIP to represent the data being passed from FMIPX to the subprogram. IPDD contains the 01 level groups IP-DD and TAPE-DATA (which includes a definition of CARD-DATA) as will as separate 01 level groups for each of the FFT and DST tables which are actually within the buffer of the AREA1-DD file (DSD-AREA). This permits subprograms to referr to any field in any table once FMIPCD has read the tables in and FMIPX calls the subprogram with a USING DSD-CHAR (IP-xxx-HOP) for each table.

(4) Flowchart.


ENTER
FMIPX
CALL FMSET TO SET ENTRY POINT OF I/O ROUTINE
CALL FMIPTRN TO OPEN TRANSACTION FILE
1
CALL-IPCD
CALL FMSCN TO SCAN IP CONTROL CARD
CALL FMIPCD TO ANALYZE FMIP ACCEPT CARD
SET UP AREA1 FILE DCB WITH WORK AREA SIZE
OPEN DUMMY AREA1 FILE
CALL FMIPCD TO READ TABLES AND ANALYZE CONTROL CARDS
2


2A
2B
GET-NEXT-DATA
IS THERE A CONTROL CARD ERROR?
NO
YES
SET ERROR RETURN SWITCH FOR FM
RETURN
2C
IS INPUT VIA TAPE ?
NO
YES
2D
TAPE-CALLS
UNIT UPDATE?
NO
YES
UNIT UPDATE?
NO
YES
CALL-IPREC
CALL FMIPREC WITH CARD IMAGE
CALL-IPUN
CALL FMIPUN WITH CARD IMAGE
CALL-IPREC-TAPE
CALL FMIPREC WITH TAPE IMAGE
CALL-IPUN-TAPE
CALL FMIPUN WITH TAPE IMAGE
CK-TRN-CALL
IS SUBORDINATE GROUP COMPLETE?
NO
YES
3
CALL FMIPTRN TO WRITE TRANSACTION RECORDS
RESET POINTERS AND COUNTERS
3


3
PROCESS-DATA
IS TAPE INPUT NEEDED?
NO
YES
READ-TAPE
IS INPUT GROUP COMPLETE ?
NO
YES
IS TAPE FILE OPEN ?
NO
YES
RESET-PSEUDO
CLEAR-PSEUDO
RESET PSEUDO PSS NUMBERS
CHANGE BLANK PSS NUMBERS TO LOW VALUES
SET UP DCB INFORMATION FOR TAPE
CALL-IPVAL
CALL FMIPVAL TO FURTHER VALIDATE RECORD
OPEN TAPE FILE
HAVE ANY TRANSACTIONS BEEN GENERATED ?
NO
YES
2B
READ TAPE FILE
CALL FMIPTRN TO WRITE TRANSACTIONS
CHECK-EOF
EOF?
NO
YES
2D
END OF DATA?
NO
YES
2B
CALL-FMCRD
IS INPUT DATA ON TAPE?
NO
YES
SET END OF DATA SWITCH
4
READ CARD FILE
2C


4
TEST-END
ARE THERE MORE FMIP CONTROL CARDS ?
NO
YES
DELETE-SUBTRNS
DELETE USER SUBROUTINES
CALL-IPSRT
HAVE ANY TRANSACTIONS BEEN GENERATED ?
NO
YES
PRINT-ERROR
INDICATE ERROR
CALL FMIPTRN TO CLOSE TRANSACTION FILE
CLOSE DUMMY AREA1 FILE
RESET-SWITCHES
RESET CERTAIN SWITCHES
DOES FILE NEED SORTING ?
NO
YES
SET SWITCH FOR FM
1
RETURN

b. FMIPCD

(1) Function. To anlyze parameters on FMIP control cards, indicating their specifications by switch settings and reading File Format and Data Source Tables.

(2) Calling Sequence.

```
ENTRY 'IPCD'    USING FM-DD IP-DD DSD-AREA MAJOR-LR11.
CALL  'LB'      USING CALLING-SEQ area.
CALL  'FMCRD'   USING FM-INPUT END-SW.
CALL  'FMPRT'   USING PRT-LINE CC.
CALL  'FMSCAN'  USING FM-DD.
CALL  'IPBIN'   USING FM-DD IP-DD MAJOR-LR11.
```

(3) Capabilities.

FMIPCD is called twice by FMIPX (through the IPCD entry point) for each source of input. The first time, the FMIP ACCEPT card must be processed and the relative location (high order position, HOP, values) of each table in the File Format Table (FFT) and the Data Source Table (DST) must be determined. For the second entry processing at RE-ENTRY-PT reads FFT and DST tables into their allotted spaces and subsequent FMIP cards are validated. When the second word of the first FMIP card is not "ACCEPT", an error message is printed and processing is terminated. Due to the setting of the IP-ERROR switch to 1, any error located by FMIPCD immediately aborts an entire File Maintenance run.

In the ACCEPT-CARD paragraph, the ACCEPT card is checked for a file or FFT name in word 3 and DSD, DST, FREE, or UNIT in word 4. Any other contents of either word is an error.

If DSD or DST is specified (indicating Batch Update), the GET-DST paragraph checks for a valid form of a DST name (last character of name must be "D"). If valid, the control table of the DST (named xxxxDØ1) is read by the Library Subroutine (LB) into area DSDØ1. If the table is not available, a message is printed and the run terminated.

The CHECK-TAPE paragraph determines whether or not the word "TAPE" is specified. If so, TAPE-SW is set to 1; otherwise it is set to 0. If an override value has been specified to increase the size of the transaction text area, the VALIDATE-OVER-RIDE paragraph is executed.

For any valid specification in word 4, the READ-LR1 paragraph reads FFT Logical Record 1 into MAJOR-LR1 through LB. If it is not present, a message is printed and FM is aborted. If LR1 is found, space requirements for other FFT and DST tables must be calculated. For Unit Update, processing continues at UNIT-HOPS. Batch processing uses COMPUTE-HOPS.

The RE-ENTRY-PT paragraph, which is executed during the second entry to FMIPCD, reads FFT LR2 and LR11 into the space previously allotted by UNIT-HOPS or COMPUTE-HOPS. At this point, Unit Update reads the next FMIP control card while Batch Update processing includes reading more Data Source Tables based on the control fields in DSDØ1. All of the tables are placed into DSD-AREA, which is originally defined as the buffer of the dummy AREA1-DD file. The exact location of each table within DSD-AREA was determined by UNIT-HOPS or COMPUTE-HOPS. The GET-DSD paragraph calls LB to read each required table into its proper position, determines that each table is actually in the library, and computes the high order position of the next table segment, if needed. The DSD tables are read in the following order under the conditions specified.

The GROUP-TYPES-LIST (xxxxD02) is always read into the beginning of DSD-AREA.

The REC-TYPE-LIST (xxxxD10) is always read. If the number of record types exceeds 150 (which is beyond the current capability of Language Processor), a second portion of the REC-TYPE-LIST is read (xxxxD11). The high order position of REC-TYPE-LIST is in IP-REC-TYPE-HOP.

The FIELD-LIST is always read. Since it may be divided into segments, multiple reads (xxxxD20, xxxxD21, ...) may be required until all 300-field segments are in memory. The high order position of FIELD-LIST is in IP-FIELD-HOP.

If a name entry was used in a file level OPR specification, the FIL-OPR-NAME-LIST (xxxxD07) and the REC-FIL-OPR-LIST (xxxxD08) are read. The high order positions are IP-FIL-OPR-HOP and IP-REC-FIL-OPR-HOP.

The VAL-LIST is always read. Since it may be divided into segments, multiple reads (xxxxDØØ, xxxxD31, ...) may be required until all segments of 600 validity checks are in core. The high order position is IP-VAL-HOP.

The VAL-LIT-AREA is always read. It may be divided into multiple segments of 5000 characters so that xxxxD40, xxxxD41, ... are read as required. The high order position is IP-VAL-LIT-HOP.

The GET-NEXT-CARD paragraph reads all subsequent FMIP control cards through FMCRD. If a non-FMIP card is found, the error message "FMIP CNTRLEND CARD MISSING" is printed and File Maintenance is aborted. Any FMIP card is printed and its characters are broken up into words by the FMSCN subroutine (FMSCAN entry point). The second word is checked to determine the type of FMIP card.

A CONFIRM card may either specify "ALL," which sets CONFIRM-SW to 2, or may list individual fields to be confirmed during Maintenance Proper. Each field must be defined in the FFT (checking is carried out by FMIPBIN) or the FM run will be terminated after a message is printed. Multiple CONFIRM cards are permitted. If "ALL" is specified and another CONFIRM card is present, an advisory message will be printed and processing will continue with the next card.

The SUBRTN card is permitted only for Unit Update. Its parameters must be of the form name#subrS or name=subrS (# is 3-8 punch; = is 6-8 punch). The required name is a one to five character field name; the subroutine name must end with "S." The presence of the field in the FFT is determined by the FMIPBIN subroutine. Paragraphs SUBRTN-CARD to FLD-5 analyze the SUBRTN card's specifications.

LM, INHIBIT, SOURCE, and RESERVE cards are acceptable to FMIPCD for compatibility with Mark III FFS. The parameters on these cards are ignored.

The CNTRLEND card must be the last card of a packet of FMIP control cards. When the CNTRLEND card is found, RE-ENTRY-SW is reset to space for the next input source, if any, and all subroutines that were specified on LOAD cards are actually brought into memory through the LOAD subroutine. The entry points for each subroutine are saved in SUB-ADDRESS. After loading, control passes to the END-IPCD paragraph which returns the FMIPX.

A CLASS card, which may only be used once for each input source, causes a header image to be set up using the literal on the CLASS card and the current date. DATESUB is used to obtain the execution date in Julian format. YMDSUB converts the Julian date to YYMMDD form. MOVALF moves the CLASS literal to the header image. FMPRT is used to force a page eject and print the new classification header at the top of the page. CLASS-SW is set to 1 to indicate that a CLASS card has occurred.

A LOAD card sets up a table to cause the permanent loading, throughout a single input source, of up to ten conversion and validation subroutines. Each subroutine name, except the last, must be followed by a comma or an error message will be printed and the FM run aborted. Multiple LOAD cards are permitted.

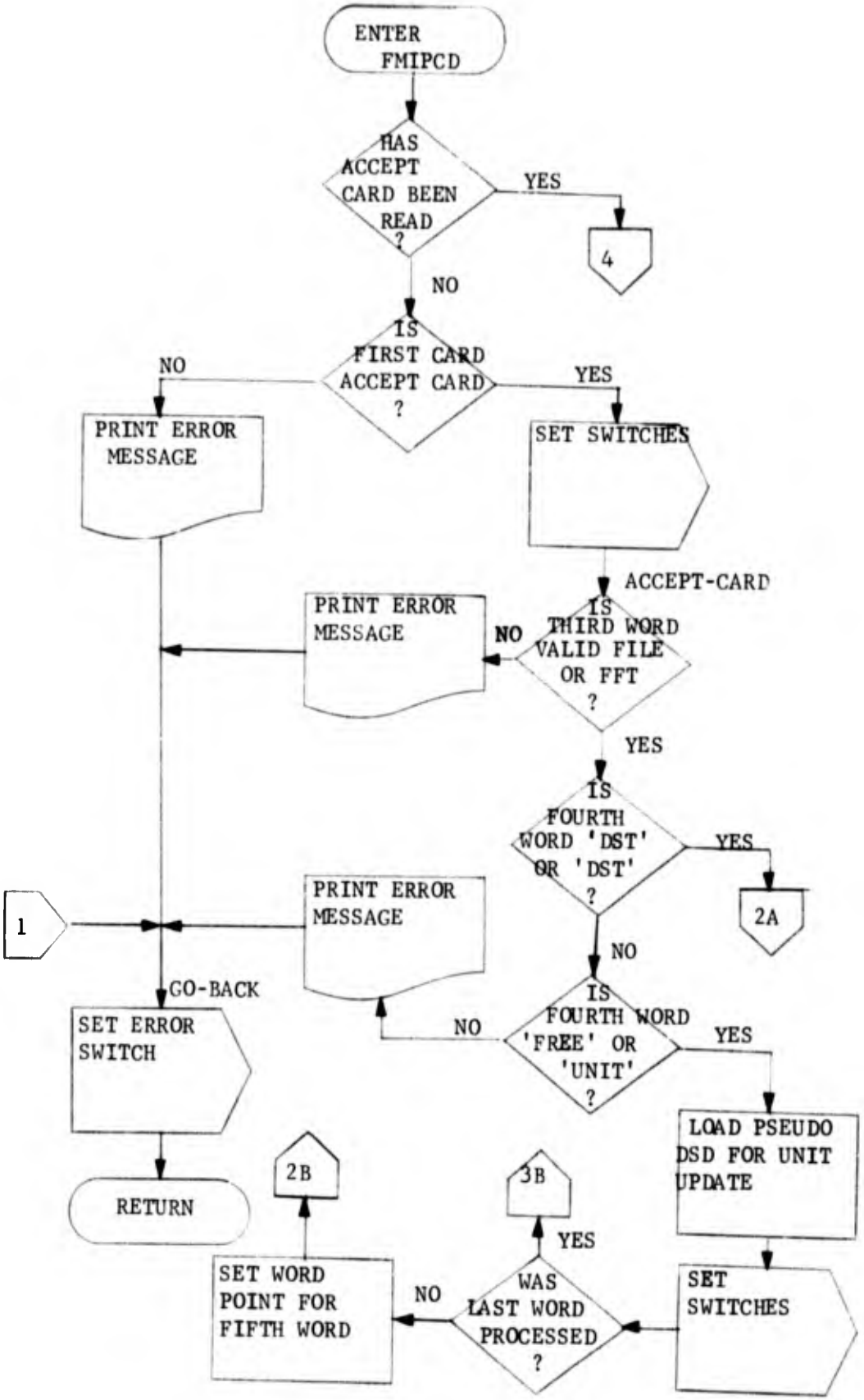
The UNIT-HOPS paragraph sets up the high order position fields for each of special DST tables required by FMIPUN for Unit Update runs. It also determines the HOP values for FFT LR2 and LR11. These values are based on the size of DST tables generated by FMIPUN and are IBM 360 dependent. The size of the transaction text area is set to a default of 2000 characters unless a higher override value was specified.

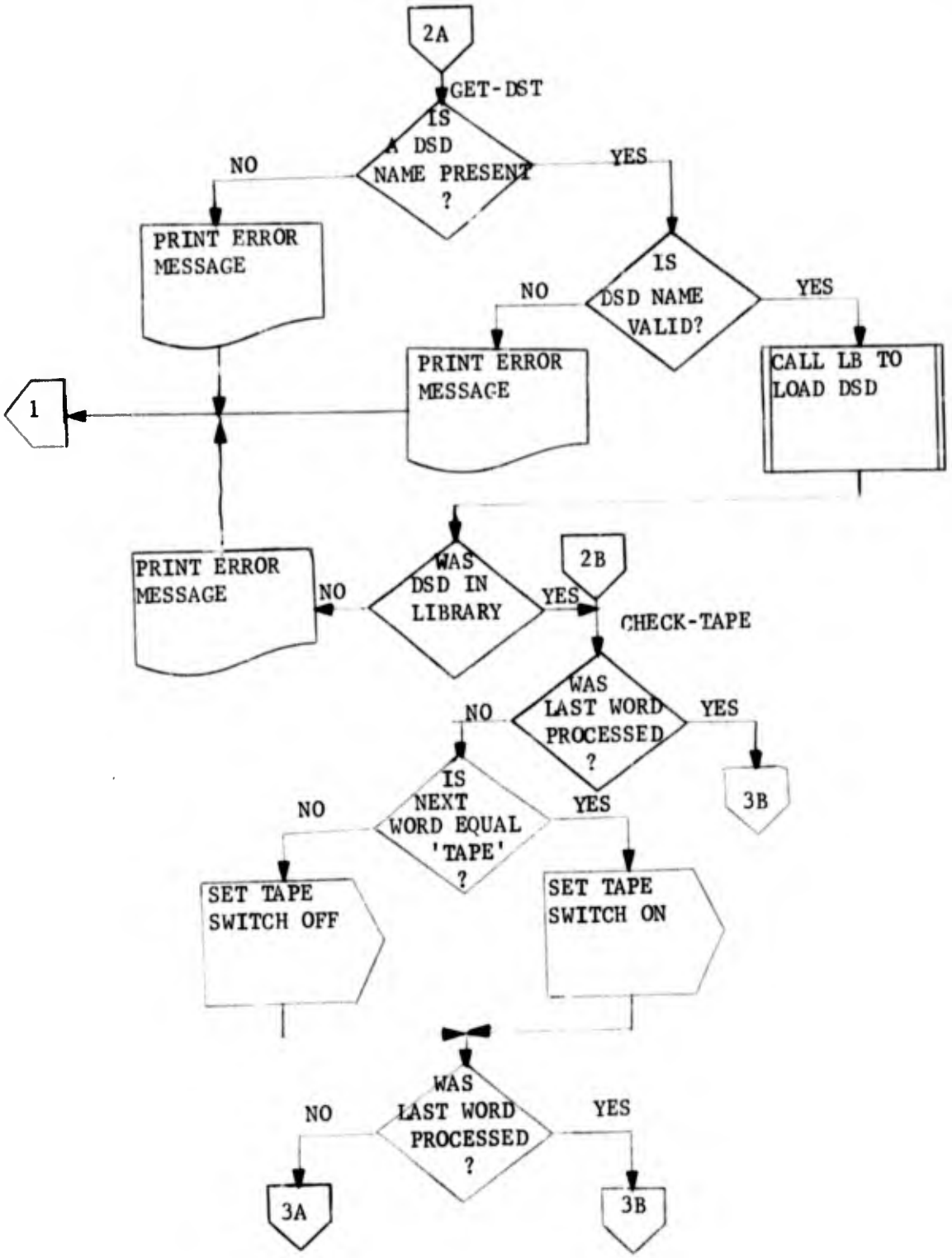
The PSEUDO-INIT paragraph is used for both Unit and Batch Update. It performs INIT-PSEUDO to initialize two tables for pseudo subset sequence number processing. Each GEN-PSEUDO field is set to 600; the USER-PSEUDO-TABLE is set to low-value. In addition, file sequence or order fields are initialized in accordance with DSD requirements.

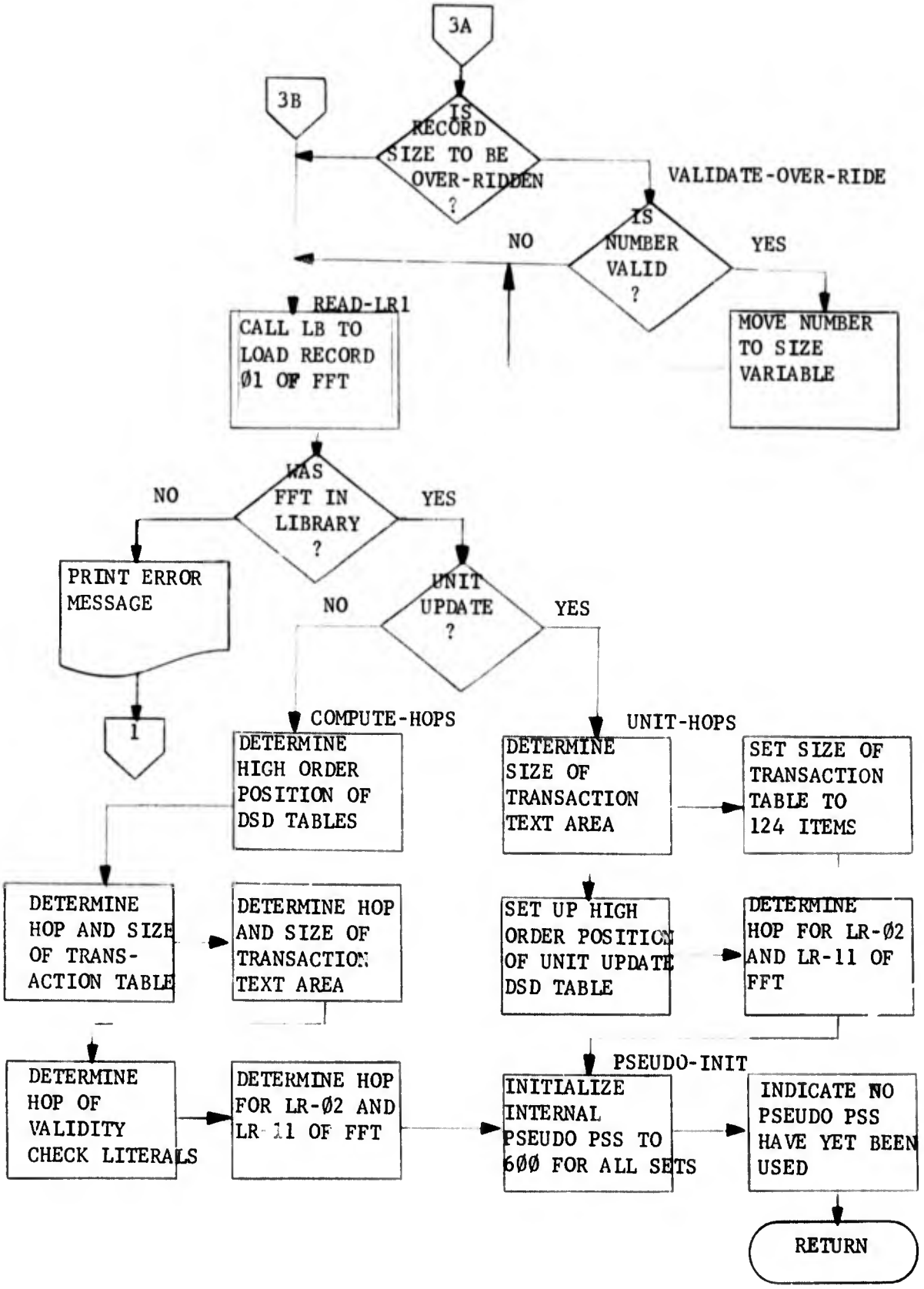
The COMPUTE-HOPS paragraph sets up the high order position fields for each of the FFT and DST tables required by Batch Update. These values are based on the IBM 360 dependent entry sizes for the tables. The size of the transaction text area is set to the largest of the following: the greatest possible amount of data in any source input record (MAX-TOT-FLD-LEN), the user-specified override value (OVER-RIDE-MTFL), or 2000 characters. For compatibility with an earlier version of MIDMS which did not compute MAX-TOT-FLD-LEN in Language Processor, a zero value in MAX-TOT-FLD-LEN causes a default to 9999 characters. This affects only very old DST's which have never been rerun under LP.

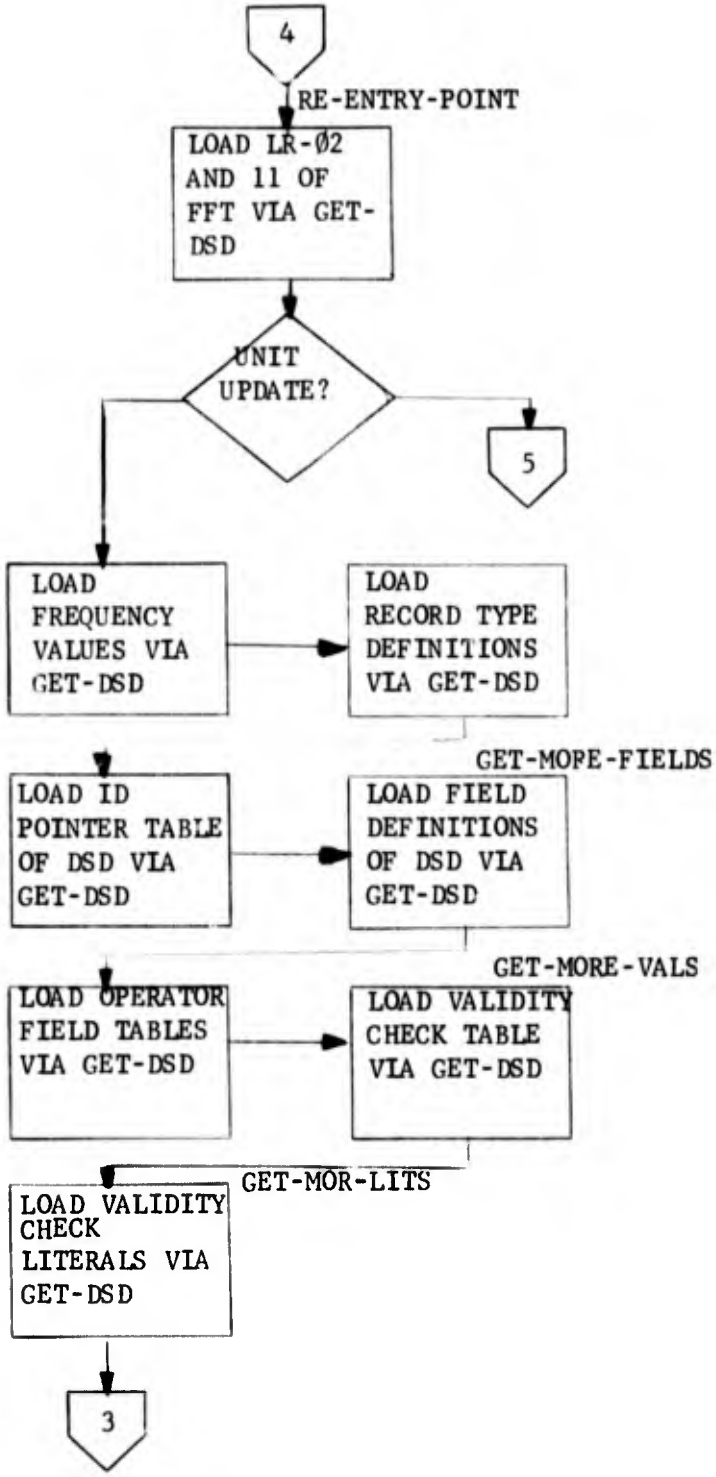
VALIDATE-OVER-RIDE verifies whether each of four characters, in what is presumed to be an override value for the size of the transaction text area, is actually numeric. Numeric overrides values are moved to OVER-RIDE-MTFL-X, which is the alphanumeric definition for OVER-RIDE-MTFL. Non-numeric values are ignored. Processing continues at READ-LR1 in either case.

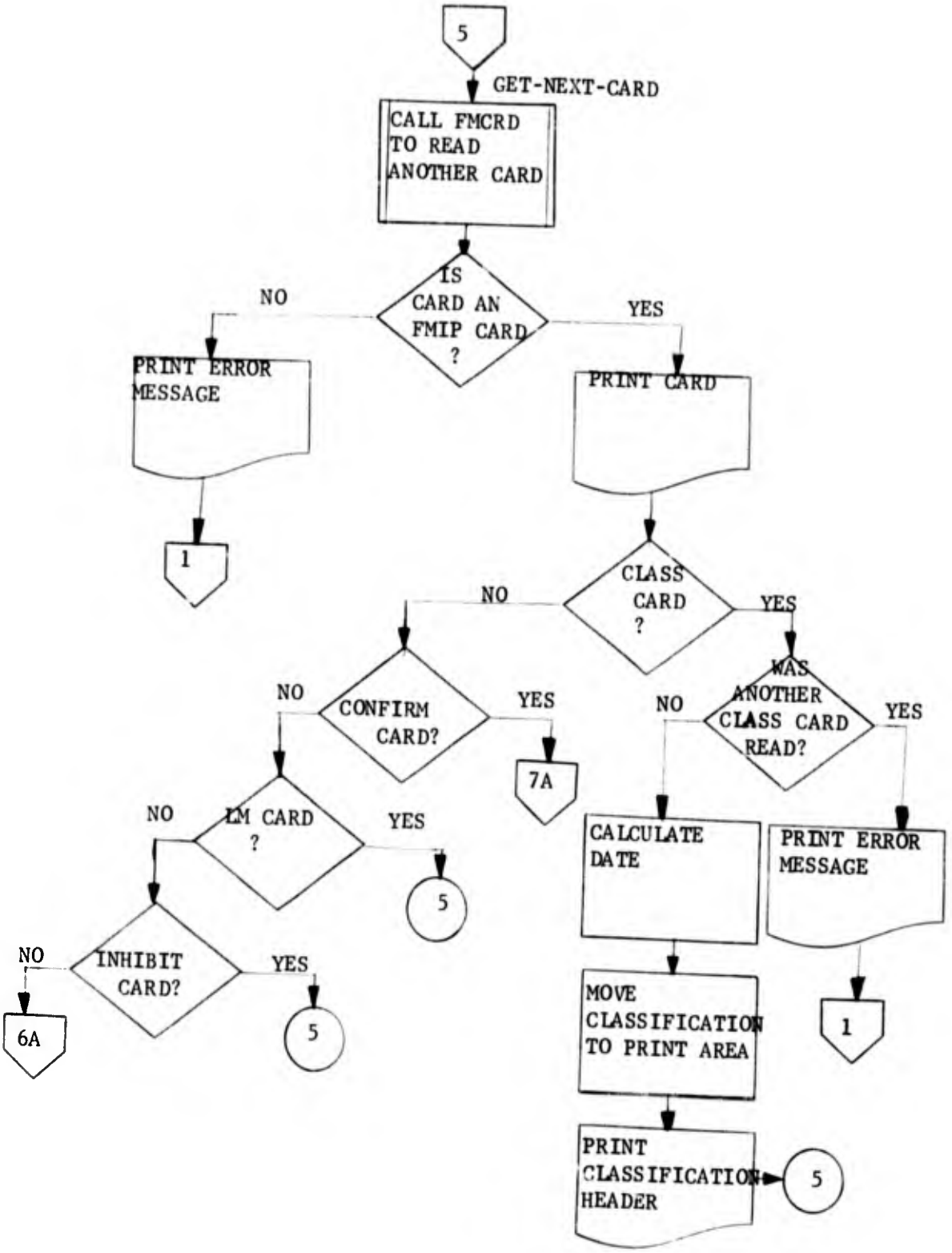
(4) Flowchart.

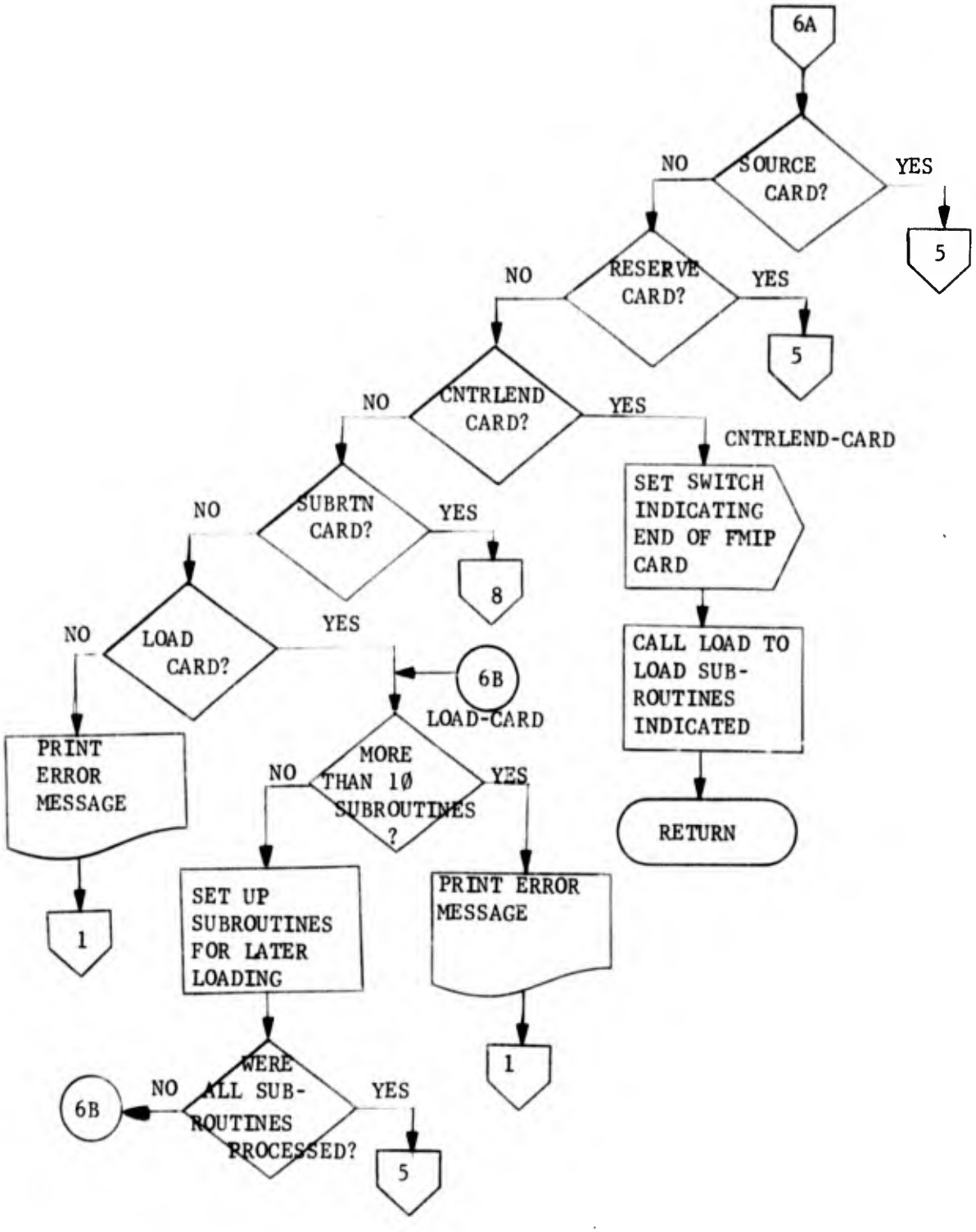

ENTER FMIPCD
HAS ACCEPT CARD BEEN READ ?
YES
4
NO
IS FIRST CARD ACCEPT CARD ?
NO
YES
PRINT ERROR MESSAGE
SET SWITCHES
ACCEPT-CARD
PRINT ERROR MESSAGE
NO
IS THIRD WORD VALID FILE OR FFT ?
YES
IS FOURTH WORD 'DST' OR 'DST' ?
YES
2A
1
PRINT ERROR MESSAGE
NO
GO-BACK
SET ERROR SWITCH
NO
IS FOURTH WORD 'FREE' OR 'UNIT' ?
YES
LOAD PSEUDO DSD FOR UNIT UPDATE
RETURN
2B
3B
YES
SET WORD POINT FOR FIFTH WORD
NO
WAS LAST WORD PROCESSED ?
SET SWITCHES

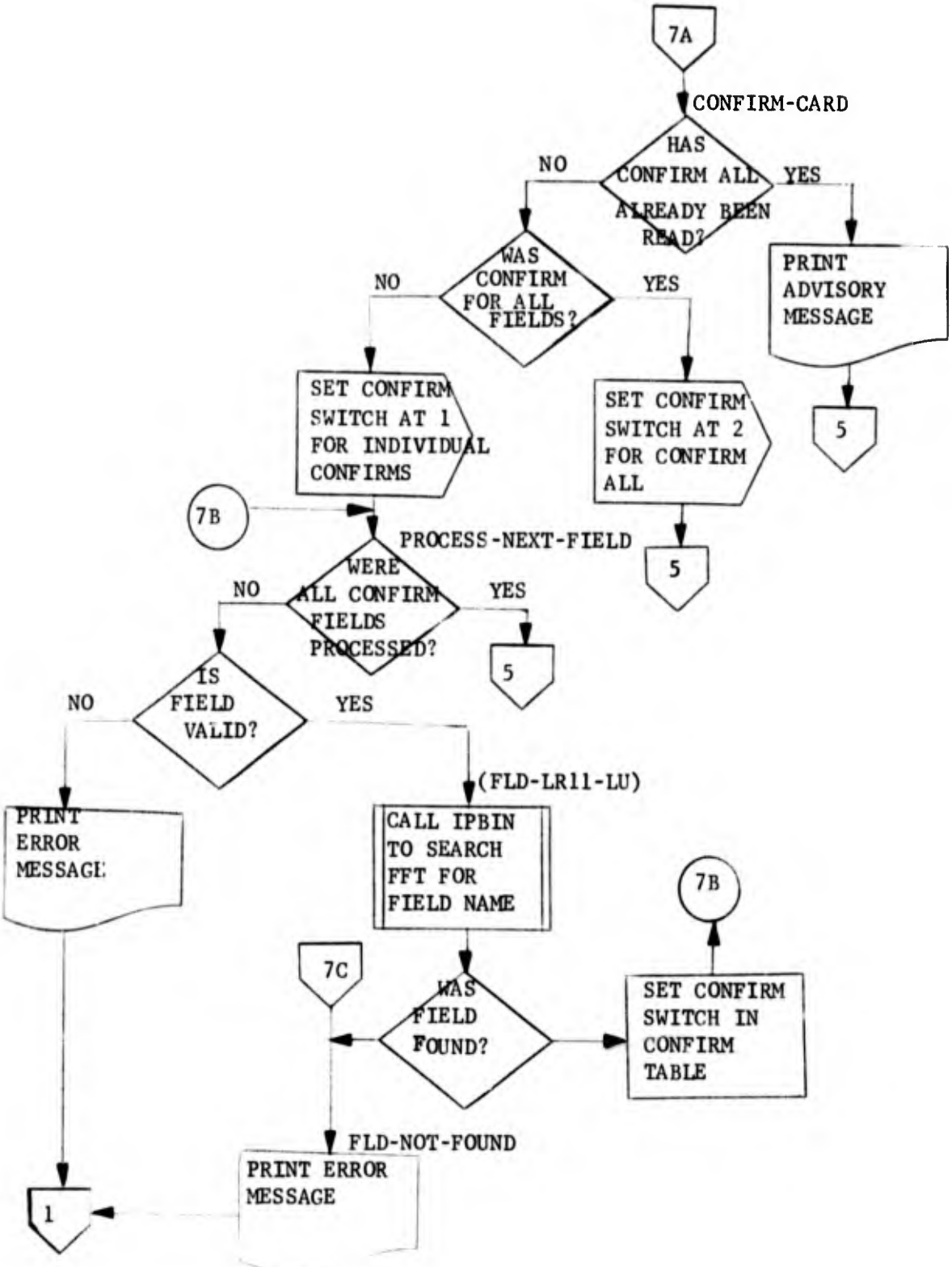

2A
GET-DST
IS A DSD NAME PRESENT ?
NO
YES
PRINT ERROR MESSAGE
IS DSD NAME VALID?
NO
YES
PRINT ERROR MESSAGE
CALL LB TO LOAD DSD
1
PRINT ERROR MESSAGE
NO
WAS DSD IN LIBRARY
YES
2B
CHECK-TAPE
WAS LAST WORD PROCESSED ?
NO
YES
3B
IS NEXT WORD EQUAL 'TAPE' ?
NO
YES
SET TAPE SWITCH OFF
SET TAPE SWITCH ON
WAS LAST WORD PROCESSED ?
NO
YES
3A
3B

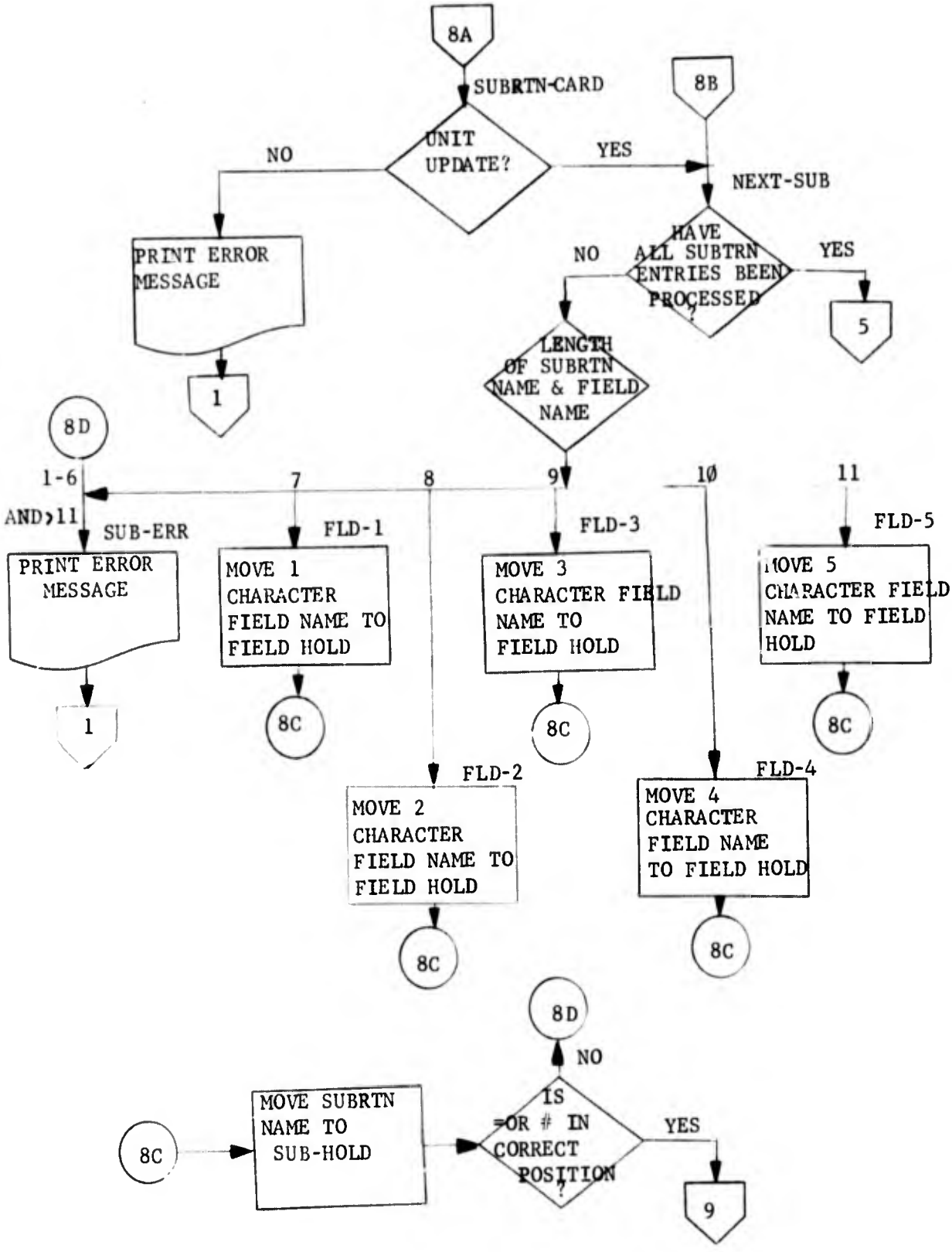

3A
3B
IS RECORD SIZE TO BE OVER-RIDDEN ?
VALIDATE-OVER-RIDE
IS NUMBER VALID ?
NO
YES
READ-LR1
CALL LB TO LOAD RECORD Ø1 OF FFT
MOVE NUMBER TO SIZE VARIABLE
WAS FFT IN LIBRARY ?
NO
YES
PRINT ERROR MESSAGE
UNIT UPDATE ?
NO
YES
1
COMPUTE-HOPS
UNIT-HOPS
DETERMINE HIGH ORDER POSITION OF DSD TABLES
DETERMINE SIZE OF TRANSACTION TEXT AREA
SET SIZE OF TRANSACTION TABLE TO 124 ITEMS
DETERMINE HOP AND SIZE OF TRANS-ACTION TABLE
DETERMINE HOP AND SIZE OF TRANSACTION TEXT AREA
SET UP HIGH ORDER POSITION OF UNIT UPDATE DSD TABLE
DETERMINE HOP FOR LR-Ø2 AND LR-11 OF FFT
DETERMINE HOP OF VALIDITY CHECK LITERALS
DETERMINE HOP FOR LR-Ø2 AND LR-11 OF FFT
PSEUDO-INIT
INITIALIZE INTERNAL PSEUDO PSS TO 6ØØ FOR ALL SETS
INDICATE NO PSEUDO PSS HAVE YET BEEN USED
RETURN

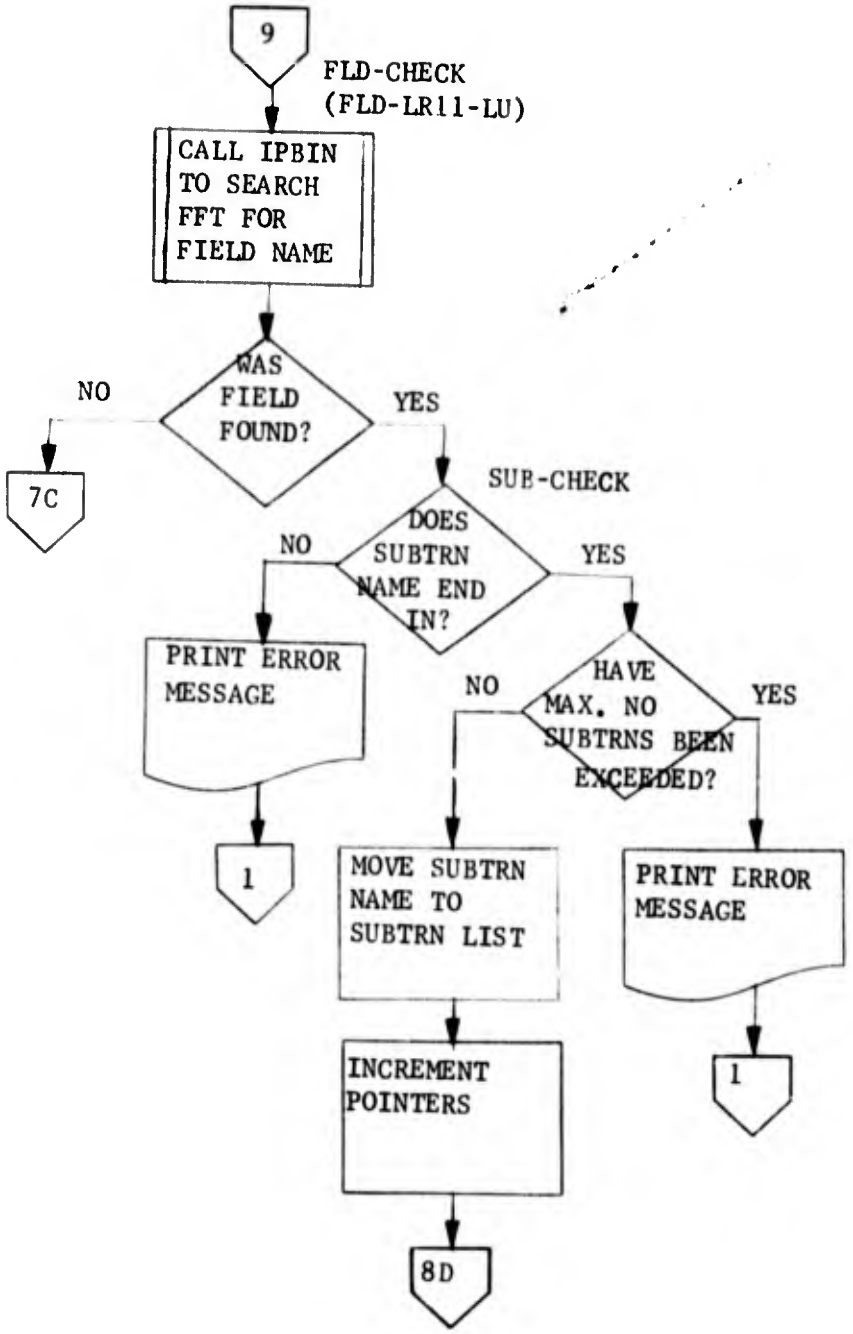

4
RE-ENTRY-POINT
LOAD LR-Ø2 AND 11 OF FFT VIA GET-DSD
UNIT UPDATE?
5
LOAD FREQUENCY VALUES VIA GET-DSD
LOAD RECORD TYPE DEFINITIONS VIA GET-DSD
GET-MOPE-FIELDS
LOAD ID POINTER TABLE OF DSD VIA GET-DSD
LOAD FIELD DEFINITIONS OF DSD VIA GET-DSD
GET-MORE-VALS
LOAD OPERATOR FIELD TABLES VIA GET-DSD
LOAD VALIDITY CHECK TABLE VIA GET-DSD
GET-MOR-LITS
LOAD VALIDITY CHECK LITERALS VIA GET-DSD
3


5
GET-NEXT-CARD
CALL FMCRD TO READ ANOTHER CARD
IS CARD AN FMIP CARD ?
NO
YES
PRINT ERROR MESSAGE
PRINT CARD
1
CLASS CARD ?
NO
YES
CONFIRM CARD?
NO
YES
7A
WAS ANOTHER CLASS CARD READ?
NO
YES
LM CARD ?
NO
YES
5
CALCULATE DATE
PRINT ERROR MESSAGE
INHIBIT CARD?
NO
YES
6A
5
MOVE CLASSIFICATION TO PRINT AREA
1
PRINT CLASSIFICATION HEADER
5


6A
SOURCE CARD?
NO
YES
5
RESERVE CARD?
NO
YES
5
CNTRLEND CARD?
NO
YES
CNTRLEND-CARD
SET SWITCH INDICATING END OF FMIP CARD
CALL LOAD TO LOAD SUB-ROUTINES INDICATED
RETURN
SUBRTN CARD?
NO
YES
8
LOAD CARD?
NO
YES
6B
LOAD-CARD
PRINT ERROR MESSAGE
1
MORE THAN 10 SUBROUTINES ?
NO
YES
SET UP SUBROUTINES FOR LATER LOADING
PRINT ERROR MESSAGE
1
WERE ALL SUB-ROUTINES PROCESSED?
NO
YES
6B
5


7A
CONFIRM-CARD
HAS CONFIRM ALL ALREADY BEEN READ?
NO
YES
PRINT ADVISORY MESSAGE
5
WAS CONFIRM FOR ALL FIELDS?
NO
YES
SET CONFIRM SWITCH AT 1 FOR INDIVIDUAL CONFIRMS
SET CONFIRM SWITCH AT 2 FOR CONFIRM ALL
5
7B
PROCESS-NEXT-FIELD
WERE ALL CONFIRM FIELDS PROCESSED?
NO
YES
5
IS FIELD VALID?
NO
YES
(FLD-LR11-LU)
PRINT ERROR MESSAGE
CALL IPBIN TO SEARCH FFT FOR FIELD NAME
7B
7C
WAS FIELD FOUND?
SET CONFIRM SWITCH IN CONFIRM TABLE
FLD-NOT-FOUND
PRINT ERROR MESSAGE
1


8A
SUBRTN-CARD
UNIT UPDATE?
NO
YES
8B
NEXT-SUB
PRINT ERROR MESSAGE
1
HAVE ALL SUBTRN ENTRIES BEEN PROCESSED?
NO
YES
5
LENGTH OF SUBRTN NAME & FIELD NAME
8D
1-6
7
8
9
10
11
AND>11
SUB-ERR
PRINT ERROR MESSAGE
1
FLD-1
MOVE 1 CHARACTER FIELD NAME TO FIELD HOLD
8C
FLD-2
MOVE 2 CHARACTER FIELD NAME TO FIELD HOLD
8C
FLD-3
MOVE 3 CHARACTER FIELD NAME TO FIELD HOLD
8C
FLD-4
MOVE 4 CHARACTER FIELD NAME TO FIELD HOLD
8C
FLD-5
MOVE 5 CHARACTER FIELD NAME TO FIELD HOLD
8C
8C
MOVE SUBRTN NAME TO SUB-HOLD
IS =OR # IN CORRECT POSITION?
8D
NO
YES
9


9
FLD-CHECK
(FLD-LR11-LU)
CALL IPBIN TO SEARCH FFT FOR FIELD NAME
WAS FIELD FOUND?
NO
YES
7C
SUB-CHECK
DOES SUBTRN NAME END IN?
NO
YES
PRINT ERROR MESSAGE
HAVE MAX. NO SUBTRNS BEEN EXCEEDED?
NO
YES
1
MOVE SUBTRN NAME TO SUBTRN LIST
PRINT ERROR MESSAGE
INCREMENT POINTERS
1
8D

c. FMIPREC

(1) Function. To perform the high level processing of Batch Update source data records.

(2) Calling Sequence.

```
ENTRY 'IPREC'      USING FM-DD IP-DD
      GRP-TYPES-LIST            SUB-TAG-LIST
      SUB-ID-LIST               SUB-TYPES-LIST
      REC-TYPE-LIST             REC-IDS-LIST
      FIELD-LIST                FIL-OPR-NAME-LIST
      REC-FIL-OPR-LIST          VAL-LIST
      VAL-LIT-AREA
      INPUT-RECORD              TAPE-DATA.
CALL  'FMCRD'      USING CARD-DATA END-SW.
CALL  'FMPRT'      USING PRT-LINE CC.
CALL  'MOVARAYS'   USING ...*
CALL  'COMARAYS'   USING ...*
CALL  'OPTRN'      USING IP-DD UNSORTED-TRANS-TAB TEXT-TAB.
```

* Several calls are made to MOVARAYS and COMARAYS subroutines.

(3) Capabilities.

The FMIPREC subprogram handles the preliminary processing of source data records and sets up certain control fields and switches for use by FMIPX and FMIPVAL. The following explains the major control fields used or set by FMIPREC (in the general order they are first referenced in the program):

TAPE-SW indicates the state of processing of a tape source data file. TAPE-SW=0 for card data, 1 for tape data, or 2 when FMIPREC requests FMIPX to read a tape record.

EOF-SW indicates whether or not the end of file has been reached for tape input source data. Ø = not end of file, 1 = end of file.

NEW-GROUP-SW indicates whether or not the previous input group is terminated. (An input group is a series of consecutive input records applying to a single data file record and having the same group ID value.) NEW-GROUP-SW is Ø for not a new group or 1 for a new group. Upon first entry to FMIPREC, NEW-GROUP-SW is set to 1 until the first group ID is determined.

ACTUAL-FREQ-TABLE is a collection of ACTUAL-FREQ entries containing the number of times a record has appeared in an input group. Its entries correspond to the entries of the GRP-TYPES-LIST and are validated by testing the codes in GRP-FREQ.

TRANS-ITEM-CNT contains the number of transaction entries in UNSORTED-TRANS-TAB. It is cleared to zero by FMIPREC for each input group.

TEXT-COUNT contains the number of characters of transaction text in TEXT-TAB. It is cleared to zero by FMIPREC for each input group.

REJ-GROUP-SW indicates whether or not the current group has been rejected due to an error. It is 0 for no reject, 1 for reject the group, or 2 to reject the balance of the group. (FMIPVAL sets the latter when transactions have exceeded the capacity of UNSORTED-TRANS-TAB.) Whenever REJ-GROUP-SW is not Ø, FMIPREC eliminates from further processing all records in the input group.

SORT-SUB contains the number of current subordinate group for use in the sort key of transaction records. (A subordinate group is a subdivision of an input group resulting from the possibility that transactions from an input record will exceed the allotted item table or text area. Subordinate groups written by FMIPTRN will be rejoined by OMMPTRN.) SORT-SUB is a counter beginning with 1 for each group. It is set to 0 when a group rejection requires that previously written transaction records be logically eliminated from further processing. (OM will ignore transaction records with non-zero SORT-SUB values for a group when it finds a zero in SORT-SUB.)

GRP-ID-HOLD contains the group ID as it comes from a new input record. It is compared against GRP-ID which contains the ID of the current input group.

GRP-SEQ-ORD-SW indicates whether a group level sequence or order test is required. It is 0 for no test, 1 for sequence, 2 for order.

GRP-SEQ is the field containing the group sequence number when sequence checking was specified in the DSD GROUP card. It is set to 0 for every input group.

GRP-ORD is the group order field when order checking was specified on the DSD GROUP card. It is set to low-value for every input group.

GRP-RECID contains the file ID as it is built up from the data source records in an input group. Each part of GRP-RECID is obtained once with further appearance of ID fields ignored. To insure that each ID field is obtained once and only once, CDMS-ID-SWS is set to all zeroes for each group and a CDMS-ID-SW corresponding to a MIDMS ID specified on the file card if the DSD is set to 1 when the ID field is processed. GRP-RECID is used as the MIDMS file record ID in the sort key of the transaction file.

INPUT-EOF-SW indicates the logical end of file condition after an actual end of source data card or tape file occurs. This permits complete processing of the last input group when a physical end of file is located and reinitialization in FMIPREC at the beginning of a new data source. INPUT-EOF-SW is 0 normally and 1 at the end of data and when a new data source is processed.

GRP-NUM contains a group number counter for use in the sort key of transaction records. It is used to associate all transaction records for all subordinate groups of a group and to order transactions within a record ID when multiple groups occur for one record.

END-SW indicates whether or not a logical end of file has been reached for card input source data. 0 = not end of file, not 0 = end of file. The logical end of file usually occurs when a FMIP or FMMP card is found. (A physical end of file may be valid if the user supplied the appropriate JCL to save the transaction file.)

Since the control fields in FMIPREC have been described in some detail, processing within paragraphs will be summarized.

START-IPREC determines whether a tape record has just been read, its end of file status, and whether or not a new group is being started.

START-GROUP initializes control fields and switches at the beginning of an input group.

CHECK-SIZE and SET-DCB-POINTERS are obsolete.

READ-RECORD determines whether a card or tape data record is to be read.

READ-CARD reads a card through FMCRD.

PROCESS-EOF is used when an end of file occurs.

READ-TAPE causes reading of a tape record by returning to FMIPX.

CHECK-TYPE and TEST-REC-TYPE validate the record code in the input record by using the COMARAYS subroutine.

BUILD-GROUP-ID determines whether the input record contains a group ID.

GET-GROUP-FIELD obtains the data in the group ID fields and places it into GRP-ID-HOLD by using the MOVARAYS subroutine.

GET-GRP-FLD is used when RECORD CODE NONE was specified in the DSD. The file ID is treated as the group ID and its fields are placed into GRP-ID-HOLD by the MOVARAYS subroutine.

CHECK-GROUP determines whether or not the group ID has changed.

TEST-GROUP is used when the group ID has changed to begin validation of the old group.

CHECK-FREQ and NEXT-FREQ determine that all record frequencies are valid according to DSD specifications.

GROUP-FREQ-ERR is used when any record frequency is invalid.

CHECK-ID-START and CHECK-ID determine that all MIDMS ID fields have been found in the input.

CDMS-ID-ERR is used when the MIDMS ID has not been completely specified.

PROCESS-RECORD to MOVE-GRP-ORD validate file or group sequence/order as specified in the DSD. The COMARAYS subroutine is used for the compare.

CHECK-REC/CHECK-REC-FREQ determines if the DSD-specified record frequency is exceeded.

CHECK-CDMS-ID, GET-CDMS-ID, GET-CDMS-FIELD, and GET-NEXT-CDMS-ID obtain the MIDMS ID fields from the record by using the MOVARAYS subroutine.

TEST-MAXES determines whether the transactions which may be produced by the current record could exceed the allocated limits of the item table or the text area.

SET-TRN-SW sets a switch to tell FMIPX to call FMIPTRN to write a subordinate group of transaction records when TEXT-MAKES determines that area limits may be exceeded.

FINISH-IPREC is the last paragraph of FMIPREC to be executed. It returns to FMIPX.

PROCESS-ERROR is used for every error condition found by FMIPREC and determines the type of error action specified in the DSD.

REJECT-GROUP is used when the "G" error code was specified in the DSD.

REJECT-RECORD is used when the "R" error code was specified in the DSD.

CONTINUE is used when the "C" error code was specified in the DSD.

CONTINUE-SW is a GO TO which is altered to one of four paragraphs depending on the type of error to be ignored after the error message is printed.

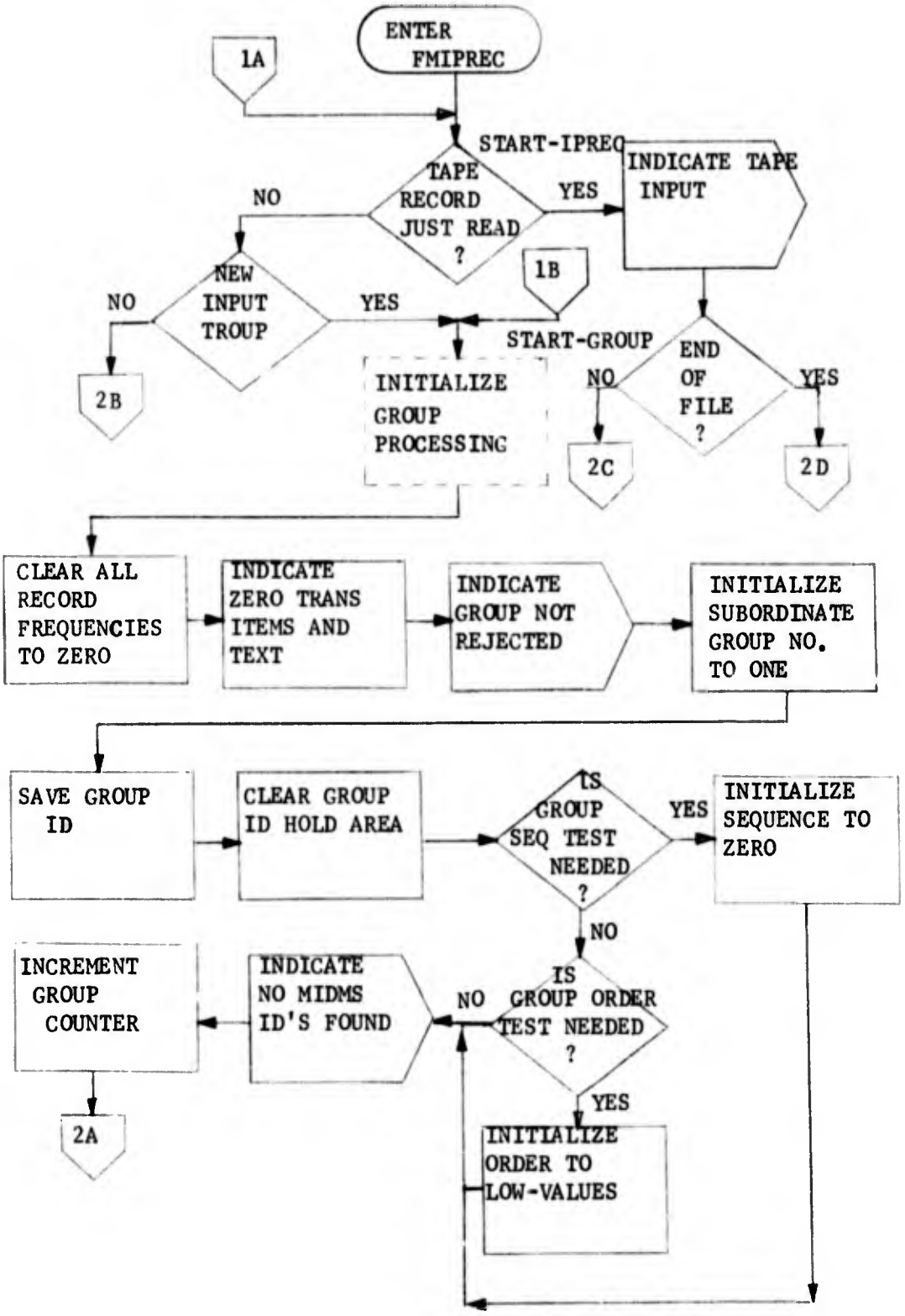
PRINT-RECID moves GRP-RECID to an error message area.

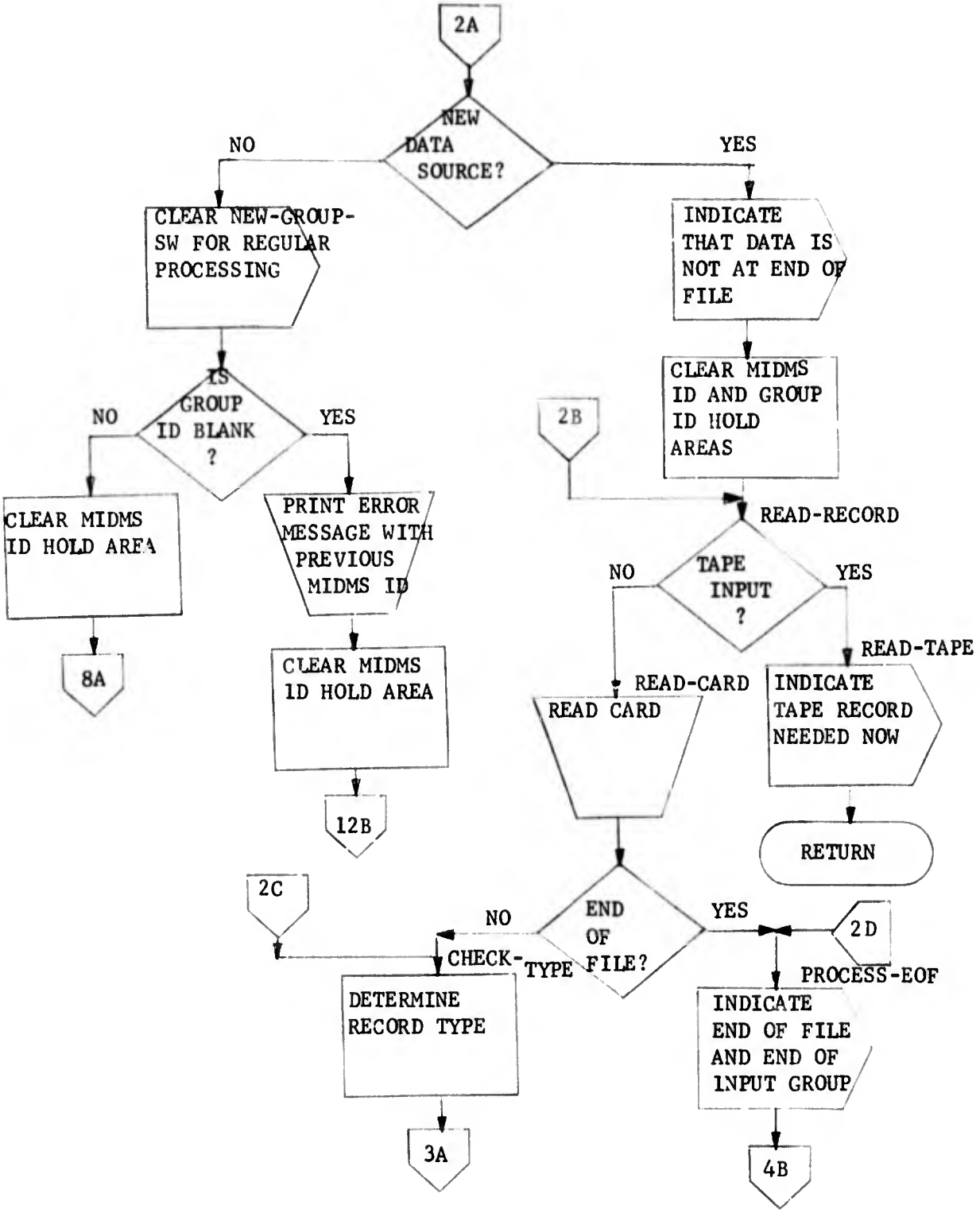
PRINT-ERROR moves the error message area to PRT-LINE for printing.

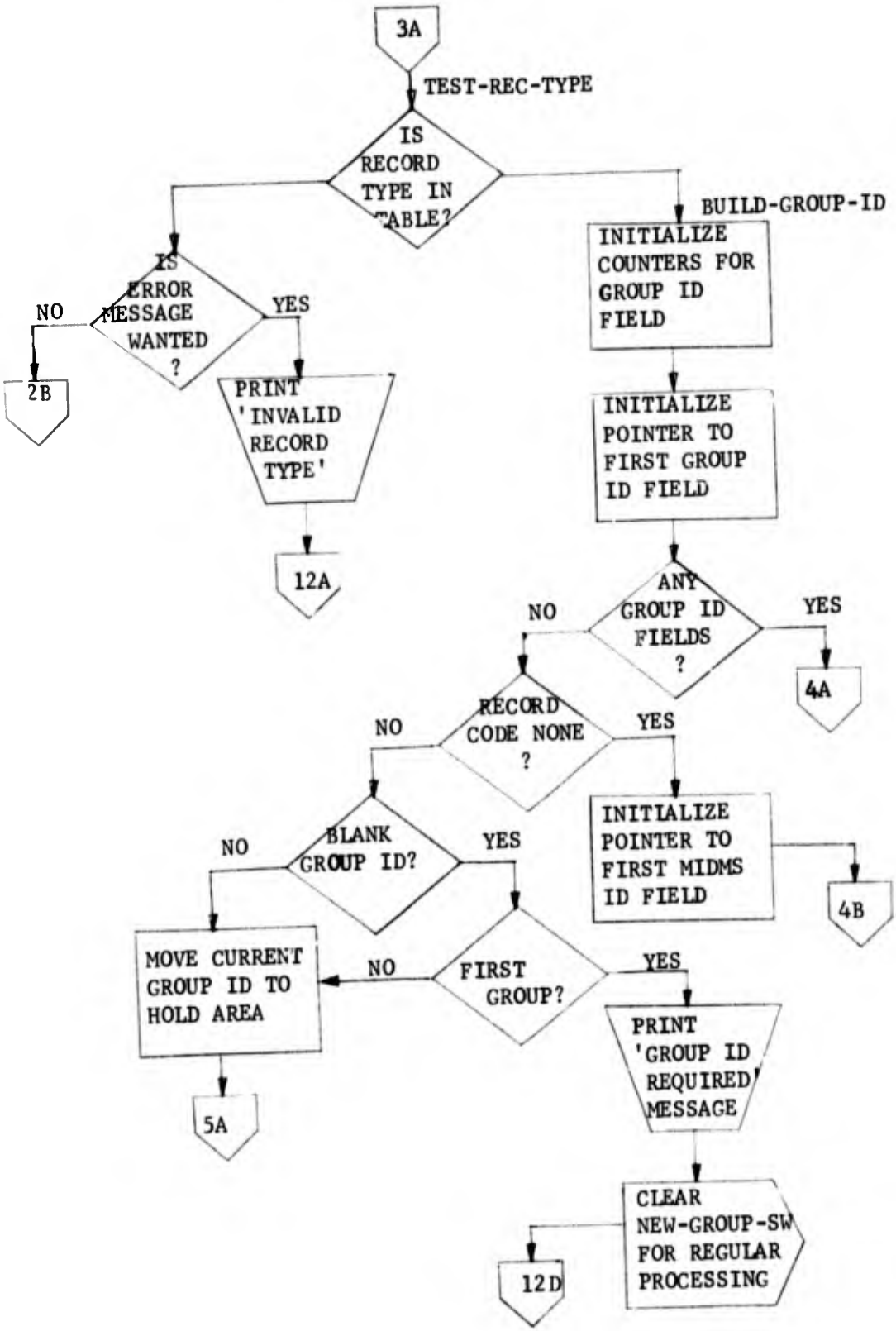
PRINT-LINE prints PRT-LINE through FMPRT.

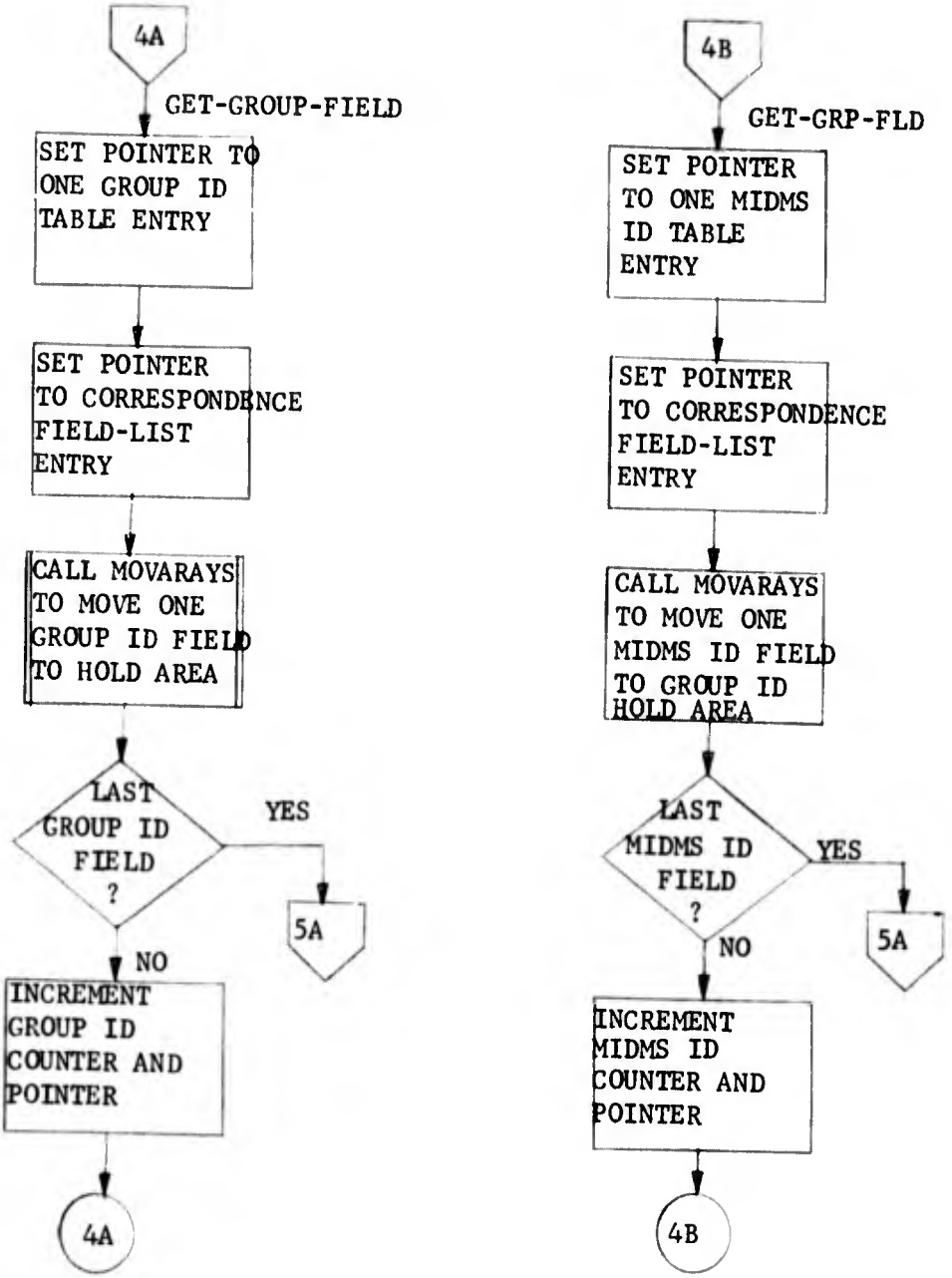
CHECK-FOR-END determines whether processing can continue with the next input group or processing terminates with an end of data condition.

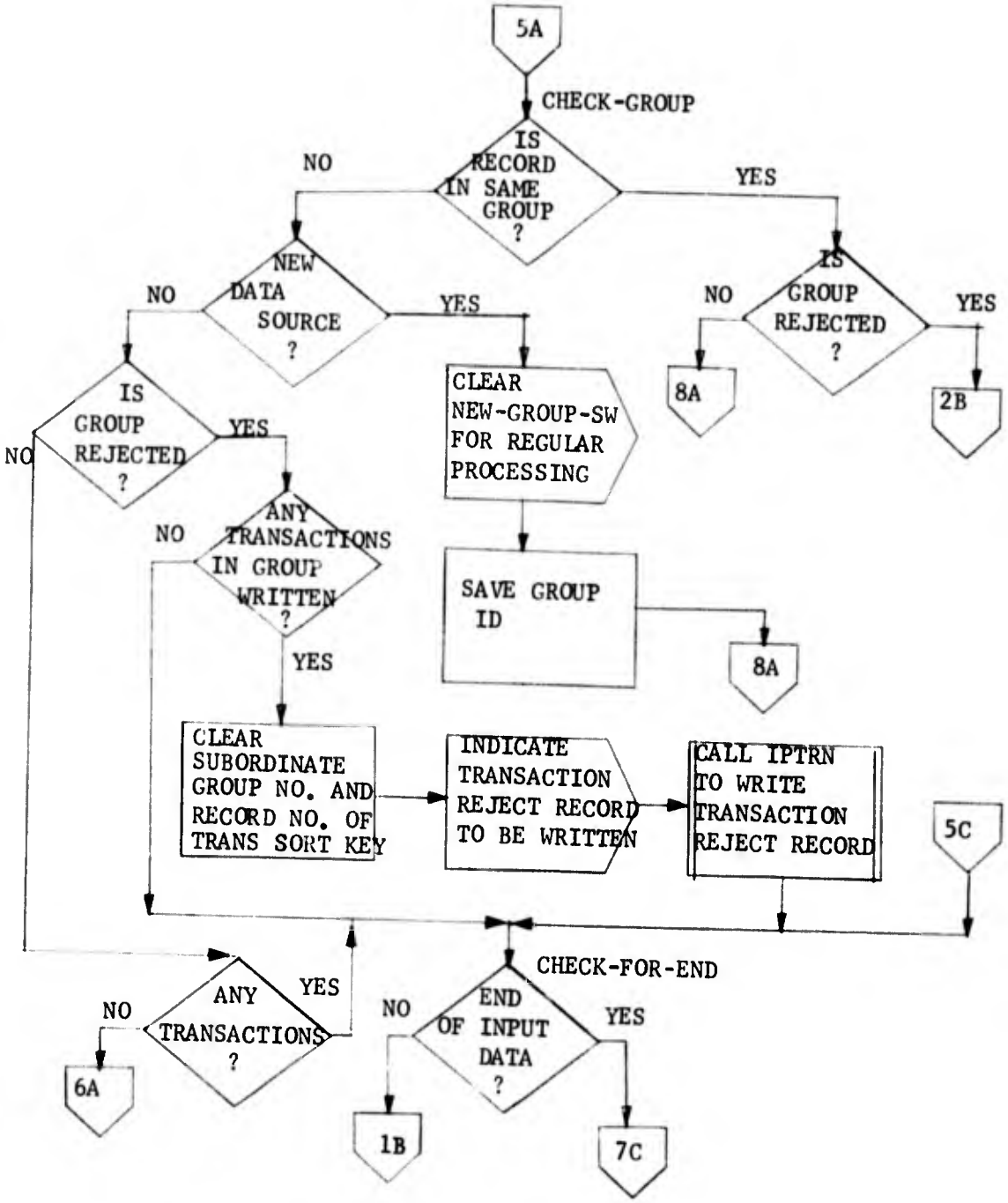
(4) Flowchart.

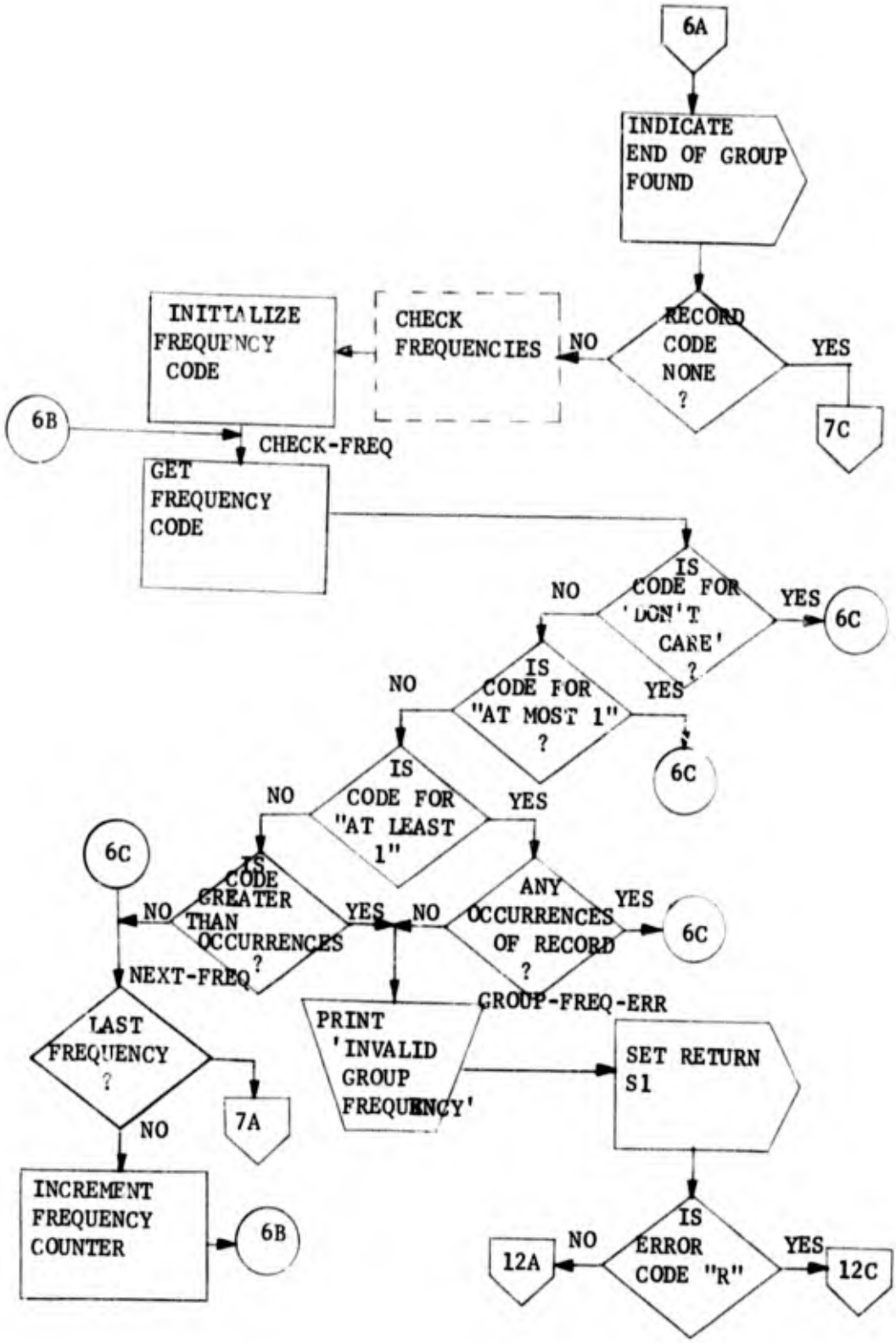

ENTER FMIPREC
1A
START-IPREC
TAPE RECORD JUST READ ?
NO
YES
INDICATE TAPE INPUT
NEW INPUT TROUP
NO
YES
1B
2B
START-GROUP
INITIALIZE GROUP PROCESSING
END OF FILE ?
NO
YES
2C
2D
CLEAR ALL RECORD FREQUENCIES TO ZERO
INDICATE ZERO TRANS ITEMS AND TEXT
INDICATE GROUP NOT REJECTED
INITIALIZE SUBORDINATE GROUP NO. TO ONE
SAVE GROUP ID
CLEAR GROUP ID HOLD AREA
IS GROUP SEQ TEST NEEDED ?
YES
INITIALIZE SEQUENCE TO ZERO
NO
INCREMENT GROUP COUNTER
INDICATE NO MIDMS ID'S FOUND
NO
IS GROUP ORDER TEST NEEDED ?
YES
INITIALIZE ORDER TO LOW-VALUES
2A

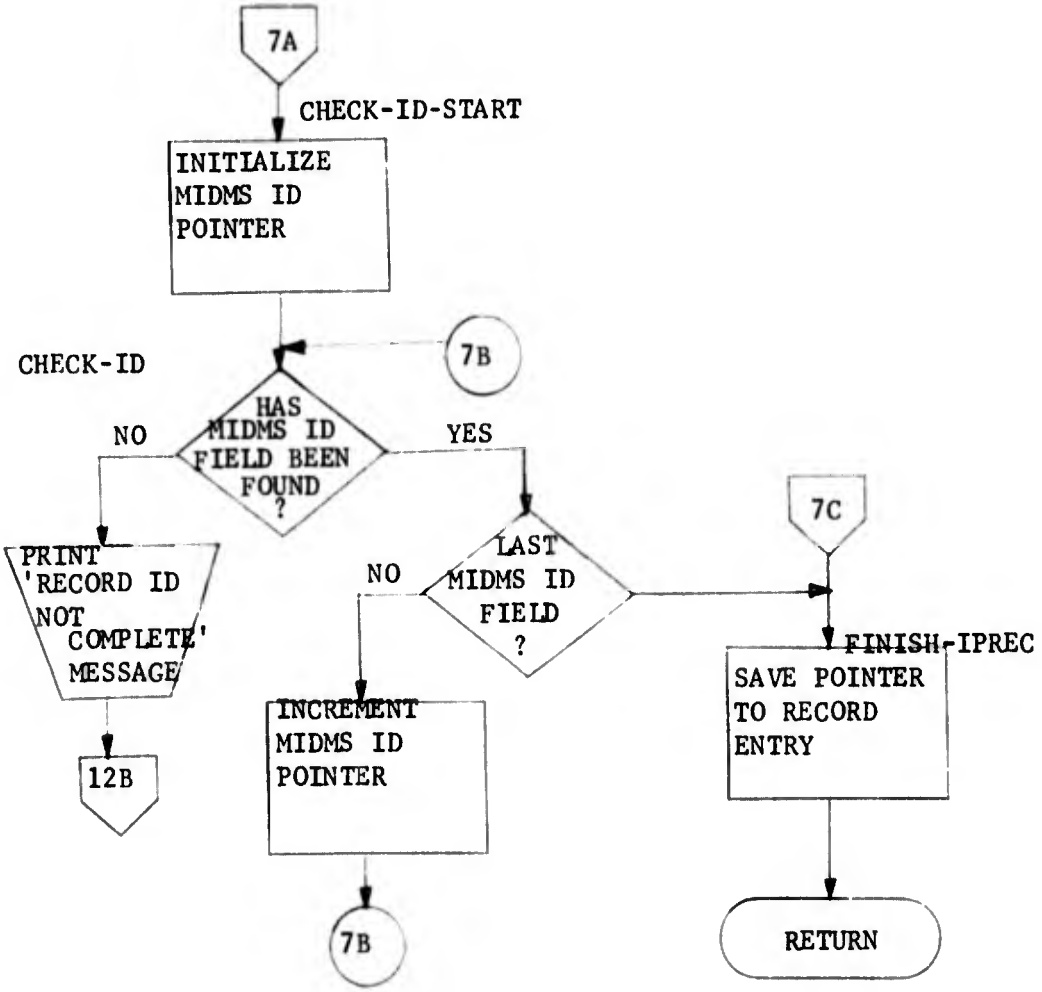

2A
NEW DATA SOURCE?
NO
YES
CLEAR NEW-GROUP-SW FOR REGULAR PROCESSING
INDICATE THAT DATA IS NOT AT END OF FILE
IS GROUP ID BLANK ?
NO
YES
CLEAR MIDMS ID AND GROUP ID HOLD AREAS
2B
CLEAR MIDMS ID HOLD AREA
PRINT ERROR MESSAGE WITH PREVIOUS MIDMS ID
READ-RECORD
TAPE INPUT ?
NO
YES
8A
CLEAR MIDMS ID HOLD AREA
READ-CARD
READ CARD
READ-TAPE
INDICATE TAPE RECORD NEEDED NOW
12B
RETURN
2C
END OF FILE?
NO
YES
2D
CHECK-TYPE
PROCESS-EOF
DETERMINE RECORD TYPE
INDICATE END OF FILE AND END OF INPUT GROUP
3A
4B

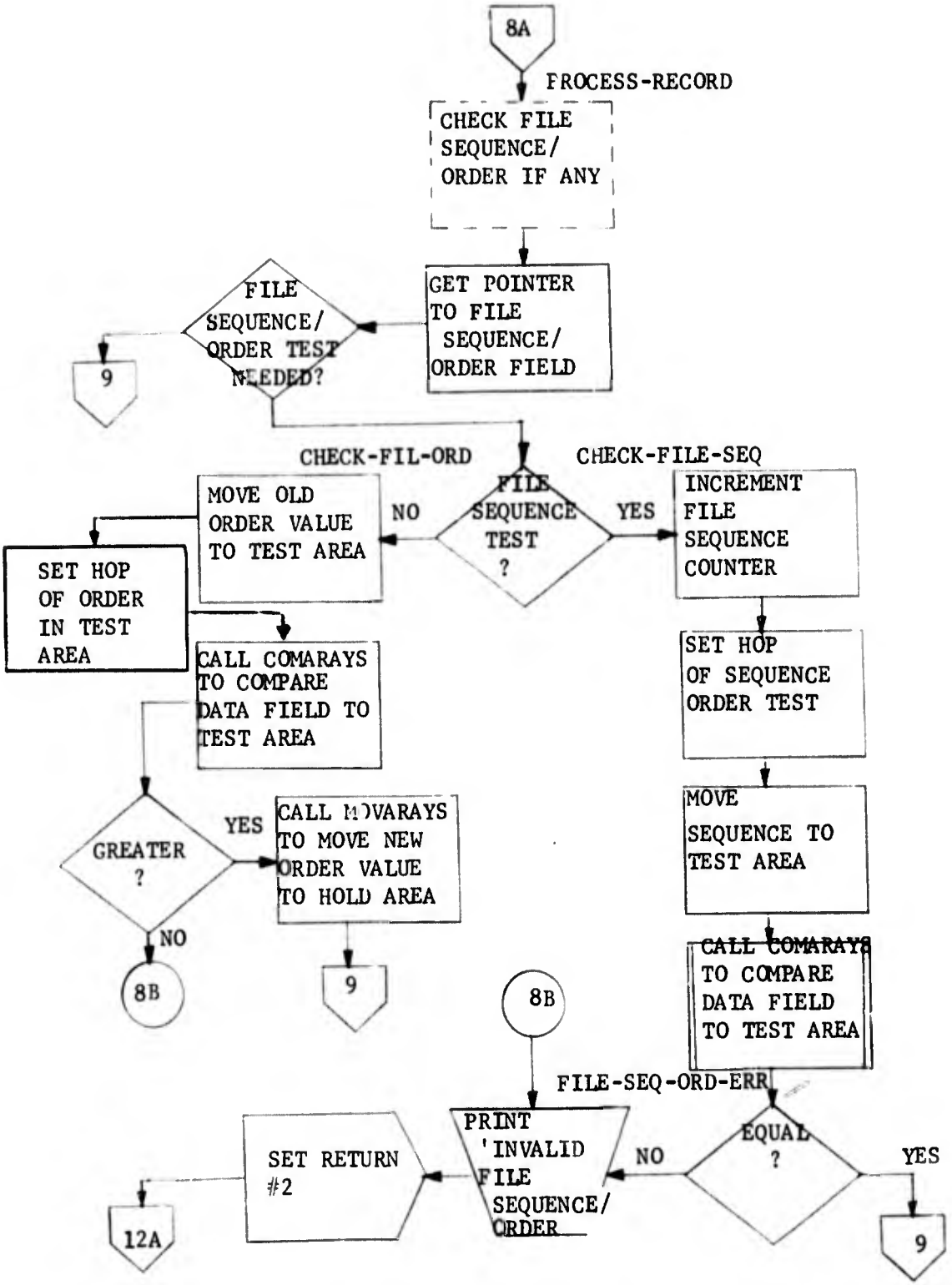

3A
TEST-REC-TYPE
IS RECORD TYPE IN TABLE?
BUILD-GROUP-ID
INITIALIZE COUNTERS FOR GROUP ID FIELD
IS ERROR MESSAGE WANTED ?
NO
YES
2B
PRINT 'INVALID RECORD TYPE'
12A
INITIALIZE POINTER TO FIRST GROUP ID FIELD
ANY GROUP ID FIELDS ?
NO
YES
4A
RECORD CODE NONE ?
NO
YES
INITIALIZE POINTER TO FIRST MIDMS ID FIELD
4B
BLANK GROUP ID?
NO
YES
MOVE CURRENT GROUP ID TO HOLD AREA
FIRST GROUP?
NO
YES
PRINT 'GROUP ID REQUIRED' MESSAGE
5A
CLEAR NEW-GROUP-SW FOR REGULAR PROCESSING
12D

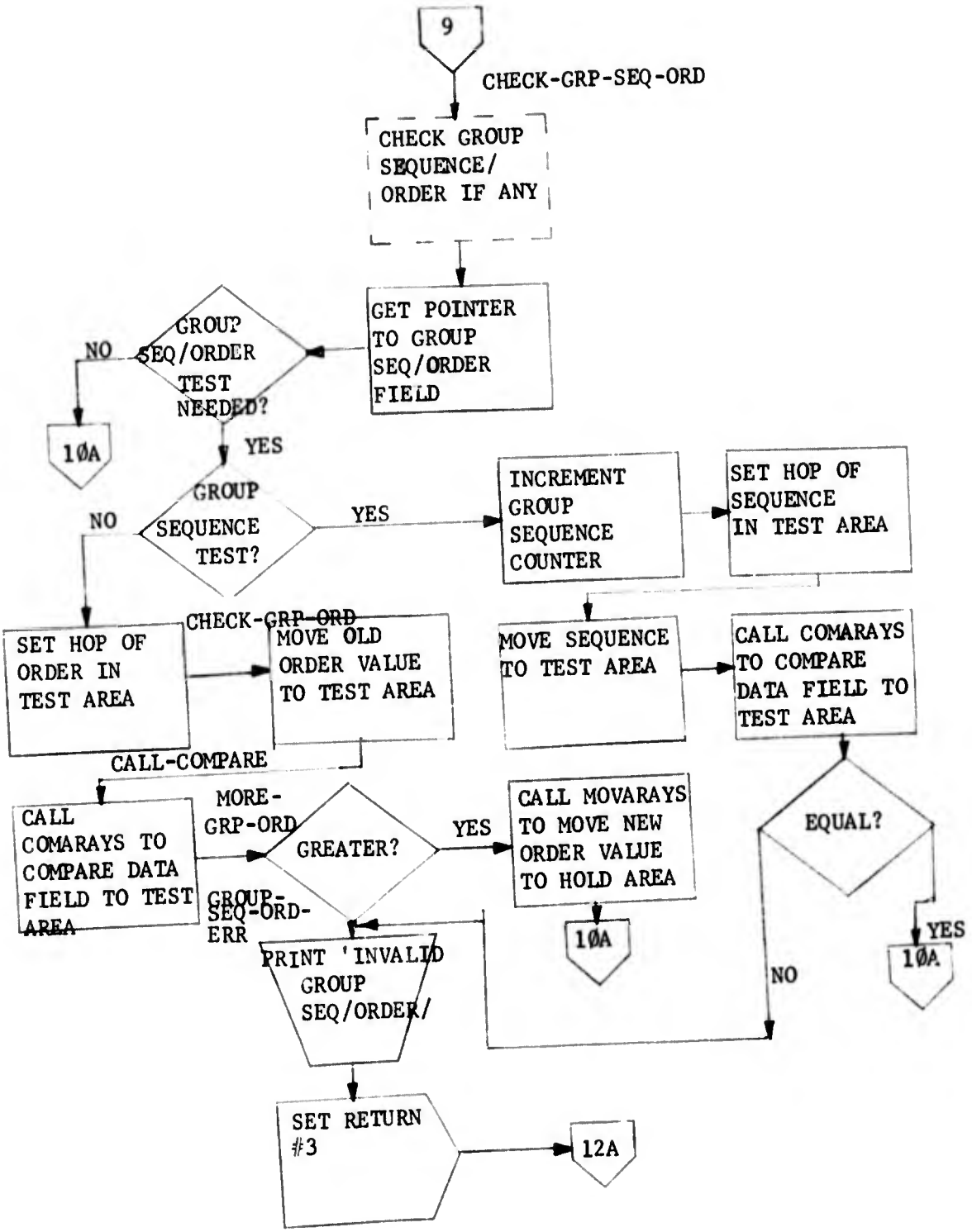

4A
GET-GROUP-FIELD
SET POINTER TO ONE GROUP ID TABLE ENTRY
SET POINTER TO CORRESPONDENCE FIELD-LIST ENTRY
CALL MOVARAYS TO MOVE ONE GROUP ID FIELD TO HOLD AREA
LAST GROUP ID FIELD ?
YES
5A
NO
INCREMENT GROUP ID COUNTER AND POINTER
4A
4B
GET-GRP-FLD
SET POINTER TO ONE MIDMS ID TABLE ENTRY
SET POINTER TO CORRESPONDENCE FIELD-LIST ENTRY
CALL MOVARAYS TO MOVE ONE MIDMS ID FIELD TO GROUP ID HOLD AREA
LAST MIDMS ID FIELD ?
YES
5A
NO
INCREMENT MIDMS ID COUNTER AND POINTER
4B

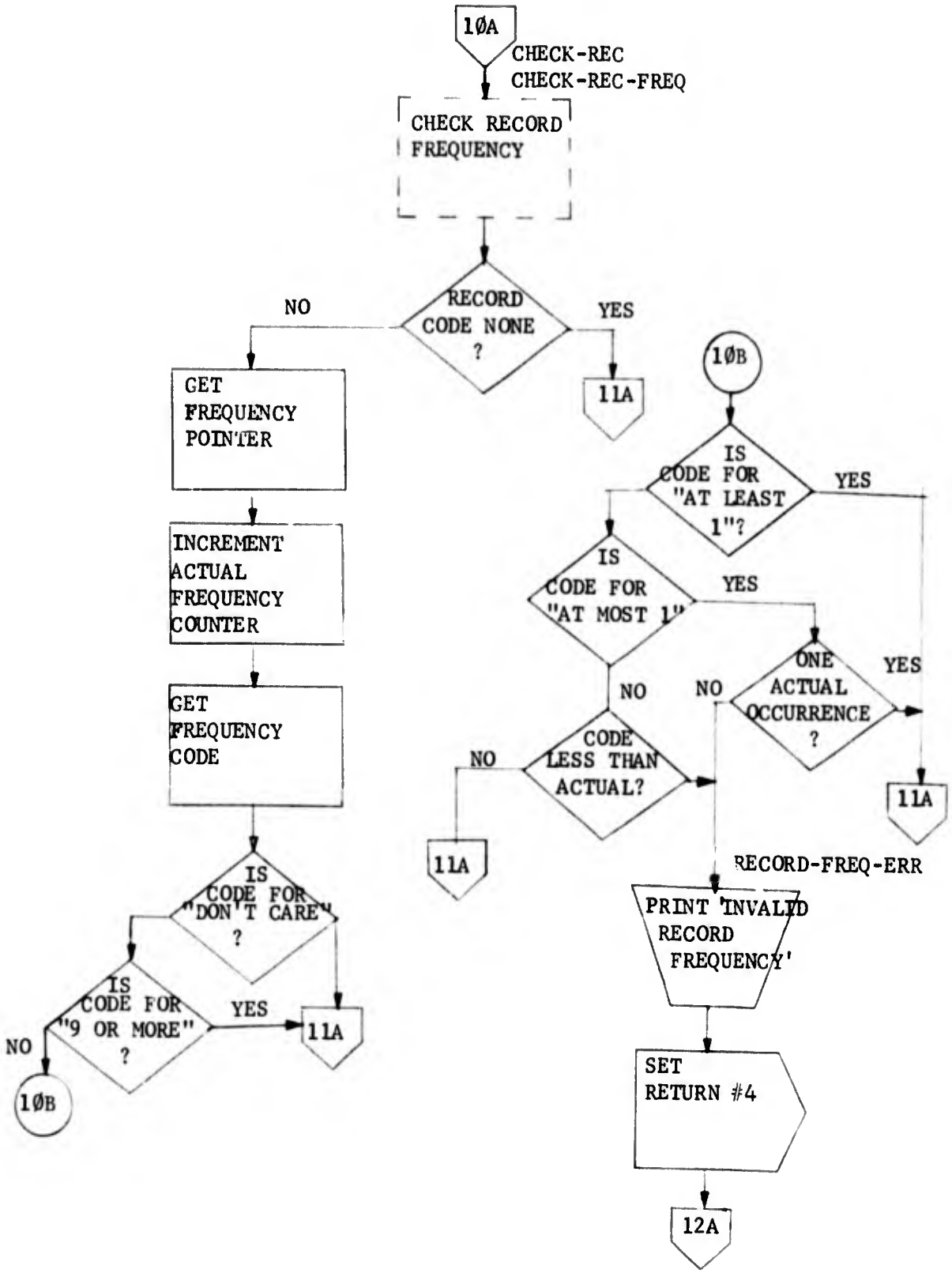

5A
CHECK-GROUP
IS RECORD IN SAME GROUP ?
NO
YES
NEW DATA SOURCE ?
NO
YES
IS GROUP REJECTED ?
NO
YES
8A
2B
CLEAR NEW-GROUP-SW FOR REGULAR PROCESSING
IS GROUP REJECTED ?
NO
YES
ANY TRANSACTIONS IN GROUP WRITTEN ?
NO
YES
SAVE GROUP ID
8A
CLEAR SUBORDINATE GROUP NO. AND RECORD NO. OF TRANS SORT KEY
INDICATE TRANSACTION REJECT RECORD TO BE WRITTEN
CALL IPTRN TO WRITE TRANSACTION REJECT RECORD
5C
CHECK-FOR-END
ANY TRANSACTIONS ?
YES
NO
6A
END OF INPUT DATA ?
NO
YES
1B
7C

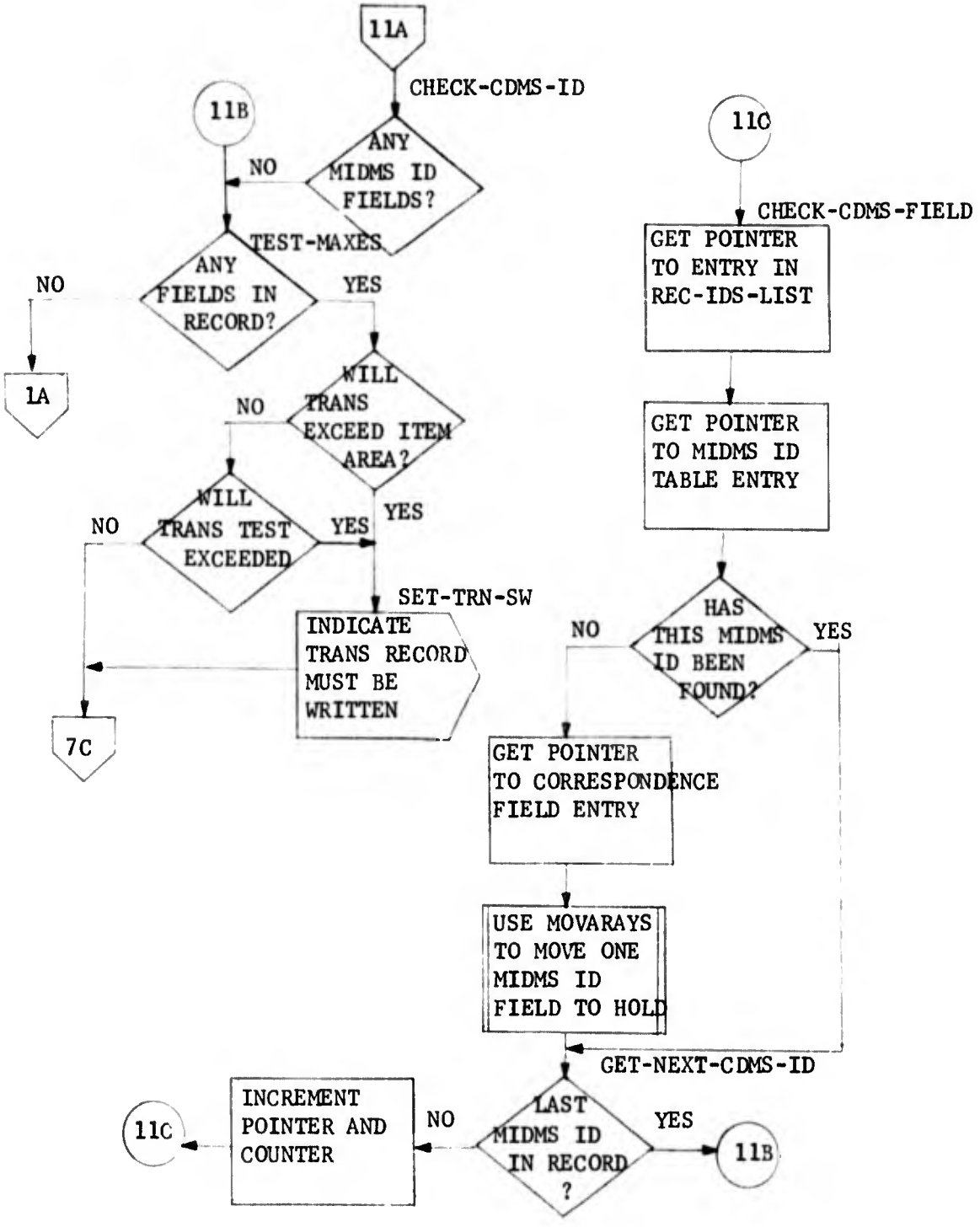

6A
INDICATE END OF GROUP FOUND
RECORD CODE NONE ?
NO
YES
CHECK FREQUENCIES
INITIALIZE FREQUENCY CODE
7C
6B
CHECK-FREQ
GET FREQUENCY CODE
IS CODE FOR 'DON'T CARE' ?
NO
YES
6C
IS CODE FOR "AT MOST 1" ?
NO
YES
6C
IS CODE FOR "AT LEAST 1"
NO
YES
6C
IS CODE GREATER THAN OCCURRENCES ?
NO
YES
ANY OCCURRENCES OF RECORD ?
NO
YES
6C
NEXT-FREQ
LAST FREQUENCY ?
NO
7A
GROUP-FREQ-ERR
PRINT 'INVALID GROUP FREQUENCY'
SET RETURN S1
INCREMENT FREQUENCY COUNTER
6B
IS ERROR CODE "R"
NO
YES
12A
12C

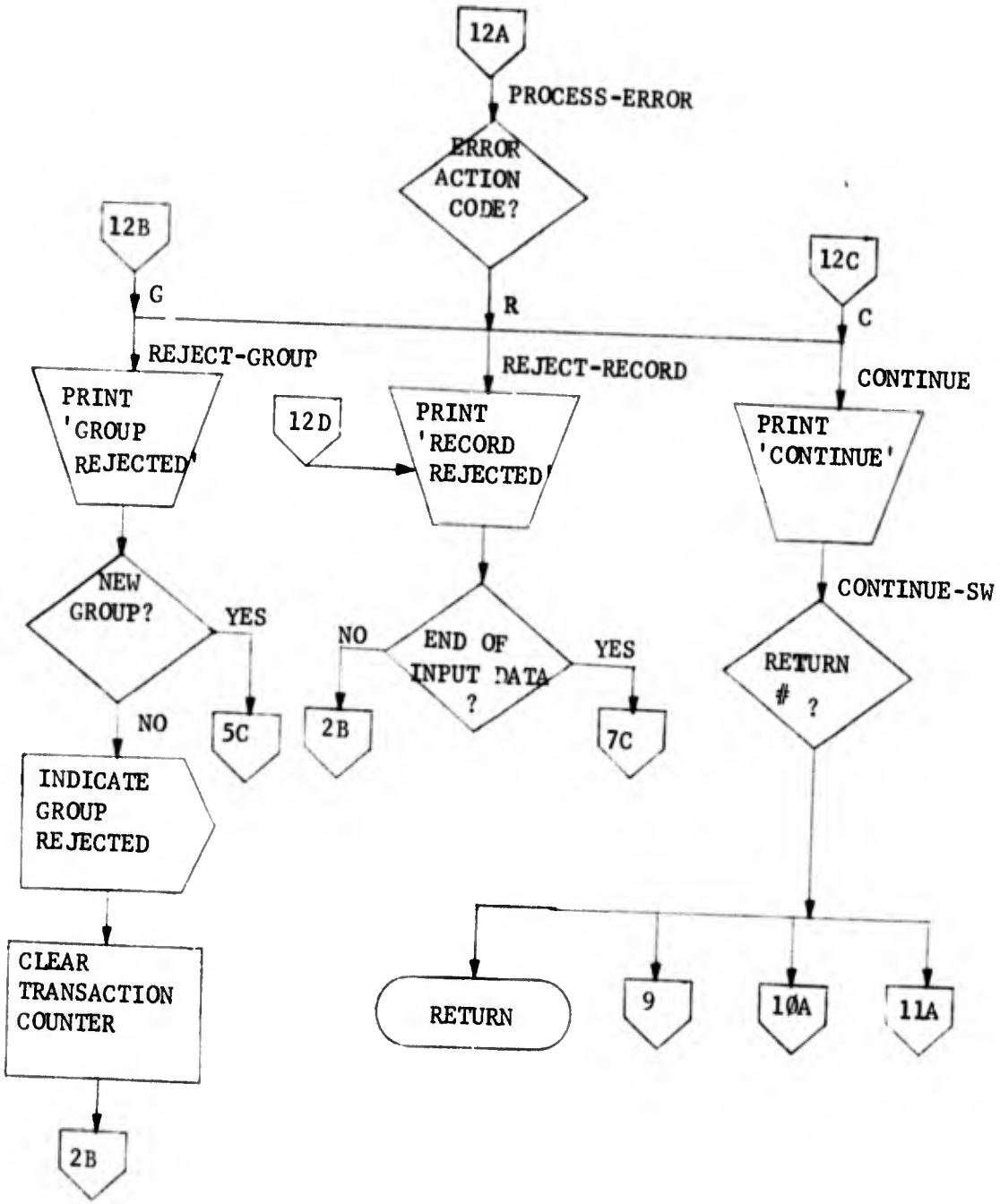

7A
CHECK-ID-START
INITIALIZE MIDMS ID POINTER
CHECK-ID
7B
HAS MIDMS ID FIELD BEEN FOUND ?
NO
YES
PRINT 'RECORD ID NOT COMPLETE' MESSAGE
12B
LAST MIDMS ID FIELD ?
NO
INCREMENT MIDMS ID POINTER
7B
7C
FINISH-IPREC
SAVE POINTER TO RECORD ENTRY
RETURN


8A
PROCESS-RECORD
CHECK FILE SEQUENCE/ ORDER IF ANY
GET POINTER TO FILE SEQUENCE/ ORDER FIELD
FILE SEQUENCE/ ORDER TEST NEEDED?
9
CHECK-FIL-ORD
CHECK-FILE-SEQ
FILE SEQUENCE TEST ?
NO
YES
MOVE OLD ORDER VALUE TO TEST AREA
INCREMENT FILE SEQUENCE COUNTER
SET HOP OF ORDER IN TEST AREA
CALL COMARAYS TO COMPARE DATA FIELD TO TEST AREA
SET HOP OF SEQUENCE ORDER TEST
MOVE SEQUENCE TO TEST AREA
GREATER ?
YES
CALL MOVARAYS TO MOVE NEW ORDER VALUE TO HOLD AREA
NO
8B
9
CALL COMARAYS TO COMPARE DATA FIELD TO TEST AREA
8B
FILE-SEQ-ORD-ERR
PRINT 'INVALID FILE SEQUENCE/ ORDER
NO
EQUAL ?
YES
SET RETURN #2
12A
9


9
CHECK-GRP-SEQ-ORD
CHECK GROUP SEQUENCE/ ORDER IF ANY
GET POINTER TO GROUP SEQ/ORDER FIELD
GROUP SEQ/ORDER TEST NEEDED?
NO
YES
10A
GROUP SEQUENCE TEST?
NO
YES
INCREMENT GROUP SEQUENCE COUNTER
SET HOP OF SEQUENCE IN TEST AREA
SET HOP OF ORDER IN TEST AREA
CHECK-GRP-ORD
MOVE OLD ORDER VALUE TO TEST AREA
MOVE SEQUENCE TO TEST AREA
CALL COMARAYS TO COMPARE DATA FIELD TO TEST AREA
CALL-COMPARE
CALL COMARAYS TO COMPARE DATA FIELD TO TEST AREA
MORE-GRP-ORD
GREATER?
YES
CALL MOVARAYS TO MOVE NEW ORDER VALUE TO HOLD AREA
EQUAL?
YES
NO
10A
10A
GROUP-SEQ-ORD-ERR
PRINT 'INVALID GROUP SEQ/ORDER/
SET RETURN #3
12A


10A
CHECK-REC
CHECK-REC-FREQ
CHECK RECORD FREQUENCY
RECORD CODE NONE ?
NO
YES
11A
GET FREQUENCY POINTER
INCREMENT ACTUAL FREQUENCY COUNTER
GET FREQUENCY CODE
IS CODE FOR "DON'T CARE" ?
IS CODE FOR "9 OR MORE" ?
YES
NO
11A
10B
10B
IS CODE FOR "AT LEAST 1"?
YES
IS CODE FOR "AT MOST 1"
YES
NO
ONE ACTUAL OCCURRENCE ?
YES
NO
CODE LESS THAN ACTUAL?
NO
11A
11A
RECORD-FREQ-ERR
PRINT 'INVALID RECORD FREQUENCY'
SET RETURN #4
12A


11A
CHECK-CDMS-ID
11B
ANY MIDMS ID FIELDS?
NO
TEST-MAXES
ANY FIELDS IN RECORD?
NO
YES
1A
WILL TRANS EXCEED ITEM AREA?
NO
YES
WILL TRANS TEST EXCEEDED
NO
YES
SET-TRN-SW
INDICATE TRANS RECORD MUST BE WRITTEN
7C
11C
CHECK-CDMS-FIELD
GET POINTER TO ENTRY IN REC-IDS-LIST
GET POINTER TO MIDMS ID TABLE ENTRY
HAS THIS MIDMS ID BEEN FOUND?
NO
YES
GET POINTER TO CORRESPONDENCE FIELD ENTRY
USE MOVARAYS TO MOVE ONE MIDMS ID FIELD TO HOLD
GET-NEXT-CDMS-ID
LAST MIDMS ID IN RECORD ?
NO
YES
INCREMENT POINTER AND COUNTER
11C
11B


12A
PROCESS-ERROR
ERROR ACTION CODE?
12B
G
REJECT-GROUP
PRINT 'GROUP REJECTED'
NEW GROUP?
YES
5C
NO
INDICATE GROUP REJECTED
CLEAR TRANSACTION COUNTER
2B
R
REJECT-RECORD
12D
PRINT 'RECORD REJECTED'
NO
END OF INPUT DATA ?
YES
2B
7C
12C
C
CONTINUE
PRINT 'CONTINUE'
CONTINUE-SW
RETURN # ?
RETURN
9
10A
11A

d. FMIPUN

(1) Function. To preprocess Unit Update records for validation by FMIPVAL.

(2) Calling Sequence.

```
ENTRY 'IPUN'       USING FM-DD IP-DD
      MAJOR-LR2            MAJOR-LR11
      REC-TYPE-LIST        FIELD-LIST
      VAL-LIST             VAL-LIT-AREA
      INPUT-RECORD         TAPE-DATA.
CALL  'FMCRD'      USING CARD-DATA END-SW.
CALL  'FMPRT'      USING PRT-LINE CC.
CALL  'IPBIN'      USING FM-DD IP-DD MAJOR-LR11.
CALL  'MOVARAYS'   USING ... *
CALL  'MOVALF'     USING UNIT-REC-ID ONE IP-CNT
      GRP-ID-HOLD ONE THIRTY.
```

* Several calls are made to the MOVARAYS subroutine.

(3) Capabilities.

The FMIPUN subprogram handles the preliminary processing of Unit Update records and builds a special collection of DST tables for each field to be updated. In addition, it contains many of the functions of FMIPREC and sets the same control fields and switches required by FMIPX and FMIPVAL. For this reason, several of the paragraphs in FMIPUN have the same names as the FMIPREC paragraphs performing the same function.

While the overall processing of FMIPUN paragraphs START-GROUP to TEST-GROUP is similar to those paragraphs in FMIPREC, some processing variations are used to reflect the Unit Update emphasis. The main variations are the use of the field name as a record type code and the record ID portion of the Unit Update record as a group ID as well as a MIDMS ID field.

The other FMIPUN paragraphs are described below.

PROCESS-RECORD initializes FMIPUN control fields and separates the processing of fixed and record control updates from periodic and variable field updates.

MOVE-FIXED-INFO clears FLD-PSS-HOP and FLD-PSS-LEN for fixed fields.

TEST-FOR-3 sets up control information for periodic and variable field updates.

PROCESS-FIELD sets up control information for the DEL operation.

PROCESS-FIELD-LEN determines the length of the update data field by looking for a record mark character (0-2-8 punch).

MOVE-FIRST sets up control information for the first record (non-continuation) of a field's update.

MOVE-RECORD moves information from the Unit Update card image to INPUT-RECORD. For continuation cards, only actual data characters are moved. Checking for continuations is also performed in this paragraph.

CHECK-SUB determines if a conversion subroutine is required for the field. If so, entries are placed in the special DST to tell FMIPVAL to convert the field.

CHECK-SIZE insures that the amount of data supplied does not exceed the field length specified in the FFT.

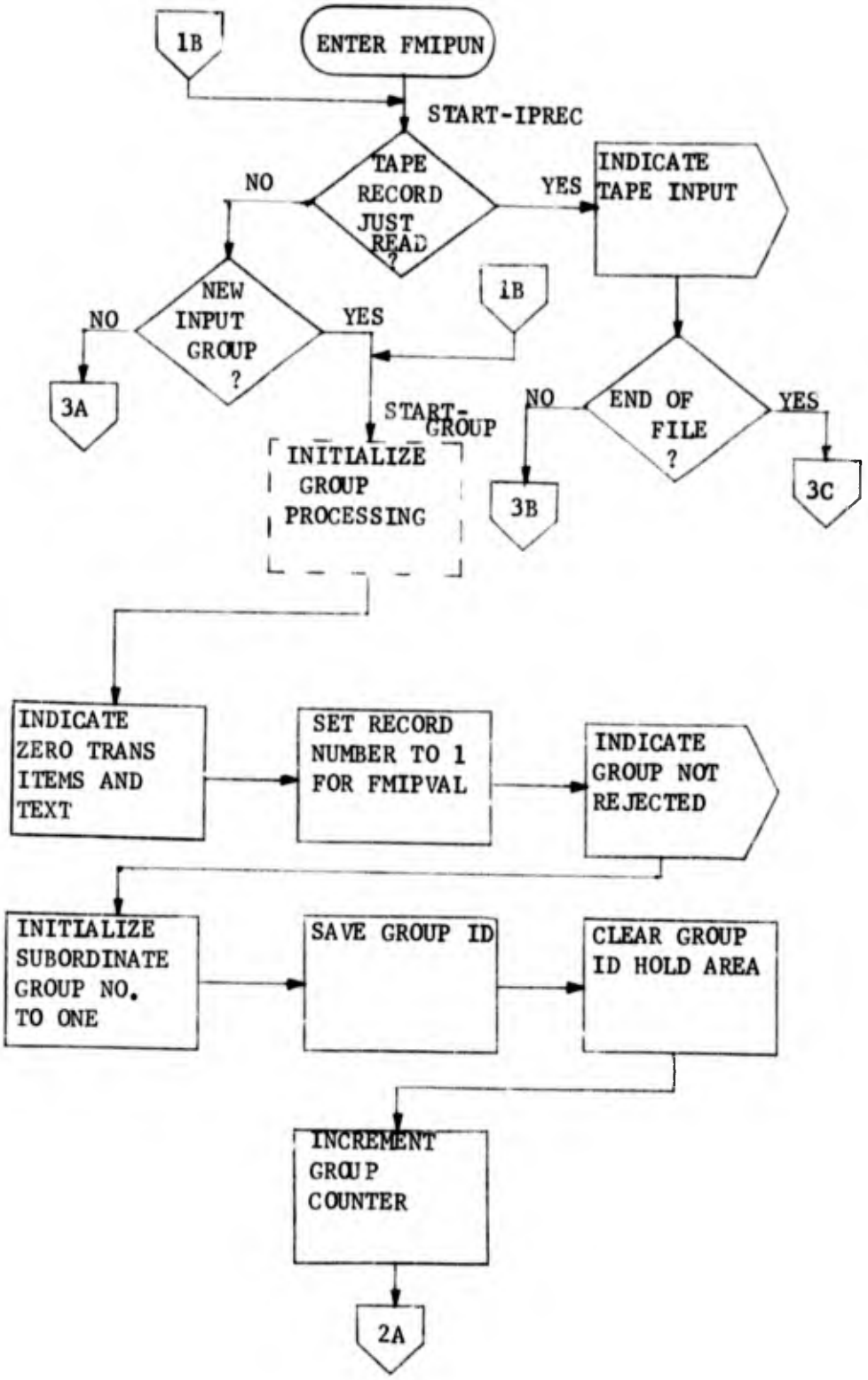
NO-CONTINUATION prints an error message when an end of file condition occurs after a continuation is specified.

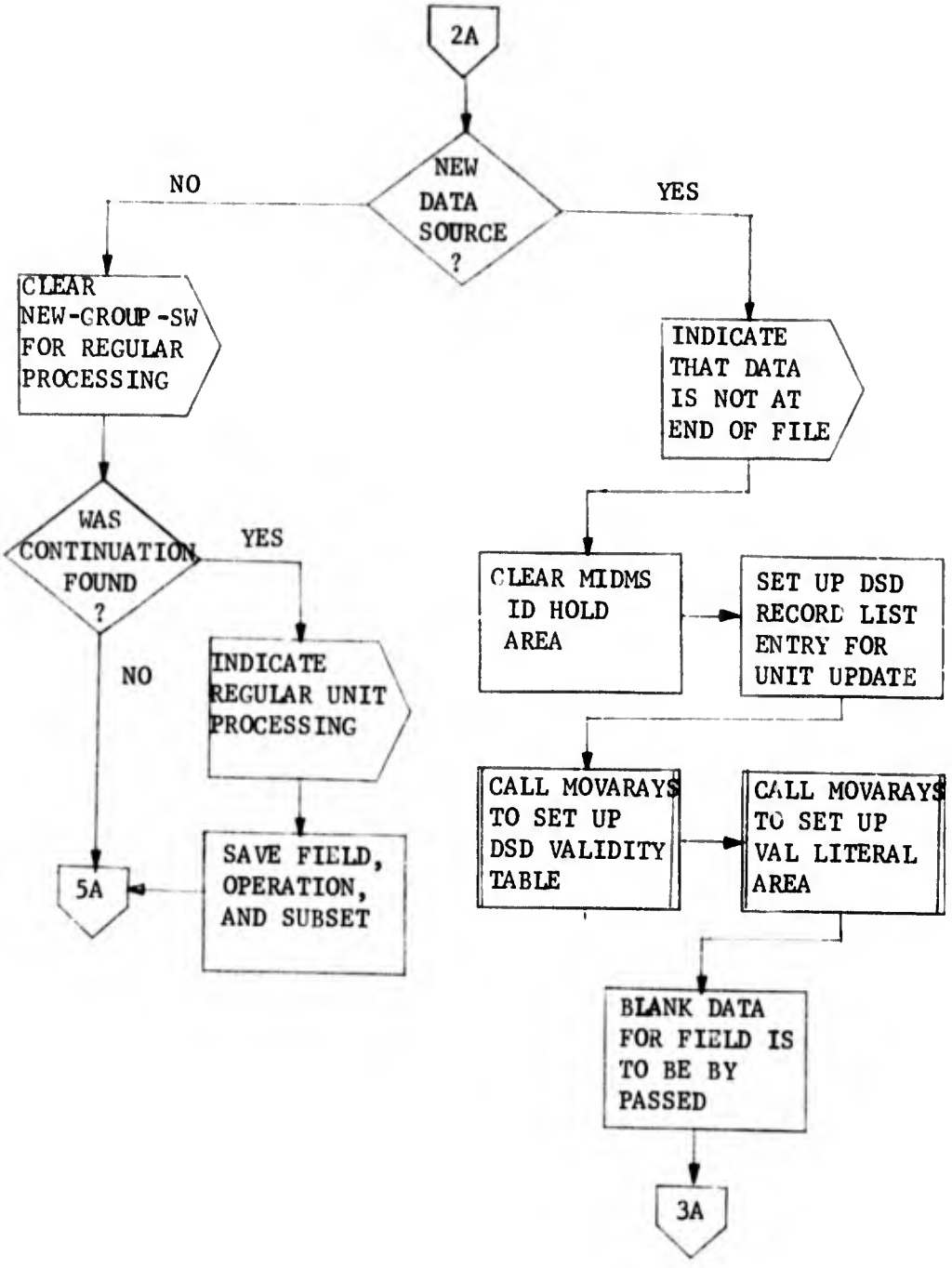
CHECK-CONTINUATION determines that an expected continuation actually occurs.

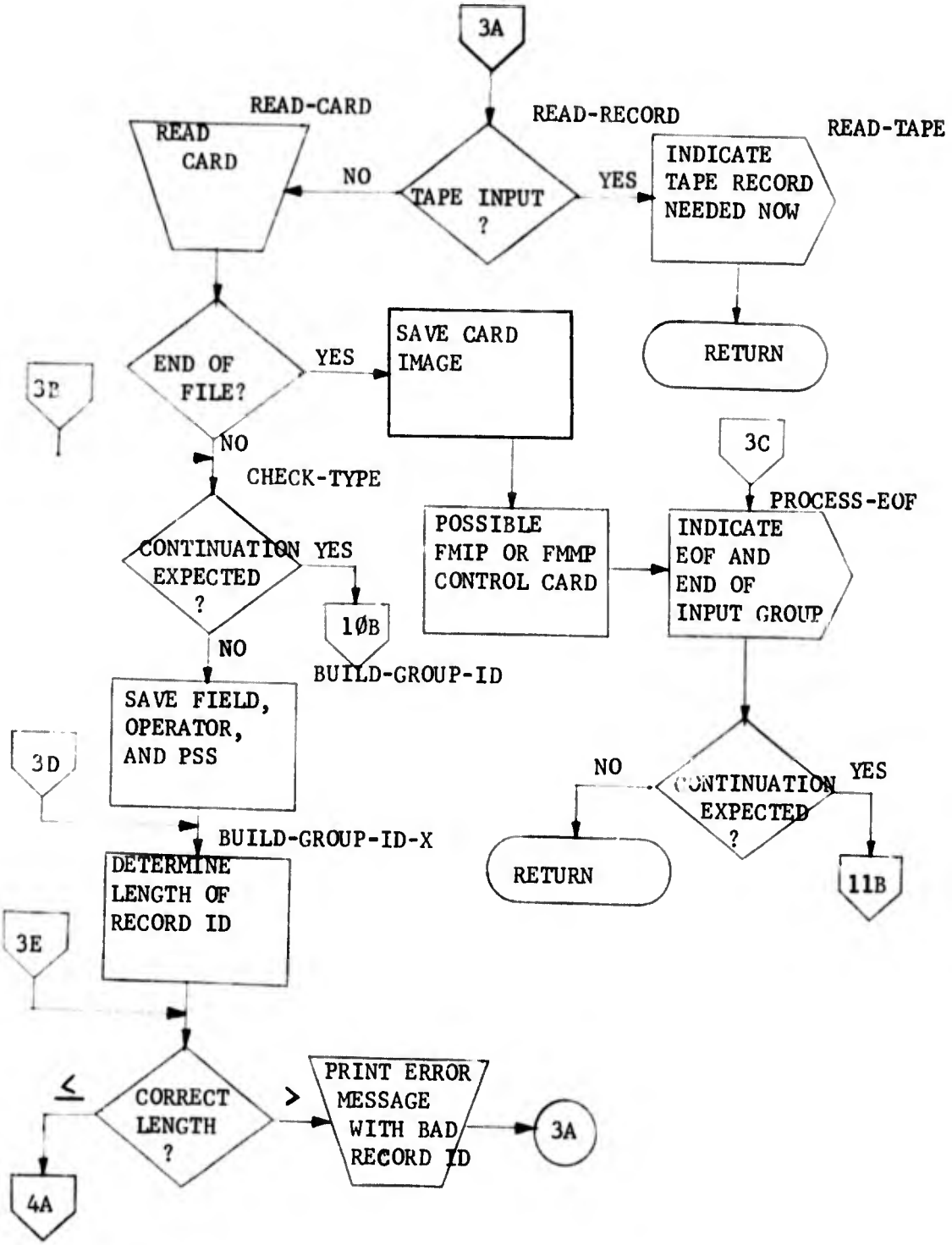
CHECK-ID determines that a possible continuation contains the same record ID as the previous record.

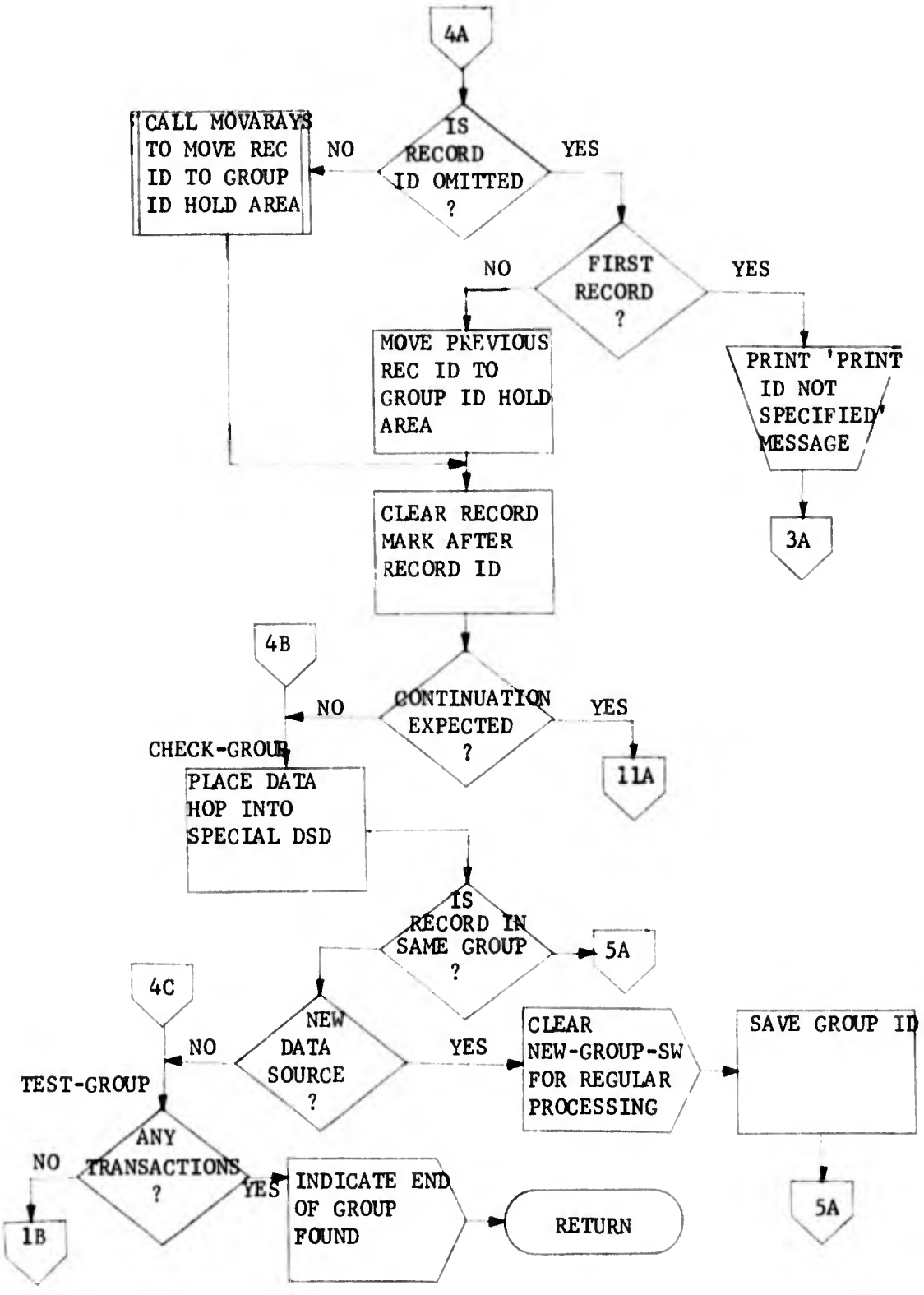
FINISH-IPUNIT terminates the processing of the FMIPUN subroutine.

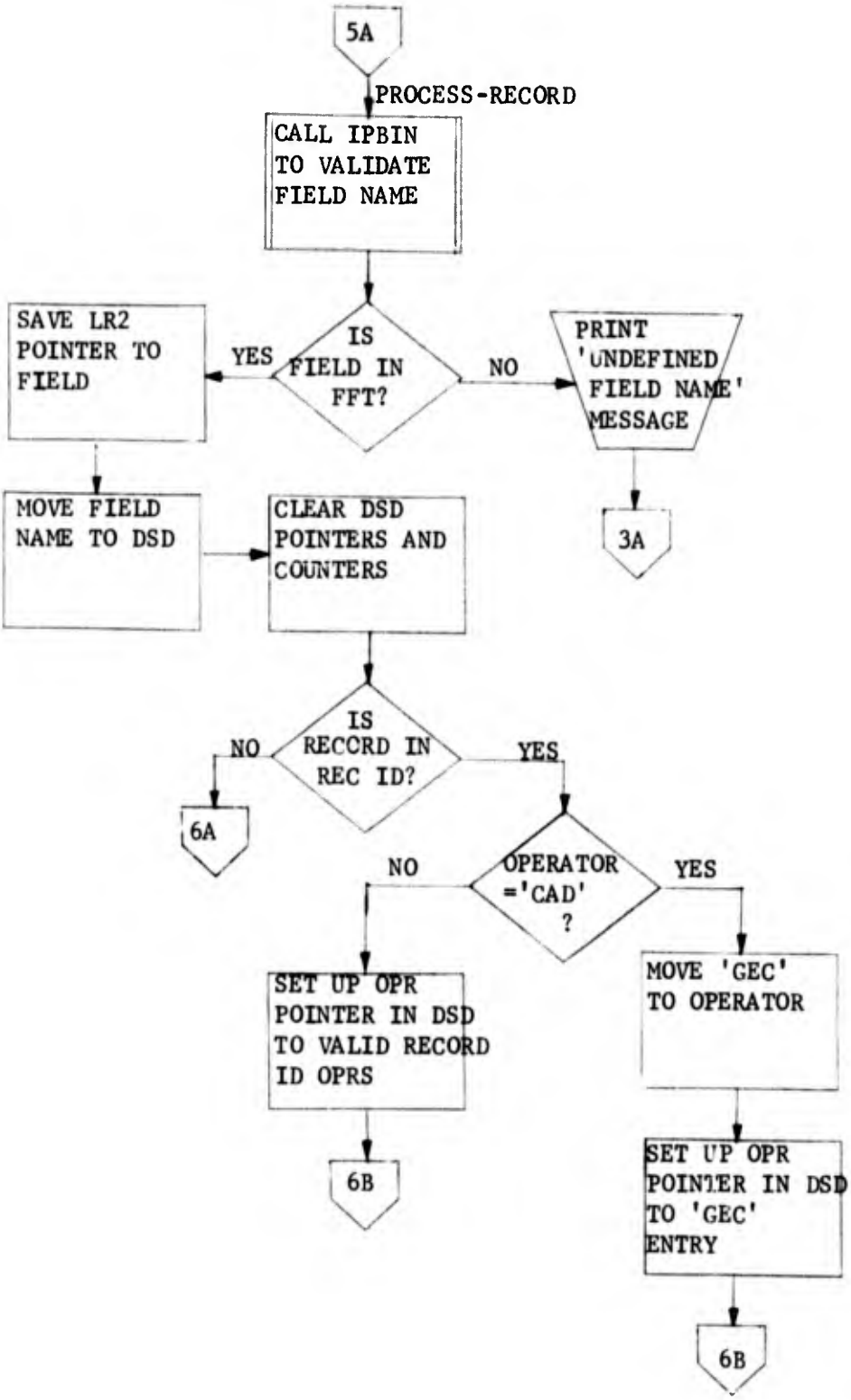
(4) Flowchart.

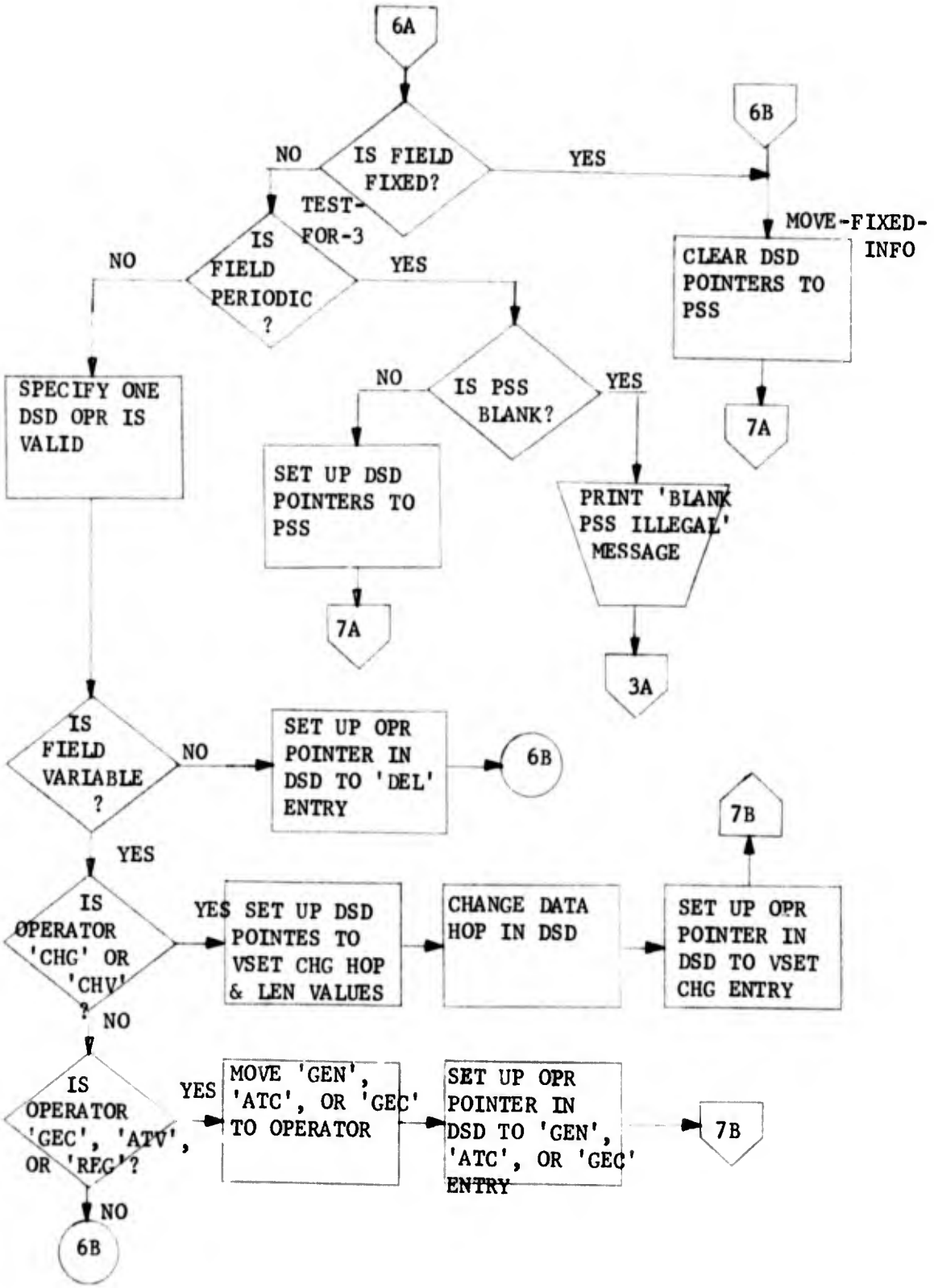

1B
ENTER FMIPUN
START-IPREC
TAPE RECORD JUST READ ?
NO
YES
INDICATE TAPE INPUT
NEW INPUT GROUP ?
NO
YES
1B
3A
START-GROUP
INITIALIZE GROUP PROCESSING
END OF FILE ?
NO
YES
3B
3C
INDICATE ZERO TRANS ITEMS AND TEXT
SET RECORD NUMBER TO 1 FOR FMIPVAL
INDICATE GROUP NOT REJECTED
INITIALIZE SUBORDINATE GROUP NO. TO ONE
SAVE GROUP ID
CLEAR GROUP ID HOLD AREA
INCREMENT GROUP COUNTER
2A

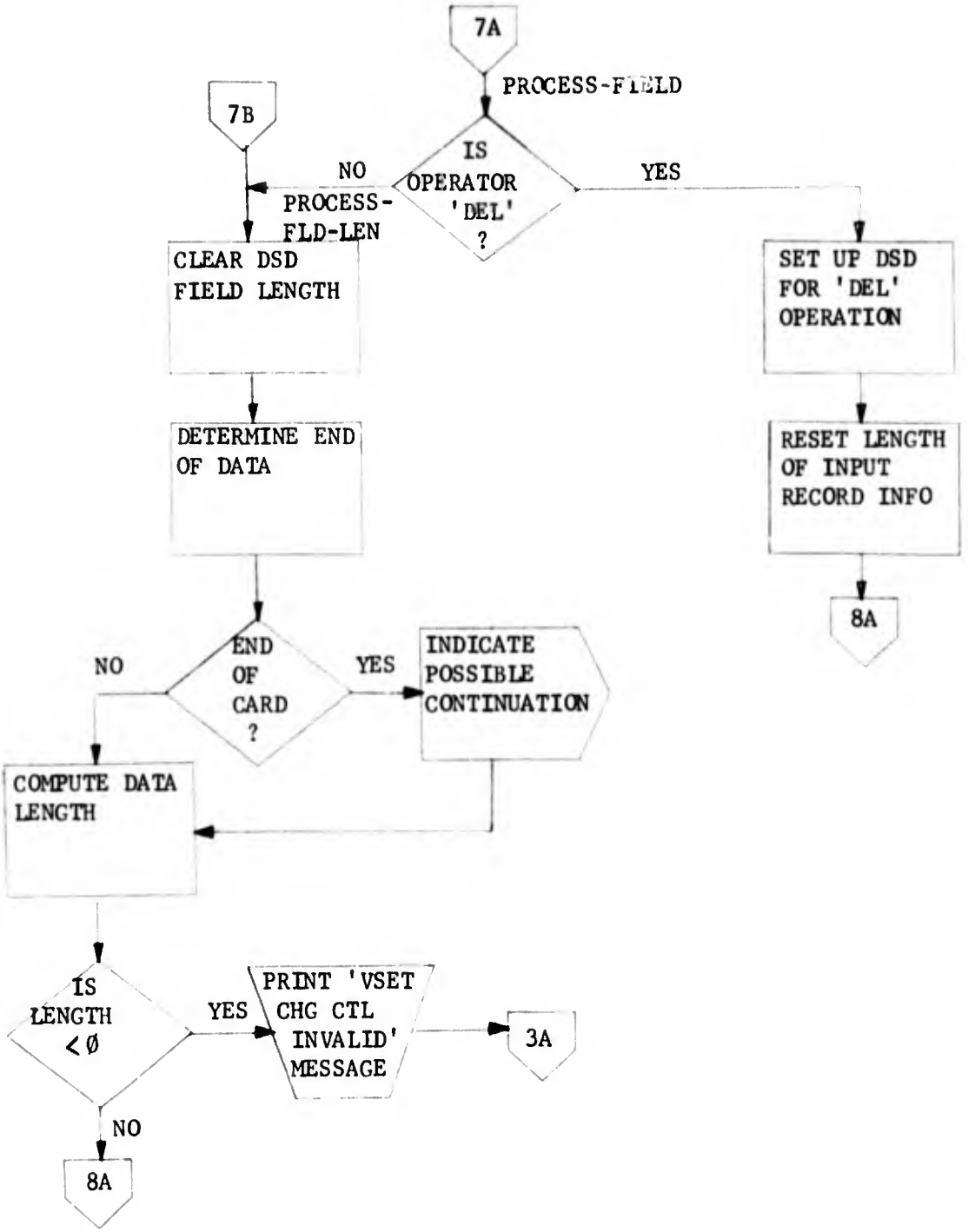

2A
NEW DATA SOURCE ?
NO
YES
CLEAR NEW-GROUP-SW FOR REGULAR PROCESSING
INDICATE THAT DATA IS NOT AT END OF FILE
WAS CONTINUATION FOUND ?
YES
NO
INDICATE REGULAR UNIT PROCESSING
CLEAR MIDMS ID HOLD AREA
SET UP DSD RECORD LIST ENTRY FOR UNIT UPDATE
CALL MOVARAYS TO SET UP DSD VALIDITY TABLE
CALL MOVARAYS TO SET UP VAL LITERAL AREA
SAVE FIELD, OPERATION, AND SUBSET
5A
BLANK DATA FOR FIELD IS TO BE BY PASSED
3A

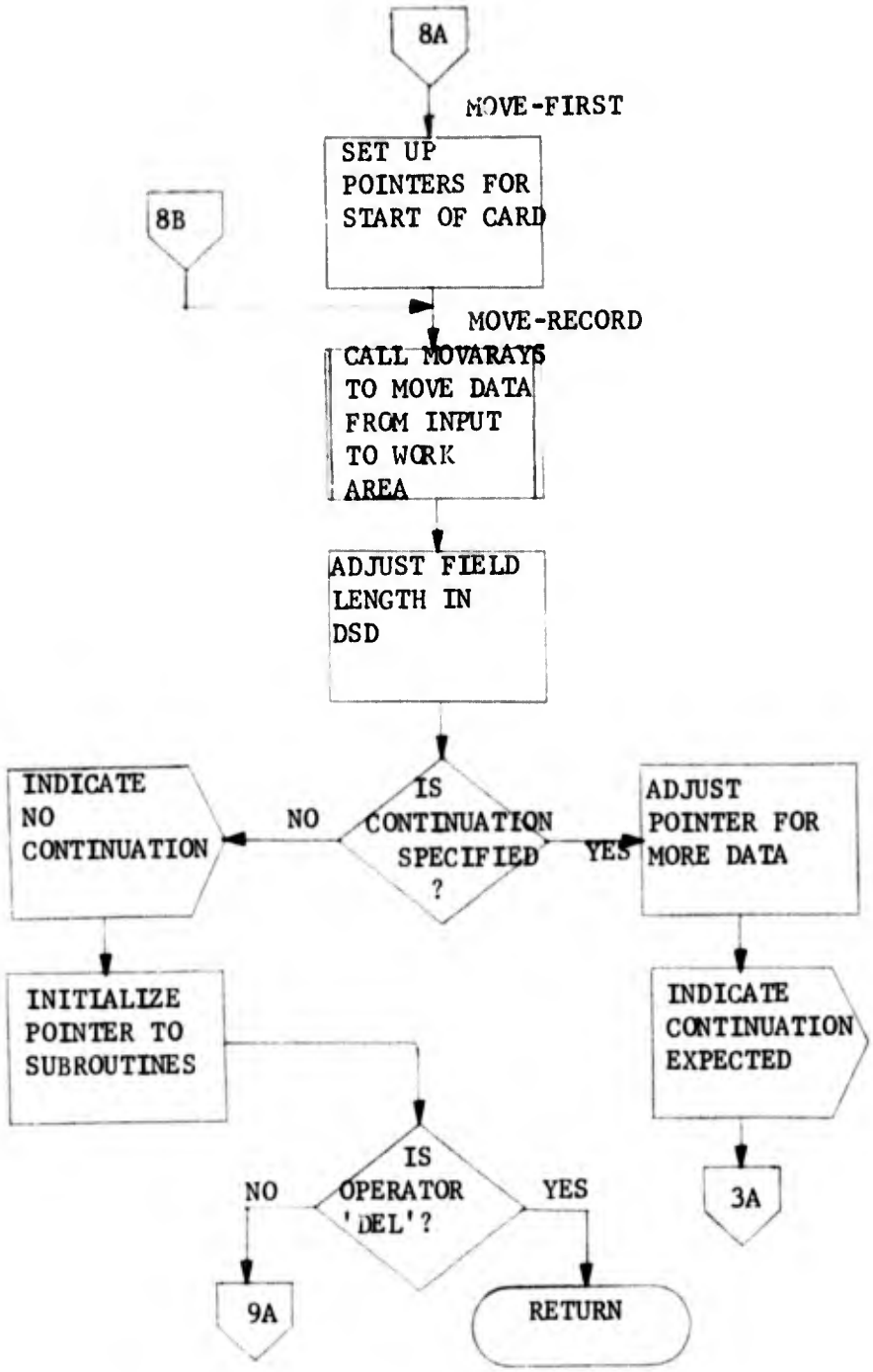

3A
READ-CARD
READ-RECORD
READ-TAPE
READ CARD
NO
TAPE INPUT ?
YES
INDICATE TAPE RECORD NEEDED NOW
RETURN
3B
END OF FILE?
YES
SAVE CARD IMAGE
NO
CHECK-TYPE
3C
PROCESS-EOF
CONTINUATION EXPECTED ?
YES
10B
BUILD-GROUP-ID
POSSIBLE FMIP OR FMMP CONTROL CARD
INDICATE EOF AND END OF INPUT GROUP
NO
SAVE FIELD, OPERATOR, AND PSS
3D
NO
CONTINUATION EXPECTED ?
YES
BUILD-GROUP-ID-X
RETURN
11B
DETERMINE LENGTH OF RECORD ID
3E
≤
CORRECT LENGTH ?
>
PRINT ERROR MESSAGE WITH BAD RECORD ID
3A
4A

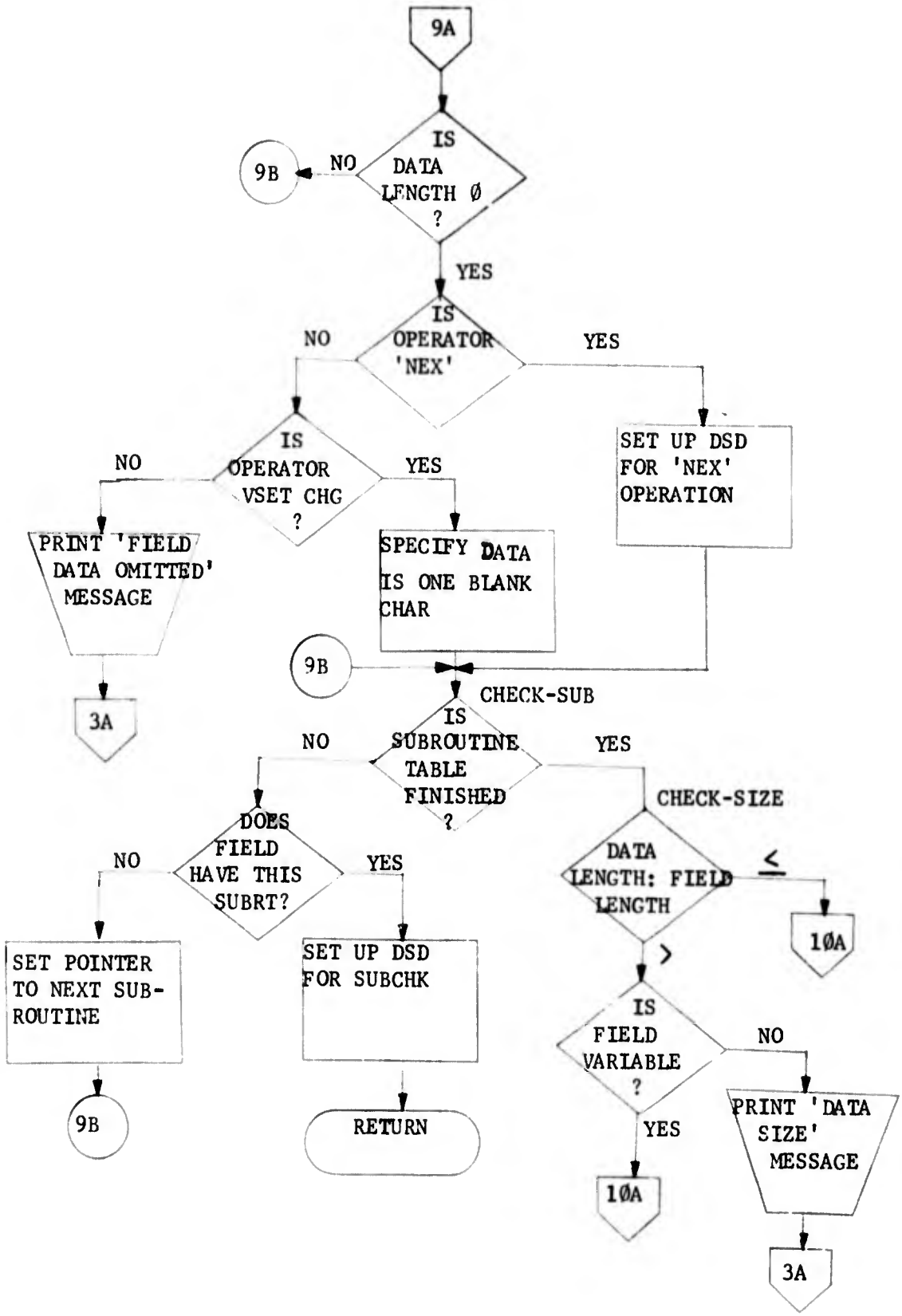

4A
IS RECORD ID OMITTED ?
NO
YES
CALL MOVARAYS TO MOVE REC ID TO GROUP ID HOLD AREA
FIRST RECORD ?
NO
YES
MOVE PREVIOUS REC ID TO GROUP ID HOLD AREA
PRINT 'PRINT ID NOT SPECIFIED' MESSAGE
3A
CLEAR RECORD MARK AFTER RECORD ID
4B
CONTINUATION EXPECTED ?
NO
YES
11A
CHECK-GROUP
PLACE DATA HOP INTO SPECIAL DSD
IS RECORD IN SAME GROUP ?
5A
4C
NEW DATA SOURCE ?
NO
YES
CLEAR NEW-GROUP-SW FOR REGULAR PROCESSING
SAVE GROUP ID
5A
TEST-GROUP
ANY TRANSACTIONS ?
NO
YES
1B
INDICATE END OF GROUP FOUND
RETURN

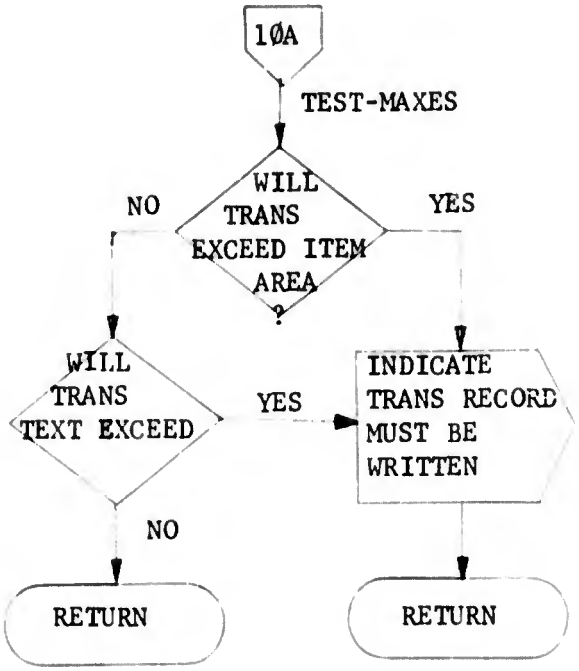

5A
PROCESS-RECORD
CALL IPBIN TO VALIDATE FIELD NAME
IS FIELD IN FFT?
YES
NO
SAVE LR2 POINTER TO FIELD
PRINT 'UNDEFINED FIELD NAME' MESSAGE
3A
MOVE FIELD NAME TO DSD
CLEAR DSD POINTERS AND COUNTERS
IS RECORD IN REC ID?
NO
YES
6A
OPERATOR ='CAD' ?
NO
YES
SET UP OPR POINTER IN DSD TO VALID RECORD ID OPRS
MOVE 'GEC' TO OPERATOR
6B
SET UP OPR POINTER IN DSD TO 'GEC' ENTRY
6B

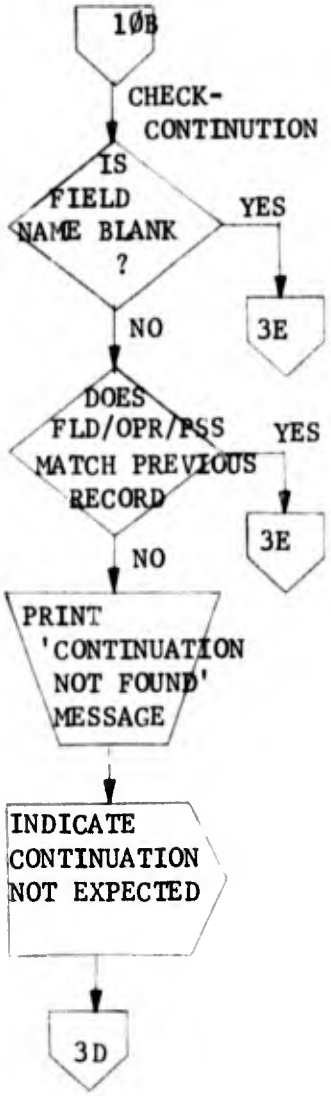

6A
IS FIELD FIXED?
NO
YES
6B
MOVE-FIXED-INFO
CLEAR DSD POINTERS TO PSS
7A
TEST-FOR-3
IS FIELD PERIODIC ?
NO
YES
SPECIFY ONE DSD OPR IS VALID
IS PSS BLANK?
NO
YES
SET UP DSD POINTERS TO PSS
7A
PRINT 'BLANK PSS ILLEGAL' MESSAGE
3A
IS FIELD VARIABLE ?
NO
SET UP OPR POINTER IN DSD TO 'DEL' ENTRY
6B
YES
7B
IS OPERATOR 'CHG' OR 'CHV' ?
YES
SET UP DSD POINTES TO VSET CHG HOP & LEN VALUES
CHANGE DATA HOP IN DSD
SET UP OPR POINTER IN DSD TO VSET CHG ENTRY
NO
IS OPERATOR 'GEC', 'ATV', OR 'REG'?
YES
MOVE 'GEN', 'ATC', OR 'GEC' TO OPERATOR
SET UP OPR POINTER IN DSD TO 'GEN', 'ATC', OR 'GEC' ENTRY
7B
NO
6B

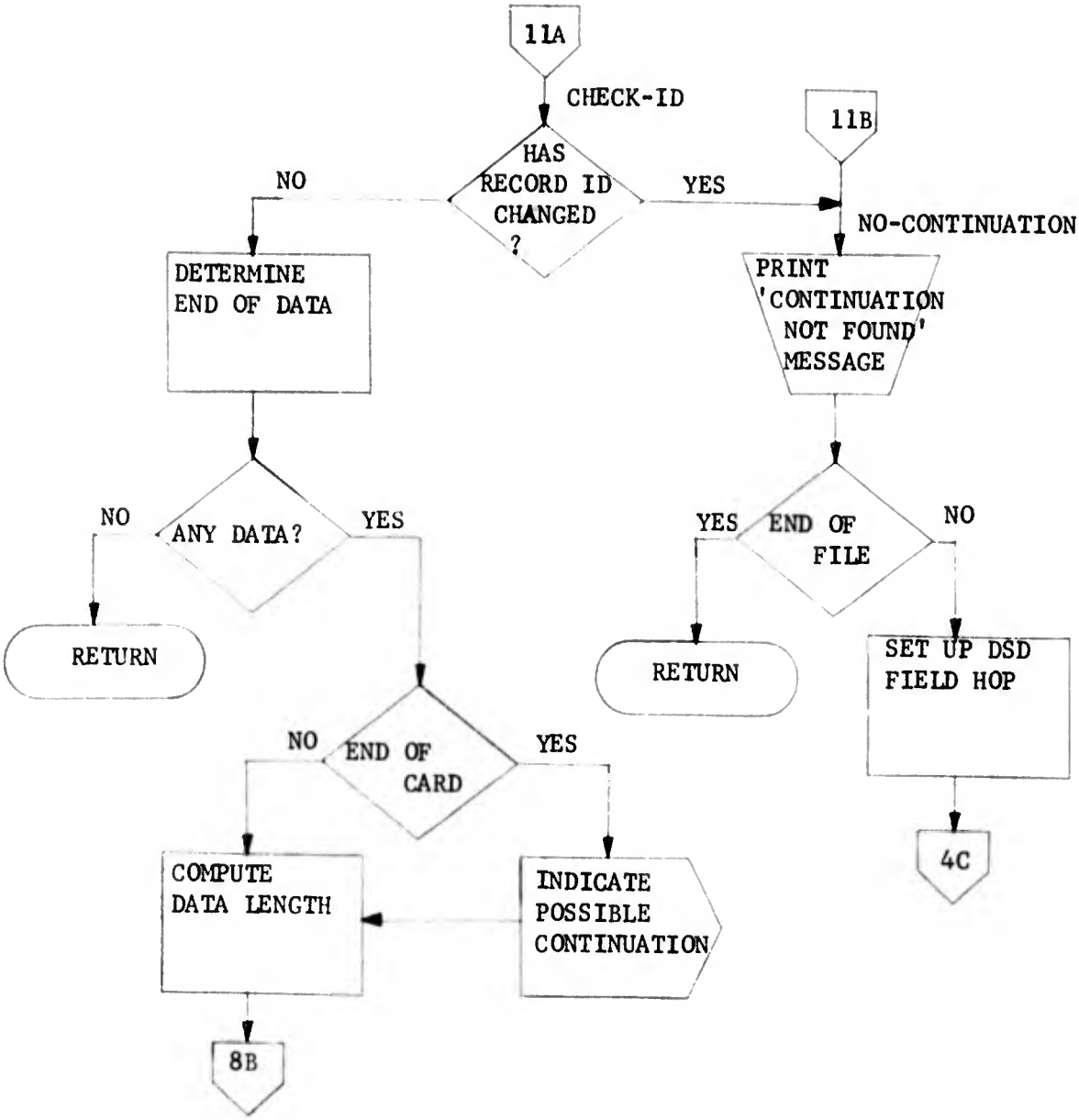

7A
PROCESS-FIELD
7B
IS OPERATOR 'DEL' ?
NO
YES
PROCESS-FLD-LEN
CLEAR DSD FIELD LENGTH
SET UP DSD FOR 'DEL' OPERATION
DETERMINE END OF DATA
RESET LENGTH OF INPUT RECORD INFO
8A
END OF CARD ?
NO
YES
INDICATE POSSIBLE CONTINUATION
COMPUTE DATA LENGTH
IS LENGTH < Ø
YES
PRINT 'VSET CHG CTL INVALID' MESSAGE
3A
NO
8A


8A
MOVE-FIRST
SET UP POINTERS FOR START OF CARD
8B
MOVE-RECORD
CALL MOVARAYS TO MOVE DATA FROM INPUT TO WORK AREA
ADJUST FIELD LENGTH IN DSD
INDICATE NO CONTINUATION
NO
IS CONTINUATION SPECIFIED ?
YES
ADJUST POINTER FOR MORE DATA
INITIALIZE POINTER TO SUBROUTINES
INDICATE CONTINUATION EXPECTED
IS OPERATOR 'DEL'?
NO
YES
3A
9A
RETURN


9A
IS DATA LENGTH Ø ?
NO
9B
YES
IS OPERATOR 'NEX'
NO
YES
SET UP DSD FOR 'NEX' OPERATION
IS OPERATOR VSET CHG ?
NO
YES
PRINT 'FIELD DATA OMITTED' MESSAGE
SPECIFY DATA IS ONE BLANK CHAR
3A
9B
CHECK-SUB
IS SUBROUTINE TABLE FINISHED ?
NO
YES
CHECK-SIZE
DOES FIELD HAVE THIS SUBRT?
NO
YES
DATA LENGTH: FIELD LENGTH
≤
>
1ØA
SET POINTER TO NEXT SUB-ROUTINE
SET UP DSD FOR SUBCHK
IS FIELD VARIABLE ?
NO
YES
PRINT 'DATA SIZE' MESSAGE
9B
RETURN
1ØA
3A


1ØA
TEST-MAXES
WILL TRANS EXCEED ITEM AREA ?
NO
YES
WILL TRANS TEXT EXCEED
YES
INDICATE TRANS RECORD MUST BE WRITTEN
NO
RETURN
RETURN


1ØB
CHECK-CONTINUTION
IS FIELD NAME BLANK ?
YES
3E
NO
DOES FLD/OPR/PSS MATCH PREVIOUS RECORD
YES
3E
NO
PRINT 'CONTINUATION NOT FOUND' MESSAGE
INDICATE CONTINUATION NOT EXPECTED
3D


11A
CHECK-ID
HAS RECORD ID CHANGED ?
NO
YES
11B
NO-CONTINUATION
DETERMINE END OF DATA
PRINT 'CONTINUATION NOT FOUND' MESSAGE
NO
ANY DATA?
YES
YES
END OF FILE
NO
RETURN
RETURN
SET UP DSD FIELD HOP
4C
NO
END OF CARD
YES
COMPUTE DATA LENGTH
INDICATE POSSIBLE CONTINUATION
8B

e. FMIPVAL

(1) Function. To validate, convert and format incoming data into transactions via information obtained form the FFT and the DSD for the file to be updated.

(2) Calling Sequence.

```
CALL 'IPVAL' FM-DD              IP-DD
             REC-TYPE-LIST      REC-IDS-LIST
             FIELD-LIST         FIL-OPR-NAME-LIST
             REC-FIL-OPR-LIST   VAL-LIST
             VAL-LIT-AREA       INPUT-RECORD
             UNSORTED-TRANS-TAB TEXT-TAB.
```

FM-DD. Not used in FMIPVAL.
IP-DD. Major elements used are:

REJ-GROUP-SW
TRANS-ITEM-CNT
TEXT-COUNT
REC-POINT
REC-INDEX
SET-ID-2
FIL-REC-HOP
FIL-REC-LEN
VAL-LIT-CNT
FIL-OPR-CT
FIL-OPR-POINT
FIELD-CNT
VAL-CNT

REC-TYPE-LIST. Major elements used are:

REC-FIELD-CT
REC-FIELD-POINT
REC-OPR-CT
REC-OPR-POINT
REC-FIL-OPR

REC-IDS-LIST. Not Used.
FIELD-LIST. All fields within this group are used extensively. They primarily control the processing of the data file fields.

FLD-HOP
FLD-LEN
FLD-PSS-HOP
FLD-PSS-LEN
FLD-LR2-POINT
FLD-VAL-CT

FLD-VAL-POINT
FLD-SKIP
FLD-NAME

FIL-OPR-NAME-LIST. Not Used.
REC-FIL-OPR-LIST. Not Used.
VAL-LIST. Contains pointers to the literals in VAL-LIT-AREA.

VAL-LIT-HOP
VAL-LIT-LEN
VAL-TYPE
VAL-ERR

VAL-LIT-AREA. All literals, constants, and values used in input processing.

INPUT-RECORD. The input record.

UNSORTED-TRANS-TAB. Output area for FMIPVAL containing the basic control information required to produce transactions.

TRANS-SETNO
TRANS-SUBSET
TRANS-FLDNO
TRANS-OPCODE
TRANS-LENGTH
TRANS-HOP

TEXT-TAB. Contains the transaction data to be inserted into fields by Maintenance Proper.

(3) Capabilities.

FMIPVAL has an extensive error checking capability. The input source data may be validated through RANGE, VALUE, PROFILE or any user-written subroutine(s) (SUBCHK) specified in the DSD. FMIPVAL provides the user with an unlimited range of data conversion. These routines may be user written or standard utility subroutines provided with MIDMS.

RANGE.

RANGE specifies numeric (unsigned) limits within which a field in the data source should fall. The RANGE checking function is divided into two functional operations by OMLPVAL. These operations are LOWVALUE and HIGHVALUE. They are also indicated in the program by paragraphs named accordingly.

LOWVALUE and HIGHVALUE are not related in FMIPVAL. These functions are distinct and are controlled entirely by the error action code supplied by OMLP. However, in reality the two operations are connected by an 'AND' (8) error action code, and together accomplish the RANGE checking.

RANGE checking employs the use of one ALC subroutine (COMARAYS). COMARAYS compares two strings of characters logically. The length of both strings are assumed equal in length.

VALUE.

The most absolute form of validity checking is the VALUE. FMIPVAL provides three routines/methods for checking values. These methods are internally controlled by FMIPVAL and OMLPVAL. These routines will automatically control the process. These three routines/methods are used to provide speed in the input processing. "Normal" value checking is provided by two ALC routines -- COMARAYS and COMLIST; however, if the field is only one character long the validation is done in COBOL (FMIPVAL paragraph VALUE-S).

Before the COMARAYS subroutine compares the data field with a value, another subroutine, MOVRAY, isolates the control information used to determine the location in the input record to be compared. If the control information from VAL-LIT-AREA is equal to zero, the entire field is compared. Otherwise, a partial field is implied and the location information represents the leftmost position in the input record (INPUT-RECORD), relative to 1, to be compared to the value supplied. The number of characters to compare is taken from the length of the literal in VAL-LIT-AREA (VAL-LIT-LEN (II)) minus 4.

For each value that is compared in the value list a call to the subroutine COMARAYS is made and the error action code is checked.

The Special Value processing approach (paragraph SPECIAL-VALUE) provides a more sophisticated method to check values. This approach provides a list of values (e.g. FLD1A VALUE A, C, D, 1, 2). By definition A, C, D, 1, and 2 are implied 'OR' conditions. Therefore, all of the values

(whole field values) may be checked during one CALL of the subroutine COMLIST until a match is found or the list is exhausted. This process eliminates repetitive CALLS, error action code checking, field value isolation and storage requirement. The formats of VAL-LIT-AREA for VALUE checking are:

Regular Format:

NNNN-VALUE1,NNNN-VALUE2,NNNN-VALUE3

where NNNN may be zero for full-field specification and non-zero for partials.

SPECIAL-VALUE Format:

OONN-VALUE1,VALUE2,VALUE3

where OONN is the number of items (maximum value 99).

PROFILE.

PROFILE is the most powerful of the validity checking routines. It validates the character types which should make up a field in the input source data. A field may be any size; however, a maximum number of characters is imposed by LP. FMIPVAL does not require the full field specifications to be provided but the first character checked is the high order character (left most) and continue until the specification list is exhausted.

The internal number (computational unaligned) is moved to PROFILE-PICTURE-PART2 (right most part of PROFILE-PICTURES) and LOW VALUES (zeros) are moved to the left most. The GO TO ... DEPENDING ON PROFILE-PICTURE (PROCESS-INPUT) isolates the type(s) of characters permitted. When the type(s) of characters are determined the GO TO's for SPECIAL-CHARACTER, NUMERIC-CHARACTER, ALPHABETIC-CHARACTER and BLANK-CHARACTER are set to proceed to VALID-CHARACTER or INVALID-CHARACTER. Profile will validate 15 types/combination of characters. These combinations are as follows:

INTERNAL EXTERNAL CORRESPONDING TYPES OF CHARACTERS

01	A	ALPHABETIC
02	N	NUMERIC
03	S	SPECIAL
04	B	BLANK
05	E	FULL SET - MINUS BLANK
06	F	FULL SET
07	C	ALPHA OR BLANK
08	L	NUMERIC OR BLANK
09	M	ALPHA OR NUMERIC
10	T	ALPHA OR SPECIAL
11	U	SPECIAL OR BLANK
12	X	SPECIAL OR NUMERIC
13	K	ALPHA, NUMERIC OR BLANK
14	V	SPECIAL, ALPHA OR BLANK
15	Z	SPECIAL, NUMERIC OR BLANK

SUBCON/SUBCHK.

SUBCON indicates that an input conversion routine is to be used on a data field. The name of this routine is stored in VAL-LIT-AREA (format XXXXS).

SUBCHK indicates that an input checking or validation is to be performed via the subroutine. This specification allows the user to design his own input validation. The name as mentioned above is stored in VAL-LIT-AREA. The subroutine is passed to LINK (an ALC subroutine designed to load and execute the subroutine requested in paragraph RETURN-TO-SUBCON). Data is passed through the MIDMS standard calling sequence, CALLING-SEQUENCE. When control is returned, the length of the output and the EXIT-FLAG are checked. If successful, a one is moved to SUBCON-SW to signal to the Transaction Builder to take the data from the SUBCON output area and not the input area.

SUBCHK functions on the same order as SUBCON, except data is not passed back thru OUT-DATA valid or not.

Graphic display of VAL-LIT-AREA:

VAL-TYPE		FORMAT
1	RANGE	NN...NN Low value of range
9	RANGE	NN...NN High value of range
2	PROFILE	PP...PP (Binary) profile code
3	SUBCON	XXXXS Name of conversion subroutine
4	SUBCHK	XXXXS Name of VAL Check Subroutine
5	VALUE	NNNN VALUE Partial Value
6	OPR	1, 2, 3, 4, 5 and 6
	OPR 1	(1 OP) OP=Opcode
	OPR 2	(2 OP LLLL MMMM)
	OPR 3	3 NNNN L LITERAL OP NNNN = HOP of value to compare L = Length of literal (1-9)
	OPR 4	4 SSSS L LITERAL OP SSSS = Subscript of FLD-NAME entry in FIELD-LIST L = Length of literal (1-9)
	OPR 5	5 SSSS TTTT OP SSSS, TTTT = Subscripts of FLD-NAME entry in FIELD-LIST
	OPR 6	6 L LITERAL OP L = Length of literal
7	VALUE	OONN List where OONN = Number of elements.

Periodic Set Assignment.

The set assignment is very simply done. As previously mentioned all transactions passed to update a record must be sorted from left (RECORD ID) to right (Variable Set if present). The FFT set notation will not permit this; therefore, the values are changed for this purpose only. Basically, we are concerned about four kinds of sets -- FIXED, PERIODIC, VARIABLE and LMCON.

Set Notation.

FFT NOTATION		FMIPVAL
SET TYPE	VALUE	
FIXED	+00	000
PERIODIC	-NN	0NN
VARIABLE	+NN	1NN
LMCON		200

Periodic Subset Sequence (PSSQ) Number Assignment.

There are five different ways to assign PSSQ's:

Numeric Pseudos (600-699)
Ann Pseudos (A01-A99)
Actual PSSQ (000-599)
Blank PSSQ (ΔΔΔ)
No PSSQ Mentioned

There are two tables used by FMIPVAL and FMIPX to control the assignment of PSEUDO PSSQ's but before these tables can be used FMIPVAL must determine the characteristics of the PSSQ. The PSSQ may be from zero to three positions long and can exist in four forms.

If the PSSQ length is zero, a blank PSSQ is assumed. If the length is less than three and greater than zero, the PSSQ must be actual PSSQ or blank PSSQ. Numeric pseudo (600-699) and Ann pseudos must be three characters.

Numeric pseudos, Ann pseudos and blank pseudos cannot be used with the following opcodes and field types:

ILLEGAL OPCODES WITH PSEUDO

OPCODE	FIELD TYPES	
ADD	Periodic Field	
SUB	Periodic Field	
CHG	Periodic Field	Subset
CHGU	Periodic Field	Subset
DEL	Periodic Field	Subset
NEX	Periodic Field	Subset

FMIPVAL controls this condition by testing BAD-PSSQ-SW switch each time a blank PSSQ, numeric pseudo and Ann pseudo is found. BAD-PSSQ-SW switch is set to one when one of the above opcode is assigned. If an error is detected, an appropriate error message is given and since the transactions have been built pointers must be adjusted to eliminate

the transaction. This is accomplished by subtracting T-LENGTH (the length of the transaction) from the test pointer (T-POINTER) and the text counter (TEXT-COUNT).

The process for determining what PSSQ's to assign for blank, numeric pseudo and Ann pseudos is described below. There are two tables as previously mentioned containing 49 elements (for 49 sets) named GEN-PSEUDO and USER-PSEUDO. The table keeps track of the previous pseudo used for each [illegible].

The subscript that points to the current set is M. If the PSSQ provided is blank or greater than 599 and equal to the previous one (PSSQ) used for that set, use the GEN-PSEUDO for that set. If they are not equal, a test is made to determine if the previous one generated (GEN-PSEUDO) is equal to 600. If that condition is true, the generated pseudo is increased by one. The USER-PSEUDO is assigned the input PSSQ and the PSSQ for the transaction is taken from GEN-PSEUDO (paragraph MOVE-GEN).

If the PSSQ provided is an Ann pseudo and equals the previous user pseudo for that set, the previously generated GEN-PSEUDO for that set is used. If previous USER-PSEUDO was not of the Ann type, the value nn is added to the GEN-PSEUDO. If the previous USER-PSEUDO was a different Ann, the value of GEN-PSEUDO is adjusted up or down reflecting the difference of the nn's. The new input PSSQ value is saved in the USER-PSEUDO for later comparison. The subset is then given the GEN-PSEUDO and PSSQ processing is terminated.

Operation Code (OPR).

MIDMS provides the user with a wide choice of operation codes (21) which can be attached to any field of the data source. Since some of these opcodes would attempt to execute impossible or illegal operations, each opcode must be fully checked for validity in the

situation used. The following table contains all the opcode/field type possible and the invalid combinations are denoted by an asterisk (*). The operation code described as OP contains the two digit number specified after the operation abbreviation in the left column.

	RECID	FIXED	PER	F/G	VSET	SUBSET	PSCNN	VSCNN	LMCON
NOP 00	001	002	003	004	005	006	007	008
ADD 01	011*	012	013	014*	015*	016*	017*	018
SUB 02	021*	022	023	024*	025*	026*	027*	028
CHG 03	031*	032	033	034	035	036*	037*	038
CHGU04	041*	042	043	044	045	046*	047	048
GEN 05	051	052	053	054	055	056*	057*	058*
CRE 05								
GENU06	061*	062	063	064	065	066*	067*	068*
GEC 07	071	072	073	074	075	076*	077*	078*
CAD 07								
GECU08	081*	082	083	084	085	086*	087*	088*
CID 09	091	092*	093*	094*	095*	096*	097*	098*
DEL 10	101	102	103	104*	105	106	107	108*
ATC 11	111*	112*	113*	114	115*	116*	117*	118*
NEX 12	121*	122	123	124*	125	126*	127*	128
CMA 13	131*	132	133	134*	135	136*	137*	138*
ADS 14	141*	142*	143	144*	145	146*	147*	148*
REP 15	151	152*	153*	154	155*	156*	157*	158*
REG 16	161	162*	163*	164	165*	166*	167*	168*
GER 17	171	172*	173*	174*	175*	176*	177*	178*
GEV 18	181*	182*	183*	184	185*	186*	187*	188*
ATV 19	191*	192*	193*	194	195*	196*	197*	198*
CHV 20	201*	202*	203*	204	205*	206*	207*	208*

NOTE * DENOTES ERROR.

Opcodes can be conditioned. That is, they may be selected if certain tests are valid. Opcodes, either conditional or unconditional may be specified on the FILE, RECORD, or FIELD level. The field level opcode overrides the other two if they are present. When the opcode is processed for a transaction, the PSSQ, if any, is next determined for inclusion in TRANS-DATA.

OPR Types.

OPR Type 1 - Paragraph OPR1
Format: 1,OP
where 1 denotes the type of OPR and OP is the unconditional opcode.

OPR Type 2 - Paragraph OPR2

Format: 2,LLLL,MMMM,OP

where 2 denotes the type and OP is the unconditional opcode for the variable set. TYPE 2 opcode is only valid for VSET. If data is not variable, an error message is printed. LLLL is a pointer to another number. This number will represent the position that characters will be removed and MMMM is a pointer to a number telling how many characters to remove.

OPR Type 3 - Paragraph OPR3

Format: 3-NNNN-L-LITERAL-OP

where NNNN is the subscript (HOP) of the data to be tested, L is the length of the literal to be compared and OP is the opcode. The maximum length of L is 9. Subroutine OPR34 is used to isolate and convert these values which are stored in VAL-LIT-AREA. OPR34 is passed the HOP of the subscript (NNNN) and OPR34 returns the value of NNNN (POSIT), L (LENGTHY), the literal (LIT-HOLD) and the OPCODE (OPR-NUM) in binary. LIT-HOLD and the subscript of the INPUT-RECORD area (POSIT) are passed to COMARAYS for comparison. If the values are equal, the opcode is converted by the OPR-CONVERT routine and the transaction is built. Otherwise, control is returned and the next opcode is processed.

OPR Type 4 - Paragraph OPR4

OPR4 is basically the same as OPR3 except when the subscript (POSIT) is returned by OPR34, POSIT does <u>not</u> point at the INPUT-RECORD area but a field name subscript in the FIELD-LIST area. The position of that field is then used in the subroutine COMARAYS to compare the logical information. Note when paragraph OPR3 is PERFORMed from paragraph OPR4, INDEX1 contains the subscripted value. This value is then used as a subscript (MOVE FLD-HOP (INDEX1) TO INDEX3, SAVE-SUBSCRIPT) for the field in question. At this point, we acquire the position of the data we wish to compare.

OPR Type 5 - Paragraph OPR5

Format: 5-SSSS-TTTT-OP

where SSSS is the first subscript of a field, TTTT is the second and OP is the opcode.

OPR Type 6 - Paragraph OPR6

Format: 6-L-LITERAL-OP

where L is the length of the literal and OP is the conditional opcode. OPR6 is generated by OMLPOPR to save storage area in VAL-LIT-AREA and processing time. OPR6 uses the subscript value stored in SAVE-SUBSCRIPT by a previous OPR3 or OPR4 to locate the data in the INPUT-RECORD and compare the data to the literal found in OPR6. If the values are not equal, control is passed to start the process again. However, if they are equal, the opcode is converted and stored for the Transaction Builder.

The Process.

First SSSS and TTTT are taken out of VAL-LIT-AREA and put into INDEX1 and INDEX 2 respectively.

SSSS ---- INDEX1
TTTT ---- INDEX2

Now using INDEX1 and INDEX2 find the length of the fields we are addressing in FIELD-LIST area and see if they are the same length. If they are, continue; otherwise, print appropriate error message. Obtain the position of each field in question and pass these values to subroutine COMARAYS (MOVE FLD-HOP (INDEX1) TO INDEX3 MOVE FLD-HOP (INDEX2) TO INDEX4).

Call the subroutine and if the result is not equal, return control to start the process again.

If the fields are equal, control is passed to paragraph FORMAT-OPR5, where the opcode is converted.

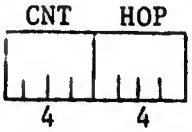
Logical Maintenance Constants (LMCON) are supposed to be used to change values that have been defined in a Logical Maintenance package. (Changing of LMCON's has not been implemented in FM Maintenance Proper.) The disposition of these constants requires a special format for the data. The format of the date is as follows:

NAME	VALUE
FLDA[illegible]	12345

Therefore, the length of the field is increased by six.

All non-FFT fields are considered LMCONs.

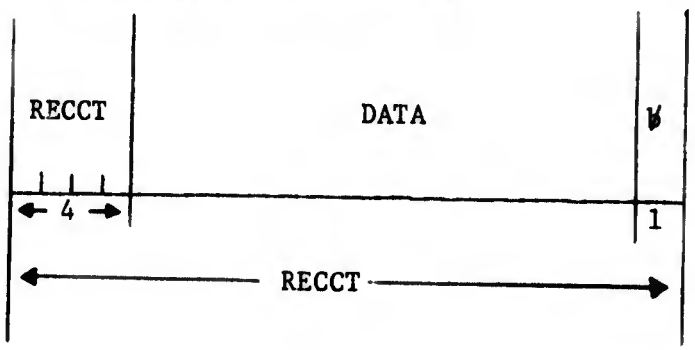
Variable Length Data. FLD-HOP is 0 only when variable length input is used. PSS-HOP and PSS-LEN specify a variable control field of the following format:



This control field is 8 digits with leading zeros assumed to fill the 8 standard positions. When the control field is blank, or the count portion is 0, a NOP is assumed and processing of the field is terminated. CNT contains the number of characters of input data. HOP contains the position (relative to 1) of the left-most character of the data field in this record. Only REP, REG, GEV, ATV, and CHV used against variable sets are valid with a variable length input field.

Variable set operands REP, REG, GEV, ATV, and CHV operators can be used against variable sets with the data in either variable length (see above) or standard fixed position formats. CHV may have either of the two CHG formats.

Whole Record Operations. Three operations receive as data a complete MIDMS data file record. These operations are REPlace an existing record (REP), REplace or Generate a record (REG), and GEnerate a new Record (GER). REP, REG, and GER must have record ID operands to be whole record operations. However, the location of the data specified in FLD-HOP must refer to the record character count (RECCT) field of the record. The format of a MIDMS data file record is:



All periodic and variable set control fields (PSC's and VSC's) must be exactly in the format normally produced by MIDMS File Maintenance since no changes will be made to this data before it is inserted into the output file by Maintenance Proper. When whole record operations are used, no other data file field may have a transaction against the same record. This includes the input record, the input group, as well as other input groups against the data file record in the same File Maintenance job. LMCON's may have transactions. To insure this requirement, TRANS-FLDNO in the transaction entry is set to 1 whenever REP, REG, and GER refer to a Record ID. One is used as the field type. The LR2 entry of 1 specifies the RECCT field which cannot have any other transactions against it and at the same time causes these new operations to sort first among the transaction items.

Transaction Building. The building of the transaction is the ultimate goal of FMIPVAL. Before these transactions can be built, all validation, conversion and the operation code must be assigned.

The output of FMIPVAL consist basically of one array (UNSORTED-TRANS-TAB) and a string of characters (TEXT-TAB). UNSORTED-TRANS-TAB and TEXT-TAB are defined as follows:

```
01  UNSORTED-TRANS-TAB.
    03  TRANS-ITEM OCCURS 1000 TIMES.
      04  TRANS-SORT.
        05  TRANS-SETNO PICTURE 999          COMPUTATIONAL.
        05  TRANS-SUBSET PICTURE 9999        COMPUTATIONAL.
        05  TRANS-FLDNO PICTURE 9999         COMPUTATIONAL.
      04  TRANS-OPCODE  PICTURE 999          COMPUTATIONAL.
      04  TRANS-LENGTH  PICTURE 999          COMPUTATIONAL.
      04  TRANS-HOP     PICTURE 9999         COMPUTATIONAL.
01  TEXT-TAB.
    03  TEXT-CHAR OCCURS 10000 TIMES         PICTURE X.
01  TEXT-TAB-X REDEFINES TEXT-TAB.
    03  TEXT-DATA OCCURS 10 TIMES            PICTURE X(1000).
```

Each entry in UNSORTED-TRANS-TAB defines a transaction. The field against which the transaction is applied is indicated by TRANS-FLDNO, which is the entry in MAJOR-LR2 for that field. The operation to be performed is indicated by TRANS-OPCODE.

TRANS-OPCODE is a three digit number -- the first (HIGH ORDER), the second digit defines the opcode and the third (RIGHT MOST) defines the field type.

TRANS-SET-NO is produced in accordance with specifications provided in the discussion of SET-ASSIGNMENT.

Periodic transactions have their subsets in TRANS-SUBSET.

If the transaction includes any data (and it usually does), the data is sorted in TEXT-TAB. TRANS-HOP indicates the point in TEXT-TAB where the data from this transaction starts while TRANS-LENGTH contains the number of characters in the data.

Variable set data does not require the use of field number (TRANS-FLDNO) and subset (TRANS-SUBSET); however, TRANS-FLDNO is used to show Maintenance Proper the position in the variable set where the text is to be inserted and TRANS-SUBSET is used to store the number of characters to remove. (The number of characters removed may be different from the length of the input transactions.)

The TRANS-OPCODE field contains values combining operation codes and field type codes as indicated below.

Operation Codes.

NOP	00
ADD	01
SUB	02
CHG	03
CHU	04
GEN	05
CRE	05
GENU	06
GEC	07
CAD	07
GECU	08
CID	09
DEL	10
ATC	11
NEX	12

CMA	13
ADS	14
REP	15
REG	16
GER	17
GEV	18
ATV	19
CHV	20

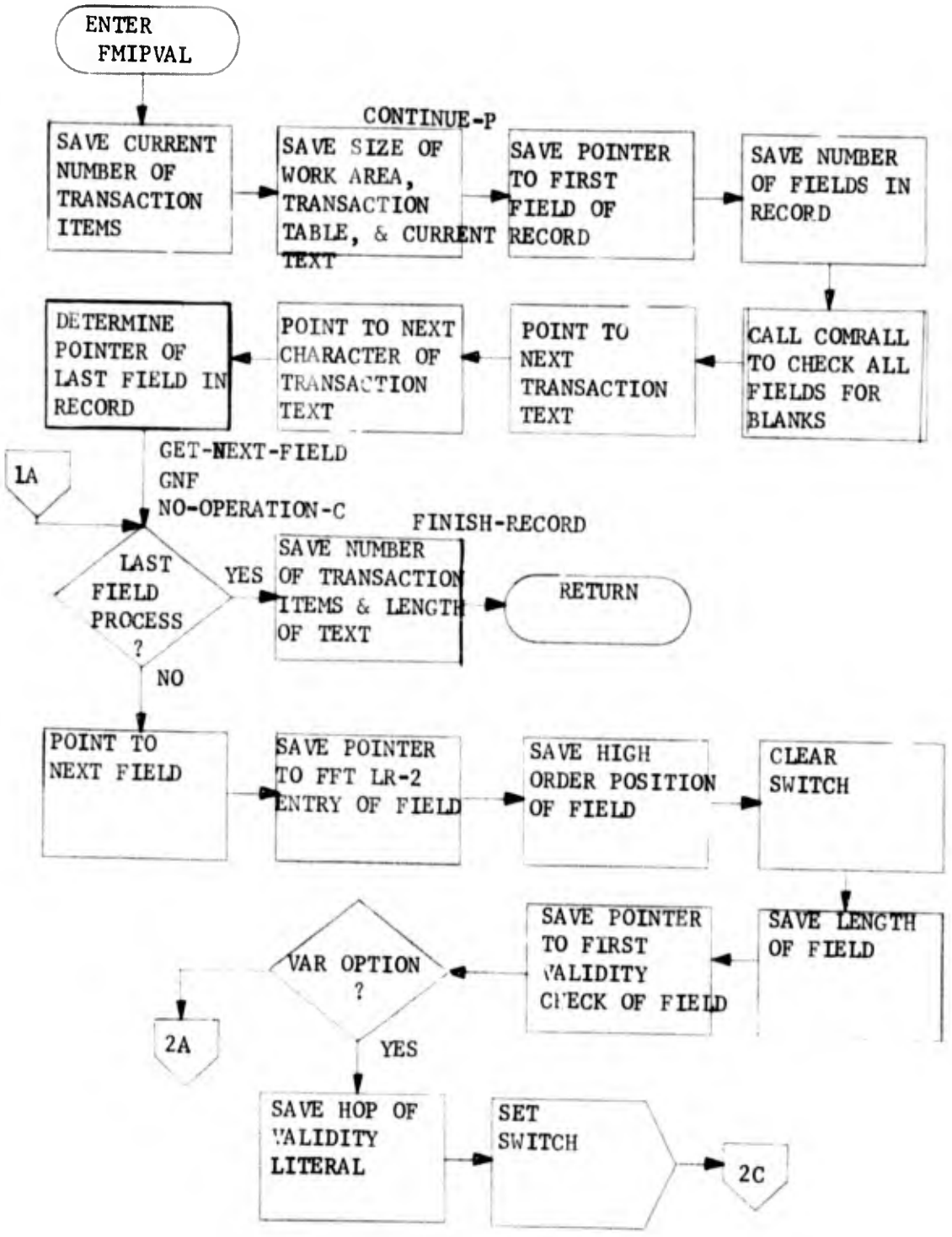
Field Type Codes.

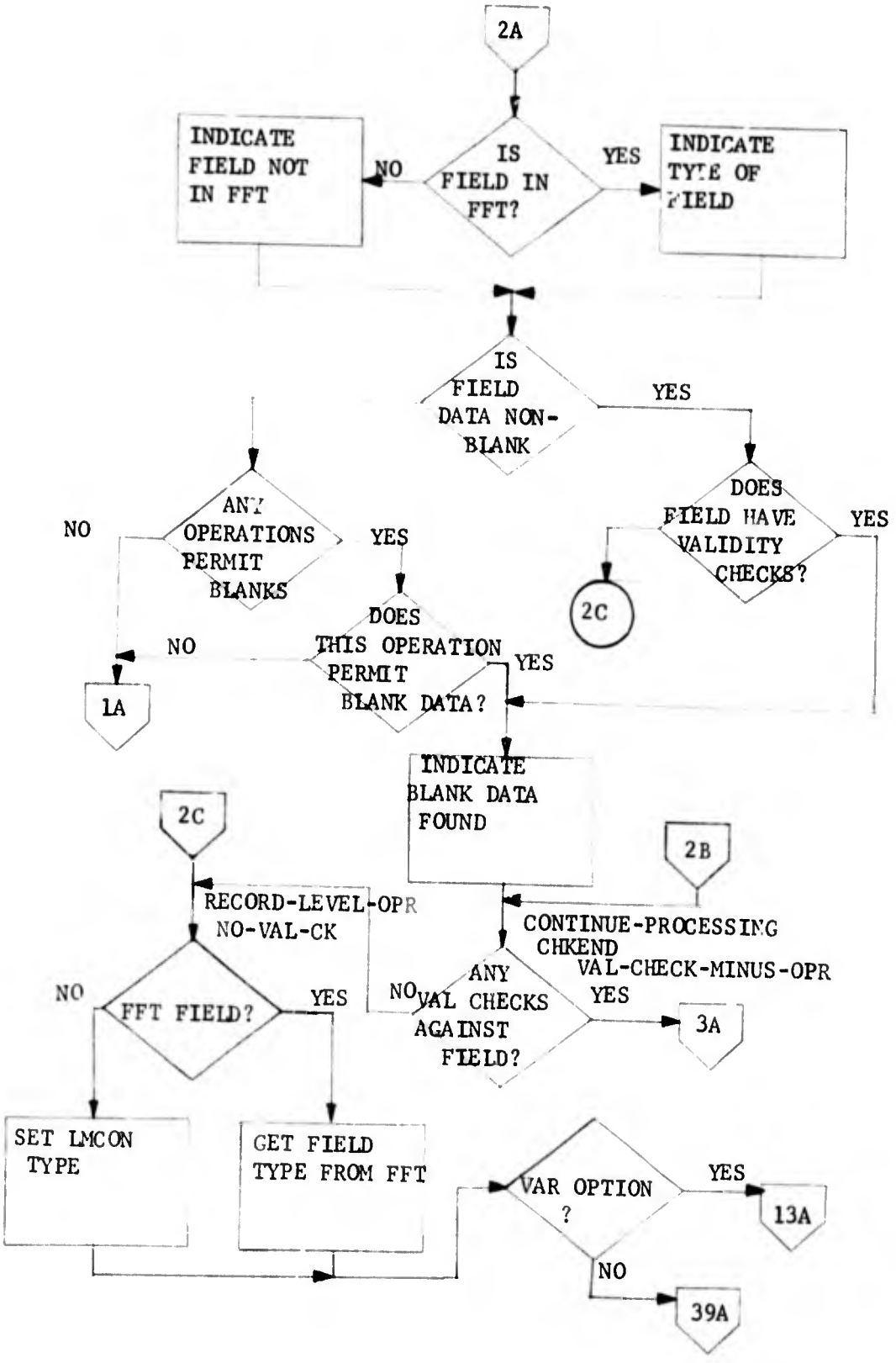
1 - RECORD ID
2 - FIXED
3 - PERIODIC FIELD/GROUP
4 - VARIABLE SET
5 - PERIODIC SUBSET
6 - PERIODIC SET CONTROL
7 - VARIABLE SET CONTROL
8 - LMCON

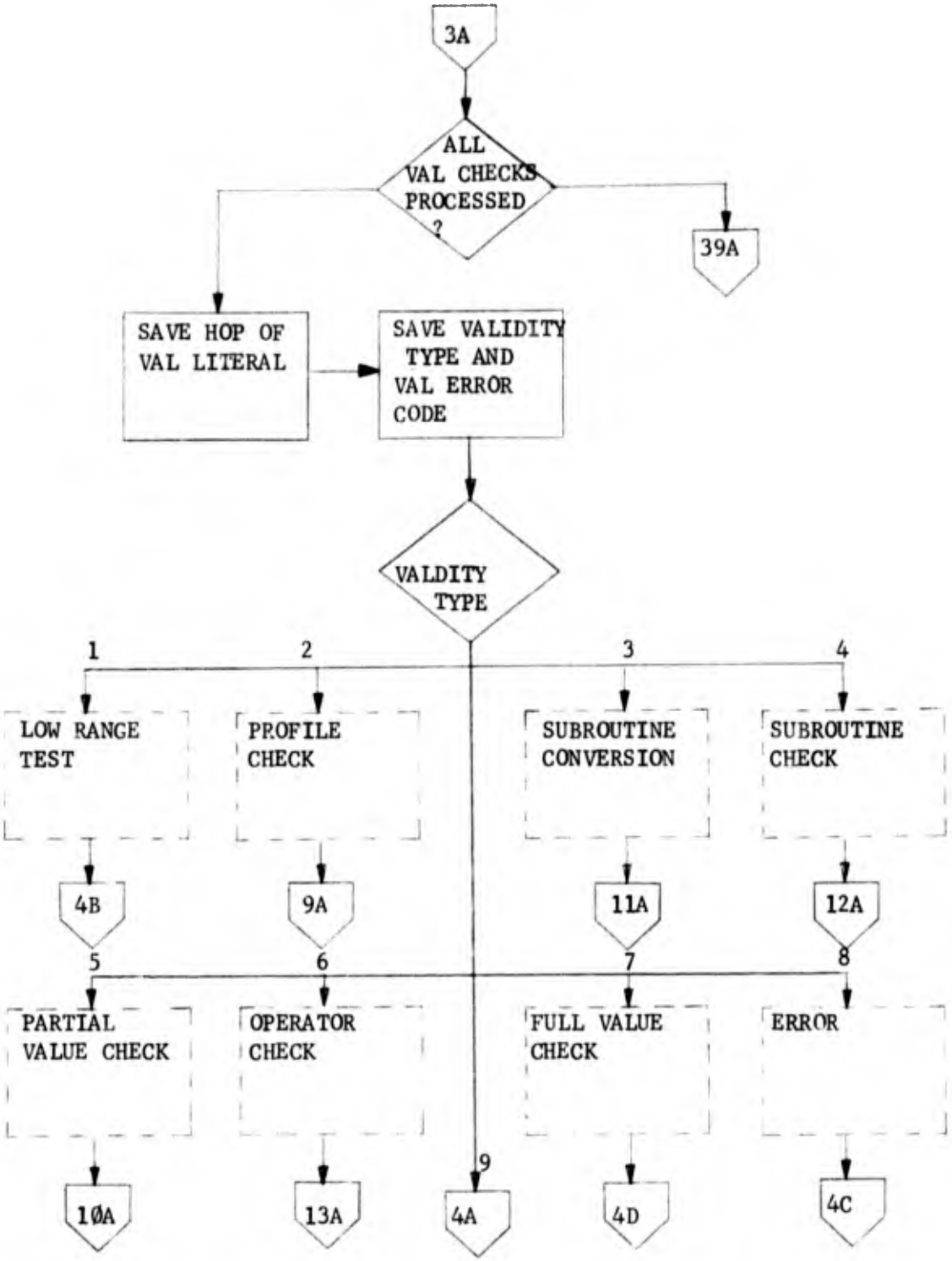
Error Codes. Throughout the program, frequent reference is made to the error action codes. There are nine error action codes as follows:

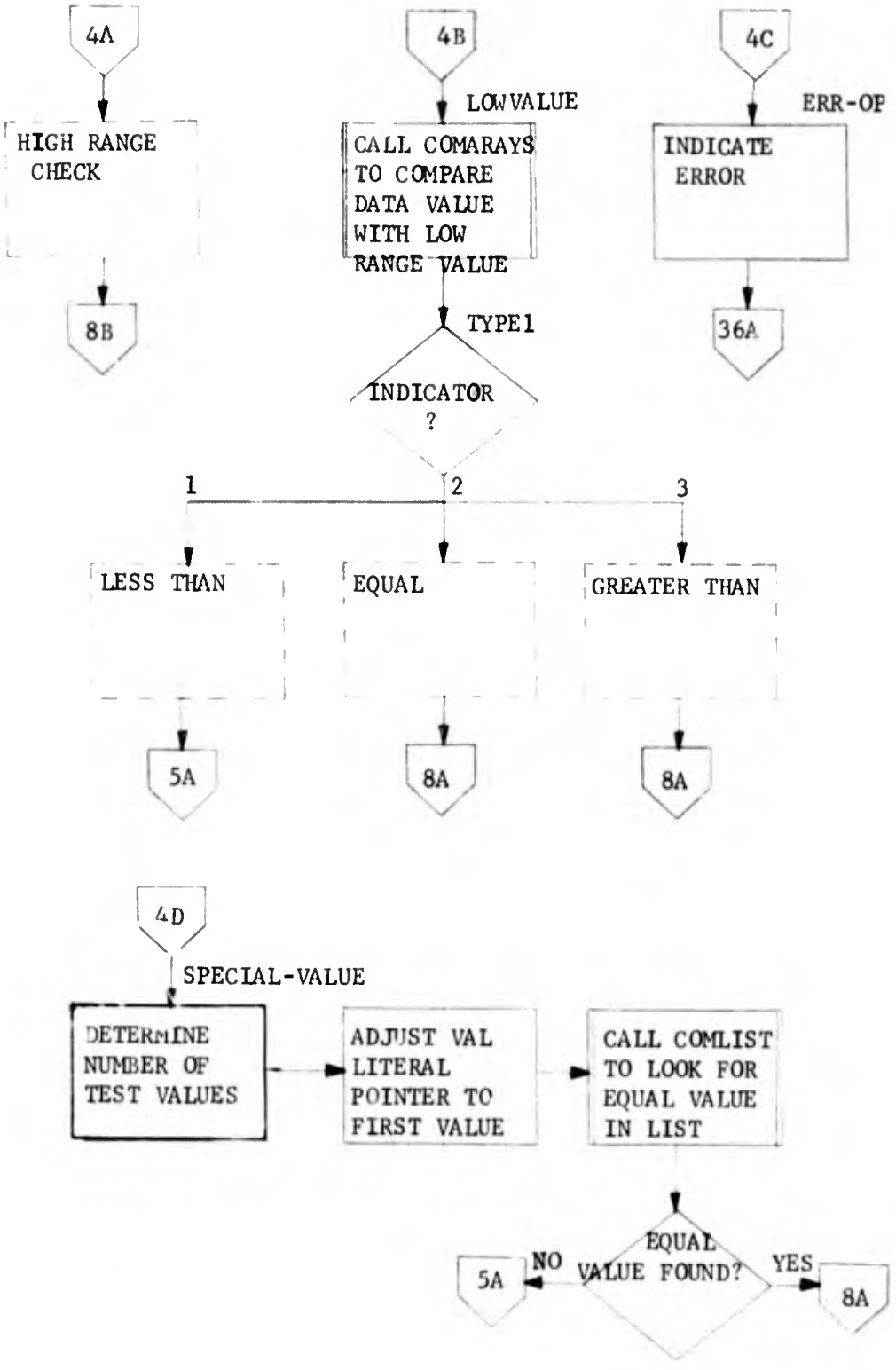
EXTERNAL	INTERNAL	ACTION TO BE TAKEN IN CASE OF ERROR
D	1	DISCONTINUE PROCESSING OF THE FIELD IN SOURCE REC
R	2	RECORD AS INPUT IS TO BE REJECTED
S	3	SUBGROUP AS INPUT IS TO BE REJECTED (Not implemented)
G	4	GROUP AS INPUT IS TO BE REJECTED
C	5	CONTINUE (ACCEPT THE FIELD) AS IF NO ERROR
N	6	NO COMMENT DISCONTINUE PROCESSING
O	7	OR CONDITION
A	8	AND CONDITION.
B	9	SAME AS D EXCEPT BLANK OUT INPUT FIELD.

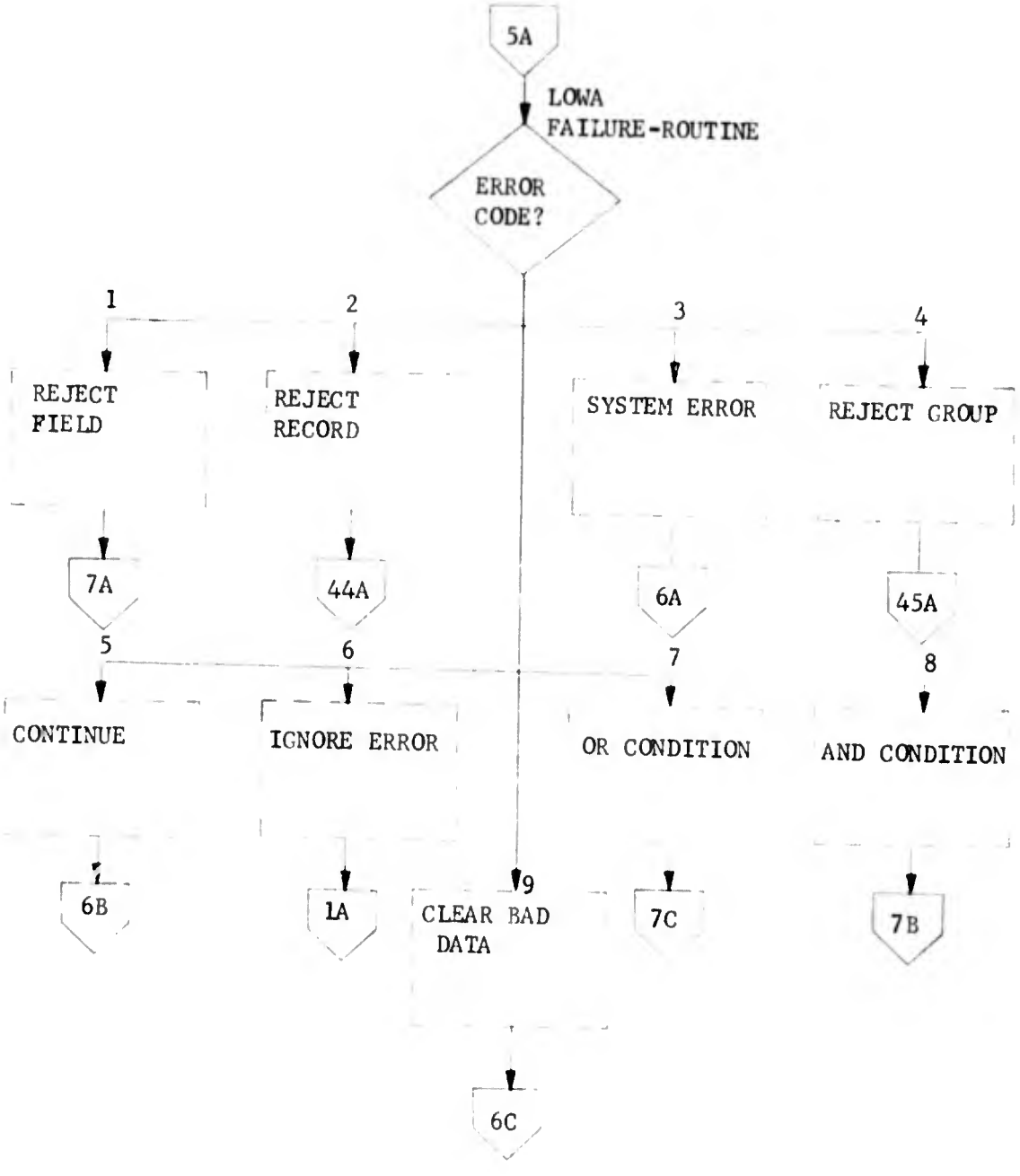
(4) Flowchart.

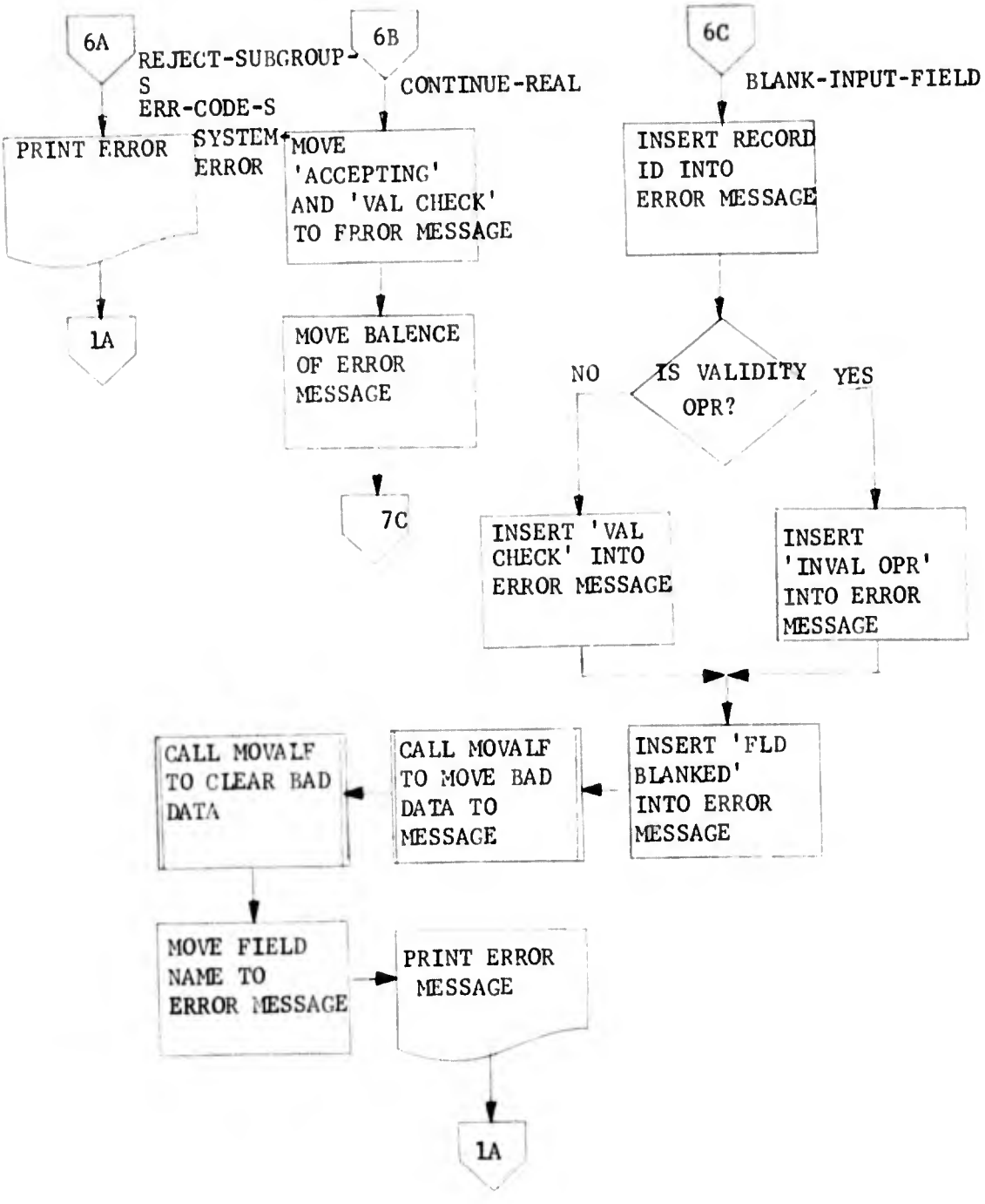

ENTER FMIPVAL
SAVE CURRENT NUMBER OF TRANSACTION ITEMS
CONTINUE-P
SAVE SIZE OF WORK AREA, TRANSACTION TABLE, & CURRENT TEXT
SAVE POINTER TO FIRST FIELD OF RECORD
SAVE NUMBER OF FIELDS IN RECORD
CALL COMRALL TO CHECK ALL FIELDS FOR BLANKS
POINT TO NEXT TRANSACTION TEXT
POINT TO NEXT CHARACTER OF TRANSACTION TEXT
DETERMINE POINTER OF LAST FIELD IN RECORD
GET-NEXT-FIELD
GNF
NO-OPERATION-C
1A
LAST FIELD PROCESS ?
YES
FINISH-RECORD
SAVE NUMBER OF TRANSACTION ITEMS & LENGTH OF TEXT
RETURN
NO
POINT TO NEXT FIELD
SAVE POINTER TO FFT LR-2 ENTRY OF FIELD
SAVE HIGH ORDER POSITION OF FIELD
CLEAR SWITCH
SAVE LENGTH OF FIELD
SAVE POINTER TO FIRST VALIDITY CHECK OF FIELD
VAR OPTION ?
2A
YES
SAVE HOP OF VALIDITY LITERAL
SET SWITCH
2C

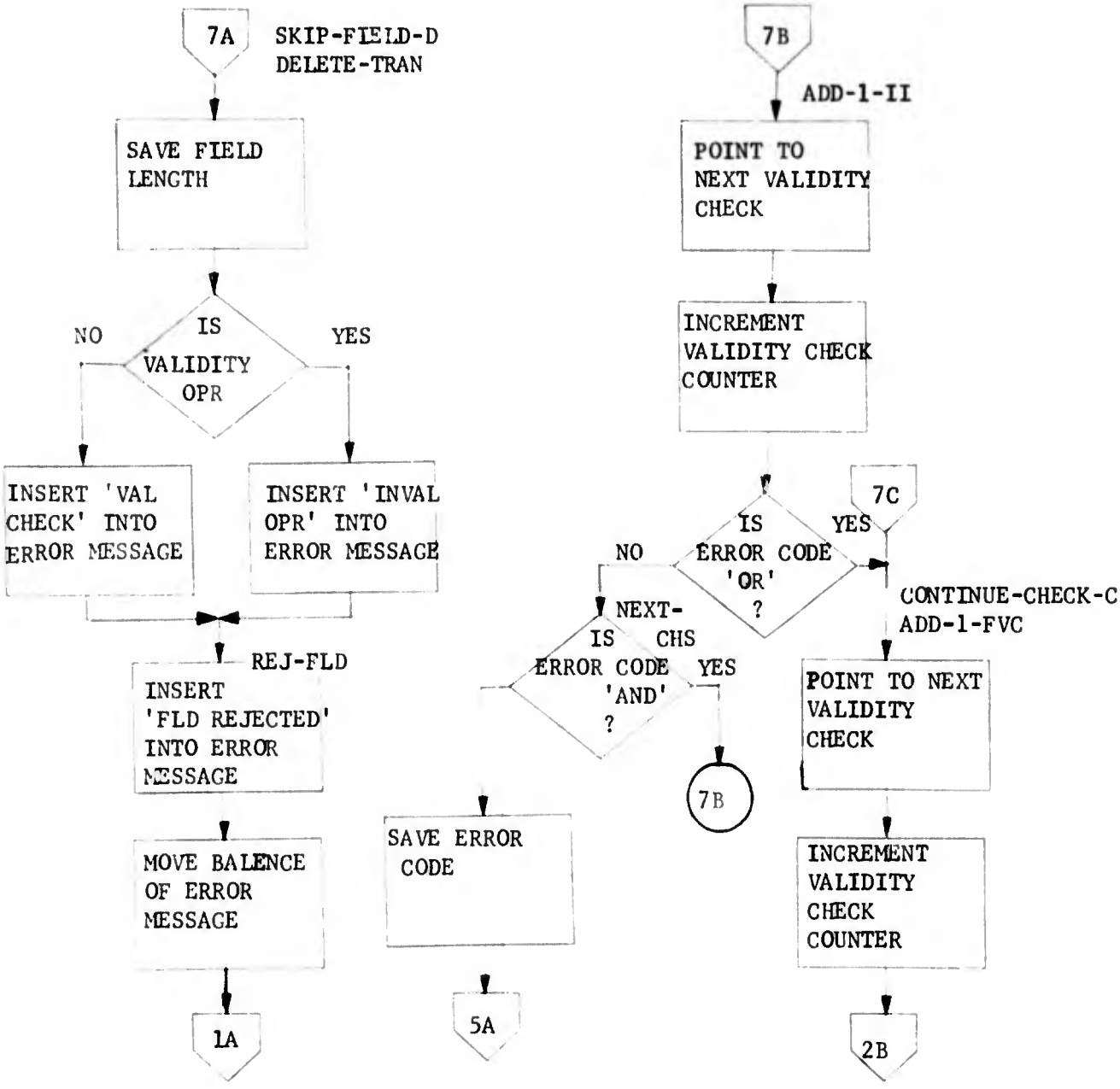

2A
INDICATE FIELD NOT IN FFT
NO
IS FIELD IN FFT?
YES
INDICATE TYPE OF FIELD
IS FIELD DATA NON-BLANK
YES
DOES FIELD HAVE VALIDITY CHECKS?
YES
2C
NO
ANY OPERATIONS PERMIT BLANKS
YES
NO
DOES THIS OPERATION PERMIT BLANK DATA?
YES
1A
INDICATE BLANK DATA FOUND
2C
2B
RECORD-LEVEL-OPR NO-VAL-CK
CONTINUE-PROCESSING CHKEND
VAL-CHECK-MINUS-OPR
NO
FFT FIELD?
YES
NO
ANY VAL CHECKS AGAINST FIELD?
YES
3A
SET LMCON TYPE
GET FIELD TYPE FROM FFT
VAR OPTION ?
YES
13A
NO
39A

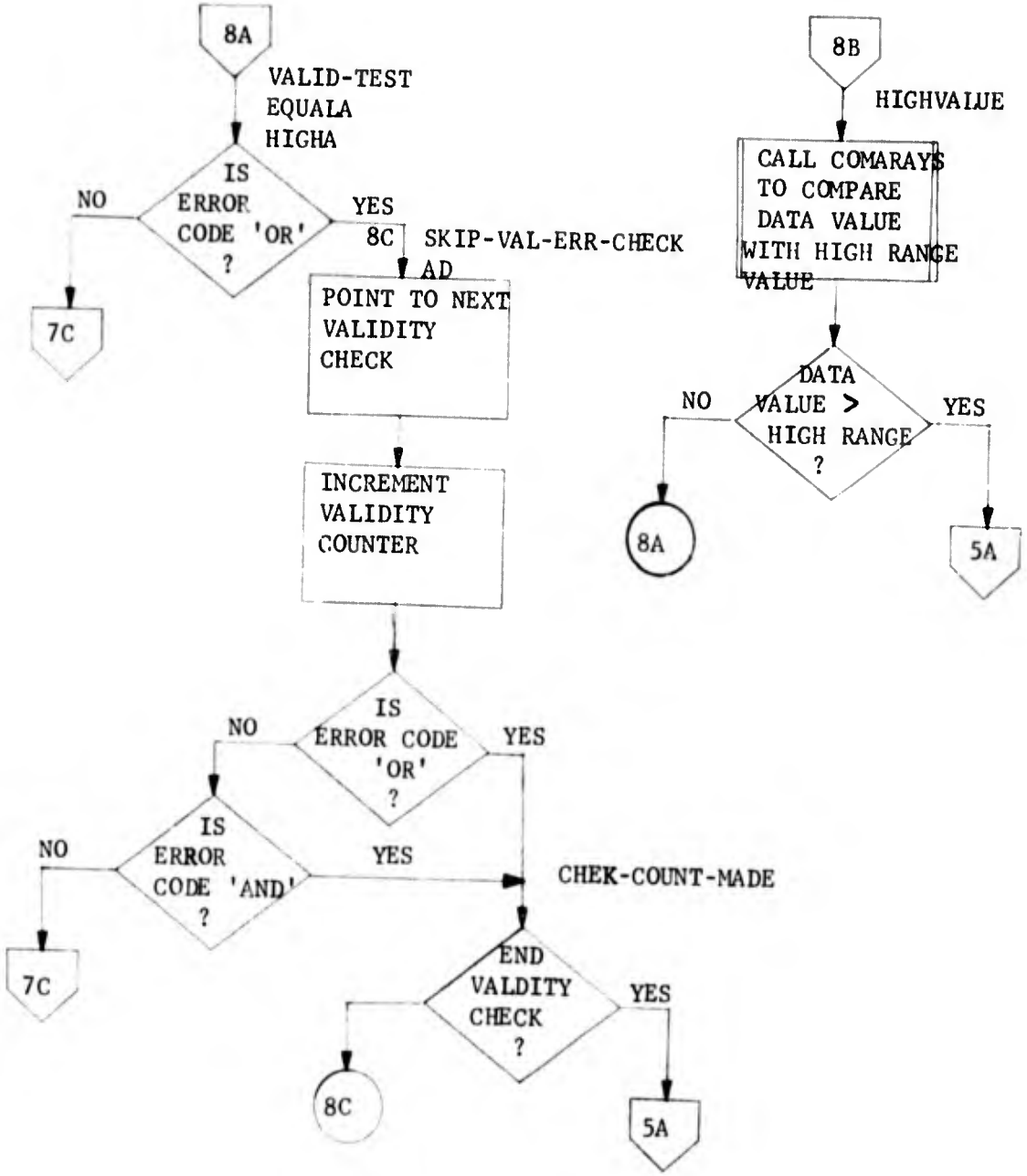

3A
ALL VAL CHECKS PROCESSED ?
39A
SAVE HOP OF VAL LITERAL
SAVE VALIDITY TYPE AND VAL ERROR CODE
VALDITY TYPE
1
2
3
4
LOW RANGE TEST
PROFILE CHECK
SUBROUTINE CONVERSION
SUBROUTINE CHECK
4B
9A
11A
12A
5
6
7
8
PARTIAL VALUE CHECK
OPERATOR CHECK
FULL VALUE CHECK
ERROR
9
10A
13A
4A
4D
4C

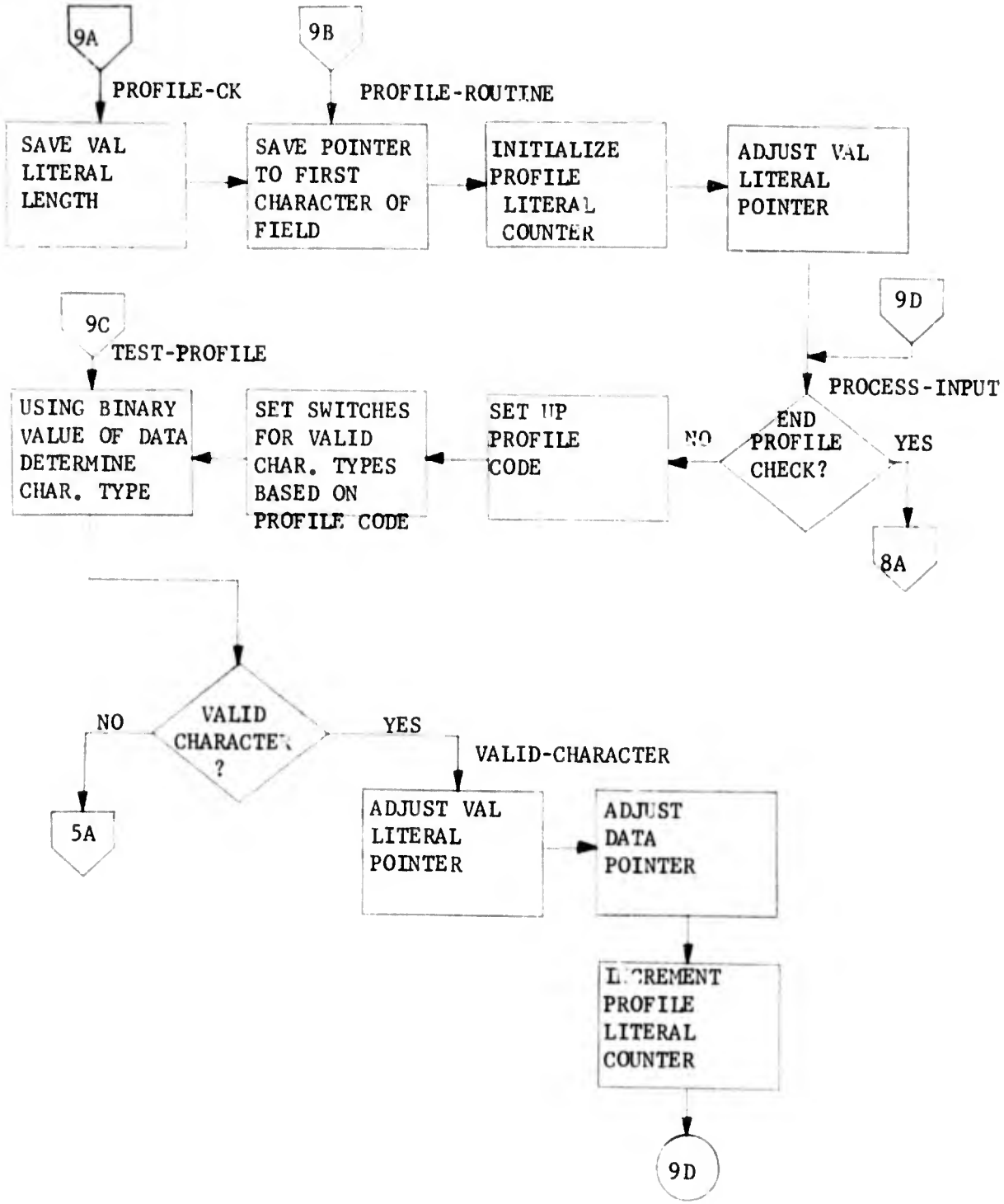

4A
HIGH RANGE CHECK
8B
4B
LOWVALUE
CALL COMARAYS TO COMPARE DATA VALUE WITH LOW RANGE VALUE
TYPE1
INDICATOR ?
1
2
3
LESS THAN
EQUAL
GREATER THAN
5A
8A
8A
4C
ERR-OP
INDICATE ERROR
36A
4D
SPECIAL-VALUE
DETERMINE NUMBER OF TEST VALUES
ADJUST VAL LITERAL POINTER TO FIRST VALUE
CALL COMLIST TO LOOK FOR EQUAL VALUE IN LIST
EQUAL VALUE FOUND?
NO
YES
5A
8A

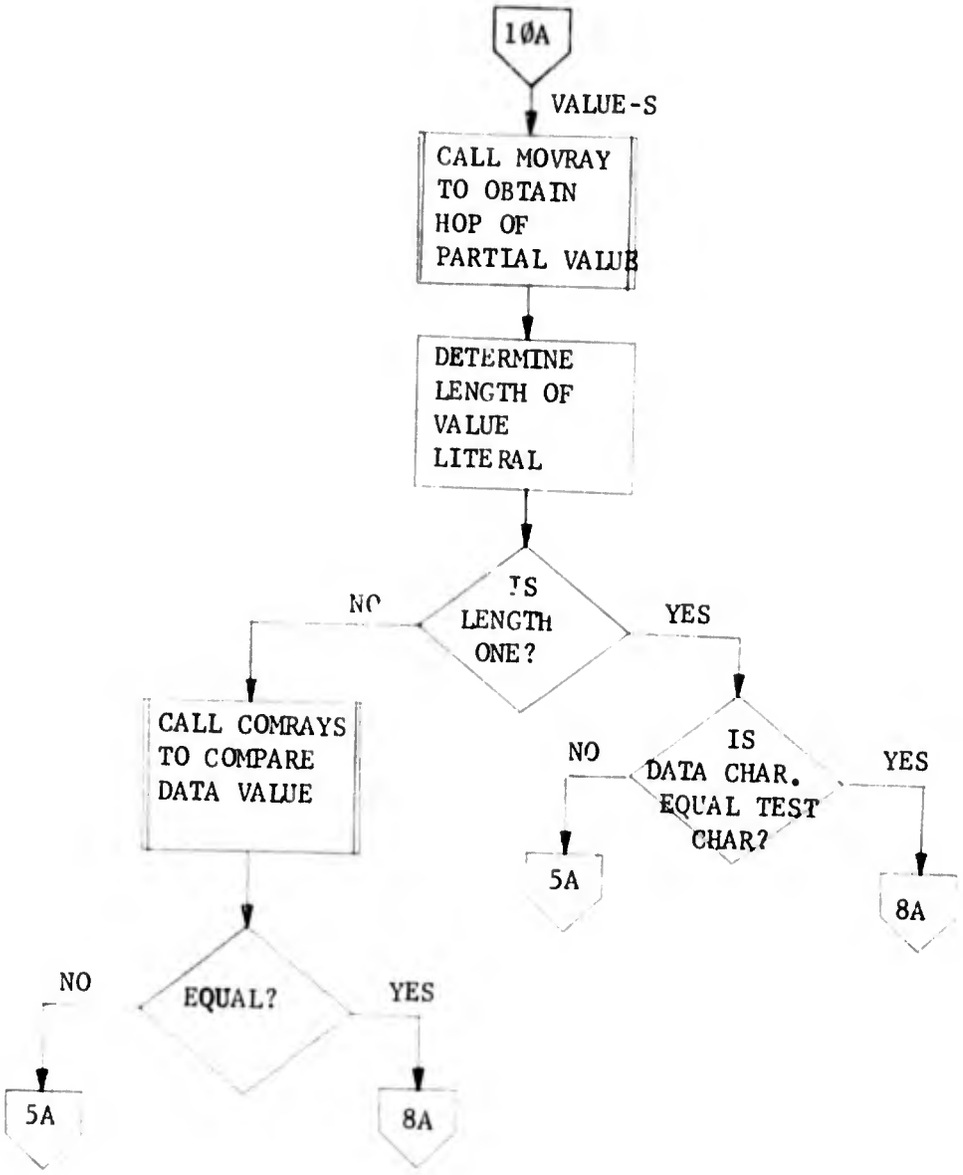

5A
LOWA
FAILURE-ROUTINE
ERROR
CODE?
1
2
3
4
REJECT
FIELD
REJECT
RECORD
SYSTEM ERROR
REJECT GROUP
7A
44A
6A
45A
5
6
7
8
CONTINUE
IGNORE ERROR
OR CONDITION
AND CONDITION
6B
1A
9
CLEAR BAD
DATA
7C
7B
6C

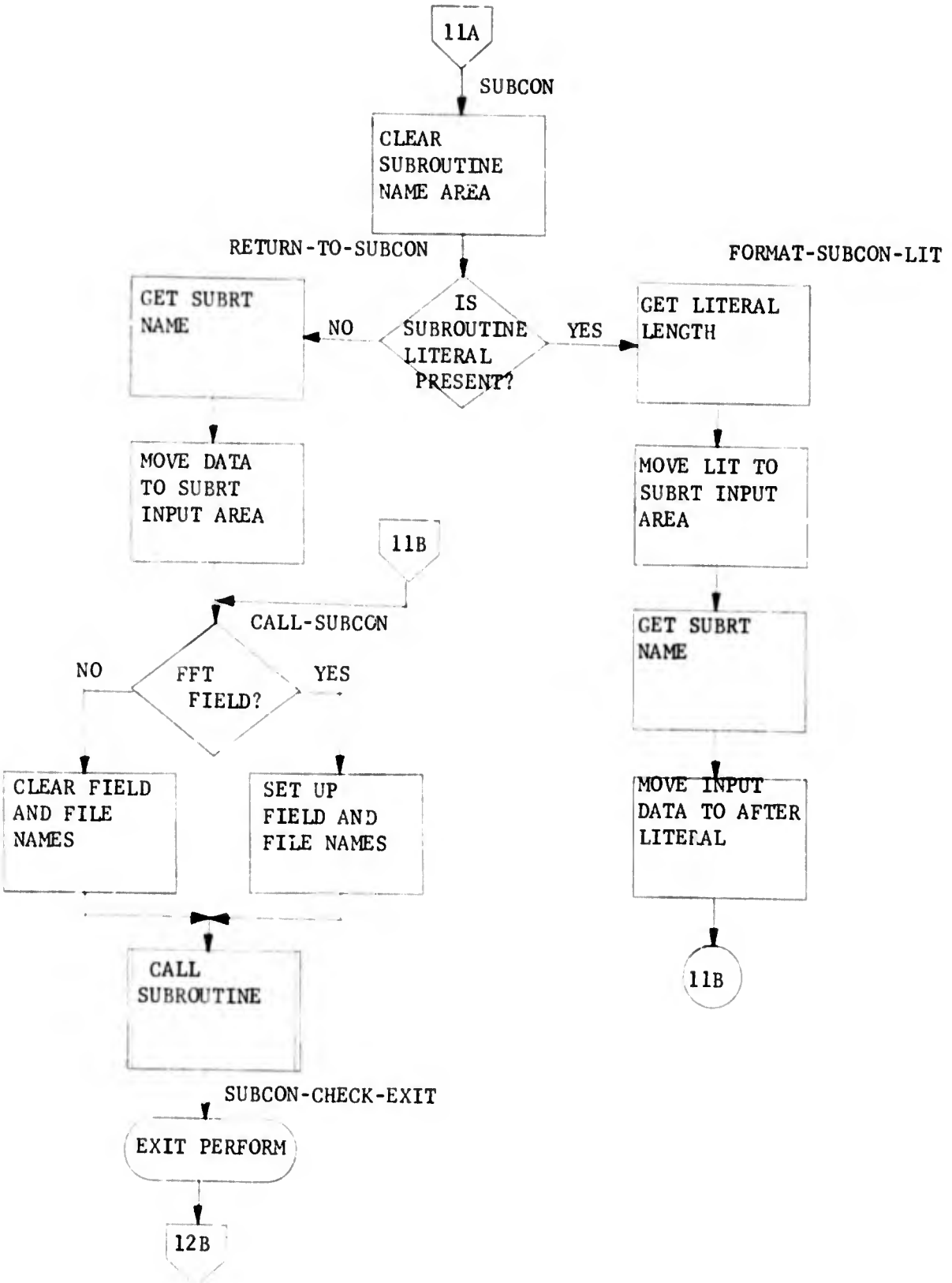

6A
REJECT-SUBGROUP-S
ERR-CODE-S
SYSTEM ERROR
6B
CONTINUE-REAL
6C
BLANK-INPUT-FIELD
PRINT ERROR
MOVE 'ACCEPTING' AND 'VAL CHECK' TO FRROR MESSAGE
INSERT RECORD ID INTO ERROR MESSAGE
1A
MOVE BALENCE OF ERROR MESSAGE
IS VALIDITY OPR?
NO
YES
7C
INSERT 'VAL CHECK' INTO ERROR MESSAGE
INSERT 'INVAL OPR' INTO ERROR MESSAGE
INSERT 'FLD BLANKED' INTO ERROR MESSAGE
CALL MOVALF TO MOVE BAD DATA TO MESSAGE
CALL MOVALF TO CLEAR BAD DATA
MOVE FIELD NAME TO ERROR MESSAGE
PRINT ERROR MESSAGE
1A

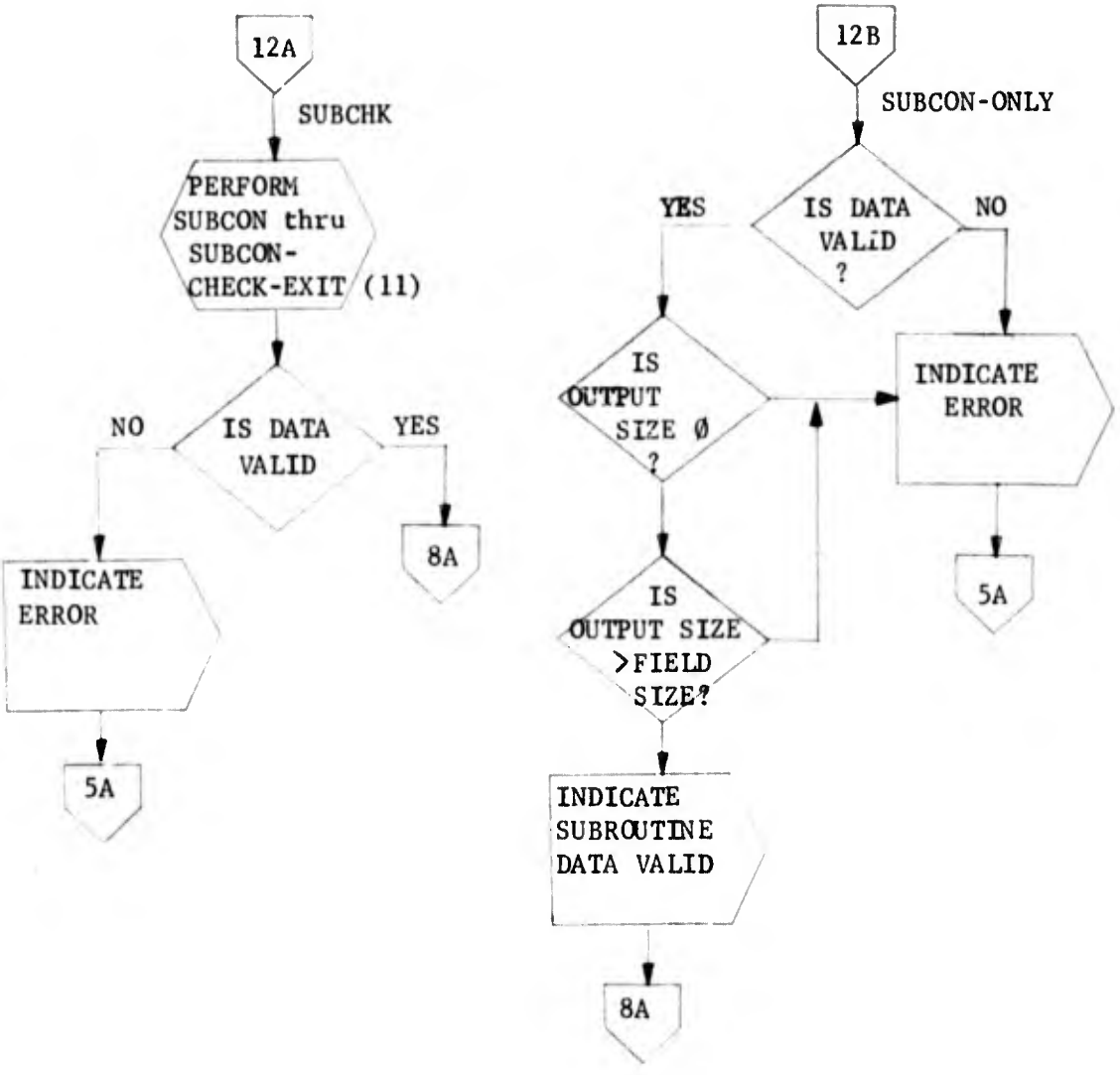

7A
SKIP-FIELD-D
DELETE-TRAN
SAVE FIELD LENGTH
IS VALIDITY OPR
NO
YES
INSERT 'VAL CHECK' INTO ERROR MESSAGE
INSERT 'INVAL OPR' INTO ERROR MESSAGE
REJ-FLD
INSERT 'FLD REJECTED' INTO ERROR MESSAGE
MOVE BALENCE OF ERROR MESSAGE
1A
7B
ADD-1-II
POINT TO NEXT VALIDITY CHECK
INCREMENT VALIDITY CHECK COUNTER
7C
IS ERROR CODE 'OR' ?
YES
NO
CONTINUE-CHECK-C
ADD-1-FVC
NEXT-IS CHS ERROR CODE 'AND' ?
YES
7B
SAVE ERROR CODE
5A
POINT TO NEXT VALIDITY CHECK
INCREMENT VALIDITY CHECK COUNTER
2B

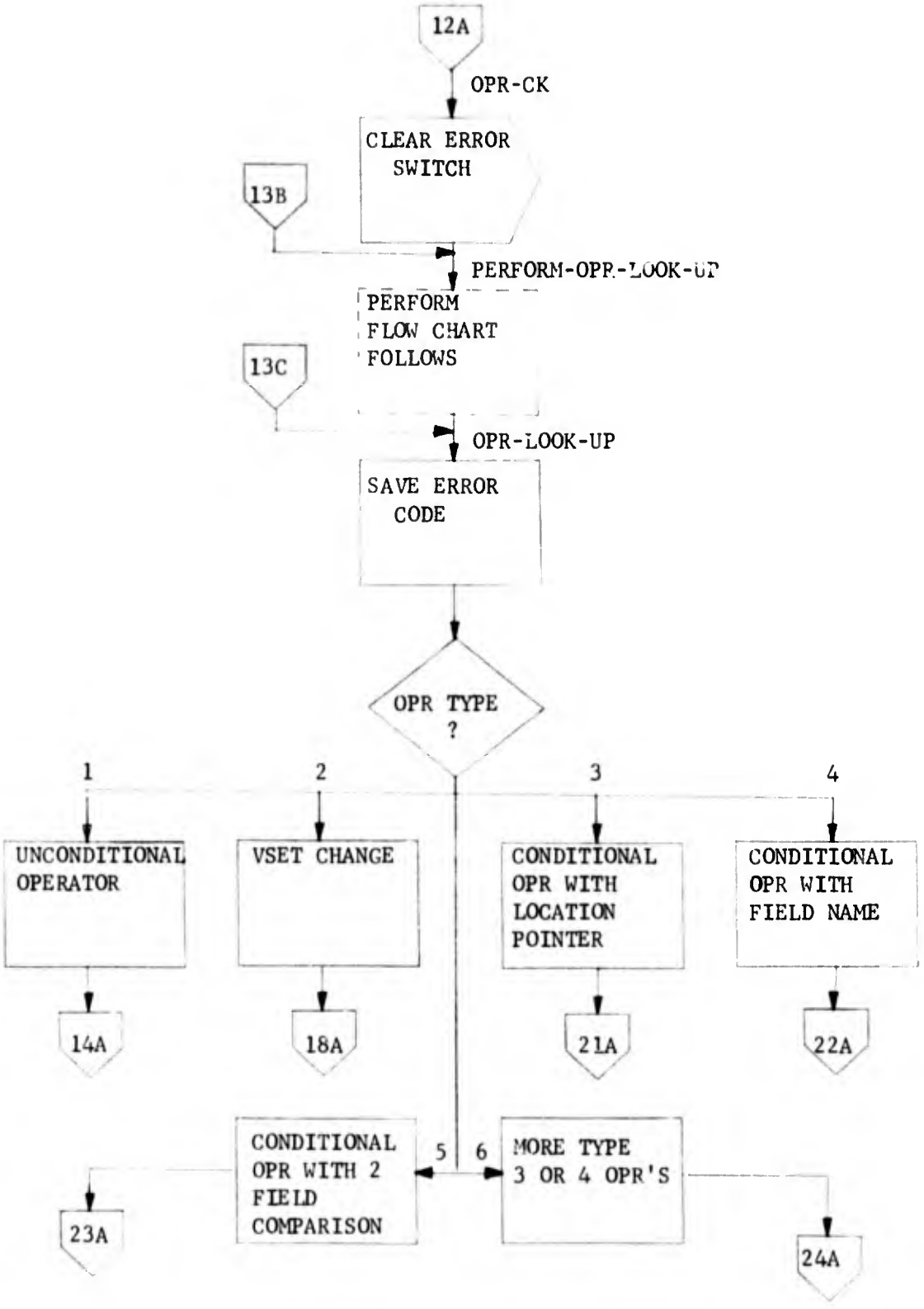

8A
VALID-TEST
EQUALA
HIGHA
IS ERROR CODE 'OR' ?
NO
YES
7C
8C SKIP-VAL-ERR-CHECK
AD
POINT TO NEXT VALIDITY CHECK
INCREMENT VALIDITY COUNTER
IS ERROR CODE 'OR' ?
NO
YES
IS ERROR CODE 'AND' ?
NO
YES
7C
CHEK-COUNT-MADE
END VALDITY CHECK ?
YES
8C
5A
8B
HIGHVALUE
CALL COMARAYS TO COMPARE DATA VALUE WITH HIGH RANGE VALUE
DATA VALUE > HIGH RANGE ?
NO
YES
8A
5A

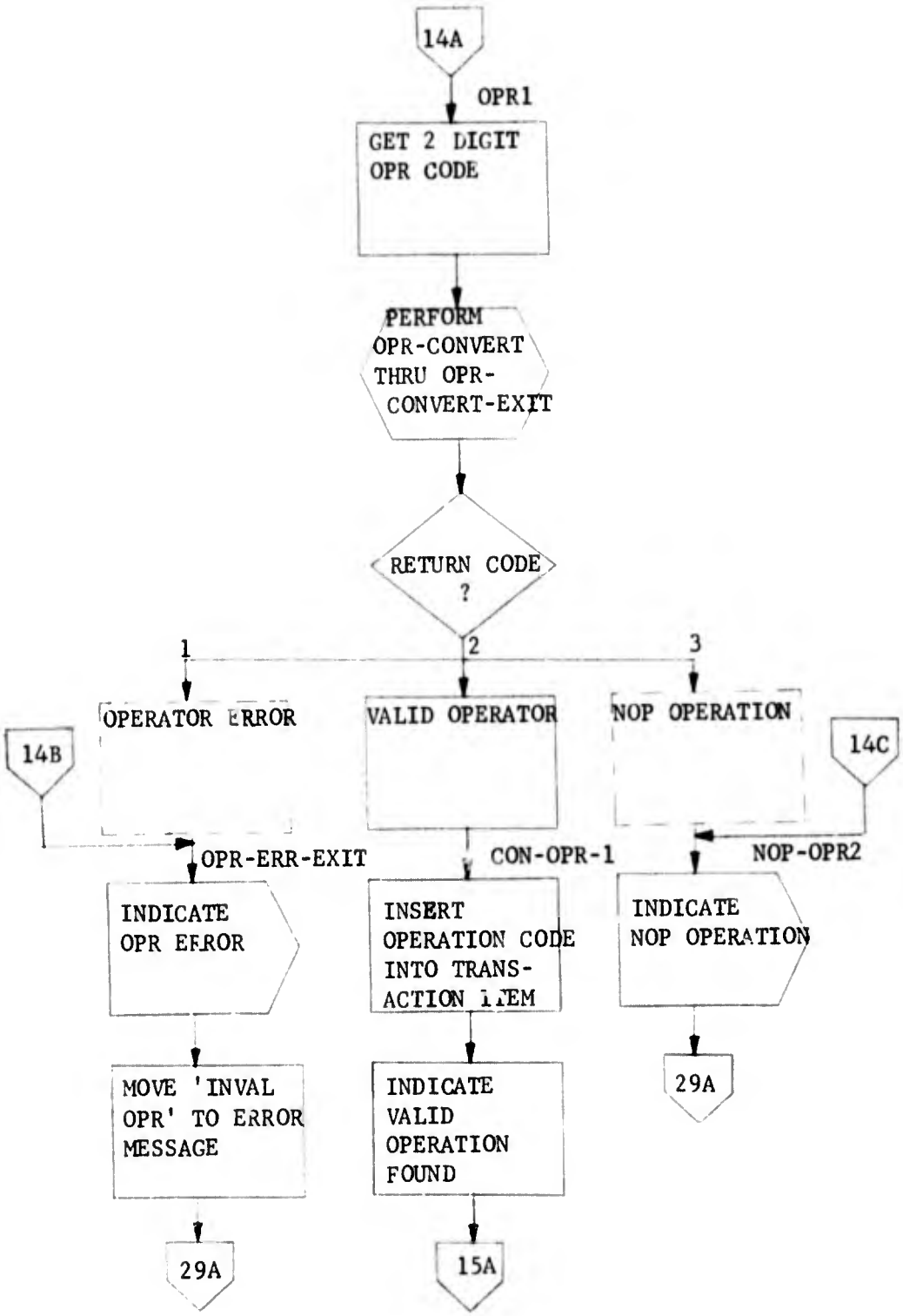

9A
PROFILE-CK
9B
PROFILE-ROUTINE
SAVE VAL LITERAL LENGTH
SAVE POINTER TO FIRST CHARACTER OF FIELD
INITIALIZE PROFILE LITERAL COUNTER
ADJUST VAL LITERAL POINTER
9D
PROCESS-INPUT
9C
TEST-PROFILE
USING BINARY VALUE OF DATA DETERMINE CHAR. TYPE
SET SWITCHES FOR VALID CHAR. TYPES BASED ON PROFILE CODE
SET UP PROFILE CODE
NO
END PROFILE CHECK?
YES
8A
NO
VALID CHARACTER ?
YES
VALID-CHARACTER
5A
ADJUST VAL LITERAL POINTER
ADJUST DATA POINTER
INCREMENT PROFILE LITERAL COUNTER
9D

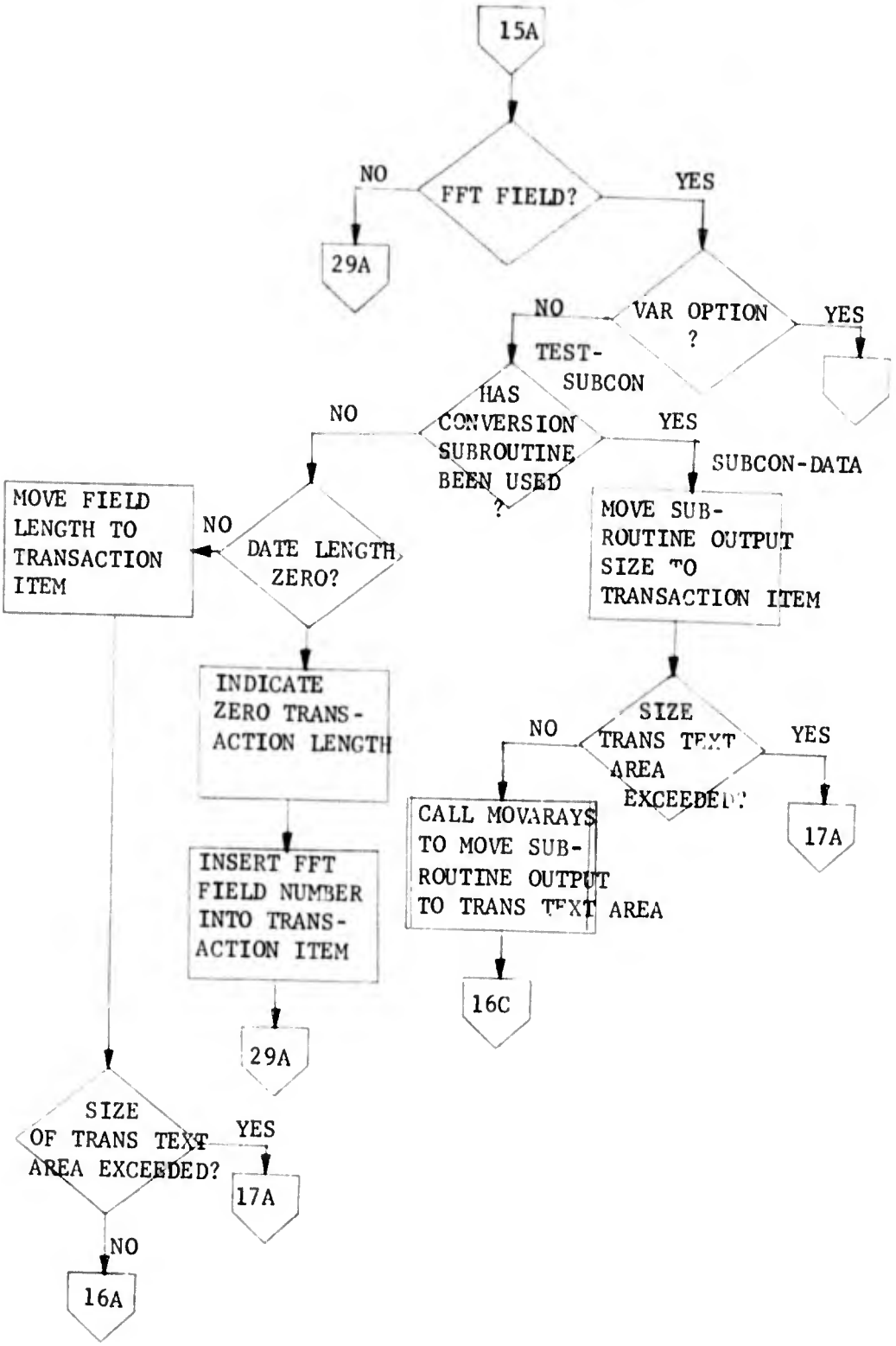

10A
VALUE-S
CALL MOVRAY TO OBTAIN HOP OF PARTIAL VALUE
DETERMINE LENGTH OF VALUE LITERAL
IS LENGTH ONE?
NO
YES
CALL COMRAYS TO COMPARE DATA VALUE
IS DATA CHAR. EQUAL TEST CHAR?
NO
YES
5A
8A
EQUAL?
NO
YES
5A
8A

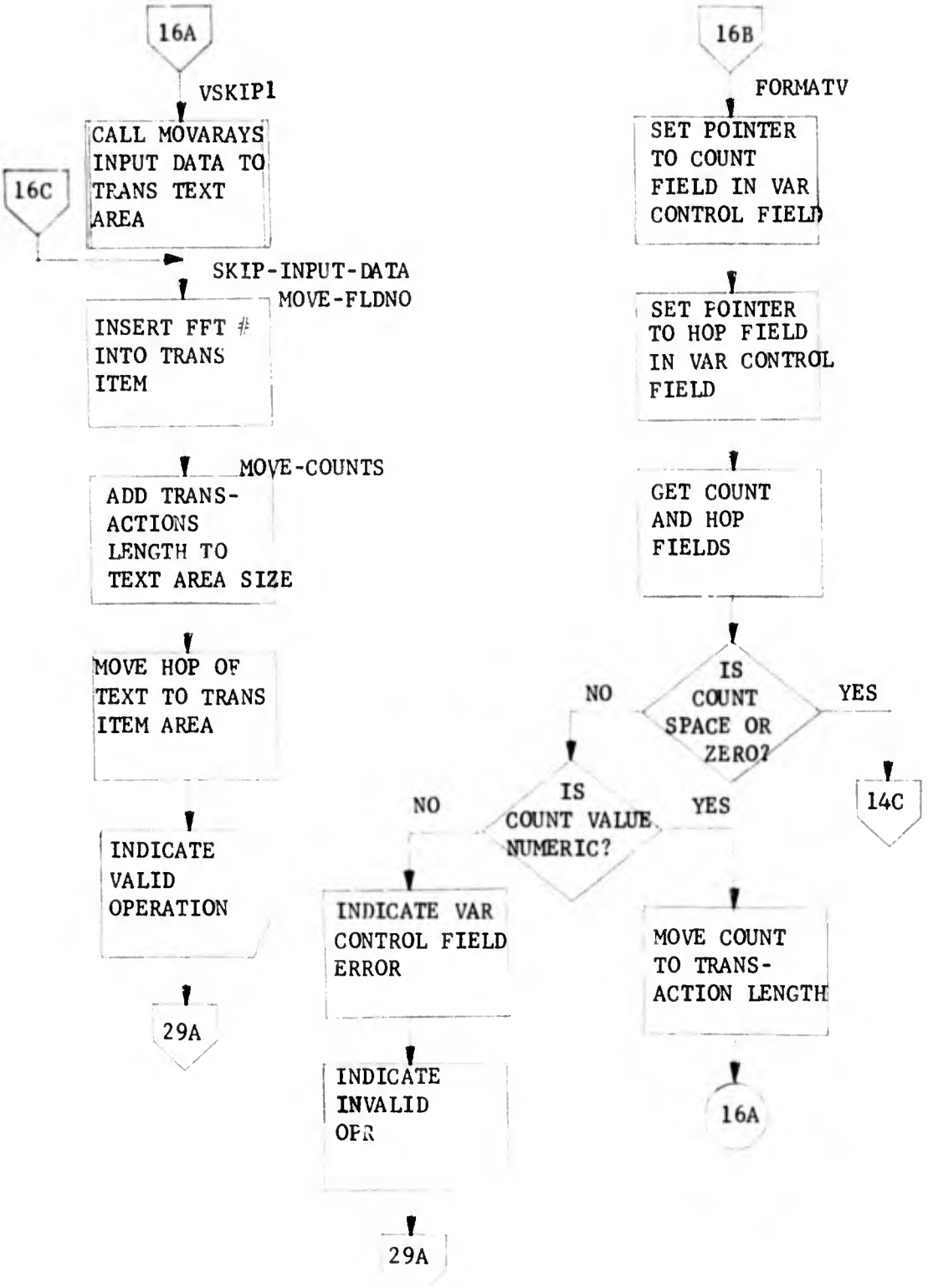

11A
SUBCON
CLEAR SUBROUTINE NAME AREA
RETURN-TO-SUBCON
FORMAT-SUBCON-LIT
GET SUBRT NAME
NO
IS SUBROUTINE LITERAL PRESENT?
YES
GET LITERAL LENGTH
MOVE DATA TO SUBRT INPUT AREA
MOVE LIT TO SUBRT INPUT AREA
11B
CALL-SUBCON
GET SUBRT NAME
NO
FFT FIELD?
YES
CLEAR FIELD AND FILE NAMES
SET UP FIELD AND FILE NAMES
MOVE INPUT DATA TO AFTER LITERAL
CALL SUBROUTINE
11B
SUBCON-CHECK-EXIT
EXIT PERFORM
12B

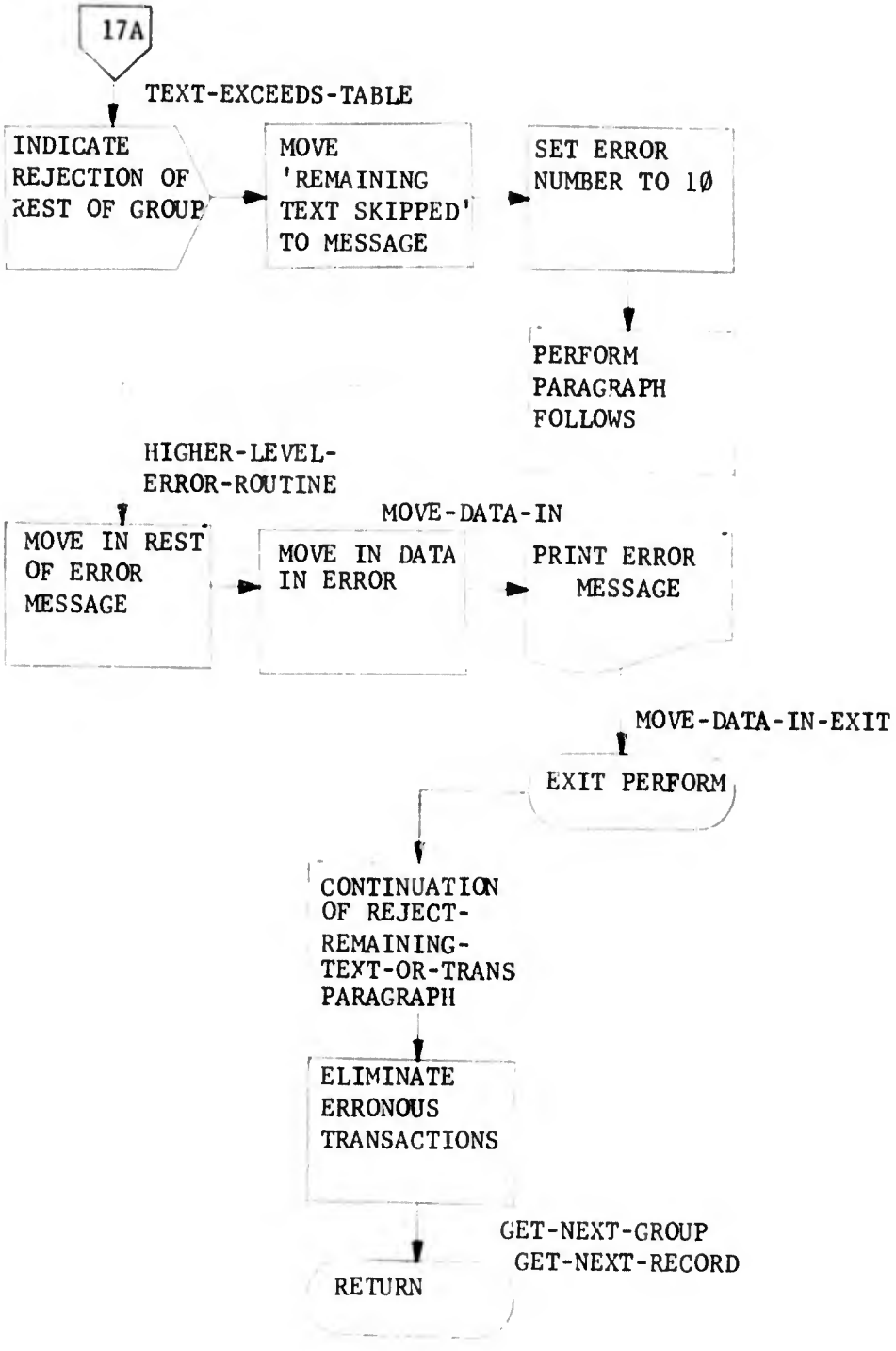

12A
SUBCHK
PERFORM SUBCON thru SUBCON-CHECK-EXIT (11)
IS DATA VALID
NO
YES
INDICATE ERROR
5A
8A
12B
SUBCON-ONLY
IS DATA VALID ?
YES
NO
IS OUTPUT SIZE Ø ?
INDICATE ERROR
5A
IS OUTPUT SIZE >FIELD SIZE?
INDICATE SUBROUTINE DATA VALID
8A

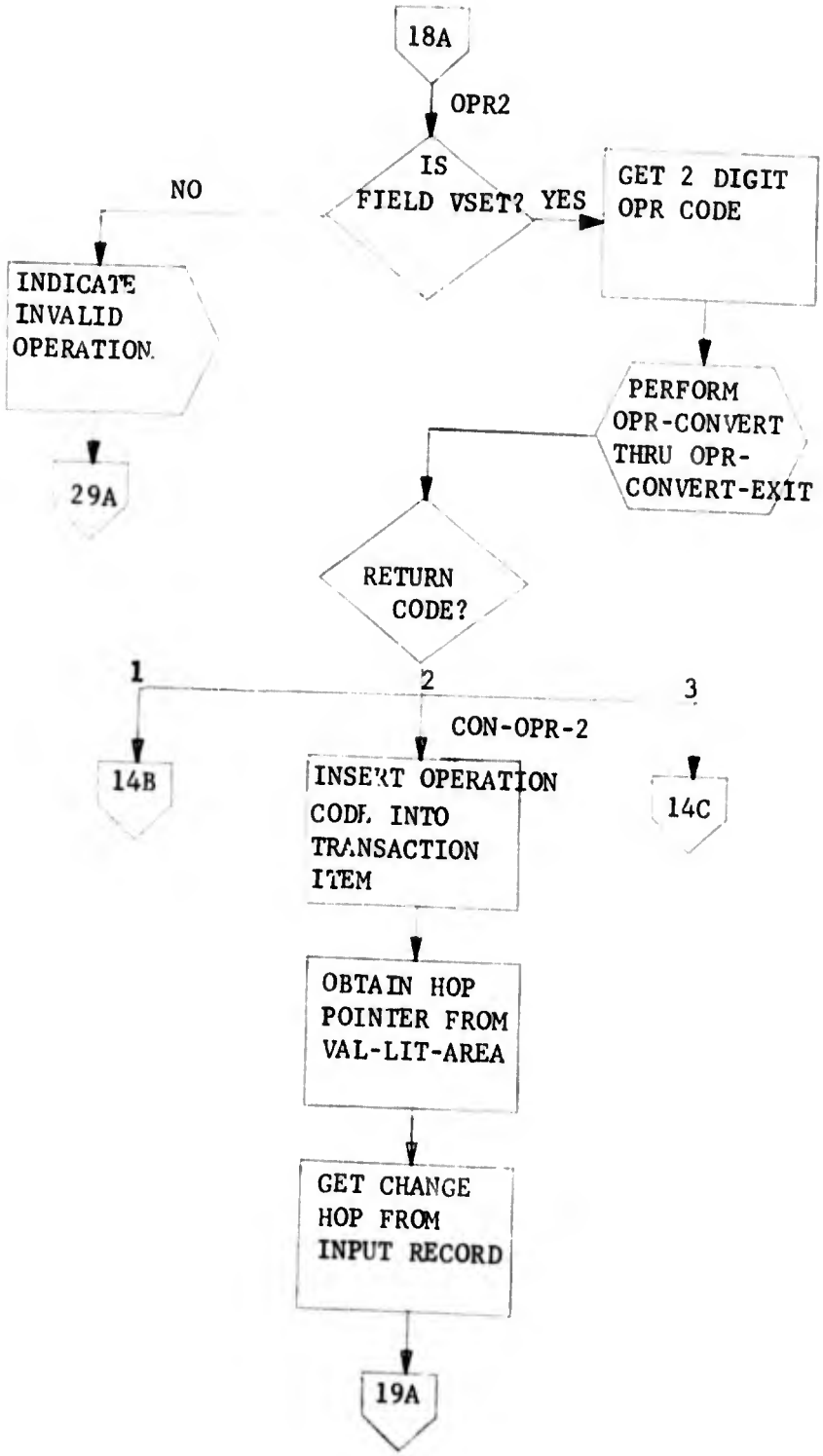

12A
OPR-CK
CLEAR ERROR SWITCH
13B
PERFORM-OPR-LOOK-UP
PERFORM FLOW CHART FOLLOWS
13C
OPR-LOOK-UP
SAVE ERROR CODE
OPR TYPE ?
1
2
3
4
UNCONDITIONAL OPERATOR
VSET CHANGE
CONDITIONAL OPR WITH LOCATION POINTER
CONDITIONAL OPR WITH FIELD NAME
14A
18A
21A
22A
CONDITIONAL OPR WITH 2 FIELD COMPARISON
5
6
MORE TYPE 3 OR 4 OPR'S
23A
24A

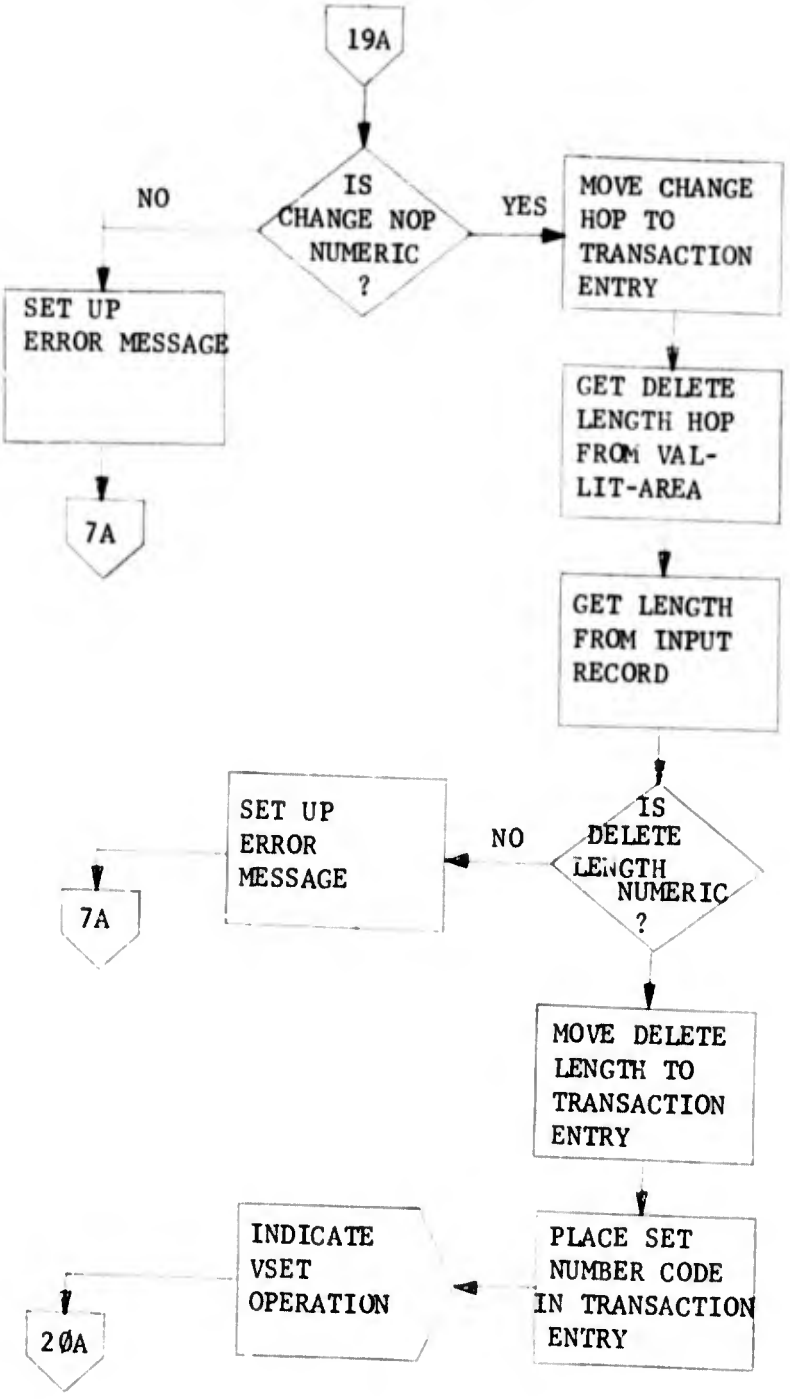

14A
OPR1
GET 2 DIGIT OPR CODE
PERFORM OPR-CONVERT THRU OPR-CONVERT-EXIT
RETURN CODE ?
1
2
3
OPERATOR ERROR
VALID OPERATOR
NOP OPERATION
14B
14C
OPR-ERR-EXIT
CON-OPR-1
NOP-OPR2
INDICATE OPR ERROR
INSERT OPERATION CODE INTO TRANS-ACTION ITEM
INDICATE NOP OPERATION
MOVE 'INVAL OPR' TO ERROR MESSAGE
INDICATE VALID OPERATION FOUND
29A
29A
15A

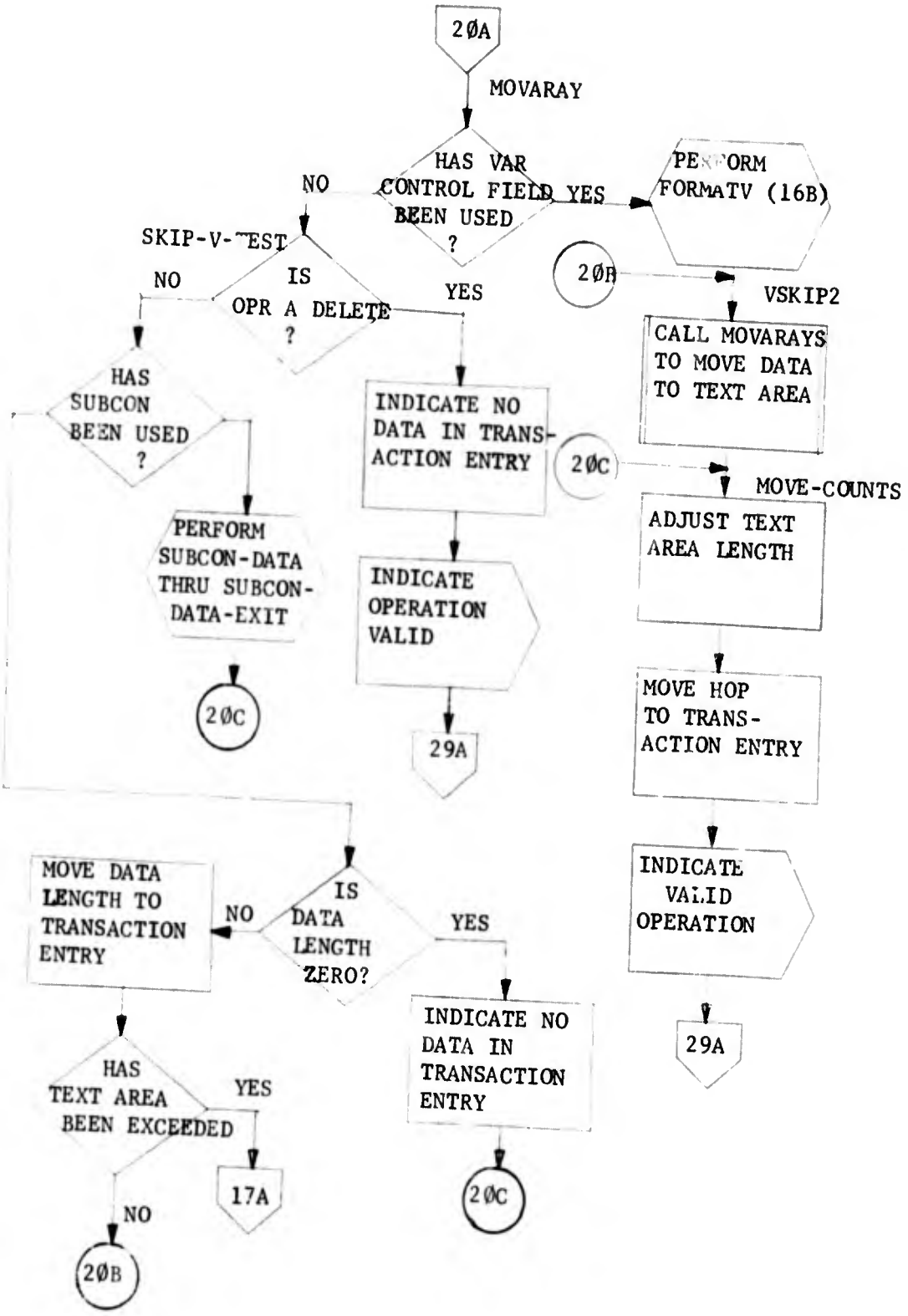

15A
FFT FIELD?
NO
YES
29A
VAR OPTION ?
NO
YES
TEST-SUBCON
HAS CONVERSION SUBROUTINE BEEN USED ?
NO
YES
SUBCON-DATA
MOVE FIELD LENGTH TO TRANSACTION ITEM
NO
DATE LENGTH ZERO?
MOVE SUB-ROUTINE OUTPUT SIZE TO TRANSACTION ITEM
INDICATE ZERO TRANS-ACTION LENGTH
SIZE TRANS TEXT AREA EXCEEDED?
NO
YES
17A
CALL MOVARAYS TO MOVE SUB-ROUTINE OUTPUT TO TRANS TEXT AREA
INSERT FFT FIELD NUMBER INTO TRANS-ACTION ITEM
16C
29A
SIZE OF TRANS TEXT AREA EXCEEDED?
YES
17A
NO
16A

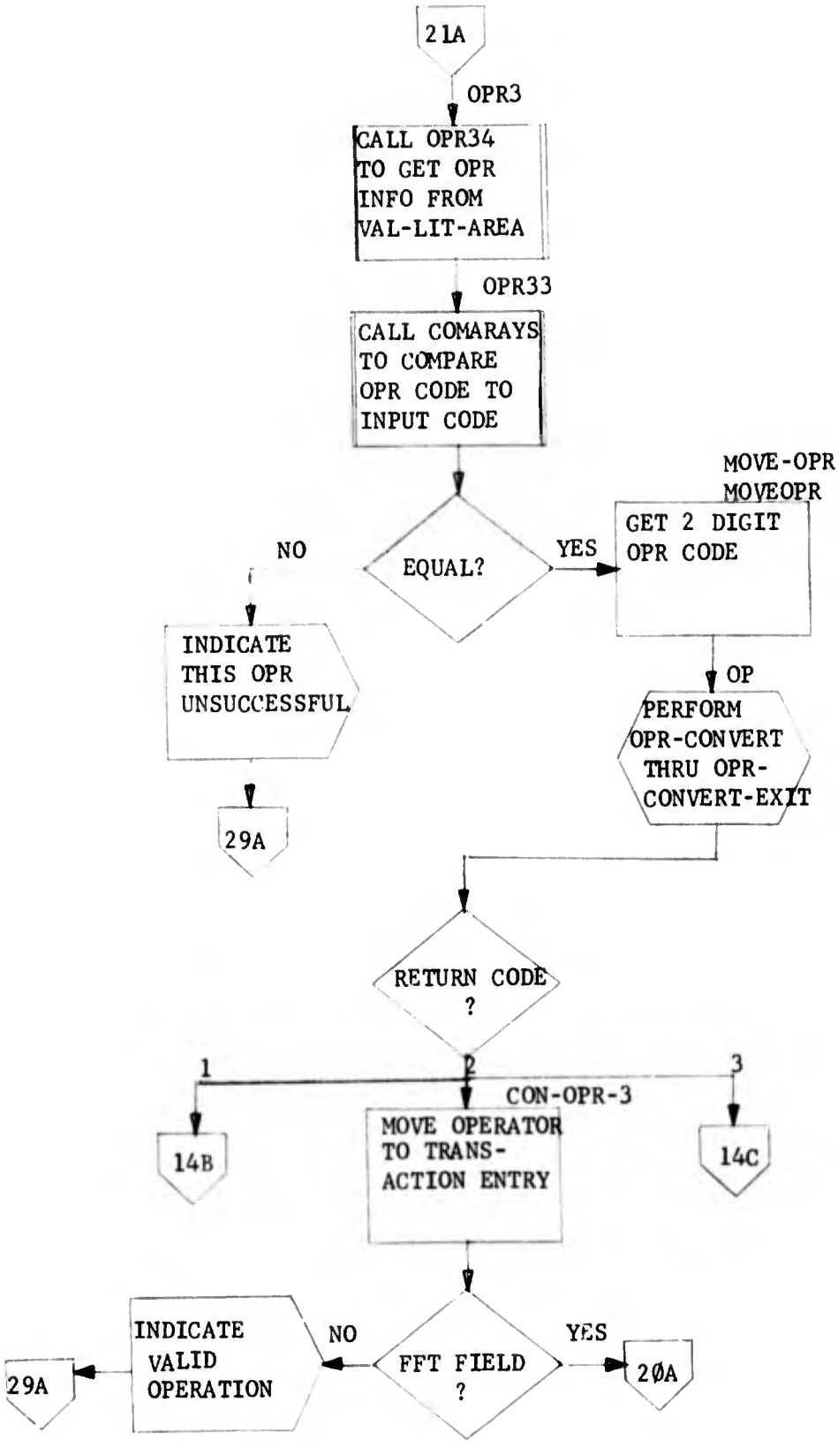

16A
VSKIP1
CALL MOVARAYS INPUT DATA TO TRANS TEXT AREA
16C
SKIP-INPUT-DATA
MOVE-FLDNO
INSERT FFT # INTO TRANS ITEM
MOVE-COUNTS
ADD TRANS-ACTIONS LENGTH TO TEXT AREA SIZE
MOVE HOP OF TEXT TO TRANS ITEM AREA
INDICATE VALID OPERATION
29A
16B
FORMATV
SET POINTER TO COUNT FIELD IN VAR CONTROL FIELD
SET POINTER TO HOP FIELD IN VAR CONTROL FIELD
GET COUNT AND HOP FIELDS
IS COUNT SPACE OR ZERO?
NO
YES
14C
IS COUNT VALUE NUMERIC?
NO
YES
INDICATE VAR CONTROL FIELD ERROR
INDICATE INVALID OPR
29A
MOVE COUNT TO TRANS-ACTION LENGTH
16A

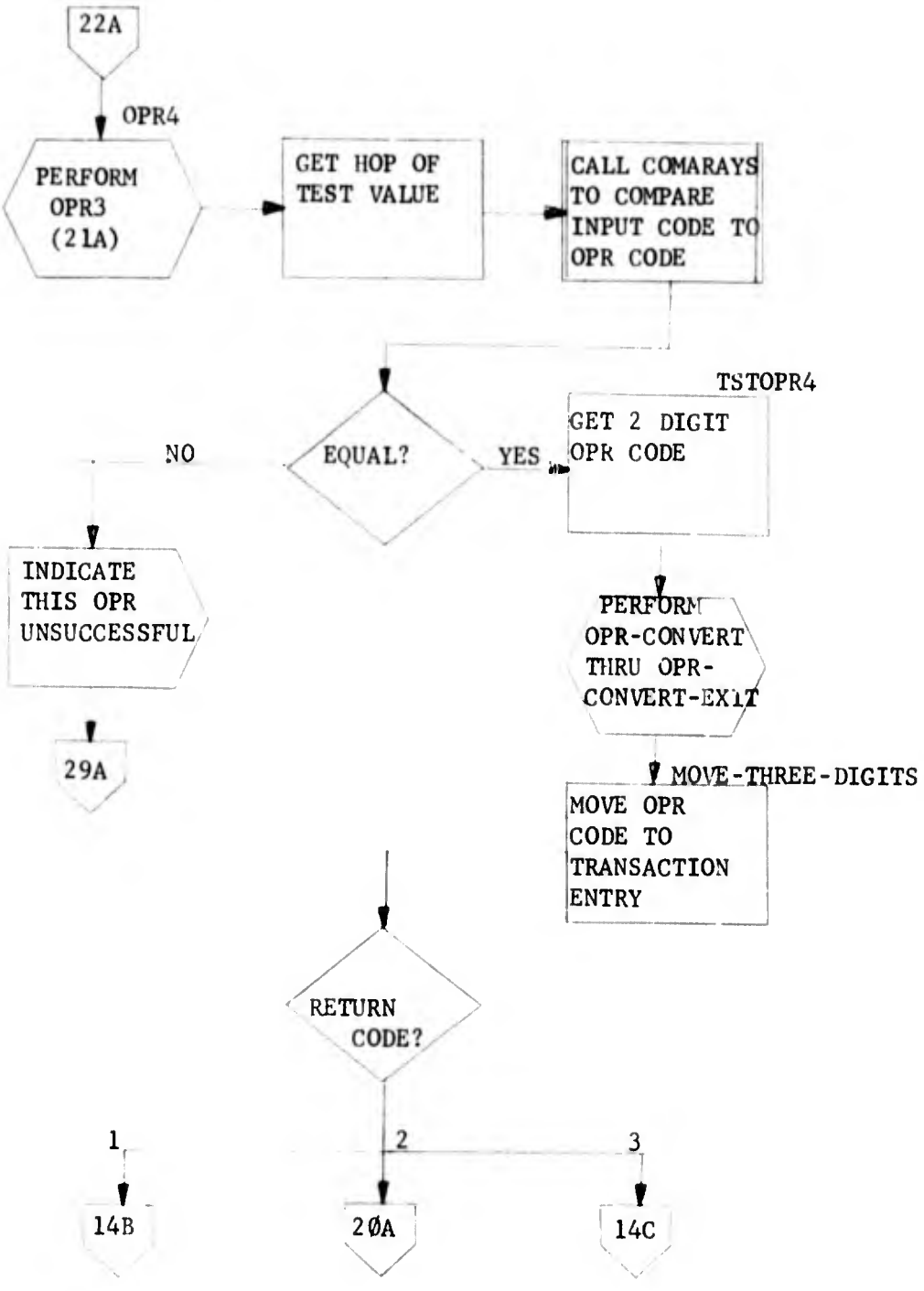

17A
TEXT-EXCEEDS-TABLE
INDICATE REJECTION OF REST OF GROUP
MOVE 'REMAINING TEXT SKIPPED' TO MESSAGE
SET ERROR NUMBER TO 1Ø
PERFORM PARAGRAPH FOLLOWS
HIGHER-LEVEL-ERROR-ROUTINE
MOVE IN REST OF ERROR MESSAGE
MOVE-DATA-IN
MOVE IN DATA IN ERROR
PRINT ERROR MESSAGE
MOVE-DATA-IN-EXIT
EXIT PERFORM
CONTINUATION OF REJECT-REMAINING-TEXT-OR-TRANS PARAGRAPH
ELIMINATE ERRONOUS TRANSACTIONS
GET-NEXT-GROUP
GET-NEXT-RECORD
RETURN

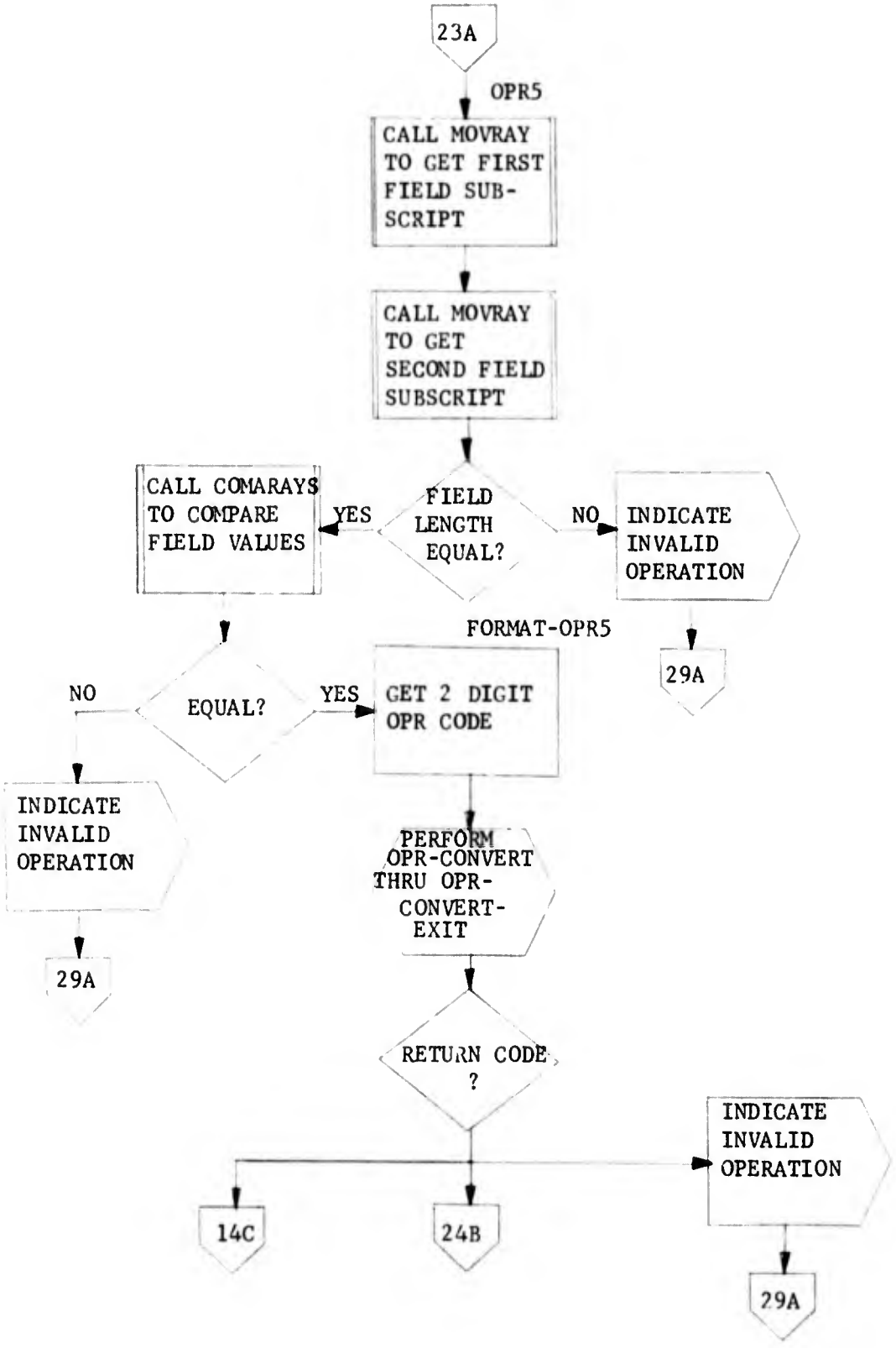

18A
OPR2
IS FIELD VSET?
NO
YES
GET 2 DIGIT OPR CODE
INDICATE INVALID OPERATION
29A
PERFORM OPR-CONVERT THRU OPR-CONVERT-EXIT
RETURN CODE?
1
2
3
CON-OPR-2
14B
INSERT OPERATION CODE INTO TRANSACTION ITEM
14C
OBTAIN HOP POINTER FROM VAL-LIT-AREA
GET CHANGE HOP FROM INPUT RECORD
19A

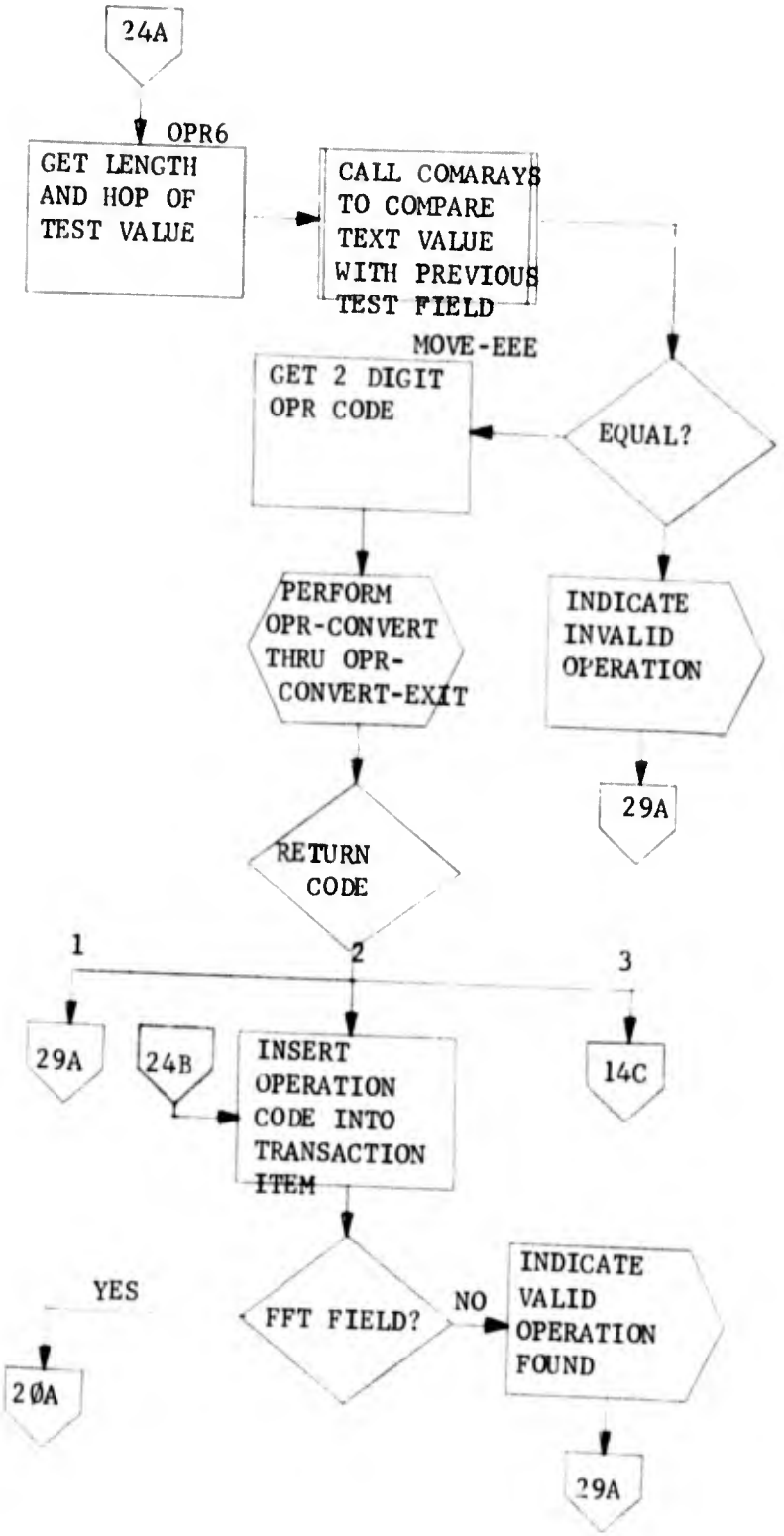

19A
IS CHANGE NOP NUMERIC ?
NO
YES
MOVE CHANGE HOP TO TRANSACTION ENTRY
SET UP ERROR MESSAGE
7A
GET DELETE LENGTH HOP FROM VAL-LIT-AREA
GET LENGTH FROM INPUT RECORD
IS DELETE LENGTH NUMERIC ?
NO
SET UP ERROR MESSAGE
7A
MOVE DELETE LENGTH TO TRANSACTION ENTRY
PLACE SET NUMBER CODE IN TRANSACTION ENTRY
INDICATE VSET OPERATION
2ØA

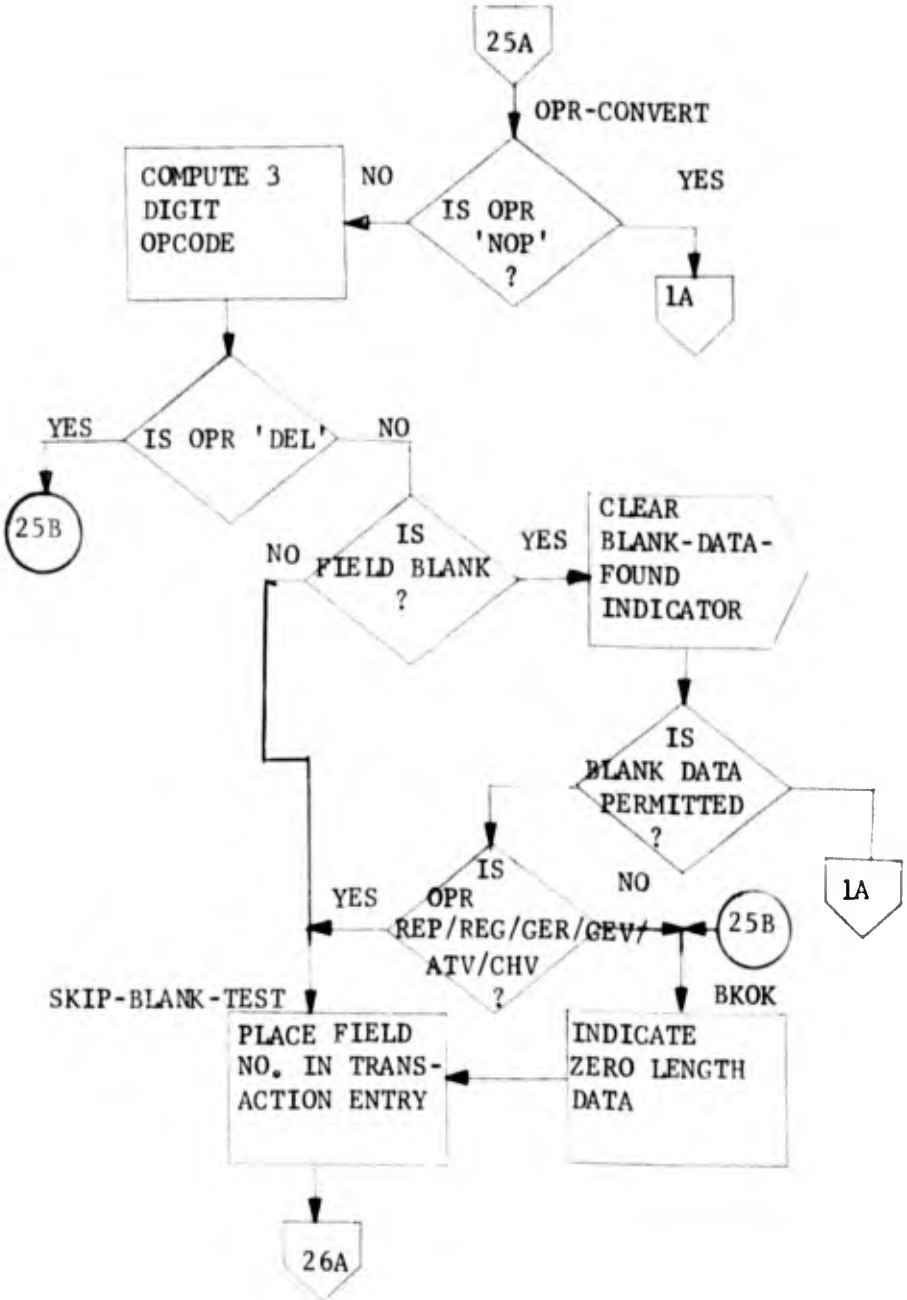

2ØA
MOVARAY
HAS VAR CONTROL FIELD BEEN USED ?
YES
PERFORM FORMATV (16B)
NO
SKIP-V-TEST
IS OPR A DELETE ?
NO
YES
2ØB
VSKIP2
CALL MOVARAYS TO MOVE DATA TO TEXT AREA
HAS SUBCON BEEN USED ?
INDICATE NO DATA IN TRANS-ACTION ENTRY
2ØC
MOVE-COUNTS
ADJUST TEXT AREA LENGTH
PERFORM SUBCON-DATA THRU SUBCON-DATA-EXIT
INDICATE OPERATION VALID
MOVE HOP TO TRANS-ACTION ENTRY
2ØC
29A
INDICATE VALID OPERATION
MOVE DATA LENGTH TO TRANSACTION ENTRY
NO
IS DATA LENGTH ZERO?
YES
29A
INDICATE NO DATA IN TRANSACTION ENTRY
HAS TEXT AREA BEEN EXCEEDED
YES
17A
2ØC
NO
2ØB

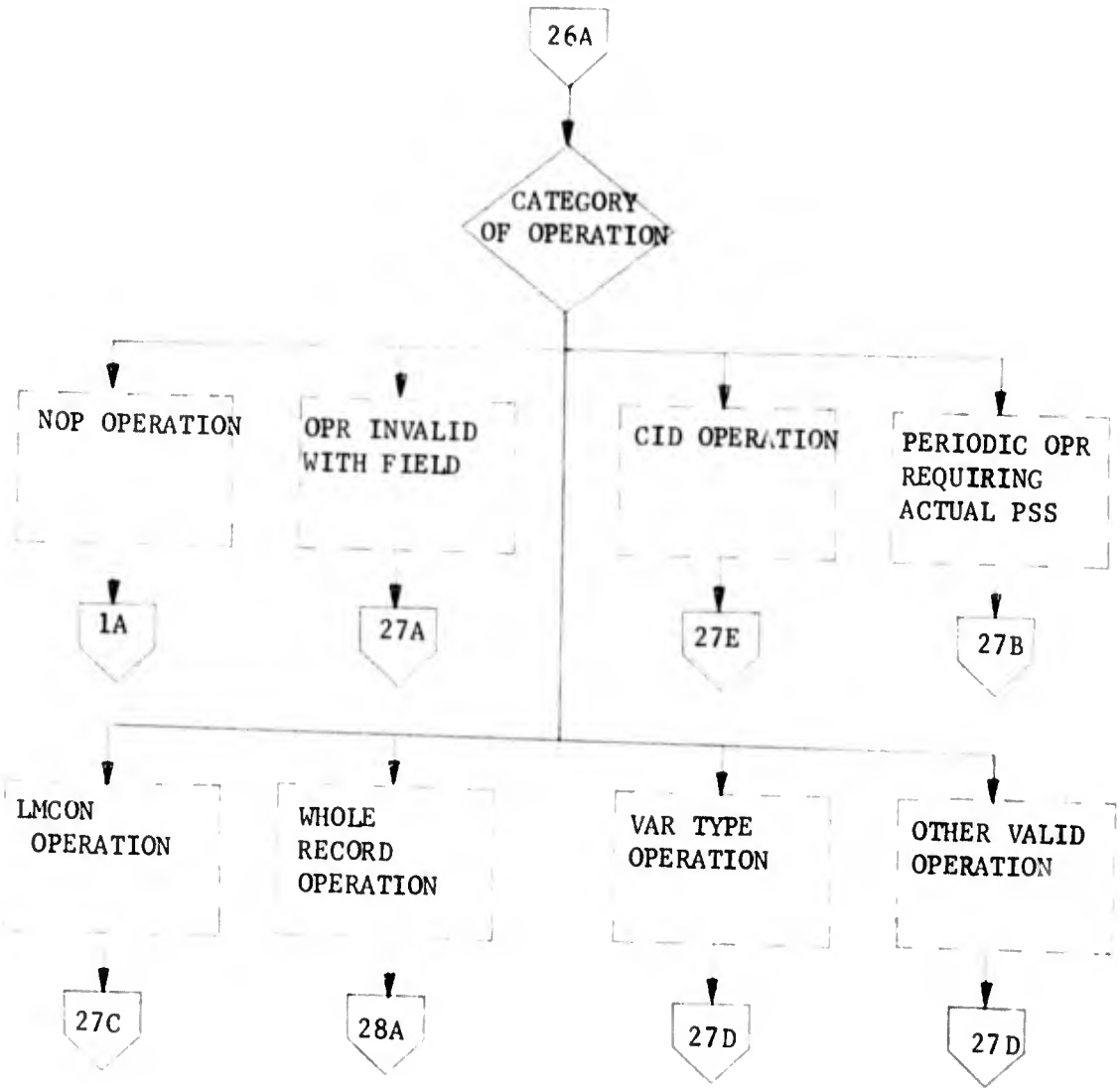

21A
OPR3
CALL OPR34 TO GET OPR INFO FROM VAL-LIT-AREA
OPR33
CALL COMARAYS TO COMPARE OPR CODE TO INPUT CODE
EQUAL?
NO
YES
MOVE-OPR
MOVEOPR
GET 2 DIGIT OPR CODE
INDICATE THIS OPR UNSUCCESSFUL
29A
OP
PERFORM OPR-CONVERT THRU OPR-CONVERT-EXIT
RETURN CODE ?
1
2
3
CON-OPR-3
14B
MOVE OPERATOR TO TRANS-ACTION ENTRY
14C
INDICATE VALID OPERATION
NO
FFT FIELD ?
YES
29A
20A

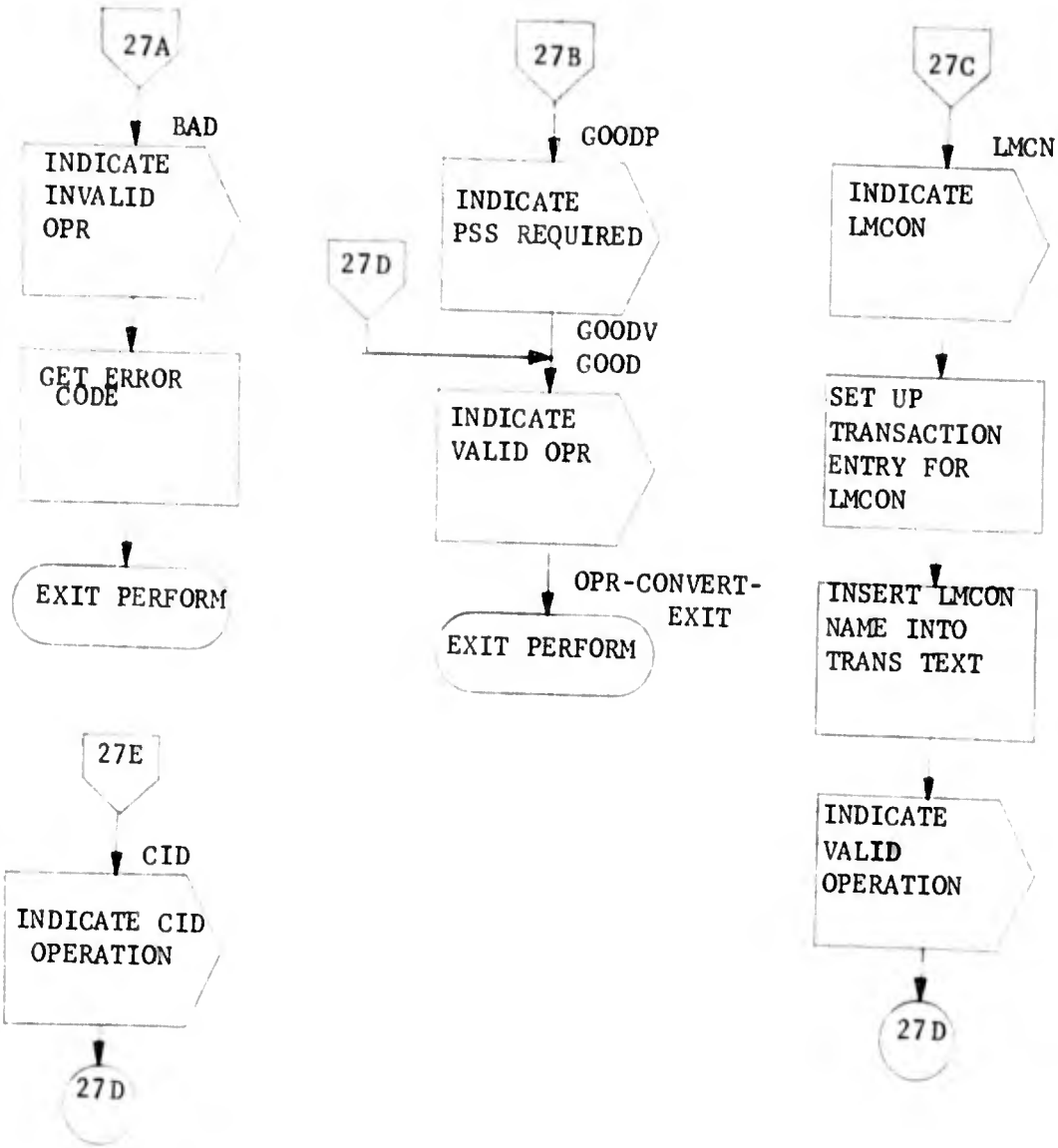

22A
OPR4
PERFORM OPR3 (21A)
GET HOP OF TEST VALUE
CALL COMARAYS TO COMPARE INPUT CODE TO OPR CODE
TSTOPR4
EQUAL?
NO
YES
GET 2 DIGIT OPR CODE
INDICATE THIS OPR UNSUCCESSFUL
29A
PERFORM OPR-CONVERT THRU OPR-CONVERT-EXIT
MOVE-THREE-DIGITS
MOVE OPR CODE TO TRANSACTION ENTRY
RETURN CODE?
1
2
3
14B
20A
14C

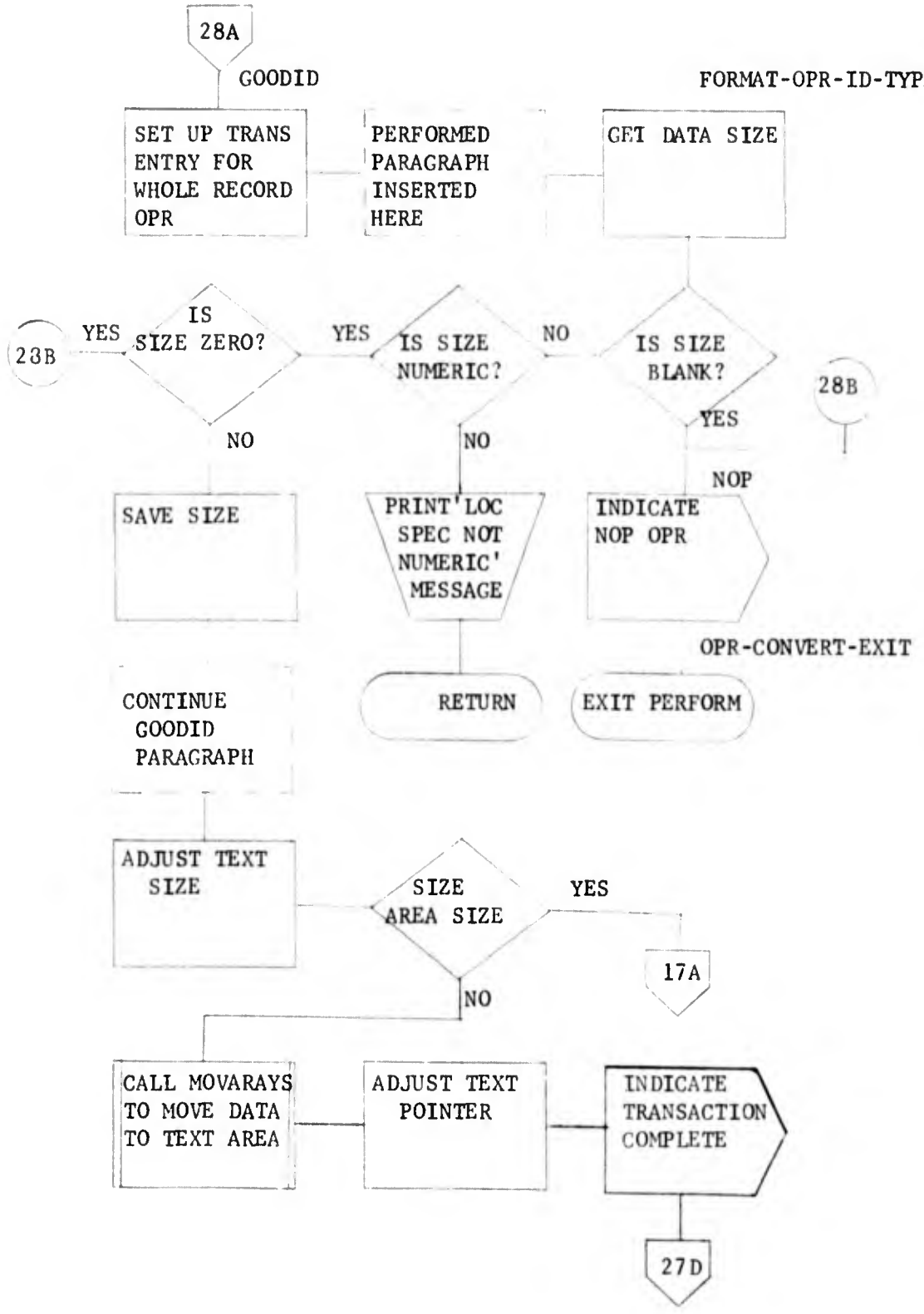

23A
OPR5
CALL MOVRAY TO GET FIRST FIELD SUB-SCRIPT
CALL MOVRAY TO GET SECOND FIELD SUBSCRIPT
FIELD LENGTH EQUAL?
YES
NO
CALL COMARAYS TO COMPARE FIELD VALUES
INDICATE INVALID OPERATION
29A
FORMAT-OPR5
EQUAL?
NO
YES
GET 2 DIGIT OPR CODE
INDICATE INVALID OPERATION
29A
PERFORM OPR-CONVERT THRU OPR-CONVERT-EXIT
RETURN CODE ?
INDICATE INVALID OPERATION
14C
24B
29A

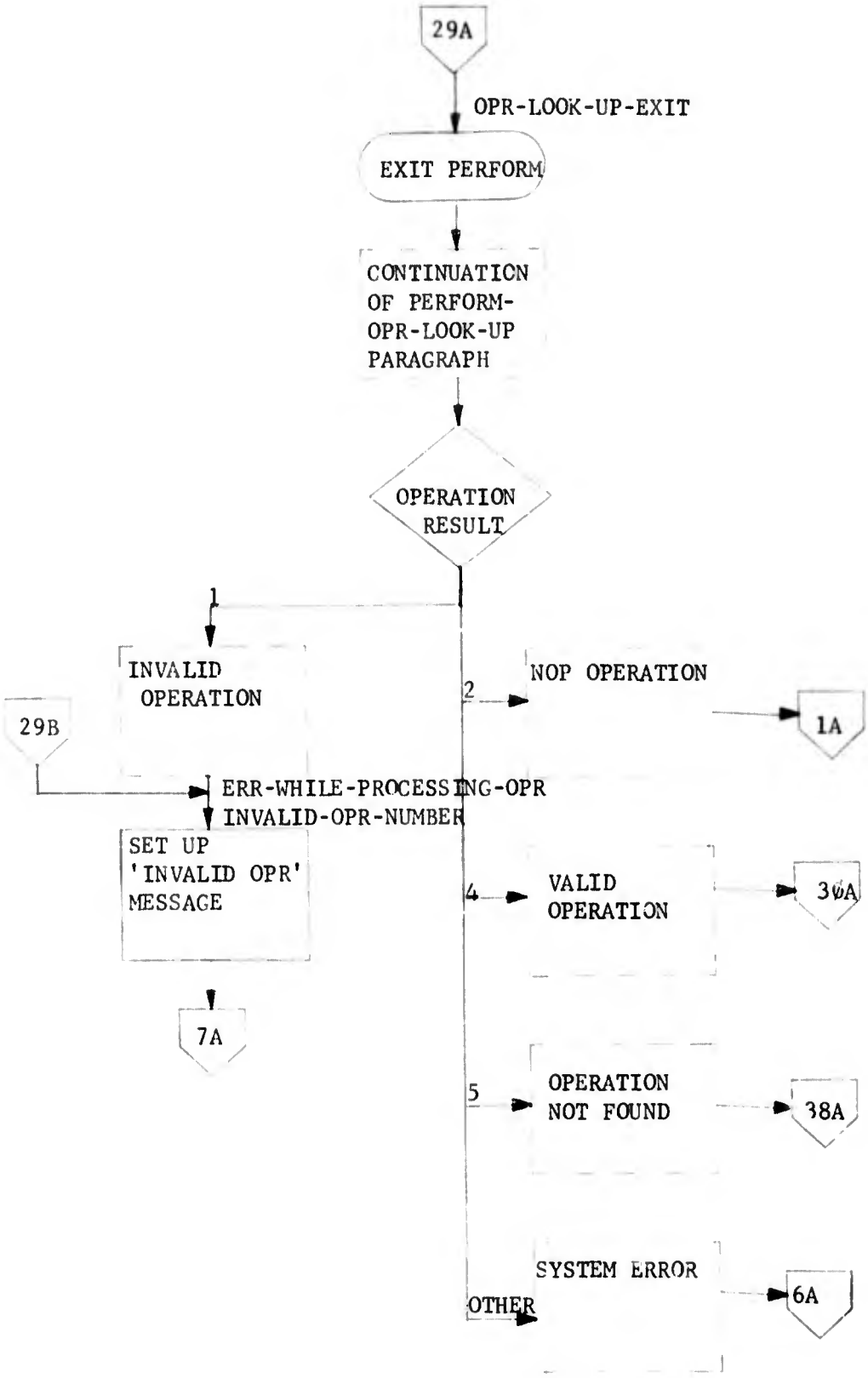

24A
OPR6
GET LENGTH AND HOP OF TEST VALUE
CALL COMARAYS TO COMPARE TEXT VALUE WITH PREVIOUS TEST FIELD
MOVE-EEE
GET 2 DIGIT OPR CODE
EQUAL?
PERFORM OPR-CONVERT THRU OPR-CONVERT-EXIT
INDICATE INVALID OPERATION
29A
RETURN CODE
1
2
3
29A
24B
INSERT OPERATION CODE INTO TRANSACTION ITEM
14C
YES
FFT FIELD?
NO
INDICATE VALID OPERATION FOUND
20A
29A

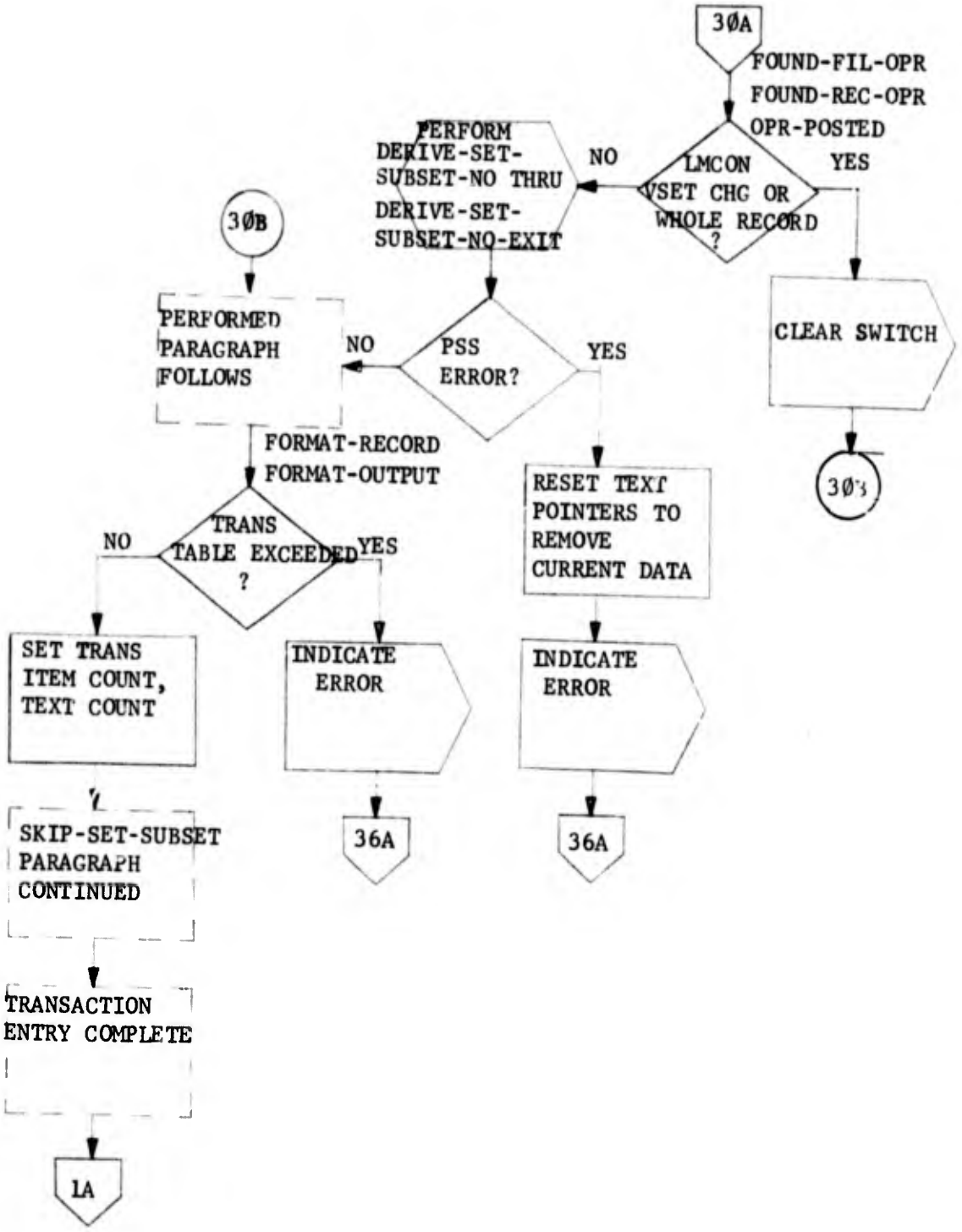

25A
OPR-CONVERT
IS OPR 'NOP' ?
NO
YES
COMPUTE 3 DIGIT OPCODE
1A
IS OPR 'DEL'
YES
NO
25B
IS FIELD BLANK ?
NO
YES
CLEAR BLANK-DATA-FOUND INDICATOR
IS BLANK DATA PERMITTED ?
1A
IS OPR REP/REG/GER/GEV/ ATV/CHV ?
YES
NO
25B
SKIP-BLANK-TEST
BKOK
PLACE FIELD NO. IN TRANS-ACTION ENTRY
INDICATE ZERO LENGTH DATA
26A

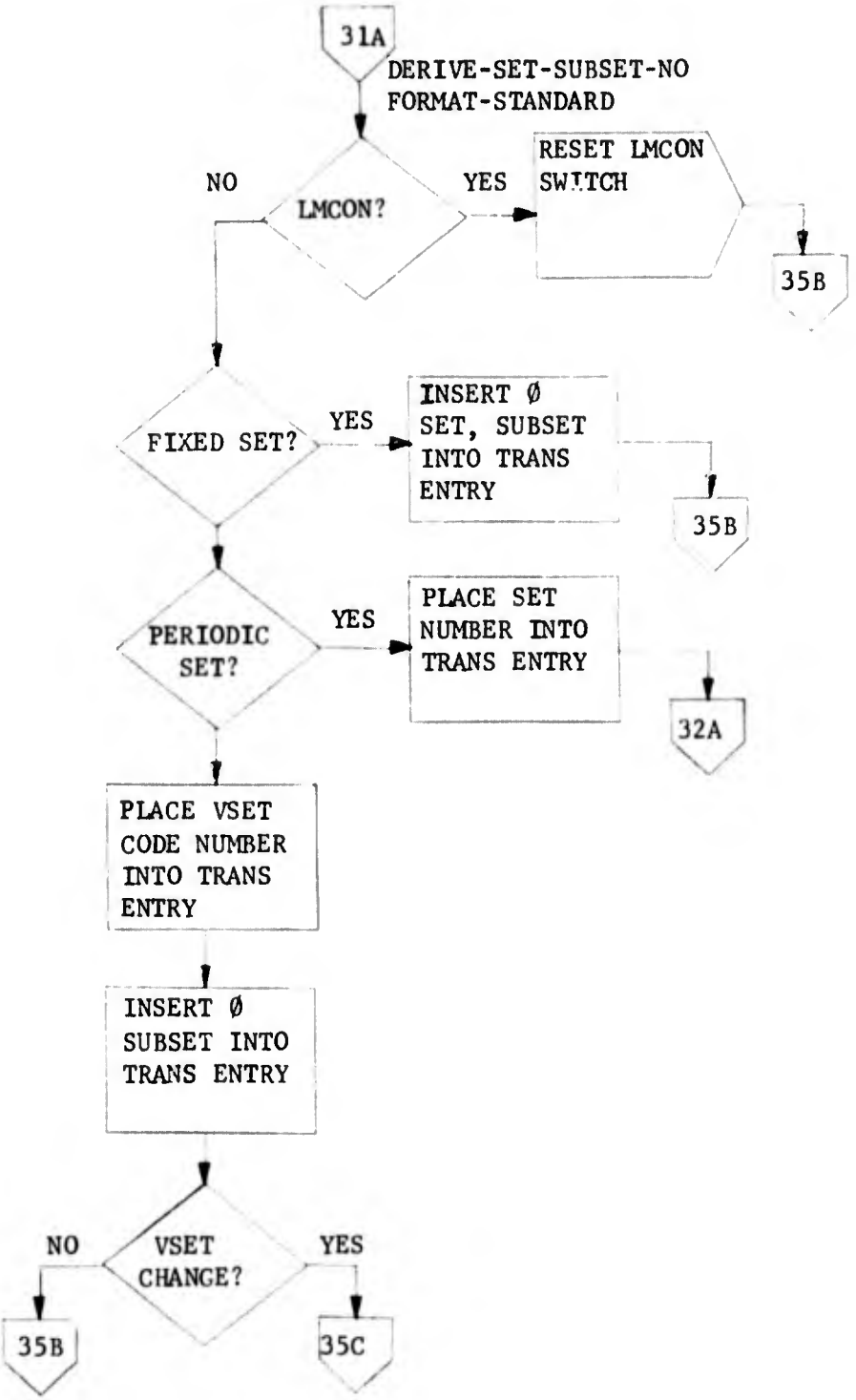

26A
CATEGORY OF OPERATION
NOP OPERATION
OPR INVALID WITH FIELD
CID OPERATION
PERIODIC OPR REQUIRING ACTUAL PSS
1A
27A
27E
27B
LMCON OPERATION
WHOLE RECORD OPERATION
VAR TYPE OPERATION
OTHER VALID OPERATION
27C
28A
27D
27D

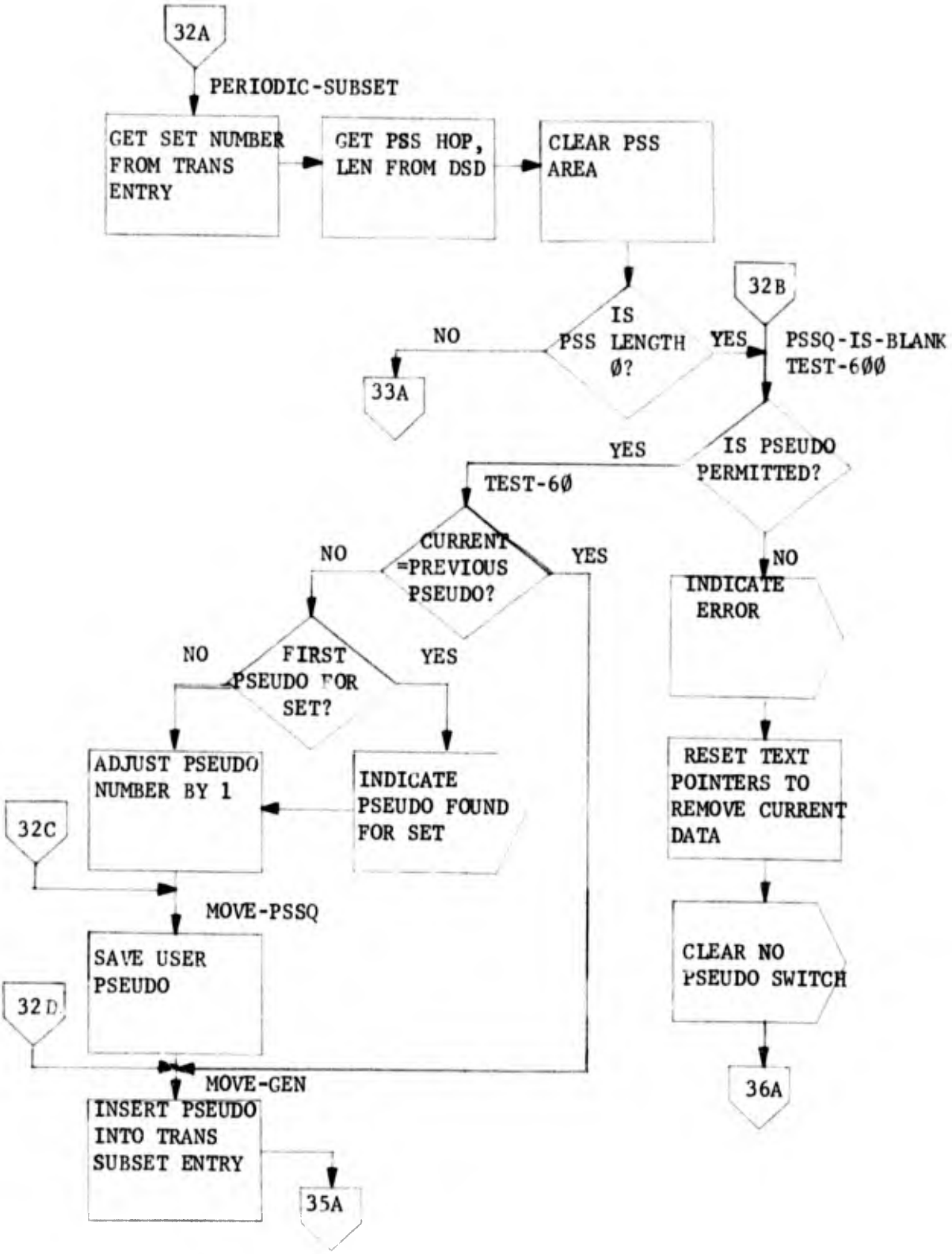

27A
BAD
INDICATE INVALID OPR
GET ERROR CODE
EXIT PERFORM
27B
GOODP
INDICATE PSS REQUIRED
27D
GOODV
GOOD
INDICATE VALID OPR
OPR-CONVERT-EXIT
EXIT PERFORM
27C
LMCN
INDICATE LMCON
SET UP TRANSACTION ENTRY FOR LMCON
INSERT LMCON NAME INTO TRANS TEXT
INDICATE VALID OPERATION
27D
27E
CID
INDICATE CID OPERATION
27D

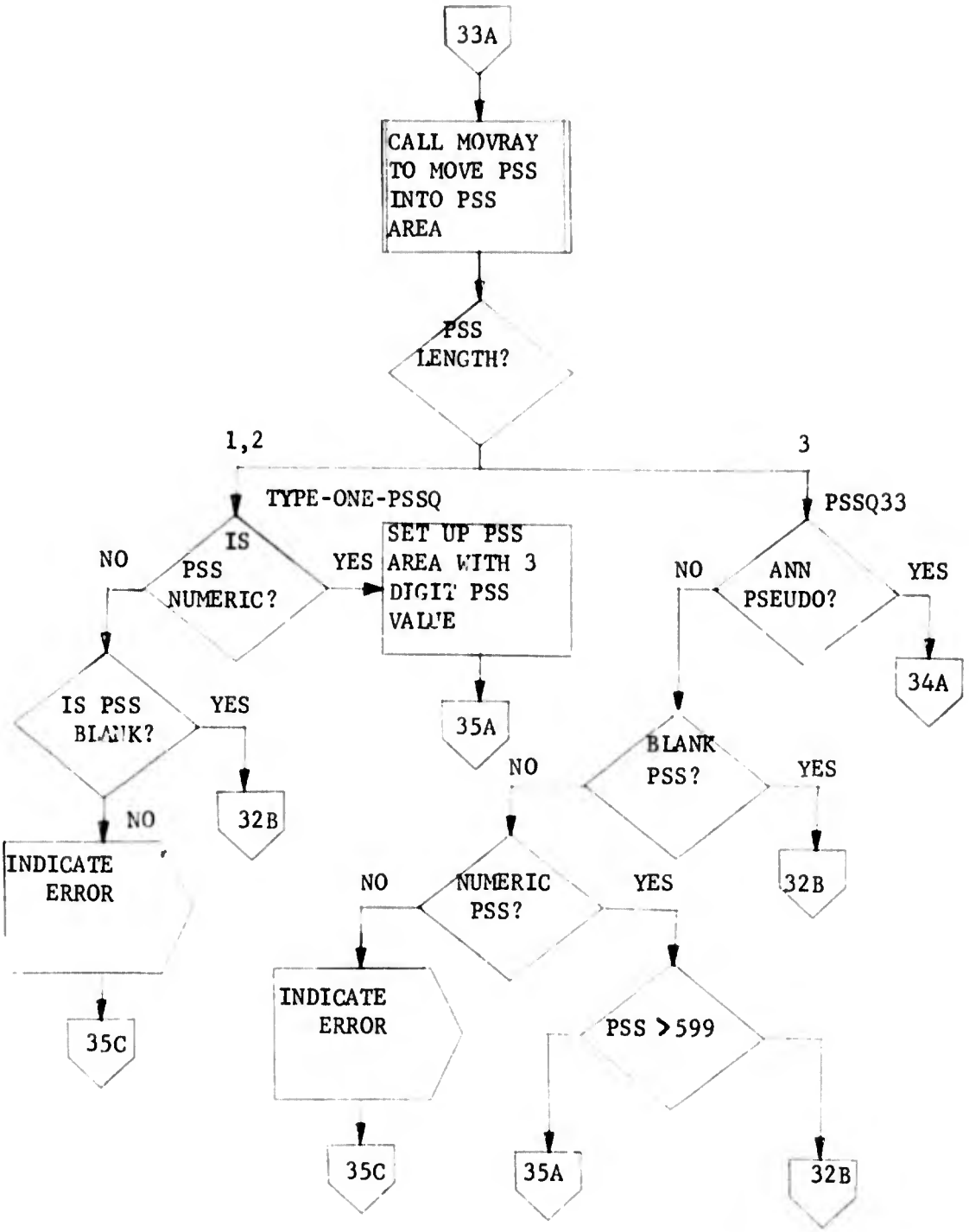

28A
GOODID
FORMAT-OPR-ID-TYPE
SET UP TRANS ENTRY FOR WHOLE RECORD OPR
PERFORMED PARAGRAPH INSERTED HERE
GET DATA SIZE
28B
YES
IS SIZE ZERO?
YES
IS SIZE NUMERIC?
NO
IS SIZE BLANK?
28B
YES
NO
NO
NOP
SAVE SIZE
PRINT 'LOC SPEC NOT NUMERIC' MESSAGE
INDICATE NOP OPR
OPR-CONVERT-EXIT
CONTINUE GOODID PARAGRAPH
RETURN
EXIT PERFORM
ADJUST TEXT SIZE
SIZE AREA SIZE
YES
17A
NO
CALL MOVARAYS TO MOVE DATA TO TEXT AREA
ADJUST TEXT POINTER
INDICATE TRANSACTION COMPLETE
27D

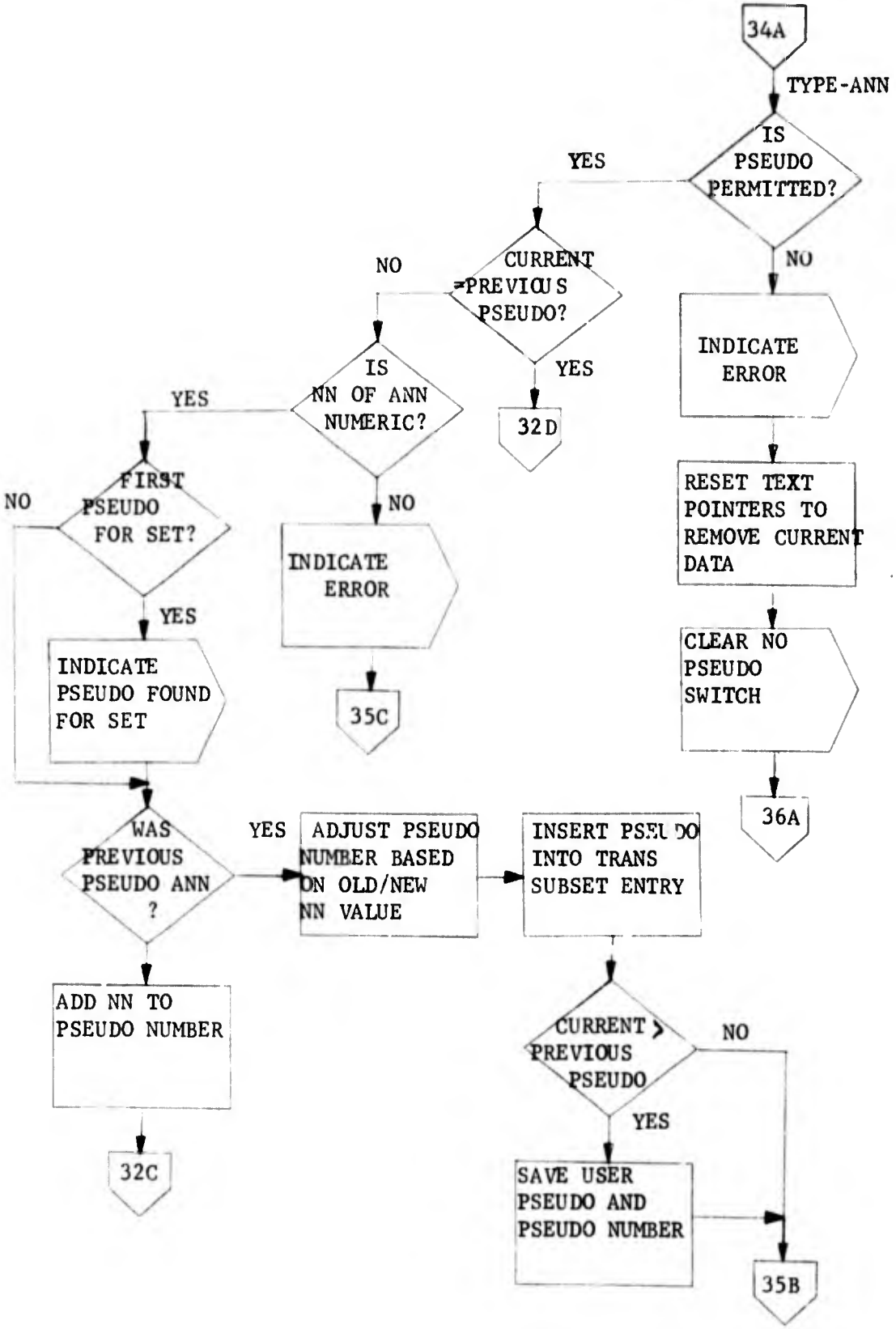

29A
OPR-LOOK-UP-EXIT
EXIT PERFORM
CONTINUATION OF PERFORM-OPR-LOOK-UP PARAGRAPH
OPERATION RESULT
1
INVALID OPERATION
29B
ERR-WHILE-PROCESSING-OPR
INVALID-OPR-NUMBER
SET UP 'INVALID OPR' MESSAGE
7A
2
NOP OPERATION
1A
4
VALID OPERATION
3ØA
5
OPERATION NOT FOUND
38A
SYSTEM ERROR
OTHER
6A

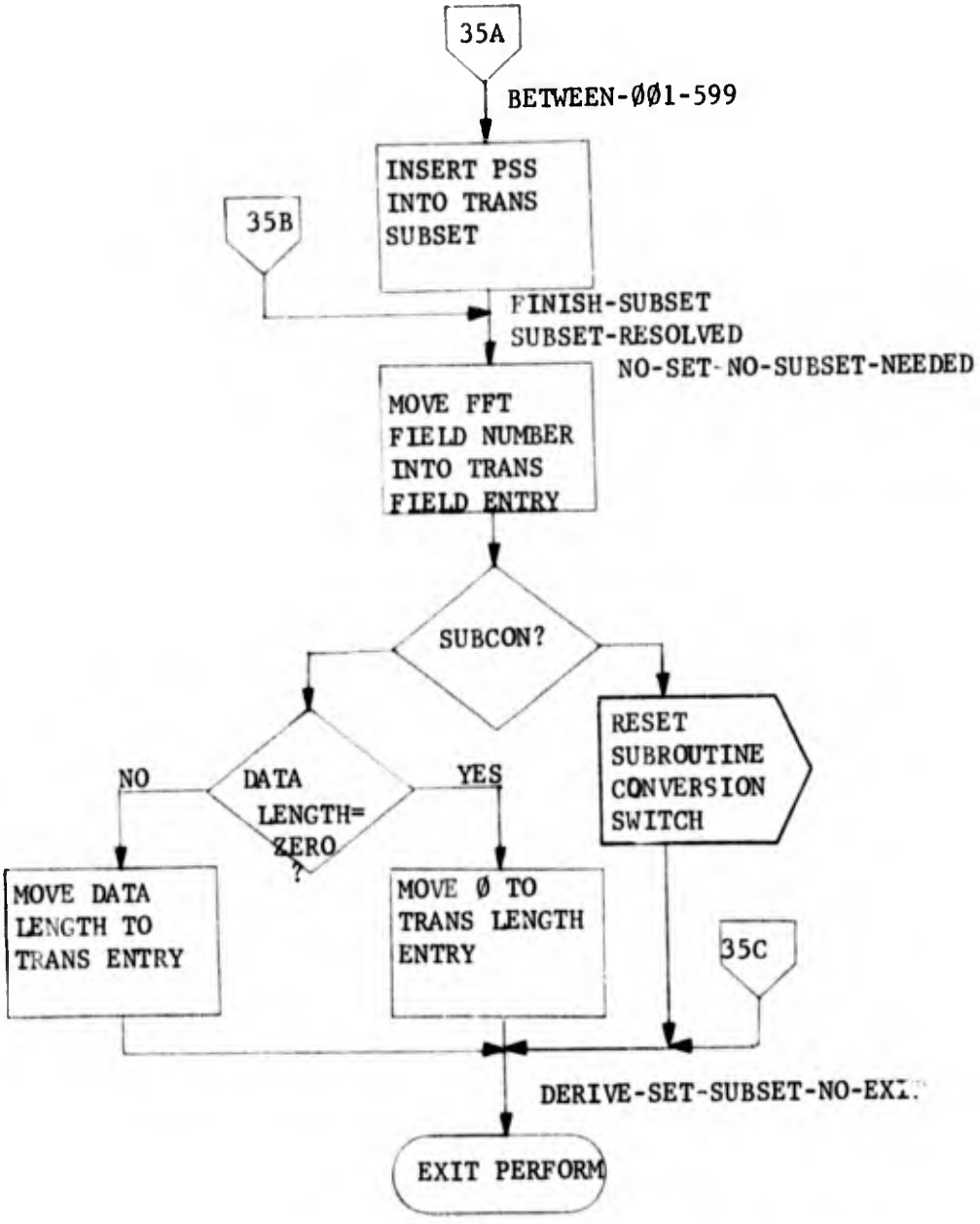

3ØA
FOUND-FIL-OPR
FOUND-REC-OPR
OPR-POSTED
LMCON VSET CHG OR WHOLE RECORD ?
YES
NO
PERFORM DERIVE-SET-SUBSET-NO THRU DERIVE-SET-SUBSET-NO-EXIT
CLEAR SWITCH
3Ø3
3ØB
PERFORMED PARAGRAPH FOLLOWS
NO
PSS ERROR?
YES
FORMAT-RECORD
FORMAT-OUTPUT
RESET TEXT POINTERS TO REMOVE CURRENT DATA
TRANS TABLE EXCEEDED ?
NO
YES
SET TRANS ITEM COUNT, TEXT COUNT
INDICATE ERROR
INDICATE ERROR
SKIP-SET-SUBSET PARAGRAPH CONTINUED
36A
36A
TRANSACTION ENTRY COMPLETE
1A

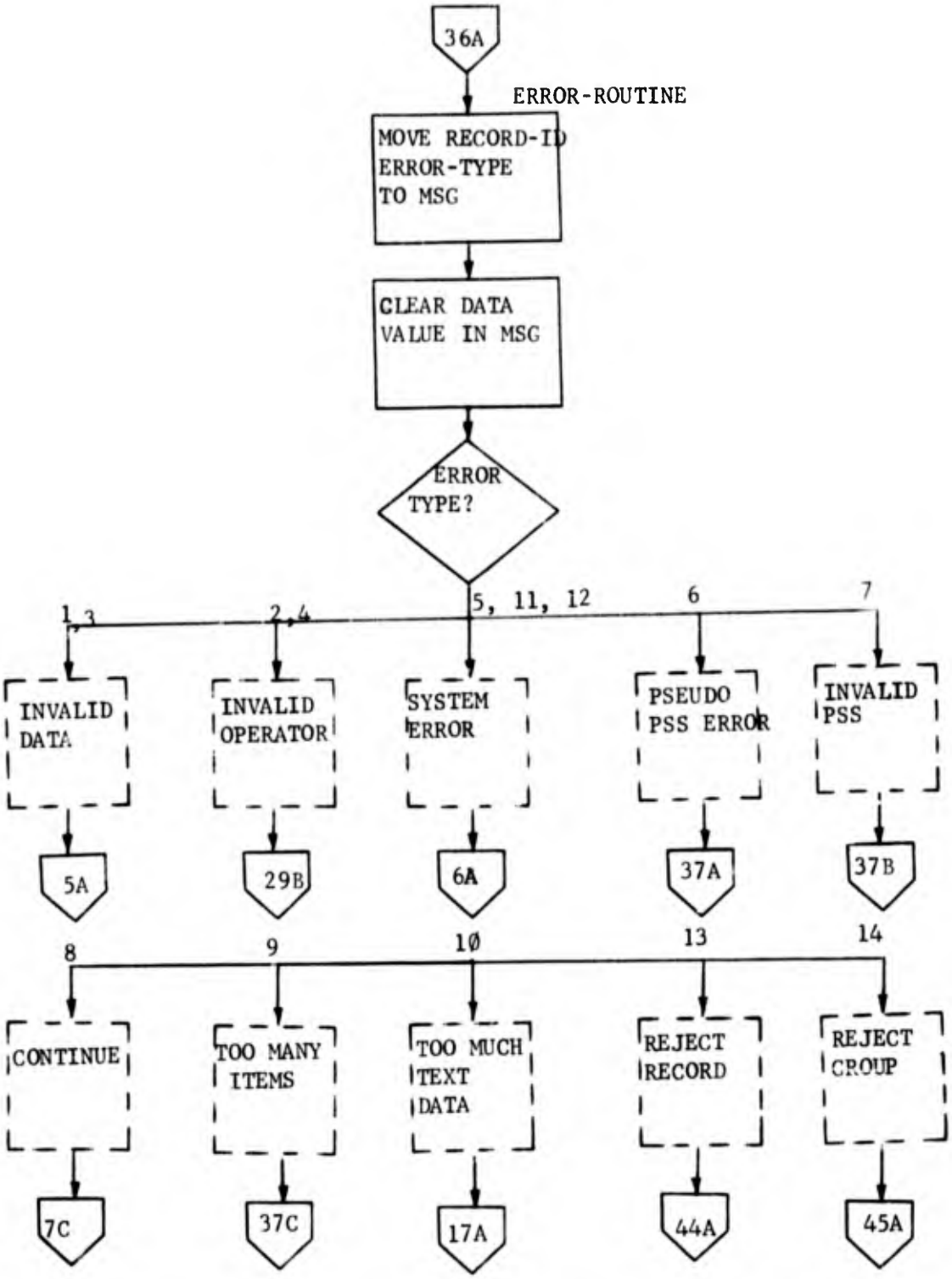

31A
DERIVE-SET-SUBSET-NO
FORMAT-STANDARD
LMCON?
NO
YES
RESET LMCON SWITCH
35B
FIXED SET?
YES
INSERT Ø SET, SUBSET INTO TRANS ENTRY
35B
PERIODIC SET?
YES
PLACE SET NUMBER INTO TRANS ENTRY
32A
PLACE VSET CODE NUMBER INTO TRANS ENTRY
INSERT Ø SUBSET INTO TRANS ENTRY
VSET CHANGE?
NO
YES
35B
35C

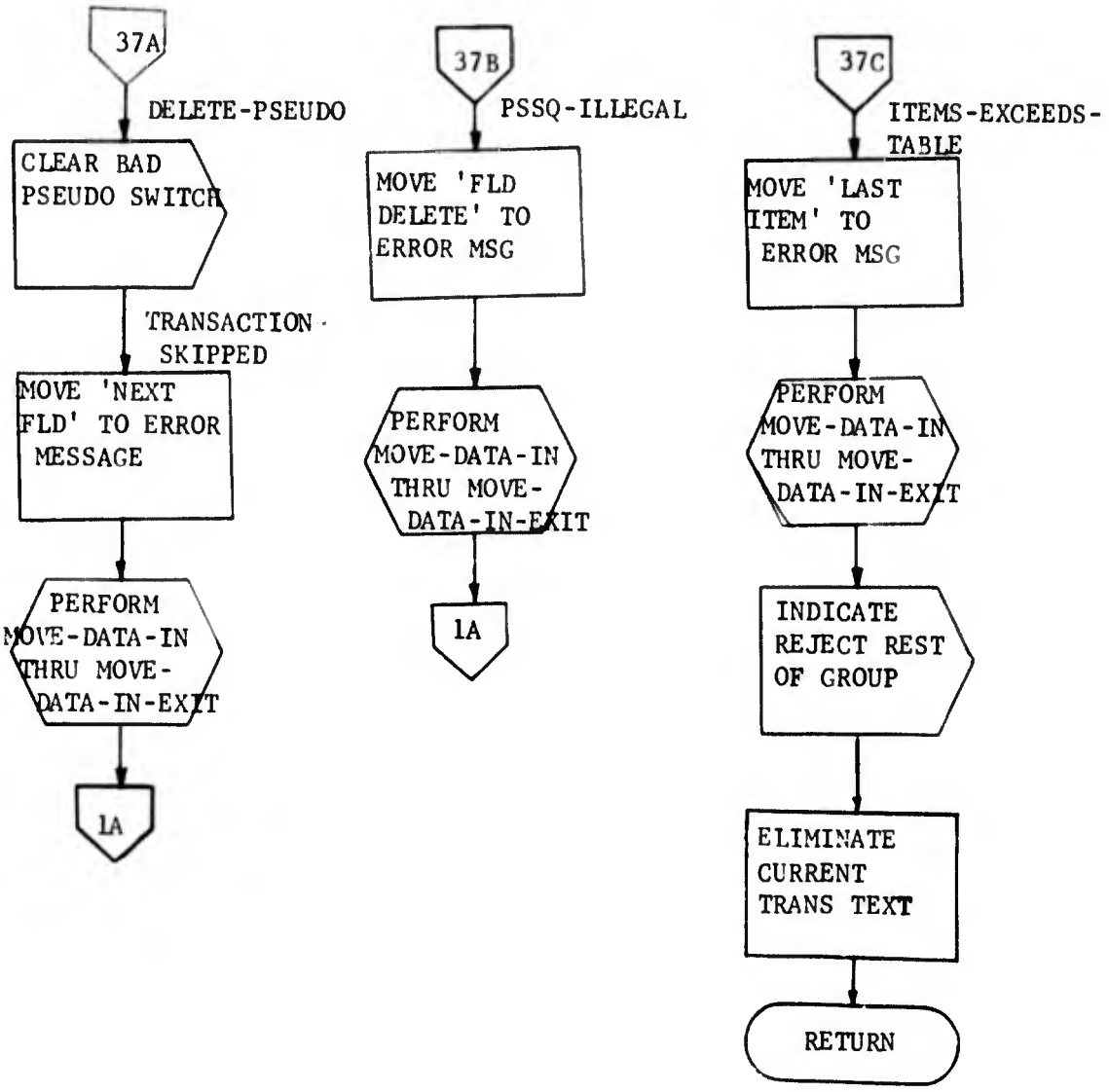

32A
PERIODIC-SUBSET
GET SET NUMBER FROM TRANS ENTRY
GET PSS HOP, LEN FROM DSD
CLEAR PSS AREA
IS PSS LENGTH Ø?
NO
33A
YES
32B
PSSQ-IS-BLANK TEST-6ØØ
IS PSEUDO PERMITTED?
YES
TEST-6Ø
NO
INDICATE ERROR
CURRENT =PREVIOUS PSEUDO?
NO
YES
FIRST PSEUDO FOR SET?
NO
YES
ADJUST PSEUDO NUMBER BY 1
INDICATE PSEUDO FOUND FOR SET
32C
MOVE-PSSQ
SAVE USER PSEUDO
32D
MOVE-GEN
INSERT PSEUDO INTO TRANS SUBSET ENTRY
35A
RESET TEXT POINTERS TO REMOVE CURRENT DATA
CLEAR NO PSEUDO SWITCH
36A

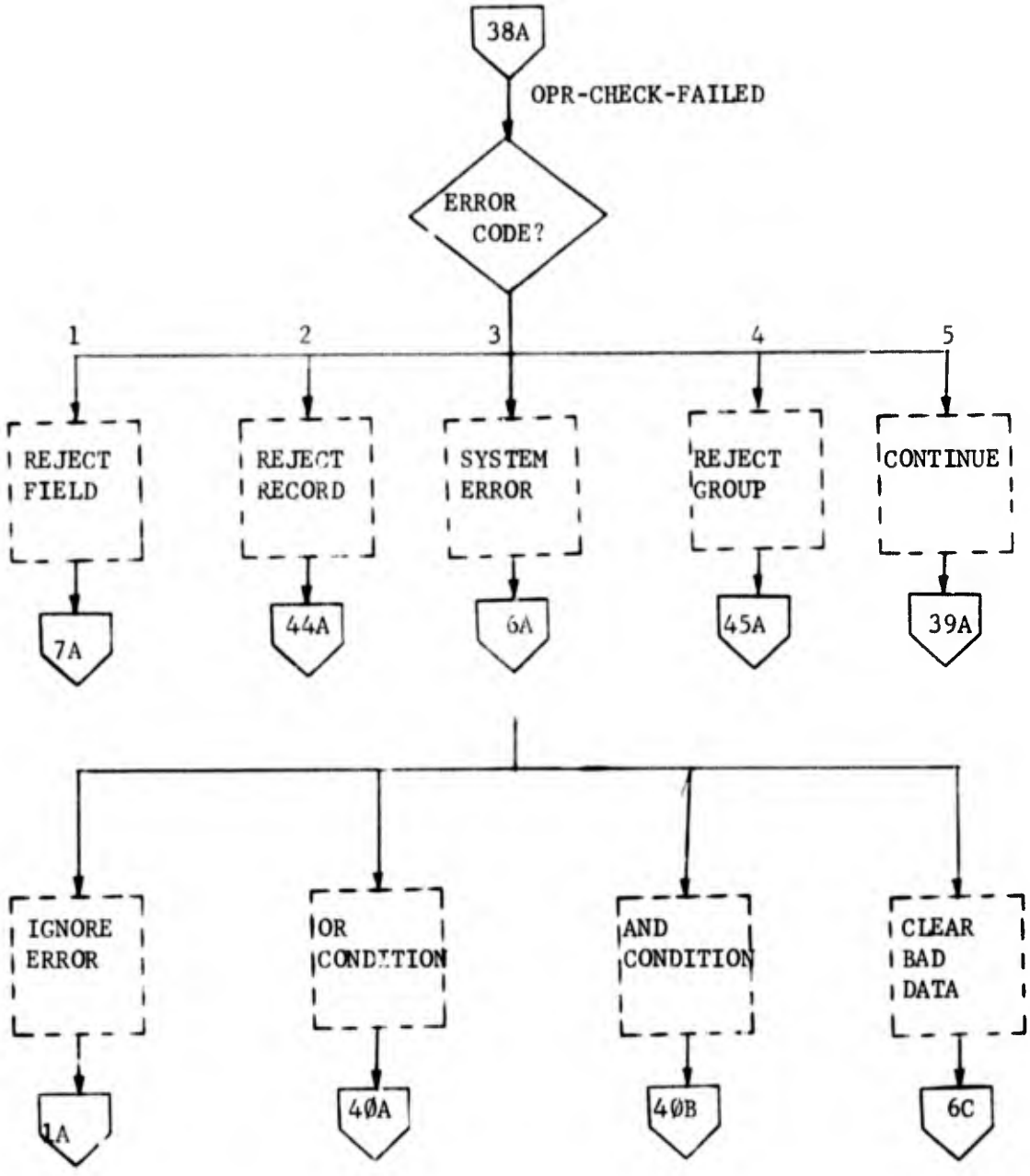

33A
CALL MOVRAY TO MOVE PSS INTO PSS AREA
PSS LENGTH?
1,2
3
TYPE-ONE-PSSQ
PSSQ33
IS PSS NUMERIC?
NO
YES
SET UP PSS AREA WITH 3 DIGIT PSS VALUE
ANN PSEUDO?
NO
YES
34A
35A
IS PSS BLANK?
YES
NO
32B
INDICATE ERROR
35C
BLANK PSS?
NO
YES
32B
NUMERIC PSS?
NO
YES
INDICATE ERROR
35C
PSS >599
35A
32B

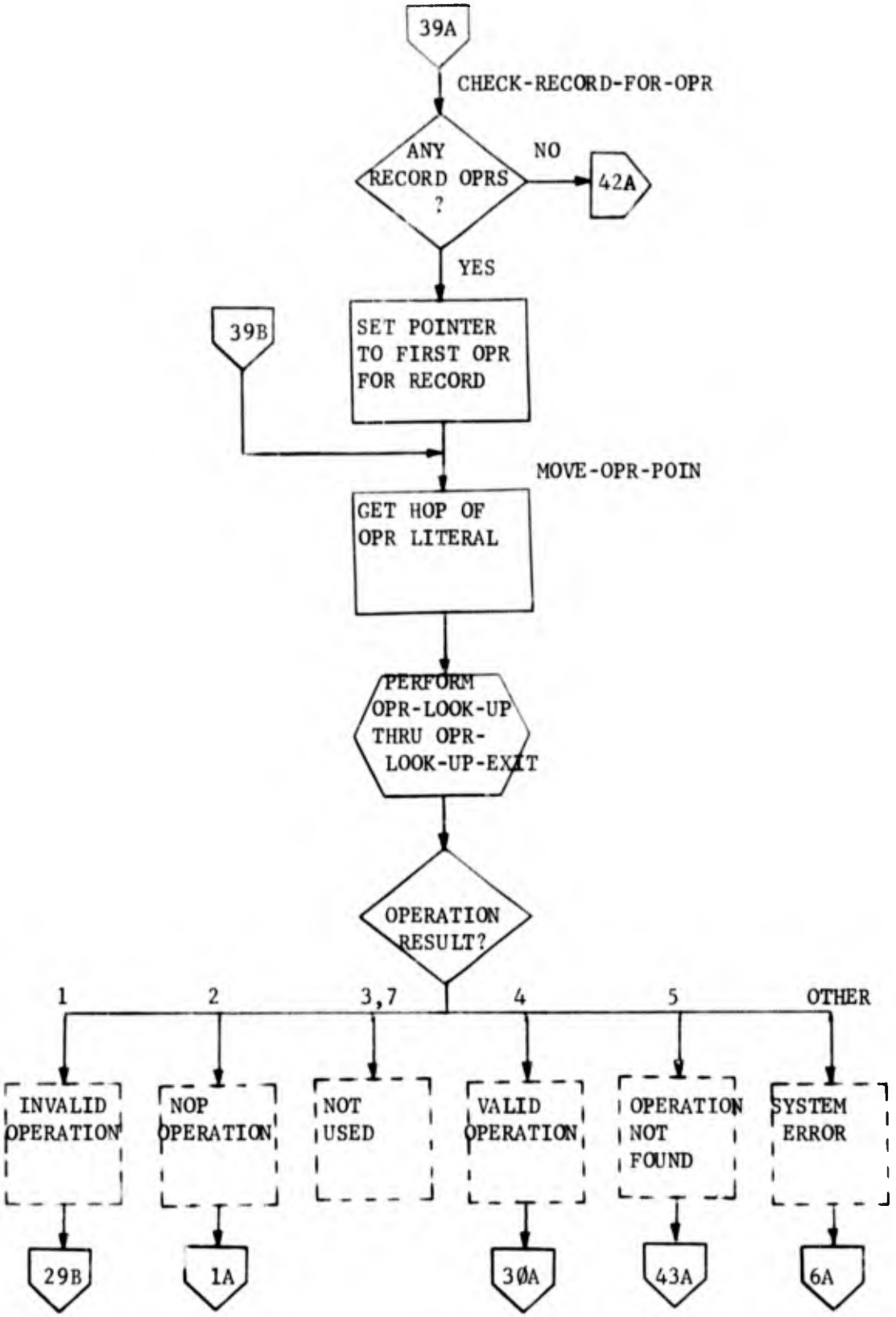

34A
TYPE-ANN
IS PSEUDO PERMITTED?
YES
NO
CURRENT PREVIOUS PSEUDO?
NO
YES
32D
INDICATE ERROR
RESET TEXT POINTERS TO REMOVE CURRENT DATA
CLEAR NO PSEUDO SWITCH
36A
IS NN OF ANN NUMERIC?
YES
NO
INDICATE ERROR
35C
FIRST PSEUDO FOR SET?
NO
YES
INDICATE PSEUDO FOUND FOR SET
WAS PREVIOUS PSEUDO ANN ?
YES
ADJUST PSEUDO NUMBER BASED ON OLD/NEW NN VALUE
INSERT PSEUDO INTO TRANS SUBSET ENTRY
ADD NN TO PSEUDO NUMBER
32C
CURRENT > PREVIOUS PSEUDO
NO
YES
SAVE USER PSEUDO AND PSEUDO NUMBER
35B

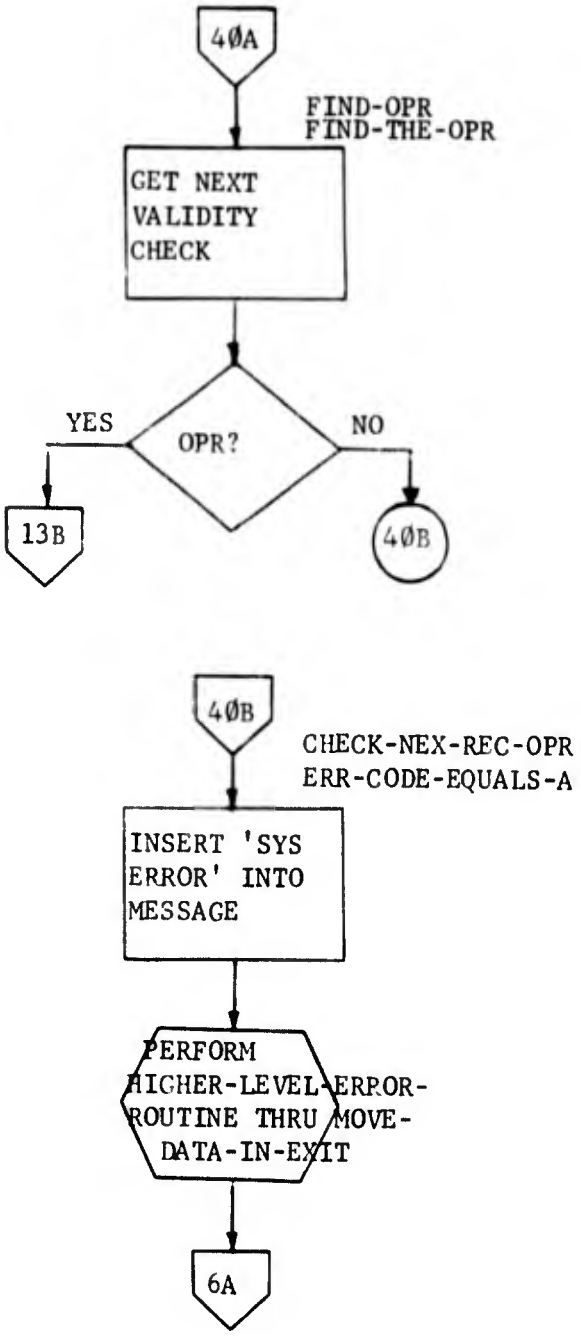

35A
BETWEEN-ØØ1-599
INSERT PSS INTO TRANS SUBSET
35B
FINISH-SUBSET
SUBSET-RESOLVED
NO-SET-NO-SUBSET-NEEDED
MOVE FFT FIELD NUMBER INTO TRANS FIELD ENTRY
SUBCON?
RESET SUBROUTINE CONVERSION SWITCH
NO
DATA LENGTH= ZERO ?
YES
MOVE DATA LENGTH TO TRANS ENTRY
MOVE Ø TO TRANS LENGTH ENTRY
35C
DERIVE-SET-SUBSET-NO-EXI
EXIT PERFORM

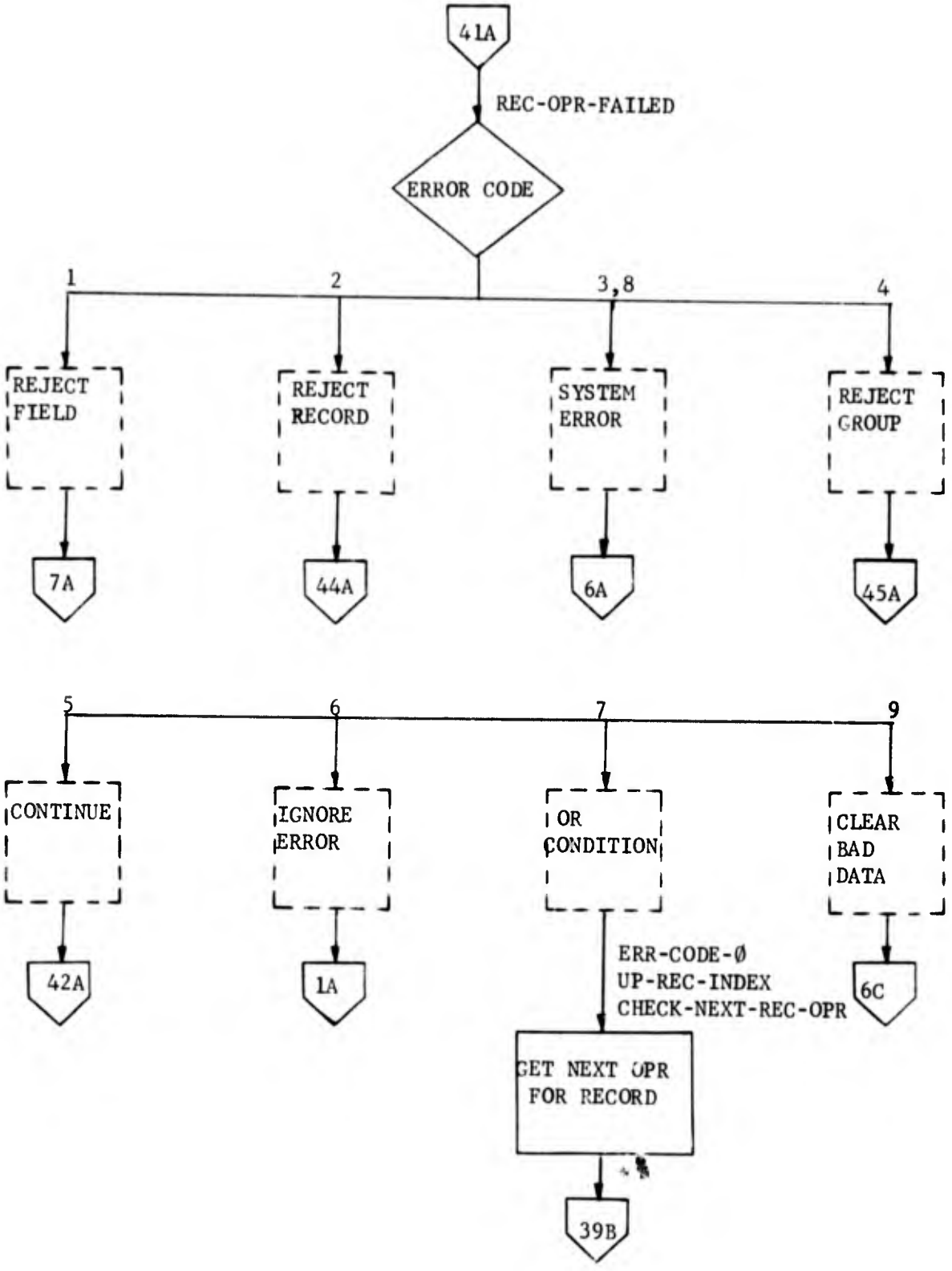

36A
ERROR-ROUTINE
MOVE RECORD-ID
ERROR-TYPE
TO MSG
CLEAR DATA
VALUE IN MSG
ERROR
TYPE?
1, 3
2, 4
5, 11, 12
6
7
INVALID
DATA
INVALID
OPERATOR
SYSTEM
ERROR
PSEUDO
PSS ERROR
INVALID
PSS
5A
29B
6A
37A
37B
8
9
1Ø
13
14
CONTINUE
TOO MANY
ITEMS
TOO MUCH
TEXT
DATA
REJECT
RECORD
REJECT
GROUP
7C
37C
17A
44A
45A

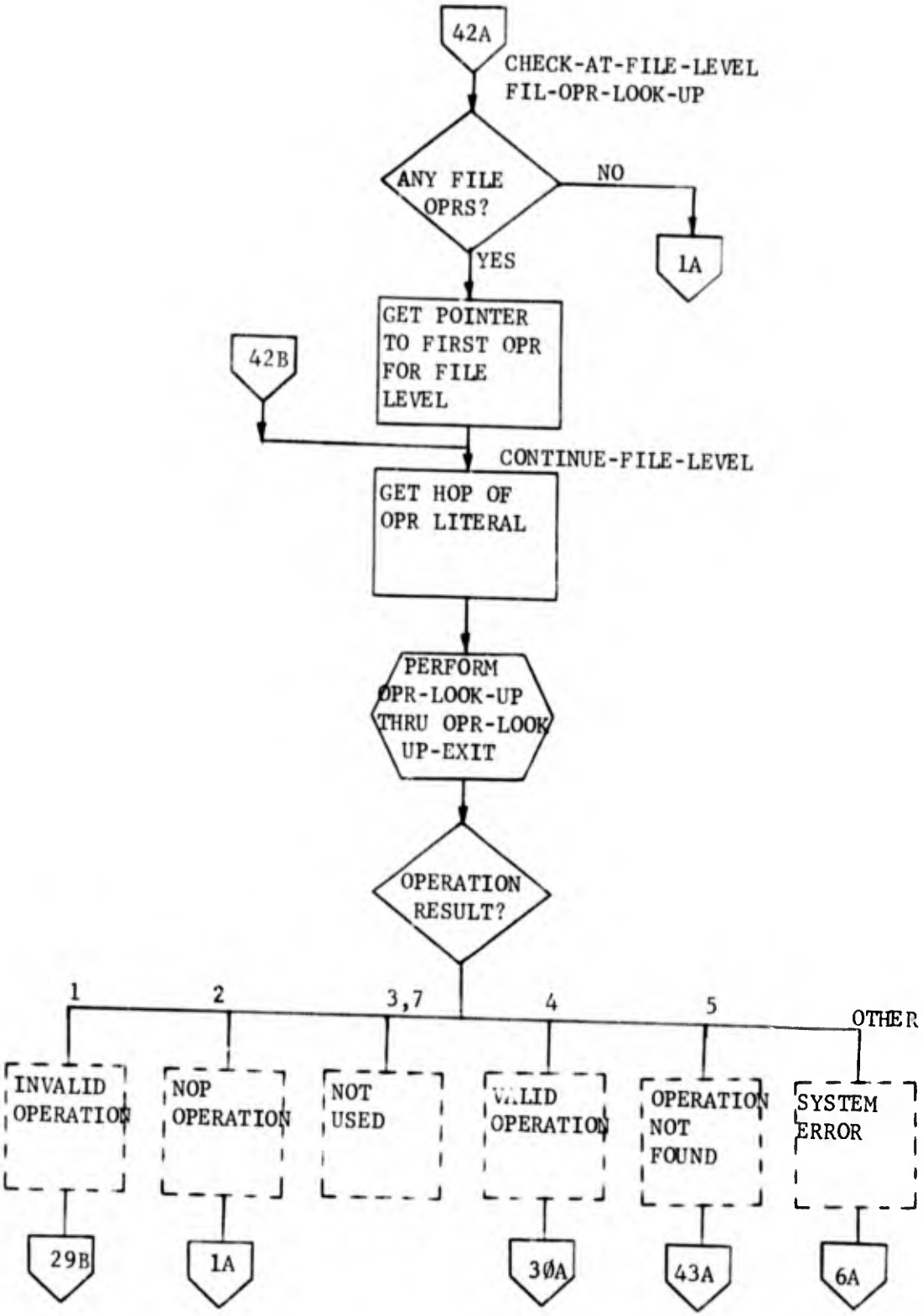

37A
DELETE-PSEUDO
CLEAR BAD PSEUDO SWITCH
TRANSACTION SKIPPED
MOVE 'NEXT FLD' TO ERROR MESSAGE
PERFORM MOVE-DATA-IN THRU MOVE-DATA-IN-EXIT
1A
37B
PSSQ-ILLEGAL
MOVE 'FLD DELETE' TO ERROR MSG
PERFORM MOVE-DATA-IN THRU MOVE-DATA-IN-EXIT
1A
37C
ITEMS-EXCEEDS-TABLE
MOVE 'LAST ITEM' TO ERROR MSG
PERFORM MOVE-DATA-IN THRU MOVE-DATA-IN-EXIT
INDICATE REJECT REST OF GROUP
ELIMINATE CURRENT TRANS TEXT
RETURN

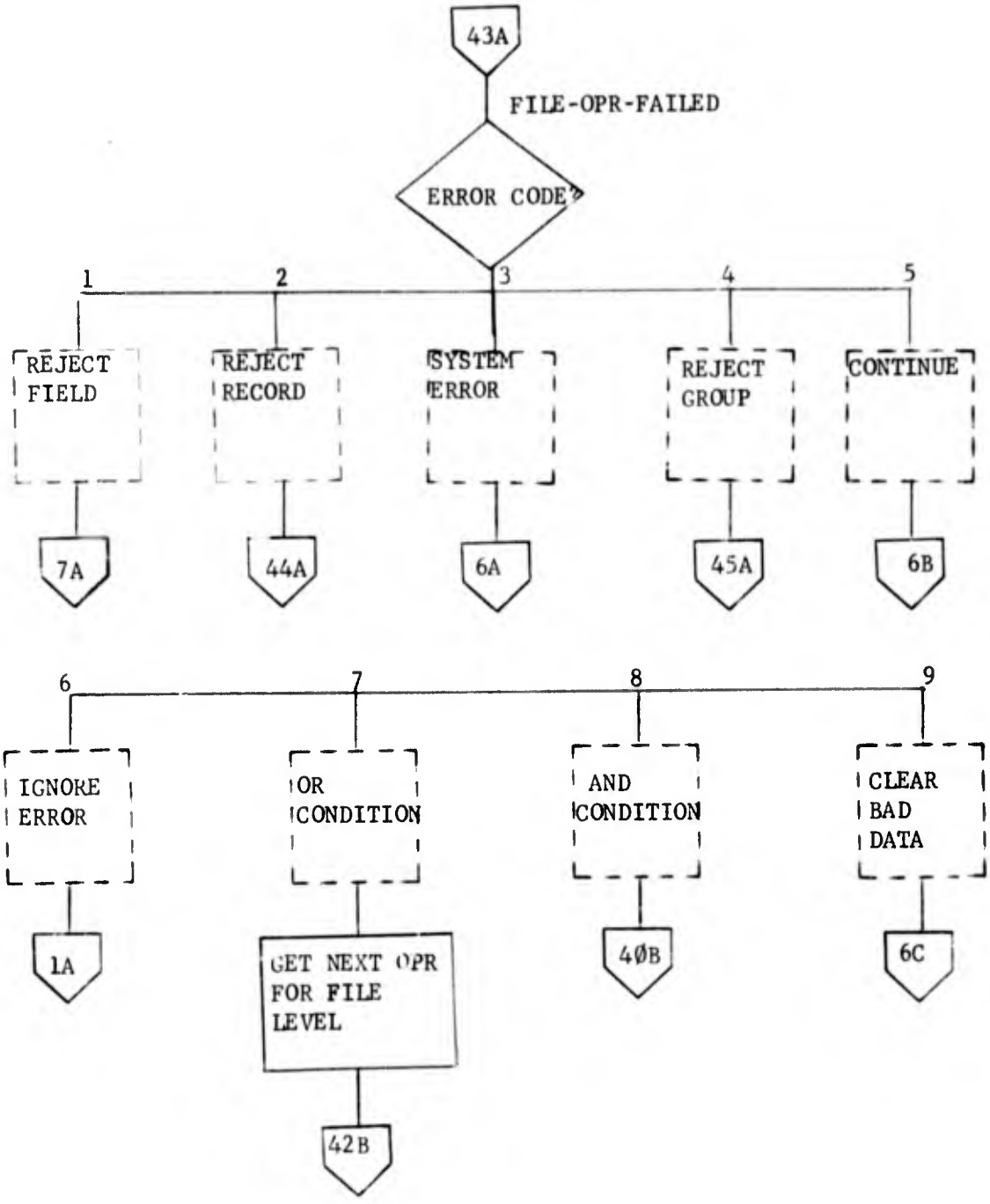

38A
OPR-CHECK-FAILED
ERROR CODE?
1
2
3
4
5
REJECT FIELD
REJECT RECORD
SYSTEM ERROR
REJECT GROUP
CONTINUE
7A
44A
6A
45A
39A
IGNORE ERROR
OR CONDITION
AND CONDITION
CLEAR BAD DATA
1A
40A
40B
6C

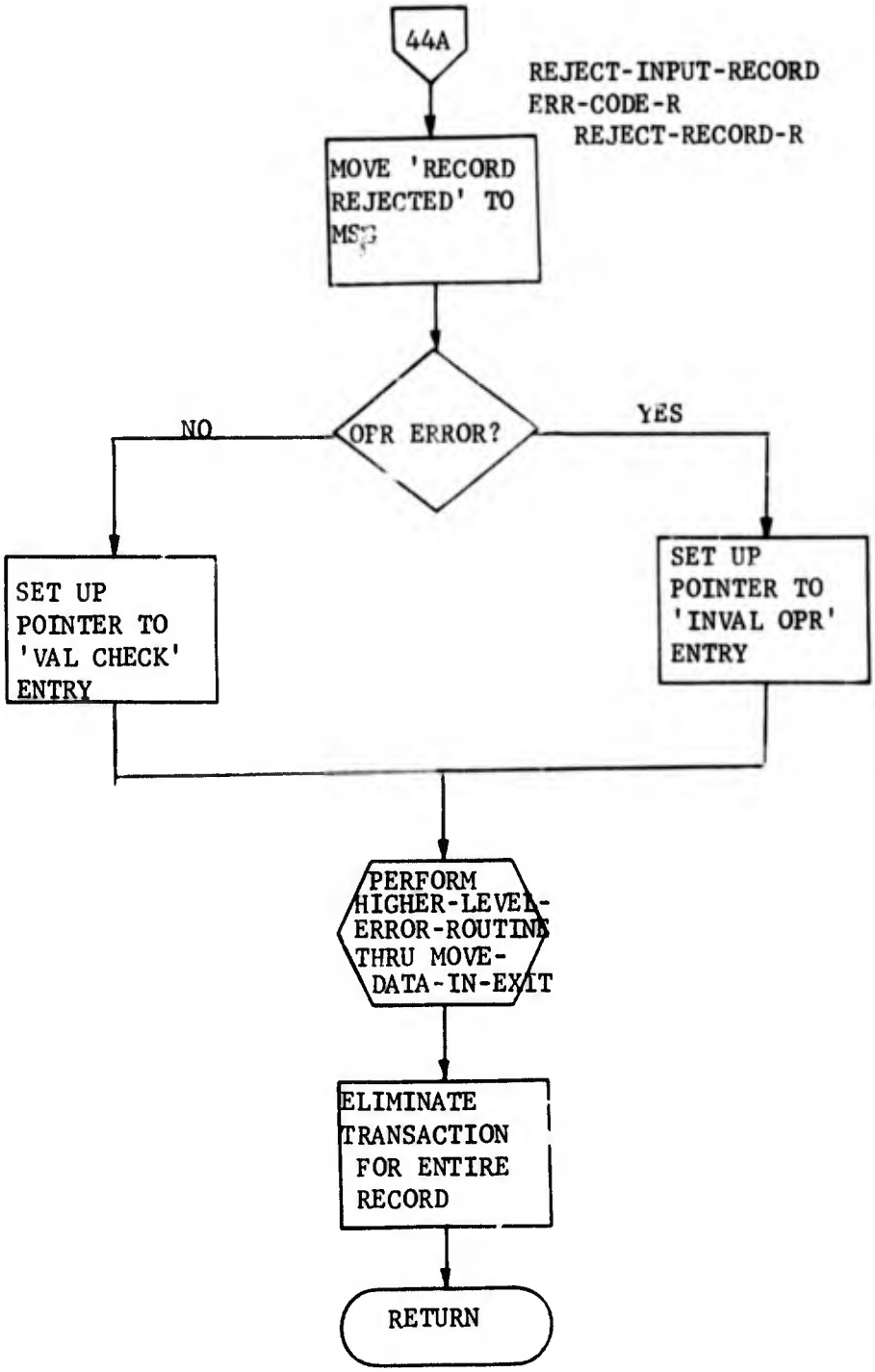

39A
CHECK-RECORD-FOR-OPR
ANY RECORD OPRS ?
NO
42A
YES
39B
SET POINTER TO FIRST OPR FOR RECORD
MOVE-OPR-POIN
GET HOP OF OPR LITERAL
PERFORM OPR-LOOK-UP THRU OPR-LOOK-UP-EXIT
OPERATION RESULT?
1
2
3,7
4
5
OTHER
INVALID OPERATION
NOP OPERATION
NOT USED
VALID OPERATION
OPERATION NOT FOUND
SYSTEM ERROR
29B
1A
30A
43A
6A

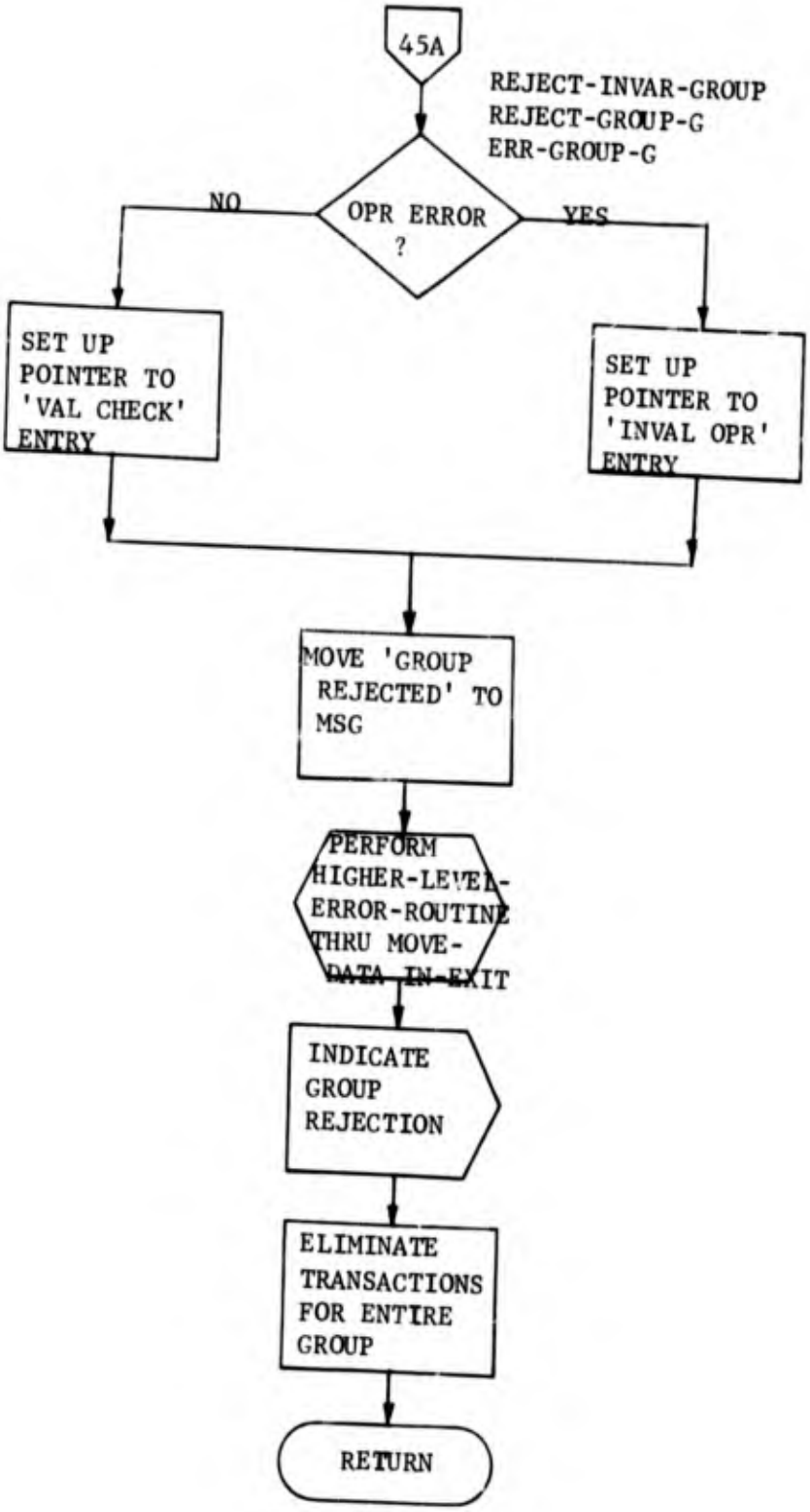

4ØA
FIND-OPR
FIND-THE-OPR
GET NEXT
VALIDITY
CHECK
YES
OPR?
NO
13B
4ØB
4ØB
CHECK-NEX-REC-OPR
ERR-CODE-EQUALS-A
INSERT 'SYS
ERROR' INTO
MESSAGE
PERFORM
HIGHER-LEVEL-ERROR-
ROUTINE THRU MOVE-
DATA-IN-EXIT
6A


41A
REC-OPR-FAILED
ERROR CODE
1
2
3,8
4
REJECT FIELD
REJECT RECORD
SYSTEM ERROR
REJECT GROUP
7A
44A
6A
45A
5
6
7
9
CONTINUE
IGNORE ERROR
OR CONDITION
CLEAR BAD DATA
42A
1A
ERR-CODE-Ø
UP-REC-INDEX
CHECK-NEXT-REC-OPR
6C
GET NEXT OPR FOR RECORD
39B


42A
CHECK-AT-FILE-LEVEL
FIL-OPR-LOOK-UP
ANY FILE OPRS?
NO
YES
1A
GET POINTER TO FIRST OPR FOR FILE LEVEL
42B
CONTINUE-FILE-LEVEL
GET HOP OF OPR LITERAL
PERFORM OPR-LOOK-UP THRU OPR-LOOK UP-EXIT
OPERATION RESULT?
1
2
3,7
4
5
OTHER
INVALID OPERATION
NOP OPERATION
NOT USED
VALID OPERATION
OPERATION NOT FOUND
SYSTEM ERROR
29B
1A
30A
43A
6A


43A
FILE-OPR-FAILED
ERROR CODE?
1
2
3
4
5
REJECT FIELD
REJECT RECORD
SYSTEM ERROR
REJECT GROUP
CONTINUE
7A
44A
6A
45A
6B
6
7
8
9
IGNORE ERROR
OR CONDITION
AND CONDITION
CLEAR BAD DATA
1A
GET NEXT OPR FOR FILE LEVEL
42B
4ØB
6C


44A
REJECT-INPUT-RECORD
ERR-CODE-R
REJECT-RECORD-R
MOVE 'RECORD REJECTED' TO MSG
OPR ERROR?
NO
YES
SET UP POINTER TO 'VAL CHECK' ENTRY
SET UP POINTER TO 'INVAL OPR' ENTRY
PERFORM HIGHER-LEVEL-ERROR-ROUTINE THRU MOVE-DATA-IN-EXIT
ELIMINATE TRANSACTION FOR ENTIRE RECORD
RETURN


45A
REJECT-INVAR-GROUP
REJECT-GROUP-G
ERR-GROUP-G
OPR ERROR ?
NO
YES
SET UP POINTER TO 'VAL CHECK' ENTRY
SET UP POINTER TO 'INVAL OPR' ENTRY
MOVE 'GROUP REJECTED' TO MSG
PERFORM HIGHER-LEVEL-ERROR-ROUTINE THRU MOVE-DATA-IN-EXIT
INDICATE GROUP REJECTION
ELIMINATE TRANSACTIONS FOR ENTIRE GROUP
RETURN

f. FMIPTRN

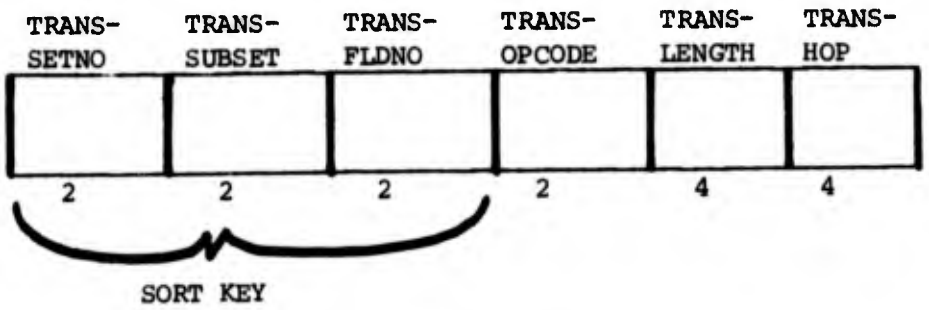
(1) Function. To sort transaction table entries and write transaction file records.

(2) Calling Sequence.

```
ENTRY 'IPTRN' USING IP-DD UNSORTED-TRANS-TAB TEXT-TAB.
```

(3) Capabilities.

FMIPTRN is called by FMIPX to sort a transaction table for an input group and write transaction file records for the input group. Upon completion of input processing, FMIPTRN is instructed to close the transaction file. WRITE-SW is used to control these alternatives. WRITE-SW is initially 4, causing the transaction file to be opened and causing WRITE-SW to be changed to 0. When WRITE-SW is 0 upon entry to FMIPTRN, the Confirmation Record is written on the transaction file, WRITE-SW is changed to 1, and the first subordinate group of transactions is processed. When WRITE-SW is 1 (as is normally the case), a subordinate group of transactions is processed. WRITE-SW=3 causes a Reject Group Record to be written and WRITE-SW to be changed to 1. WRITE-SW=2 indicates that the transaction file is to be closed. (WRITE-SW is cleared to 0 if no sort of transaction records is required.) The transaction table sorting process in paragraphs SET-UP-SORT to REDO orders the fields to be updated so that they can be handled in sequence by Maintenance Proper. Three TRANS-ITEM fields take part in the sort -- TRANS-SETNO, TRANS-SUBSET, and TRANS-FLDNO. TRANS-SETNO orders the transaction entries by set, TRANS-SUBSET orders the transaction entries by subsets within a set, and TRANS-FLDNO orders by positions within a single subset. Figure 3-3-3 shows the possible values for these fields.



TRANS-SETNO = 000 for Fixed Set
0NN for Periodic Set NN
1NN for Variable Set NN
200 for LMCON

TRANS-SUBSET = 0000 for Fixed, Whole Vset, or LMCON
0NNN for Periodic Subset Number
(Pseudo if NNN is greater than 600)
NNNN for HOP (relative to 1) of Partial
Vset Change

TRANS-FIELD = Subscript of Field in FFT LR2
Delete Length for Partial Vset Change
1 for Whole Record Operations
0 for LMCON

TRANS-OPCODE = Code Number of Operation/Field-Type.

TRANS-LENGTH = Length of Text String for this item.

TRANS-HOP = HOP of Text in Total Text String (relative to 1).

Figure 3-3-3. Transaction Item Format

The balance of the FMIPTRN subprogram writes out six types of transaction records into a transaction file. These records are variable in length and are blocked. (For the IBM 360, BLOCK CONTAINS 0 RECORDS is specified in the COBOL File Description for the transaction file so that the maximum block size is determined from the Job Control Language DD card. The minimum permitted block size is 1060 characters.)

The six transaction record types are indicated below. Their formats are specified in figure 3-3-4.

The Confirmation Record is written at the beginning of the transaction file and specifies the fields to be confirmed by Maintenance Proper. CONSW has a value of "0" when no confirmations are specified, "1" for individual field confirmations, or "2" when all fields will be confirmed. CONFIRM INFO contains one digit entry for each of 299 possible fields. Each position normally contains a "0" value except that when CONSW is "1" and the field is to be confirmed, the entry will have a "1".

A Reject Group Record is written when a group rejection is encountered after one or more subordinate groups have been written on the transaction file. IP sort will place this record in front of all of the subordinate group control records so that Maintenance Proper will skip over the transaction records for all subordinate groups of the input group.

A Subordinate Group Control Record is written at least once for each input group, once for each subordinate group. It contains the number of transaction items in the transaction table, the total number of characters required for the table entries, and the total size of the text entries for the subordinate group. IP sort will place all subordinate group control records together so that MP will process all subordinate groups of a single input group as if it occurred on a single control record.

Item Information Records subdivide the transaction table into segments of 62 items per record. Any subordinate group may require multiple Item Information Records to hold the entire transactio table.

Text Information Records subdivide the transaction text area into segments of 1000 characters without regard to the transaction entries they belong to.

The Last Record is written at the end of the transaction file.

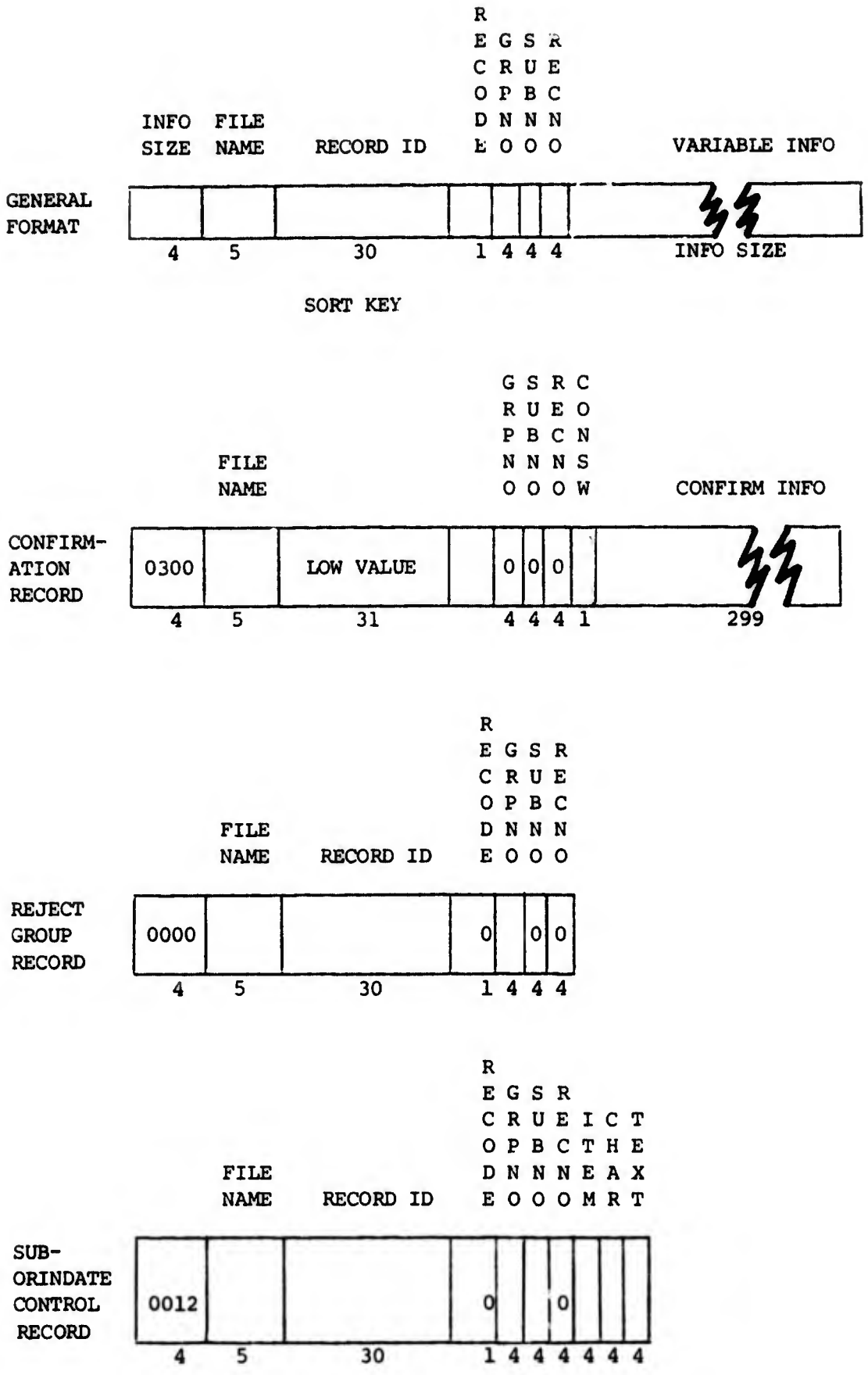


Figure 3-3-4. Transaction File Record Formats (IBM 360)

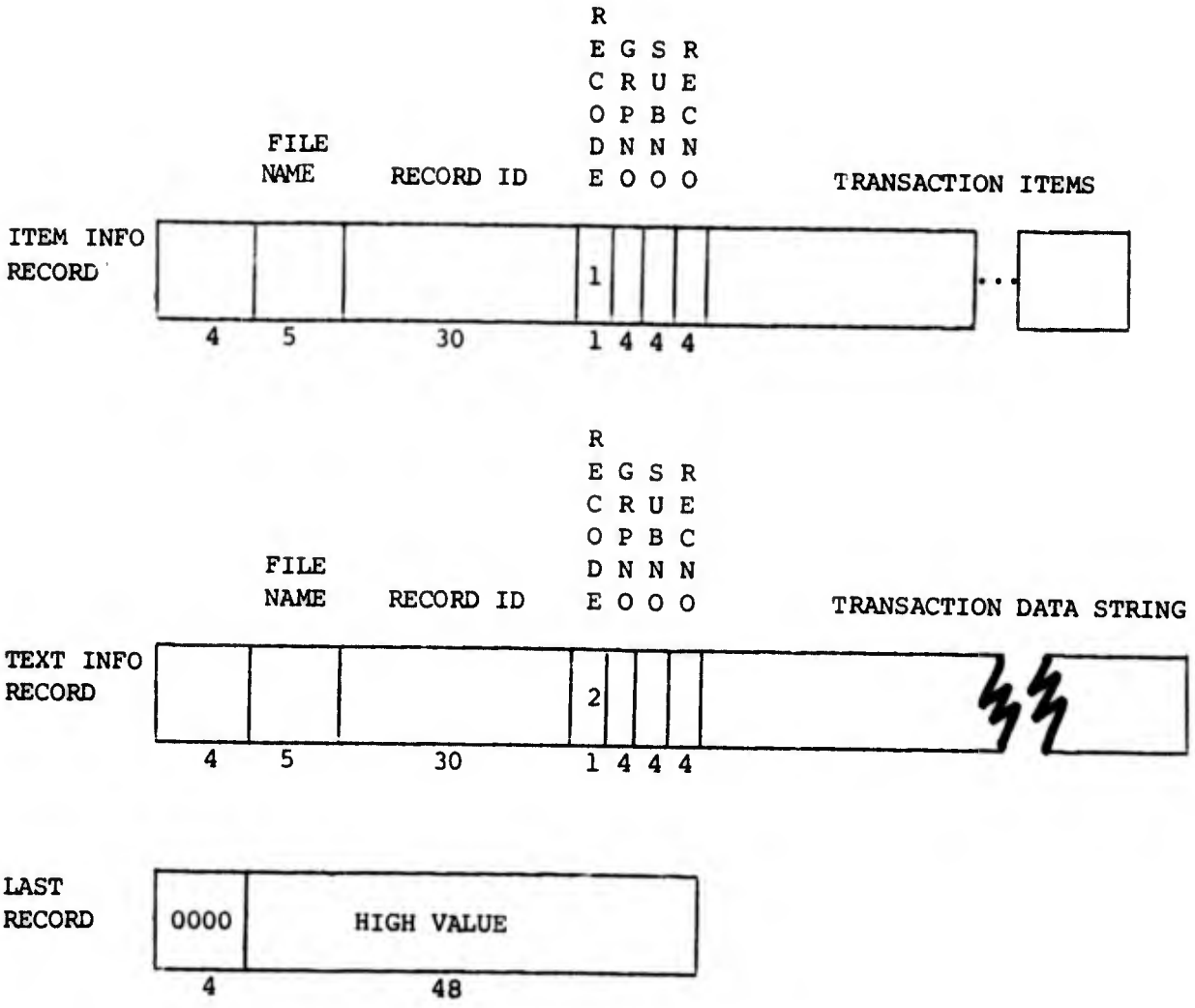


Figure 3-3-4. Transaction File Record Formats (IBM 360) (Continued)

The following fields are placed in transaction records to control the actions of IP Sort and Maintenance Proper. (All format information applies to the IBM 360 version of MIDMS only.)

INFO SIZE is four decimal digits in length and contains the number of characters in the variable portion of the transaction record beyond the sort key. It is used as the OCCURS...DEPENDING variable so that COBOL can compute the logical record size of each transaction record. The maximum value for INFO SIZE is 1000. The Reject Group Record and the Last Record have no variable data so that INFO SIZE is 0000 for these records. The Confirmation Record contains exactly 0300 characters of variable data and the Subordinate Group Control Record contains exactly 0012 characters. INFO SIZE is a multiple of 16 (between 0016 and 0992) for Item Information Records. Text Information Records may contain 0001-1000 characters.

FILE NAME is a five character field which would be the major sort key if updates for multiple files could be placed in a single transaction file. Since only one file is permitted, all records except the last contain the standard MIDMS file name ending with "A". The Last Record contains highest value alphanumeric characters (all binary ones) in the FILE NAME field to assure that it remains the last record after sorting.

RECORD ID is a thirty character field normally containing a record control field value so that records will be updated by Maintenance Proper in ascending record ID order. The value is left justified with trailing blanks when the defined control fields total less than thirty characters. The value of the RECORD ID field is derived from data in the MIDMS ID fields specified in the DSD or from the record ID information in Unit Update cards. As special cases, the Confirmation Record contains low values (binary zeroes) in the RECORD ID field and the Last Record contains high values (binary ones). The low values assure that confirmation information will be received first by MP. The high values in the Last Record are essentially ignored by the sort since FILE NAME already contains high values.

RECODE is a one character code number to reorder and distinguish types of transaction records. Reject Group and Subordinate Group Control Records contain the digit "0" in the RECODE field. In Item Information Records, RECODE has the digit "1". Text Information Records are identified by the digit "2" in RECODE. FMIPTRN generates one Subordinate Group Control Record, one or more Item

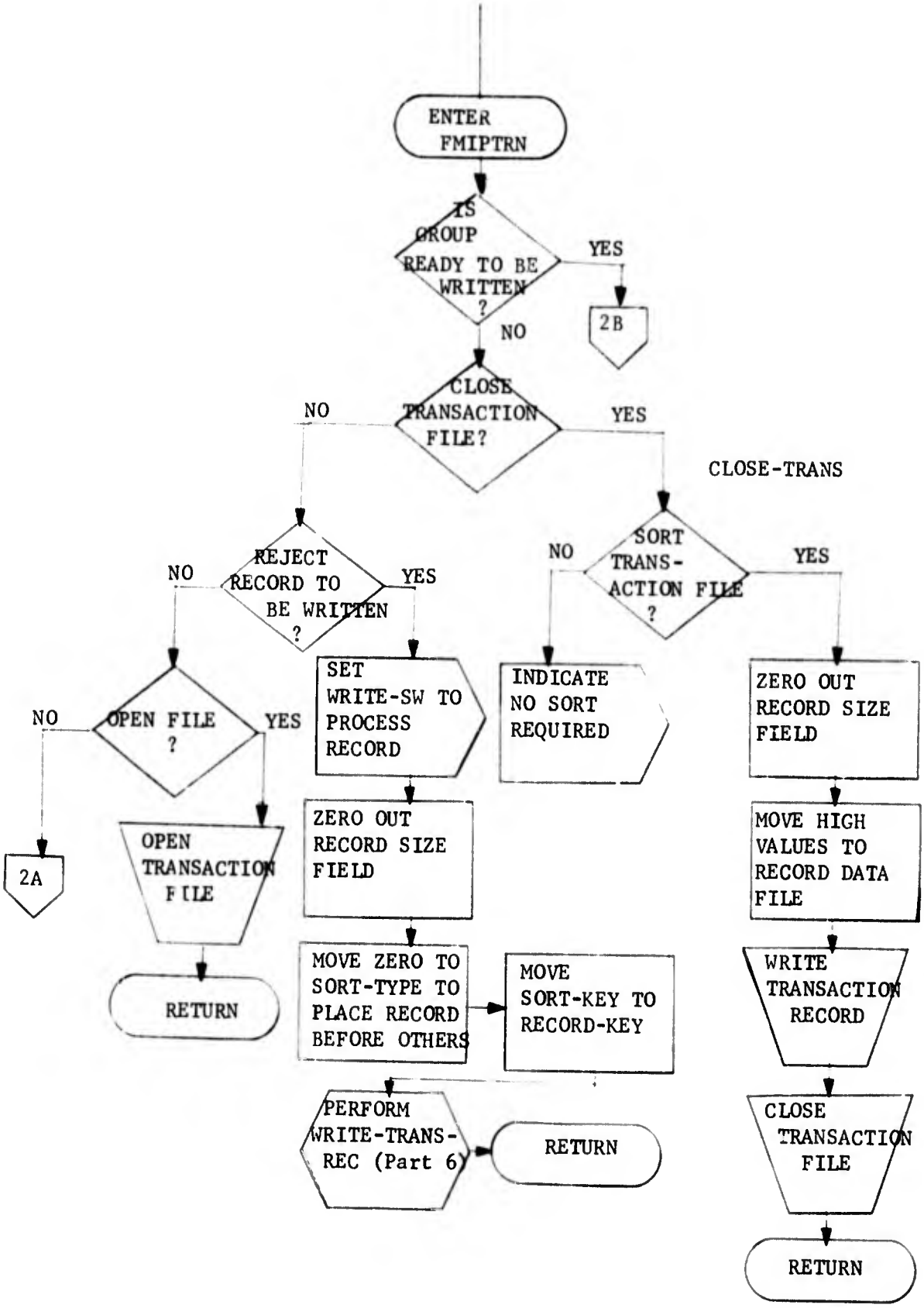
Information Records, and zero or more Text Information Records whenever requested by FMIPX. When more than one subordinate group are created for a source input group or more than one source input group apply to the same MIDMS file record, the sort will cause all type "0" records for a particular record ID to be placed before all type "1" records and both will be placed before type "2" records. This is the sequence expected by MP, which will combine all of the information obtained into a single series of field updates. The Reject Group Record, when present for any group, will be sorted in front of all Subordinate Group Control Records for the group because of a SUBNO value of zero. As special cases, RECODE has binary zeroes for the Confirmation Record and binary ones for the Last Record.

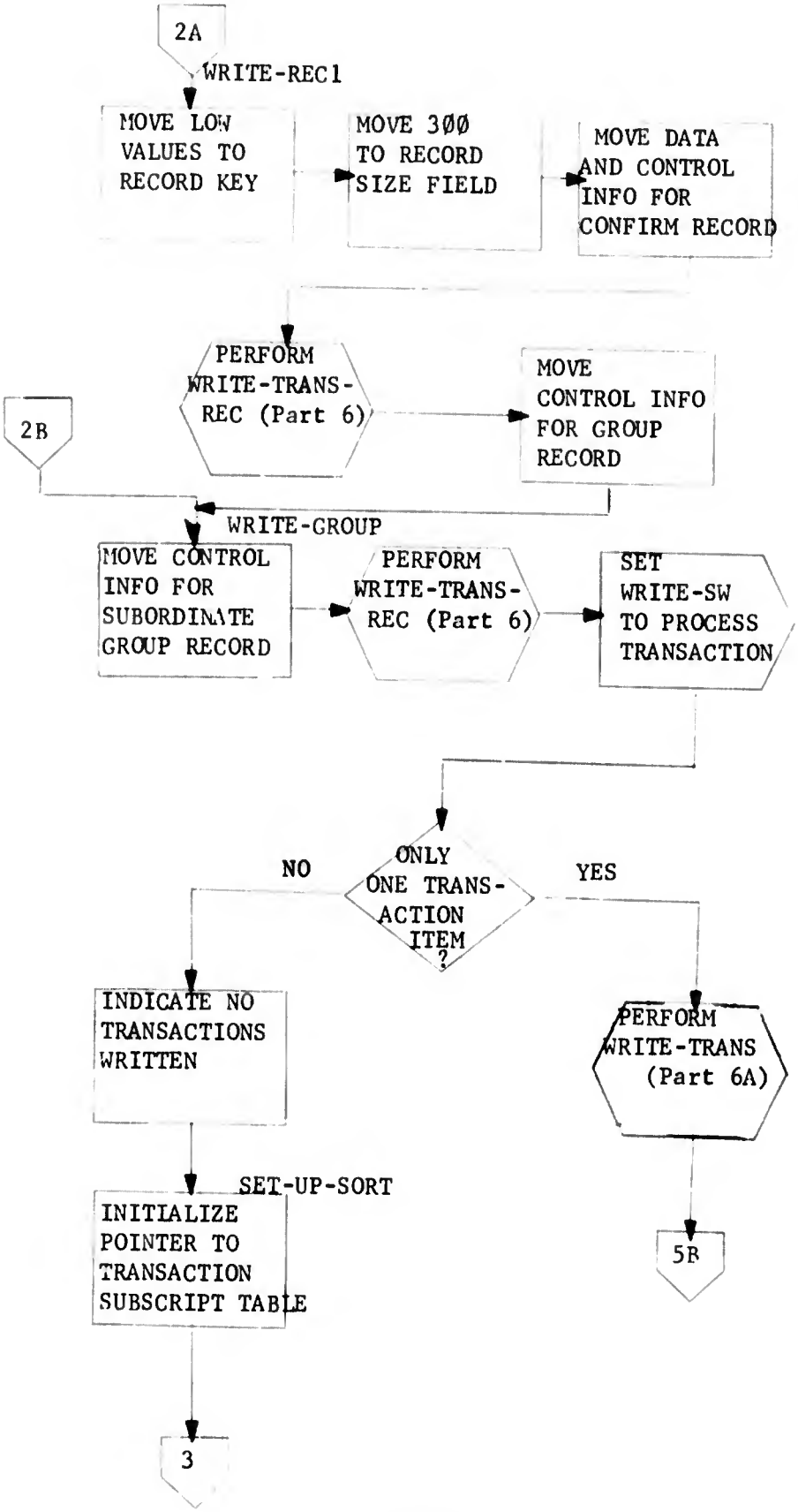
GRPNO is a four character binary number which is generally unaligned because of blocking of transaction records. It is derived from a counter for source input groups as determined by FMIPREC or FMIPUN. After sorting, similar types of records will be ordered by group number so that field update precedence in MP will be partly determined by input sequence. GRPNO has binary zeroes for the Confirmation Record and binary ones for the Last Record.

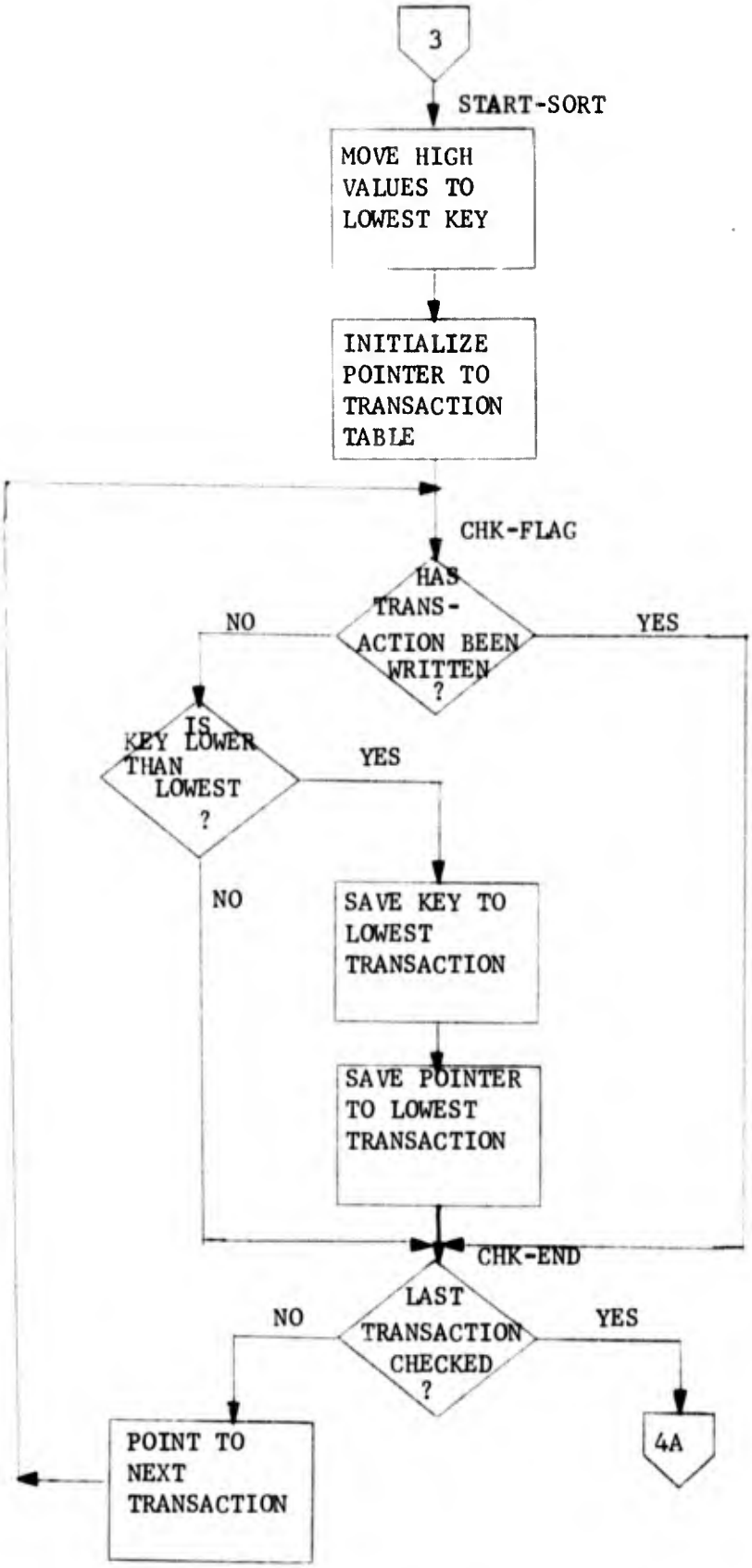
SUBNO is a four character binary number assigned (starting from 1) to each subordinate group of an input group. SUBNO fields are generally unaligned. When a group is rejected by FMIPREC or FMIPVAL based on DSD criteria, binary zeroes are placed in SUBNO so that the Reject Group Record will sort before Subordinate Group Control Records causing elimination of the group by MP. SUBNO has binary zeroes for the Confirmation Record and binary ones for the Last Record.

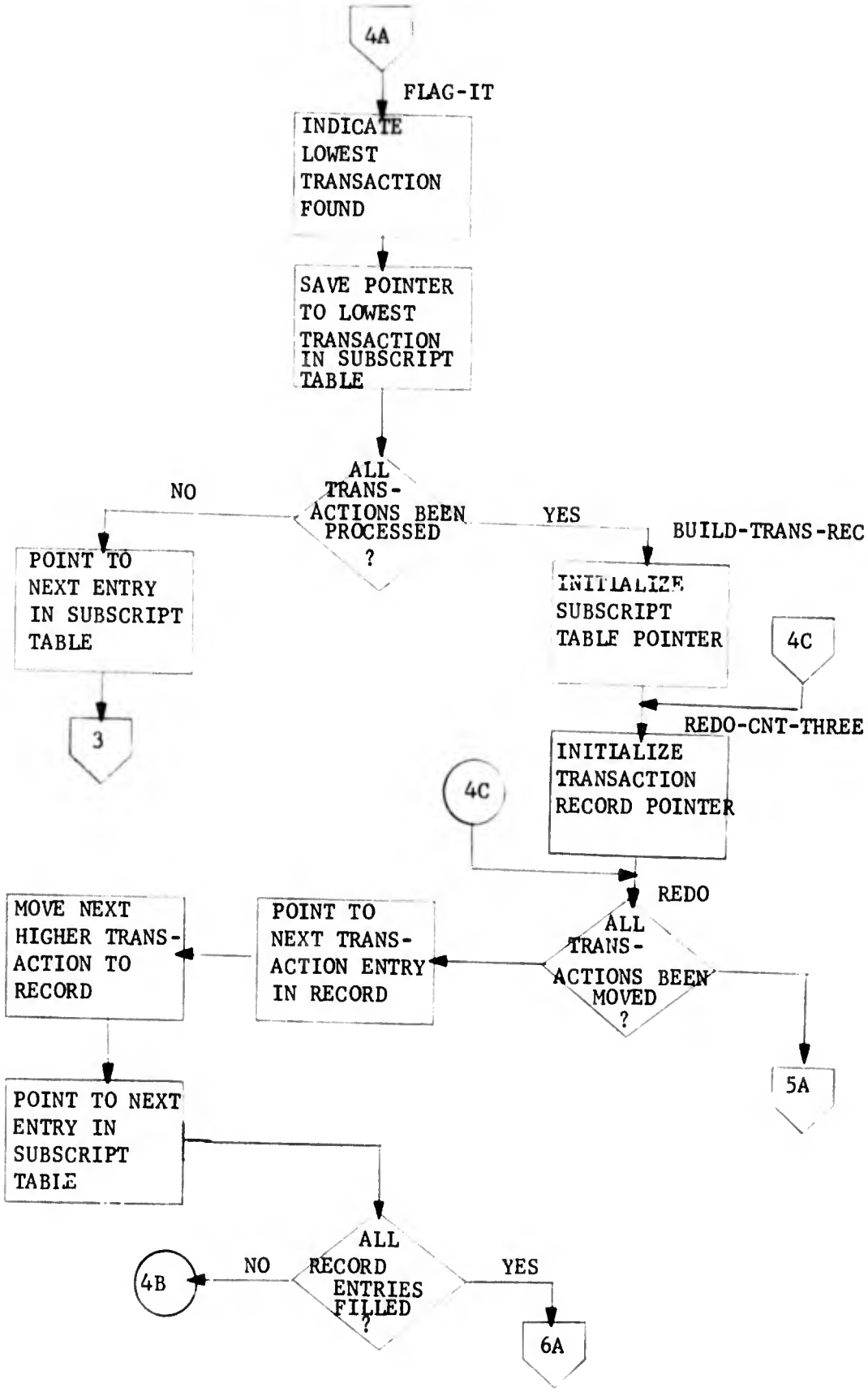
RECNO is a four character binary number (generally unaligned). Within each type of record in a subordinate group, RECNO counts the records starting from zero. That is, RECNO is zero for the Subordinate Group Control Record; it is 0, 1, 2 ..., for Item Information Records; and it is 0, 1, 2 ..., for Text Information Records. The sort will not change the order of individual Item or Text Information Records. A Reject Group Record also has a RECNO of zero. A zero value in RECNO is used by MP to determine the end of a previous series of records of a particular type. RECNO has binary zeroes for the Confirmation Record and binary ones for the Last Record.

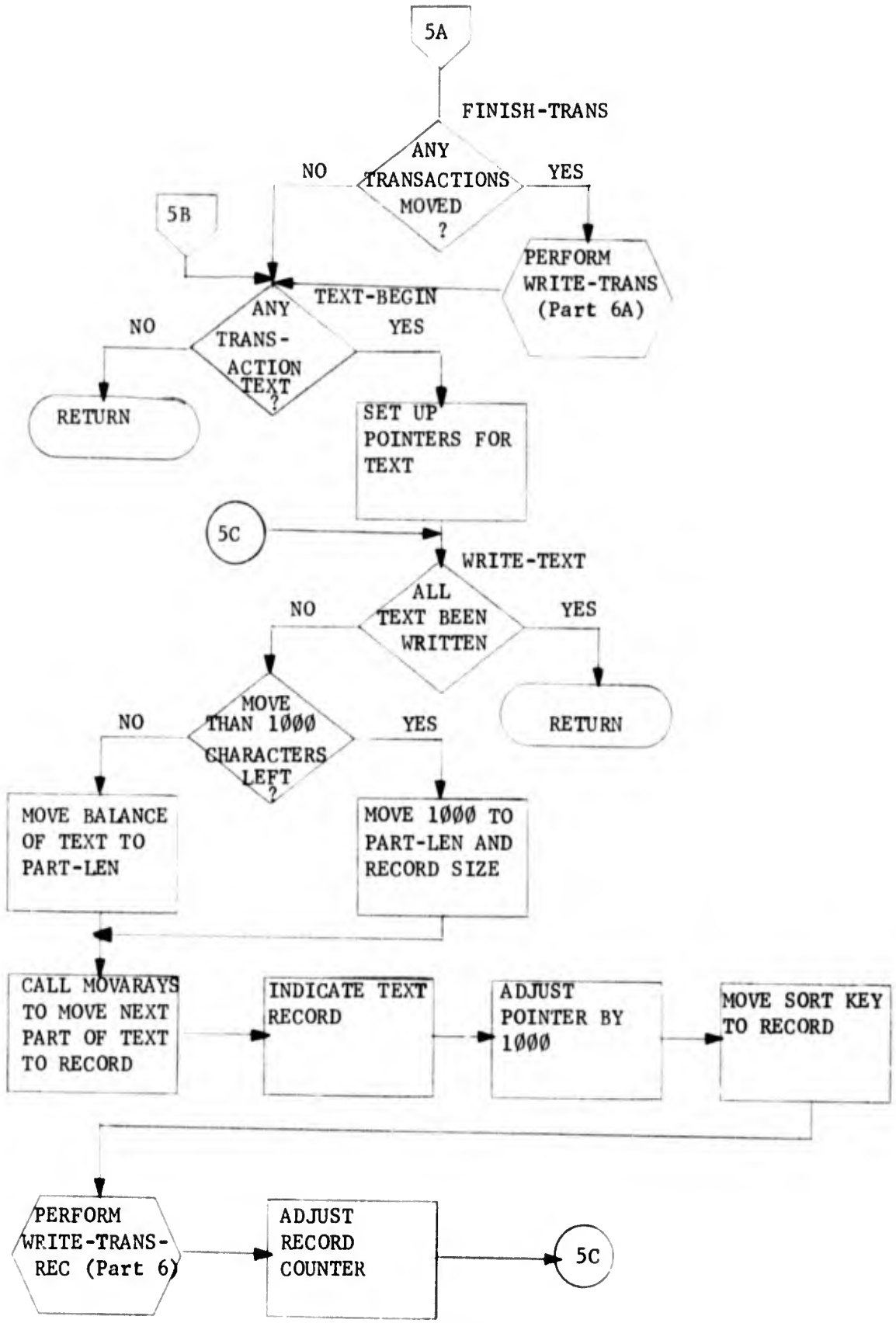
(4) Flowchart.

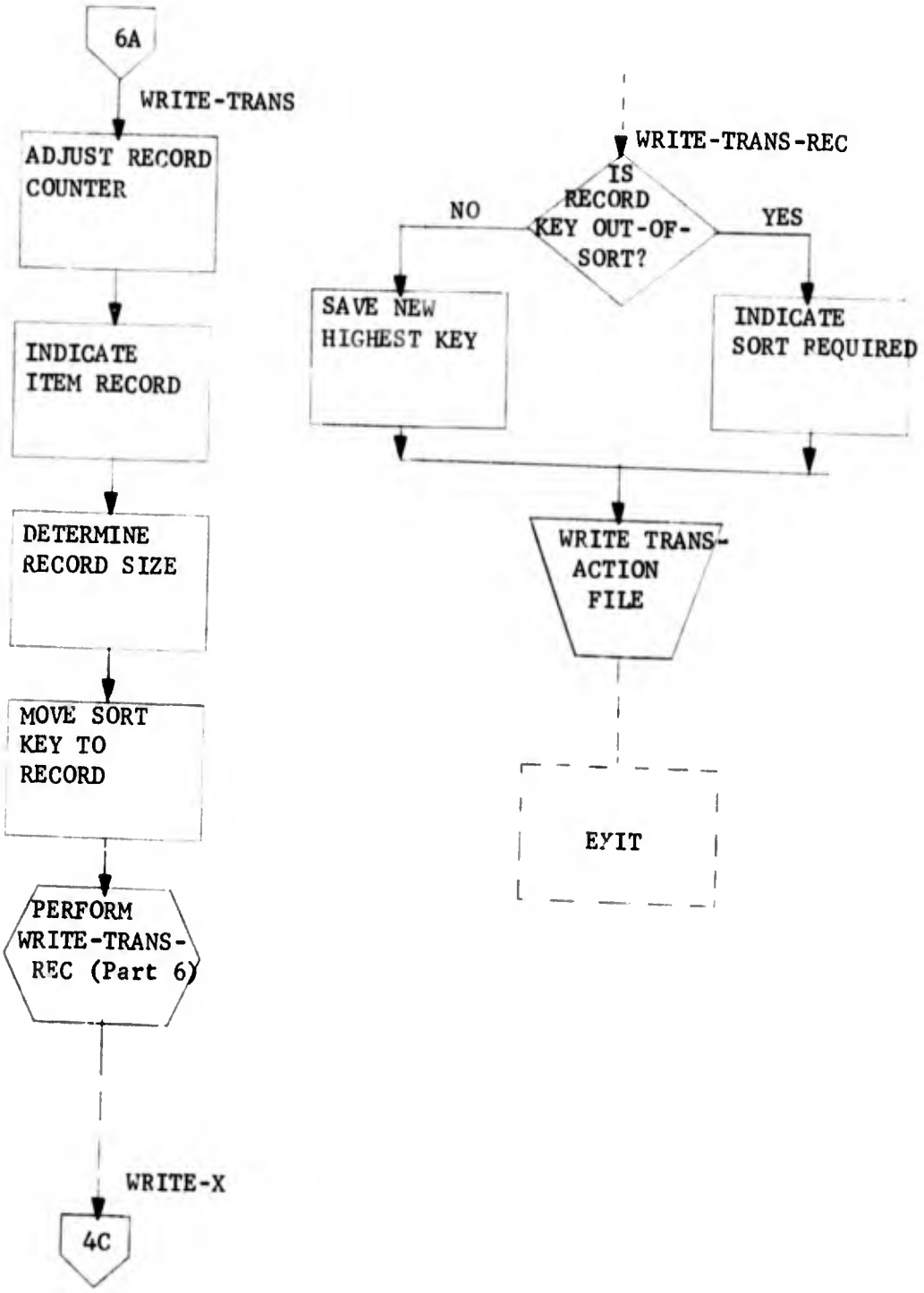

ENTER FMIPTRN
IS GROUP READY TO BE WRITTEN ?
YES
2B
NO
CLOSE TRANSACTION FILE?
NO
YES
CLOSE-TRANS
REJECT RECORD TO BE WRITTEN ?
NO
YES
SORT TRANS-ACTION FILE ?
NO
YES
OPEN FILE ?
NO
YES
2A
SET WRITE-SW TO PROCESS RECORD
INDICATE NO SORT REQUIRED
ZERO OUT RECORD SIZE FIELD
OPEN TRANSACTION FILE
ZERO OUT RECORD SIZE FIELD
MOVE HIGH VALUES TO RECORD DATA FILE
RETURN
MOVE ZERO TO SORT-TYPE TO PLACE RECORD BEFORE OTHERS
MOVE SORT-KEY TO RECORD-KEY
WRITE TRANSACTION RECORD
PERFORM WRITE-TRANS-REC (Part 6)
RETURN
CLOSE TRANSACTION FILE
RETURN


2A
WRITE-REC1
MOVE LOW VALUES TO RECORD KEY
MOVE 300 TO RECORD SIZE FIELD
MOVE DATA AND CONTROL INFO FOR CONFIRM RECORD
PERFORM WRITE-TRANS-REC (Part 6)
MOVE CONTROL INFO FOR GROUP RECORD
2B
WRITE-GROUP
MOVE CONTROL INFO FOR SUBORDINATE GROUP RECORD
PERFORM WRITE-TRANS-REC (Part 6)
SET WRITE-SW TO PROCESS TRANSACTION
ONLY ONE TRANS-ACTION ITEM ?
NO
YES
INDICATE NO TRANSACTIONS WRITTEN
PERFORM WRITE-TRANS (Part 6A)
SET-UP-SORT
INITIALIZE POINTER TO TRANSACTION SUBSCRIPT TABLE
5B
3


3
START-SORT
MOVE HIGH VALUES TO LOWEST KEY
INITIALIZE POINTER TO TRANSACTION TABLE
CHK-FLAG
HAS TRANS-ACTION BEEN WRITTEN ?
NO
YES
IS KEY LOWER THAN LOWEST ?
YES
NO
SAVE KEY TO LOWEST TRANSACTION
SAVE POINTER TO LOWEST TRANSACTION
CHK-END
LAST TRANSACTION CHECKED ?
NO
YES
POINT TO NEXT TRANSACTION
4A


4A
FLAG-IT
INDICATE LOWEST TRANSACTION FOUND
SAVE POINTER TO LOWEST TRANSACTION IN SUBSCRIPT TABLE
ALL TRANS-ACTIONS BEEN PROCESSED ?
NO
YES
BUILD-TRANS-REC
POINT TO NEXT ENTRY IN SUBSCRIPT TABLE
3
INITIALIZE SUBSCRIPT TABLE POINTER
4C
REDO-CNT-THREE
INITIALIZE TRANSACTION RECORD POINTER
4C
REDO
ALL TRANS-ACTIONS BEEN MOVED ?
POINT TO NEXT TRANS-ACTION ENTRY IN RECORD
MOVE NEXT HIGHER TRANS-ACTION TO RECORD
5A
POINT TO NEXT ENTRY IN SUBSCRIPT TABLE
ALL RECORD ENTRIES FILLED ?
NO
YES
4B
6A


5A
FINISH-TRANS
ANY TRANSACTIONS MOVED ?
NO
YES
5B
PERFORM WRITE-TRANS (Part 6A)
TEXT-BEGIN
ANY TRANS-ACTION TEXT ?
NO
YES
RETURN
SET UP POINTERS FOR TEXT
5C
WRITE-TEXT
ALL TEXT BEEN WRITTEN
NO
YES
RETURN
MOVE THAN 1ØØØ CHARACTERS LEFT ?
NO
YES
MOVE BALANCE OF TEXT TO PART-LEN
MOVE 1ØØØ TO PART-LEN AND RECORD SIZE
CALL MOVARAYS TO MOVE NEXT PART OF TEXT TO RECORD
INDICATE TEXT RECORD
ADJUST POINTER BY 1ØØØ
MOVE SORT KEY TO RECORD
PERFORM WRITE-TRANS-REC (Part 6)
ADJUST RECORD COUNTER
5C


6A
WRITE-TRANS
ADJUST RECORD COUNTER
INDICATE ITEM RECORD
DETERMINE RECORD SIZE
MOVE SORT KEY TO RECORD
PERFORM WRITE-TRANS-REC (Part 6)
WRITE-X
4C
WRITE-TRANS-REC
IS RECORD KEY OUT-OF-SORT?
NO
YES
SAVE NEW HIGHEST KEY
INDICATE SORT REQUIRED
WRITE TRANS-ACTION FILE
EXIT

g. FMIPBIN

(1) Function. IPBIN is a binary search routine to look for a field name in LR11 of an FFT.

(2) Calling Sequence.

Called by 'IPCD' and 'IPUN'
Entry Point 'IPBIN'
Calling Sequence FM-DD IP-DD MAJOR-LR11

(3) Capabilities.

After entry point, FLD-LR11-LU is performed. CNT1 is initialized at 0 and CNT-LR (the total number of field names) plus one is divided in half and stored in TALLY.

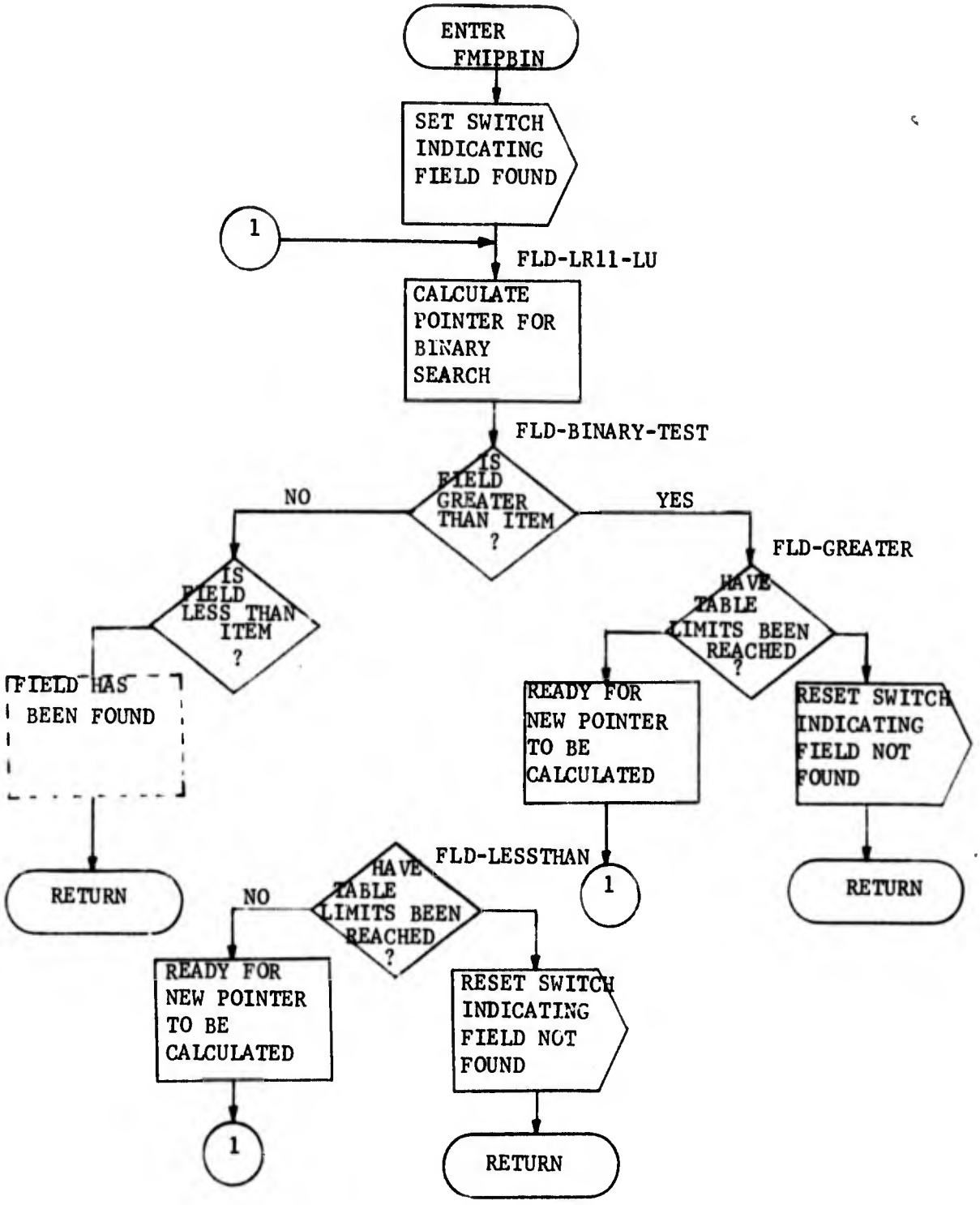
FLD-BINARY-TEXT adds TALLY to CNT1 and then compares the field name with the LR11 table entry at CNT1. If the names are equal, the processing is complete. If the field name is greater, the program proceeds to FLD-GREATER; if it is less, FLD-LESSTHAN.

FLD-GREATER computes a new TALLY value and loops back to FLD-BINARY-TEST. The new TALLY value is the smallest number which when multiplied by a power of two will result in a product value no less than CNT-LR. This assures that every entry in LR-11 required to be tested will actually be tested.

FLD-LESSTHAN computes a new TALLY value in the same manner as FLD-GREATER but negates the value so that an earlier table entry can be tested. Processing loops back to FLD-BINARY-TEST.

The processing continues with increasing powers of two until a match is made or until TALLY is less than 2 or greater than -2 indicating that the test name is not in the table.

(4) Flowchart.


ENTER FMIPBIN
SET SWITCH INDICATING FIELD FOUND
1
FLD-LR11-LU
CALCULATE POINTER FOR BINARY SEARCH
FLD-BINARY-TEST
IS FIELD GREATER THAN ITEM ?
NO
YES
FLD-GREATER
IS FIELD LESS THAN ITEM ?
HAVE TABLE LIMITS BEEN REACHED ?
FIELD HAS BEEN FOUND
READY FOR NEW POINTER TO BE CALCULATED
RESET SWITCH INDICATING FIELD NOT FOUND
RETURN
1
RETURN
FLD-LESSTHAN
HAVE TABLE LIMITS BEEN REACHED ?
NO
READY FOR NEW POINTER TO BE CALCULATED
RESET SWITCH INDICATING FIELD NOT FOUND
1
RETURN

h. FMIPSRT

(1) Function. To sort transaction file records.

(2) Calling Sequence. There is none. FMIPSRT is entered through its PROGRAM-ID and no parameters are passed.

(3) Capabilities.

FMIPSRT is dynamically loaded and executed by the File Maintenance Supervisor after FMIPX determines that the transaction file must be sorted. A sort key of 48 characters is used. (See figure 3-3-4 with the FMIPTRN subprogram for the format of the sort key.) The transaction file is directly sorted by a COBOL SORT statement with no INPUT PROCEDURE or OUTPUT PROCEDURE. NOTE: For the IBM 360, BLOCK CONTAINS 0 RECORDS is specified for the input and the output transaction files so that records can be blocked. A DCB BLKSIZE parameter is required on the SORTIN and SORTOUT DD cards at execution time to specify the maximum block size. The BLKSIZE value may be no smaller than 1060.

3. IP ERROR MESSAGES.

a. "ACCEPTING"

(1) FMIPVAL: CONTINUE-REAL, CON-OPR-2

(2) Advisory message indicating value accepted.

b. "BLANK GROUP ID ILLEGAL"

(1) FMIPREC: START-GROUP

(2) A group ID which contains all blanks was found.

c. "BLANK PSEUDO PSS ILLEGAL"

(1) FMIPUN: TEST-FOR-3

(2) Location in the Unit Update record normally containing the PSS number contains blanks.

d. "COMMA MISSING AFTER SUBROUTINE"

(1) FMIPCD: LOAD-CARD

(2) More than one subroutine name appears on a load card and is not separated by a comma.

e. "CONTINUED"

(1) FMIPVAL: CONTINUE

(2) An error has been detected, but processing will continue.

f. "CONTINUING"

(1) FMIPREC: CONTINUE

(2) An error has been detected, but processing is continuing.

g. "DATA SIZE EXCEEDS FIELD SIZE"

(1) FMIPUN: CHECK-SIZE

(2) Length of data portion of unit-record exceeds field length.

h. "DSD NAME OMITTED"

(1) FMIPCD: GET-DST

(2) Name of DSD did not follow DSD keywork on ACCEPT card.

i. "DSD NOT IN TABLE LIBRARY"

(1) FMIPCD: GET-DST, GET-DSD

(2) Librarian could not find DSP given within the user's library specified by JCL.

j. "EXPECTED CONTINUATION NOT FOUND"

(1) FMIPUN: NO-CONTINUATION, CHECK-CONTINUATION

(2) Continuation of a unit record input is expected when position 80 of the previous record is not blank. It is found when the new record matches the previous one from position through the record ID delimiter (first record-mark or 0-2-8 punch).

k. "FFT NOT IN TABLE LIBRARY"

(1) FMIPCD: READ-LR1

(2) Librarian could not find FFT given within the users library specified by JCL.

l. "FIELD DATA OMITTED"

(1) FMIPUN: MOVE-RECORD

(2) Zero length data field in a transaction not DEL or NEX.

m. "FIRST FMIP CARD MUST BE ACCEPT"

(1) FMIPCD

(2) First FMIP control card read was not an ACCEPT card.

n. "FLD DELETED"

(1) FMIPVAL: PSSQ-ILLEGAL

(2) An error has occurred, field rejected as per DSD description.

o. "FIELD REJECTED"

(1) FMIPVAL: CON-OPR-2

(2) An error has occurred, field rejected as per DSD description.

p. "FIELD WILL BE SKIPPED"

(1) FMIPVAL: CON-OPR-2

(2) An error has occurred and the field indicated will be skipped.

q. "FIRST REC MUST HAVE GROUP ID"

(1) FMIPREC: BUILD-GROUP-ID

(2) First record read had a blank group ID.

r. "FLD BLANKED"

(1) FMIPVAL: BLANK-INPUT-FIELD

(2) Field specified has had blanks moved to it.

s. "FLD REJECTED"

(1) FMIPVAL: DELETE-TRAN

(2) A field has been rejected due to an error.

t. "FMIP CNTRLEND CARD MISSING"

(1) FMIPCD: GET-NEXT-CARD

(2) A card other than an FMIP control card has been read without first indicating that all FMIP cards had been read with an FMIP CNTRLEND card.

u. "GROUP FREQUENCY ERROR"

(1) FRIPREC: GROUP-FREQ-ERR

(2) Frequency of a record is larger than specified on DSD group card.

v. "GROUP REJ"

(1) FRIPREC: REJECT-GROUP

(2) An error has occurred, group rejected as per DSD specifications.

w. "ID EXCEEDS LENGTH OF REC CTL"

(1) FMIPUN: BUILD-GROUP-ID-X

(2) A record ID has been specified larger than the record ID control field.

x. "ILLEGAL SOURCE CODE"

(1) FMIPCD: SOURCE-CARD

(2) Source code given on source card was not between 1 and 9.

y. "INDIVIDUAL CONFIRMS IGNORED"

(1) FMIPCD: CONFIRMS-CARD

(2) A CONFIRM ALL card has already been processed.

z. "INPUT GROUP REJECTED"

(1) FMIPVAL: ERR-CODE-G

(2) An error has occurred, input group rejected as per DSD description.

aa. "INVALID DSD NAME"

(1) FMIPCD: GET-DST

(2) DSD name given does not end in D.

ab. "INVALID FIELD NAME"

(1) FMIPCD: PROCESS-NEXT-FIELD

(2) Length of a field on a CONFIRM card exceeded 5 characters.

ac. "INVALID FILE SEQ/ORDER"

(1) FMIPREC: FILE-SEQ-ORD-ERR

(2) Values for file level, sequence or order, test are not in order.

ad. "INVALID FILE NAME"

(1) FMIPCD: ACCEPT-CARD

(2) Name of file on ACCEPT card does not end in A or T.

ae. "INVALID FMIP CONTROL CARD"

(1) FMIPCD: GET-NEXT-CARD

(2) An FMIP control card has been read which was not a CLASS, CONFIRM, LM, INHIBIT, SOURCE, RESERVE, CNTRLEND, SUBTRN, or LOAD card.

af. "INVALID GROUP SEQ/ORDER"

(1) FMIPREC: GROUP-SEQ-ORD-ERR

(2) Values for group level, sequence or order, test are not in order.

ag. "INVALID RECORD TYPE"

(1) FMIPREC: TEST-REC-TYPE

(2) Record code type does not match any in DSD.

ah. "INVALID SUBTRN SPECIFICATION"

(1) FMIPCD: SUB-ERR

(2) The word following SUBTRN must consist of a field name, an =, and a subroutine name example: FLD01=SUBRS. The field name must be longer than 5 characters and the subroutine name mmust end in S. The offending word appears as error word.

ai. "ITEM CNT"

(1) FMIPVAL: FORMAT-OUTPUT

(2) IP is trying to build a transaction in excess of the maximum number of transactions allowed for this installation.

aj. "LAST ITEM"

(1) FMIPVAL: ITEM-EXCEEDS-TABLE

(2) This message appears along with "ITEM CNT".

ak. "LOC SPEC NOT NUMERIC"

(1) FMIPVAL: CON-OPR-2, FORMAT-OPR-ID-TYPE

(2) A source data record field which is supposed to contain a numeric pointer, does not have numeric contents.

al. "MUST BE DSD OR UNIT"

(1) FMIPCD: NO-FOURTH

(2) Keyword of DSD or UNIT is missing from ACCEPT card.

am. "MULTIPLE CLASS CARDS NOT ALLOWED"

(1) FMIPCD: CLASS-CARD

(2) Only one class card is allowed per FMIP run.

an. "NEXT FLD"

(1) FMIPVAL: TRANSACTION-SKIPPED

(2) A transaction has been skipped, next field will be validated.

ao. "NO TRANSACTIONS GENERATED"

(1) FMIPX: CALL-IPSRT

(2) No transaction records have been created to be passed to FMMP. Normally due to user error.

ap. "PERMITTED ONLY FOR UNIT UPDATE"

(1) FMIPCD: SUBTRN-CARD

(2) A SUBTRN card is not allowed for batch update.

aq. "RECORD FREQUENCY ERROR"

(1) FMIPREC: RECORD-FREQ-ERR

(2) Frequency of a record is fewer than specified on DSD group card.

ar. "RECORD ID NOT COMPLETE"

(1) FMIPREC: CDMS-ID-ERR

(2) Not enough records (of various) types were supplied to produce a complete record ID.

as. "RECORD ID NOT SPECIFIED"

(1) FMIPUN: BUILD-GROUP-ID-X

(2) Record ID portion of first unit update record has zero length. Check for record mark (0-2-8) in first position.

at. "RECORD REJ"

(1) FMIPREC: REJECT-RECORD

(2) An error has occurred, record rejected as per DSD specifications.

au. "RECORD REJECTED"

(1) FMIPVAL: REJECT-RECORD-R

(2) An error has occurred, record rejected as per DSD description.

av. "REMAINING TEXT SKIPPED"

(1) FMIPVAL: REJECT-REMAINING-TEXT-OR-TRANS

(2) A record has been built which is larger than allowed, remaining text for record is skipped.

aw. "SOURCE OMITTED"

(1) FMIPCD: SOURCE-CARD

(2) Source code not given on SOURCE card.

ax. "SUBROUTINE LIMIT EXCEEDED"

(1) FMIPCD: LOAD-CARD

(2) More than 10 subroutines were requested to be loaded from LOAD cards.

ay. "SUBROUTINE MISSING AFTER COMMA"

(1) FMIPCD: LOAD-CARD

(2) A comma was punched after the last subroutine name listed, indicating more subroutine names to follow of which there were none.

az. "SUBTRN MAXIMUM EXCEEDED"

(1) FMIPCD: SUB-CHECK

(2) Maximum number of subroutines from SUBTRN card per run was exceeded.

ba. "SUBTRN NAME MUST END WITH S"

(1) FMIPCD: SUB-CHECK

(2) Subroutine name given on SUBTRN card does not end in S.

bb. "SYS ERROR"

(1) FMIPVAL: SYSTEM-ERROR, ERR-CODE-S, ERROR-CODE-EQUALS-A

(2) Program malfunction. A programmer should be consulted.

bc. "UNDEFINED FIELD NAME"

(1) FMIPCD: FLD-NOT-FOUND

(2) Field given on CONFIRM card could not be found within the FFT of the user library specified.

bd. "UNDEFINED FIELD NAME"

(1) FMIPUN-PROCESS-RECORD

(2) A field name has been specified which is not in the FFT. The field name appears as error-word.

be. "VSET CHG CTL INVALID"

(1) FMIPUN: PROCESS-FIELD-LEN

(2) Control field for a variable set change is not valid.

bf. "ZERO HOP ILLEGAL FOR VSET"

(1) FMIPVAL: CON-OPR-2

(2) A variable set change control field has a zero as a high order position.

CHAPTER 3

Section IV

File Maintenance Maintenance Proper (FMMP)

1. OVERVIEW. File Maintenance Maintenance Proper (FMMP) generates or updates a MIDMS data file. It is divided into two general areas: Ordinary Maintenance (OM) and Logical Maintenance (LM). OM uses the transaction file produced by Input Processor to specify records and fields to be modified. LM uses a load module produced from a COBOL program originally generated by Language Processor to determine the specific changes to the MIDMS file.

a. Maintenance Proper consists of a series of load modules which are loaded and deleted as needed. They are dynamically loaded and executed by either the File Maintenance Supervisor (the FM load module) or FMMP (file Maintenance Maintenance Proper Supervisor). In addition, the FMIO load module is used for reading FMMP control cards and printing confirmations and error messages. Figure 3-4-1 shows the detailed structure of the load modules and their relationship to FM, FMMP and FMIO. The primary functions of the load modules/programs controlled by FMMP are as follows:

b. FMMPX is the Maintenance Proper Supervisor and controls the execution of Ordinary Maintenance and subroutine mode Logical Maintenance. In addition, it handles the reading and writing of the MIDMS file being updated.

c. MPCD1 and MPCD2 analyze the parameters on FMMP control cards, set appropriate switches, and read File Format and LM Define Tables. Modification of LM define values is also handled by MPCD2 in accordance with redefine card specifications.

d. OMMPX performs the detailed field updates as specified by the transactions in the transaction file.

e. OMMPTIO reads the transaction file and acts as intermediary between FMMPX and OMMPTRN.

f. OMMPTRN causes the reading of transaction records, reconstructs the transaction table and text area originally produced in FMIP, and logically combines multiple transaction groups applying to the same MIDMS file record.

g. LMMPX handles Logical Maintenance processing by causing the dynamic loading and execution of Logic Package load modules.

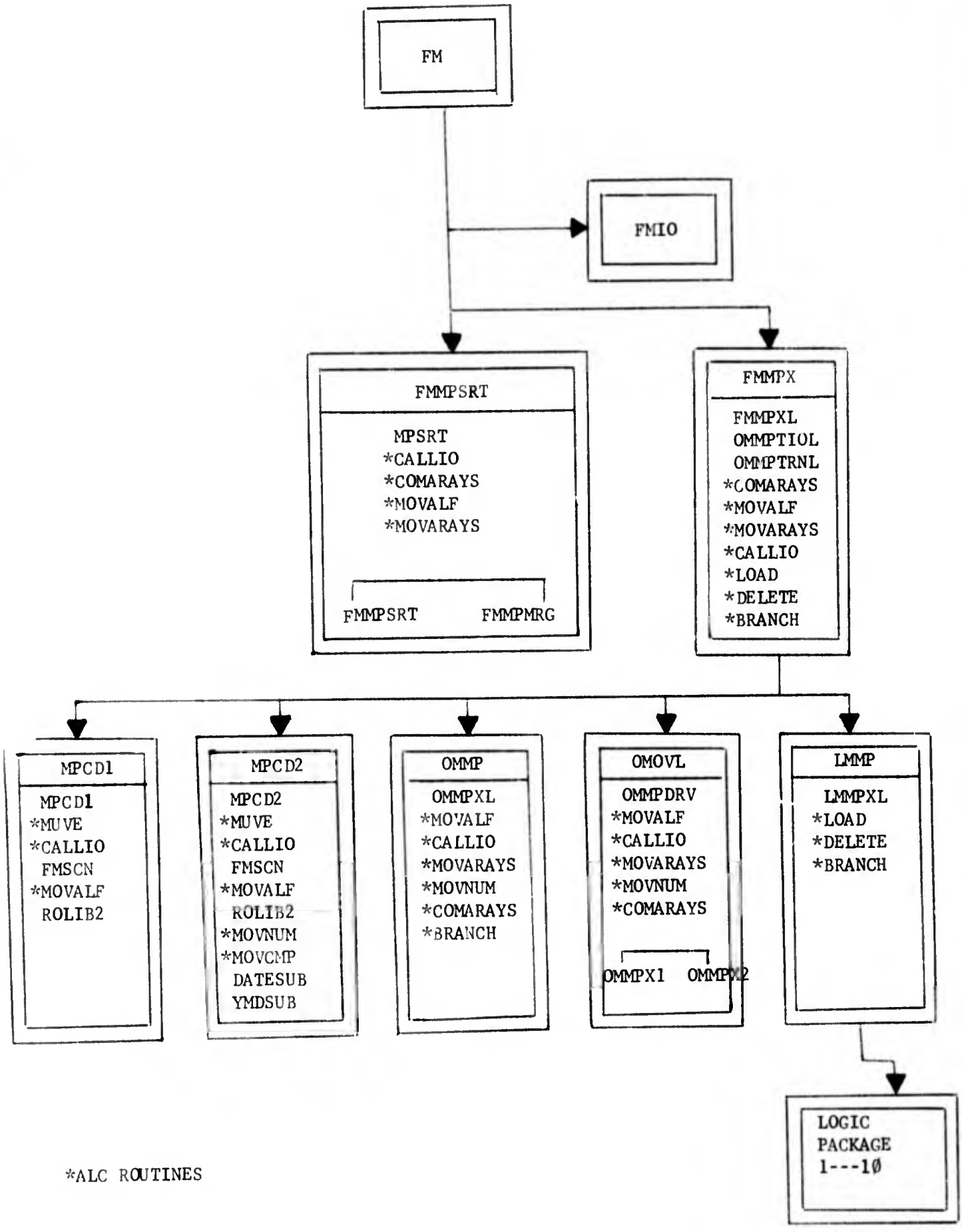


Figure 3-4-1. FM File Maintenance Proper (FMMP) Structure

h. MPSRTX calls the FMMPSRT and FMMPMRG subprograms when records in the updated file must be sorted.

i. FMMPSRT sorts keys pointing to a file of out-of-sort records from the MIDMS file.

j. FMMPMRG merges the just-sorted out-of-sort records with the main MIDMS file.

k. The relationships between the FFMP subprograms are indicated by figure 3-4-2.

2. FILE MAINTENANCE PROPER (FMMP) SUBPROGRAMS.

a. FMMPX

(1) Function. To control Ordinary Maintenance and subroutine mode Logical Maintenance execution and to read and write the MIDMS file being updated.

(2) Calling Sequence.

```
ENTRY   'FMMP'      USING FM-DD.
CALL    'FMSET'     USING FMIO.
CALL    'FMPRT'     USING PRT-LINE CC.
CALL    'MOVARAYS'  USING ...*
CALL    'COMARAYS'  USING ...*
CALL    'DELETE'    USING ...*
CALL    'MOVALF'    USING ...*
CALL    'LOAD'      USING ...*
CALL    'BRANCH'    USING ...*
CALL    'MPTIO'     USING MP-DD AREA2-BUFFER.
```

*The subroutines MOVARAYS, COMARAYS, and MOVALF are called several times by FMMPX.

(3) Program Description.

FMMPX.

Is the first program executed for the Maintenance Proper process. FMMP, the entry point of FMMPX, is also the entry point for the FMMP load module which is dynamically loaded by the File Maintenance Supervisor when a Maintenance Proper control card is read.

The beginning of FMMPX execution consists of five calls to subroutines which initialize the program processing. FMSET inserts the FMIO subprogram entry address into the CALLIO

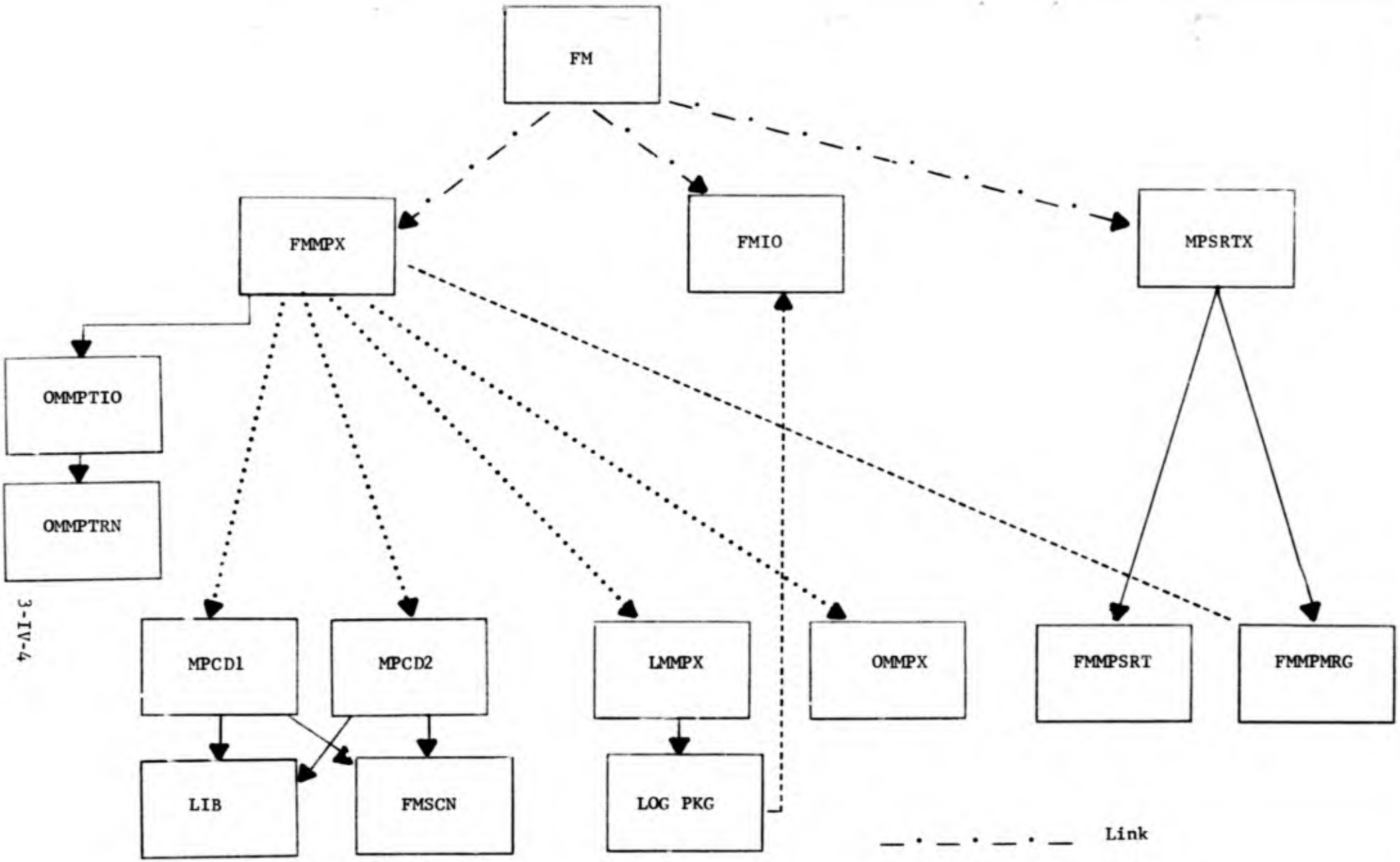


Figure 3-4-2. FM Maintenance Proper (FMMP) Subprogram Relationships

subroutine so that subsequent calls in MP to FMCRD and FMPRT will result in branching to FMIO.

MPCD1.

Analyzes the FMMP control cards and determines how much core is needed for LR2, LR3 and the defines for logical maintenance. MPCD1 then returns to FMMP with this computed size. MPCD1 is deleted. AREA1 is then opened with the computed size as its blocksize. MPCD2 is loaded and branched to. MPCD2 processes any LM redefines and reads from the library LR2, LR3, and any LM defines into AREA1. Upon return to FMMP, MPCD2 is deleted. Any subroutines that were specified by the LOAD option on the FFMP card are permanently loaded for this run; two calls to MOVARAYS clear IBM 360 DCB LRECL and BLKSIZE fields for the input and output MIDMS data files so that records can be blocked and buffer space can be allocated based on parameters at execution time in Job Control Language cards.

HALL-INIT thru READ-HIST-LOOP.

Handle history file processing. The old history file is first read and copied onto the new history file. When a record is changed in the maintenance run it will be appended to the new history file in paragraphs WRITE-HIST thru HIST-EXIT.

Regular processing starts by opening the MIDMS output file so that either Ordinary Maintenance or Logical Maintenance will have an area to place the output record. MPTIO is called. The transacting file is opened and the first transaction is fetched. AREA2 is used similar to AREA1 only for transactions. An input file is not opened for an OM create run (OMCRE)--OMMP is called directly. If a file revision LM run is indicated, no OM is needed so LMMP is called.

CALL-RDMAJ.

This paragraph performs the paragraphs required to read an input file record.

TEST-OM.

Initializes record-processing switches and determines if OM is to be executed: if there is no transaction for the record read then LM is checked.

The next series of paragraphs determines what is needed to be loaded, deletes modules that are not needed, and loads the modules that are needed.

NO-MAJOR-FILE.

Is used to process all remaining transactions when either there is no previous major file (OMCRE) or when the input file has reached an end-of-file condition.

TEST-IDS.

Determines if there is a transaction for the input record just read; if so, OMMP is called to process it.

TEST-LM.

Determines first if there was any change in the record through OMMP; if so, change-date is maintained. Secondly, if there is no LM indicated, the record is written to the output file; finally, the record is considered as changed (since FMMP cannot foresee what LM does to it) and is passed to LMMP.

The next several paragraphs determine which load modules are present; unnecessary modules are deleted while absent modules which are needed get loaded.

CALL-WRTMAJ.

Writes the record, then passes to TEST-TRANS-SW to determine if there are more transactions or MIDMS-file input records. If there are either, control loops back to the appropriate paragraph to continue processing; otherwise, the output file is closed and statistics are printed giving the number of records in the input file; the number generated, deleted, and updated; the number in the output file; and the maximum record size of the records in the output file.

FINISH-FMMP.

Returns control to the File Maintenance Supervisor upon completion of Maintenance Proper.

SUBR-DELETE.

Deletes any subroutines that were permanently loaded.

SORT-CALL.

Sets FM-LINK-SW to 2 when out of sort records occurred. This signals FM to execute the Maintenance Proper sort.

RDMAJ.

Reads a MIDMS file record. If the input file is not open, it is opened before reading. It also puts the record-ID of the record read into PRESENT-RECID.

SET-EOF.

Sets the end of file switch (MAJOR-EOF-SW) when the input file has been completely read.

RESET-SWITCHES.

Clears LM switches SWITCHA to SWITCHZ for each record.

WRTMAJ.

Is the beginning of the routine to write an output record. If the input record is to be deleted from the file, control passes to DELETE-RECORD. If the record ID of an input record was changed for output (CID operation), the updated record is assumed to be out of sort and is handled by WRT-UNSORT thru FMMP-2A. When the unsorted file is to be opened, those load modules which are loaded and deleted and are presently loaded must be deleted prior to opening the unsorted file. They then must be loaded again. This will prevent core fragmentation. If no change was made to an input record, processing continues with NO-CHANGE. Otherwise, the output record ID is compared

(through COMARAYS) with the previously written ID with one of three possible outcomes: If the last record ID is less than the current one, WRT-OUT is executed; if equal, control passes to RECID-ERROR; if greater, WRT-UNSORT. If the record was generated, the create date is inserted in the record.

WRT-OUT.

Writes an updated record and saves its record-ID in LAST-RECID.

WRT-UNSORT thru FMMP-2A.

Are used to write records which are out of sort relative to the main output file. There are two files associated with this capability: UNSORTED-FILE and KEY-FILE. UNSORTED-FILE, in the IBM 360, is a variable length unblocked ORGANIZATION DIRECT file with a maximum record size of 2004 characters. A MIDMS record larger than 2004 characters is subdivided into segments with the following characteristics: The first 2000 characters of the MIDMS record are placed in the first segment record but the record is given a special size of 2001 characters. The balance of the MIDMS record is placed in the succeeding segments. Each record segment is assigned a SYMBOLIC KEY of its sequence number within the file. (The ACTUAL KEY -- track number -- is handled automatically by IBM 360 output routines.) KEY-FILE is an unblocked, fixed length record file containing the record ID's of out-of-sort records and their keys to the UNSORTED-FILE. KEY-FILE will be sorted in FMMPSRT when normal MP processing is complete.

RECID-ERROR.

Moves the current record ID into an error message area because a duplicate record ID has occurred.

MOVE-ERROR.

Moves the words "DUPLICATE RECORD ID, 2ND RECORD DELETED" into the error message area and causes the printing of the diagnostic. The record with the duplicate ID is not written into the output file.

CLSMAJ.

Starts the process of closing files in FMMPX. At the present time, it only contains a NOTE regarding a possible future requirement to specifically handle as deletes the portion of an input data file which is not read in by file revision Logical Maintenance.

CLOSE-FILES.

Closes all of the open files handled by FMMPX. When there are out-of-sort records, the number of records written on UNSORTED-FILE is printed.

FINISH-INOUT.

Is the last paragraph executed for the perform of both reading and writing MIDMS file records.

DELETE-RECORD.

Clears DELETE-RECORD SW and goes to FINISH-INOUT. In the future, specific additional processing of record deletes may be required in this paragraph.

DELETE-RECORDS.

This paragraph is for a possible future capability (see CLSMAJ).

NO-CHANGE.

WRT-OLD.

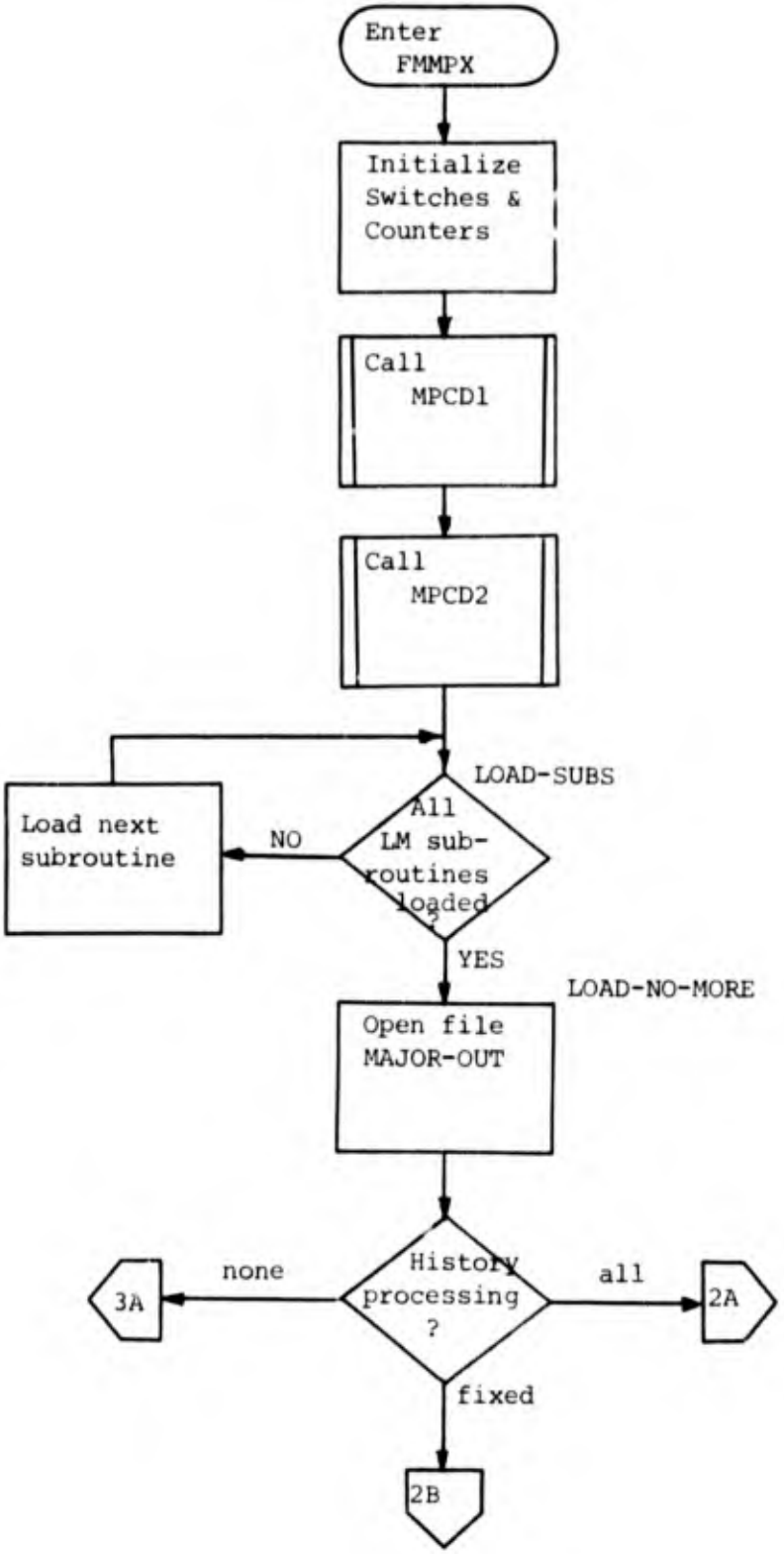
OLD-UNSORT.

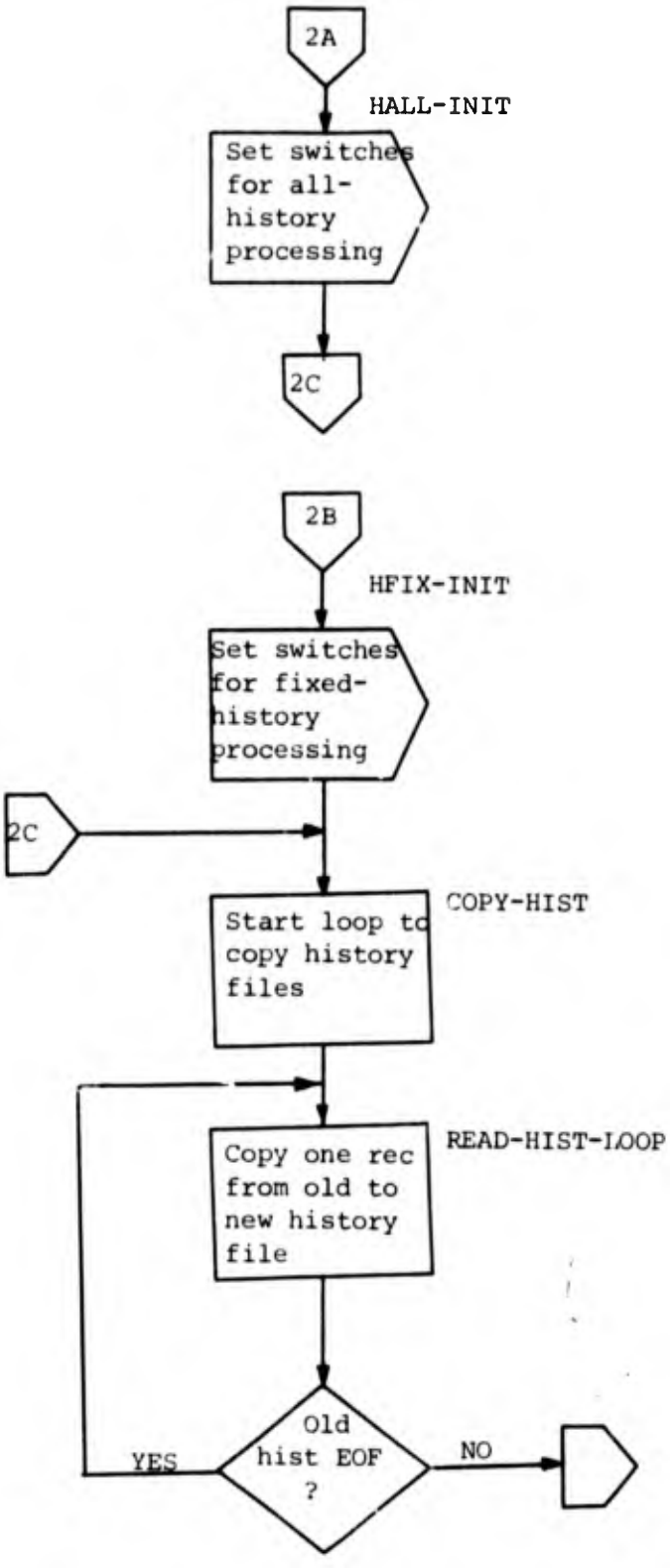
OLDID-ERROR.

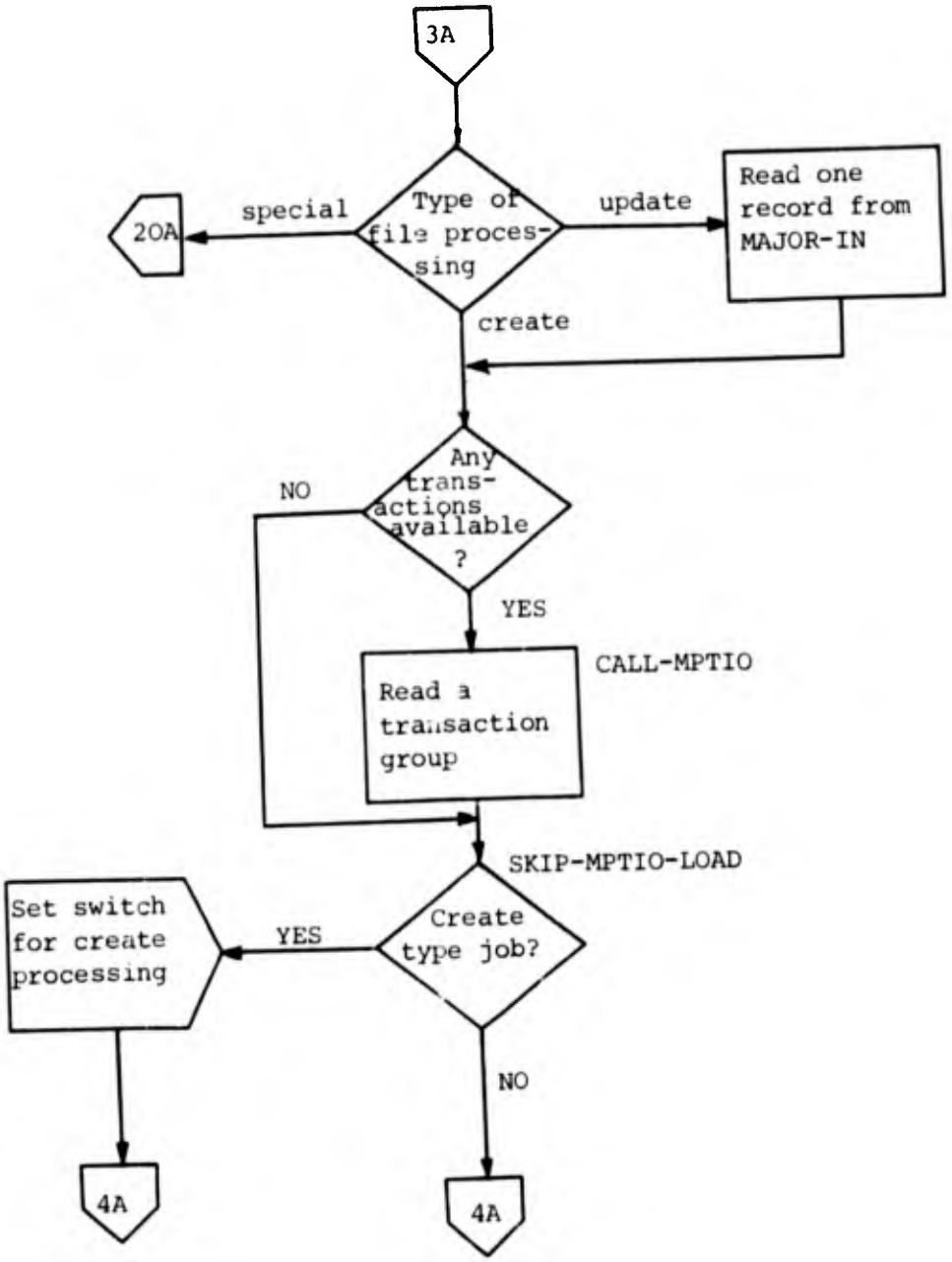
Have the same functions as the end of WRTMAJ, WRT-OUT, WRT-UNSORT, and RECID-ERROR. The only difference between these paragraphs is that the "OLD" paragraphs handle the writing of non-updated MIDMS file records.

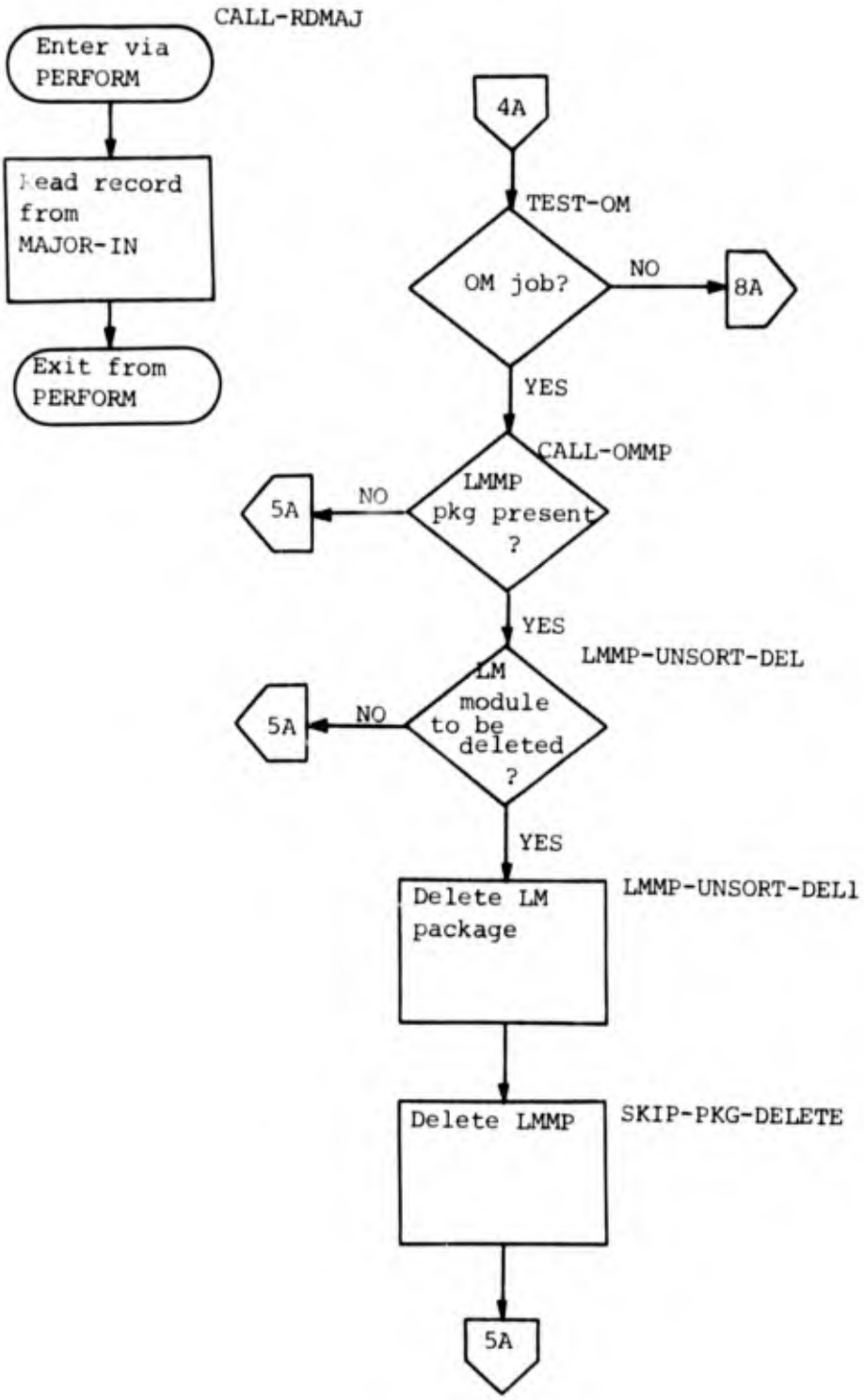
The greater part of the Working-Storage Section of FMMPX is contained in the COBOL COPY library under the name MPDDV. MPDDV has definitions for all fields used by FMMPX as well as those required between calls to other MP subprograms. MPDD is the COPY library name for the Linkage Section of most other MP routines. Under the group name MP-DD, MPDD includes a field by field and character by character duplicate of MPDDV except that MPDD excludes VALUE clauses. In addition, MPDD includes definitions for the input and output MIDMS records. These are associated with the FD records in FMMPX through the calls to routines requiring these records.

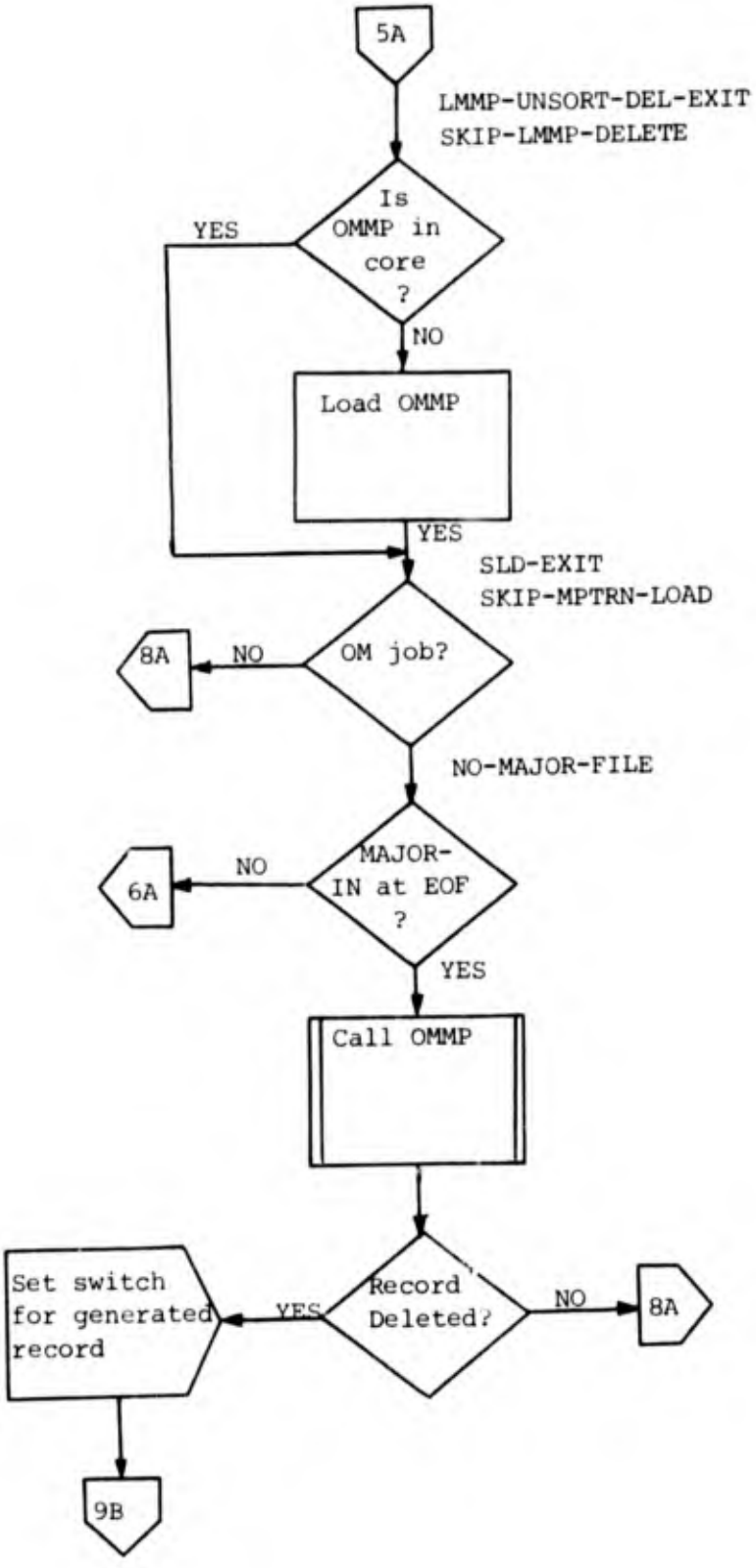
(4) Limitations. None.

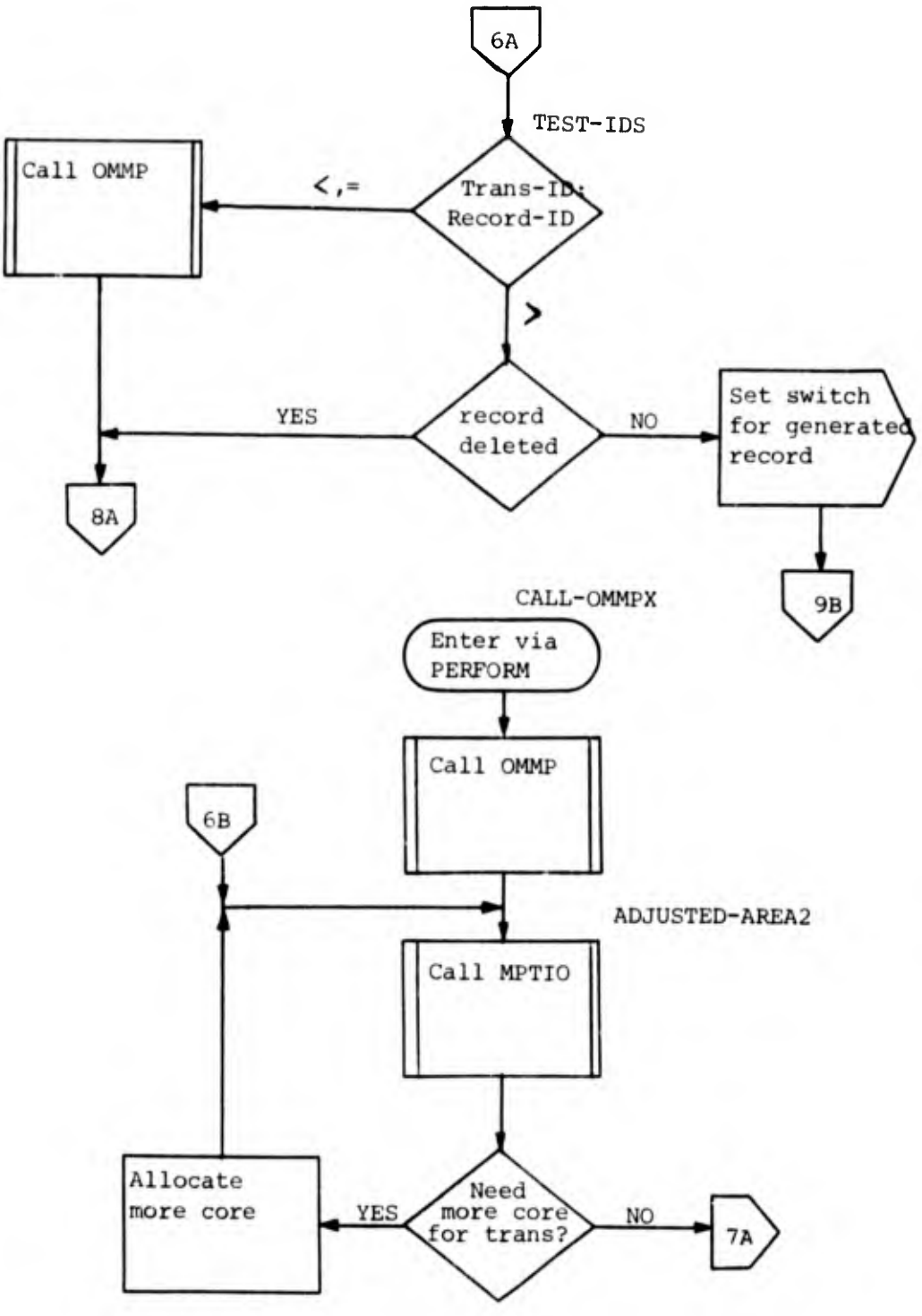

Enter
FMMPX
Initialize
Switches &
Counters
Call
MPCD1
Call
MPCD2
LOAD-SUBS
All
LM sub-
routines
loaded
?
NO
Load next
subroutine
YES
LOAD-NO-MORE
Open file
MAJOR-OUT
History
processing
?
none
3A
all
2A
fixed
2B

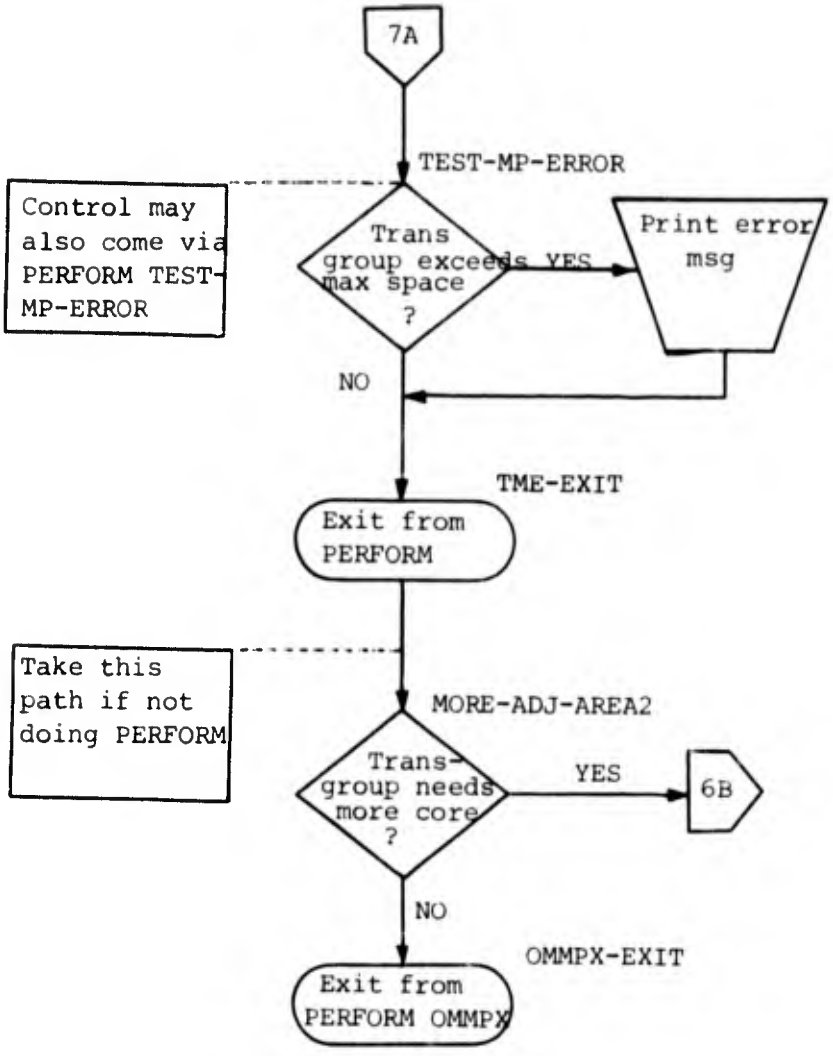

2A
HALL-INIT
Set switches for all-history processing
2C
2B
HFIX-INIT
Set switches for fixed-history processing
2C
COPY-HIST
Start loop to copy history files
READ-HIST-LOOP
Copy one rec from old to new history file
Old hist EOF ?
YES
NO

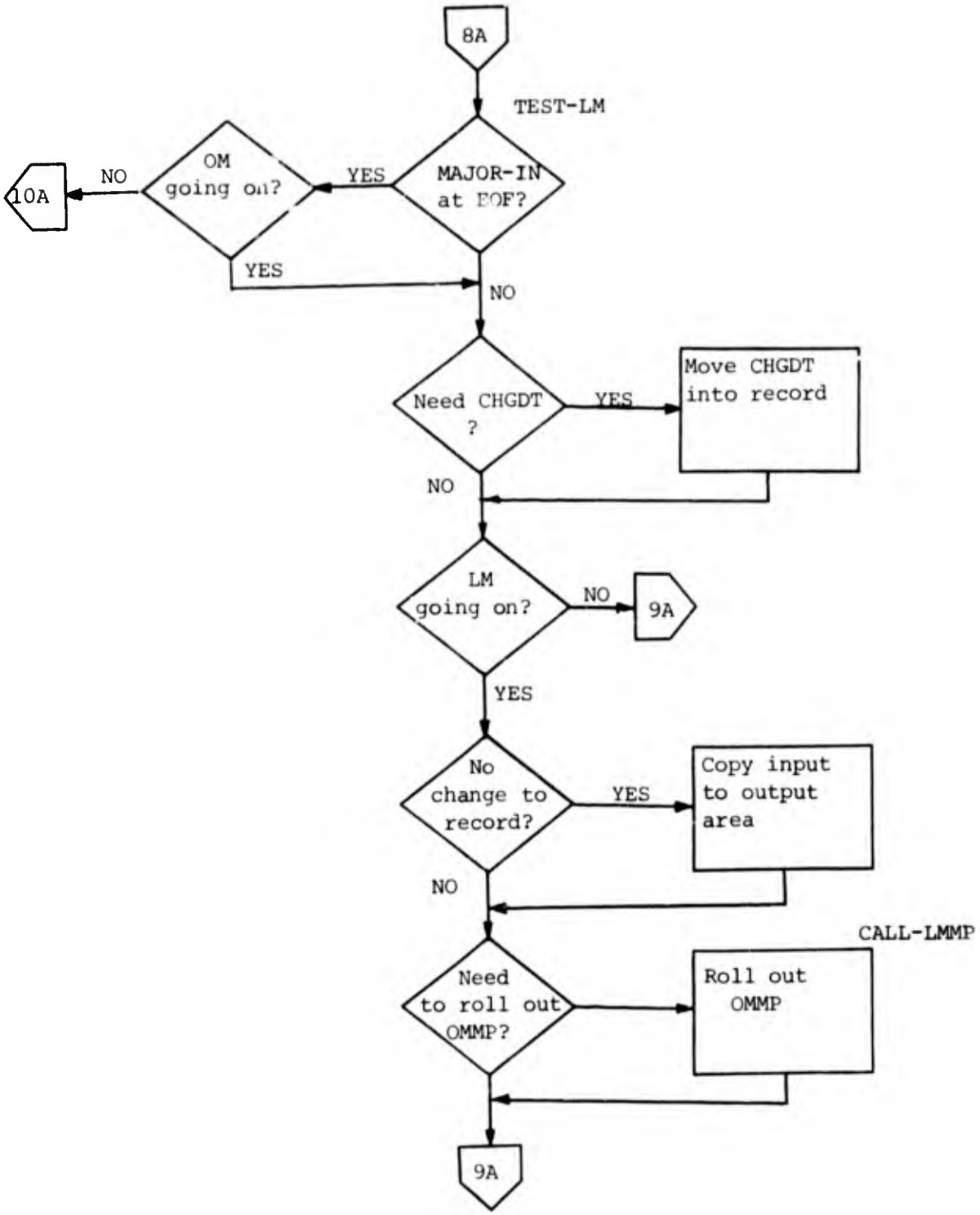

3A
Type of file processing
special
20A
update
Read one record from MAJOR-IN
create
Any transactions available ?
NO
YES
CALL-MPTIO
Read a transaction group
SKIP-MPTIO-LOAD
Create type job?
YES
Set switch for create processing
NO
4A
4A

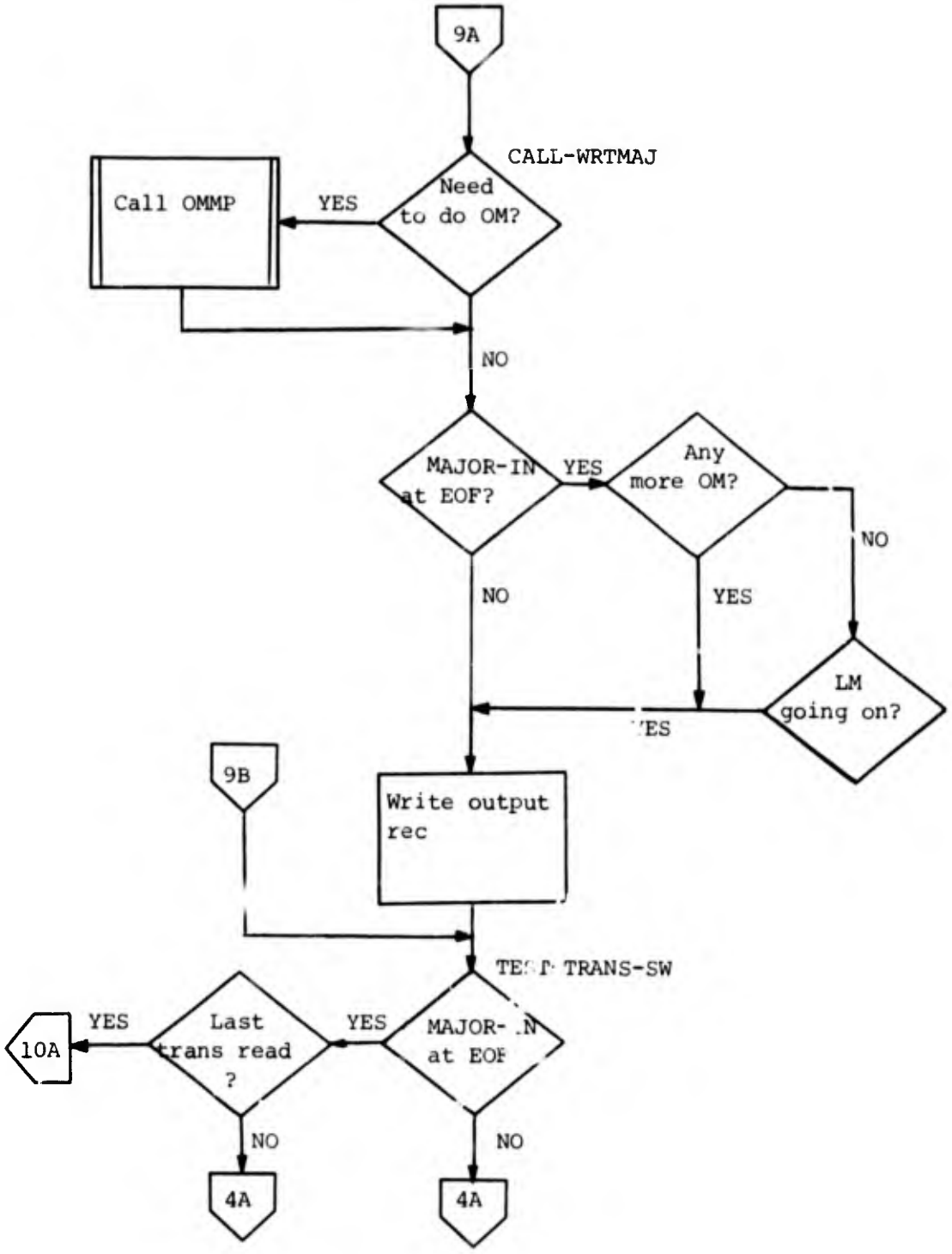

CALL-RDMAJ
Enter via PERFORM
Read record from MAJOR-IN
Exit from PERFORM
4A
TEST-OM
OM job?
NO
8A
YES
CALL-OMMP
LMMP pkg present ?
NO
5A
YES
LMMP-UNSORT-DEL
LM module to be deleted ?
NO
5A
YES
Delete LM package
LMMP-UNSORT-DEL1
Delete LMMP
SKIP-PKG-DELETE
5A

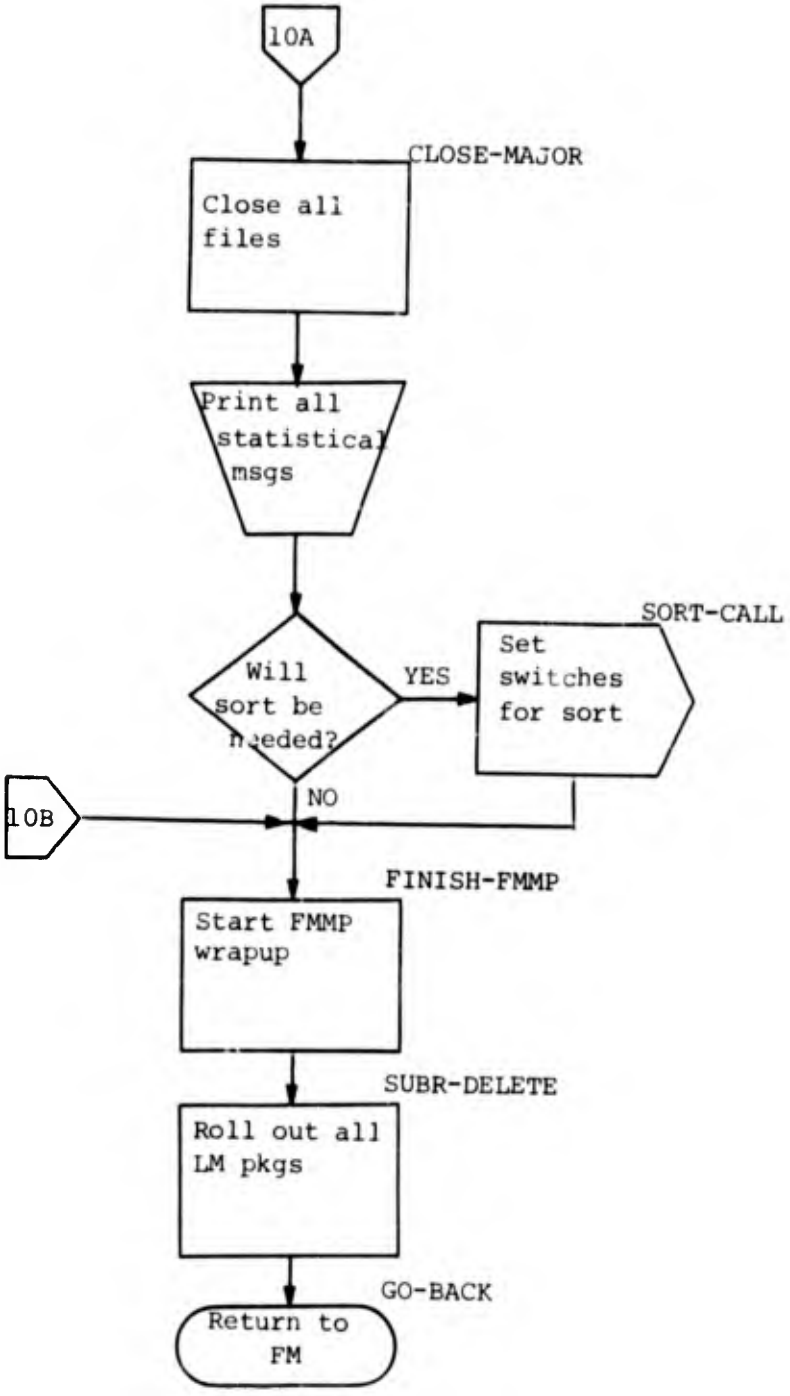

5A
LMMP-UNSORT-DEL-EXIT
SKIP-LMMP-DELETE
Is OMMP in core ?
YES
NO
Load OMMP
YES
SLD-EXIT
SKIP-MPTRN-LOAD
OM job?
8A
NO
NO-MAJOR-FILE
MAJOR-IN at EOF ?
6A
NO
YES
Call OMMP
Record Deleted?
YES
Set switch for generated record
NO
8A
9B

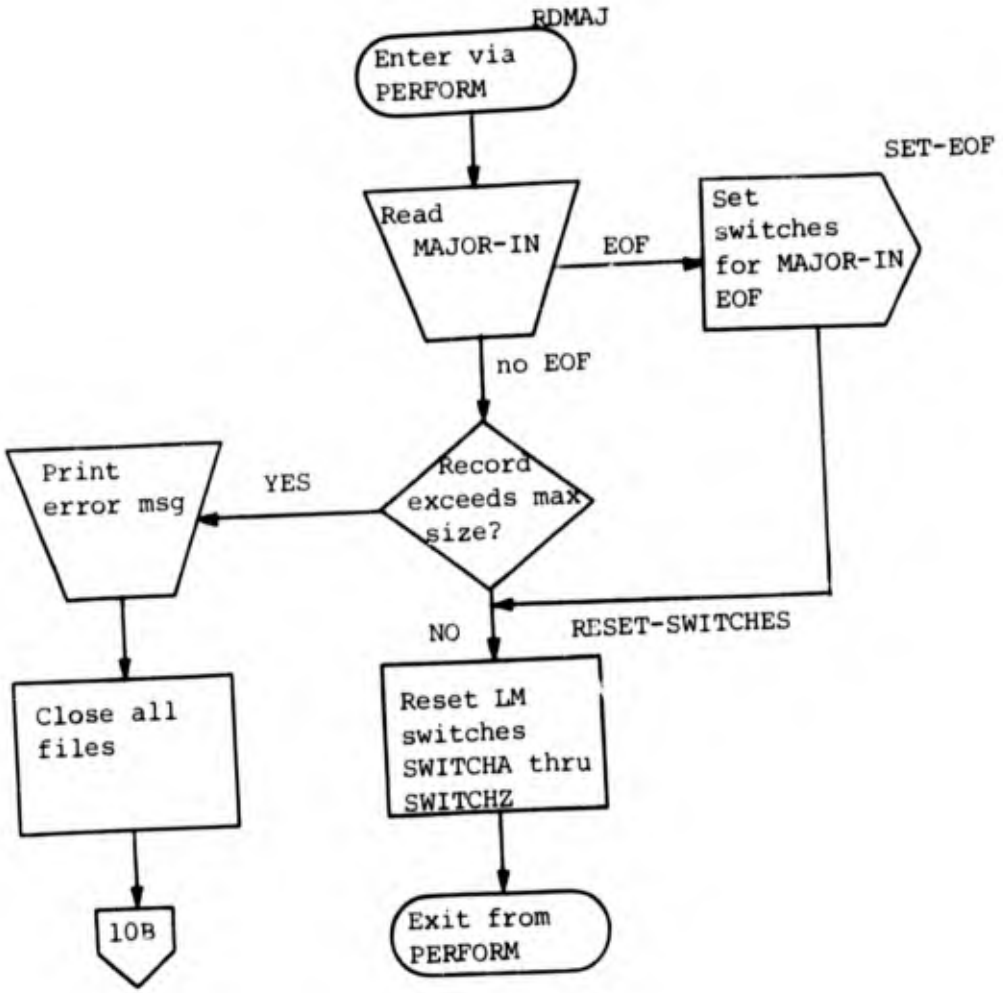

6A
TEST-IDS
Call OMMP
<,=
Trans-ID:
Record-ID
>
YES
record
deleted
NO
Set switch
for generated
record
8A
9B
CALL-OMMPX
Enter via
PERFORM
Call OMMP
6B
ADJUSTED-AREA2
Call MPTIO
Allocate
more core
YES
Need
more core
for trans?
NO
7A

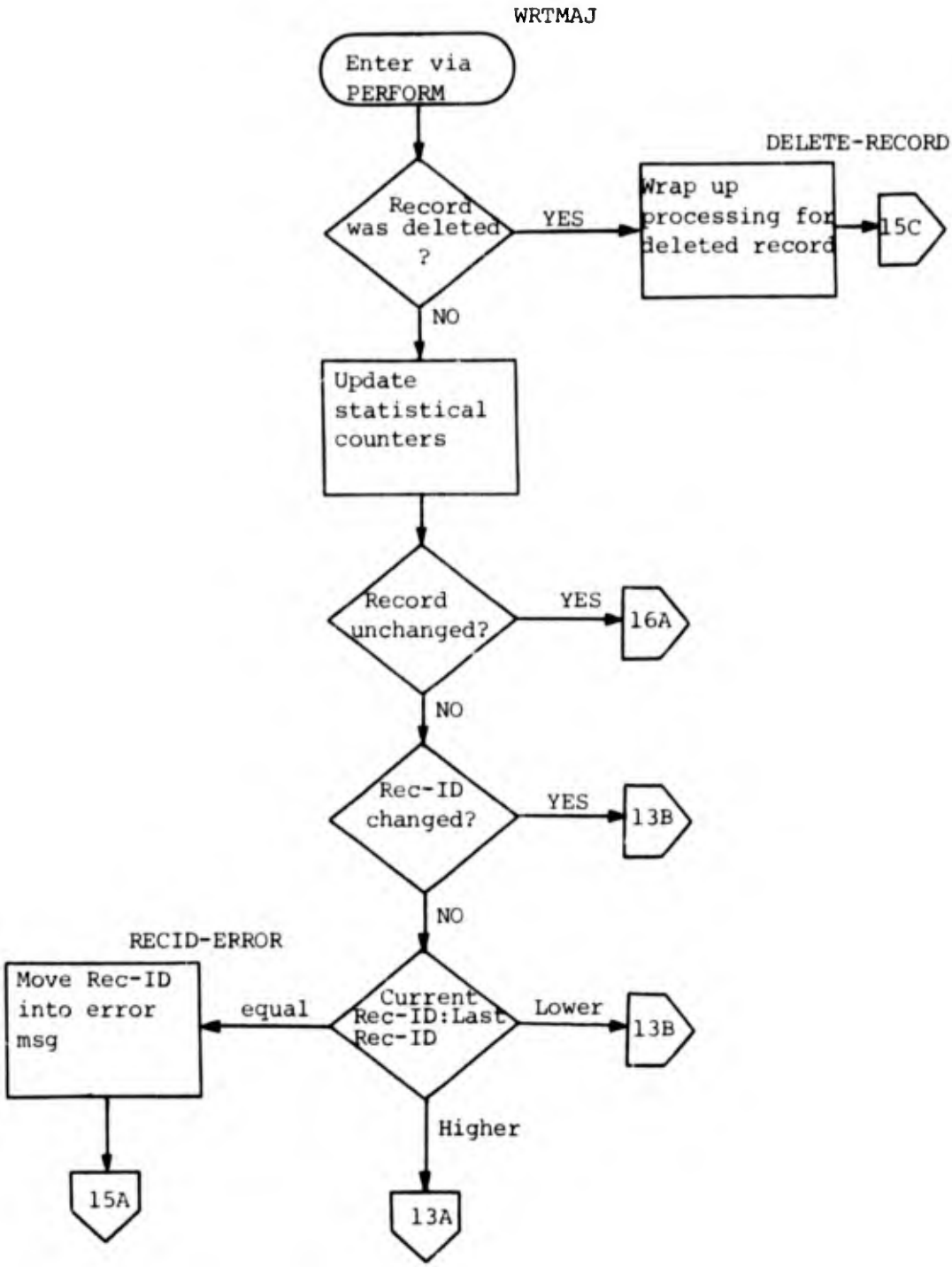

7A
TEST-MP-ERROR
Control may also come via PERFORM TEST-MP-ERROR
Trans group exceeds max space ?
YES
Print error msg
NO
TME-EXIT
Exit from PERFORM
Take this path if not doing PERFORM
MORE-ADJ-AREA2
Trans-group needs more core ?
YES
6B
NO
OMMPX-EXIT
Exit from PERFORM OMMPX

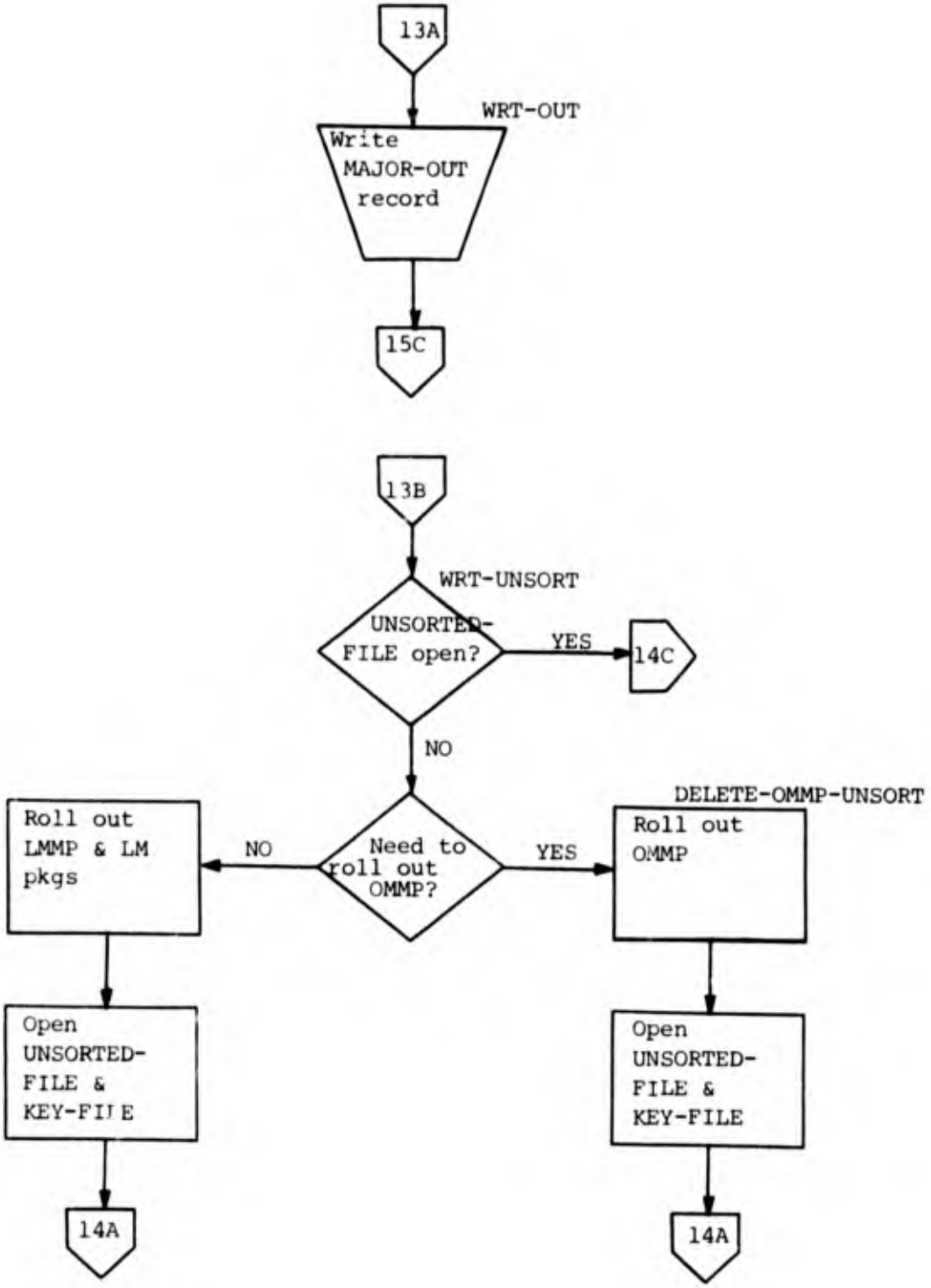

8A
TEST-LM
MAJOR-IN at EOF?
YES
OM going on?
NO
10A
YES
NO
Need CHGDT ?
YES
Move CHGDT into record
NO
LM going on?
NO
9A
YES
No change to record?
YES
Copy input to output area
NO
CALL-LMMP
Need to roll out OMMP?
Roll out OMMP
9A

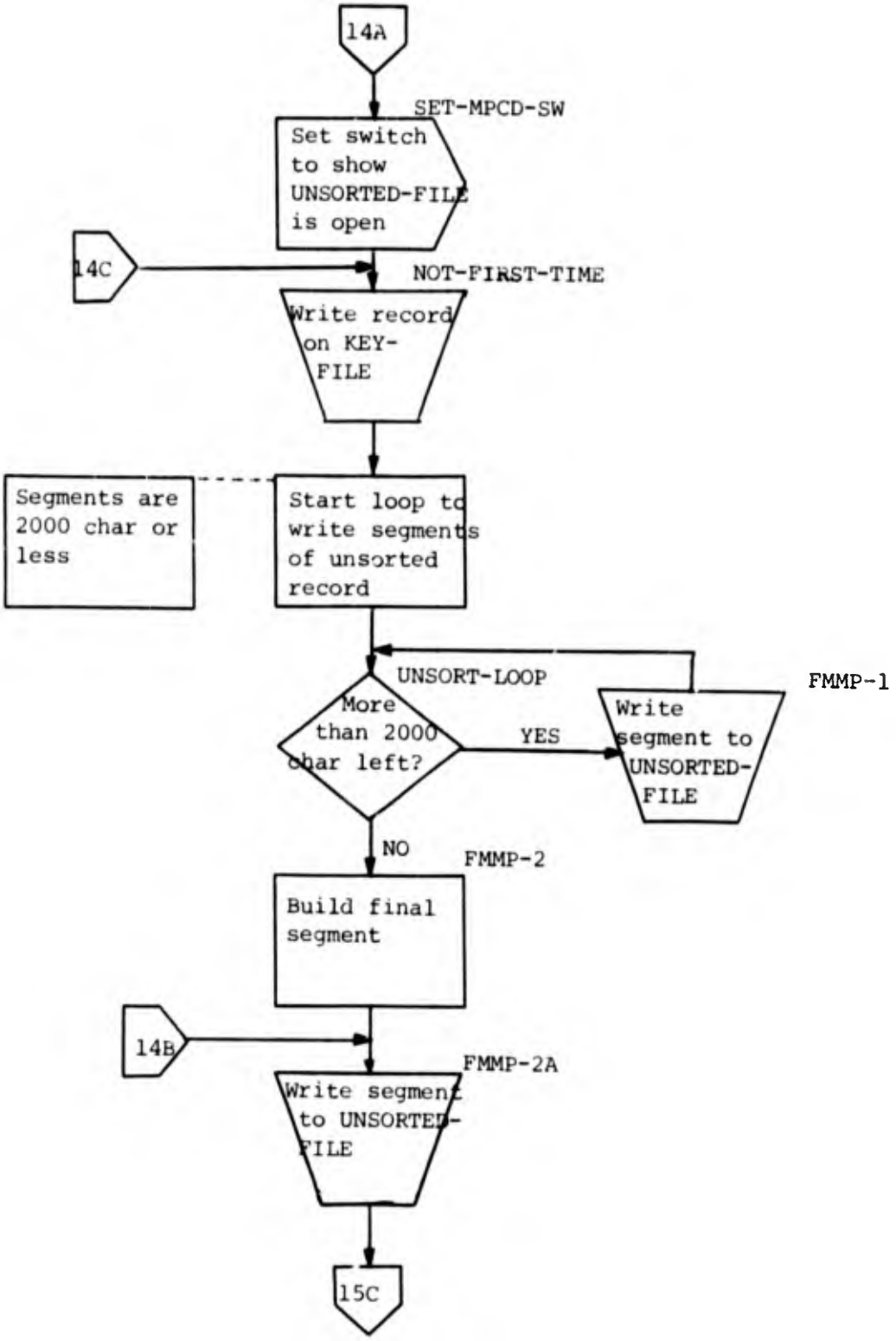

9A
CALL-WRTMAJ
Need to do OM?
YES
Call OMMP
NO
MAJOR-IN at EOF?
YES
Any more OM?
NO
YES
NO
LM going on?
YES
9B
Write output rec
TEST TRANS-SW
YES
Last trans read ?
YES
MAJOR-IN at EOF
10A
NO
NO
4A
4A

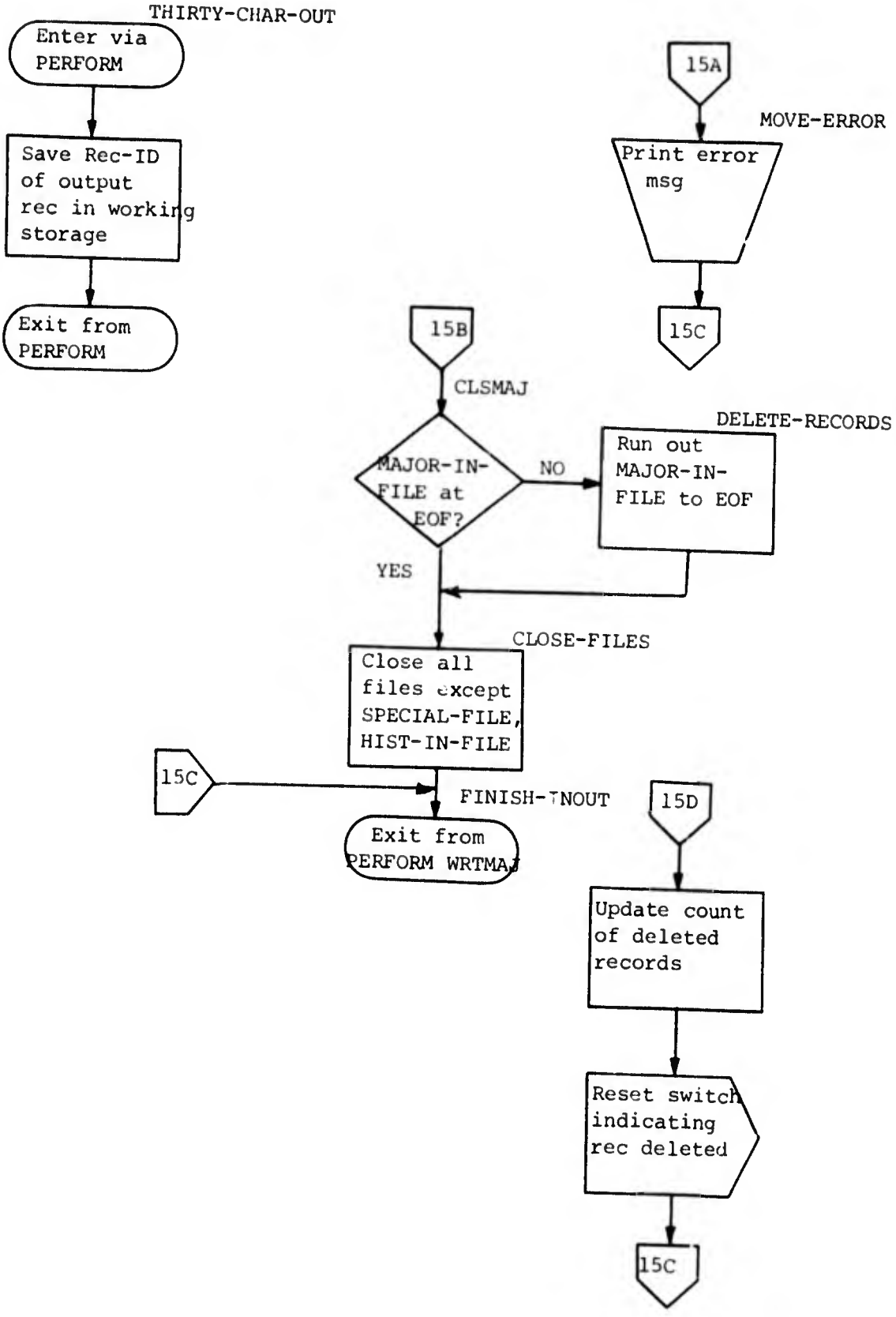

10A
CLOSE-MAJOR
Close all files
Print all statistical msgs
SORT-CALL
Will sort be needed?
YES
Set switches for sort
NO
10B
FINISH-FMMP
Start FMMP wrapup
SUBR-DELETE
Roll out all LM pkgs
GO-BACK
Return to FM

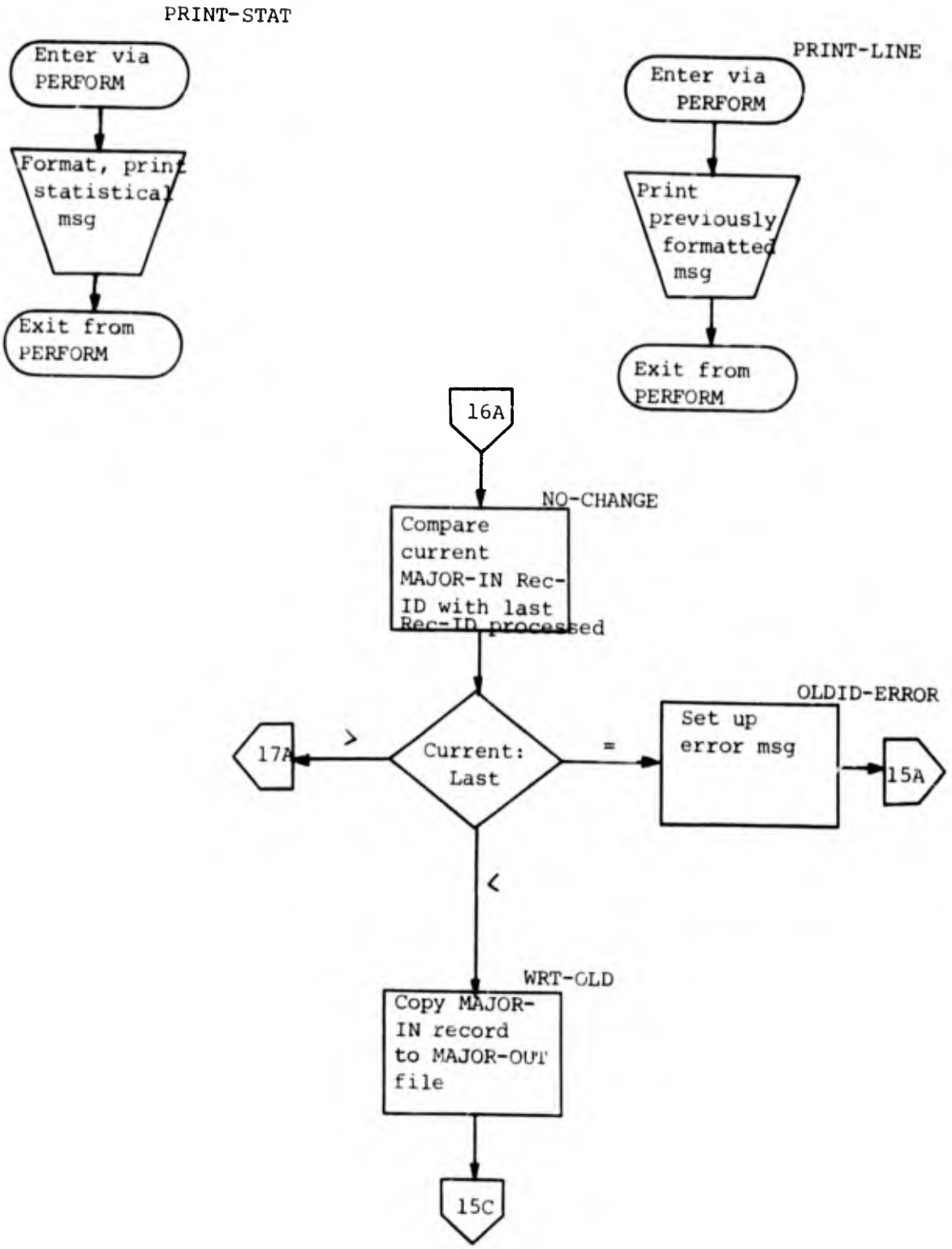

RDMAJ
Enter via PERFORM
Read MAJOR-IN
EOF
SET-EOF
Set switches for MAJOR-IN EOF
no EOF
Record exceeds max size?
YES
Print error msg
NO
RESET-SWITCHES
Close all files
Reset LM switches SWITCHA thru SWITCHZ
10B
Exit from PERFORM

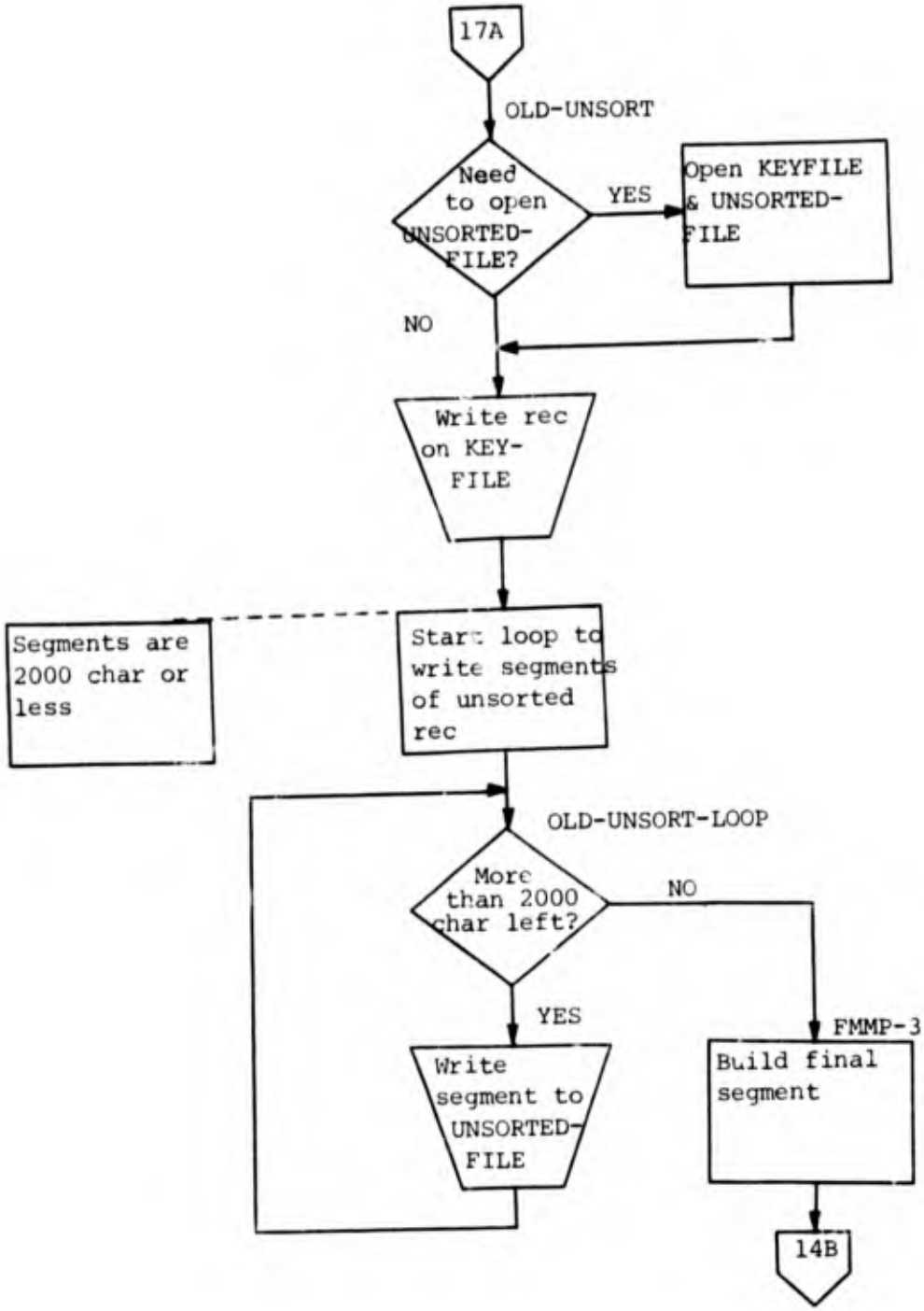

WRTMAJ
Enter via PERFORM
DELETE-RECORD
Record was deleted ?
YES
Wrap up processing for deleted record
15C
NO
Update statistical counters
Record unchanged?
YES
16A
NO
Rec-ID changed?
YES
13B
NO
RECID-ERROR
Move Rec-ID into error msg
equal
Current Rec-ID:Last Rec-ID
Lower
13B
Higher
15A
13A

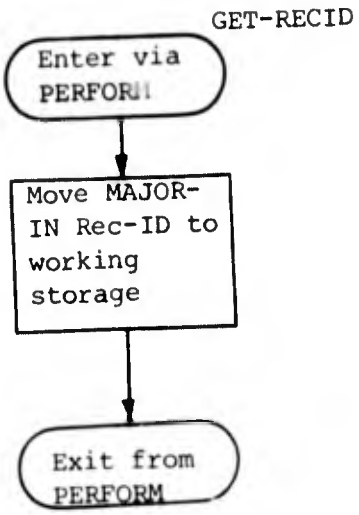

13A
WRT-OUT
Write MAJOR-OUT record
15C
13B
WRT-UNSORT
UNSORTED-FILE open?
YES
14C
NO
DELETE-OMMP-UNSORT
Roll out LMMP & LM pkgs
NO
Need to roll out OMMP?
YES
Roll out OMMP
Open UNSORTED-FILE & KEY-FILE
Open UNSORTED-FILE & KEY-FILE
14A
14A

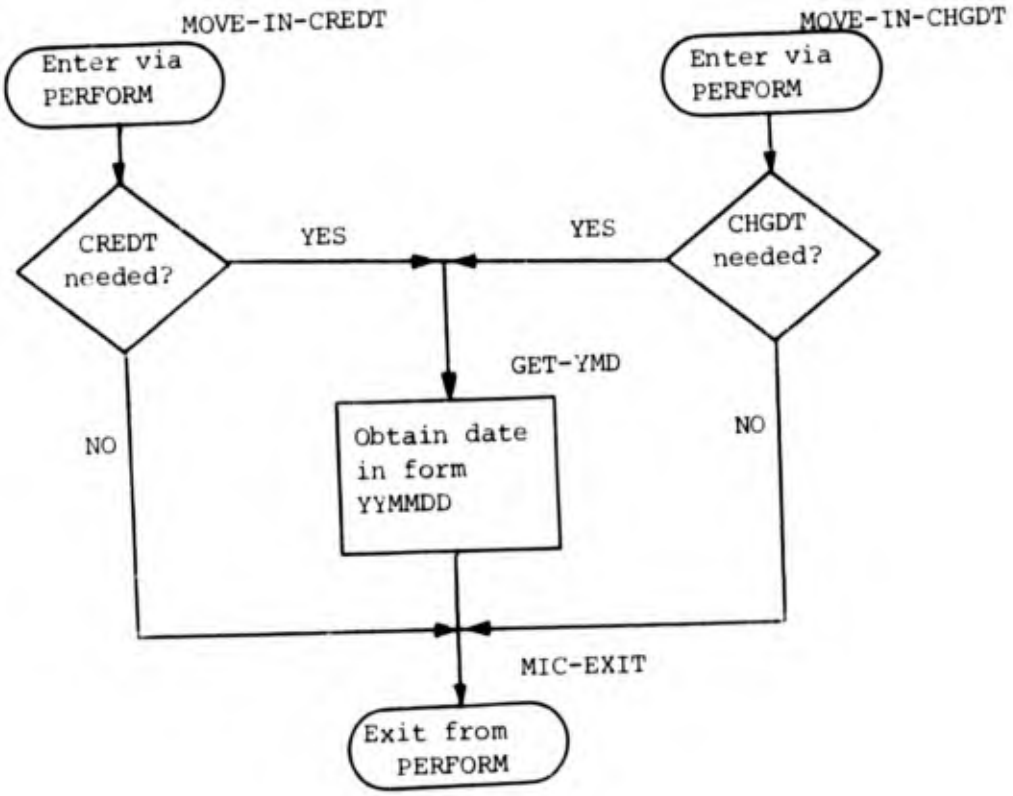

14A
SET-MPCD-SW
Set switch to show UNSORTED-FILE is open
14C
NOT-FIRST-TIME
Write record on KEY-FILE
Segments are 2000 char or less
Start loop to write segments of unsorted record
UNSORT-LOOP
FMMP-1
More than 2000 char left?
YES
Write segment to UNSORTED-FILE
NO
FMMP-2
Build final segment
14B
FMMP-2A
Write segment to UNSORTED-FILE
15C

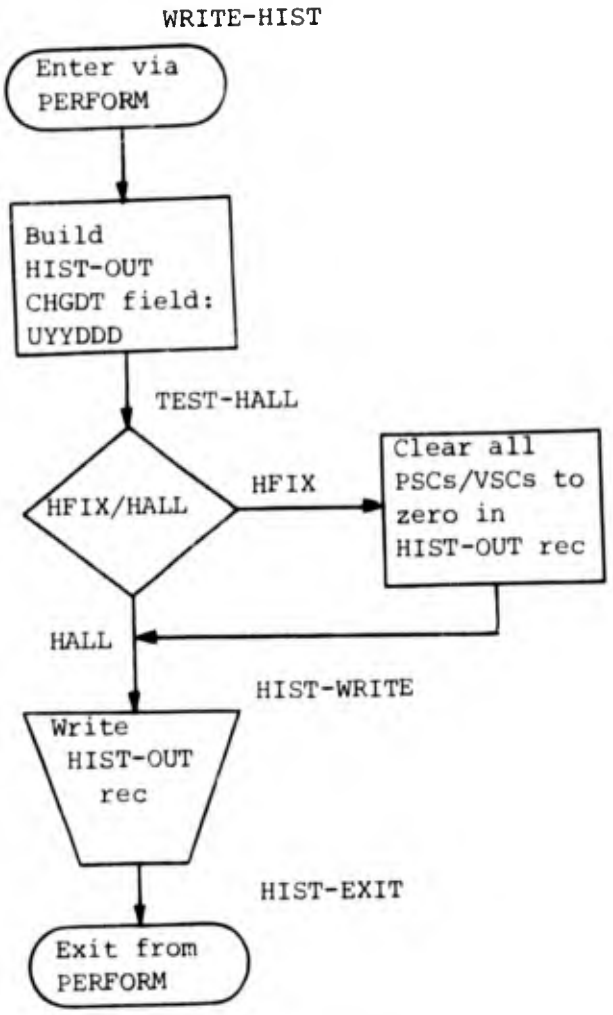

THIRTY-CHAR-OUT
Enter via PERFORM
Save Rec-ID of output rec in working storage
Exit from PERFORM
15A
MOVE-ERROR
Print error msg
15C
15B
CLSMAJ
DELETE-RECORDS
MAJOR-IN-FILE at EOF?
NO
Run out MAJOR-IN-FILE to EOF
YES
CLOSE-FILES
Close all files except SPECIAL-FILE, HIST-IN-FILE
15C
FINISH-TNOUT
15D
Exit from PERFORM WRTMAJ
Update count of deleted records
Reset switch indicating rec deleted
15C

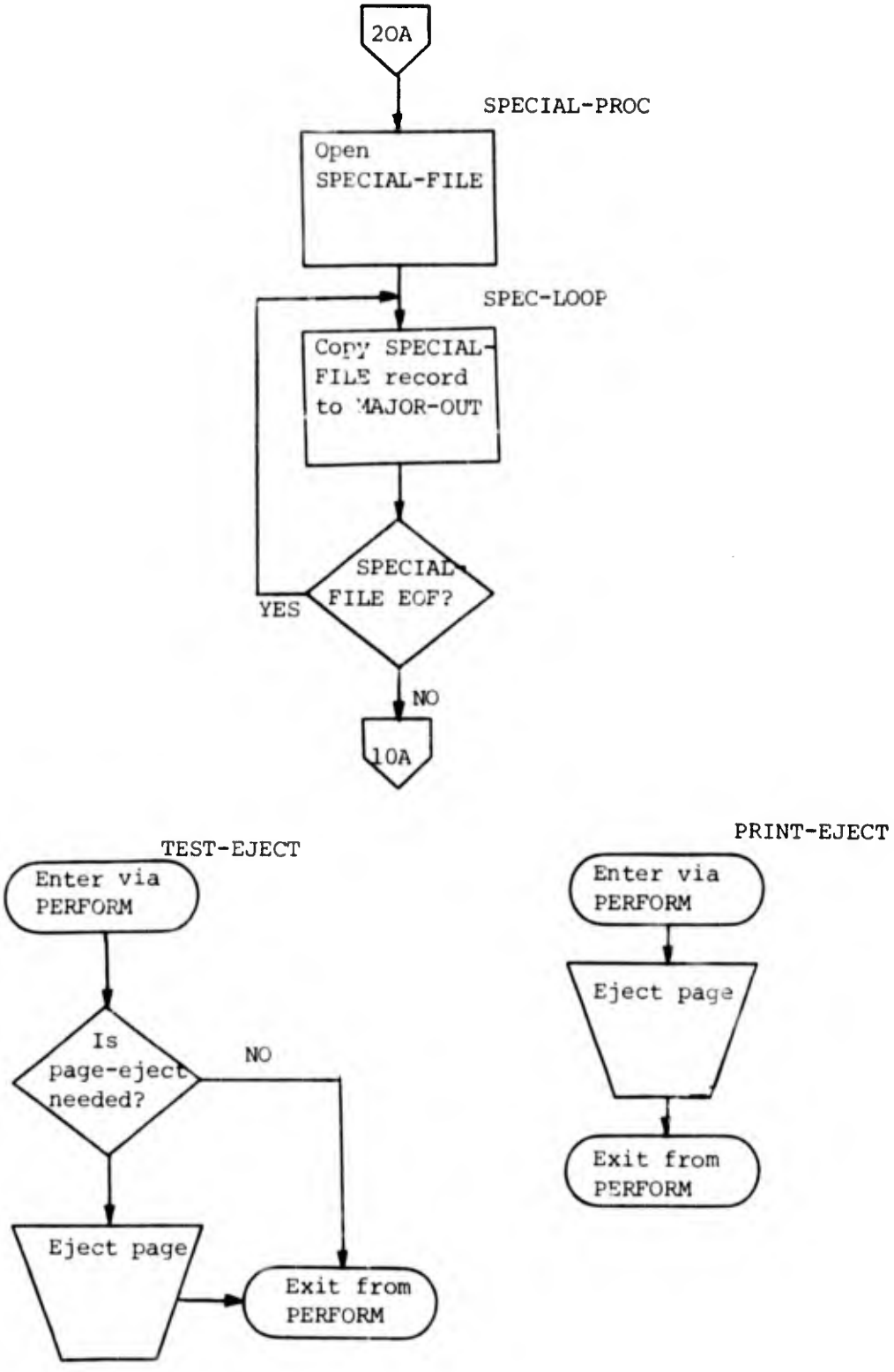

PRINT-STAT
Enter via PERFORM
Format, print statistical msg
Exit from PERFORM
PRINT-LINE
Enter via PERFORM
Print previously formatted msg
Exit from PERFORM
16A
NO-CHANGE
Compare current MAJOR-IN Rec-ID with last Rec-ID processed
Current: Last
>
17A
=
OLDID-ERROR
Set up error msg
15A
<
WRT-OLD
Copy MAJOR-IN record to MAJOR-OUT file
15C

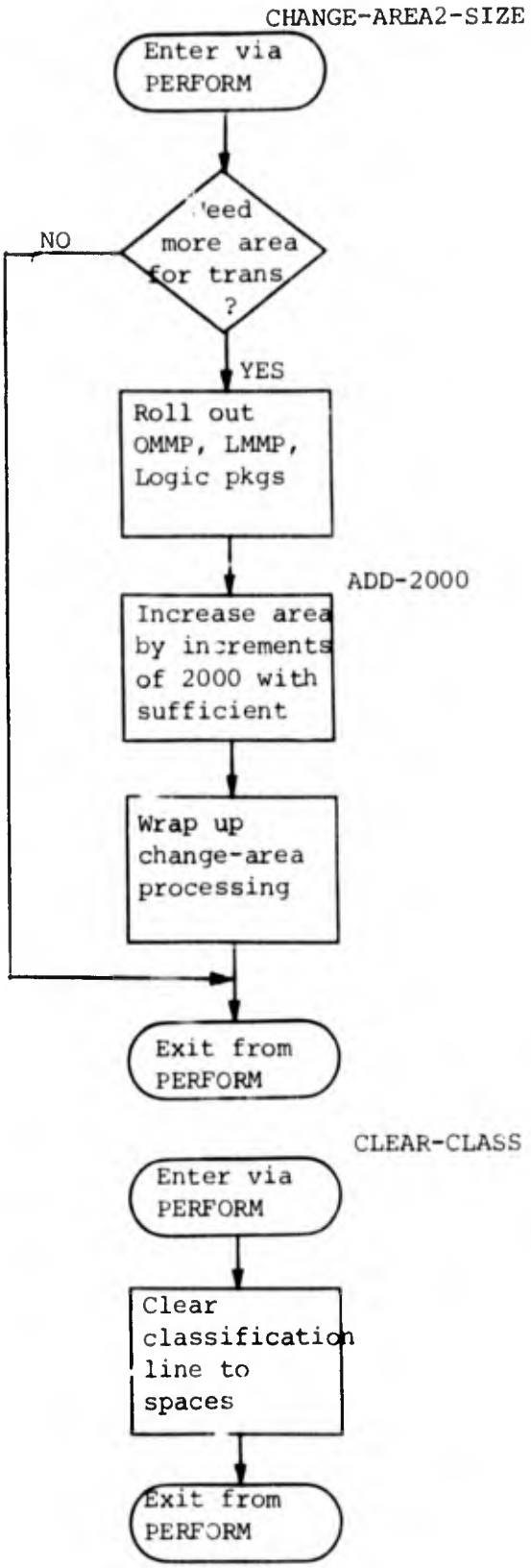

17A
OLD-UNSORT
Need to open UNSORTED-FILE?
YES
Open KEYFILE & UNSORTED-FILE
NO
Write rec on KEY-FILE
Segments are 2000 char or less
Start loop to write segments of unsorted rec
OLD-UNSORT-LOOP
More than 2000 char left?
NO
YES
Write segment to UNSORTED-FILE
FMMP-3
Build final segment
14B


GET-RECID
Enter via PERFORM
Move MAJOR-IN Rec-ID to working storage
Exit from PERFORM


MOVE-IN-CREDT
Enter via PERFORM
CREDT needed?
YES
NO
MOVE-IN-CHGDT
Enter via PERFORM
CHGDT needed?
YES
NO
GET-YMD
Obtain date in form YYMMDD
MIC-EXIT
Exit from PERFORM


WRITE-HIST
Enter via PERFORM
Build HIST-OUT CHGDT field: UYYDDD
TEST-HALL
HFIX/HALL
HFIX
Clear all PSCs/VSCs to zero in HIST-OUT rec
HALL
HIST-WRITE
Write HIST-OUT rec
HIST-EXIT
Exit from PERFORM


20A
SPECIAL-PROC
Open
SPECIAL-FILE
SPEC-LOOP
Copy SPECIAL-
FILE record
to MAJOR-OUT
SPECIAL-
FILE EOF?
YES
NO
10A
TEST-EJECT
Enter via
PERFORM
Is
page-eject
needed?
NO
Eject page
Exit from
PERFORM
PRINT-EJECT
Enter via
PERFORM
Eject page
Exit from
PERFORM


CHANGE-AREA2-SIZE
Enter via PERFORM
'eed more area for trans ?
NO
YES
Roll out OMMP, LMMP, Logic pkgs
ADD-2000
Increase area by increments of 2000 with sufficient
Wrap up change-area processing
Exit from PERFORM
CLEAR-CLASS
Enter via PERFORM
Clear classification line to spaces
Exit from PERFORM

b. MPCD1

(1) Function. To interpret FMMP control cards, set control switches, and determines the core needed for the logical records and LM Define Tables.

(2) Calling Sequence.

```
ENTRY 'MPCD1'  USING FM-DD  MP-DD  MAJOR-LR2
                     MAJOR-LR3  DEFINE-AREA.
CALL  'LB'     USING CALLING-SEQ DATA-AREA.*
CALL  'FMCRD'  USING FM-INPUT END-SW.
CALL  'FMSCAN' USING FM-DD.
CALL  'FMPRT'  USING PRT-LINE  CC.
CALL  'MOVALF' USING ...
```

*LB is called several times by MPCD1 referencing different data areas.

(3) Program Description. Since the basic purpose of MPCD1 is to analyze FMMP control cards, most paragraphs of this subprogram validate the contents of the control cards. Therefore, the activities of each paragraph are summarized.

CHECK-FILE.
Validates the form of a file or FFT name on the FMMP card.

CHECK-FILE-EXIT.
Is a GO TO which is altered to distinguish between the first control card and a continuation card.

READ-FFT.
Reads Logical Record 1 of the FFT. If it is not found on the library, an error message is printed and the MP run is terminated.

TEST-FOR-LAST-WORD.
Determines if all words have been processed.

TEST-FOR-LAST-WORD.

CHECK-PARAMETERS.

GET-NEXT-WORD.
Look for the presence of most of the MP option specifications. Certain of the options (NOFND, NOXCH) are tested for compatibility with Mark III FFS, but are not presently used.

CHECK-CONTN.
Determines if there is a continuation specification on the FMMP card. If so, another control card is read in the GET-NEXT-CARD paragraph. If not, an error message is printed indicating that an invalid parameter is on the control card and the MP run is terminated.

GET-NEXT-CARD.
Reads a card through FMCRD and uses FMSCAN to divide it into words.

MORE-CONTN.
Determines that the card read after a continuation is an FMMP card. If not, a message is printed.

CHECK-OM.

Distinguishes between OM and LM runs.

MORE-LM.

Determines if more than one Logic Package name is specified. Only one is permitted for file-to-file LM.

CHECK-LMFR.

Looks for a File Revision job. OM is not permitted with File Revision.

MORE-LMFR.

Determines if more Logic Packages are specified for File Revision. This is illegal.

CHECK-PKG.

Validates the general format of a Logic Package name and determines if too many names have been specified.

VALIDATE-PKG.

Determines if control information for the Logic Package is on the Table Library. This will only be present when the Logic Package was built in Subroutine mode by Language Processor.

MOVE-VERSION.

Insures that only one file-to-file Logic Package was specified and finds the size of the define values for the Logic Package in question. The LOP for these defines in DEFINE-AREA is saved in the DEP-HOP field of the LM-ITEM for the Logic Package.

VAL-PKG-EXIT.

Is an altered GO TO which will distinguish between the processing of ordinary LM names and File Revision names.

LAST-WORD.

Contains final validation of parameter combinations and omissions.

PRINT-ERROR.

Moves the error message area to the print area and sets the FM-ERROR switch so that the FM run will be terminated due to the error.

PRINT-LINE.

Prints a line by means of the FMPRT subroutine.

TEST-END-SW.

Completes the processing of MPCD1 by returning to FMMPX.

CHECK-LITERAL thru CL-EXIT.

Place the classification literal, if specified on the FMMP card, into an area to be passed to FMIO for setting up the writing of headers and trailers for the rest of the run.

GET-LR-HOPS.

Computes the size of LR2 and LR3 from the information provided in LR1 (i.e., the number of items in LR2 times 21 is the size of LR2, the number of items in LR3 times 10 is the size of LR3). LR2 and LR3 sizes are added together with the total define LM sizes. TOTAL-DEFINE has these combined sizes. This size will be used by FMMP to open AREA1.

Because of the extensive use of LM-PKG-LIST and REDEF-LIST within the MPCD1 and MPCD2 subprograms, their entries will be explained here.

```
Ø2  LM-COUNT            PICTURE 99       COMPUTATIONAL.
Ø2  LM-PKG-LIST.
  Ø3  LM-ITEM OCCURS 1Ø TIMES.
    Ø4  DEF-HOP         PICTURE 9(5)     COMPUTATIONAL.
    Ø4  REDEF-FIRST     PICTURE 999      COMPUTATIONAL.
    Ø4  REDEF-LAST      PICTURE 999      COMPUTATIONAL.
    Ø4  LM-PKG          PICTURE X(5).
Ø2  REDEF-COUNT         PICTURE 999      COMPUTATIONAL.
Ø2  REDEF-LIST.
  Ø3  REDEF-ITEM OCCURS 1ØØ TIMES.
    Ø4  REDEF-POINT     PICTURE 9999     COMPUTATIONAL.
    Ø4  REDEF-SIZE      PICTURE 99       COMPUTATIONAL.
    Ø4  REDEF-NEXT      PICTURE 999      COMPUTATIONAL.
    Ø4  REDEF-NAME      PICTURE X(6).
```

LM-COUNT.
Contains the number of Logic Packages specified on the FMMP control card.

DEF-HOP.
Contains the first position subscript of DEFINE-CHAR and indicates the beginning of the define values for this Logic Package.

REDEF-FIRST.
Contains the subscript of the first entry in REDEF-LIST applying to this Logic Package. REDEF-FIRST is zero when no redefines have been specified.

REDEF-LAST.
Contains the subscript of the last entry in REDEF-LIST applying to this Logic Package.

LM-PKG.
Contains the name of the Logic Package.

REDEF-COUNT.
Contains the number of REDEF cards which have been processed.

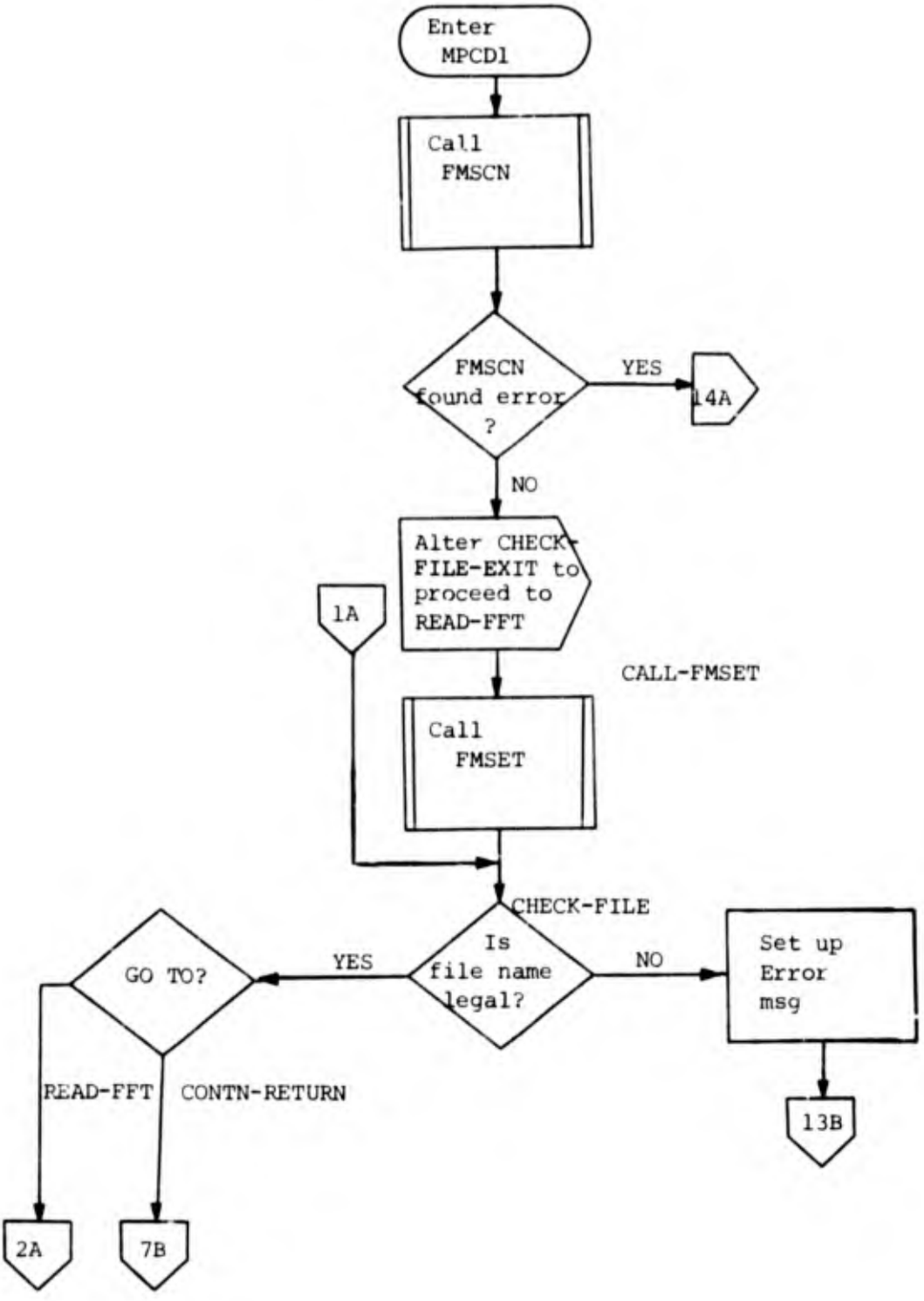
REDEF-POINT.
Contains the first position subscript of REDEF-CHAR and indicates the beginning of a particular redefine value.

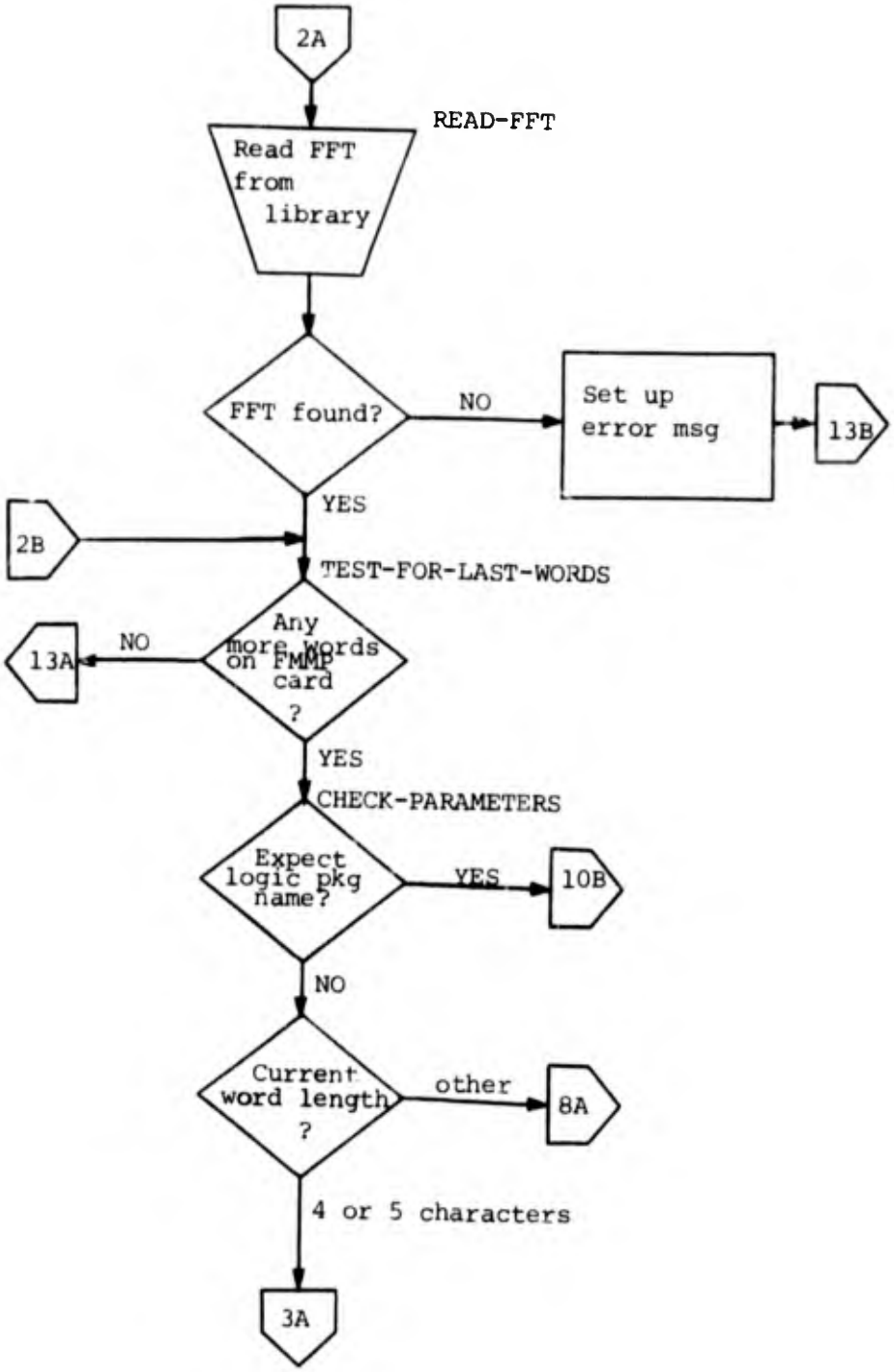
REDEF-SIZE.
Contains the number of characters of redefine data.

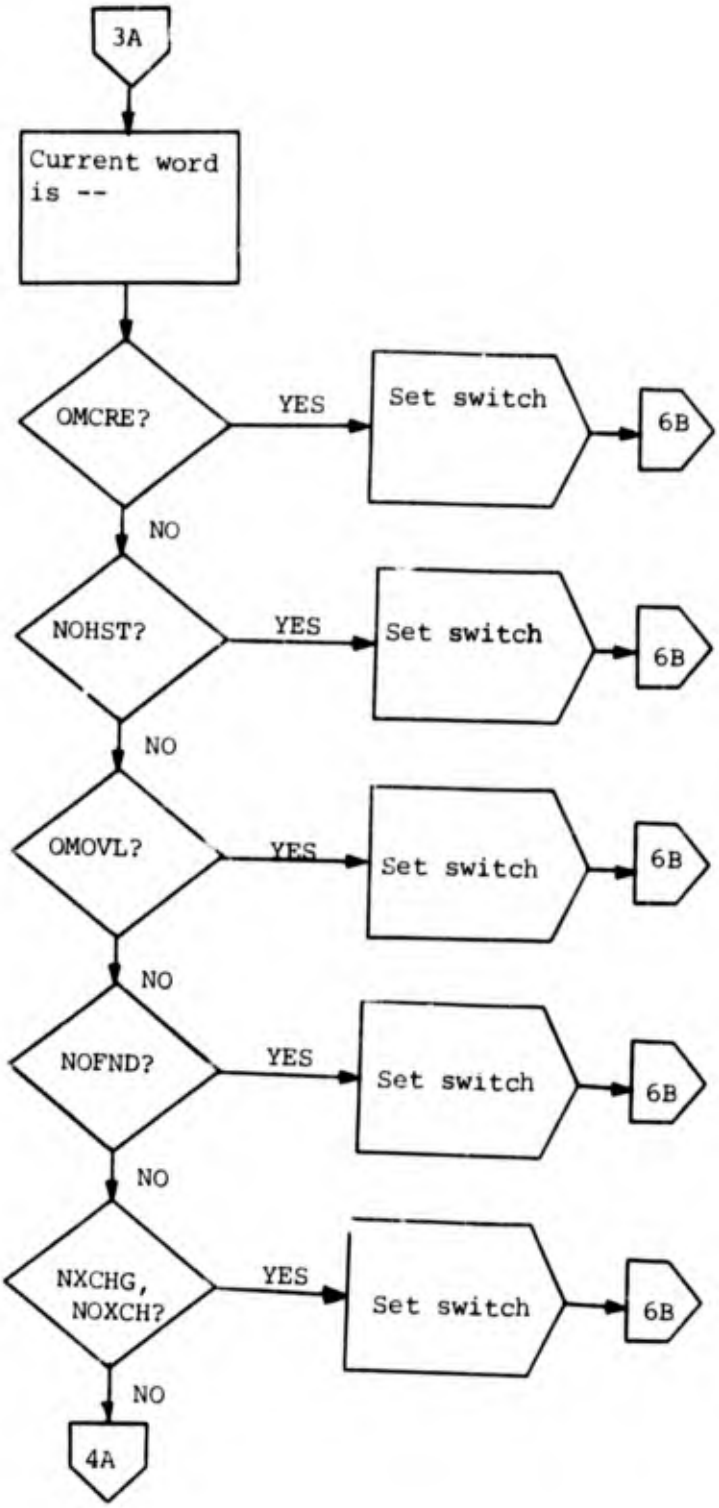
REDEF-NEXT.
Contains the subscript of the next REDEF-ITEM applying to the same Logic Package. It is zero when there is no next redefine.

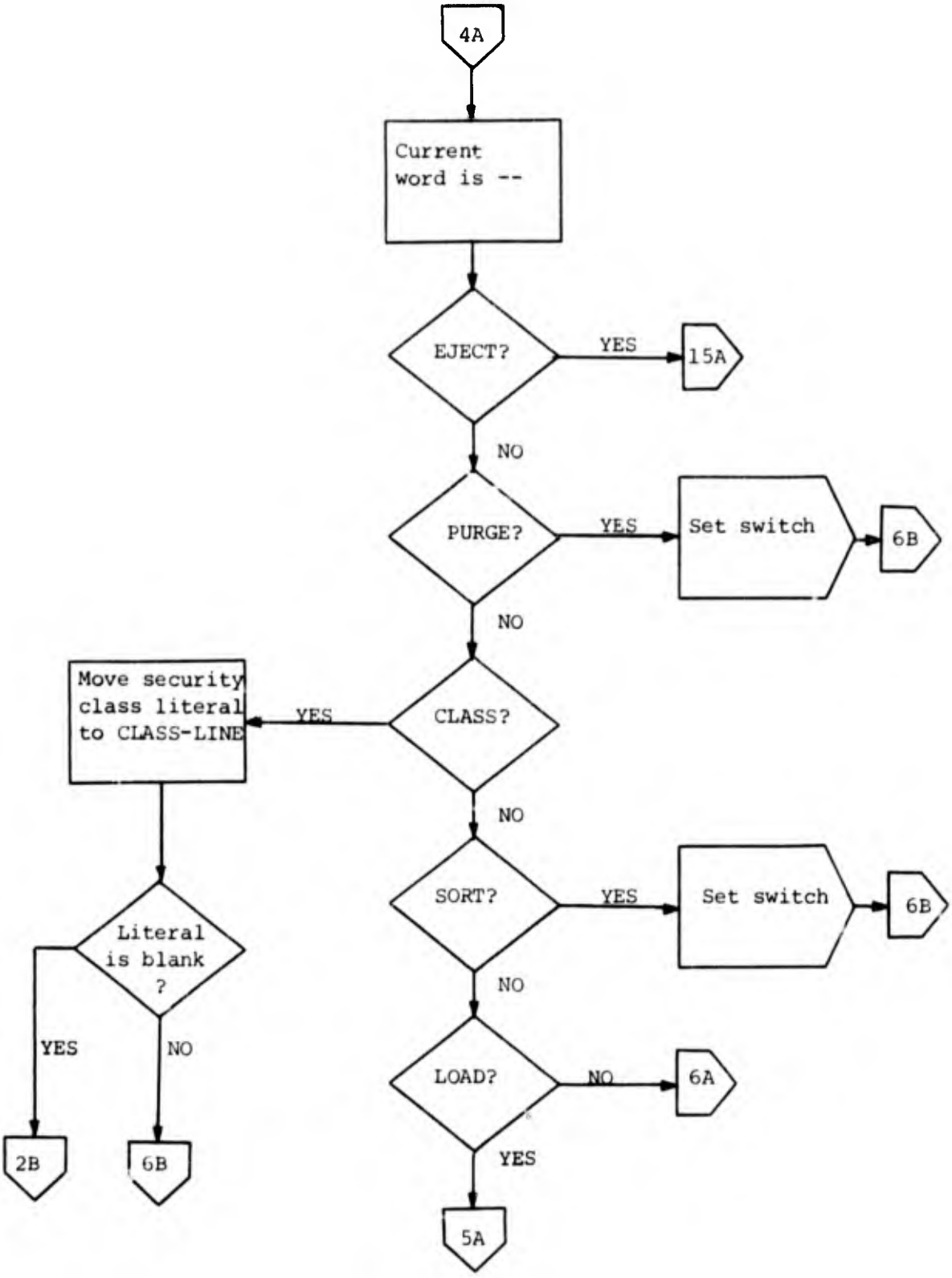
REDEF-NAME.
Is the define name specified on the REDEF card.

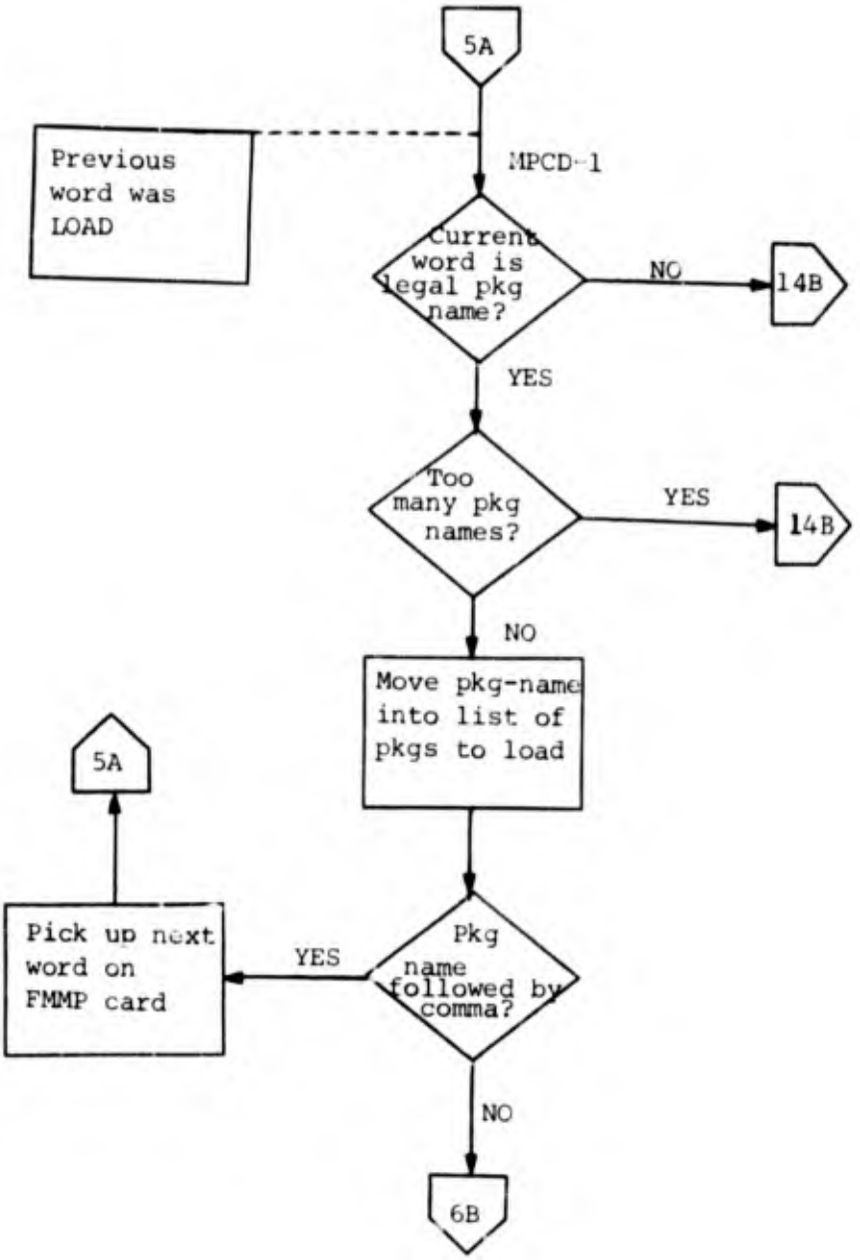
(4) Limitations. None.

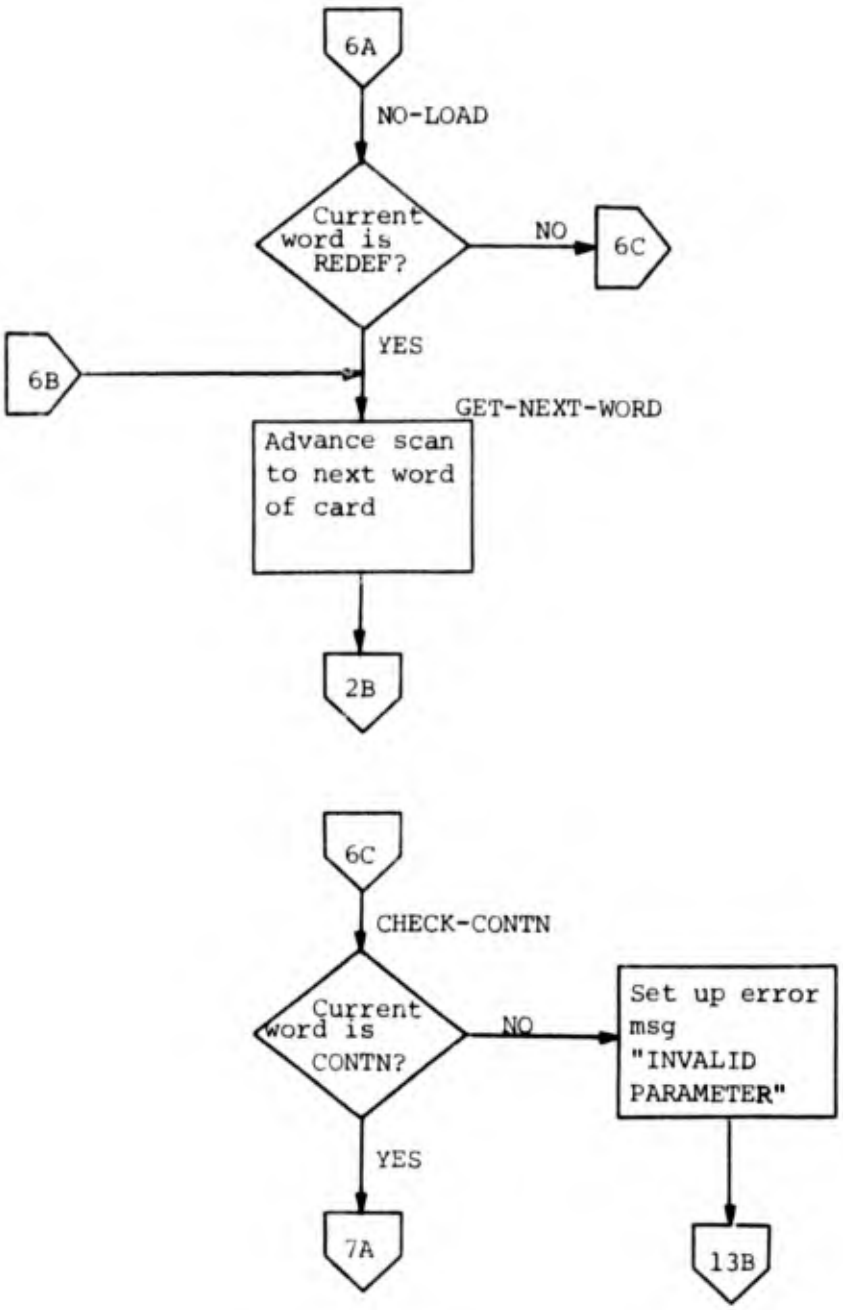

Enter
MPCD1
Call
FMSCN
FMSCN
found error
?
YES
14A
NO
Alter CHECK-
FILE-EXIT to
proceed to
READ-FFT
1A
CALL-FMSET
Call
FMSET
CHECK-FILE
Is
file name
legal?
YES
NO
GO TO?
Set up
Error
msg
READ-FFT
CONTN-RETURN
13B
2A
7B

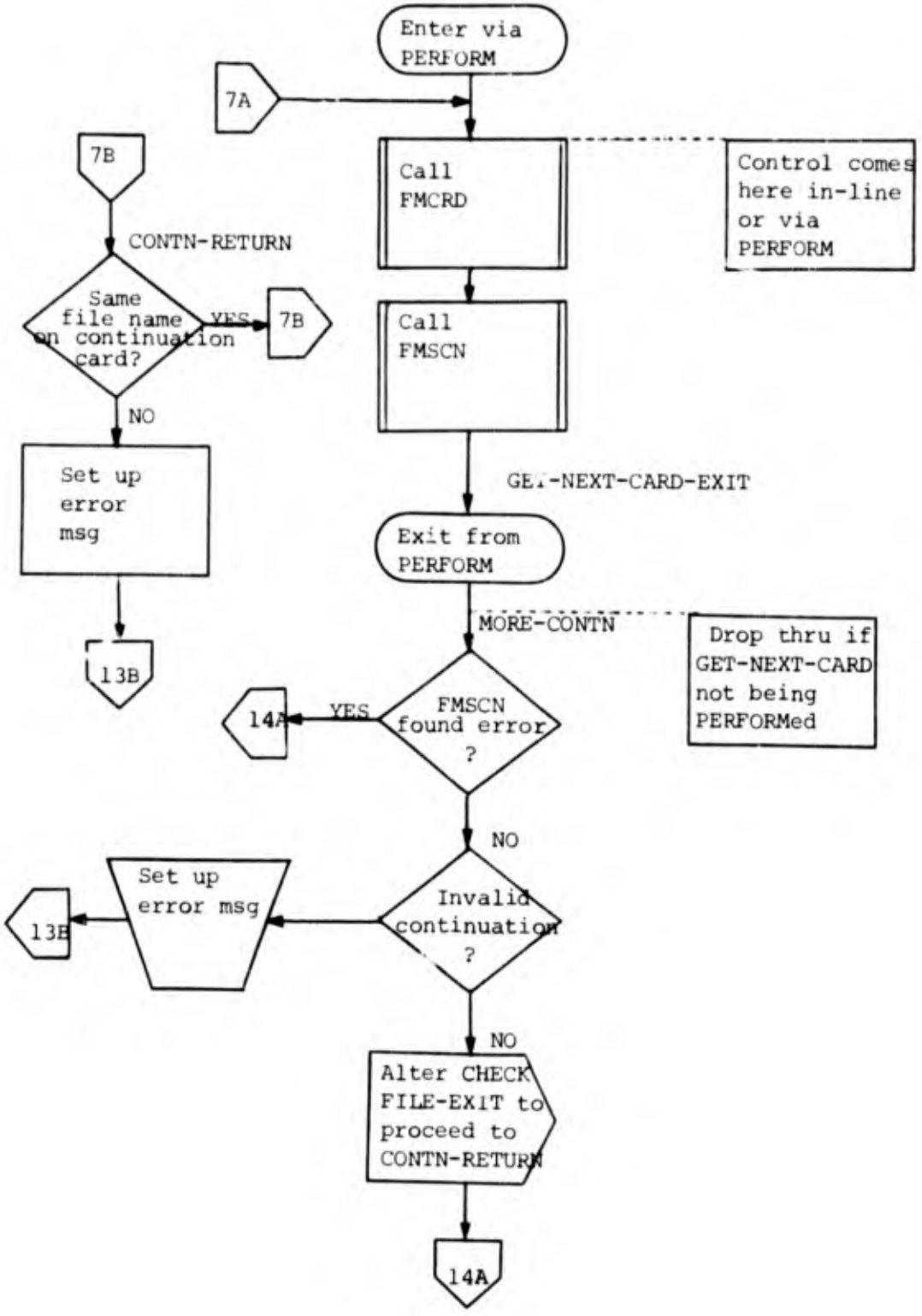

2A
READ-FFT
Read FFT from library
FFT found?
NO
Set up error msg
13B
YES
2B
TEST-FOR-LAST-WORDS
Any more words on FMMP card ?
NO
13A
YES
CHECK-PARAMETERS
Expect logic pkg name?
YES
10B
NO
Current word length ?
other
8A
4 or 5 characters
3A

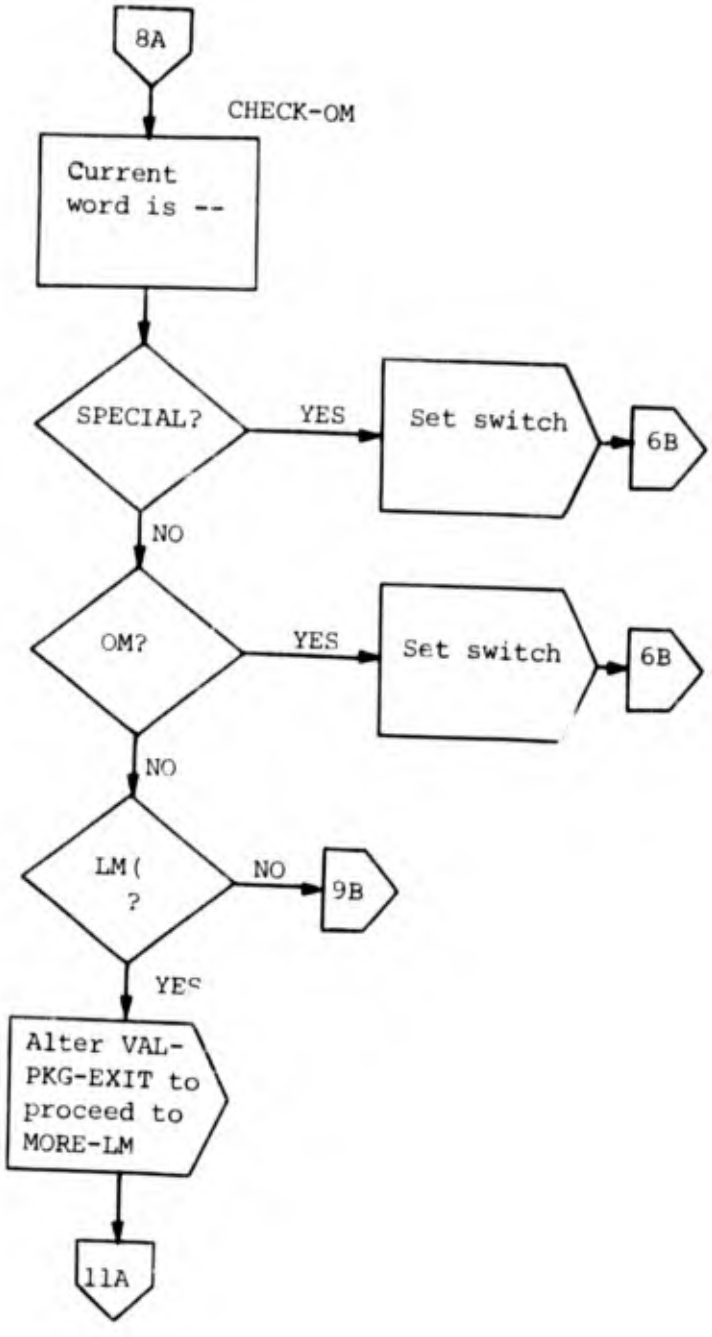

3A
Current word is --
OMCRE?
YES
Set switch
6B
NO
NOHST?
YES
Set switch
6B
NO
OMOVL?
YES
Set switch
6B
NO
NOFND?
YES
Set switch
6B
NO
NXCHG, NOXCH?
YES
Set switch
6B
NO
4A

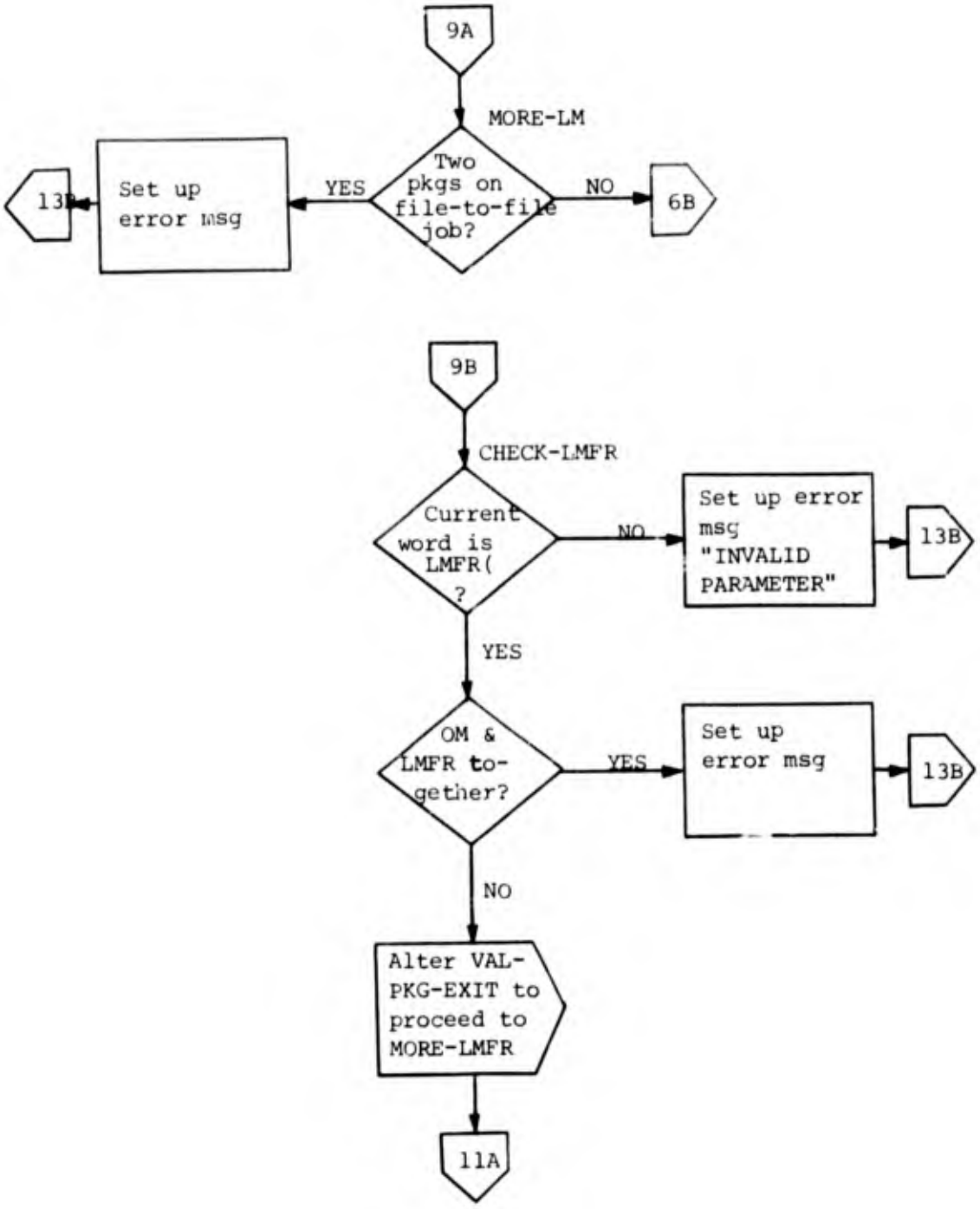

4A
Current word is --
EJECT?
YES
15A
NO
PURGE?
YES
Set switch
6B
NO
CLASS?
YES
Move security class literal to CLASS-LINE
NO
SORT?
YES
Set switch
6B
NO
LOAD?
NO
6A
YES
5A
Literal is blank ?
YES
NO
2B
6B

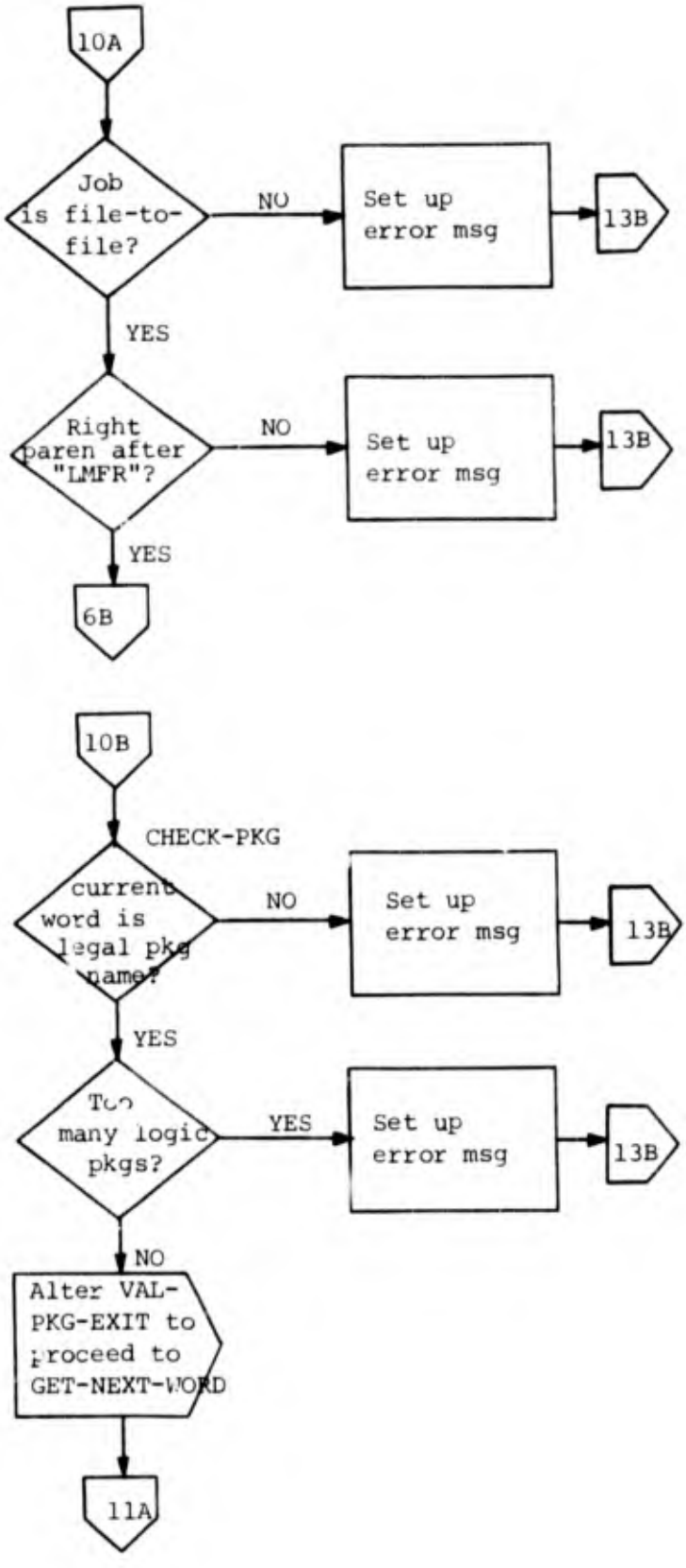

5A
Previous word was LOAD
MPCD-1
Current word is legal pkg name?
NO
14B
YES
Too many pkg names?
YES
14B
NO
Move pkg-name into list of pkgs to load
5A
Pick up next word on FMMP card
YES
Pkg name followed by comma?
NO
6B

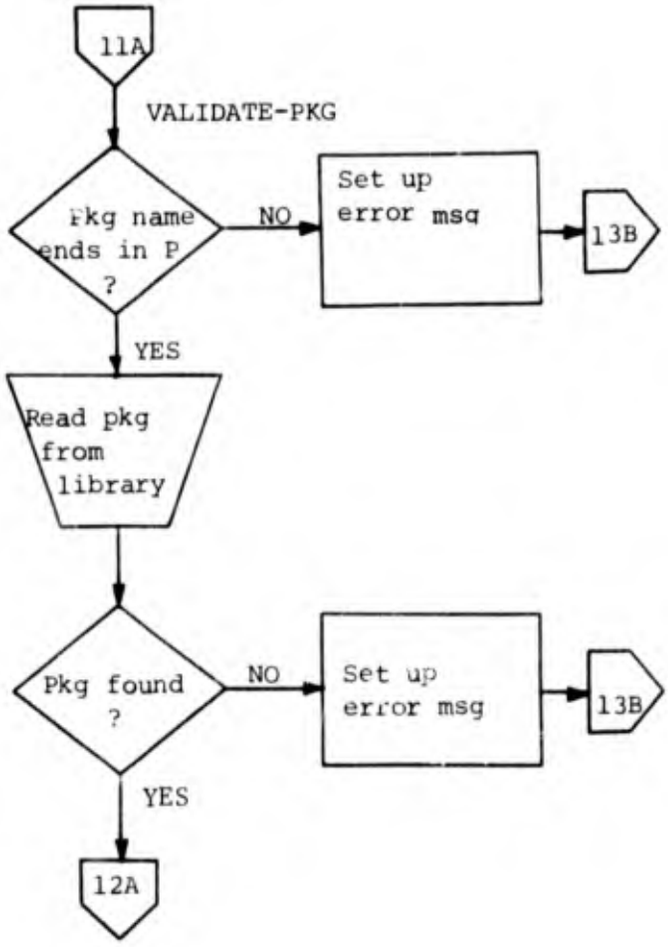

6A
NO-LOAD
Current word is REDEF?
NO
6C
YES
6B
GET-NEXT-WORD
Advance scan to next word of card
2B
6C
CHECK-CONTN
Current word is CONTN?
NO
Set up error msg "INVALID PARAMETER"
YES
7A
13B

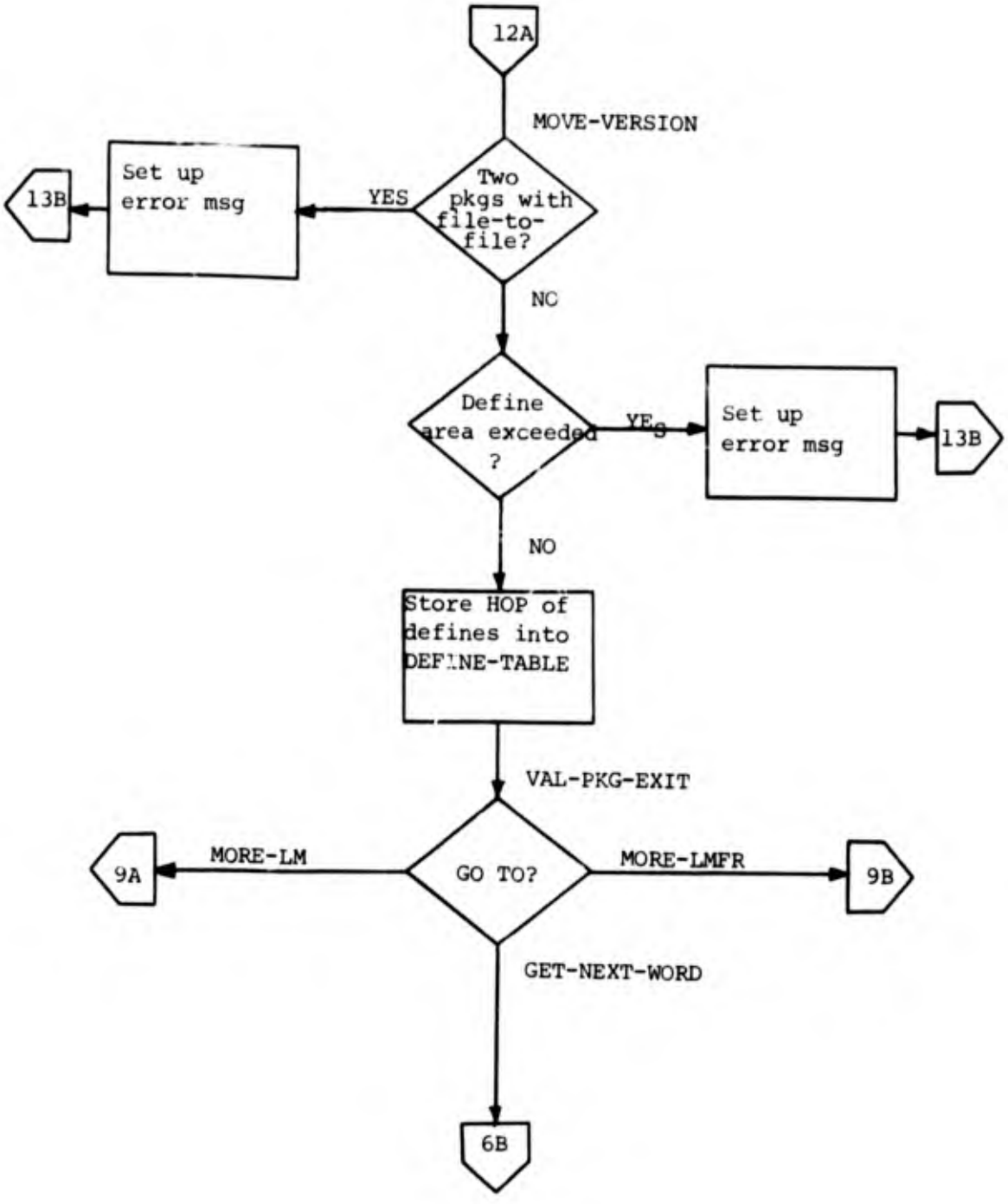

Enter via PERFORM
7A
7B
Call FMCRD
Control comes here in-line or via PERFORM
CONTN-RETURN
Same file name on continuation card?
YES
7B
Call FMSCN
NO
Set up error msg
GET-NEXT-CARD-EXIT
Exit from PERFORM
13B
MORE-CONTN
Drop thru if GET-NEXT-CARD not being PERFORMed
14A
YES
FMSCN found error ?
NO
Set up error msg
13B
Invalid continuation ?
NO
Alter CHECK FILE-EXIT to proceed to CONTN-RETURN
14A

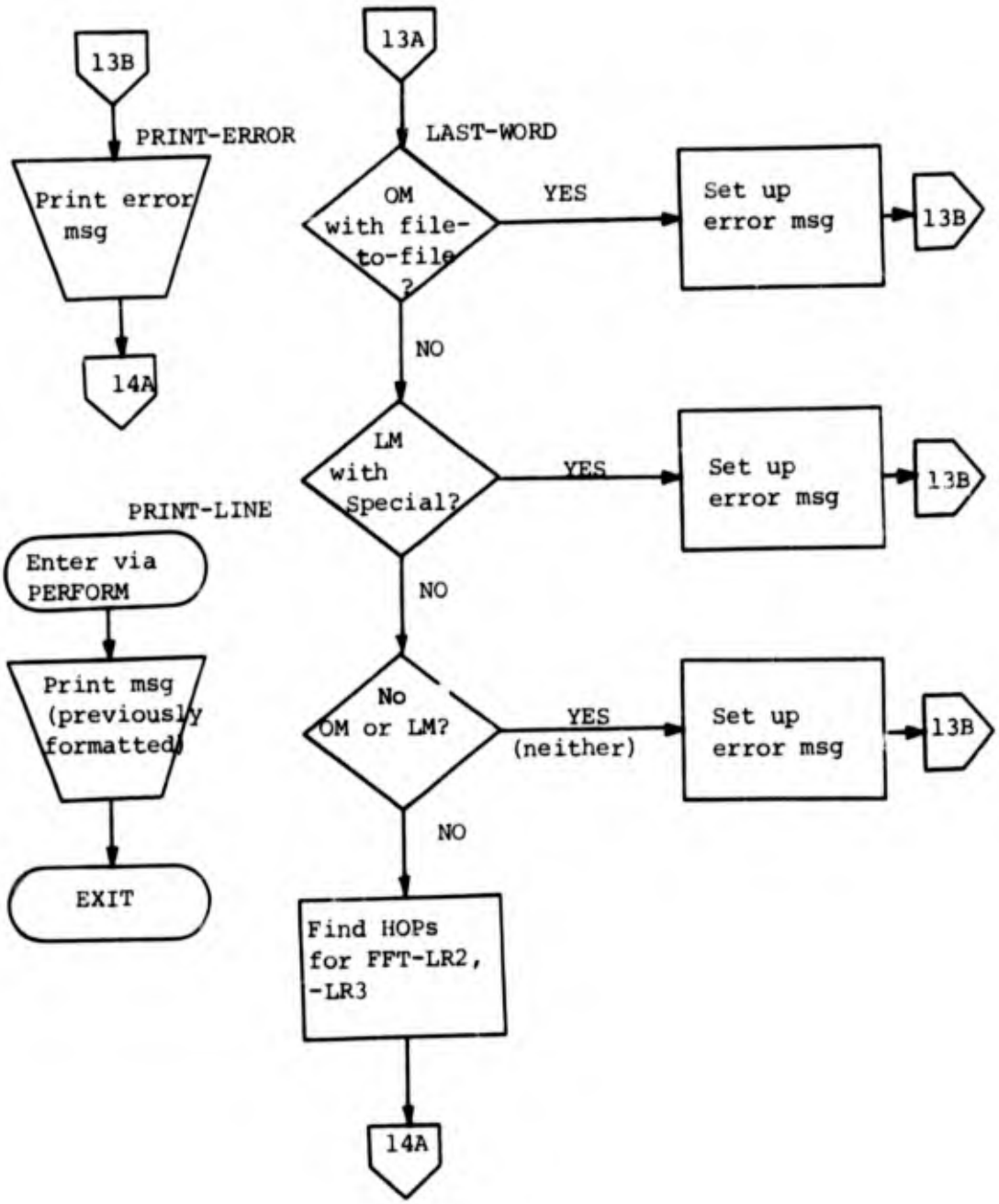

8A
CHECK-OM
Current word is --
SPECIAL?
YES
Set switch
6B
NO
OM?
YES
Set switch
6B
NO
LM(?
NO
9B
YES
Alter VAL-PKG-EXIT to proceed to MORE-LM
11A

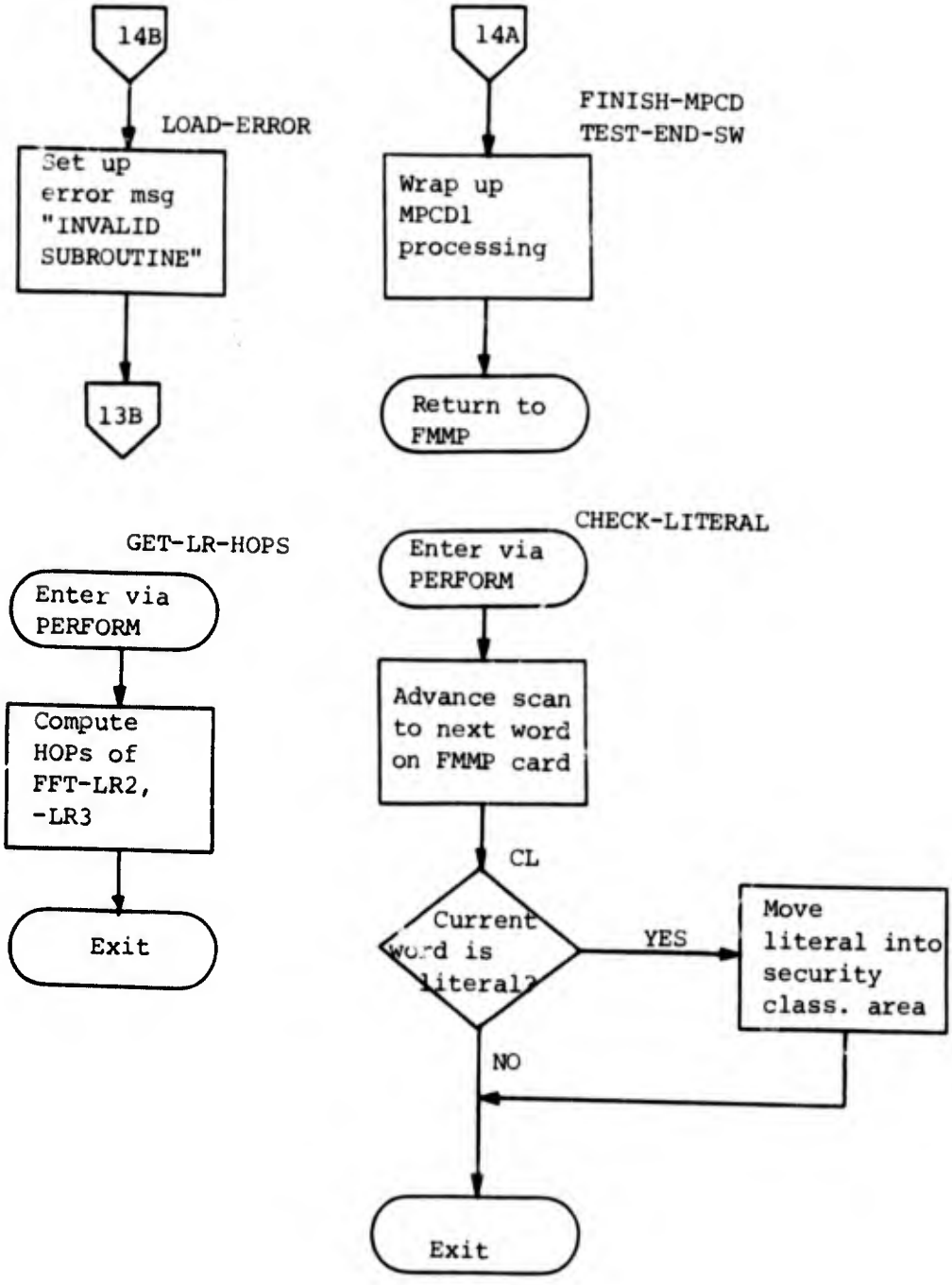

9A
MORE-LM
Two pkgs on file-to-file job?
YES
NO
Set up error msg
13B
6B
9B
CHECK-LMFR
Current word is LMFR(?
NO
Set up error msg "INVALID PARAMETER"
13B
YES
OM & LMFR to-gether?
YES
Set up error msg
13B
NO
Alter VAL-PKG-EXIT to proceed to MORE-LMFR
11A

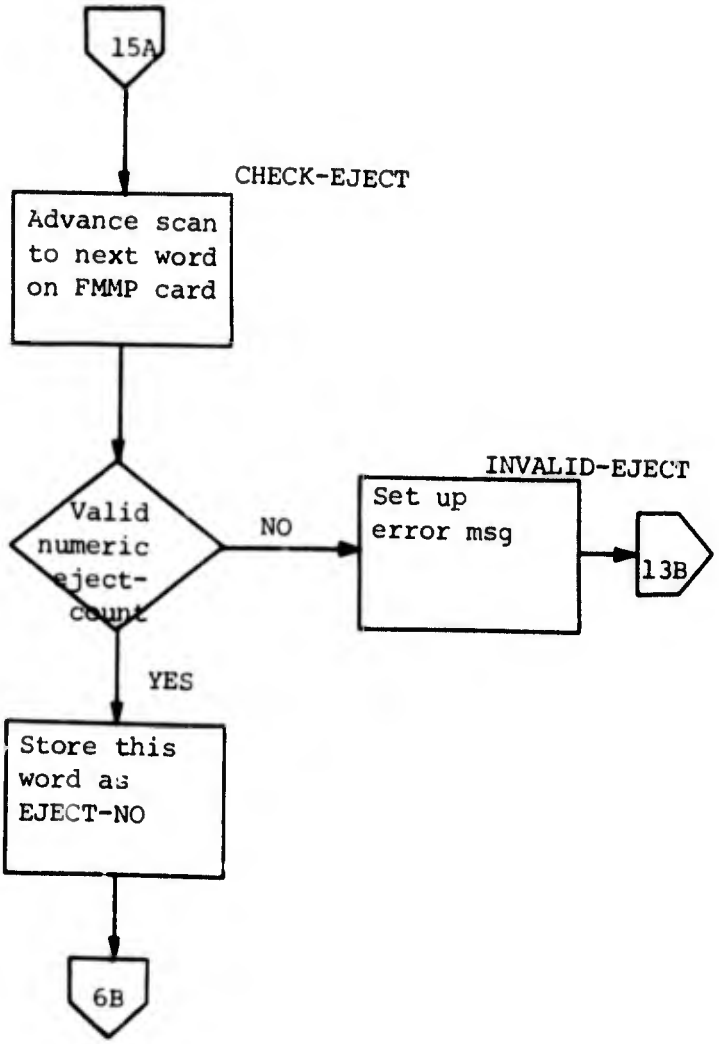

10A
Job is file-to-file?
NO
Set up error msg
13B
YES
Right paren after "LMFR"?
NO
Set up error msg
13B
YES
6B
10B
CHECK-PKG
current word is legal pkg name?
NO
Set up error msg
13B
YES
Too many logic pkgs?
YES
Set up error msg
13B
NO
Alter VAL-PKG-EXIT to proceed to GET-NEXT-WORD
11A


11A
VALIDATE-PKG
Pkg name ends in P ?
NO
Set up error msg
13B
YES
Read pkg from library
Pkg found ?
NO
Set up error msg
13B
YES
12A


12A
MOVE-VERSION
Two pkgs with file-to-file?
YES
Set up error msg
13B
NO
Define area exceeded ?
YES
Set up error msg
13B
NO
Store HOP of defines into DEFINE-TABLE
VAL-PKG-EXIT
GO TO?
MORE-LM
9A
MORE-LMFR
9B
GET-NEXT-WORD
6B


13B
PRINT-ERROR
Print error msg
14A
13A
LAST-WORD
OM with file-to-file ?
YES
Set up error msg
13B
NO
LM with Special?
YES
Set up error msg
13B
NO
No OM or LM?
YES
(neither)
Set up error msg
13B
NO
Find HOPs for FFT-LR2, -LR3
14A
PRINT-LINE
Enter via PERFORM
Print msg (previously formatted)
EXIT


14B
LOAD-ERROR
Set up error msg "INVALID SUBROUTINE"
13B
14A
FINISH-MPCD
TEST-END-SW
Wrap up MPCD1 processing
Return to FMMP
GET-LR-HOPS
Enter via PERFORM
Compute HOPs of FFT-LR2, -LR3
Exit
CHECK-LITERAL
Enter via PERFORM
Advance scan to next word on FMMP card
CL
Current word is literal?
YES
Move literal into security class. area
NO
Exit


15A
CHECK-EJECT
Advance scan to next word on FMMP card
Valid numeric eject-count
NO
INVALID-EJECT
Set up error msg
13B
YES
Store this word as EJECT-NO
6B

c. MPCD2

(1) Function. To interpret FMMP REDEF control cards, set control switches, and read File Format and LM Define Tables into AREA1 whose size was computed by MPCD1. This allows utilization of only that amount of core required and precludes the use of fixed length tables in the data division.

(2) Calling Sequence.

```
ENTRY   'MPCD2'      USING FM-DD MP-DD MAJOR-LR2 MAJOR-LR3
                           DEFINE-AREA.
CALL    'LB'         USING CALLING-SEQ Data-Area.*
CALL    'FMCRD'      USING FM-INPUT END-SW.
CALL    'FMSCAN'     USING FM-DD.
CALL    'FMPRT'      USING PRT-LINE  CC.
CALL    'MOVARAYS'   USING ...
CALL    'MOVNUM'     USING ...
CALL    'MOVCMP'     USING ...
CALL    'MOVALF'     USING ...
CALL    'DATESUB'    USING JUL-DATE-X
CALL    'YMDSUB'     USING JUL-DATE-X YMD-DATE-X.
```

*LB is called several times by MPCD2 referencing different data areas.

(3) Program Description. Since the basic purpose of MPCD2 is to analyze FMMP REDEF control cards, most paragraphs of this subprogram validate the contents of the control cards. Therefore, the activities of each paragraph are summarized.

GET-NEXT-CARD.
Reads a card through FMCRD and uses FMSCAN to divide it into words.

MORE-REDEF.
Validates REDEF cards by checking for the word "REDEF" and the presence of a Logic Package name with the proper format.

TEST-NAME.
Insures that the Logic Package name was specified on the FMMP control card.

TEST-LABEL.
Checks the format of the define name and the presence of a literal. It also sets up a REDEF-ITEM with control information about the redefine and moves (using MOVARAYS) the REDEF data to REDEF-AREA.

CHECK-ENDREDEF.
Looks for an ENDREDEF card terminating REDEF cards. Any other card is illegal.

READ-LRS.
Reads FFT Logical Records 2 and 3 into the first part of DEFINE-AREA. Their relative high order position are contained in LR2-HOP and LR3-HOP.

EXECUTE-REDEF.

Starts the processing of LM-PKG-LIST and REDEF-LIST information.

NEXT-PKG.

Determines if the Logic Package has redefines.

ADJUST-PKG.

Determines if all Logic Packages have been tested for redefines. If not, the next one is checked.

READ-DEF-TABLE.

Reads the define table for the Logic Package from the Table Library.

FIRST-DEFINE.

Initializes the search for the define name.

TEST-FOR-END.

Determines the present of a name in the define table. If the name is found, the size of the redefine data is compared to the define field size to determine that it fits. Then the type of define data is checked and one of three subroutines is used to move the redefine data into the define data area. MOVNUM is used for ordinary numeric defines to right justify the number and insert leading zeros. MOVCMP (called in the COMP-DEFINE paragraph converts numeric characters to computational form. MOVALF (called in the ALPHA-DEFINE paragraph) left justifies alphanumeric defines and pads with trailing blanks. If the MOVNUM or MOVCMP subroutines determine that the redefine data is not actually numeric, an error message is printed and the MP run is aborted. Ordinarily, the NEXT-REDEFINE paragraph is executed to locate the next redefine.

PRINT-ERROR.

Moves the error message area to the print area and sets the FM-ERROR switch so that the FM run will be terminated due to the error.

PRINT-LINE.

Prints a line by means of FMPRT subroutine.

FINISH-MPCD.

Sets up the classification header and the YYMMDD date. If CREDT and CHGDT are specified for the FFT, the fixed-set HOP's for these fields are also determined.

TEST-END-SW.

Wraps up processing for this module and returns control to FMMP.

CHECK-FOR-CREDT-CHGDT.

CFCC thru CFCC-EXIT.

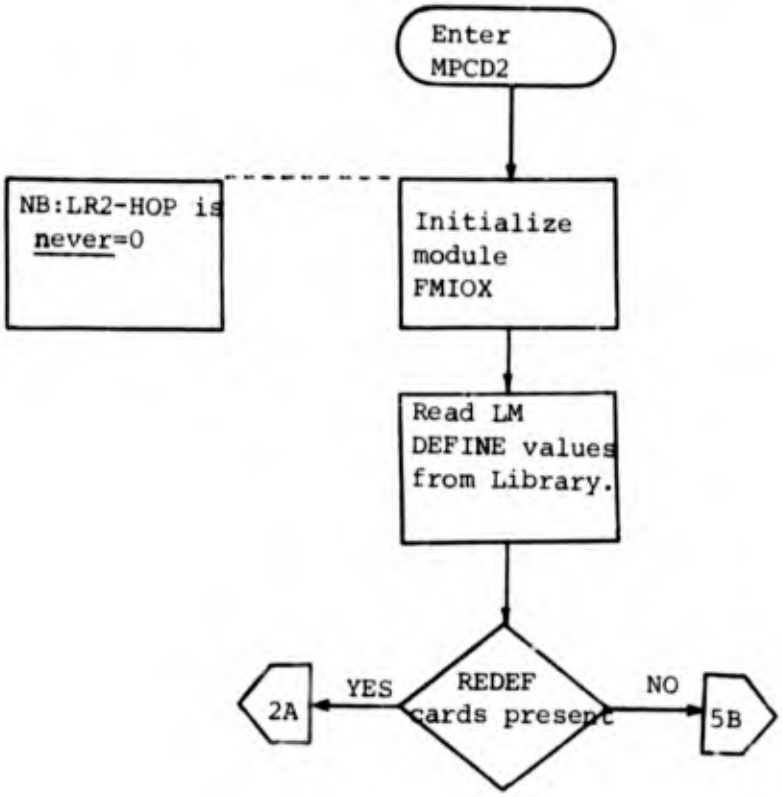
These paragraphs obtain the Julian date from the Operating System, convert it to YYMMDD format, and determine whether CREDT and CHGDT are present in the FFT. If they are, their fired-set HOP's are determined.

MOVE-IN-LM-VALUES.

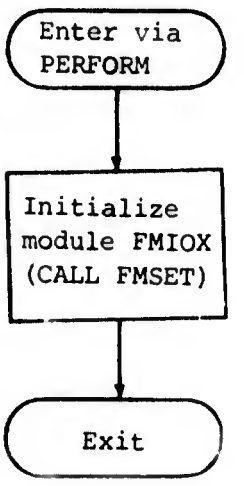
MOVE-THEM thru MT-EXIT.

These paragraphs fetch REDEF values from the library for all Logic Packages.

(4) Limitations. None.

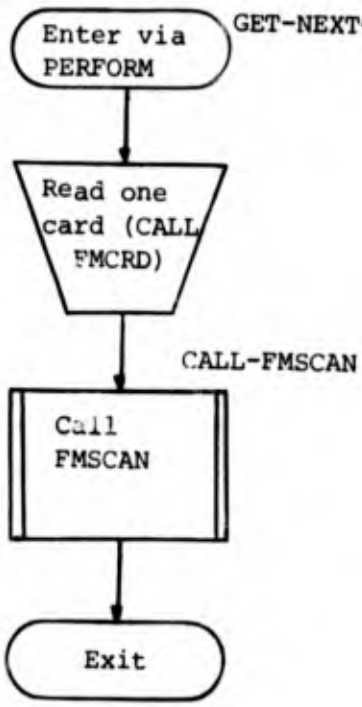


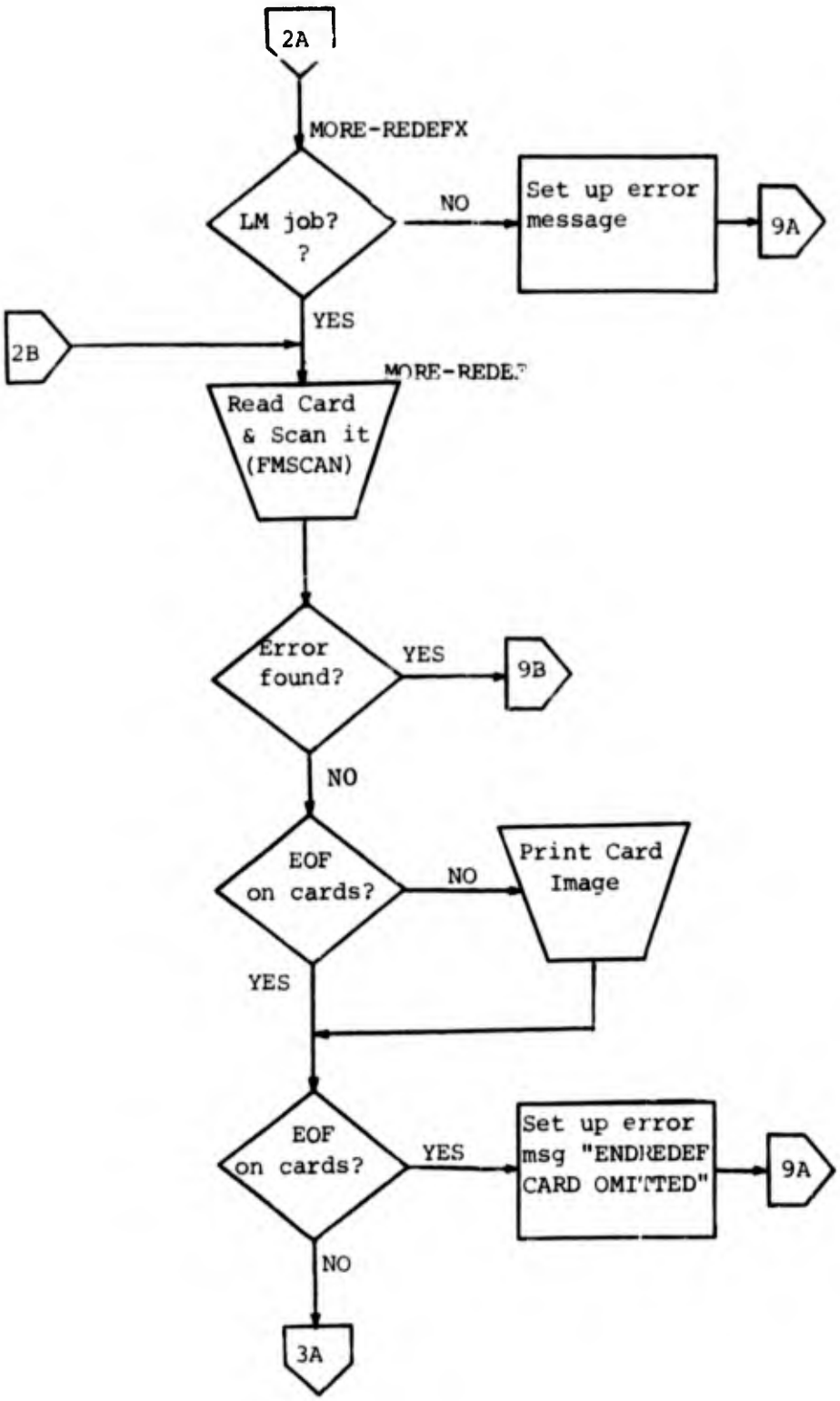
CALL-FMSET

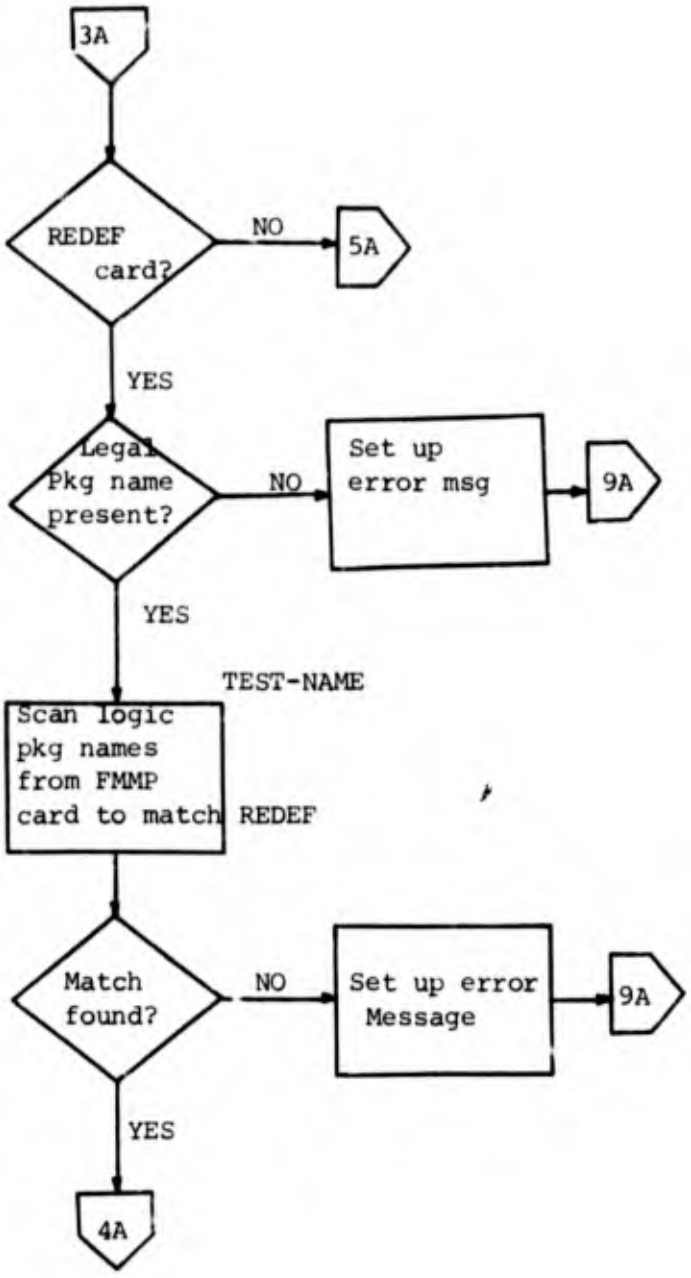


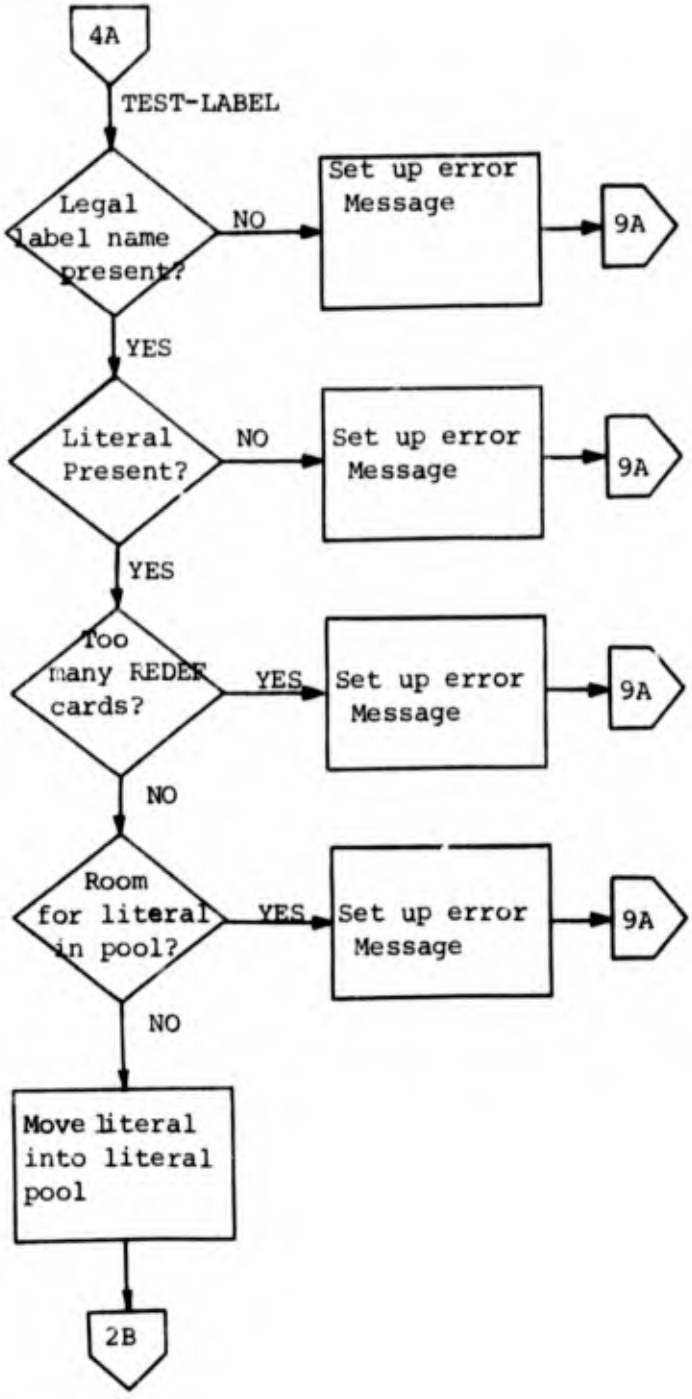
GET-NEXT-CARD

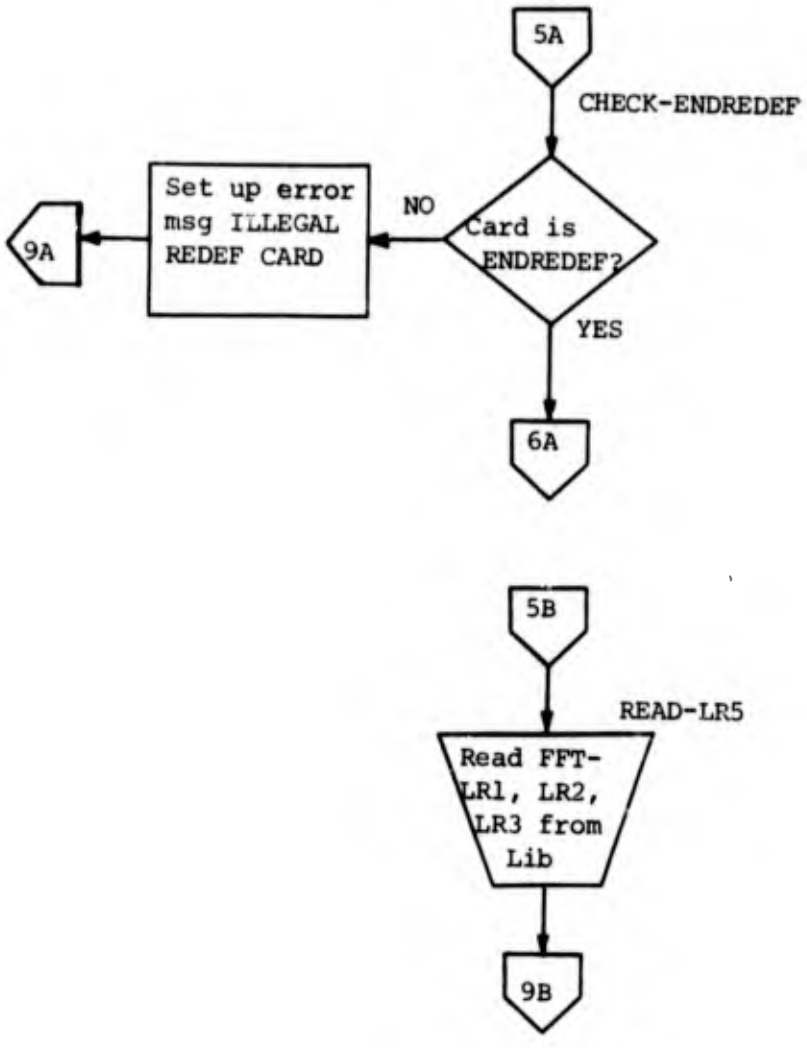
CALL-FMSCAN

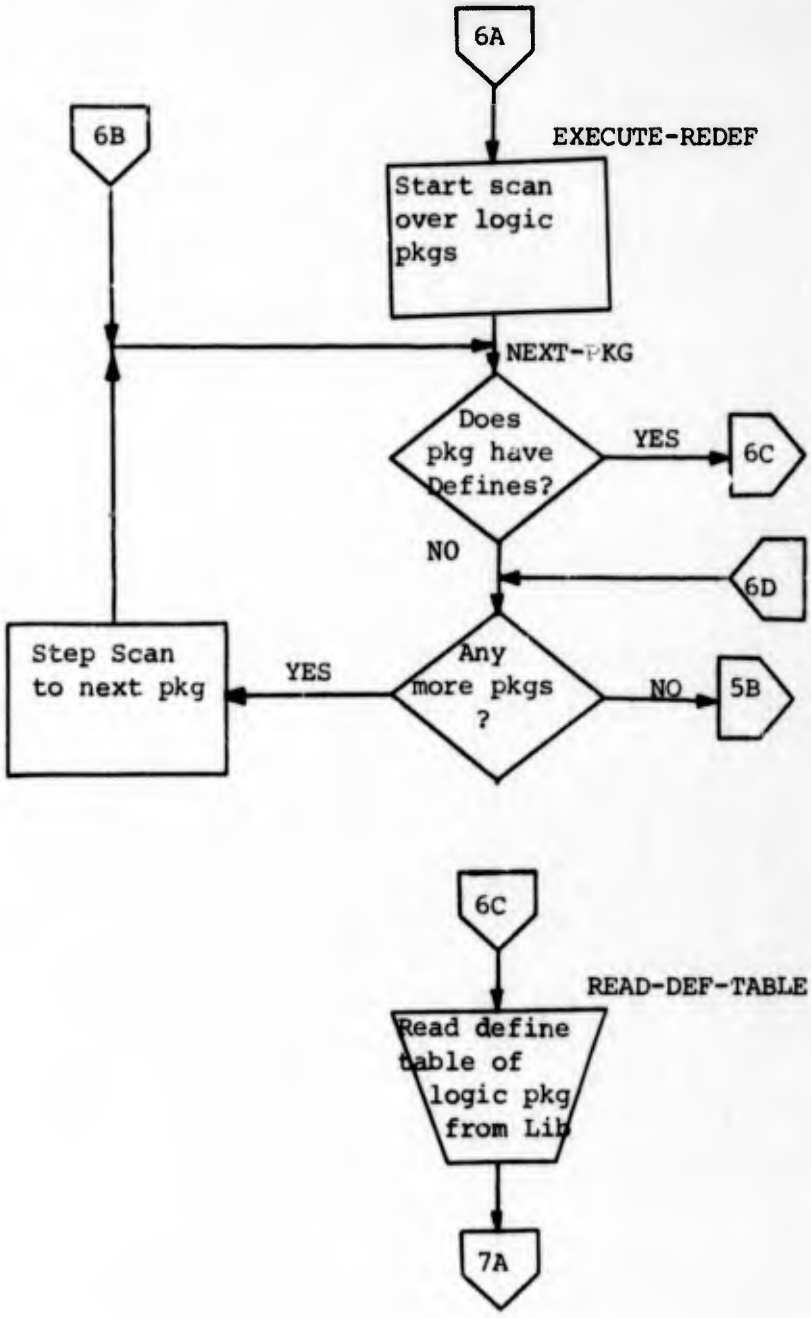


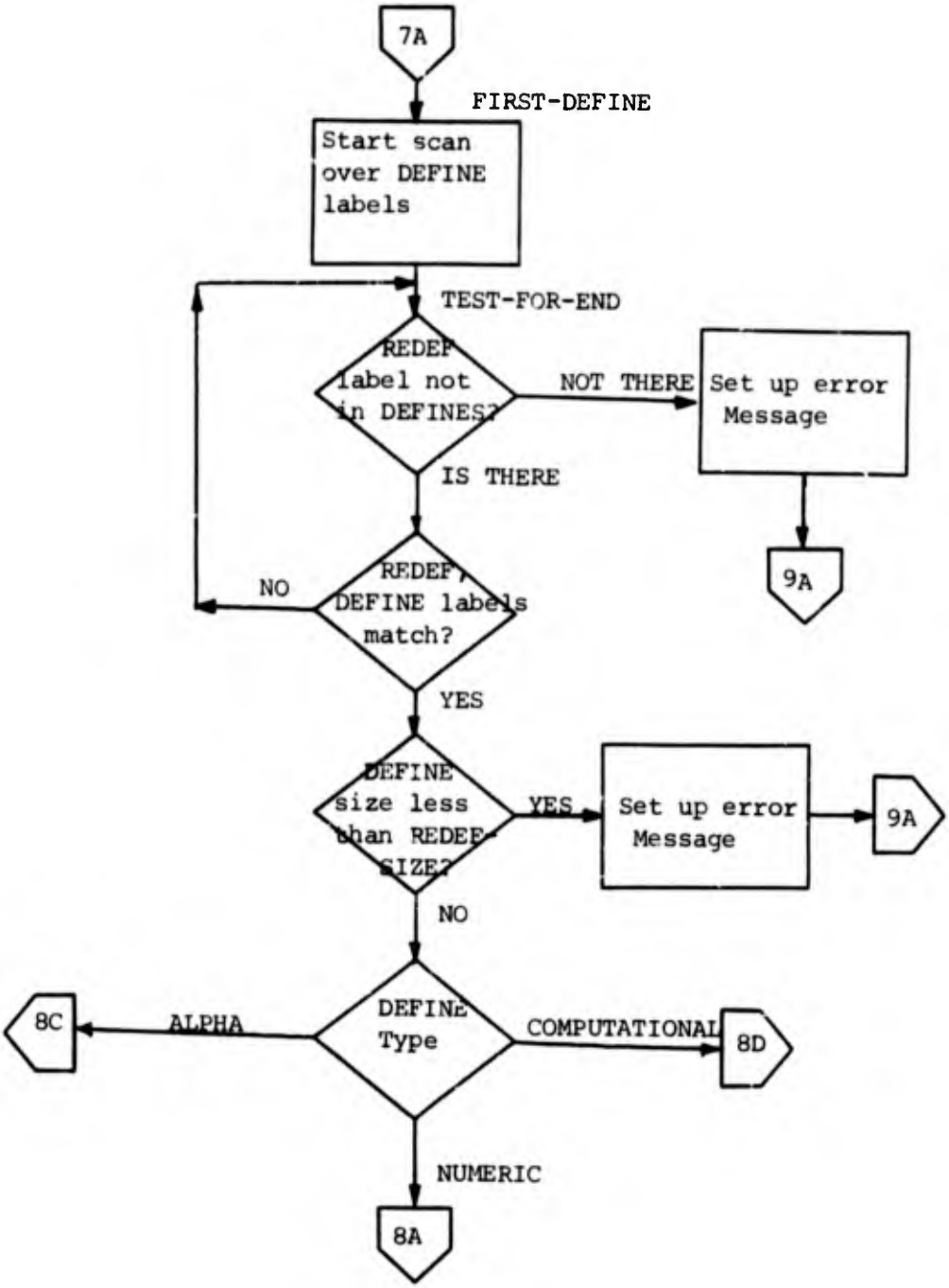

2A
MORE-REDEFX
LM job?
?
NO
Set up error message
9A
YES
2B
MORE-REDE?
Read Card & Scan it (FMSCAN)
Error found?
YES
9B
NO
EOF on cards?
NO
Print Card Image
YES
EOF on cards?
YES
Set up error msg "ENDREDEF CARD OMITTED"
9A
NO
3A

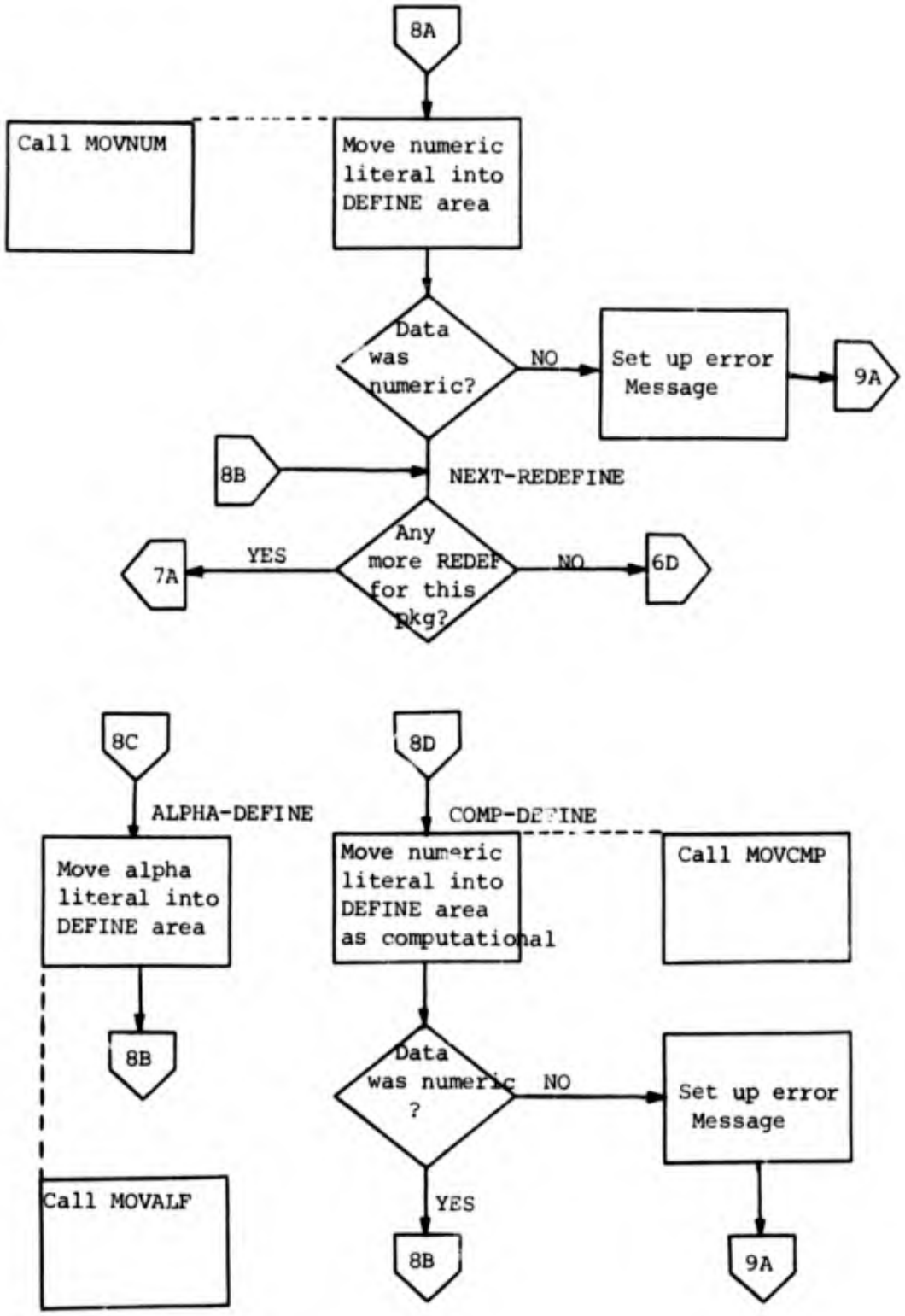

3A
REDEF card?
NO
5A
YES
Legal Pkg name present?
NO
Set up error msg
9A
YES
TEST-NAME
Scan logic pkg names from FMMP card to match REDEF
Match found?
NO
Set up error Message
9A
YES
4A

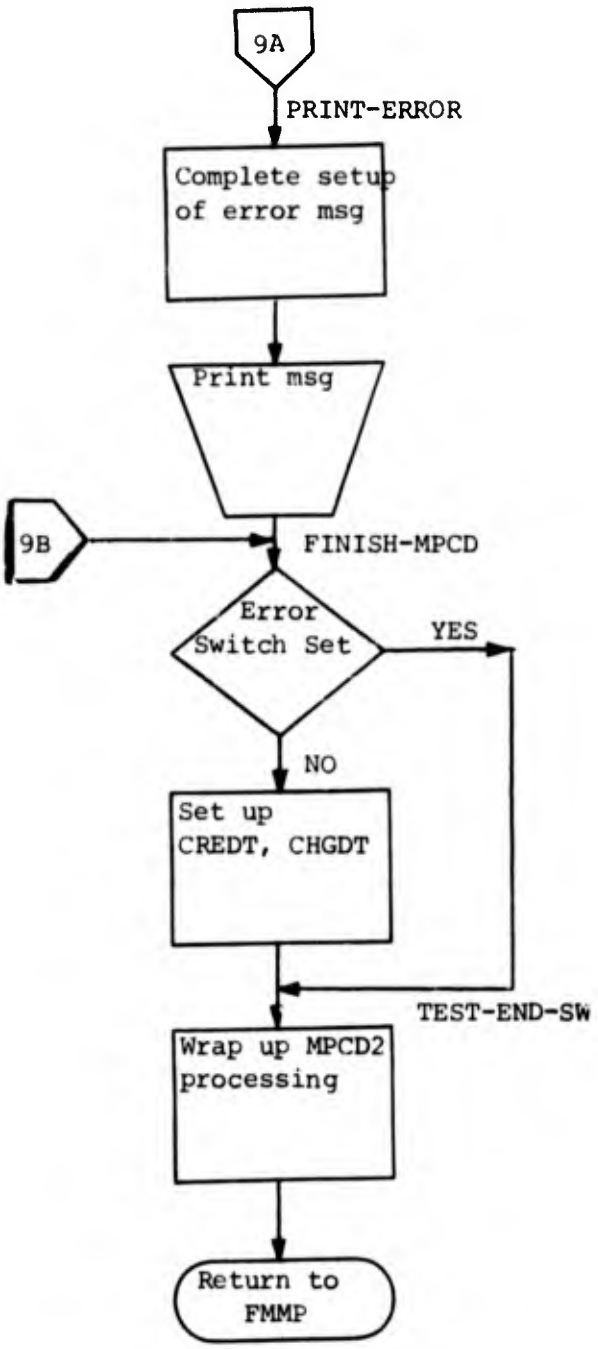

4A
TEST-LABEL
Legal label name present?
NO
Set up error Message
9A
YES
Literal Present?
NO
Set up error Message
9A
YES
Too many REDEF cards?
YES
Set up error Message
9A
NO
Room for literal in pool?
YES
Set up error Message
9A
NO
Move literal into literal pool
2B


5A
CHECK-ENDREDEF
Card is ENDREDEF?
NO
YES
Set up error msg ILLEGAL REDEF CARD
9A
6A
5B
READ-LR5
Read FFT-LR1, LR2, LR3 from Lib
9B

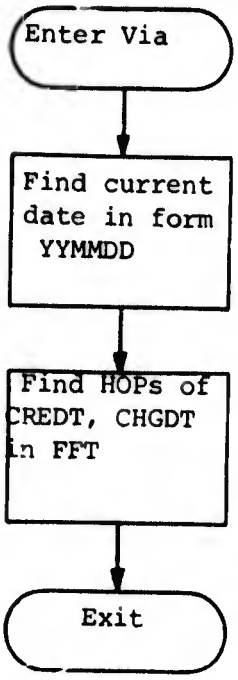

6A
EXECUTE-REDEF
6B
Start scan over logic pkgs
NEXT-PKG
Does pkg have Defines?
YES
6C
NO
6D
Step Scan to next pkg
YES
Any more pkgs ?
NO
5B
6C
READ-DEF-TABLE
Read define table of logic pkg from Lib
7A


7A
FIRST-DEFINE
Start scan over DEFINE labels
TEST-FOR-END
REDEF label not in DEFINES?
NOT THERE
Set up error Message
9A
IS THERE
REDEF, DEFINE labels match?
NO
YES
DEFINE size less than REDEF SIZE?
YES
Set up error Message
9A
NO
DEFINE Type
8C
ALPHA
COMPUTATIONAL
8D
NUMERIC
8A

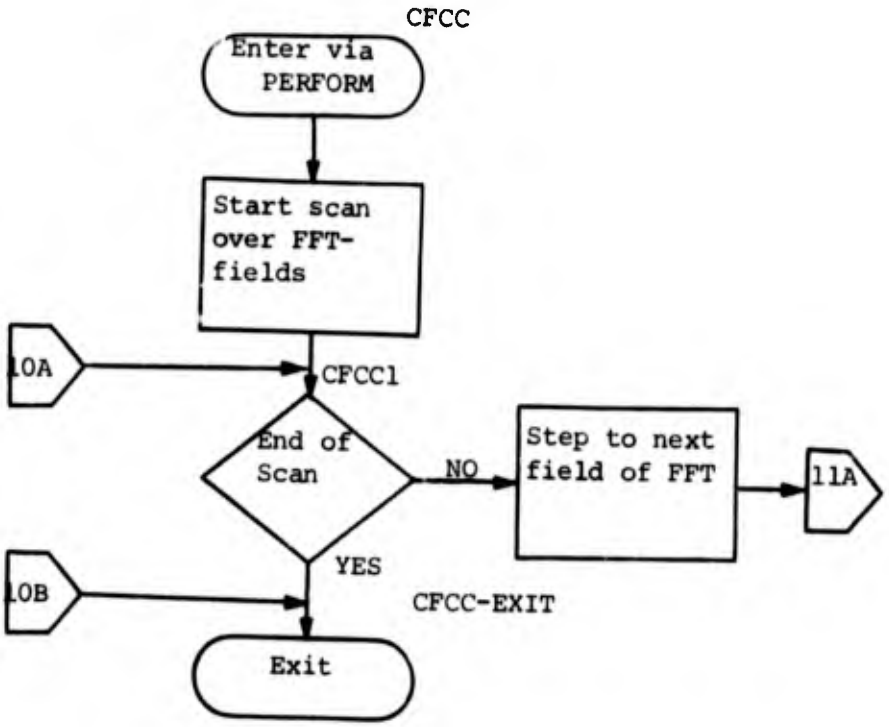

8A
Call MOVNUM
Move numeric literal into DEFINE area
Data was numeric?
NO
Set up error Message
9A
8B
NEXT-REDEFINE
Any more REDEF for this pkg?
YES
7A
NO
6D
8C
ALPHA-DEFINE
Move alpha literal into DEFINE area
8B
Call MOVALF
8D
COMP-DEFINE
Move numeric literal into DEFINE area as computational
Call MOVCMP
Data was numeric ?
NO
Set up error Message
YES
8B
9A

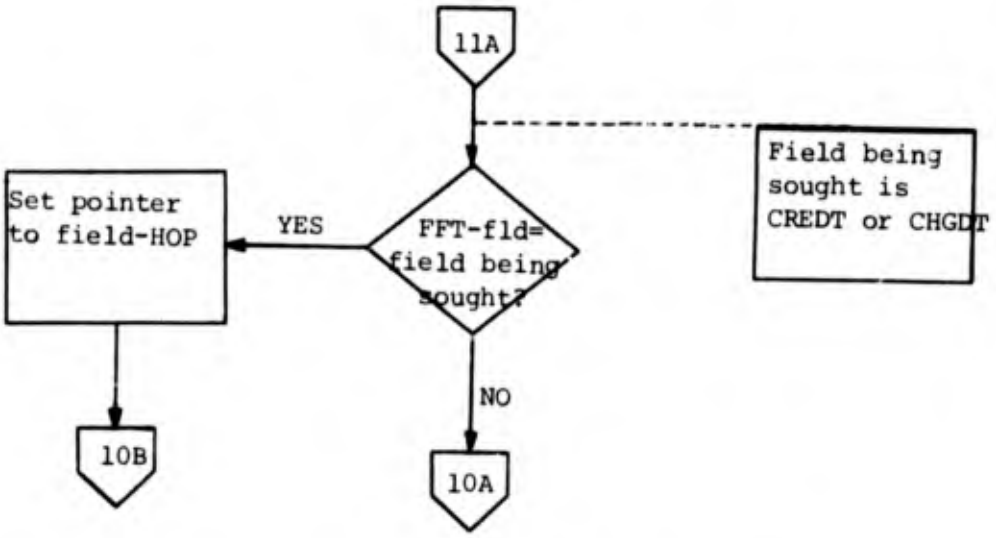

9A
PRINT-ERROR
Complete setup of error msg
Print msg
9B
FINISH-MPCD
Error Switch Set
YES
NO
Set up CREDT, CHGDT
TEST-END-SW
Wrap up MPCD2 processing
Return to FMMP

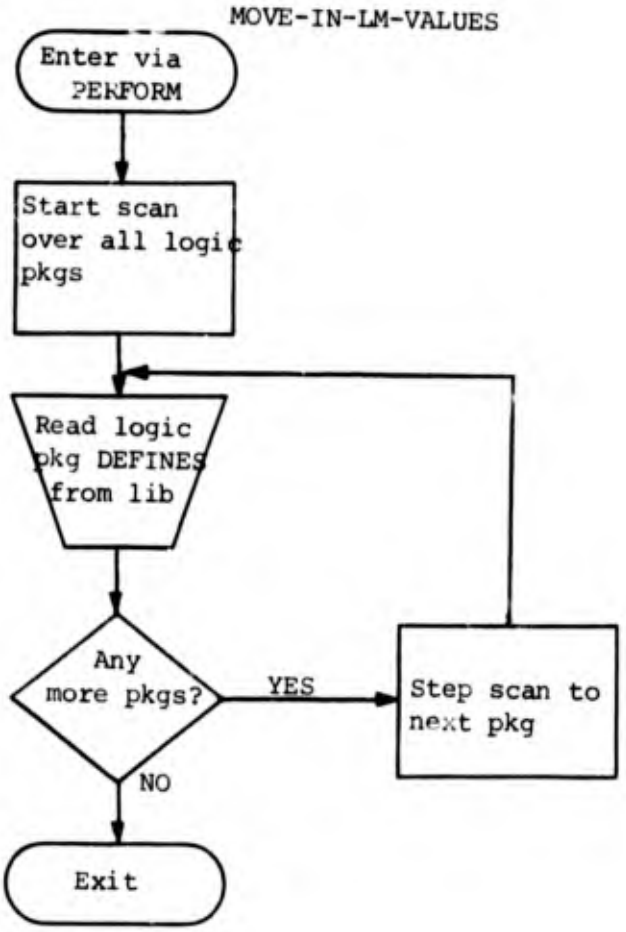
CHECK-FOR-CREDT-CHGDT



CFCC




11A
Field being sought is CREDT or CHGDT
Set pointer to field-HOP
YES
FFT-fld= field being sought?
NO
10B
10A


MOVE-IN-LM-VALUES
Enter via PERFORM
Start scan over all logic pkgs
Read logic pkg DEFINES from lib
Any more pkgs?
YES
Step scan to next pkg
NO
Exit

d. OMMPX

(1) Function. This routine performs three basic functions:

(a) It puts transactions together to make new data file records. Where the transaction-ID is matched by an old record-ID, an updated record results. If there is no match, the result is a generated record.

(b) Any confirmations which have been requested are formatted and printed.

(2) Calling Sequence.

(a) There is one entry-point: OMMP. OMMP is loaded by FMMP whenever a new record-ID is to be processed.

```
ENTRY 'OMMP'     USING MP-DD INPUT-RECORD MAJOR-RECORD
                       MAJOR-LR2 MAJOR-LR3 TRANS-TABLE FM-DD.
CALL  'FMSET'    USING FMIO.
CALL  'FMPRT'    USING PRT-LINE  CC.
CALL  'MOVALF'   USING ...
CALL  'MOVNUM'   USING ...
CALL  'COMARAYS' USING ...
```

(b) Some of the more important fields passed to OMMP are listed below, with their functions:

1. MAJOR-LR1, MAJOR-LR2, MAJOR-LR3 are the first three logical records for the FFT of the file being maintained. The latter two are passed by DEFINE-AREA and received in MAJOR-LR2 and MAJOR-LR3.

2. MAJOR-EOF-SW indicates when end of file has been reached on the old data file.

3. GENERATED-RECORD-SW indicates that a new record has been generated on the new data file.

4. DELETE-RECORD-SW indicates a data file record (input) is being deleted from the output.

5. CHANGE-ID-SW indicates that the record-ID of an input data file record is being changed in output.

6. TRANS-SW indicates whether no transactions have been processed yet, some transactions have been processed (but not all), or all transactions have been processed. It is mainly used for communication with the transaction-input routine OMMPTRN.

7. CONFIRM-SW indicates whether confirmation was requested for no fields, some fields, or all fields. If some fields are to be confirmed, the fields in question are indicated by the CONFIRM-ITEM switches. Each of these switches corresponds to the field with the same entry-number in MAJOR-LR2.

8. OM-SW indicates the status of Ordinary Maintenance:

0 = no OM
1 = OM
2 = LMCONS
3 = Record deleted by OM; do not write out
4 = Transactions need more core - open AREA2 with 2K more.

(c) The transactions are held in fields TRANS-TABLE, TRANS-TEST and TRANS-LIST. For details, see section on "Transaction Formats."

(3) Program Description.

Within OMMP there are certain switches that designate the status of the transaction, the record and the operation to be performed. These switches will be discussed first since they are applicable to all processing within OMMP. Each transaction is given a designation of three numeric characters which indicates the opr:

ADD	010	ATC	110
SUB	020	NEX	120
CHG	030	CMA	130
CHGV	040	ADS	140
GEN	050	REP	150
GENU	060	REG	160
GEC	070	GER	170
GECU	080	GEV	180
CID	090	ATV	190
DEL	100	CHV	200

and the type of field that the operation is to be performed on (this value is added to the opr indicator):

RECORD ID	(ID)	1
FIXED FIELD	(FX)	2
PERIODIC FIELD	(PE)	3
VARIABLE SET	(VS)	4
PERIODIC SET	(SV)	5
PSC	(PC)	6
VSC	(VC)	7
LMCON		8

The second important switch is called SW-BOX. Depending on the operator-field-type value (as described above), a predetermined SW-BOX value is moved in. This contains three numeric characters:

OLD-RECORD-SW.
Indicating the required status of the record (1 if must be present, 2 if must be absent, 3 for don't care).

OLD-SUBSET-SW.
Indicating the required status of the subset (same code as OLD-RECORD-SW).

OPR-ACTION-SW.
Indicating the operation to be done:

1 = NEX
2 = CHG
3 = DEL-ID
4 = DEL-PSC or DEL-VSC
5 = DEL-SUBSET
6 = Arithmetic (ADD or SUB)
7 = CHG-VSET
8 = Whole record transaction
9 = Skip transaction (set only when an error-condition is detected).

Therefore, if the opr code was given as 055, the SW-BOX would contain 322.

The transaction opr (055) indicates a GENerate on a periodic SUBset. In order for this transaction to be valid, the record may be either absent or present (SW-BOX(1)=3), the subset must be absent (SW-BOX(2)=2), and the opr-action is CHG (SW-BOX(3)=2). To validate that this is the case two other switches are checked which indicate the status of the record. These are REC-STAT-SW and SUB-STAT-SW. They are set as follows:

1 = record not present
2 = record present
3 = record successfully generated
4 = record provisionally generated.

By testing the required values against the actual status values, it is determined whether the transaction is valid or invalid. One additional switch is used chiefly for the record confirmation trailer indication. GEN-CHG-SW is set to:

1 = generated
2 = updated
3 = deleted.

START-OMMP.

Does the initialization that is needed once per run, and for transaction record when a combination OM-LM run is indicated. From LR3, it is determined which periodic set is the last one and also which set is the first variable set.

INIT-FOR-TRANS.

Does the initialization needed once per transaction. It also determines if the transaction is to be treated as a generate or as a change. If a change, the old fixed set is moved into the output area; otherwise, in MP-3 the transaction record-ID is moved to the output area.

GET-NEXT-TRANS.

Determines if there are more transactions for this record and if so goes to MP-5; otherwise, the wrapup of the record is performed. This process (MP-3A thru MP-4B) does the confirmation for the last processed variable set (MP-3A1); prints the confirmation trailer, i.e. updated, generated, deleted (MP-3B); determines if a record is to be written out (MP-3B1); copies from the input record to the output record all remaining sets if applicable (MP-4, MP-4A); and inserts the record mark at the end of the record and moves in the record character count (MP-4B).

MP-5.

Finds the next sort-order transaction for this record and moves it into a non-subscripted area. A check is then made to determine if a whole-record transaction has been followed by a non-whole-record transaction on the same rec-ID.

MP-7.

Does a Computed Go To depending on the operator for this transaction and moves in the required absent/present indicators as well as an indicator for the type of operation action. If there is a valid operation code, then control is passed to MP-8; otherwise, an invalid code has been passed from IP and FALSOP forces an abnormal termination.

MP-8.

Determines if the operation code is consistent with the absence/presence of the record, i.e. illegal to GEN record on a record that exists. The different conditions are checked for validity in CK-REC1 thru SET-UP-ERROR.

If no errors are detected, FIXED-FIELDS determines if the present transaction is for a fixed field. If not control goes to PERIODIC-FIELDS; otherwise, the high order position of the field is set up for the input record and the output record.

FIXED-AND-PERIODIC.

Sets up the pointers to move the changed field into the output record. It determines if the transaction is on the record-ID. if so goes to RECID-PROCESSING. It also checks to see if the transaction is trying to change the PSS.

MP-9.

Goes to the appropriate paragraph depending on the value of opr action code.

CHG-OPR.

Determines if the transaction is attempting to update the RECCT or put alphanumeric data into a numerically defined FFT field.

CALL-MOVALF.

Handles all the moving of transaction data to the output data record and sets NO-CHANGE-SW to Ø to indicate that the record has been changed.

MP-1Ø.

Makes all provisional generates definite generates after the valid transaction has been processed.

CONFIRM-CK.

Determines if any confirmations are desired and if the confirmation header has been printed.

NEX-CONF.

Handles the action for the NEX opcode. It prints out the value of the field in the old record and what the value would be if the transaction were applied.

MP-11 thru MP-12.

Determine if the transaction data is strictly numeric.

MP-13 thru MP-13A.

Reset switches REC-STAT-SW and SUB-STAT-SW from "provisional generate" to "not present" for transactions which have been invalidated.

MP-14.

Sets CHANGE-ID-SW when a change to record-ID has been validated (see RECID-PROCESSING below).

RECID-PROCESSING.

First check for the CID (change id) opcode. If it is CID and is valid control is passed back to MP-9 which will eventually move in the transaction data. If the opcode is DEL-ID control goes to DEL-RECID. MP-14A thru MP-14B checks to see if the transaction data is the same as the data file record, if so, there is no need to move the transaction data in as it has been moved in already in MP-3; otherwise control goes eventually back to MP-9 where the data will be moved in.

DEL-RECID.

Moves blanks to the record-id and sets DELETE-RECORD-SW to 1.

DEL-PSC-VSC.

Moves 'D' to the left-most position of the PSC/VSC. At a later time (when the set is reached) a check is made for the 'D' and if present the old set is skipped over.

WHOLE-RECORD.

Checks to insure that this is the only transaction against this record. A determination is then made as to whether the transaction data record-id is a change or a generate. If a change condition is present the record is considered as out of sort and CHANGE-ID-SW is set to 1 to indicate this. In any case the transaction data is moved into the output area as an entire

record. The first four characters are assumed to be the correct RECCT - no check is made for validity of the RECCT or the control fields.

PERIODIC-FIELDS.

First determines if the transaction is on a variable set. If so, control goes to VSET-FIELDS; otherwise processing is performed to search thru the old record until the desired periodic set is reached. If the previous transaction was rejected on a periodic subset (SUB-STAT-SW = 1) the pointer must be backed up to the prceding subset. If the set is to be deleted it is skipped over.

MP-15.

Checks for pseudo subset notations and goes to PSEUDO-PROCESSING if present. If there is no old file or no old record a blank subset is built with the desired PSS number; otherwise the old subsets are stepped thru, copying each one onto the output file, until the desired location in the old record is reached where the blank subset is built with the desired PSS number.

MP-16 thru CK-SUB3.

Validate that the requirements for presence/absence of the subset are met.

MP-17.

Controls the loop for stepping thru the subsets.

MP-19 thru MP-19-EXIT.

Builds the blank subset with the desired PSS number.

MOVE-SUBSETS.

Compares the current PSS number of the subset we are pointing at, to the one desired. INDICATOR equal to 3 signifies that the desired location in the set has not been reached.

MOVE-SUBSET-DATA.

Moves the collection of subsets, up to the position of insertion or change, into the output record.

NO-OLD-SET.

Determines if the subset desired is a generate or a change.

PSEUDO-PROCESSING thru MP-21A1.

Handles processing of all subset numbers greater than 599 and converts the PSS number to the next sequential PSS number. A pseudo subset is placed at the end of all existing subsets in that particular set. This entails copying all of the old existing subsets into the output record.

MOVE-SETS-NOT-PROCESSED thru MP-23B.

Handle the processing which locates the desired transaction set. First is a determination of whether any subsets from the last processed set have been skipped over; if so, the number of characters remaining in that set is computed by multiplying the number of remaining subsets by the subset length. These characters are then moved into the output record by MP-20 thru MP-20-EXIT. This initial process finishes up the processing of the last set. MP-22A now moves in the control field for the last processed set. MP-23 loops thru subsequent non-transaction sets in a similar manner (i.e.

computing the number of characters to be moved in, moving them in and generating the new control field) until the desired set is reached. GET-RIGHT-SET thru MP-23B handles the same process when there is no old input record.

VSET-FIELDS.
First determines if the transaction is on the same variable set. If not, the last transaction must be confirmed (if confirmation was specified). Confirmation on variable sets is only done upon the change of transaction set. There will be only one confirmation per variable set.

MP-24.
Controls the processing for copying into the output record the remaining characters of the previous transaction set and further determines if the present transaction set is present or absent.

MP-26 thru CK33.
Check the validity of the transaction against the presence/absence requirements.

VSET-OPR.
Checks for invalid combinations of operators on the variable set and goes to the appropriate paragraph for processing.

CHG-VSET.
Handles opcodes REP, REG, CHV, and CHG and determines if partial variable set processing is called for. It furthermore initializes pointers for multiple change operations.

MP-27.
Is the entry for GEC opcodes as well as handling the above cases that fall through. If the old variable set exists and there have been no good transactions on this variable set yet, skip over the old variable set.

MP-28.
Sets up the correct new character count in the variable set control field. MP-28A initializes the high order position of the variable set on the first transaction for the vset. MP-29 sets up the pointers and moves in the transaction data.

ATTACH-VSET.
Copies any existing vset data into the output record and then appends the transaction data onto the end.

PARTIAL-VSET thru MP-31.
Handle partial processing by copying the original vset data up to the first character of the change (MP-29B). The transaction data is then appended (MP-30). The number of old characters to be deleted is computed and are skipped over. Finally the remainder, if any, of the existing vset characters are copied into the output record (MP-29C).

DEL-SUBSET.
Blanks out the existing subset except for the PSS number. SUB-STAT-SW is set to 1 and saved in CNT4, then set to 2. SUB-STAT-SW equal to 2 will signify during confirmations that the old

subset exists. However, CNT4 will be used to restore SUB-STAT-SW to 1 when the next transaction is fetched, which will cause the pointers to be moved back to the beginning of the subset; subsequent transactions will overlay the old subset position, thus effectively causing deletion.

ARITHMETIC.

Handles the ADD and SUB opcodes. First the FFT field must be defined numerically. The old field is then fetched and is validated as containing strictly numeric data. If not, it is treated as Ø. The transaction data is next validated as numeric. Overflow conditions are checked for. The character to the left of the first significant digit is checked for sign indication. If a sign is present the FFT field must be defined as SGNUM. MP-32 moves in the updated field.

MP-33A.

Returns to FMMP.

PRT-ERR-MSG thru PRT-EXIT.

Handle the formatting and printing of all error messages. For exact format see MIDMS User's Reference Manual.

CONFIRMATIONS thru CONF-VAR-2.

Handle the formatting and printing of all confirmations. For exact format see MIDMS User's Reference Manual.

CONF-UPDATE-OR-GEN.

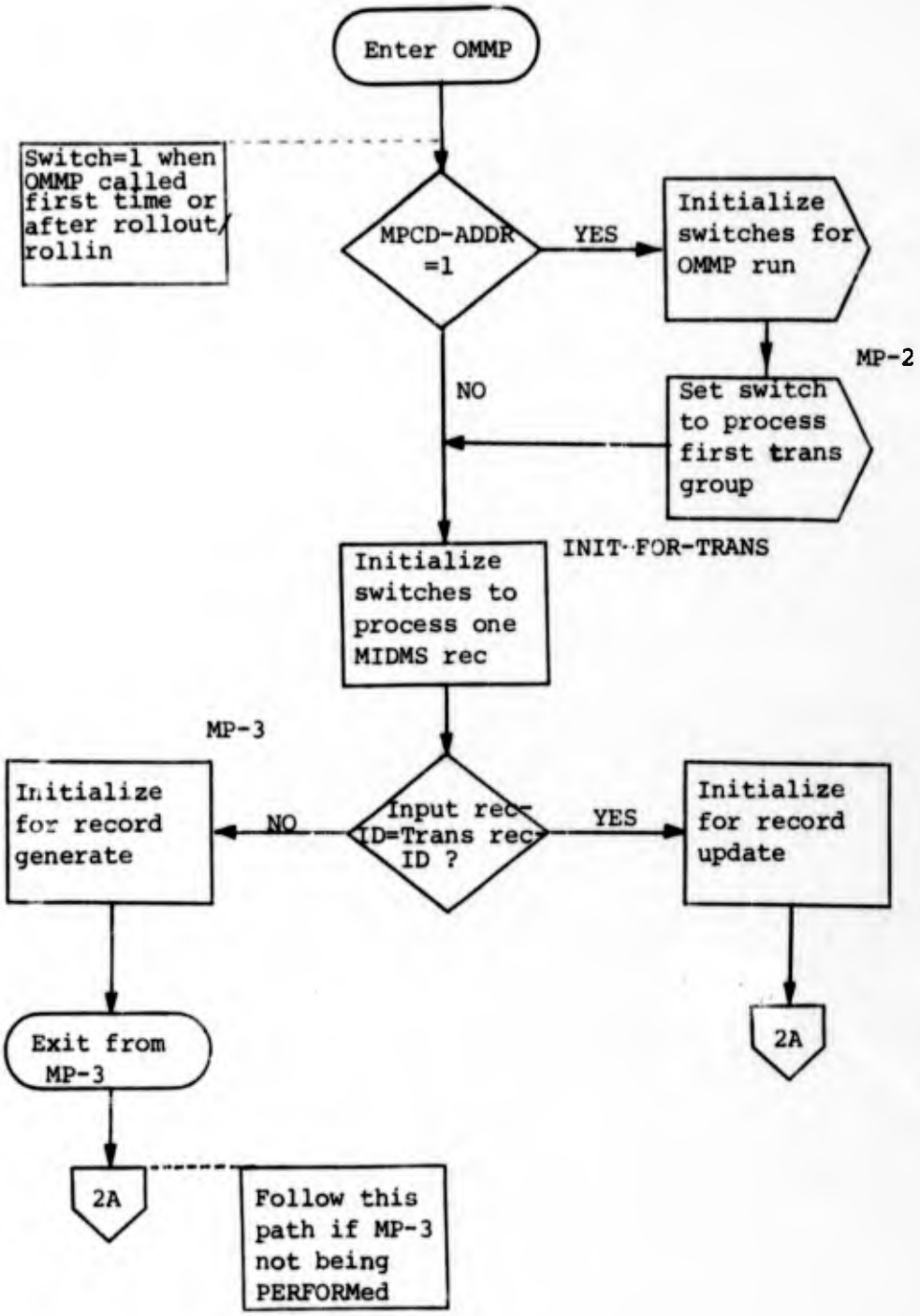
Produces the confirmation trailer (i.e., "generated," "updated," or "deleted").

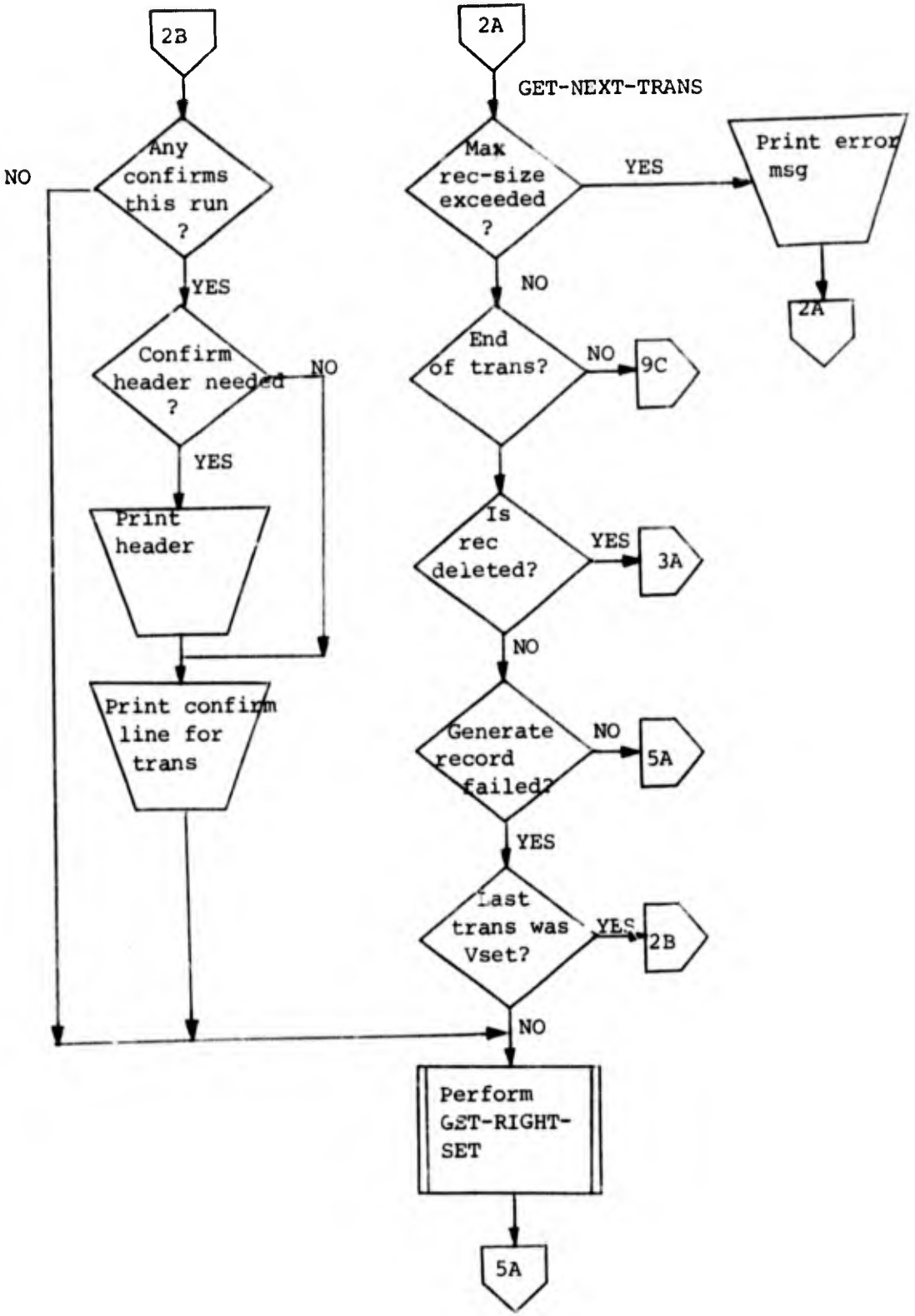
GET-VSET-FIELD.

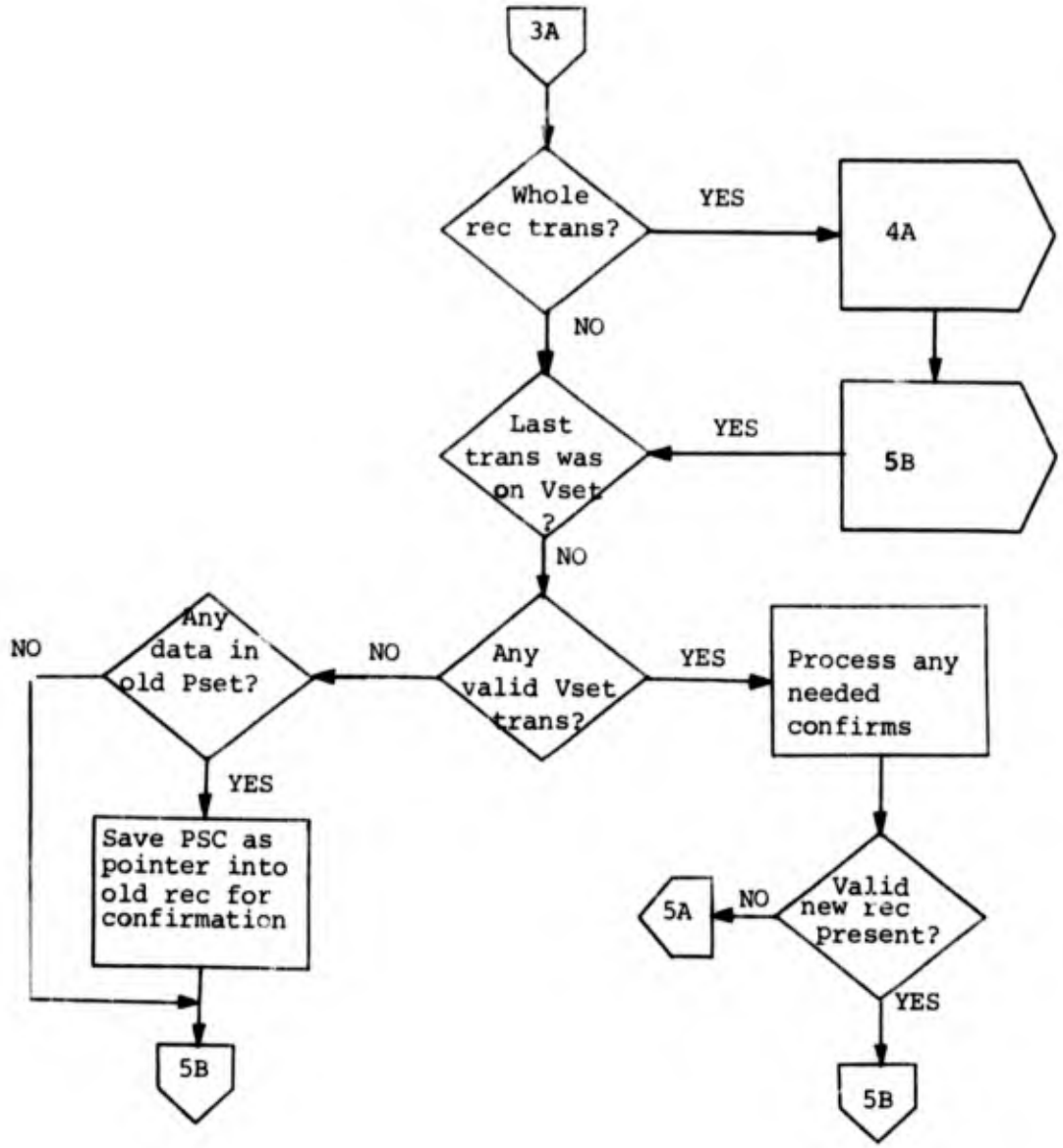
Is used to find the real variable set name when there is a partial change transaction since in this case T-FLDNO contains the delete length instead of the LR2 entry.

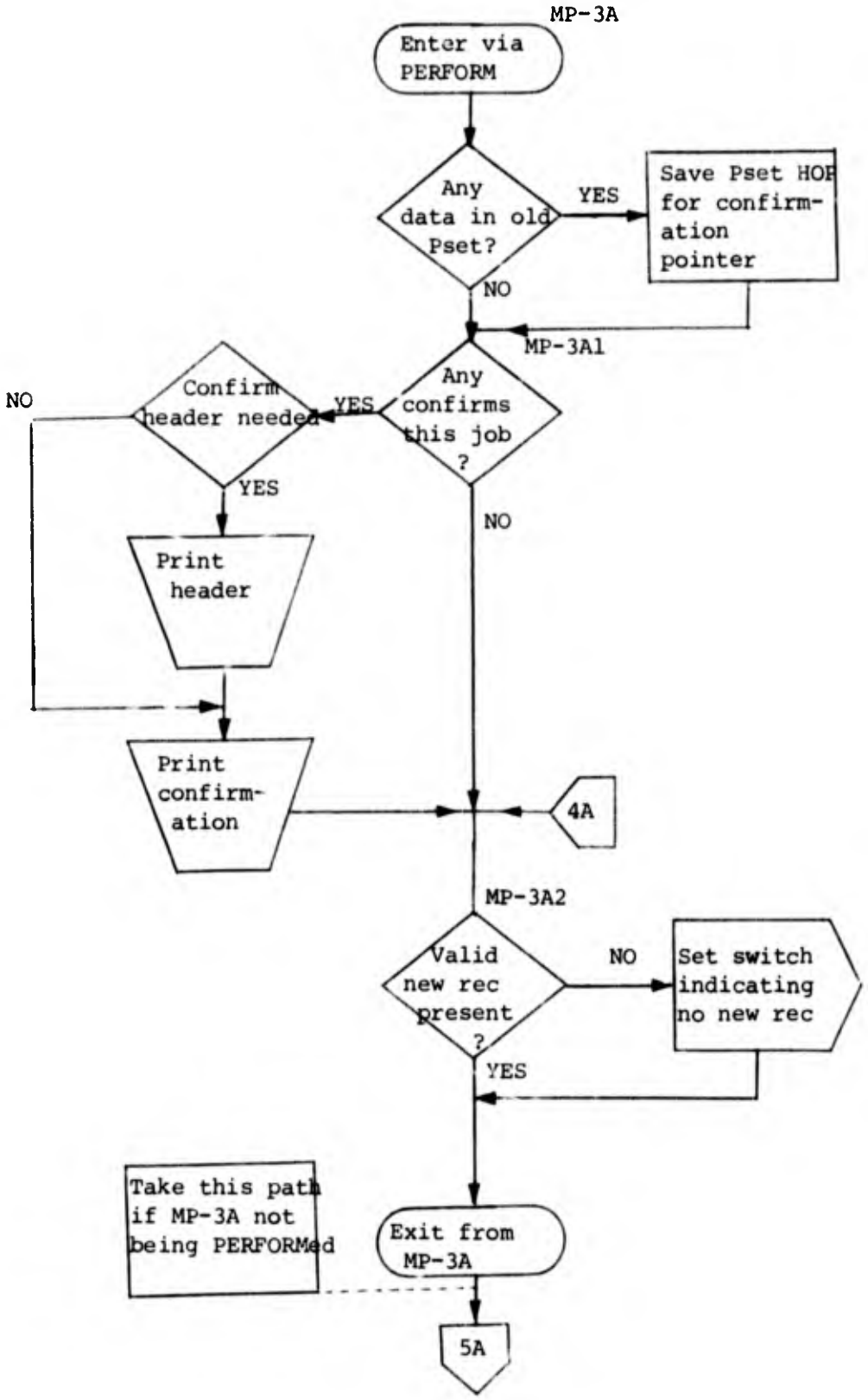
WHOLE-RECORD-CONF.

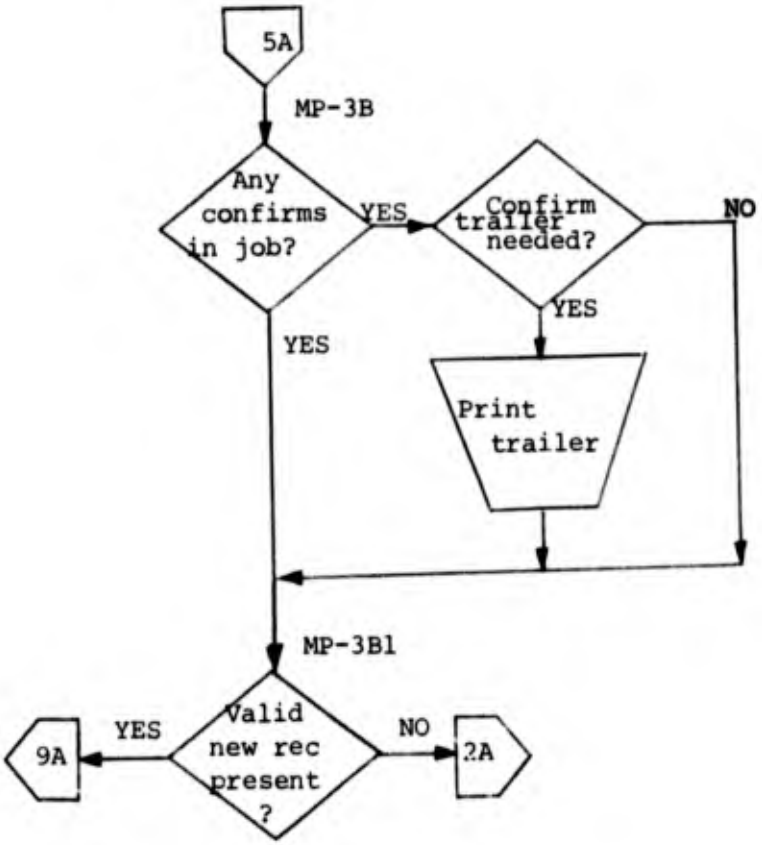
Sets up processing in the special case when whole record operators are used and the original record is larger than the replacement record. This processing will enable the total old record to be printed out in the confirmation listing.

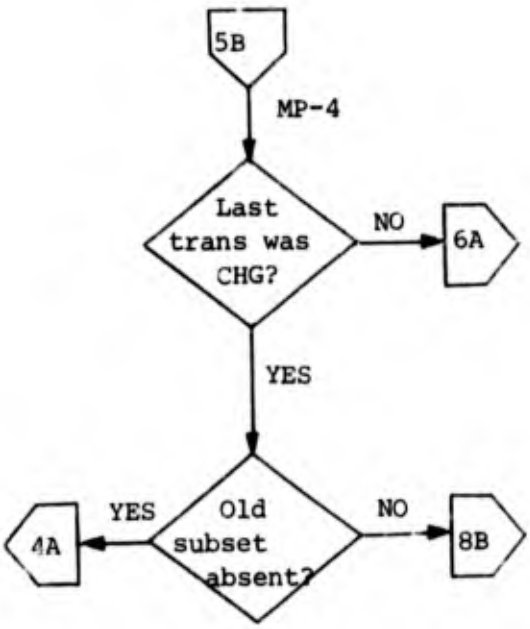

Enter OMMP
Switch=1 when OMMP called first time or after rollout/ rollin
MPCD-ADDR =1
YES
Initialize switches for OMMP run
MP-2
NO
Set switch to process first trans group
Initialize switches to process one MIDMS rec
INIT-FOR-TRANS
MP-3
Initialize for record generate
NO
Input rec-ID=Trans rec-ID ?
YES
Initialize for record update
Exit from MP-3
2A
2A
Follow this path if MP-3 not being PERFORMed

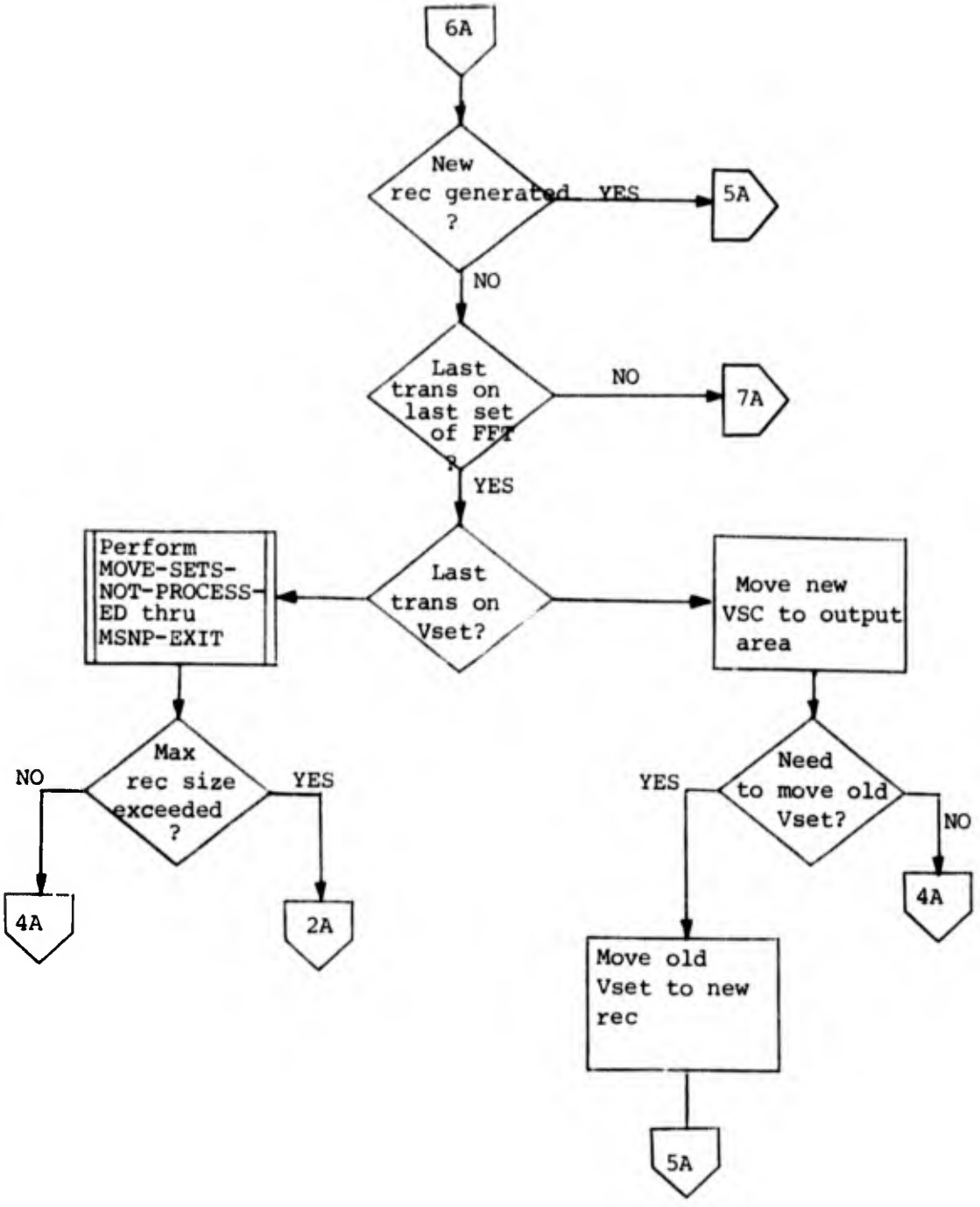

2B
Any confirms this run ?
NO
YES
Confirm header needed ?
NO
YES
Print header
Print confirm line for trans
2A
GET-NEXT-TRANS
Max rec-size exceeded ?
YES
Print error msg
2A
NO
End of trans?
NO
9C
Is rec deleted?
YES
3A
NO
Generate record failed?
NO
5A
YES
Last trans was Vset?
YES
2B
NO
Perform GET-RIGHT-SET
5A

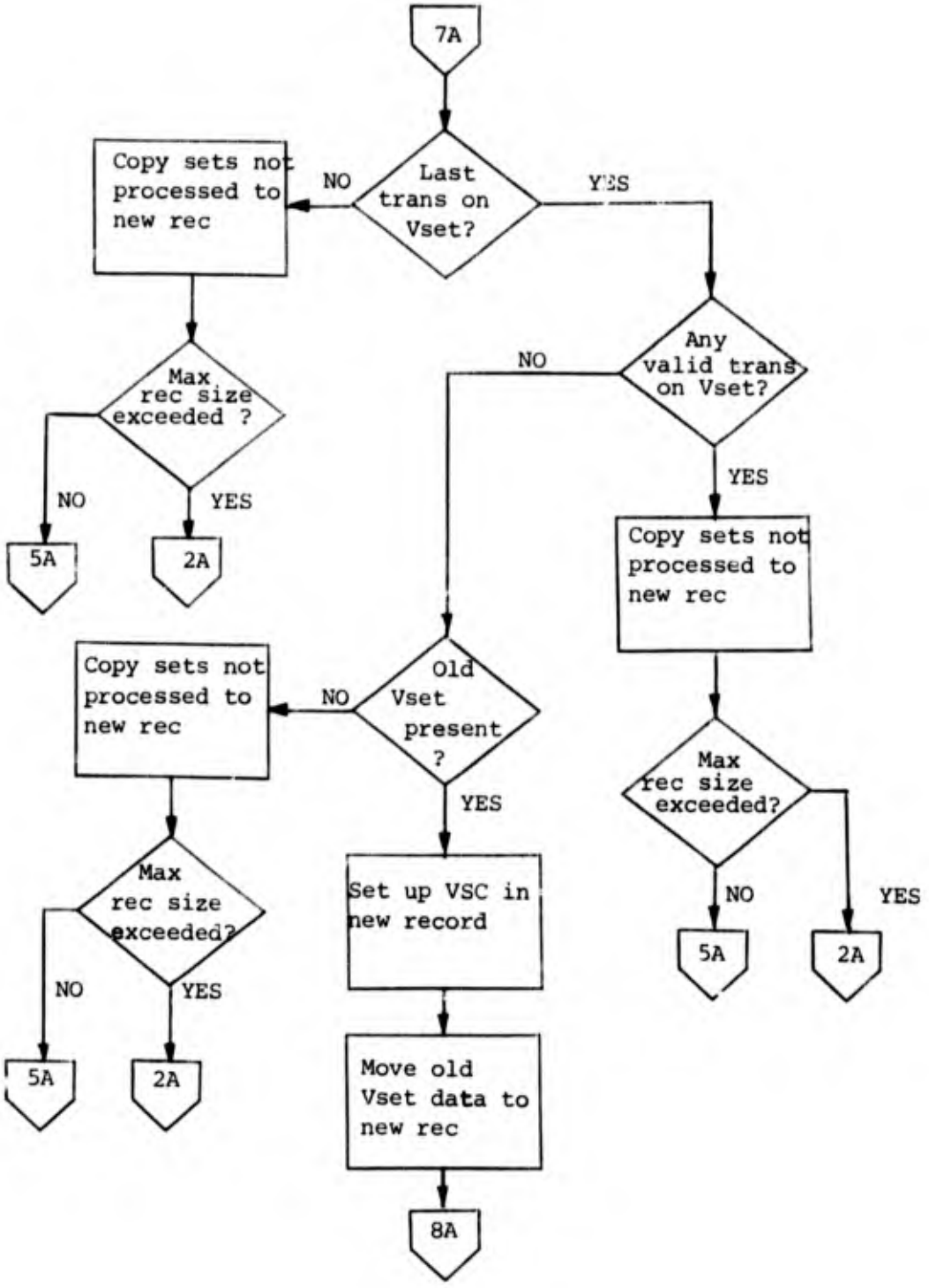

3A
Whole rec trans?
YES
4A
NO
Last trans was on Vset ?
YES
5B
NO
Any valid Vset trans?
NO
Any data in old Pset?
NO
YES
Save PSC as pointer into old rec for confirmation
5B
YES
Process any needed confirms
Valid new rec present?
NO
5A
YES
5B

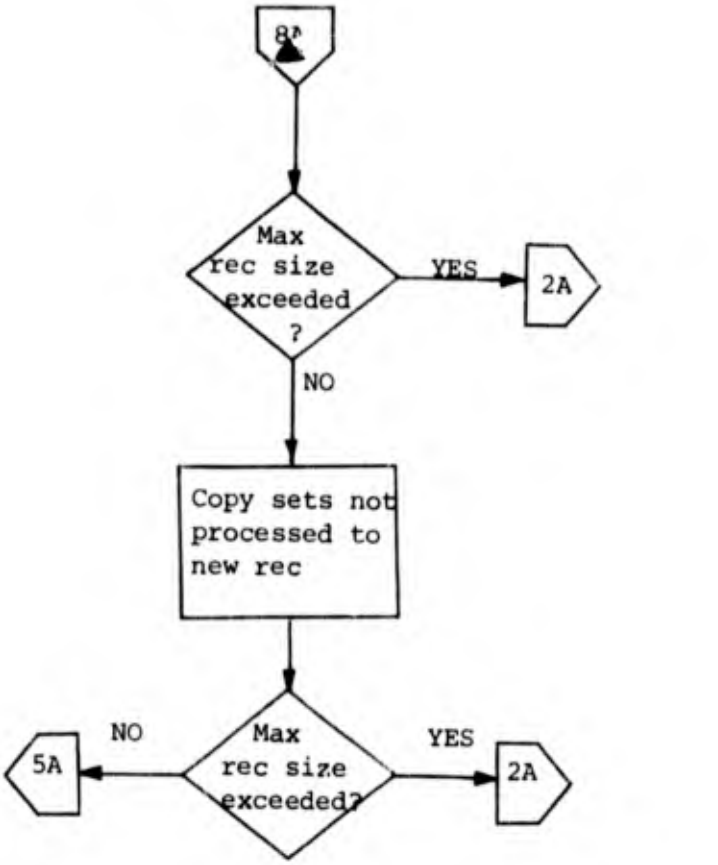

MP-3A
Enter via PERFORM
Any data in old Pset?
YES
Save Pset HOP for confirm-ation pointer
NO
MP-3A1
Any confirms this job ?
YES
Confirm header needed
NO
YES
Print header
NO
Print confirm-ation
4A
MP-3A2
Valid new rec present ?
NO
Set switch indicating no new rec
YES
Take this path if MP-3A not being PERFORMed
Exit from MP-3A
5A

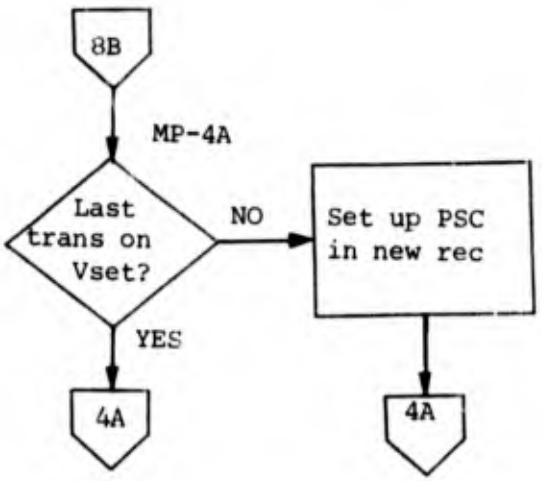

5A
MP-3B
Any confirms in job?
YES
Confirm trailer needed?
NO
YES
YES
Print trailer
MP-3B1
Valid new rec present ?
YES
9A
NO
2A

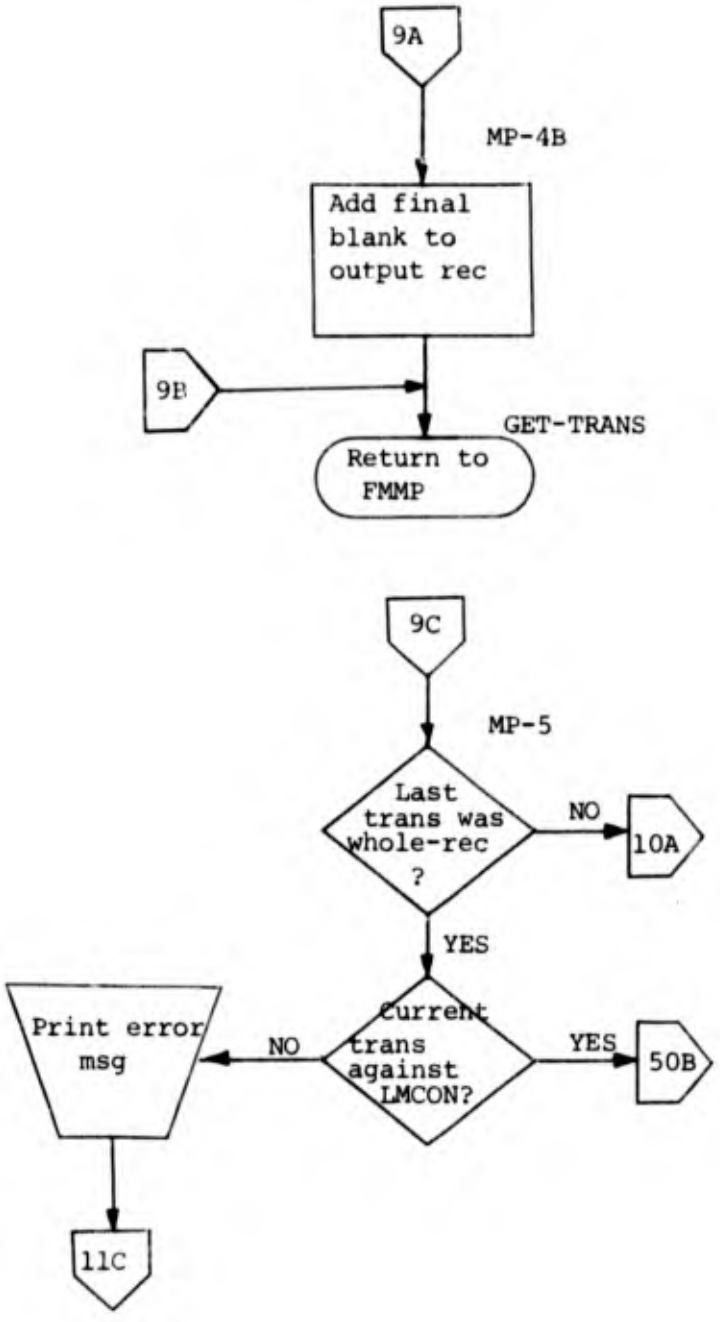

5B
MP-4
Last trans was CHG?
NO
6A
YES
Old subset absent?
YES
4A
NO
8B

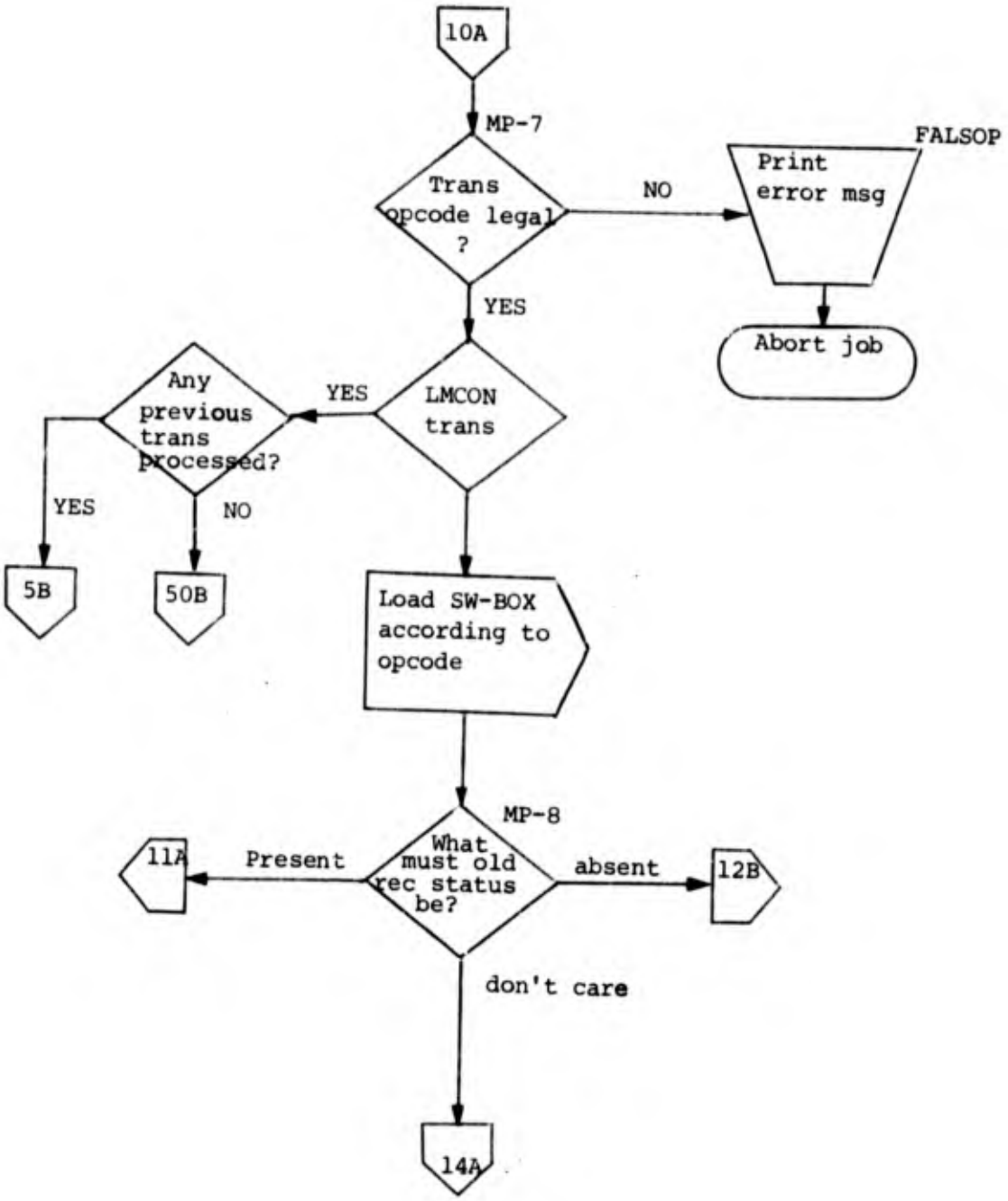

6A
New rec generated ?
YES
5A
NO
Last trans on last set of FFT ?
NO
7A
YES
Last trans on Vset?
Perform MOVE-SETS-NOT-PROCESSED thru MSNP-EXIT
Move new VSC to output area
Max rec size exceeded ?
NO
YES
4A
2A
Need to move old Vset?
YES
NO
4A
Move old Vset to new rec
5A

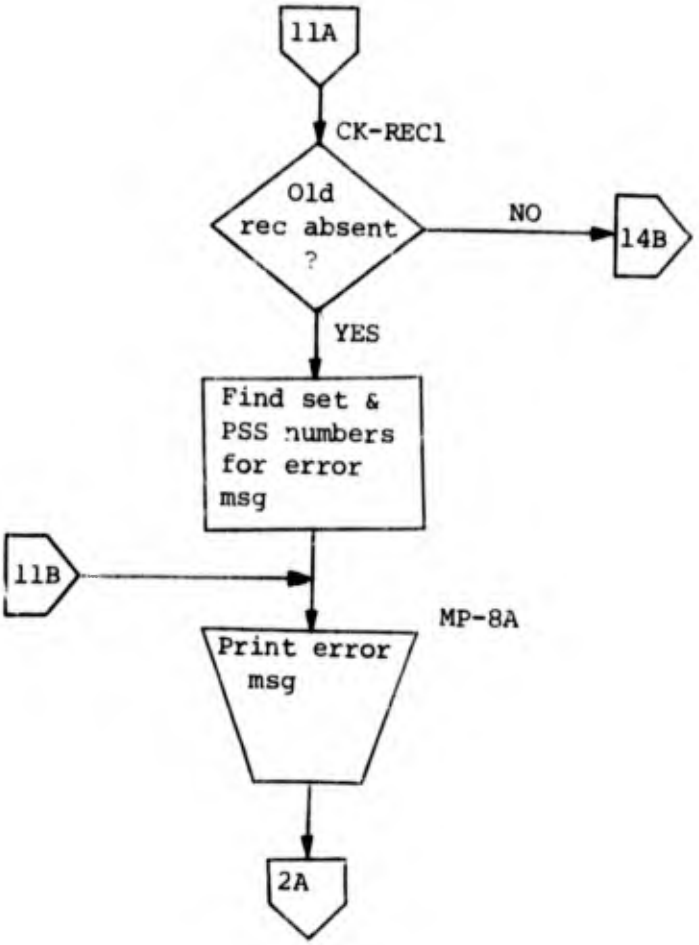

7A
Last trans on Vset?
NO
YES
Copy sets not processed to new rec
Max rec size exceeded ?
NO
YES
5A
2A
Any valid trans on Vset?
NO
YES
Copy sets not processed to new rec
Max rec size exceeded?
NO
YES
5A
2A
Old Vset present ?
NO
YES
Copy sets not processed to new rec
Max rec size exceeded?
NO
YES
5A
2A
Set up VSC in new record
Move old Vset data to new rec
8A

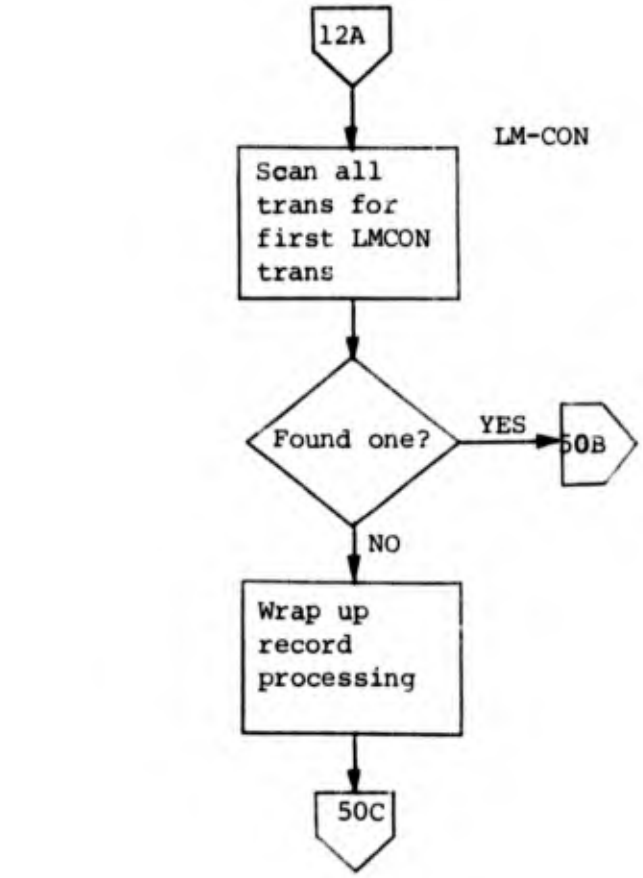

8A
Max rec size exceeded ?
YES
2A
NO
Copy sets not processed to new rec
Max rec size exceeded?
NO
5A
YES
2A

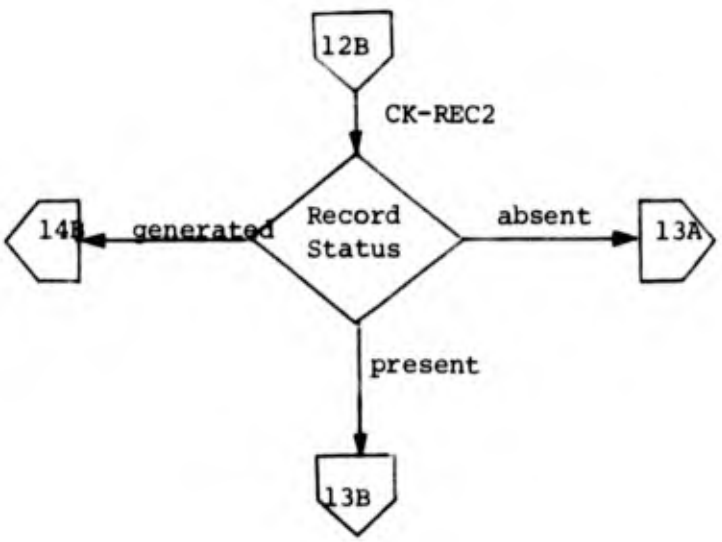

8B
MP-4A
Last trans on Vset?
NO
Set up PSC in new rec
YES
4A
4A

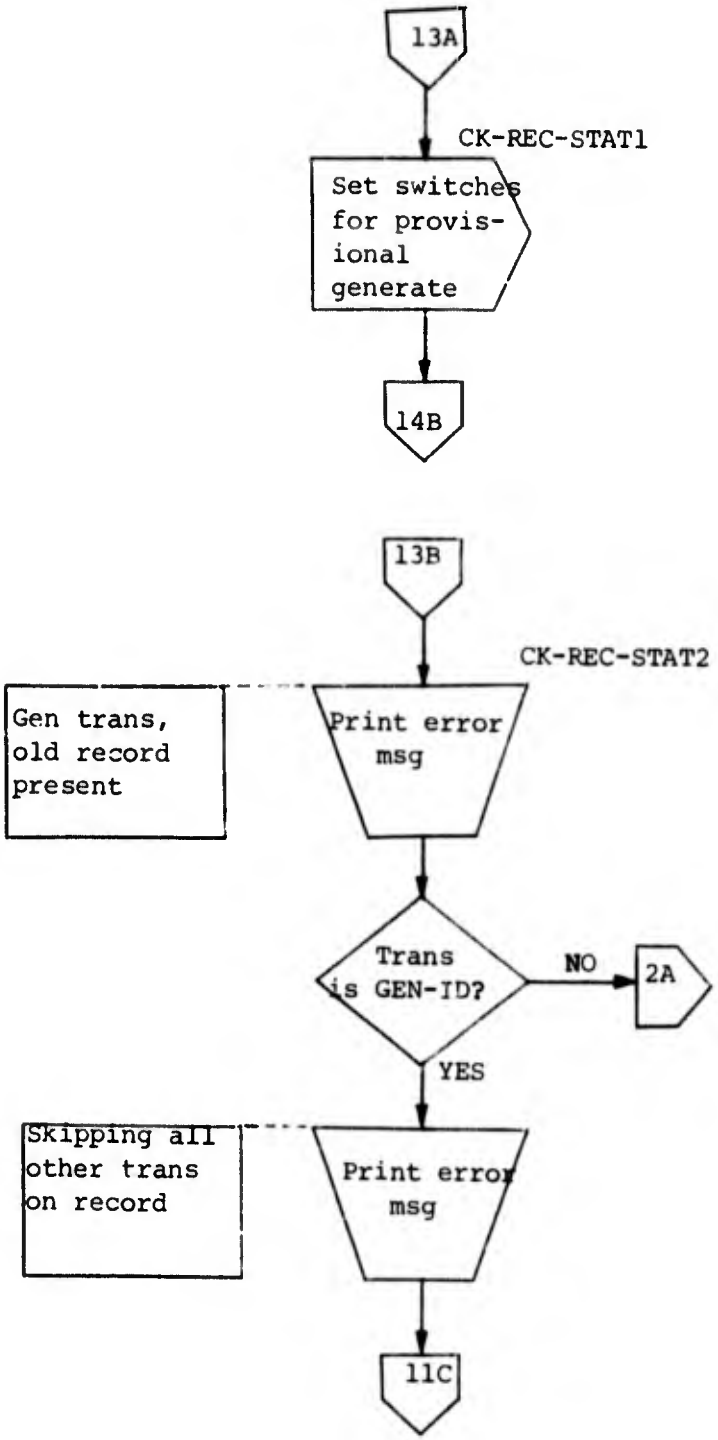

9A
MP-4B
Add final blank to output rec
9B
GET-TRANS
Return to FMMP
9C
MP-5
Last trans was whole-rec ?
NO
10A
YES
Current trans against LMCON?
NO
Print error msg
YES
50B
11C

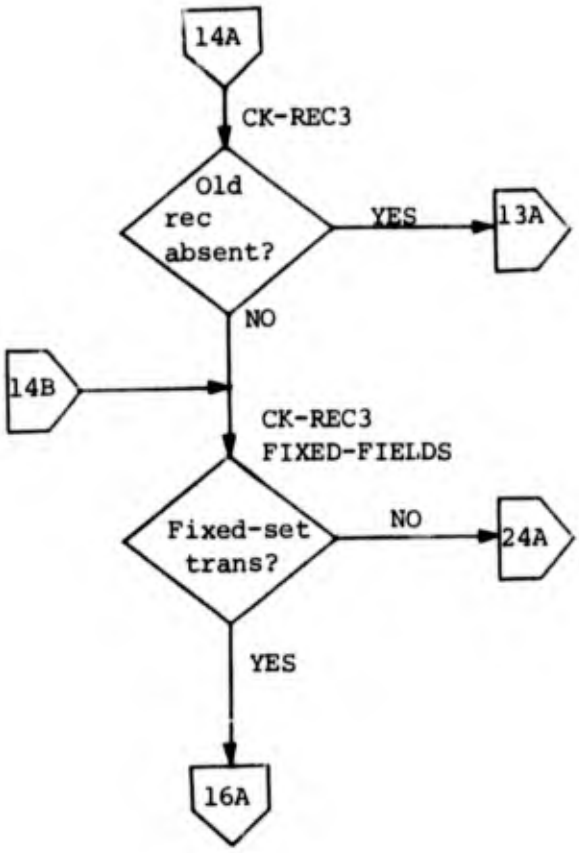

10A
MP-7
Trans opcode legal ?
NO
FALSOP
Print error msg
Abort job
YES
LMCON trans
YES
Any previous trans processed?
YES
NO
5B
50B
Load SW-BOX according to opcode
MP-8
What must old rec status be?
Present
11A
absent
12B
don't care
14A

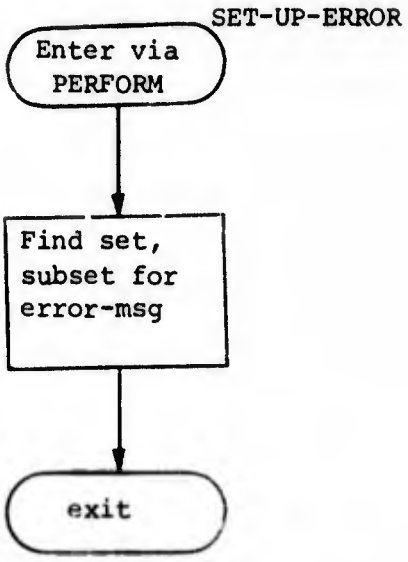

11A
CK-REC1
Old rec absent ?
NO
14B
YES
Find set & PSS numbers for error msg
11B
MP-8A
Print error msg
2A
11C
SKIP-REST-TRANS
Set switch indicating last
12A

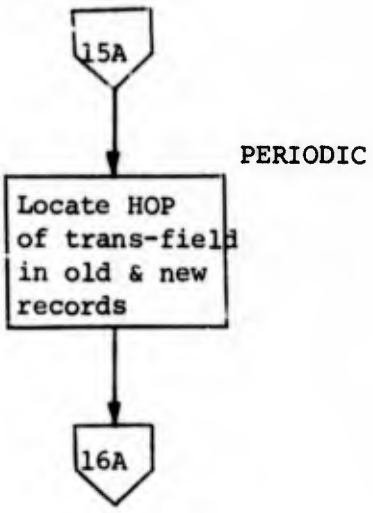

12A
LM-CON
Scan all trans for first LMCON trans
Found one?
YES
50B
NO
Wrap up record processing
50C

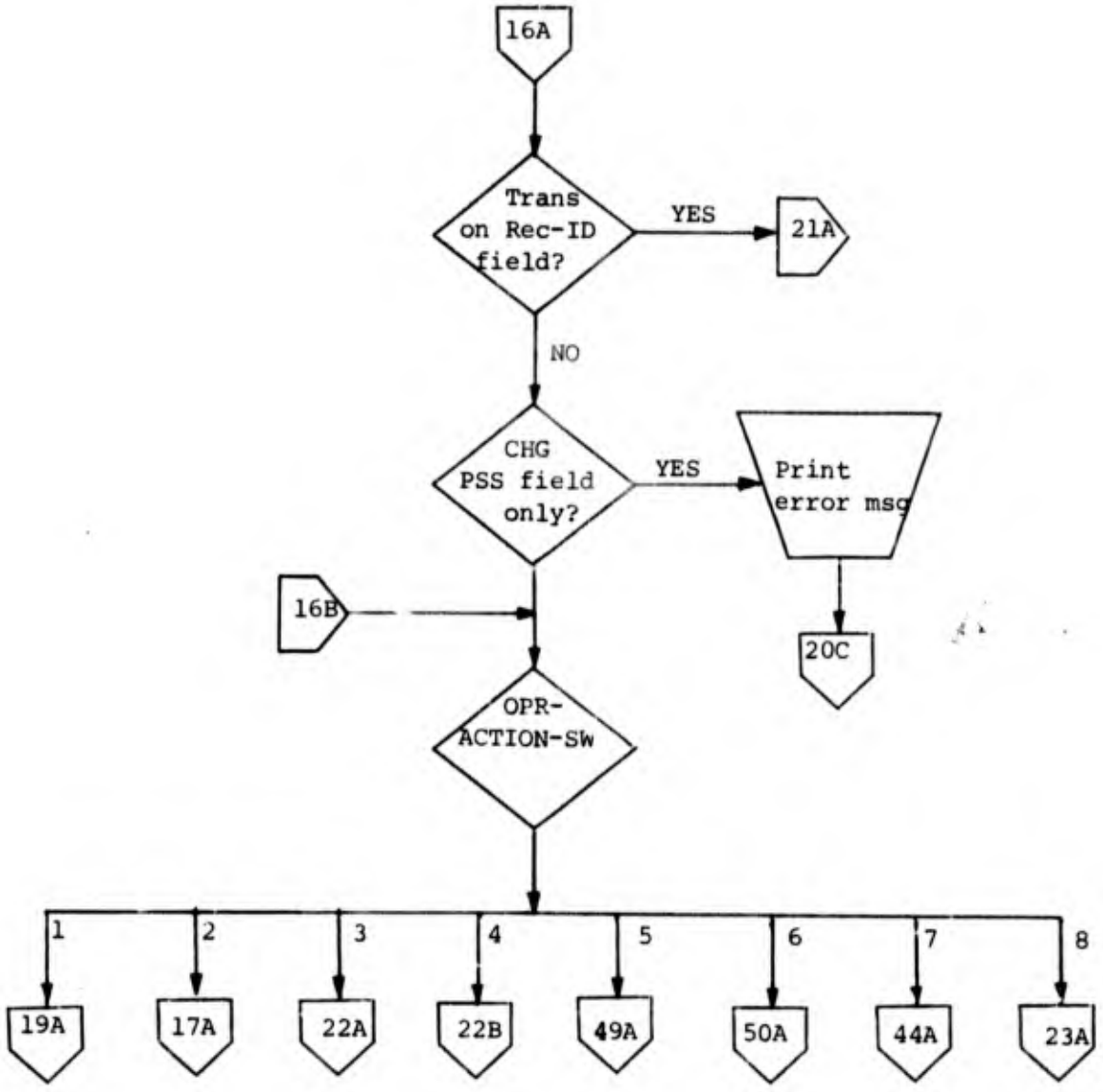

12B
CK-REC2
Record Status
generated
14B
absent
13A
present
13B

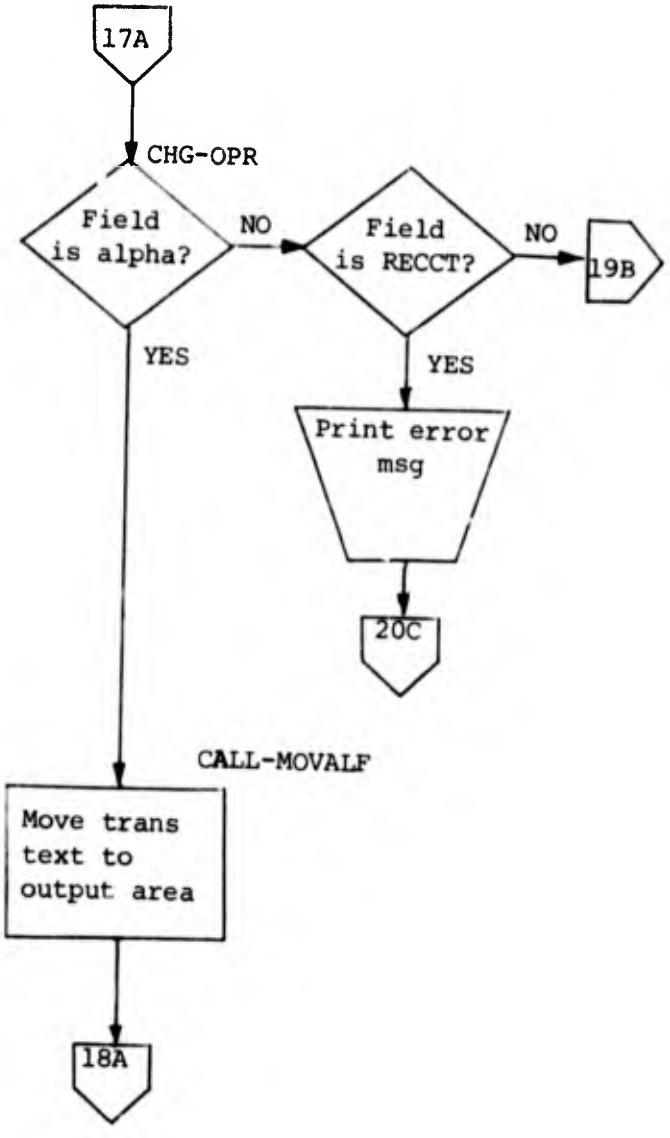

13A
CK-REC-STAT1
Set switches for provis-ional generate
14B
13B
CK-REC-STAT2
Gen trans, old record present
Print error msg
Trans is GEN-ID?
NO
2A
YES
Skipping all other trans on record
Print error msg
11C

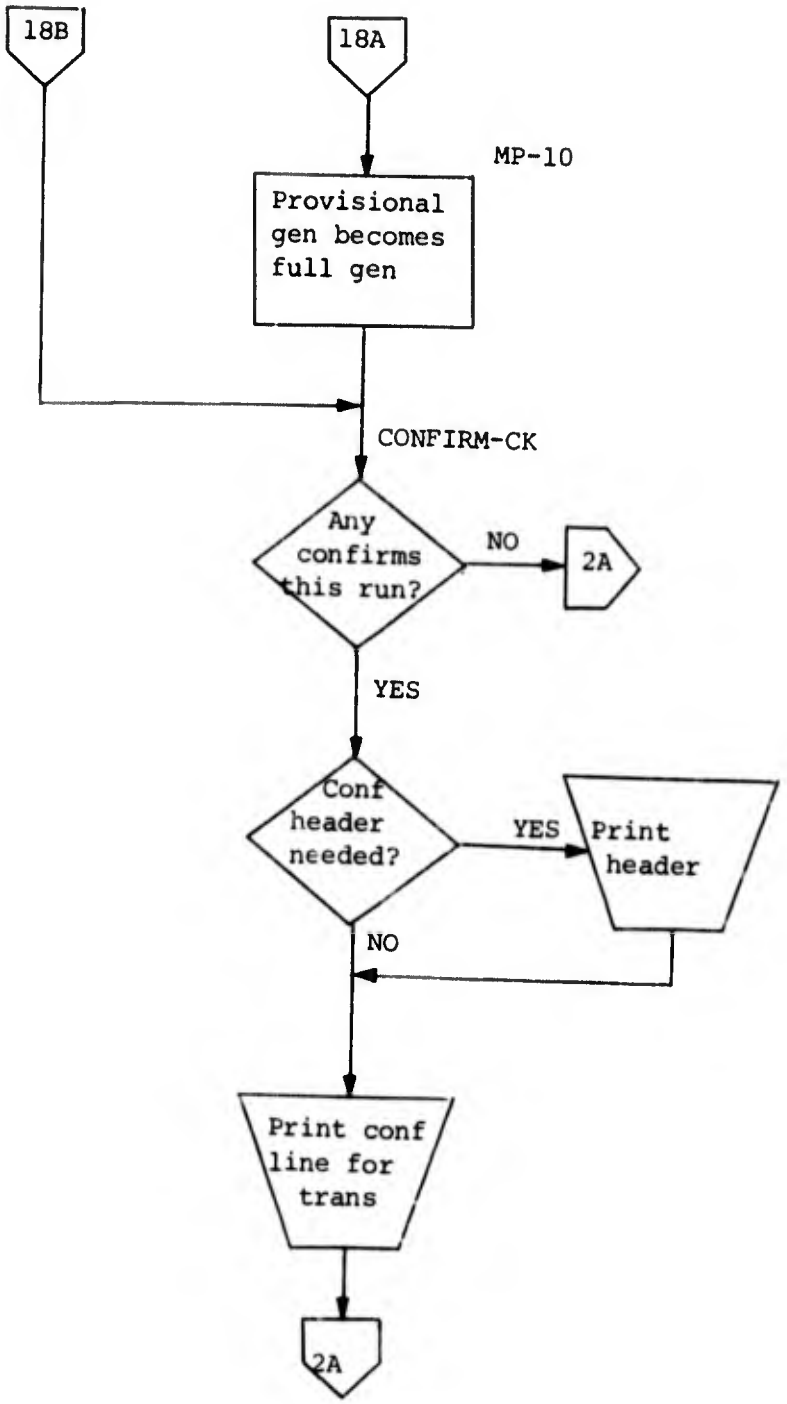

14A
CK-REC3
Old rec absent?
YES
13A
NO
14B
CK-REC3 FIXED-FIELDS
Fixed-set trans?
NO
24A
YES
16A

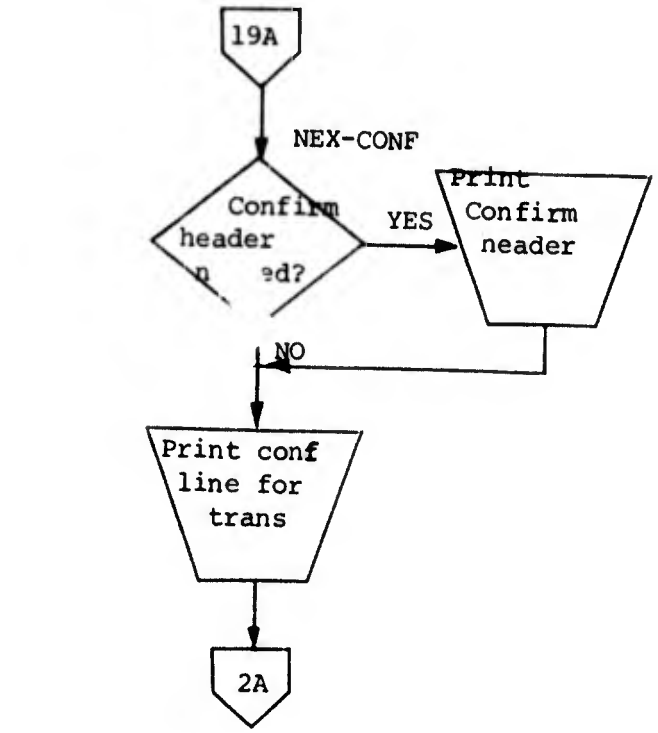

SET-UP-ERROR
Enter via PERFORM
Find set, subset for error-msg
exit

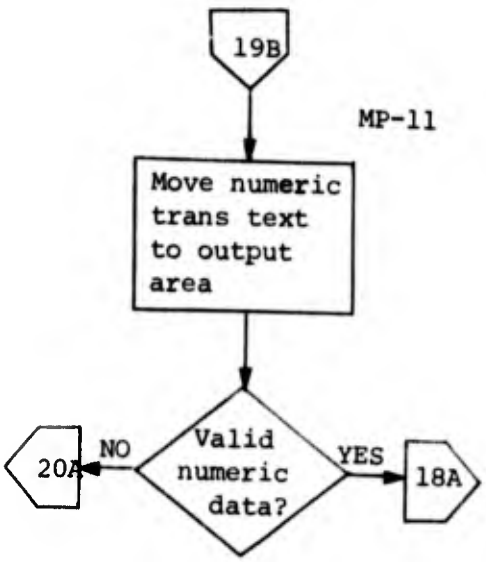

15A
PERIODIC
Locate HOP of trans-field in old & new records
16A

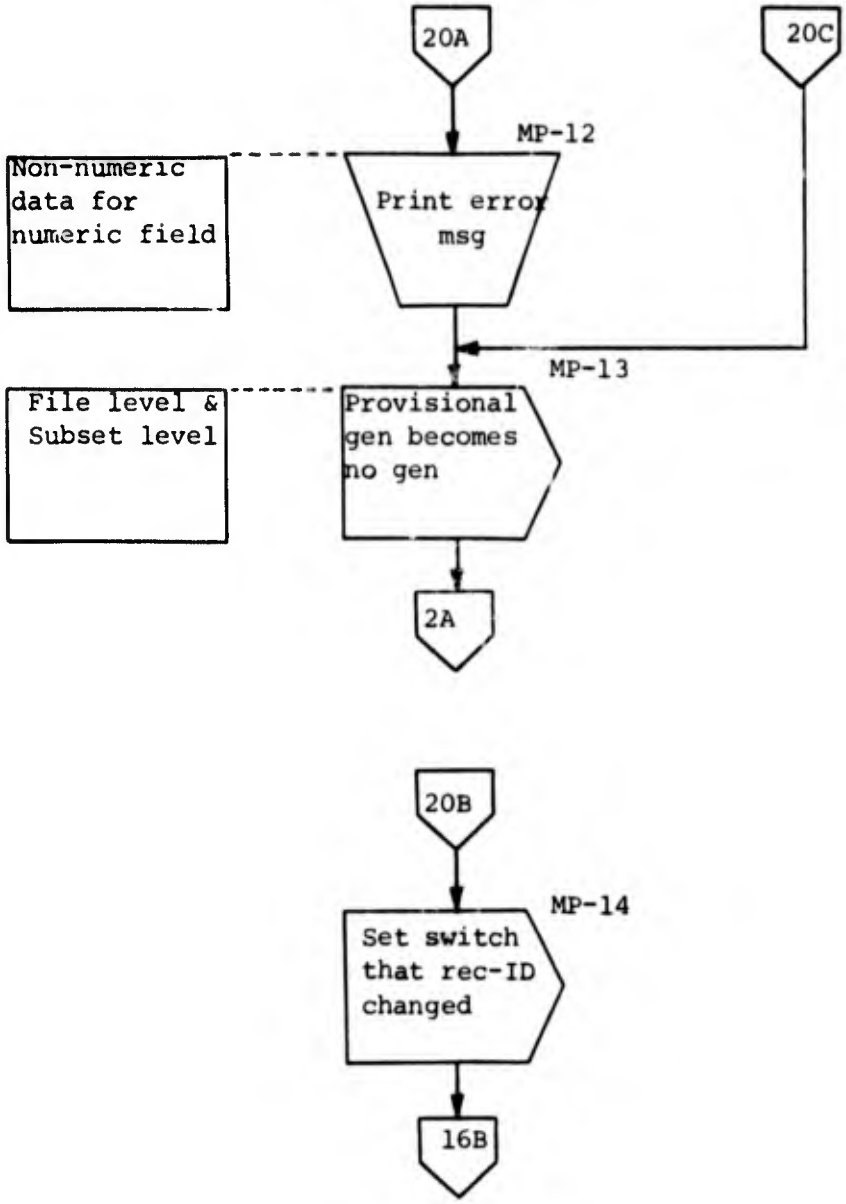

16A
Trans on Rec-ID field?
YES
21A
NO
CHG PSS field only?
YES
Print error msg
20C
16B
OPR-ACTION-SW
1
2
3
4
5
6
7
8
19A
17A
22A
22B
49A
50A
44A
23A

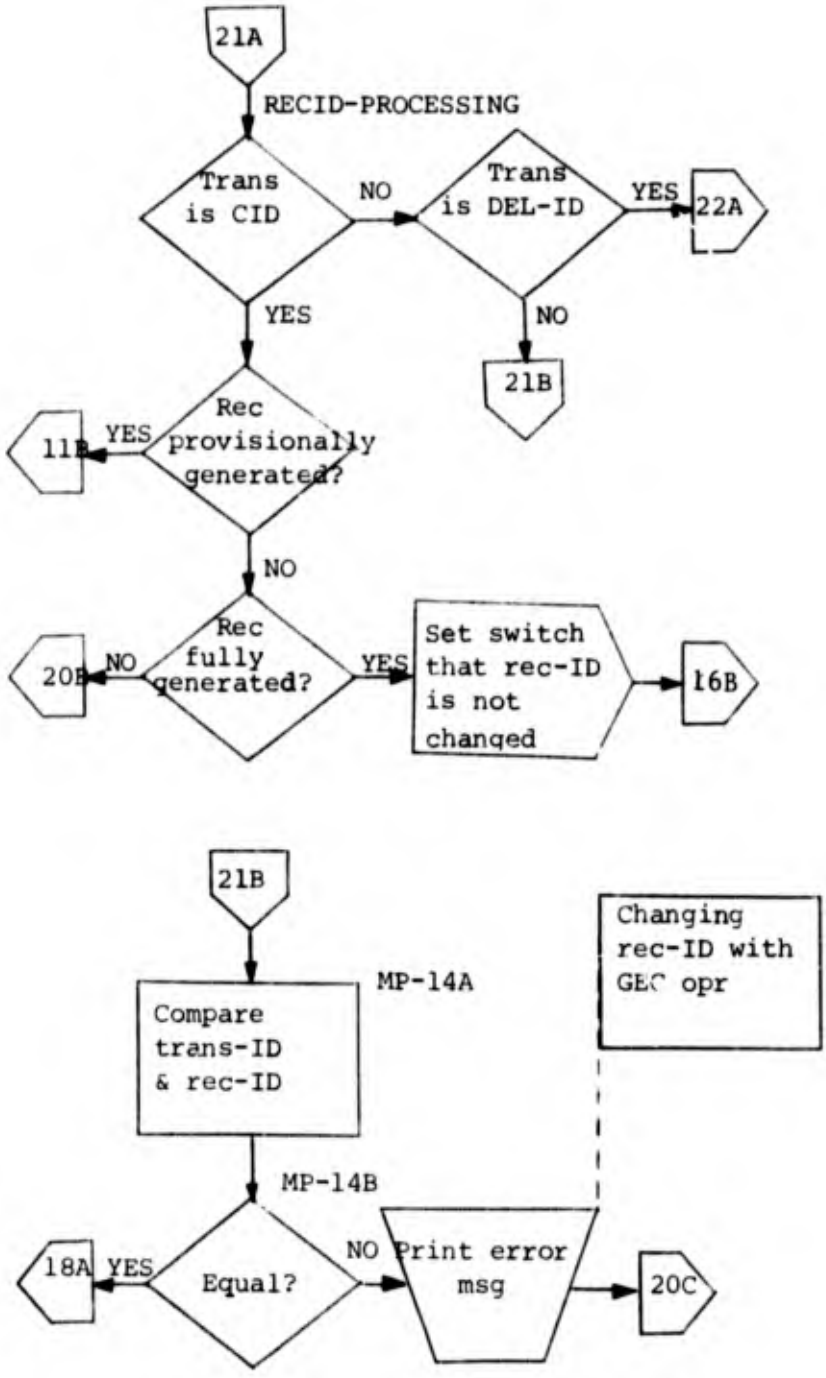

17A
CHG-OPR
Field is alpha?
NO
Field is RECCT?
NO
19B
YES
YES
Print error msg
20C
CALL-MOVALF
Move trans text to output area
18A

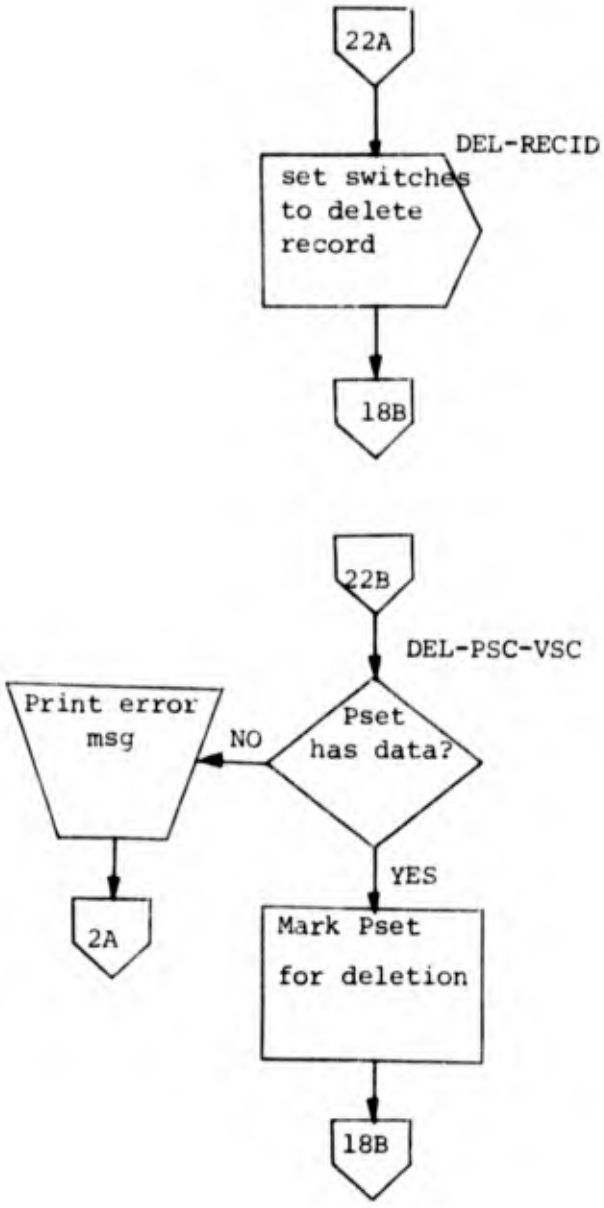

18B
18A
MP-10
Provisional gen becomes full gen
CONFIRM-CK
Any confirms this run?
NO
2A
YES
Conf header needed?
YES
Print header
NO
Print conf line for trans
2A

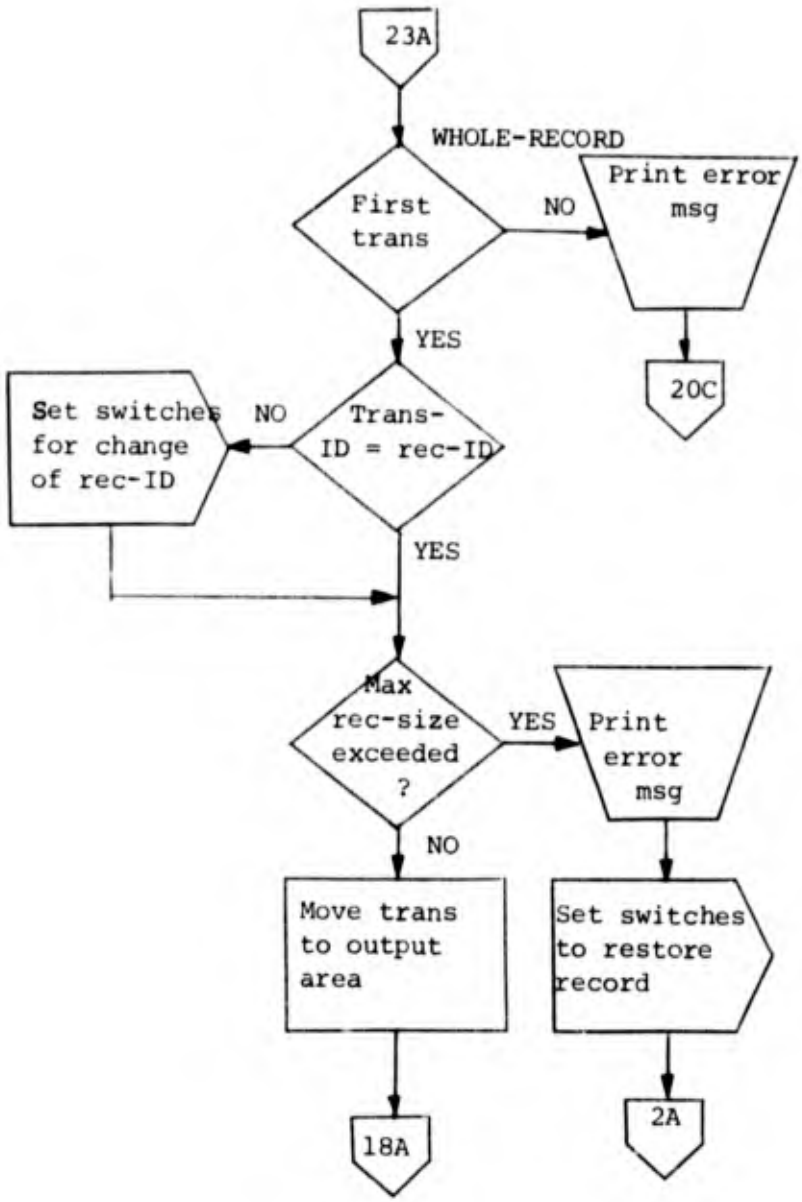

19A
NEX-CONF
Confirm header
YES
Print Confirm header
NO
Print conf line for trans
2A

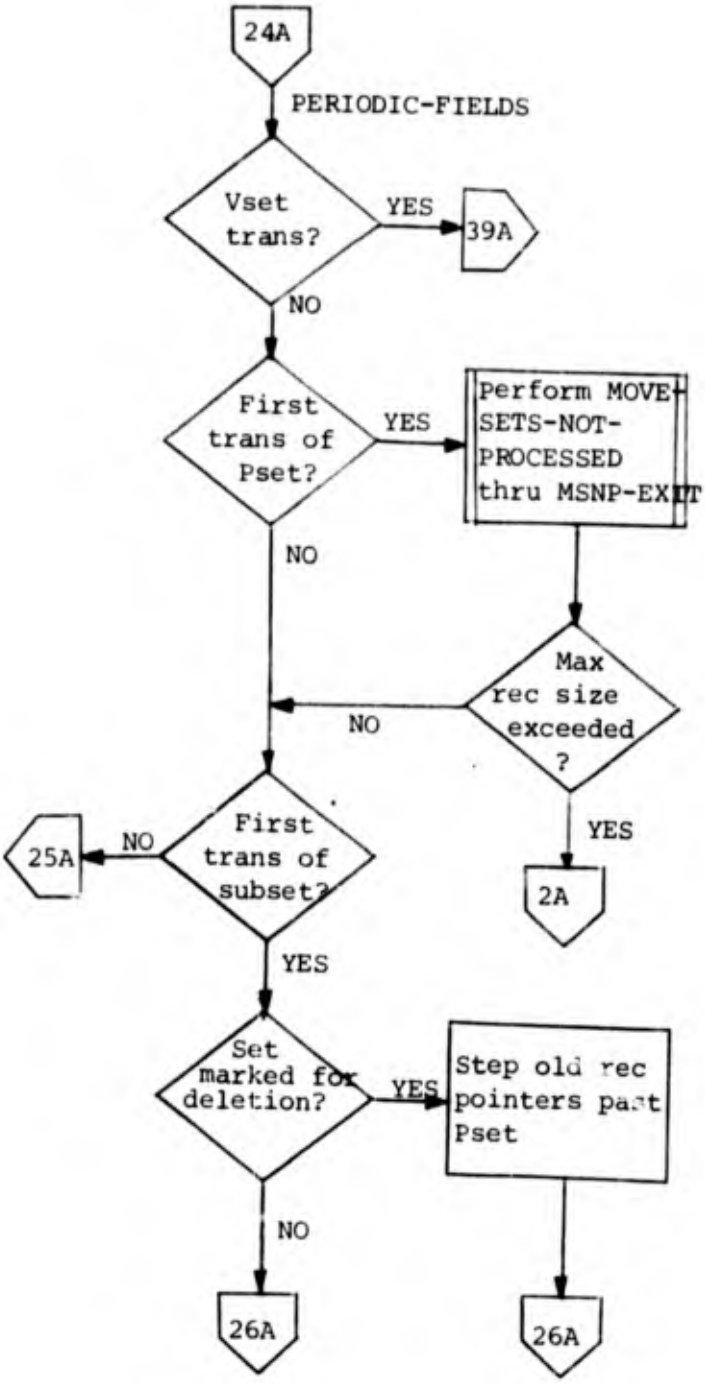

19B
MP-11
Move numeric trans text to output area
Valid numeric data?
NO
20A
YES
18A

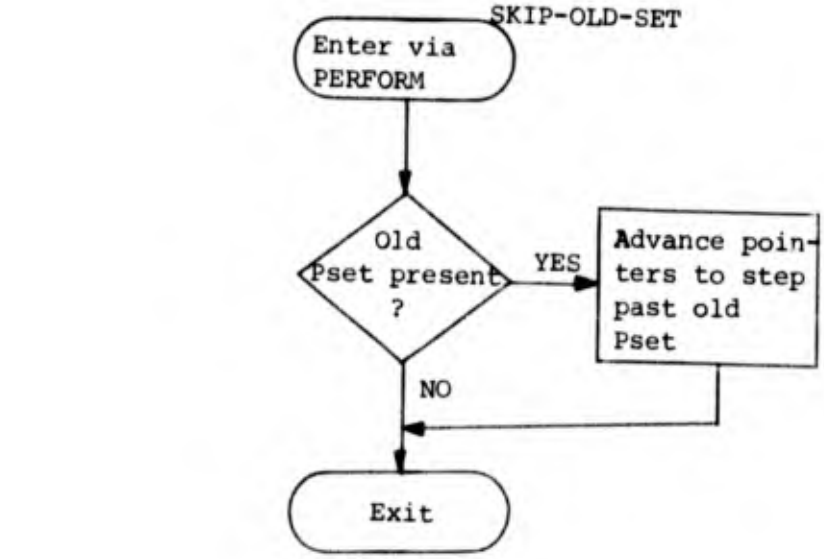

20A
20C
MP-12
Non-numeric data for numeric field
Print error msg
MP-13
File level & Subset level
Provisional gen becomes no gen
2A
20B
MP-14
Set switch that rec-ID changed
16B

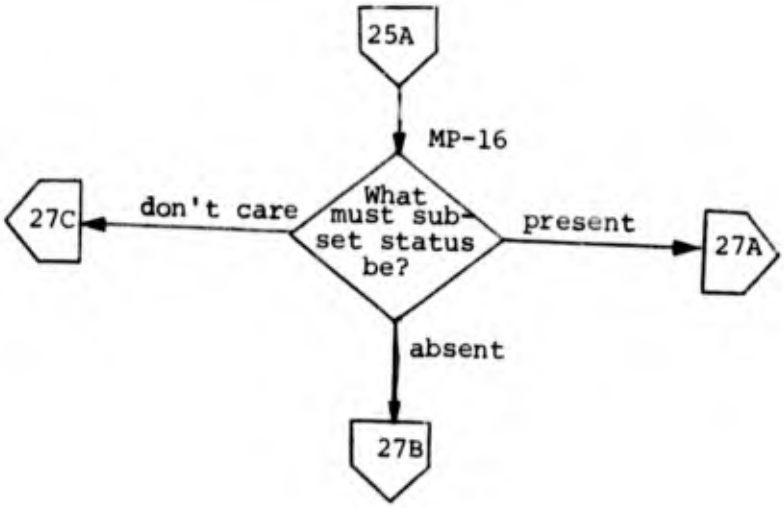

21A
RECID-PROCESSING
Trans is CID
NO
Trans is DEL-ID
YES
22A
YES
NO
21B
Rec provisionally generated?
YES
11B
NO
Rec fully generated?
NO
20B
YES
Set switch that rec-ID is not changed
16B
21B
MP-14A
Compare trans-ID & rec-ID
Changing rec-ID with GEC opr
MP-14B
Equal?
YES
18A
NO
Print error msg
20C

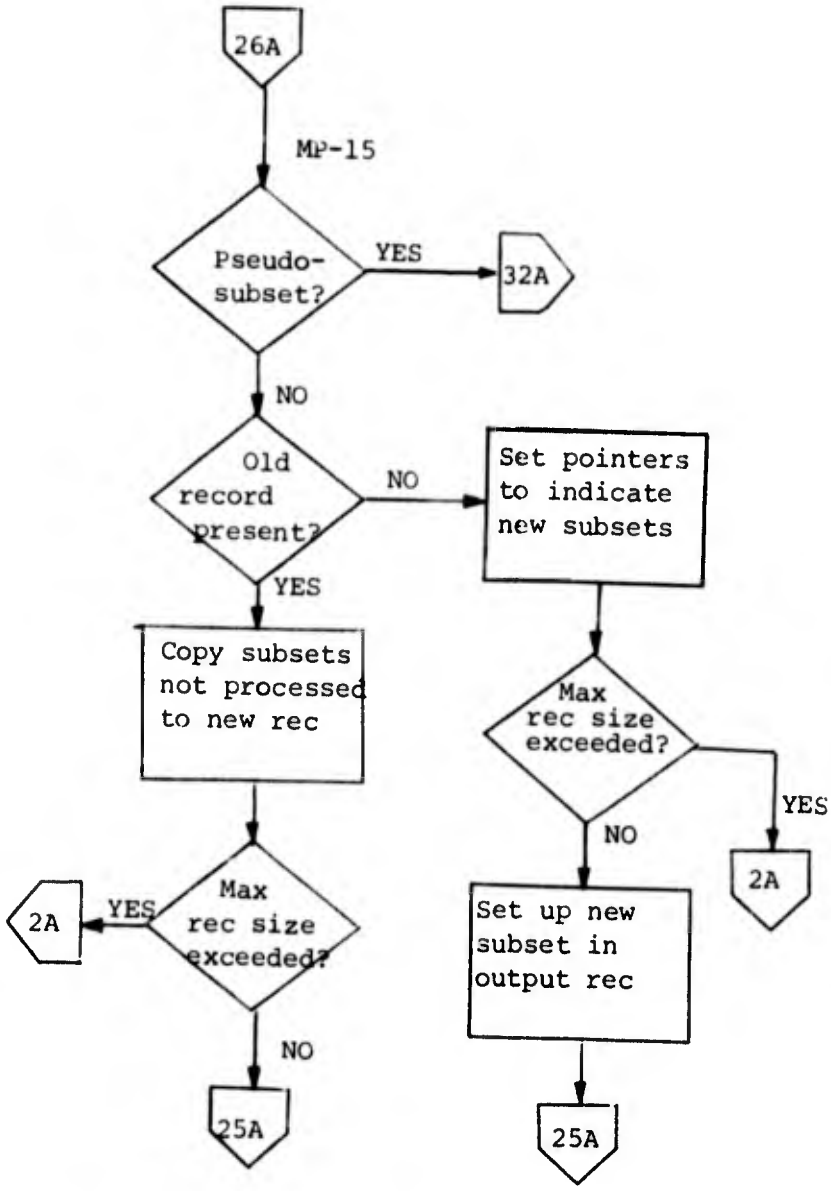

22A
DEL-RECID
set switches to delete record
18B
22B
DEL-PSC-VSC
Pset has data?
NO
Print error msg
YES
2A
Mark Pset for deletion
18B

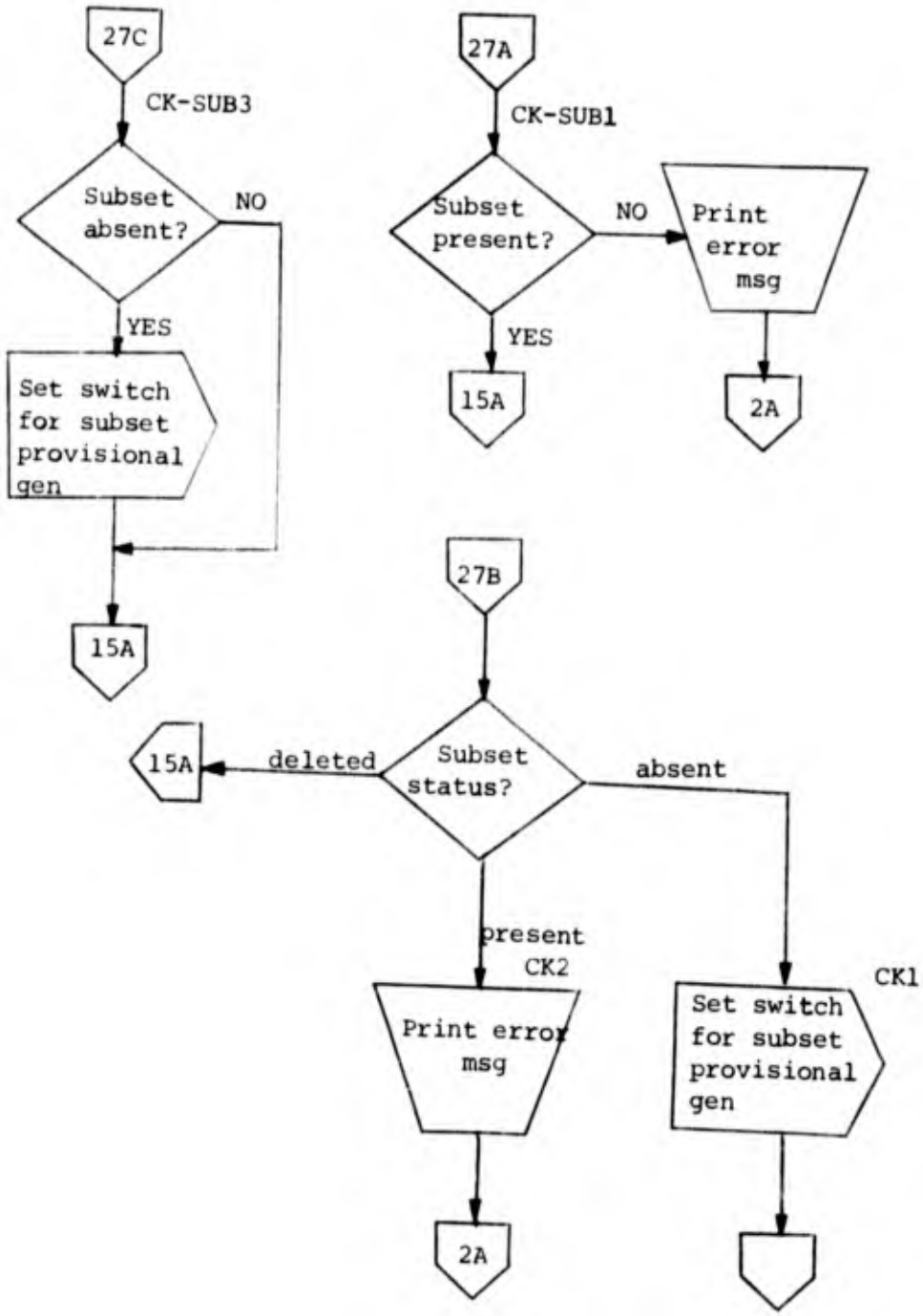

23A
WHOLE-RECORD
First trans
NO
Print error msg
20C
YES
Trans-ID = rec-ID
NO
Set switches for change of rec-ID
YES
Max rec-size exceeded ?
YES
Print error msg
NO
Move trans to output area
Set switches to restore record
18A
2A

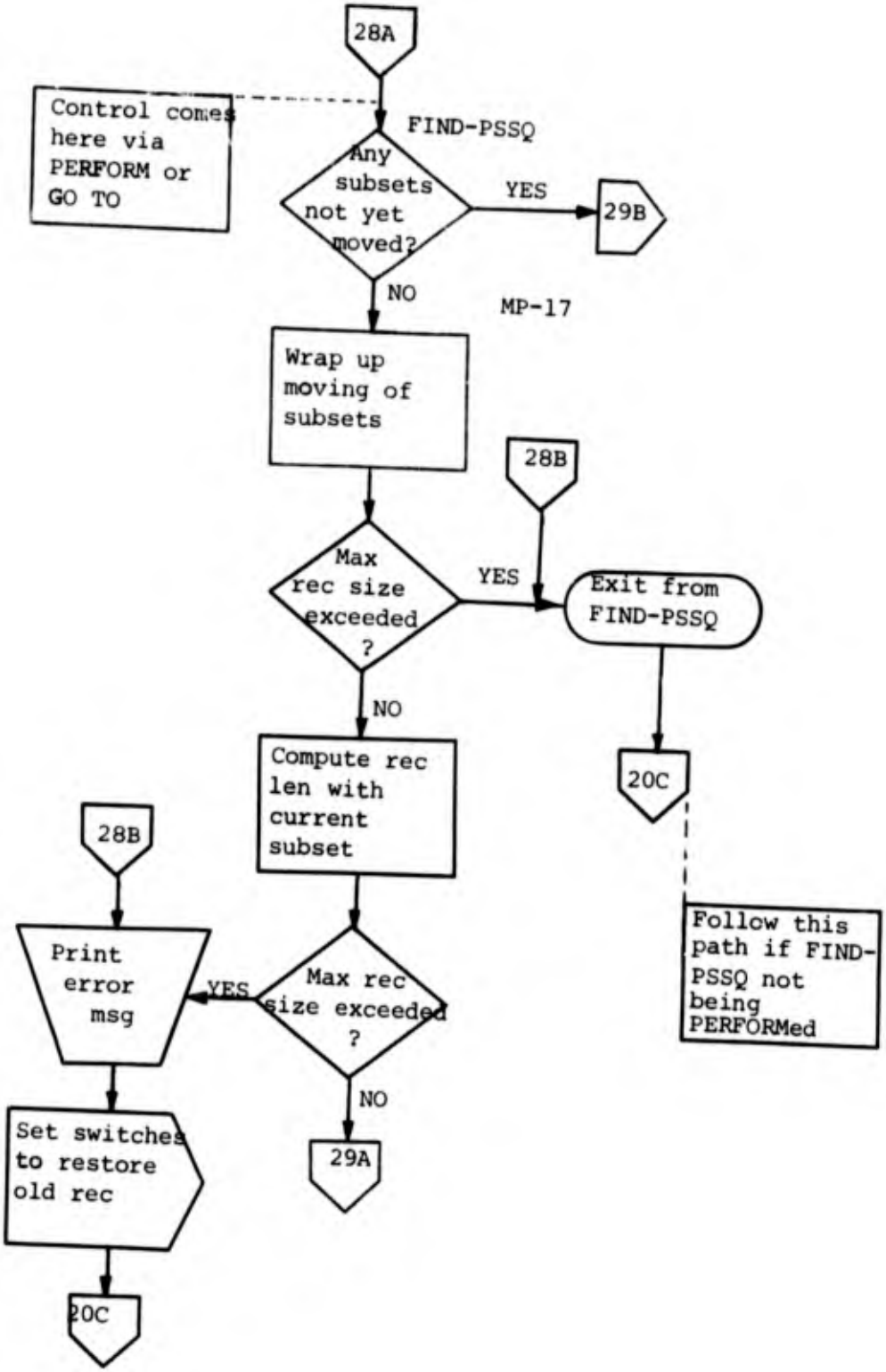

24A
PERIODIC-FIELDS
Vset trans?
YES
39A
NO
First trans of Pset?
YES
Perform MOVE-SETS-NOT-PROCESSED thru MSNP-EXIT
NO
Max rec size exceeded ?
NO
YES
2A
First trans of subset?
NO
25A
YES
Set marked for deletion?
YES
Step old rec pointers past Pset
NO
26A
26A

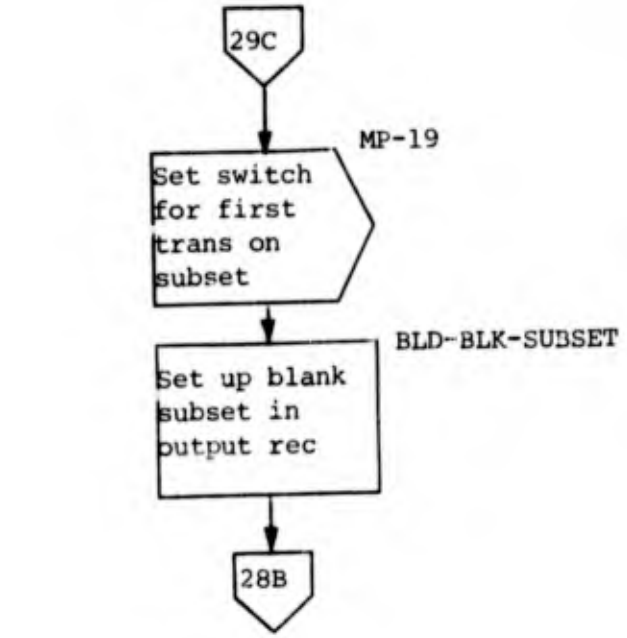

SKIP-OLD-SET
Enter via PERFORM
Old Pset present ?
YES
Advance pointers to step past old Pset
NO
Exit

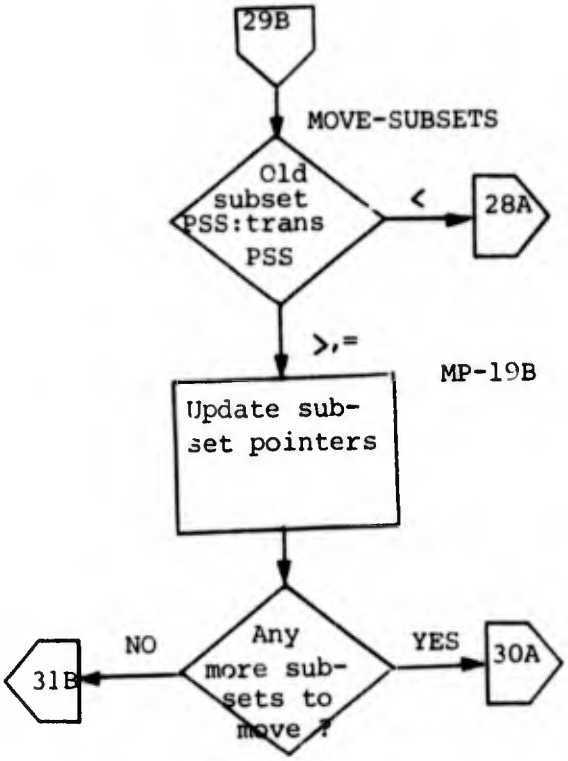

25A
MP-16
What must subset status be?
don't care
27C
present
27A
absent
27B

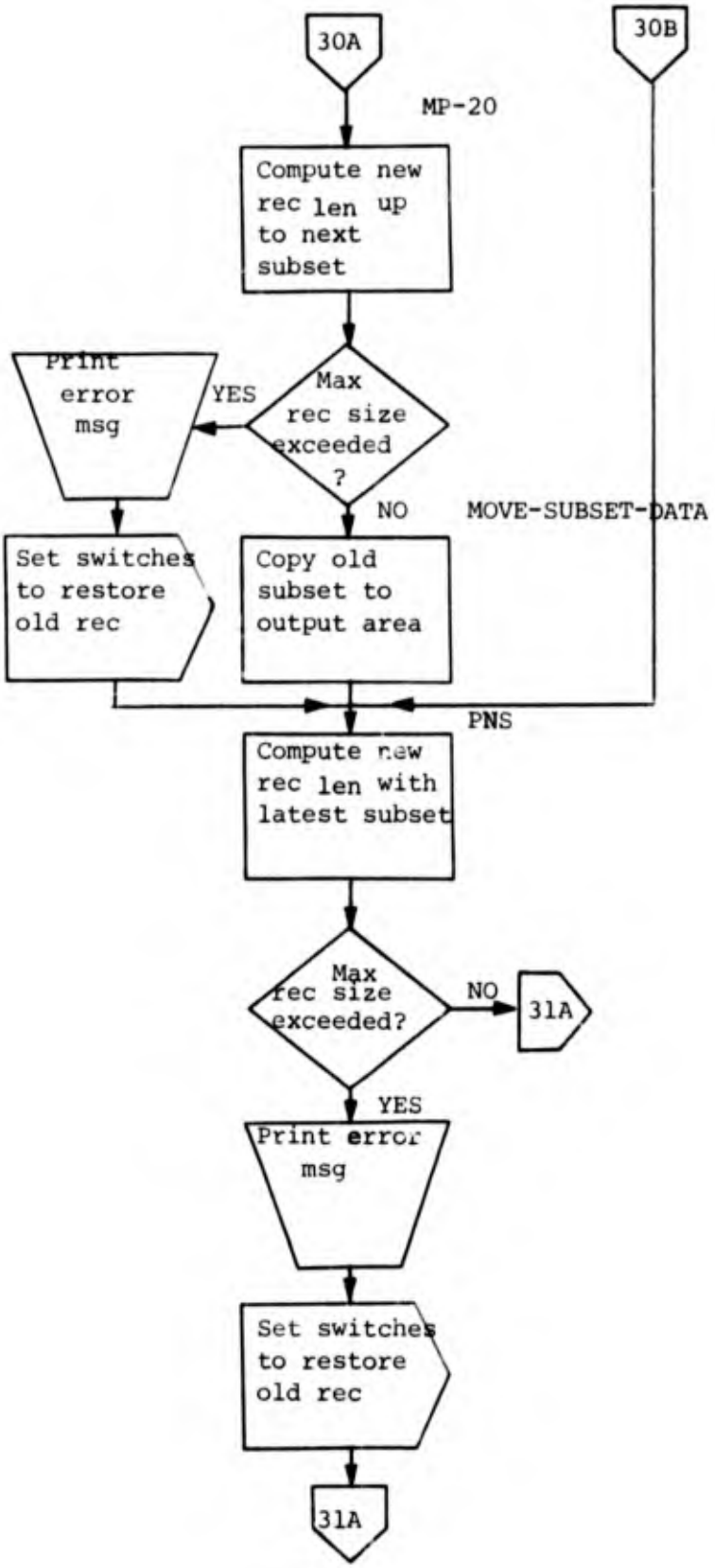

26A
MP-15
Pseudo-subset?
YES
32A
NO
Old record present?
NO
Set pointers to indicate new subsets
YES
Copy subsets not processed to new rec
Max rec size exceeded?
YES
NO
2A
Max rec size exceeded?
YES
2A
Set up new subset in output rec
NO
25A
25A

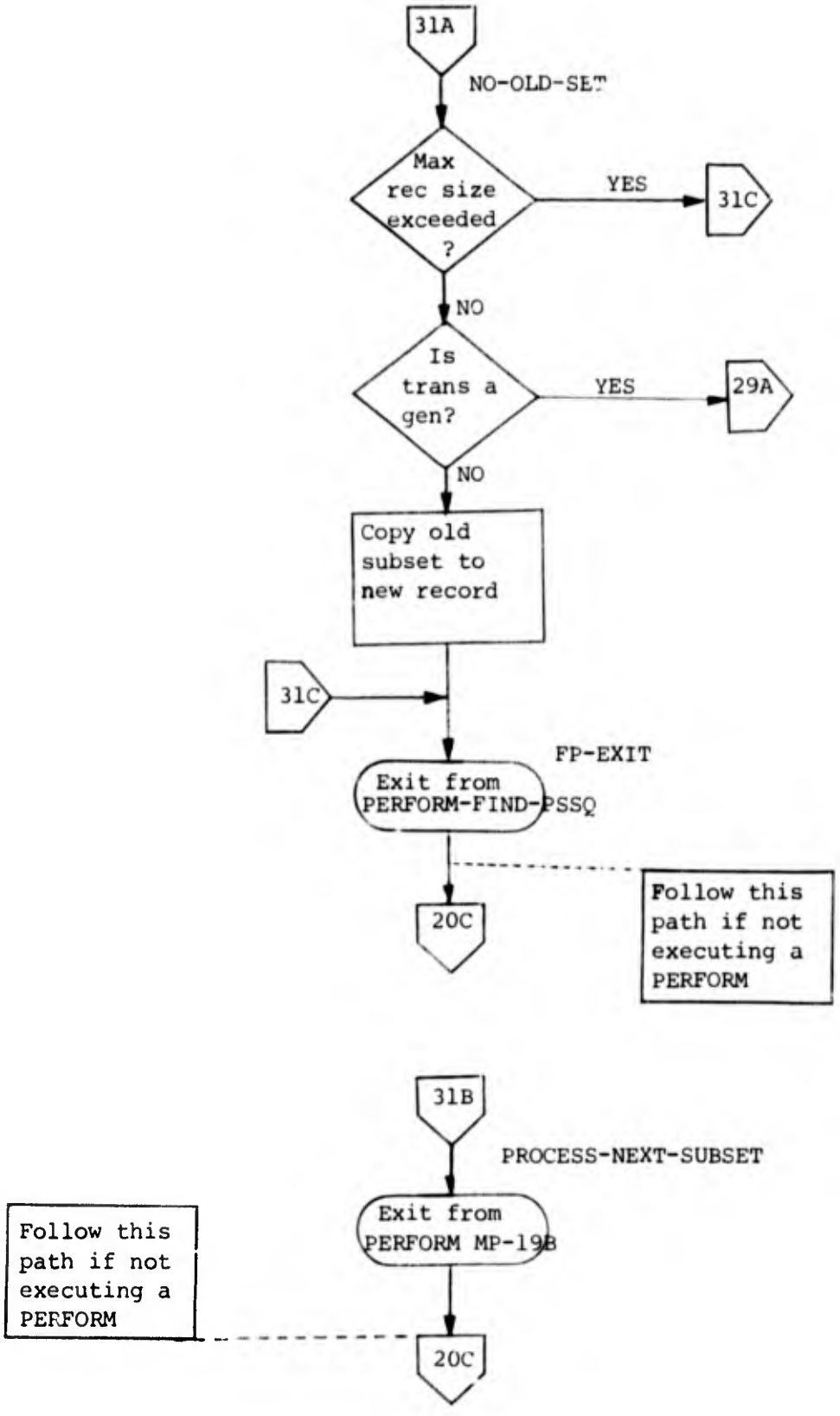

27C
CK-SUB3
Subset absent?
NO
YES
Set switch for subset provisional gen
15A
27A
CK-SUB1
Subset present?
NO
Print error msg
YES
15A
2A
27B
Subset status?
deleted
15A
absent
present
CK2
Print error msg
2A
CK1
Set switch for subset provisional gen

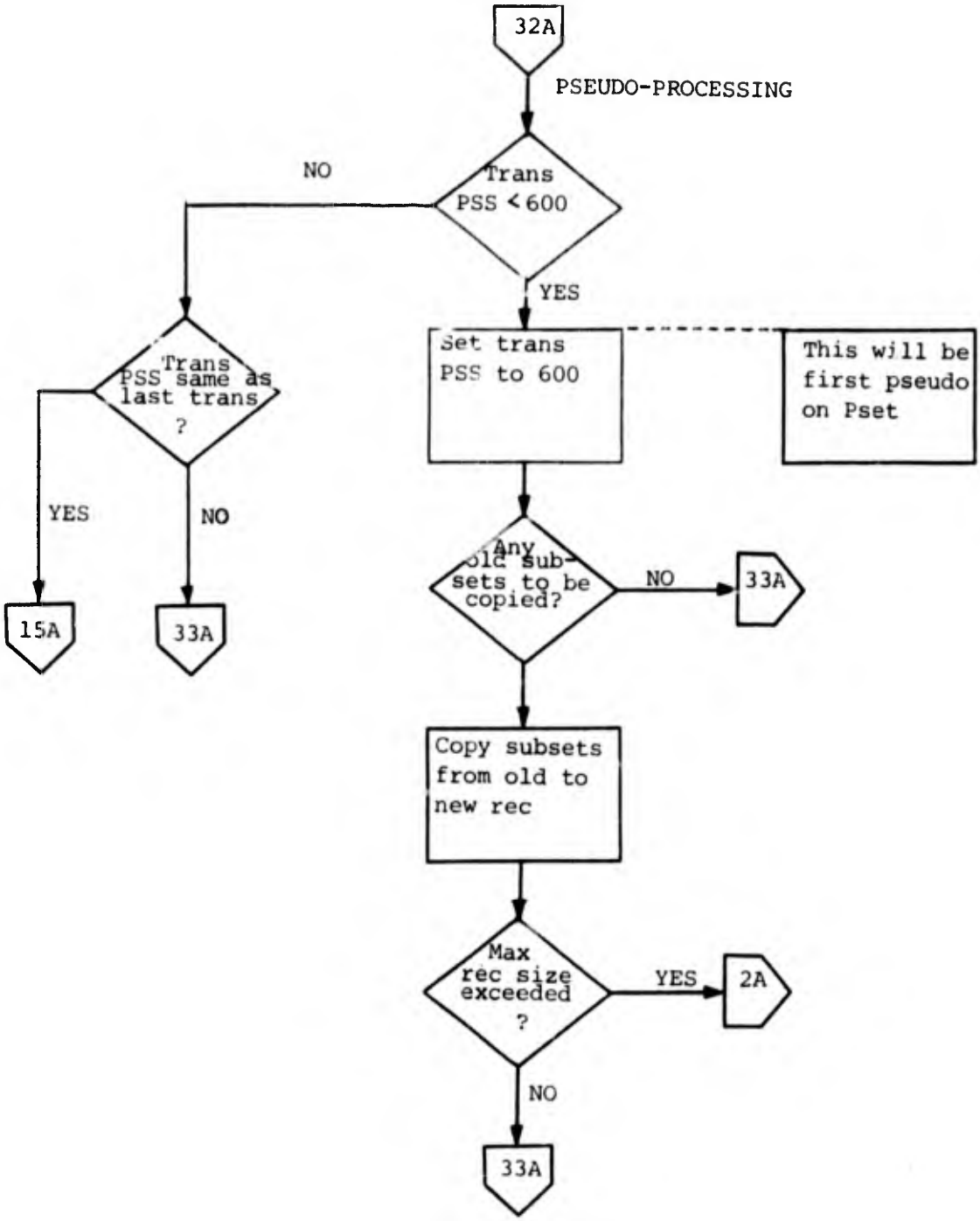

28A
Control comes here via PERFORM or GO TO
FIND-PSSQ
Any subsets not yet moved?
YES
29B
NO
MP-17
Wrap up moving of subsets
28B
Max rec size exceeded ?
YES
Exit from FIND-PSSQ
NO
Compute rec len with current subset
20C
28B
Follow this path if FIND-PSSQ not being PERFORMed
Print error msg
YES
Max rec size exceeded ?
NO
Set switches to restore old rec
29A
20C

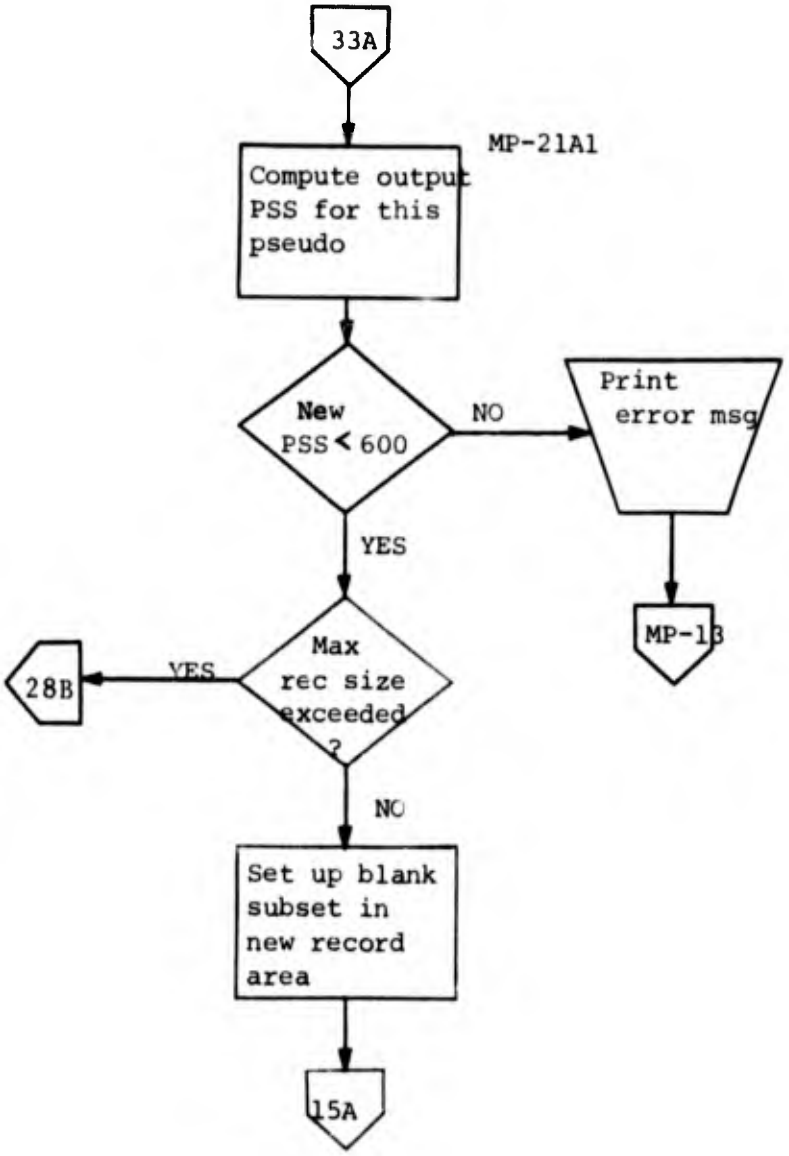

29C
MP-19
Set switch for first trans on subset
BLD-BLK-SUBSET
Set up blank subset in output rec
28B

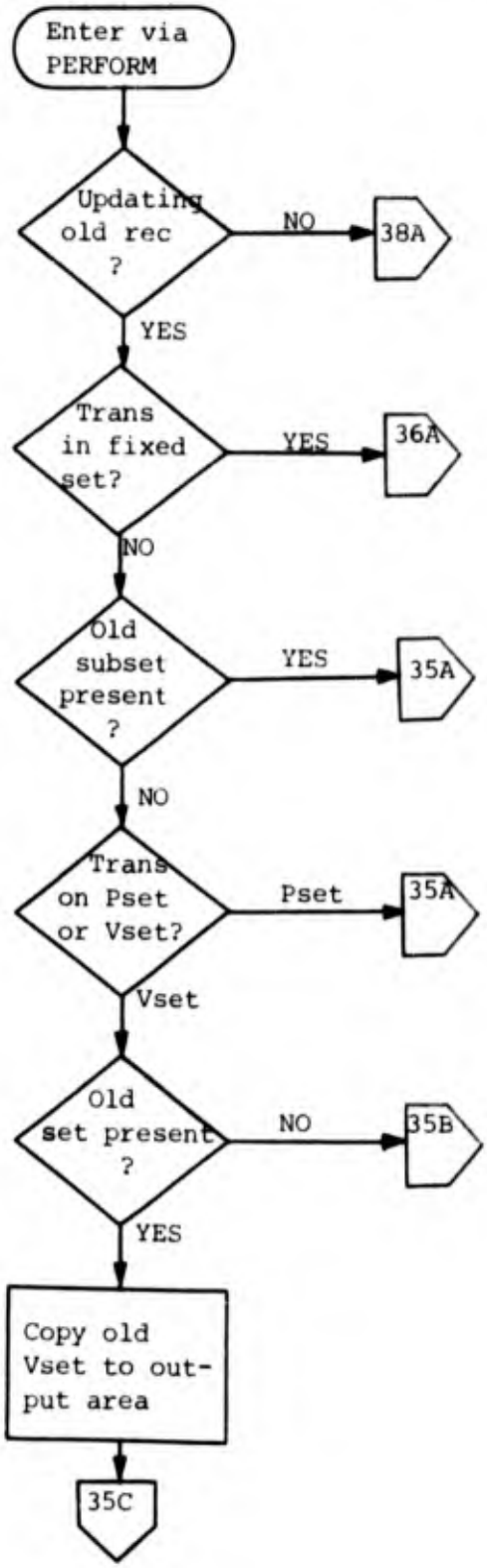

29B
MOVE-SUBSETS
Old subset PSS:trans PSS
<
28A
>,=
MP-19B
Update sub-set pointers
Any more sub-sets to move ?
NO
31B
YES
30A

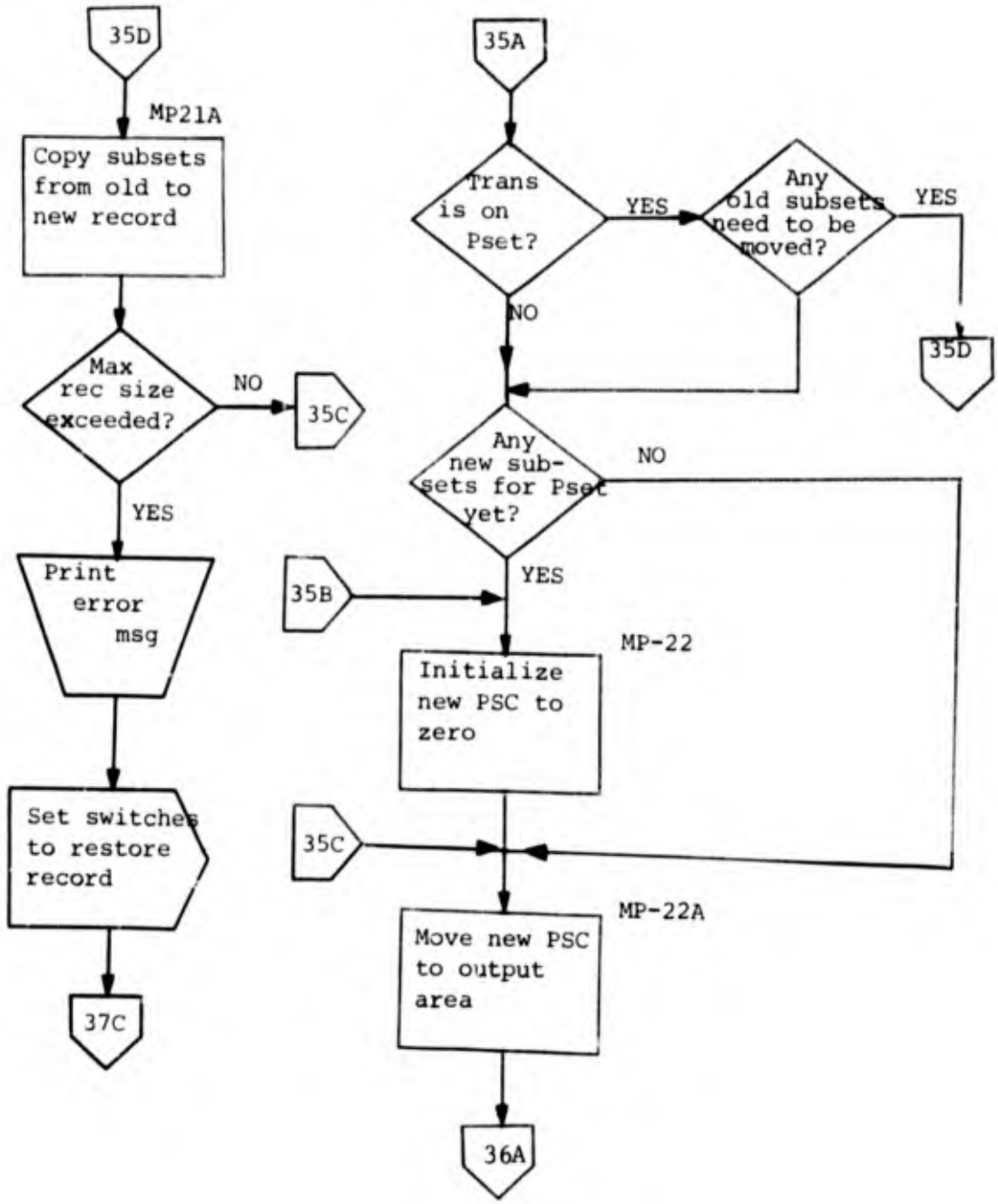

30A
30B
MP-20
Compute new rec len up to next subset
Print error msg
YES
Max rec size exceeded ?
NO
MOVE-SUBSET-DATA
Set switches to restore old rec
Copy old subset to output area
PNS
Compute new rec len with latest subset
Max rec size exceeded?
NO
31A
YES
Print error msg
Set switches to restore old rec
31A

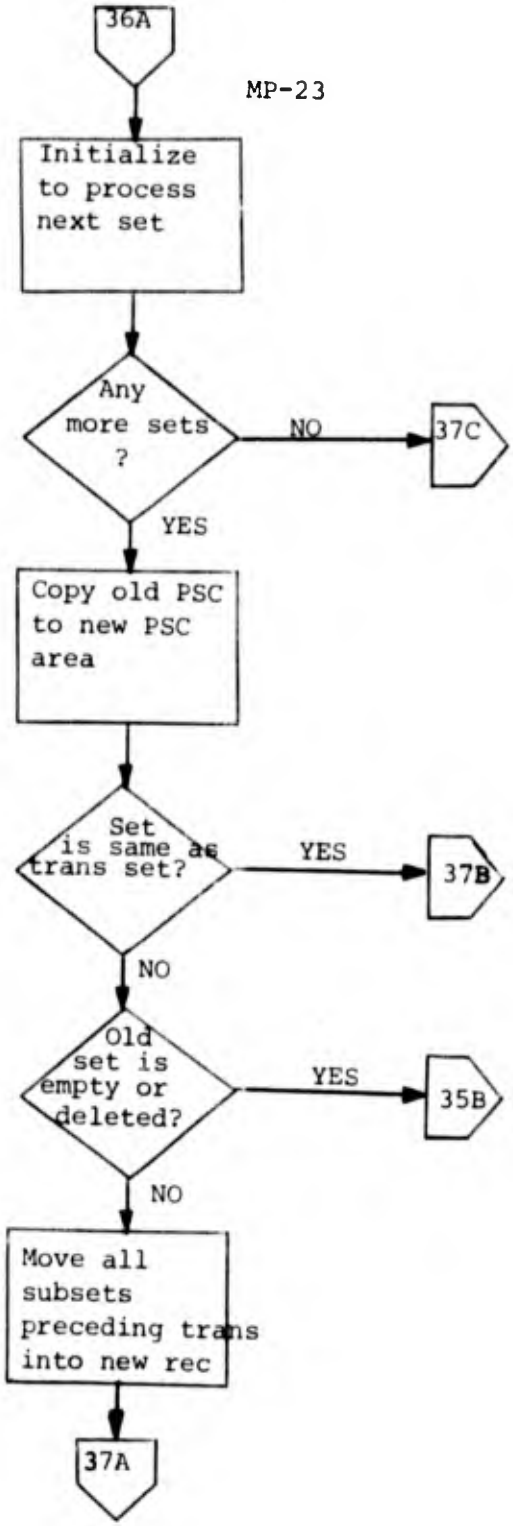

31A
NO-OLD-SET
Max rec size exceeded ?
YES
31C
NO
Is trans a gen?
YES
29A
NO
Copy old subset to new record
31C
FP-EXIT
Exit from PERFORM-FIND-PSSQ
20C
Follow this path if not executing a PERFORM
31B
PROCESS-NEXT-SUBSET
Exit from PERFORM MP-19B
Follow this path if not executing a PERFORM
20C

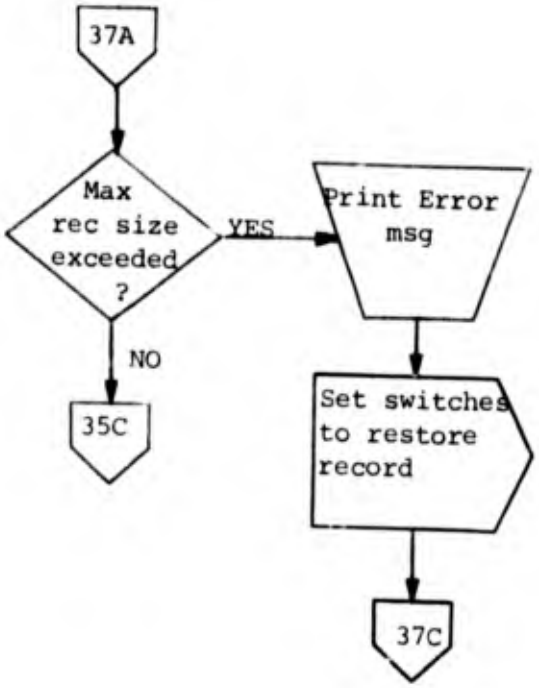

32A
PSEUDO-PROCESSING
Trans PSS < 600
NO
YES
Trans PSS same as last trans ?
YES
NO
15A
33A
Set trans PSS to 600
This will be first pseudo on Pset
Any old sub-sets to be copied?
NO
33A
Copy subsets from old to new rec
Max rec size exceeded ?
YES
2A
NO
33A

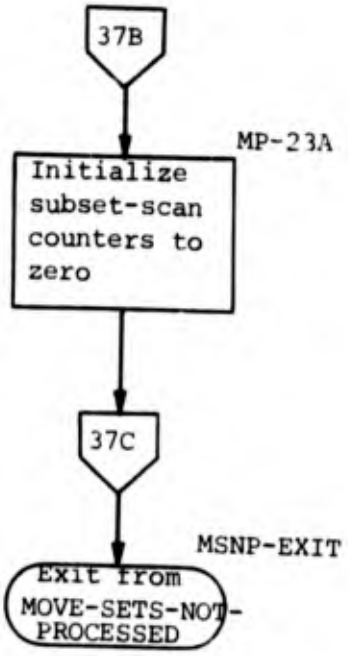

33A
MP-21A1
Compute output PSS for this pseudo
New PSS < 600
NO
Print error msg
MP-13
YES
Max rec size exceeded ?
YES
28B
NO
Set up blank subset in new record area
15A

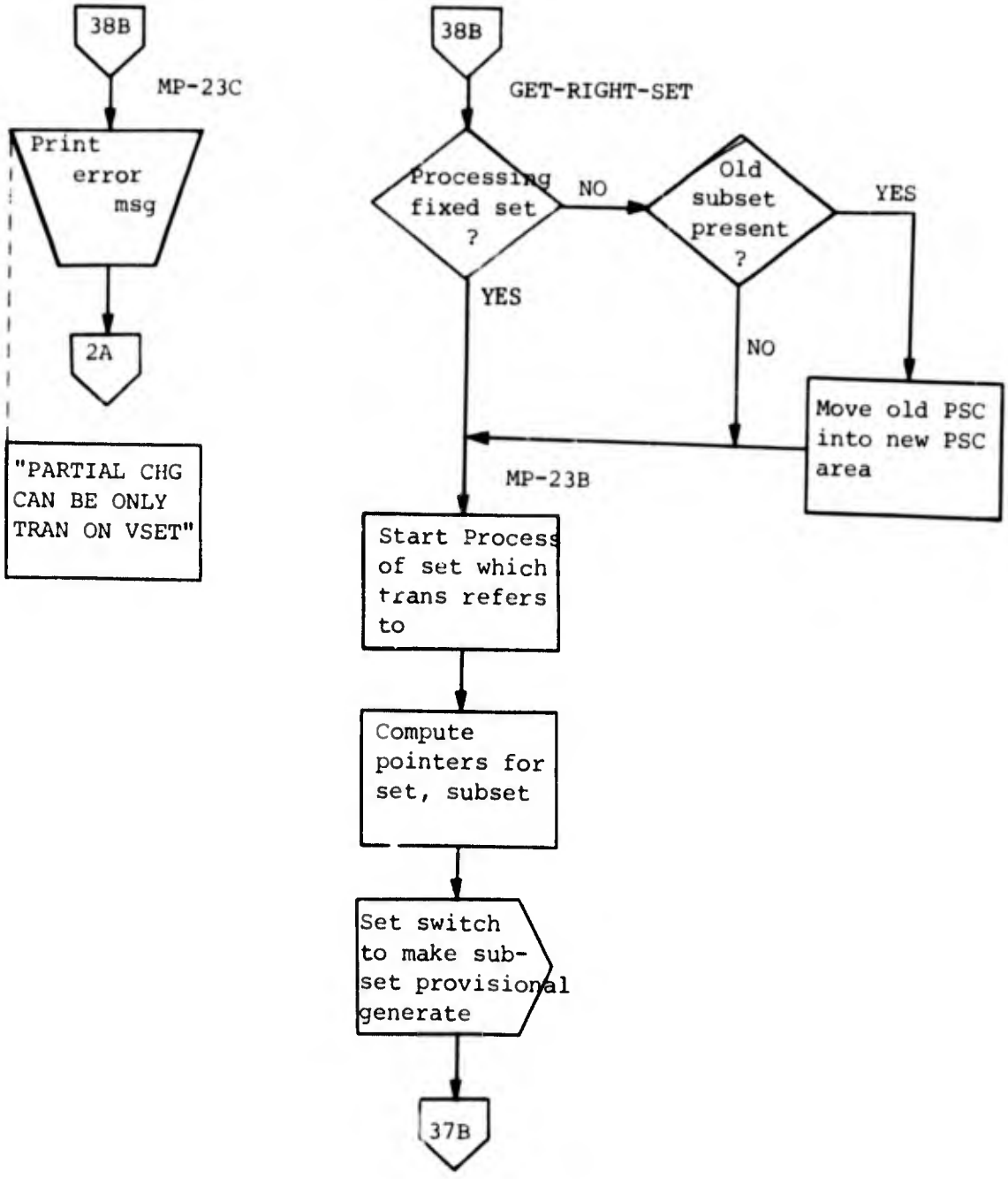

Enter via PERFORM
Updating old rec ?
NO
38A
YES
Trans in fixed set?
YES
36A
NO
Old subset present ?
YES
35A
NO
Trans on Pset or Vset?
Pset
35A
Vset
Old set present ?
NO
35B
YES
Copy old Vset to output area
35C

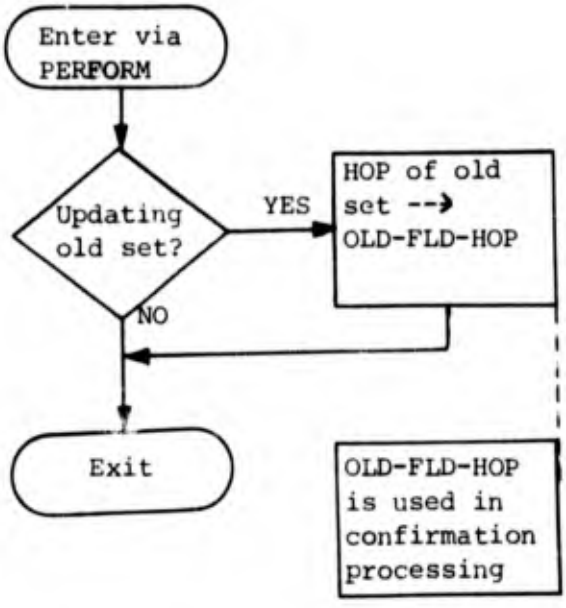

35D
MP21A
Copy subsets from old to new record
Max rec size exceeded?
NO
35C
YES
Print error msg
Set switches to restore record
37C
35A
Trans is on Pset?
YES
Any old subsets need to be moved?
YES
35D
NO
Any new sub-sets for Pset yet?
NO
YES
35B
MP-22
Initialize new PSC to zero
35C
MP-22A
Move new PSC to output area
36A

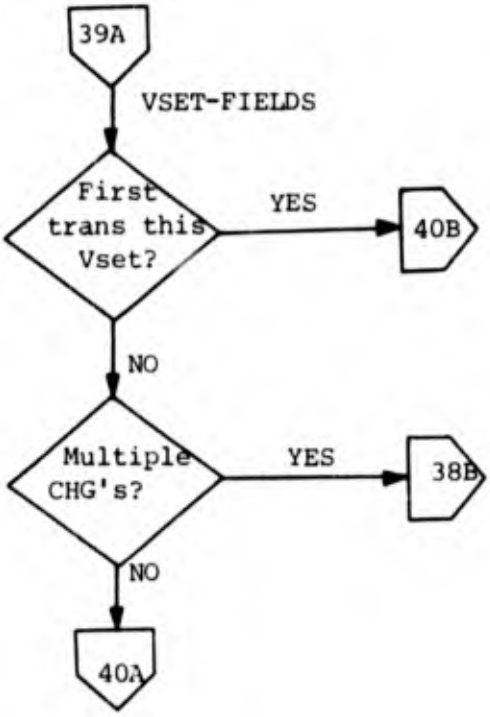

36A
MP-23
Initialize to process next set
Any more sets ?
NO
37C
YES
Copy old PSC to new PSC area
Set is same as trans set?
YES
37B
NO
Old set is empty or deleted?
YES
35B
NO
Move all subsets preceding trans into new rec
37A

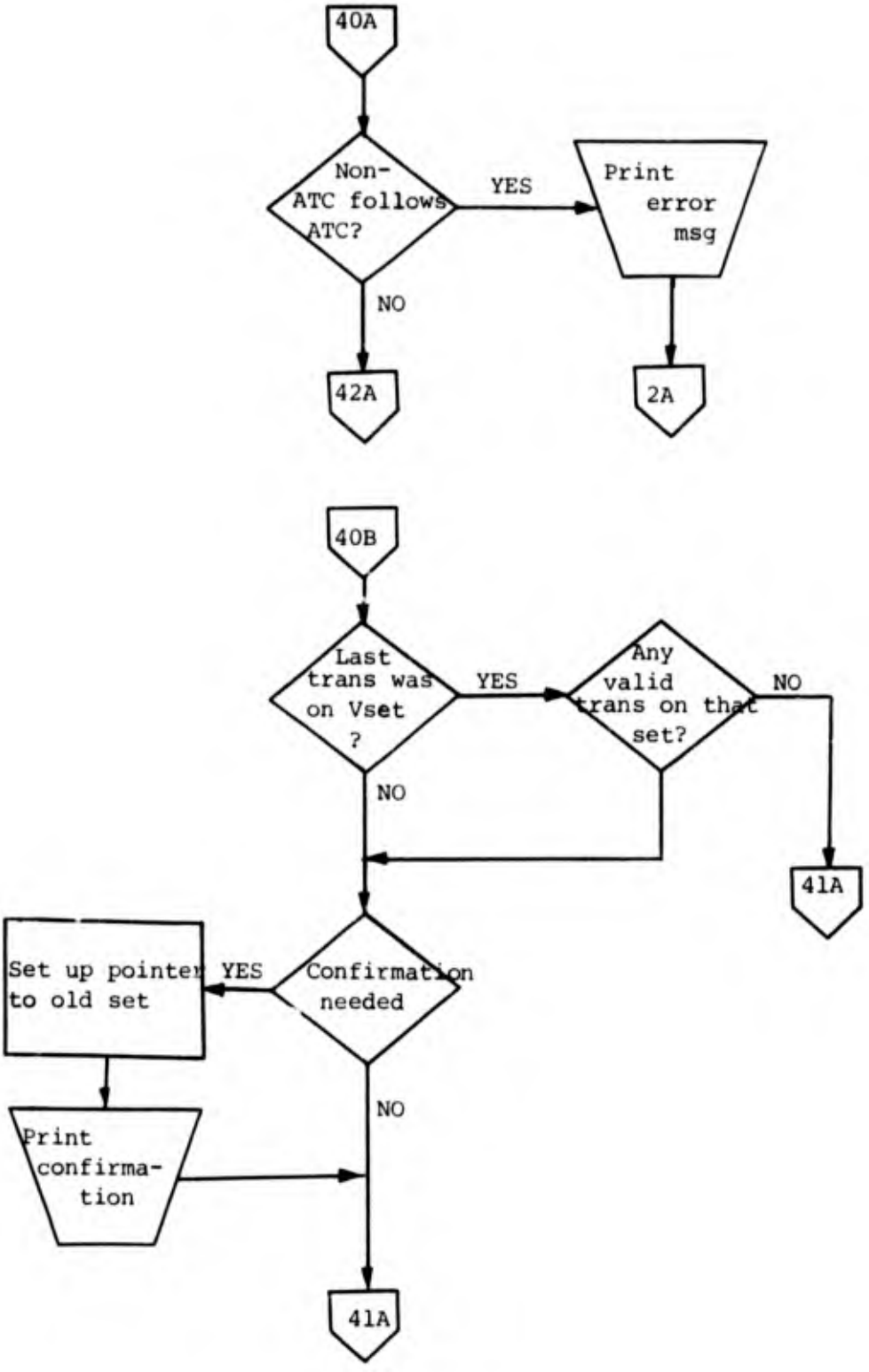

37A
Max rec size exceeded ?
YES
Print Error msg
NO
35C
Set switches to restore record
37C

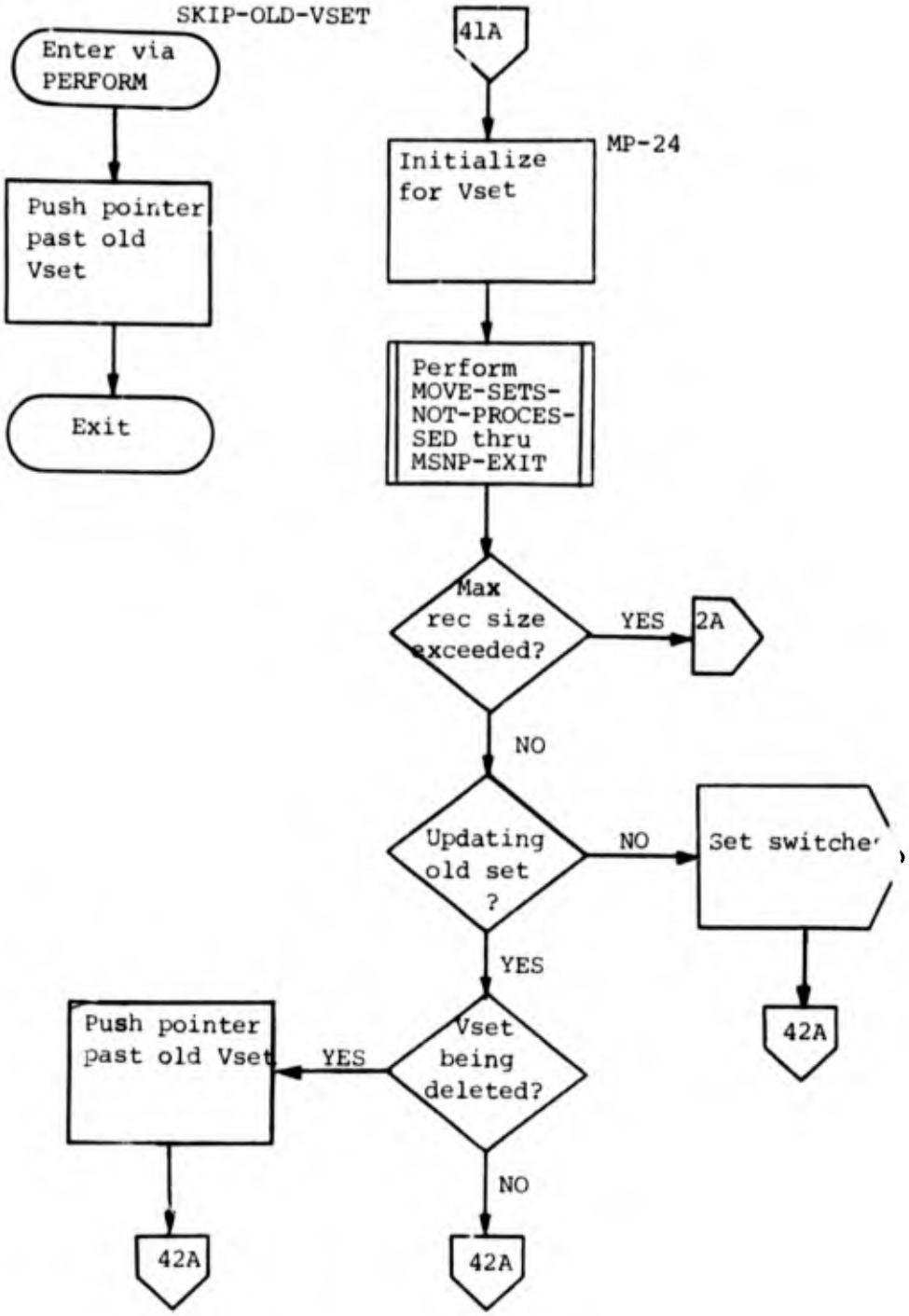

37B
MP-23A
Initialize subset-scan counters to zero
37C
MSNP-EXIT
Exit from MOVE-SETS-NOT-PROCESSED

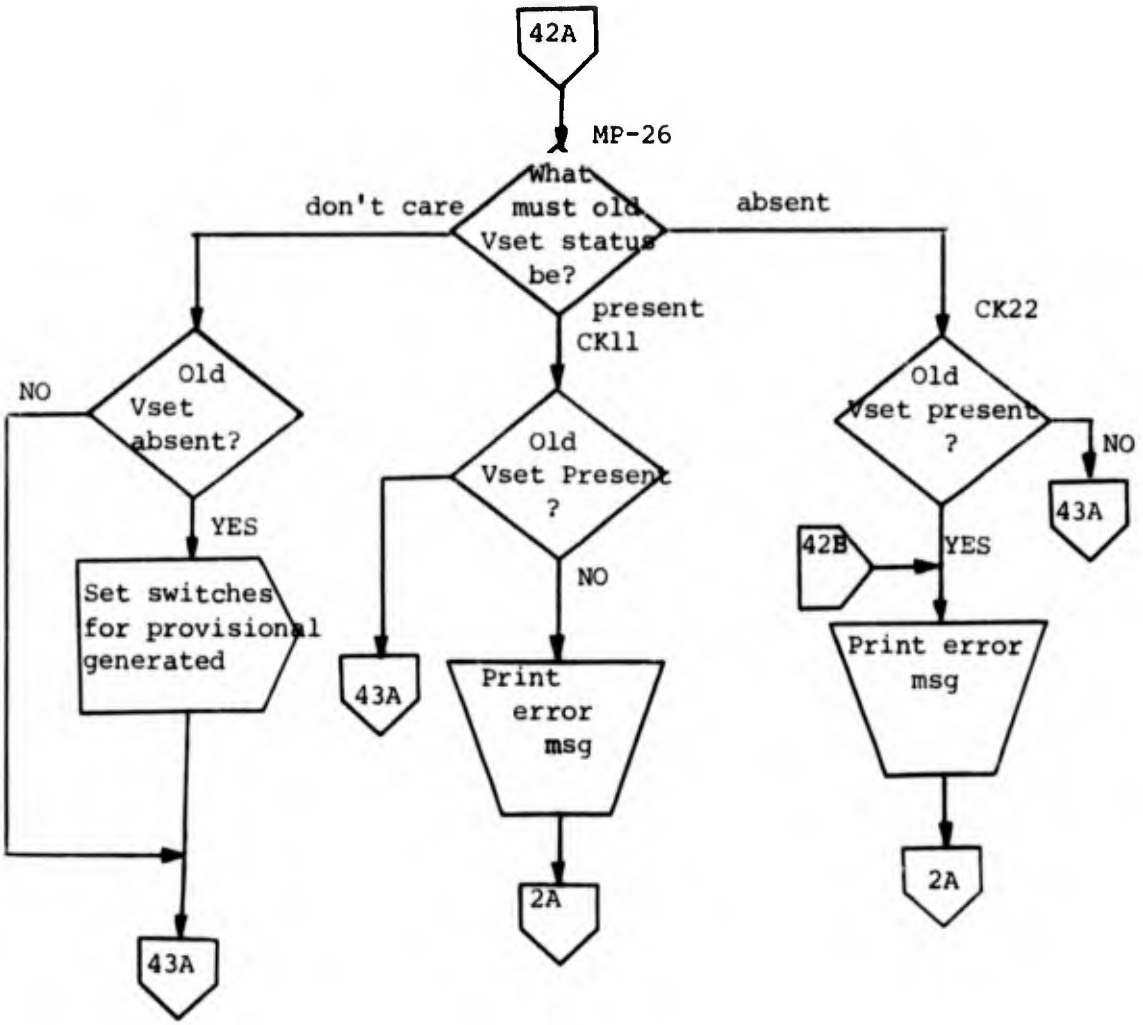

38B
MP-23C
Print error msg
2A
"PARTIAL CHG CAN BE ONLY TRAN ON VSET"
38B
GET-RIGHT-SET
Processing fixed set ?
NO
Old subset present ?
YES
YES
NO
Move old PSC into new PSC area
MP-23B
Start Process of set which trans refers to
Compute pointers for set, subset
Set switch to make sub-set provisional generate
37B

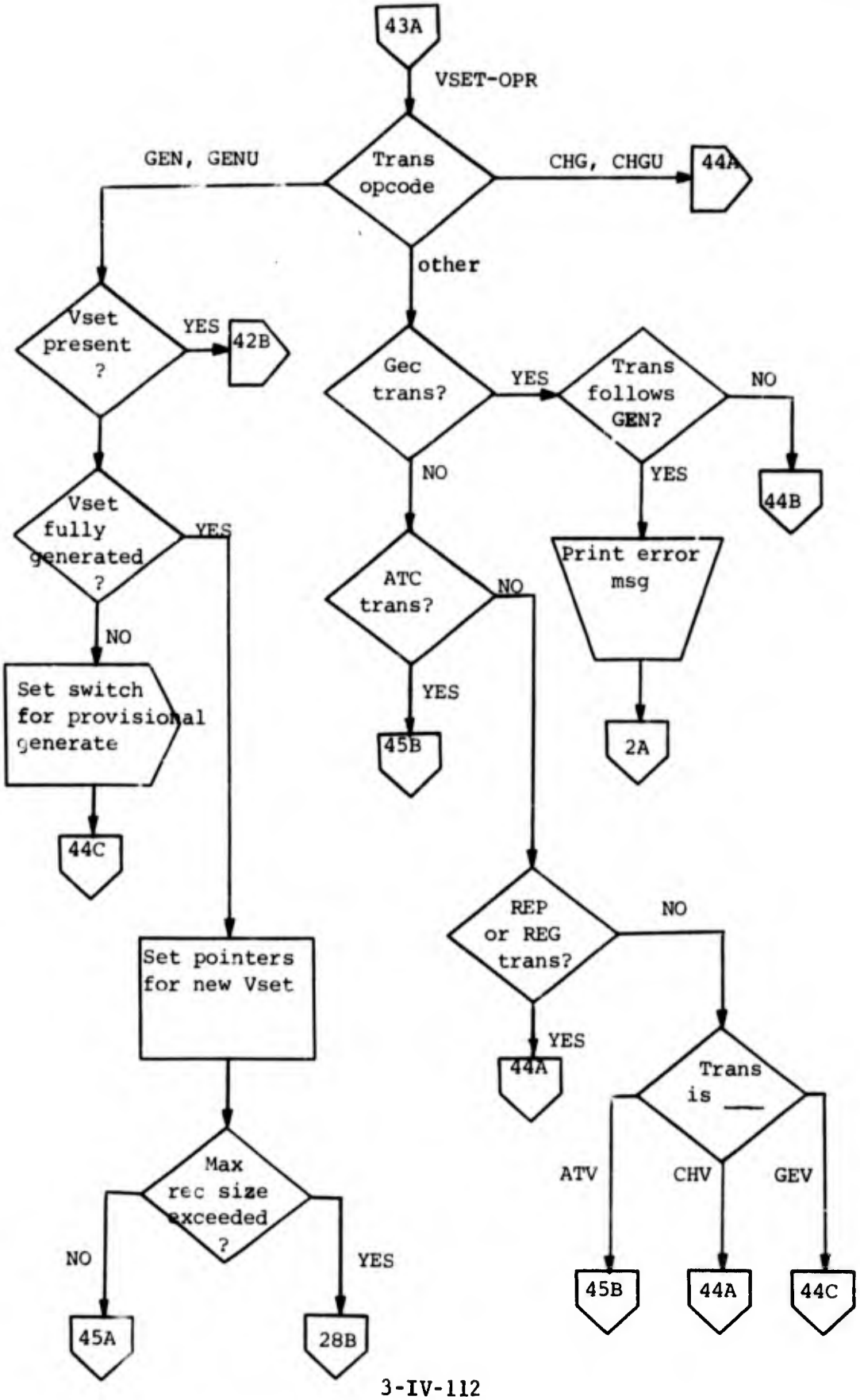

Enter via PERFORM
Updating old set?
YES
HOP of old set --> OLD-FLD-HOP
NO
Exit
OLD-FLD-HOP is used in confirmation processing

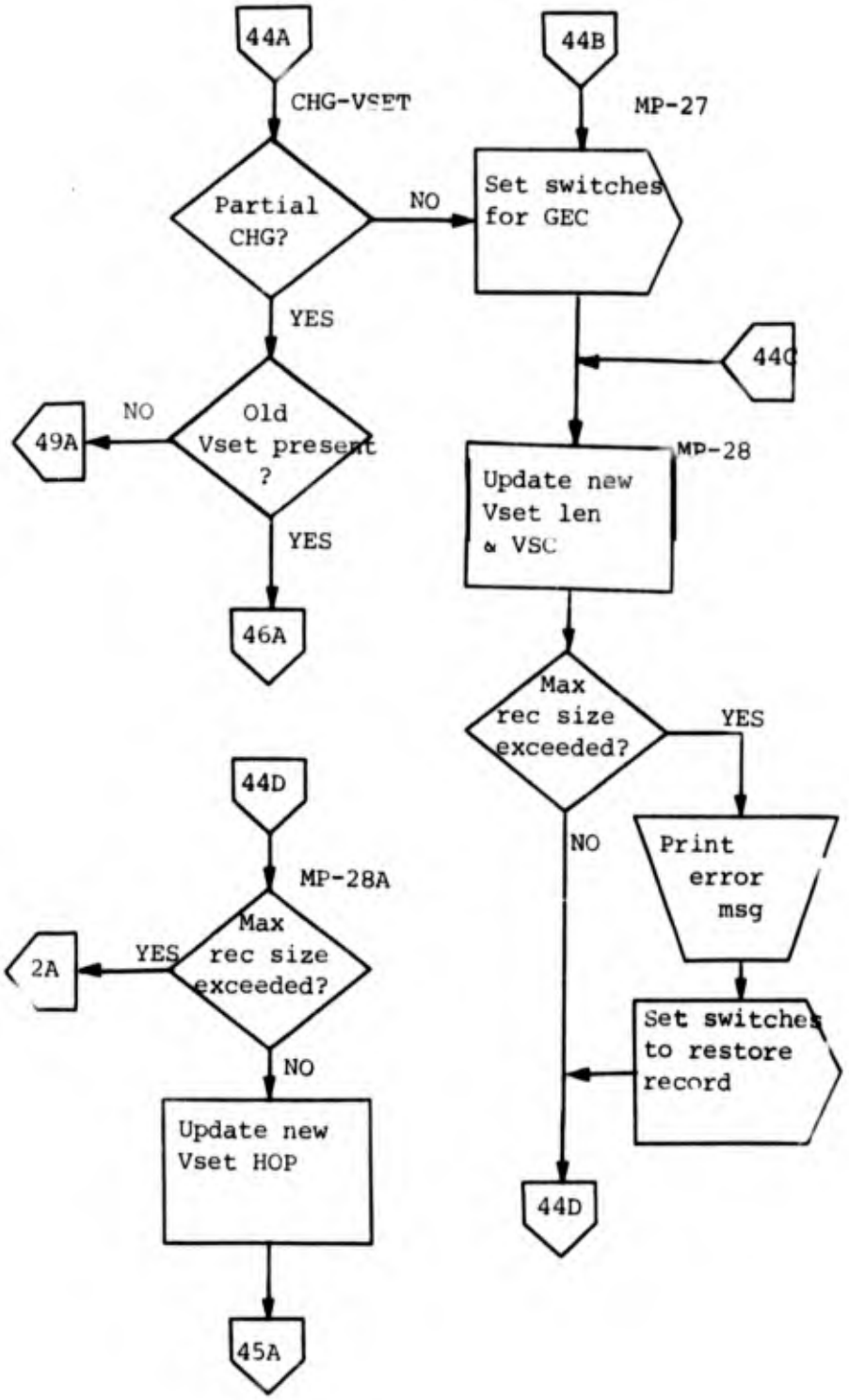

39A
VSET-FIELDS
First trans this Vset?
YES
40B
NO
Multiple CHG's?
YES
38B
NO
40A

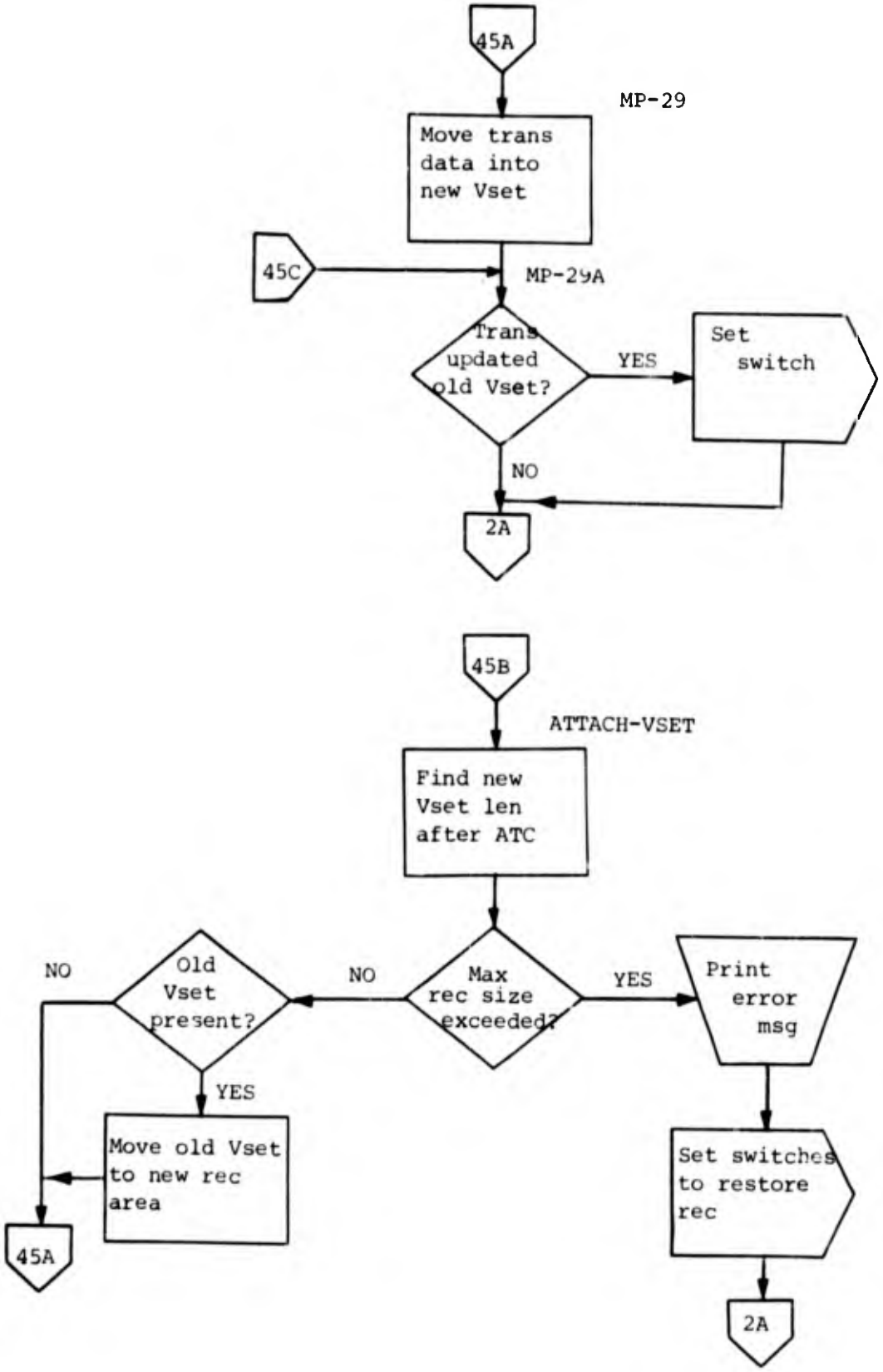

40A
Non-ATC follows ATC?
YES
Print error msg
NO
42A
2A
40B
Last trans was on Vset ?
YES
Any valid trans on that set?
NO
NO
41A
Set up pointer to old set
YES
Confirmation needed
NO
Print confirma-tion
41A

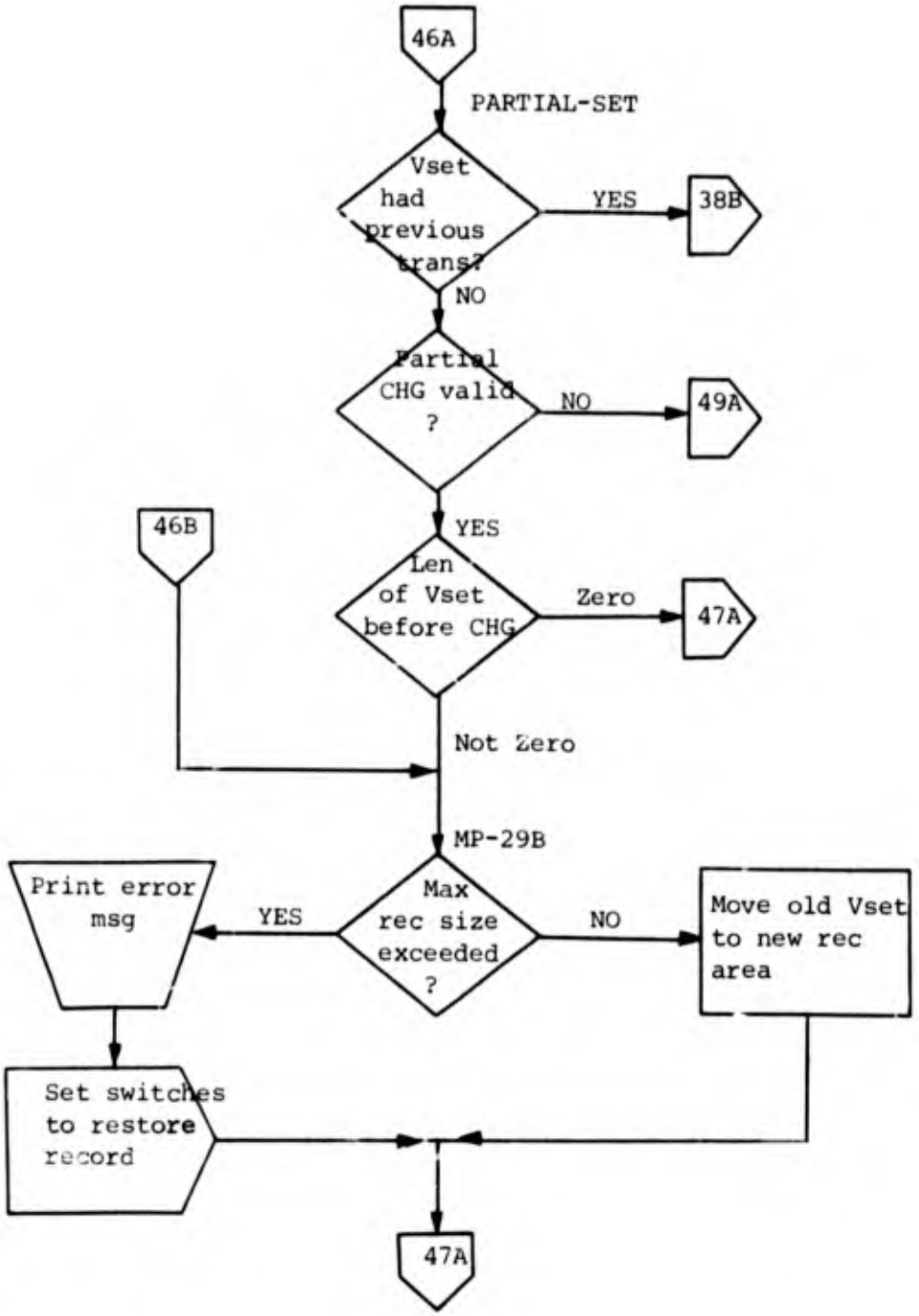

SKIP-OLD-VSET
Enter via PERFORM
Push pointer past old Vset
Exit
41A
Initialize for Vset
MP-24
Perform MOVE-SETS-NOT-PROCES-SED thru MSNP-EXIT
Max rec size exceeded?
YES
2A
NO
Updating old set ?
NO
Set switche
YES
Push pointer past old Vset
YES
Vset being deleted?
42A
NO
42A
42A

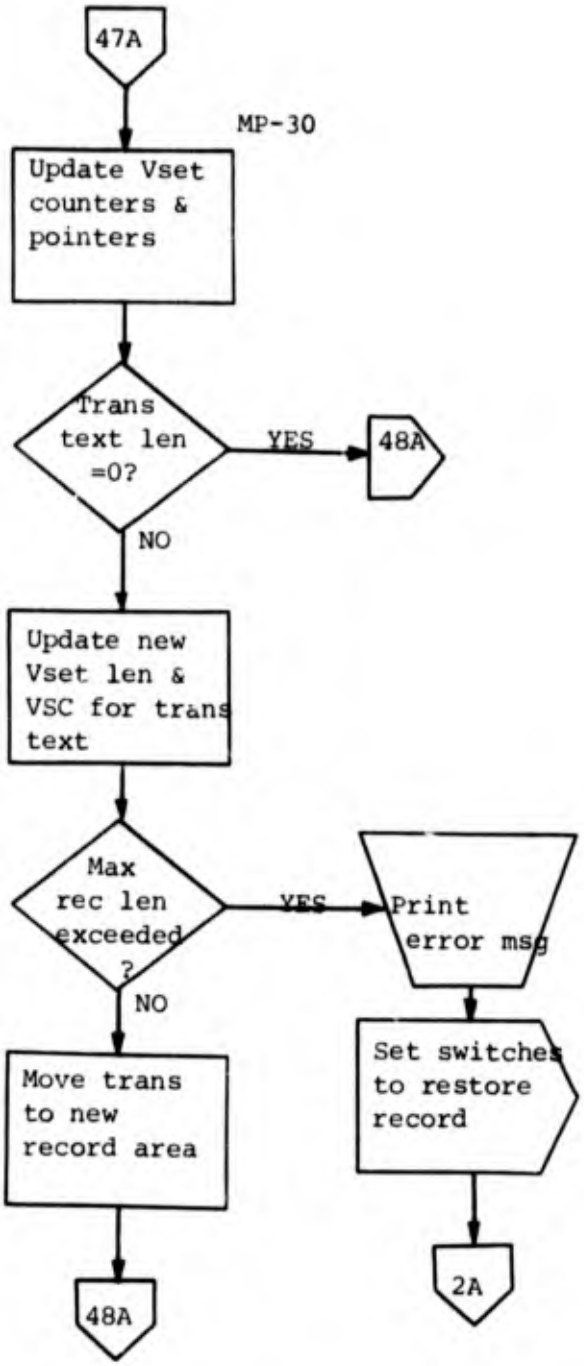

42A
MP-26
What must old Vset status be?
don't care
absent
present
CK11
CK22
Old Vset absent?
NO
YES
Set switches for provisional generated
43A
Old Vset Present ?
NO
43A
Print error msg
2A
Old Vset present ?
NO
43A
42B
YES
Print error msg
2A

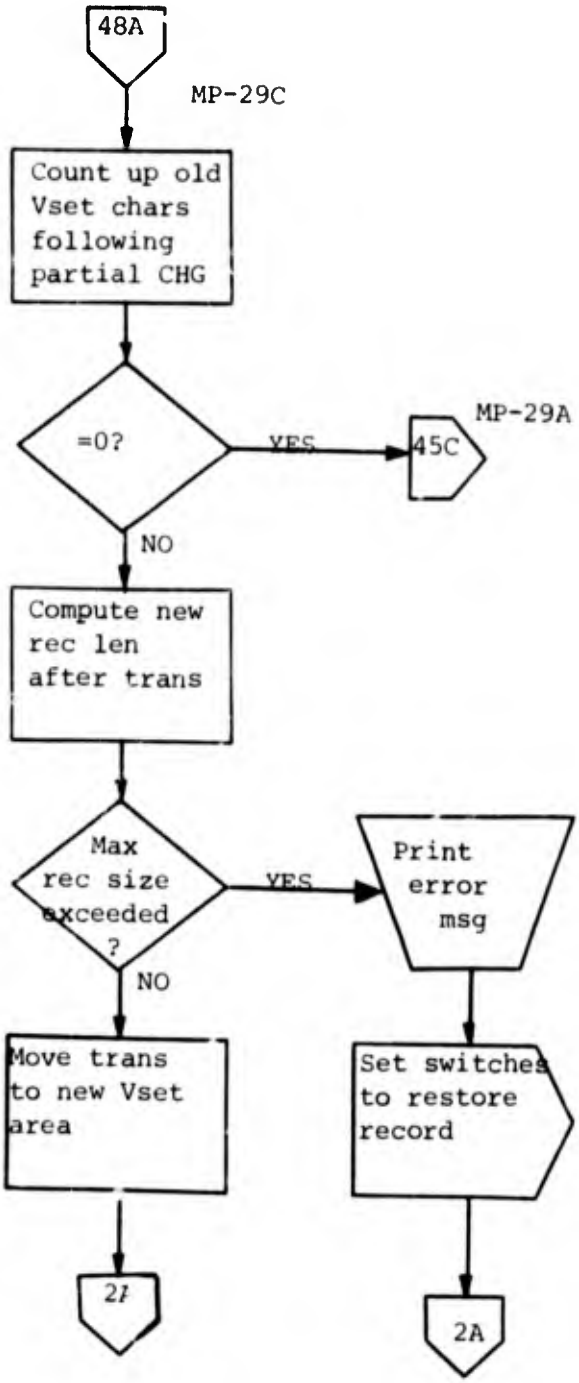

43A
VSET-OPR
Trans opcode
GEN, GENU
CHG, CHGU
44A
other
Vset present ?
YES
42B
Gec trans?
YES
Trans follows GEN?
NO
44B
NO
YES
Vset fully generated ?
YES
ATC trans?
NO
Print error msg
NO
Set switch for provisional generate
YES
45B
2A
44C
REP or REG trans?
NO
Set pointers for new Vset
YES
44A
Trans is ___
Max rec size exceeded ?
ATV
CHV
GEV
NO
YES
45B
44A
44C
45A
28B

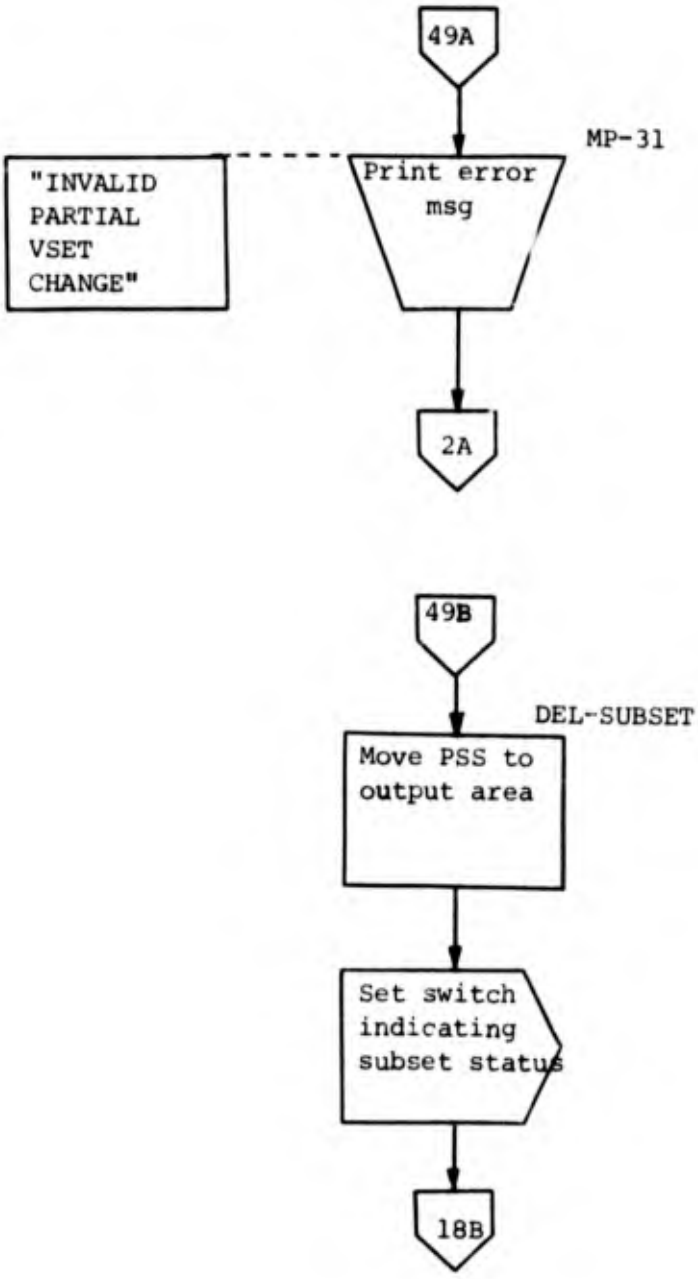

44A
CHG-VSET
Partial CHG?
NO
YES
Old Vset present ?
NO
49A
YES
46A
44B
MP-27
Set switches for GEC
44C
MP-28
Update new Vset len & VSC
Max rec size exceeded?
YES
Print error msg
NO
Set switches to restore record
44D
44D
MP-28A
Max rec size exceeded?
YES
2A
NO
Update new Vset HOP
45A

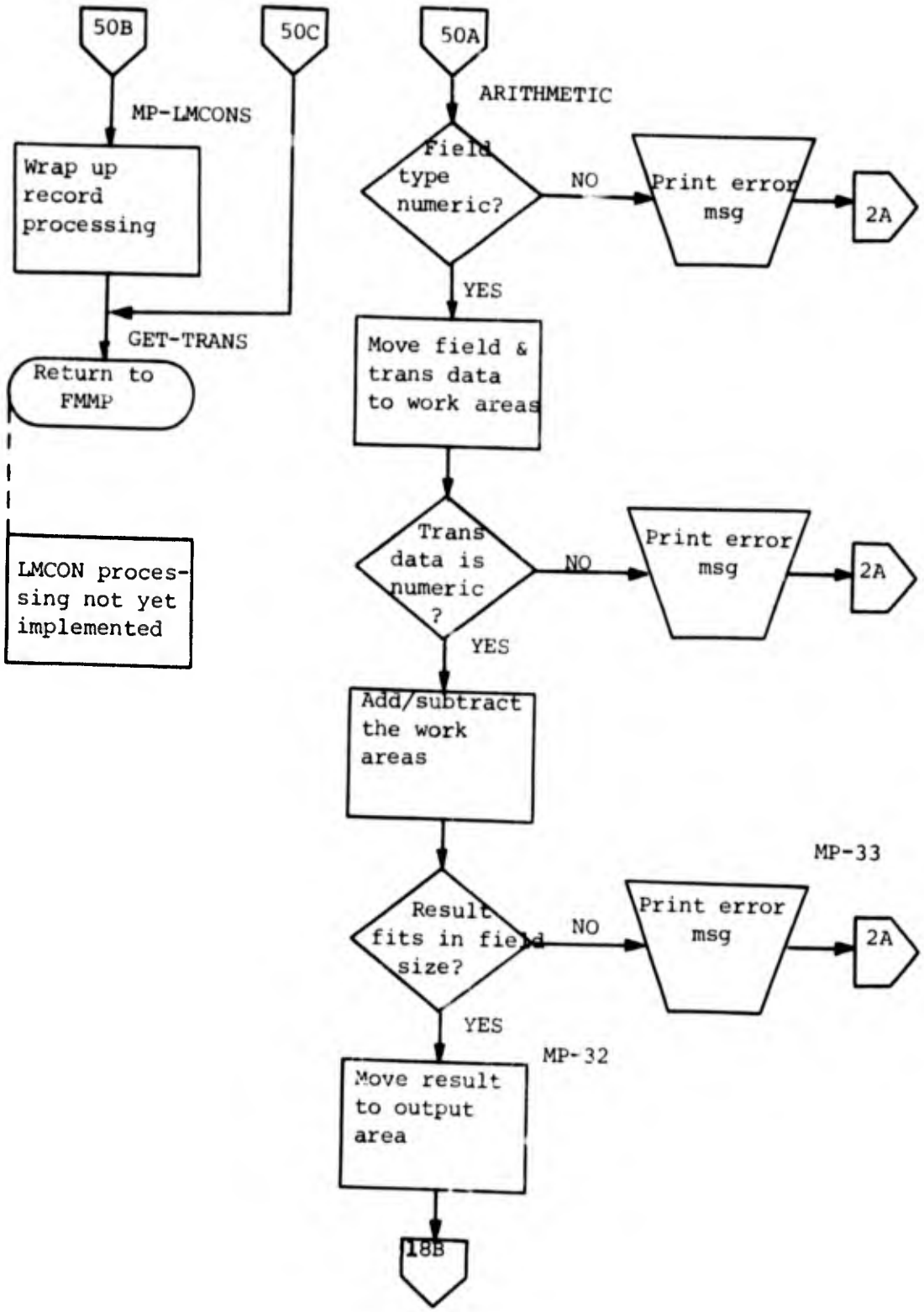

45A
MP-29
Move trans data into new Vset
45C
MP-29A
Trans updated old Vset?
YES
Set switch
NO
2A
45B
ATTACH-VSET
Find new Vset len after ATC
Max rec size exceeded?
NO
YES
Print error msg
Old Vset present?
NO
YES
Move old Vset to new rec area
45A
Set switches to restore rec
2A

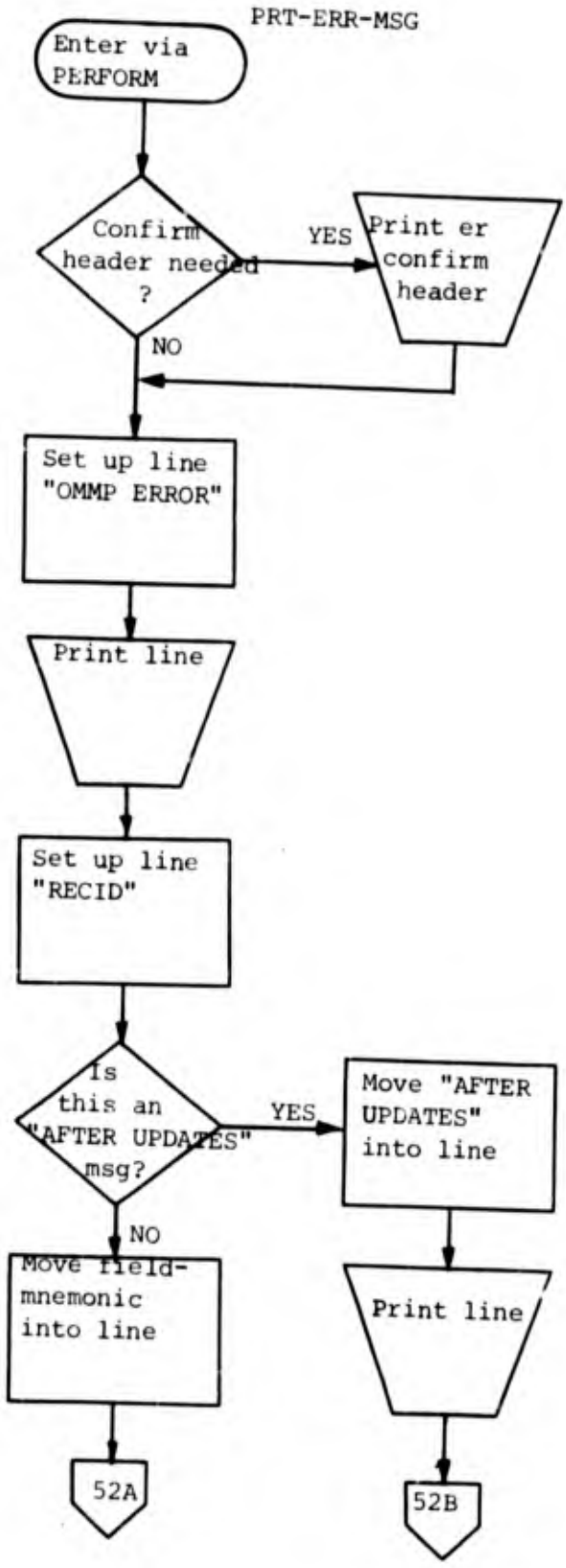

46A
PARTIAL-SET
Vset had previous trans?
YES
38B
NO
Partial CHG valid ?
NO
49A
YES
46B
Len of Vset before CHG
Zero
47A
Not Zero
MP-29B
Print error msg
YES
Max rec size exceeded ?
NO
Move old Vset to new rec area
Set switches to restore record
47A

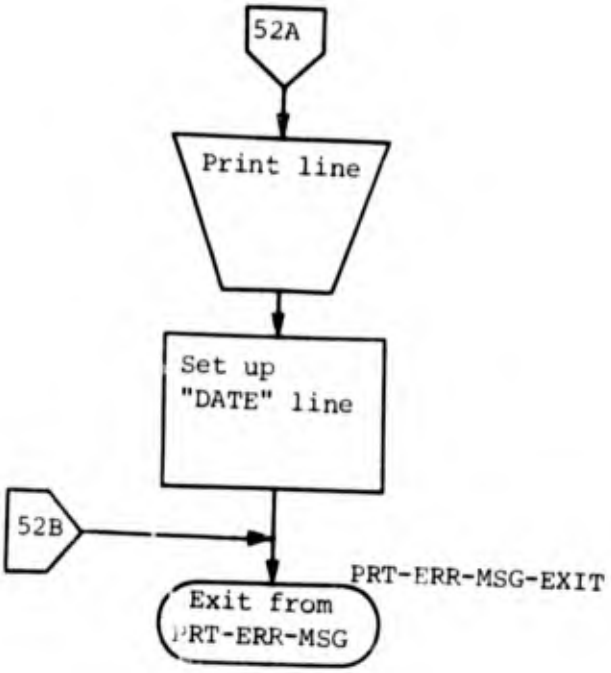

47A
MP-30
Update Vset counters & pointers
Trans text len =0?
YES
48A
NO
Update new Vset len & VSC for trans text
Max rec len exceeded ?
YES
Print error msg
NO
Move trans to new record area
Set switches to restore record
48A
2A

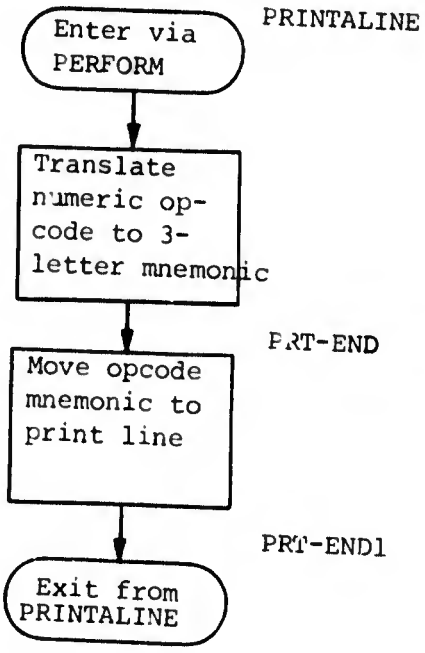

48A
MP-29C
Count up old Vset chars following partial CHG
=0?
YES
MP-29A
45C
NO
Compute new rec len after trans
Max rec size exceeded ?
YES
Print error msg
NO
Move trans to new Vset area
Set switches to restore record
2I
2A

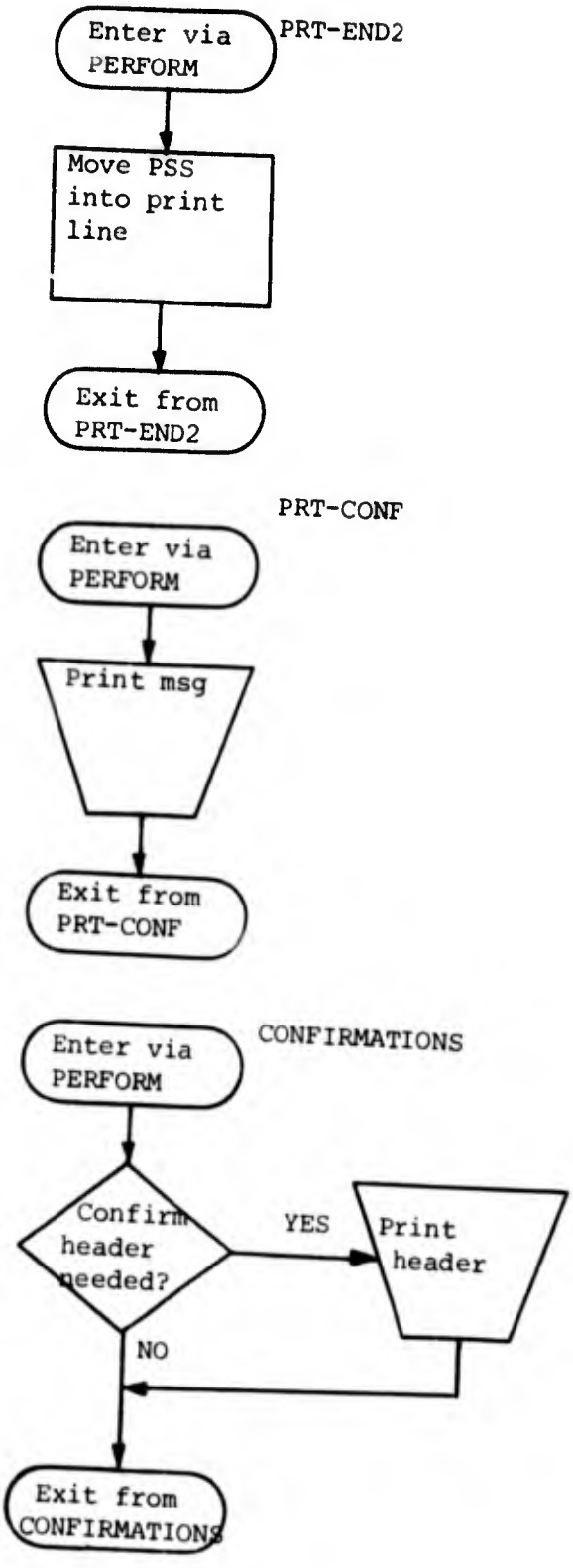

49A
MP-31
"INVALID
PARTIAL
VSET
CHANGE"
Print error
msg
2A
49B
DEL-SUBSET
Move PSS to
output area
Set switch
indicating
subset status
18B

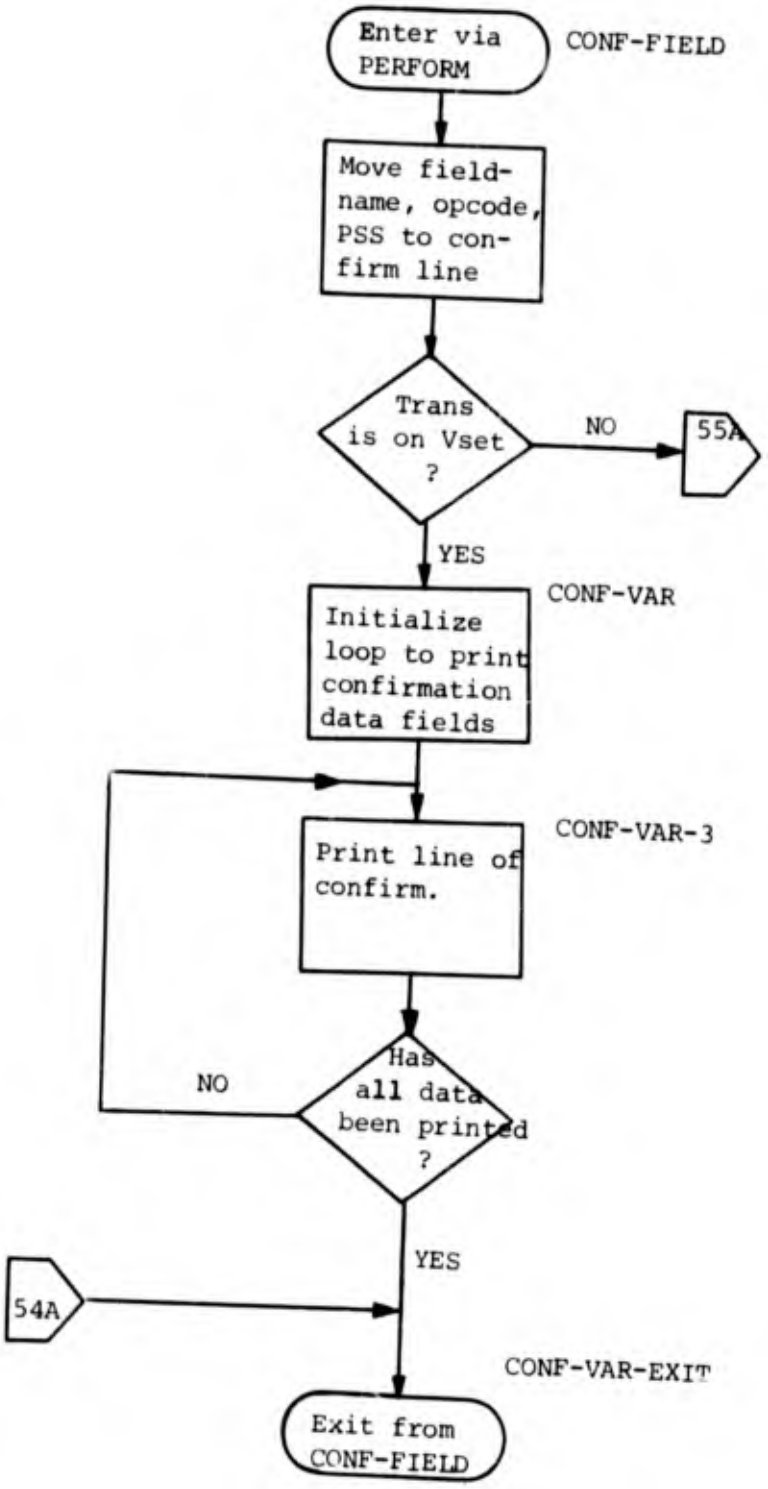

50B
50C
50A
MP-LMCONS
ARITHMETIC
Wrap up record processing
Field type numeric?
NO
Print error msg
2A
YES
GET-TRANS
Move field & trans data to work areas
Return to FMMP
Trans data is numeric ?
NO
Print error msg
2A
YES
LMCON processing not yet implemented
Add/subtract the work areas
MP-33
Result fits in field size?
NO
Print error msg
2A
YES
MP-32
Move result to output area
18B

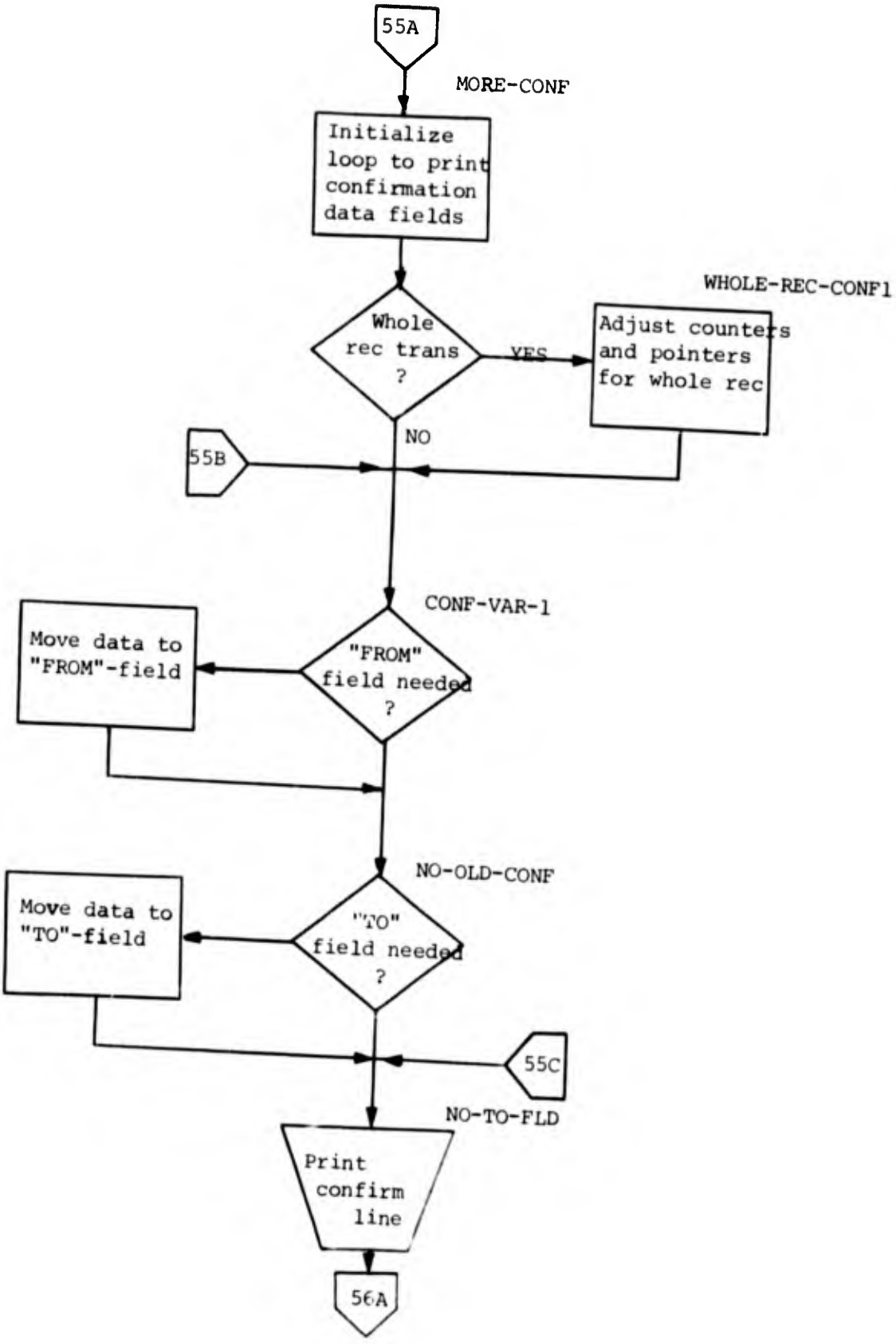

PRT-ERR-MSG
Enter via PERFORM
Confirm header needed ?
YES
Print er confirm header
NO
Set up line "OMMP ERROR"
Print line
Set up line "RECID"
Is this an "AFTER UPDATES" msg?
YES
Move "AFTER UPDATES" into line
NO
Move field-mnemonic into line
Print line
52A
52B


52A
Print line
Set up
"DATE" line
52B
PRT-ERR-MSG-EXIT
Exit from
PRT-ERR-MSG


Enter via
PERFORM
PRINTALINE
Translate
numeric op-
code to 3-
letter mnemonic
PRT-END
Move opcode
mnemonic to
print line
PRT-END1
Exit from
PRINTALINE


Enter via PERFORM
PRT-END2
Move PSS into print line
Exit from PRT-END2
PRT-CONF
Enter via PERFORM
Print msg
Exit from PRT-CONF
Enter via PERFORM
CONFIRMATIONS
Confirm header needed?
YES
Print header
NO
Exit from CONFIRMATIONS


Enter via PERFORM
CONF-FIELD
Move field-name, opcode, PSS to con-firm line
Trans is on Vset ?
NO
55A
YES
CONF-VAR
Initialize loop to print confirmation data fields
CONF-VAR-3
Print line of confirm.
Has all data been printed ?
NO
YES
54A
CONF-VAR-EXIT
Exit from CONF-FIELD


55A
MORE-CONF
Initialize loop to print confirmation data fields
WHOLE-REC-CONF1
Whole rec trans ?
YES
Adjust counters and pointers for whole rec
NO
55B
CONF-VAR-1
Move data to "FROM"-field
"FROM" field needed ?
NO-OLD-CONF
Move data to "TO"-field
"TO" field needed ?
55C
NO-TO-FLD
Print confirm line
56A

e. OMMPTIO

(1) Function. To read transaction file records and write overflow transaction records.

(2) Calling Sequence.

```
ENTRY 'MPTIO'  USING MP-DD  TRANS-TABLE.
CALL  'MPTRN'  Using MP-DD  TRANS-TABLE LINK-TRANS-REC.
```

(3) Program Description.

OMMPTIO.

Handles the input processing of transaction records and the outputting of overflow transactions, records exceeding the table sizes. OMMPTIO acts according to the setting of MPTIO-SW.

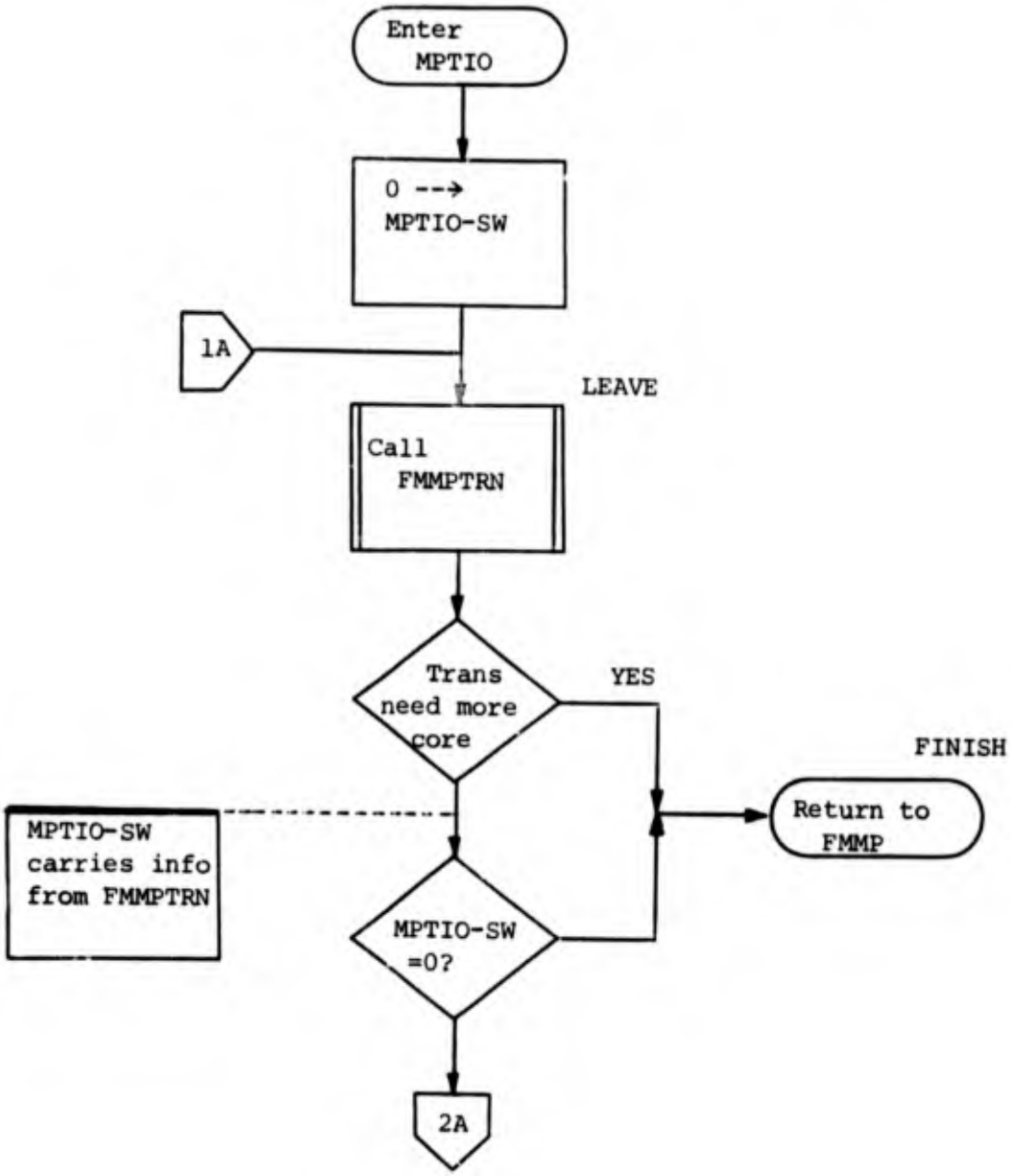
Ø = Return to OMMPX
1, 2, 3 = Read one transaction record

TRANS-SW.

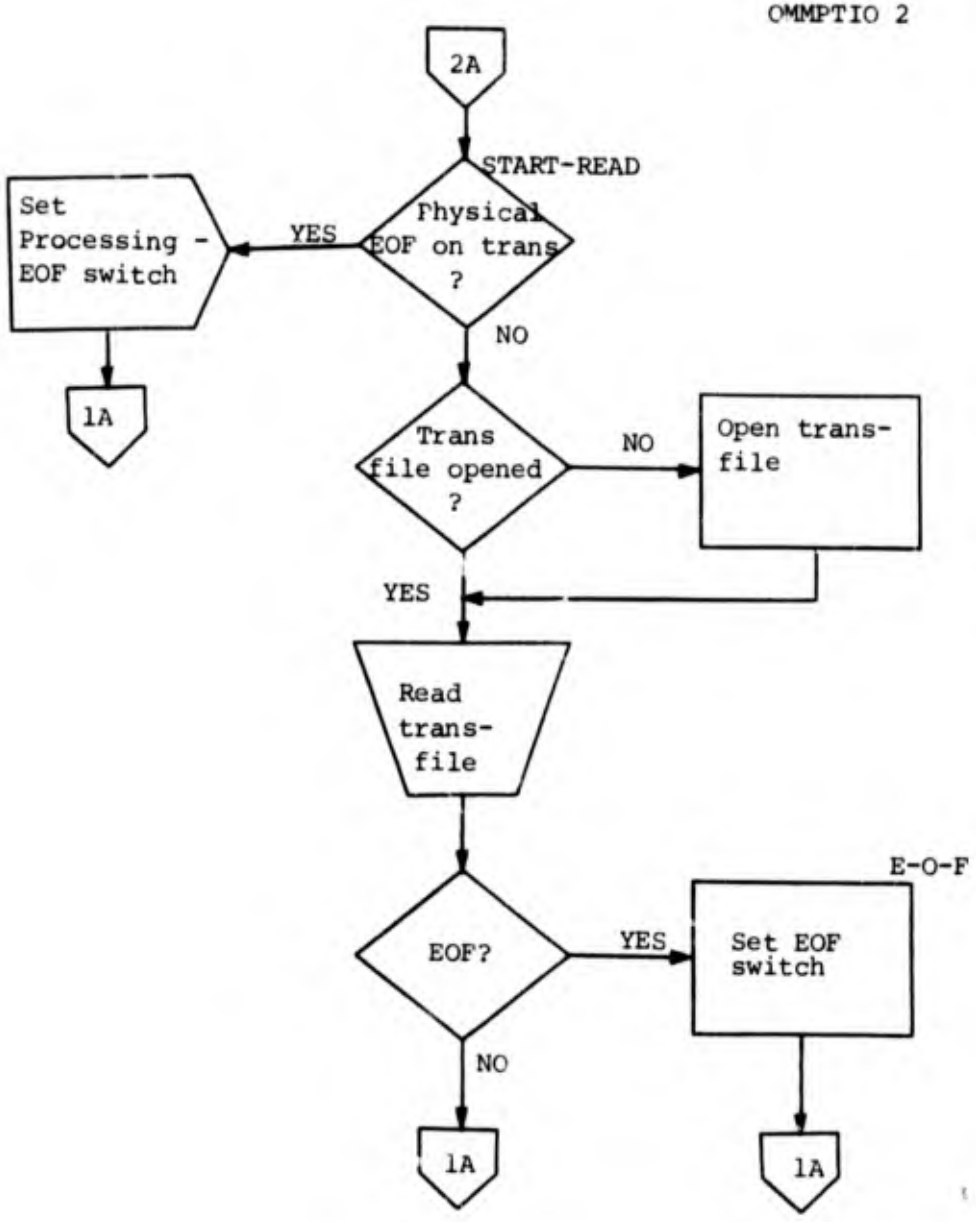
Is a switch set within OMMPX, OMMPTIO, OMMPTRN. It controls the processing of all three subprograms depending on its value:

Ø = Start processing, open transaction file
1 = Normal processing
2 = Processing end of file, terminate OM
3 = Physical end of file, transactions still to be processed

OMMPTIO 1


Enter MPTIO
0 --> MPTIO-SW
1A
LEAVE
Call FMMPTRN
Trans need more core
YES
FINISH
Return to FMMP
MPTIO-SW carries info from FMMPTRN
MPTIO-SW =0?
2A

OMMPTIO 2


2A
START-READ
Physical EOF on trans ?
YES
Set Processing - EOF switch
1A
NO
Trans file opened ?
NO
Open trans-file
YES
Read trans-file
EOF?
YES
E-O-F
Set EOF switch
NO
1A
1A

f. OMMPTRN

(1) Function. To set up transactions against a single MIDMS file record so that OMMPX can process the transactions.

(2) Calling Sequence.

```
ENTRY 'MPTRN'       USING MP-DD TRANS-TABLE LINK-RECORD.
CALL  'MOVARAYS'    USING ...
```

(3) Program Description. OMMPTRN performs the processing necessary to build a single group of transactions to update a MIDMS file record even when several input groups originally produced the transactions. As OMMPTRN decides to read a transaction record, it will return to OMMPTIO to have the record read. (OMMPTIO will recall OMMPTRN for processing.) If a combined group exceeds the capacity of the area holding it, additional transactions for the same data file record are placed in an overflow file by OMMPTIO. Upon completion of an entire group, OMMPTIO will return to OMMPX. The value of the switch MPTIO-SW determines the action of OMMPTIO after the return from OMMPTRN:

Ø = Return to OMMPX
1 = Read one transaction record

Upon entry to OMPTRN, MP-ERROR is tested to determine if processing should continue. If OM-SW is equal to 4, AREA2 has been open with the new size needed for the transaction. MPTRIO-SW is tested to determine where to continue.

ENTER-MPTRANS.
Initializes various control fields for a new input group. When TRANS-SW is 1, the next transaction record is already in memory so that processing can begin immediately. A physical end of file condition is changed to processing end of file by the modification of TRANS-SW from 3 to 2.

READ-RETURN.
Starts processing of a record obtained by OMMPTIO. Processing end of file, and physical end of file conditions are tested. If LINK-RECID is low value, Confirmation Record information is saved and a new record is read.

GRPID-TEST.
First determines if the record is an item record. If not it is a control record. If it is for the same group as the previous control record was for, is this a good group? If the total number of subgroups has been exceeded stop processing this subgroup and perform procedure to eliminate, forward and backward, processing for this group. Set FIRST-SW to 1 for the first item in a new group. Make appropriate adjustments to TOT-ITEM-CNT, TOT-TEXT-CNT and TOT-CHAR-CNT.

FIND-TOT-SIZE.

Determines if AREA2 is presently large enough to hold the transactions for this MIDMS record. If not AREA2 size is readjusted by returning to FMMP.

ADJ-BUFFER-SIZE.

Provides a return point for those cases in which AREA2 had to be enlarged.

ITEM-REC.

Is the return point when an item record has been processed for a subgroup and determines what kind of record is now present from LINK-TYPE (0=control, 1=item, 2=text). A control record at this point signifies the beginning of a new MIDMS record and all the transactions for the previous transaction record must be merged (there is no text). If LINK-TYPE is equal to 2, control passes to SET-UP-TEXT. If LINK-TYPE is 1 then more items are moved in by paragraph MOVE-ITEMS.

SET-UP-TEXT.

Sets up pointers to move the textual data in. If any is present (i.e., none would be for a DELETE) BLD-SUP-TEXT actually does the moving.

TEXT-REC.

Is the return point when there has previously been a text record for this subgroup. If this record is not a control record, the text is moved in via BLD-SUP-TEXT. If it is the end of a group merge the transactions via START-MERGE otherwise set up the pointers and move in the text.

PRE-MERGE.

Initializes TRANS-LIST pointers prior to the operations of START-MERGE.

START-MERGE.

Initializes pointers for the merging of all transactions for a given MIDMS record.

START-MERGE.

Initiates each pass of the merge process.

GETNEX-GRP.

Determines whether an item in one group has a smaller key than the previous items from other groups tested during this pass.

ENDGRP-TST.

Determines whether all groups have been tested in this pass. If not, the next group is obtained.

UN-FLAG.

Clears FLAG-SET to Ø indicating that a pseudo subset sequence number is not the minimum item entry.

ADJ-GRPCNT.

Saves the number of the group containing the minimum item, the value of the minimum key, and the item number of the minimum item.

BUILD-SUPTRANS.

Modifies the TRANS-TEXT-HOP of the item with the minimum key (to reflect the position of the group's text in TRANS-TEXT) and determines if the item refers to a pseudo subset sequence

number. If not, the subscript of the item is placed in TRANS-LIST. If not all transaction items have been handled, the pointer to the first item of the group containing the minimum item is adjusted by one (so that the same item will not be referenced again) and control is given to START-MERGE.

ADJ-PSEUDO and MOVE-GRPSUB.

Handle pseudo subset sequence numbers. First they add proper adjustment to the subset number to reflect pseudos in the previous groups. If more transaction items are to be processed, the next item in the group is checked to see if it refers to the same set (it will also have a pseudo subset). If not, the pseudo adjustment (PSEUDO-ADJ) for the next group is computed as 600 less than the last pseudo subset number.

ENDGRP-TST2.

Determines whether all groups have been processed after a pseudo has been located. If not, the first unprocessed item of the next group is checked to see if the same set is referenced.

FOUND-MIN thru FG1.

Find the minimum pseudo in all the groups and readjusts its pseudo number. This loop must be done and the pseudo readjusted for every pseudo present.

UN-FLAG2.

Clears PSEUDO-ADJ before beginning the next pass of the merge.

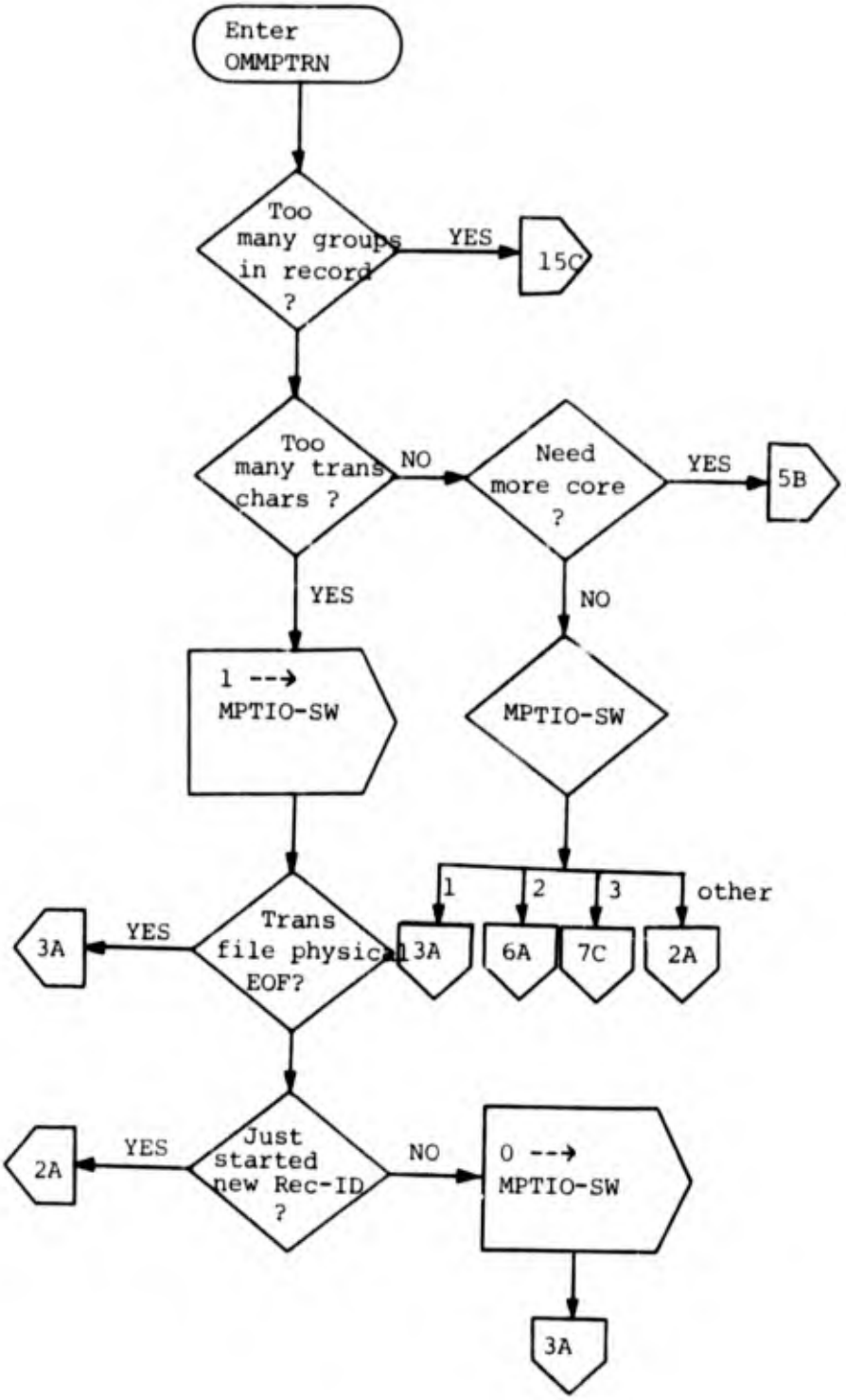
BACK-UP thru BACK-UP-EXIT.

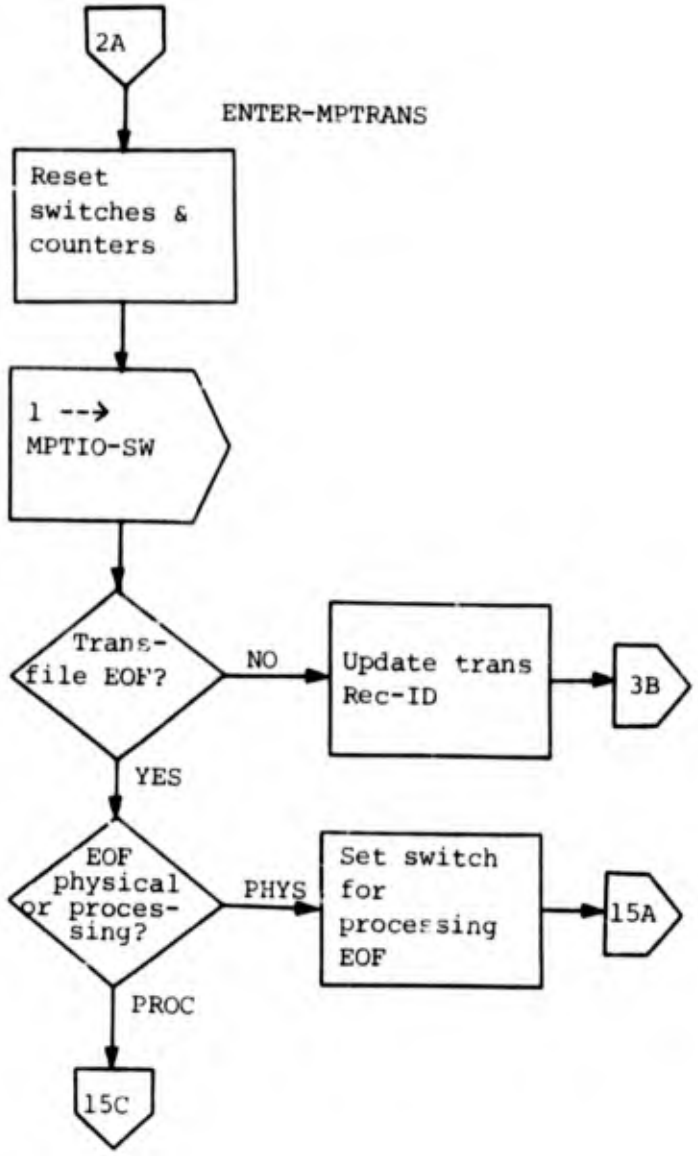
Reset TOT-CHAR-CNT, TOT-ITEM-CNT and TOT-GRP-CNT when the maximum number of subgroups has been exceeded. All subgroups within this group are eliminated.

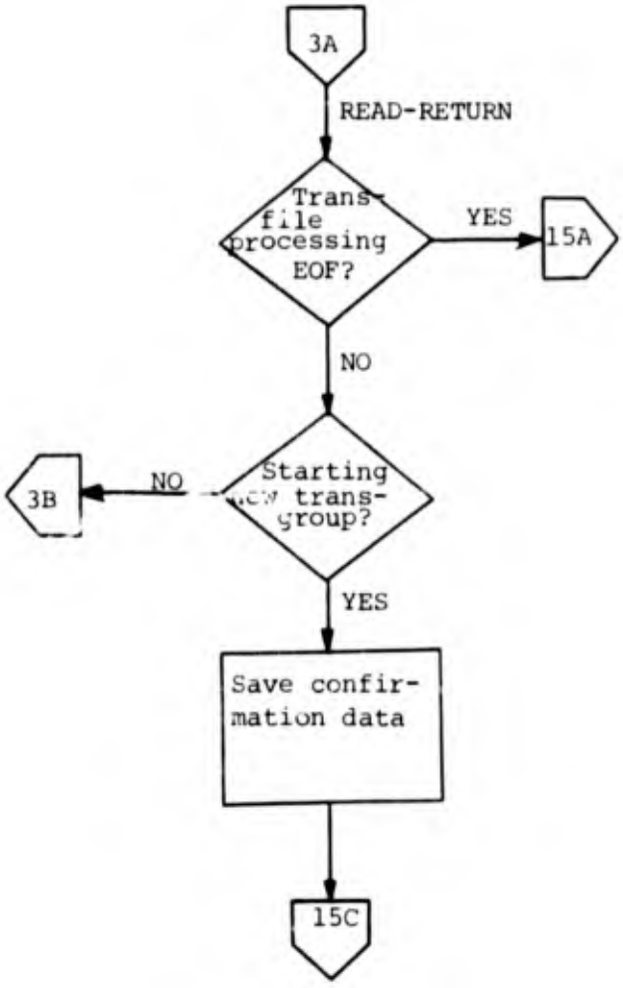
LEAVE-TRANS.

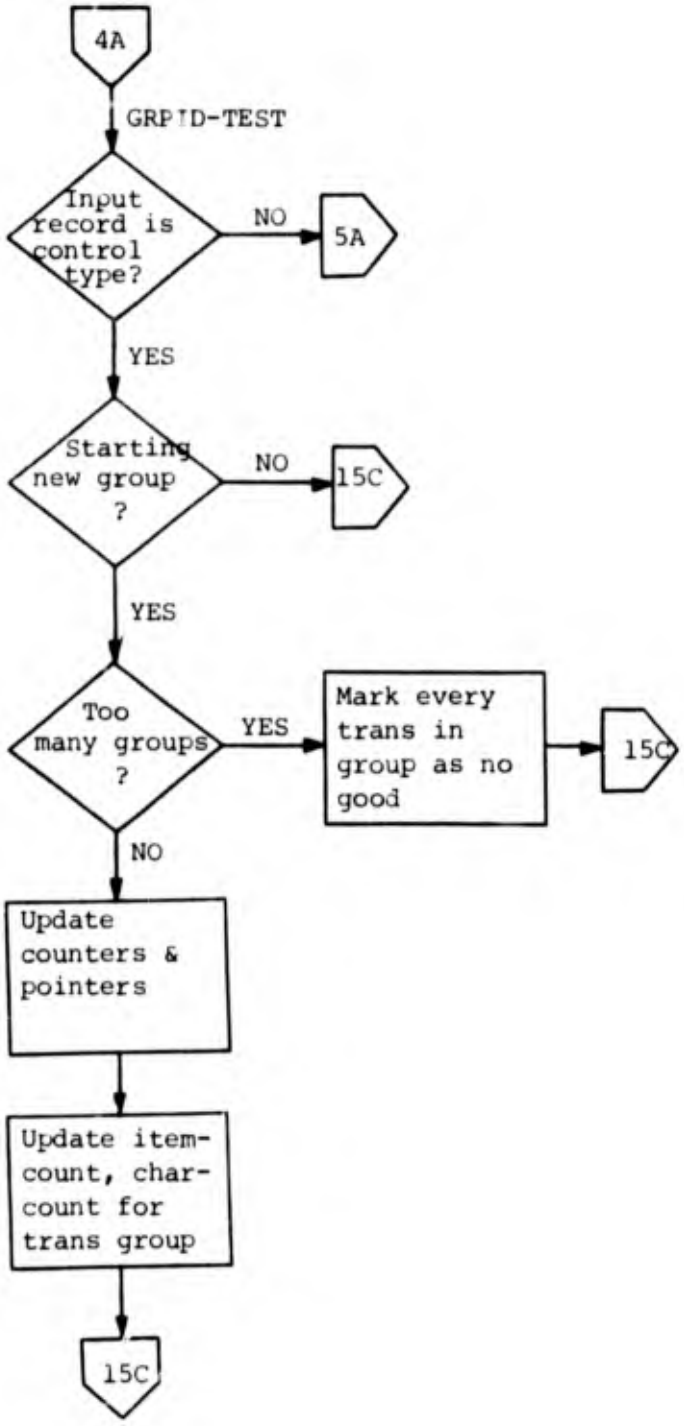
Completes the processing of OMMPTRN by moving 0 to MPTIO-SW and returns to OMMPTIO.

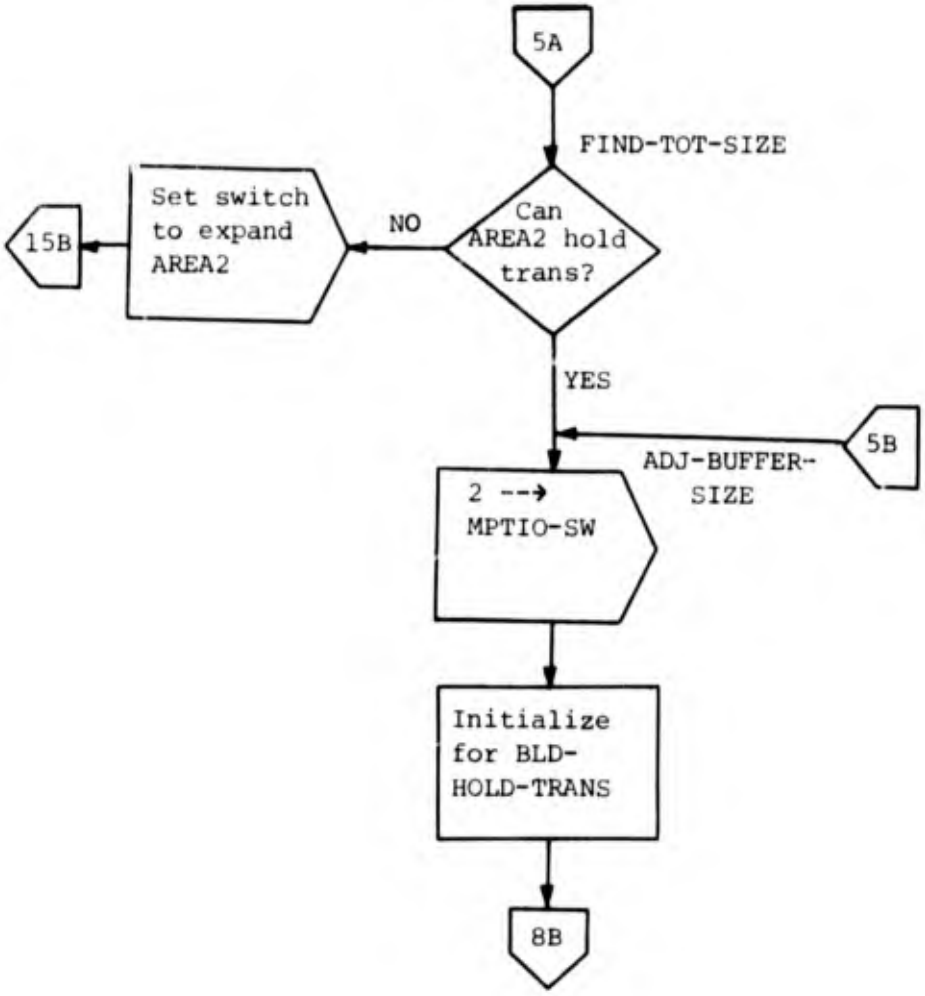
(4) Limitations. None.

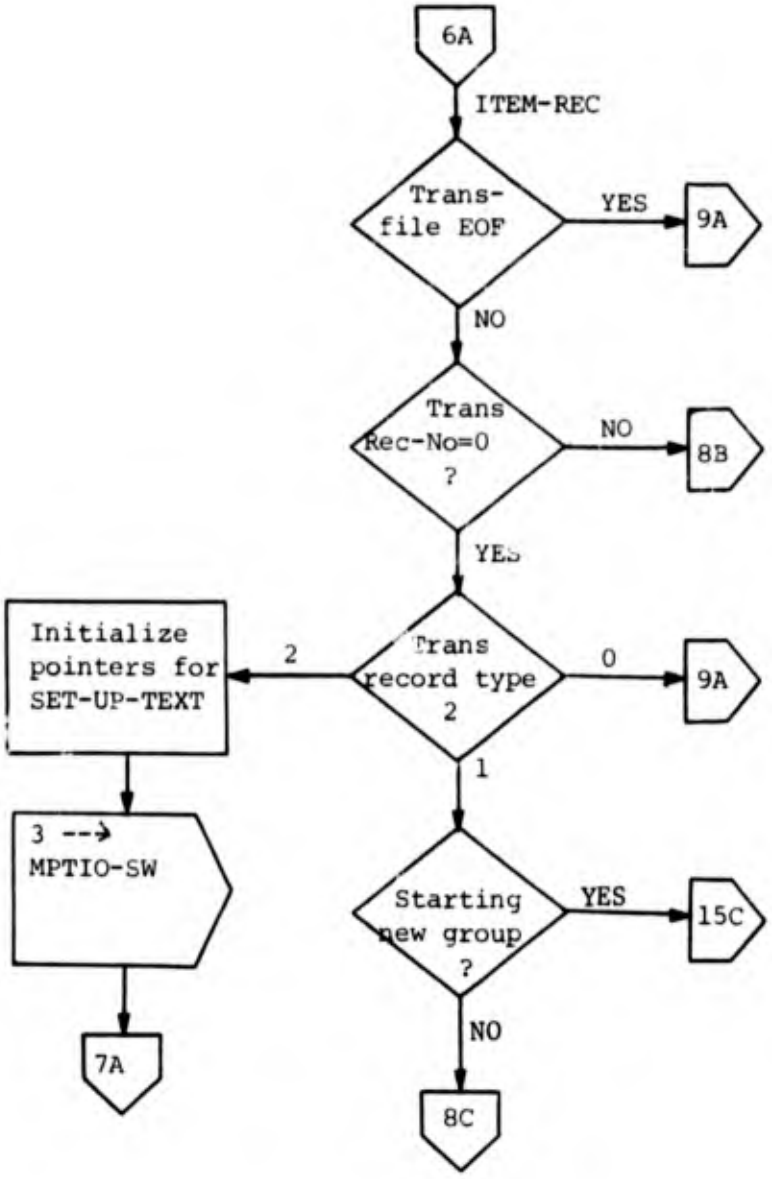

Enter OMMPTRN
Too many groups in record ?
YES
15C
Too many trans chars ?
NO
Need more core ?
YES
5B
YES
NO
1 --> MPTIO-SW
MPTIO-SW
1
2
3
other
Trans file physic EOF?
YES
3A
13A
6A
7C
2A
Just started new Rec-ID ?
YES
2A
NO
0 --> MPTIO-SW
3A

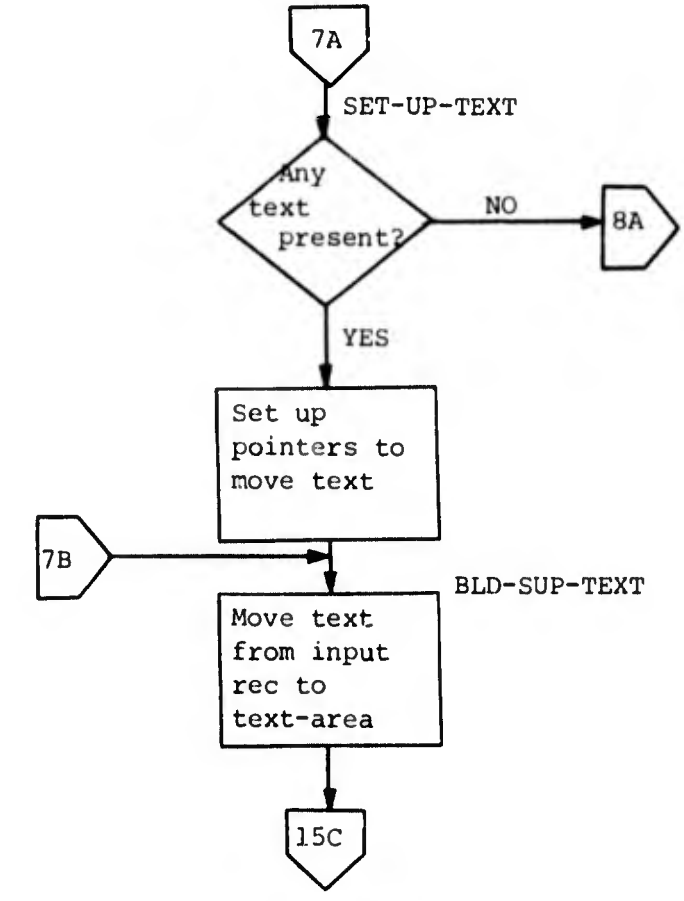

2A
ENTER-MPTRANS
Reset switches & counters
1 --> MPTIO-SW
Trans-file EOF?
NO
Update trans Rec-ID
3B
YES
EOF physical or proces-sing?
PHYS
Set switch for processing EOF
15A
PROC
15C

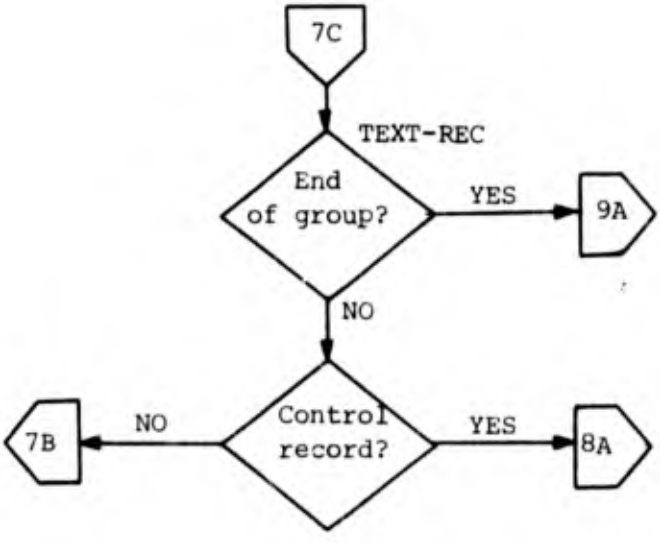

3A
READ-RETURN
Trans-
file
processing
EOF?
YES
15A
NO
3B
NO
Starting
new trans-
group?
YES
Save confir-
mation data
15C

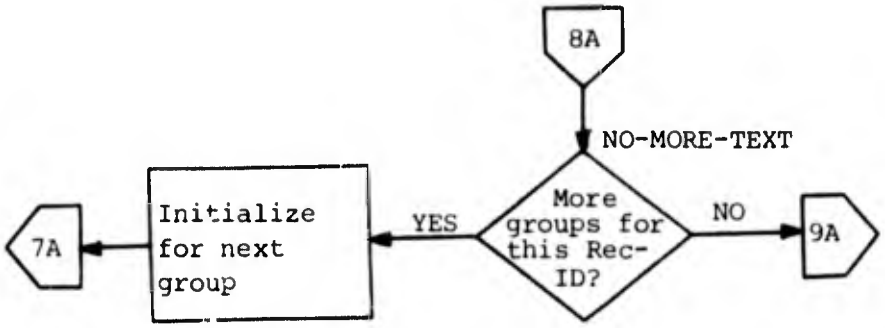

4A
GRPID-TEST
Input record is control type?
NO
5A
YES
Starting new group ?
NO
15C
YES
Too many groups ?
YES
Mark every trans in group as no good
15C
NO
Update counters & pointers
Update item-count, char-count for trans group
15C

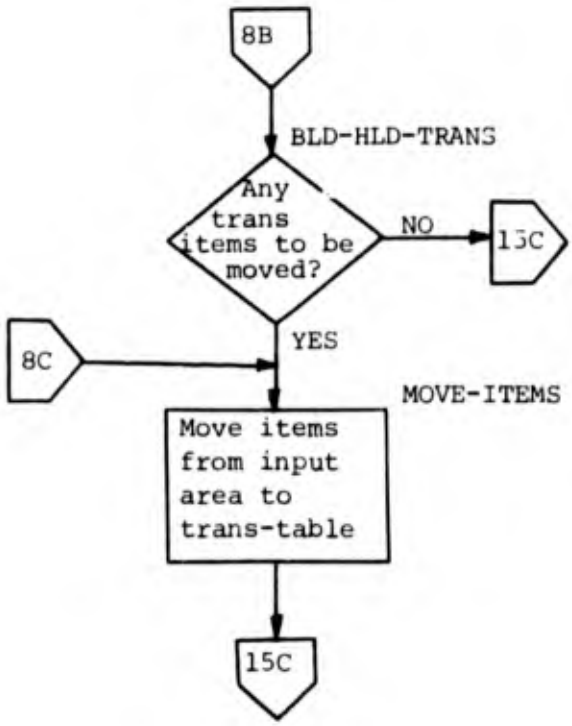

5A
FIND-TOT-SIZE
Can AREA2 hold trans?
NO
Set switch to expand AREA2
15B
YES
5B
ADJ-BUFFER-SIZE
2 --→ MPTIO-SW
Initialize for BLD-HOLD-TRANS
8B

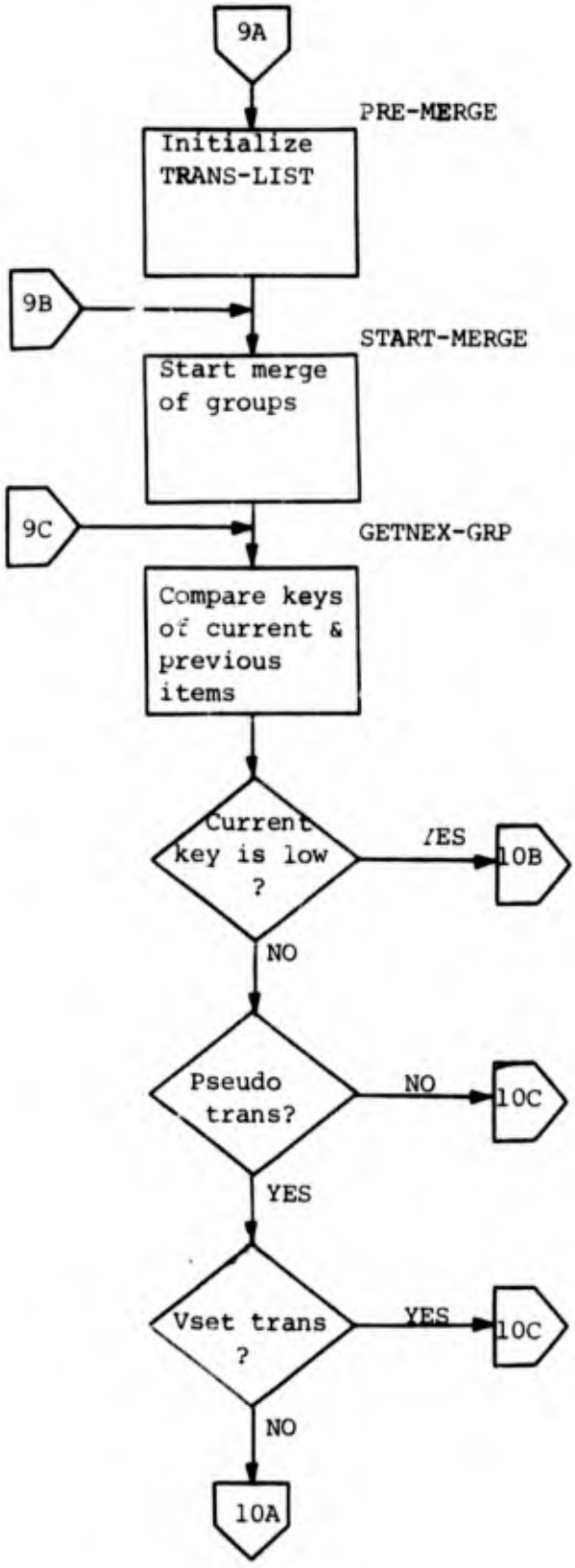

6A
ITEM-REC
Trans-file EOF
YES
9A
NO
Trans Rec-No=0 ?
NO
8B
YES
Initialize pointers for SET-UP-TEXT
2
Trans record type 2
0
9A
1
3 --> MPTIO-SW
Starting new group ?
YES
15C
7A
NO
8C

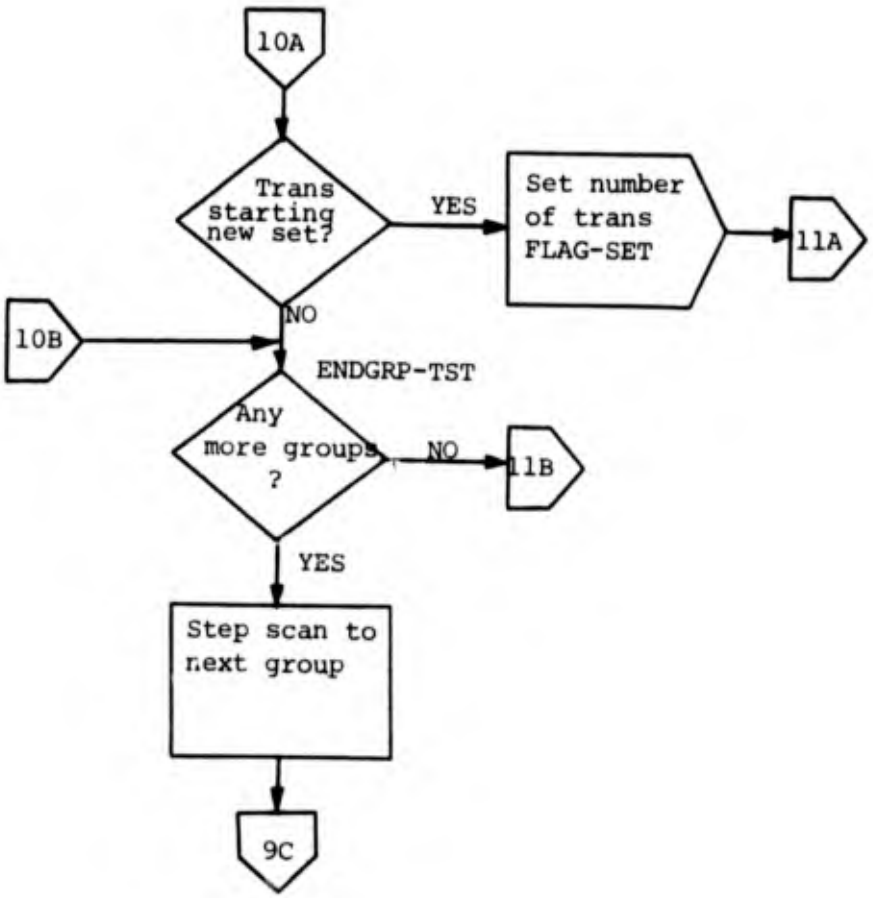

7A
SET-UP-TEXT
Any text present?
NO
8A
YES
Set up pointers to move text
7B
BLD-SUP-TEXT
Move text from input rec to text-area
15C

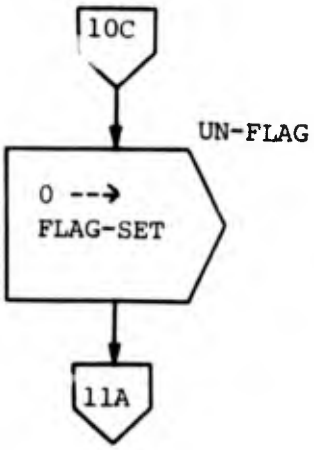

7C
TEXT-REC
End of group?
YES
9A
NO
7B
NO
Control record?
YES
8A

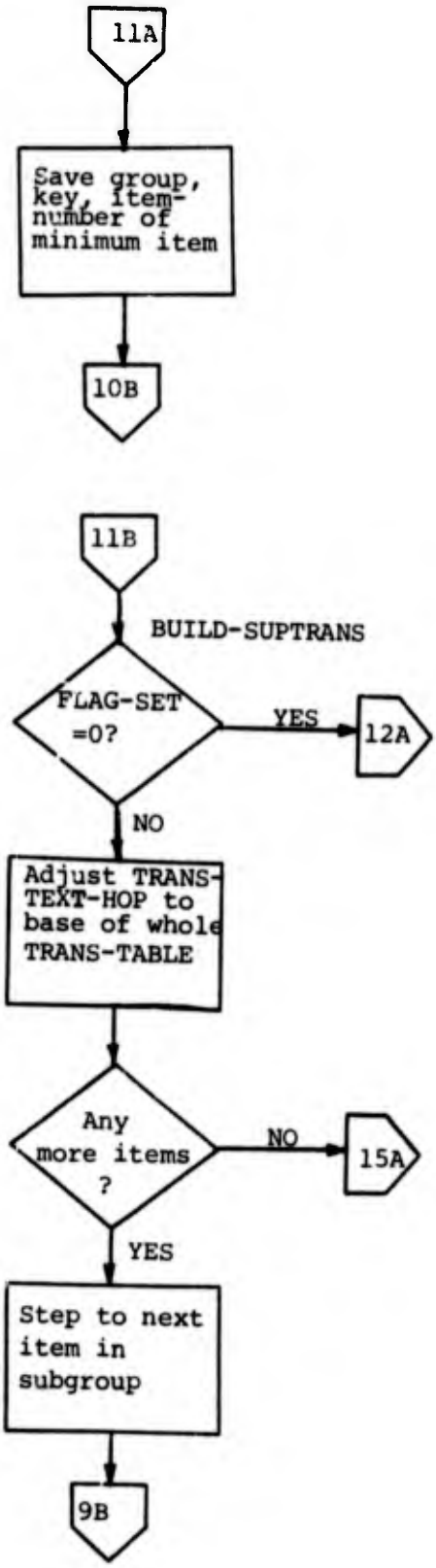

8A
NO-MORE-TEXT
More groups for this Rec-ID?
YES
NO
Initialize for next group
7A
9A

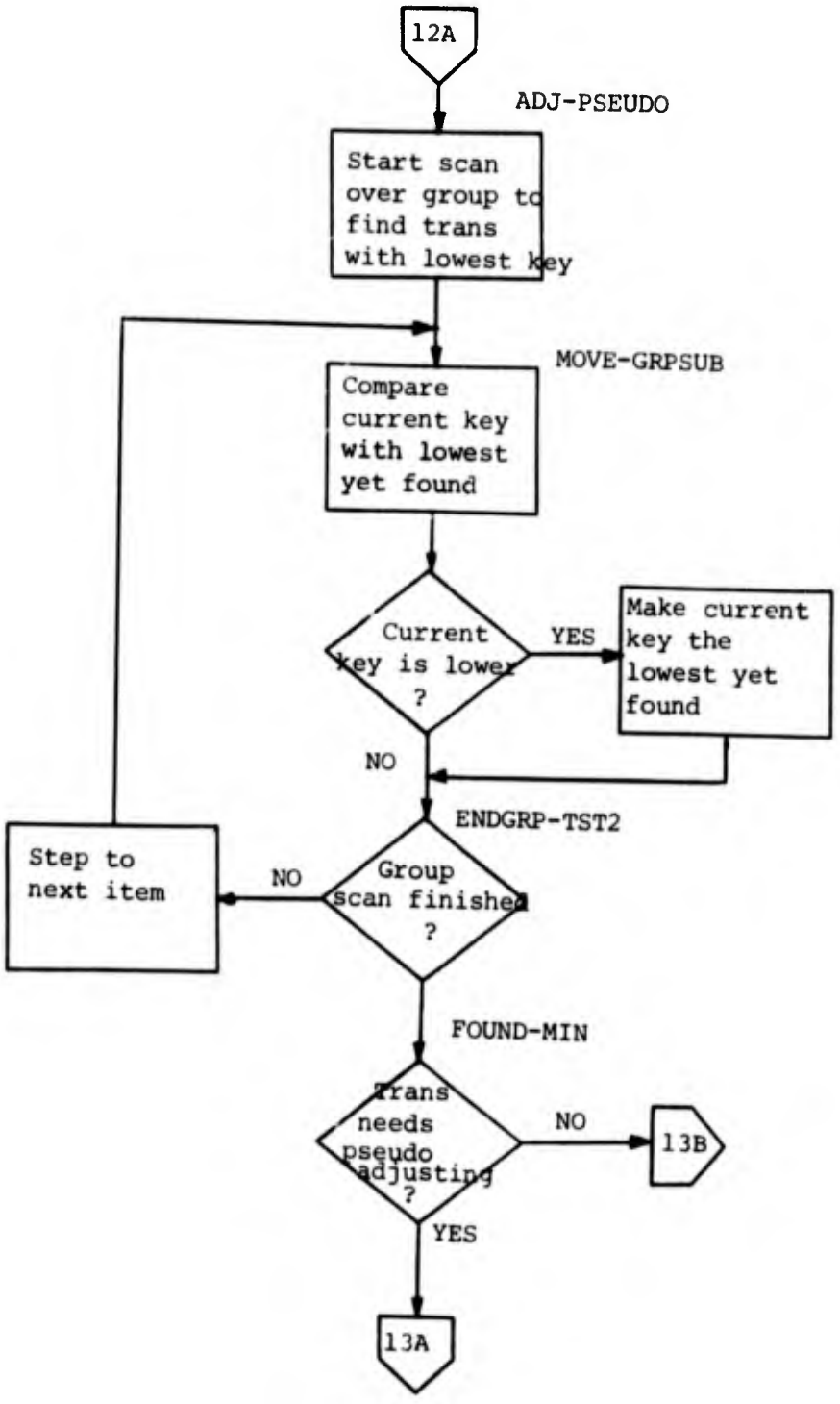

8B
BLD-HLD-TRANS
Any trans items to be moved?
NO
13C
YES
8C
MOVE-ITEMS
Move items from input area to trans-table
15C

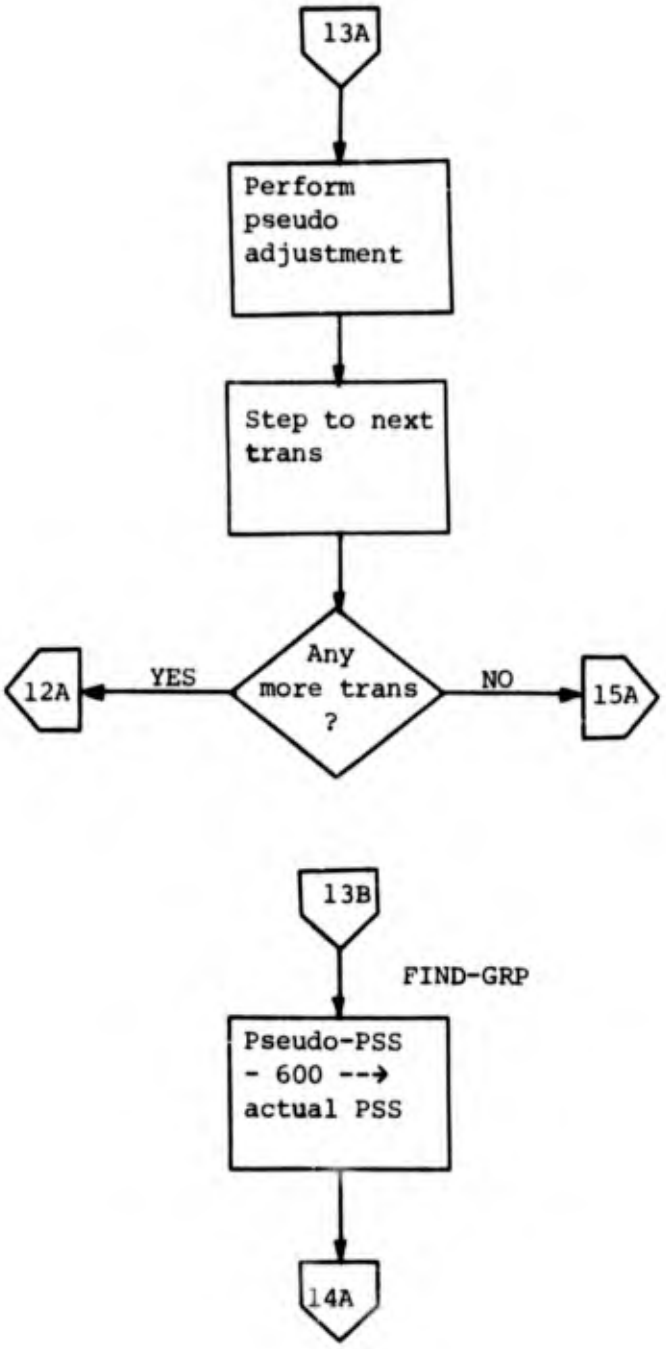

9A
PRE-MERGE
Initialize TRANS-LIST
9B
START-MERGE
Start merge of groups
9C
GETNEX-GRP
Compare keys of current & previous items
Current key is low ?
YES
10B
NO
Pseudo trans?
NO
10C
YES
Vset trans ?
YES
10C
NO
10A

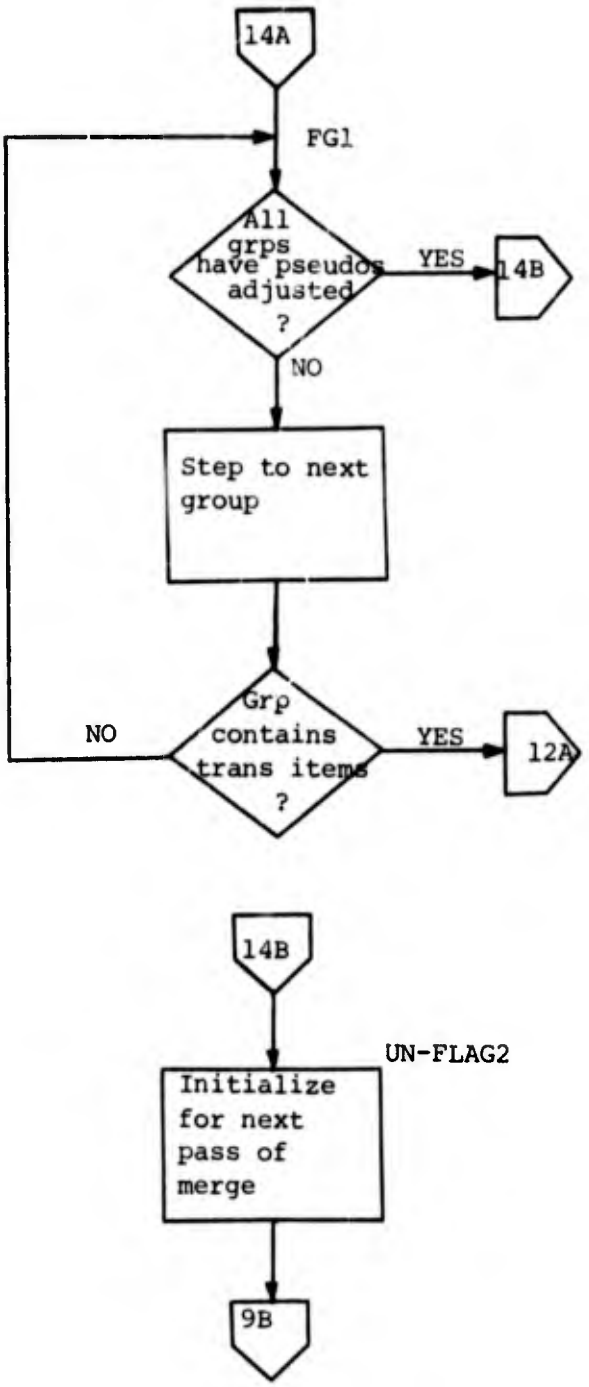

10A
Trans starting new set?
YES
Set number of trans FLAG-SET
11A
10B
NO
ENDGRP-TST
Any more groups ?
NO
11B
YES
Step scan to next group
9C

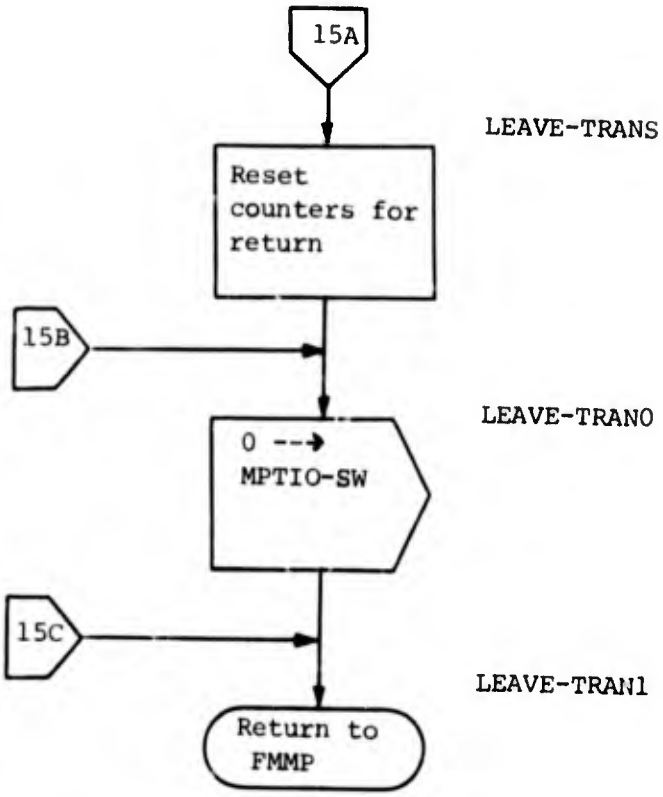

10C
UN-FLAG
0 --> FLAG-SET
11A

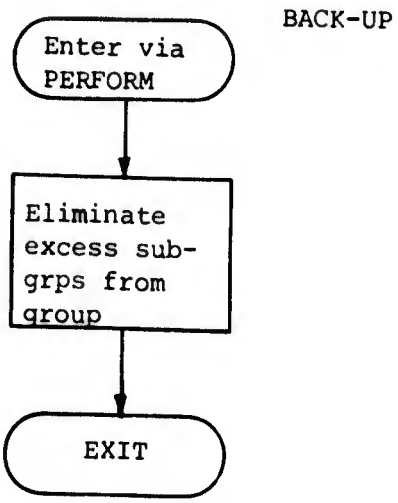

11A
Save group, key, item-number of minimum item
10B
11B
BUILD-SUPTRANS
FLAG-SET =0?
YES
12A
NO
Adjust TRANS-TEXT-HOP to base of whole TRANS-TABLE
Any more items ?
NO
15A
YES
Step to next item in subgroup
9B


12A
ADJ-PSEUDO
Start scan over group to find trans with lowest key
MOVE-GRPSUB
Compare current key with lowest yet found
Current key is lower ?
YES
Make current key the lowest yet found
NO
ENDGRP-TST2
Group scan finished ?
NO
Step to next item
FOUND-MIN
Trans needs pseudo adjusting ?
NO
13B
YES
13A


13A
Perform pseudo adjustment
Step to next trans
Any more trans ?
YES
12A
NO
15A
13B
FIND-GRP
Pseudo-PSS - 600 --> actual PSS
14A


14A
FG1
All grps have pseudos adjusted ?
YES
14B
NO
Step to next group
Grp contains trans items ?
NO
YES
12A
14B
UN-FLAG2
Initialize for next pass of merge
9B


15A
LEAVE-TRANS
Reset counters for return
15B
LEAVE-TRAN0
0 --> MPTIO-SW
15C
LEAVE-TRAN1
Return to FMMP


BACK-UP
Enter via PERFORM
Eliminate excess sub-grps from group
EXIT

g. LMMPX

(1) Function. To execute Logic Package within Maintenance Proper.

(2) Calling Sequence.

```
ENTRY 'LMMP'     USING MP-DD INPUT-RECORD MAJOR-RECORD
                       FMIO DEFINE-AREA
CALL  'BRANCH'   USING PKG-ADDR LMMP-DD
                       MAJOR-RECORD INPUT-RECORD
                       FMIO DEFINE-CHAR (DEF-POINT).
```

(3) Program Description.

LMMPX.

Is loaded by FMMPX through the LMMP entry point to dynamically load and execute Logic Packages in round-robin fashion. Parameters needed by the Logic Packages are taken from areas within the Linkage Section of LMMPX. Of the six parameters passed to the LINK subroutine, the last five are actually given to the Logic Package.

The six parameters are described as follows:

PKG-ADDR.

Is the address of the Logic Package load module to be loaded and executed.

LMMP-DD.

Contains those switches passed between Logic Packages and between calls to the same Logic Package.

MAJOR-RECORD.

Contains the work area for the record being maintained.

INPUT-RECORD.

Contains the original record being updated. For single file Logic Packages, it is treated as a minor file. For file-to-file Logic Packages, it is used only for the RELOAD operation.

FMIO.

Is passed to Logic Packages so that the FMIO load module will be available for the read card or print line capabilities of LM. The value of the entry point for FMIO is placed in the CALLIO subroutine by the Logic Package.

DEFINE-CHAR (DEF-POINT).

Specifies the high order position of the define area for the Logic Package so that values can be saved between uses of the Logic Package even it must be overlaid. Detailed fields in the define area appear only in the Logic Package.

(4) Limitations. None.

h. MPSRTX

(1) Function. To call FMMPSRT and FMMPMRG when a Maintenance Proper Sort is required.

(2) Calling Sequence.

```
ENTRY  'XMPSRT'   USING FM-DD.
CALL   'FMSET'    USING FMIO.
CALL   'FMMPSRT'.
CALL   'FMPMRG'   USING FM-DD.
```

(3) Program Description.

MPSRTX.

Is called by FM through the entry point XMPSRT after FMMPX discovers that out-of-sort records were to be written upon the output MIDMS file. MPSRTX is the first program executed in the FMMPSRT load module. The only processing in MPSRTX are calls. FMSET sets up the entry point of FMIO in the CALLIO subroutine so that the FMMPSRT subprogram can print using FMPRT. FMMPSRT sorts the key file pointing to the out-of-sort records. MPMRG merges the out-of-sort records with the main MIDMS file.

(4) Limitations. None.

i. FMMPSRT

(1) Function. To sort the key file to the out-of-sort MIDMS file records.

(2) Calling Sequence. None. FMMPSRT is entered directly through its PROGRAM-ID name with no parameters.

(3) Program Description.

FMMPSRT.

Uses a COBOL SORT to order the keys to the out-of-sort records. The SORT Input Procedure consists of reading the key file records and releasing them to the sort. The Output Procedure consists of writing out records returned from the sort.

(4) Limitations. None.

j. FMMPMRG

(1) Function. To merge out-of-sort records with the main MIDMS output file.

(2) Calling Sequence.

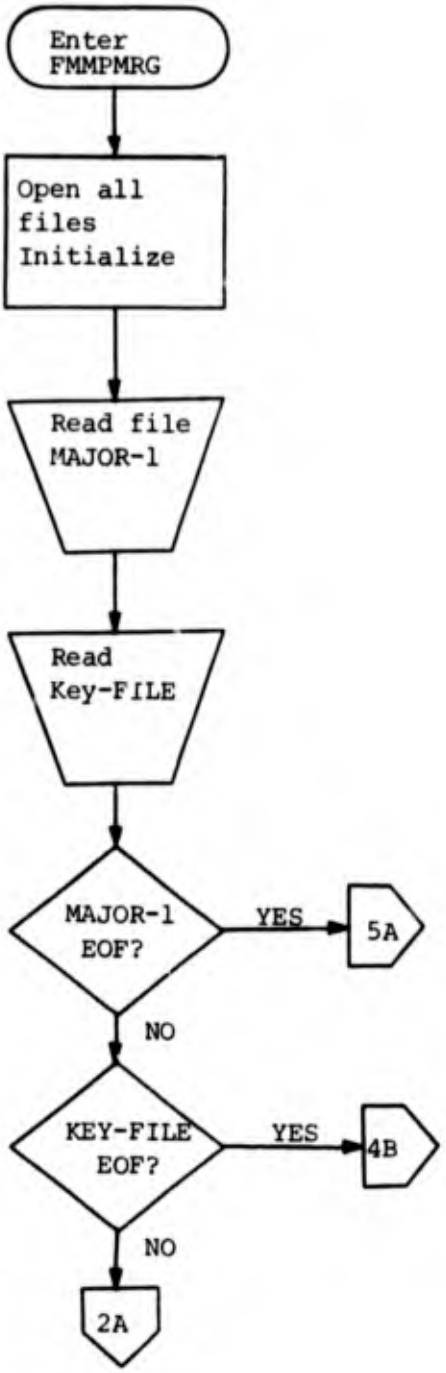
```
ENTRY 'MPMRG'     USING FM-DD.
CALL  'MOVARAYS' USING ...
CALL  'COMARAYS' USING ...
CALL  'MUVE'      USING ...
CALL  'FMPRT'     USING PRT-LINE  CC.
```

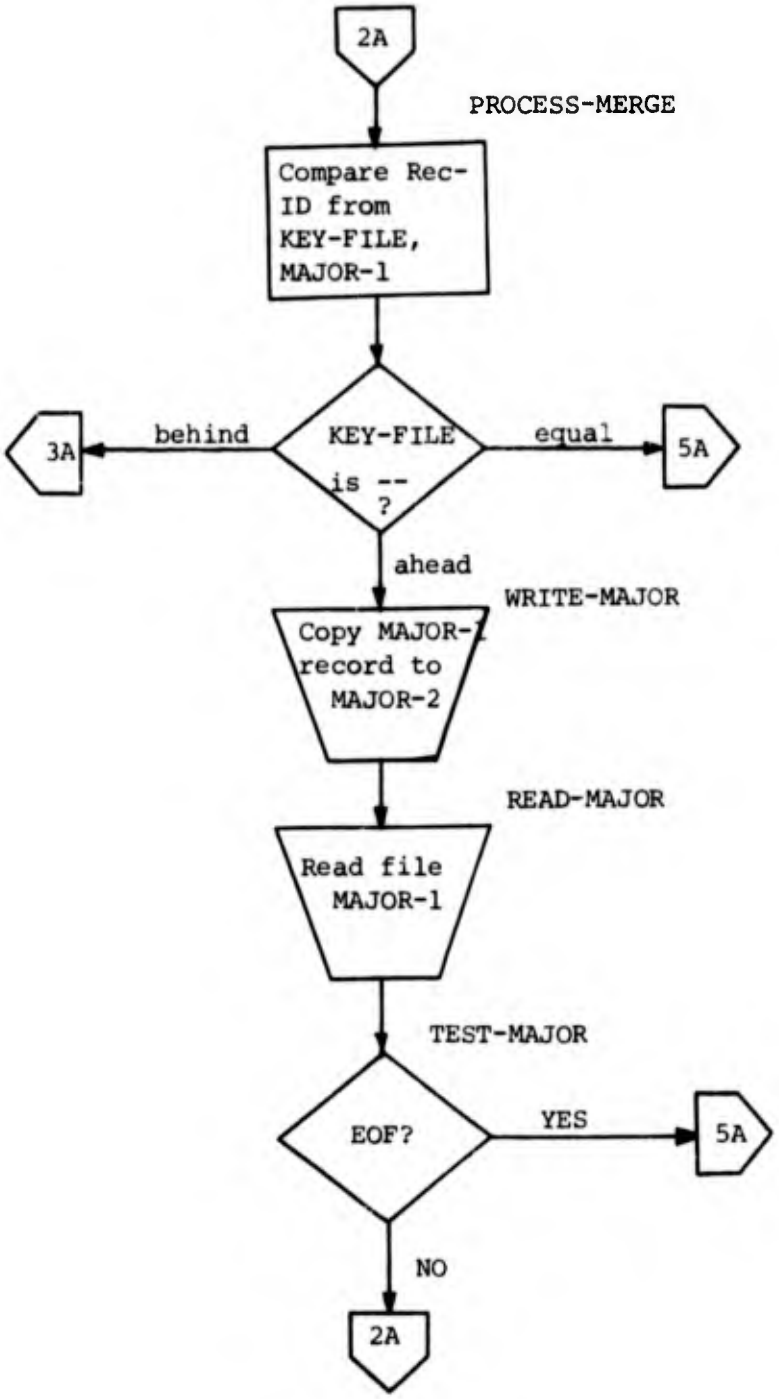
(3) Program Description.

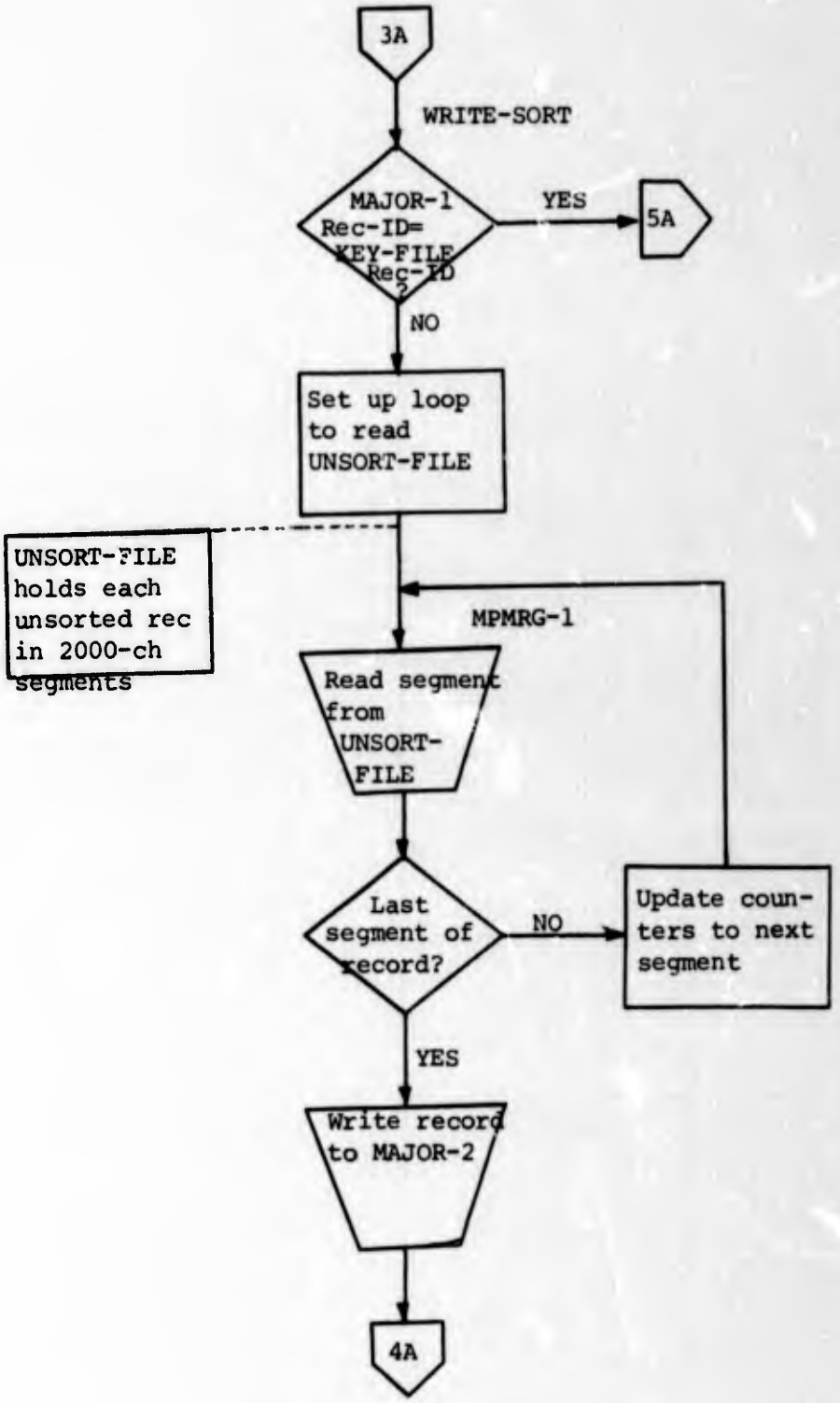
FMMPMRG.

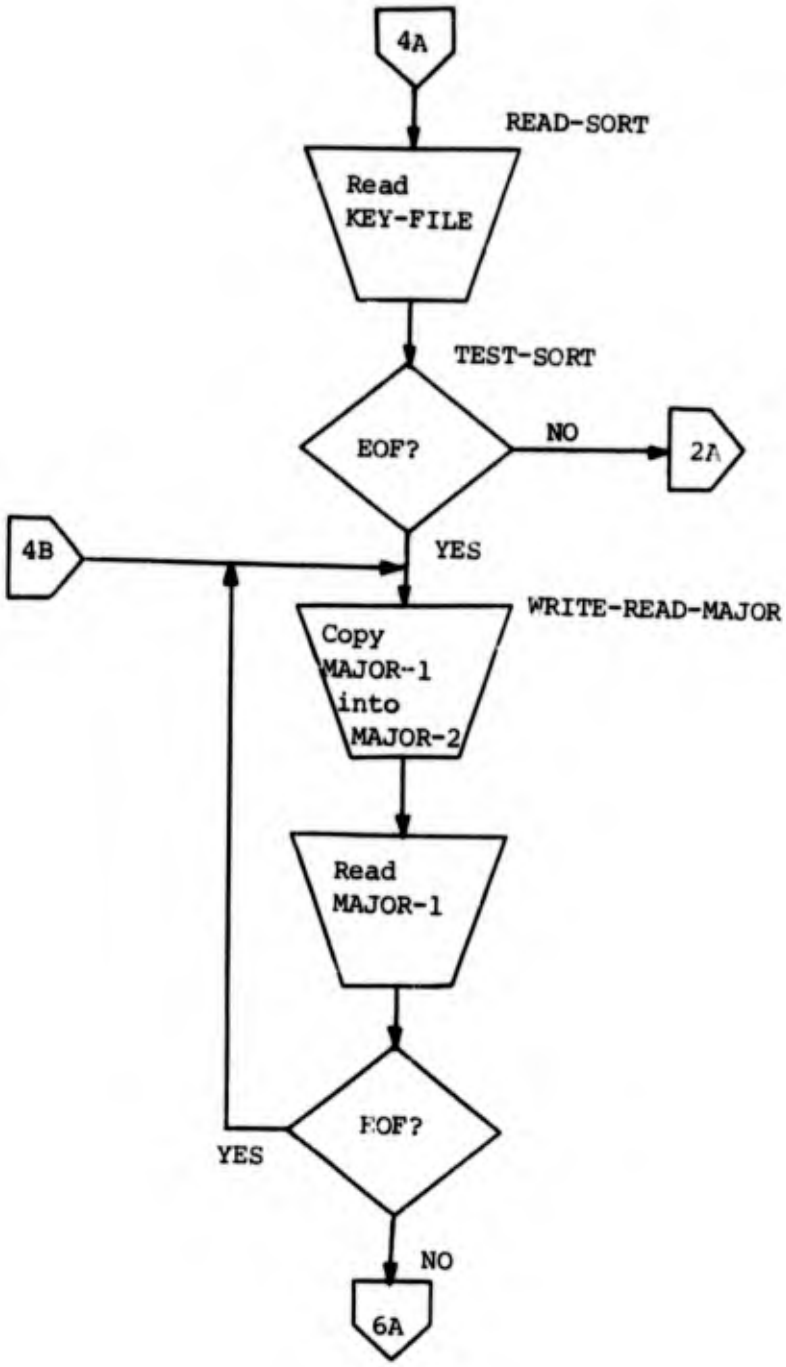
Is called by MPSRTX through the entry point MPMRG. After entering this subprogram in the IBM 360, MOVARAYS is called twice to clear the input and output DCB LRECL fields to 0 so that MIDMS records can be blocked. The processing of FMMPMRG reads the MP output file and the sorted key file determining lower record ID's for each record. Equal ID's are rejected as duplicates and the sort file records are eliminated from the final output file with a message.

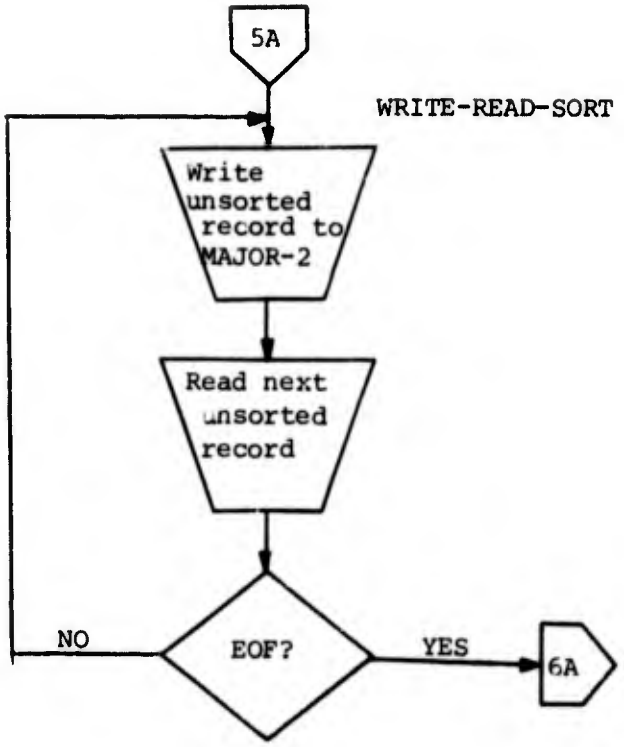
(4) Limitations. None.


Enter
FMMPMRG
Open all
files
Initialize
Read file
MAJOR-1
Read
Key-FILE
MAJOR-1
EOF?
YES
5A
NO
KEY-FILE
EOF?
YES
4B
NO
2A

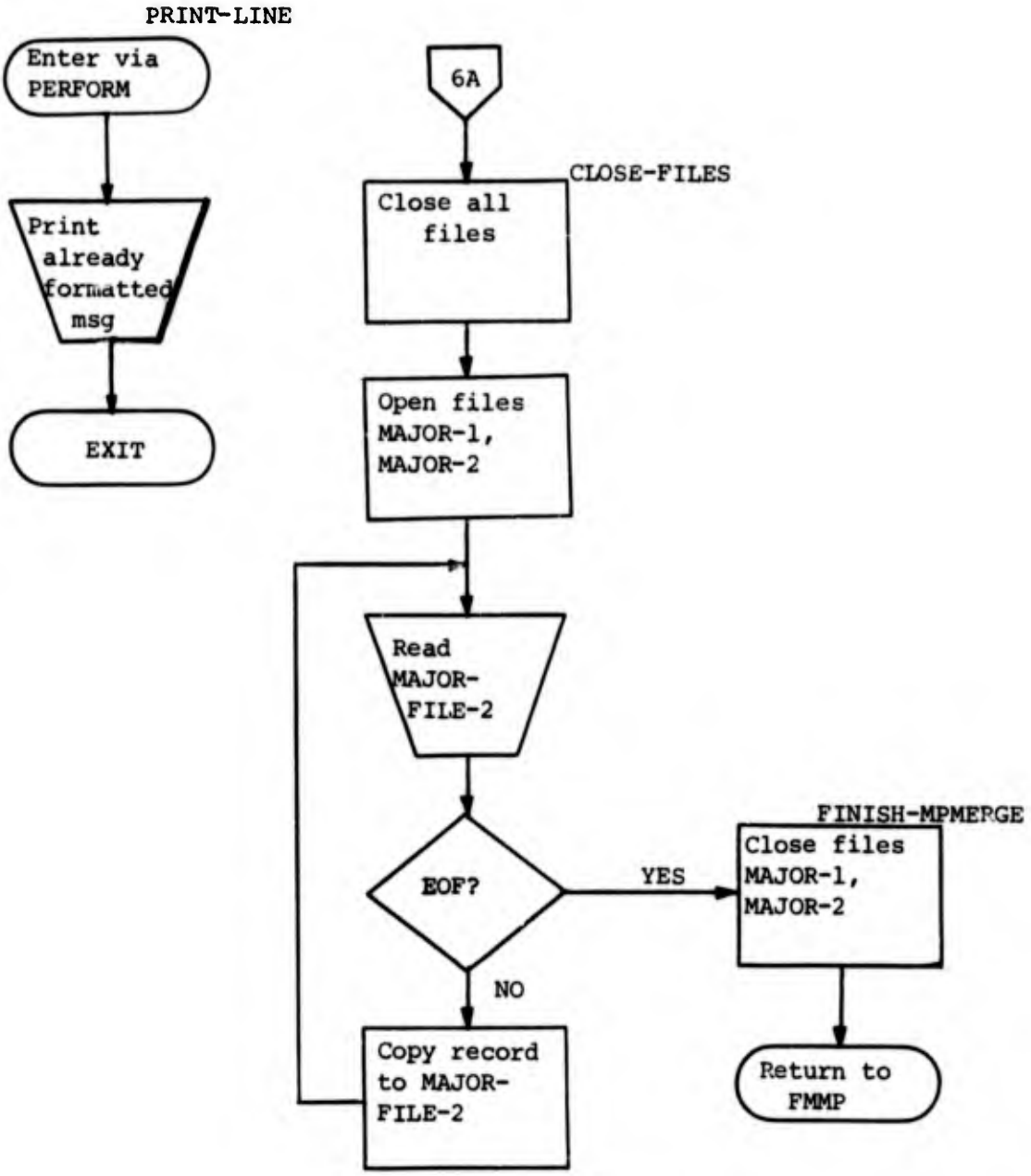

2A
PROCESS-MERGE
Compare Rec-ID from KEY-FILE, MAJOR-1
KEY-FILE is -- ?
behind
3A
equal
5A
ahead
WRITE-MAJOR
Copy MAJOR-1 record to MAJOR-2
READ-MAJOR
Read file MAJOR-1
TEST-MAJOR
EOF?
YES
5A
NO
2A


3A
WRITE-SORT
MAJOR-1 Rec-ID= KEY-FILE Rec-ID 2
YES
5A
NO
Set up loop to read UNSORT-FILE
UNSORT-FILE holds each unsorted rec in 2000-ch segments
MPMRG-1
Read segment from UNSORT-FILE
Last segment of record?
NO
Update counters to next segment
YES
Write record to MAJOR-2
4A


4A
READ-SORT
Read
KEY-FILE
TEST-SORT
EOF?
NO
2A
4B
YES
WRITE-READ-MAJOR
Copy
MAJOR-1
into
MAJOR-2
Read
MAJOR-1
EOF?
YES
NO
6A


5A
WRITE-READ-SORT
Write unsorted record to MAJOR-2
Read next unsorted record
EOF?
NO
YES
6A


5B
Print error msg
4A


PRINT-LINE
Enter via PERFORM
Print already formatted msg
EXIT
6A
CLOSE-FILES
Close all files
Open files MAJOR-1, MAJOR-2
Read MAJOR-FILE-2
EOF?
YES
NO
FINISH-MPMERGE
Close files MAJOR-1, MAJOR-2
Return to FMMP
Copy record to MAJOR-FILE-2

k. OMOVL

(1) Function. This load module is comprised of three subprograms (OMMPDRV, OMMPX1, OMMPX2). They perform the same function as OMMPX. In the overlay format 6K less is required.

(2) Calling Sequence.

```
ENTRY  'OMOVL'     USING MP-DD INPUT-RECORD
                         MAJOR-RECORD MAJOR-LR2 MAJOR-LR3
                         TRANS-TABLE FM-DD
CALL   'OMMP1'     USING MP-DD INPUT-RECORD
                         MAJOR-RECORD MAJOR-LR2 MAJOR-LR3
                         TRANS-TABLE FM-DD OMMP-DDV.
CALL   'OMMP2'     USING MP-DD INPUT-RECORD
                         TRANS-TABLE FM-DD OMMP-DDV.
```

(a) OMMPX1 and OMMPX2 are entered with the same parameters as they are called with except receive OMMP-DD instead OMMP-DDV.

(b) The same ALC subroutines are used as in OMMPX.

(3) Program Descriptions.

The same capabilities are present in the OMOVL load module as exist in the OMMPX load module. Paragraph names, functions and methodology are identical, except for the situations which cause overlaying. In these cases OMMPDRV-SW is set; the processing program (either OMMPX1 or OMMPX2) returns to OMMPDRV which will call the appropriate program depending upon the setting of OMMPDRV-SW. Upon entry into the subprogram the desired paragraph is reached depending upon OMMPDRV-SW.

OMMPDRV.

Is just a supervisor which controls the calling of the two processing subprograms.

OMMPX1.

Handles all the ordinary processing and will transfer control via OMMPDRV to OMMPX2 on the following conditions:

A detected error.
Variable set processing.
Arithmetic operation.
For wrap up processing of the record.

When using the overlay version there will be at least one overlay for each MIDMS record for which there is a transaction. Additional overlays will be required whenever any of the first three conditions as described above occur.

(4) Limitations. None.

3. FILE MAINTENANCE CONFIRMATIONS AND ERROR MESSAGES.

a. Confirmations.

(1) Whenever confirmation of any field is requested, a confirmation header is produced for every record against which transactions are issued. The header is in the form of:

CONFIRMATION OF RECORD ***record id***

(2) Following this, each field which had a valid transaction performed against it and for which confirmation is desired is printed out with the supporting information present when appropriate in the following format:

1 - 5	7 - 15	17 - 19	21 - 23	28 - 31	33 - 75	77 - 78	80 - 123
field-name	subset or partial range on Vset	OPR	field-type	'FROM'	original contents	'TO'	present contents

(3) After all field confirmations have been printed a confirmation trailer will be printed provided that there were valid transactions carried out against this record. This will be in the form of:

CONFIRMATION OF RECORD ***record id*** GENERATED
UPDATED
DELETED

NOTE: Confirmation for a variable set is done only after all transactions for that VSET have been processed. Therefore, the opcode that appears in the OPR position of the confirmation will be the last OPR against the VSET. The FROM contents are those in the original record and the TO contents represents the end result of all the transactions applied against the specific VSET. In addition, if any field exceed 43 characters as defined in the FFT (except in the variable set where the exact number of characters present is used), multiple FROM and TO lines will be produced to accommodate the maximum size of either field.

b. Error Diagnostics.

(1) The error diagnostics are designed to provide the user with enough information so that he may identify the exact transaction that was found to be in error as well as the reason for its rejection. The messages are in a three line format as follows:

```
OMMP ERROR error message      ***ERROR***   OPR IS opr
**RECID** record id           field-name    PSS IS pssq
DATA   data (up to first 120 characters)
```

(2) The following are the specific error messages giving the error message and an explanation of its probable cause:

ERROR MESSAGE	EXPLANATION
Ø1 ARITHMETIC OVERFLOW	The result of an arithmetic operation exceeded the size of the FFT field.
Ø2 DATA NOT NUMERIC	Either an arithmetic operation or a change was attempted on a numerically defined FFT field and the input data was not numeric.
Ø3 FIELD TYPE NOT NUMERIC	An arithmetic operation is being attempted on an FFT field not defined as numeric.
Ø4 GEC ILLEGAL FOLLOWING GEN	Illegal sequence for variable set operators, use GEC.
Ø5 GEC OPR CANNOT CHANGE EXISTING RECID	GEC cannot change the ID of an existing record.
Ø6 GEC OPR ILLEGAL - RECORD PRESENT	An attempt was made to generate a record that is present.
Ø7 GEN OPR ILLEGAL - VSET EXISTS	Self-explanatory.
Ø8 ILLEGAL OPR SET NOT PRESENT	A delete PSC or VSC was attempted on a set containing no data.
Ø9 ILLEGAL SUBSET GEN, SUBSET IS PRESENT	An attempt was made to generate a subset which already existed.
1Ø INVALID PARTIAL VSET CHANGE	(1) Either the change range exceeds the original vset size or (2) there has already been a transaction against this vset.
11 NEGATIVE NUMERIC FIELD ILLEGAL	The result of an arithmetic operation was negative and the FFT field was not defined as SGNUM.

	ERROR MESSAGE	EXPLANATION
12	NEW RECORD EXCEEDS MAX RECORD LENGTH	While building a record the maximum record size, as specified in the JCL, was exceeded.
13	OLD RECORD RESTORED	If, while building a new record or updating an existing record the maximum size record was exceeded, all updates to the record are canceled.
14	ONLY ATC/ATV CAN FOLLOW ATC/ATV	Self-explanatory.
15	VSET NOT PRESENT	An attempt was made to replace an empty Vset.
16	OPR ILLEGAL, OLD SUBSET IS NOT PRESENT	An attempt was made to change a nonexistent subset.
17	OPR ILLEGAL WHEN RECORD NOT PRESENT	The opr specified an action requiring an old record.
18	PARTIAL CHANGE CAN BE ONLY TRAN ON VSET	Self-explanatory.
19	PSSQ CANNOT EXCEED 599	An attempt was made to assign an actual subset number greater than 599 from a pseudo subset number.
20	RECCT CANNOT BE UPDATED	An operation was attempted against the RECCT.
21	RECORD SIZE EXCEEDS MAXIMUM	Whole record operation data exceeds the maximum record size as specified in JCL.
22	TRANSACTION ILLEGAL WITH WHOLE RECORD OPR	Whole record transactions must be the first and only transactions on a record.
23	WHOLE RECORD OPR MUST BE FIRST TRANS	Whole record operations must be the first and only transaction on a the record.

CHAPTER 3

SECTION V

Logical Maintenance

1. SUBSYSTEM STRUCTURE.

a. LMLP. This program controls the processing for LM runs. It prints all logical maintenance cards through the LMLPPG1 and LMLPPG2 (which generate the initial portion of the generated COBOL program), LMLPSCN is called to read a card and build a word table composed of the words on the logic package card. Upon return from LMLPSCN an operation lookup is executed to determine what subprogram is needed for the particular operation indicated on the card. The appropriate subprogram is called where the necessary COBOL code is generated. Each logic package card is read and processed independently.

b. LMLPASN. This subroutine generates COBOL instructions to assign index numbers to periodic sets.

c. LMLPATC. This subroutine generates COBOL to attach data to a variable set.

d. LMLPBLD. This subroutine generates COBOL instructions to build a subset.

e. LMLPCNG. This program controls the processing of CHANGE and DELETE field operations. LMLPFMT1 is utilized to verify the field names. COBOL code is generated depending on the type of field or fields.

f. LMLPDDS. This program takes the COBOL Data Division generated by FS and writes it out on DATAOUT or LINKOUT in the case of single file. The fixed set is written whenever a MAJOR or MINOR file is used on the FMLP control card. This is done from LMLPPG1 or LMLPPG2 depending on the run type. The Data Division for each periodic set is written only if the set is referenced by a BUILD statement. This is done from LMLPWP2.

g. LMLPDEF. When the keyword DEFINE is found, control is passed here. The define word is put into the define table and the appropriate COBOL Data Division code is generated.

h. LMLPDEL. This subroutine generates COBOL coding to delete a record, periodic set, or periodic subset.

i. LMLPFLD. This subroutine is used by the other processing subroutines to qualify major and minor fixed and periodic field names and defines. It also builds COBOL Data Division statements whenever a partial field is specified.

j. LMLPFMT (LMLPFMT1). This program is used to verify all field names by the routines subordinate to LMLPMOV, LMLPSUB, LMLPCNG. Two copies of this program are used.

k. LMLPGEN. This subroutine produces the Data Division Code, Linkage Section Code, and Procedure Division Code determined necessary by the other functional processing routines.

l. LMLPGRP. If DEFINE GROUP is indicated, this subroutine generates the appropriate COBOL Data Division code and remains in control until an ENDGRP card is encountered.

m. LMLPMLT. If a MULTIPLY or DIVIDE is indicated, this subroutine generates the appropriate Procedure Division COBOL coding.

n. LMLPMOV. This program is the driver for all processing of all MOVE/PUT statements. It selectively calls LMLPPUT, LMLPNM0, LMLPNM1, LMLPNM2, LMLPNM3, LMLPZN0, LMLPZN1, LMLPZN2, LMLPZN3.

o. LMLPNM0. This program controls the processing for MOVE NUM or MOVE NUMERICS statements. The purpose is to replace the magnitude of field 2 by that of field 1 without changing the sign of field 2. A further breakdown for processing is determined according to the field types.

(1) LMLPNM1. In case of an alphanumeric receiving field.

(2) LMLPNM2. In case of a numeric defined receiving field.

(3) LMLPNM3. In the case of numeric receiving field.

Each of these generate the code appropriate for the particular type of receiving field.

p. LMLPPG1, LMLPPG2. These routines are used once each per logic package and together generate the IDENTIFICATION DIVISION, ENVIRONMENT DIVISION, the fixed portion of the WORKING-STORAGE SECTION (which is needed for every generated program), the LINKAGE SECTION (when the type of run needs one), and the initial part of the PROCEDURE DIVISION. The presence of two routines to do the task is to facilitate operating in less core. Control is passed to LMLPPG1 and then LMLPPG2 sequentially from LMLP.

q. LMLPPRT. This subroutine generates the appropriate code for a PRINT or XPRINT operation.

r. LMLPPUT. This program processes the key word PUT and generates appropriate code depending on the field types.

s. LMLPRFD. This program isolates the two fields and verifies them.

t. LMLPRLN. This program verifies the combination of relational words and sets the relational switches to indicate the standard relation to be used.

u. LMLPRMN. This program contains the three entry points (IF, AND, OR) and controls the processing for conditional statements. It initializes the areas used by the subordinate routines.

v. LMLPRMS. This program checks for operations of (NOT) IN RANGE, (NOT) IN TABLE, DELETED RECORD, FIND RECORD, GENERATED RECORD, EOF, and generates the appropriate COBOL code.

w. LMLPRST. This program processes LM statements of the following types, checking for valid statements, and generating COBOL input:

READ
WRITE
SAVE
SCRAP
STOP
RELOAD
PUNCH
CONVERT
STOP SFL

Each of these verbs is assigned a specific entry point. Thus, to process "READ", the program LMLP executes a call to entry-point LMREAD.

(1) READ. Reads in a record from the major file, minor file, or card reader, tape, as indicated by the user.

(2) WRITE. Writes a record on the major file.

(3) SAVE. Saves the contents of the MP work area so that it will be available for further processing after the area is used for a WRITE.

(4) SCRAP. Abandons the building of a major file record, so that it will not be written out.

(5) STOP. In single file LM, the STOP statement causes the processing of a record to be terminated, and that record to be written out on the major file. Control is then passed to MP supervisor, which reads another record and gives control to the first statement in the package. In file-to-file LM, the STOP statement causes all files online to be closed.

(6) RELOAD. Restores the major file record to its input state, cancelling all updating previously done to that record.

(7) PUNCH. Punches a card from a card-image area defined by the user.

(8) CONVERT. Takes data from one operand, calls a user-subroutine to process it, stores the result into another operand, and passes control to a failure exit if the subroutine indicates that an error was detected.

(9) STOP SFL. Terminates any further use of a single file logic package. When used in file-to-file LM, it acts exactly like the STOP instruction.

x. LMLPRT1, LMLPRT2. These programs generate the output for the condition FLDA relation FLDB format. Depending on the combination of fields found in LMLPRFD either LMLPRT1 or LMLPRT2 is used. LMLPRT2 is used only when the first field is a numeric data field.

y. LMLPSCN. This program reads each card by use of FMCRD then scans each card, builds the words (word terminators are blanks or a comma) and places them in FM-WORD-AREA. Next, it examines the second word to determine if it is DEFINE and if so, checks the third word for GROUP. Appropriate switches are set to notify LMLP what operation is indicated.

z. LMLPSP0. This program is called to check for BIT, SWITCH, and PROFILE operations. If found, validity checking is done for the particular operation and code generated for SWITCH. PROFILE and BIT require field validation through LMLPRFD and then call LMLPSP1 for code generation. If none of the preceding operations were found upon return from LMLPSP0, the only valid operation remaining must be in the form of: condition FLDA relation FLDB.

aa. LMLPSP1. Called by LMLPSP0 for COBOL code generation.

ab. LMLPSTE. This subroutine generates COBOL coding for the STEP INDEX instruction.

ac. LMLPSUB. This program controls the processing for both ADD and SUBTRACT operations. LMLPFMT1 is used to verify the field names. Code is generated depending on the type of fields.

ad. LMLPTAB. This program verifies the card format and generates the appropriate code. If the RANGE operation was found in LMLPRMS, the field name is verified through LMLPRFD and the appropriate code is generated.

ae. LMLPVPT. This subroutine allows a specified variable set to be printed (displayed) in whole or part.

af. LMLPWP0, LMLPWP1, LMLPWP2, LMLPWP3, LMLPWP4. These subroutines generate the post-processing COBOL code by interpretation of tables which are built and switches which are set during the processing of the logic package. This post-processing is divided into multiple subroutines only for the purpose of the overlay structure and thus, to require less core. The generated code is needed by only once per logic package and is generated by these routines.

(1) LMLPWP0. This program verifies that for all FAILURE EXITS there exists corresponding LABELS. It also scans the BUILD table, STEP table, and DELETE table to insure that the proper set index combinations have been assigned.

(2) LMLPWP1, LMLPWP5. These programs generate the Procedure Division Code for BUILD and STEP statements by using the tables previously built. LMLPWP1 calls LMLPWP5.

(3) LMLPWP2. This program generates the Procedure Division Code for ATTACH statement by use of switches previously set. Generates code for expand-record, attach major-to-major, attach LIT-to-major, attach minor-to-major.

(4) LMLPWP3. This program generates the Procedure Division Code for DELETE statement by use of DELETE table previously built. Generates code for CONTRACT-RECORD, and all disassigns. (This latter function allows user to use the same index any number of times logically but only one combination will be active at any one time.)

(5) LMLPWP4. This program generates Data Division Code needed for user calling sequence, I/O areas and VPRINT. It also generates Procedure Division Code to open and close I/O files and to do the I/O functions.

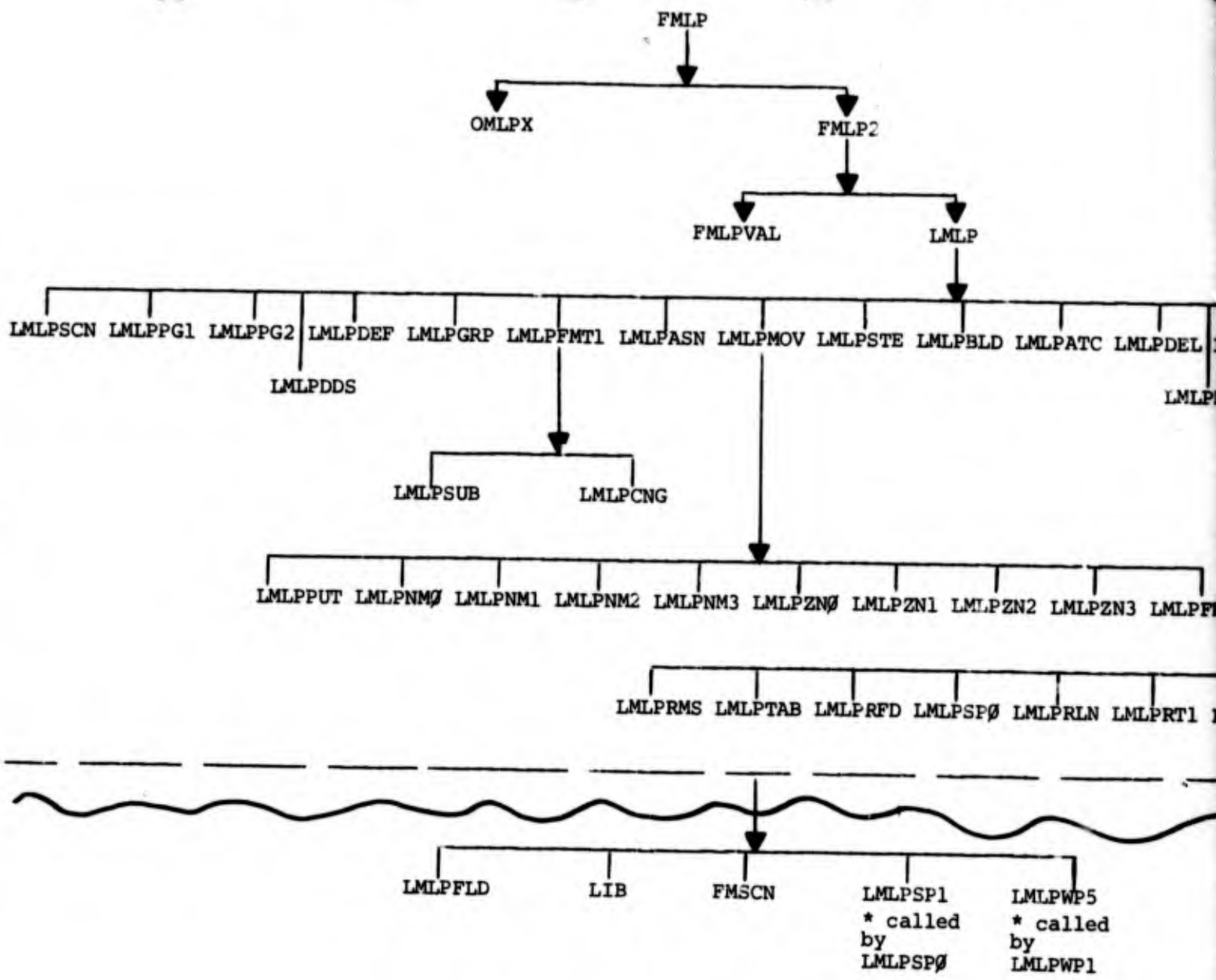
ag. LMLPZN0. This program controls the processing for MOVE ZON or MOVE ZONES statements. The purpose is to replace the sign of field 2 by that of field 1 without changing the magnitude of field 2. A further breakdown for processing is determined according to the field types.

(1) LMLPZN1. In the case of an alphanumeric receiving field.

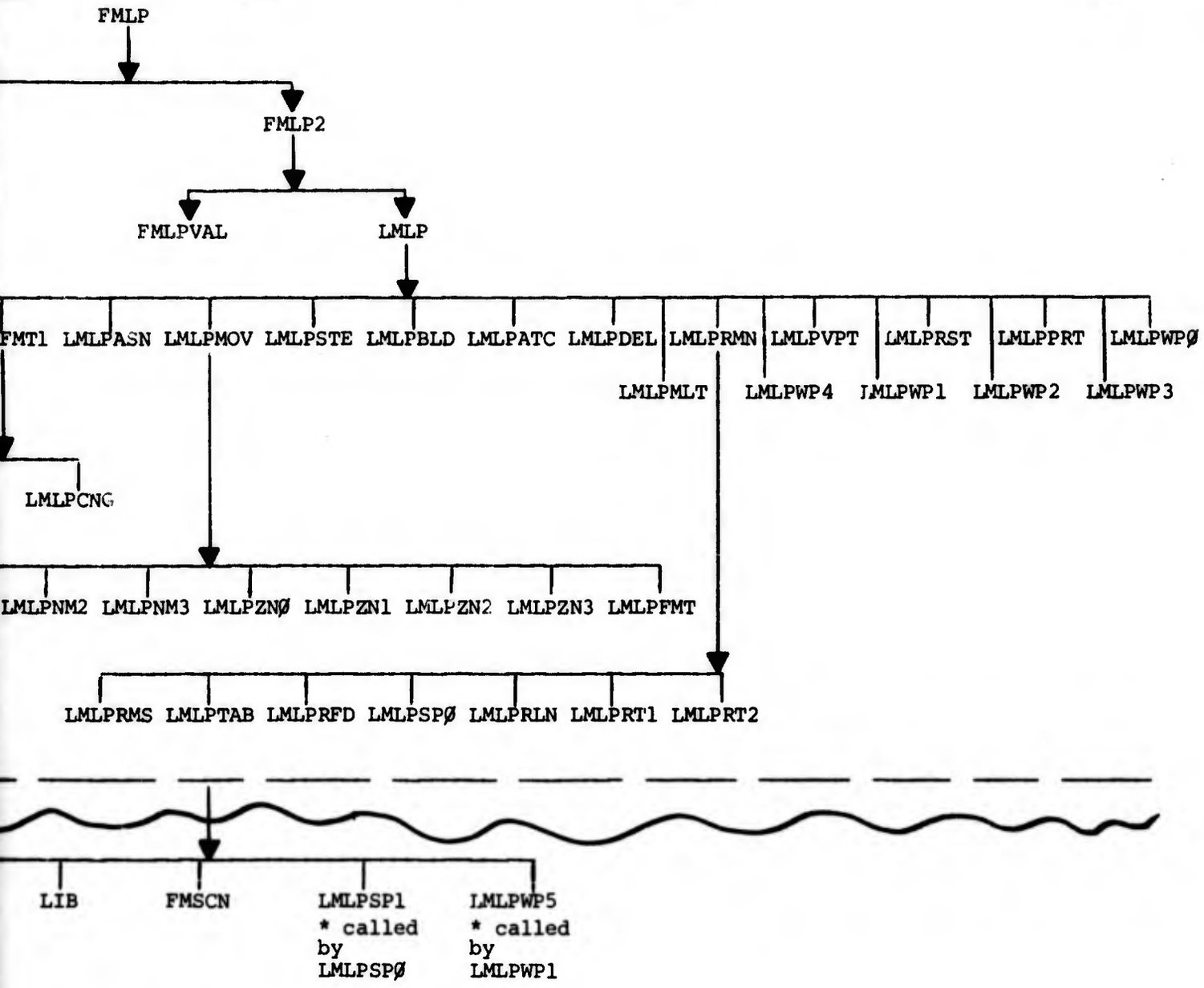
(2) LMLPZN2. In the case of a numeric defined receiving field.

(3) LMLPZN3. In the case of a numeric receiving field.

Each of these generate the code appropriate for the particular type of receiving field.



LMLP Overlay Structure



LMLP Overlay Structure

2. SUBPROGRAM IDENTIFICATION AND DESCRIPTION.

a. LMLP.

(1) Function. This program is the logical maintenance supervisor. It prints all input cards. It analyzes the FM-WORD-AREA by doing a table lookup for key second words. According to the second word, it calls the appropriate processing subprograms.

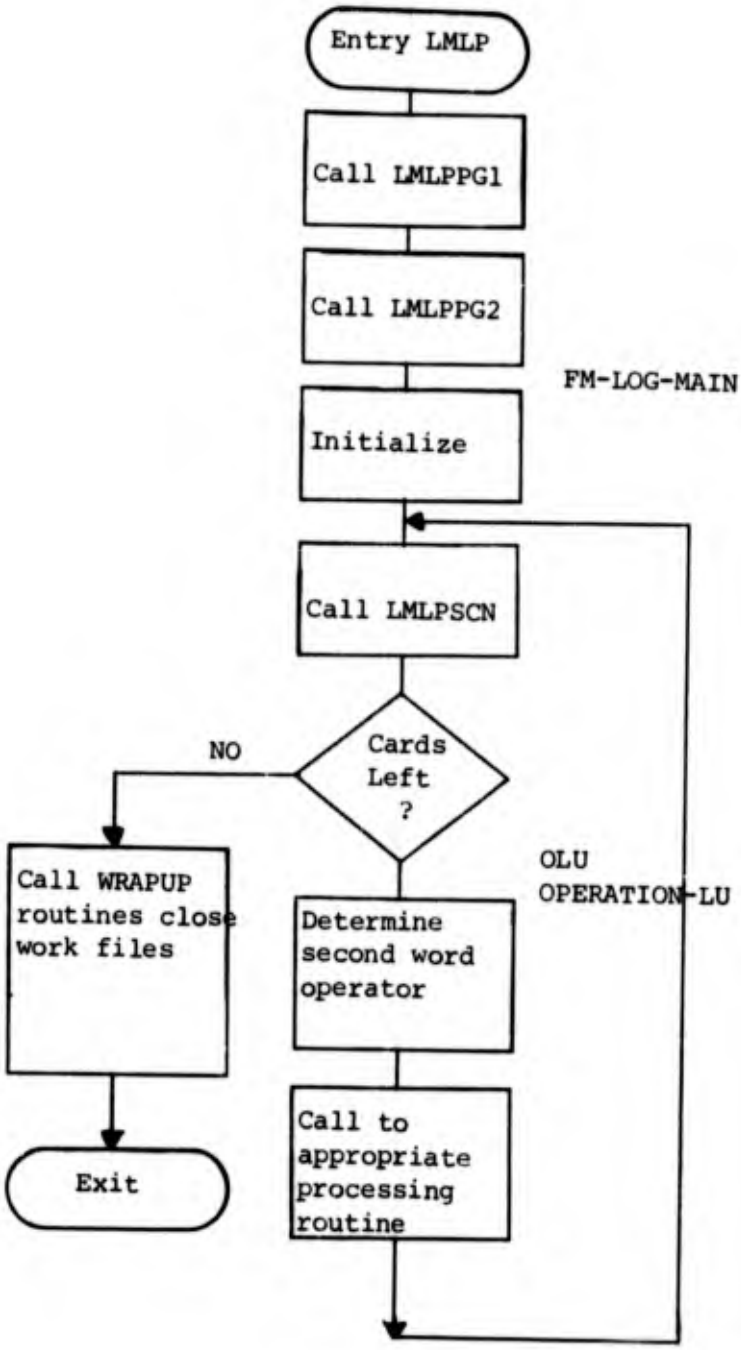
(2) Calling Sequence.

```
ENTRY   'LMLP'      USING FM-DD   LP-DD   LM-DD
CALL    'LMLPPG1'   USING LP-DD   LM-DD
CALL    'LMLPPG2'   USING LP-DD   LM-DD
CALL    'LMLPSCN'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPPRT'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPGRP'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPPFL'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPRDR'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPWRT'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPSCP'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPCVT'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPPCH'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPSAV'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPSTO'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPRLD'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPASN'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPSTE'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPBLD'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPATC'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPADD'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPSUB'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPMOV'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPCNG'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPRMN'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPDEF'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPRMN'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPRMN'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPVPT'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPDEL'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPMLT'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPWP0'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPWP1'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPWP2'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPWP3'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPWP4'   USING FM-DD   LP-DD   LM-DD
CALL    'LMLPGEN'   USING OUTPUT-LINE
CALL    'FMIO'      USING PRNT-CARDS    CC
CALL    'LMLPDDS'   USING FM-DD   LP-DD   LM-DD
```

(3) Program Description.

CALL-PG1.
CALL-PG2.
For generation of Identification Division, Environment Division, part of the Data Division, and part of the Procedure Division.
FM-LOG-MAIN.
Initialization of switches, counters, and hold areas.
RETURN-TO-LMSCN.
Calls LMSCN to build table of words from input logic package.
GENERATE-PRINT.
Determines if card is for comments, end-of-job, or if logic card has a label.
OLU.
OPERATION-LU.
Determines which processing subroutine should be called depending on card type.
STOPIT.
Call wrap up procedures.
SKIP-WRAPUPS.
Call LMCLO to close three work files (Dataout, Linkout, Procout) and return.

(4) Limitations. Processes one card at a time.


Entry LMLP
Call LMLPPG1
Call LMLPPG2
FM-LOG-MAIN
Initialize
Call LMLPSCN
Cards Left ?
NO
Call WRAPUP routines close work files
Exit
OLU OPERATION-LU
Determine second word operator
Call to appropriate processing routine

b. LMLPASN

(1) Function. Handles the ASSIGN instruction.

(2) Calling Sequence.

```
ENTRY 'LMASN'  USING FM-DD  LP-DD  LM-DD
CALL  'LMPRO'  USING CARD-IMAGE
CALL  'FMPRT'  USING PRNT-CARDS  CC
```

(3) Program Description.

GENERATE-ASSIGN.

This paragraph performs the following error checking on the fields of the ASSIGN card:

The first word is a label, it is counted as field one whether or not it is present.

The second word must be equal to 'ASSIGN'.

The third word must be equal to 'INDEX'.

The fourth word must be a two digit index number, whose value is not zero.

The fifth word must be equal to 'TO'.

The sixth word must be one of the following: PSETmm, VSETmm, $PSETmm, or $VSETmm. The set number mm must be two digits, not equal to zero, and must have been defined in the FFT as a periodic or variable set.

The seventh word must be a period.

The eighth word must be a failure exit. No checking is performed here.

GENERATE-ASSIGN-CODING and GEN-MOVE.

The paragraphs generate the in-line coding for the ASSIGN procedure.

INDEX-INFO-LOOKUP.

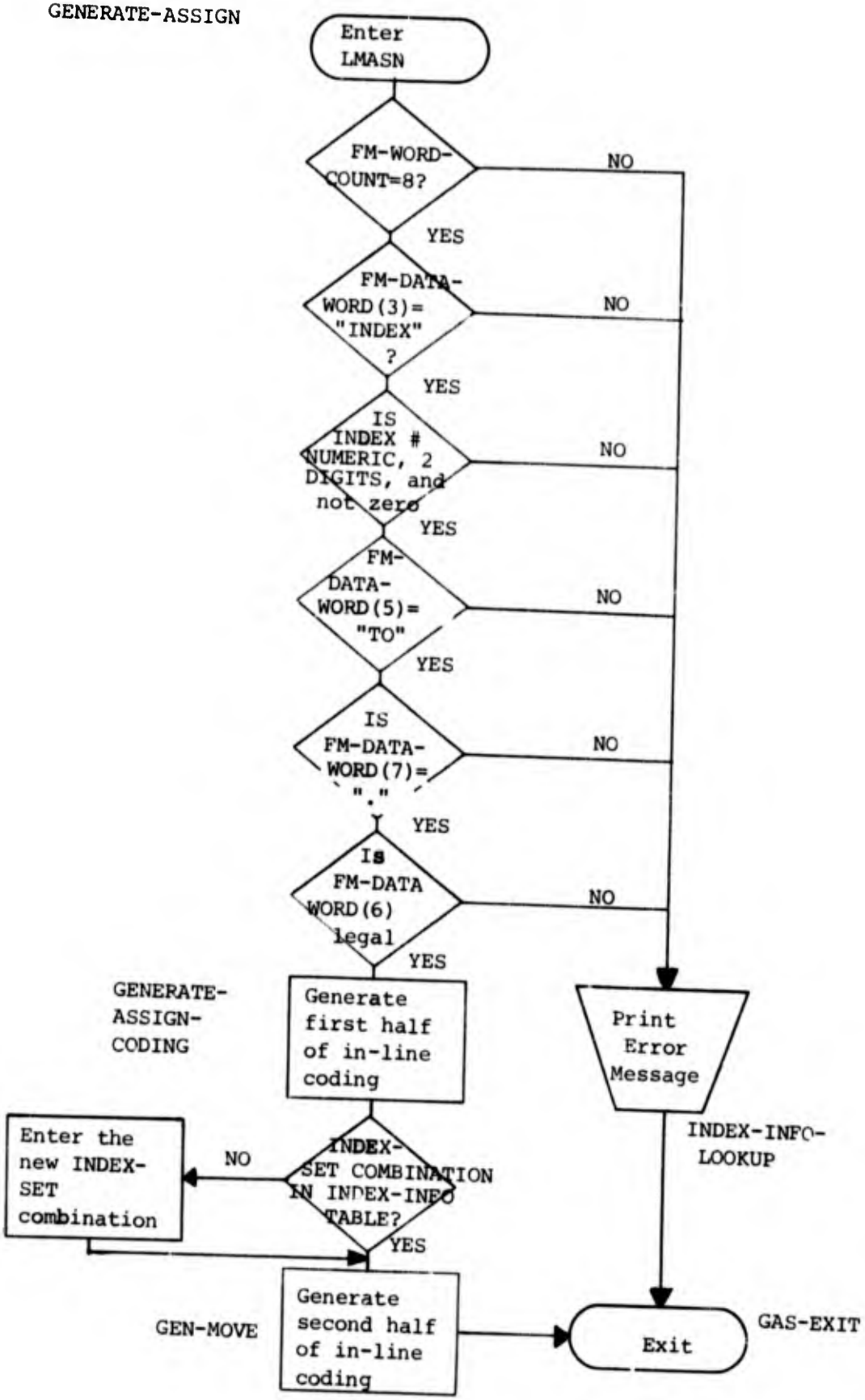
This paragraph searches the INDEX-INFO table for the set/index combination defined on this card. If they are not already in the table, they are entered.

GEN-MOVE.

This paragraph continues the COBOL code generation.

(4) Limitations. Two hundred (200) ASSIGNS per Logic Package.

GENERATE-ASSIGN



c. LMLPATC

(1) Function. Handles the ATTACH instruction.

(2) Calling Sequence.

```
ENTRY 'LMATC'  USING FM-DD  LP-DD  LM-DD
CALL  'LMFLD'  USING FM-DD  LP-DD  LM-DD
CALL  'LMPRO'  USING CARD-IMAGE
CALL  'FMPRT'  USING PRNT-CARDS  CC
```

(3) Program Description.

GENERATE-ATTACH thru GEN-ATCH-2.

This paragraph performs the following error checking on the attach statement:

The first word is the label. It is counted as word 1 regardless of its presence.

The second word must be 'ATTACH'.

The third word may be:

VSETnn or a variable set name. nn is the variable set number.

$VSETnn or a minor variable set name, nn is the minor variable set number.

A field which is not a variable set field.

The third word cannot be numeric literal, $$ORC or $$NRC, nor a field with a computational value.

The fourth word must be 'TO'.

The fifth word must be 'VSETmm', where mm is the number of a major field variable set. This is the set to be changed by the ATTACH action.

The sixth word must be a period.

The seventh word must be a failure exit. No checking is performed here.

GENERATE-ATTACH-CODING THRU GAT-EXIT.

Generates the COBOL coding for the ATTACH procedure.

SWITCH VALUES:

VARIABLE-SW

Equal to '1' - The field to be attached is a ($) variable set name or ($)VSETOO.

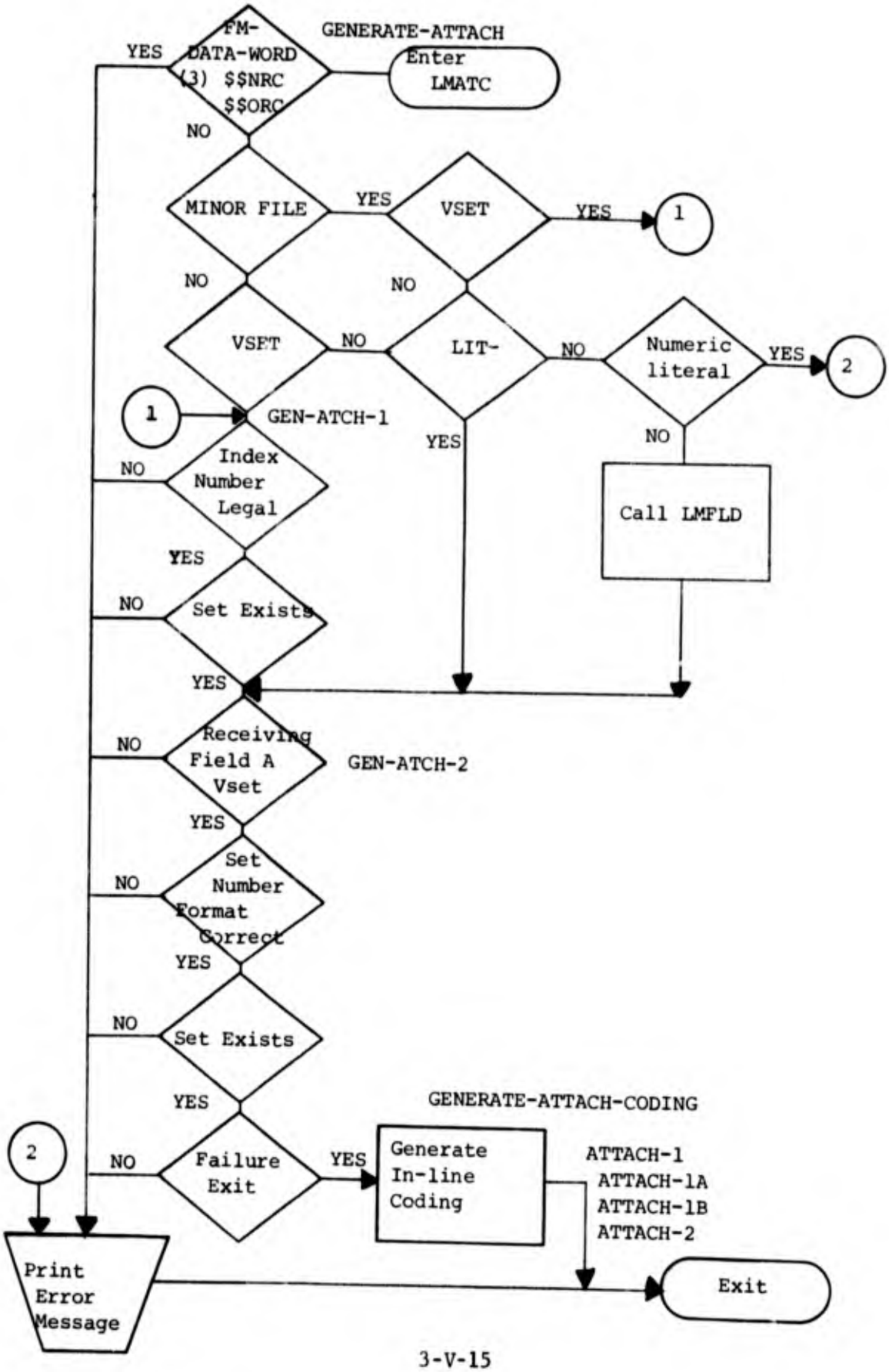
Equal to '0' - Other.

MINOR-SW

Equal to 1 - The field to be attached is from the minor file.

Equal to 0 - From the major file.

(4) Limitations. None.


GENERATE-ATTACH
Enter LMATC
FM-DATA-WORD (3) $$NRC $$ORC
YES
NO
MINOR FILE
YES
VSET
YES
1
NO
NO
VSET
NO
LIT-
NO
Numeric literal
YES
2
1
GEN-ATCH-1
YES
NO
Call LMFLD
Index Number Legal
NO
YES
Set Exists
NO
YES
Receiving Field A Vset
NO
GEN-ATCH-2
YES
Set Number Format Correct
NO
YES
Set Exists
NO
YES
GENERATE-ATTACH-CODING
2
Failure Exit
NO
YES
Generate In-line Coding
ATTACH-1
ATTACH-1A
ATTACH-1B
ATTACH-2
Print Error Message
Exit

d. LMLPBLD

(1) Function. Handles the BUILD SUBSET instruction.

(2) Calling Sequence.

```
ENTRY  'LMBLD'  USING FM-DD  LP-DD  LM-DD
CALL   'LMPRO'  USING CARD-IMAGE
CALL   'FMPRT'  USING PRNT-CARDS  CC
```

(3) Program Description.

GENERATE-BUILD and GEN-BLD2.

This paragraph performs the following error checking on the words of the BUILD card:

There must be eight (8) words on the card (the label field is always counted as number 1, even though it may not be present).

The first word is the label field. No checking is performed at this time.

The second word must be 'BUILD'.

The third word must be 'SUBSET'.

The fourth word must be a 2 digit set number. It must have been defined during File Structuring.

The fifth word must be an 'X' followed immediately by a two digit index number (it cannot be zero).

The sixth word can be a numeric minor file field, a numeric major file field, or a numeric define. Partial fields may not be used on a BUILD card.

The seventh word must be a period.

The eighth word must be a failure exit. No checking is performed here.

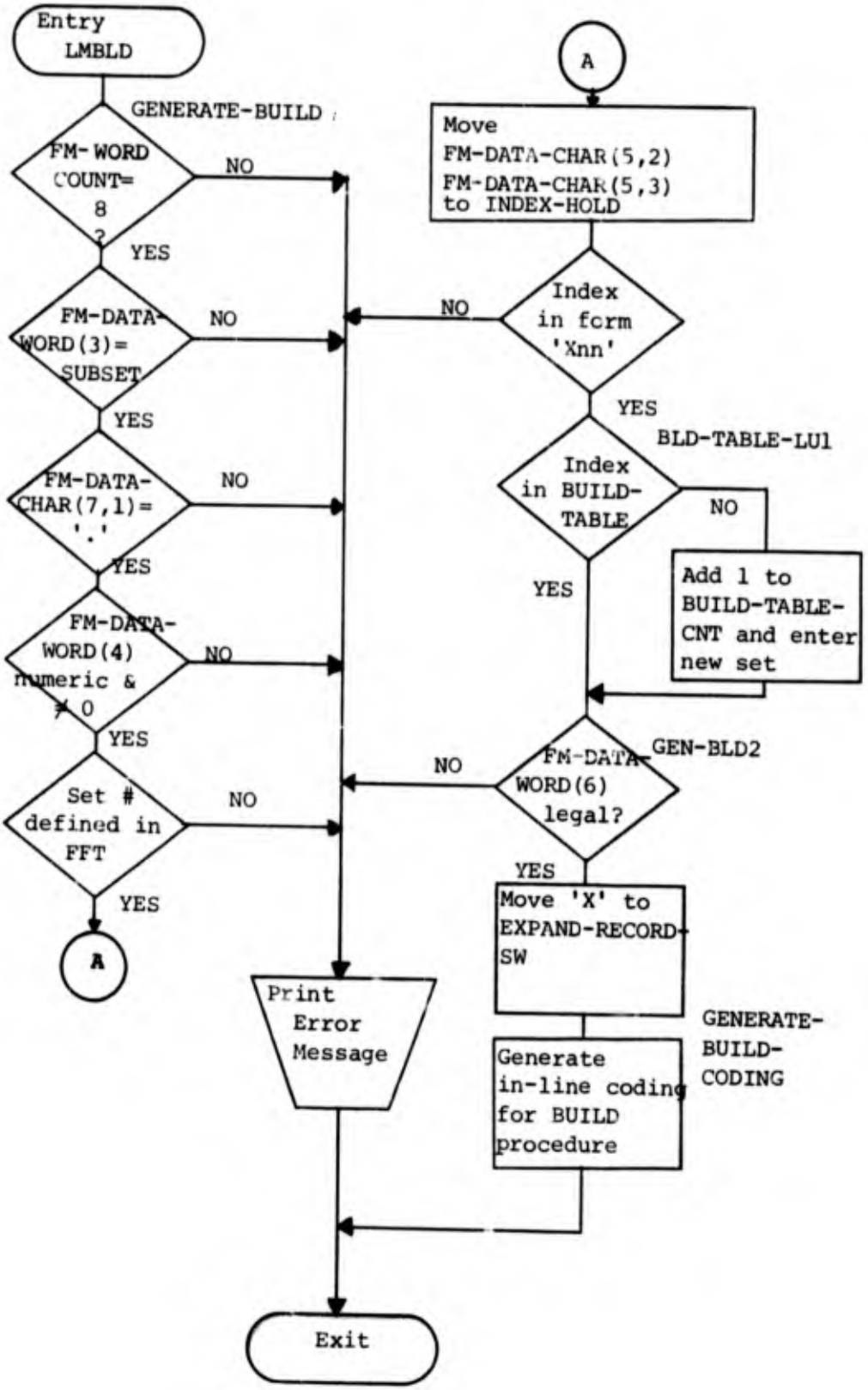
BLD-TABLE-LU1.

Scans the BUILD-TABLE looking for the combination of index and set numbers specified on this BUILD card. If it is already in the table, it will not be reentered.

GENERATE-BUILD-CODING.

EXPAND-RECORD-SW is set to indicate that the EXPAND-RECORD paragraph must be generated in LMWRAPUP. The in-line coding for the BUILD procedure is generated here.

(4) Limitations. One hundred (100) BUILDS per Logic Package.


Entry
LMBLD
GENERATE-BUILD
FM-WORD COUNT= 8 ?
NO
YES
FM-DATA-WORD(3)= SUBSET
NO
YES
FM-DATA-CHAR(7,1)= '.'
NO
YES
FM-DATA-WORD(4) numeric & ≠ 0
NO
YES
Set # defined in FFT
NO
YES
A
Print Error Message
A
Move FM-DATA-CHAR(5,2) FM-DATA-CHAR(5,3) to INDEX-HOLD
Index in form 'Xnn'
NO
YES
BLD-TABLE-LU1
Index in BUILD-TABLE
NO
Add 1 to BUILD-TABLE-CNT and enter new set
YES
GEN-BLD2
FM-DATA-WORD(6) legal?
NO
YES
Move 'X' to EXPAND-RECORD-SW
GENERATE-BUILD-CODING
Generate in-line coding for BUILD procedure
Exit

e. LMLPCNG

(1) Function. Handles the following instructions:

DELETE field
CHANGE SWITCH
SET SWITCH

(2) Calling Sequence.

```
ENTRY 'LMCNG'   USING FM-DD  LP-DD  LM-DD
ENTRY 'LMDFL'   USING FM-DD  LP-DD  LM-DD
ENTRY 'LMSET'   USING FM-DD  LP-DD  LM-DD
ENTRY 'LMPRO'   USING FM-DD  LP-DD  LM-DD
CALL  'LMPRO'   USING FM-DD  LP-DD  LM-DD
CALL  'FMPRT'   USING FM-DD  LP-DD  LM-DD
CALL  'LMFMT1'  USING FM-DD  LP-DD  LM-DD
```

(3) Program Description.

DELETE field

Paragraph LM-DELETE-ROUTINE initializes all the switches and hold areas needed for the DELETE field instruction and validates the card for errors. Logic flow branches to paragraph LDR-1 for the generation of COBOL coding if the field to be deleted is a numeric field or to paragraph LDR-2 if the field is an alphanumeric field.

CHANGE SWITCH

Processing starts in paragraph LM-CHANGE-ROUTINE where all switches and hold areas are initialized and the card validated for errors. Logic flow continues to paragraph LCR-1 where the COBOL coding is generated.

SET SWITCH

Processing starts in paragraph LM-SET-ROUTINE. All switches and hold areas are initialized and the card validated for errors. Logic flow branches to paragraph LSS-1 for COBOL code generation. Here it is determined if the switch is set 'ON' or 'OFF' and appropriate COBOL coding is generated.

PARTIAL-IN-FLD2.

This paragraph produces procedure division coding for the second half of two lines of coding needed for a partial FIELD-2.

PERIOD-GEN-1.

Generates a period at the end of a generated procedure division statement for output WORD-1.

CHAR-CK-1.

Generates a period at the end of a generated procedure division statement and is initiated by paragraph PERIOD-GEN-1 by a perform routine.

PERIOD-GEN-2.

Generates a period at the end of a generated procedure division statement for the output holding area FM-WORD-2.

CHAR-CK-2.

Generates a period at the end of a generated procedure division statement and is initiated by paragraph PERIOD-GEN-1 by a perform routine.

COBOL-OUT1.

COBOL-OUT2.

1 and 2 produce the print out of the generated COBOL statements.

INITIALIZER.

Generates all the switches and counters used in the ADD, SUBTRACT, MOVE, DELETE field, SET and CHANGE routines.

CK-PARTIAL-FLD2.

This paragraph determines if FIELD-2 is a partial and channels the logic flow to paragraph PARTIAL-IN-FLD2 if a partial is found.

LM-DELETE-ROUTINE

Calls LMLPFMT1 to determine the field type of the field to be deleted. Tests for illegal format.

LDR-1

Deletes a numeric data field or numeric defined literal by generating code which will move zeros to the field.

LDR-2

Deletes an alphanumeric data field or alphanumeric defined literal by generating code which will move space to the field.

LM-CHANGE-ROUTINE

Checks the switch name for format and spelling.

LCR-1

Generates code to change the switch value.

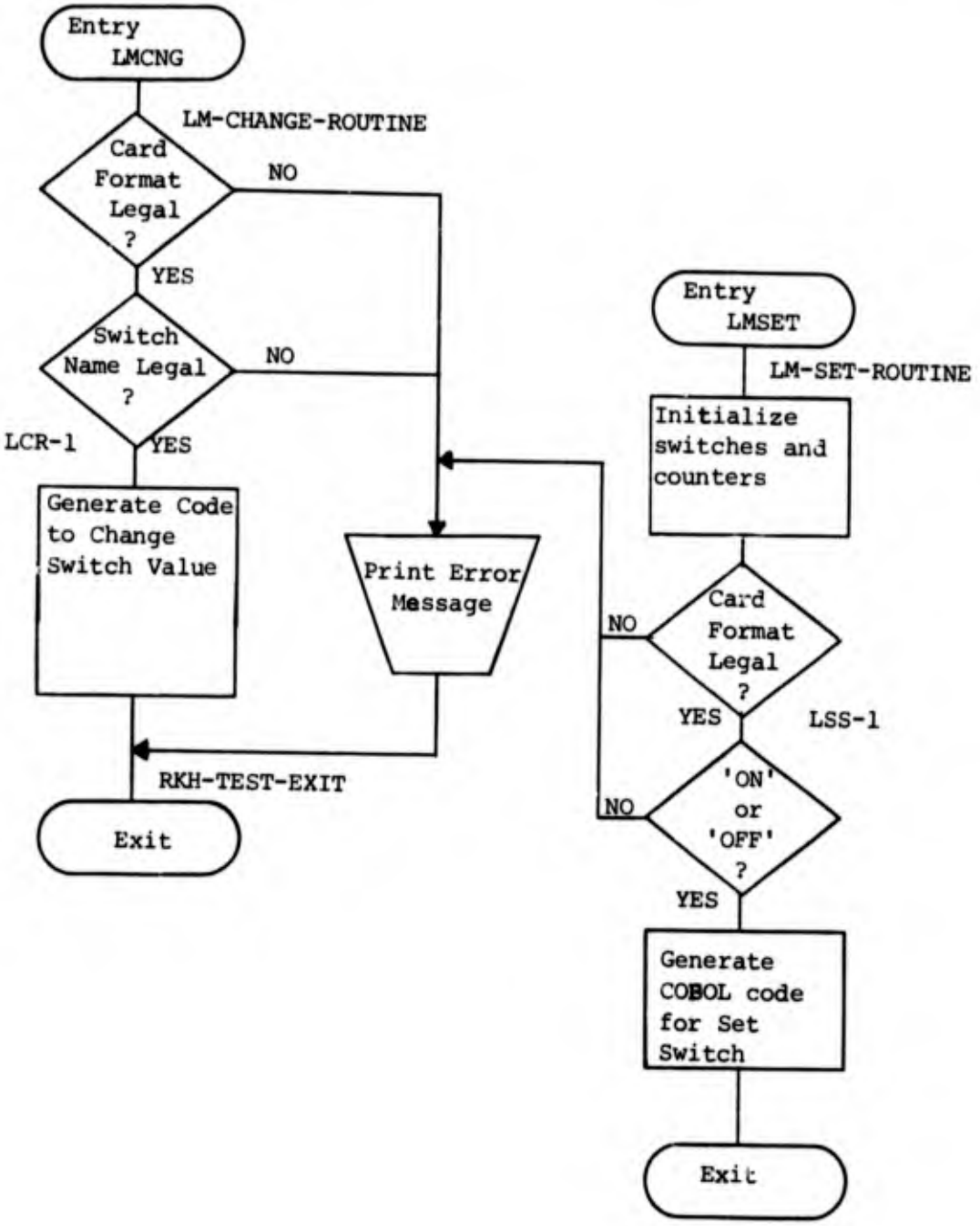
LM-SET-ROUTINE

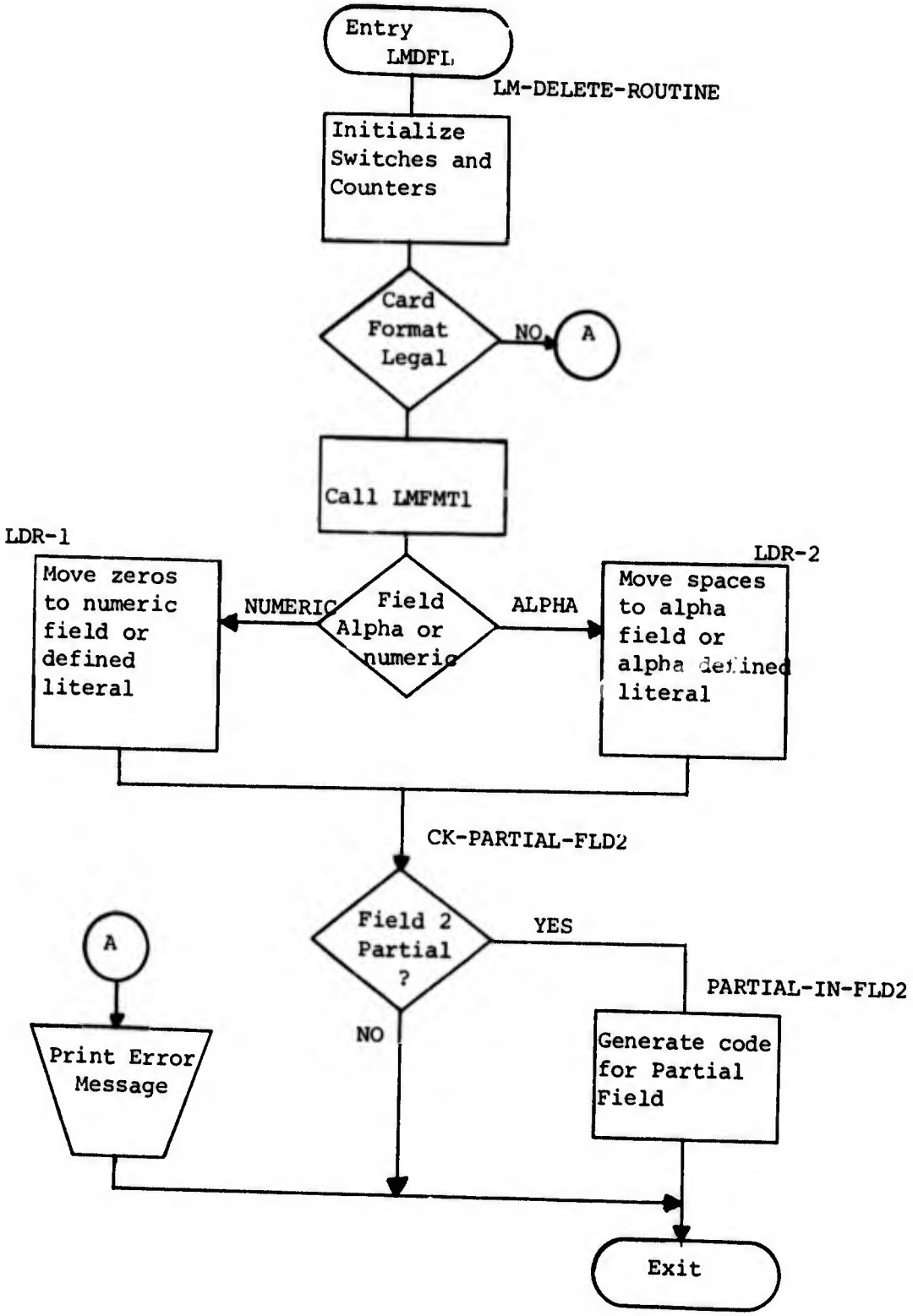
Checks the switch name for format and spelling.

LSS-1

Generates code to set the switch 'ON' or 'OFF'.

(4) Limitations. None.


Entry
LMCNG
LM-CHANGE-ROUTINE
Card
Format
Legal
?
NO
YES
Switch
Name Legal
?
NO
LCR-1
YES
Generate Code
to Change
Switch Value
Print Error
Message
RKH-TEST-EXIT
Exit
Entry
LMSET
LM-SET-ROUTINE
Initialize
switches and
counters
Card
Format
Legal
?
NO
YES
LSS-1
'ON'
or
'OFF'
?
NO
YES
Generate
COBOL code
for Set
Switch
Exit


Entry
LMDFL
LM-DELETE-ROUTINE
Initialize Switches and Counters
Card Format Legal
NO
A
Call LMFMT1
LDR-1
Move zeros to numeric field or defined literal
NUMERIC
Field Alpha or numeric
ALPHA
LDR-2
Move spaces to alpha field or alpha defined literal
CK-PARTIAL-FLD2
Field 2 Partial ?
YES
PARTIAL-IN-FLD2
NO
A
Print Error Message
Generate code for Partial Field
Exit

f. LMLPDDS

(1) Function. This program takes the COBOL Data Division generated by File Structure and writes it out into a work file where the generated code is produced.

(2) Calling Sequence.

```
ENTRY   'LMDDS'    USING  SET-NUMBER, FILE-NUMEA,
                          HOP-HOLD1, INDEX-HOLD,
                          RUN-TYPE
CALL    'FMIO'     USING  ERROR-DIAGNOSTIC
CALL    'LMLPGEN'  USING  DD-LINE(CNT2)
```

(3) Program Description.

CALL-LIB

Calls the library and receives one block for the designated set number. An 'N' operation indicates that the set and block combinations do not exist. The first line generated must be in the form

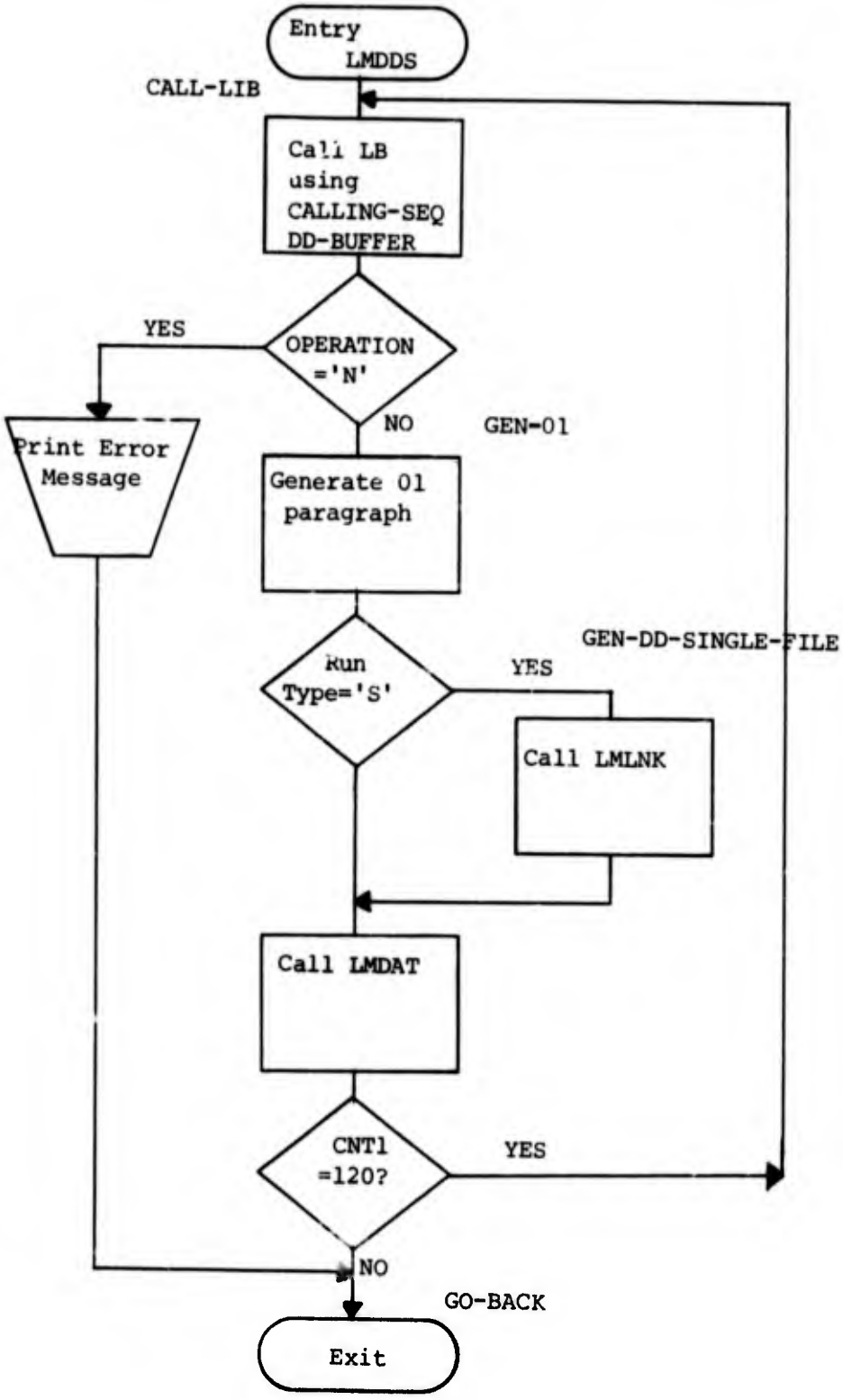
```
01  MAJOR-  file name-RECORD
    MINOR-
```

GEN-Ø1

SKIP-FIXED

These paragraphs perform the function of generating that first line. There can be a maximum of 120 80-characters lines per block. The remainder of these (the first one has already been written) are written out by paragraph GEN-DDS (in the case of file-to-file) or GEN-DDS-SINGLE-FILE (in the case of single file). A test provided to determine if there is another block associated with the same periodic or fixed set. If so control is looped to CALL-LIB which will get the next block.

(4) Limitations. None.


Entry
LMDDS
CALL-LIB
Call LB
using
CALLING-SEQ
DD-BUFFER
YES
OPERATION
='N'
NO
GEN-01
Print Error
Message
Generate 01
paragraph
GEN-DD-SINGLE-FILE
Run
Type='S'
YES
Call LMLNK
Call LMDAT
CNT1
=120?
YES
NO
GO-BACK
Exit

g. LMLPDEF

(1) Function. DEFINE value instruction.

(2) Calling Sequence.

```
ENTRY    'LMDEF'      USING  FM-DD  LP-DD  LM-DD
CALL     'MINBSRCH'   USING  FM-DD  LP-DD  LM-DD
CALL     'MAJBSRCH'   USING  FM-DD  LP-DD  LM-DD
CALL     'FMPRT'      USING  PRNT-CARDS  CC
CALL     'LMDAT'      USING  FM-CARD
CALL     'LMLNK'      USING  FM-CARD
CALL     'MOVALF'     USING  DEFINED-LITERAL-CHARS
                             HOP-HOLD1  HOP-HOLD2
                             DEF-VALUES  HOP-HOLD
                             HOP-HOLD2
CALL     'MOVNUM'     USING  DEFINED-LITERAL-CHARS
                             HOP-HOLD1  HOP-HOLD2
                             DEF-VALUES  HOP-HOLD
                             HOP-HOLD2  HOLD-IT
```

(3) Program Description.

CHAR-CHECK.

This paragraph determines if the field on the 'DEFINE' card is the field name or value. Logic flow branches to paragraphs GENERATE-DEF-TAB if the field is a name or to MOVE-LITERAL if it is a value.

The processing of a possible name continues in GENERATE-DEF-TAB where it is validated for proper leading characters, then on to DEF-WORD-BUILD and TEST-NO-7 where it is error checked in the major or minor file's FFT. The Define Table is then checked in paragraph DEF-LU and DEF-TAB-LU to see if the DEFINE name has already been used. If the name is unique it will be added to the Define Table in paragraph DEF-TAB-ADD and logic flow returns to paragraph LM-FIRST-WORD to start processing the value portion of the define.

The actual processing of the value portion of the define is in paragraph MOVE-LITERAL. In this paragraph an initial breakdown of the various types of allowable value types is made. An assumption is made that the define will be numeric until proven differently. The first check is to determine if the value is an area (numeric or alphanumeric). Logic branches to paragraph CK-PLUS if the first character is not a # sign indicating space reservation. Logic flow then proceeds to paragraph FINISH-CARD for COBOL Code generation.

CK-PLUS.

This paragraph is used when the define value is not reserving space but actually contains a numeric or alphanumeric value. Error checking for invalid format for various types of possible values are made in paragraphs CK-PLUS, CK-4AT-SIGN, and CHECK-SIGN-SPACE. Logic flow branches to paragraph BUILD-LITERAL to build the actual value into a hold area for use in the code generation further on in the logic. Error checking continues in the portion of the logic as well as the building of the literal value. Paragraph BUILD-LITERAL is 'performed' by paragraph MAKE-VALUE. Logic flow continues in paragraph FINISH-CARD at the completion of the building of the literal value.

FINISH-CARD.

This paragraph generates part of the COBOL Data Division Coding needed for a define. If an area is being defined the logic flow branches to paragraph POUND-SIGN. See below. A check is made to determine if the value is alphanumeric and applicable indicators set. For an alphanumeric value a PICTURE clause with an 'X' will be generated, otherwise a PICTURE with a S9 will be generated. Further checking of a numeric value is made in paragraph IS-IT-COMP to determine if it has been defined as computational.

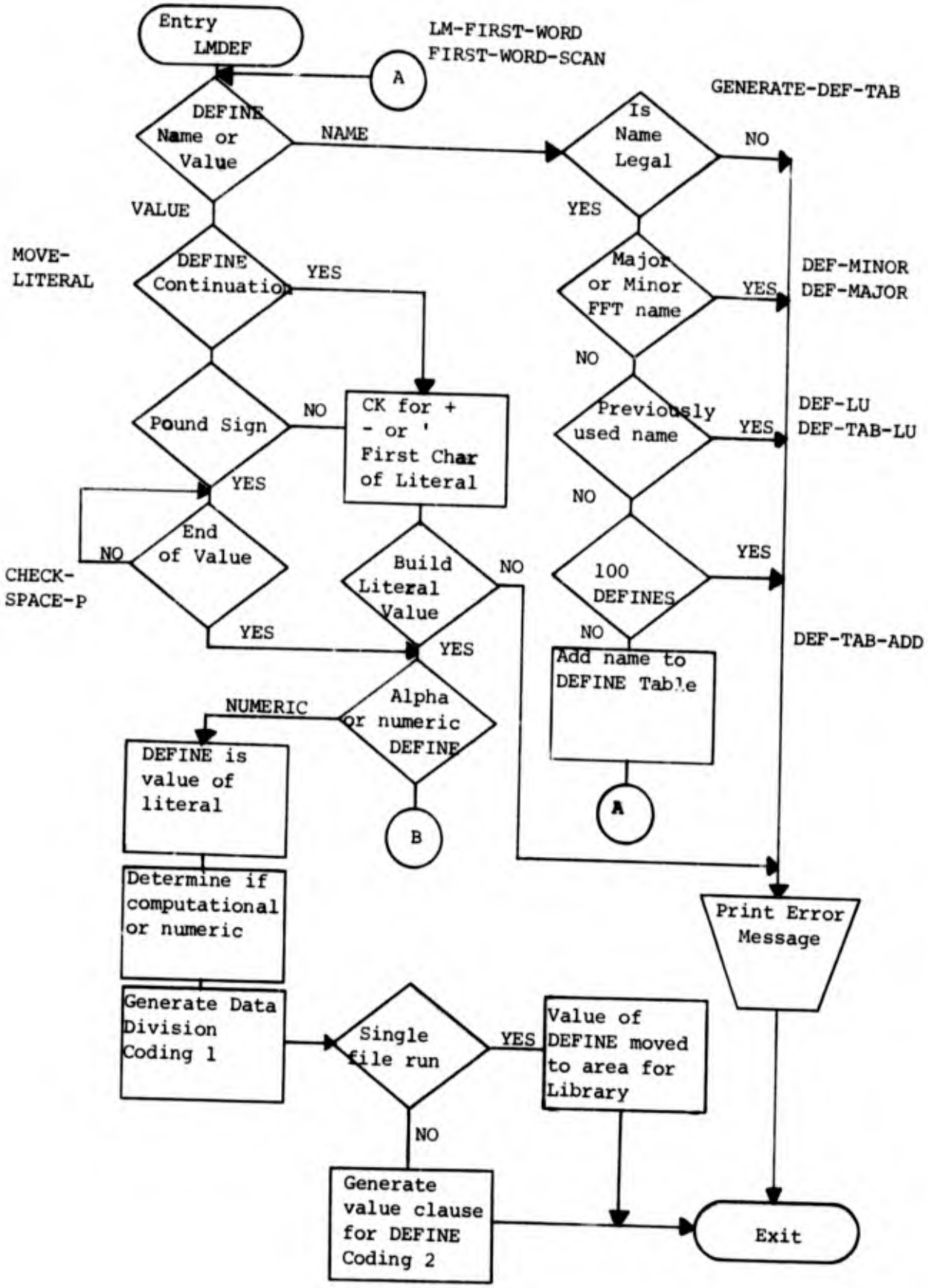
MOVE-PAREN.

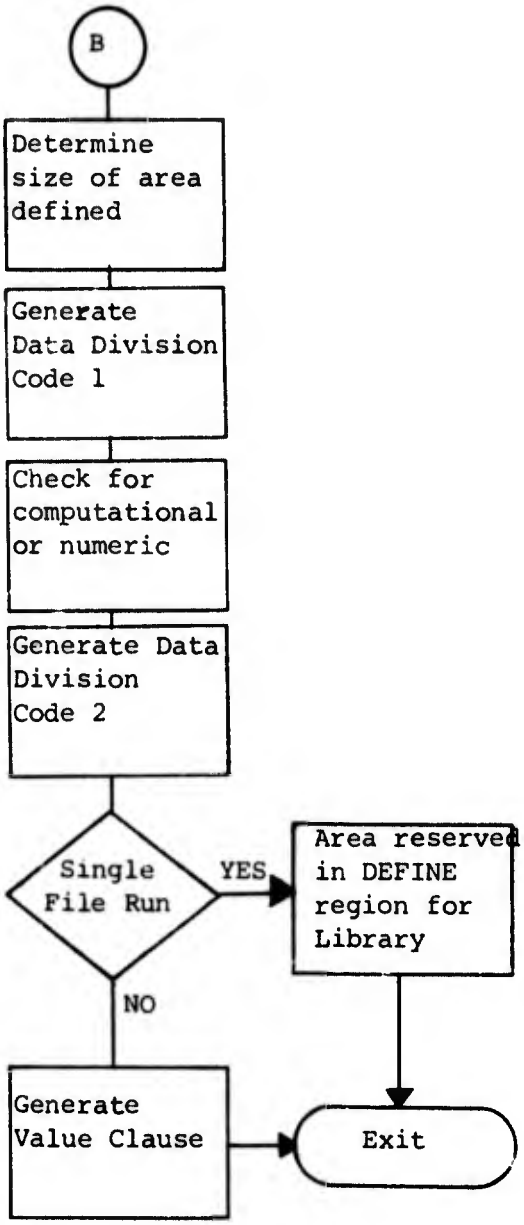
This paragraph finishes the building of the Data Division PICTURE clauses. If the run type is single-file the data value is made available to the MIDMS library for storage and use. For file-to-file runs a 'value clause' is attached to the generated COBOL Data Division statements. Paragraphs GEN-VALUE-START handles the building of a data value string for alphanumeric data for single-file runs. Paragraph MOVE-NUMERIC-DATA moves numeric data. For file-to-file runs logic flow branches to paragraph GEN-VAL-START where the value of the define is moved to a hold area. The 'value clause' is generated in paragraph TERMINATE-CARD. Logic flow continues to paragraph DEF-RESET where switches and holding areas are reinitialized prior to the end of processing.

POUND-SIGN.

Generates COBOL code for a 'DEFINE' area. In this paragraph and paragraph MOVE-POUND-VAL1 the size of the area being defined is determined and validated for errors. Paragraph IS-IT-COMP determines if the area is being defined as a numeric computational area and is 'performed' in paragraph MOVE-POUND-VAL1. Logic flow continues in paragraph MOVE-POUND-VAL where the first portion of the COBOL Data Division.

(4) Limitations. None.


Entry
LMDEF
LM-FIRST-WORD
FIRST-WORD-SCAN
A
GENERATE-DEF-TAB
DEFINE
Name or
Value
NAME
Is
Name
Legal
NO
YES
VALUE
MOVE-
LITERAL
DEFINE
Continuation
YES
Major
or Minor
FFT name
YES
DEF-MINOR
DEF-MAJOR
NO
Pound Sign
NO
CK for +
- or '
First Char
of Literal
Previously
used name
YES
DEF-LU
DEF-TAB-LU
NO
YES
End
of Value
NO
CHECK-
SPACE-P
Build
Literal
Value
NO
100
DEFINES
YES
YES
NO
DEF-TAB-ADD
Add name to
DEFINE Table
YES
Alpha
or numeric
DEFINE
NUMERIC
DEFINE is
value of
literal
A
B
Determine if
computational
or numeric
Print Error
Message
Generate Data
Division
Coding 1
Single
file run
YES
Value of
DEFINE moved
to area for
Library
NO
Generate
value clause
for DEFINE
Coding 2
Exit


B
Determine size of area defined
Generate Data Division Code 1
Check for computational or numeric
Generate Data Division Code 2
Single File Run
YES
Area reserved in DEFINE region for Library
NO
Generate Value Clause
Exit

h. LMLPDEL

(1) Function. Handles the following delete instructions:

DELETE SUBSET
DELETE PSETnn
DELETE VSETnn
DELETE RECORD

(2) Calling Sequence.

```
ENTRY  'LMDEL'  USING FM-DD  LP-DD  LM-DD
CALL   'LMPRO'  USING CARD-IMAGE
CALL   'FMPRT'  USING PRNT-CARDS  CC
```

(3) Program Description.

GENERATE-DELETE.

This paragraph tests DELETE-SWITCH and goes to the appropriate routine:

DELETE-SWITCH=1 A delete subset is being requested, go to GENERATE-DELETE-S.

DELETE-SWITCH=2 A delete periodic set is being requested, go to GENERATE-DELETE-P.

DELETE-SWITCH=3 A delete variable set is being requested, go to GENERATE-DELETE-V.

GENERATE-DELETE-S.

This paragraph performs the following error checking on the words of the DELETE SUBSET card:

There must be six words on the card.

The first word is the label field. It is counted as number one whether or not it is present. No checking is performed here.

The second word must be 'DELETE'.

The third word must be 'SUBSET'.

The fourth word must be the set and index numbers in the form 'ss&Xnn', where ss=set number and nn=index number. The set number must be defined in FFT as a periodic set.

The fifth word must be a period.

The sixth word must be an error exit label. No checking is performed here.

The in-line coding for the DELETE SUBSET routine is generated here also. Enter the set/index combination in the DELETE-TABLE.

GENERATE-DELETE-P.

This paragraph peforms the following error checking on the words of the DELETE SET card:

There must be three words on the card.

The first word is the label field. It is counted as number one whether or not it is present. No checking is performed here.

The second word must be 'DELETE'.

The third word must be 'PSETss', where ss is the periodic set number. It must be two digits, not equal to zero, and defined in the FFT as a periodic set.

The in-line coding for the DELETE PSET routine is also generated here.

GENERATE-DELETE-V.

This paragraph performs the following error checking on the words of the DELETE VSET card:

There must be three words on the card.

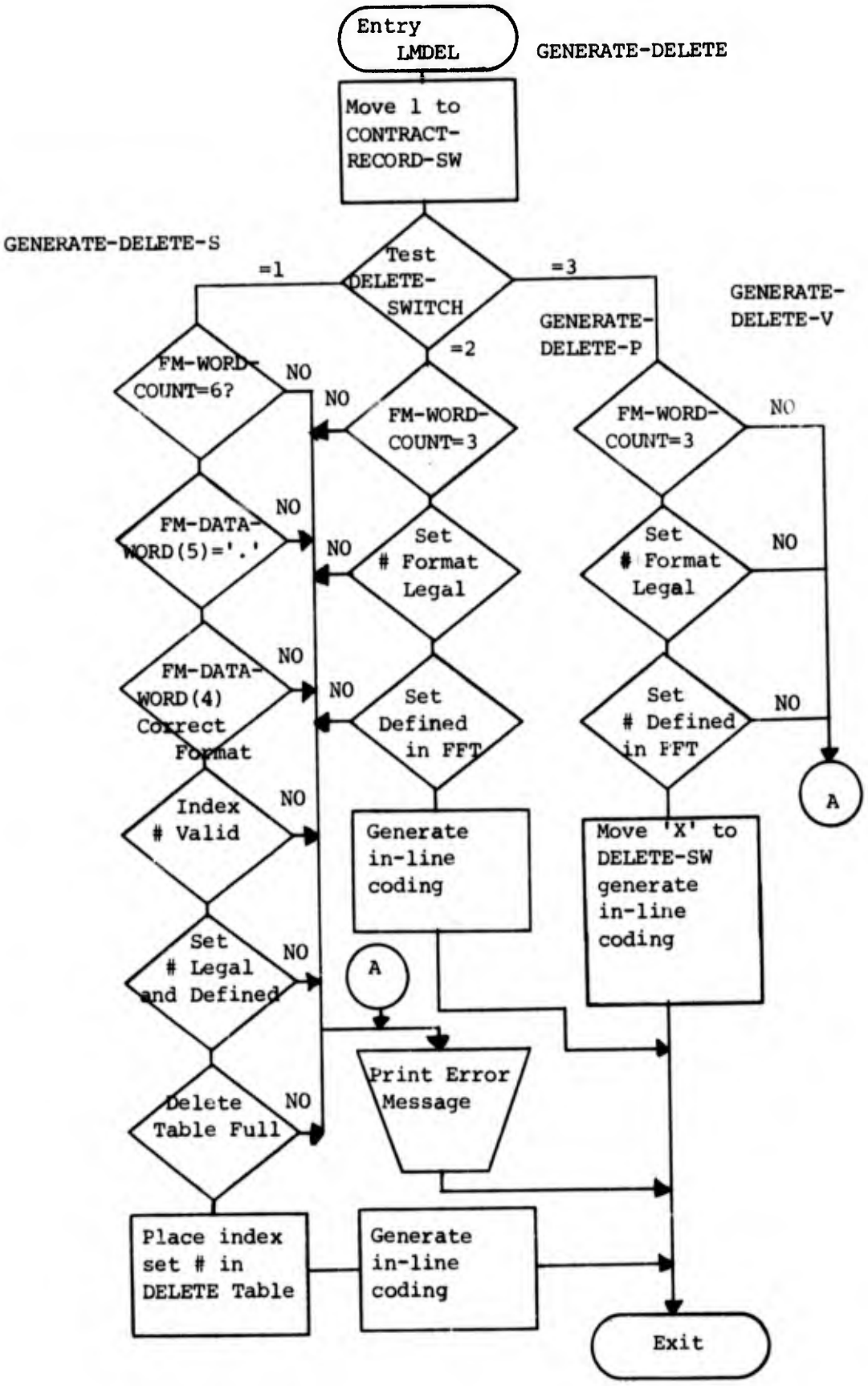
The first word is counted as number one whether or not it is present. No checking is performed here.

The second word must be 'DELETE'.

The third word must be 'VSETss', where ss is the variable set number. It must be 2 digits, not equal to zero, and defined in the FFT as a variable set.

The in-line coding is also generated here.

(4) Limitations. Twenty-five (25) DELETEs per Logic Package.


Entry
LMDEL
GENERATE-DELETE
Move 1 to CONTRACT-RECORD-SW
GENERATE-DELETE-S
Test DELETE-SWITCH
=1
=2
=3
GENERATE-DELETE-P
GENERATE-DELETE-V
FM-WORD-COUNT=6?
NO
FM-WORD-COUNT=3
NO
FM-WORD-COUNT=3
NO
FM-DATA-WORD(5)='.'
NO
Set # Format Legal
NO
Set # Format Legal
NO
FM-DATA-WORD(4) Correct Format
NO
Set Defined in FFT
NO
Set # Defined in FFT
NO
A
Index # Valid
NO
Generate in-line coding
Move 'X' to DELETE-SW generate in-line coding
Set # Legal and Defined
NO
A
Print Error Message
Delete Table Full
NO
Place index set # in DELETE Table
Generate in-line coding
Exit

i. LMLPFLD

(1) Function. This program determines a major, minor or defined field type and also checks for partial field types. A switch is set to indicate the field type to the calling program.

(2) Calling Sequence.

```
ENTRY  'LMFLD'      USING FM-DD  LP-DD  LM-DD
ENTRY  'MINBSRCH'   USING FM-DD  LP-DD  LM-DD
ENTRY  'MAJBSRCH'   USING FM-DD  LP-DD  LM-DD
CALL   'FMPRT'      USING PRNT-CARDS  CC
CALL   'LMDAT'      USING FM-CARD
```

(3) Program Description.

START-GE-CODE.
Entry points for MAJSCH, MINSCH, and LMFLD.

CHECK-FIELD-DEFINITION
Initialize counters, pointers, etc. If the first character is '$', a minor field is to be processed and MAJOR-MINOR-SW is set to 1; go to IS-MINOR-INDEXED. If first character is not a '$', the word can be a major field name or a define. The length of the word is checked to make sure it is not greater than nine characters long (FLDAA+XNN). If it is, an error has occurred and the subroutine is exited from. If the length is valid, perform IS-MAJOR-INDEXED.

IS-MAJOR-INDEXED.
If any character after the first and before the third character from the end of the word is a plus sign, this indicates that the word is an indexed major periodic field. INDEX-SW is set to 1 to indicate indexing and MAJOR-FIELD-BUILD is gone to. Else, if the sixth character of the word is gotten to and '+' has not been found, the word can still be a define or a major fixed field and MAJOR-IS-IT-SPACE is branched to.

MAJOR-IS-IT-SPACE.
Since a define can be 6 characters long, MAJOR-IS-IT-SPACE and MAJOR-SPACE scans the word through the seventh character looking for the space. If it is not found by the time 7 characters have been checked, the word is too long and an error has occurred and the subroutine is left. If a space is found, go to MAJOR-FIELD-BUILD.

MAJOR-FIELD-BUILD and BUILD-MAJOR-FIELD.
Moves the word (can still be a define or major fixed field) to FIELD-DEFINE-NAME.

LOOKUP-DEFINE and DEFINE-LOOKUP.
Checks to see if the word is a define. If it is not, EXIT-SW is set to 1 and MAJOR-LR2-LOOKUP is gone to. If word is a define and INDEX-SW=1, an error has occurred and the subroutine exit is taken.

Location in the define table is stored in LR2-DEF-POINTER. If the word is a define and it is numeric (D[illegible]-TYPE=9) set NUMERIC-SW to 1, MOVE 'D' to FIELD-OR-DEFINE, qualify define with '-U' and go to PRINT-WORD. If the define is not numeric, leave NUMERIC-SW blank and go to PRINT-WORD.

BUILD-30-CHARACTERS

Qualifies major/minor field name with '-U'. If field is a minor field (MAJOR-MINOR-SW=1), move 'OF MINOR-' to output/save area; else, move 'OF MAJOR-' to output/save area. If the field is a fixed field (FIXED-FIELD-SW=1), move '-RECORD' to output/save area; else, go to BUILD-PSETNN. If the field is a minor field (MAJOR-MINOR-SW=1), move the minor file name to the output/save area; else move the major file name to the output/save area. Go to PRINT-WORD.

BUILD-PSETNN.

If the periodic field is not indexed (INDEX-SW≠1) an error has occurred and the subroutine exit is taken. Move 'PSET' to the output/save area. If the periodic field is a minor field (MAJOR-MINOR-SW=1), move the minor periodic set number to the output save area; else, move the major periodic set number to the output/save area. Move '-X' to output/save area. Move the index number to the output area.

PRINT-WORD.

Move the output/save area to FIELD-SPECIFICATION.

MAJOR-LR2-LOOKUP.

Performs LAB-MAJOR-LU thru LAB-MAJOR-LU-EXIT which checks LR11 of the major file to see if word is a major field name. If it is not (HIT-OR-MISS≠2) an error has occurred and the subroutine exit is taken. If it is, it is indicated by setting FIELD-OR-DEFINE to 'F,' its location in LR11 is stored in LR2-DEF-POINTER.

IS-IT-MAJOR-FIX

If the field is fixed (SET-ID-2=Ø), FIXED-FIELD-SW is set to 1.
If the field is periodic (INDEX-SW=1, SET-ID-2 Ø) FIXED-FIELD-SW is set to 2.
If field is variable (SET-ID-2=Ø), FIXED-FIELD-SW is set to 3 and VARIABLE SW to 1.
If none of the above conditions are met, an error has occurred and the subroutine exit is taken.

PERFORM-BUILD-WORD.

Performs BUILD-3Ø-CHARACTERS thru BUILD-3Ø-CHARACTERS-EXIT which builds the qualified word (see examples) in an output area. Go to IS-IT-PARTIAL-FIELD.

IS-MINOR-INDEXED.

Checks to see if the word to be processed exceeds its limit of ten characters ($FLDAA+XØ1). If it does, an error has occurred and the subroutine exist is taken.

Performs MINOR-INDEXED to see if any character after the first and before the third character from the end of the word is a plus sign. If it is, this indicates that the word is an indexed minor periodic field. INDEX-SW is set to 1 to indicate an index and MINOR-FIELD-BUILD is gone to.

MINOR-IS-IT-SPACE.

Performs MINOR-SPACE to see if a space is found before the eighth character of the word is reached. If not an error has occurred and the subroutine exit is taken.

MINOR-FIELD-BUILD and BUILD-MINOR-FIELD

Moves the minor field name to FIELD-DEFINE-NAME after removing the dollar sign prefix.

MINOR-LR2-LOOKUP.

Performs LAB-MINOR-LU thru LAB-MINOR-LU-EXIT which checks LR11 of the minor file to see if word is a minor field name. If it is not (HIT-OR-MISS≠2) an error has occurred and the subroutine exit is taken. If it is, it is indicated by setting FIELD-OR-DEFINE to 'F', MAJOR-MINOR-SW is set to 1, and its location in LR11 is stored in LR2-DEF-POINTER.

IS-IT-MINOR-FIXED.

If the field is fixed (SET-ID-2=Ø), FIXED-FIELD-SW is set to 1.
If the field is periodic (INDEX-SW=1, SET-ID-2=Ø), FIXED-FIELD-SW is set to 2.
If the field is variable (SET-ID-2=Ø), FIXED-FIELD-SW is set to 3 and VARIABLE-SW to 1.
If none of the above conditions are met, an error has occurred and the subroutine exit is taken.
If any condition is met, PERFORM-BUILD-WORD is gone to.

IS-IT-PARTIAL-FIELD

If the field was a variable field (PARTIAL-SW=1), go to SET-FIELD-SWITCHES.
Update the word table counter (FM-WORD-POINTER).
If there are no more words in the word table (FM-WORD-POINTER= FM-WORD-COUNT), go to SET-FIELD-SWITCHES.

If the word to be processed now is a partial field, it is to be processed to generate COBOL Data Division statements. To qualify as partial field notation, the word can have a minimum of 4 numeric characters followd by a dash followed by a maximum of 4 numeric characters.

The length of the word is checked to see if it exceeds its limit of 9 characters. If it does it is not a partial field. The word table pointer (FM-WORD-POINTER) is set to the previous word in the word table and SET-FIELD-SWITCHES is gone to.

The word is examined for a dash. If the dash occurs after the fourth character or no dash is found, the word is not a partial field, the word table pointer (FM-WORD-POINTER) is set to the previous word in the word table and SET-FIELD-SWITCHES is gone to.

TALLY-3 is set to point at the dash in the word, TALLY contains the number of characters preceeding the dash.

SAVE-PARTIAL-1

The job now is to build the picture of the first Ø2 level in the generated COBOL Data Division statements. The number of leading zeros to put into the 4 character Ø2 level filler is stored in TALLY-1. Then MOVE-ZEROES is performed to move these zeroes into a 4 position save area (CHARACTER-4).

STOP-ZERO-1 and MOVE-NUMBER-1.

The first part of the partial field for the first Ø2 filler is now moved into the save area (CHARACTER-4) behind the zeroes previously moved in. The save area (CHARACTER-4) is moved to SAVE-1. SAVE-1 now contains the picture of the first Ø2 filler.

EXAMINE-1 and MOVE-ZEROES-2 and STOP-ZERO-2.

Next the data word is examined to determine how many characters are in the second portion of the partial field. This number is stored in TALLY-2. The number of leading zeroes required in the second Ø2 filler is stored in CNT2.

The save area (CHARACTER-4) is built which contains the picture of the second Ø2 filler to generate. The save area is stored in SAVE-2.

CHECK-PARATIALS.

If the first part of the partial field (SAVE-1) is zero or the first part (SAVE-1) is greater than the second part (SAVE-2), an error has occurred and the subroutine exit is taken.

If the partial is for a define (FIELD-OR-DEFINE=D), go to CHECK-PARTIAL-DEFINE.

If the partial is for a minor field (MAJOR-MINOR-SW=1), go to MINOR-PARTIAL-FIELD.

If the major field is numeric, set NUMERIC-SW to 1.

Put the major field size in SAVE-3.

Go to CONTINUE-PARTIAL.

MINOR-PARTIAL-FIELD.
If the minor field is numeric, set NUMERIC-SW to 1.

Put the minor field size in SAVE-3.

Go to CONTINUE-PARTIAL.

CHECK-PARTIAL-DEFINE.
If the define is numeric, set NUMERIC-SW to 1.
Move the define size to SAVE-3.

CONTINUE-PARTIAL.
Checks to see if the second part of the partial (SAVE-2) is larger than the size of the field (SAVE-3). If so an error has occurred and the subroutine exit is taken.

BUILD-PARTIAL-DATA-DIVISION thru BUILD-Ø2-2.
These paragraph use SAVE-1, SAVE-2, and SAVE-3 to generate the Ø1 level, Ø2 level fillers, and the Ø2 level partial field (see examples of output).

SET-FIELD-SWITCHES.
This paragraph proceeds through a series of levels of IF and ELSE statements testing combinations of previously set switches to determine exactly what type of field is present. The switches are tested in the following order:

LAB-MAJOR-LU thru LAB-MAJOR-LU-EXIT.
This is a binary search designed to search major LR11 to see if the word being processed is a major field. If it is, FIELD-SW is set to 1 and HIT-OR-MISS is set to 2.

LAB-MINOR-LU thru LAB-MINOR-LU-EXIT.
This is a binary search designed to search minor LR11 to see if the word being processed is a minor field. If it is, FIELD-SW is set to 1 and HIT-OR-MISS is set to 2.

SWITCHES USED IN LMLPFLD

FIELD-OR-DEFINE

=F	LR2 Mnemonic
≠F	define

FIXED-FIELD-SW

=3	Variable Field
≠3	Non-Variable Field

MAJOR-MINOR-SW

=1	Minor Field
≠1	Major Field

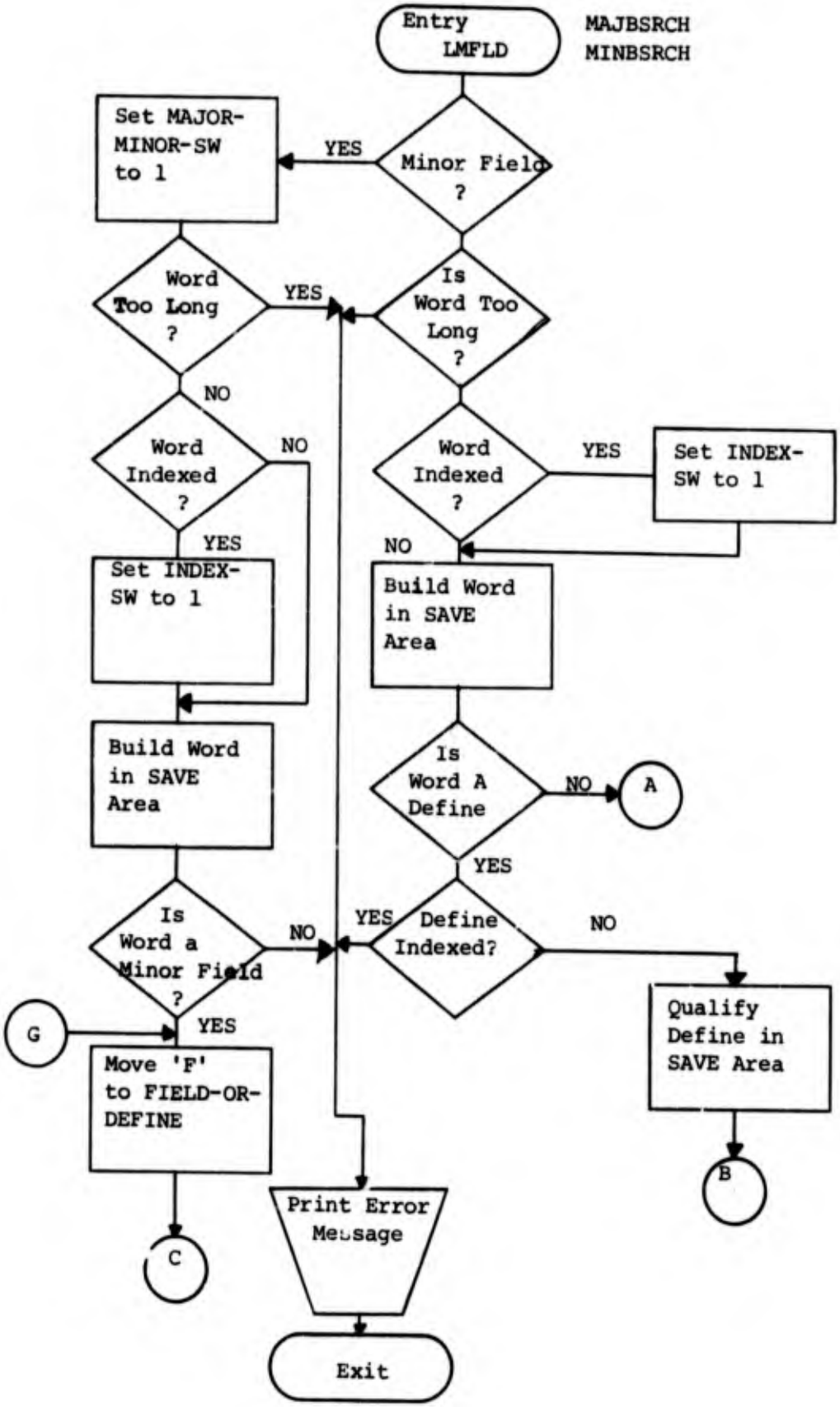
```
DATA-TYPE (for LR2 Mnemonic Fields)
=2              Numeric
≠2              Alphanumeric

DEF-TYPE  (for Define Fields)
=9              Numeric
=C              Numeric
≠9 and ≠C       Alphanumeric

FLD-SWITCH
=1              First Field from Calling Program
≠1              Second Field from Calling Program
```

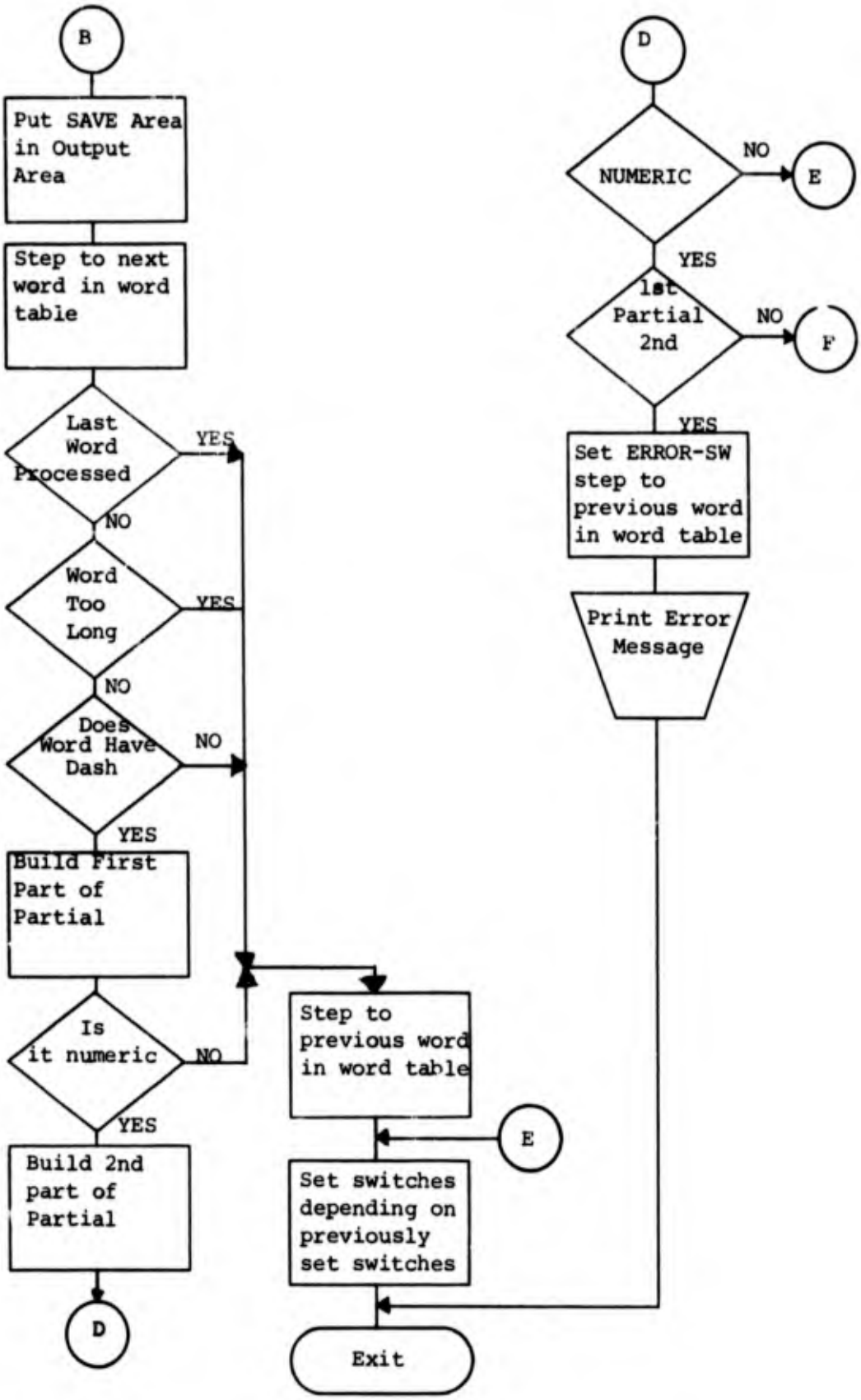
Depending on the combination of these switches, one of the following values is given to FIELD1-TYPE-SW (for FLD-SWITCH=1) or FIELD2-TYPE-SW (for FLD-SWITCH≠1).

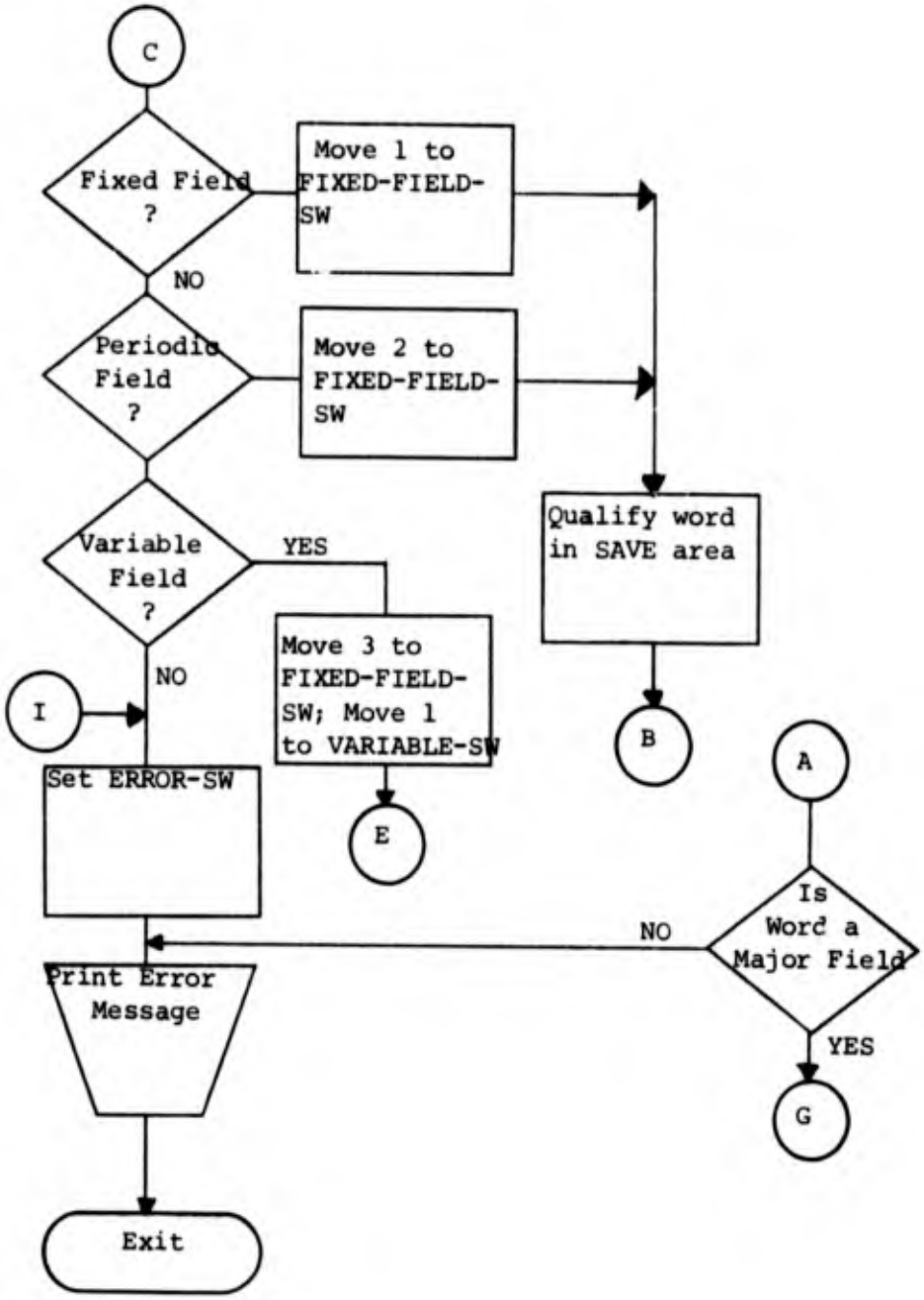
```
1=non-variable alphanumeric LR2 mnemonic or alphanumeric define
3=numeric define
5=non-variable numeric LR2 mnemonic
6=variable minor LR2 mnemonic
7=variable major LR2 mnemonic
```

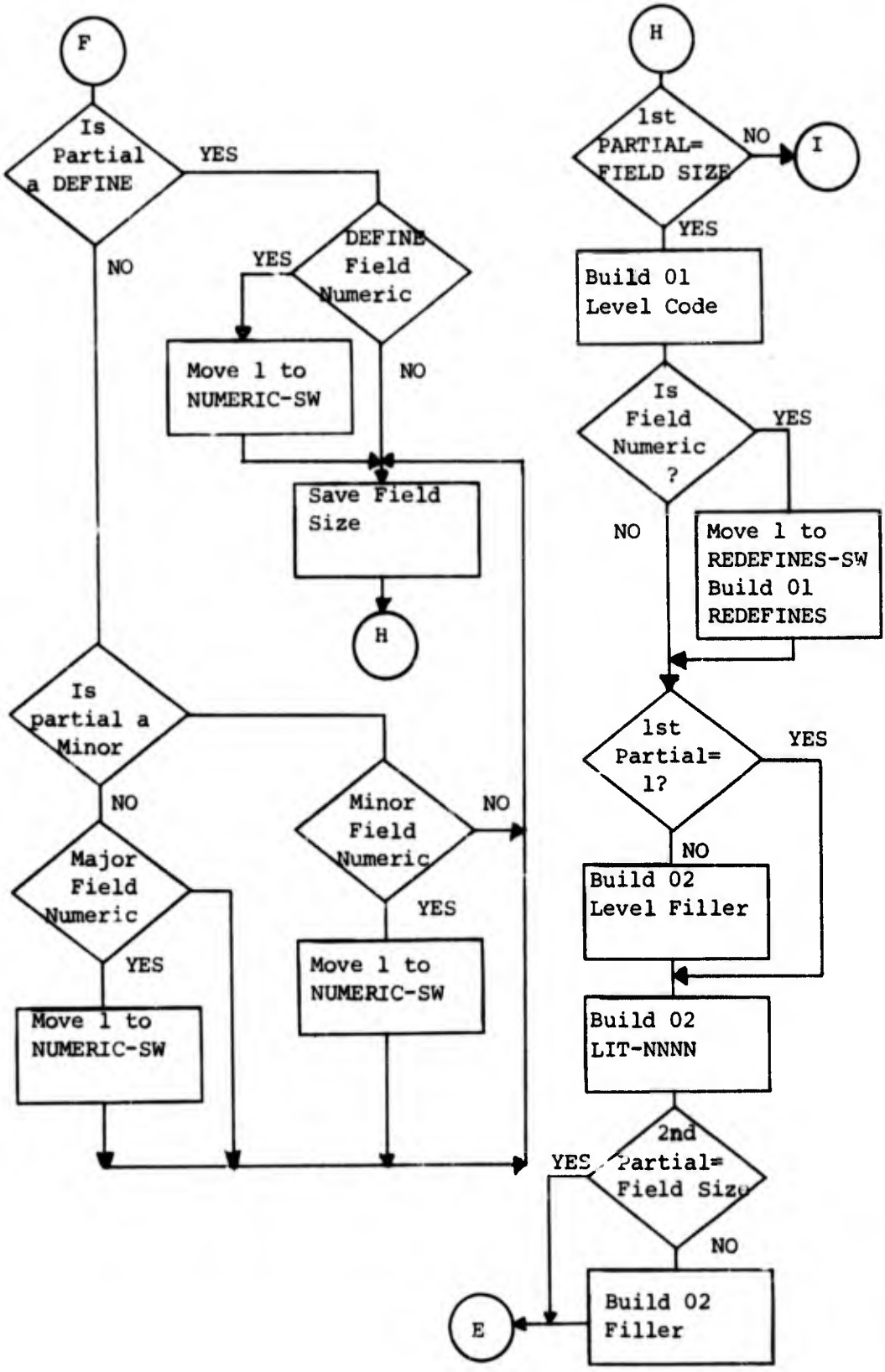
Final processing is done to place the field size in either FIELD-1-SIZE or FIELD-2-SIZE depending on the setting of FLD-SWITCH.

(4) Limitations. None.


Entry
LMFLD
MAJBSRCH
MINBSRCH
Minor Field ?
YES
Set MAJOR-MINOR-SW to 1
Is Word Too Long ?
YES
Word Too Long ?
NO
Word Indexed ?
NO
YES
Set INDEX-SW to 1
Word Indexed ?
YES
Set INDEX-SW to 1
NO
Build Word in SAVE Area
Build Word in SAVE Area
Is Word A Define
NO
A
YES
Define Indexed?
YES
NO
Is Word a Minor Field ?
NO
YES
G
Move 'F' to FIELD-OR-DEFINE
Qualify Define in SAVE Area
B
C
Print Error Message
Exit


B
Put SAVE Area in Output Area
Step to next word in word table
Last Word Processed
YES
NO
Word Too Long
YES
NO
Does Word Have Dash
NO
YES
Build First Part of Partial
Is it numeric
NO
YES
Build 2nd part of Partial
D
D
NUMERIC
NO
E
YES
1st Partial 2nd
NO
F
YES
Set ERROR-SW step to previous word in word table
Print Error Message
Step to previous word in word table
E
Set switches depending on previously set switches
Exit

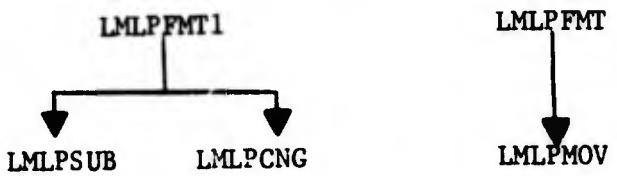

C
Fixed Field ?
Move 1 to FIXED-FIELD-SW
NO
Periodic Field ?
Move 2 to FIXED-FIELD-SW
Variable Field ?
YES
Move 3 to FIXED-FIELD-SW; Move 1 to VARIABLE-SW
NO
I
Set ERROR-SW
E
Qualify word in SAVE area
B
A
Is Word a Major Field
NO
YES
G
Print Error Message
Exit


F
Is Partial a DEFINE
YES
NO
DEFINE Field Numeric
YES
Move 1 to NUMERIC-SW
NO
Save Field Size
H
Is partial a Minor
NO
Major Field Numeric
YES
Move 1 to NUMERIC-SW
Minor Field Numeric
NO
YES
Move 1 to NUMERIC-SW
H
1st PARTIAL= FIELD SIZE
NO
I
YES
Build 01 Level Code
Is Field Numeric ?
YES
NO
Move 1 to REDEFINES-SW
Build 01 REDEFINES
1st Partial= 1?
YES
NO
Build 02 Level Filler
Build 02 LIT-NNNN
2nd Partial= Field Size
YES
NO
Build 02 Filler
E

j. LMLPFMT and LMLPFMT1

(1) Function. This program is used to determine a field for programs LMLPSUB, LMLPCNG and LMLPMOV.

System Flow



(2) Calling Sequence.

```
ENTRY 'LMFMT' or 'LMFMT1' USING FM-DD  LP-DD  LM-DD
CALL  'LMFLD'             USING FM-DD  LP-DD  LM-DD
CALL  'LMPRO'             USING COBOL-FORMAT1,2,3
CALL  'FMPRT'             USING PRNT-CARDS  CC
```

(3) Program Description.

(a) Paragraph LM-FORMAT-TEST initially establishes the possible field type by examining the first character of the field. Logic branches to one of four paragraphs depending on the possible field type. If the first character of the field is a '$', logic flow passes to paragraph LMR-1 where a check is made to determine if the field is the system references $$ORC or $$NRC. If the field is not one of the two system references, it is assumed to be a minor field and the logic flow continues in paragraph FIELD-PERFORM where a call is made to LMLPFLD for determination of the actual minor field type. (For the continued logic flow after the return from LMLPFLD skip down to the section labeled return from field.) Switches are set to indicate this type field.

(b) If the first character is a numeric digit, logic flow branches to paragraph LMR-2. Paragraphs LMR-2 thru LMR-2B1 validate a numeric literal. Valid numeric literals are moved into an output holding area and processing returned to the program which called LMLPFMT(1). If, in paragarph LM-FORMAT-TEST, the first character of a field being examined, should be a '-' or '+' sign, logic flow would first proceed to paragraph LMR-1B to test for numeric length limitation. Logic flow would then proceed to paragraph LMR-2A for continued validation as shown above for a numeric literal. The assumption is made that the field is a signed numeric literal. Switches are set to indicate this type field.

(c) If the first characters of the field were alphabetic, logic flow would branch to LMR-4 to test a major data field. Switches are set to indicate this type field and logic flow continues to paragraph FIELD-PERFORM to actual determination of major field type

by calling program module LMLPFLD.

(d) Return from LMLPFLD. Upon return from LMLPFLD for the determination of minor or major data field types, a test is made in paragraph FIELD-PERFORM to determine if the minor or major field has a partial field notation attached with it. If no partial field is indicated, no further processing by LMLPFMT(1) is necessary and logic flow returns to the calling program. Logic flow branches to paragraph CFD-PARTIAL for determination of a partial field for field1 or field2 of an input logical maintenance statement. If field2 is in question, logic flow branches to CFD-4. Field1 is processed in paragraphs CFD-PARTIAL and CFD-3A for the generation of COBOL Procedure Division coding for a partial field in field1. Paragraphs CFD-4 generates the Procedure Division COBOL Coding necessary for a partial field in field2 position.

(e) Processing is terminated at the FORMAT-TEST-EXIT paragraph and logic flow returns to the calling program.

LM-FORMAT-TEST

This paragraph tests the first characters of an input field to initially determine if the first character is $ (FOR $$ORC, $$NRC, or a minor field), -, +, or numeric digit (for a numeric literal), alphabetic (for special literal or major field). The FORMAT TEST routine checks a field for special literal, numeric literal, $$ORC, $$NRC, $$YMD, $$DAT, and major or minor field format. In the case of a major or minor field control is returned to the scan routine through the CHECK-FIELD-DEFINITION perform for further determination of the field format. LM-FORMAT-TEST is initiated when needed by a perform instruction and terminated upon identification of a correct field format or determination of an error.

LMR-1

This paragraph determines if the field being tested is $$NRC, $$ORC, or a minor field. Appropriate data is moved to an output holding area for display for $$NRC and $$ORC, $$MYMD, $$DAT, minor fields are tested further in other paragraphs.

LMR-1B

This paragraph tests a signed numeric literal to insure it does not exceed the maximum length for numeric literals in COBOL.

LMR-2

This paragraph tests an unsigned numeric literal to insure it does not exceed the maximum length for numeric literals in COBOL.

LMR-1A1, LMR-2A, LMR-3B1
Are used together to validate a numeric literal as consisting of all numeric data.

LMR-2B, LMR-2B1
Removes a '-' sign from numeric literals. Moves the field to an output holding area and sets switches to indicate a numeric literal in field1 position.

LMR-3
Removes '+' sign from numeric literals. Moves the field to an output holding area and sets switches to indicate a numeric literal in field2 position.

LMR-4
Determines if a field starting with an alphabetic character is an alphanumeric literal or a major field name. Major fields are processed in FIELD-PERFORM.

LMR-4A
Processes an alphanumeric literal by moving it to an output holding area and setting appropriate switches.

FIELD-PERFORM
A switch and pointer are set to indicate which field is being tested and which word of the input instruction is to be tested. LMLPFLD is called to actually determine the field type. The field is moved to an output holding area and a test is made to determine if a partial field is present.

CFD-2.
CFD-PARTIAL.
CFD-3A.
Generates Procedure Division coding to handle a partial field in field1 position.

CFD-4
This paragraph generates the first of two COBOL statements needed to handle a partial field in field2 position. The second line is generated in the program which generates the actual instruction coding.

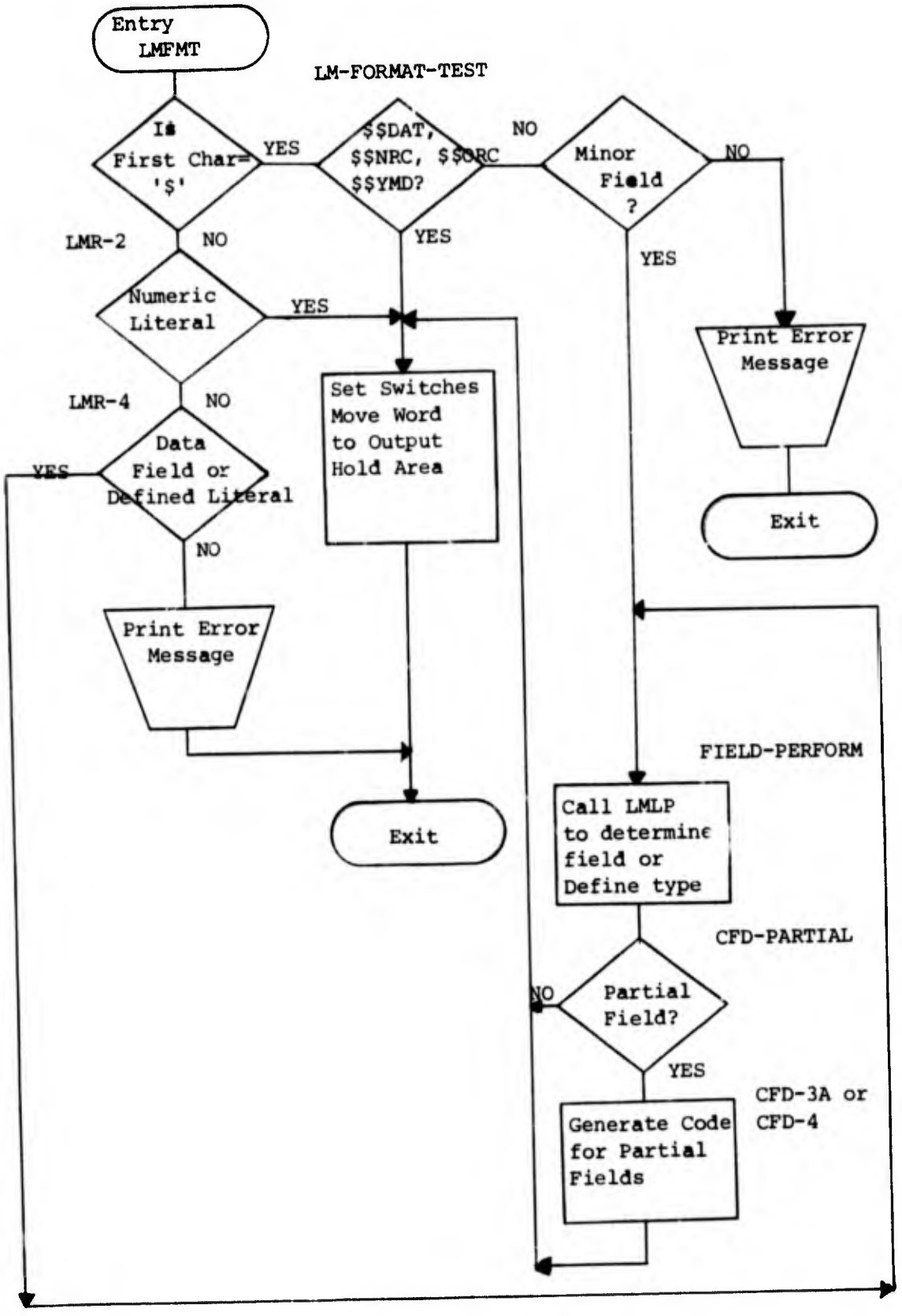
FORMAT-TEST-EXIT
Returns logic flow to the program which called LMLPFMT(1).

PERIOD-GEN-1 thru PERIOD-GEN-1-EXIT
Places a period at the end of a COBOL statement.

PERIOD-GEN-2 thru PERIOD-GEN-2-EXIT
Same as above.

COBOL-OUT1.
COBOL-OUT2.
COBOL-OUT3.
Outputs COBOL generated coding.

(4) Limitations. None.


Entry
LMFMT
LM-FORMAT-TEST
Is
First Char=
'$'
YES
$$DAT,
$$NRC, $$ORC
$$YMD?
NO
Minor
Field
?
NO
LMR-2
NO
YES
YES
Numeric
Literal
YES
Print Error
Message
LMR-4
NO
Set Switches
Move Word
to Output
Hold Area
YES
Data
Field or
Defined Literal
Exit
NO
Print Error
Message
FIELD-PERFORM
Exit
Call LMLP
to determine
field or
Define type
CFD-PARTIAL
NO
Partial
Field?
YES
CFD-3A or
CFD-4
Generate Code
for Partial
Fields

k. LMLPGEN

(1) Function. This program is used to write the Data Division, Procedure Division and Linkage Section COBOL generated coding. It is called by the various program modules whenever a line of COBOL coding has been built and is ready for generation.

(2) Calling Sequence.

```
ENTRY 'LMDAT'  USING OUTPUT-LINE
ENTRY 'LMLNK'  USING OUTPUT-LINE
ENTRY 'LMPRO'  USING OUTPUT-LINE
ENTRY 'LMCLO'  USING OUTPUT-LINE
```

(3) Program Description.

ENTRY 'LMDAT'

The paragraph WRITE-DATA opens the output files and writes a line of COBOL Data Divison coding which has been built in one of the generating program modules. Processing returns to the calling program for further code generation.

ENTRY 'LMLNK'

Paragraph WRITE-LINK writes LINKAGE SECTION COBOL coding as built prior to the call to 'LMLNK.' After writing the line of code processing returns to the calling program.

ENTRY 'LMPRO'

Paragraph writes a line of Procedure Division coding which has been previously built in the calling program, processing returns to the calling program after the write.

ENTRY 'LMCLO'

The paragraph WRITE-CLOSE closes the three output areas for the Data Division, Linkage Section, and Procedure Division.

(4) Limitations. None.

1. LMLPGRP

(1) Function. This program allows for the defining of a large area for the reading in of tape or card records.

(2) Calling Sequence.

```
ENTRY   'LMGRP'       USING FM-DD  LP-DD  LM-DD
CALL    'MINBSRCH'    USING FM-DD  LP-DD  LM-DD
CALL    'MAJBSRCH'    USING FM-DD  LP-DD  LM-DD
CALL    'LMDAT'       USING FM-CARD
CALL    'LMLNK'       USING FM-CARD
CALL    'FMPRT'       USING PRNT-CARDS  CC
CALL    'FMCRD'       USING FM-INPUT    END-SW
```

(3) Program Description.

CONTINUE-TO-SCAN.
FIRST-WORD-SCAN.
CHEK-NAME.
The input card is validated for the keywords 'DEFINE GROUP'.
VALIDATE-FIELD-NAME.
Determines validity and syntax of the group name.
DEF-MINOR.
In a file-to-file LM run, the minor file's FFT is searched to determine if the defined group's name is also a minor field or group name. If they are the same, an error message is issued.
DEF-MAJOR.
Determines if the define group name is the same as a major file or group name. If they are the same, an error message is issued.
IS-IT-ZERO.
DEF-TAB-LU.
Determines if the define group name has previously been used as a define.
DEF-TAB-ADD.
ADD-U.
If the define group is a new group which has met all the above tests successfully, the group name is entered into the define table.
CHECK-FOR-POUND.
Determines if an area is being defined or it is one of the subdefines.
VALIDATE-SPACE.
If the area from the group card is being defined, it will then be checked for errors.
GENERATE-01.
Data division statements are generated.

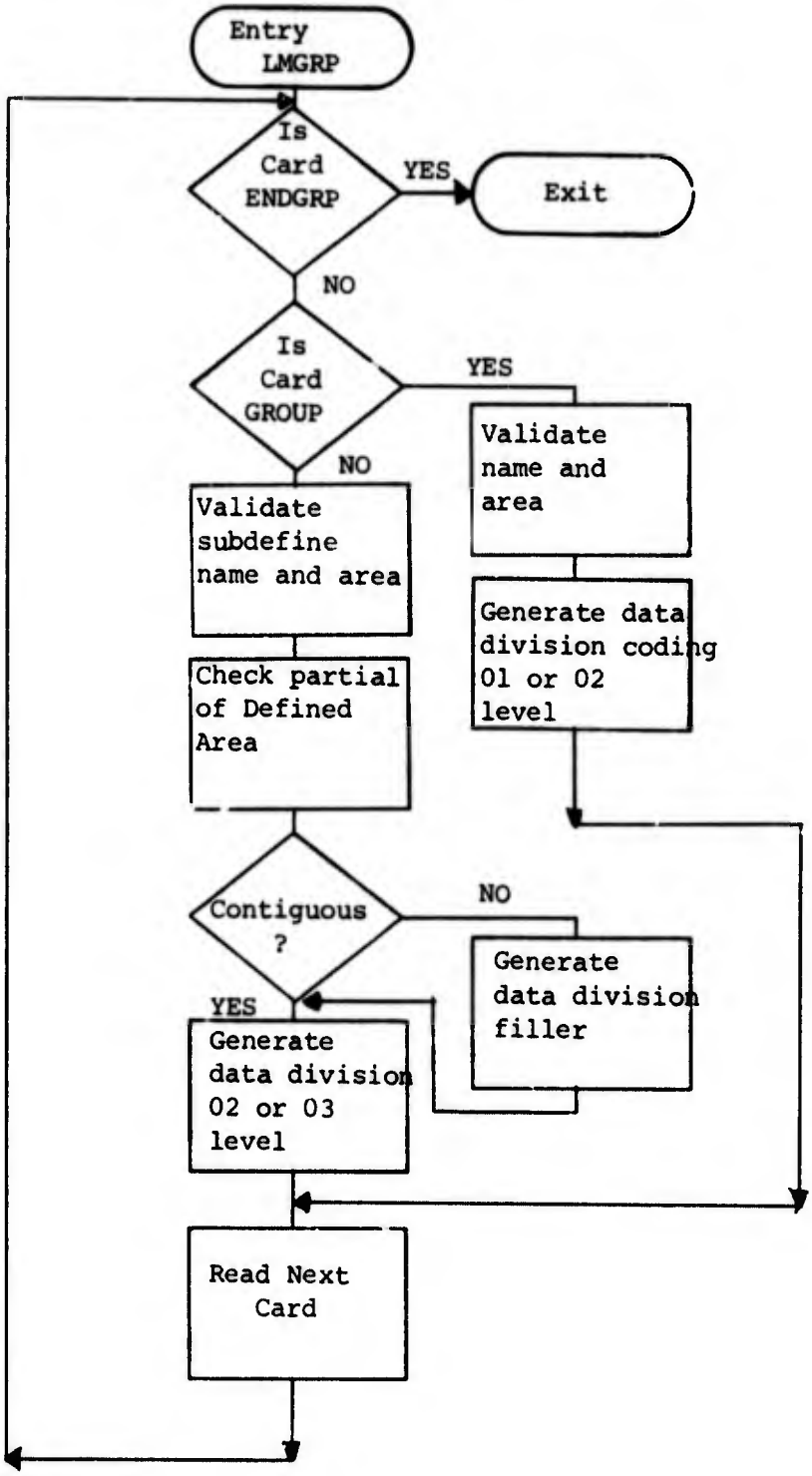
(a) If the define is a subdefine of the Group, the area will be checked for numeric or alphanumeric and validated for errors. Data Division coding will then be generated for the subdefines. Paragraph CHECK-FOR-POUND will generate a FILLER statement to define the area between subdefines. LMLPGRP will read in all cards after a DEFINE GROUP is found and will return control to LMLPSCN when the ENDGRP card is identified in paragraph CONTINUE-TO-SCAN.

(b) The user may have as many GROUP DEFINES as desired. However, only 100 DEFINES are allowed per logic package. Similar to the normal individual DEFINE, a GROUP DEFINE must be located in the logic package prior to any other logical maintenance statements. A DEFINE GROUP is initiated with the DEFINE GROUP card and is terminated with the ENDGRP statement. A Group cannot have another group contained within it. The partial ranges of the subdefines need not be contiguous but must be in ascending order. It is dependent upon the file monitor to properly define his subdefines to properly reflect his input data. It is possible to read alphanumeric data into a numeric subdefined area thereby presenting the danger of a core dump.

(c) For the subdefine fields the file monitor may specify NUMER to define numeric data; SGNUM for signed numeric data; or ALPHA for alphanumeric data following the partial area indicator. If this area is left blank, the data area will be assumed to be alphanumeric.

(d) Any type of formatted data may be read into a DEFINE GROUP area. A file could be input for the group.

(4) Limitations. None.


Entry
LMGRP
Is
Card
ENDGRP
YES
Exit
NO
Is
Card
GROUP
YES
NO
Validate
name and
area
Validate
subdefine
name and area
Generate data
division coding
01 or 02
level
Check partial
of Defined
Area
Contiguous
?
NO
YES
Generate
data division
filler
Generate
data division
02 or 03
level
Read Next
Card

m. LMLPMLT

(1) Function. Handles the MULTIPLY and DIVIDE instructions.

(2) Calling Sequence.

```
ENTRY  'LMDIV'    USING FM-DD  LP-DD  LM-DD
ENTRY  'LMMLT'    USING FM-DD  LP-DD  LM-DD
CALL   'LMPRO'    USING COBOL-FORMAT
CALL   'LMFMT1'   USING FM-DD  LP-DD  LM-DD
```

(3) Program Description.

CK-FLD2.
Error checking is performed for illegal receiving fields. A computed GO TO is used for determining the combination type of field2.

NDEF-FLD2.
Determine field1 type, field2 is a numeric defined literal.

NFLD-FLD2
Determine field1 type, field2 is a numeric data field.

ND-ND.
NF-ND.
ND-NF.
NF-NF.
Generates COBOL code dependent on combinations of field1 and field2. ND=numeric defined literal; NF=numeric data field.

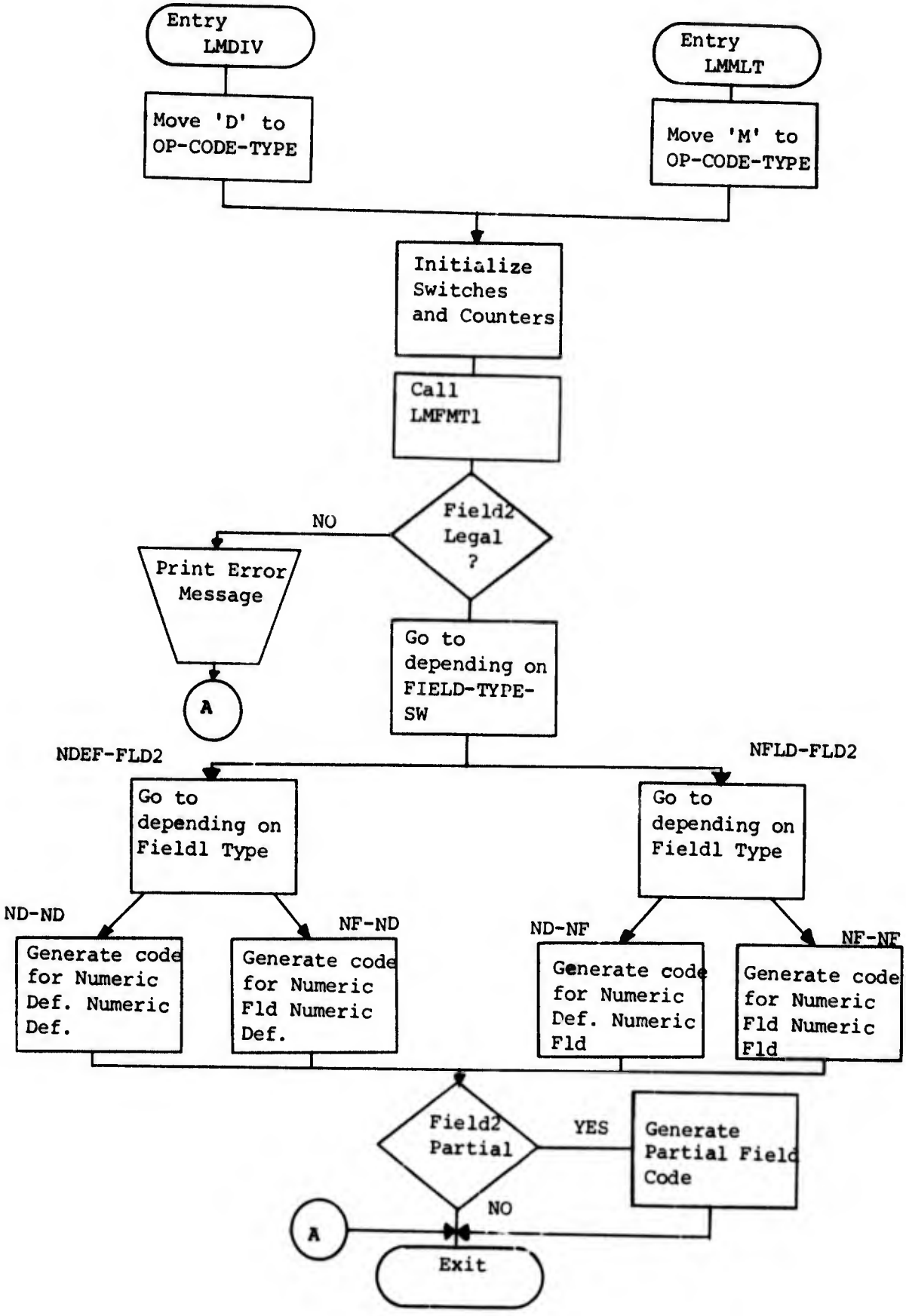
CK-PARTIAL-FLD2.
Determines if field2 is a partial; if so, generates appropriate COBOL code.

LINE-1 thru LINE-5.
Moves COBOL coding into point line position.

COBOL-OUT1.
COBOL-OUT2.
COBOL-OUT3.
Prints appropriate COBOL coding according to a predefined format.

INITIALIZER.
Initializes counters and switches.

(4) Limitations. None.


Entry
LMDIV
Move 'D' to
OP-CODE-TYPE
Entry
LMMLT
Move 'M' to
OP-CODE-TYPE
Initialize
Switches
and Counters
Call
LMFMT1
Field2
Legal
?
NO
Print Error
Message
A
Go to
depending on
FIELD-TYPE-
SW
NDEF-FLD2
NFLD-FLD2
Go to
depending on
Field1 Type
Go to
depending on
Field1 Type
ND-ND
NF-ND
ND-NF
NF-NF
Generate code
for Numeric
Def. Numeric
Def.
Generate code
for Numeric
Fld Numeric
Def.
Generate code
for Numeric
Def. Numeric
Fld
Generate code
for Numeric
Fld Numeric
Fld
Field2
Partial
YES
Generate
Partial Field
Code
NO
A
Exit

n. LMLPMOV

(1) Function. This program initially processes the MOVE/PUT instructions when the statement is in the following form:

```
                    { NUM      }
                    { ZON      }
MOVE/PUT            { field1   }      TO/INTO      field2
                    { ZONES    }
                    { NUMERICS }
```

(2) Calling Sequence.

```
ENTRY 'LMMVE'  USING FM-DD  LP-DD  LM-DD
CALL  'LMPUT'  USING FM-DD  LP-DD  LM-DD
CALL  'LMNMØ'  USING FM-DD  LP-DD  LM-DD
CALL  'FMPRT'  USING PRNT-CARDS  CC
CALL  'LMNM1'  USING FM-DD  LP-DD  LM-DD
CALL  'LMNM2'  USING FM-DD  LP-DD  LM-DD
CALL  'LMNM3'  USING FM-DD  LP-DD  LM-DD
CALL  'LMZNØ'  USING FM-DD  LP-DD  LM-DD
CALL  'LMZN1'  USING FM-DD  LP-DD  LM-DD
CALL  'LMZN2'  USING FM-DD  LP-DD  LM-DD
CALL  'LMZN3'  USING FM-DD  LP-DD  LM-DD
```

(3) Program Description.

(a) LMLPMOV determines in paragraph LM-MOVE-ROUTINE if a move statement is MOVE NUM, MOVE NUMERICS, MOVE ZON or MOVE ZONES. If the move is one of the types shown above, logic flow goes to paragraph NUM-AND-ZON-MOVE-RTN for validation of the first field of the input card. LMLPFMT is called to determine the first field type. Continued processing determines if there are any errors on the card and what position field2 is in. LMLPFMT is called again in paragraph NUM-OR-ZON to determine field2 type. The third word on the card is checked for ZONES or ZONE and logic flow goes to CALL-ZON if it is, otherwise the logic flow goes to CALL-NUM. These paragraphs will call LMLPZNØ or LMLPNMØ. A computed GO TO is used to select the program (LMLPZN1,2, or 3 or LMLPNM1,2 or 3) for the actual code generation.

(b) If in paragraph LM-MOVE-ROUTINE it is determined that the input card is a MOVE field, instruction LMLPFMT is called to determine field1 type and the card validated for proper format. TEST-FLD2-RTN calls LMLPFMT to determine field2 type. Logic flow continues to NEXT-IF where LMLPPUT is called to generate COBOL coding.

INITIALIZER
Initializes all the switches, counters, and storage areas.

LM-MOVE-ROUTINE
Determines which type of MOVE instruction is being processed. Calls LMLPFMT to determine field1 type. Checks the connector between fields for the word 'TO' or 'INTO.'

TEST-FLD2-TRN.
LMLPFMT is called to determine field2 type. Error messages are generated for invalid field combinations.

NEXT-IF
LMLPPUT is called to generate coding for the MOVE field instruction.

NUM-AND-ZON-MOVE-RTN
Logic flow branches here from LM-MOVE-ROUTINE for MOVE ZON/NUM type instructions. Field2 type is determined by calling LMLPFMT. Checking is done to determine if the connector between fields is 'TO' or 'INTO'.

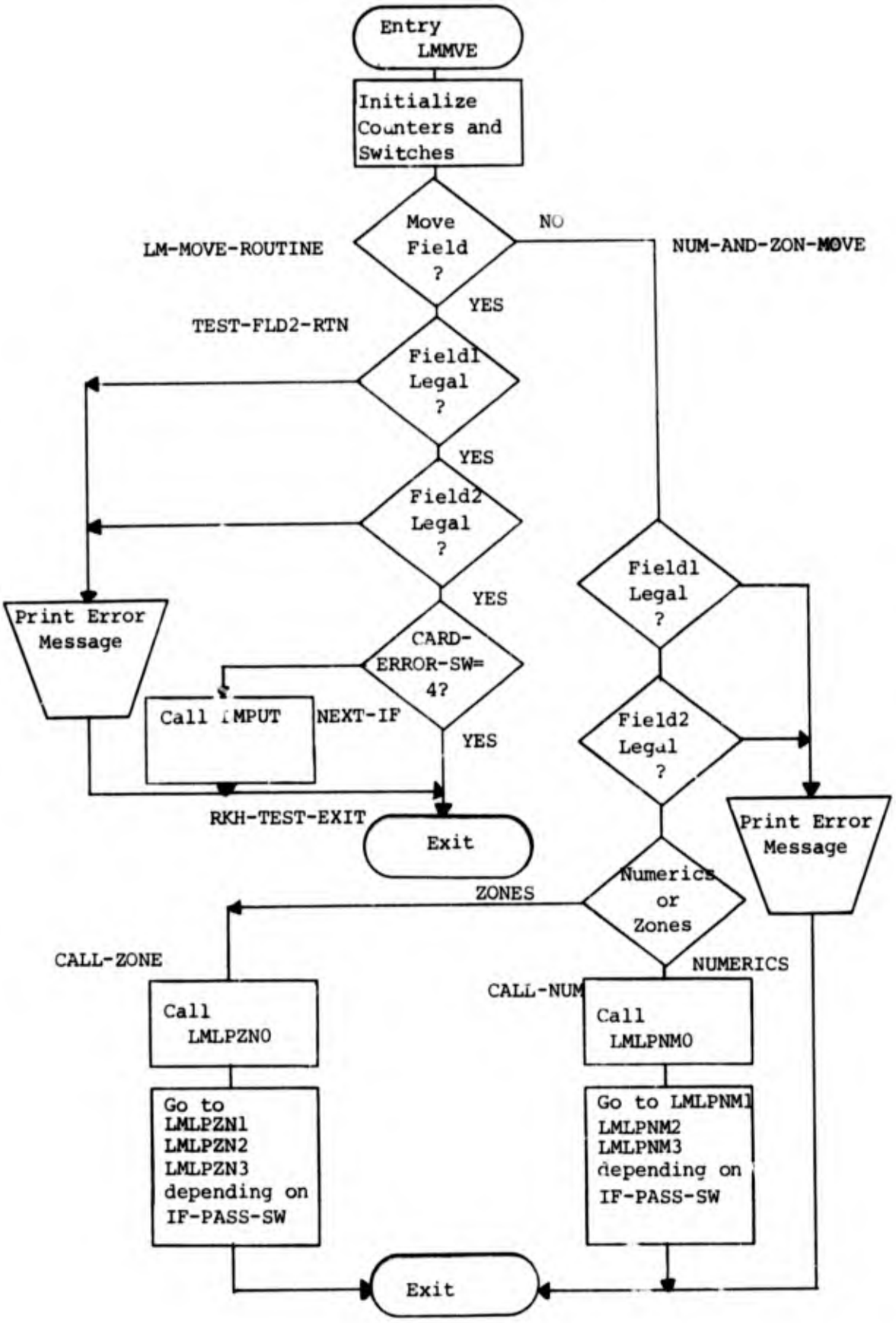
NUM-OR-ZON.
LMLPFMT is called to determine field2 type. Word 3 is checked to determine if the MOVE instruction is a MOVE ZON or MOVE NUM.

CALL-NUM
LMLPNMØ is called for the MOVE NUM instruction to determine what COBOL coding is to be generated. A computed GO TO is used to call the appropriate program for generating the COBOL coding. IF-PASS-SW is set in LMLPNMØ and used to select the program to be called.

RKH-TEST-EXIT.
Common exit point.

CALL-ZON
LMLPZNØ is called for the MOVE ZON instruction to determine which COBOL coding is to be generated. A Computed Go To is used to call the appropriate program for actually generating the COBOL coding. IF-PASS-SW is set in LMLPZNØ, and is used to determine which program is called.

(4) Limitations. None.


Entry
LMMVE
Initialize
Counters and
Switches
Move
Field
?
NO
LM-MOVE-ROUTINE
NUM-AND-ZON-MOVE
YES
TEST-FLD2-RTN
Field1
Legal
?
YES
Field2
Legal
?
YES
Print Error
Message
CARD-
ERROR-SW=
4?
Call IMPUT
NEXT-IF
YES
RKH-TEST-EXIT
Exit
Field1
Legal
?
Field2
Legal
?
Print Error
Message
Numerics
or
Zones
ZONES
NUMERICS
CALL-ZONE
CALL-NUM
Call
LMLPZN0
Call
LMLPNM0
Go to
LMLPZN1
LMLPZN2
LMLPZN3
depending on
IF-PASS-SW
Go to LMLPNM1
LMLPNM2
LMLPNM3
depending on
IF-PASS-SW
Exit

o. LMLPNMØ

(1) Function. This program controls the processing for the MOVE NUM (NUMERICS) instruction. It performs an error checking function and determines the field type of field1 and field2.

(2) Calling Sequence.

```
ENTRY  'LMNMØ'  USING FM-DD  LP-DD  LM-DD
CALLS  'FMPRT'  USING PRNT-CARDS  CC
```

(3) Program Description. Entry into LMLPNMØ is through a CALL from LMLPMOV. A Computed Go To in paragraph NUM-FLD2-RTN determines where the logic flow goes next dependent upon field2 type. A switch is set according to the field type of field2 and logic flow is returned to LMLPMOV for the calling of LMLPNM1, 2, or 3 for the actual COBOL generation.

NUM-FLD2-RTN.
Error checking is performed for illegal receiving field types. A Computed Go To is used for determining the combination type of field2.

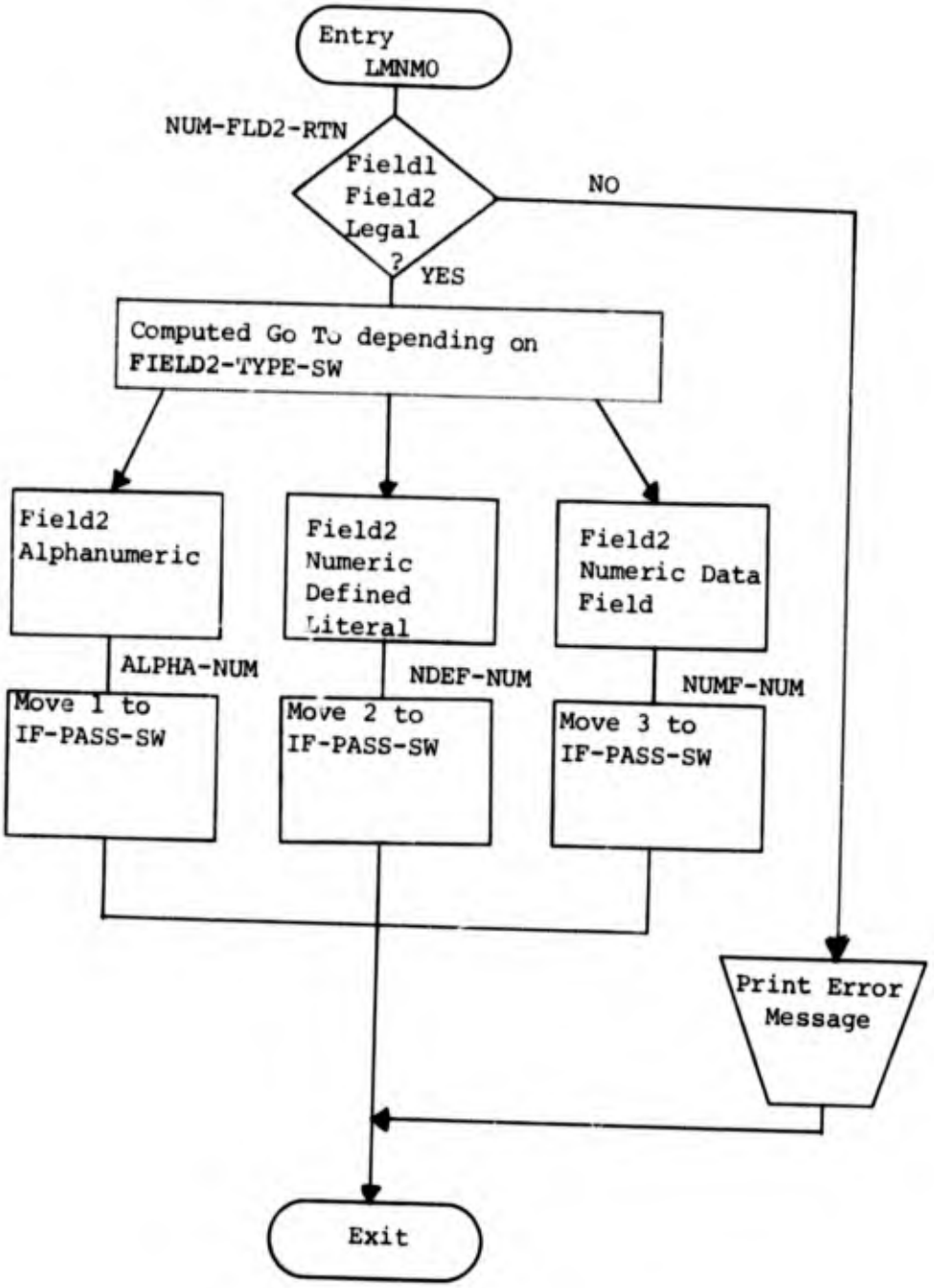
ALPHA-NUM.
A switch is set to indicate which program is needed to generate COBOL coding for an alphanumeric receiving field (IF-PASS-SW=1).

NDEF-NUM.
A switch is set to indicate which program is needed to generate COBOL coding for a numeric defined literal as a receiving field (IF-PASS-SW=2).

NUMF-NUM
A switch is set to indicate which program is needed to generate COBOL coding for a numeric data field as a receiving field (IF-PASS-SW=3).

RKA-TEST-EXIT.
Returns logic flow to LMLPMOV which will call the required program to generate the COBOL coding dependent upon IF-PASS-SW.

(4) Limitations. None.


Entry
LMNM0
NUM-FLD2-RTN
Field1
Field2
Legal
?
NO
YES
Computed Go To depending on
FIELD2-TYPE-SW
Field2
Alphanumeric
Field2
Numeric
Defined
Literal
Field2
Numeric Data
Field
ALPHA-NUM
NDEF-NUM
NUMF-NUM
Move 1 to
IF-PASS-SW
Move 2 to
IF-PASS-SW
Move 3 to
IF-PASS-SW
Print Error
Message
Exit

p. LMLPNM1

(1) Function. Handles a portion of the MOVE NUMERICS instruction. Generates COBOL code for an alphanumeric receiving field.

(2) Calling Sequence.

```
ENTRY  'LMNM1'  USING FM-DD  LP-DD  LM-DD
CALL   'LMPRO'  COBOL-FORMAT1,2,3
CALL   'FMPRT'  PRNT-CARDS  CC
CALL   'LMDAT'  FM-CARD
```

(3) Program Description. Entry is made due to a call from LMLPMOV. In paragraph CK-PARTIAL-FLD2 a Computed Go To determines which paragraph the logic flow will go to depending on field1 type. See the 'MOVE NUM TABLE' for paragraph names which actually generate the code. Some paragraphs are broken into segments. In these cases an alphabetic character is attached to the paragraph number (i.e. LMR-3A). These breakdowns are made to take advantage of commonly generated statements which may be generated by PERFORMing a paragraph rather than duplicating it.

PARTIAL-IN-FLD2.
Produces Procedure Division coding for the second half of two lines of coding needed for a partial in field2.

DATA-GENERATION-1.
Generates a Data Division statement to define an alphanumeric literal.

DG-REDEFINES-1.
Generates a Data Division statement to redefine the alphanumeric literal as a signed numeric literal.

DATA-GENERATION-2.
Same as DATA-GENERATION-1.

DG-REDEFINES-2.
Same as DG-REDEFINES-1.

DG-SINGLE-1.
Generates a Data Division statement to define a signed numeric literal.

PERIOD-GEN-1 thru PERIOD-GEN-1-EXIT.
Places a period at the end of a variable length field name when that field name is the end of a COBOL statement requiring a period for field1.

PERIOD-GEN-2 thru PERIOD-GEN-2-EXIT.
Same as above except it is used for field2.

COBOL-OUT1.
COBOL-OUT2.
COBOL-OUT3.
Generates the printed COBOL generated output line.

CK-PARTIAL-FLD2.
Contains program entry point. A Computed Go To is used to determine the coding which will be generated dependent upon the field type of field1 and an alphanumeric receiving field.

LNR-3 thru LNR-11
Generates COBOL coding for the various combinations of field1 and 2 types. See table.

RKH-TEST-EXIT.
Returns logic flow to LMLPMOV.

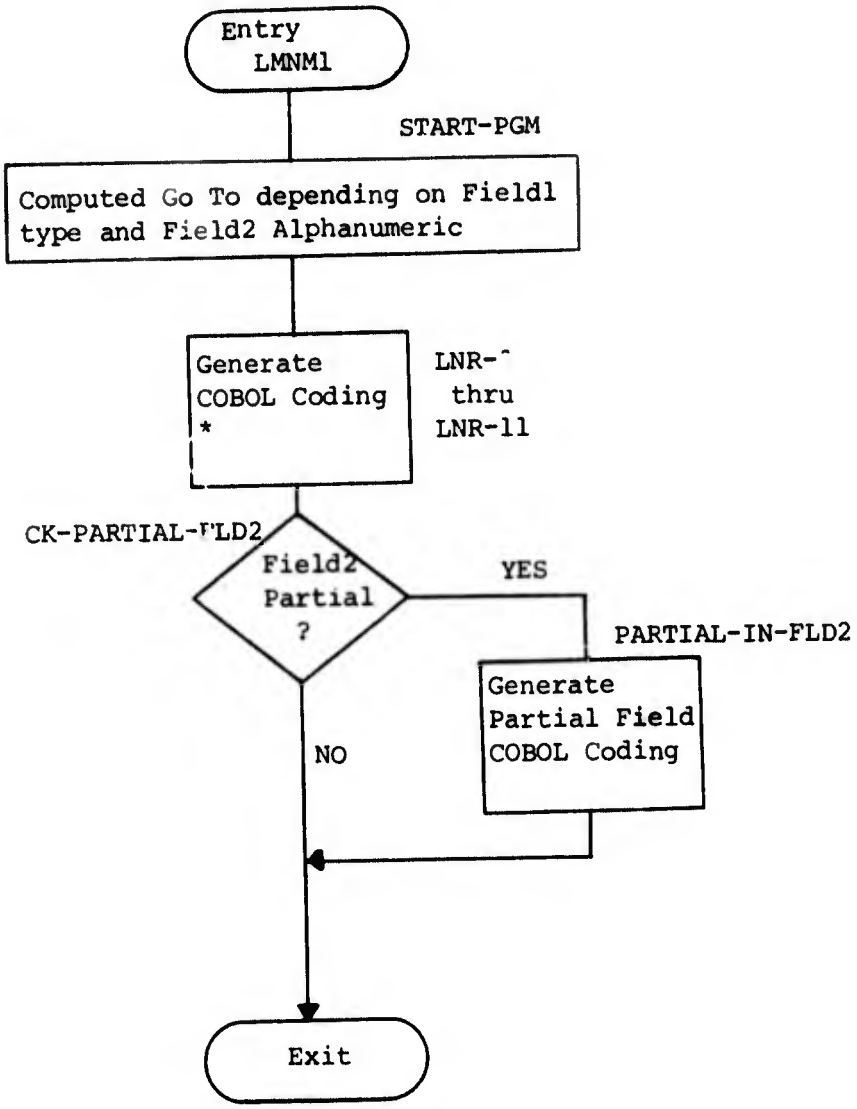
MOVE NUM TABLE
This table shows the paragraph names which generate COBOL coding for the MOVE instruction format:

{MOVE / PUT} {NUM / NUMERICS} field1 {TO / INTO} field2

FIELD1 \ FIELD2	A	L	ND	NL	N
A	LNR-3	ERROR	LNR-7	ERROR	LNR-6
L	ERROR	ERROR	ERROR	ERROR	ERROR
ND	LNR-11	ERROR	LNR-13	ERROR	LNR-12 THRU LNR-12A
NL	LNR-4	ERROR	LNR-16	ERROR	LNR-5
N	LNR-8 THRU LNR-8D	ERROR	LNR-10	ERROR	LNR-9 THRU LNR-9A

NOTE: The table is derived from a combination of paragraph names from three different programs: LMLPNM1, LMLPNM2, and LMLPNM3.

(4) Limitations. None.



*SEE MOVE NUM TABLE ON PREVIOUS PAGE

q. LMLPNM2

(1) Function. Handles the MOVE NUMERICS instruction. Generates code for a numeric defined literal receiving field.

(2) Calling Sequence.

```
ENTRY  'LMNM2'  USING FM-DD  LP-DD  LM-DD
CALL   'LMPRO'  USING COBOL-FORMAT1,2,3
CALL   'FMPRT'  USING PRNT-CARDS  CC
CALL   'LMDAT'  USING FM-CARD
```

(3) Program Description. Entry is made through a call from LMLPMOV. In paragraph CK-PARTIAL-FLD2 a Computed Go To determines which paragraph the logic flow will go to depending on field1 type. See the 'MOVE NUM TABLE' for paragraph names which actually generate the code. Some paragraphs are broken into segments. In these cases an alphabetic character is attached to the paragraph number (i.e. LMR-3A). These breakdowns are made to take advantage of commonly generated statements which may be generated by PERFORMing a paragraph rather than duplicating it.

PARTIAL-IN-FLD2.
DATA-GENERATION-1.
DG-REDEFINES-1.
DG-SINGLE-1.
PERI D-GEN-2 thru PERIOD-GEN-2-EXIT.
COBOL-OUT1.
COBOL-OUT2.
COBOL-OUT3.
All above paragraphs perform the same function as in LMLPNM1.
CK-PARTIAL-FLD2.
Same as LMLPNM1 except field2 will be a numeric defined literal.
LNR-6A thru LNR-16.
Generates COBOL coding depending upon field1 and field2 types. See table.

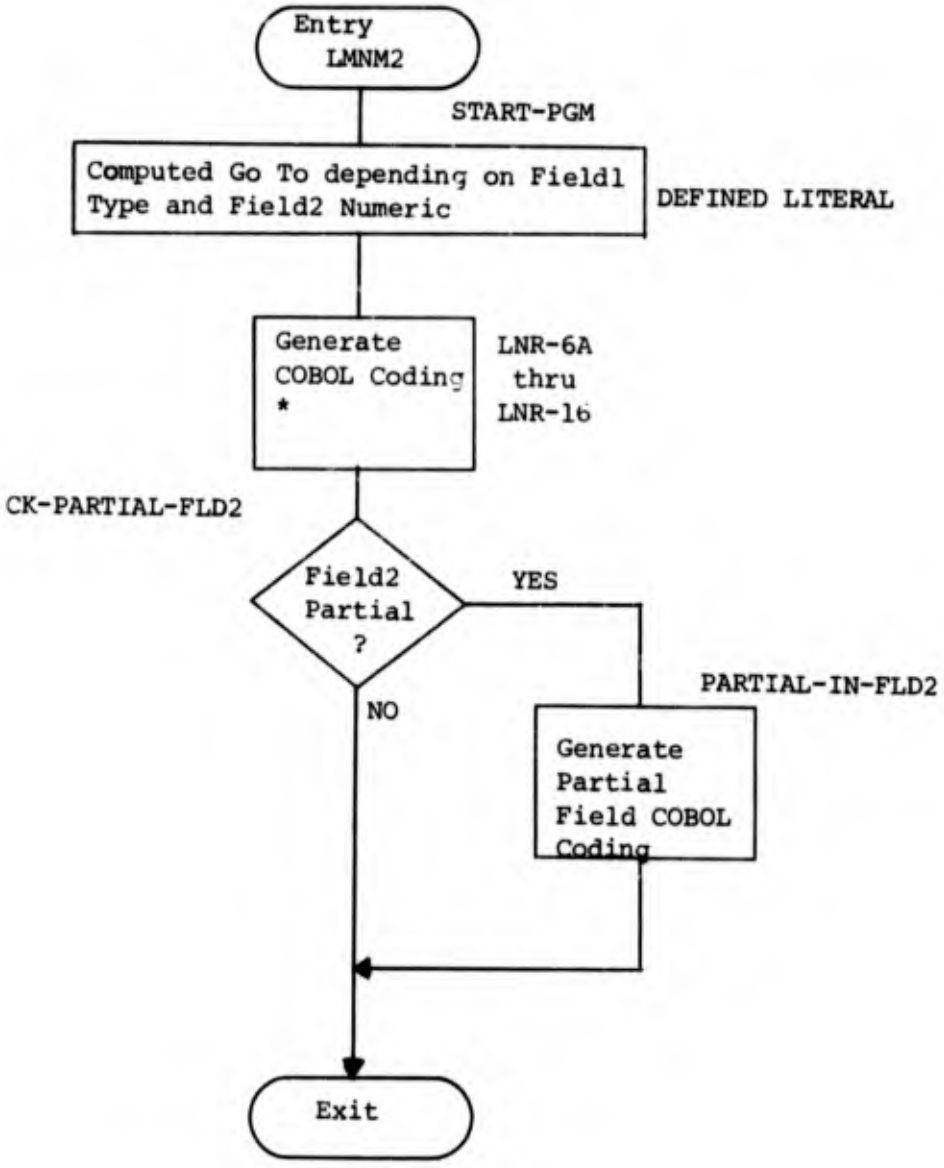
MOVE NUM TABLE

This table shows the paragraph names which generate COBOL coding for the MOVE instruction format:

{MOVE / PUT} {NUM / NUMERICS} field1 {TO / INTO} field2

See LMLPNM1 for MOVE NUM TABLE.

(4) Limitations. None.



*SEE MOVE NUM TABLE UNDER LMLPNM1

r. LMLPNM3

(1) Function. Handles the MOVE NUMERIC instruction. Generates coding for a numeric data field as a receiving field.

(2) Calling Sequence.

```
ENTRY  'LMNM3'  USING FM-DD  LP-DD  LM-DD
CALL   'LMPRO'  USING COBOL-FORMAT1,2,3
CALL   'FMPRT'  USING PRNT-CARDS  CC
CALL   'LMDAT'  USING FM-CARD
```

(3) Program Description. Entry is made due to a call from LMLPMOV. In paragraph CK-PARTIAL-FLD2 a Computed Go To determines which paragraph the logic flow will go to depending on field1 type. See the 'MOVE NUM TABLE' for paragraph names which actually generate the code. Some paragraphs are broken into segments. In these cases an alphabetic character is attached to the paragraph number (i.e., LMR-3A). These breakdowns are made to take advantage of commonly generated statements which may be generated by PERFORMing a paragraph rather than duplicating it.

PARTIAL-IN-FLD2.
DATA-GENERATION-1.
DG-REDEFINES-1.
DG-SINGLE-1.
PERIOD-GEN-2 thru PERIOD-GEN-2-EXIT.
COBOL-OUT1.
COBOL-OUT2.
COBOL-OUT3.

All above paragraphs perform the same functions as in LMLPNM1.

CK-PARTIAL-FLD2.

Same as LMLPNM1 except field2 will be a numeric data field.

LNR-5 thru LNR-12A

Generates COBOL coding depending upon field1 and 2 types. See table.

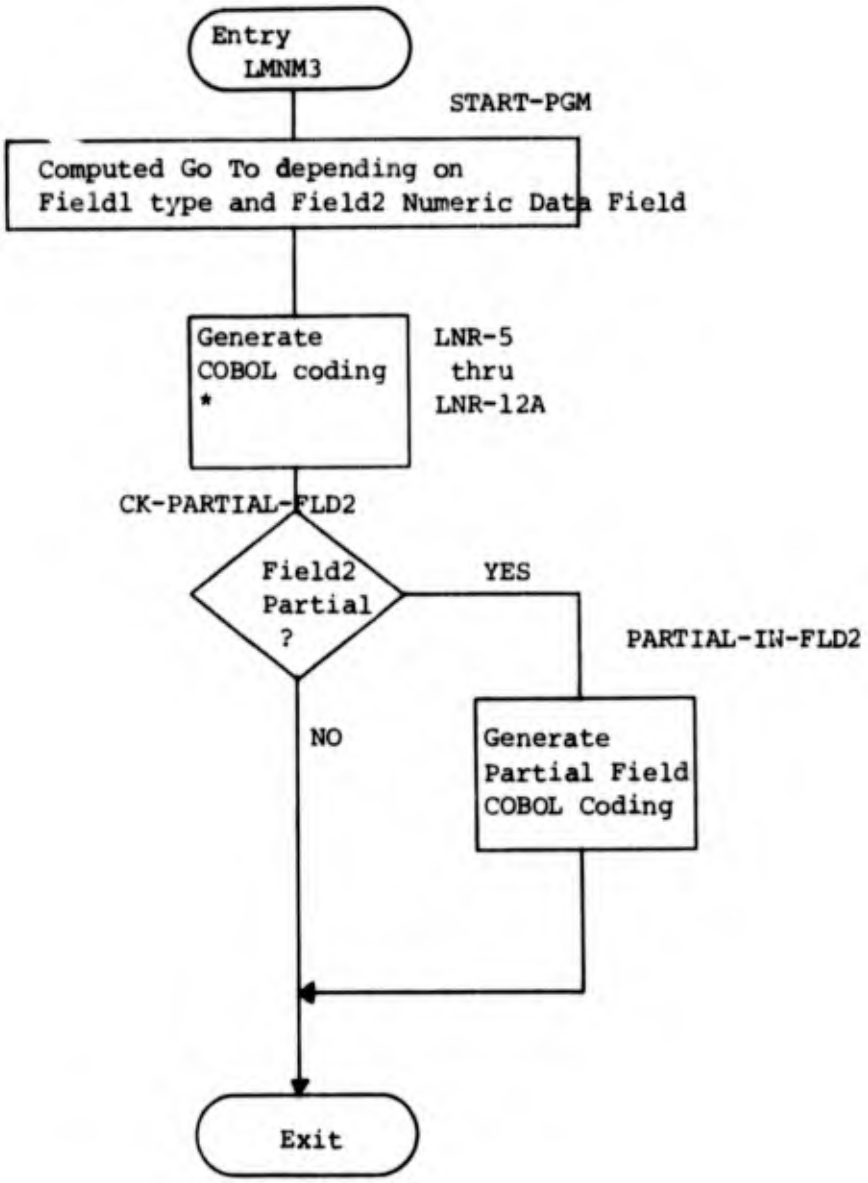
MOVE NUM TABLE

This table shows the paragraph names which generate COBOL coding for the MOVE instruction format:

{MOVE / PUT} {NUM / NUMERICS} field1 {TO / INTO} field2

See LMLPNM1 for MOVE NUM TABLE.

(4) Limitations. None.



*SEE MOVE NUM TABLE UNDER LMLPNM1.

s. LMLPPG1

(1) Function. This program generates the initial part of the generated COBOL program. It generates that code necessary (depending on the run type) through the WORKING-STORAGE SECTION.

(2) Calling Sequence.

```
ENTRY  'LMLP1'   USING LP-DD  ML-DD
CALLS  'LMLPGEN' USING GENERATED-PROG
```

(3) Program Description.

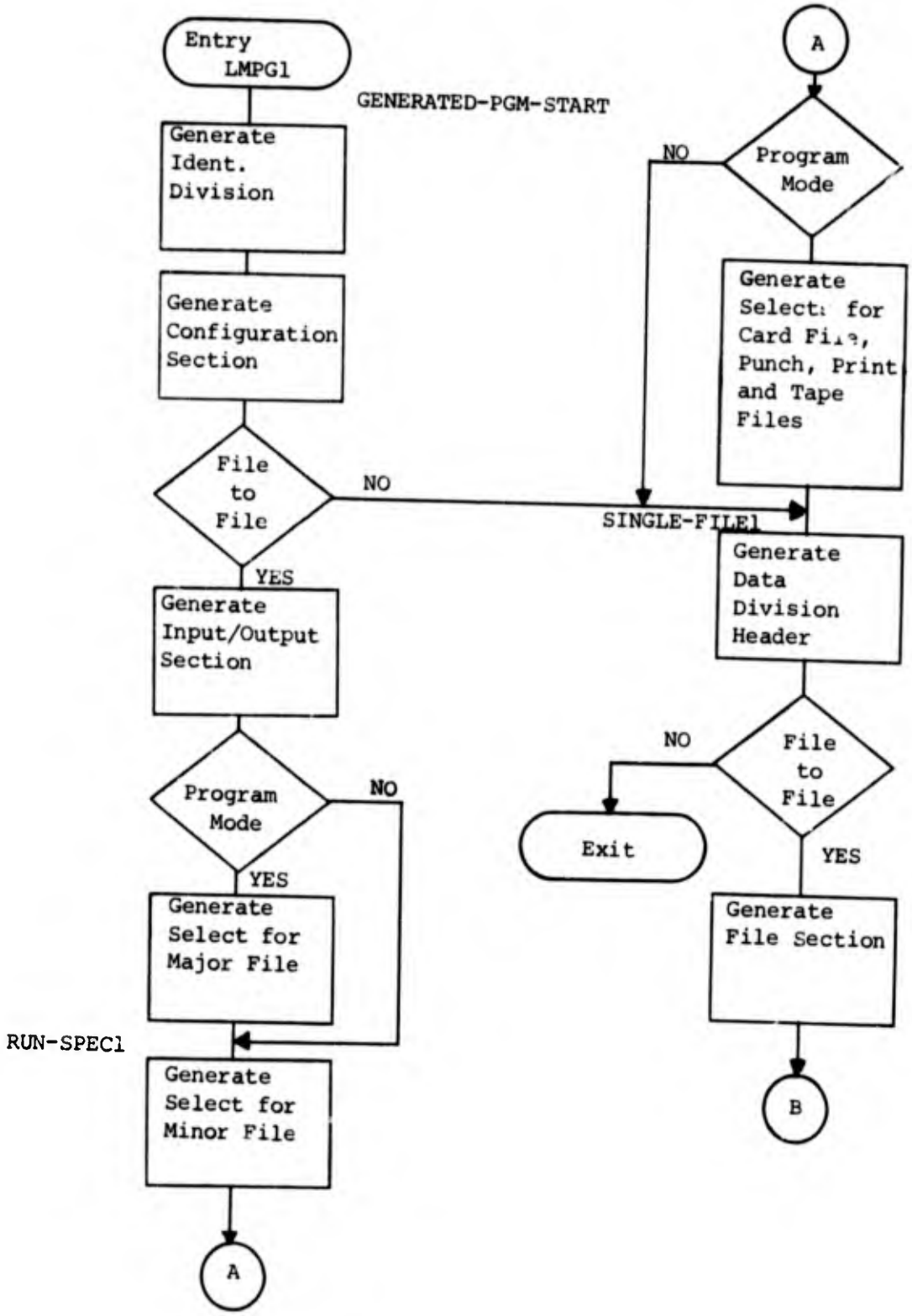
GENERATED-PGM-START
Generates Identification Division, Configuration Section, Input-Output Section and Select Clause for major file.

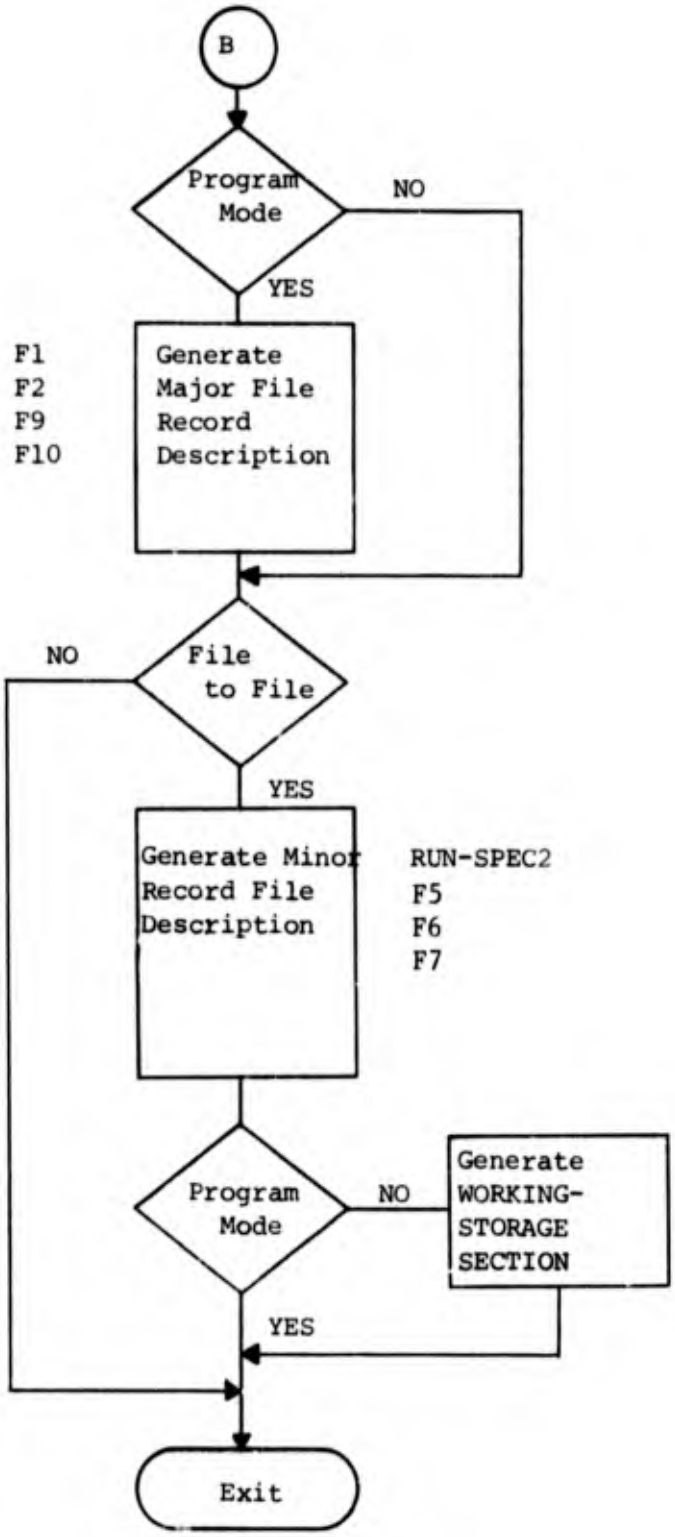
RUN-SPEC1
Generates select clause for minor-file, card file, punch file, tape file.

SINGLE-FILE1
Generates DATA DIVISION header. Generates FILE SECTION.

F1,F2,F9,F1Ø
Generates the major record file description RUN-SPEC2,F5-F6,FY. Generates the minor record file description. Generates the WORKING-STORAGE SECTION.

(4) Limitations. None.


Entry
LMPG1
GENERATED-PGM-START
Generate
Ident.
Division
Generate
Configuration
Section
File
to
File
NO
YES
SINGLE-FILE1
Generate
Input/Output
Section
Program
Mode
NO
YES
Generate
Select for
Major File
RUN-SPEC1
Generate
Select for
Minor File
A
A
Program
Mode
NO
Generate
Select for
Card File,
Punch, Print
and Tape
Files
Generate
Data
Division
Header
File
to
File
NO
YES
Exit
Generate
File Section
B


B
Program Mode
NO
YES
F1
F2
F9
F10
Generate Major File Record Description
NO
File to File
YES
Generate Minor Record File Description
RUN-SPEC2
F5
F6
F7
Program Mode
NO
Generate WORKING-STORAGE SECTION
YES
Exit

t. LMLPPG2

(1) Function. This program generates the remainder of the fixed portion of the COBOL program down through initialization in the Procedure Division. If it is a single-file run, a LINKAGE SECTION and ENTER statements are generated.

(2) Calling Sequence.

```
ENTRY 'LMPG2'   USING LP-DD  LM-DD
CALLS 'LMLPGEN' USING GENERATED-PROG
```

(3) Program Description. A Computed Go To is executed upon entry and a branch taken to GEN-LINKAGE for a file-to-file run subroutine mode (TYPE1), F8 for a file-to-file run program mode (TYPE2), or SINGLE-FILE2 for a single file run (TYPE3).

F8,F3,F4.
Generate card, punch, print, and tape file record descriptions.

SINGLE-FILE2.
Generates the WORKING-STORAGE SECTION header.

F11.
SINGLE FILE3.
SINGLE FILE5.
Generates the system defined Data Division names (SWITCH A-Z, 0-9, $$ORC, $$NRC, etc.) for TYPE2 runs. Generates the Data Division entry for the fixed portion of the major record. Generates the PROCEDURE DIVISION header; opens card, print, punch files.

INIT-PARA.
Generates code to initialize values.

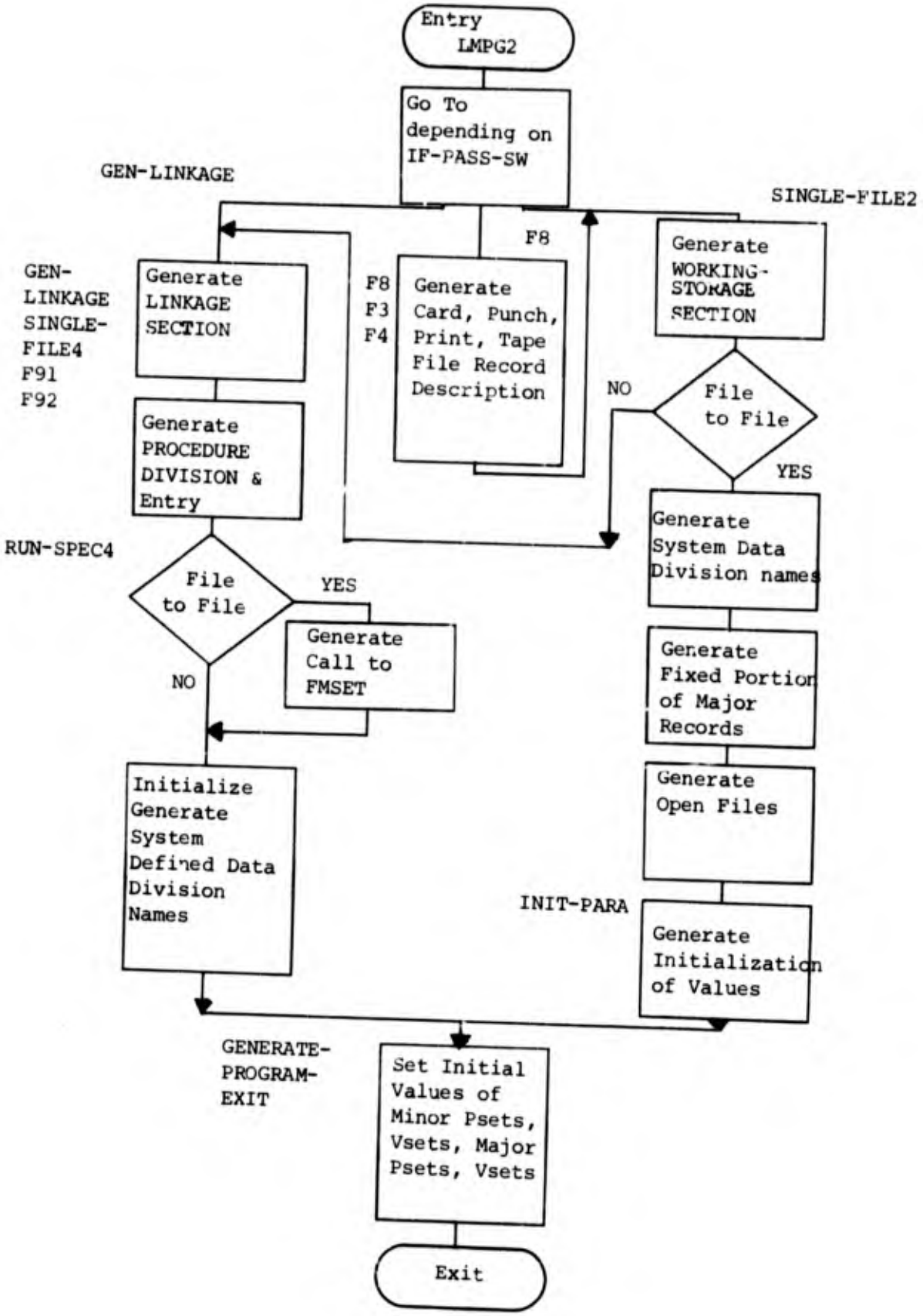
GEN-LINKAGE.
SINGLE-FILE4.
F92.
Generates the necessary LINKAGE-SECTION code for TYPE1 and TYPE3 runs. Generates PROCEDURE DIVISION header and entry Logic Package 'name' using parameters.

RUN-SPEC4.
Generates code to initialize values.

GENERATED-PROGRAM-EXIT
Determines initial values for number of minor psets and vsets, and major psets and vsets.

GEN-SYS-DATES.
Determines the Julian date of the execution run and converts it into a year, month, day format.

(4) Limitations. None.


Entry
LMPG2
Go To
depending on
IF-PASS-SW
GEN-LINKAGE
SINGLE-FILE2
F8
GEN-
LINKAGE
SINGLE-
FILE4
F91
F92
Generate
LINKAGE
SECTION
F8
F3
F4
Generate
Card, Punch,
Print, Tape
File Record
Description
Generate
WORKING-
STORAGE
SECTION
NO
File
to File
YES
Generate
PROCEDURE
DIVISION &
Entry
Generate
System Data
Division names
RUN-SPEC4
File
to File
YES
Generate
Call to
FMSET
NO
Generate
Fixed Portion
of Major
Records
Initialize
Generate
System
Defined Data
Division
Names
Generate
Open Files
INIT-PARA
Generate
Initialization
of Values
GENERATE-
PROGRAM-
EXIT
Set Initial
Values of
Minor Psets,
Vsets, Major
Psets, Vsets
Exit

u. LMLPPRT

(1) Function. This program handles the PRINT and XPRINT instructions.

(2) Calling Sequence.

```
ENTRY  'LMPRT'  USING FM-DD  LP-DD  LM-DD
CALL   'LMFLD'  USING FM-DD  LP-DD  LM-DD
CALL   'LMPRO'  USING PROCEDURE-RECORD
CALL   'FMPRT'  USING PRNT-CARDS CC
CALL   'LMDAT'  USING FM-CARD
```

(3) Program Description.

GENERATE-PRINT.

In this paragraph DSN-PTR is used to accumulate the total number of characters to be printed per line (132 maximum). Begins processing the 'PRINT' verb by developing entries in the print-setup tables from the first operand of the print verb.

GPR-OPERAND-CYCLE.

All fields are presumed alpha until proven numeric.

GPR-LIT-OPERAND.

Builds the print-setup tables from operands 2,3... of the print verb. When control is transferred to GPR-PRODUCE-SETUPS, these tables are complete.

SETUP-TABLE-DESCRIPTION.

GPR-PRODUCE-SETUPS.

Generates setup statements from the setup tables.

GPR-PRODUCE-ONE-SETUP.

Generates one COBOL setup statement out of the setup-tables.

GPR-PRODUCE-PRINT-LIT.

Produces the level 01 data definition for the literals destined to be moved to the print-area. It also directs the production of level 02 data definitions and saves LIT-XXXX, the numeric part of the level 01 data name, for subsequent use in GENERATE-PRINT-PROPER.

GPR-PRODUCE-PRINT-LIT-02.

Generates a level 02 data definition for each entry in the setup tables for which such an entry is needed. When length is given as zero, the data definition is not needed.

GENERATE-PRINT-PROPER.

Generates the last two lines of COBOL output for the print verb.

GPR-SETUP-ONE-FIELD.

Handles partial fields in the print statement.

CHECK-SPECIAL-FIELD.

Tests for a special literal and systems references (i.e., $$ORC, $$NRC, etc.) and converts numeric literals to alphanumeric literals for printing.

BUILD-NUM-LIT.

Generates the numeric literal for later insertion into a value clause, counting the digits.

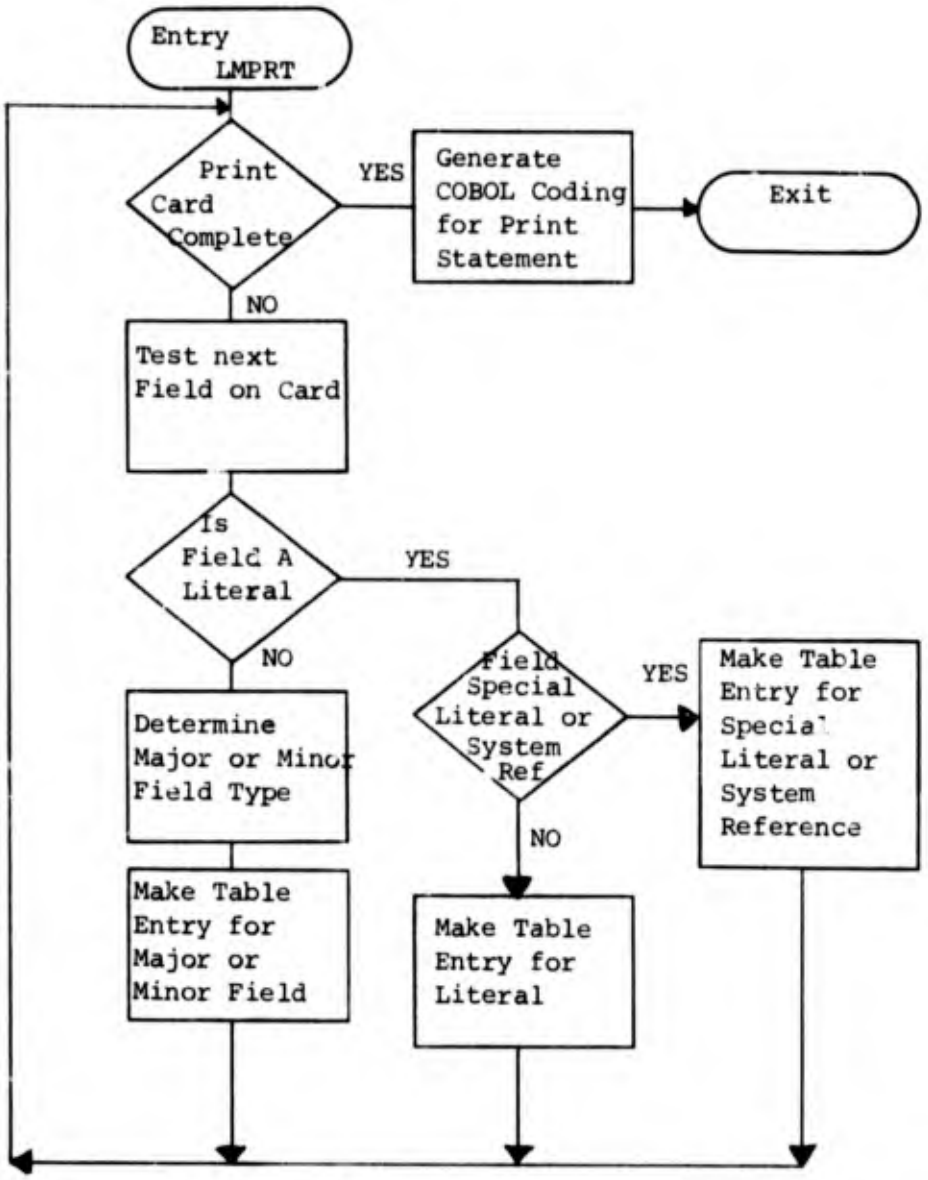
OUTPUT-PROCEDURE.

Outputs one card image to the Procedure Division of the generated program.

INSERT-PERIOD.
INSERT-PERIOD-1.
INSERT-PERIOD-2.
INSERT-PERIOD-EXIT.

The above paragraphs place a period after the right most nonblank character in the sentence.

(4) Limitations. None.


Entry
LMPRT
Print
Card
Complete
YES
Generate
COBOL Coding
for Print
Statement
Exit
NO
Test next
Field on Card
Is
Field A
Literal
YES
NO
Field
Special
Literal or
System
Ref
YES
Make Table
Entry for
Special
Literal or
System
Reference
Determine
Major or Minor
Field Type
NO
Make Table
Entry for
Major or
Minor Field
Make Table
Entry for
Literal

v. LMLPPUT

(1) Function. Processes the PUT/MOVE field instruction LMLPMOV calls LMLPPUT.

(2) Calling Sequence.

```
ENTRY 'LMPUT'  USING FM-DD  LP-DD  LM-DD
CALL  'LMPRT'  USING PRNT-CARDS  CC
CALL  'LMPRO'  USING COBOL-OUT1  COBOL-OUT2  COBOL-OUT3
CALL  'LMDAT'  USING FM-CARD
```

(3) Program Description. Paragraph CK-PARTIAL-FLD2 contains the entry point for the MOVE/PUT field instruction. A Computed Go To determines initially the field2 type of the field combinations needed. Logic flow branches to ALPHA-MOV, NDEF-MOV or NUM-MOV depending on the field type of field2. A Computed Go To in each of the paragraphs determines the final field type and the logic flow branches to the paragraph which will generate COBOL coding for the specific combination of field1 and 2 types. See the MOVE FIELD TABLE for paragraph names of the code generating paragraphs.

PARTIAL-IN-FLD2
Produces a PROCEDURE DIVISION coding for the second half of two statements needed for a partial notation in field2.

DATA-GENERATION-1
Generates DATA DIVISION coding to define an alphanumeric literal.

DG-REDEFINES-1
Generates DATA DIVISION coding to redefine the alphanumeric literal defined by DATA-GENERATION-1 as a signed numeric literal.

DATA-GENERATION-2
Same as DATA-GENERATION-1.

DG-REDEFINES-2
Same as DG-REDEFINES-1.

PERIOD-GEN-1 thru PERIOD-GEN-1-EXIT
Places a period at the end of a variable length field name when that field name is the end of a COBOL statement requiring a period for field1.

PERIOD-GEN-2 thru PERIOD-GEN-2-EXIT
Same as above except it is used for field2.

COBOL-OUT1.
COBOL-OUT2.
COBOL-OUT3.
Generates the printed COBOL generated output line.

CK-PARTIAL-FLD2.
Contains the program entry point. A Computed Go To is used establish a partial combination type for field2.

ALPHA-MOV.
This paragraph uses a Computed Go To to select the paragraph for generating COBOL coding depending upon field1 type

with field2 an alpahanumeric field or defined literal.

NDEF-MOV.

Uses a Computed Go To to select the paragraph necessary for generating COBOL coding determined by field1 type with field2 a numeric defined literal.

NUM-MOV.

Use a Computed Go To to select the paragraph necessary for generating COBOL coding determined by field1 type with field2 a numeric data field.

LFR-2 thru LFR-6

Generates COBOL coding depending on field1 and 2 types. See MOVE field TABLE 1.

RKH-TEST-EXIT.

Returns logic flow to LMLPMOV.

MOVE field TABLE 1

This table shows the paragraph names which generate COBOL coding for the MOVE/PUT instruction format:

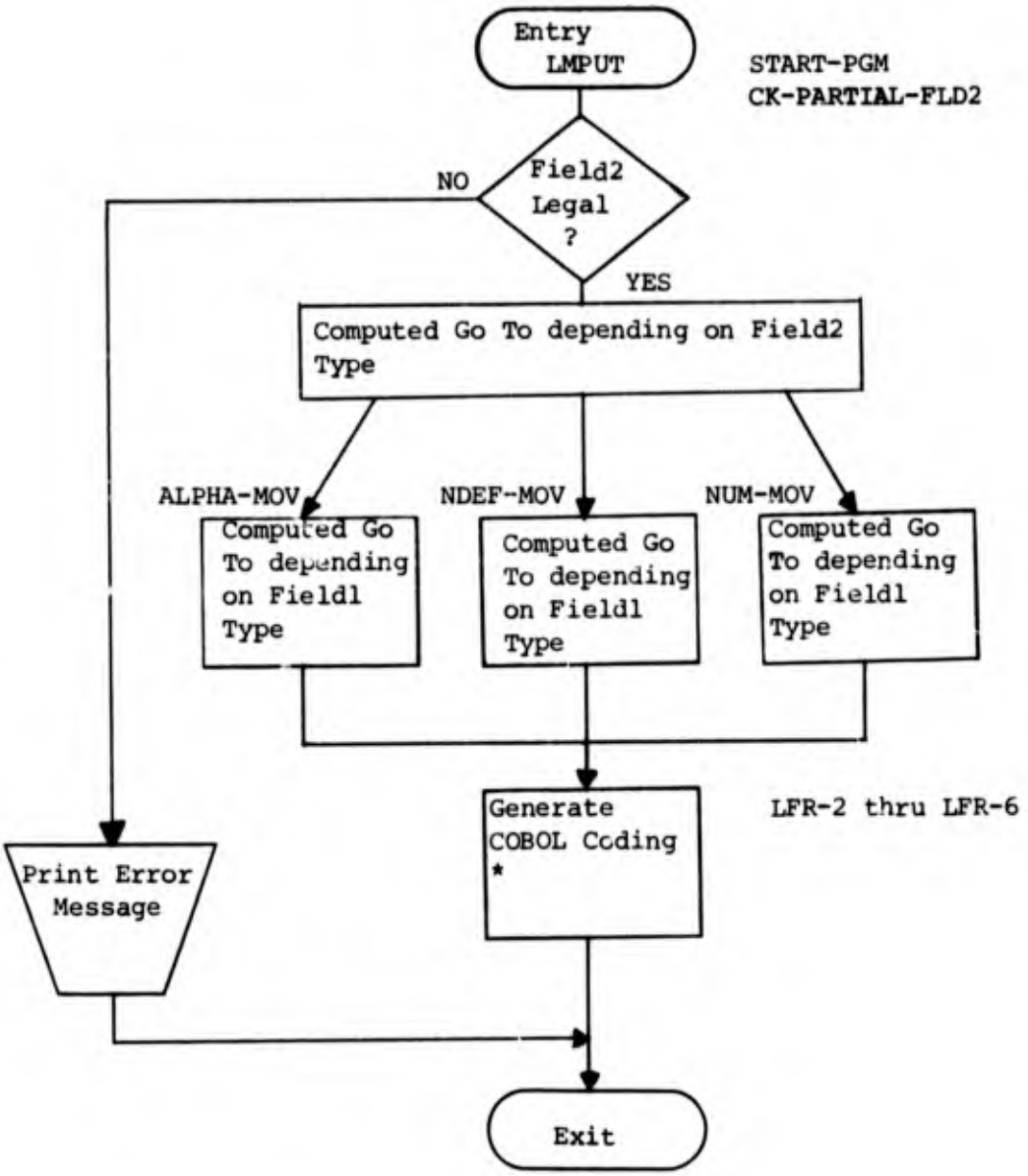
$$\left\{\begin{matrix}\text{MOVE}\\\text{PUT}\end{matrix}\right\} \quad \text{field1} \quad \left\{\begin{matrix}\text{TO}\\\text{INTO}\end{matrix}\right\} \quad \text{field2}$$

FIELD2

FIELD1

	A	N	ND	L	NL
A	LFR-2	LFR-4	LFR-4	ERROR	ERROR
N	LFR-5	LFR-3	LFR-3	ERROR	ERROR
ND	LFR-6	LFR-2	LFR-2	ERROR	ERROR
L	LFR-2	ERROR	ERROR	ERROR	ERROR
NL	ERROR	LFR-2	LFR-2	ERROR	ERROR

(4) Limitations. None.



*SEE MOVE FIELD TABLE ON PREVIOUS PAGE.

w. LMLPRFD

(1) Function. This program generates the code for the IN RANGE and NOT IN RANGE operations. It furthermore is the program that does all the field name validation for the conditionals.

(2) Calling Sequence.

```
ENTRY 'LMRFD'      USING FM-DD  LP-DD  LM-DD
CALL  'LMLPFLD'    USING FM-DD  LP-DD  LM-DD
CALL  'FMPRT'      USING PRNT-CARDS  CC
CALL  'LMLPGEN'    USING IF-CARD1
CALL  'LMLPGEN'    USING IF-CRD3
```

(3) Program Description. If the RANGE statement was found, go to the processing for RANGE. Determine the first field type. If control was passed here for verification of the BIT or PROFILE field go to the exit. The next section of processing locates the second field by checking from right to left determining if there is a failure exit and/or partial field notation. The second field is now validated and control is passed back to LMRMN.

GENERATE-IF-CODE.
Generates COBOL code before the 'IF' conditional.

IF-END.
Generates COBOL code after teh 'IF' conditional.

GENERATE-OR-CODE.
Generates COBOL code before the 'OR' conditional.

OR-END.
Generates COBOL code after the 'OR' conditional.

GENERATE-AND-CODE.
Generates COBOL code before the 'AND' conditional.

AND-END.
Generates COBOL code after the 'AND' conditional.

GENERATE-ORIF-CODE.
Generates COBOL code before the 'OR IF' conditional.

ORIF-END.
Generates the COBOL code after the 'OR IF' conditional.

FIELD-GENER.
Put the word pointer back on the field name then verify that field. A unique value is accumulated which will signify RANGE or NOT RANGE and if the first part of the range is zero. Control is passed to the appropriate paragraph for code generation.

PARTIAL-TYPE.
Checks to insure that no partial notation is used on a RANGE card.

RANGE-NO-ZERO
BREAK1
BREAK3
BREAK4

Generates the code for IN RANGE NNNN-MMMM where NNNN is greater than zero.

NOT-RANGE-NO-ZERO
BREAK2

Generates the code for NOT IN RANGE NNNN-MMMM where NNNN is greater than zero.

RANGE-ZERO.

Generates the code for IN RANGE NNNN-MMMM where NNNN is equal to zero.

NOT-RANGE-ZERO

Generates the code for NOT IN RANGE NNNN-MMMM where NNNN is equal to zero.

CREATE-QUOTES
QUOTES1

Generates the right quote closed up to the last character.

FM-DETERMINE-TYPE-FIELD

Determines if the field begins with a + or - sign and if it is a number.

IF-NUM-NEG-POS.

Sets negative or positive switch based on field.

IF-NUM-LIT-OUT

Determines if the field is a special literal (in form LIT-NNNN), i.e., $$ORC, $$NRC, $$YMD, or $$DAT.

IF-PARTIAL-FIELD-FORMAT.

Determines if the word being examined is in the form NNNN-MMMM, where NNNN and MMMM are numeric and NNNN is less than MMMM.

IF-PASS.

When none of the previous field types have been found, the only valid type field is a define or data field. LMFLD is called to verify the field. If a valid field is found, the desired field name is generated in the output area.

IF-NUM-LIT

Verifies that each character is numeric and moves the input to the output area without the plus sign if one is present.

CHECK-CARD-TYPE-SWITCHES.

Does a Computed Go To depending on the conditional word and generates COBOL code after the relational testing.

IF-SEQ1-GENER.

This paragraph outputs COBOL coding for field1.

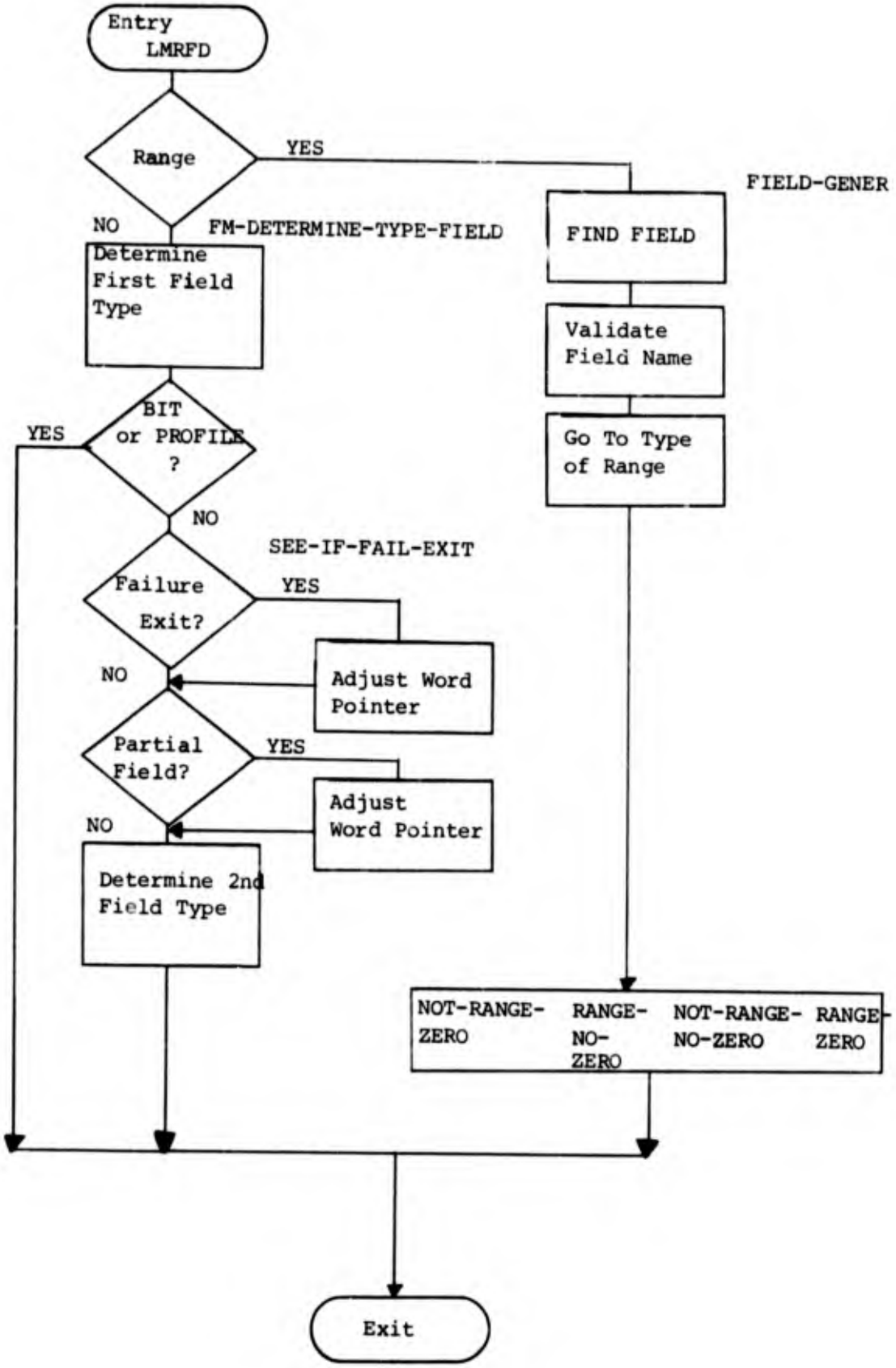
IF-SEQ3-GENER.

Outputs COBOL coding for both field1 and field2 when their format does not fit in the previous output area.

GENERATE-PARTIAL-MOVE.

Generates the moves when a partial field is specified so the literals can be used in place of the fields.

(4) Limitations. None.


Entry
LMRFD
Range
YES
FIELD-GENER
NO
FM-DETERMINE-TYPE-FIELD
FIND FIELD
Determine
First Field
Type
Validate
Field Name
BIT
or PROFILE
?
YES
Go To Type
of Range
NO
SEE-IF-FAIL-EXIT
Failure
Exit?
YES
NO
Adjust Word
Pointer
Partial
Field?
YES
NO
Adjust
Word Pointer
Determine 2nd
Field Type
NOT-RANGE-
ZERO
RANGE-
NO-
ZERO
NOT-RANGE-
NO-ZERO
RANGE-
ZERO
Exit

x. LMLPRLN

(1) Function. This program validates the relational words and generates a standard relation depending on the combination of relational words.

(2) Calling Sequence.

```
ENTRY  'LMRLN'  USING FM-DD  LP-DD  LM-DD
CALLS  'FMPRT'  USING PRNT-CARDS  CC
```

(3) Program Description.

RESTORE-TEMP
This paragraph determines the actual number of words in the relational, it must be less than 7.

FM-RELATION-CHECK
Rejects the invalid combination of fields such as comparing two numeric literals.

SCAN-RELATION
Controls stepping through the relation words to determine if they are valid.

RELATION-LU
Performs a table lookup of all the valid relational words and determines if the particular input relation word is valid. If it is valid, the appropriate switch or switches are set. Relations 'IS', 'OR', 'THAN' and 'TO' are considered as noise words and do not set switches.

SET-EQUAL.
Found relational EQUALS, EQUAL ON or EQ.

SET-GREATER.
Found relational GREATER, AFTER, LATER, GT.

SET-LESS.
Found relational LESS, BEFORE, EARLIER, LT.

SET-GE.
Found relational GE.

SET-LE.
Found relational LE.

SET-NE.
Found relational NE.

SET-NOT.
Found relational NOT.

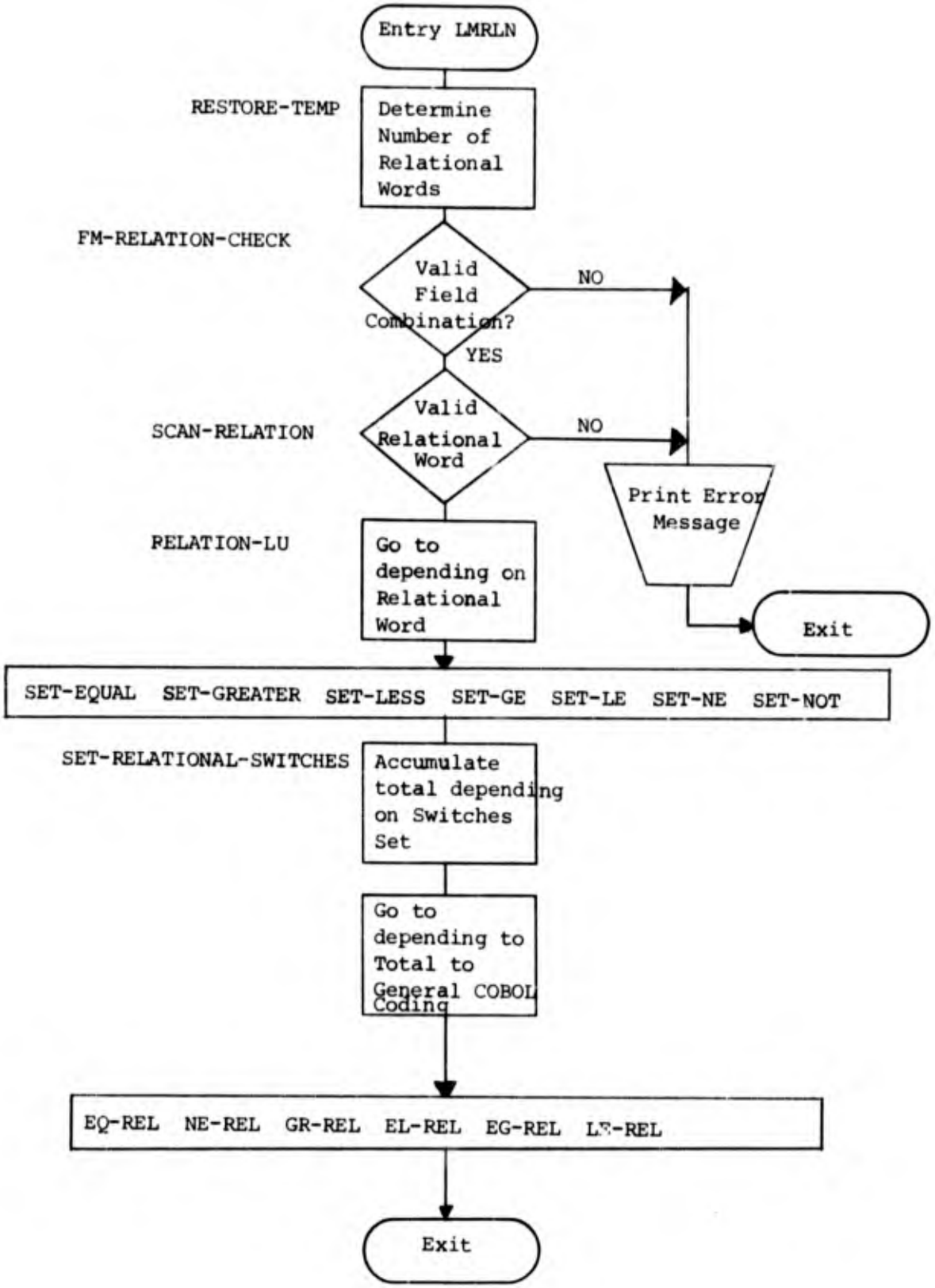
SET-RELATION-SWITCHES.
Accumulates a total by adding a different value for each of the four switches. By this method a unique value is reached for each group of relationals. A standard relational term is generated depending on this accumulated total.

EQ-REL.
Generates code for IS EQUAL TO.

NE-REL.
Generates code for NOT EQUAL TO.

GR-REL.
     Generates code for IS GREATER THAN.
EL-REL.
     Generates code for IS NOT GREATER THAN.
EG-REL.
     Generates code for IS NOT LESS THAN.
LE-REL.
     Generates code for IS LESS THAN.

(4) Limitations. None.


Entry LMRLN
RESTORE-TEMP
Determine Number of Relational Words
FM-RELATION-CHECK
Valid Field Combination?
NO
YES
SCAN-RELATION
Valid Relational Word
NO
Print Error Message
RELATION-LU
Go to depending on Relational Word
Exit
SET-EQUAL SET-GREATER SET-LESS SET-GE SET-LE SET-NE SET-NOT
SET-RELATIONAL-SWITCHES
Accumulate total depending on Switches Set
Go to depending to Total to General COBOL Coding
EQ-REL NE-REL GR-REL EL-REL EG-REL LE-REL
Exit

y. LMLPRMN1

(1) Function. LMLPRMN1 is the supervisor for all of the conditional statements beginning with IF, AND, OR or OR IF. It controls the processing of all these types of statements by calling the appropriate subprogram to check for keywords and code generation.

(2) Calling Sequence.

```
ENTRY 'LMRMN'   USING FM-DD  LP-DD  LM-DD
CALL  'LMLPRMS' USING FM-DD  LP-DD  LM-DD
CALL  'LMLPSPØ' USING FM-DD  LP-DD  LM-DD
CALL  'LMLPRFD' USING FM-DD  LP-DD  LM-DD
CALL  'LMLPRLN' USING FM-DD  LP-DD  LM-DD
CALL  'LMLPRT1' USING FM-DD  LP-DD  LM-DD
CALL  'LMLPRT2' USING FM-DD  LP-DD  LM-DD
CALL  'LMLPTAB' USING FM-DD  LP-DD  LM-DD
```

(3) Program Description.

FM-IF
FM-AND
FM-OR

Is the supervisor for conditional statements. It contains the entry points for IF, AND, OR and OR IF statements and sets a switch depending on which condition is indicated.

INITIAL-CARD
INITIAL-CARD1

Initialization of counters and switches.

START-PROCESSING

Calls LMRMS to check for third and fourth keywords (RANGE, TABLE, DELETED RECORD, FIND RECORD, GENERATED RECORD, EOF). Testing of MAJOR-SW2 tells whether keywords have been found and appropriate code generated. If TABLE was found, LMTAB is called to generate the appropriate code. If RANGE was found, LMRFD is called to verify the field name and generate the appropriate code.

CALL-BSP

If none of the above keywords were found LMSPØ is called. This checks for BIT, PROFILE and SWITCH field name verification must be done by calling LMRFD. Code for SWITCH is generated here and code for BIT and PROFILE is generated by calling LMSP1.

CALL-RLN

Calls LMRLN to verify the relational words. LMRT1 is called and depending on the field combinations code is generated. If the combinations are not found in LMRT1, LMRT2 is called to generate the code.

FM-IF-EXIT.

This paragraph checks for failure exit.

CALL-LMLPTAB.

Calls LMLPTAB if a TABLE-LOOKUP is indicated.

CALL-DETERMINE-FLD

Calls LMRFD to generate the code for RANGE operations and to validate all fields. In the case of BIT and PROFILE return goes back to LMSPØ otherwise it goes to LMRLN.
Each of the subprograms called by LMRMN contains much duplicated coding. For this reason, these commonly used paragraphs will be discussed separately form particular subprograms. Each of the conditionals are controlled by truth switches (TRUTH-Ø, TRUTH-1, and TRUTH-2). It need not concern us as to the specific function of each but only that collectively they govern the status of the total condition at any one time, and indicate whether following conditions should even be tested (i.e. with an IF conditional and an AND conditional, if the IF is false there is no point in testing the AND statement). Depending on the type of conditional, certain switches are initialized preceding the actual conditional statement and switches set according to the truth of the condition.

GENERATE-IF-CODE
GENERATE-OR-CODE
GENERATE-AND-CODE
GENERATE-ORIF-CODE

These paragraphs perform the initialization function of the truth switches depending on the particular conditional.

IF-END
OR-END
AND-END
ORIF-END

These paragraphs perform the setting of the truth switches depending on the truth of the total condition so far.
When the exact code to be generated has been determined, GENER-FIRST-PART thru GENER-FIRST-PART-EXIT is performed and this generates the appropriate initialization of the truth switches. The code for the logic statement is then generated. Finally, CHECK-CARD-TYPE-SWITCHES thru CHECK-END is performed to generate the code to set the truth switches based upon the truth of the condition. Other common paragraphs include:

SEE-IF-FALL-EXIT

Which determines if there is a failure exit on the statement and if so, generate the appropriate code. It also sets MAJOR-SW2 which will indicate that processing is complete for the card.

IF-SEQ1-GENER
IF-SEQ2-GENER
IF-SEQ3-GENER

These all produce the Procedure Division generated code. The only difference is the format of the output.

IF-CLOSE-UP
IF-CLOSE-UP1
These paragraphs close up the spaces between the last word generated and a period or right parenthesis when needed.

SWITCHES USED IN RELATIONAL TESTING:

FM-IF-SW
=1 Initiation of IF condition

FIRST-PART-SW
=1 AND conditional
=2 OR conditional
=3 OR IF conditional
=4 IF conditional

IF-FIELD-SW
=1 First field
=2 Second field

PARTIAL-SW
=1 Second field is partial

IF-PARTIAL-INDICATOR
=1 First field is partial

ERROR-SW
=4 Error encountered; do not generate any code

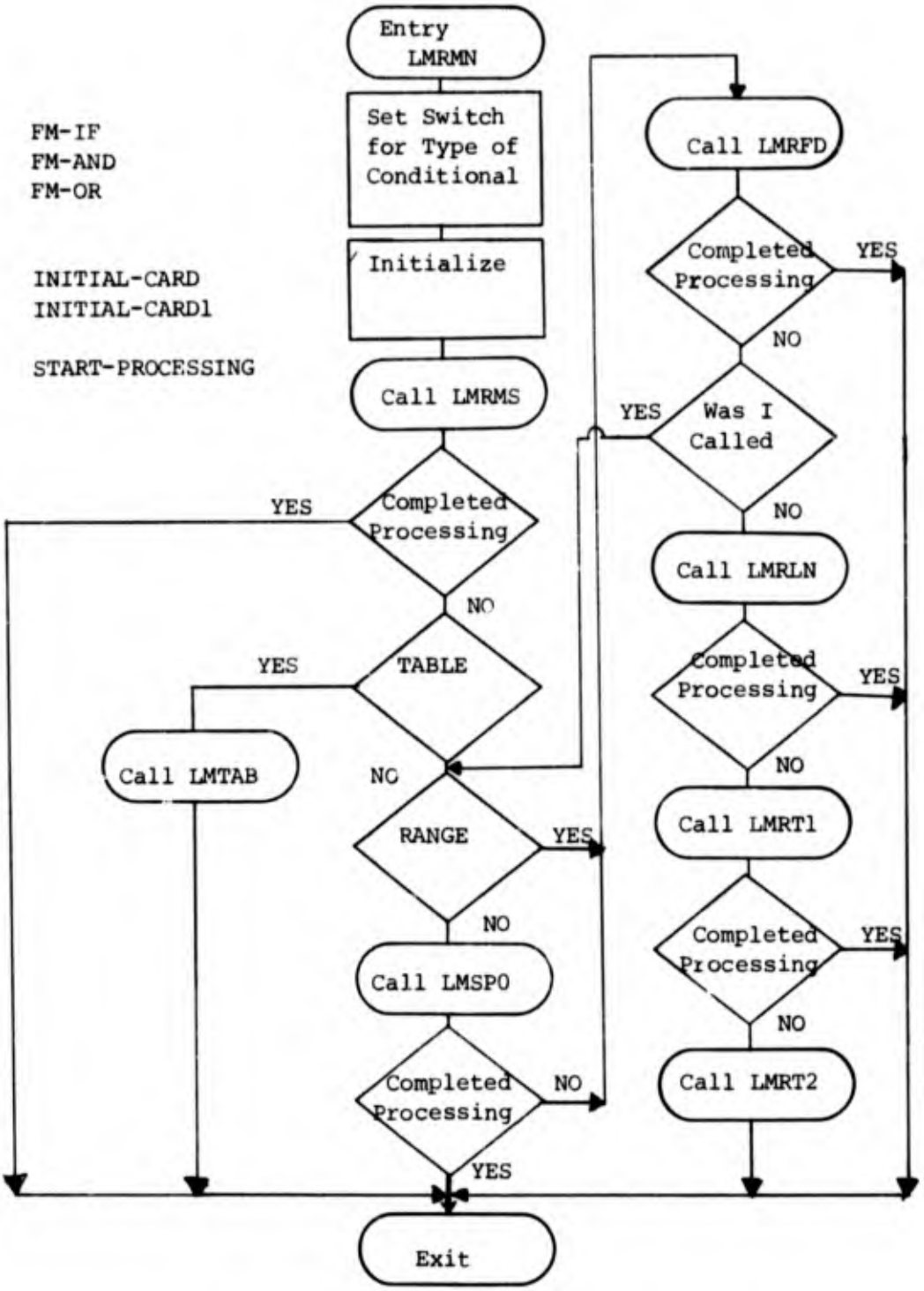
FM-RELATION-SW
=1 IS EQUAL TO
=2 IS NOT EQUAL TO
=3 IS GREATER THAN
=4 IS NOT GREATER THAN
=5 IS LESS THAN
=6 IS NOT LESS THAN

IF-NEGPOS-SW
=1 Negative nonzero number
=2 Positive nonzero number
=3 Negative zero
=4 Positive zero
=5 Zero

FIELD1-TYPE-SW and FIELD2-TYPE-SW
=1 Alphanumeric data field or alphanumeric define field
=2 Special literal
=3 Numeric define
=4 Number
=5 Numeric data field

```
EQUAL-SW
     =1    Relational word 'equal'

GREATER-SW
     =1    Relational word 'greater'

LESS-SW
     =1    Relational word 'less'

NOT-SW
     =1    Relational word 'not'

SWITCH-SW
     =1    Keyword SWITCHX found

IF-NOT-IN-RANGE-SW
     =1    Keywords 'NOT IN RANGE' found

PRINT-ERR-SW
     =1    Error was detected while processing a field in LMFLD

PROF-SW
     =1    Detected an invalid profile character

PROF-NOT-SW
     =1    Keyword 'NOT' found

IF-PARTIAL-FIELD1-SW
     =     The number of non-numeric characters in a word

IF-PARTIAL-FIELD2-SW
     =     The number of '_' in a word
```

(4) Limitations. None.


FM-IF
FM-AND
FM-OR
INITIAL-CARD
INITIAL-CARD1
START-PROCESSING
Entry LMRMN
Set Switch for Type of Conditional
Initialize
Call LMRMS
Completed Processing
YES
NO
TABLE
YES
Call LMTAB
NO
RANGE
YES
NO
Call LMSP0
Completed Processing
NO
YES
Exit
Call LMRFD
Completed Processing
YES
NO
Was I Called
YES
NO
Call LMRLN
Completed Processing
YES
NO
Call LMRT1
Completed Processing
YES
NO
Call LMRT2

z. LMLPRMS

(1) Function. This program checks for the keywords of DELETED, FIND, GENERATED, RECORD, EOF in the third position. If found, generates the appropriate COBOL code. It further checks for keywords of RANGE and TABLE.

(2) Calling Sequence.

```
ENTRY  'LMRMS'     USING FM-DD  LP-DD  LM-DD
CALL   'FMPRT'     USING PRINT-CARDS  CC
CALL   'LMLPGEN'   USING IF-CARD2
CALL   'LMLPGEN'   USING IF-CRD3
```

(3) Program Description.

GENERATE-IF-CODE.
Generates COBOL coding for the 'IF' condition.
IF-END.
Generates COBOL coding for the conclusion of the 'IF' condition.
GENERATE-OR-CODE.
Generates COBOL coding for the 'OR' condition.
OR-END.
Generates COBOL coding for the conclusion of the 'OR' condition.
GENERATE-AND-CODE.
Generates the COBOL coding for the 'AND' condition.
AND-END.
Generates COBOL coding for the conclusion of the 'AND' condition.
GENERATE-ORIF-CODE.
Generates COBOL coding for the 'OR IF' condition.
ORIF-END.
Generates COBOL coding for the conclusion of the 'OR IF' condition.
FM-FIELD-CHECK
This paragraph checks for the various keywords in the third position (DELETED, FIND, GENERATED, RECORD, EOF) and for the keywords RANGE and TABLE in their various formats.
NOT-LM-ORD.
Checks for keywords BIT, PROFILE, or RANGE as the fourth word.
LM-OR-MAINT.
Determines if LM-OM run.
FM-DELETED-PROCESS
Generates the code for DELETED RECORD.
FM-FIND-PROCESS
Generates the code for FIND RECORD.
FM-GENERATED-RECORD-PROCESS
Generates the code for GENERATED RECORD.
FM-RECORD-ID-PROCESS
Generates the code for RECORD ID.

FM-EOF-PROCESS

Generates the code for EOF MAJOR, EOF MINOR.

CHECK-EOF-CARDS

Generates the code for EOF TAPE, EOF CARD.

FM-IN-RANGE-PROCESS

Takes the Range portion of the RANGE statement in the form NNNN-MMMM and divides it into the two separate ranges.

MOVE-M1

If the word (NNNN-MMMM) is greater than twenty characters the remainder of the word must be retrieved from the overflow area and concatenated with the characters in the word.

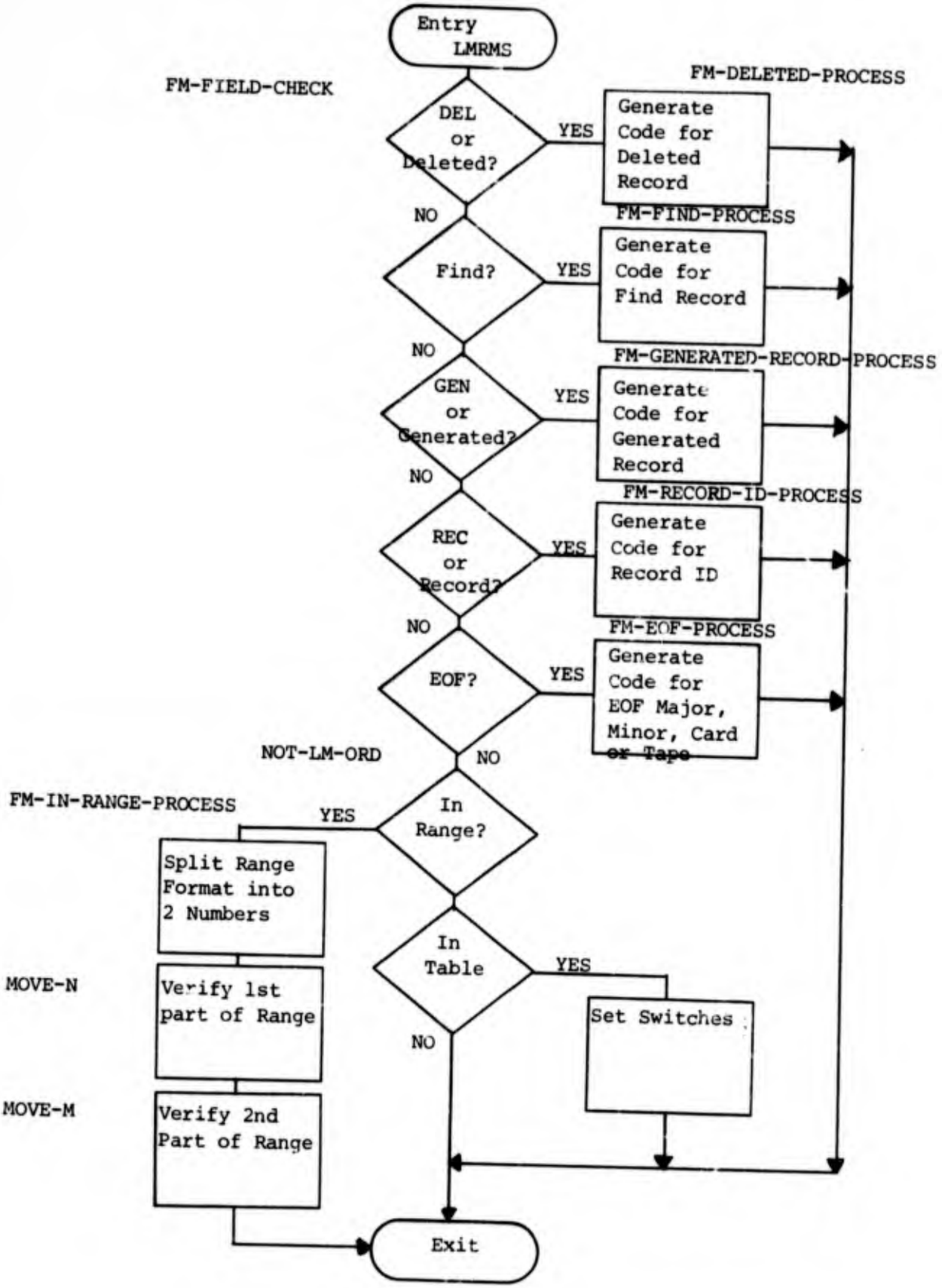
MOVE-N

Validates the format of the first part of the range.

MOVE-M

Validates the format of the second part of the range.

(4) Limitations. None.


Entry
LMRMS
FM-FIELD-CHECK
FM-DELETED-PROCESS
DEL
or
Deleted?
YES
Generate
Code for
Deleted
Record
NO
FM-FIND-PROCESS
Find?
YES
Generate
Code for
Find Record
NO
FM-GENERATED-RECORD-PROCESS
GEN
or
Generated?
YES
Generate
Code for
Generated
Record
NO
FM-RECORD-ID-PROCESS
REC
or
Record?
YES
Generate
Code for
Record ID
NO
FM-EOF-PROCESS
EOF?
YES
Generate
Code for
EOF Major,
Minor, Card
or Tape
NOT-LM-ORD
NO
FM-IN-RANGE-PROCESS
In
Range?
YES
Split Range
Format into
2 Numbers
In
Table
YES
MOVE-N
Verify 1st
part of Range
Set Switches
NO
MOVE-M
Verify 2nd
Part of Range
Exit

aa. LMLPRST

(1) Function. Handles the following

READ (CARD,TAPE,MAJOR,MINOR)
WRITE
SAVE
SCRAP
STOP
RELOAD MAJOR
STOP SFL

(2) Calling Sequence.

```
ENTRY 'LMRDR'  USING FM-DD  LP-DD  LM-DD
CALL  'LMWRT'  USING FM-DD  LP-DD  LM-DD
CALL  'LMSAV'  USING FM-DD  LP-DD  LM-DD
CALL  'LMSCP'  USING FM-DD  LP-DD  LM-DD
CALL  'LMSTO'  USING FM-DD  LP-DD  LM-DD
CALL  'LMRLD'  USING FM-DD  LP-DD  LM-DD
CALL  'LMPCH'  USING FM-DD  LP-DD  LM-DD
CALL  'LMCVT'  USING FM-DD  LP-DD  LM-DD
CALL  'LMFLD'  USING FM-DD  LP-DD  LM-DD
CALL  'LMFLD'  USING FM-DD  LP-DD  LM-DD
CALL  'FMPRT'  USING PRNT-CARDS  CC
CALL  'LMDAT'  USING FM-CARDS
CALL  'LMPRO'  USING PROCEDURE-RECORD
```

(3) Program Description.

GENERATE-READ.
This paragraph does a check for the type of 'READ' being processed (i.e., READ CARD), and for errors of format or spelling. If a READ MAJOR or READ MINOR is detected further tests are made in paragraph GEN-READ-MAJOR-MINOR for type run and the appropriate Procedure Divison COBOL code for a READ MAJOR or MINOR statement is generated by 'performing' paragraph BUILD-RETURN-CODE-SENTENCE.

GENERATE-READ thru GEN-READ-CARD.
If it is determined that the READ is for tape or cards, logic flow branches from paragraph GENERATE-READ to GEN-READ-CARD. Here the receiving field is tested to be a defined area of proper type and format. The appropriate COBOL Procedure Division coding is then generated.

GENERATE-WRITE.
This paragraph performs error checking for a file-to-file Logic Package 'WRITE' statement. The generated COBOL coding is produced by performing paragraph BUILD-RETURN-CODE-SENTENCE.

GENERATE-SAVE.

This paragraph handles the 'SAVE' instruction. Error checking is performed and the appropriate COBOL code generated.

GENERATE-SCRAPE.

Run type is determined in this paragraph and appropriate COBOL Procedure Division coding for the SCRAP statement is generated when the run type is single file. Logic then branches to paragraph GENERATE-STOP where coding for the 'STOP' instruction is also generated. For file-to-file runs only the 'STOP' instruction coding is generated. No error checking is necessary.

GENERATE-STOP.

Generates COBOL Procedure Division coding for the 'STOP' or 'STOP SFL' statements. No error checking is necessary. If the STOP SFL instruction is used, a switch value is generated which will indicate to FM not to recall the single-file LM package. For file-to-file runs the STOP SFL instruction acts exactly like the STOP instruction.

GENERATE-RELOAD.

This paragraph handles the 'RELOAD MAJOR' statement. Error checking to determine RUN-TYPE is made. If no error is detected COBOL Procedure Division Code is generated.

GENERATE-PUNCH.

This paragraph validates for errors and generates COBOL Procedure Division coding for the PUNCH instruction along with paragraphs GP-FULL-FIELD and GP-PROPER. These paragraphs allow for the punching directly from field, partial field or group areas.

GP-FULL-FIELD.

This paragraph is used in conjunction with GENERATE-PUNCH and is for punching a full field.

GP-PROPER.

This paragraph is used in conjunction with GENERATE-PUNCH and is for generating COBOL code to punch a card from an output area.

CHECK-SPECIAL-FIELD.

This paragraph is used in conjunction with GENERATE-CONVERT to determine the field type of the field to be converted.

BUILD-NUM-LIT.

This paragraph builds a numeric literal value for a CONVERT operation.

GENERATE-CONVERT.

Determines the type of field which is to be converted. It 'performs' paragraph CHECK-SPECIAL-FIELD to determine if the field is a special literal or system reference. Program LMLPFLD is called to determine major or minor field type. If a partial field is indicated, logic branches to paragraph GC-MOVE-PART-FLD1. If a system reference ($$ORC) or literal is detected, logic flow branches to paragraph GC-MOVE-SPEC-FLD1. See below. For any nonpartial field or defined literal, COBOL coding needed initially for the convert statement is generated. Logic Flow continues in paragraph GC-MOVE-FILE-NAME and then paragraph GC-FLD2 to generate additional Procedure Division coding.

If the output area for CONVERT data is numeric, logic flow branches to paragraph GC-FLD2-NUMERIC where additional Data Division coding is generated, otherwise processing is completed at paragraph GC-EXIT.

GEN-NAME-LINE.

This paragraph works in conjunction with GENERATE-CONVERT to generate COBOL coding for a field name for any nonpartial field or defined literal.

GC-MOVE-PART-FLD1.

Generates COBOL Procedure Division coding for a partial field. From here logic branches to paragraph GC-MOVE-FILE-NAME for generation of additional COBOL coding. Logic continues into paragraph GC-FLD2-ALPH to handle Procedure Division coding for partial fields. From here logic flow continues to paragraph GC-EXIT and processing is completed.

GC-MOVE-SPEC-FLD1.

Generates COBOL Procedure Division coding for a CONVERT of a system reference, special literal or literal. Logic branches to paragraph GC-MOVE-FILE-NAME and processing continues as in a CONVERT FIELD statement shown above.

GC-MOVE-FILE-NAME.

This paragraph generates COBOL Procedure Division coding to initiate the CONVERT activity in the generated COBOL program.

GC-FLD2.

This paragraph does error checking of the result field of the CONVERT statement.

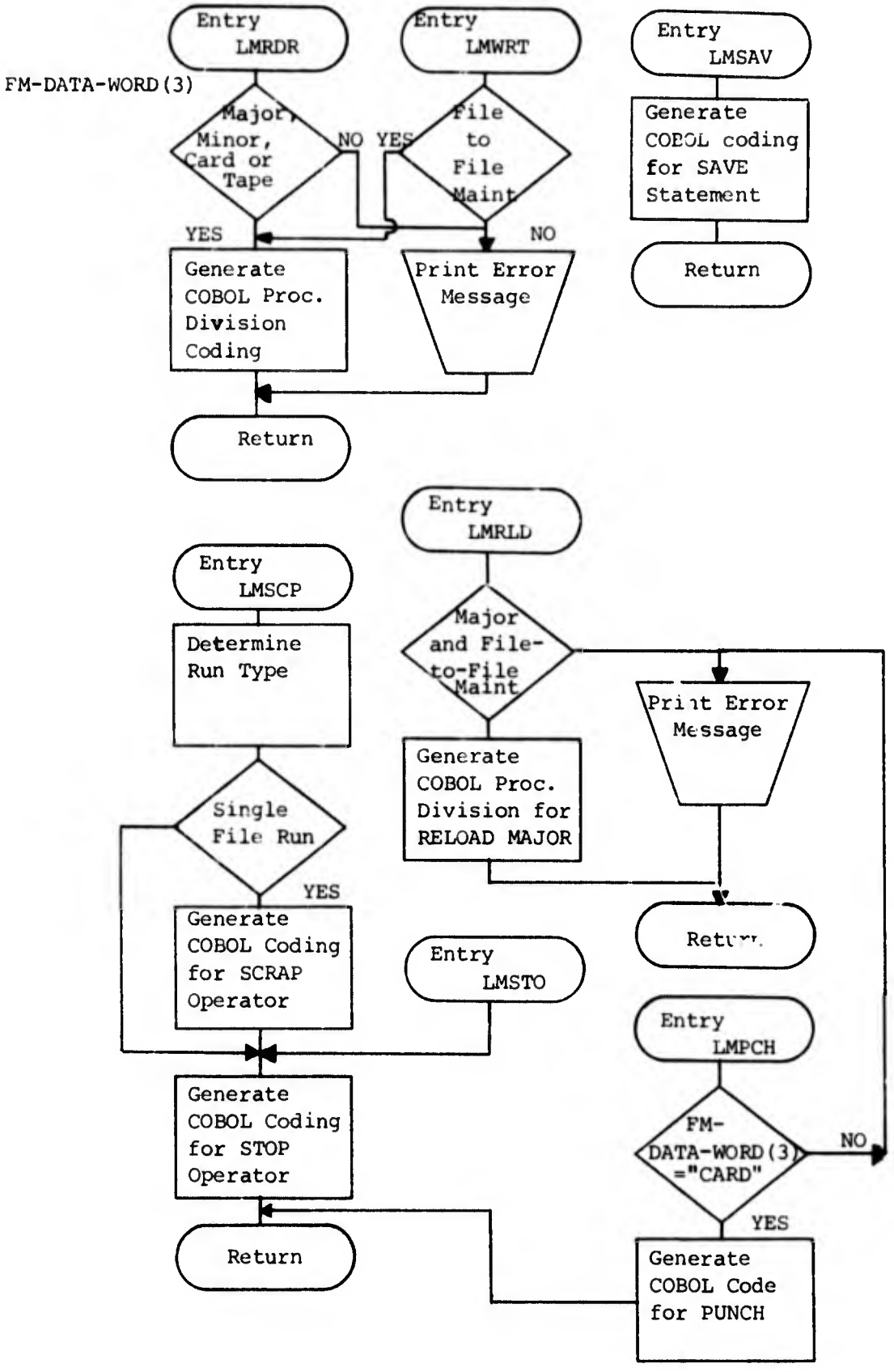
GC-FLD2-ALPH.

This paragraph generates COBOL coding for the CONVERT instruction to move the subroutine output into the result field when the result field is alphanumeric.

GC-FLD2-NUMERIC.

This paragraph generates COBOL coding for the CONVERT instruction to move the subroutine output into a numeric result field.

(4) Limitations. None.


Entry LMRDR
Entry LMWRT
Entry LMSAV
FM-DATA-WORD(3)
Major, Minor, Card or Tape
NO
YES
File to File Maint
YES
NO
Generate COBOL coding for SAVE Statement
Generate COBOL Proc. Division Coding
Print Error Message
Return
Return
Entry LMRLD
Entry LMSCP
Major and File-to-File Maint
Determine Run Type
Print Error Message
Generate COBOL Proc. Division for RELOAD MAJOR
Single File Run
YES
Generate COBOL Coding for SCRAP Operator
Entry LMSTO
Return
Entry LMPCH
Generate COBOL Coding for STOP Operator
FM-DATA-WORD(3) ="CARD"
NO
YES
Return
Generate COBOL Code for PUNCH

ab. LMLPRT1

(1) Function. This program generates the appropriate code when the statement is in the format

conditional FLDA relational FLDB .FAILURE

The only case it does not handle is when FLDA is a numeric data field. This is handled in LMLPRT2.

(2) Calling Sequence.

```
ENTRY 'LMPRT1'       USING FM-DD  LP-DD  LM-DD
CALL  'FMPRT'        USING PRNT-CARDS  CC
CALL  'LMLPGEN'      USING IF-CARD1
CALL  'LMLPGEN'      USING IF-CARD2
CALL  'LMLPGEN'      USING IF-CRD3
```

(3) Program Descriptions.

GENERATE-IF-CODE.
Generates COBOL code before the 'IF' conditional.
IF-END.
Generates COBOL code after the 'IF' conditional.
GENERATE-OR-CODE.
Generates COBOL code before the 'OR' conditional.
OR-END.
Generates COBOL code after the 'OR' conditional.
GENERATE-AND-CODE.
Generates COBOL code before the 'AND' conditional.
AND-END.
Generates COBOL code after the 'AND' conditional.
GENERATE-ORIF-CODE.
Generates COBOL code before the 'OR IF' conditional.
ORIF-END.
Generates COBOL code after the 'OR IF' conditional.
FIELD-DECISION.
Controls the spliting for FIELD1 types. Branches to the designated FIELD-DECISION subparagraphs depending on the type of the first field.
FIELD-DECISION1.
Comes here if the first field is alphanumeric and then proceeds to the appropriate GENERATE code paragraph depending on FIELD2.
FIELD-DECISION3.
Comes here if the first field is numeric define and then proceeds to the appropriate GENERATE-CODE paragraph depending on FIELD2.

FIELD-DECISION5.

Comes here if the first field is number (numeric literal) and then proceeds to the appropriate GENERATE-CODE paragraph depending on FIELD2.

FIELD-DECISION6.

Comes here if the first field is a special literal (enclosed in quotes) and then proceeds to the appropriate GENERATE-CODE paragraph depending on FIELD2.

GENERATE-CODE1.

Generates appropriate code when FIELD1 is an alphanumeric data field, a special literal, or alphanumeric defined literal, and FIELD2 is an alphanumeric data field, a special literal, or alphanumeric defined literal. (FIELD1 and FIELD2 cannot both be special literals.) When FIELD1 is a numeric defined literal and FIELD2 is a numeric defined literal or a numeric literal. When FIELD1 is a numeric literal and FIELD2 is a numeric defined literal.

GENERATE-CODE4.

Generates appropriate code when FIELD1 is a numeric defined literal and FIELD2 is a numeric data field.

GENERATE-CODE11.

Generates initial code when field1 is a numeric literal and FIELD2 is a numeric data field. The rest of the code is generated in GENERATE-CODE8.

GENERATE-CODE8.

Generates the appropriate code when FIELD1 is an alphanumeric data field or defined literal and FIELD2 is a numeric data field. Additional code is generated when the relation is equal, not less, not greater and other code generated when the relational is not equal, less, or greater.

GENERATE-CODE10.

Generates appropriate code when FIELD2 is a numeric defined literal and FIELD1 is an alphanumeric data field or defined literal.

GENERATE-CODE9.

Generates the appropriate code when FIELD1 is a numeric defined literal and FIELD2 is an alphanumeric defined literal or data field.

REDEFINES-GENER.

Generates an 01 literal for the alphanumeric field and redefines 01 literal as numeric.

IF-SEQ1-GENER.

Outputs COBOL coding for field1.

IF-SEQ2-GENER.

Outputs the COBOL coding for the relation and field2.

IF-SEQ3-GENER.

Outputs the COBOL coding for both field1 and field2 when their format does not fit in the previous two output areas.

GENERATE-PARTIAL-MOVE.

Generates the moves when a partial field is specified so the literals can be used in place of the fields.

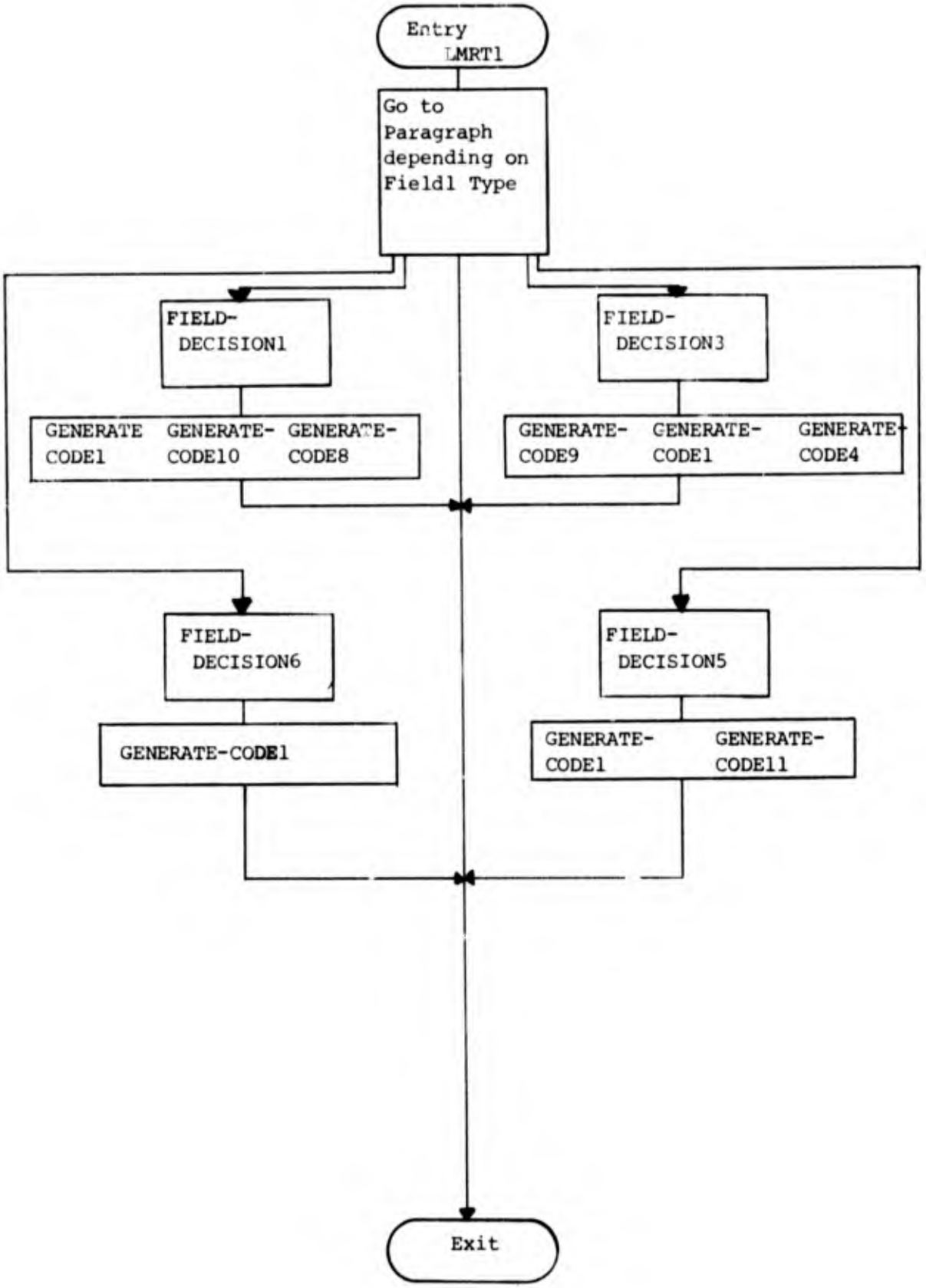
GENER-FIRST-PART.

Does a Computed Go To depending on the conditional and generates the appropriate code previous to the relation testing.

CHECK-FIRST-PARTIAL.

This paragraph is used when the first field is an alphanumeric partial and the second is numeric.

(4) Limitations. None.


Entry
LMRT1
Go to
Paragraph
depending on
Field1 Type
FIELD-
DECISION1
FIELD-
DECISION3
GENERATE
CODE1
GENERATE-
CODE10
GENERATE-
CODE8
GENERATE-
CODE9
GENERATE-
CODE1
GENERATE-
CODE4
FIELD-
DECISION6
FIELD-
DECISION5
GENERATE-CODE1
GENERATE-
CODE1
GENERATE-
CODE11
Exit

ac. LMLPRT2

(1) Function. This program generates the appropriate code when the statement is in the form:

conditional FLDA relational FLDB .FAILURE

and FLDA is a numeric data field.

(2) Calling Sequence.

```
ENTRY  'LMRT2'    USING FM-DD  LP-DD  LM-DD
CALL   'FMPRT'    USING PRNT-CARDS  CC
CALL   'LMLPGEN'  USING IF-CARD1
CALL   'LMLPGEN'  USING IF-CARD2
CALL   'LMLPGEN'  USING IF-CARD3
```

(3) Program Description. Control only comes here if the first field is a numeric data field. In any other case the appropriate code would be generated in LMRT1.

```
GENERATE-IF-CODE.
IF-END.
GENERATE-OR-CODE.
OR-END.
GENERATE-AND-CODE.
AND-END.
GENERATE-ORIF-CODE.
ORIF-END.
```

For descriptions of the above paragraph headings, see LMLPRT1 program description.

FIELD-DECISION4.

Comes here when the first field is numeric data field and then proceeds to the appropriate GENERATE-CODE paragraph depending on FIELD2.

GENERATE-CODE3.

Generate appropriate code when FIELD1 is a numeric data field and FIELD2 is a numeric defined literal.

GENERATE-CODE5.

Generates appropriate code when FIELD1 is a numeric data field and FIELD2 is a numeric literal and when the combination of the relation and FIELD2 allows FIELD1 to have a value of zero. The second part of this paragraph generates the appropriate code when FIELD1 and FIELD2 are the same as in previous note but the combination of the relation and FIELD2 does not allow FIELD1 to have a value of zero.

GENERATE-CODE6.

Generates the appropriate code when FIELD1 and FIELD2 are numeric data fields and the relation is equal, not less or not greater. The second part of this section generates the code for the above fields when the relation is not equal, less or greater.

GENERATE-CODE7.

Generates appropriate code when FIELD1 is a numeric data field and FIELD2 is an alphanumeric data field or defined literal. Additional code is generated when the relation is equal, not less, or not greater and other code generated when the relation is not equal, less or greater.

REDEFINES-GENER.

Generates an 01 literal for the alphanumeric field and a redefines 01 literal as numeric.

IF-SEQ1-GENER.

IF-SEQ2-GENER.

IF-SEQ3-GENER.

Outputs the COBOL coding for field1, field2, and the relational.

GENERATE-PARTIAL-MOVE.

Generates the moves when a partial field is specified so the literals can be used in place of the fields.

GENER-FIRST-PART.

Does a Computed Go To depending on the conditional and generates the appropriate code previous to the relation testing.

CHECK-FIRST-PARTIAL.

This paragraph is used when the first field is an alphanumeric partial and the second is numeric.

TABLE 1

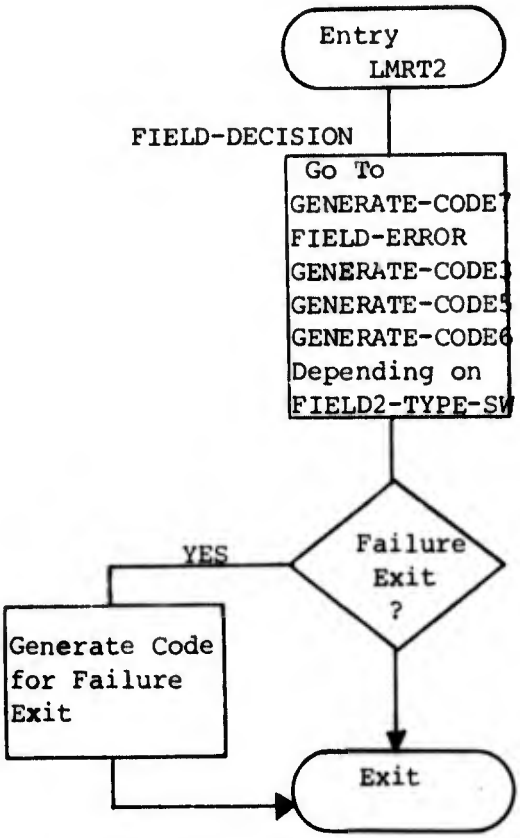
Is a conditional output table and shows where control branches to depending on field1 and field2.

(4) Limitations. None.

CONDITIONAL OUTPUT TABLE

FIELD1	FIELD2: ALPHA-NUMERIC FIELD OR DEFINE	FIELD2: SPECIAL LITERAL	FIELD2: NUMERIC DEFINE	FIELD2: NUMBER	FIELD2: NUMERIC FIELD
ALPHA-NUMERIC FIELD OR DEFINE	GENERATE-CODE1	GENERATE-CODE1	GENERATE-CODE10	ERROR	GENERATE-CODE8
SPECIAL LITERAL	GENERATE-CODE1	ERROR	ERROR	ERROR	ERROR
NUMERIC DEFINE	GENERATE-CODE9	ERROR	GENERATE-CODE1	GENERATE-CODE1	GENERATE-CODE4
NUMBER	ERROR	ERROR	GENERATE-CODE1	ERROR	GENERATE-CODE11 GENERATE-CODE8
NUMERIC FIELD	GENERATE-CODE7	ERROR	GENERATE-CODE3	GENERATE-CODE5	GENERATE-CODE6

Table 1


Entry
LMRT2
FIELD-DECISION
Go To
GENERATE-CODE7
FIELD-ERROR
GENERATE-CODE1
GENERATE-CODE5
GENERATE-CODE6
Depending on
FIELD2-TYPE-SW
Failure
Exit
?
YES
Generate Code
for Failure
Exit
Exit

ad. LMLPSCN

(1) Function. This program scans an input card to determine the type of instruction involved. Appropriate switches are set to indicate the type instruction to program LMLP.

(2) Calling Sequence.

```
ENTRY  'LMSCN'  USING FM-DD  LP-DD  LM-DD
CALL   'LMCRD'  USING FM-INPUT  END-SW
CALL   'FMPRT'  USING PRNT-CARD  CC
CALL   'LMPRO'  USING IF-CRD3
CALL   'LMDAT'  USING FM-CARD
CALL   'LMLINK' USING FM-CARD
```

(3) Program Description.

FM-LOG-MAIN.
Controls reentry from LMLP. The switch SEQ-OK is set according to the type card being processed (i.e., PRINT, DELETE, etc.).

FM-READ.
Calls FMCRD which reads and prints each logic package card. If there is an asterisk in column 1, it is treated as a comment and the next card is read.

FM-READ1.
Performs the initialization. If column is not equal to a space then it is assumed that this is the beginning of a LABEL.

LM-LABEL.
LM-LABEL1.
CHECK-7.
LU-LAB-TAB.
LAB-TAB-LU.
These paragraphs collectively validate a label entry for proper format and syntax. A table of labels is checked to insure a label has not been previously used.

LABEL-FAILURE-LU.
LABEL-FAILURE-LU1.
Determine if there has been a reference to this label and if so delete it from the failure-table.

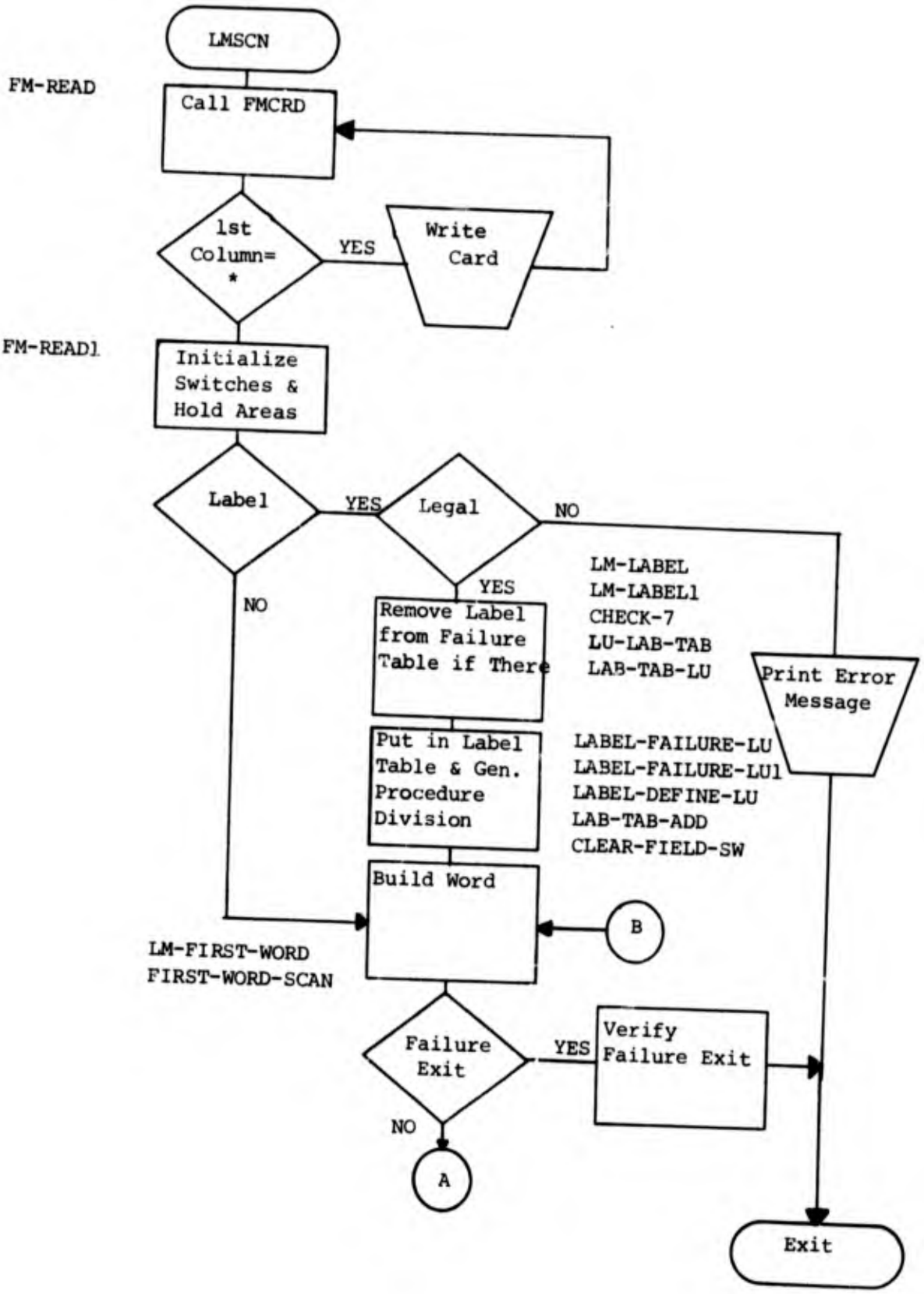
LAB-TAB-ADD.
Enters the valid label in the Label Table.

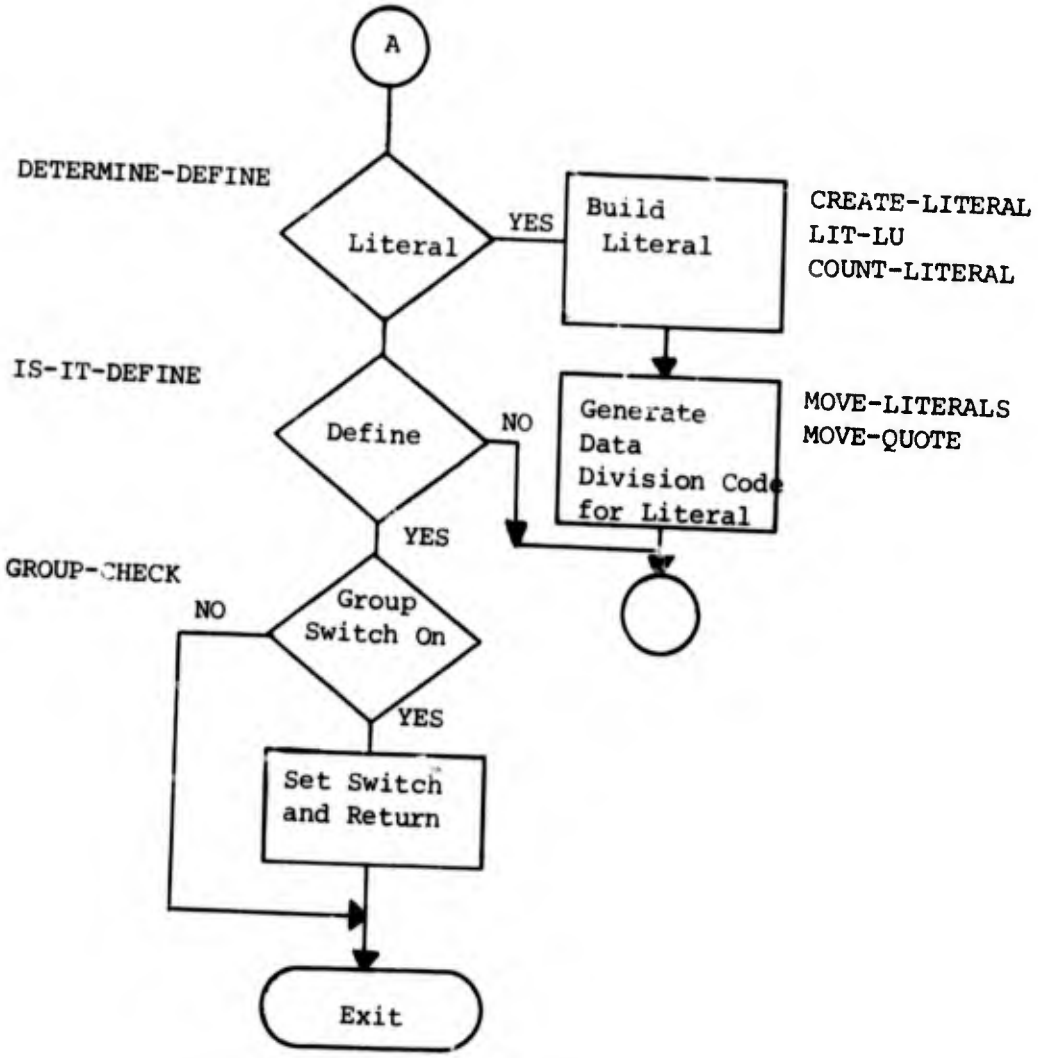
MOVE-DASH-F-C.
Qualifies the label by adding a '-L'.

CLEAR-FIELD-SW.
Produces the PROCEDURE DIVISION entry.

LM-FIRST-WORD.
FIRST-WORD-SCAN.
Scan the card until spaces are found terminating a word, puts these words in the word-table. A period is considered as a word and signifies the presence of a failure exit.
SKIP-PARA1.
CARD-GO-TO.
GO-TO-WHERE.
LABEL-TABLE-LOOKUP.
FAIL-TABLE-LOOKUP.
FAIL-TABLE-LU.
Each Failure Exit is checked against the Label Table to determine if there is a corresponding label. If a corresponding label is found, no other action is necessary. If no corresponding label is found, the Failure Exit is entered into the FAIL TABLE.
DETERMINE-DEFINE.
Determines if a card entry contains a literal value and validates a literal for proper quote or At-sign.
CREAT-LITERAL.
LIT-LU.
COUNT-LITERAL.
These paragraphs scan the literal value until a closing quote or At-sign is found. The literal is then built for use in the generated COBOL PROCEDURE DIVISION entries.
MOVE-LITERAL3.
MOVE-QUOTE.
Generate the COBOL Data Division code for the literal.
IS-IT-DEFINE.
Checks to see if the word is DEFINE.
GROUP-CHECK.
If DEFINE was found, the next word is checked for GROUP. Switches are set to tell LMLP what conditions was found.

(4) Limitations. None.


LMSCN
FM-READ
Call FMCRD
1st Column= *
YES
Write Card
FM-READ1
Initialize Switches & Hold Areas
Label
YES
Legal
NO
NO
YES
LM-LABEL
LM-LABEL1
CHECK-7
LU-LAB-TAB
LAB-TAB-LU
Remove Label from Failure Table if There
Print Error Message
Put in Label Table & Gen. Procedure Division
LABEL-FAILURE-LU
LABEL-FAILURE-LU1
LABEL-DEFINE-LU
LAB-TAB-ADD
CLEAR-FIELD-SW
Build Word
B
LM-FIRST-WORD
FIRST-WORD-SCAN
Failure Exit
YES
Verify Failure Exit
NO
A
Exit


A
DETERMINE-DEFINE
Literal
YES
Build Literal
CREATE-LITERAL
LIT-LU
COUNT-LITERAL
IS-IT-DEFINE
Define
NO
Generate Data Division Code for Literal
MOVE-LITERALS
MOVE-QUOTE
YES
GROUP-CHECK
NO
Group Switch On
YES
Set Switch and Return
Exit

ae. LMLPSPO

(1) Function. This program handles the processing of BIT, SWITCH, and PROFILE operations.

(2) Calling Sequence.

```
ENTRY 'LMSPO'      USING FM-DD  LP-DD  LM-DD
CALL  'FMPRT'      USING PRNT-CARDS  CC
CALL  'LMLPGEN'    USING IF-CARD1
CALL  'LMLPGEN'    USING IF-CRD3
CALL  'LMLPGEN'    USING FM-CARD
CALL  'LMLPSP1'    USING FM-DD  LP-DD  LM-DD
```

(3) Program Description.

START-LMSPO.
First determines if LMRFD was called to validate the field names for BIT or PROFILE operations, if so return to the point where they left and continue processing those statements.

FM-FIELD-CHECK.
Checks for the SWITCH operation.

FM-CHECK-WORD4
Checks for BIT operation or PROFILE operation and if found goes to the appropriate processing paragraph for those statements.

FM-SWITCH-PROCESS
Controls the processing of the SWITCH operation. It utilizes much of the BIT processing paragraphs to validate the words after SWITCH. A unique total is returned and the appropriate paragraph is branched to generate the desired code.

ON4
Generates the code for SWITCH ON or NOT OFF.

OFF4
Generates the code for SWITCH OFF or NOT ON.

FM-BIT-PROCESS
Returns to LMRMN which calls LMRFD for validation of the field name on the BIT operation.

RETURN1
Validates that the field is a legal type.

SWITCH-TYPE
Determines the number of words after BIT or SWITCH. Controls the table lookup to validate those words.

DETERMINE-SWITCH-TYPE
Does a table lookup for each of the words after SWITCH or BIT. They must be +, NOT, ON, OFF, -.

SWITCH-SET

Goes to the appropriate paragraph which adds a number to the total.

SWPLUS

Adds 1 to the total count.

SWNOT

Adds 3 to the total count.

SWON

Adds 4 to the total count.

SWOFF

Adds 5 to the total count.

SWMINUS

Adds 6 to the total count.

SWITCH-TYPE-SET

Adjusts the total count to create a unique number which reflects the combination of words and field type for the BIT statement. Calls LMSP1 which generates the code for BIT depending on the value of the total accumulated.

GOTO-RETURN2.

Generates first part of COBOL Data Division depending on switch or bit settings.

REDEFINES-GENER

Generates the numeric redefine statement for the Data Division that is needed when the field type on the BIT operation is alphanumeric.

FM-PROFILE-PROCESS

Calls LMRFD to validate the field name for the PROFILE OPERATION.

RETURN2

Validates the field types and the format of the rest of the PROFILE statement. It also generates the Data Division entry necessary calls LMSP1 for generation of appropriate code.

GOTO-RETURN3.

This paragraph generates the 01 Data Division for the profile card.

CHECK-CARD-TYPE-SWITCHES.

Does a Computed Go To depending on the conditional word and generates the appropriate code after the relational testing.

IF-SEQ1-GENER.

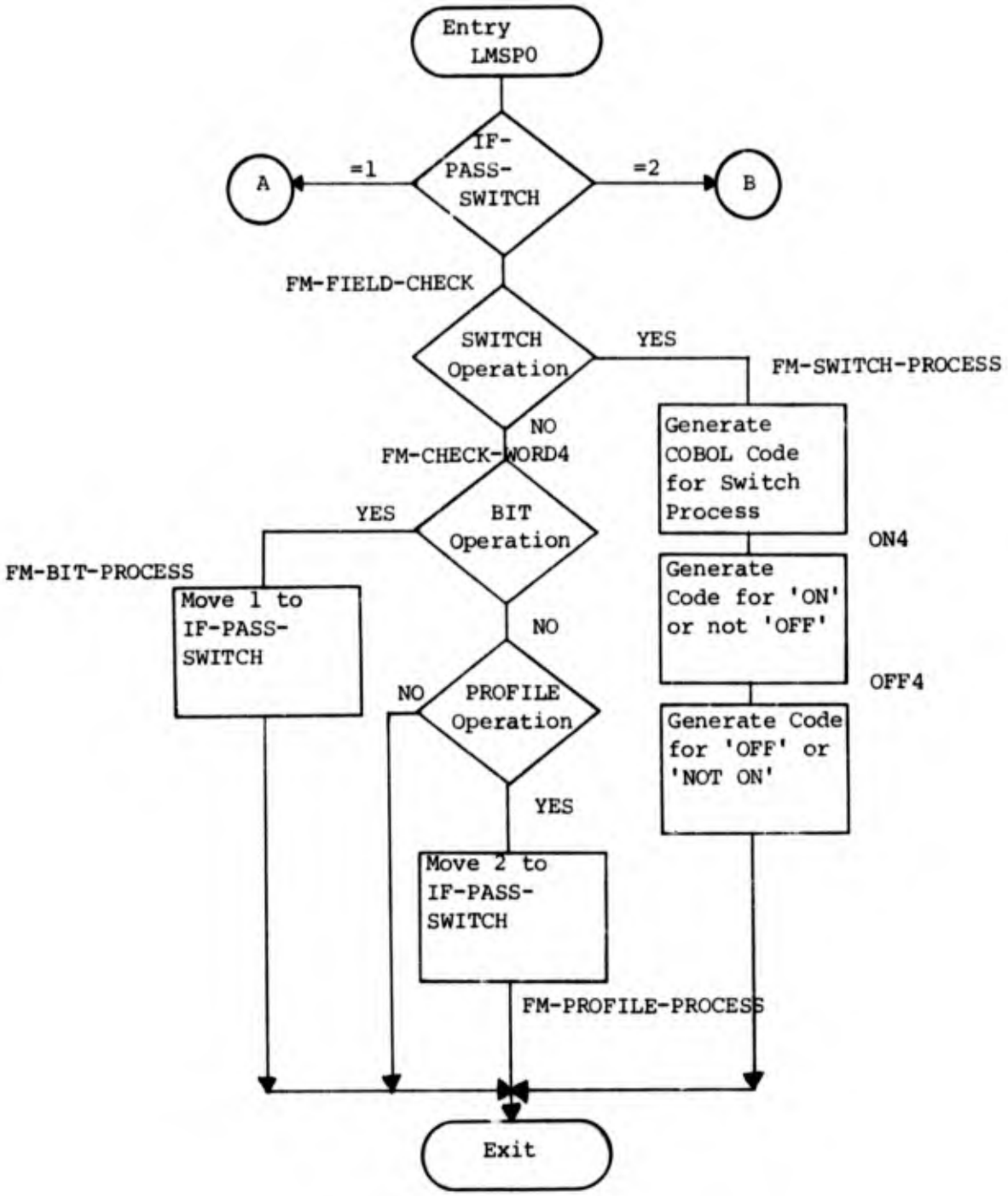
Outputs the coding for field1.

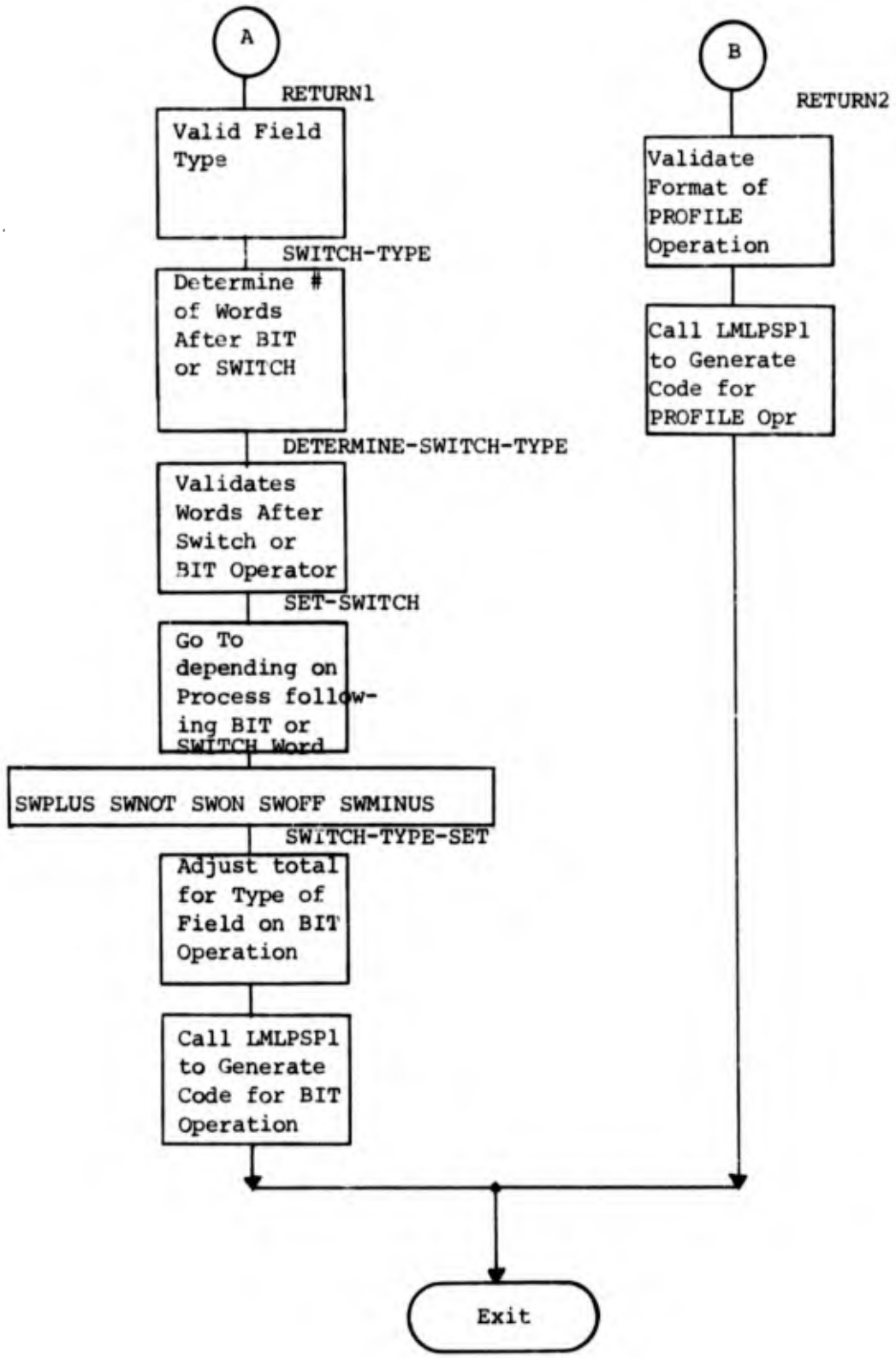
IF-SEQ3-GENER.

Outputs the coding for both field1 and field2 when their format does not fit in the previous two output areas.

IF-CLOSE-UP.
ICU1-PFRM.
IF-CLOSE-UP1.
IF-CLOSE-UP-EXIT.
IF-ERROR.
GENERATE-PARTIAL-MOVE.
Generates the moves when a partial field is specified so the literals can be used in place of the fields.

(4) Limitations. None.


Entry
LMSP0
IF-
PASS-
SWITCH
=1
=2
A
B
FM-FIELD-CHECK
SWITCH
Operation
YES
FM-SWITCH-PROCESS
NO
FM-CHECK-WORD4
Generate
COBOL Code
for Switch
Process
YES
BIT
Operation
ON4
FM-BIT-PROCESS
Generate
Code for 'ON'
or not 'OFF'
Move 1 to
IF-PASS-
SWITCH
NO
OFF4
NO
PROFILE
Operation
Generate Code
for 'OFF' or
'NOT ON'
YES
Move 2 to
IF-PASS-
SWITCH
FM-PROFILE-PROCESS
Exit


A
RETURN1
Valid Field Type
SWITCH-TYPE
Determine # of Words After BIT or SWITCH
DETERMINE-SWITCH-TYPE
Validates Words After Switch or BIT Operator
SET-SWITCH
Go To depending on Process follow-ing BIT or SWITCH Word
SWPLUS SWNOT SWON SWOFF SWMINUS
SWITCH-TYPE-SET
Adjust total for Type of Field on BIT Operation
Call LMLPSP1 to Generate Code for BIT Operation
B
RETURN2
Validate Format of PROFILE Operation
Call LMLPSP1 to Generate Code for PROFILE Opr
Exit

af. LMLPSP1

(1) Function. This program generates the appropriate COBOL coding for BIT and PROFILE operations.

(2) Calling Sequence.

```
ENTRY 'LMSP1'    USING FM-DD  LP-DD  LM-DD
CALL  'FMPRT'    USING PRNT-CARDS  CC
CALL  'LMLPGEN'  USING IF-CARD1
CALL  'LMLPGEN'  USING IF-CRD3
CALL  'LMLPGEN'  USING FM-CARD
```

(3) Program Description.

LMLPSP1

has two functions: (a) to generate the code BIT statements and (b) to generate the code for PROFILE statements. Upon entry, a determination is made as to which code generation is desired. The generation of BIT code proceeds through the following path.

ENTER1

This paragraph goes to the desired paragraph which will generate the appropriate code for the BIT statement depending on the words following BIT and the type of field. The paragraphs are in the form

```
PLUS     ON     1
MINUS    OFF    2
                3
```

The first bracket is determined by the key symbols + or -. The second bracket is determined by the keyword or words ON or OFF with NOT OFF equivalent to ON and NOT ON equivalent to OFF. The third bracket is determined by the type of field present; 1 indicates an alphanumeric field, 2 indicates a numeric data field, and 3 indicates a numeric define.

LIT-Ø2-PIC
LIT-FILLER-ST
LIT-FILLER

If PROFILE was indicated, generate the Data Division code required for PROFILE.

ADD-1-HM

Verifies each PROFILE character and replaces the alphabetic PROFILE character by a two digit numeric indicator which will be used during execution to verify the input data.

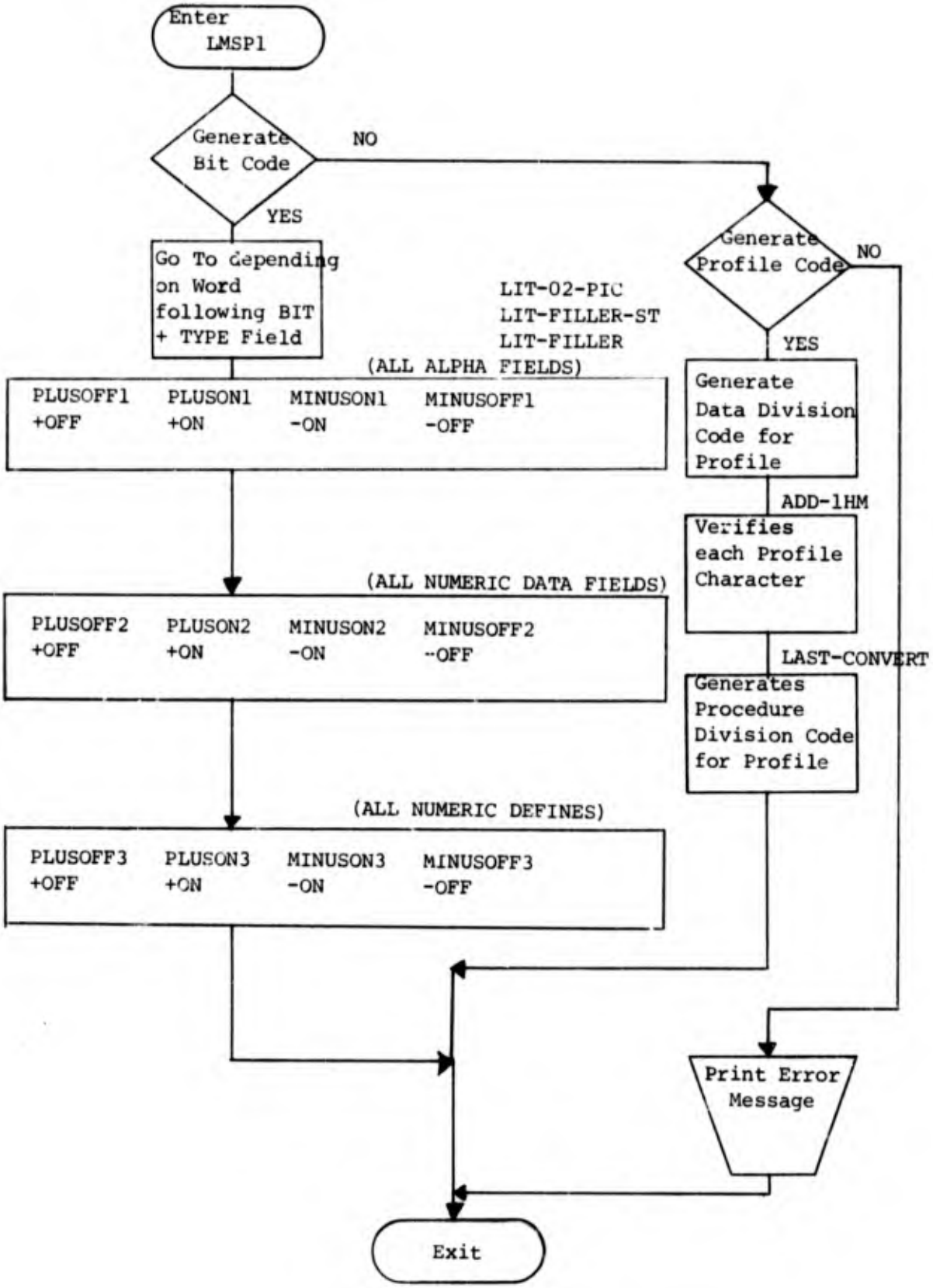
LAST-CONVERT
Generates the Procedure Division code for the PROFILE operation.

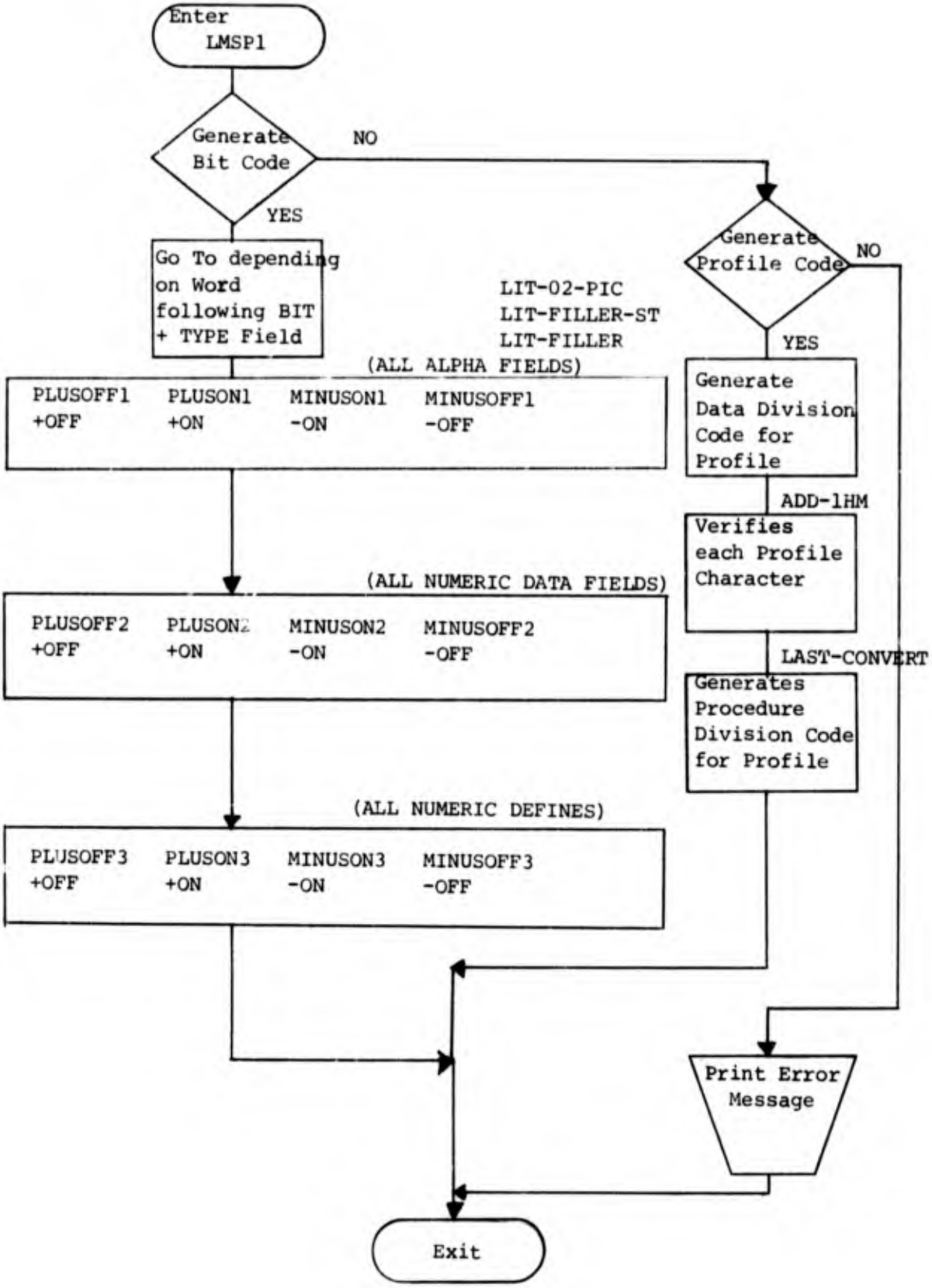
FINISH-DD
Generates a COBOL 02 Data Division entry for each of the PROFILE characters specified. The decoded alpahabetic PROFILE character is used in the VALUE clause.

IF-SEQ1-GENER.
This paragraph outputs the coding for FIELD1.

IF-SEQ3-GENER.
This paragraph outputs the coding for both FIELD1 and FIELD2 when their format does not fit in the previous two output areas.

(4) Limitations. None.


Enter
LMSP1
Generate
Bit Code
NO
YES
Go To depending
on Word
following BIT
+ TYPE Field
Generate
Profile Code
NO
YES
LIT-02-PIC
LIT-FILLER-ST
LIT-FILLER
(ALL ALPHA FIELDS)
PLUSOFF1
+OFF
PLUSON1
+ON
MINUSON1
-ON
MINUSOFF1
-OFF
Generate
Data Division
Code for
Profile
ADD-1HM
Verifies
each Profile
Character
(ALL NUMERIC DATA FIELDS)
PLUSOFF2
+OFF
PLUSON2
+ON
MINUSON2
-ON
MINUSOFF2
-OFF
LAST-CONVERT
Generates
Procedure
Division Code
for Profile
(ALL NUMERIC DEFINES)
PLUSOFF3
+OFF
PLUSON3
+ON
MINUSON3
-ON
MINUSOFF3
-OFF
Print Error
Message
Exit


Enter
LMSP1
Generate
Bit Code
NO
YES
Go To depending
on Word
following BIT
+ TYPE Field
LIT-02-PIC
LIT-FILLER-ST
LIT-FILLER
(ALL ALPHA FIELDS)
PLUSOFF1
+OFF
PLUSON1
+ON
MINUSON1
-ON
MINUSOFF1
-OFF
(ALL NUMERIC DATA FIELDS)
PLUSOFF2
+OFF
PLUSON2
+ON
MINUSON2
-ON
MINUSOFF2
-OFF
(ALL NUMERIC DEFINES)
PLUSOFF3
+OFF
PLUSON3
+ON
MINUSON3
-ON
MINUSOFF3
-OFF
Generate
Profile Code
NO
YES
Generate
Data Division
Code for
Profile
ADD-1HM
Verifies
each Profile
Character
LAST-CONVERT
Generates
Procedure
Division Code
for Profile
Print Error
Message
Exit

ag. LMLPSTE

(1) Function. Handles the STEP INDEX instruction.

(2) Calling Sequence.

```
ENTRY 'LMSTE' USING FM-DD  LP-DD  LM-DD
CALL  'LMPRO' USING FM-CARDS
CALL  'FMPRT' USING PRNT-CARDS  CC
```

(3) Program Description.

GENERATE-STEP.

This paragraph performs the following error checking on the words of the STEP card:

There must be eight (8) words on the card (the label field is always counted as 1, but may or may not be present).

The first word is the label field. No checking is done on this field at this time.

The second word must be 'STEP'.

The fourth word must be a two digit index number and not zero.

The fifth word must be 'BY'.

The sixth word must be an unsigned or signed numeric literal from 1 to 3 digits.

The seventh word must be a period.

The eighth word must be a failure exit. The validation of the eighth word is performed by LMLP.

CHECK-3.

CHECK-1.

Validates increment and insures that it is not greater than 599.

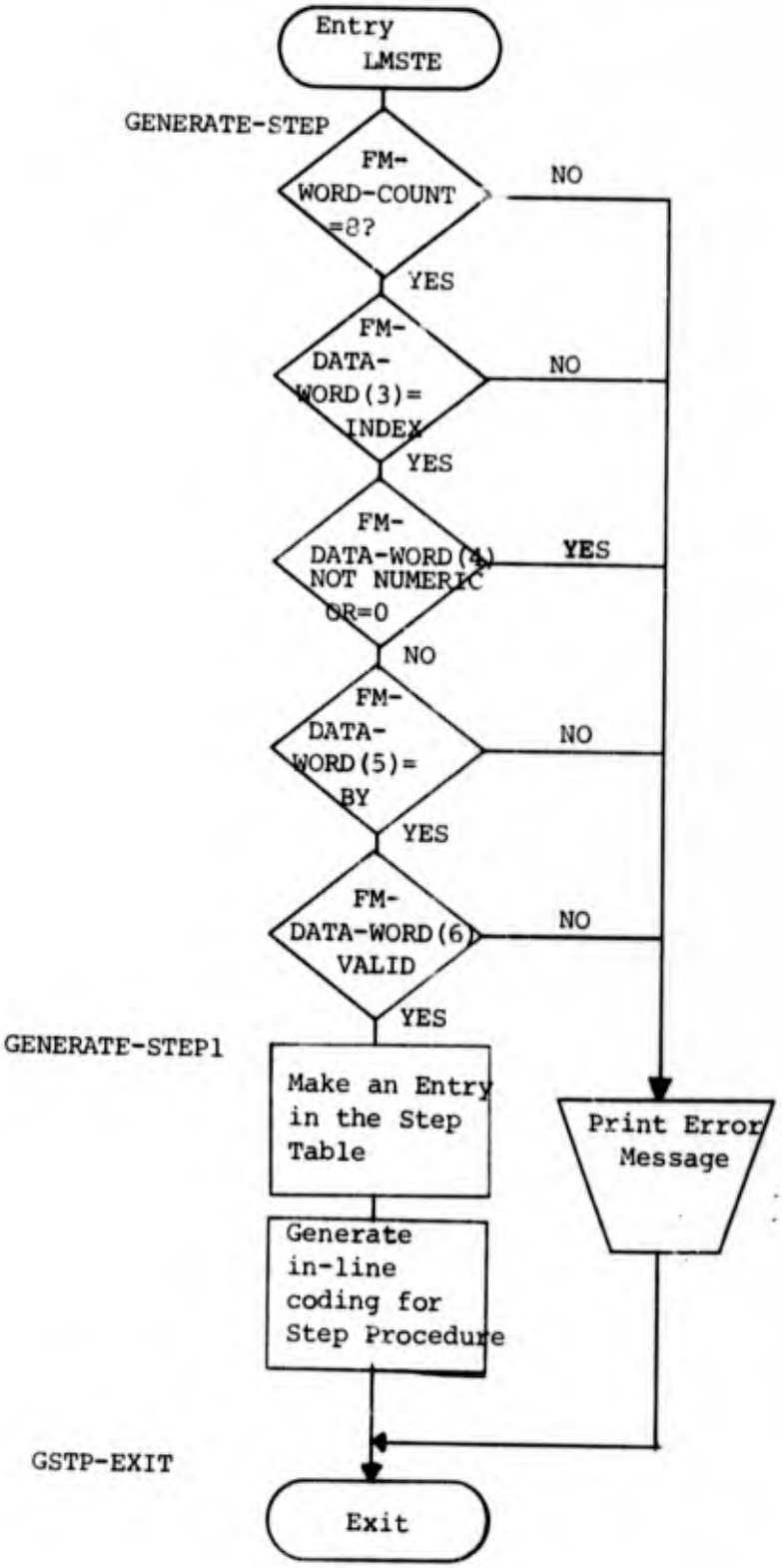
GENERATE-STEP1.

This paragraph generates the in-line coding for the STEP procedure.

STEP-BY-ZERO.

Checks to insure that increment of step is not zero. If it is, warning message is issued indicating that nothing has changed.

(4) Limitations. None.


Entry
LMSTE
GENERATE-STEP
FM-
WORD-COUNT
=0?
NO
YES
FM-
DATA-
WORD(3)=
INDEX
NO
YES
FM-
DATA-WORD(4)
NOT NUMERIC
OR=0
YES
NO
FM-
DATA-
WORD(5)=
BY
NO
YES
FM-
DATA-WORD(6)
VALID
NO
YES
GENERATE-STEP1
Make an Entry
in the Step
Table
Print Error
Message
Generate
in-line
coding for
Step Procedure
GSTP-EXIT
Exit

ah. LMLPSUB

(1) Function. Handles the following two instructions:

ADD field TO field
SUBTRACT field FROM field

(2) Calling Sequence.

```
ENTRY 'LMADD'   USING FM-DD  LP-DD  LM-DD
ENTRY 'LMSUB'   USING FM-DD  LP-DD  LM-DD
CALL  'LMPRO'   USING COBOL-FORMAT1,2,3
CALL  'FMPRT'   USING PRNT-CARDS  CC
CALL  'LMFMT1'  USING FM-DD  LP-DD  LM-DD
```

(3) Program Description.

(a) Paragraph LM-ADD-ROUTINE initializes switches and areas used for the ADD instruction. The field type for the first field is determined by calling LMLPFMT1 and the connecting word is check to be 'TO'. FIELD2 type is determined in CK-FLD2-ADD by calling LMLPFMT1 and a Computed Go To is used to determine initially the first field combination type. Computed Go To statements are used in CK-FLD1-ADD and CK-NDEF-ADD to determine logic flow for the generation of COBOL coding. Paragraph LAR-2 is used only when FIELD1 is a numeric data field and paragraph LAR-3 is used only when FIELD2 is a numeric data field. Paragraph LAR-4 is used when FIELD1 is a numeric literal or numeric defined literal and FIELD2 is a numeric defined literal. Paragraph LAR-5 is used when FIELD1 is a numeric data field and FIELD2 is a numeric defined literal. See ADD TABLE.

(b) Paragraph LM-SUBTRACT-ROUTINE initializes switches and hold areas used for the SUBTRACT instruction. FIELD1 type is determined by calling LMLPFMT1 and the connecting word between fields is checked for the word 'FROM'. The second field type is determined in CK-FLD2-SUB by calling LMLPFMT1 and a Computed Go To is used to initially determine the field types combination. A Computed Go To in CK-FLD2-SUB and CK-NDEF-SUB determines the actual field combination types and logic flow is directed to the appropriate paragraphs for the actual COBOL code generation. Paragraph LSR-2 is used only when FIELD1 is a numeric data field. Paragraph LSR-3 is used when FIELD2 is a numeric data field. Paragraph LSR-4 is used to generate COBOL coding when FIELD1 is a numeric literal or numeric defined literal and FIELD2 is a numeric defined literal. LSR-5 is used when FIELD1 is a numeric data field and FIELD2 is a numeric defined literal. See SUBTRACT TABLE.

PARTIAL-IN-FLD2.

This paragraph generates procedure division coding for the second half of two lines of coding needed for a partial in field2.

PERIOD-GEN-1.

Generates a period at the end of a generated Procedure Division statement for OUTPUT-WORD-1.

CHAR-CK-1.

Generates a period at the end of a generated Procedure Division statement and is initiated by paragraph PERIOD-GEN-1 by a perform routine.

PERIOD-GEN-2.

Generates a period at the end of a generated Procedure Division statement for the output holding area FM-WORD-2.

CHAR-CK-2.

Generates a period at the end of a generated Procedure Division statement.

COBOL-OUT1.

COBOL-OUT2.

COBOL-OUT3.

Prints out the generated COBOL coding.

MOVE-FLD1-TO.

MOVE-FLD2-TO.

Generates the first half of a COBOL move statement.

GIVING-FLD2.

Generates the receiving field of a COBOL move statement.

INITIALIZER.

Initializes all the switches and counters used in the ADD, SUBTRACT, MOVE, DELETE field, SET, and CHANGE routines.

ERROR-OUT1 thru ERROR-OUT10.

Generates error messages.

CK-PARTIAL-FLD2.

Determines if field2 is a partial and channels the logic flow to paragraph PARTIAL-IN-FLD2 if a partial is found.

LM-ADD-ROUTINE THRU LAR-5.

Handles the ADD instruction.

LM-ADD-ROUTINE.

This paragraph initializes switches and storage areas.
LMLPFMT1 is called to determine the field type of FIELD1.
The connector between fields is tested for the word 'TO'.

CK-FLD2-ADD.

LMLPFMT1 is called to determine the field type of FIELD2.
A Computed Go To is used to establish the field combination type for FIELD2.

CK-FLD1-ADD.

A Computed Go To is used to determine what coding is to be generated when FIELD2 is a numeric data field.

CK-NDEF-ADD.

A Computed Go To is used to determine what coding is to be generated when FIELD2 is a numeric defined literal.

LAR-2 .

Generates code for a combination of two numeric data fields.

LAR-3 thru LAR-3A2.

Generates code for a combination of field2 as a numeric data field and field1 as a numeric defined literal or a numeric literal.

LAR-4.

Generates code for a combination of field2 as a numeric defined literal and field1 as a numeric defined literal or numeric literal.

LAR-5.

Generates code for a combination of field2 as a numeric defined literal and field1 as a numeric data field.

LM-SUBTRACT-ROUTINE THRU LSR-5.

Handles the SUBTRACT instruction.

LM-SUBTRACT-ROUTINE.

This paragraph initializes switches and storage areas. LMLPFMT1 is called to determine the field type of field1. The connector between fields is checked for the word 'FROM'.

CK-FLD2-SUB.

LMLPFMT is called to determine the field type of field2. A Computed Go To is used to establish the field combination type of field2.

CK-FLD1-SUB.

A Computed Go To is used to determine which coding is being generated when field2 is a numeric data field.

CK-NDEF-SUB.

A Computed Go To is used to determine which coding is generated when field2 is a numeric defined literal or a numeric literal.

LSR-2 thru LSR-5.

Generate the actual COBOL coding. Which paragraphs are used is dependent upon the combination of field1 and 2 types.

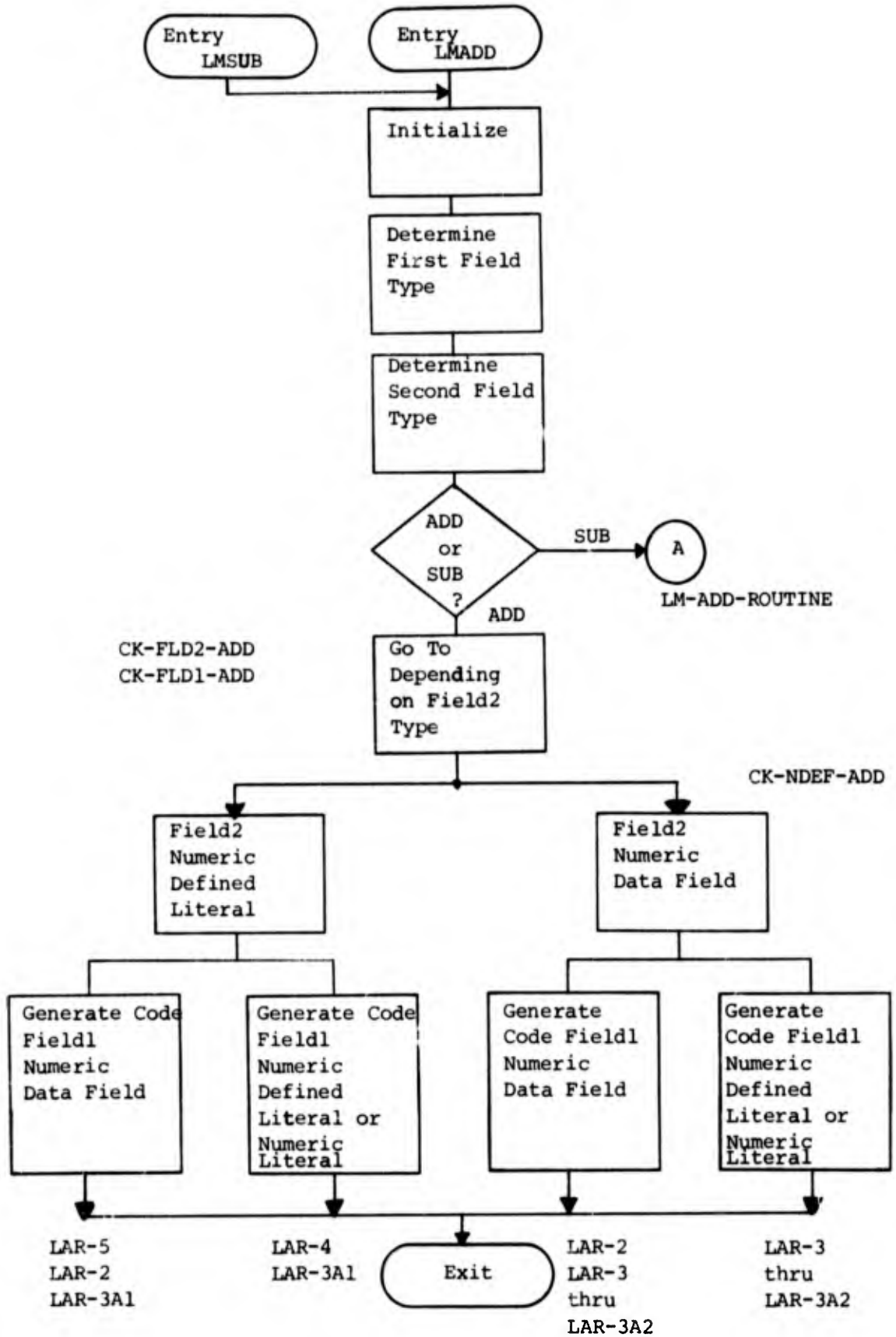
ADD TABLE

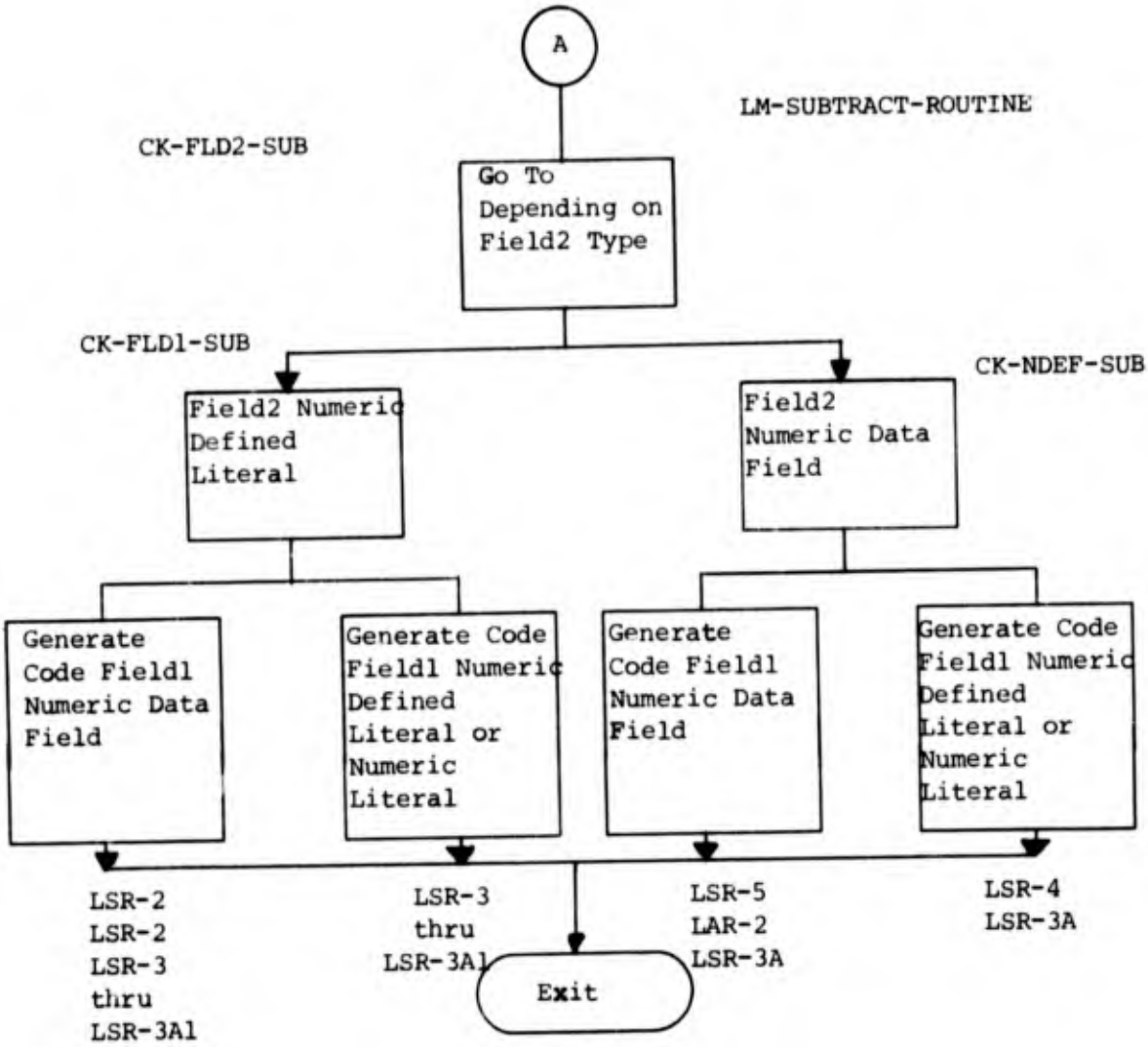
FIELD1 \ FIELD2	A	L	ND	NL	N
A	ERROR	ERROR	ERROR	ERROR	ERROR
L	ERROR	ERROR	ERROR	ERROR	ERROR
ND	ERROR	ERROR	LAR-4	ERROR	LAR-3
NL	ERROR	ERROR	LAR-4	ERROR	LAR-3
N	ERROR	ERROR	LAR-5	ERROR	LAR-2

SUBTRACT TABLE

FIELD1 \ FIELD2	A	L	ND	NL	N
A	ERROR	ERROR	ERROR	ERROR	ERROR
L	ERROR	ERROR	ERROR	ERROR	ERROR
ND	ERROR	ERROR	LSR-3	ERROR	LSR-4
NL	ERROR	ERROR	LSR-3	ERROR	LSR-4
N	ERROR	ERROR	LSR-2	ERROR	LSR-5

(4) Limitations. None.


Entry
LMSUB
Entry
LMADD
Initialize
Determine
First Field
Type
Determine
Second Field
Type
ADD
or
SUB
?
SUB
A
LM-ADD-ROUTINE
ADD
CK-FLD2-ADD
CK-FLD1-ADD
Go To
Depending
on Field2
Type
CK-NDEF-ADD
Field2
Numeric
Defined
Literal
Field2
Numeric
Data Field
Generate Code
Field1
Numeric
Data Field
Generate Code
Field1
Numeric
Defined
Literal or
Numeric
Literal
Generate
Code Field1
Numeric
Data Field
Generate
Code Field1
Numeric
Defined
Literal or
Numeric
Literal
LAR-5
LAR-2
LAR-3A1
LAR-4
LAR-3A1
Exit
LAR-2
LAR-3
thru
LAR-3A2
LAR-3
thru
LAR-3A2


A
LM-SUBTRACT-ROUTINE
CK-FLD2-SUB
Go To Depending on Field2 Type
CK-FLD1-SUB
CK-NDEF-SUB
Field2 Numeric Defined Literal
Field2 Numeric Data Field
Generate Code Field1 Numeric Data Field
Generate Code Field1 Numeric Defined Literal or Numeric Literal
Generate Code Field1 Numeric Data Field
Generate Code Field1 Numeric Defined Literal or Numeric Literal
LSR-2
LSR-2
LSR-3
thru
LSR-3A1
LSR-3
thru
LSR-3A1
LSR-5
LAR-2
LSR-3A
LSR-4
LSR-3A
Exit

ai. LMLPTAB

(1) Function. This program generates the appropriate COBOL code for IN TABLE and NOT IN TABLE operations.

(2) Calling Sequence.

```
ENTRY 'LMTAB'    USING FM-DD  LP-DD  LM-DD
CALL  'FMPRT'    USING PRNT-CARDS  CC
CALL  'LMLPGEN'  USING IF-CRD3
```

(3) Program Description.

GENERATE-IF-CODE.
Generates appropriate COBOL code before the 'IF' conditional.
IF-END.
Generates appropriate code after the 'IF' conditional.
GENERATE-OR-CODE.
Generates appropriate code before the 'OR' conditional.
OR-END.
Generates appropriate code after the 'OR' conditional.
GENERATE-AND-CODE.
Generates the appropriate code before the 'AND' conditional.
AND-END.
generates the appropriate code after the 'AND' conditional.
GENERATE-ORIF-CODE.
Generates the appropriate code before the 'OR IF' conditional.
ORIF-END.
Generates the appropriate code after the 'OR IF' conditional.
SEE-IF-FAIL-EXIT.
This paragraph generates the code for a failure exit.
CHECK-CARD-TYPE-SWITCHES.
This paragraph does a Computed Go To depending on the conditional word and generates the appropriate code after the relational testing.
CHECK-END.
IF-SEQ3-GENER.
This paragraph outputs the coding for both FIELD1 and FIELD2 when their format does not fit in the previous two output areas.
IF-CLOSE-UP.
ICU1-PFRM.
IF-CLOSE-UP1.
IF-CLOSE-UP-EXIT.
GENER-FIRST-PART.
This paragraph does a Computed Go To depending on the conditional and generates the appropriate code previous to the relation testing.
FM-IN-TABLE-PROCESS.
Checks for field in table.

SKIP-ONE-MORE.

Checks to see if the table name is five characters.

NOTEXX.

NOTEXX-END.

Table name must begin with an alphabetic character and end with a 'L.' Checks for a failure exit with no result field.

CHECK-FT-PART-SW.

Positions word-pointer on field name and calls LMFLD to verify the field name. Generates part of the Procedure Division code.

MOVE-DATA-TO.

Generates Procedure Division code. Goes to a paragraph depending on the card format.

OR-IF-CONDITION.

Checks for failure exit on 'OR IF' condition.

NO-RESULT-NO-FAILURE.

Card Type IF FLDA IN TABLE AXXXL.

NO-RESULT-FAILURE.

Card Type IF FLDA IN TABLE AXXXL .FAIL.

RESULT-NO-FAILURE.

Card Type IF FLDA IN TABLE AXXXL RESULT.

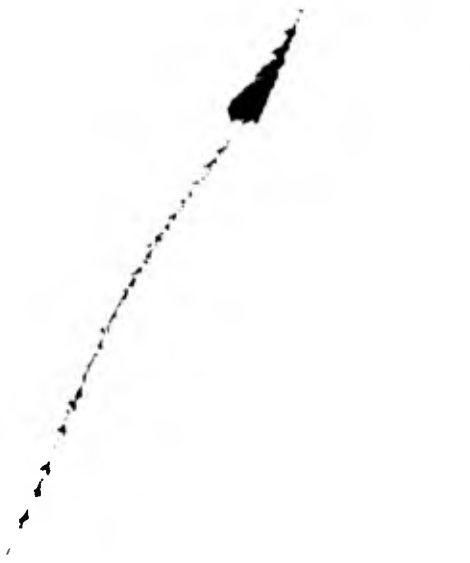
RESULT-FAILURE.

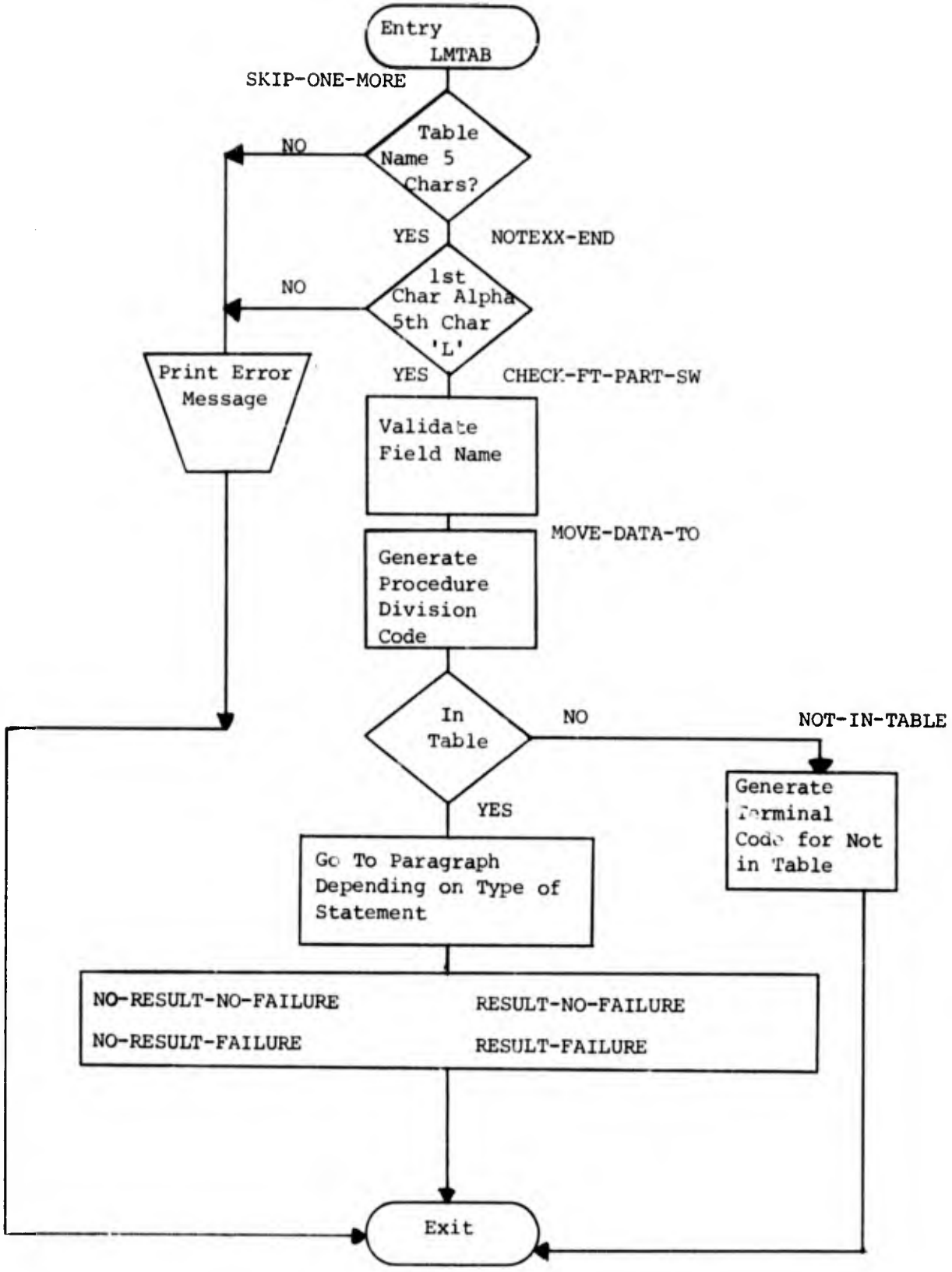
Card Type IF FLDA IN TABLE AXXXL RESULT .FAIL.

NOT-IN-TABLE.

Sets switches for NOT IN TABLE.

(4) Limitations. None.




Entry
LMTAB
SKIP-ONE-MORE
Table
Name 5
Chars?
NO
YES
NOTEXX-END
1st
Char Alpha
5th Char
'L'
NO
YES
CHECK-FT-PART-SW
Print Error
Message
Validate
Field Name
MOVE-DATA-TO
Generate
Procedure
Division
Code
In
Table
NO
NOT-IN-TABLE
YES
Generate
Terminal
Code for Not
in Table
Go To Paragraph
Depending on Type of
Statement
NO-RESULT-NO-FAILURE
RESULT-NO-FAILURE
NO-RESULT-FAILURE
RESULT-FAILURE
Exit

aj. LMLPVPT

(1) Function. Handles the VPRINT instruction. Allows for the print of all or part of the variable set.

(2) Calling Sequence.

```
ENTRY 'LMVPT'  USING FM-DD  LP-DD  LM-DD
CALL  'LMFLD'  USING FM-DD  LP-DD  LM-DD
CALL  'LMPRO'  USING CARD-IMAGE
CALL  'FMPRT'  USING PRNT-CARDS  CC
```

(3) Program Description.

VPRINT-PROCESS.
Validates the field name. Checks to insure that the field name is a VSET. Builds the first part of the partial (in form mmmm-NNNN).

PUT-IN-ZEROS-1.
Inserts leading zeroes to fill out a four character field for the first part of the partial STOP-MOVING-ZEROS-1.

MOVE-MMMM.
Verify that each character is numeric and move them to the output area.

CONTINUE-EXAMINING.
Builds the second part of the partial PUT-IN-ZEROS-2.
Inserts leading zeroes to fill out a four character field for the second part of the partial.

STOP-MOVING-ZEROS-2.
Verifies that each character is numeric and moves them to the output area.

VERIFY-VSET-PARTIAL.
The first part of the partial must be smaller than the second part.

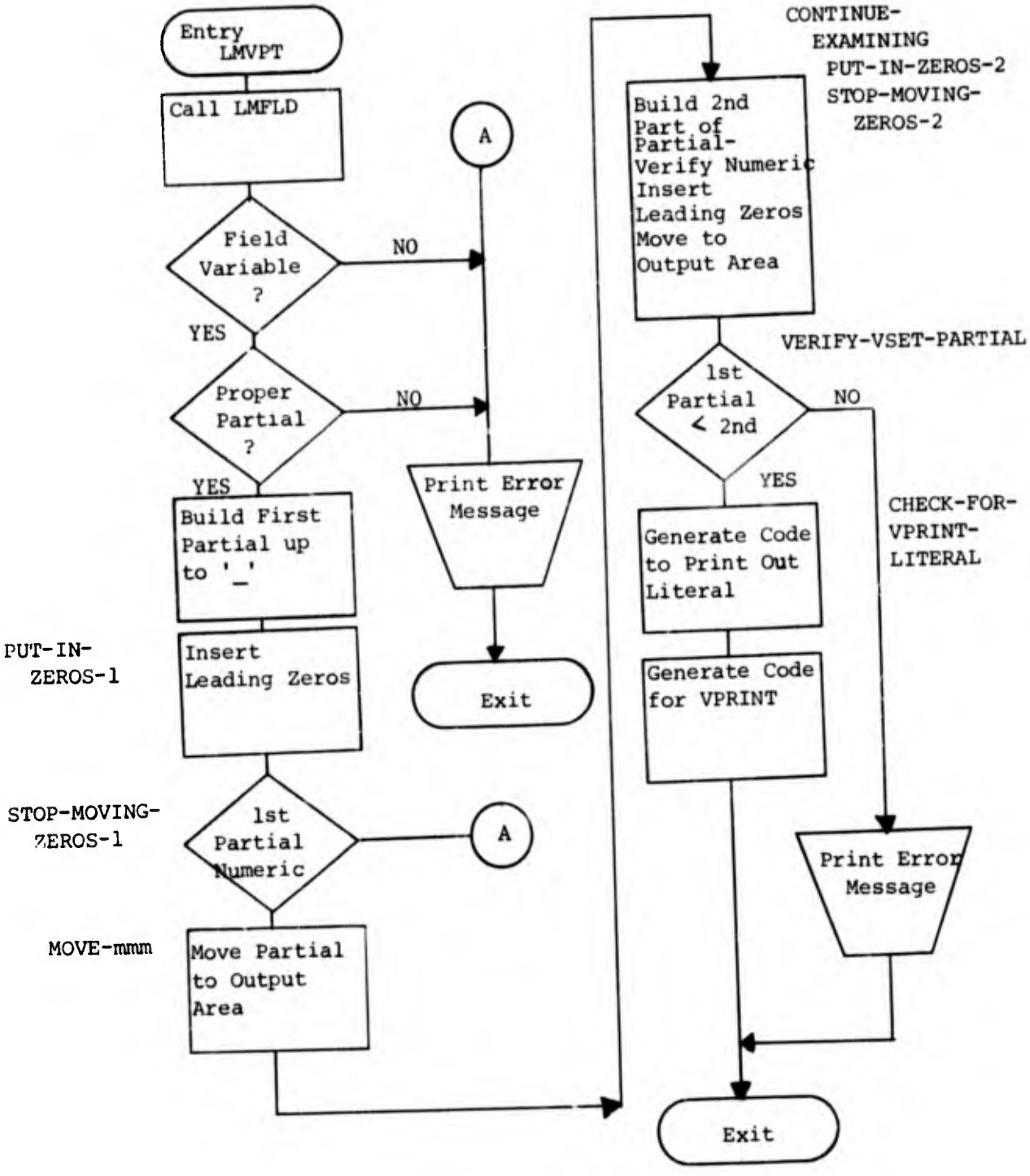
CHECK-FOR-VPRINT-LITERAL.
Determines if there is a literal associated with the portion of the variable set desired. If found, produces the code that will write that literal out on a line by itself.

NO-VPRINT-LITERAL-CODE.
Generates COBOL coding when there is no literal associated with the VSET.

CHECK-FILE-TYPE.
Determines if VSET is part of the major file or minor file.

CHECK-FILE-TYPE-EXIT.
Generates the code to print out 132 characters per line of the variable set. If a number greater than the number of characters in the variable set is specified, it is replaced by the actual number of characters in the variable set.

(4) Limitations. None.


Entry
LMVPT
Call LMFLD
Field
Variable
?
NO
YES
Proper
Partial
?
NO
YES
Build First
Partial up
to '_'
Print Error
Message
A
PUT-IN-
ZEROS-1
Insert
Leading Zeros
Exit
STOP-MOVING-
ZEROS-1
1st
Partial
Numeric
A
MOVE-mmm
Move Partial
to Output
Area
CONTINUE-
EXAMINING
PUT-IN-ZEROS-2
STOP-MOVING-
ZEROS-2
Build 2nd
Part of
Partial-
Verify Numeric
Insert
Leading Zeros
Move to
Output Area
VERIFY-VSET-PARTIAL
1st
Partial
< 2nd
NO
YES
CHECK-FOR-
VPRINT-
LITERAL
Generate Code
to Print Out
Literal
Generate Code
for VPRINT
Print Error
Message
Exit

ak. LMLPWP0

(1) Function. This program verifies that for all FAILURE EXITS there exist corresponding LABELS. It also scans the BUILD table, STEP table, and DELETE table to insure that the proper set index combinations have been assigned.

(2) Calling Sequence.

```
ENTRY 'LMWPØ'      USING FM-DD  LP-DD  LM-DD
CALL  'FMPRT'      USING PRNT-CARDS  CC
CALL  'DATESUB'    USING ITEM-DATE
CALL  'LB'         USING CALLING-SEQ  FM-DEF-AREA
CALL  'LB'         USING CALLING-SEQ  DEF-VALUES
CALL  'LB'         USING CALLING SEQ  XXXX-PØØ
CALL  'LMPRO'      USING CARD-IMAGE
```

(3) Program Description.

PERFORM-FAIL-LU.
Determines if all labels have been defined.

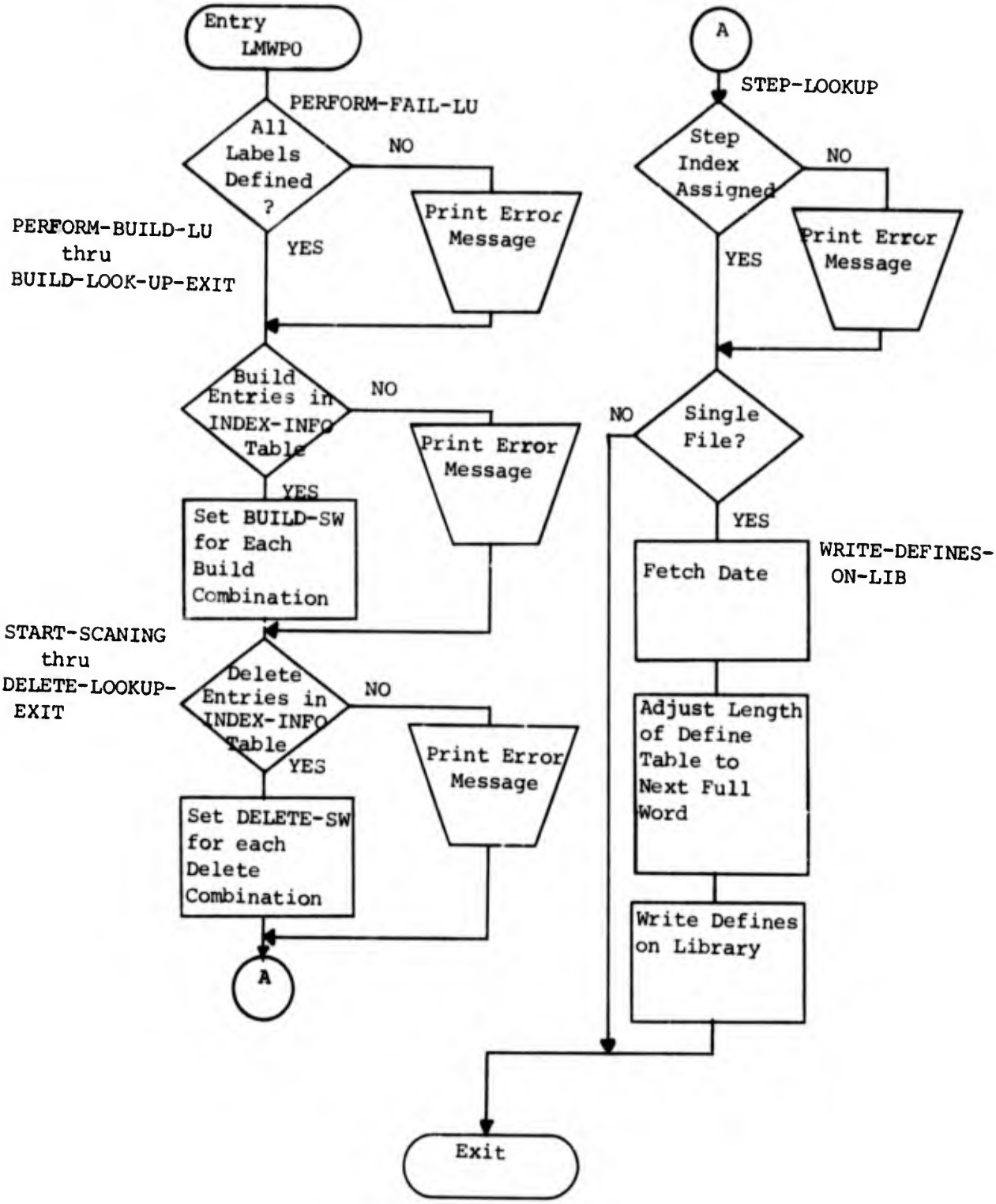
PERFORM-BUILD-LU thru BLD-LOOK-UP-EXIT.
Insures that all index/set combinations in the BUILD-TABLE are defined in the INDEX-INFO table. BUILD-SW is set in INDEX-INFO for each of these combinations.

START-SCANING thru DELETE-LOOKUP-EXIT.
Insures that all index/set combinations in the DELETE-TABLE are defined in the INDEX-INFO table. DELETE-SW is set in INDEX-INFO for each of these combinations.

STEP-LOOKUP
Insures that all step/index combinations in the STEP-TABLE have been assigned.

WRITE-DEFINES-ON-LIB
Call DATESUB to get current date. Adjust length of the DEFINE-TABLE to the next full word so that it can be passed. It writes the define table out on the library. Three entries are put on the library ØØ's the logic package name with which the defines are associated, Ø1 is the define names and sizes, Ø2 is the string of values for all of the defines. Also, contains the total length of the define area as adjusted for boundary alignment.

(4) Limitations. None.


Entry
LMWP0
PERFORM-FAIL-LU
All Labels Defined ?
NO
Print Error Message
YES
PERFORM-BUILD-LU
thru
BUILD-LOOK-UP-EXIT
Build Entries in INDEX-INFO Table
NO
Print Error Message
YES
Set BUILD-SW for Each Build Combination
START-SCANING
thru
DELETE-LOOKUP-
EXIT
Delete Entries in INDEX-INFO Table
NO
Print Error Message
YES
Set DELETE-SW for each Delete Combination
A
A
STEP-LOOKUP
Step Index Assigned
NO
Print Error Message
YES
NO
Single File?
YES
Fetch Date
WRITE-DEFINES-
ON-LIB
Adjust Length of Define Table to Next Full Word
Write Defines on Library
Exit

a1. LMLPWP1

(1) Function. This program in conjunction with LMLPWP5 generates the Procedure Division code for the BUILD and STEP statements using the tables previously built.

(2) Calling Sequence.

```
ENTRY 'LMWP1'  USING FM-DD  LP-DD  LM-DD
CALL  'LMPRO'  USING CARD-IMAGE
CALL  'FMPRT'  USING PRNT-CARDS  CC
CALL  'LMWP5'  USING FM-DD  LP-DD  LM-DD
```

(3) Program Description.

START-SCANNING1
: Initiates the scanning of the LAST-CODE table for non-zero entries, each of these indicates that an index (which is the value of the subscript for that entry) has been assigned. Various step, build, attach, assign, and delete paragraphs must be generated.

INDEX-SCAN
: Initiates the generation of step, build, assign, and delete paragraphs as needed for the various indexes.

GEN-STEP-XNN
: Generates the coding for the step paragraphs (STEP-XNN and STEP-SUBSET-XNN) for this index, NN.

GO-STEP.
: Determines if the index being pointed at in INDEX-INFO table is the one being processed for a step operation. If so, an entry is made in STEP-TABLE and one of the following is generated:

  Major file periodic set - generate STEP-aa-XNN where NN is the index number and aa is the code-number from INDEX-INFO table.

  Major file variable set - generates STEP-ØØ-XNN where NN is the index number.

  Minor file periodic set - generates STEP-aa-XNN where NN is the index number and aa is the code-number from INDEX-INFO table.

  Minor file variable set - generates STEP-99-XNN where NN is the index number.

INDEX-INFO
: Scans the INDEX-INFO table for each assignment of an index, NN. (The value of the index (NN) is given by SUB1.) The number of entries that will be found in the INDEX-INFO table is equal to the value of the entry in LAST-CODE table subscripted by SUB1.

GEN-BUILD-XNN

Generates the coding for the BUILD-XNN.

INFO-CHECK.

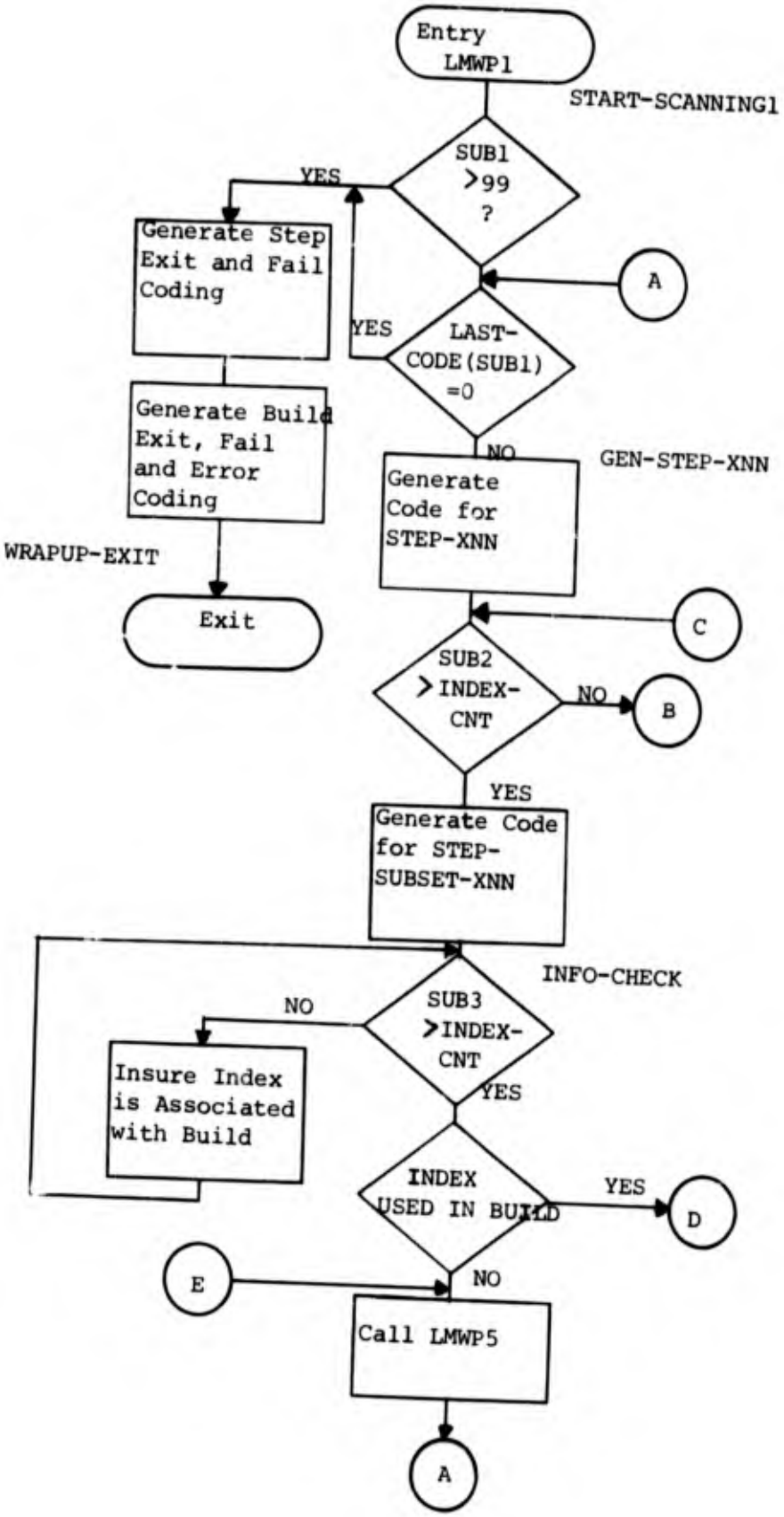
Scans the INDEX-INFO table for each index and checks if it is associated with a build card (build-switch not equal to space).

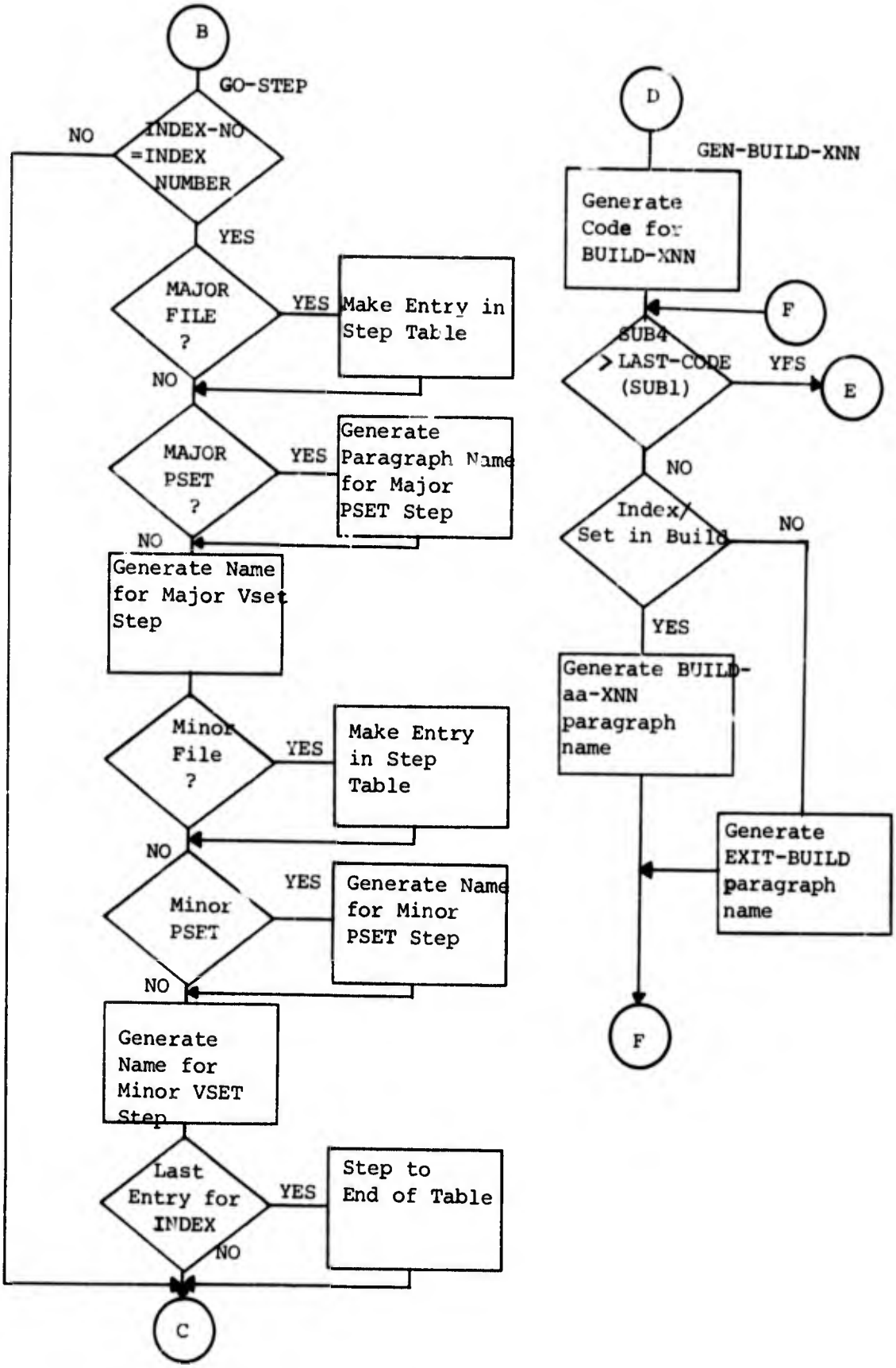
A pointer is placed in SUB-NO-TABLE at the entry in INDEX-INFO.

GO-GEN.

Generates the BUILD-aa-XNN paragraph names for each index (NN) and set combination used in a build operation. For each set assigned to this index which is not used in a build, "EXIT-BUILD" is generated. "aa" is the number of sets previously assigned to this index, including the current one.

(4) Limitations. None.


Entry
LMWP1
START-SCANNING1
SUB1
>99
?
YES
Generate Step
Exit and Fail
Coding
A
YES
LAST-
CODE(SUB1)
=0
Generate Build
Exit, Fail
and Error
Coding
NO
GEN-STEP-XNN
Generate
Code for
STEP-XNN
WRAPUP-EXIT
Exit
C
SUB2
>INDEX-
CNT
NO
B
YES
Generate Code
for STEP-
SUBSET-XNN
INFO-CHECK
SUB3
>INDEX-
CNT
NO
Insure Index
is Associated
with Build
YES
INDEX
USED IN BUILD
YES
D
E
NO
Call LMWP5
A


B
GO-STEP
INDEX-NO =INDEX NUMBER
NO
YES
MAJOR FILE ?
YES
Make Entry in Step Table
NO
MAJOR PSET ?
YES
Generate Paragraph Name for Major PSET Step
NO
Generate Name for Major Vset Step
Minor File ?
YES
Make Entry in Step Table
NO
Minor PSET
YES
Generate Name for Minor PSET Step
NO
Generate Name for Minor VSET Step
Last Entry for INDEX
YES
Step to End of Table
NO
C
D
GEN-BUILD-XNN
Generate Code for BUILD-XNN
F
SUB4 > LAST-CODE (SUB1)
YES
E
NO
Index/ Set in Build
NO
YES
Generate BUILD-aa-XNN paragraph name
Generate EXIT-BUILD paragraph name
F

am. LMLPWP2

(1) Function. This program generates Procedure Division code for the ATTACH statement by use of switches previously set. Generates code for expand-record, attach major-to-major, attach LIT-to-major, attach minor-to-major.

(2) Calling Sequence.

```
ENTRY 'LMWP2'  USING FM-DD  LP-DD  LM-DD
CALL  'LMDAT'  USING CARD-IMAGE
CALL  'LMPRO'  USING CARD-IMAGE
CALL  'FMPRT'  USING PRNT-CARDS  CC
```

(3) Program Description.

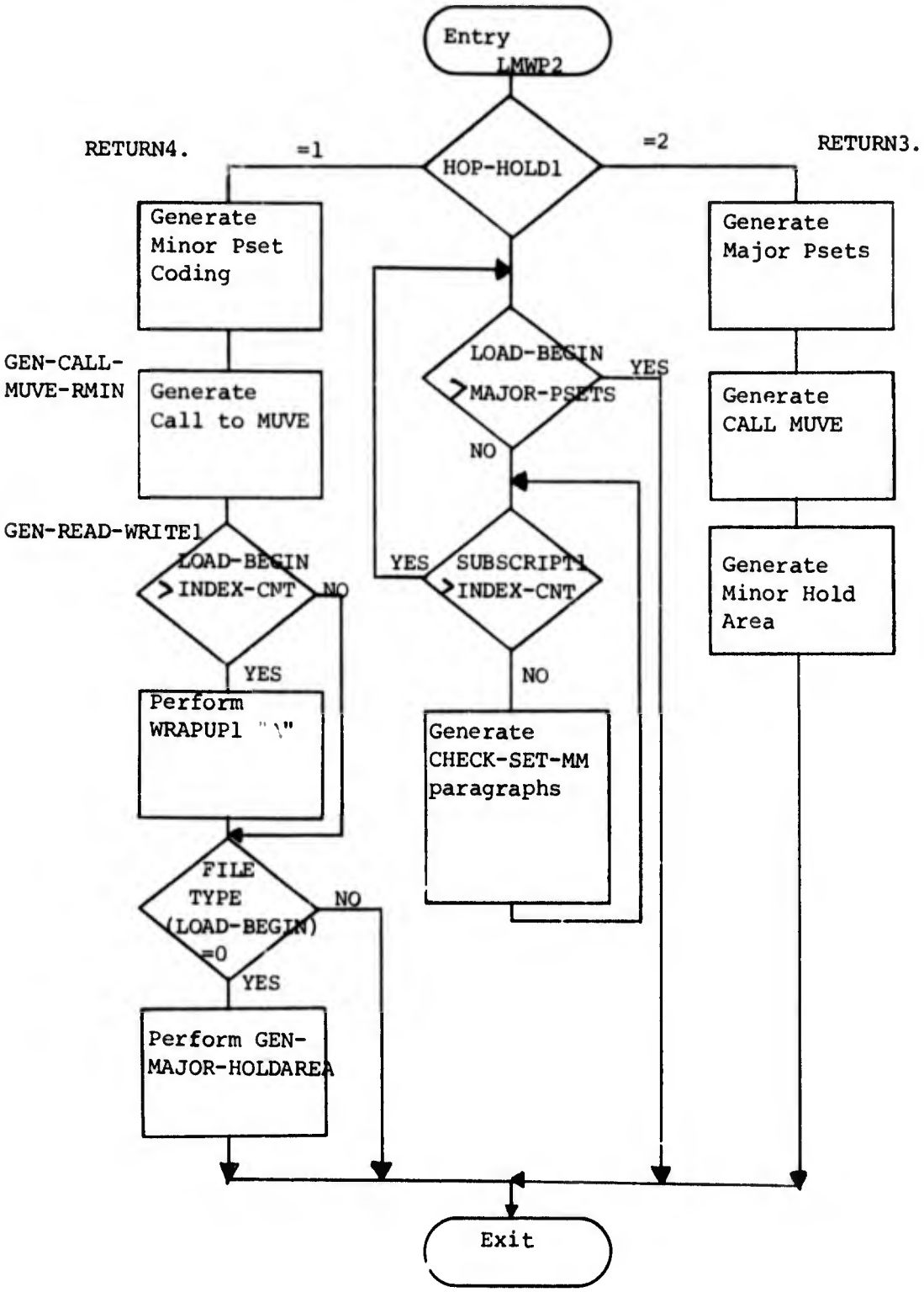
GEN-CHECK-SET-MM thru GEN-SET-EXIT.
Generates the CHECK-SET-MM for each major file periodic set (MM). This coding insures that all work areas created for subsets are written back into the record.

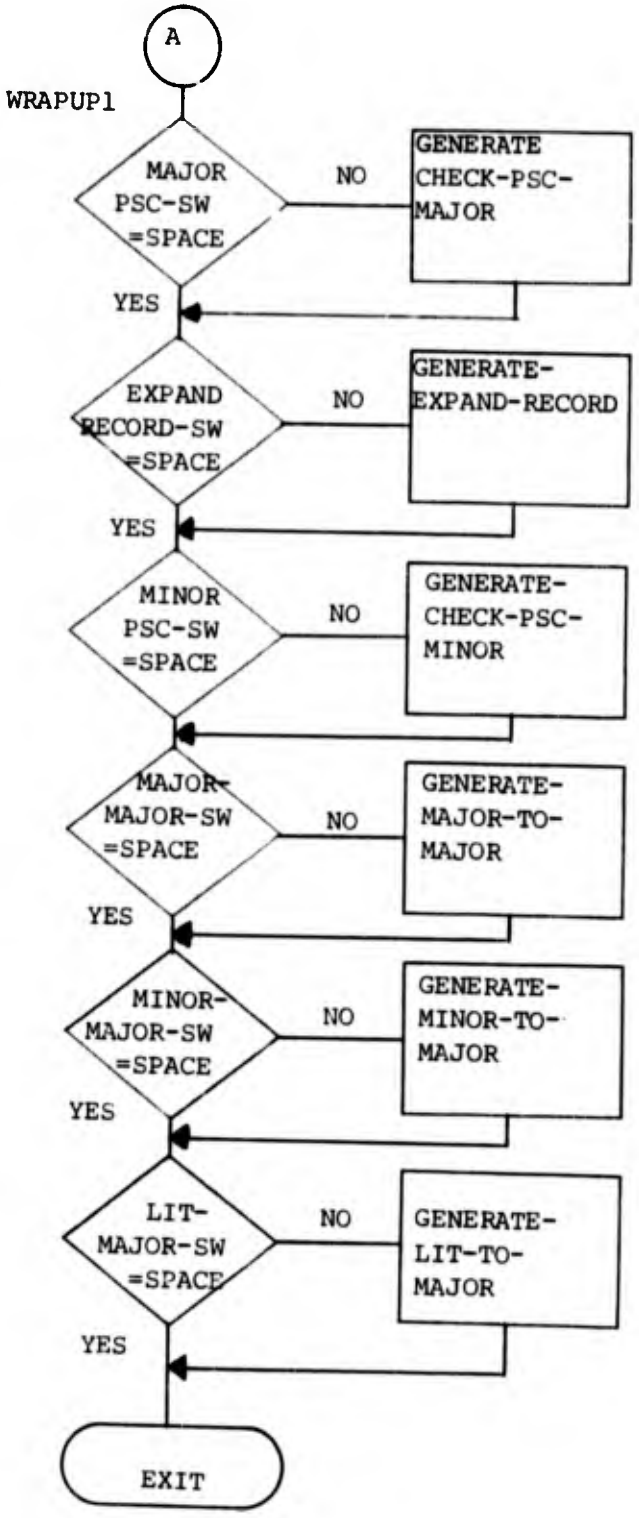
GEN-READ-AND-WRITES. and
GEN-READ-WRITE1 thru GEN-READ-WRITE1-EXIT.
Generates the coding to move data from a major file periodic set, and move it back in. Also to move data from a minor file periodic set to a work area.

WRAPUP1
Initiates the generation of the coding for the following paragraphs:

GEN-CHECK-PSC-MAJOR
GENERATE-EXPAND-RECORD
GENERATE-CHECK-PSC-MINOR
GENERATE-MAJOR-TO-MAJOR
GENERATE-MINOR-TO-MAJOR
GENERATE-LIT-TO-MAJOR

(4) Limitations. None.


Entry
LMWP2
HOP-HOLD1
RETURN4.
=1
=2
RETURN3.
Generate
Minor Pset
Coding
Generate
Major Psets
GEN-CALL-
MUVE-RMIN
Generate
Call to MUVE
LOAD-BEGIN
>MAJOR-PSETS
YES
NO
Generate
CALL MUVE
GEN-READ-WRITE1
LOAD-BEGIN
>INDEX-CNT
NO
YES
SUBSCRIPT1
>INDEX-CNT
YES
NO
Generate
Minor Hold
Area
Perform
WRAPUP1 "\"
Generate
CHECK-SET-MM
paragraphs
FILE
TYPE
(LOAD-BEGIN)
=0
NO
YES
Perform GEN-
MAJOR-HOLDAREA
Exit


A
WRAPUP1
MAJOR PSC-SW =SPACE
NO
GENERATE CHECK-PSC-MAJOR
YES
EXPAND RECORD-SW =SPACE
NO
GENERATE-EXPAND-RECORD
YES
MINOR PSC-SW =SPACE
NO
GENERATE-CHECK-PSC-MINOR
MAJOR-MAJOR-SW =SPACE
NO
GENERATE-MAJOR-TO-MAJOR
YES
MINOR-MAJOR-SW =SPACE
NO
GENERATE-MINOR-TO-MAJOR
YES
LIT-MAJOR-SW =SPACE
NO
GENERATE-LIT-TO-MAJOR
YES
EXIT

an. LMLPWP3

(1) Function. This program generates Procedure Division code for the DELETE statement by use of the DELETE table which was previously built. Generates code for contract-record, and all disassigns. (This latter function allows the user to use the same index any number of times logically with only one combination active at any one time.)

(2) Calling Sequence.

```
ENTRY 'LMWP3'  USING FM-DD  LP-DD  LM-DD
CALLS 'LMPRO' USING CARD-IMAGE
```

(3) Program Description.

GEN-CONTRACT-RECORD.
Generates the COBOL code used during the DELETE operation. Performs the function of closing up the record after a portion has been deleted.

GENERATE-DELETE-SET.
Generates the code which controls the processing of the DELETE operation.

GEN-WRITE-DISASSIGNS.
Generates a logic package paragraph to perform the disassignings required as part of the implementation of the 'WRITE' verb.

GEN-DSN-PARA.
Generates sentences of the form:

```
PERFORM DISASSIGN-X** THRU DISASSIGN-EXIT
MOVE ZERO TO SET-X(**)
```

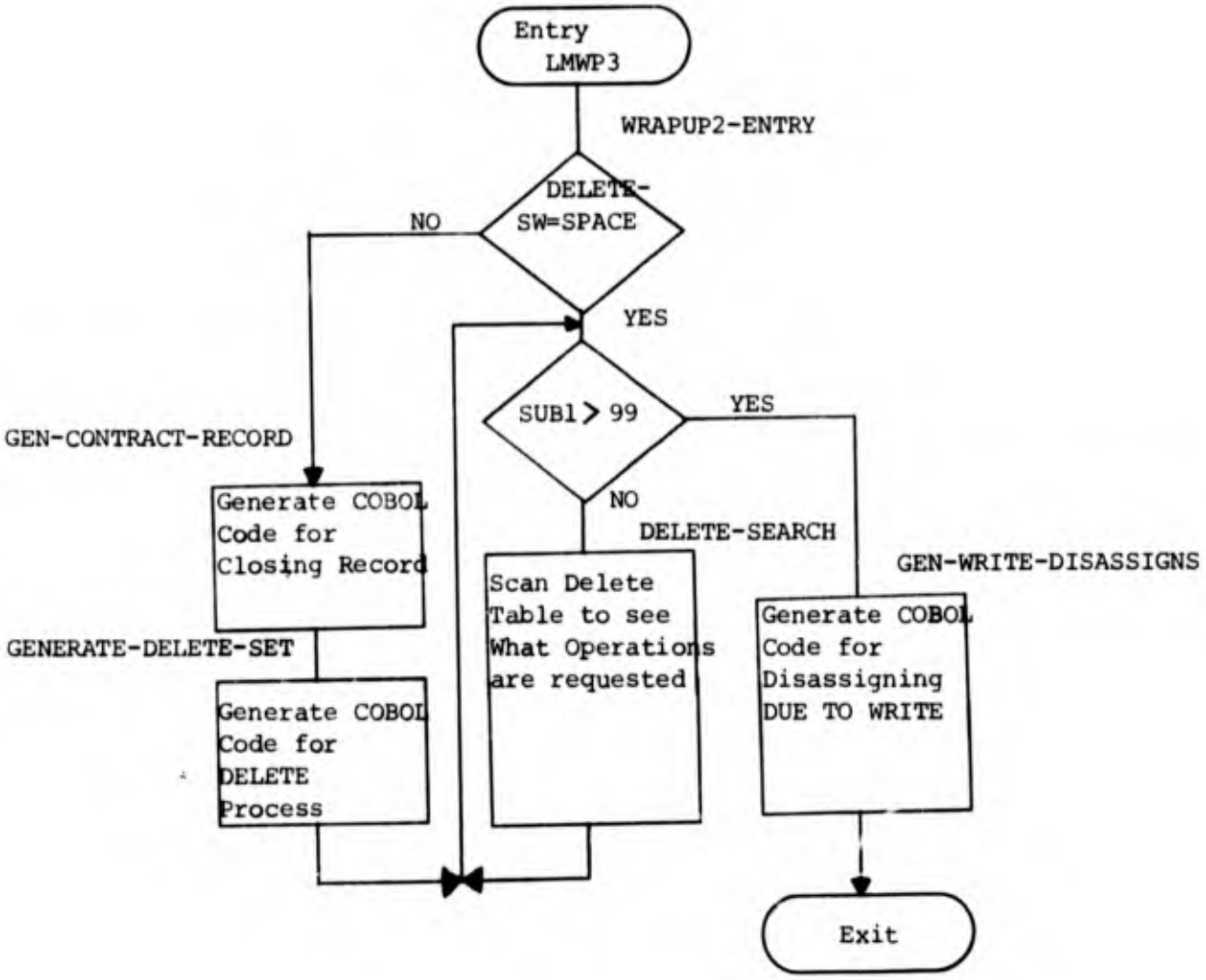
where ** is some two digit number for which an index must be disassigned at the processing of the write verb.

GEN-DELETE-XNN.
Generates the codes to delete the desired periodic set/subset, variable set, fixed set or record.

DELETE-LU.

DELETE-SEARCH.
Scans the DELETE table to see what DELETE operations are requested.

(4) Limitations. None.


Entry
LMWP3
WRAPUP2-ENTRY
DELETE-
SW=SPACE
NO
YES
SUB1 > 99
YES
NO
GEN-CONTRACT-RECORD
Generate COBOL
Code for
Closing Record
GENERATE-DELETE-SET
Generate COBOL
Code for
DELETE
Process
DELETE-SEARCH
Scan Delete
Table to see
What Operations
are requested
GEN-WRITE-DISASSIGNS
Generate COBOL
Code for
Disassigning
DUE TO WRITE
Exit

ao. LMLPWP4

(1) Function. This program generates Data Division coding needed for user calling sequence, I/O areas, and VPRINT. It also generates Procedure Division code to open and close I/O files and to do I/O functions.

(2) Calling Sequence.

```
ENTRY 'LMWP4'  USING FM-DD  LP-DD  LM-DD
CALL  'LMDAT'  USING CARD-IMAGE
CALL  'LMPRO'  USING CARD-IMAGE
```

(3) Program Description.

(a) The initial part of LMWP4 generates the Data Division entries that are needed only one time. The most important of these is INDEX-INFO which is used during the processing of ASSIGNS, BUILDS, ATTACHES, STEPS and DELETES. Also generated as the standard USER-CALLING-SEQUENCE, the output areas used in subroutine mode, the data division elements needed for VPRINT.

(b) The remainder of the program generates the procedure division code used to control the input/output functions. I/O functions are controlled by use of RETRN-CODE and a unique value is placed in it. SET-EXIT thru EXIT-RETURN is then performed. These paragraphs do a Computed Go To depending on the value in RETRN-CODE and thereby get to the functional paragraph which will perform the desired I/O operation. The remainder of the program generates the code for the individual I/O paragraphs. In case of single-file runs much of the I/O is illegal within the logic package as it is handled in MP. Therefore, only the paragraph name and an exit is generated.

CLOSE-FILE-START
: Generates the closing of the major-file, card-file, punch-file, and print-file for file-to-file runs.

CLOSE-FILE-SKIP
: For single-file runs generates the return linkage, otherwise generates STOP RUN. Generates the call to LINK for user called subroutines. For single-file runs generates a call to FMPRT for printing.

PRINT-SKIP
: For file-to-file runs generates the code to read the major file and controls the opening and closing of the file.

MAJOR-READ-SKIP

For file-to-file runs generates the code to read the minor file and controls the opening and closing of the file. For single file runs generates a call to FMCRD in order to read a card. For file-to-file runs generates the code to read a card.

CARD-SKIP

For file-to-file runs generates the code to read a tape and controls the opening and closing of the tape.

TAPE-SKIP

For file-to-file runs generates the code to write the major record and blanks out the output area after writing.

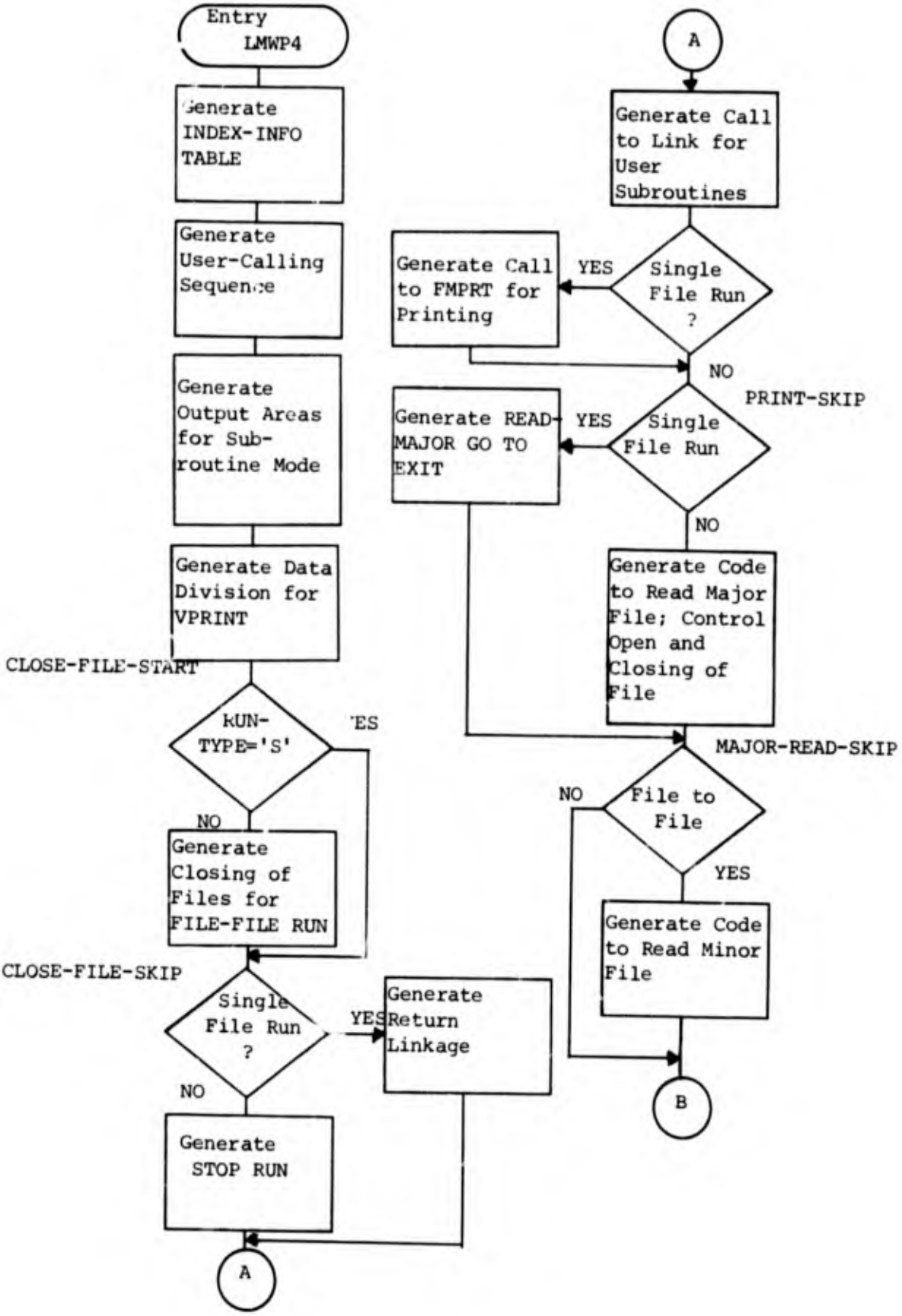
MAJOR-WRITE-SKIP

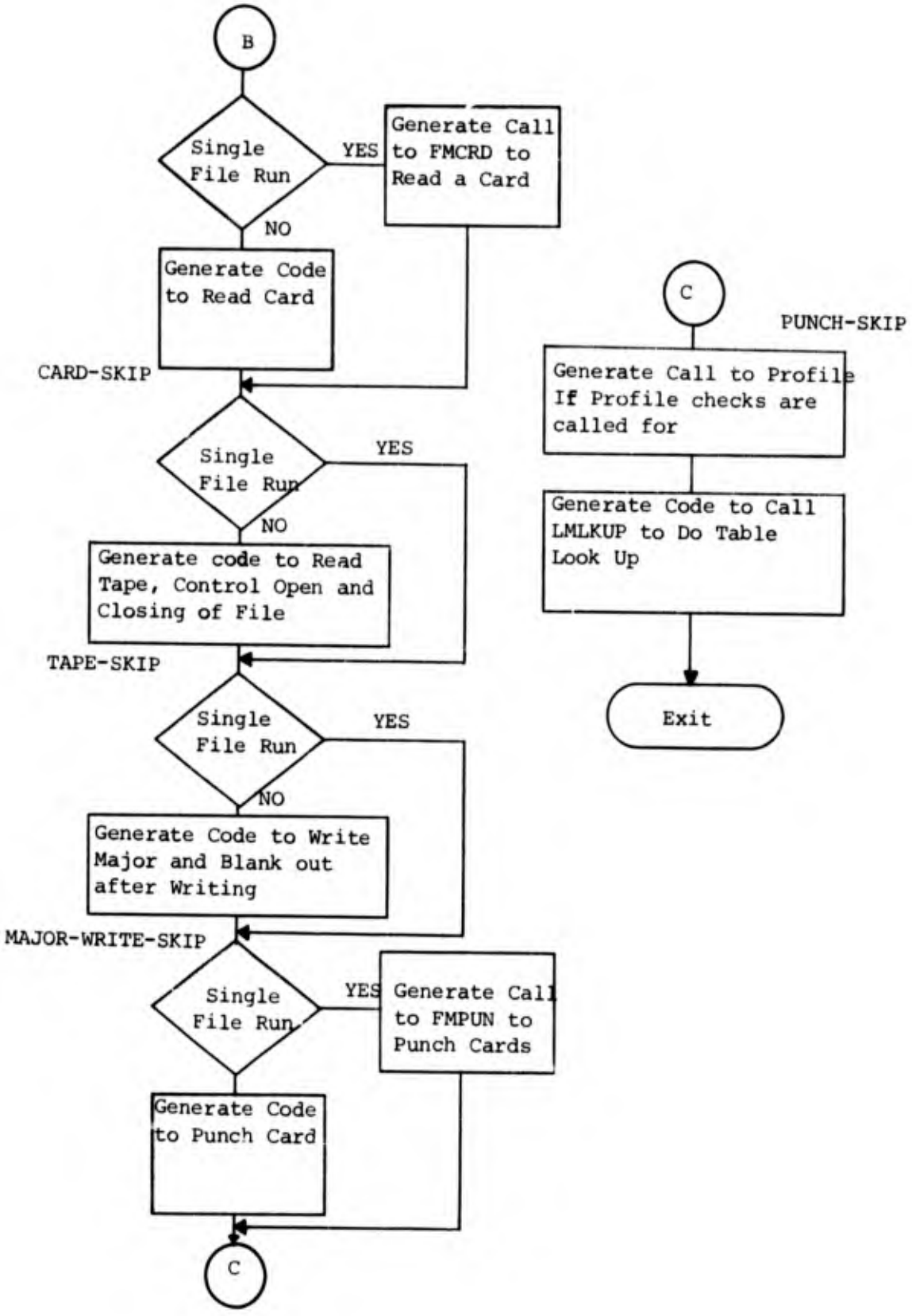
For single file runs generates the code to call FMPUN in order to punch cards. Otherwise generates the code to punch the cards.

PUNCH-SKIP

Generates a call to PROFILE if the PROFILE operation was called for in the logic package. Generates the code to call LMLKUP in order to do table-lookups.

(4) Limitations. None.


Entry
LMWP4
Generate INDEX-INFO TABLE
Generate User-Calling Sequence
Generate Output Areas for Sub-routine Mode
Generate Data Division for VPRINT
CLOSE-FILE-START
RUN-TYPE='S'
YES
NO
Generate Closing of Files for FILE-FILE RUN
CLOSE-FILE-SKIP
Single File Run ?
YES
Generate Return Linkage
NO
Generate STOP RUN
A
A
Generate Call to Link for User Subroutines
Single File Run ?
YES
Generate Call to FMPRT for Printing
NO
PRINT-SKIP
Single File Run
YES
Generate READ-MAJOR GO TO EXIT
NO
Generate Code to Read Major File; Control Open and Closing of File
MAJOR-READ-SKIP
File to File
NO
YES
Generate Code to Read Minor File
B


B
Single File Run
YES
Generate Call to FMCRD to Read a Card
NO
Generate Code to Read Card
CARD-SKIP
Single File Run
YES
NO
Generate code to Read Tape, Control Open and Closing of File
TAPE-SKIP
Single File Run
YES
NO
Generate Code to Write Major and Blank out after Writing
MAJOR-WRITE-SKIP
Single File Run
YES
Generate Call to FMPUN to Punch Cards
Generate Code to Punch Card
C
C
PUNCH-SKIP
Generate Call to Profile If Profile checks are called for
Generate Code to Call LMLKUP to Do Table Look Up
Exit

ap. LMLPWP5

(1) Function. This program generates Procedure Division code in conjunction with LMLPWP1 for the BUILD and STEP statements by using tables previously built.

(2) Calling Sequence.

```
ENTRY 'LMWP5'  USING FM-DD  LP-DD  LM-DD
CALL  'LMPRO'  USING CARD-IMAGE
CALL  'FMPRT'  USING PRNT-CARDS  CC
```

(3) Program Description.

GEN-DISASSIGN-XNN
Generates the DISASSIGN-XNN for each index used in a logic package. A subparagraph is generated for each set associated with this index (NN).

INDEX-PAR
Generates the STEP-aa-XNN paragraphs for this index (NN), and the BUILD-aa-XNN paragraphs if the index was used in a build operation, i.e. BUILD-SW≠b.

GEN-STEP-aa-XNN
Generates the in-line coding for the step (index/set) paragraphs:

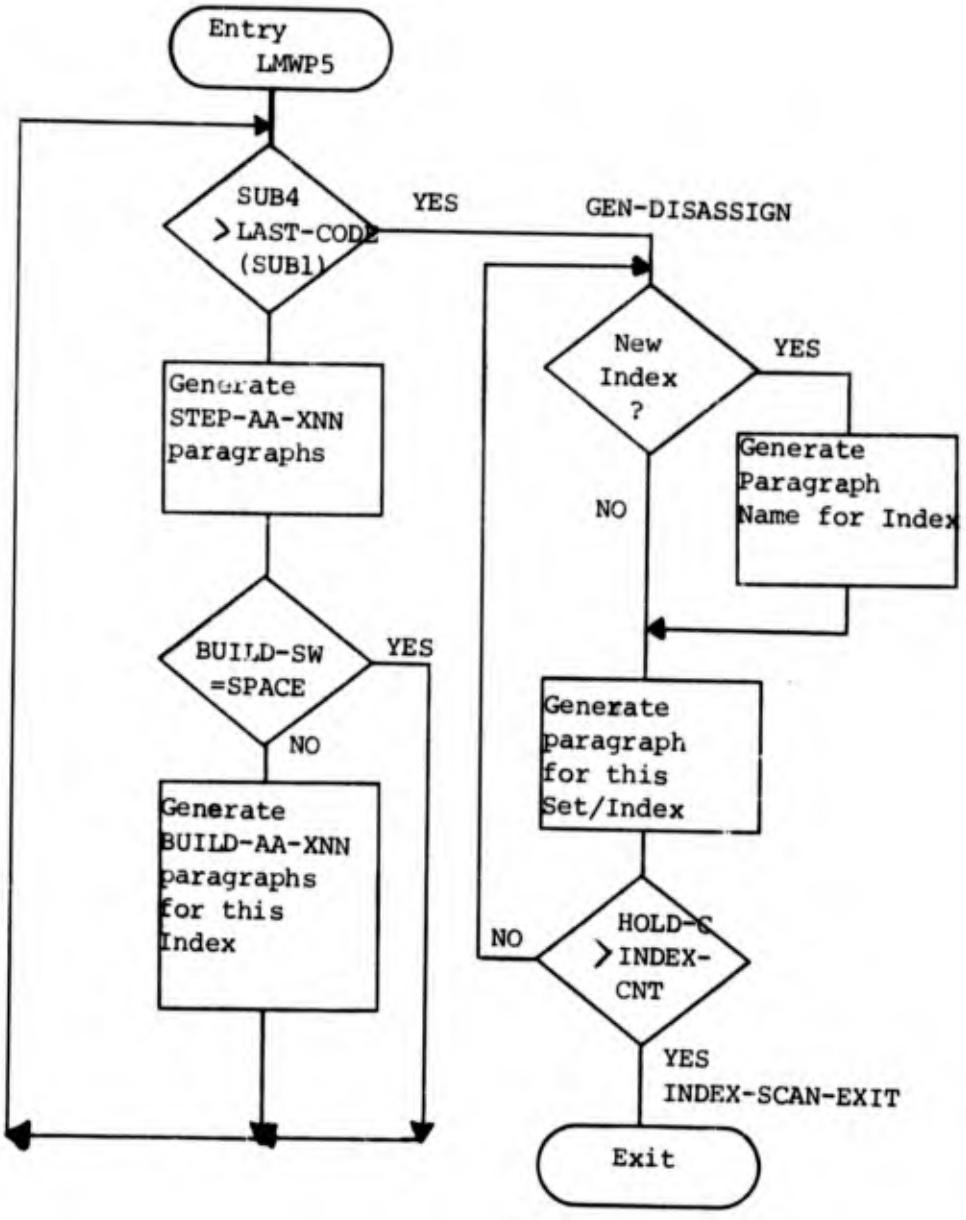
STEP-aa-XNN (where NN is the index number) for major file and minor file periodic sets.

STEP-ØØ-XNN for major file variable sets.

STEP-99-XNN for minor file variable sets.

BUILD-aa-XNN.
Generates the in-line coding for every assignment of index NN used in a build operation. "aa" is the CODE-NO entry in the INDEX-INFO table.

(4) Limitations. None.


Entry
LMWP5
SUB4
>LAST-CODE
(SUB1)
YES
GEN-DISASSIGN
Generate
STEP-AA-XNN
paragraphs
BUILD-SW
=SPACE
YES
NO
Generate
BUILD-AA-XNN
paragraphs
for this
Index
New
Index
?
YES
Generate
Paragraph
Name for Index
NO
Generate
paragraph
for this
Set/Index
HOLD-C
>INDEX-
CNT
NO
YES
INDEX-SCAN-EXIT
Exit

aq. MLMPZNO

(1) Function. This program controls the processing for the MOVE ZON (ZONES) instruction. It performs error checking functions and determines the field types of field1 and field2.

(2) Calling Sequence.

```
ENTRY 'LMZNO'  USING FM-DD  LP-DD  LM-DD
CALLS 'FMPRT'  USING PRNT-CARDS  CC
```

(3) Program Description. Entry is made to LMLPZNO through a CALL from LMLPMOV. The type receiving field is determined in paragraph ZON-FLD2-RTN and logic flow is directed from that paragraph using a Computed Go To depending on field2 type. A switch is set indicating the field2 type. Logic flow returns to LMLPMOV for the calling of LMLPZN1, 2 or 3 dependent upon the switch setting.

ZON-FLD2-RTN.
Error checking is performed for illegal receiving field types. A Computed Go To is used for determining the combination type of field2.

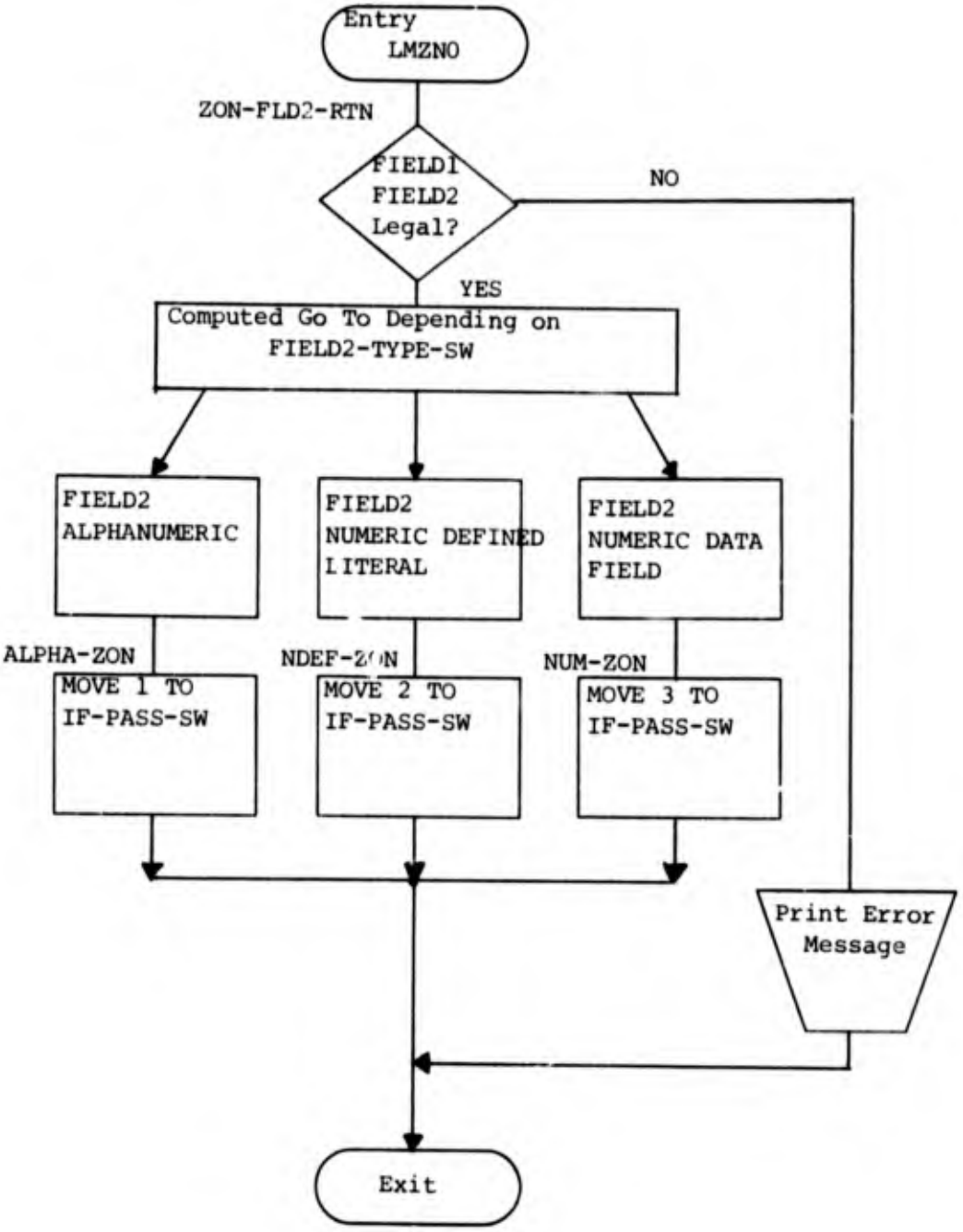
ALPHA-ZON.
A switch is set to indicate which program is needed to generate COBOL coding for an alphanumeric receiving field (IF-PASS-SW=1).

NDEF-ZON.
A switch is set to indicate which program is needed to generate COBOL coding for numeric defined literal as a receiving field (IF-PASS-SW=2).

NUM-ZON.
A switch is set to indicate which program is needed to generate COBOL coding for a numeric data field as a receiving field (IF-PASS-SW=3).

RKH-TEST-EXIT.
Returns logic flow to LMLPMOV which will call required program to generate COBOL coding dependent upon IF-PASS-SW.

(4) Limitations. None.


Entry
LMZNO
ZON-FLD2-RTN
FIELD1
FIELD2
Legal?
NO
YES
Computed Go To Depending on
FIELD2-TYPE-SW
FIELD2
ALPHANUMERIC
FIELD2
NUMERIC DEFINED
LITERAL
FIELD2
NUMERIC DATA
FIELD
ALPHA-ZON
NDEF-ZON
NUM-ZON
MOVE 1 TO
IF-PASS-SW
MOVE 2 TO
IF-PASS-SW
MOVE 3 TO
IF-PASS-SW
Print Error
Message
Exit

ar. LMLPZ.1

(1) Function. This program generates COBOL code for the MOVE ZON (ZONES) instruction when the receiving field is alphanumeric.

(2) Calling Sequence.

```
ENTRY   'LMZN1'  USING FM-DD  LP-DD  LM-DD
CALL    'LMPRO'  USING COBOL-FORMAT1,2,3
CALL    'FMPRT'  USING PRNT-CARDS  CC
CALL    'LMDAT'  USING FM-CARD
```

(3) Program Description. Entry is made through a call from LMLPMOV. In paragraph ALPHA-ZON a Computed Go To determines which paragraph the logic flow will go to depending on field1 type. See the 'MOVE ZON TABLE' for paragraph names which actually generate the code. Some paragraphs are broken into segments. In these cases an alphabetic character is attached to the paragraph number (i.e. LMR-3A). These breakdowns are made to take advantage of commonly generated statements which may be generated by PERFORMing a paragraph rather than duplicating it.

PARTIAL-IN-FLD2.
Produces Procedure Division coding for the second half of two lines of coding needed for a partial in field2.

DATA-GENERATION-1.
Generates a Data Division statement to define an alpahnumeric literal.

DG-REDEFINES-1.
Generates a Data Division statement to redefine the alphanumeric literal as a signed numeric literal.

DATA-GENERATION-2.
Same as DATA-GENERATION-1.

DG-REDEFINES-2.
Same as DG-REDEFINES-1.

DG-SINGLE-1.
Generates a Data Division statement to define a signed numeric literal.

DG-SINGLE-2.
Same as DG-SINGLE-1.

PERIOD-GEN-1 thru PERIOD-GEN-1-EXIT.
Places a period at the end of a variable length field name when that field name is the end of a COBOL statement requiring a period for field1.

PERIOD-GEN-2 thru PERIOD-GEN-2-EXIT.
Same as above except it is used for field2.

COBOL-OUT1.
COBOL-OUT2.
COBOL-OUT3.
Generates the printed COBOL generated output line.

CK-PARTIAL-FLD2.

ALPHA-ZON.

A Computed Go To is used to determine the coding which will be generated dependent upon field type of field1 and an alphanumeric receiving field. Contains program entry point.

LZR-3 thru LZR-12C

Generate COBOL coding for the various combinations of field1 and 2 types. See table.

RKH-TEST-EXIT.

Returns logic flow to LMLPMOV.

MOVE ZON TABLE

This table shows the paragraph names which generate COBOL coding for the MOVE instruction format:

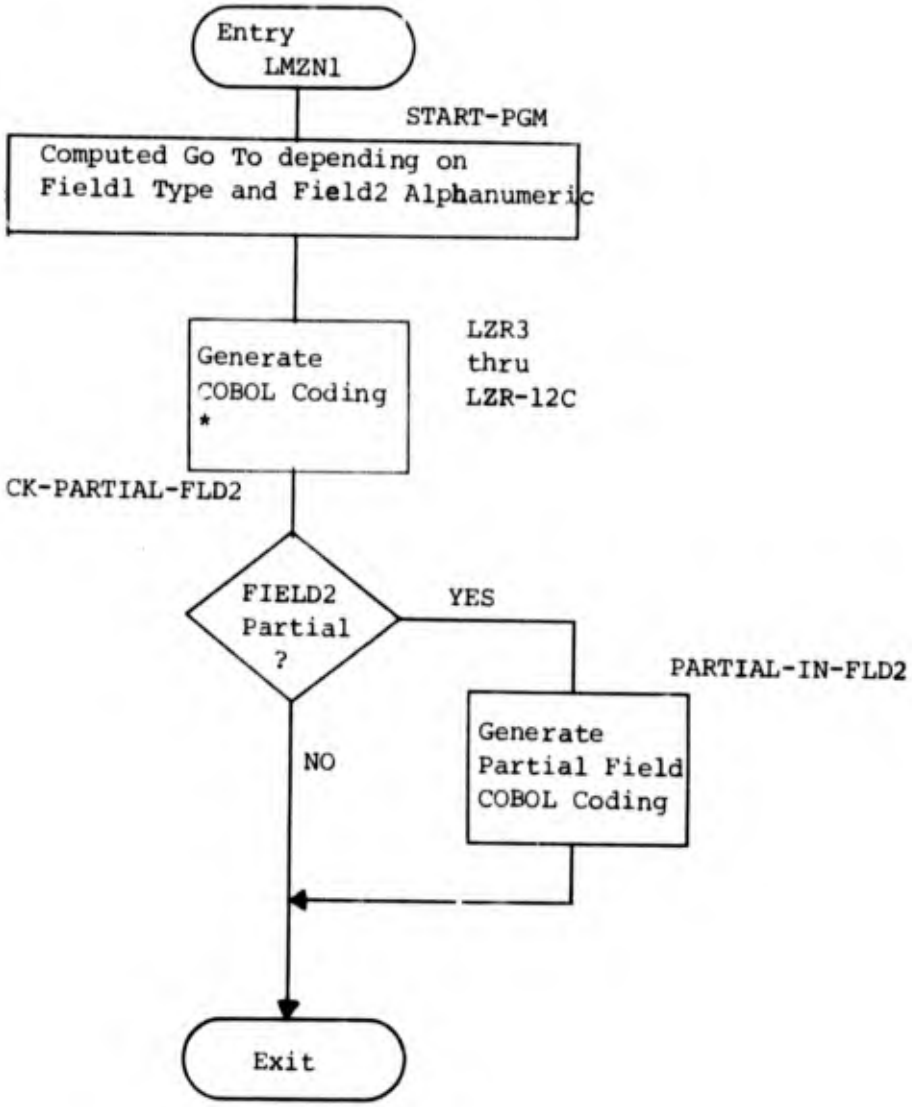
{MOVE / PUT} {ZON / ZONES} field1 {TO / INTO} field2

FIELD2

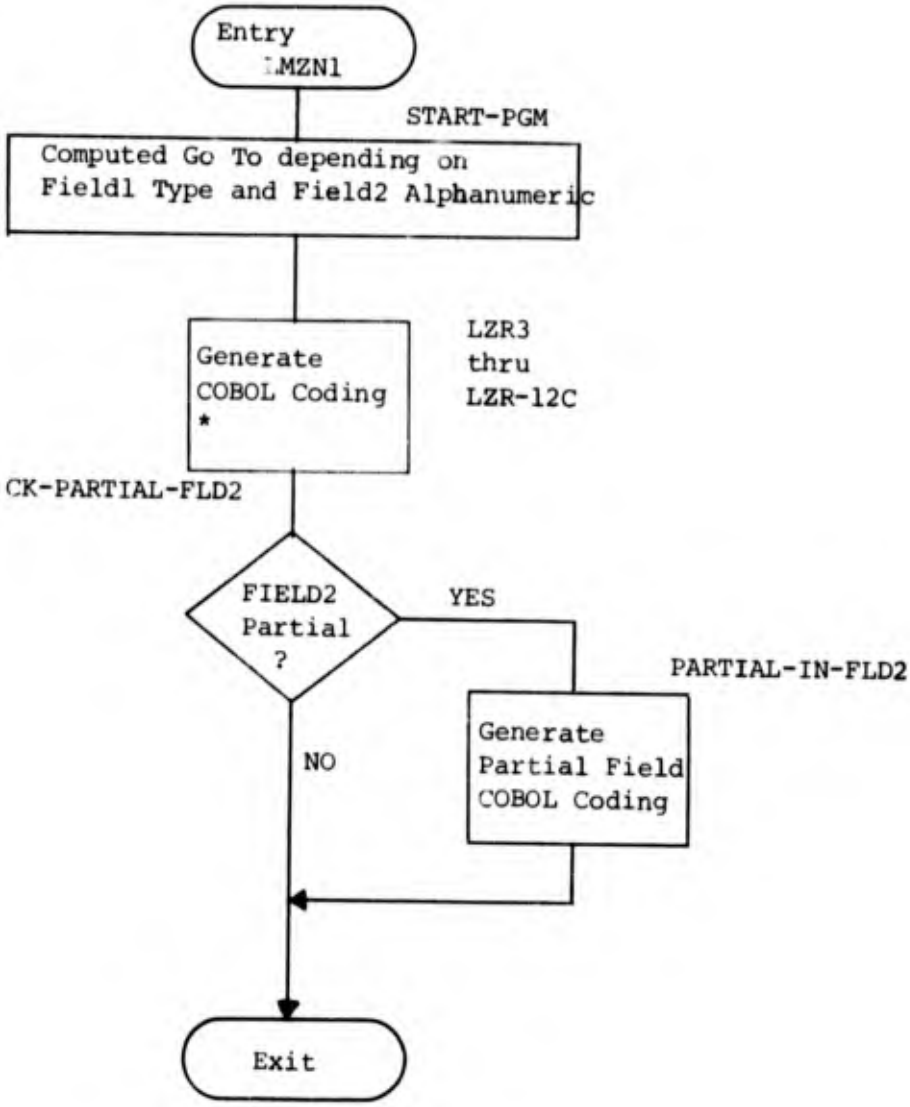
FIELD1	A	L	ND	NL	N
A	LZR-3 THRU LZR-3F	ERROR	LZR-8 THRU LZR-8B	ERROR	LZR-7 THUR LZR-7B
L	ERROR	ERROR	ERROR	ERROR	ERROR
ND	LZR-12 THRU LZR-12C	ERROR	LZR-14	ERROR	LZR-13 THRU LZR-13B
NL	LZR-4 THRU LZR-4D	ERROR	LZR-6 THRU LZR-6A	ERROR	LZR-5
N	LZR-10 THRU LZR-10D	ERROR	LZR-11 THRU LZR-11A	ERROR	LZR-9 THRU LZR-9B

NOTE: This table is derived from a combination of paragraph names from three different programs--LMLPZN1, LMLPZN2, and LMLPZN3.

(4) Limitations. None.



* SEE MOVE ZON TABLE ON PREVIOUS PAGE.



* SEE MOVE ZON TABLE ON PREVIOUS PAGE.

as. LMLPZN2

(1) Function. This program generates COBOL code for the MOVE ZON (ZONES) instruction when the receiving field is a numeric defined literal.

(2) Calling Sequence.

```
ENTRY  'LMZN2'  USING FM-DD  LP-DD  LM-DD
CALL   'LMPRO'  USING COBOL-FORMAT1,2,3
CALL   'FMPRT'  USING PRNT-CARDS  CC
CALL   'LMDAT'  USING FM-CARD
```

(3) Program Description. Entry is made through a call from LMLPMOV. In paragraph NDEF-ZON a Computed Go To determines which paragraph the logic flow will go to depending on fieldl type. See the 'MOVE ZONE TABLE' for paragraph names which actually generate the code. Some paragraphs are broken into segments. In these cases an alphabetic character is attached to the paragraph number (i.e. LMR-3A). These breakdowns are made to take advantage of commonly generated statements which may be generated by PERFORMing a paragraph rather than duplicating it.

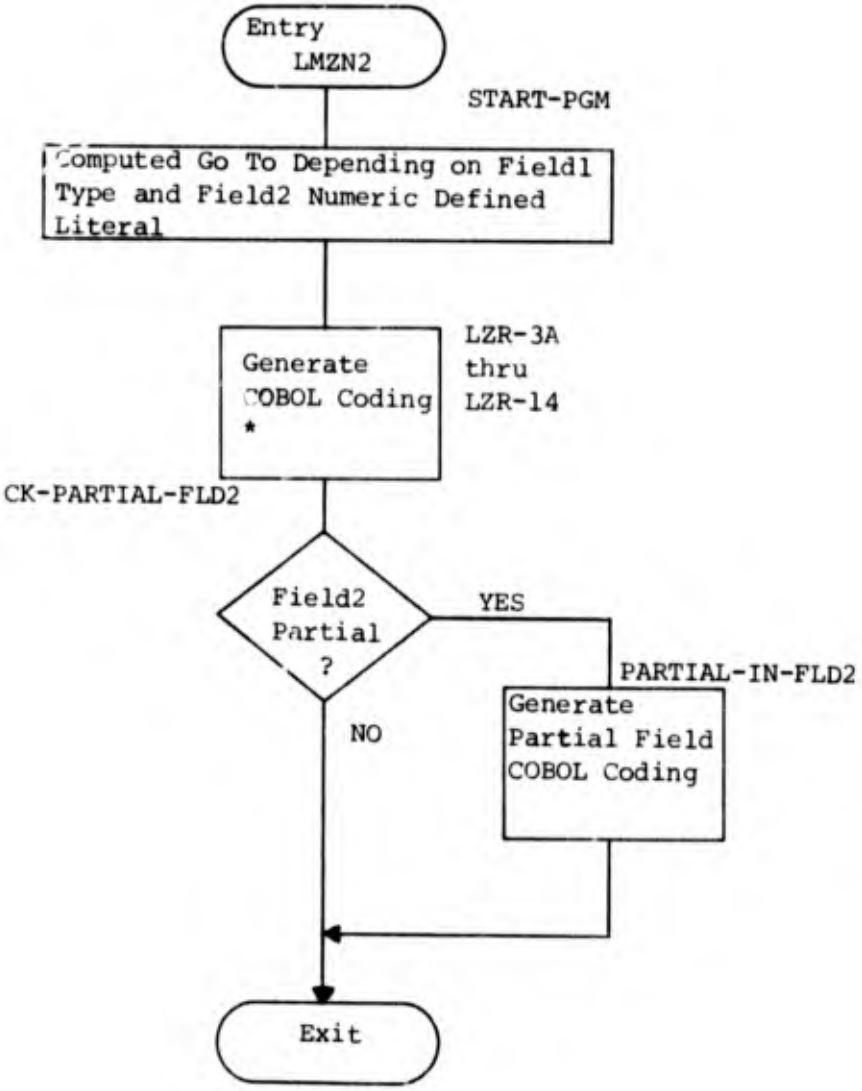
PARTIAL-IN-FLD2.
DATA-GENERATION-1.
DG-REDEFINES-1.
DATA-GENERATION-2.
DG-REDEFINES-2.
DG-SINGLE-1.
DG-SINGLE-2.
PERIOD-GEN-1 thru PERIOD-GEN-1-EXIT.
PEIROD-GEN-2 thru PERIOD-GEN-2-EXIT.
COBOL-OUT1.
COBOL-OUT2.
COBOL-OUT3.
All above paragraphs perform the same functions as in LMLPZN1.

CK-PARTIAL-FLD2.
Determines if field2 is a partial and channels the logic flow to paragraph PARTIAL-IN-FLD2 if a partial is found.

NDEF-ZON.
A Computed Go To is used to determine the coding which will be generated dependent upon field type of fieldl and a numeric defined literal as a receiving field.

LZR-3A thru LZR-14.
Generates COBOL coding for the various combinations of fieldl and 2 types. See MOVE ZON TABLE in LMLPZN1.

(4) Limitations. None.



* SEE MOVE ZON TABLE UNDER LMLPZN1.

at. LMLPZN3

(1) Function. This program generates COBOL code for the MOVE ZON (ZONES) instruction when the receiving field is a numeric data field.

(2) Calling Sequence.

```
ENTRY  'LMZN3' USING FM-DD  LP-DD  LM-DD
CALL   'LMPRO' USING COBOL-FORMAT1,2,3
CALL   'FMPRT' USING PRNT-CARDS  CC
CALL   'LMDAT' USING FM-CARDS
```

(3) Program Description. Entry is made through a call from LMLPMOV. In paragraph NUM-ZON a Computed Go To determines which paragraph the logic flow will go to depending on field1 type. See the 'MOVE TABLE' for paragraph names which actually generate the code. Some paragraphs are broken into segments. In these cases an alphabetic character is attached to the paragraph number (i.e., LMR-3A). These breakdowns are made to take advantage of commonly generated statements which may be generated by PERFORMing a paragraph rather than duplicating it.

PARTIAL-IN-FLD2.
DATA-GENERATION-1.
DG-REDEFINES-1.
DATA-GENERATION-2.
DG-REDEFINES-2.
DG-SINGLE-1.
DG-SINGLE-2.
PERIOD-GEN-1 thru PERIOD-GEN-1-EXIT.
PERIOD-GEN-2 thru PERIOD-GEN-2-EXIT.
COBOL-OUT1.
COBOL-OUT2.
COBOL-OUT3.
CK-PARTIAL-FLD2.

All above paragraphs perform the same functions as in LMLPZN1.

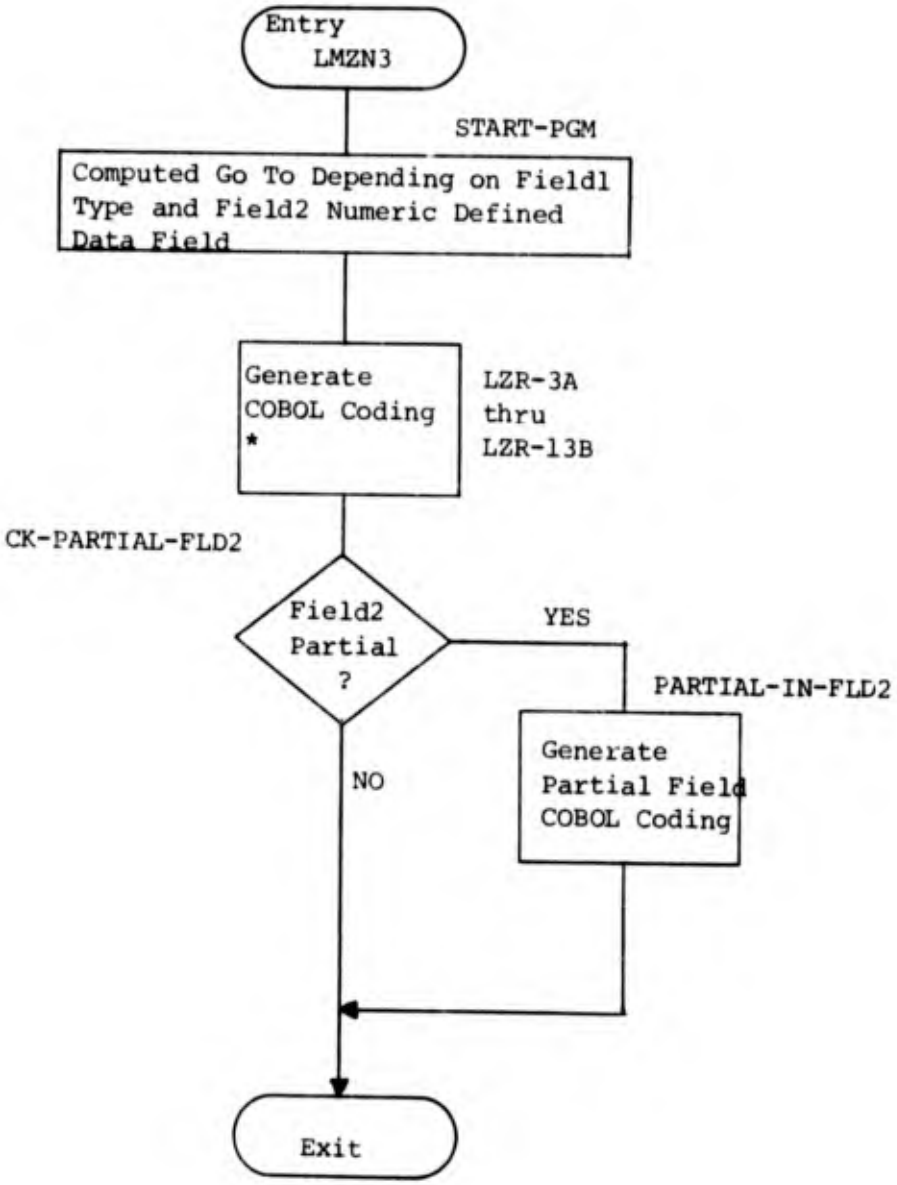
NUM-ZON.

A Computed Go To is used to determine the coding which will be generated dependent upon field type of field1 and a numeric data field as a receiving field.

LZR-3A thru LZR-13B.

Generate COBOL coding for the various combinations of field1 and 2 types. See MOVE ZON TABLE in LMLPZN1.

(4) Limitations. None.



* SEE MOVE ZON TABLE UNDER LMLPZN1.

3. LM ERROR MESSAGES.

a. ASSIGN INDEX TO PSETS ONLY

(1) LMLPASN: GENERATE-ASSIGN.

(2) An index may only be assigned to periodic sets for use as a pointer. The entry must be designated in the form "PSETnn," where nn is the desired periodic set number to be associated with the index number. Ninety-nine indexes are available.

b. AT-SIGN MISSING

(1) LMLPDEF: BUILD-LITERAL.

(2) An '@' or quote has been detected as part of a defined literal but no corresponding opending or closing sign is present. @ or quotes may be used interchangeably. A space following an @ or quote is considered a termination sign and any characters following a termination sign and any characters following the space will be ignored. Consequently, the @ or quote must be in or before card column 79. @ signs or quotes are required for continuation of defined literals. Do not mix @ signs and quotes.

c. CHARACTER NOT NUMERIC

(1) LMLPRMS: MOVE-M, MOVE-M1.

(2) Characters in range values are not numeric. Characters may only be numeric digits except for the hyphen between the two values.

d. COMP NUMERIC DEFINE ILLEGAL

(1) LMLPATC: GEN-ATCH-TEST.

(2) An LM constant which is defined as computational (signed binary) on a DEFINE card with a C cannot be used in the first operand on the ATTACH card.

e. COMPUTATIONAL NOT ALLOWED IN SFL

(1) LMLPDEF: SFL-COMP.

(2) "C" on DEFINE card is valid only in file-to-file LM. Computational (signed binary) is not allowed in single file logic packages.

f. CONNECTOR -BY- REQUIRED

(1) LMLPMLT: ERROR-OUT2.

(2) Keyword -BY- is missing, misplaced, or misspelled on MULTIPLY card. "BY" must follow first operand.

g. CONNECTOR -INTO- REQUIRED

(1) LMLPMLT: ERROR-OUT4.

(2) Keyword -INTO- is missing, misplaced, or misspelled on DIVIDE card. "INTO" must follow first operand

h. CONNECTOR MUST BE -FROM-

(1) LMLPSUB: ERROR-OUT4.

(2) The word following the first operand on the SUBTRACT card must be "FROM."

i. CONNECTOR MUST BE -TO-

(1) LMLPSUB: ERROR-OUT2.

(2) The word following the first operand on the ADD card must be "TO."

j. CONNECTOR MUST BE -TO- OR -INTO-

(1) LMLPMOV: LM-MOVE-ROUTE, NUM-AND-ZON-MOVE-RTN

(2) The connnector between fields on a MOVE card is required and must be either the word "TO" or "INTO." Check for misplaced, misspelled or missing words.

k. DATA DIVISION FOR FFT NOT ON LIBRARY

(1) LMLPDDS: CALL-LIB.

(2) FFT referenced on FMLP card is not on library. Restructure FFT under FS or check spelling on card.

l. DATA NOT ALLOWED AFTER DELETE FIELD

(1) LMLPCNG: LM-DELETE-ROUTINE.

(2) Extraneous data has been detected following the field which is to be deleted. Check for an illegal partial field format.

m. DEFINE CANNOT BE INDEXED

(1) LMLPFLD: DEFINE-LOOKUP.

(2) A defined value cannot be indexed. Remove index from card.

n. DEFINE IS EQUAL TO MAJOR FIELD NAME

(1) LMLPDEF: DEF-MAJOR.

(2) Define name is the same as a major file field name. Change define name.

o. DEFINE IS EQUAL TO MINOR FIELD NAME

(1) LMLPDEF: DEF-MINOR.

(2) Define name is the same as a minor file field name. Change define name.

p. DEFINE NAME TOO LARGE

(1) LMLPDEF: TEST-NO-7.

(2) Define name over six characters. Shorten name to 1-6 characters long; the first character alphabetic.

q. DEFINE NOT ALLOWED HERE

(1) LMLPSCN: IS-IT-DEFINE.

(2) All defines must be placed at the start of the logic package. Once a nondefine statement is used, no more defines are acceptable.

r. DEFINE PREVIOUSLY DEFINED

(1) LMLPDEF, DEF-TAB-LU, LMLPGRP: DEF-TAB-LU.

(2) Define name has already been used for a previous define. Change name of define. If continuation of a define is desired, a card with the same name must immediately follow the first.

s. DEFINE TYPE NOT NUMERIC

(1) LMLPRLD: CK-DEF.

(2) The periodic subset sequence number defined area is a nonnumeric defined field. Use a numeric file field or a numeric constant not to exceed three digits.

t. DEFINE VALUE MISSING

(1) LMLPDEF: FWS-PFRM.

(2) No value statement follows define name. Insure there is at least one space between the name and value.

u. DEFINE VALUE TOO LARGE

(1) LMLPDEF: GEN-CONTINUE.

(2) Value of define exceeds 132 character limit. Reduce size of DEFINE.

v. DEFINE VALUE TABLE EXCEEDED

(1) LMLPDEF: GEN-VALUE-START, MOVE-POUND-VAL.

(2) Single file defined storage area maximum of 5000 characters exceeded. Reduce the number of defines.

w. DIVISION OVERFLOW ERROR

(1) LMLPMLT: MULT-EXIT.

(2) First operand on DIVIDE card is equal to zero; cannot divide by zero. Insure that first operand is not zero.

x. DST, DSD, OR LOG REQUIRED

(1) FMLPVAL: FMLP.

(2) Keyword missing, misplaced, or misspelled on FMLP card. Second word on card must be DST, DSD, or LOG depending on type run desired.

y. EXPECTED -TO-

(1) LMLPATC: GEN-ATCH-2.

(2) The connector "TO" is required on the ATTACH card after the data to be attached is indicated.

z. EXPECTED -.-

(1) LMLPDEL: GENERATE-DELETE-S.

(2) A period is required prior to the failure exit. Check for extraneous words in the statement.

aa. EXTRANEOUS WORD ON STOP CARD

(1) LMLPRST: GENERATE-STOP.

(2) Word other than SFL is on stop card. Remove extra word.

ab. FAILURE EXIT EXCEEDS 6 CHARACTERS

(1) LMLPSCN: SKIP-PARA1.

(2) Failure exit must be from 1-6 characters in length, with the first character being alphabetic. No special characters are allowed.

ac. FAILURE EXIT OMITTED ON PREVIOUS CARD

(1) LMLPSCN: OPERATION-LU1.

(2) Failure exit must be present on last card of a consecutive series of relationship test cards. If only one card, then failure exit must be on it.

ad. FIELD CANNOT BE NUMERIC

(1) LMLPRST: GEN-READ-CARD.

(2) The receiving area for the READ CARD/TAPE must be an alphanumeric defined area large enough to hold the largest value to be read.

ae. FIELD MUST BE IN THE FORM MMMM-NNNN

(1) LMLPVPT: VPRINT-PROCESS.

(2) Partial field notation of the first operand is illegal.

af. FIELD MUST BE VARIABLE

(1) LMLPVPT: VPRINT-PROCESS.

(2) Only a variable set as defined in the FFT may be printed using VPRINT.

ag. FIELD NOT NUMERIC

(1) LMLPMLT: ERR2.

(2) First and/or second operand on MULTIPLY or DIVIDE card is not a numerically defined field or field name.

ah. FIRST CHARACTER MUST BE ALPHABETIC

(1) LMLPDEF: GENERATE-DEF-TAB.

(2) First character of a defined constant name must be alphabetic.

ai. FIRST FIELD NOT NUMERIC

(1) LMLPSUB: ERROR-OUT3.

(2) The first operand of an ADD or SUBTRACT operation is not numeric. Check to see if the field in question has been defined as numeric. Defined literals which have a value in @ signs or quotes are considered alphanumeric even if the value contains all numeric digits.

aj. FIRST WORD MUST BE *DEFINE*

(1) LMLPGRP: CHECK-NAME.

(2) Only DEFINE statements are allowed following a GROUP DEFINE until an ENDGRP statement is encountered.

ak. GROUP DEFINE EQUAL TO MAJOR FIELD NAME

(1) LMLPGRP: DEF-MAJOR.

(2) The name chosen for the define is already used as major field name in the major FFT. Choose a unique and different DEFINE name.

al. GROUP DEFINE EQUAL TO MINOR FIELD NAME

(1) LMLPGRP: DEF-MINOR.

(2) The name chosen for the define is already used as a minor field name in the minor FFT. Choose a unique and different DEFINE name.

am. ILLEGAL CARD FORMAT

(1) LMLPCNG: LM-CHANGE-ROUTINE.

(2) An excessive number of words are on the CHANGE SWITCH card. Check for imbedded blanks in the switch name.

an. ILLEGAL CHAR IN GROUP NAME

(1) LMLPGRP: VALIDATE-FIELD-NAME.

(2) The GROUP DEFINE name must have an alphabetic first character with the remaining characters (2-6) either numeric digits or alphabetic (or combination of both). No special characters allowed.

ao. ILLEGAL CHARACTER IN FIRST FIELD

(1) LMLPFMT: ERROR-OUT7, LMLPFMT1: ERROR-OUT7.

(2) First character of the first field must be a $ (for $$ORC, $$NRC, $$YMD, or a minor field), =, +, or numeric digit (for a numeric literal), or alphabetic (for special literal or major field). Field names must be alphabetic or numeric.

ap. ILLEGAL CHARACTER IS SET INDEX

(1) LMLPDEL: GENERATE-DELETE-S.

(2) After the subset number, the characters "&X" or "+X" are required on the DELETE card.

aq. ILLEGAL CHARACTER IN SPACE PARAMETER

(1) LMLPGRP: VALIDATE-SPACE.

(2) The size entry on a DEFINE GROUP card following the pound sign must be a number designating the group size. No spaces are allowed between the pound sign and the first numeric digit.

ar. ILLEGAL CONTINUATION

(1) LMLPDEF: FINISH-CARD.

(2) Literals which cannot be placed on only one define card may be continued on the immediately following card by finishing the literal with an @ sign or quote on the first card, repunching DEFINE and the LM defined constant name, and specifying another literal where the previous one terminated.

as. ILLEGAL DEFINE NAME

(1) LMLPDEF: DEF-MINOR.

(2) Define name contains special characters or does not have an alphabetic character in the first position. Change name so characters are alphabetic and numeric.

at. ILLEGAL DEFINE OPERATION

(1) LMLPDEF: CHAR-CHECK.

(2) All defines must be placed at the start of the logic package. Once a non-define statement is used, no more defines are acceptable.

au. ILLEGAL DEFINE VALUE

(1) LMLPDEF: BLDLIT-PFRM.

(2) Define value cannot extend into column 80 of card. At-sign or quote cannot extend beyond card column 79.

av. ILLEGAL FIELD FOR RANGE CARD

(1) LMLPRFD: PARTIAL-TYPE.

(2) First operand of RANGE test must be a numeric file field, system field, constant, or literal.

aw. ILLEGAL FIELD SPECIFICATIONS

(1) LMLPNM0: ERROR-OUT10.
LMLPNM1: ERROR-OUT10.
LMLPNM2: ERROR-OUT10.
LMLPNM3: ERROR-OUT10.
LMLPPUT: ERROR-OUT10.
LMLPRT1: FIELD-ERROR.
LMLPRT2: FIELD-ERROR.
LMLPSUB: ERROR-OUT10.
LMLPZN0: ERROR-OUT10.
LMLPZN1: ERROR-OUT10.
LMLPZN2: ERROR-OUT10.
LMLPZN3: ERROR-OUT10.

(2) The field type combinations are not compatible.

ax. ILLEGAL FIELD TYPE COMBINATION

(1) FM-RELATION-CHECK, LMLPRLN, LMLPMLT: ERROR-OUT3.

(2) The field types of the two values being compared in the relationship test are not compatible. Also, applies to MULTIPLY or DIVIDE instructions.

ay. ILLEGAL FIELD TYPE FOR BIT OPERATION

(1) LMLPSPO: RETURN1.

(2) Numeric or special literals cannot be used for the bit test. First field must be a file field, constant, or literal.

az. ILLEGAL FIELD TYPE FOR SWITCH CARD

(1) LMLPSPO: SWITCH-TYPE-SET.

(2) Switch name must take the form of SWITCHn where "n" is any alphabetic or numeric character.

ba. ILLEGAL FILE FOR RELOAD

(1) LMLPRST: GENERATE-RELOAD.

(2) Only the major file may be reloaded.

bb. ILLEGAL FIRST FIELD

(1) LMLPCNG: ERROR-OUT8.
LMLPNMO: ERROR-OUT8.
LMLPSUB: ERROR-OUT8.
LMLPZNO: ERORR-OUT8.
LMLPZN1: ERROR-OUT8.
LMLPZN2: ERROR-OUT8.
LMLPZN3: ERROR-OUT8.

(2) The first operand field type is not legal for the statement operation.

bc. ILLEGAL FORMAT ON VPRINT CARD

(1) LMLPVPT: VPRINT-PROCESS.

(2) VPRINT must have as a minimum, the symbolic name of the variable set to be printed and partial notation.

bd. ILLEGAL INDEX

(1) LMLPASN: GENERATE-ASSIGN.

(2) An index number must be two digit numeric characters greater than zero.

be. ILLEGAL LABEL

(1) LMLPSCN: LAB-ERROR, LU-LAB-TAB.

(2) A label must be from 1-6 characters in length, with the first character being alphabetic. No special characters are allowed.

bf. ILLEGAL LINE SKIP VALUE

(1) LMLPCNG: EJECT-LINES.

(2) Skip value must be 1-9.

bg. ILLEGAL NUMERIC DATA

(1) LMLPDEF: FIN-NUMERIC.

(2) Non-numeric data in a computational numeric define value.

bh. ILLEGAL PARTIAL FIELD NOTATION

(1) LMLPGRP: VS-PFRM3.

(2) Partial notation must have all numeric ranges separated by a hyphen with the first range less than or equal to the second range.

bi. ILLEGAL PARTIAL FIELD SPECIFICATION

(1) LMLPFLD: MOVE-NUMBER-1, STOP-ZERO-2, CHECK-PARTIALS.

(2) Partial notation must have all numeric ranges separated by a hyphen with the first range less than or equal to the second range.

bj. ILLEGAL PROFILE CHARACTER

(1) LMLPSP1: ADD-1-HM.

(2) Profile mask characters must be one of the following: A, B, C, E, F, K, L, M, N, S, T, U, V, X, Z.

bk. ILLEGAL RECEIVING FIELD

(1) LMLPFMT: ERROR-OUT1.
LMLPMLT: ERR1.
LMLPMOV: TEST-FLD2-RTN.
LMLPNMO: ERROR-OUT1.
LMLPSUB: ERROR-OUT1.
LMLPZNO: ERROR-OUT1.
LMLPZN1: ERROR-OUT1.
LMLPZN2: ERROR-OUT1.
LMLPZN3: ERROR-OUT1.

(2) The PSET or VSET number must be a two digit numeric number greater than zero.

bl. ILLEGAL RUN SPECIAL---CODE IGNORED

(1) FMLPVAL: CHK-RUN-SPCL.

(2) Illegal run type; check FMLP card.

bm. ILLEGAL RUN TYPE

(1) FMLPVAL: CHK-5TH-WRD.

(2) Only the characters "0", "S", or "F" can follow the major file name.

bn. ILLEGAL SET NUMBER

(1) LMLPASN: GENERATE-ASSIGN.
LMLPATC: GEN-ATCH-1.
LMLPDEL: GENERATE-DELETE-S,
GENERATE-DELETE-P,
GENERATE-DELETE-V.

(2) The PSET or VSET number must be a two digit numeric number greater than zero.

bo. ILLEGAL SUBROUTINE NAME

(1) LMLPRST: GC-MOVE-FILE-NAME.

(2) Subroutine name must be five characters long and end in "S."

bp. ILLEGAL SWITCH NAME

(1) LMLPCNG: ERROR-OUT11.

(2) Switch name must take the form of "SWITCHn" where "n" is any alphabetic or numeric character.

bq. ILLEGAL TO CONVERT INTO MINOR FILE FIELD

(1) LMLPRST: GC-FLD2.

(2) Receiving field can only be a major file field or a constant.

br. ILLEGAL TO CONVERT INTO THIS FIELD

(1) LMLPRST: GC-FLD2.

(2) A special literal cannot be used as a receiving field for converted data.

bs. ILLEGAL VALUE FOR BUILD

(1) LMLPBLD: CK-DEF.

(2) The periodic subset sequence number is not a numeric file field or numeric constant.

bt. ILLEGAL WORD FOLLOWING *READ*

(1) LMLPRST: GENERATE-READ.

(2) Only the word MAJOR, MINOR, CARD, or TAPE is allowed.

bu. ILLEGAL WORD FOR SWITCH OR BIT OPERATION

(1) LMLPSPO: DETERMINE-SWITCH-TYPE.

(2) One of the keyword parameters is incorrect.

bv. ILLEGAL WORDS ON SWITCH CARD

(1) LMLPSP1: ENTER1.

(2) Illegal combination or number of words on SWITCH card.

bw. INCORRECT INDEX NUMBER

(1) LMLPBLD: GENERATE-BUILD.

(2) Index number must be in the form "Xnn," where nn is a two digit number greater than zero assigned to the periodic set.

bx. INCORRECT NUMBER OF WORDS

(1) LMLPATC: GEN-ATCH-2.

(2) Insure that failure exit on ATTACH card is immediately following the period on the card. No space allowed.

by. INCORRECT WORD COUNT

(1) LMLPASN
LMLPBLD
LMLPDEL: GENERATE-DELETE-S,
GENERATE-DELETE-P,
GENERATE-DELETE-V.
LMLPSPO: SWITCH-TYPE.
LMLPSTE: GENERATE-STEP.

(2) An incorrect card format has been detected for this instruction.

bz. INDEX IS ILLEGAL

(1) LMLPDEL: GENERATE-DELETE-S.

(2) Periodic or variable set index number must be numeric and greater than zero. Must be two digits in length.

ca. INDEX MUST BE 2 DIGITS AND NUMERIC

(1) LMLPSTE: GENERATE-STEP.

(2) Index number on STEP card must be numeric and greater than zero. Must be two digits in length.

cb. INDEX NOT PREVIOUSLY ASSIGNED

(1) LMLPFI): CIA-PFRM.

(2) All periodic field references must include an index designation to properly access a subset; this index must be previously assigned to tell the system what subset is of interest.

cc. INDEX-SET NEVER ASSIGNED FOR A BUILD

(1) LMLPWP0: BLD-LOOK-UP.

(2) The user is attempting to build a periodic subset without first assigning an index number. Assign an index number prior to executing this instruction.

cd. INDEX-SET NEVER ASSIGNED FOR A DELETE

(1) LMLPWP0: DELETE-LOOKUP.

(2) The user is attempting to build a periodic subset without first assigning an index number. Assign an index number prior to executing this instruction.

ce. INVALID COMBINATION OF RELATIONAL WORDS

(1) LMLPRLN: SET-REL-ERROR.

(2) Only one operator and/or one negative operator is allowed per logic connector.

cf. INVALID DST OR LOG NAME

(1) FMLPVAL: SCAN-3RD-WORD.

(2) Logic package name must be five characters long, ending in "P," and be unique to the site.

cg. INVALID EOF TYPE

(1) LMLPRMS: INVALID-EOF.

(2) For Single File Logic, only a READER EOF may be specified; for file-to-file, TAPE, MAJOR, or MINOR may additionally be used.

cb. INDEX NOT PREVIOUSLY ASSIGNED

(1) LMLPFLD: CIA-PFRM.

(2) All periodic field references must include an index designation to properly access a subset; this index must be previously assigned to tell the system what subset is of interest.

cc. INDEX-SET NEVER ASSIGNED FOR A BUILD

(1) LMLPWP0: BLD-LOOK-UP.

(2) The user is attempting to build a periodic subset without first assigning an index number. Assign an index number prior to executing this instruction.

cd. INDEX-SET NEVER ASSIGNED FOR A DELETE

(1) LMLPWP0: DELETE-LOOKUP.

(2) The user is attempting to build a periodic subset without first assigning an index number. Assign an index number prior to executing this instruction.

ce. INVALID COMBINATION OF RELATIONAL WORDS

(1) LMLPRLN: SET-REL-ERROR.

(2) Only one operator and/or one negative operator is allowed per logic connector.

cf. INVALID DST OR LOG NAME

(1) FMLPVAL: SCAN-3RD-WORD.

(2) Logic package name must be five characters long, ending in "P," and be unique to the site.

cg. INVALID EOF TYPE

(1) LMLPRMS: INVALID-EOF.

(2) For Single File Logic, only a READER EOF may be specified; for file-to-file, TAPE, MAJOR, or MINOR may additionally be used.

ch. INVALID MAJOR FILE NAME

(1) FMLPVAL: CHK-4TH-WORD.

(2) Major file name must be five characters long with the first character alphabetic and ending in "A" or "T."

ci. INVALID MINOR FILE NAME

(1) FMLPVAL: CHK-RUN-TYPE.

(2) Minor file name must be five characters long with the first character alphabetic and ending in "A" or "T."

cj. INVALID OPERATION

(1) LMLP: OLU.

(2) No operation exists in the LM instruction set as indicated by the user statement.

ck. INVALID PROFILE

(1) LMLPSPO: RETURN2.

(2) Profile field size is less than the profile mask. Both fields must be equal.

cl. INVALID RELATIONAL WORD

(1) LMLPRLN: SCAN-RELATION.

(2) Invalid operator in relational test statement.

cm. INVALID SET ID

(1) LMLPFLD: IS-IT-MINOR-FIXED.

(2) A two digit number period set number greater than zero must be used.

cn. INVALID SIZE

(1) LMLPDEF: POUND-SIZE.

(2) The size of a defined area has not been indicated. Only the pound sign was detected. Check for space between pound sign and value. No space is allowed.

co. INVALID WORD COUNT

(1) LMLPSP0: IF-ERROR.
LMLPSP1: IF-ERROR.

(2) An incorrect card format has been detected for this instruction.

cp. LABEL NOT DEFINED

(1) LMLPWT0: PERFORM-FAIL-LU.

(2) A failure exit label has not been defined. There is no statement or instruction to branch to.

cq. LABEL PREVIOUSLY USED

(1) LMLPSCN: LAB-TAB-LU.

(2) Label is not unique to the program. Change label and insure that all references to it are changed.

cr. LABEL TOO LARGE

(1) LMLPSCN: CHECK-7.

(2) A label must be from 1-6 characters in length with the first character being alphabetic.

cs. LITERAL EXCEEDS 132 CHARACTERS

(1) LMLPDEF: BUILD-LITERAL.

(2) A defined literal cannot exceed 132 characters; size of one print line.

ct. LITERAL HAS A LENGTH OF ZERO

(1) LMLPSCN: COUNT-LITERAL.

(2) A literal must be at least one character long.

cu. LITERAL TOO LARGE

(1) LMLPSCN: COUNT-LITERAL.

(2) Literal value cannot extend into card column 80. At-sign or quote cannot extend beyond card column 79.

cv. LOG NAME MUST END IN P

(1) FMLPVAL: SCAN-3RD-WORD.

(2) Logic package name must end in "P," be five characters long, and be unique to the site.

cw. MAJOR FFT NOT ON LIBRARY

(1) FMLPVAL: CHK-OP-CODE1.

(2) The major FFT does not reside on the library. A file structuring run is necessary to place the FFT on the library.

cx. MAJOR FIELD TYPE NOT NUMERIC

(1) LMLPBLD: CK-MAJ.

(2) The periodic subset sequence number must be either a numeric file field or numeric constant.

cy. MAJOR PSET NOT DEFINED

(1) LMLPASN: GENERATE-ASSIGN.

(2) No periodic set has been defined for the major PSET designated.

cz. MAJOR VSET NOT DEFINED

(1) LMLPASN: GENERATE-ASSIGN.

(2) No periodic set has been defined for the major VSET designated.

da. MAXIMUM DELETES EXCEEDED (25)

(1) LMLPDEL: GENERATE-DELETE-S.

(2) The maximum number of deletes allowed for a logic package is 25.

db. MINOR FFT NOT ON LIBRARY

(1) FMLPVAL: CHK-OP-CODE2.

(2) The Minor FFT does not reside on the library. A File Structuring run is necessary to place the FFT on the library.

dc. MINOR FIELD CANNOT RECEIVE DATA

(1) LMLPMLT: ERROR-OUT1.

(2) Minor file fields cannot be updated, only major file fields.

dd. MINOR FIELD TYPE NOT NUMERIC

(1) LMLPBLD: GEN-BLD2.

(2) The periodic subset sequence number must be either a numeric file field or numeric constant.

de. MINOR PSET NOT DEFINED

(1) LMLPASN: GENERATE-ASSIGN.

(2) No periodic set has been defined for the minor PSET designated.

df. MINOR VSET NOT DEFINED

(1) LMLPASN: GENERATE-ASSIGN.

(2) No periodic set has been defined for the minor VSET designated.

dg. MMMM MUST BE SMALLER THAN NNNN

(1) LMLPVPT: VERIFY-VSET-PARTIAL.

(2) When utilizing partial notation on the VPRINT card, the first numeric must be smaller than the second numeric.

dh. NAME FOR GROUP DEFINE MISSING

(1) LMLPGRP: CHECK-NAME.

(2) Following the word "GROUP," a unique name must be designated for the GROUP.

di. NUMBER CAN CONTAIN A MAXIMUM OF 18 DIGITS

(1) LMLPRFD: IF-NUM-LIT.

(2) A numeric literal is a series of not more than 18 digits. A plus sign (or ampersand) or a minus sign may precede the numeric literal.

dj. NUMBER EXCEEDS MAXIMUM SIZE

(1) LMLPFMT: ERROR-OUT13.
LMLPFMT1: ERROR-OUT13.

(2) A numeric literal is a series of not more than 18 digits. A plus sign (or ampersand) or a minus sign may precede the numeric literal.

dk. NUMBER ILLEGAL

(1) LMLPATC: GENERATE-ATTACH.

(2) The field following the ATTACH operator may only be an LM constant which is not defined as computational (C), or an FFT defined mnemonic (RMRKS), including a mnemonic of another variable set.

dl. NUMBER CONTINUATION ILLEGAL

(1) LMLPDEF: IS-IT-ZERO.

(2) Numeric defines cannot be continued.

dm. NUMERIC FIELD CAN HAVE MAX OF 18 CHARS

(1) LMLPGRP: VS-PFRM3.

(2) COBOL limits numeric digit length to 18 digits. Reduce numeric literal length.

dn. NUMERIC FIELD CANNOT EXCEED 18 CHARS

(1) LMLPDEF: FINISH-CARD, MOVE-POUND-VAL.

(2) COBOL limits numeric fields to a maximum of 18 digits. Reduce size of numeric field to 18 digits not counting the sign if any.

do. OPERAND MISSING

(1) LMLPSCN: FWS-PFRM.

(2) An operand had been misspelled, misplaced, or is illegal and not acceptable in the MIDMS instruction set.

dp. PARTIAL EXCEEDS FIELD SIZE

(1) LMLPFLD: CONTINUE-PARTIAL.

(2) The indicated partial field notation is larger than the actual field size.

dq. PARTIAL FIELD NOT ALLOWED

(1) LMLPRST: GEN-READ-CARD.

(2) A partial field is not allowed on the receiving area for the card image on the READ CARD instruction.

dr. PERIOD IS REQUIRED

(1) LMLPSTE: GENERATE-STEP1.

(2) A period must preceed the failure exit. No space is allowed between the period and the failure exit.

ds. PERIODIC FIELD INDEX MISSING

(1) LMLPFLD: IS-IT-MAJOR-FIX, IS-IT-MINOR-FIX.

(2) All periodic type fields must have an index number with the format FIELD+Xnn where nn is greater than zero.

dt. PERIODIC FIELD IS NOT INDEXED

(1) LMLPFLD: BUILD-PSETNN.

(2) All periodic type fields must have an index number with the format FIELD+Xnn where nn is greater than zero.

du. POUND SIGN REQUIRED

(1) LMLPGRP: CHECK-FOR-POUND.

(2) The pound sign is not on the size entry of the GROUP card. The size entry must take the form of "#nnn," where "n" is a one, two, or three digit number designating the group size.

dv. PRINT STATEMENT EXCEEDS 132 CHAR

(1) LMLPPRT: GPR-PRODUCE-SETUPS.

(2) Line length cannot exceed 132 characters. Reduce size of line and use a new PRINT statement for each 132 character line.

ec. RANGE FIELD NOT NUMERIC

(1) LMLPRMS: MOVE-N.

(2) The field which is being tested for a RANGE value must be numeric and must be the same size as the range values.

ed. RANGE VALUE CANNOT EXCEED 37 CHARACTERS

(1) LMLPRMS: FM-IN-RANGE-PROCESS.

(2) Maximum size of range values are 18 digits for each number separated by a hyphen (18+1+18 = 37).

ee. READ CARD MUST BE INTO DEFINED AREA

(1) LMLPRST: GEN-READ-CARD.

(2) The receiving area for the card image must be a previously defined alphanumeric constant.

ef. READ REQUIRES FILE-TO-FILE OPERATION

(1) LMLPRST: GEN-READ-MAJOR-MINOR.

(2) All READ statements except READ CARD are only valid operations in file-to-file logic packages. FMLP LOG card must have an "F" in the logic package type field.

eg. READ TAPE REQUIRES PROGRAM MODE

(1) LMLPRST: GEN-READ-CARD.

(2) File-to-file logical Maintenance in the program mode is required for READ tape operations. On the FMLP control card specify run type as "F" and indicate program mode.

eh. RECEIVING FIELD MISSING

(1) LMLPRST: GEN-READ-CARD.

(2) An alphanumeric defined field large enough to hold the largest record to be read is missing.

ei. RECEIVING FIELD MUST BE A VSET

(1) LMLPATC: GEN-ATCH-2.

(2) Data to be attached can only be appended to the end of a variable set. The receiving variable set must be specified in the form "VSETnn," where nn is the number assigned to the variable set in the major file FFT.

ej. RECEIVING FIELD MUST BE NUMERIC

(1) LMLPMOV: TEST-FLD2-RTN.

(2) If the first operand is a special literal, the receiving operand cannot be numeric.

ek. RECEIVING FIELD NOT NUMERIC

(1) LMLPMOV: TEST-FLD2-RTN.
LMLPSUB: ERROR-OUT5.

(2) The second operand of an ADD or SUBTRACT operation must be either a numeric data field or numeric defined literal. In a MOVE operation, first field and receiving field are not compatible.

el. RELATION CONTAINS TOO MANY WORDS

(1) LMLPRLN: RESTORE-TEMP.

(2) There can be no more than six words (counting a label, whether present or not) per relational statement.

em. RELOAD REQUIRES FILE-TO-FILE RUN

(1) LMLPRST: GENERATE-RELOAD.

(2) The RELOAD MAJOR statement reinitializes the major file record as it was input and places it into the MP work area. This operator is available in file-to-file runs only (run type "F" on FMLP control card).

en. RUN TYPE MUST BE S OR F

(1) FMLPVAL: CHK-5TH-WORD.

(2) The run type entry may only have one of two forms: S and F.

eo. SAVE REQUIRES FILE-TO-FILE RUN

(1) LMLPRST: GENERATE-SAVE.

(2) SAVE is only used after a WRITE operation which is a valid operation only in file-to-file runs.

ep. SET DOES NOT EXIST

(1) LMLPATC: GEN-ATCH-1.
LMLPDEL: GENERATE-DELETE-P,
GENERATE-DELETE-S,
GENERATE-DELETE-V.

(2) The indicated VSET or PSET does not exist and has not been defined in the FFT. Restructure the FFT to include this set.

eq. SET NUMBER ILLEGAL

(1) LMLPATC: GEN-ATCH-2.

(2) The variable set index number must be a two digit number greater than zero.

er. SET NUMBER MUST BE 2 NUMERIC DIGITS

(1) LMLPBLD: GENERATE-BUILD.

(2) Periodic set number contains non-numeric digits or contains more than two numeric digits. Check for omission of a blank between the set number and index number.

es. SINGLE RANGE VALUE CANNOT EXCEED 18 CHARACTERS

(1) LMLPRMS: FM-IN-RANGE-PROCESS.

(2) The range value on either side of the hyphen cannot exceed 18 numeric digits due to a COBOL limitation.

et. SIZE NOT NUMERIC

(1) LMLPDEF: POUND-SIZE-ERROR.

(2) The size parameter of a DEFINE area contains non-numeric characters following the pound sign.

eu. SIZE TOO LARGE

(1) LMLPDEF: POUND-SIGN.

(2) Size value following pound sign cannot exceed four numeric digits.

ev. SPACE CANNOT FOLLOW SIGN

(1) LMLPDEF: CHECK-SIGN-SPACE.

(2) A signed numeric literal cannot have a space after the sign. Make sign and numerics contiguous.

ew. STEP INDEX NOT ASSIGNED

(1) LMLPWPO: STEP-LOOKUP.

(2) Assigning an index to a set must be done before attempting to step the index.

ex. STEP MUST BE NUMERIC VALUE LESS THAN 600

(1) LMLPSTE: STEP-ERR.

(2) Increment on STEP card cannot exceed 599.

ey. SWITCH MAY BE ONLY -ON- or -OFF-

(1) LMLPCNG: ERROR-OUT16.

(2) The switch condition indicator can only be set to ON or OFF.

ez. SYNTAX ERROR ON TABLE LOOKUP

(1) LMLPTAB: TABLE-ERROR2.

(2) IN TABLE operator card has a format error. Check documentation for proper format.

fa. TABLE LOOKUP PARTIAL FIELD NOT ALLOWED

(1) LMLPRMS: NOT-LM-ORD.

(2) Use of partial field notation in the first field and/or the receiving field is not allowed.

fb. TABLE NAME IN ERROR

(1) LMLPTAB: TABLE-ERROR.

(2) Incorrect table name. Table name must be a five character name ending in "L."

fc. TO MUST FOLLOW GO

(1) LMLPSCN: CARD-GO-TO.

(2) GO TO is the correct operator in the MIDMS instruction set. Anything else will be ignored and flagged as an error.

fd. TOO MANY LABELS

(1) LMLPSCN: LABEL-FAILURE-LU1, LAB-TAB-ADD.

(2) A maximum of 100 labels per logic package is allowed.

fe. TOO MANY LABELS REFERENCED

(1) LMLPSCN: LABEL-DEFINE-LU.

(2) There can only be 100 unique failure exits within a single logic package.

ff. TOTAL CONDITION MUST BEGIN WITH AN IF

(1) LMLPRMN: FM-AND, FM-OR.

(2) A conditional statement has been detected which does not start with an "IF." Check for an "OR" or "AND" statement which is not preceded by an "IF" statement.

fg. VARIABLE FIELD NOT ALLOWED

(1) LMLPFMT: FIELD-PERFORM.
LMLPFMT1: FIELD-PERFORM.

(2) The only operators that allow use of the variable set in their fields are: ATTACH, DELETE, and VPRINT (or PRINT for entire VSET).

fh. VPRINT REQUIRED TO PRINT VARIABLE SET

(1) LMLPPRT: GPR-OPERAND-CYCLE.

(2) In order to print out a partial listing of the variable set, the VPRINT instruction is necessary. Change PRINT to VPRINT.

fi. WORD -BY- IS REQUIRED

(1) LMLPSTE: GENERATE-STEP.

(2) The index number must be followed by the word "BY."

fj. WORD EXCEEDS CARD LIMITATION

(1) LMLPSCN: BUILD-OVERFLOW1.

(2) Only columns 1-78 are available. Except for an at-sign in column 79, data cannot be placed in 79-80.

fk. WORD *INDEX* MISSING

(1) LMLPASN: GENERATE-ASSIGN.

(2) The word INDEX must follow the ASSIGN operator.

fl. WORD MUST BE *INDEX*

(1) LMLPSTE: GENERATE-STEP.

(2) The word INDEX must follow the STEP operator.

fm. WORD MUST BE SPELLED -SWITCH-

(1) LMLPCNG: ERROR-OUT15.

(2) On the change switch card, the word switch must be spelled correctly.

fn. WORD NOT FIELD OR DEFINE

(1) LMLPFLD: MAJOR-LR2-LOOKUP, MINOR-LR2-LOOKUP.

(2) The field has not been defined in the logic package or in a major or minor FFT. If not spelling error, then FFT should be restructured or define the field in the logic package.

fo. WORD -SUBSET- IS REQUIRED

(1) LMLPBLD: GENERATE-BUILD.

(2) BUILD SUBSET is the correct operator. Insure that subset is after build and that it is spelled correctly.

fp. WORD -TO- MISSING

(1) LMLPASN: GENERATE-ASSIGN.

(2) The operand TO connector is required following the index number.

fq. WORD TOO LARGE

(1) LMLPFLD: CHECK-FIELD-DEFINITION.

(2) If not indexed the word cannot exceed six characters; if indexed word cannot exceed nine characters.

fr. WRITE REQUIRES FILE-TO-FILE RUN

(1) LMLPRST: GENERATE-WRITE.

(2) The WRITE operator is valid only in file-to-file maintenance. The logic package type must be "F" on the FMLP control card.

fs. XPRINT CARD OUT OF SEQUENCE

(1) LMLPSCN: PRT-TEST.

(2) An XPRINT card must be preceded by a PRINT card or another XPRINT card.

ft. ZERO LENGTH NOT VALID

(1) LMLPDEF: GEN-CONTINUE.

(2) Continue card has no value following the defined name.

fu. -.- REQUIRED FOR FAILURE EXIT

(1) LMLPASN: GENERATE-ASSIGN.
LMLPATC: GEN-ATCH-2.
LMLPBLD: GENERATE-BUILD.

(2) The period was missing prior to the failure exit.

fv. $$NRC AND $$ORC ARE ILLEGAL

(1) LMLPATC: GENERATE-ATTACH.

(2) System fields cannot be attached to variable sets.

fw. $$ORC $$NRC ILLEGAL RECEIVING FIELDS

(1) LMLPFMT: ERROR-OUT9.
LMLPFMT1: ERROR-OUT9.

(2) The system fields cannot be used as the receiving fields in an action statement.

fx. 20 CHARACTERS MAXIMUM PROFILE SIZE

(1) LMLPSPO: RETURN2.

(2) Profile mask can only be twenty characters long.

fy. 100 DEFINES LIMIT EXCEEDED

(1) LMLPDEF: DEF-TAB-AD.

(2) Maximum number of defines in a given logic package is limited to 100.