Xerox BPM/BTM/UTS

Sigma 5-9 Computers

System Generation
Technical Manual

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

XEROX

Xerox BPM/BTM/UTS

Sigma 5-9 Computers

System Generation
Technical Manual

90 18 77B

September 1973

Price: $13.00

Printed in U.S.A.

REVISION

The System Generation processors described in this manual operate under the D00 version of UTS and the H00 version of BPM/BTM.

RELATED PUBLICATIONS

Title	Publication No.
Xerox Sigma 5 Computer/Reference Manual	90 09 59
Xerox Sigma 6 Computer/Reference Manual	90 17 13
Xerox Sigma 7 Computer/Reference Manual	90 09 50
Xerox Sigma 8 Computer/Reference Manual	90 17 49
Xerox Sigma 9 Computer/Reference Manual	90 17 33
Xerox Batch Processing Monitor (BPM) and Batch Time-Sharing Monitor (BTM)/SM Reference Manual	90 17 41
Xerox Control Program-Five (CP-V)/SM Reference Manual	90 16 74
Xerox Batch Processing Monitor (BPM)/System Technical Manual	90 15 28
Xerox BPM/BTM Subsystems and Utilities Technical Manual	90 30 61
Xerox BPM/BTM/UTS/Overlay Loader Technical Manual	90 18 03
Xerox BPM/BTM/UTS Peripheral Conversion Language (PCL) Technical Manual	90 19 32
Xerox Batch Time-Sharing Monitor (BTM)/Delta Subsystem Technical Manual	90 18 79
Xerox Batch Time-Sharing Monitor (BTM)/Edit Subsystem Technical Manual	90 19 11
UTS Overview and Index Technical Manual	90 19 84
UTS Basic Control and Basic I/O Technical Manual	90 19 85
UTS System and Memory Management Technical Manual	90 19 86
UTS Symbiont and Job Management Technical Manual	90 19 87
UTS Operator Communication and Monitor Services Technical Manual	90 19 88
UTS File Management Technical Manual	90 19 89
UTS Reliability and Maintainability Technical Manual	90 19 90
UTS Initialization and Recovery Technical Manual	90 19 92
UTS Command Processors Technical Manual	90 19 93
UTS System Processors Technical Manual	90 19 94
UTS Data Bases Technical Manual	90 19 95

Manual Content Codes: BP – batch processing, LN – language, OPS – operations, RP – remote processing, RT – real-time, SM – system management, TS – time-sharing, UT – utilities.

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

CONTENTS

APPENDIXES

FIGURES

TABLES

PREFACE

This document describes the purpose and architecture of the System Generation processors that operate under BPM/BTM/UTS. It is assumed that the user is familiar with information contained in other operating system manuals, particularly those listed on the related publications page.

1.0 SYSGEN OVERVIEW.

1.1 INTRODUCTION.

SYSGEN comprises a series of processors capable of forming a UTS or BPM/BTM system tailored to a specified installation. These processors are PASS2, PASS3, DEF and LOCCT, each of which has various control commands described in detail in the following chapters. Files are accessed from Xerox-supplied or user tapes or for BPM/BTM private disc packs via the Peripheral Conversion Language (PCL) processor. Discussion of this processor is not included in this manual.

1.2 SYNTACTICAL REQUIREMENTS

In general, the various SYSGEN processors have certain common syntactical rules and requirements that may be applied to their control commands. Any deviation for a given processor is noted under the detailed discussion of it.

1. The legitimate characters that may be used in names are:

   Alpha: A-Z, a-z

   Numeric: 0-9, X'A'-X'F'

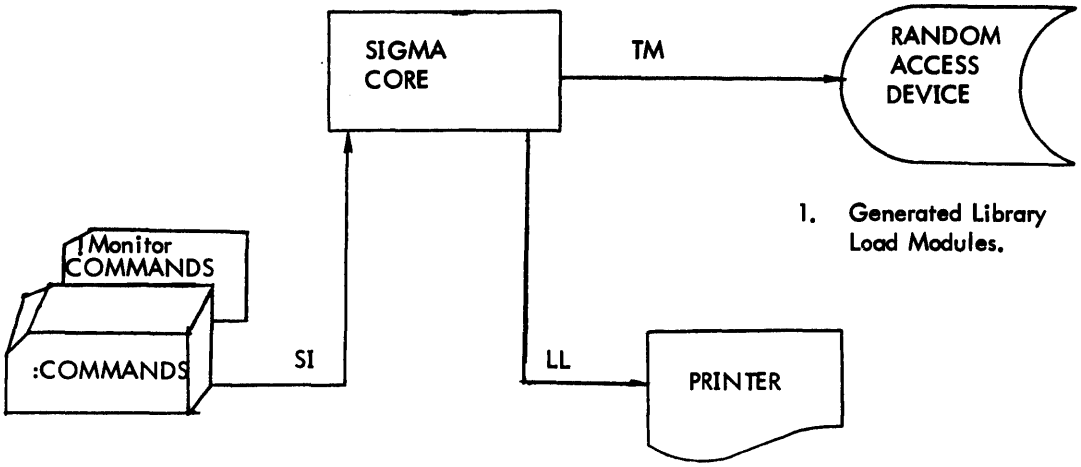
   Special (alpha): $ - | _ : # @
2. All control commands to processors begin with a : in column 1.
3. When the options for a particular control command do not all fit on a physical image (80 column), they may be continued on one or more images.
4. Continuation is indicated by the use of a semicolon and the continuation command must have a : in column 1. A semicolon may be placed anywhere within a command as well as anywhere within a name. If the semicolon is found within a name, the continuation command must contain the remainder of the name, starting with the second character position so that the name may be reformed.
5. There are three methods of incorporating comments with a processor control command sequence.
   a. Comments are accepted following a semicolon.
   b. If no semicolon is used, comments are accepted if preceeded by a period.
   c. If an entire command is a comment, the first character in the image must be an asterisk.

1.3 THE SYSGEN PROCESSORS.

1.3.1 PASS2.

This SYSGEN processor receives as input various parameters concerning a target system. PASS2 then generates the library modules (nearly all of which are load modules) which identify the system variables. These modules are incorporated into the target system's Monitor and any other processor which requires a knowledge of the target system's configuration. The target system parameters include: peripheral definitions, operational label assignments, real-time information, symbiont device information, core size, number of index and blocking buffers.

The following diagram shows what operational labels and corresponding peripherals are accessed during a PASS2.



1. Generated Library Load Modules.

1. PASS2 Control Commands.

1. Display control Information.

PASS2 is entered via the Monitor control command !PASS2 and terminates input from the SI device when a Monitor control command is encountered (i.e., a control with "!" in column 1).

The PASS2 generated Library Modules consist of:

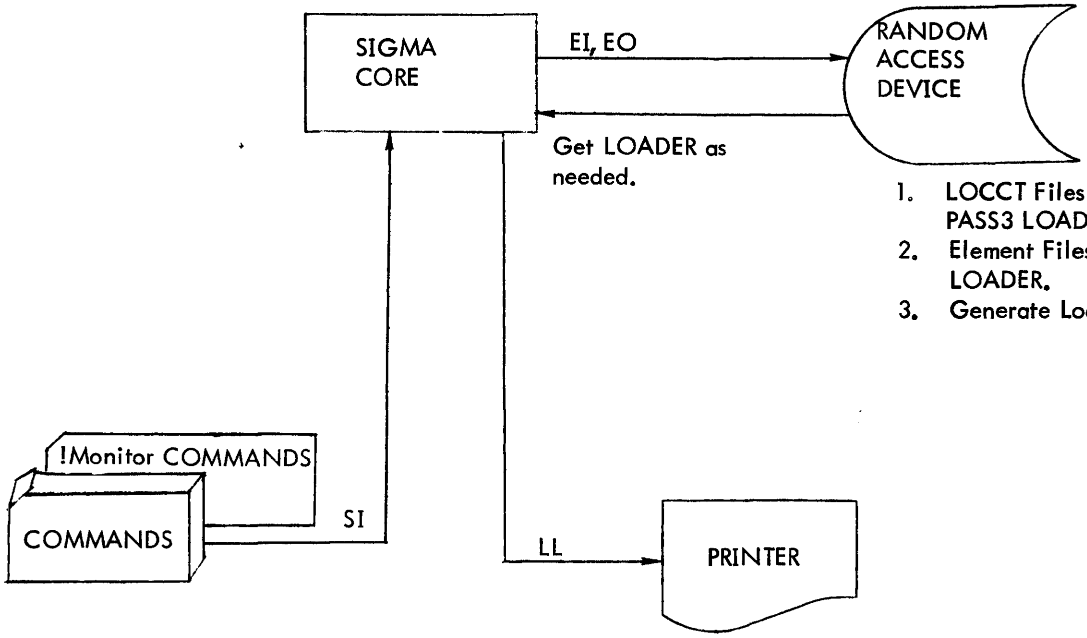
NAME	*GENERATED BY	MODULE TYPE
IOTABLE	CHAN, DEVICE, STDLB, and OSTDLB (UTS)	Load Module
SPEC:HAND	DEVICE	Data
M:HGP	CHAN, DEVICE	Load Module
M:SDEV	SDEVICE	Load Module
M:CPU	UTM (UTS) MONITOR (BPM/BTM)	Load Module
MON::ORG	UTM (UTS) MONITOR (BPM/BTM)	Relocatable Object Module
M:SYMB	UTM (UTS)	Load Module
M:BIG9	UTM (UTS)	Load Module
M:BLIMIT	BLIMIT (UTS)	Load Module
M:OLIMIT	OLIMIT (UTS)	Load Module
M:ELIMIT	ELIMIT (UTS)	Load Module
M:DLIMIT	DLIMIT (BPM/BTM)	Load Module
M:ABS	ABS (BPM/BTM)	Load Module
M:BTM	BTM (BPM/BTM)	Load Module
M:FRGD	FRGD, INTLB (BPM/BTM)	Load Module
M:COC	COC (UTS)	Load Module
M:IMC	IMC (UTS)	Load Module
M:SPROCS	SPROCS (UTS)	Load Module
M:PART	PARTITION (UTS)	Load Module

*NOTE: When a specific system type is specified (in parentheses), the preceding control command is the one that generates or helps in generating the Module NAME. Otherwise, if no system is specified, the assumption is BPM/BTM/UTS.

1.3.2 PASS3.

This SYSGEN processor communicates to the system LOADER the necessary information to load a specific Monitor, processor or library. Each PASS3 control command identifies a file which contains information for the LOADER. Such a file is referred to as an LOCCT (see LOCCT processor overview 1.3.3). These LOCCT files eliminate the maintaining of Monitor, processor or library LOAD/TREE control command structures. An LOCCT conveys to the LOADER the information which the original LOAD/TREE control commands contained as parameters (e.g., element file names, load parameters, tree structure).

The following diagram shows what operational labels and corresponding peripherals are accessed during a PASS3.



1. LOCCT Files used by PASS3 LOADER.
2. Element Files used by LOADER.
3. Generate Load Module.

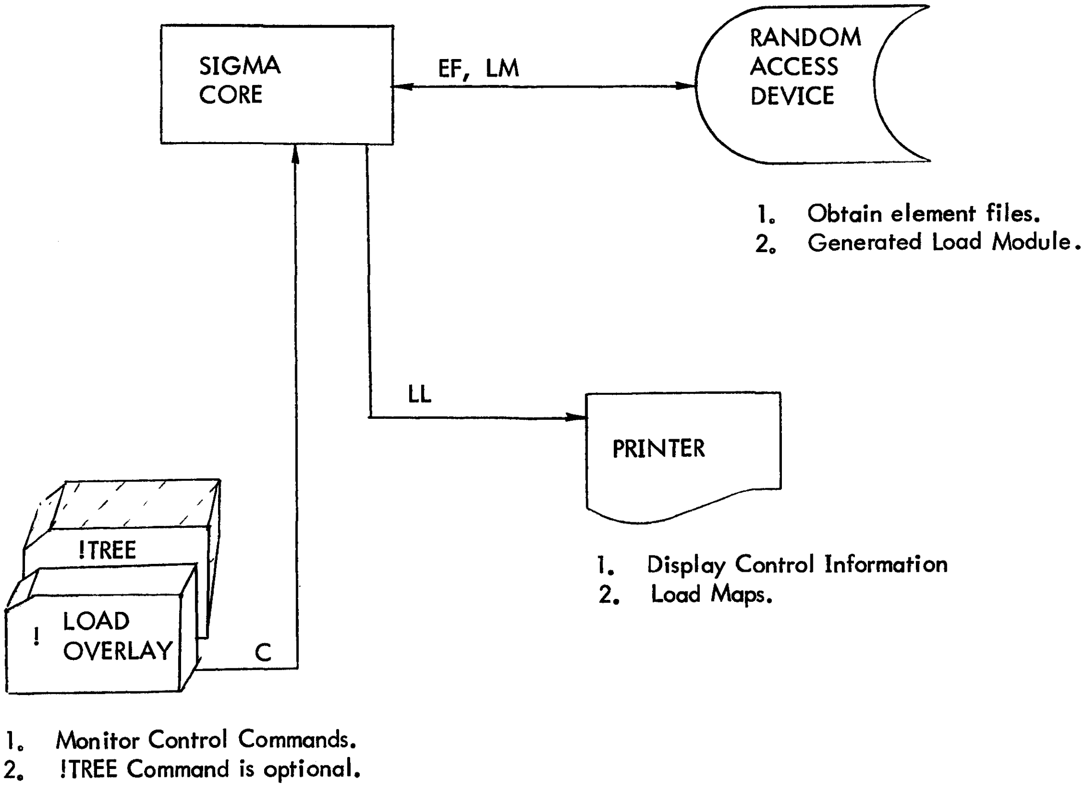
1. PASS3 Control Commands.

1. Display Control Information.
2. Load Maps.

PASS3 is entered via the Monitor control command !PASS3 and terminates input from the SI device when a Monitor control command is encountered (i.e., a control command with "!" in column 1).

Processors and Libraries may also be loaded by the system LOADER and eventually included (through DEF processor) in a target system without the use of the SYSGEN PASS3/LOCCT processors. This can be accomplished by using the !LOAD (!OVERLAY) and !TREE Monitor control commands.

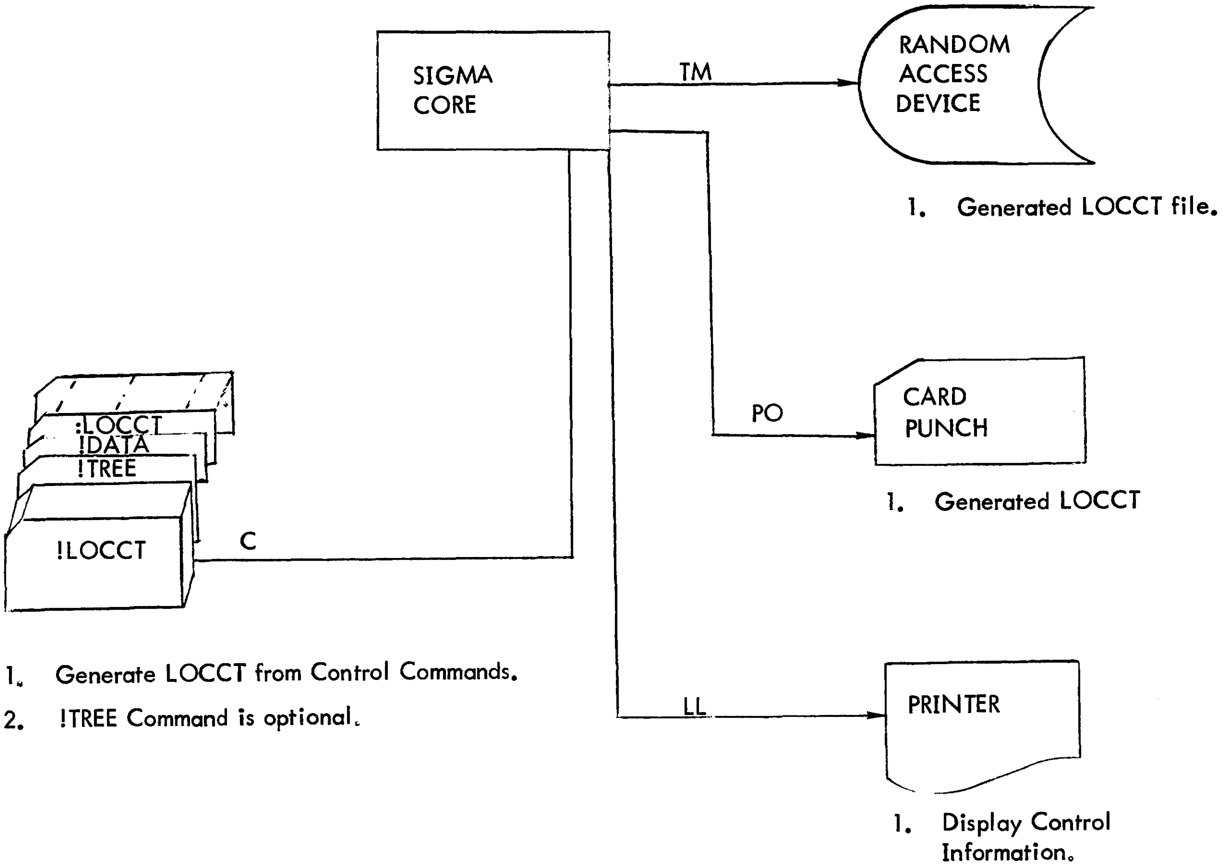
The following diagram shows what operational labels and corresponding peripherals are accessed during this procedure.


SIGMA CORE
EF, LM
RANDOM ACCESS DEVICE
1. Obtain element files.
2. Generated Load Module.
LL
PRINTER
1. Display Control Information
2. Load Maps.
!TREE
! LOAD OVERLAY
C
1. Monitor Control Commands.
2. !TREE Command is optional.

1.3.3 LOCCT.

This SYSGEN processor intercepts from the system's Control Command Interpreter (CCI) the table of information generated from LOAD (!LOCCT)/ !TREE control commands for the system LOADER. This table of information is referred to as an LOCCT (Loader Control Command Table). An LOCCT, when obtained from CCI, is converted into a permanent file. Therefore, such a file may be used by SYSGEN PASS3 for purposes of loading a Monitor, processor or library.

The following diagram shows what operational labels and corresponding peripherals are accessed during a LOCCT process.



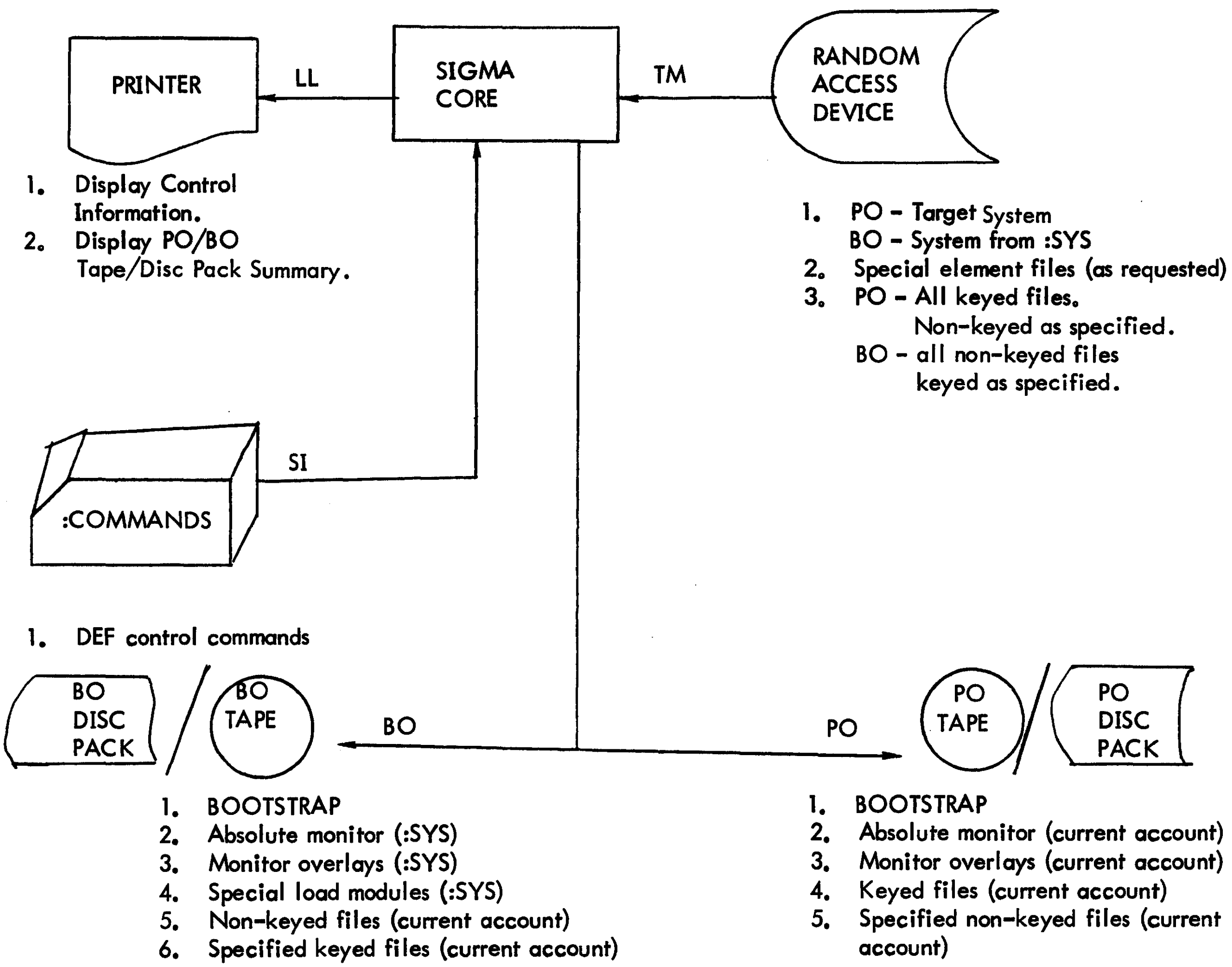
1.3.4 DEF.

Currently, there are two versions of DEF; one for UTS (D00 version) and pre-H00 versions of BPM/BTM as described in Chapter 4, and one for BPM/BTM (H00 release) as described in Appendix C.

This SYSGEN processor generates either a target system tape (a PO tape) or a BO Tape which may eventually be used as a new BI tape for subsequent SYSGENs. DEF may create multiple tapes in any given run. In addition, for BPM/BTM only, DEF generates either a BO disc pack or PO disc pack, which are functionally synonymous to BO/PO tapes.

If the tape/disc pack being created is a BO tape/disc pack, it contains a bootstrap, absolute monitor, the monitor overlays and the load modules PCL, CCI, LOADER, PASS2, LOCCT, PASS3, DEF, FMGE, ERRMSG, M:MON:LIB – :DIC containing the system DCBs. VOLINIT is also included if a BO disc pack is being generated. In addition, if the system is a UTS system, then the following are also automatically included: XDELTA, LOGON, TEL, SUPER, DEFCOM, SUPER, JIT0, JIT1, JIT2, JIT3, JIT6, ANLZ, GHOST1, RECOVER, M:SPROCS (containing the overlays of M:MON). For BO tapes the null file LASTLM terminates the load module portion of the tape. All of the named files are obtained from the System account (:SYS). For BO disc packs, LASTLM contains the names of all the files on the BO pack that are to go into the :SYS account. This is necessary because disc pack files are accessed through the alphabetized file directory. DEF then obtains all of the non-keyed files from the current account and any keyed files that have been specifically identified via are :INCLUDE command and writes these on the tape or disc pack.

The following diagram shows what operational labels and corresponding peripherals are accessed during a DEF.



1. Display Control Information.
2. Display PO/BO Tape/Disc Pack Summary.

1. PO - Target System
   BO - System from :SYS
2. Special element files (as requested)
3. PO - All keyed files.
   Non-keyed as specified.
   BO - all non-keyed files
   keyed as specified.

1. DEF control commands

1. BOOTSTRAP
2. Absolute monitor (:SYS)
3. Monitor overlays (:SYS)
4. Special load modules (:SYS)
5. Non-keyed files (current account)
6. Specified keyed files (current account)

1. BOOTSTRAP
2. Absolute monitor (current account)
3. Monitor overlays (current account)
4. Keyed files (current account)
5. Specified non-keyed files (current account)

If the tape/disc pack being created is a PO tape/disc pack, it contains a bootstrap, or absolute monitor (for the target system), the monitor overlays (for the target system), special element files if requested and all of the keyed files (normally load module) from the current account unless certain of these are specifically identified to be :IGNORED. The null file, LASTLM, terminates the PO tape/disc pack. Since all files on the tape/disc pack are to go into the :SYS account, LASTLM is the same for PO tapes and disc packs.

DEF is entered via the MONITOR control command !DEF and terminates input from the SI device when a Monitor control command is encountered (i.e., a control command with a ! in column 1). DEF requires assignments for PO and/or BO.

1.4 FUNCTIONAL FLOW OF SYSGEN

The following diagram represents a functional flow of the SYSGEN operation. Note, the LOCCT process may not be needed and the PASS3 function may be replaced or augmented by using the system LOAD control commands to form a processor or library although PASS3 must be used to load M:MON.

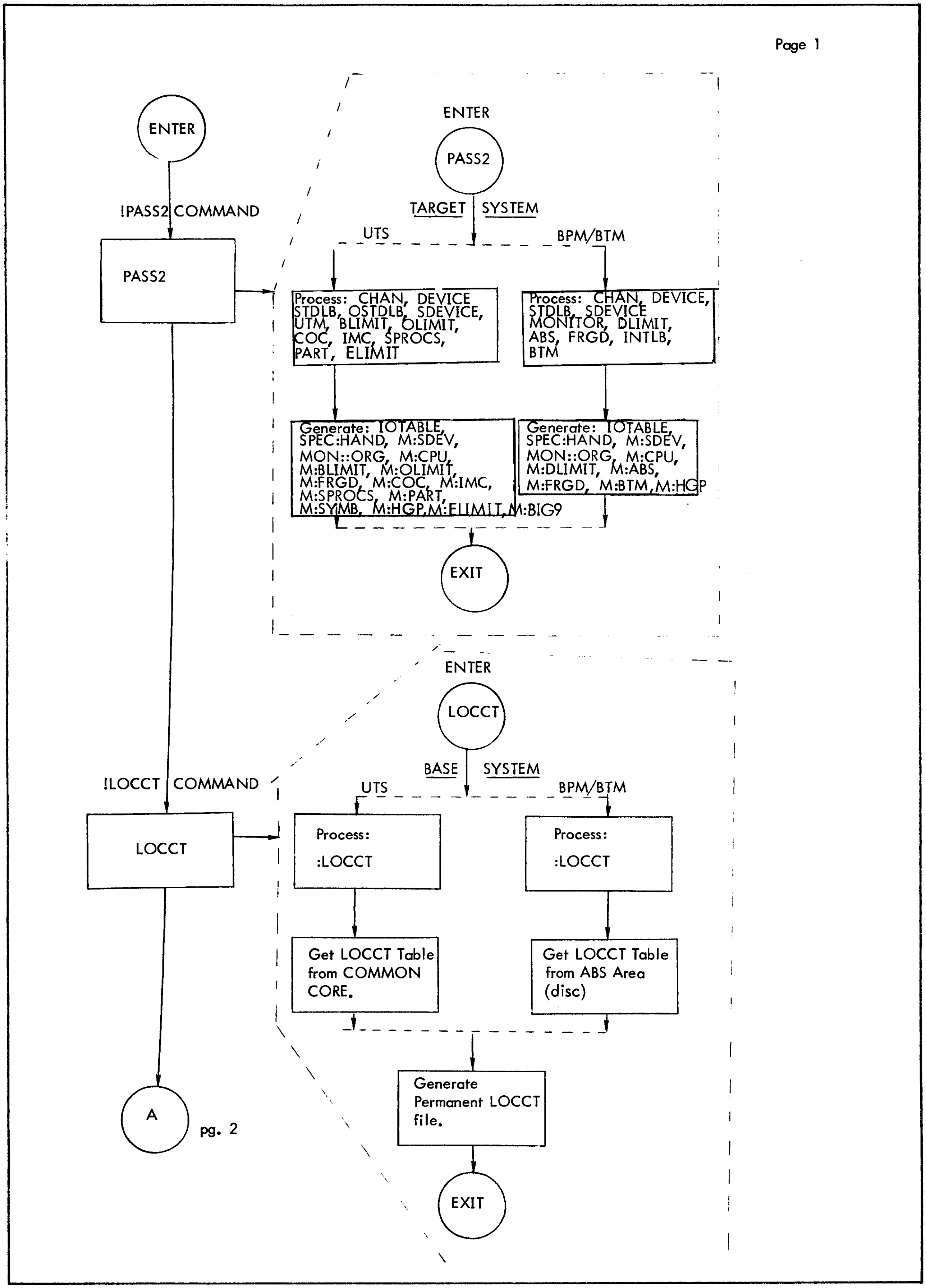

Page 1
ENTER
!PASS2 COMMAND
PASS2
ENTER
PASS2
TARGET SYSTEM
UTS
BPM/BTM
Process: CHAN, DEVICE STDLB, OSTDLB, SDEVICE, UTM, BLIMIT, OLIMIT, COC, IMC, SPROCS, PART, ELIMIT
Process: CHAN, DEVICE, STDLB, SDEVICE MONITOR, DLIMIT, ABS, FRGD, INTLB, BTM
Generate: IOTABLE, SPEC:HAND, M:SDEV, MON::ORG, M:CPU, M:BLIMIT, M:OLIMIT, M:FRGD, M:COC, M:IMC, M:SPROCS, M:PART, M:SYMB, M:HGP,M:ELIMIT,M:BIG9
Generate: IOTABLE, SPEC:HAND, M:SDEV, MON::ORG, M:CPU, M:DLIMIT, M:ABS, M:FRGD, M:BTM,M:HGP
EXIT
!LOCCT COMMAND
LOCCT
ENTER
LOCCT
BASE SYSTEM
UTS
BPM/BTM
Process: :LOCCT
Process: :LOCCT
Get LOCCT Table from COMMON CORE.
Get LOCCT Table from ABS Area (disc)
Generate Permanent LOCCT file.
EXIT
A
pg. 2

Figure 1-1. Flow Diagram of SYSGEN

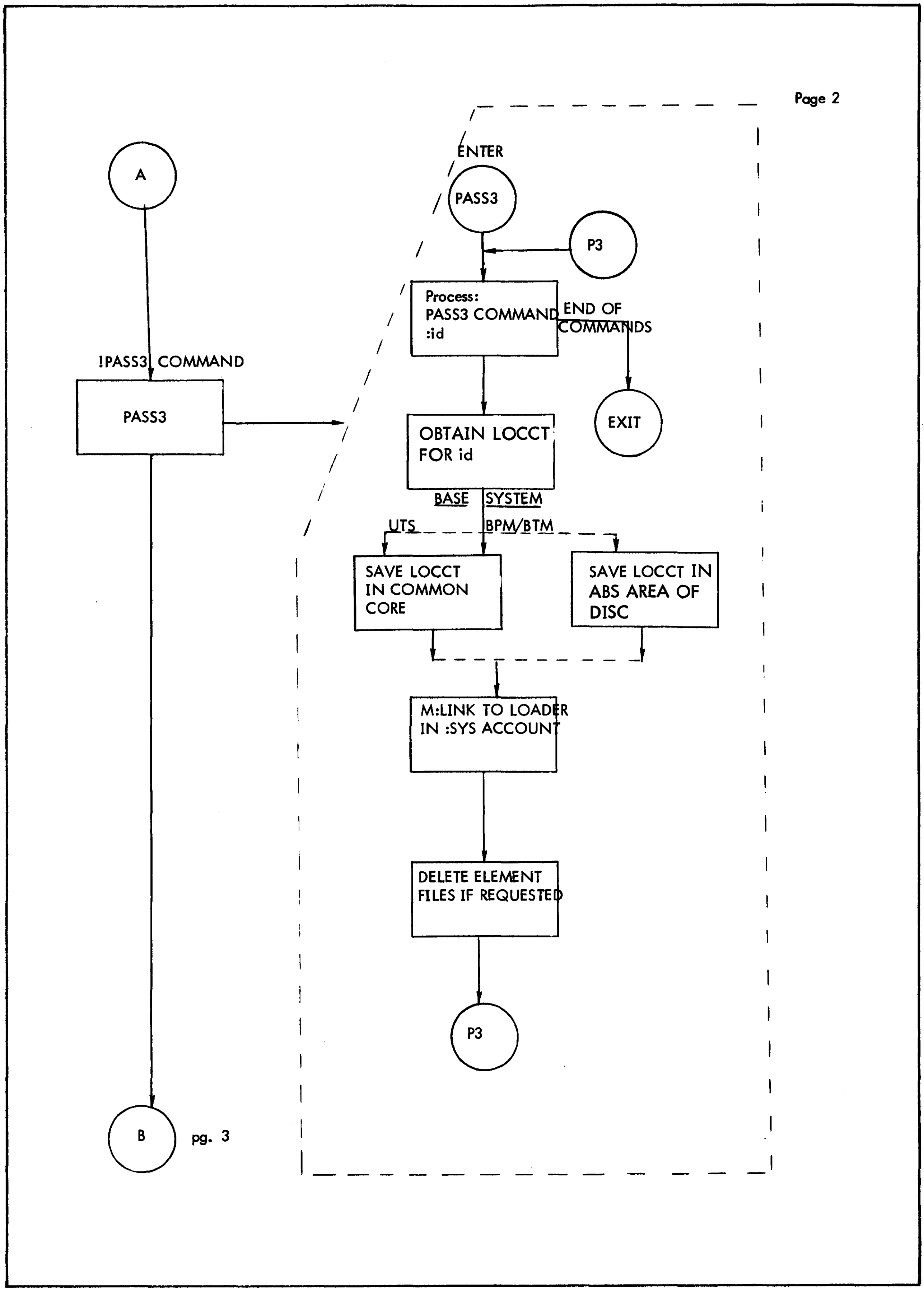


Figure 1-1. Flow Diagram of SYSGEN (Cont)

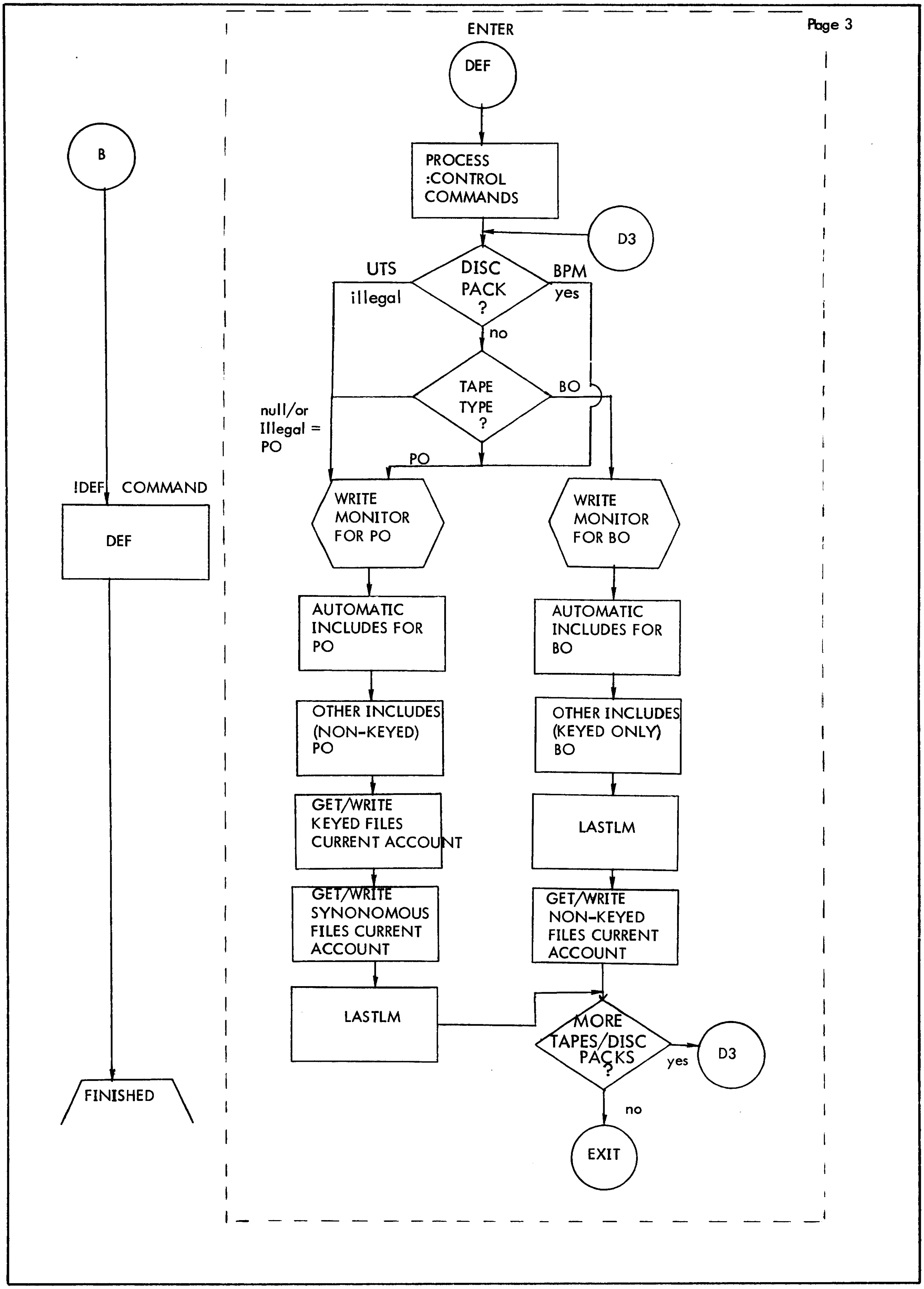

Page 3
ENTER
DEF
B
PROCESS :CONTROL COMMANDS
D3
UTS
illegal
DISC PACK ?
BPM
yes
no
TAPE TYPE ?
BO
null/or Illegal = PO
PO
!DEF COMMAND
DEF
WRITE MONITOR FOR PO
WRITE MONITOR FOR BO
AUTOMATIC INCLUDES FOR PO
AUTOMATIC INCLUDES FOR BO
OTHER INCLUDES (NON-KEYED) PO
OTHER INCLUDES (KEYED ONLY) BO
GET/WRITE KEYED FILES CURRENT ACCOUNT
LASTLM
GET/WRITE SYNONOMOUS FILES CURRENT ACCOUNT
GET/WRITE NON-KEYED FILES CURRENT ACCOUNT
LASTLM
MORE TAPES/DISC PACKS ?
yes
D3
no
EXIT
FINISHED

Figure 1-1. Flow Diagram of SYSGEN (Cont)

2.0 PASS2

2.1 P2CCI

2.1.1 Purpose

To read PASS2 control commands and call the appropriate processors to handle them. P2CCI is the root segment of PASS2.

2.1.2 Calling Sequence

P2CCI is called by the monitor control command:

```
!PASS2...
```

which runs the load module PASS2 of which P2CCI is the initial part.

2.1.3 Return

P2CCI exits when it encounters an end of data, that is, a card with a "!" in column 1 or an END command. It may also make earlier error exits to the monitor if significant faults are found.

2.1.4 Input

Register 0 contains the address of the pointer to PASS2's temp stack.

2.1.5 Output

Display of PASS2 control information to the LL device.

2.1.6 Subroutines Used

NAMSCAN (Used to get PASS2 "type" field from !PASS2 control command.)

2.1.7 Base Registers

Register 3 = Address of data in PASS2's temp stack.

Register 2 = PASS2 type index where:

0 = BPM

2 = UTMBPM

Register 7 = Address of FETCHLST

2.1.8 Description

P2CCI is entered when the monitor control command !PASS2 is encountered. Its first action is to move its dynamic data to the temp stack. The resulting form of the stack is as shown in Table 2-1.

Table 2-1. PASS2 STACK ALLOCATION

BASESTAC	EQU	0	REL•DISPLACEMENT TO STACK BASE
SSIZE	EQU	BASESTAC+1	DISC SECTOR SIZE (FROM :DEVICE)
CORE	EQU	BASESTAC+2	CORE SIZE (FROM :MONITOR)
SDGANSG	EQU	BASESTAC+3	#GRAN/PER, #SEC2/GRAN (FROM :DEVICE)
OPTNWD	EQU	BASESTAC+4	TEMP STORAGE
72FLAG	EQU	BASESTAC+5	#7202-04 RADS

```
LASTSPEC   EQU   BASESTAC+6
CCBUFRS    EQU   LASTSPEC+1                    BASE OF BUFFERS

BUFFADDR   EQU   CCBUFRS+0                     POINTER TO CC BUFFER (=BUFFER)
BUFFER     EQU   CCBUFRS+1                     CONTROL COMMAND BUFFER (20 WORDS)
FETCHLST   EQU   CCBUFRS+21                    CC  PROCESSOR PARAM. LIST  (7 WORDS)
FETCHCCP   EQU   CCBUFRS+24                    CURNT•CHAR•POS•PROCESSED IN CC
FETCHCSL   EQU   CCBUFRS+26                    CHAR•STRING LENGTH OF FIELD IN CC
FETCHBUF   EQU   CCBUFRS+28                    CC CHAR•STRING BUFFER (9 WORDS)
FETCHADR   EQU   CCBUFRS+37                    POINTER TO FETCHLST (=FETCHLST)
XBUFADDR   EQU   CCBUFRS+38                    POINTER TO XBUFFER (=XBUFFER)
XBUFFER    EQU   CCBUFRS+39                    ERROR BUFFER FOR $ CHAR.
P2FLAGS    EQU   CCBUFRS+59                    BASE OF PASS2 FLAGS

CCFLAGS    EQU   P2FLAGS+0                     CC FLAGS FOR CC TYPE FOUND
*    00000000000000000000000000000000            < ALL CC TYPES MISSING >
*    00000000000000000000000000000001                  :CHAN      FOUND   (00000001)
*    00000000000000000000000000000010                  :DEVICE    FOUND   (00000002)
*    00000000000000000000000000000100                  :STDLB     FOUND   (00000004)
*    00000000000000000000000000001000                  :SDEVICE   FOUND   (00000008)
*    00000000000000000000000000010000                  :MONITOR   FOUND   (00000010)
*    00000000000000000000000000100000                  :DLIMIT    FOUND   (00000020)
*    00000000000000000000000001000000                  :ABS       FOUND   (00000040)
*    00000000000000000000000010000000                  :BTM       FOUND   (00000080)
*    00000000000000000000000100000000                  :FRGD      FOUND   (00000100)
*    00000000000000000000001000000000                  :INTLB     FOUND   (00000200)
*    00000000000000000000010000000000                  :COC       FOUND   (00000400)
*    00000000000000000000100000000000                  :IMC       FOUND   (00000800)
*    00000000000000000001000000000000                  :SPROCS    FOUND   (00001000)
*    00000000000000000010000000000000                  :BLIMIT    FOUND   (00002000)
*    00000000000000000100000000000000                  :OLIMIT    FOUND   (00004000)
*    00000000000000001000000000000000                  :UTM       FOUND   (00008000)
*    00000000000000010000000000000000                  :OSTDLB    FOUND   (00010000)
*    00000000000000100000000000000000                  :PART      FOUND   X'20000'
*    00000000000001000000000000000000                  :ELIMIT    FOUND   X'40000'
*    00000000100000000000000000000000                  BATCH HEADER FLAG
*    00000000001000000000000000000000                  ON-LINE HEADER FLAG
*    01000000000000000000000000000000           :DEVICE CC JUST PROC         (40000000)
*    10000000000000000000000000000000           END OF CC's < EOF >          (80000000)
```

CHANFLG	EQU	P2FLAGS+1	CHAN CC ENCOUNTERED
STDFLG	EQU	P2FLAGS+2	STDLB CC BEING PROCESSED
P2TYPE	EQU	P2FLAGS+3	PASS2 TYPE (BPM, UTMBPM)
OSTD:STD	EQU	P2FLAGS+4	STDLB/OSTDLB FLAG
P2CNTRS	EQU	P2FLAGS+5	BASE OF PASS2 COUNTERS
RCHAN	EQU	P2CNTRS+0	# CHAN CC'S
SDEVFLG	EQU	P2CNTRS+1	# SYMBIONT DEVICES
P2CORE	EQU	P2CNTRS+2	BASE OF PASS2 CORE VALUES
DYSTORND	EQU	P2CORE+0	END OF AVAILABLE CORE FOR PASS2
SAVEPAGE	EQU	P2CORE+1	# PAGES OF CORE, 1ST PAGE ADDR.
DEVS	EQU	P2CORE+3	DP/7T/9T/0
COCS	EQU	P2CORE+4	COC ADDRESS NDD
SWAPBTM	EQU	P2CORE+8	SWAPPER FLAG FOR BTM
AVTBLGTH	EQU	P2CORE+9	AVRTABLE SIZE
LORBIN	EQU	P2CORE+10	LOW RBT DCT INDEX
HIRBIN	EQU	P2CORE+11	HIGH RBT DCT INDEX
#RBTS	EQU	P2CORE+12	# RBTs DEFINED
# PRDP	EQU	P2CORE+13	# PRIVATE PACKS DEFINED
DUALFLG	EQU	P2CORE+14	DUAL FLAG
SCYLPSA	EQU	P2CORE+15	# GRAN/PHYCYL; # PHYCYL
BIG9FLG	EQU	P2CORE+16	BIG9 FLAG FOR UTS
SWAPUTS	EQU	P2CORE+17	DPSWAPPER FOR UTS
P2DYNEND	EQU	P2CORE+18	END BASIC PASS2 STACK

The stack data area is pointed to by R3. That is, to obtain the contents of the word "BUFFER" one executes:

```
                    LW, REG     BUFFER, R3
```

This area is initialized to all zeros except that XBUFFER is filled with blanks and a seven word PLIST is moved to FETCHLST.

Once the data area has been initialized P2CCI reads the "type" field from the !PASS2 card and tests it for validity. If it is not valid P2CCI prints an error message and does an ERROR exit to the monitor, otherwise the type is stored in P2TYPE and P2CCI begins to read cards.

When P2CCI reads a card, a check is first made to determine whether the card was an end of data. If it was, actions discussed below are taken. Otherwise the card is checked to see if it is out of order. A :CHAN card must precede any :DEVICE card; and all :CHAN, :DEVICE, :STDLB, and :OSTDLB cards must precede :SDEVICE, and :MONITOR or :UTM cards. In addition, :UTM must precede :SPROCS and :IMC. Also, most cards may appear only once. If a card is illegal or out of order, it is listed with appropriate error information and P2CCI continues.

If the card is found acceptable it is used to obtain an index to the table of PASS2 processor overlays. This index and the PASS2 type index are used to decide the name of the required overlay which is segloaded and entered. Illegal control commands for the type of PASS2 being performed are detected at this stage. (See Table 2-2.) When the processor overlay completes it's operations it returns to P2CCI which reads another card and calls another processor overlay. Some processor overlays read cards internally for themselves.

When an end of data is found P2CCI prints a list of PASS2 commands that have not been encountered and exits to the monitor.

2.1.9 P2CCI MESSAGES

...PASS2 CCI IN CONTROL... a !PASS2 command has called P2CCI.

***UNKNOWN TYPE the type field of the !PASS2 command was not BP(M) or UT(MBPM).

***UNKNOWN OR MISPLACED CC the conditions for a CC discussed above were not met.

***CC IGNORED, PREVIOUS CC OF THIS TYPE ENCOUNTERED a CC which can appear only once has been duplicated.

***CC'S NOT ENCOUNTERED, BUT POSSIBLY NEEDED heading for a list of control commands legal for this type PASS2 that were not processed. This list is printed just before P2CCI exits.

...END OF PASS2... P2CCI has completed its task and exits.

***CANNOT READ CONTINUATION RECORD-PASS2 ABORTED self-explanatory

***UTM MUST PRECEDE SPROCS/IMC--CC IGNORED :SPROCS or :IMC has just been read but :UTM has not yet been processed.

2.1.10 Internal Subroutines

Several subroutines internal to P2CCI and used by it and/or the PASS2 processors are discussed below.

CCLOAD

CCLOAD is used by P2CCI to load processors. It is called by the following sequence:

```
        BAL,SR4   CCLOAD
        TEXTC     "Segname"
```

CCLOAD does a segload CAL on the name addressed by SR4 then converts the byte count of the "segname" to words, adds this to SR4 and exits through SR4.

LISTIT

LISTIT is called to list the current control command by:

```
        BAL,SR4   LISTIT
```

It checks to see if the CC has already been listed and exits if so. If not, it marks the CC as listed and does a print CAL through M:LL.

PRINTMSG

PRINTMSG is usually called in the following form:

```
        BAL,SR4   A
   A    BAL,SR3   PRINTMSG
        TEXTC     'MESSAGE'
```

Effectively it is:

```
   PRINTMSG  M:PRINT   (MESg, *SR3)
             B         *SR4
```

READCC

READCC is called by:

```
        BAL,SR4   READCC
```

It BAL's to LISTIT to list the last CC if it has not been listed. Then it reads the next card. If the card begins with a colon, READCC exits. If the first character is a "_" it is listed (LISTIT) and the next card read. If neither, the first character of the card is printed (LISTIT) with error information and the next card is read. The process continues until a ':' card has been found.

RDINCFCH

RDINCFCH is called by :

```
        BAL,R4   RDINCFCH
```

It BAL 's to READCC to read a card then to NAMSCAN to get the first field. If the field is not a legal name it prints the card (LISTIT) with error information and branches to the beginning of P2CCI's card checking routine. Otherwise RDINCFCH moves the first 4 bytes of the name to R1 and exits.

OUTLLERR

OUTLLERR is called by:

```
        BAL,SR4    OUTLLERR
```

It lists the current control command (LISTIT) then obtains the current character position of the scan routines. It then builds a line for printing that is blank except for a '$' at that character position, prints it, and exits.

ABNRETUR

ABNRETUR is entered when an error or abnormal condition is encountered on a read from the C device. If the card being read is the !PASS2 command it error exits. Otherwise it assumes the abnormal was an end of file and sets a flag to this effect. AB\NRETUR then determines which processor was doing the read and returns control to that processor's end of data handling routine.

SYNTAX, COREALLOC, MODGEN and WRITELM are used by PASS2 overlays to process their control commands, allocate core for building a load module, creating DEFs and REFs, and writing the load module. A detailed discussion is found in Chapter 6.8 - 6.11.

Table 2-2. PASS2 Control Commands

CONTROL COMMAND	PROCESSOR CALLED		NOTES
	BPM	UTS	
:STDLB	UBCHAN	UBCHAN	
:CHAN	UBCHAN	UBCHAN	
:DEVICE	*ILLEGAL*	*ILLEGAL*	READ BY UBCHAN
:SDEVICE	SDEVICE	SDEVICE	
:MONITOR	XMONITOR	*ILLEGAL*	
:DLIMIT	XLIMIT	*ILLEGAL*	
:ABS	ABS	*ILLEGAL*	
:FRGD	FRGD	*ILLEGAL*	
:INTLB	*ILLEGAL*	*ILLEGAL*	READ BY FRGD
:BTM	BTM	*ILLEGAL*	
:COC	*ILLEGAL*	P2COC	
:IMC	*ILLEGAL*	IMC	
:SPROCS	*ILLEGAL*	SPROCS	
:BLIMIT	*ILLEGAL*	XLIMIT	
:OLIMIT	*ILLEGAL*	XLIMIT	
:ELIMIT	*ILLEGAL*	XLIMIT	
:UTM	*ILLEGAL*	XMONITOR	
:OSTDLB	*ILLEGAL*	UBCHAN	
:PARTITION	*ILLEGAL*	XPART	

2.1.11 Flow Charts

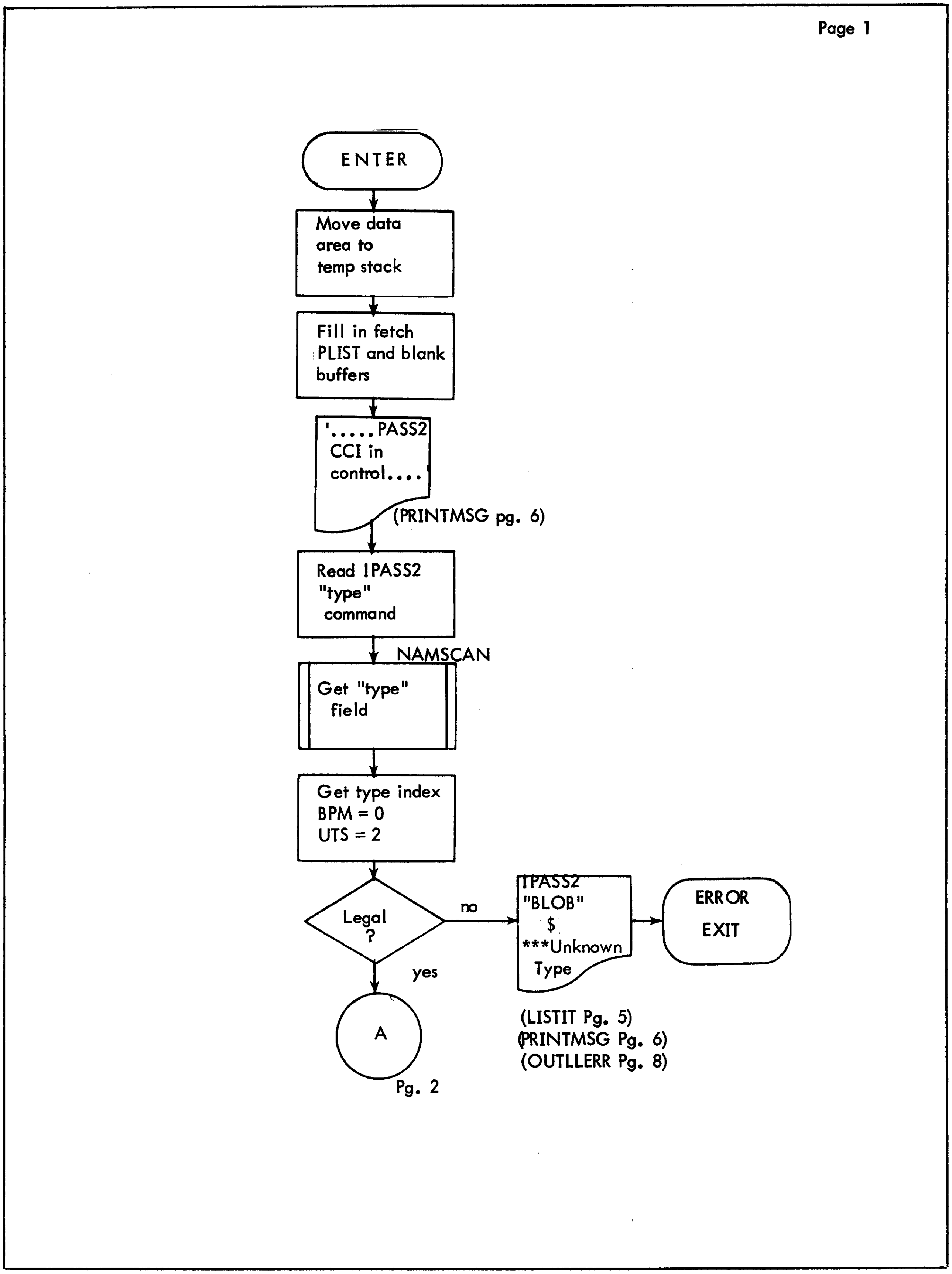

Page 1
ENTER
Move data area to temp stack
Fill in fetch PLIST and blank buffers
'.....PASS2 CCI in control....'
(PRINTMSG pg. 6)
Read !PASS2 "type" command
NAMSCAN
Get "type" field
Get type index
BPM = 0
UTS = 2
Legal ?
no
yes
!PASS2 "BLOB" $ ***Unknown Type
ERROR EXIT
(LISTIT Pg. 5)
(PRINTMSG Pg. 6)
(OUTLLERR Pg. 8)
A
Pg. 2

Figure 2-1. Flow Diagram of P2CCI

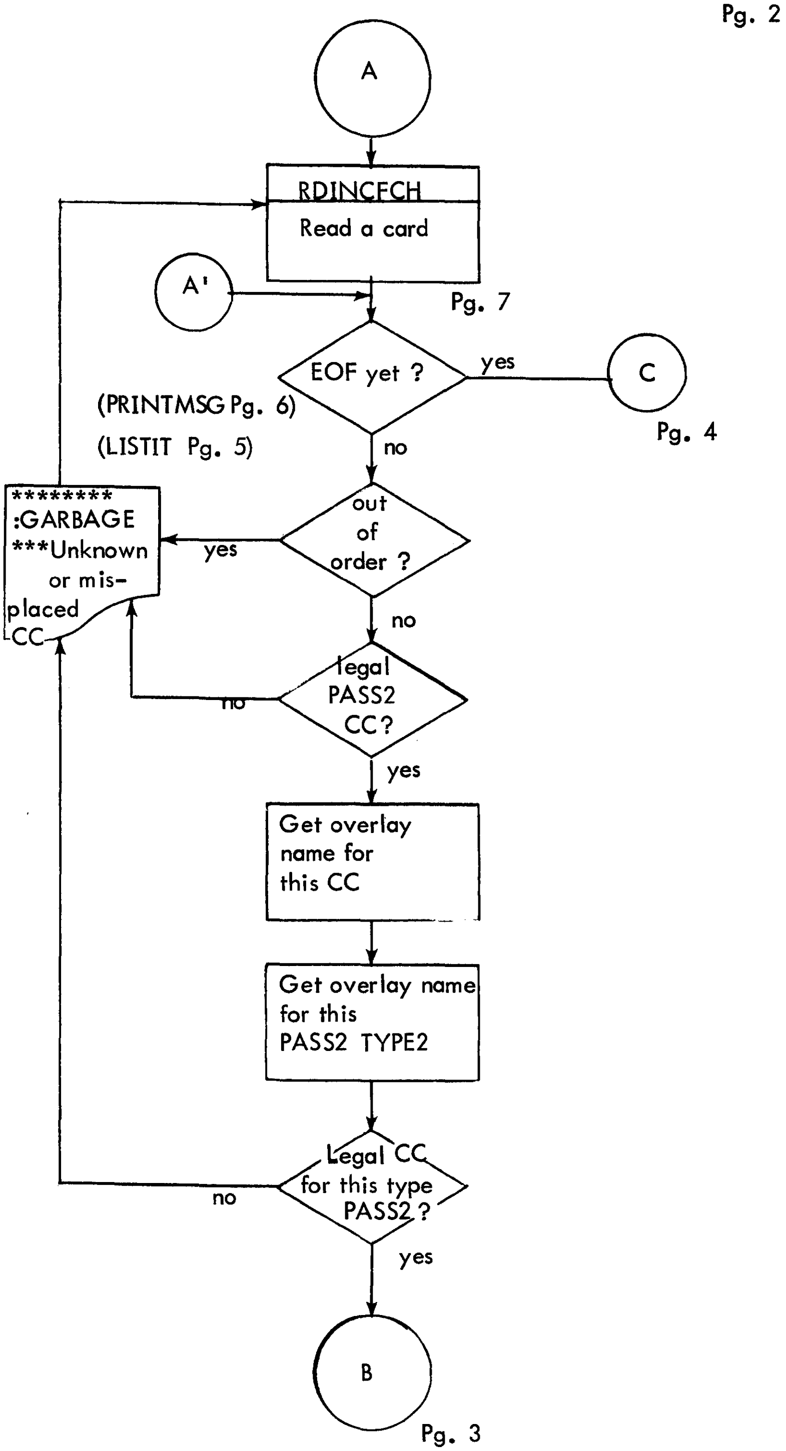


Figure 2.1. Flow Diagram of P2CCI (cont.)

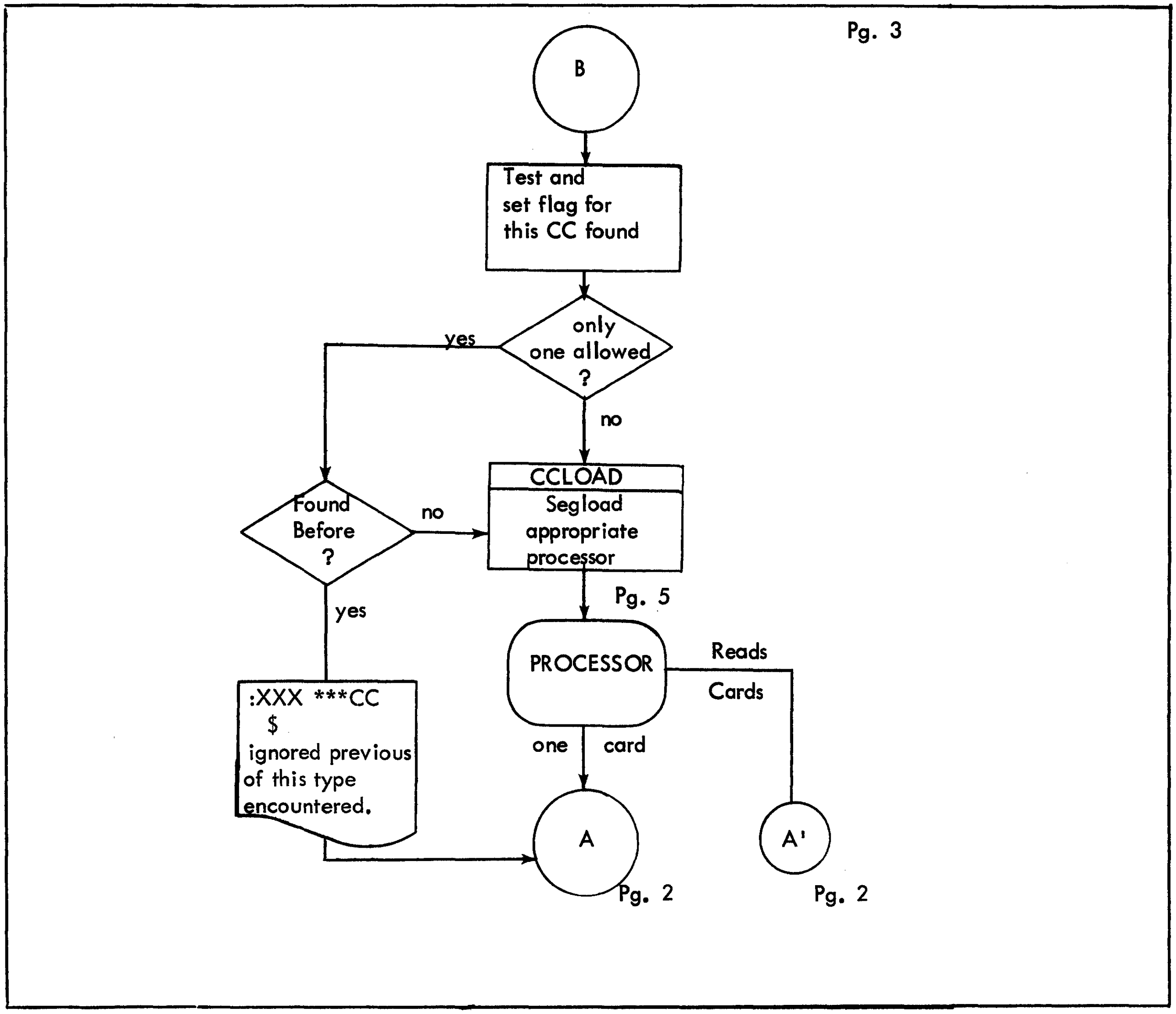


Figure 2-1 Flow Diagram of P2CCI (Cont.)

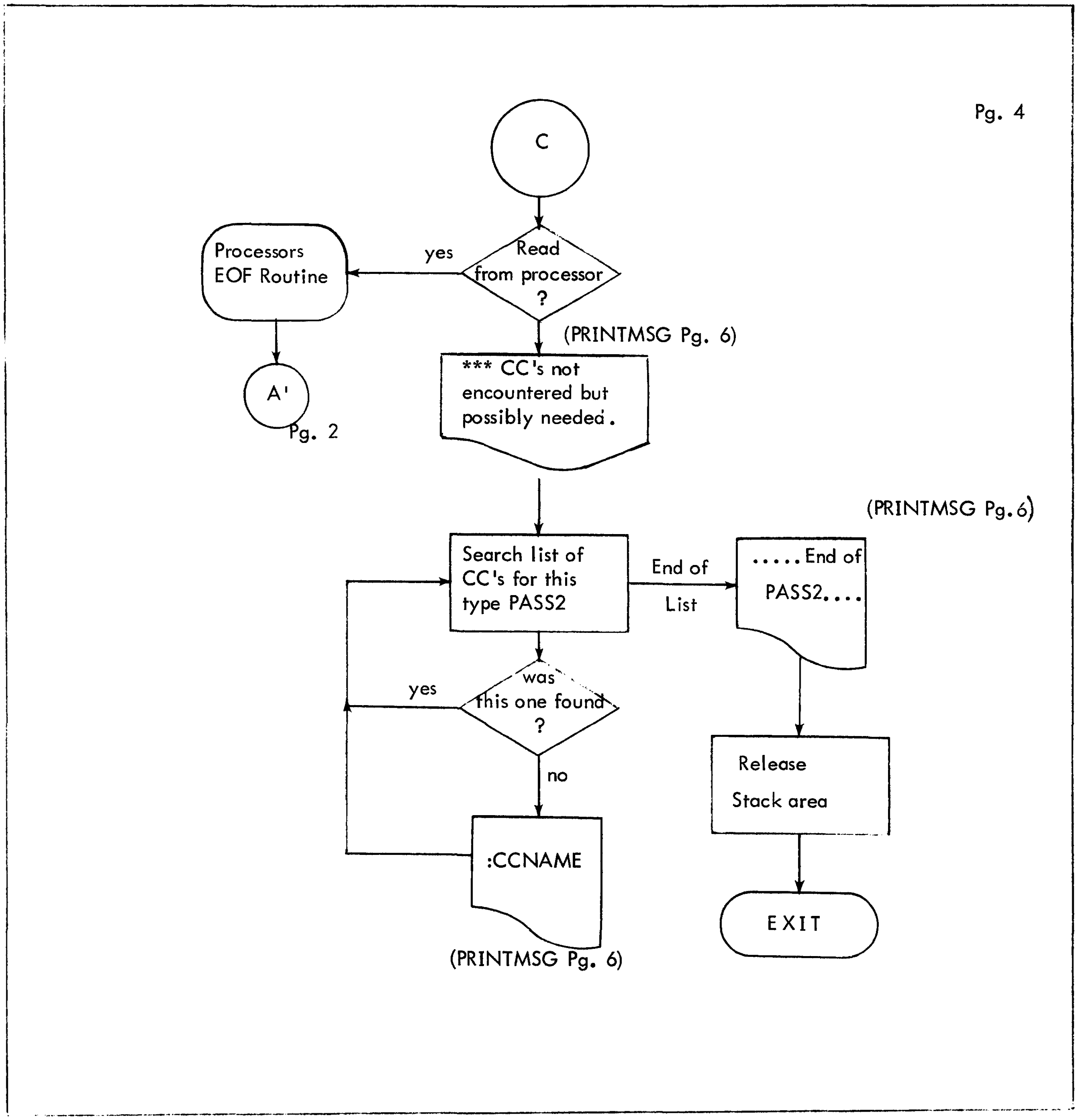


Figure 2-1. Flow Diagram of P2CCI (Cont.)

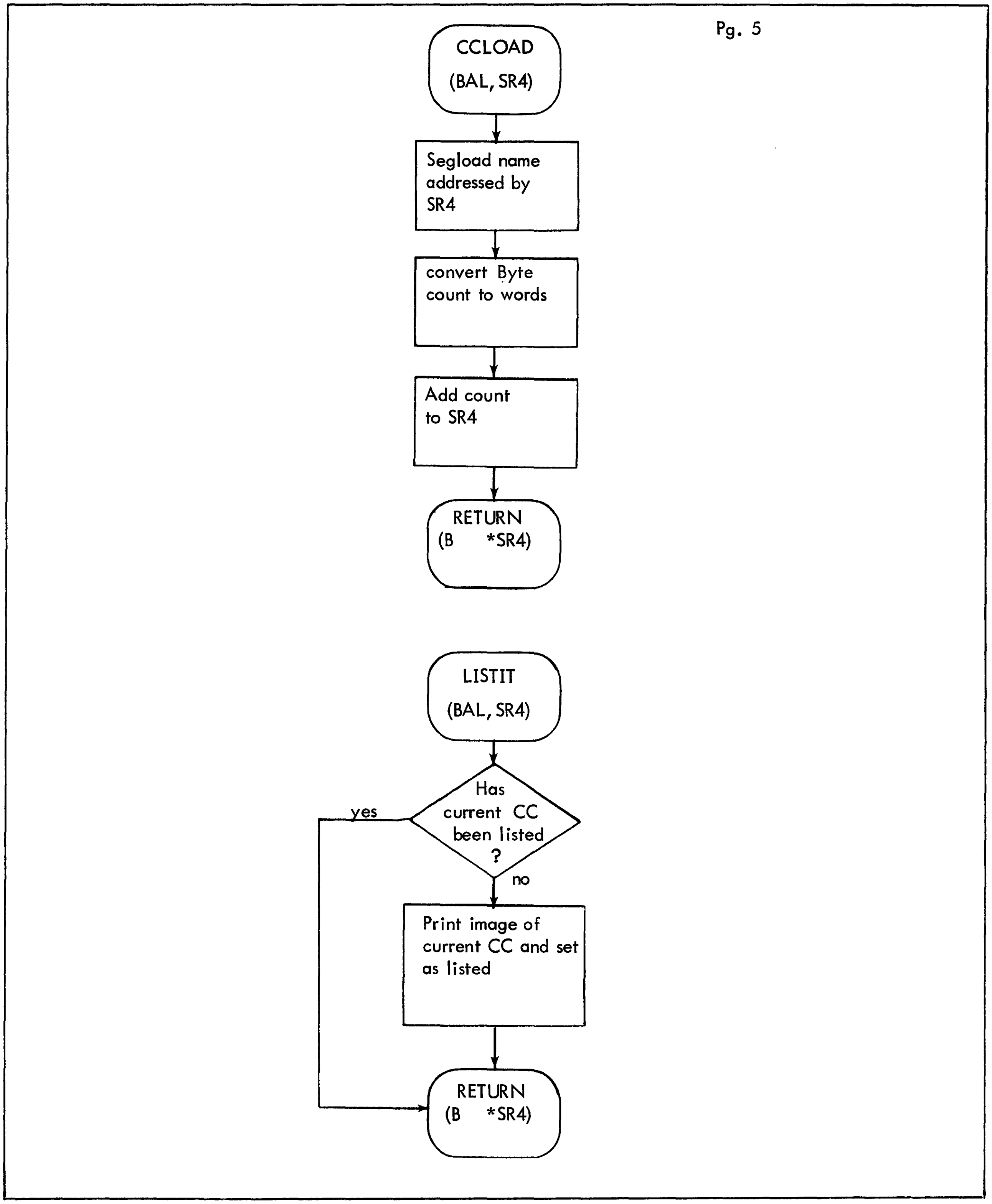

Pg. 5
CCLOAD
(BAL, SR4)
Segload name addressed by SR4
convert Byte count to words
Add count to SR4
RETURN
(B *SR4)
LISTIT
(BAL, SR4)
Has current CC been listed ?
yes
no
Print image of current CC and set as listed
RETURN
(B *SR4)

Figure 2-1 Flow Diagram of P2CCI (Cont.)

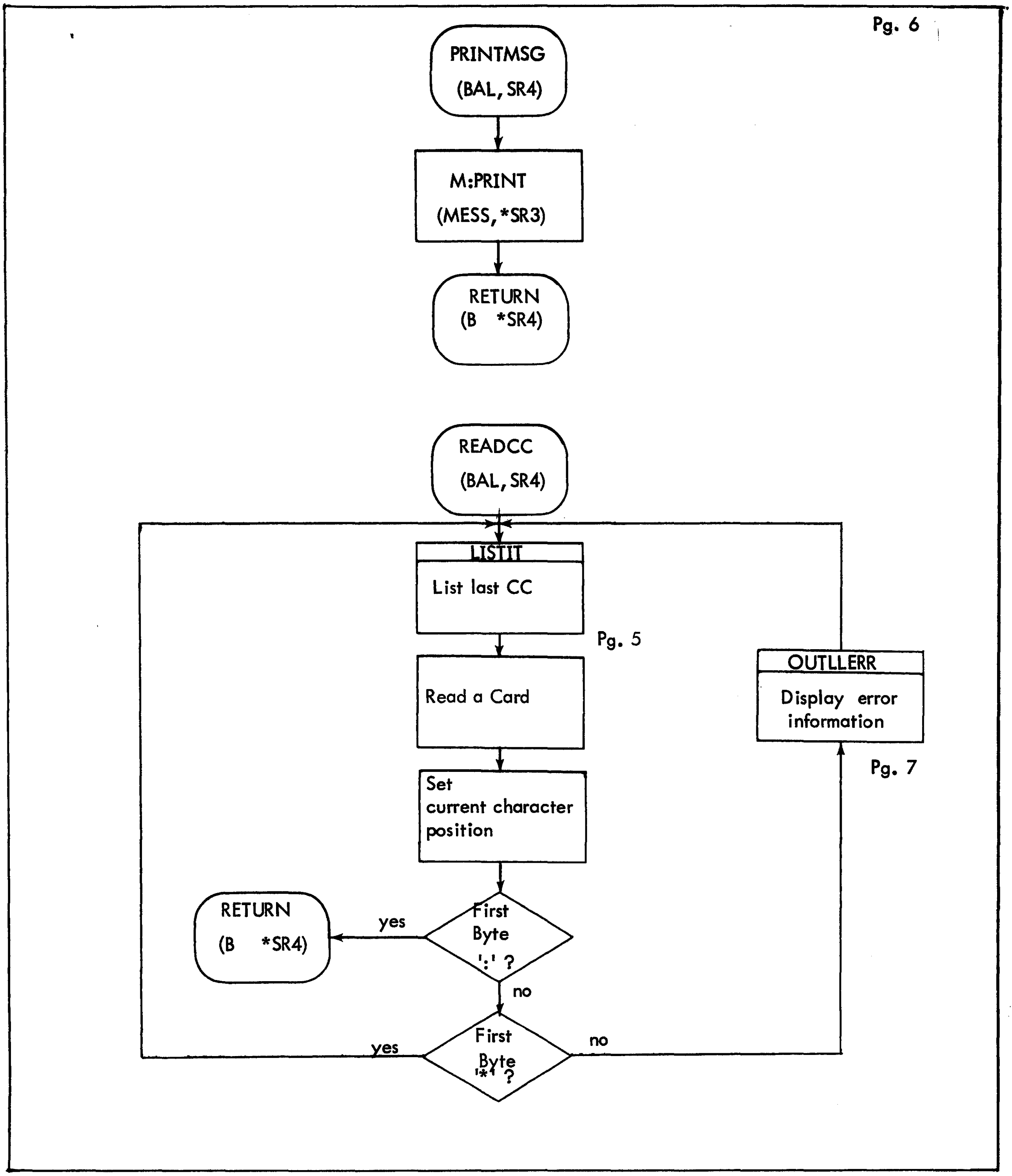


Figure 2-1 Flow Diagram of P2CCI (Cont.)

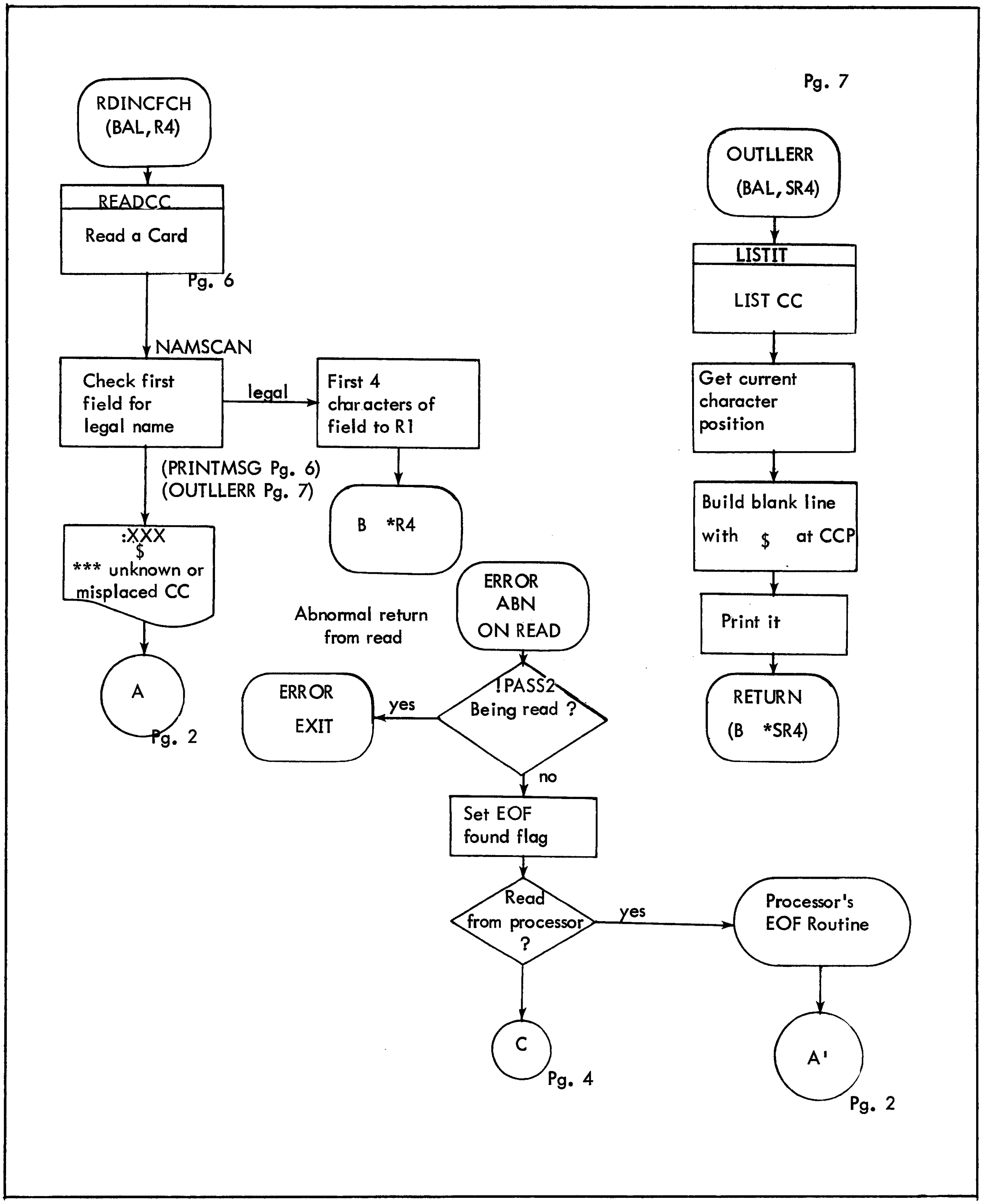


Figure 2-1 Flow Diagram of P2CCI (Cont.)

2.2 UBCHAN

2.2.1 Purpose

To process the PASS2 commands: :CHAN, :DEVICE, :STDLB and :OSTDLB (UTS) and build the load module IOTABLE containing the I/O tables for the monitor being generated. M:HGP load module is also generated containing the HGPs. The keyed file SPEC:HAND containing the device handler names is also built.

2.2.2 Usage

```
B     CHAN (BPM/BTM)
B     UBCHAN (UTS)
      with R1 = Type of control command
                1 for :CHAN
              = 4 for :STDLB
              = X'10000' for :OSTDLB

      :DEVICE cards require a preceding :CHAN and are always read by UBCHAN itself
      R0 = address of temp stack pointer
      R3 = base address of data in temp stack
      R7 = address of control card PLIST
```

2.2.3 Input

Control card images (:CHAN, :DEVICE, :STDLB, :OSTDLB)

2.2.4 Output

Display of control information to LL device
IOTABLE load module (See Table 2-3)
M:HGP load module contains all the HGPs
SPEC:HAND keyed file (see Chapter 6.3 for details).

Table 2-3. IOTABLE Load Module

Label	Entry Size (Words)	Length	Contents or Value	Target System
1. Tables				
IOTABLE (CLIST)	Variable	Variable	CLISTS for device Handlers (1/device)	Both
DCT1	1/2	#DEVICES+1	Device address (X'ndd')	Both
DCT2	1/4	#DEVICES+1	CIT index	Both
DCT3	1/4	#DEVICES+1	Bits 0-1 I/O operation Bits 6-7 access control key	Both
DCT4	1/4	#DEVICES+1	Device type index	Both
DCT5	1/4	#DEVICES+1	0	Both
DCT6	1/4	#DEVICES+1	0	Both
DCT7	1/2	#DEVICES+1	DW address of CLIST for device	Both
DCT8	1	#DECIVES+1	REF to Handler1 name	Both
DCT9	1	#DEVICES+1	REF to Handler2 name	Both
DCT10	1/2	#DEVICES+1	0	Both
DCT11	1	#DEVICES+1	0	Both
DCT12	1	#DEVICES+1	0	Both
DCT13	2	#DEVICES+1	0	Both
DCT14	1/4	#DEVICES+1	0 or 1 if dedicate	Both
DCT15	1/4	#DEVICES+1	0	Both
DCT16	2	#DEVICES+1	TAB YYNDD for Remote Batch devices N/L * RBndd	UTS
			N/L !!YYNDD	BPM/BTM
DCT17	1/2	#DEVICES+1	0	Both
DCT18	1/4	#DEVICES+1	0	Both
DCT19	1/4	#DEVICES+1	0	Both
DCT20	1/4	#DEVICES+1	0	Both
DCT21	1/2	#DEVICES+1	0	Both
DCT22	1/4	#DEVICES+1	contains disk type index 0 = not disk device 1 = 7204, 2 = 7232, 3 = 7212 4 = 7242, 5 = 7260, 6 = 7265	UTS

Table 2-3. IOTABLE Load Module (cont.)

Label	Entry Size (Words)	Length	Contents or Value	Target System
DCT23	1/2	#DEVICES+1	0 if entry not disk type device otherwise, contains displacement in words from HGP (DEF) to the in core HGP for the specific device.	UTS
DCT1P[†]	1/2	#DEVICES+1	X'ndd' from DCT1	Both
DCT1A[†]	1/2	#DEVICES+1	If pooled device X'ndd' where X'nd' specifies the secondary IOP/controller field from DUAL option; if device not pooled X'ndd' from DCT1.	Both
CIT1	1/4	#CHAN+1	0	Both
CIT2	1/4	#CHAN+1	0	Both
CIT3	1/4	#CHAN+1	0 Bit 4 = 1 if DUAL for given entry	Both
CIT4	1	#CHAN+1	0	Both
CIT5	1/4	#CHAN+1	0	Both
CIT6[††]	1/4	#CHAN+1	0	Both
OPLBTBL1	1/2	#OPLABELS+1	Text of OPLABEL	Both
OPLBTBL2	1/4	#OPLABELS+1	DCT index	Both
OPLBTBL3	1/4	#OPLABELS+1	DCT index	Both
OPLBTBL4	1/4	#OPLABELS+1	Bits 0-1 similar to DCT3 I/O flags	Both
OPLBTBL5	1/4	#OPLABELS+1	DCT index for OSTDLB	UTS
TYPMNE	1/2	#Device type mnemonics+1	Text of Mnemonic	Both
AVRTBL	2	#Tape devices +DP	0 for TAPE; for DP second word Bit 0 = Public/Private Bits 16-31 HGP displacement	Both
AVRID	1/2	# of tape devices+DP	0	UTS
SOLICIT	1/2	# of tape devices+DP	0	UTS
AVRNOU	1/2	# of tape devices +DP	0	UTS

[†] These tables generated only if DUAL specified on any :CHAN command, if not then labels equated to DCT1.

[††] CIT6 generated only if DUAL specified on any :CHAN command, if not, CIT6 is equated to CIT5.

Table 2-3. IOTABLE Load Module (cont.)

Label	Entry Size (Words)	Length	Contents or Value	Target System
AVRSID	1	#tape devices	0	UTS
HGP	variable	variable	HGPS (see Figure 2-8) Headers of public HGPs and HGPs for private dp	BPM/BTM UTS
ABSFDLL	1	1	ABSF disk address lower limit	BPM/BTM
ABSFDC	1	1	ABSF next disk address available	BPM/BTM
ABSFDUL	1	1	ABSF disk address upper limit	BPM/BTM
ABSFDISC	5	5	Word 0=DCT index of ABSF disk Word 1 = SS of ABSF disk Word 2 = NSPT of ABSF disk Word 3 = Max bytes per I/O call Word 4 = Max sectors per I/O call	BPM/BTM
BCHKLL	1	1	BCHK disk address lower limit	BPM/BTM
BCHKUL	1	1	BCHK disk address upper limit	BPM/BTM
BCHKDISC	5	5	Word 0=DCT index of BCHK disk Word 1 = SS of BCHK disk Word 2 = NSPT or BCHK disk Word 3 = Max bytes per I/O call Word 4 = Max sectors per I/O call	BPM/BTM
BCHKSIZ	1	1	Size (words) of BCHK area	BPM/BTM
BCHKFLG	1	1	0	BPM/BTM
BCHKCNT	1	1	0	BPM/BTM
IOCTQ	variable	#DEVICES+ #Tape+#DP+3	Word 0-1 stack pointer DW remainder is stack	Both
MB:GAM1[†]	1/4	#of disks with PSA	Granule address mask	UTS
MB:GAM2[†]	1/4	#of disks with PSA	(SGP words/granule position)-1	UTS
MB:GAM3[†]	1/4	#of disks with PSA	Shift for GSP index to granule position	UTS
MB:GAM4[†]	1/4	#of disks with PSA	Shift for track to granule address	UTS
MB:GAM5[†]	1/4	#of disks with PSA	Shift for disk address to track #	UTS

[†] Entries for these tables are determined by the disk type and number of tracks defined on the given disk. If the swapping device is a disk pack pseudo - 7232 type entries are made. See 2.2.7.1 for complete discussion.

Table 2-3. IOTABLE Load Module (cont.)

Label	Entry Size (Words)	Length	Contents or Value	Target System
MB:GAM6[†]	1/4	#of disks with PSA	Sector address mask	UTS
MB:GPT[†]	1/4	#of disks with PSA	Granules/Track	UTS
MB:SWAPS[†]	1/4	#of disks with PSA	Shift for granule position SGP index	UTS
MB:DWT[†]	1/4	#of disks with PSA	DW size of SGP	UTS
MB:SPACEJIT[†]	1/4	#of disks with PSA	Increment for spacing users around disk	UTS
MB:SDI	1/4	#of disks with PSA	Disk DCT index	UTS
M:GATLIM[†]	1	#of disks with PSA	Highest valid track	UTS
M:GASLIM[†]	1	#of disks with PSA	Highest sector position	UTS
M:ADRINCR[†]	1	#of disks with PSA	Increment to get from last Sector/Band to first sector next Band	UTS
M:SWPEND	1	#of disks with PSA	Address of the first sector on the next BAND/TRACK/CYLI following the PSA area data is in the format returned by a sense command	UTS
M:FREE#GRAN	1	#of disks with PSA	# of unused granules on swapping device, the first device entry = 0	UTS
M:SWAPD	1	#of disks with PSA	Disk device address	UTS
M:SNSDA	1	#of disks with PSA	0	UTS
M:HLTIC	1	# of disks with PSA	TIC to sense CDW	UTS
M:SGP	1	#of disks with	WA of granule pools (see Figure 2-8.1 for format of granule pools)	UTS
M:SBAND	1	#of disks with PSA	0	UTS

[†] Entries for these tables are determined by the disk type and number of tracks defined on the given disk. If the swapping device is a disk pack pseudo - 7232 type entries are made. See 2.2.7.1 for complete discussion.

Table 2-3. IOTABLE Load Module (cont.)

Label	Entry Size (Words)	Length	Contents or Value	Target System
M:JITPAGE	1	#of discs with PSA	0	UTS
M:CLBGN	1	#of discs with PSA	0	UTS
M:WCKBCL	1	#of discs with PSA	0	UTS
M:WCKECL	1	#of discs with PSA	0	UTS
MH:CLEND	1/2	#of discs with PSA	0	UTS
MH:LDA	1/2	#of discs with PSA	0	UTS
MB:#RTRY	1/4	#of discs with PSA	0	UTS
MB:SFC	1/4	#of discs with PSA	0	UTS
S:DP	1	1	1 if disc pack is swapping device, 0 if RAD is swapping device	UTS
S:CYLSZ[t]	1	1	The number of granules/physical cylinder	UTS
RBLIMS	2	2	If Remote Batch devices defined: word 0 = Lowest RB DCT index word 1 = Highest RB DCT index If no Remote Batch devices defined: word 0 = DCTSIZ+1 word 1 = DCTSIZ	Both
RB:FLAG[tt]	1	Highest RBT DCT1 index	Each RBT entry = 0 if HALF duplex Keyword, =X'8000' if FULL duplex Keyword, default is HALF	Both
WARBFLAG[tt]	1	1	Address of table points to first significant word of RB:FLAG	BPM/BTM
RBH:ACK[tt]	1/2	Highest RBT DCT1 index	Each RBT entry = 0	Both
RBB:SPC[tt]	1/4	Highest RBT DCT1 index	Each RBT entry = 0	Both

[t] Generated only when swapping device is disc pack.

[tt] Generated only if Remote Batch devices defined. Significant entries in tables only in the range of RBT DCT1 indices.

Table 2-3. IOTABLE Load Module (cont.)

Label	Entry Size (Words)	Length	Contents or Value	Target System
RBB:SFC[†]	1/4	Highest RBT DCT1 index	Each RBT entry = 0	Both
RBB:CPZ[†]	1/4	Highest RBT DCT1 index	Each RBT entry = X'50'	UTS
RBB:LPZ[†]	1/4	Highest RBT DCT1 index	Each RBT entry WIDTH parameter on PAPER Keyword option or default of X'80'	UTS
2. Absolute DEFs				
HGPSIZE	-	-	Total words of HGPs	BPM/TBM
DCN	-	-	DCTX of first disk	UTS
DCTSIZ	-	-	#of entries in DCT tables	Both
CITSIZ	-	-	#of entries in CIT tables	Both
NTYPMNE	-	-	#of type mnemonics	Both
BATAPE	-	-	DCTX of first tape or disk pack (BPM)	Both
NBATAPE	-	-	- BATAPE	Both
PSA	-	-	PSA size in tracks	BPM/BTM
OPLBTSIZ	-	-	#of entries in op label table	Both
AVRTBLSIZ	-	-	#of tape entries in AVRTBL	Both
AVRTBLNE	-	-	#of entries in AVRTBL (includes tape+DP)	Both
LSWAP	-	-	#of devices with PSA-1	UTS
CDP:NGC	-	-	#granules/logical cylinder	BPM/BTM
CDP:NCYL	-	-	If CDP:NGC =30, value is 399 ((12000/CDP:NSC) -1) If CDP:NGC =2, value is 5998 ((12000/CDP:NGC)-2)	BPM/BTM
BCRBFLAG[†]	-	-	Length in bytes of significant words of RB:FLAG	BPM/BTM
NUMRBTS	-	-	#RBTs defined	BPM/BTM
SC2SK[††]	-	-	((#granules per physical cylin-2)/4)*10	UTS

[†] Generated only if Remote Batch devices defined. Significant entries in tables only in the range of RBT DCT1 indices.

[††] Generated only when swapping device is disk pack.

2.2.5 Subroutines Used

NAMSCAN CHARSCAN CHSTSCAN HEXSCAN	Character scanning routines
ABNRETUR CLEARDYN OUTLLERR PRINTMSG RDINCFCH	Card reading and message printing routines
MODIFY MODGEN	Load module manipulator

2.2.6 Data Base

2.2.6.1 Temp Stack

When UBCHAN is called it allocates temp stack area as in Table 2-4. PLISTS are moved into position and pointers and addresses initialized via R3. The area from BASESTAC to P2DYNEND has already been initialized by P2CCI. See discussion of P2CCI for description of this part of data base (Table 2-1).

Table 2-4. PASS2 Stack Allocation

#TYPMNE	EQU	64	******#Entries in TYPMNE Table (Max)
TYPMNLNG	EQU	#TYPMNE	#TYPMNE entries
TYPMNEAD	EQU	P2DYNEND+1	Pointer to TYPMNE Table (=TYPMNE)
TYPMNESZ	EQU	P2DYNEND+2	#Entries in TYPMNE Table (CURRENT)
TYPMNE	EQU	P2DYNEND+3	TYPMNE Table (1/2 word/entry)
TYPMNEND	EQU	P2DYNEND+3+ (#TYPMNE/2)	End of TYPMNE Table
*******			****
#STDLB	EQU	128	******#Entries in STDLB Table (Max)
*			
* This table contains both STDLB & OSTDLB Info.			
STDLCDPT	EQU	TYPMNEND+0	Pointer to next STDLB Table entry
STDLADDR	EQU	TYPMNEND+1	Pointer to STDLB Table (=STDLCD1)
STDLCD1	EQU	TYPMNEND+2	STDLB Table (2 word/entry)

Table 2-4. PASS2 Stack Allocation (cont.)

STDLEND	EQU	TYPMNEND+2 + (#STDLB*2)	End of STDLB Table
******			****
#CHAN	EQU	32	******#Entries in CHAN Table (Max)
CHANPTR	EQU	STDLEND+0	Pointer to next CHAN Table entry
CHANADDR	EQU	STDLEND+1	Pointer to CHAN Table (=CHANTBL)
CHANTBL	EQU	STDLEND+2	CHAN Table (5 wrds/entry)
CHANEND	EQU	STDLEND+2 + (#CHAN*5)	End of CHAN Table
#DEVICE	EQU	96	******#Entries in DEVICE Table (Max)
DEVICDPT	EQU	CHANEND+0	Pointer to next DEVICE Table entry
DEVIADDR	EQU	CHANEND+1	Pointer to DEVICE Table (=DEVICD1)
DEVICD1	EQU	CHANEND+2	DEVICE TABLE (4 wrd/entry)
DEVIEND	EQU	CHANEND+2 + (#DEVICE*4)	End of DEVICE Table
DEVIXTRA	EQU	22	< Extra work area >
*******			****
#HANDLER	EQU	96	******#Entries in HANDLER Table (Max)
HANDTADR	EQU	DEVIEND + DEVIXTRA+0	Pointer to HANDLER Table (=HANDTABL)
HANDTABL	EQU	HANDTADR+1	HANDLER Table (4 wrds/entry)
HANDEND	EQU	HANDTADR+1 + (#HANDLER*4)	End of HANDLER Table
*******			****
#OPLB	EQU	128	******#Entries in OPLB Table (Max)
OPLBADDR	EQU	HANDEND+0	Pointer to OPLB Table (=OPLBTAB1)
OPLBSIZE	EQU	HANDEND+1	#Entries in OPLB Table (current)
OPLBTAB1	EQU	HANDEND+2	OPLB Table (1/2 word/entry)
OPLBEND	EQU	HANDEND+2 + (#OPLB/2)	End of OPLB Table
*******			****
#DCDP	EQU	32	******#Entries in DC/DP/CM Table (Max)
DCINPNTR	EQU	OPLBEND+0	Pointer to next DC/DP/CM Table entry
DCINADDR	EQU	OPLBEND+1	Pointer to DC/DP/CM TBL (=DCINTAB)
DCINTABL	EQU	OPLBEND+2	DC/DP/CM Table (21 words/entry)
DCINEND	EQU	OPLBEND+2 + (#DCDP*21)	End of DC/DP/CM Table
*******			****
DCT1PTR	EQU	DCINEND+0	Pointer to DCT1 Table
DCT2PTR	EQU	DCINEND+1	Pointer to DCT2 Table

Table 2-4. PASS2 Stack Allocation (cont.)

DCT3PTR	EQU	DCINEND+2	Pointer to DCT3 Table
DCT4PTR	EQU	DCINEND+3	Pointer to DCT4 Table
DCT5PTR	EQU	DCINEND+4	Pointer to DCT5 Table
DCT6PTR	EQU	DCINEND+5	Pointer to DCT6 Table
DCT7PTR	EQU	DCINEND+6	Pointer to DCT7 Table
DCT8PTR	EQU	DCINEND+7	Pointer to DCT8 Table
DCT9PTR	EQU	DCINEND+8	Pointer to DCT9 Table
DCT10PTR	EQU	DCINEND+9	Pointer to DCT10 Table
DCT11PTR	EQU	DCINEND+10	Pointer to DCT11 Table
DCT12PTR	EQU	DCINEND+11	Pointer to DCT12 Table
DCT13PTR	EQU	DCINEND+12	Pointer to DCT13 Table
DCT14PTR	EQU	DCINEND+13	Pointer to DCT14 Table
DCT15PTR	EQU	DCINEND+14	Pointer to DCT15 Table
DCT16PTR	EQU	DCINEND+15	Pointer to DCT16 Table
DCT17PTR	EQU	DCINEND+16	Pointer to DCT17 Table
DCT18PTR	EQU	DCINEND+17	Pointer to DCT18 Table
DCT19PTR	EQU	DCINEND+18	Pointer to DCT19 Table
DCT20PTR	EQU	DCINEND+19	Pointer to DCT20 Table
DCT21PTR	EQU	DCINEND+20	Pointer to DCT21 Table
DCT22PTR	EQU	DCINEND+21	Pointer to DCT22 Table
DCT23PTR	EQU	DCINEND+22	Pointer to DCT23 Table
DCT1PTR	EQU	DCINEND+23	Pointer to DCT1P Table
DCT1APTR	EQU	DCINEND+24	Pointer to DCT1A Table
DCTLAST	EQU	DCINEND+25	End DCT Tables
NDCTS	EQU	DCTLAST - DCT1PTR	# DCT Tables
*******			****
CIT1PTR	EQU	DCTLAST+1	Pointer to CIT1 Table
CIT2PTR	EQU	DCTLAST+2	Pointer to CIT2 Table
CIT3PTR	EQU	DCTLAST+3	Pointer to CIT3 Table
CIT4PTR	EQU	DCTLAST+4	Pointer to CIT4 Table
CIT5PTR	EQU	DCTLAST+5	Pointer to CIT5 Table
CIT6PTR	EQU	DCTLAST+6	Pointer to CIT6 Table
CITLAST	EQU	DCTLAST+7	End of CIT Tables
NCITS	EQU	CITLAST - CIT1PTR	#CIT Tables
*******			****
TYPINDX	EQU	CITLAST+1	Type Index to Reorder DCTs
DCTSIZE	EQU	CITLAST+2	# Entries in DCT Table
CLISTPTR	EQU	CITLAST+3	Pointer to CLIST Table
PACKRPTR	EQU	CLISTPTR	End of Tables to be packed
*******			****

Table 2-4. PASS2 Stack Allocation (cont.)

HEADADDR	EQU	PACKRPTR+1	Pointer to HEADER
TREEADDR	EQU	PACKRPTR+2	Pointer to TREE
RDEFADDR	EQU	PACKRPTR+3	Pointer to REF/DEF Stack
EXPRADDR	EQU	PACKRPTR+4	Pointer to Expr. Stack
RDICADDR	EQU	PACKRPTR+5	Pointer to REL·DICT·0
CLISTADR	EQU	PACKRPTR+6	Pointer to SECT·0
HEADLNG	EQU	PACKRPTR+7	HEADER SIZE
TREELNG	EQU	PACKRPTR+8	TREE SIZE
RDEFLNG	EQU	PACKRPTR+9	REF/DEF Stack size
EXPRLNG	EQU	PACKRPTR+10	Expr· Stack SIZE
RDICLNG	EQU	PACKRPTR+11	REL·DICT·0 SIZE
SECT0LNG	EQU	PACKRPTR+12	SECT·0 SIZE
*******			****
CLISWDCT	EQU	SECT0LNG+1	#Words in all command lists
HGP1PTR	EQU	SECT0LNG+2	Pointer to next HGP Table entry
HGP1ADDR	EQU	SECT0LNG+3	Pointer to HGP Table Base
LASTHGP1	EQU	SECT0LNG+4	Pointer to base of last HGP Table
***			
OPLBT1AD	EQU	LASTHGP1+1	Pointer to OPLBT1 Table
OPLBT2AD	EQU	LASTHGP1+2	Pointer to OPLBT2 Table
OPLBT3AD	EQU	LASTHGP1+3	Pointer to OPLBT3 Table
OPLBT4AD	EQU	LASTHGP1+4	Pointer to OPLBT4 Table
OPLBT5AD	EQU	LASTHGP1+5	Pointer to OPLBT5 Table
TYPMNPNT	EQU	LASTHGP1+6	Pointer to TYPMNE Table
AVRTBLAD	EQU	LASTHGP1+7	Pointer to AVR Table
AVRTBLSZ	EQU	LASTHGP1+8	AVF Table SIZE
FDABCHAD	EQU	LASTHGP1+9	Pointer BCHK/ABSF Info.
FDABCHSZ	EQU	18	
ABSFDLL	EQU	FDABCHAD+1	ABSF disk address lower limit
ABSFDC	EQU	FDABCHAD+2	ABSF Current disk address
ABSFDUL	EQU	FDABCHAD+3	ABSF disk address upper limit
ABSFDISC	EQU	FDABCHAD+4	ABSF DCT1 Index
BCHKLL	EQU	FDABCHAD+9	
BCHKUL	EQU	FDABCHAD+10	
BCHKDISC	EQU	FDABCHAD+11	
BCHKSIZ	EQU	FDABCHAD+16	
#SWAPDEVS	EQU	BCHKSIZ+1	Number of disks with PSA
SWAPPTR	EQU	BCHKSIZ+2	Pointer to SWAPPER Table area
SOLICIT	EQU	BCHKSIZ+3	Pointer to SOLICIT Table area

Table 2-4. PASS2 Stack Allocation (cont.)

AVRIDARA	EQU	BCHKSIZ+4	Pointer to AVRID Table area
AVRSID	EQU	BCHKSIZ+5	Pointer to AVRSID table
DCN	EQU	AVRSID+1	Index into DCT1 of 1-ST DC/DP/CM
BATAPE	EQU	DCN+1	Index into DCT1 of 1-ST 9T/7T or DP (BPM)
NBATAPE	EQU	DCN+2	Compliment of BATAPE
PSA	EQU	DCN+3	Perm Storage Size (Tracks)
******			****
LMODPLIS	EQU	PSA+1	Master PLIST
EXPRCALL	EQU	PSA+9	Expression (and MODGEN) PLIST
HEAD	EQU	PSA+19	Load Module HEAD
TREE	EQU	PSA+31	Load Module TREE
MAX00	EQU	PSA+43	End of SECT 00
RDEFCALL	EQU	PSA+44	REF PLIST
DEFCALL	EQU	RDEFCALL	DEF PLIST
DICTCALL	EQU	PSA+52	Relocation Dictionary PLIST
TREETOP	EQU	PSA+57	End of PLISTS
*******			****
RBLIMSPTR	EQU	TREETOP+1	Pointer to RBLIMS Table
RBFLAGPTR	EQU	TREETOP+2	Pointer to RB:FLAG
RBHACKPTR	EQU	TREETOP+3	Pointer to RBH:ACK
RBBSPCPTR	EQU	TREETOP+4	Pointer to RBB:SPC
RBBSFCPTR	EQU	TREETOP+5	Pointer to RBB:SFC
RBBCPZPTR	EQU	TREETOP+6	Pointer to RBB:CPZ
RBBLPZPTR	EQU	TREETOP+7	Pointer to RBB:LPZ
******			****
STACKEND	EQU	TREETOP+8	End of Stack
	*******		****
SDBEGIN	EQU	PACKRPTR	Base of available stack for SDEVICE
SDVEND	EQU	DYSTORND	End of available stack for SDEVICE
*******************	***********	********************	***********************************

2.2.6.2 Preliminary Tables

The UBCHAN processor contains several tables necessary to its operation. They are described below:

TYPCHARS - This is a list of the type mnemonics (YY or YYNDD) for standard devices. It is moved to the temp stack at TYPMNE before UBCHAN processing starts, and may be extended during processing as new types of devices are defined. The index into this table is used to index other tables and is referred to as the "device index".

HANDNAME - This table, indexed by device index, is a list of default handler names for standard devices.

IOFLOWACT - This table, indexed by device index, contains a code for the standard I/O flow of standard devices. INPUT = X'40' OUTPUT = X'80' I/O = X'C0'.

CLISTDAT - This table, indexed by device index, contains the standard CLIST length for standard devices.

DCDPCM - This table is a list of disc types.

DEFAULTS - This table, parallel to DCDPCM, contains the defaults of SIZE, SS, and NSPT for each disc type.

OPLBCHAR - This table is a list of the standard oplabels. It is moved to temp stack at OPLBTAB1 before UBCHAN processing begins and may be expanded as oplabels are defined during processing.

OPFLOWACT - This table, parallel to OPLBCHAR, contains the flow codes (as in IOFLOWACT above) for each oplabel.

STANDARD - This table contains the default assignment for all standard labels, both batch and on-line.

ABSFX1 and ABSFX2 - These are parallel tables containing values representing maximum bytes per I/O call to NEWQ used in generating words 3 and 4 in the table ABSFDISC and BCHKDISC.

2.2.6.3 Intermediate Tables

As control commands are processed UBCHAN stores its information in intermediate tables in the temp stack. This is required because in many cases final tables cannot be built until all the commands have been read. The intermediate tables are described below.

CHANTBL - The 5 word entries in this table contain device information for dual channel access and the start and end address for the DEVICD1 entries for the devices on this channel. (Described in Figure 2-2).

DEVICD1 - The entries in this table begin as 9 words and contain device information (Figure 2-3). Later the size of each entry is reduced to 3 words. (Figure 2-4). The rest of the information is moved to the DCINTBL entry if this device was a disc. One DEVICD1 entry is built for each :DEVICE command processed.

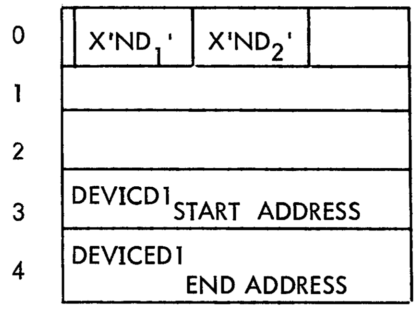
DCINTBL - The 21 word entries (only 5 of which are used) in this table contain disc information. It is moved to this table from the DEVICD1 table entry. One DCINTBL entry is formed for each :DEVICE command that defines a disc system (DC, DP, CM). (Described in Figure 2-5).

OPLBTAB1 - This table contains a list of monitor operational labels in text. It initially contains the standard oplabels and others are added as they are defined with :STDLB and :OSTDLB commands.

STDLCD1 - This table contains information defining operational labels derived from :STDLB and OSTDLB commands. The oplabel index is the index of the defined oplabel into OPLBTAB1. There are two formats since an oplabel may be assigned to a device or to another oplabel. (Described in Figure 2-6).

TYPMNE - This table contains a list of all device type mnemonics in text. Initially it contains the standard types from TYPCHARS, and more are added if they are defined with device cards.

HANDTBL - This table is a list of handler names for devices. As :DEVICE commands are read the handlers required for the devices defined have their names placed here. The names are in pairs, initial handler first then clean up handler, in TEXTC format. Housekeeping is done so that each pair of names appears only once. (Table 2-6).



Bit 0=1 if Ns are different

X'ND_1', X'ND_2' from dual option on :CHAN

Figure 2-2. CHANTBL Entry

0	NDD			Paper Size	
1	Type +1	Paper Width	X'NDD'		
2	*Flags	Handler Index			
3			'YY' Temporary		
4	DEVICD1 Index		CYLIN PRIV Type	NGC	Disk Devices Only
5	SS		NSPT		
6	SIZE		PSA		
7	PFA		PER		
8	BCHK		ABSF		

0		3	7
DE		DP	FLOW

Flags if set

Flow = 1 INPUT

= 2 OUTPUT

= 3 INPUT/OUTPUT

DE = DEDICATE

Bit 0 = 1

DP

Bit 3 = 1 if FULL DUPLEX

Figure 2-3. Initial DEVICD1 Entry

N	D	D	Paper Size
TYPMNE Index	Paper Width	Device Address	
***Flags	Handler Index	Y	Y

Figure 2-4. Final DEVICD1 Entry

0	DEVICD1 Index	CYLIN	PRIV	Type	FLINK
1	SS	NSPT			
2	SIZE	PSA			
3	PFA	PER			
4	BCHK	ABSF			

Figure 2-5. FINAL DCINTBL ENTRY

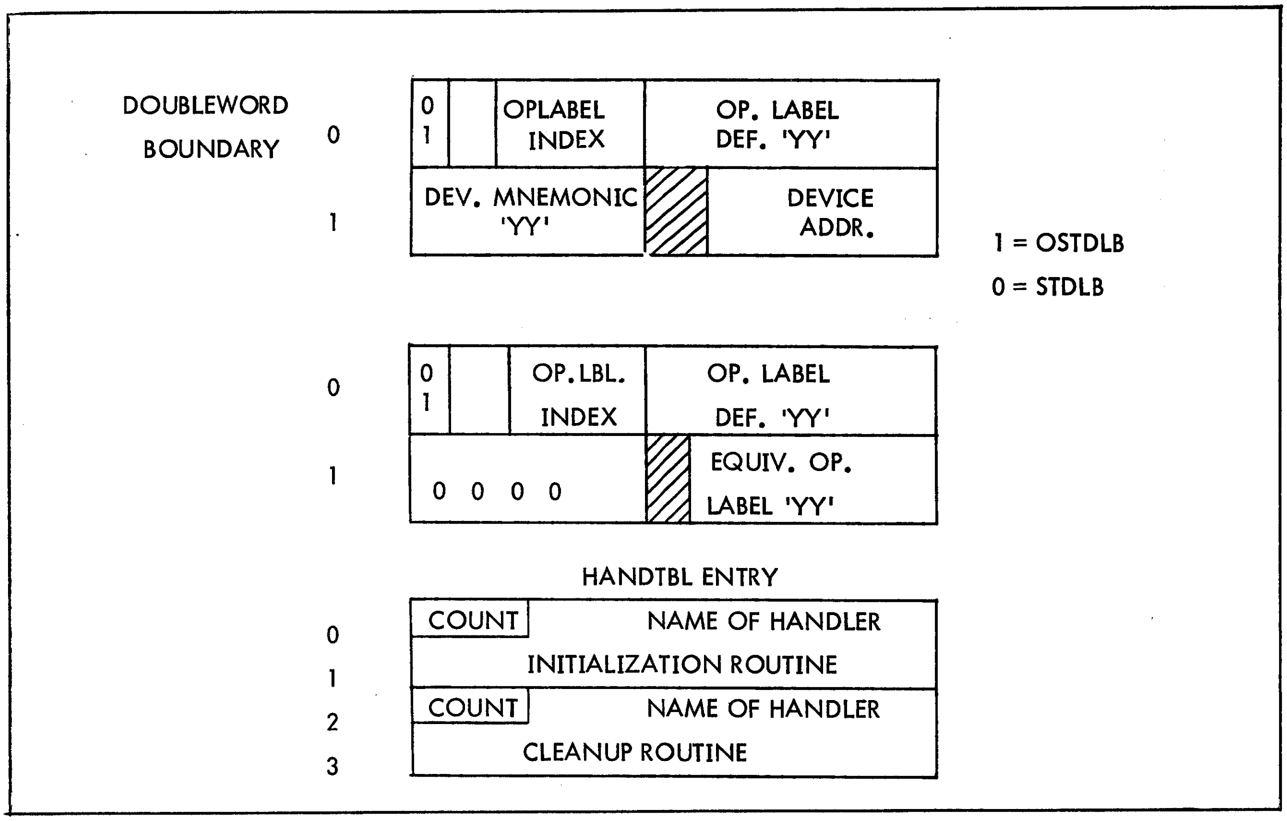


Figure 2-6. Two Types of STDLCD1 Entry and HANDTBL Entry

Table 2-5. STANDARD DEVICES

DEVICE	TYPCHARS	IO FLOWACT	HANDNAME	CLISTDAT
NO DEVICE	NO	*****ILLEGAL DEVICE DEFINITION *****		6
OPERATOR CONSOLE	TY	I/O	KBTIO, KBTCU	8
PAPERTAPE READER	PR	I	PTAP, PTAPCU	8
PAPERTAPE PUNCH	PP	O	PTAP, PTAPCU	8
CARD READER	CR	I	CRDIN, CRDINCU	2
CARD PUNCH	CP	O	CRDOUT, CRDDCU	74
LINE PRINTER	LP	O	PRTOUT, PRTCU	6
LINE PRINTER (7450)[1]	LP	O	PRTOUTL, PRTCU	6
LINE PRINTER (7446)[1]	LP	O	7446IO, 7446CU	6
RAD[3]	DC	I/O	DISCIO, DISCCU	6
9 TRACK TAPE[2]	9T	I/O	MTAP, MTAPCU	8
7 TRACK TAPE[2]	7T	I/O	7TAP, 7TAPCU	8
ANY TAPE	MT	*****ILLEGAL DEVICE DEFINITION*****		
DISC PACK (7242)[3]	DP	I/O	DPAK, DPAKCU	12
DISC PACK (7260/7265)[3]	DP	I/O	DISKAB, DSKABCU	12
REMOTE BATCH CONTROLLER	RB	I/O	DSCIO, DSCCU	10

DEVICE	TYPCHARS	IO FLOWACT	HANDNAME	CLISTDAT
COC CONTROLLER (BPM/BTM)[1]	CO	I/O	COC, COC	6
COC CONTROLLER (BATCH SWAPPING BTM)[1]	CO	I/O	COCBS, COCBS	6
COC CONTROLLER (UTS)	ME	I/O	COC, COC	6

1. User must specify handler names.
2. Inclusion of either tape handler causes additional module MAGTAPE to be added.
3. For UTS - if swapping device is RAD then module TSIO also included. If swapping device is Disk Pack then module DPSIO also included.

2.2.7 Description

UBCHAN is segloaded and entered from P2CCI when a :CHAN, :STDLB, or :OSTDLB (UTS only) control command is encountered. If P2CCI encounters a :DEVICE command, it is considered an error. An image of the command is passed in the read buffer to UBCHAN and a flag is set according to the type of command. From this point on UBCHAN reads its own input, a process which continues until a command which is neither :CHAN, :DEVICE, :STDLB, nor :OSTDLB is encountered. Upon entry UBCHAN first initializes its temp stack.

When UBCHAN reads a :CHAN command, it sets a CHAN encountered flag and calls CHANNEL to process the card. It then reads the next command.

When a :DEVICE command is encountered, UBCHAN checks to see if it was preceeded (not necessarily immediately) by a valid :CHAN. If it was not, an error message is printed and the command is ignored. Otherwise, the type mnemonic is checked. If it is "NO" or "MT" the command is in error. If it is a standard type mnemonic the index into the TYPMNE table is obtained. Otherwise, the new mnemonic is added to the TYPMNE table and the new index is noted. Whenever any table is expanded, it is checked for overflow and when this occurs error messages are printed.

Once the index of the type mnemonic has been obtained, it is used to build a DEVICD1 table entry for this device and all available default parameters are placed in it. A count is kept of the number of Remote batch devices. The rest of the command is then scanned for options, and as they are found, their values replace the defaults in DEVICD1. If the device was a disc, the disc options are also processed and when this is completed, the extra words of DEVICD1 are moved to DCINTBL. All options are checked for syntax and for legal values. When the DEVICD1 and DCINTBL entries are completed, the next card is read.

When a control command is a :STDLB or :OSTDLB command, an entry is added to STDLCD1 for each assignment on the card. In BPM/BTM, :OSTDLB cards are illegal while in UTS the high order bit of the first word of each entry is set to one for these assignments. The oplabels are checked to see if they are standard and if not, are added to the OPLBTAB1 table. When all assignments have been processed, the next card is read.

When UBCHAN reads a command that is not :CHAN, :DEVICE, :STDLB, or :OSTDLB or when an EOF is encountered during UBCHAN processing, the reading of control commands is stopped and UBCHAN begins to build its files. A check is made to determine whether CHAN/DEVICE information was read and if not UBCHAN exits. Otherwise, pages are obtained and allocated among the tables to be built. In allocating the DCT and CIT tables a check is made to determine if the DUAL flag is set (i.e., DUAL option was specified on any :CHAN command). If so, DCT1A, DCT1P and CIT6 are allocated space. If not, then the labels DCT1A and DCT1P are equated to DCT1, and CIT6 equated to CIT5.

The first CHANTBL entry is examined and the information in it and the DEVICD1 entries to which it points are used to begin building the DCT and CIT tables. When this CHANTBL entry is exhausted, the next is fetched and this process continues through all of CHANTBL. The CLIST area is also built at this time. Most devices have CLISTs of all zero and length dictated by CLISTDAT, although some devices have special CLISTs (See Figure 2-7).

While processing DEVICD1 entries when a remote batch device is first encountered, space is allocated for those RBT-dependent tables. Since table entries are significant only in the area of RBT indices, the pointers to the several tables are bumped back, overlapping other areas.

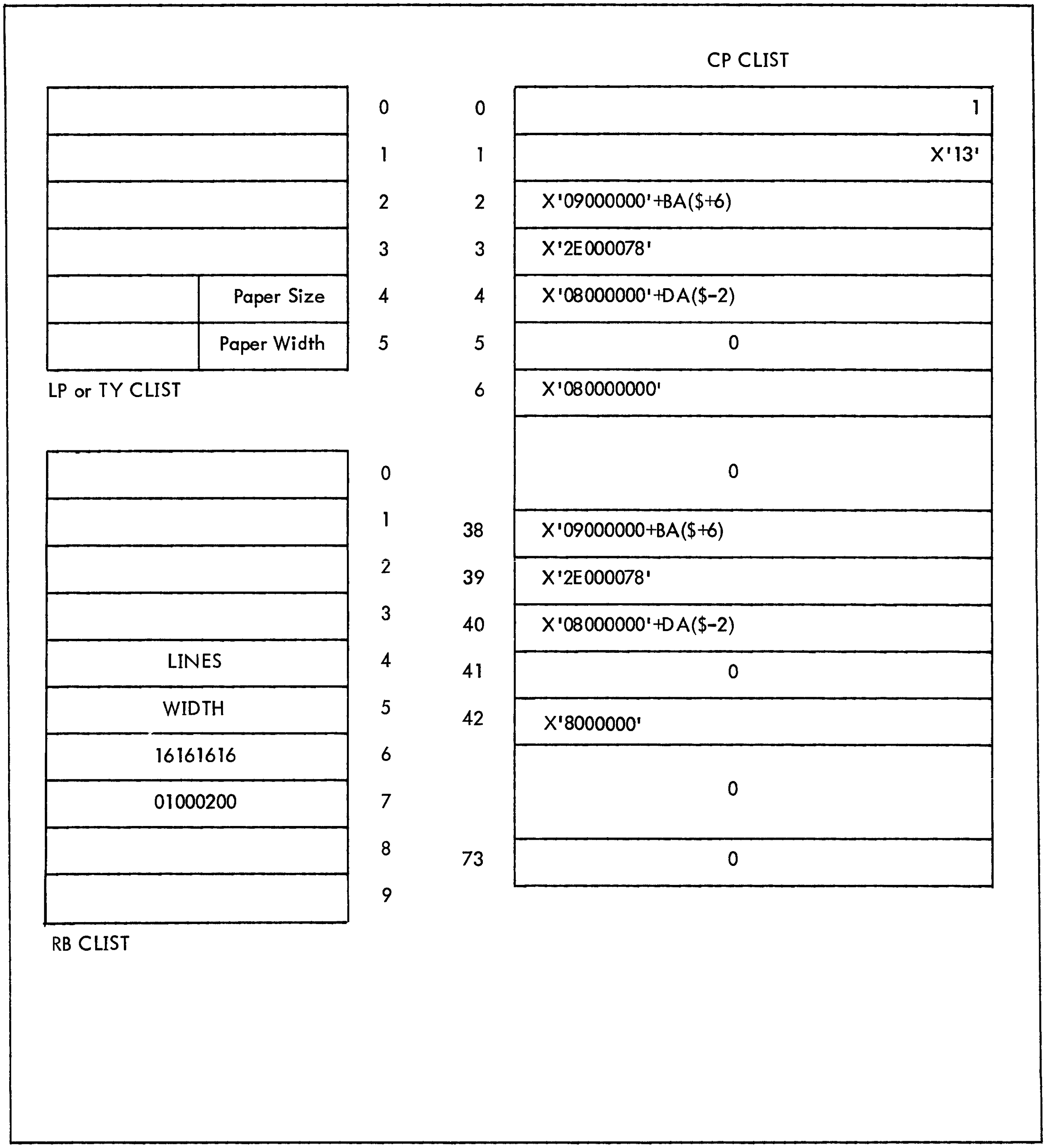
When the DCT and CIT tables (except for DCT22 and DCT23, – UTS only) have been built, RBLIMS table entries are defined. Then the DCINTBL is searched for a disk with PSA defined on it. The DCT 22 entry for the disk device is determined and stored in the appropriate location. Then if the swapping device (i.e., has PSA defined on it) for UTS is a disk pack, the #tracks of PSA is converted to #physical cylinders. If the #tracks does not equal a physical cylinder(s) the PFA and PER (if necessary) is decremented. The maximum number of PSA tracks that may be specified is X'3FC' for a 7242 and X'21C' for a 7260/7265.

The HGP for this device is generated and if the device is a disk pack, then an entry is made in the AVRTBL table. After the PSA discs are processed, DCINTBL is searched for other entries and the HGPs for them are generated (Figure 2-8).

UBCHAN then proceeds to build OPLBT1-4. For each operational label in OPLBTAB1 an entry is made in each OPLBT either from STDLCD1 or from STANDARD. Op labels assigned to "NO" are assigned to device NOA00 (DCTX=0). If this is for UTS, OPLBT5 is built in the same manner. When the oplabel tables are complete, TYPMNE is moved to its allocated area in the working pages. Then the area for the control task and temp stack are allocated. If the system is BPM/BTM, a copy of the HGPs are then written out as the load module M:HGP even though they are also included in the load module IOTABLE. UBCHAN then enters the completion routines to generate IOTABLE. If the system is UTS, the swap tables are generated (see 2.2.7.1) and then an area is allocated for the load module M:HGP which is created and written out to the disk. This load module contains the HGPs for all devices. In addition, the HGPs for private devices remain a part of IOTABLE as well as the 7 word headers of the HGPs for public devices. When M:HGP has been created, in a UTS system, the headers of the HGPs and the swap tables are squeezed into the area from where the full HGPs had been written. DCT23 entries are then computed and space is allocated for the AVRID and SOLICIT tables.

At this point the data (SECT 00) area of the load module IOTABLE is completed. Using MODIFY UBCHAN builds the head, tree, expression stack and REF/DEF stack for the module. MODGEN is used to generate the swapper tables. Finally RDICLIST is used to make final modifications to the relocation dictionary, and IOTABLE is written out via M:TM. Following this the handler names are written out into the SPEC:HAND file and UBCHAN exits.

If UBCHAN is unable to build IOTABLE, SPEC:HAND, or M:HGP, appropriate, definitive error messages are produced and PASS2 (being unable to continue without these vital tables) aborts.



LP or TY CLIST

		0
		1
		2
		3
	Paper Size	4
	Paper Width	5

RB CLIST

	0
	1
	2
	3
LINES	4
WIDTH	5
16161616	6
01000200	7
	8
	9

CP CLIST

0	1
1	X'13'
2	X'09000000'+BA($+6)
3	X'2E000078'
4	X'08000000'+DA($-2)
5	0
6	X'080000000'
	0
38	X'09000000+BA($+6)
39	X'2E000078'
40	X'08000000'+DA($-2)
41	0
42	X'8000000'
	0
73	0

Figure 2-7. Special CLISTS

0	FLINK					
1	0	DCT INDEX	CYL	PRIV	TYPE	NGC
2	No. SECTORS/TRACK					
3	NO. SECTORS/GRANULE					
4	PER MAPWL		PFA MAPWL			
5	NVAT					
	PER MAPWD		PER 1st SECT NO.			
6	PFA MAPWD		PFA 1st SECT NO.			
7	PFA BIT MAP (1 = AVAILABLE)					
N	PER BIT MAP (1 = AVAILABLE)					
	0	7 8	15 16	17	23 24	31

where

CYL indicates whether device is allocated by cylinder (bit 16=1) or granule (bit 16=0).

PRIV indicates whether device is private (bit 17=1) or public (bit 17=0).

NGC number of granules/logical cylinder, has meaning only if CYL set. The # granules/logical cylinder is a SYSGEN definable parameter. For UTS this may be 1≥n≥255. For RBM this value may be either 2 or 30.

TYPE contains device type (7=disk; B=disk pack).

PER/MAPWL/PFA/MAPWL contain the number of words in the bit map area for PER/PFA.

PER/MAPWD/PFA/MAPWD contain the word displacement from the start of this allocation table to the first word of the bit map for PER/PFA.

NVAT contains the next volumes cylinder 0 allocation table if PRIV is set.

Figure 2-8. ALLOCATION TABLE FORMAT (HGP)

2.2.7.1 Generation of Swapper Tables (UTS only)

Upon entry into the routine (ALLOSWAP) to generate the swapper tables, space is allocated first for each table. Then for the type of the given swapping device (of which there may be only one if it is defined on a disk pack) is determined. The type controls the values to be generated for the various tables. See Table 2.5.1 for the types and values being used. If the swapping device is a disk pack then pseudo-7232 values are used, that is the number of tracks of PSA specified for the 7242/7260/7265 is compared to the numbers of tracks for 7232 devices to determine the type. After building the tables, the granule pools or SGPs are generated. The starting address of each pool is generated as an entry in the M:SGP table. The SGP for the first swapping device has all bits set to 0. All subsequent pools have the bits set according to the type, See Figure 2.8.1 for the various types.

2.2.8 Error and Informational Messages

All messages are output on the LL device.

***STDLB ENTRY TABLE FULL

The STDLB control command information has overflowed the allocated area. Up to 128 standard label definitions or up to 32 unique operational labels are allowed. UBCHAN tries to continue.

***DEVICE ENTRY TABLE FULL

The DEVICE control commands have overflowed the allocated core area. Up to 96 devices may be defined. UBCHAN tries to continue

***TYPMNE ENTRY TABLE FULL

A maximum of 64 unique type mnemonics are accepted from DEVICE control commands, and more have been specified. UBCHAN tries to continue.

Table 2-5.1. Swapper Table Constants by Type

Table Name	7212 RAD Type 0	7232 RAD or 7242/7260/7265 Disk Pack		
		Type 1 (0-80)	Type 2 (81-100)	Type 3 (101-200
MB:GAM1	X'3F'	7	7	7
MB:GAM2	1	3	7	15
MB:GAM3	-1	-2	-3	-4
MB:GAM4	6	3	3	3
MB:GAM5	-7	-4	-4	-4
MB:GAM6	X'7F'	X'F'	X'F'	X'F'
MB:GPT	41	6	6	6
MB:SWAPS	0	1	2	3
MB:DWT	41	12	24	48
MB:SPACEJIT	7	1	1	1
M:GATLIM	X'3F'	X'7F'	X'FF'	X'1FF'
M:GASLIM	80	10	10	10
M:ADRINCR	X'2E'	4	4	4

Type	Device	PSA (Hex Tracks)
0	7212	0-40
1	7232 or 7242/7260/ 7265	0-80
2		81-100
3		101-200

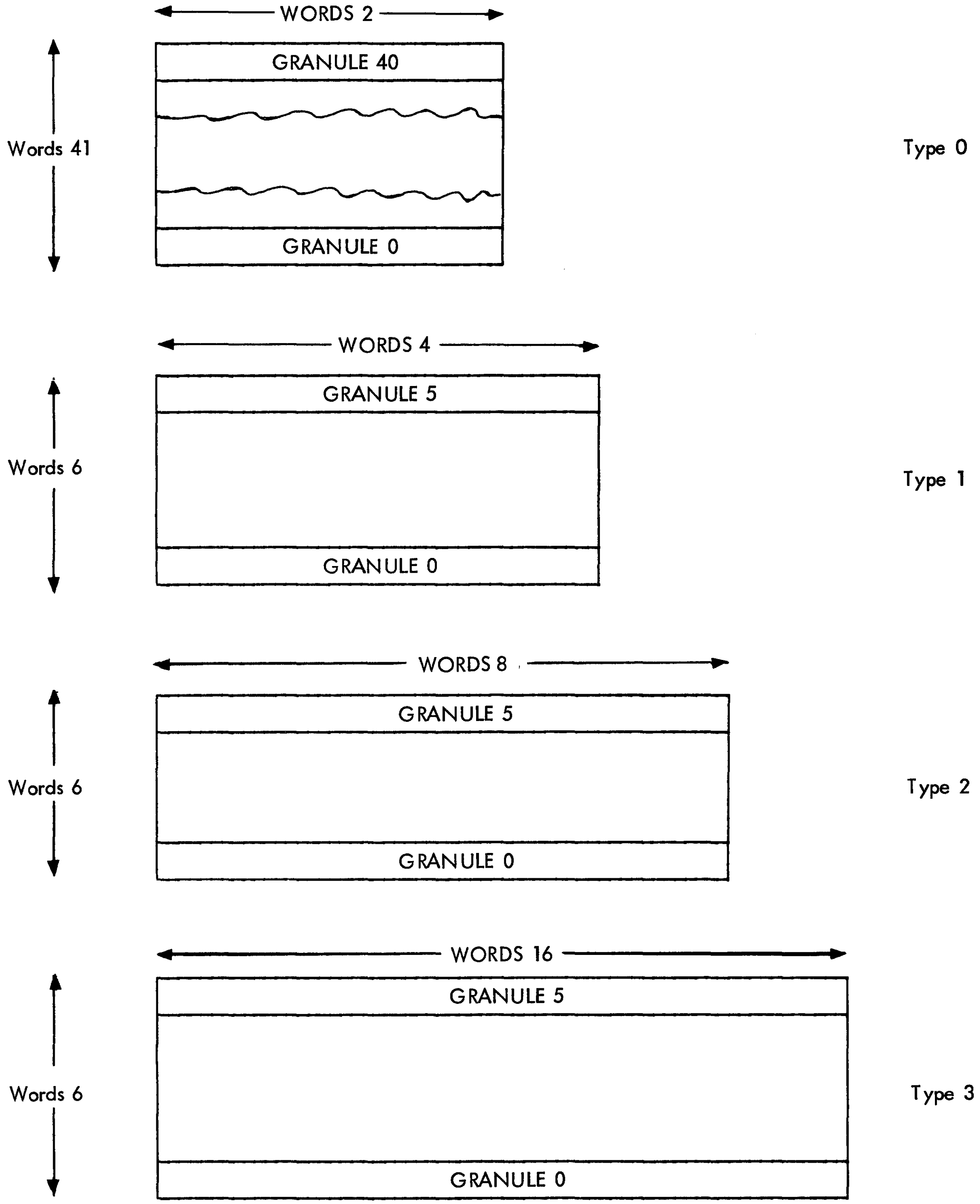


Figure 2-8.1 SGP Format and Contents by Type

Type 0

Location	Contents
Every 4th DW from 0 to 40	X'1111111111111111'
Every 4th DW from 1 to 37[t]	X'8888888888888888'
Every 4th DW from 2 to 38	X'4444444444444444'
Every 4th DW from 3 to 39	X'2222222222222222'
[t]Doubleword 37 is	X'8888888808888888'

Type (Value in Doublewords)

1	2	3	Contents
0-1	0-3	0-7	X'5555555555555555'
8-9	16-19	32-39	X'5555555555555555'
4-5	8-11	16-23[tt]	X'AAAAAAAAAAAAAAAA'
2-3	4-7	8-15	0
6-7	12-15	24-31	0
10-11	20-23	40-47	0
[tt]Doubleword 23 is			X'AAAAAAAA2AAAAAAA'

Figure 2-8.1 SGP Format and Contents by Type (Cont.)

***DISC ENTRY TABLE FULL

The DEVICE control commands defining disc units (i.e., YYNDD is of DCndd, DPndd, and CMndd types) have overflowed the allocated core area. Up to 32 discs may be defined. UBCHAN tries to continue.

***HANDLER CLIST FULL

When generating the CLIST (peripheral command list area) tables, the core area allocated is not large enough. Up to 64 handler definitions are allowed. UBCHAN aborts.

***DCT TABLE FULL

When generating the DCT tables (peripheral device information tables), the core area allocated is not large enough. UBCHAN aborts.

***HGP TABLE FULL

When generating the HGP tables for disc, disc pack or cram devices, the allocated core area is not large enough. UBCHAN aborts.

***OPLB XX EQUIVALENT YY MISSING

STDLB control command specifies that an operational label (XX) standard assignment is to another operational label (YY) that has not been defined. UBCHAN tries to continue.

***UNKNOWN DEVICE YYNDD (for LL)

The YYNDD field of a DEVICE control command is invalid, (i.e., bad syntax) or for the STDLB control command, the YYNDD referenced has not been defined by a DEVICE control command. UBCHAN tries to continue.

***INSUFFICIENT PAGES AVAILABLE

When core is being allocated for the generation of the load module, the available core is not large enough for the required allocation. UBCHAN aborts.

***ONLY XXXX PAGES OBTAINED

This message appears immediately after the preceding message. XXXX is the number of pages that was available to build the load module. UBCHAN aborts.

***LOAD MODULE CANNOT BE GENERATED

This message is produced in conjunction with the two preceding messages. UBCHAN aborts.

***SPEC:HAND CANNOT BE GENERATED

An inconsistency has occurred in building the HANDLERS record of the SPEC:HAND file. UBCHAN aborts.

***PASS2 UNABLE TO CONTINUE

This message is produced after any of the messages in which the explanation indicates that UBCHAN aborts.

***NO DISC DEFINED

This message is the result of no disc being defined by a DEVICE control command. This is only an informational message.

***NO HANDLER NAME GIVEN

When a device is being defined whose type mnemonic is unknown to PASS2, the HANDLER option must be present. UBCHAN continues to next control command.

***DEVICE TYPE YY ILLEGAL

A DEVICE control command YYNDD field contains "NO" or "MT" as its YY. UBCHAN tries to continue.

***PSA/PER INVALID ON CYLIN ALLOCATED DEVICE – PSA/PER IGNORED

An attempt has been made to define PSA/PER on a device that is allocated in logical cylinders. The options are ignored. UBCHAN continues.

***NO PSA DEFINED

***NO PER DEFINED

No PSA/PER has been defined on any disc. UBCHAN tries to continue.

***SYNTAX ERROR DUAL OPTIONS USED

The closing double parenthesis on the DUAL option of the CHAN command are in error. However, the preceeding option has been correctly processed and is used. UBCHAN continues.

***ONLY PFA VALID ON PRIVATE DEVICES

An attempt has been made to define other than PFA on a private device. All other allocations are zeroed out. UBCHAN continues.

SS AND NSPT MUST BE NONZERO – SET TO XXXX DEFAULTS

The options SS and/or NSPT have not been specified nor has the device type been specified (i.e., 7204, 7212, 7232, 7242, 7260, and 7265). The default (XXXX) for the particular type of target system (UTS-7232, BPM/ BTM - 7204) is substituted. UBCHAN continues.

***VALID 'CHAN' CC MUST PRECEDE 'DEVICE' CC

A DEVICE control command is encountered without being preceded by a valid CHAN control command. UBCHAN continues to the next control command.

***'NAME' OR SYNTAX INVALID

A CHAN control command option field has a syntax error or the DEVICE control command contains a syntax error or invalid name for HANDLER option. UBCHAN tries to continue.

***CHAN TABLE FULL

The CHAN control command has overflowed the allocated core area. Up to 32 :CHAN commands are allowed. UBCHAN tries to continue.

***NO CHAN/DEVICE INFO

No CHAN and DEVICE control commands have been encountered, although STDLB control commands have been processed. UBCHAN aborts.

***NO DEVICE FOR CHAN

A CHAN control command has been encountered without having any DEVICE definitions for this channel. UBCHAN tries to continue.

***'ABSF'/'BCHK' PREVIOUSLY DEFINED

A DEVICE control command has defined ABSF and/or BCHK and they have also been defined previously. UBCHAN continues to the next control command.

***SUM OF PSA+PER+PFA+BCHK+ABSF>SIZE

This warning message appears if there is a conflict in the summation of the given list of variables and the defined disc size. The message may appear several times for a given disc, i.e., if the conflict is determined after the summation of PSA+PER, then the message appears for this summation and once for each of the remaining summations, the overflowing value is replaced with the reamining SIZE and the processor continues.

***THIS DISC ALREADY DEFINED

A DEVICE control command is defining a disc, cram or disc pack device (i.e., YYNDD) which has already been defined. UBCHAN will try to continue.

***SYNTAX ERROR

A syntax error has been encountered on a control command. UBCHAN continues.

***NO DEVICE FOR TYPMNE YY (OPLBL=LL)

A operational label is assigned (or defaulted) to a device that has not been defined. In UTS, these messages have the headings:

---BATCH (STDLB)---for batch and

---ON LINE (OSTDLB)---for on-line.

They are printed at the end of UBCHAN processing.

***HGP CANNOT BE FORMED FOR YYNDD

The DEVICE control command defining this disc did not provide enough information to generate an HGP. UBCHAN continues.

***CYLIN VALUE INVALID--VALUE IGNORED

For BPM/BTM only, the value for number of granules per logical cylinder on the CYLIN option was not 2 or X'1E'. The value is ignored and UBCHAN continues.

***NGC=2 FOR ALL PRIVATE PACKS

***NGC=30 FOR ALL PRIVATE PACKS

For BPM/BTM only, one of the above messages always generated after the first, and only the first, command defining a private pack. All subsequent private packs are allocated in terms of the first defined pack regardless of what value is used on the subsequent commands. UBCHAN continues.

***NGC=30 FOR ALL PUBLIC PACKS

For BPM/BTM only, an attempt has been made to define public packs in terms of two granules/logical cylinder. The correct value is substituted and UBCHAN continues. This message is produced only once even if several public device CYLIN options are in error.

***PSA VALUE TOO LARGE – MAX VALUE USED – PFA INCREMENTED

For UTS only, in an attempt to define PSA on a disc pack, the number of tracks specified exceeded X'3FC' for a 7242 or X'21C' for a 7260/7265. The correct value is substituted and the extra tracks added to the total number of PFA tracks defined. UBCHAN continues.

***PSA INCREMENTED FOR DP SWAPPER – PFA/PER DECREMENTED

For UTS only, the number of tracks defined for PSA on a disc pack is not equal to a physical cylinder(s) i.e., evenly divisible by 20 (#tracks/physical cylinder). The number of tracks of PSA is incremented and PFA and PER, if necessary, are decremented. UBCHAN continues.

***PSA MUST BE 7212/7232/DISC PACK – PSA IGNORED

For UTS only, an attempt was made to define PSA on other than the above devices. The option is ignored. UBCHAN continues.

***PSA PREVIOUSLY DEFINED ON DP – PSA IGNORED

For UTS only, if PSA is defined on a disc pack it may not also be defined on a RAD or more than one disc pack. An attempt to do so has been detected. The option is ignored and UBCHAN continues.

***PSA DEFINED ON RAD, NOT ALLOWED ON DP

For UTS only a RAD has previously been defined with PSA on it. Therefore, it may not also be defined on a disc pack. The option is ignored and UBCHAN continues.

***NGC>255-- 55 USED FOR 7260/7265
***NGC>255--30 USED FOR 7242

For UTS only, the value specified for the number of granules per logical cylinder option on the CYLIN parameter is >255, the appropriate default is used and one of the above messages is generated. UBCHAN continues.

2.2.9 Major Internal Routines

UBCHAN/CHAN	Main entry, initialization, control
CHANNEL	Processes :CHAN command, builds CHANTBL entry
IODEFRD	Sets flag and reads next card and branches to appropriate routine.
IODDEVIC	Processes :DEVICE command gets YYNDD and finds type mnemonic
TYPFOUND	Gets type mnemonic index, puts default entries in DEVICD1
DEVOPTPA	Processes other options on :DEVICE command, stores valid options in DEVICD1.
DEVCDOUT	Completes building of DEVICD1 entry. If tape, increments AVRTBL size. Checks options if disc for validity and moves correct values to DCINTBL.
IOSTDLB/IOSTDLBO	Process :STDLB, :OSTDLB command. Gets options and stores in STDLCD1.
UBENDITALL/ENDITALL	Entry point when all commands for UBCHAN read. Allocates space for DCT, CIT, OPLBT, TYPMNE, and AVRTBL
GNDCTCIT	Generates DCT and CIT tables.
SETDCT1	Generates CLISTs for devices.
CPCLIST	Generates special CP and RB CLISTs.
GENCIT	Generate special setting in CIT3 if DUAL option on :CHAN command.
SRCHPSA	Searches for disk with PSA defined on it.
SRCHNXT	Checks for end of disk information. If not, then links to next DCINTBL when all PSA disks completed.
HGPENSET	Initial setting of HGP values. Makes entry in AVRTBL if disk pack.
NOAVRENT	If ABSF, BCHK areas defined, generates appropriate values.
FORMHGP	Gets number of tracks of PER, PFA BALs to SETGRANO and stores displacements in HGP.

CNVTCYL

For UTS systems in which PSA defined on a disc pack. First, checks that PSA ≤ X'BFC' for 7242 or ≤ X'21C' for 7260/7265. Then converts #tracks to # of physical cylinders.

SETGRANO

Builds HGP bit maps for PFA, PER upon entry

R6 = # sectors/track
R7 = # sectors/granule
R11 = DCINTBL WD8 addr, WD7 addr.
R12 = PSA, PSA+PFA addr.
R15 = PFA, PER addr.

This routine entered twice. In the above, the value preceeding the comma is the first entry value and that after the comma the second.

OPLBTYPM

Main entry to building OPLBT 1-5. Initialize pointers and check if any :STDLB or :OSTDLB commands.

DOSTAND

Checks STANDARD table for default Op label.

CHKSTDL

Searches STDLCD1 for Op label.

CHKASGN

Determines type of Op label assignment if it is to Oplabel then searches table for YY assignment.

OPLBNDD

Entered when oplabel assigned to device. Stores Op label in OPLBT1 and searches DCT1 for device address.

SETOPLBN

Put DCTX of device for Op label in OPLBT2 and OPLBT3. Stores flow in OPLBT4.

SETOPLB5

Generates value for OPLBT5 for UTS system only.

OPLBENDX

Finishes generating OPLBT1-5 by storing 'NO' in OPLBT1 entry 1 and sets DCT3 entry to 'NO'.

XFERTYPM

Transfers TYPMNE to SECT 00 area.

XFERFDAB

Transfers ABSF, BCHK values to SECT 00 area for BPM/BTM.

ALLTMP

Allocates area for control task and temp stack. If BPM/BTM at end sets up to HGPs as load module and branches to WRTHGP.

SWAPSET

Builds Swap tables for UTS system.

HGPSTAK/WRTHGP

Build, write M:HGP load module via M:TM.

HMOVIT

Packs headers of HGPs and Swap tables into area from which full HGPs were written (UTS only).

LOADMODL

Sets up IOTABLE load module.

GETMODFY
EQUMODFY

These routines change the MODIFY. PLIST before entering SETMODFY.

SETMODFY

This routine calls MODIFY to manipulate the load module being created.

GENEXP

Sets up the name pointed to for a call to SETMODFY placing the name in IOTABLE's expression stack and REFing it. The routine is used for handler names.

RDICLIST

Changes the relocation dictionary for special card punch CLIST words.

RDEFSWAP — Using MODGEN to set up changes to PLIST and call MODIFY, generates the Swap tables and remote batch tables.

NOSWAP — Final set up for IOTABLE includes opening file.

WRITELM — Writes IOTABLE file using M:TM then closes and saves file. Writes out SPEC:HAND file and returns to READOK in P2CCI.

OUTOFIT — Prints unable to continue message and does an error exit from PASS2.

2.2.10 Flow Chart

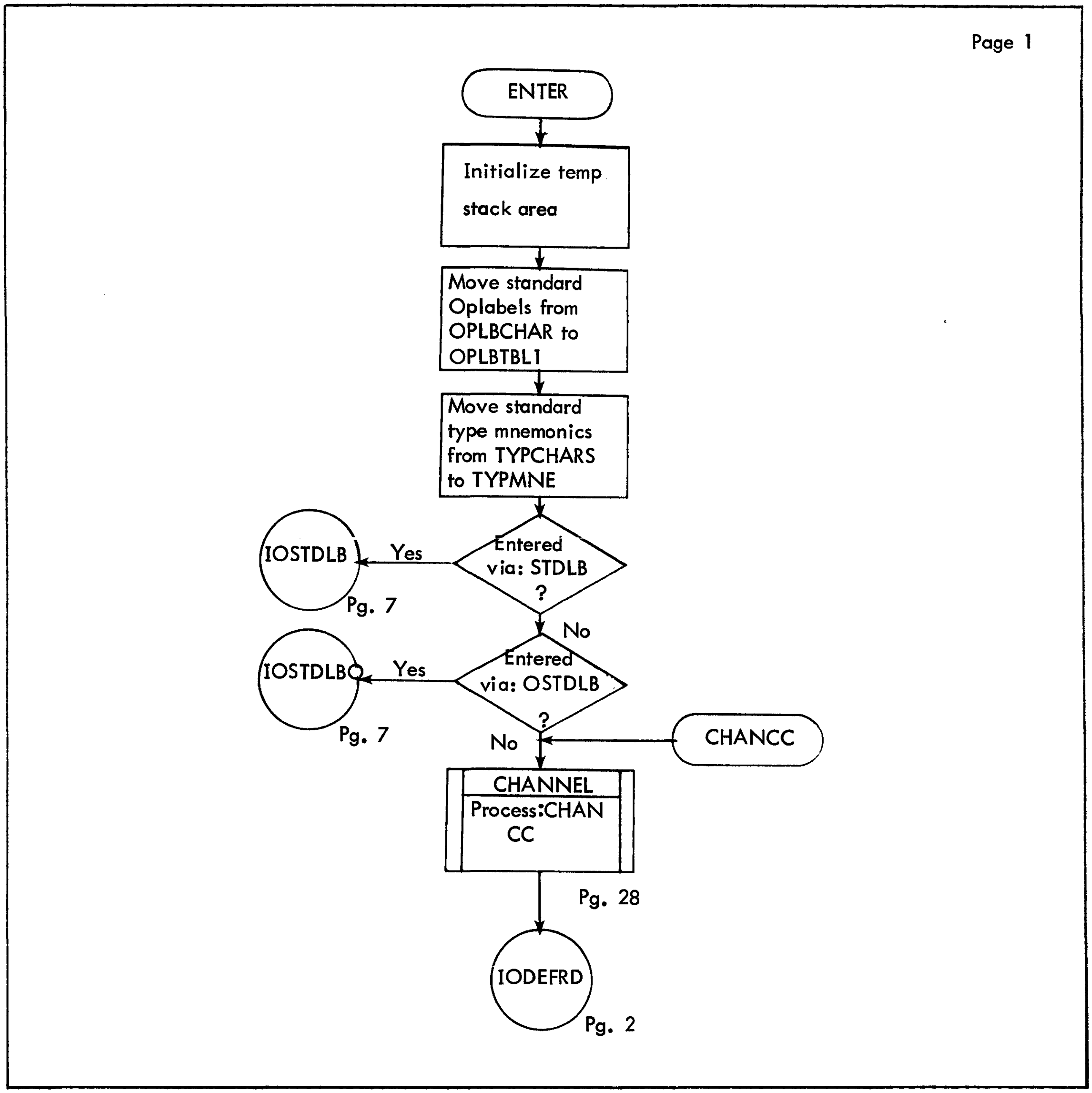


Figure 2-9. Flow Diagram of UBCHAN

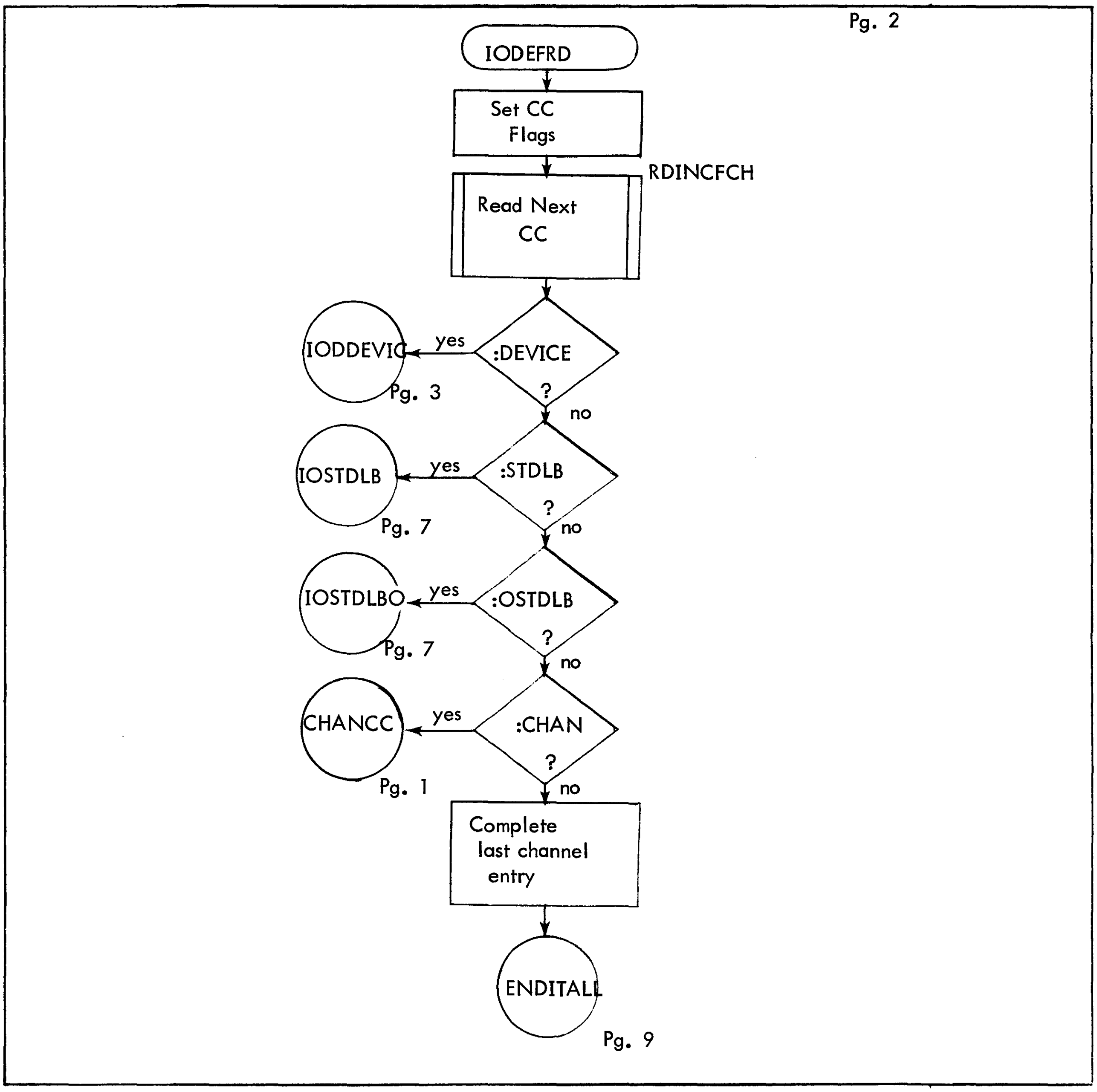


Figure 2-9. Flow Diagram of UBCHAN (Cont.)

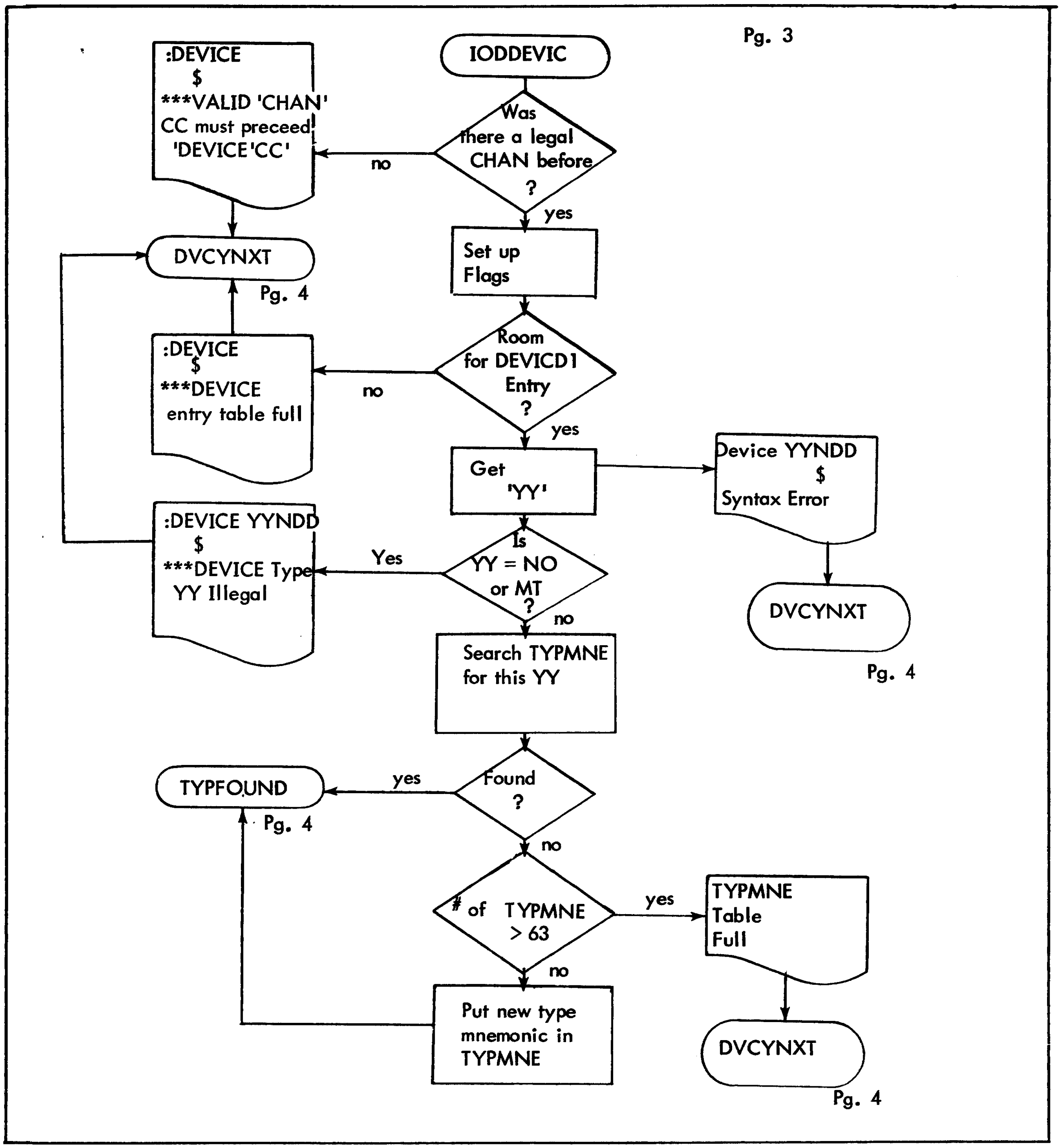
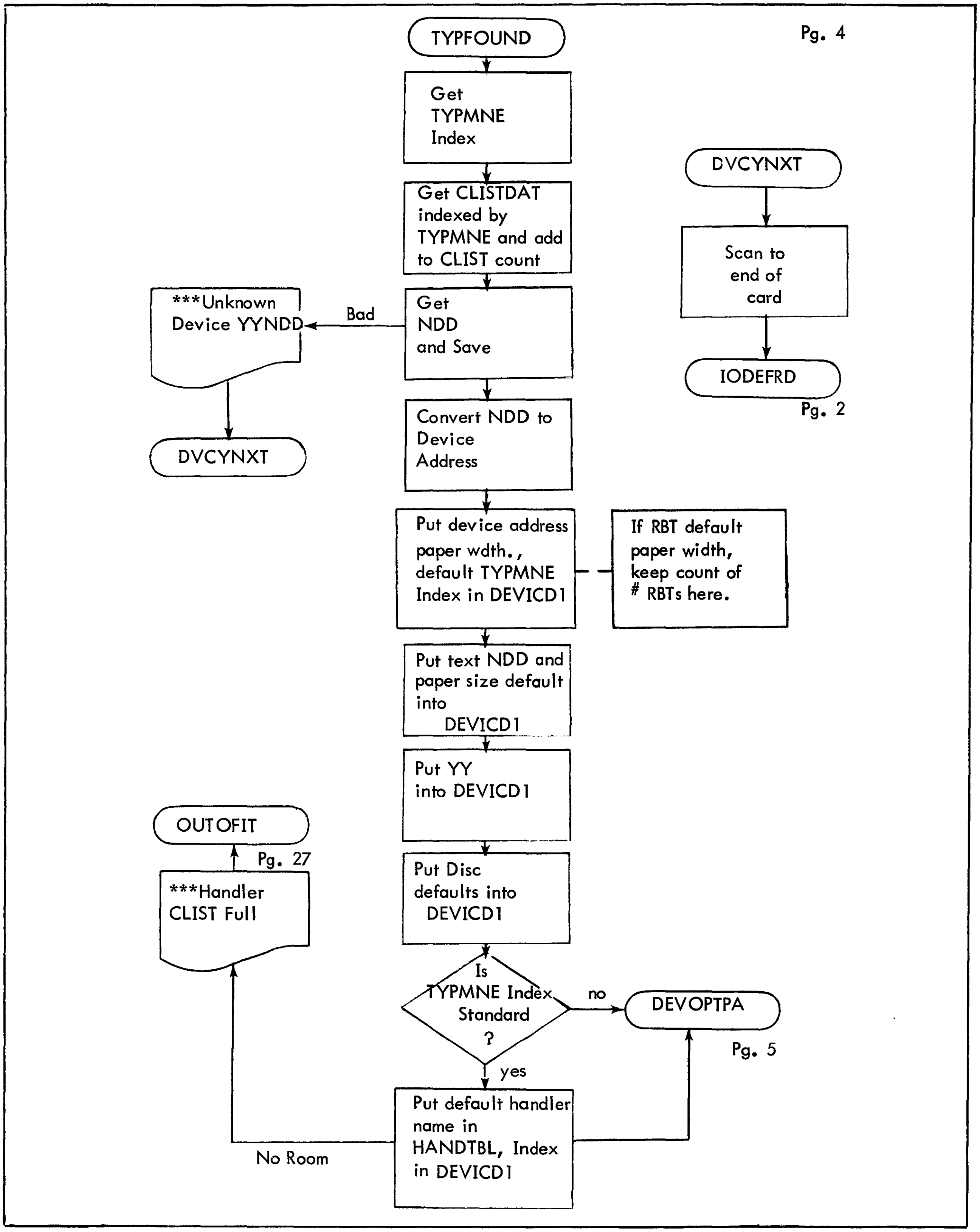


Figure 2-9. Flow Diagram of UBCHAN (Cont.)



TYPFOUND
Pg. 4
Get TYPMNE Index
Get CLISTDAT indexed by TYPMNE and add to CLIST count
Get NDD and Save
Bad
***Unknown Device YYNDD
DVCYNXT
Convert NDD to Device Address
Put device address paper wdth., default TYPMNE Index in DEVICD1
If RBT default paper width, keep count of # RBTs here.
Put text NDD and paper size default into DEVICD1
Put YY into DEVICD1
Put Disc defaults into DEVICD1
Is TYPMNE Index Standard ?
no
DEVOPTPA
Pg. 5
yes
Put default handler name in HANDTBL, Index in DEVICD1
No Room
***Handler CLIST Full
OUTOFIT
Pg. 27
DVCYNXT
Scan to end of card
IODEFRD
Pg. 2

Figure 2-9. Flow Diagram of UBCHAN (Cont.)

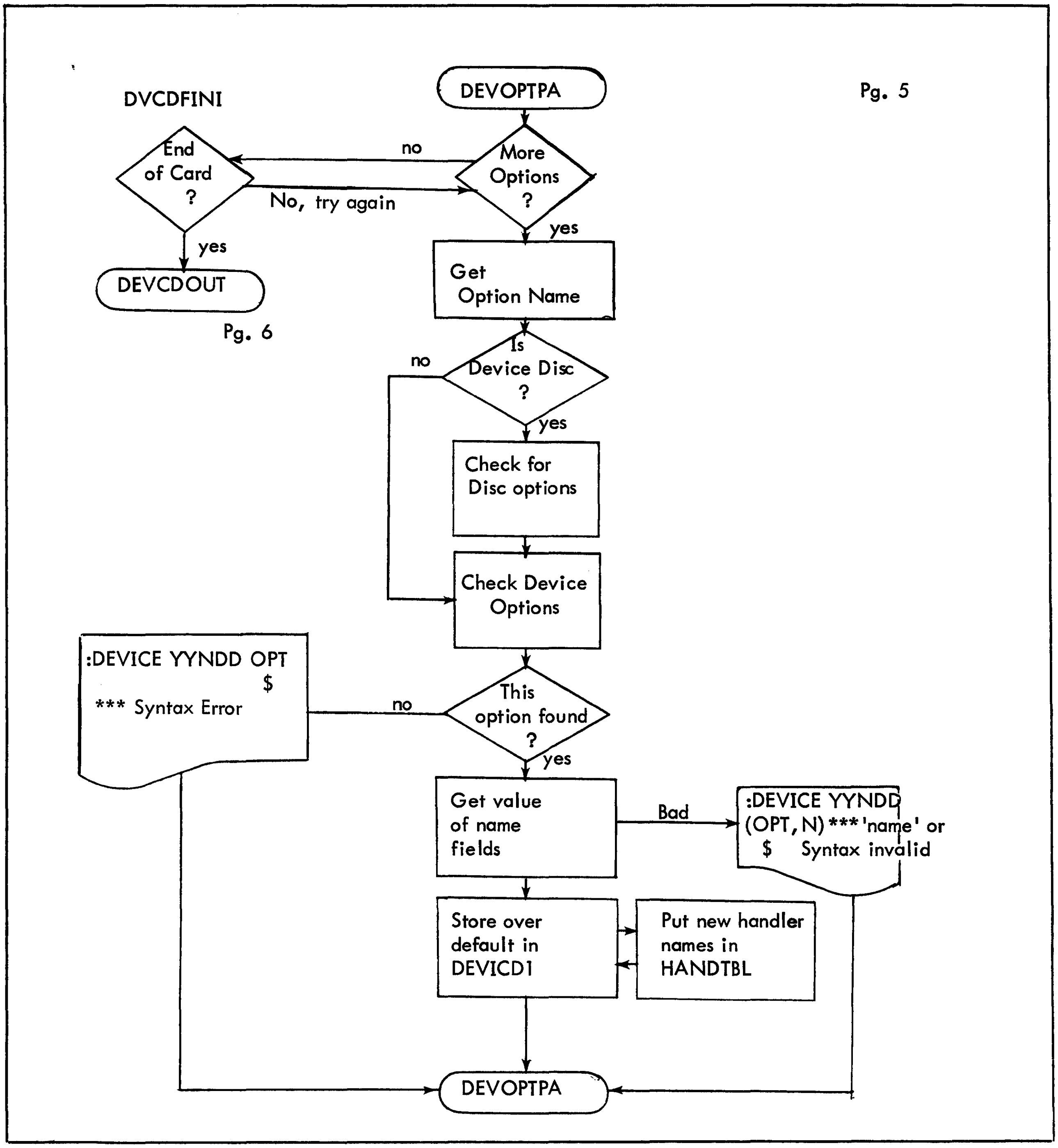


Figure 2-9. Flow Diagram of UBCHAN (Cont.)

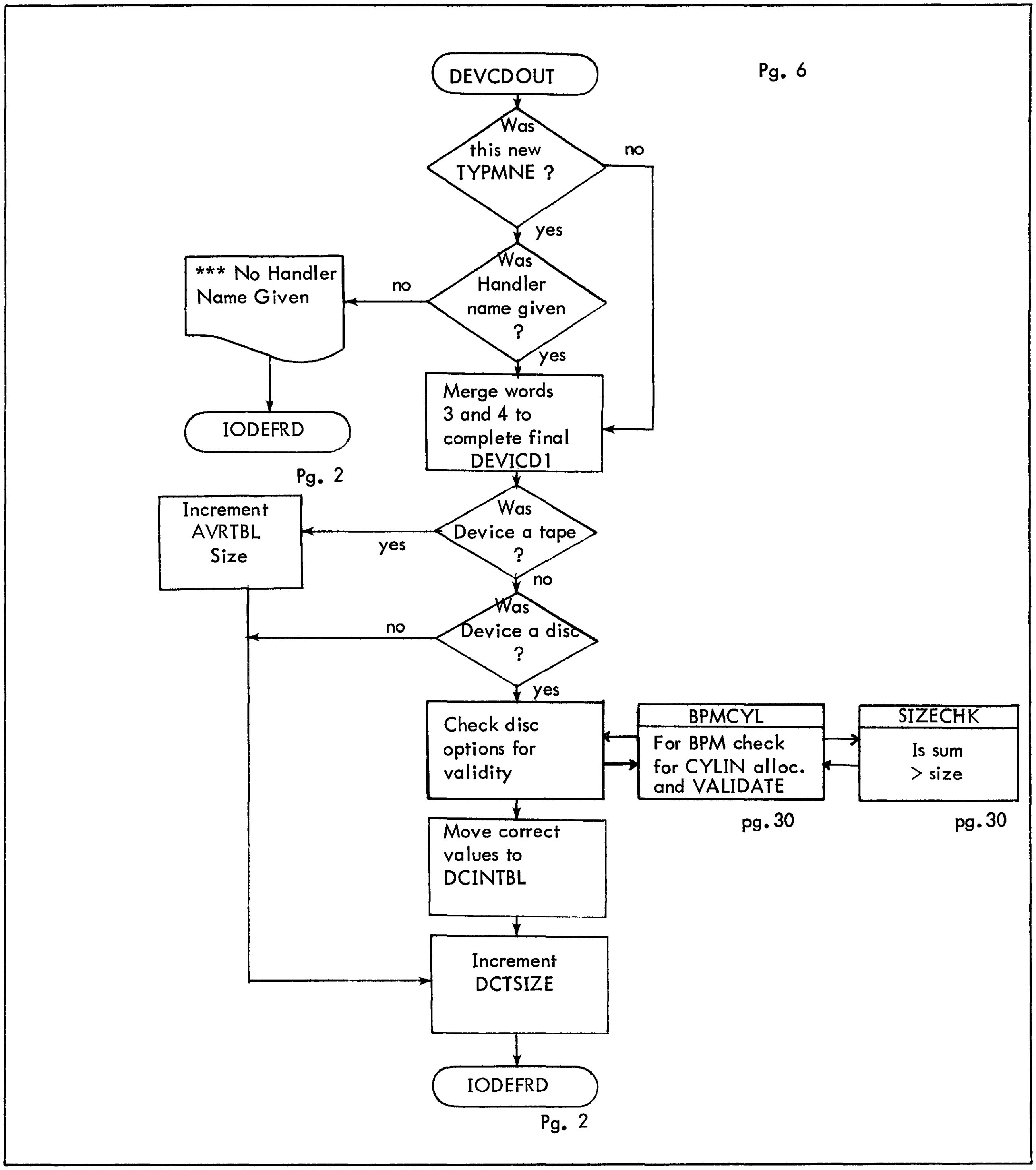


Figure 2-9. Flow Diagram of UBCHAN (Cont.)

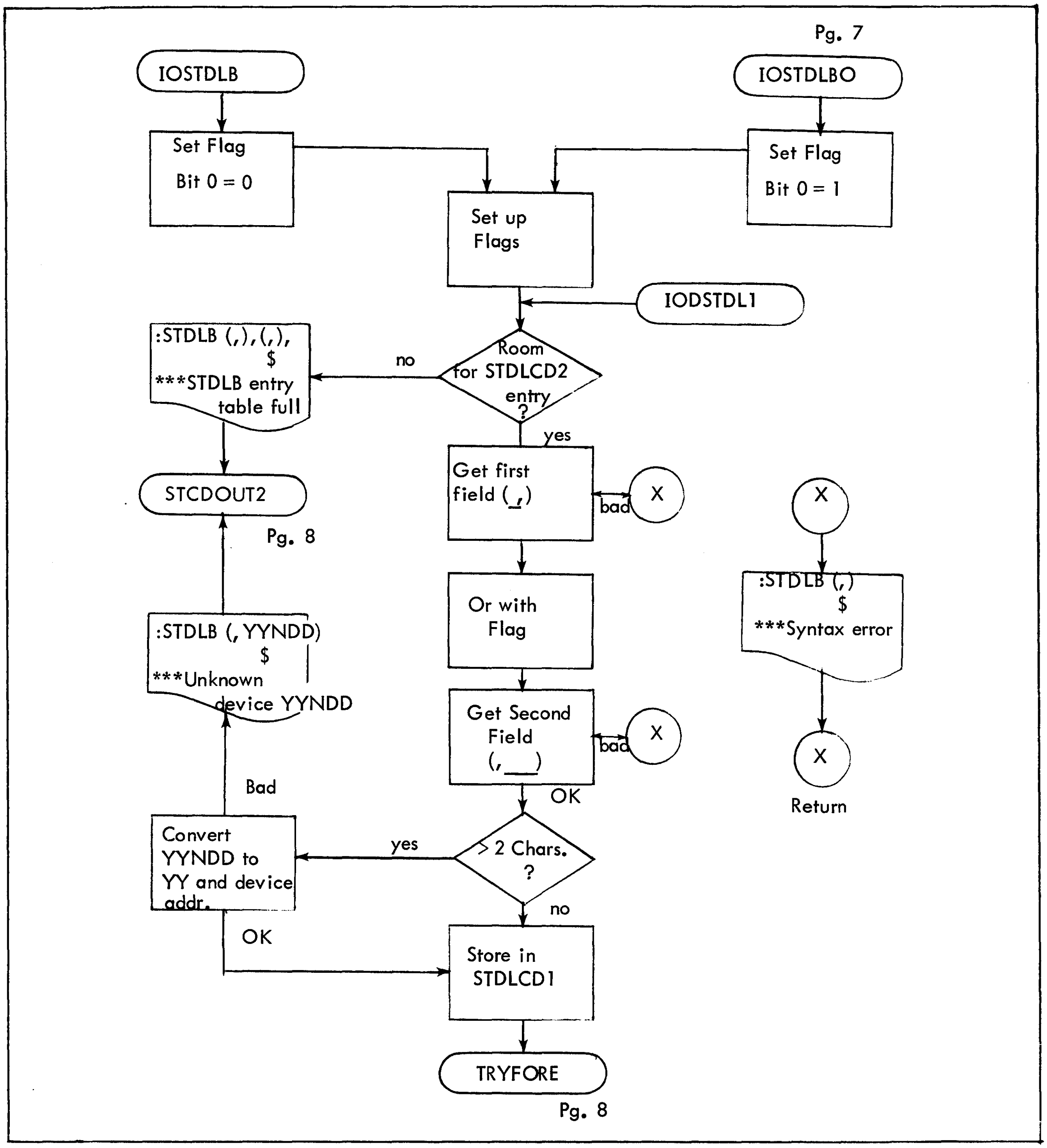


Figure 2-9. Flow Diagram of UBCHAN (Cont.)

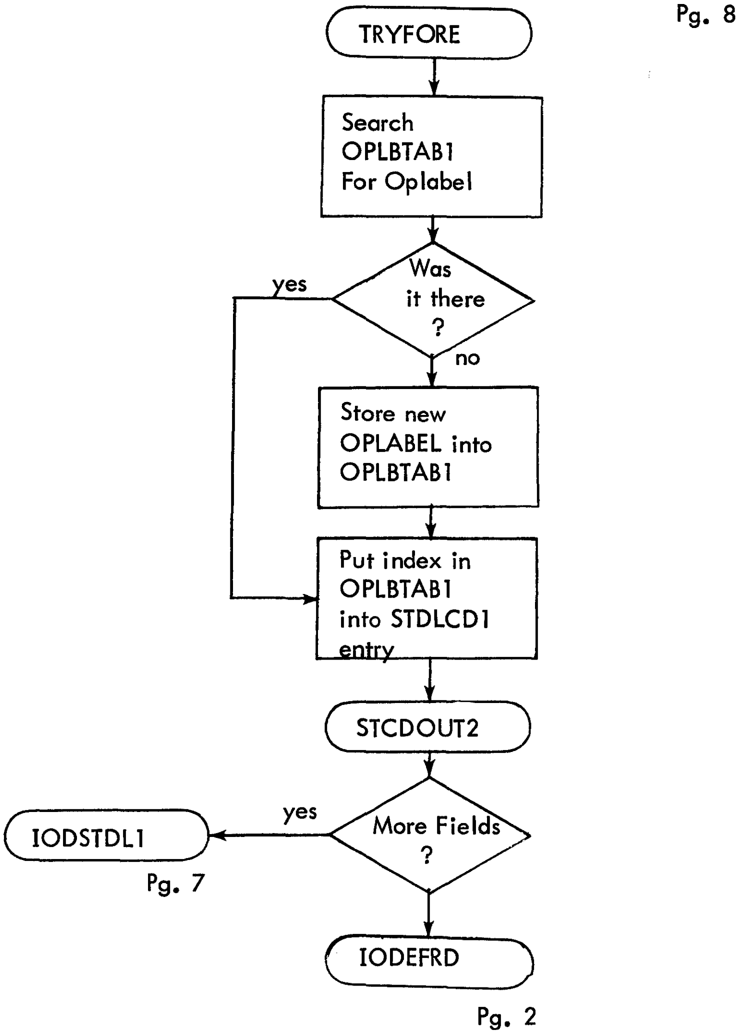

Pg. 8
TRYFORE
Search OPLBTAB1 For Oplabel
Was it there ?
yes
no
Store new OPLABEL into OPLBTAB1
Put index in OPLBTAB1 into STDLCD1 entry
STCDOUT2
More Fields ?
yes
IODSTDL1
Pg. 7
IODEFRD
Pg. 2

Figure 2-9. Flow Diagram of UBCHAN (Cont.)

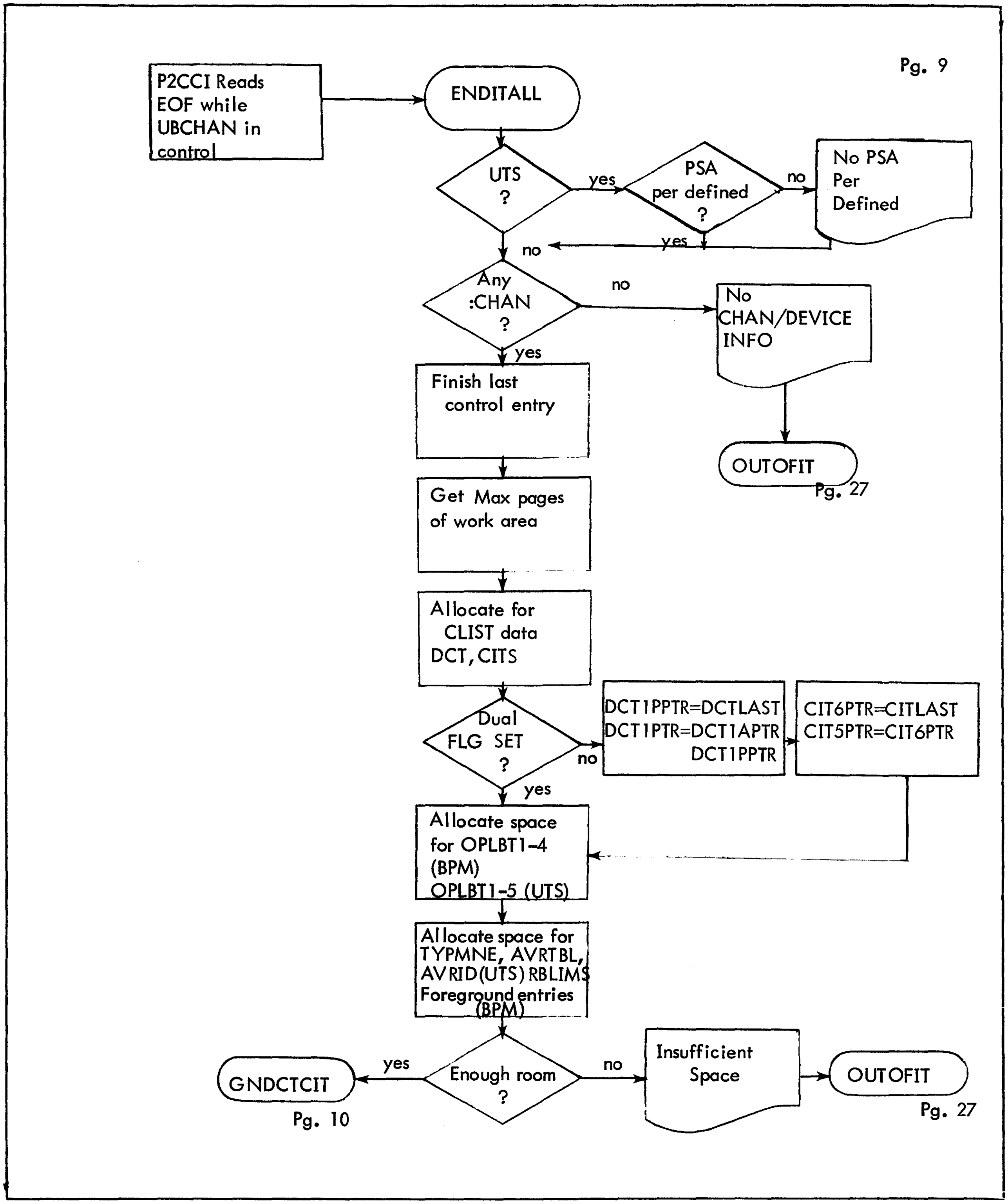


Figure 2-9. Flow Diagram of UBCHAN (Cont.)

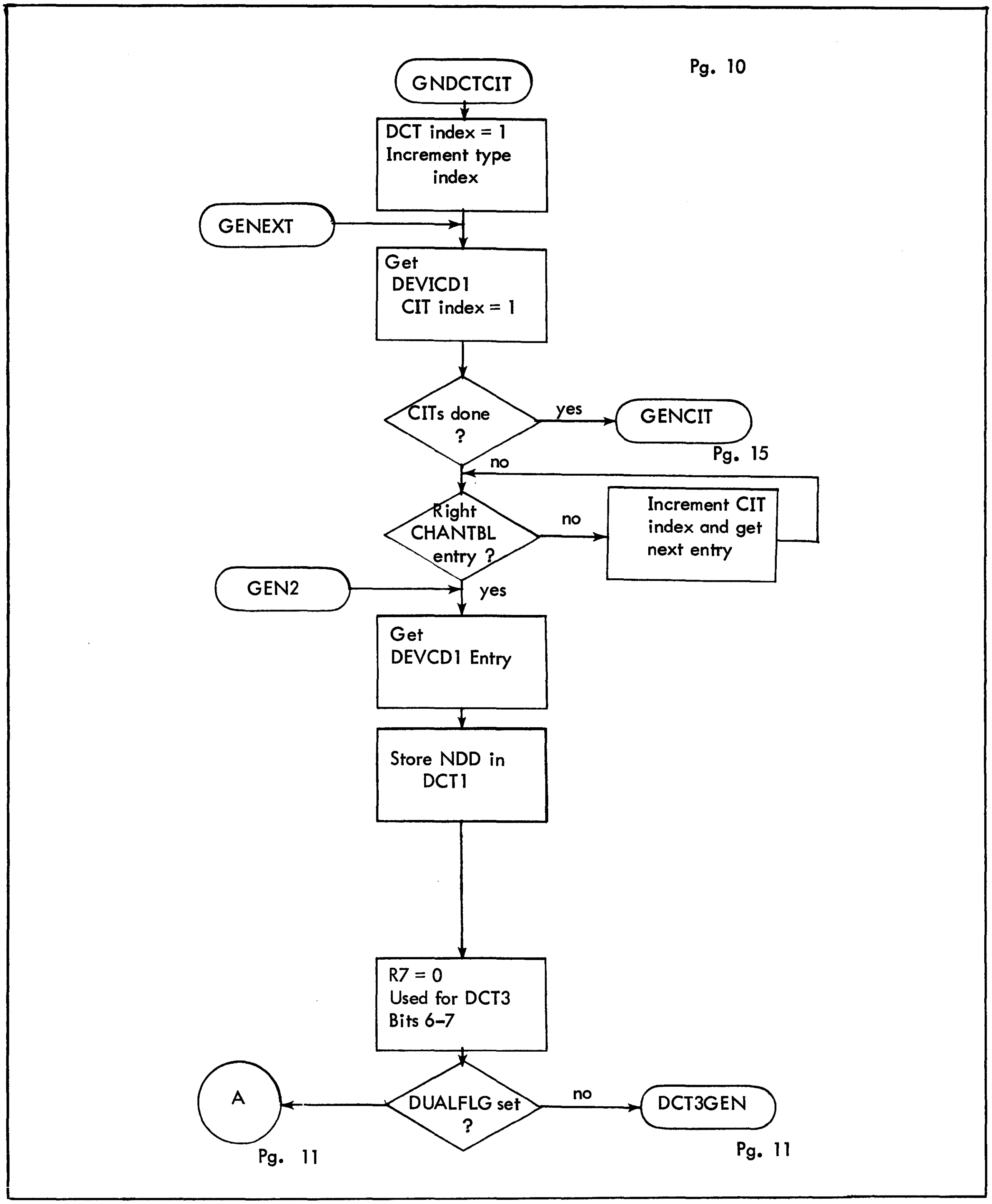


Figure 2-9. Flow Diagram of UBCHAN (Cont.)

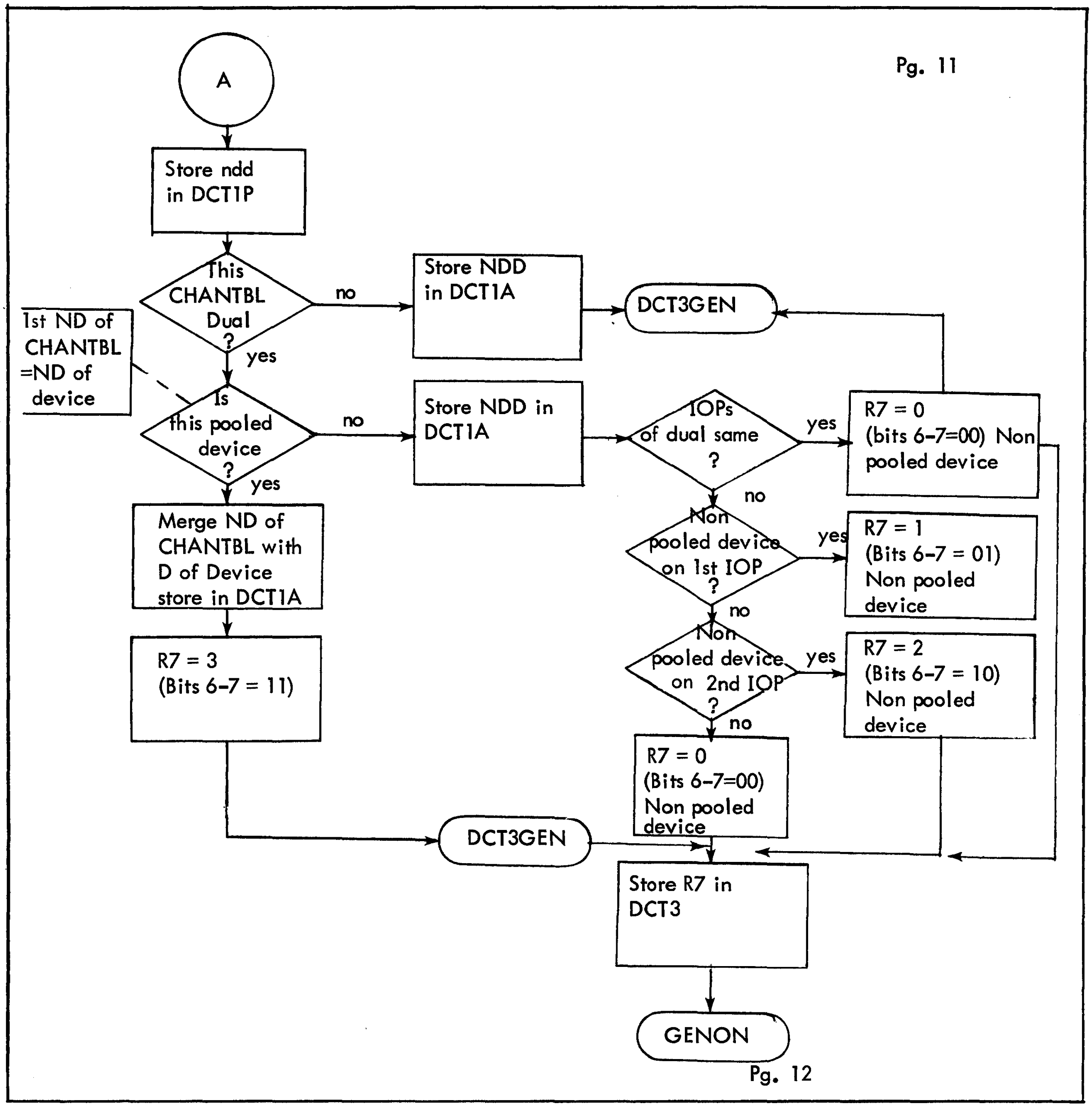

Pg. 11
A
Store ndd in DCT1P
This CHANTBL Dual ?
no
yes
Store NDD in DCT1A
DCT3GEN
1st ND of CHANTBL =ND of device
Is this pooled device ?
no
yes
Store NDD in DCT1A
IOPs of dual same ?
yes
no
R7 = 0 (bits 6-7=00) Non pooled device
Merge ND of CHANTBL with D of Device store in DCT1A
Non pooled device on 1st IOP ?
yes
no
R7 = 1 (Bits 6-7 = 01) Non pooled device
R7 = 3 (Bits 6-7 = 11)
Non pooled device on 2nd IOP ?
yes
no
R7 = 2 (Bits 6-7 = 10) Non pooled device
R7 = 0 (Bits 6-7=00) Non pooled device
DCT3GEN
Store R7 in DCT3
GENON
Pg. 12

Figure 2-9. Flow Diagram of UBCHAN (Cont.)

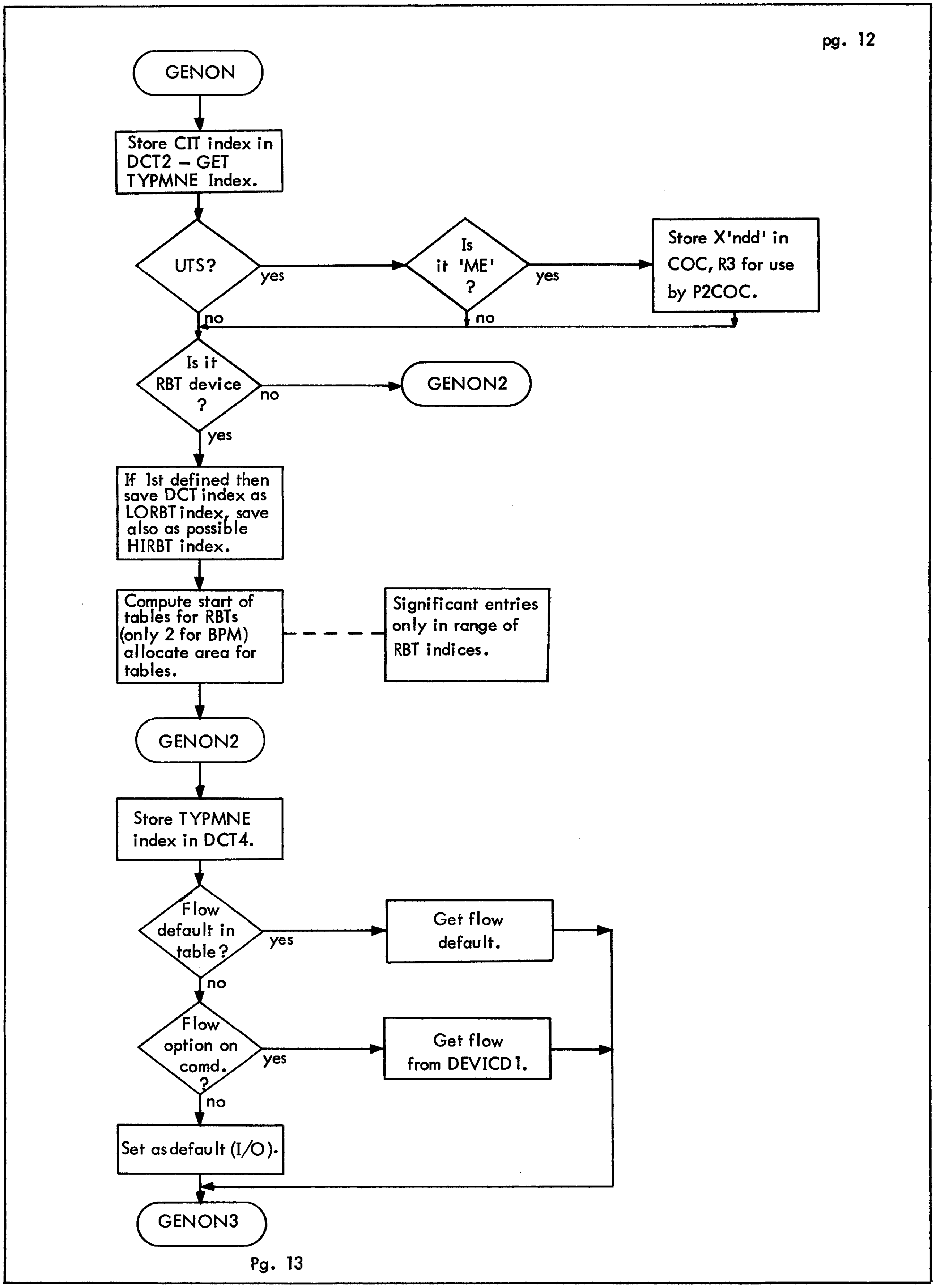

pg. 12
GENON
Store CIT index in DCT2 – GET TYPMNE Index.
UTS?
yes
no
Is it 'ME' ?
yes
no
Store X'ndd' in COC, R3 for use by P2COC.
Is it RBT device ?
no
GENON2
yes
If 1st defined then save DCT index as LORBT index, save also as possible HIRBT index.
Compute start of tables for RBTs (only 2 for BPM) allocate area for tables.
Significant entries only in range of RBT indices.
GENON2
Store TYPMNE index in DCT4.
Flow default in table?
yes
Get flow default.
no
Flow option on comd. ?
yes
Get flow from DEVICD1.
no
Set as default (I/O).
GENON3
Pg. 13

Figure 2-9. Flow Diagram of UBCHAN (cont.)

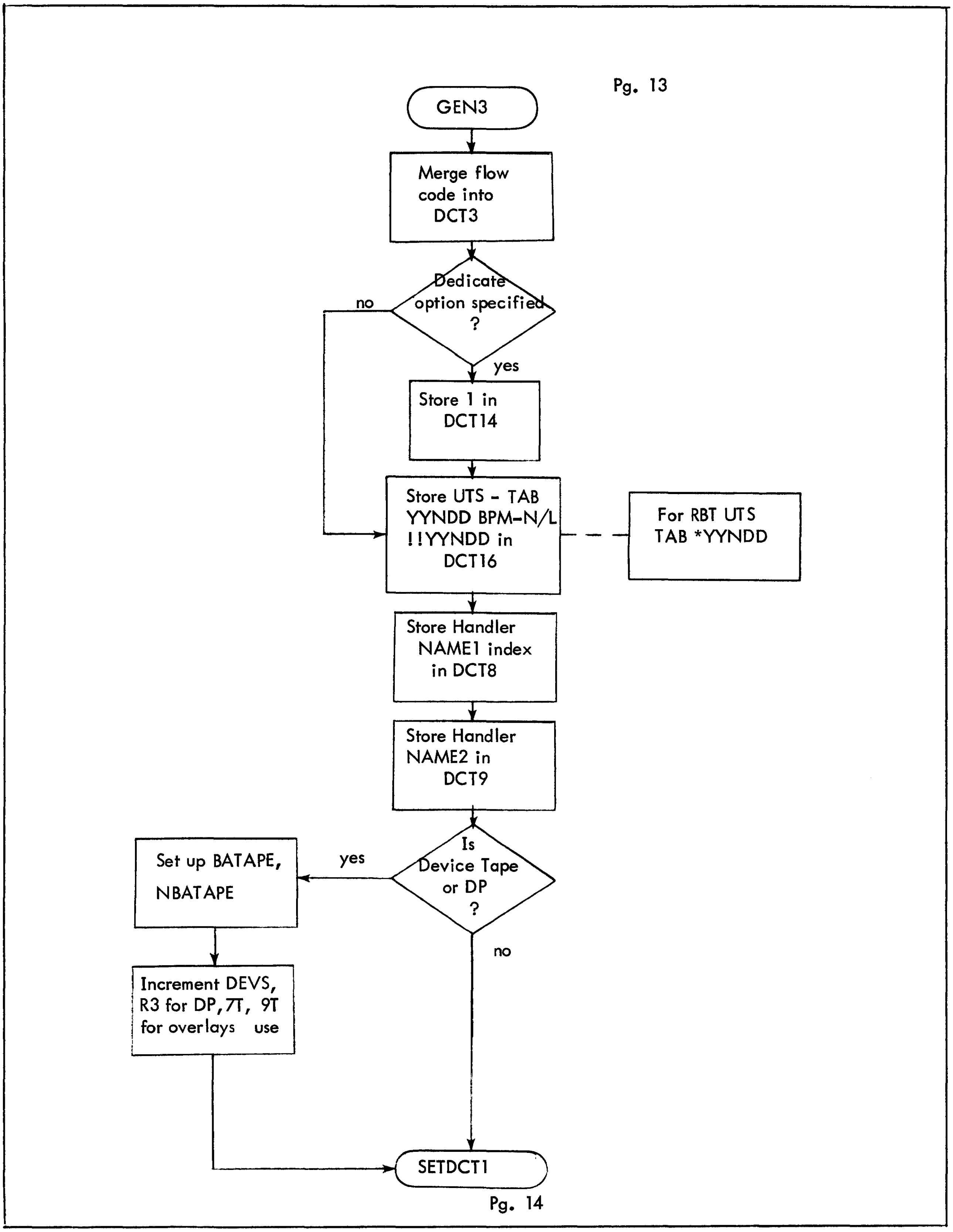

Pg. 13
GEN3
Merge flow code into DCT3
Dedicate option specified ?
no
yes
Store 1 in DCT14
Store UTS - TAB YYNDD BPM-N/L !!YYNDD in DCT16
For RBT UTS TAB *YYNDD
Store Handler NAME1 index in DCT8
Store Handler NAME2 in DCT9
Is Device Tape or DP ?
yes
no
Set up BATAPE, NBATAPE
Increment DEVS, R3 for DP, 7T, 9T for overlays use
SETDCT1
Pg. 14

Figure 2-9. Flow Diagram of UBCHAN (Cont.)

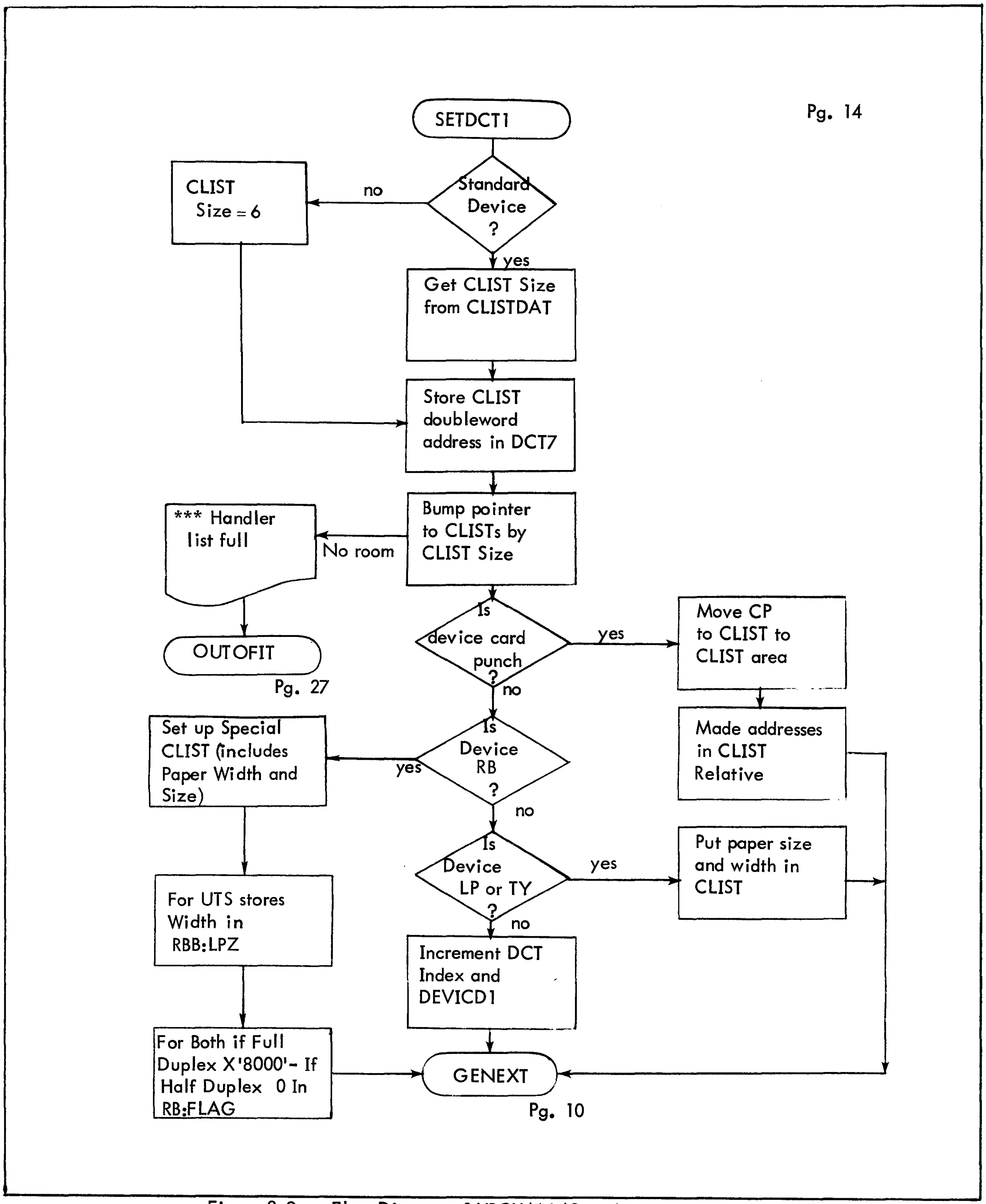

Pg. 14
SETDCT1
Standard Device ?
no
yes
CLIST Size = 6
Get CLIST Size from CLISTDAT
Store CLIST doubleword address in DCT7
Bump pointer to CLISTs by CLIST Size
No room
*** Handler list full
OUTOFIT
Pg. 27
Is device card punch ?
yes
no
Move CP to CLIST to CLIST area
Made addresses in CLIST Relative
Is Device RB ?
yes
no
Set up Special CLIST (includes Paper Width and Size)
Is Device LP or TY ?
yes
no
Put paper size and width in CLIST
For UTS stores Width in RBB:LPZ
Increment DCT Index and DEVICD1
For Both if Full Duplex X'8000'- If Half Duplex 0 In RB:FLAG
GENEXT
Pg. 10

Figure 2-9. Flow Diagram of UBCHAN (Cont.)

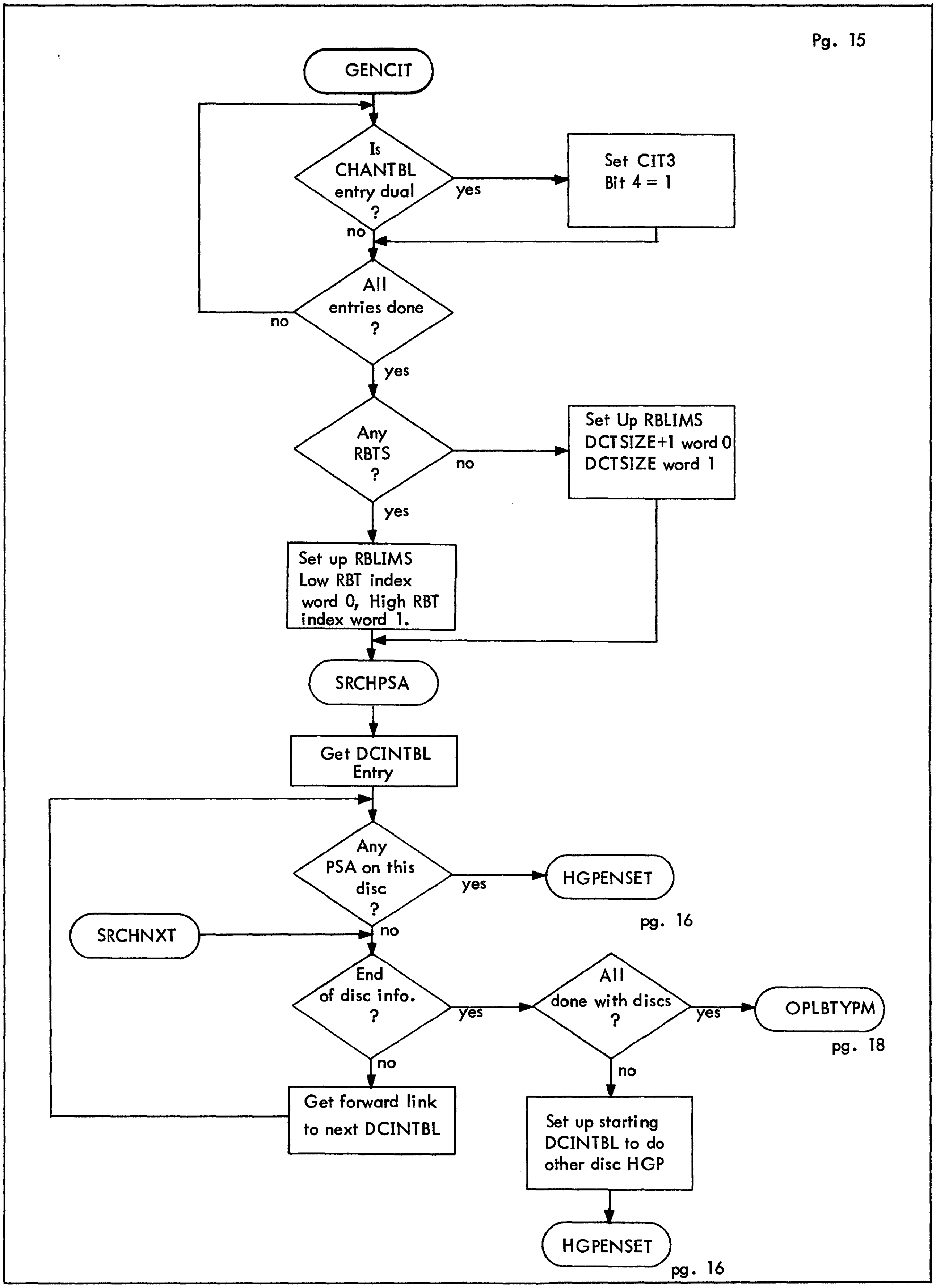


Figure 2-9. Flow Diagram of UBCHAN (cont.)

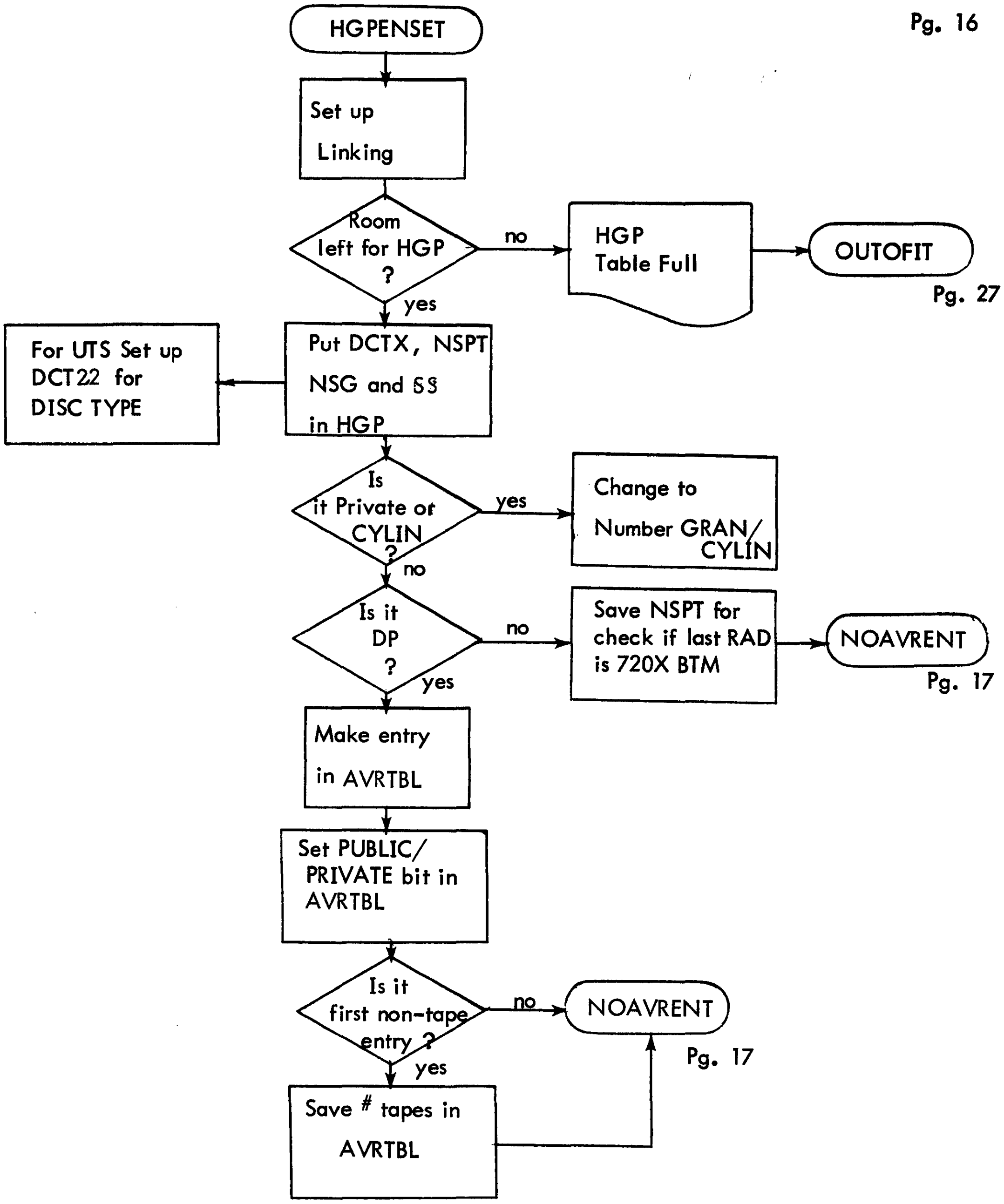
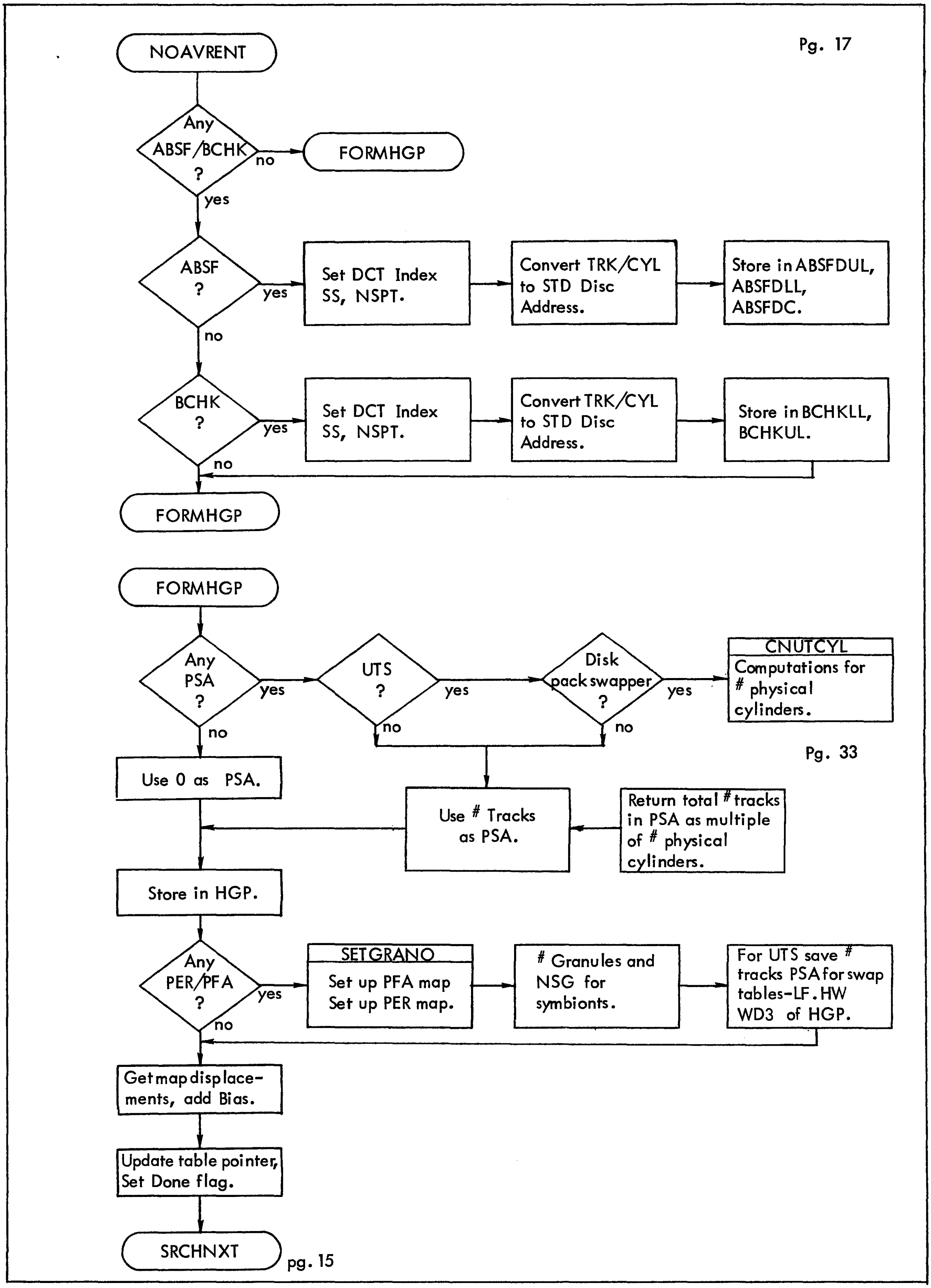

HGPENSET
Pg. 16
Set up Linking
Room left for HGP ?
no
HGP Table Full
OUTOFIT
Pg. 27
yes
Put DCTX, NSPT NSG and SS in HGP
For UTS Set up DCT22 for DISC TYPE
Is it Private or CYLIN ?
yes
Change to Number GRAN/ CYLIN
no
Is it DP ?
no
Save NSPT for check if last RAD is 720X BTM
NOAVRENT
Pg. 17
yes
Make entry in AVRTBL
Set PUBLIC/ PRIVATE bit in AVRTBL
Is it first non-tape entry ?
no
NOAVRENT
Pg. 17
yes
Save # tapes in AVRTBL

Figure 2-9. Flow Diagram of UBCHAN (Cont.)



NOAVRENT
Pg. 17
Any ABSF/BCHK ?
no
FORMHGP
yes
ABSF ?
yes
Set DCT Index SS, NSPT.
Convert TRK/CYL to STD Disc Address.
Store in ABSFDUL, ABSFDLL, ABSFDC.
no
BCHK ?
yes
Set DCT Index SS, NSPT.
Convert TRK/CYL to STD Disc Address.
Store in BCHKLL, BCHKUL.
no
FORMHGP
FORMHGP
Any PSA ?
yes
UTS ?
yes
Disk pack swapper ?
yes
CNUTCYL
Computations for # physical cylinders.
Pg. 33
no
no
no
Use 0 as PSA.
Use # Tracks as PSA.
Return total # tracks in PSA as multiple of # physical cylinders.
Store in HGP.
Any PER/PFA ?
yes
SETGRANO
Set up PFA map Set up PER map.
# Granules and NSG for symbionts.
For UTS save # tracks PSA for swap tables-LF.HW WD3 of HGP.
no
Get map displace-ments, add Bias.
Update table pointer, Set Done flag.
SRCHNXT
pg. 15

Figure 2-9. Flow Diagram of UBCHAN (cont.)

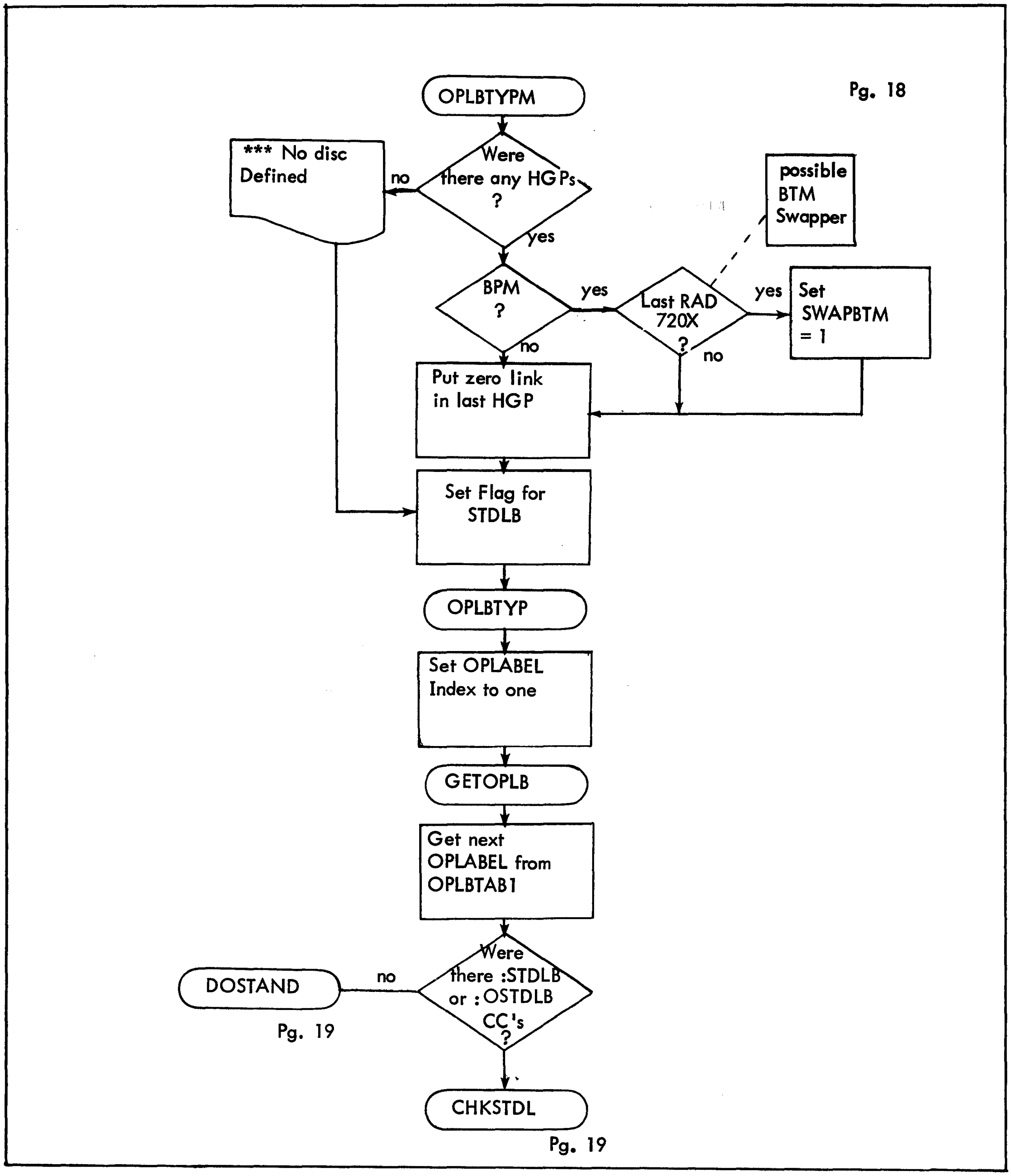


Figure 2-9. Flow Diagram of UBCHAN (Cont.)

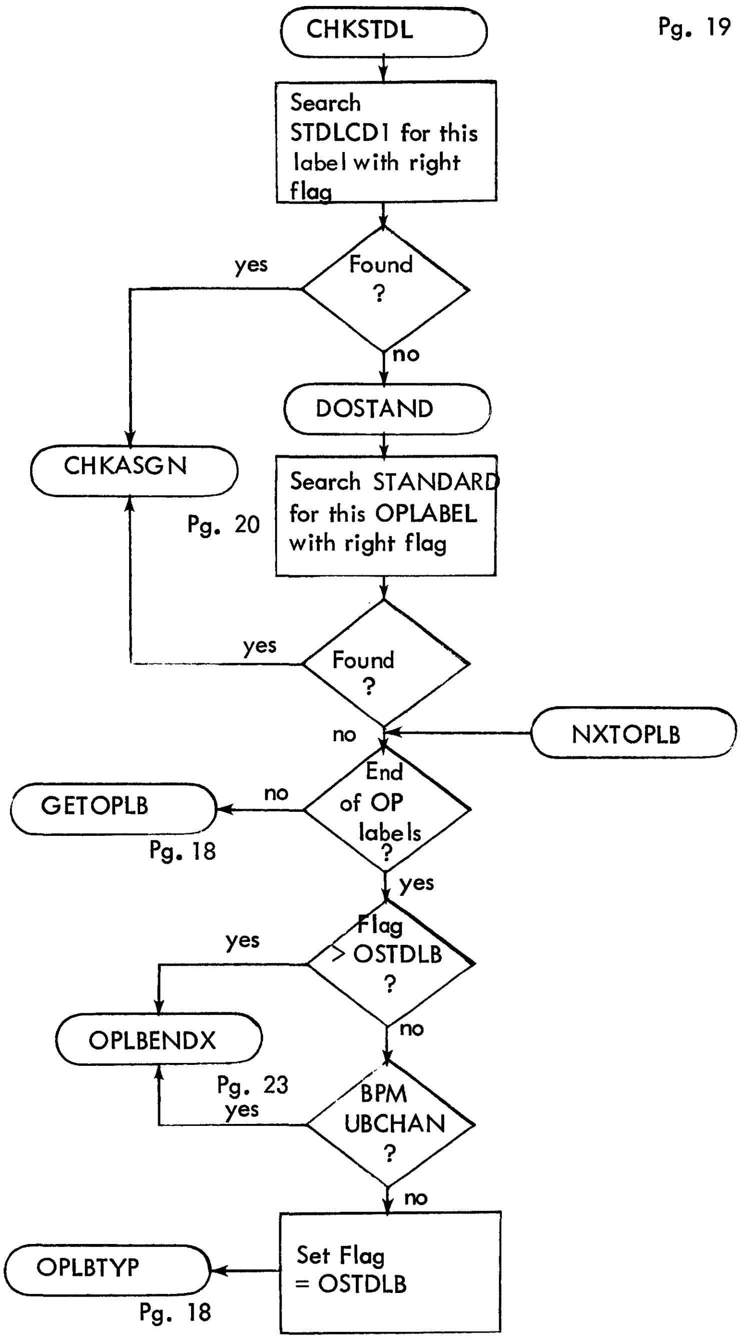
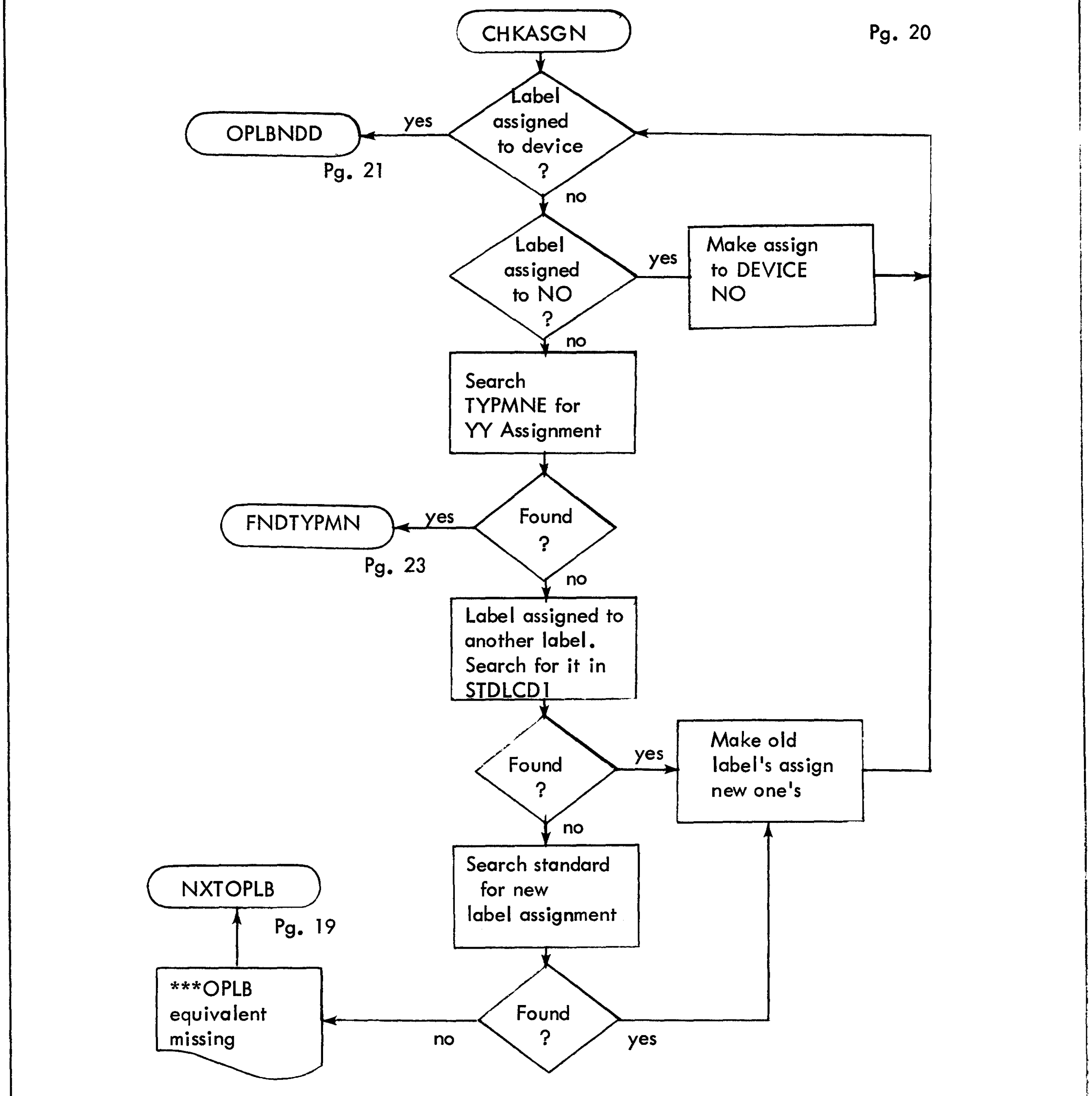


Figure 2-9. Flow Diagram of UBCHAN (Cont.)



CHKASGN
Pg. 20
Label assigned to device ?
yes
OPLBNDD
Pg. 21
no
Label assigned to NO ?
yes
Make assign to DEVICE NO
no
Search TYPMNE for YY Assignment
FNDTYPMN
yes
Found ?
Pg. 23
no
Label assigned to another label. Search for it in STDLCD1
Found ?
yes
Make old label's assign new one's
no
NXTOPLB
Pg. 19
Search standard for new label assignment
***OPLB equivalent missing
no
Found ?
yes

Figure 2-9. Flow Diagram of UBCHAN (Cont.)

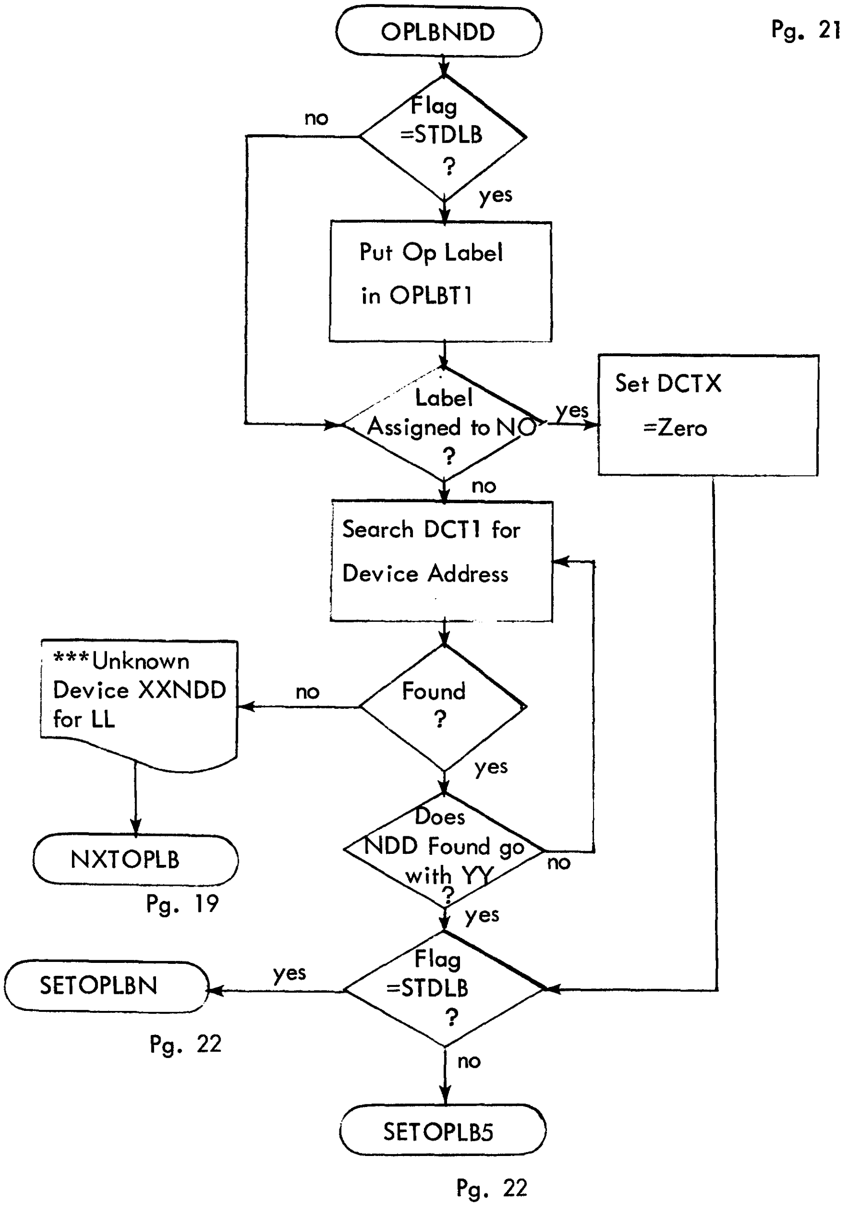


Figure 2-9. Flow Diagram of UBCHAN (Cont.)

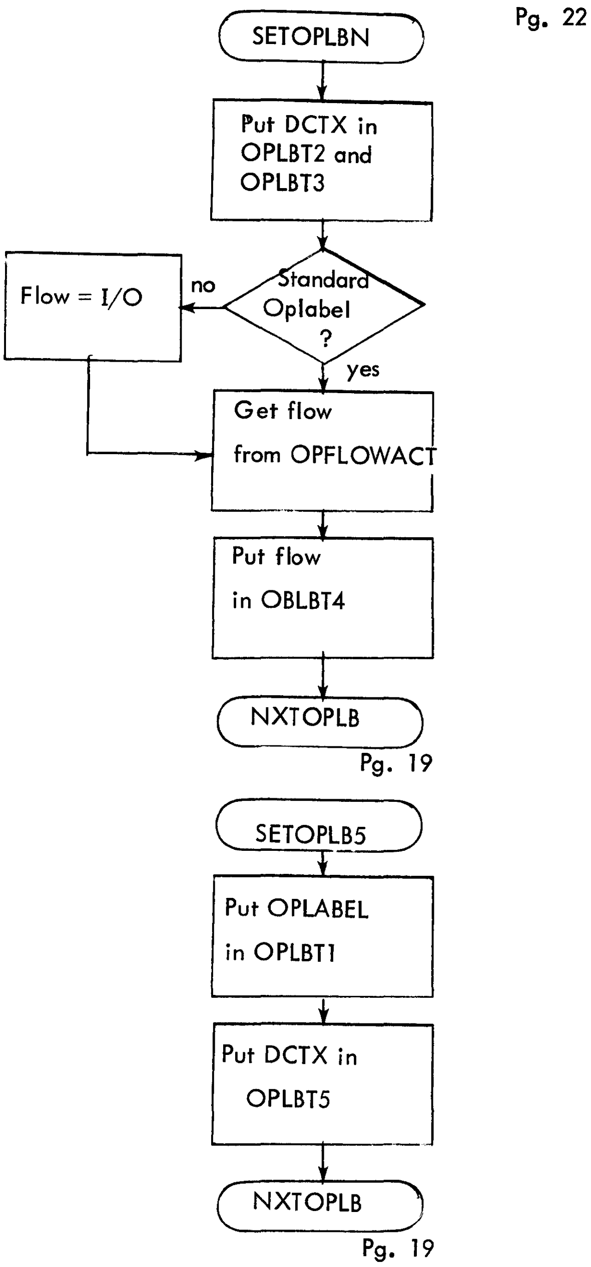


Figure 2-9. Flow Diagram of UBCHAN (Cont.)

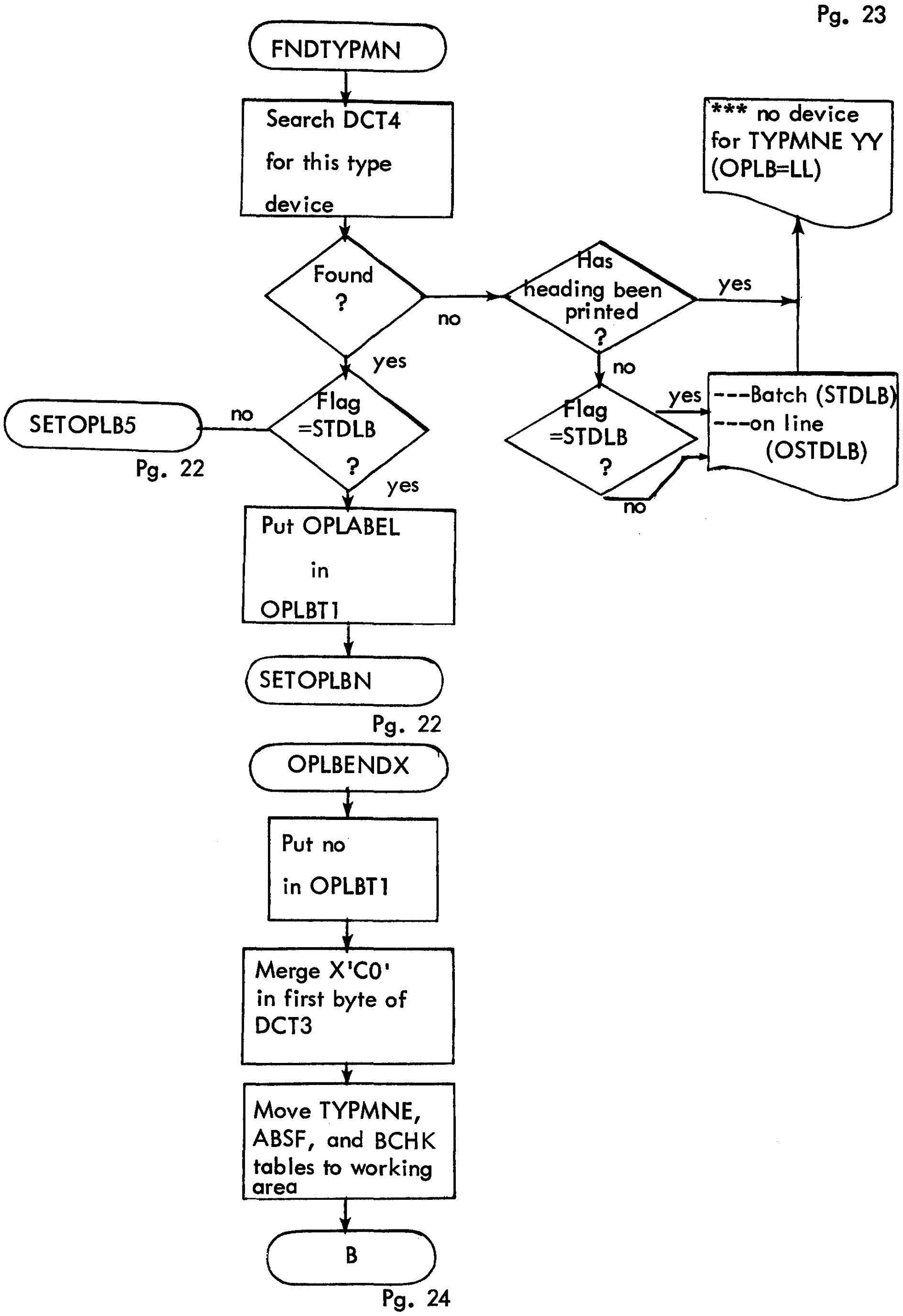


Figure 2-9. Flow Diagram of UBCHAN (Cont.)

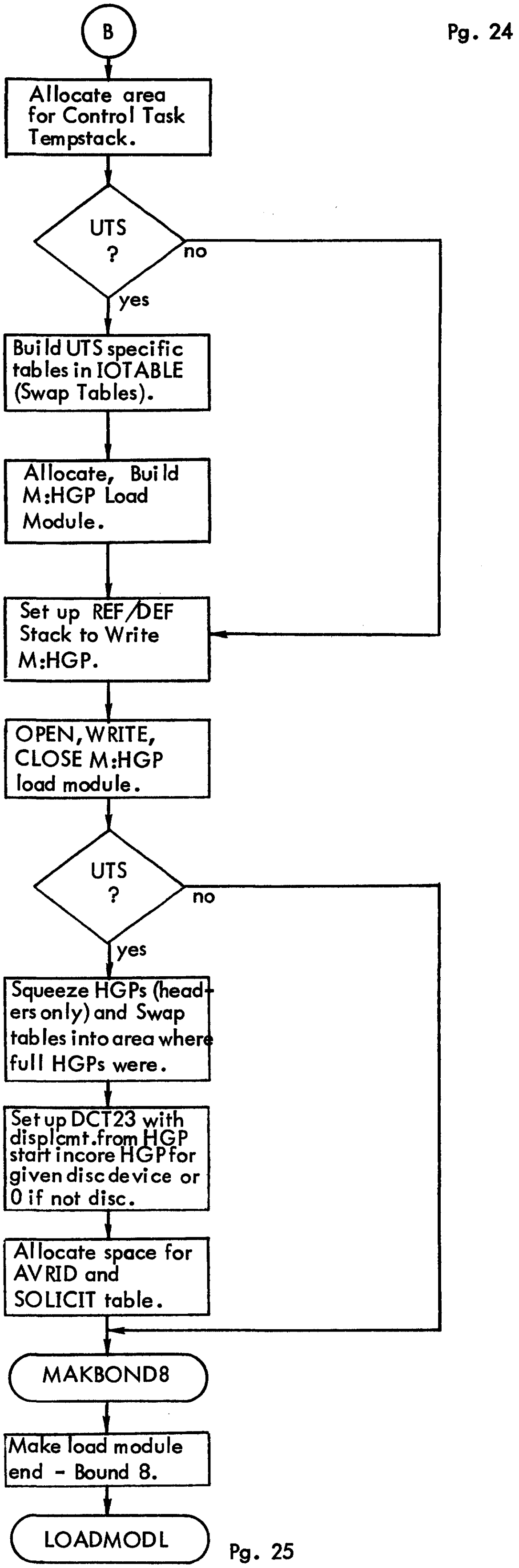

B
Pg. 24
Allocate area for Control Task Tempstack.
UTS ?
no
yes
Build UTS specific tables in IOTABLE (Swap Tables).
Allocate, Build M:HGP Load Module.
Set up REF/DEF Stack to Write M:HGP.
OPEN, WRITE, CLOSE M:HGP load module.
UTS ?
no
yes
Squeeze HGPs (headers only) and Swap tables into area where full HGPs were.
Set up DCT23 with displcmt.from HGP start incore HGP for given disc device or 0 if not disc.
Allocate space for AVRID and SOLICIT table.
MAKBOND8
Make load module end - Bound 8.
LOADMODL
Pg. 25

Figure 2-9. Flow Diagram of UBCHAN (cont.)

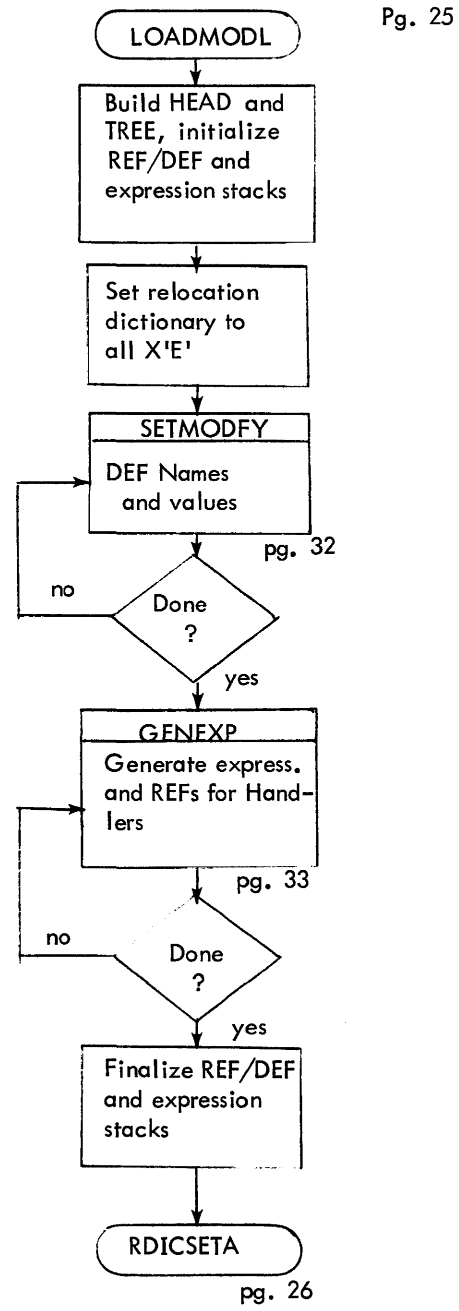

Pg. 25
LOADMODL
Build HEAD and TREE, initialize REF/DEF and expression stacks
Set relocation dictionary to all X'E'
SETMODFY
DEF Names and values
pg. 32
no
Done ?
yes
GENEXP
Generate express. and REFs for Handlers
pg. 33
no
Done ?
yes
Finalize REF/DEF and expression stacks
RDICSETA
pg. 26

Figure 2-9. Flow Diagram of UBCHAN (Cont.)

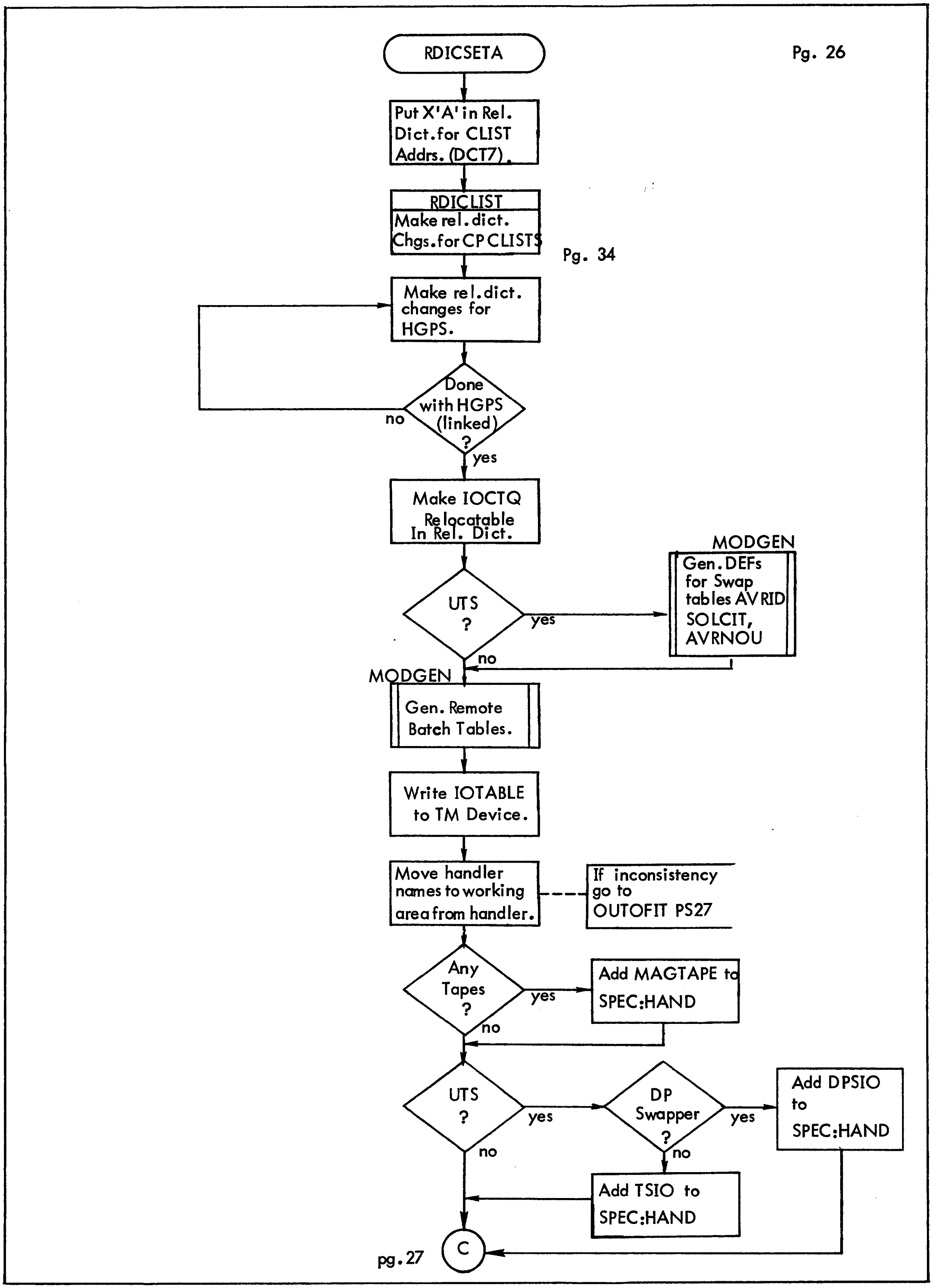

RDICSETA
Pg. 26
Put X'A' in Rel. Dict. for CLIST Addrs. (DCT7).
RDICLIST
Make rel. dict. Chgs. for CP CLISTS
Pg. 34
Make rel. dict. changes for HGPS.
Done with HGPS (linked) ?
no
yes
Make IOCTQ Relocatable In Rel. Dict.
UTS ?
yes
no
MODGEN
Gen. DEFs for Swap tables AVRID SOLCIT, AVRNOU
MODGEN
Gen. Remote Batch Tables.
Write IOTABLE to TM Device.
Move handler names to working area from handler.
If inconsistency go to OUTOFIT PS27
Any Tapes ?
yes
no
Add MAGTAPE to SPEC:HAND
UTS ?
yes
no
DP Swapper ?
yes
no
Add DPSIO to SPEC:HAND
Add TSIO to SPEC:HAND
pg. 27
C

Figure 2-9. Flow Diagram of UBCHAN (cont.)

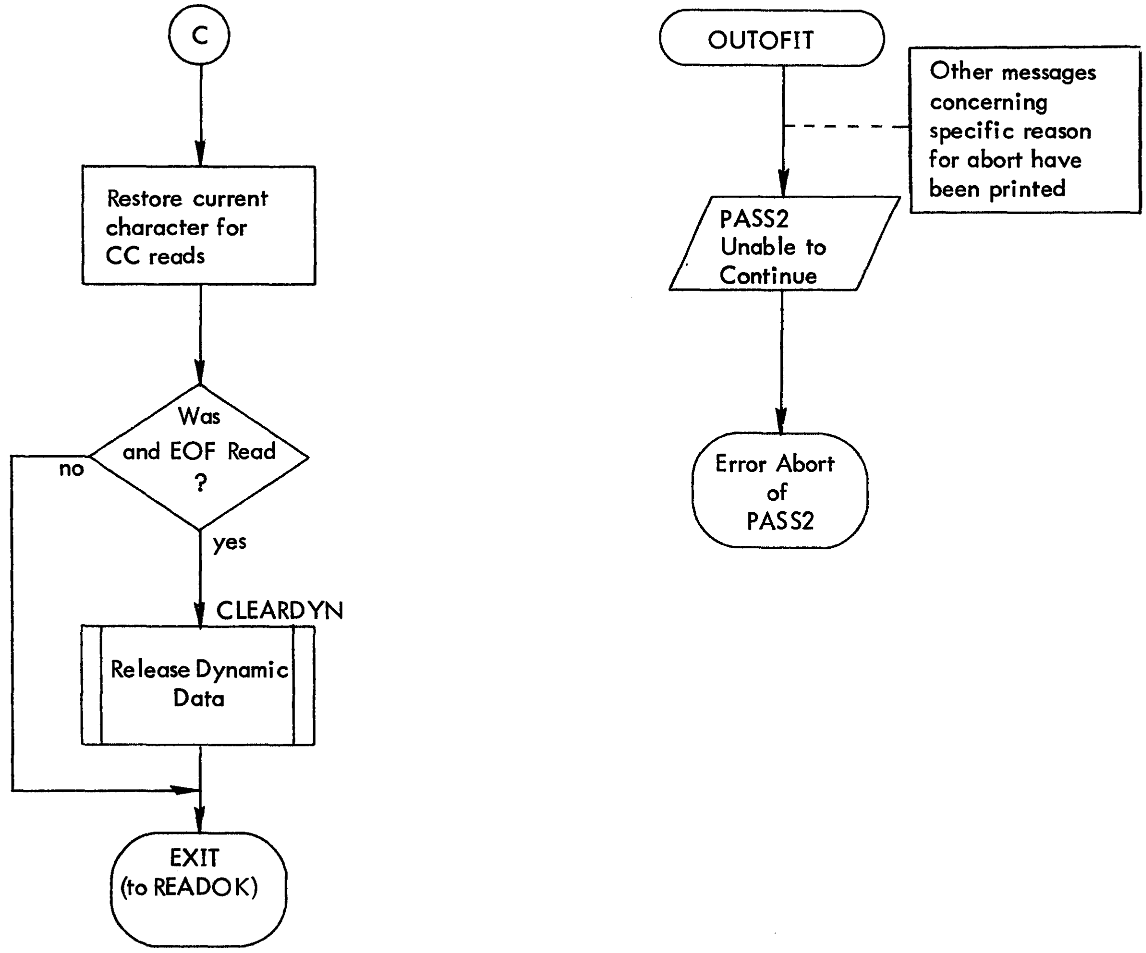


Figure 2-9. Flow Diagram of UBCHAN (cont.)

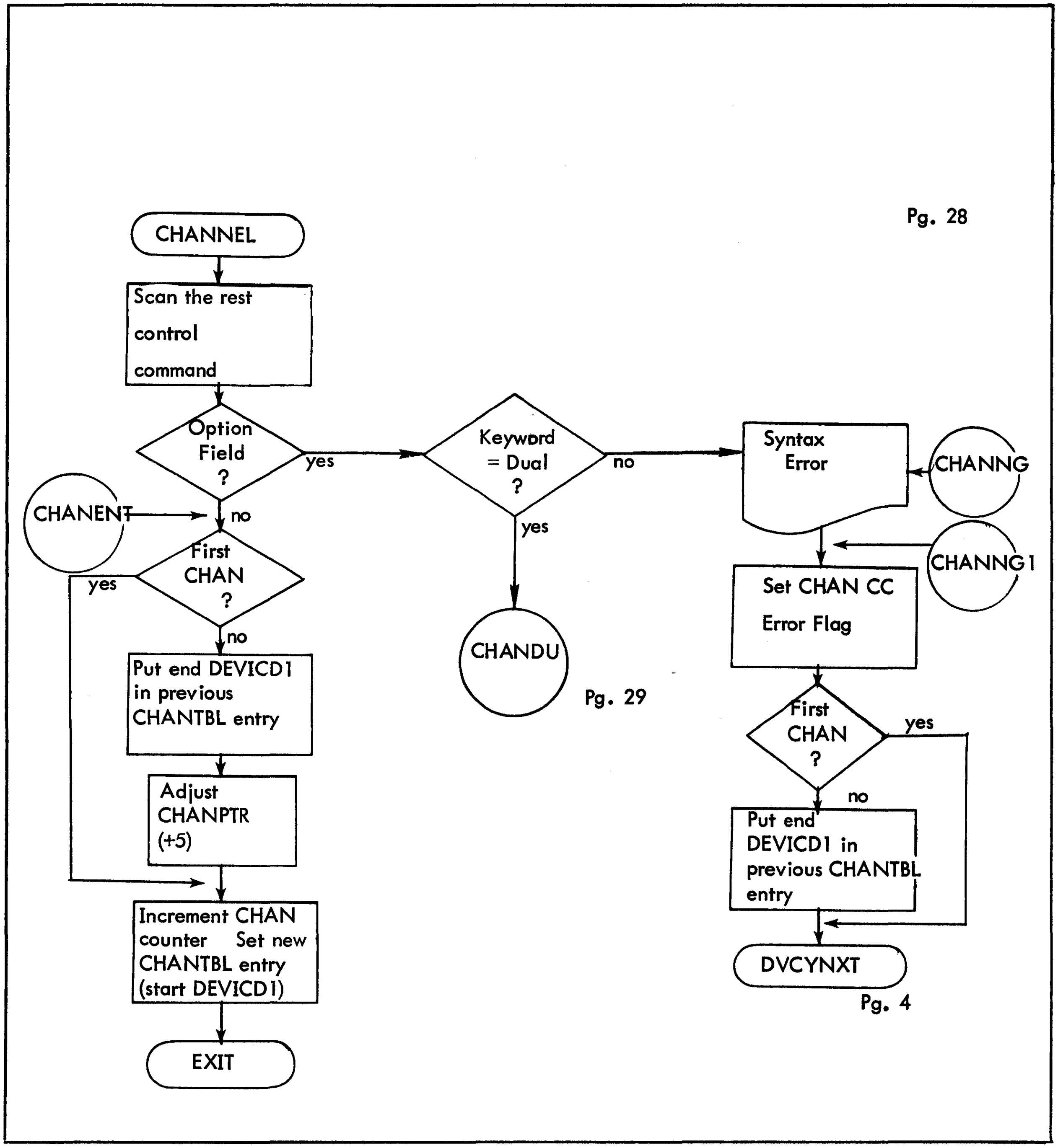


Figure 2-9. Flow Diagram of UBCHAN (Cont.)

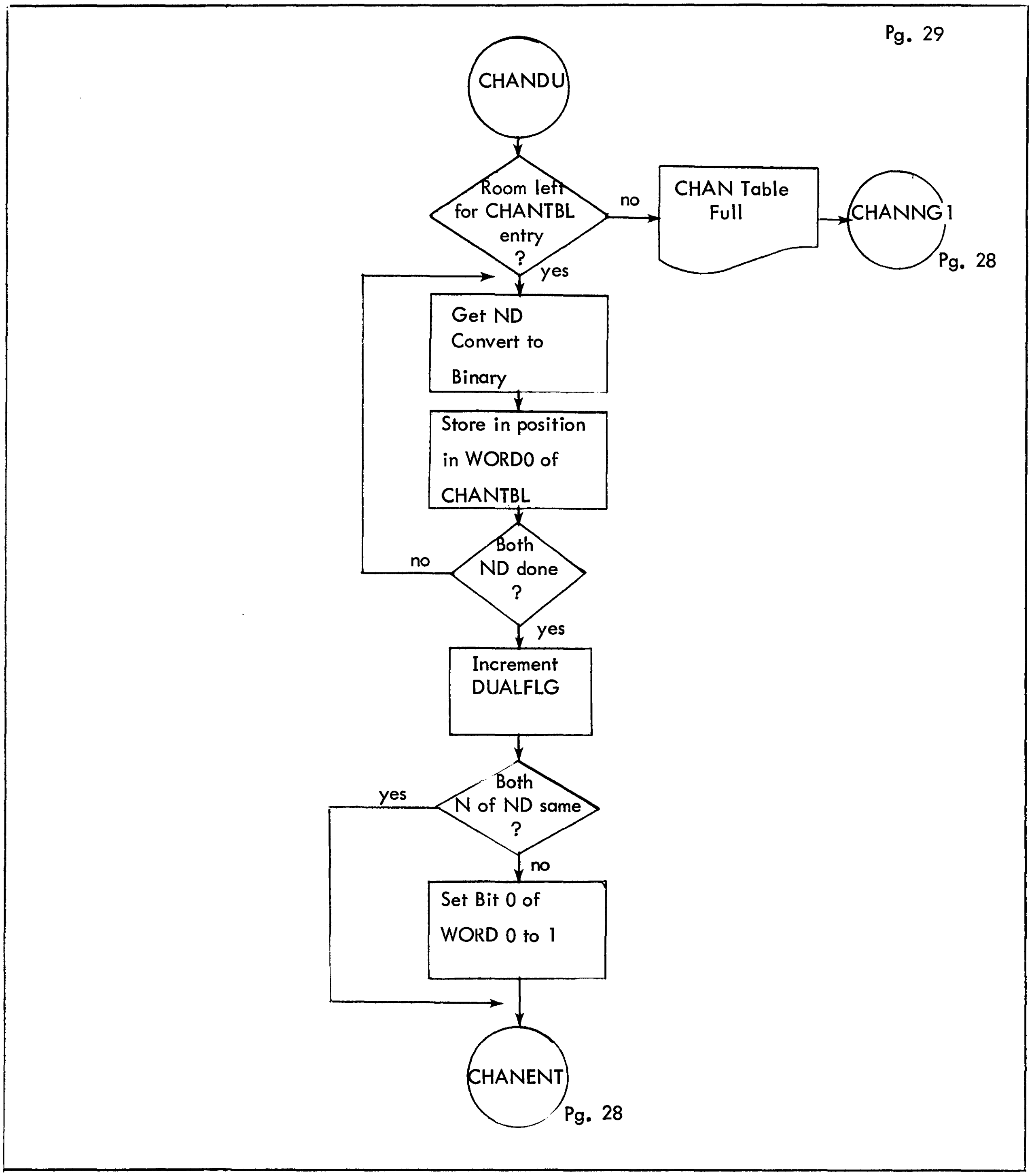


Figure 2-9. Flow Diagram of UBCHAN (Cont.)

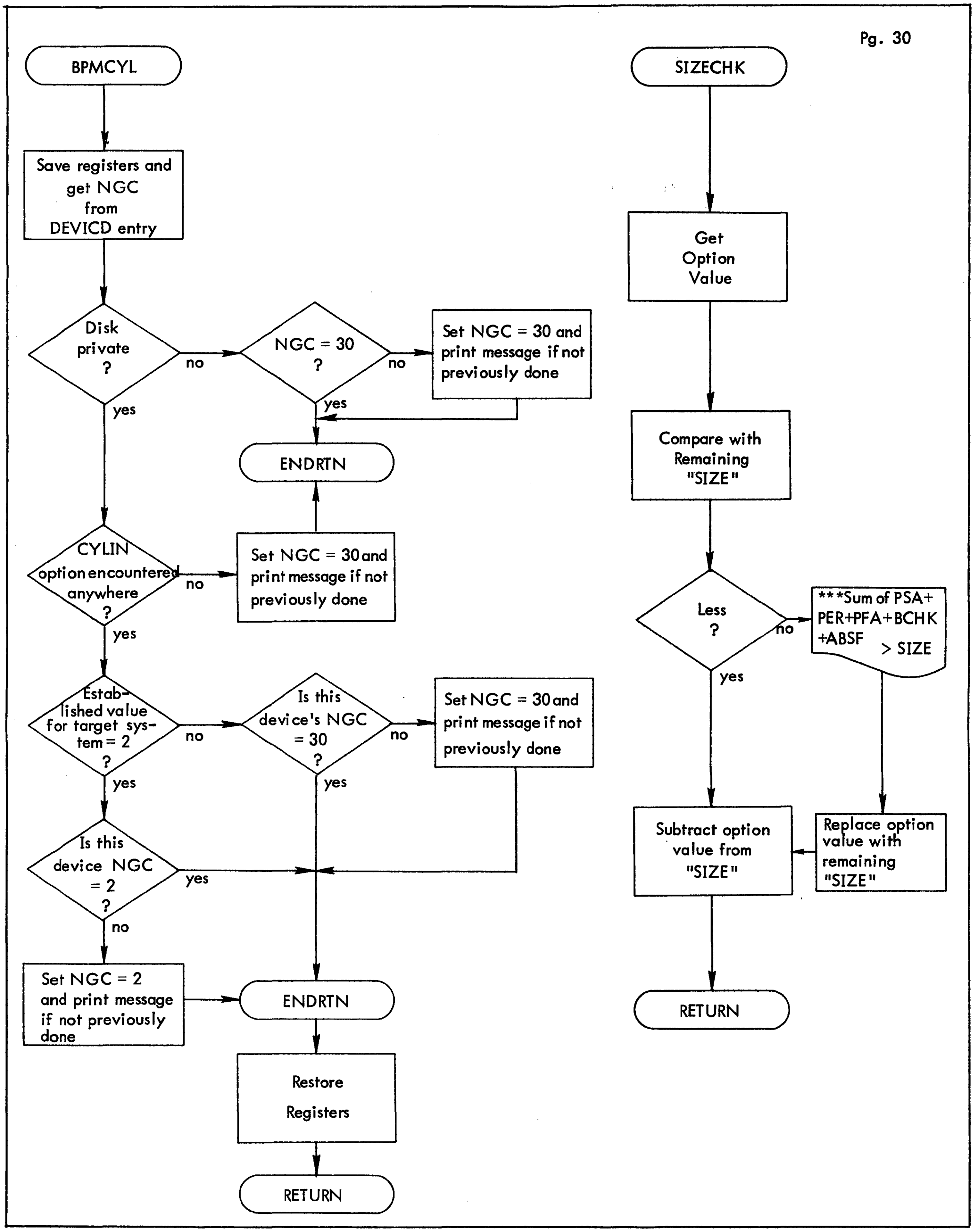

Pg. 30
BPMCYL
Save registers and get NGC from DEVICD entry
Disk private ?
no
yes
NGC = 30 ?
no
yes
Set NGC = 30 and print message if not previously done
ENDRTN
CYLIN option encountered anywhere ?
no
yes
Set NGC = 30and print message if not previously done
Estab-lished value for target sys-tem = 2 ?
no
yes
Is this device's NGC = 30 ?
no
yes
Set NGC = 30and print message if not previously done
Is this device NGC = 2 ?
yes
no
Set NGC = 2 and print message if not previously done
ENDRTN
Restore Registers
RETURN
SIZECHK
Get Option Value
Compare with Remaining "SIZE"
Less ?
no
yes
***Sum of PSA+PER+PFA+BCHK+ABSF > SIZE
Subtract option value from "SIZE"
Replace option value with remaining "SIZE"
RETURN

Figure 2-9. Flow Diagram of UBCHAN (cont.)

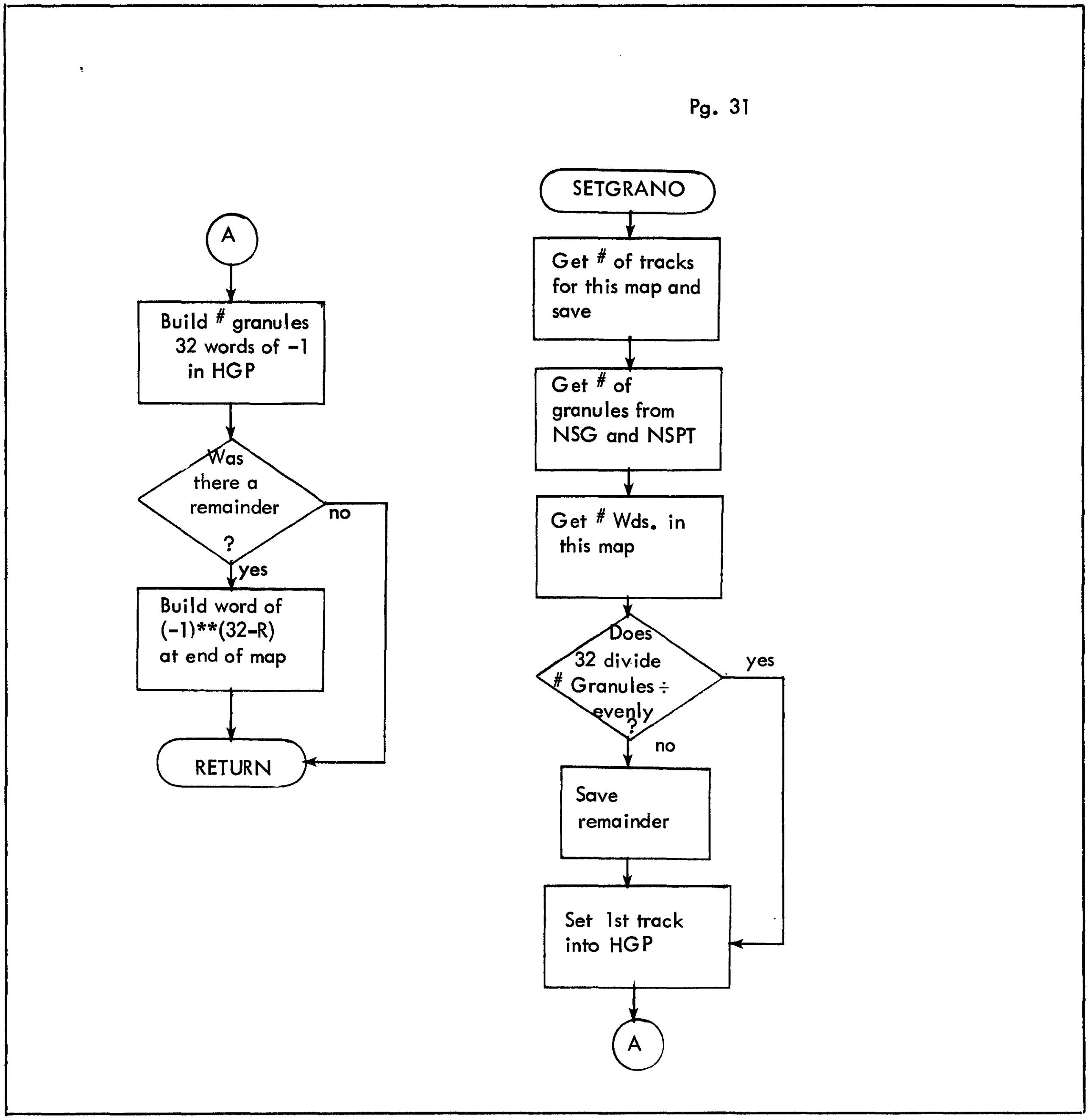

Pg. 31
A
Build # granules 32 words of -1 in HGP
Was there a remainder ?
no
yes
Build word of (-1)**(32-R) at end of map
RETURN
SETGRANO
Get # of tracks for this map and save
Get # of granules from NSG and NSPT
Get # Wds. in this map
Does 32 divide # Granules ÷ evenly ?
yes
no
Save remainder
Set 1st track into HGP
A

Figure 2-9. Flow Diagram of UBCHAN (Cont.)

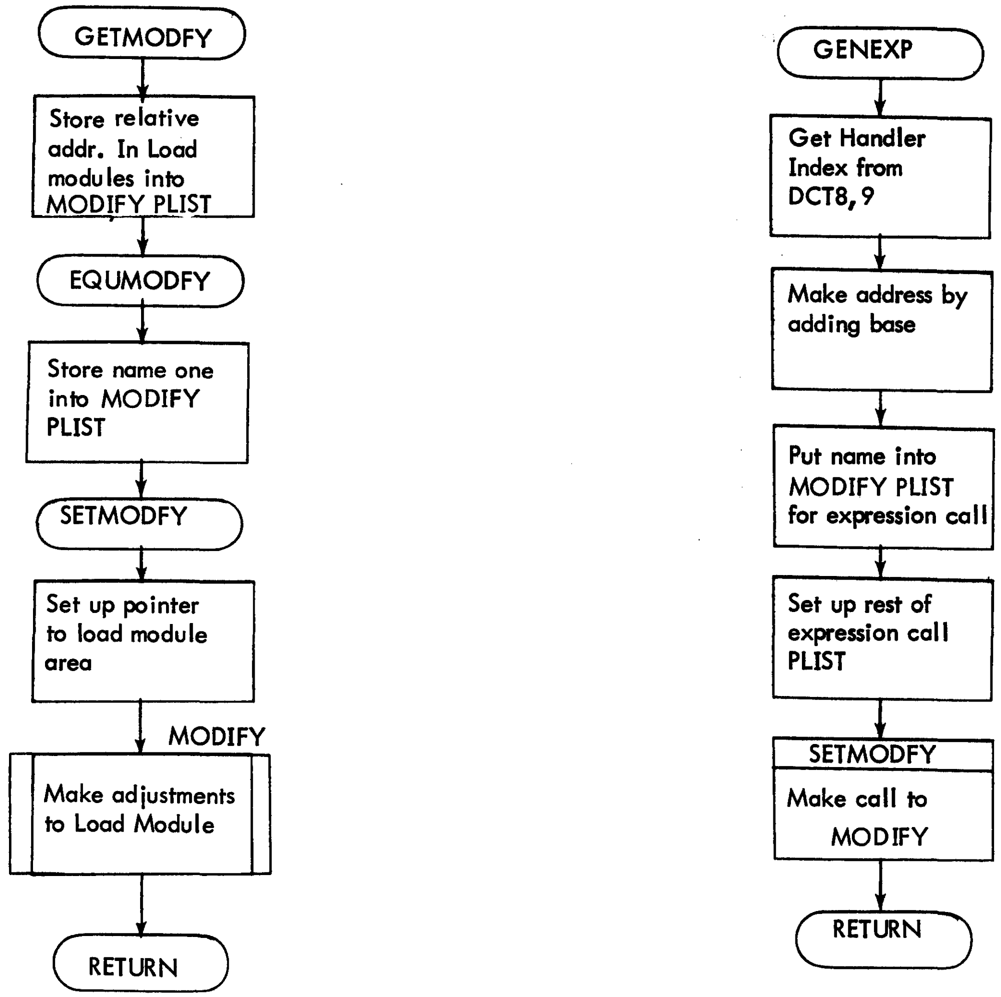

GETMODFY
Store relative addr. In Load modules into MODIFY PLIST
EQUMODFY
Store name one into MODIFY PLIST
SETMODFY
Set up pointer to load module area
MODIFY
Make adjustments to Load Module
RETURN
GENEXP
Get Handler Index from DCT8, 9
Make address by adding base
Put name into MODIFY PLIST for expression call
Set up rest of expression call PLIST
SETMODFY
Make call to MODIFY
RETURN

Figure 2-9. Flow Diagram of UBCHAN (cont.)

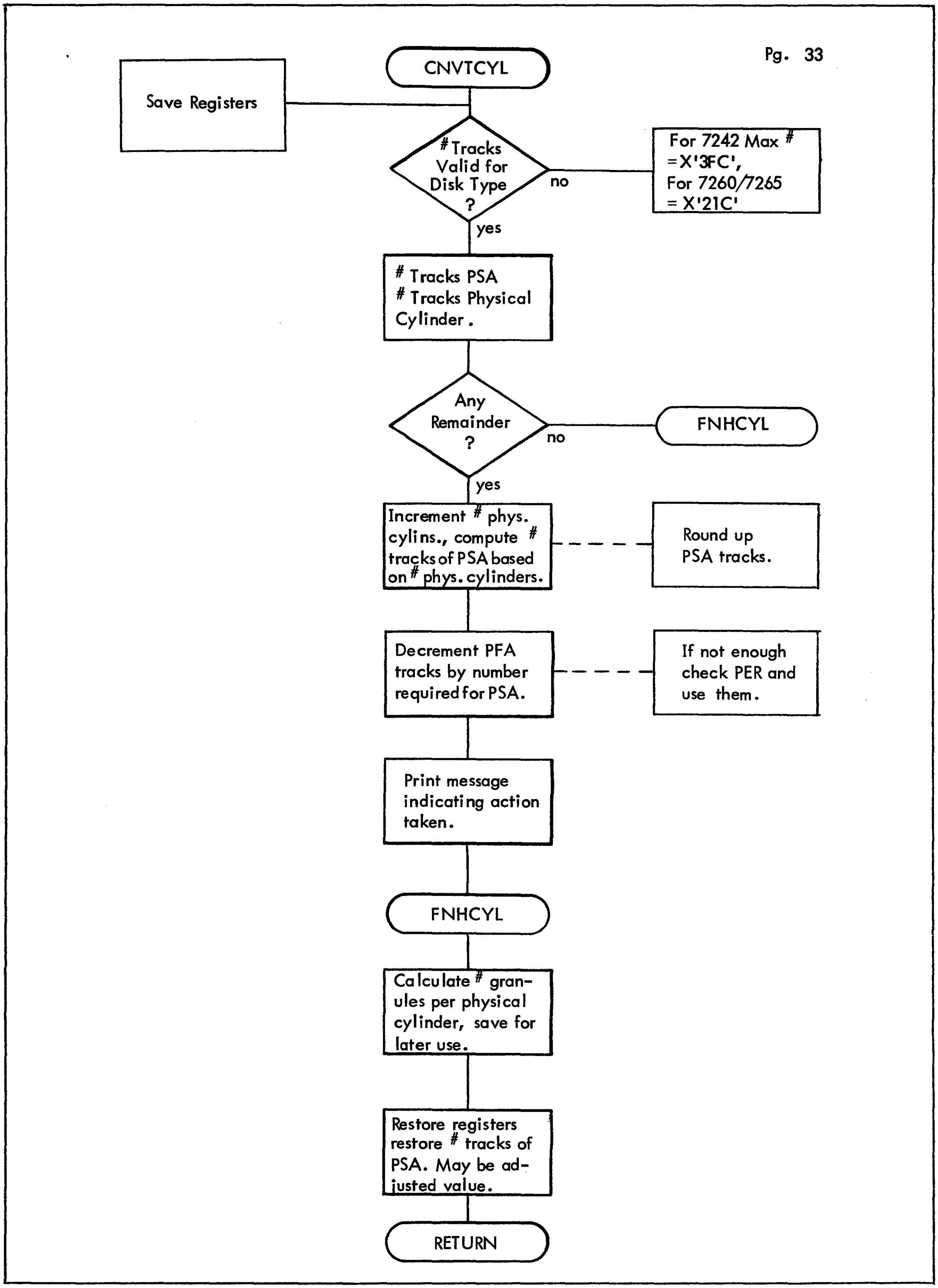

Pg. 33
CNVTCYL
Save Registers
# Tracks Valid for Disk Type ?
no
For 7242 Max # =X'3FC',
For 7260/7265 = X'21C'
yes
# Tracks PSA
# Tracks Physical Cylinder.
Any Remainder ?
no
FNHCYL
yes
Increment # phys. cylins., compute # tracks of PSA based on # phys. cylinders.
Round up PSA tracks.
Decrement PFA tracks by number required for PSA.
If not enough check PER and use them.
Print message indicating action taken.
FNHCYL
Calculate # granules per physical cylinder, save for later use.
Restore registers restore # tracks of PSA. May be adjusted value.
RETURN

Figure 2-9. Flow Diagram of UBCHAN (cont.)

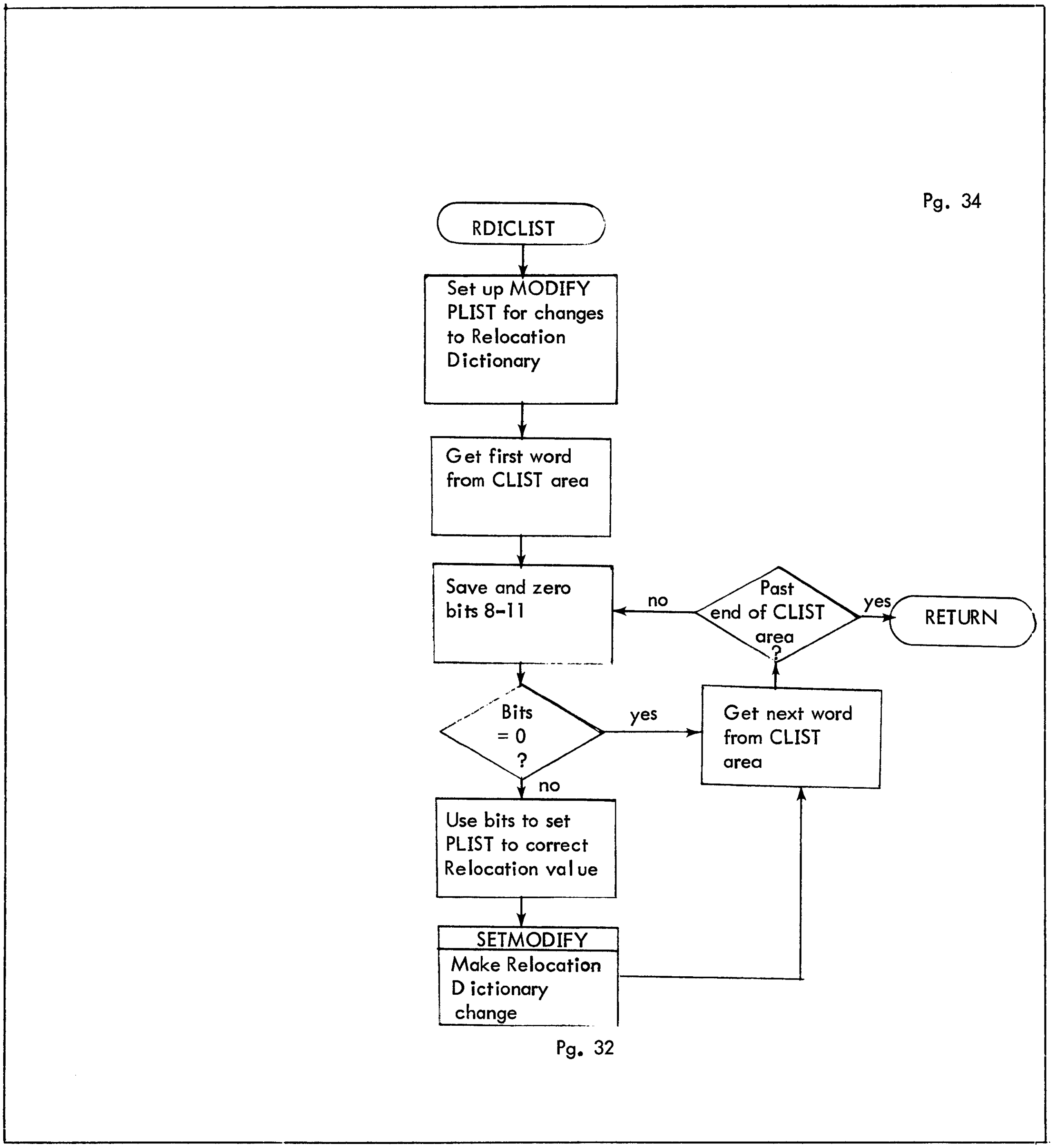

Pg. 34
RDICLIST
Set up MODIFY PLIST for changes to Relocation Dictionary
Get first word from CLIST area
Save and zero bits 8-11
no
Past end of CLIST area ?
yes
RETURN
Bits = 0 ?
yes
Get next word from CLIST area
no
Use bits to set PLIST to correct Relocation value
SETMODIFY
Make Relocation Dictionary change
Pg. 32

Figure 2-9. Flow Diagram of UBCHAN (Cont.)

2.3 SDEVICE

2.3.1 Purpose

To process the PASS2 :SDEVICE command and generate the load module M:SDEV. This load module defines the devices controlled by the symbionts.

2.3.2 Usage

```
     B    SDEVICE
          With   R7 pointing to the control card PLIST
                 R0 pointing to the temp stack pointer
                 R3 pointing to PASS2 stack data
                 R3 and R7 are saved
          Return is to SDVRETRN in P2CCI
          Error return is to the Monitor.
```

2.3.3 Input

Control card (:SDEVICE) image

2.3.4 Output

R6 contains the number of symbiont devices defined on the :SDEVICE command

M:SDEV load module (Table 2-6)

Table 2-6. M:SDEV Load Module

Label	Entry Size (Wds)	Length N=# Entries[†]	Contents or Value	Target System
SODV	VALUE	-	# of OSSEG YYNDDs plus 2 for each remote batch device	BOTH
SSTAT	1/4	N+3	0	BOTH
SRET	1	N+2	0	BOTH
SNDDX	1/4	N+3	1st entry = N+2 Others = DCT1 Index	BOTH BOTH
SSIG	1/4	N+3	0	BOTH
SCNTXT	1/2	N+3	0	BOTH
SQHD	1	1	0	BOTH
SQTL	1	1	0	BOTH
SQUE	1/4	N+3	0	BOTH
SCDA	1	N+2	0	BPM/BTM only
SSDA	1	N+2	0	BPM/BTM only
SYMX	1/4	N+3	See Insert	BOTH

[†] #Entries includes devices defined on :SDEVICE command plus 2 entries for each Remote Batch device (RBndd) defined via :DEVICE commands.

where

SYMX contains:

For UTS

Entry 0	= 0
Entry 1 through N	= 1 if ISSEG 2 if OSSEG
Last entry-1	= 3
Last entry	= 2

For BPM/BTM

Entry 0	= 0
Entry 1 through N	= REF ISSEG or OSSEG
Last Entry - 1	= REF SFSEG
Last Entry	= REF OSSEG

all entires for remote batch devices are set to 1 (UTS) or a REF to ISSEG (BPM/BTM)

2.3.5 Subroutines Used

CHARSCAN	(Used to check a specific character for legal syntax)
NAMSCAN	(Used to scan a field containing a name)
QUOTSCAN	(Used to scan a field containing a keyword)
MODIFY	(Used to generate M:SDEV load module)
PRINTMSG OUTLLERR	(display error information)

2.3.6 Other External References

SDVRETRN	-	return point from SDEVICE processor.
SDVBEGIN	-	base of work area in PASS2 stack
SDVEND	-	end of work area in PASS2 stack
DCTSIZE	-	length of DCT tables
DCT1PTR	-	pointer to DCT1 Table
DCT4PTR	-	pointer to DCT4 Table
TYPMNPNT	-	pointer to type mnemonic table
SDEVFLG	-	number of Symbiont devices (set by SDEVICE)
#RBTS	-	number of Remote batch devices defined via :DEVICE commands
LORBIN	-	DCT1 index of the first Remote batch device defined

2.3.7 Description

The SDEVICE processor is entered when P2CCI encounters a :SDEVICE command which must have been preceeded by valid :CHAN/:DEVICE commands. The work area available for generating M:SDEV is allocated by UBCHAN and its boundaries are located in PASS2's temp stack and referenced by the labels SDVBEGIN and SDVEND.

SDEVICE first determines via #RBTS if any Rembot batch devices have been defined. If there are any on the target system, LORBIN is used as an index into DCT1 table to obtain the connect X'ndd' which is merged with 'RB' and stored in the intermediate table with the symbiont name ISSEG. A count symbiont devices is incremented for each entry.

The syntax for the :SDEVICE command is:

:SDEVICE (LMN, MANE, YYNDD, YYNDD...), (LMN,...)

SDEVICE begins processing a parenthetical expression by requiring the keyword "LMN". The name which follows is that of a symbiont which controls the input or output of the device(s) identified by YYNDD. The symbiont names are ISSEG (input) and OSSEG (output). The "NAME" is obtained and syntactically checked (1 to 7 alphanumeric characters, one of which is alpha) and if legal, is saved in an intermediate table (See Table 2-7). The next field "YYNDD" is obtained and is validated by checking certain tables in the work area built by UBCHAN. Note if YY=RB, it is not processed as it is already in the table. The DCT1 table (pointed to by DCT1PTR) is searched for the value "NDD". If none is found, the "YYNDD" is invalid and SDEVICE skips to the closing parenthesis and then continues processing. However, if found, the index to the value in DCT1 is used as an index to the DCT4 table (pointed to by DCT4 PTR) where a value is obtained which in turn is an index into TYPMNE (pointed to by TYPMNPNT).

The value "YY" is then checked against the indexed value in the TYPMNE. If not equal, SDEVICE continues the search of the DCT1 table for another "NDD" equivalent. If the "YY" value is found, the "YYNDD" is saved in the intermediate table and the number of entries in the table is incremented by one.

SDEVICE then obtains the next "YYNDD". If there is none, the current parenthetical field is terminated and the syntax scan begins with the next parenthetical field. If there is another "YYNDD" in the current field, the "name" (ISSEG/OSSEG) is obtained from the previous "YYNDD" and is entered into the intermediate table. Processing continues as previously described.

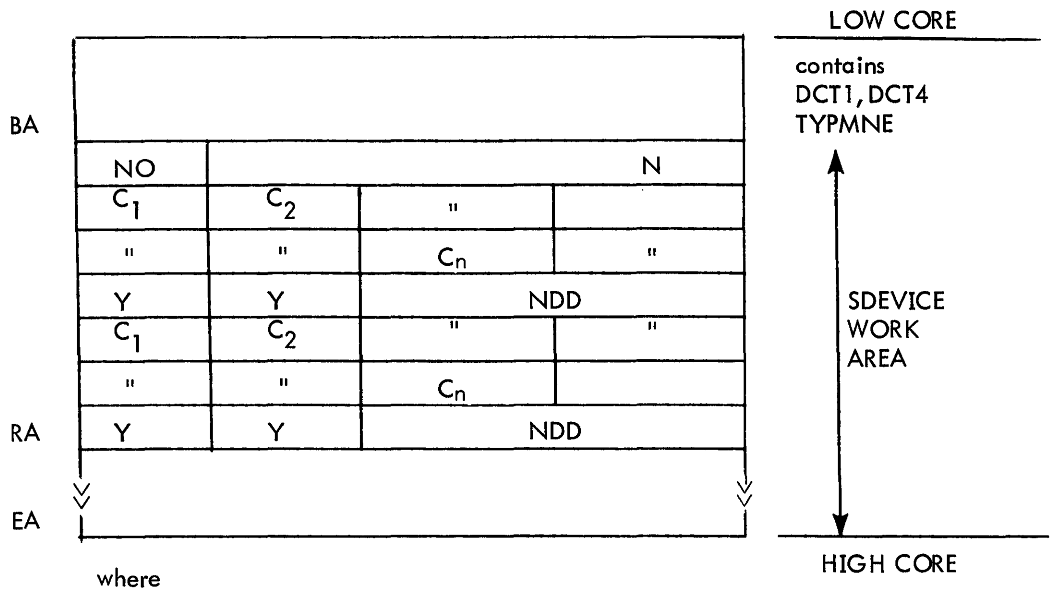
To complete processing SDEVICE, it sets up PLISTs and allocates and initializes the load module areas in the working storage area, and proceeds to generate SECT.00 using the information from the intermediate table and DCT1, DCT4, and TYPMNE tables.

The tables generated for M:SDEV are listed in the OUTPUT section of this chapter. Note that in creating SNDDX, the entries in the intermediate table are processed in the order of occurrence and, therefore, the SNDDX table is ordered accordingly. As each entry is generated, a count of the number of "OSSEG" references is maintained.

The contents of the byte table SYMX depend on whether a BPM/BTM or UTS system is being generated. For both the first and last two entries are null. For a UTS system, the entries are set to 1 for an "ISSEG" name and to 2 for an "OSSEG" name, and the first null entry at the end of the table is set to 3 and the second null entry to 2. For a BPM/BTM system, the entries in SYMX are referenced by the REFs ISSEG and OSSEG as encountered in the intermediate table. Therefore, the destination of each REF is a specific byte in SYMX The first null entry at the end of the table is the destination for the REF SFSEG and the second null entry for the REF OSSEG.

When the generation of the tables is completed, the load module M:SDEV is written. SDEVICE then puts the number of symbionts in Register 6 and checks for error conditions having been detected (ABORTFLG $\neq$ 0). If errors have occurred, SDEVICE prints message and exits back to the monitor. If error-free, SDEVICE exits back to SDVRETRN in P2CCI and PASS2 continues processing.

Table 2-7. INTERMEDIATE NAME TABLE



where

N = number of entries in intermediate table (3 words/entry)

C_1 C_2. . C_n = characters in name (i.e., ISSEG, OSSEG)

YY= EBCDIC "YY" from YYNDD value

ndd = hexadecimal equivalent of "NDD" from YYNDD value

BA = base address of work area pointed to by SDVBEGIN

EA = end address plus 1 work area pointed to by SDVEND

RA = base address of remainder of work area which is used in generating M:SDEV and is forced to a doubleword boundary.

NO = number of OSSEG name references

2.3.8 SDEVICE Messages

INVALID SYMBIONT NAME

The "name" field is either non-alphanumeric or is greater than seven characters long. SDEVICE skips to end of parenthetical field and continues processing.

INVALID KEYWORD

The keyword "LMN" is expected but not found. SDEVICE skips to end of parenthetical field and continues processing.

SYNTAX ERROR

A open parenthesis, a closing parenthesis, a comma after "LMN" or "name" is expected but not found; end of command cannot be found;or a "YYNDD" is encountered where YY = 9T, 7T, or DC and "DD" is less than X'80' or YY is not equal to 9T, 7T, or DC and "DD" is greater than or equal to X'80'. SDEVICE skips to end of parenthetical field and continues processing.

INVALID 'YYNDD'

The "YYNDD" is not alphanumeric, is greater than five characters long, is not defined in DCT1 table, the value "N" is not from A through H, or the value "DD" is not hexadecimal. SDEVICE skips to end of parenthetical field and continues processing.

NO ROOM LEFT FOR :SDEVICE

There is no work area left for generating the intermediate table. SDEVICE displays abort message and exits back to the Monitor.

REMAINDER OF CC IGNORED

This message appears with previously described messages if SDEVICE cannot find end of current parenthetical field. SDEVICE continues by attempting to generate load module.

MODIFY ERROR

There is not enough work area available to generate load module. SDEVICE displays abort message and exits back to the Monitor.

'SDEVICE' ABORTED

Message is displayed in previously described error conditions.

2.3.9 Internal Routines

SDEVICE

Main entry, initialize and control.

SDEVICE0 Process next parenthetical field.

SDEV0 Process next "YYNDD".

WRTMSDEV Load module generation completed, write it to "M:SDEV" file.

WRITEF Perform actual write of load module.

Register 1 = buffer address.

Register 2 = buffer size (bytes).

Register 3 = key address of load module record.

VALID Check current "YYNDD" against available devices as specified by DCT1PTR, DCT4PTR, and TYPMNPNT and respective tables DCT1, DCT4, and TYPMNE.

Register 1 = "YYNDD", with "NDD" converted from EBCDIC to hexadecimal.

ROOM Check if any work area remains for intermediate table.

Register 5 = address of next available entry.

Register 6 = address of last available entry plus one.

DEFX Set up DEF PLIST for MODIFY routine for generation of an external definition.

Register 1 = address of NAME_1.

Register 13 = VALUE_2.

EXPRX Set up EXPR PLIST for MODIFY routine for generation of an external reference.

Register 1 = address of NAME_2.

Register 13 = VALUE_1.

STADDR Call MODIFY to perform the function DEF or EXPR.

Register 4 = address of PLIST needed by Master PLIST.

STREGS Save contents of registers 12 through 14, and 1 through 4.

BUILDCDT Set up Master PLIST with address supplied.

Register 13 = relative word address to be stored in Master PLIST.

Register 12 = double word address to be stored in Master PLIST.

Register 2 = half word address of Master PLIST entry where final address is to be set.

Register 5 = word address of base of work area.

Special error routines include: ERR1, ERRCOMON, ERR1X, ERR2, ERR3, ERR3A, ERRX, ERR4, ERR5, ERR6, ERR7, ERR8, ERR9, ERRA1, ERRA11, ERRA2, ERRZZ, RECOVER, MODERR, ERR12.

2.3.10 Flow Chart

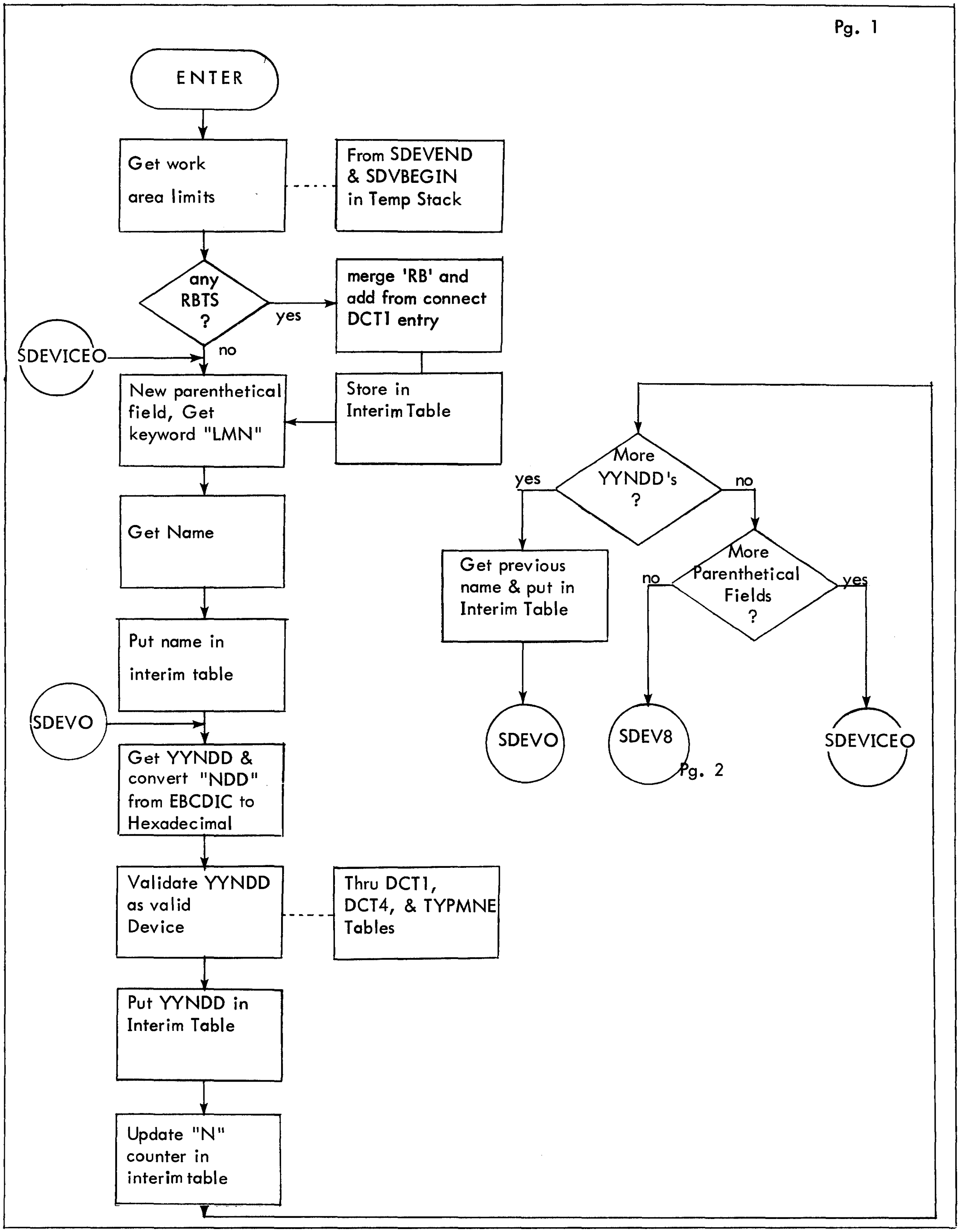

Pg. 1
ENTER
Get work area limits
From SDEVEND & SDVBEGIN in Temp Stack
any RBTS ?
yes
merge 'RB' and add from connect DCT1 entry
no
SDEVICEO
New parenthetical field, Get keyword "LMN"
Store in Interim Table
Get Name
Put name in interim table
SDEVO
Get YYNDD & convert "NDD" from EBCDIC to Hexadecimal
Validate YYNDD as valid Device
Thru DCT1, DCT4, & TYPMNE Tables
Put YYNDD in Interim Table
Update "N" counter in interim table
More YYNDD's ?
yes
no
Get previous name & put in Interim Table
More Parenthetical Fields ?
no
yes
SDEVO
SDEV8
Pg. 2
SDEVICEO

Figure 2-10. Flow Diagram of SDEVICE

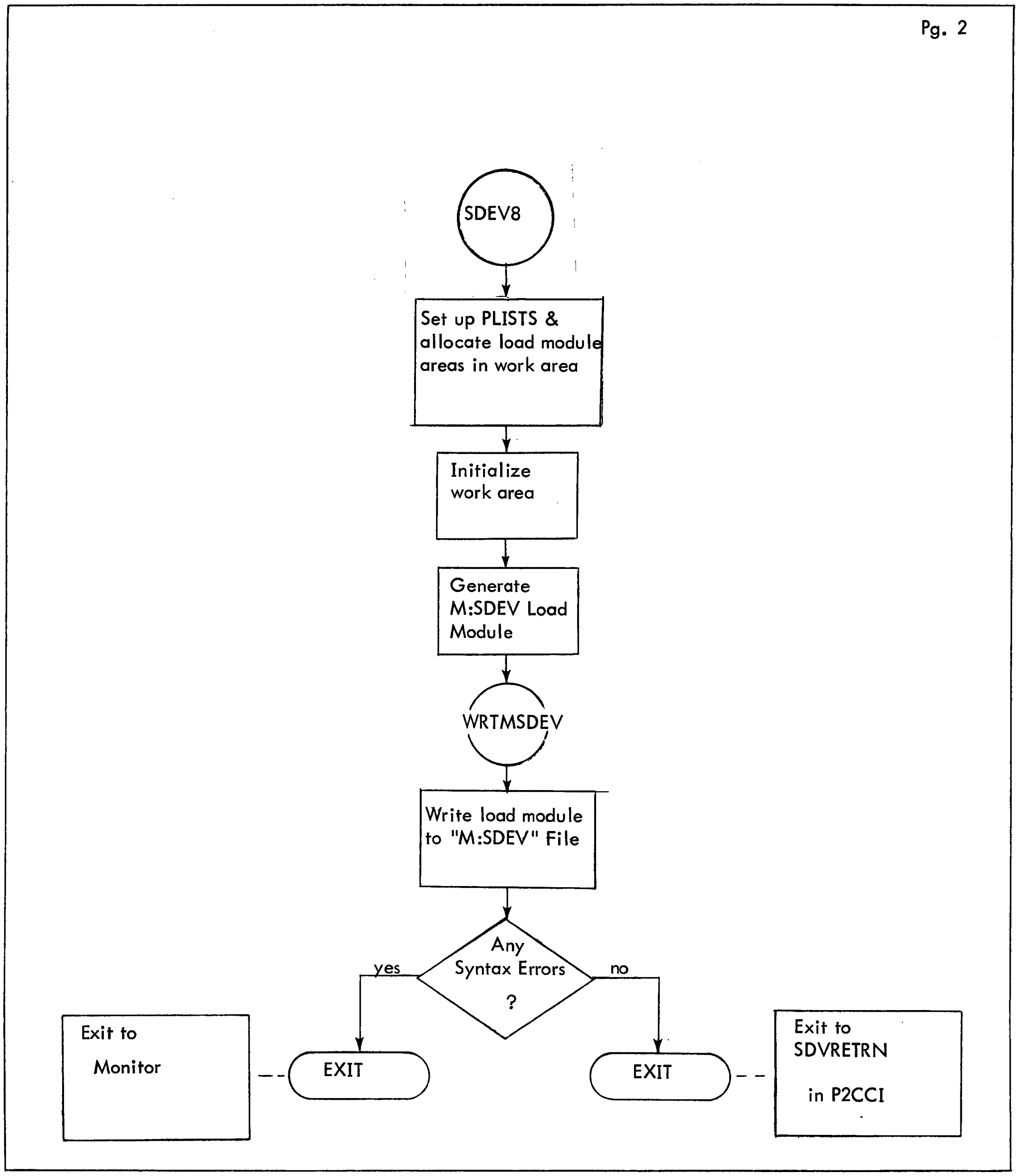

Pg. 2
SDEV8
Set up PLISTS & allocate load module areas in work area
Initialize work area
Generate M:SDEV Load Module
WRTMSDEV
Write load module to "M:SDEV" File
Any Syntax Errors ?
yes
no
EXIT
EXIT
Exit to Monitor
Exit to SDVRETRN in P2CCI

Figure 2-10. Flow Diagram of SDEVICE (Cont.)

2.4 XMONITOR

2.4.1 Purpose

To process MONITOR or UTM PASS2 control commands, creating the M:CPU and (For UTS) M:SYMB, and M:BIG9 load modules, the MON::ORG object module, and an updated SPEC:HAND file if SIG9 and or ANS specified on the UTM command.

2.4.2 Usage

B MONITOR (MONITOR command)

B UBMONITOR (UTM command)

with: R7 pointing to control card PLIST

R0 pointing to temp stack pointer

R3 pointing to PASS2 stack data

R6 containing the number of symbiont devices (BPM/BTM only)

R0 and R3 are saved

Return is to READSTRG in P2CCI

2.4.3 Input

Control card (:MONITOR or :UTM)

SDGANSG - to create SDGA value

#RBTS - number of remote batch devices defined via :DEVICE commands

LORBIN - DCT1 index of first remote batch device defined

#PRDP - number of private disc packs defined

DEVS - contains # private pack, #7 track and #9 track tapes

BIG9FLG - Set by XMONITOR for UTS systems if BIG9 option specified

2.4.4 Output

M:CPU load module (Table 2-8)

MON:ORG object module - is of length 2*((ORG+1)/2) reserved words with the last one equated to MONORG label. ORG is the keyword parameter on :UTM or :MONITOR

M:SYMB Module (Table 2-9) UTS only

SPEC:HAND file - If SIG9 is present on :UTM command, the handler name S9TRAPS and 1 or if ANS is specified on :UTM command the name ANSTP is added to the HANDLERS record of this file.

M:BIG9 module UTS only - contains an absolute DEF :9 only - This DEF is 1 if BIG9 option specified on UTM command, otherwise :9 = 0.

Table 2-8. Contents of M:CPU Load Module

Label	Entry Size (words)	Length	Contents/Value (in terms of Keywords)	Target System
1. Absolute DEFs				
MAXBM	Value	-	MPOOL-FMPOOL(if >0)	BPM/BTM
MAXCFU	Value	-	CFU	BPM/BTM
MAXBKGDCFU	Value	-	CFU-FCFU (if > 0)	BPM/BTM
MAXBQ	Value	-	QUEUE - FQUEUE (if > 0)	BPM/BTM
	Value	-	QUEUE	UTS
SDGA	Value	-	# Granules PER defined	BOTH
CORE	Value	-	CORE in words	BOTH
BCRBID[1]	Value	-	length in bytes of significant words of RBB'ID table	BPM/BTM
SSSIZE	Value	-	Sector Size	UTS
SBSECTS	Value	-	number of buffers/sector	UTS
SGSIZE	Value	-	number of sectors/granule	UTS
ANS	Value	-	1 if ANS specified 0 if ANS not specified	UTS
LAVRFMT[2]	Value	-	number of entries in AVRFNMT table	UTS
LCLX	Value	-	0 if no Remote batch devices on system else is highest RBT DCT1 index +1	UTS
RBLIMSZ[1]	Value	-	number of RBTs defined	UTS
RBLIMSIX[1]	Value	-	DCT1 index of first RBT defined	UTS
2. Tables				
TSTACK	1	TSTACK+2	Stack pointer followed by TSTACK words containing 0	BPM/BTM
ABSBASE	1	1	0	BPM/BTM
ABSEND	1	1	0	BPM/BTM
MPOOL	1	(34*MPOOL)+2	For both systems, 1st word on DW boundary contains address of 1st MPOOL buffer (also starting on DW boundary) and the 1st word of each buffer contains address of next buffer or 0 if last buffer	UTS
		(34*MPOOL-2)+2		BPM/BTM
SPOOL	1	(256*SPOOL)+2	Same as MPOOL except for buffer size	BPM/BTM
CPOOL	1	(40*CPOOL)+2	Same as MPOOL except for buffer size	BOTH
CPOOLEND	1	1	represents address of end of CPOOL table	BOTH

[1] Generated only if remote batch devices is defined for the Target System.

[2] Generated only if ANS is specified.

Table 2-8. Contents of M:CPU Load Module (cont.)

Label	Entry Size (words)	Length	Contents/Value (in terms of Keywords)	Target System
SYMFILE	1	SFIL+1	0 except first word contains SFIL	BPM/BTM
SYMFSDA	1	SFIL+1	0	BPM/BTM
RBSYMFID	1/4	SFIL+1	0	BPM/BTM
IOQ1	1/4	QUEUE+1	0	BOTH
IOQ2 QFREE	1/4	QUEUE+1	each entry= (Byte displacement from IOQ2)+1 except last entry = 0	BOTH
IOQ3	1/4	QUEUE+1	0	BOTH
IOQ4	1/4	QUEUE+1	0	BOTH
IOQ5	1/4	QUEUE+1	0	BOTH
IOQ6	1	QUEUE+1	0	BOTH
IOQ7	1/4	QUEUE+1	0	BOTH
IOQ8	1	QUEUE+1	0	BOTH
IOQ9	1/2	QUEUE+1	0	BOTH
IOQ10	1/4	QUEUE+1	0	BOTH
IOQ11	1/4	QUEUE+1	0	BOTH
IOQ12	1	QUEUE+1	0	BOTH
IOQ13	2	QUEUE+1	0	BOTH
IOQ14	1/4	QUEUE+1	0	BOTH
IOQ15	1/4	QUEUE+1	0	UTS
ACNCFU	1	19	0	BOTH
FILCFU	1	19	0	BOTH
BGRCFU	1	19*(CFU-1)	0	BOTH
LASTCFU	1	19	0	BOTH
MPATCH	1	MPATCH	0(if MPATCH > 0) starts on DW boundary	BOTH
JIT	-	153 + CORE (in 8K units)	All 0 except word 71 contains $+4 words 72-73 - M:OC(TEXTC) word 72 contains $+2 words 76-77 (M:OC DCB) contains X'200003', X60002' word 108 contains X'F0404040' words 153 to END contain all 1 bits	BPM/BTM
MX:PPUT	1/4 or 1/2	CORE (in pages) + 8 or 16 words (overleid)	The first 32 entries (8 or 16 words) overlay MPATCH or LASTCFU Remaining bytes on halfwords are forward linked same as IOQ2	UTS
RB:XFLG[1]	1	1	0	BOTH
RBB:ID	1	1	0 if no RBTs on system	BOTH
	1/4	highest RBT DCT1 index highest RBT DCT1 index +	Significant entries (containing 0) only in the RBT DCT1 index range (plus 1 for UTS)	BPM UTS
WARBBID[1]	1	1	Address of table points to word containing first significant entry of RBB:ID	BPM/BTM

[1] Generated only if remote batch devices defined on target system.

Table 2-8. Contents of M:CPU Load Module

Label	Entry Size (words)	Length	Contents/Value (in terms of Keywords)	Target System
RBD:WSN[1]	2	Highest RBT DCT1 index	Significant entries (containing 0) only in the DCT1 index range of RBTs	UTS
SITEID	2	2	name identified by SITE Keyword or blank	UTS
AVRFNMT[2]	6	#tapes defined on system	0	UTS
ANSFLGS[2]	1/4	#tapes defined on system	0	UTS
ANSPRT[2]	1	1	1 if ANSPROT 0 if keyword not specified	UTS
M:FPPH[3]	1	1	X'20' (first linked entry in MX:PPUT)	UTS
M:FPPT	1	1	CORE (in pages)-1	
M:FPPC	1	1	CORE (in pages)-X'20'	
BW:CHG	1	1	0	UTS
BL:IFS	1	1	(INFILE-1	UTS
BL:OFS	1	1	OUTFILE-1	UTS
LSERIAL	1/4	(((AVGSER*16) +3)/4)+17	(Byte displacement from LSERIAL)+1 except last entry = 0	UTS
TSERIAL	1	(((AVGSER*16) +3)/4)+17	entry 0 = # entries in LSERIAL all other entries = 0	UTS
PARPSD[4]	1	1	DW boundary PARXXXX (PREF)	UTS
PARITYCC[4]	1	1	0	UTS
PARERPSD[4]	2	4	words 0-1 = 0 word 2 = PARITYER(PREF) word 3 = X'17000000'	UTS
BUSER10[4]	2	2	word 0 = X'00400000'+ BUSER1(PREF)	UTS
PSDTEMP[4]	2	2	0	UTS
BUSTEMP[4]	2	2	0	UTS
IETPSD[4]	2	2	word 0 = IETXXXX(PREF) word 1 = 0	UTS
INSTXPSD[4]	2	4	words 0-1 = 0 word 2 = INSTXCPT (PREF) word 3 = X'17000000'	UTS
MEMFTPSD[4]	2	4	words 0-1 = 0 word 2 = MEMFAULT (PREF) word 3 = X'17000000'	UTS
PARXPSD[4]	2	2	word 0 = PARXX1 (PREF) word 1 = X'07000000'	UTS

1. Generated only if remote batch devices defined on target system
2. Generated only if ANS specified ANSPROT. Keyword ignored if ANS not specified also
3. These words are order dependent
4. Generated only if SIG9 or BIG9 option specified

Table 2-9. Contents of M:SYMB Load Module

Label	Entry Size (words)	Length	Contents/Value in terms of Keywords
1. Absolute DEFS			
MFS	Value	-	2 + 2 * #RBTs defined on system
OUTFIL	Value	-	OUTFILE + MFS
INFIL	Value	-	INFILE
1. Tables			
BD:ACCT	2	INFIL+1	0
BW:RES	1	INFIL+1	0
BH:TIME	1/2	INFIL+1	0
BH:PART	1/4	INFIL+1	0
BH:SLNK	1/2	INFIL+1	0
BH:XLNK	1/2	INFIL+1	0
BH:HPRI[1]	1/2	X'23' (HW)	all entries 0 except X'22' = 1 X'23' = INFIL+1
BH:TPRI	1/2	X'23' (HW)	all entries 0 except X'22' = INFIL X'23' = OUTFIL+INFIL
BW:SDA	1	INFIL+OUTFIL+1	0
BH:SID	1/2	INFIL+OUTFIL+1	0
BB:PI BB:DEV	1/4	INFIL+OUTFIL+1	0
BH:LINK	1/2	INFIL+OUTFIL+1	Entry 0 = 0 Entry 1 through INFIL -1 are (1+ displacement from BH:LINK) Entries INFIL + 1 through INFIL + OUTFIL -1 are (1+ displacement from BH:LINK) Entries INFIL and INFIL + OUTFIL are 0
BB:RID	1/4	INFIL+OUTFIL+1	0
S#H:LNK	1/2	(((INFIL*AVGSER) +3)/4)+1	0
S#W:SER	1	(((INFIL*AVGSER) +3)/4)+1	
RB:SPMF	1	1 # of RBTs + 1	0 if no RBTs defined significant entries (contianing 0) in range of DCT1 index for first RBT through LCLX
RB:MFAD	1	1 # of RBTs + 1	0 if no RBTs defined significent entries (containing 0) in range of DCT1 index for first RBT through LCLX
RBB:MFC[2]	1/4	#RBTs+1	Significent entries (containing 0) only in range of DCT1 index for first RBT through LCLX
RBB:MXP[2]	1/4	#RBTs=1	Significent entries (containing 0) only in range of DCT1 index for first RBT through LCLX
SYMND	1	1	address is that of test cell in M:SYMB

[1] Tables are order dependent

[2] Generated only if Remote batch devices defined for target system

2.4.5 Interaction

M:OPEN, M:READ, M:WRITE, M:CLOSE are used to write MON::ORG and to update SPEC:HAND.

SYNTAX is used to interpret input control card.

COREALLOC is used to allocate memory for M:CPU and M:SYMB.

MODGEN is used to manipulate REFDEF stacks, EXPRESSION stacks, and RELOCATION DICTIONARIES.

WRITELM is used to write M:CPU,M:SYMB and M:BIG9.

READSTRG is exit location.

2.4.6 Data Bases

KWDTBLO is a two-part input table for SYNTAX. The first part is a set of SYNTAX control halfwords that prints SYNTAX to process the non-standard format of SITE option on the UTM command as well as the standard format for the rest of the UTM options and all of the MONITOR standard options. The second part of the table is the normal SYNTAX keyword table.

DYNAM is the MONITOR virgin stack data block. It contains entries for UTM-specific keywords, but they are zeroed to cause SYNTAX to reject them.

UDYNAM is the UTM virgin stack data block containing one doubleword table pointer for the SITE option in addition to the normal keyword entries. It causes SYNTAX to reject MONITOR-specific keywords.

ORGROM is a MON::ORG module with ORG set to 0.

2.4.7 Subroutines

BUFGEN generates linked buffers (MPOOL, SPOOL, CPOOL). It is entered via a branch to BUFGEN while under MODGEN control and exits by restoring R10 (MODGEN's controlling register).

2.4.8 Description

If entered at MONITOR and R6 (number of symbiont devices) is non-zero, the default values in DYNAM for SPOOL, CPOOL, and TSTACK are adjusted appropriately. Then, or otherwise, SYNTAX is used to decode the control card. COREALLOC is used to set up the M:CPU module in memory for generation and MODGEN interaction. The M:CPU module is generated using MODGEN, checking where necessary to omit those items not belonging to the particular target system being generated (See Table 2-8). WRITELM creates the M:CPU file. If the BIG9 option has been used, XMONITOR sets BIG9FLG in the stack for subsequent use by the SPROCS and IMC overlays.

Then theORG input value is inserted into ORGROM, the checksum is adjusted and ORGROM is written as the MON::ORG file. Then if the target system is BPM/BTM, XMONITOR returns to P2CCI.

Otherwise, COREALLOC again sets up a module generation environment, and MODGEN is used to generate the M:SYMB module which is then written by WRITELM. If the SIG9 (BIG9) and/or ANS options have been specified the SPEC:HAND file is read into core and the names S9TRAPS and/or ANSTP are added to the HANDLERS record of the file which is then written out to disc. COREALLOC is then used to allocate a module generation enviornment in which only REFDEF entry space is defined and MODGEN is used to define the absolute DEF:9, which is equal to 1 if the BIG9 option has been used on the :UTM command. Otherwise, :9 is set equal to 0. WRITELM then writes the load module M:BIG9 out to disc. XMONITOR then returns to P2CCI at READSTRG.

2.4.9 XMONITOR Messages

***TROUBLE WITH SPEC:HAND–S9TRAPS AND/OR ANSTP NOT INCLUDED

In attempting to open SPEC:HAND file to add the name S9TRAPS and/or ANSTP, an error or abnormal condition was encountered. XMONITOR continues.

***FQUEUE > = QUEUE - FQUEUE IGNORED

The value specified for FQUEUE is equal or greater than specified for QUEUE. XMONITOR continues.

***FMPOOL > = MPOOL- FMPOOL IGNORED

The value for FMPOOL is equal or greater than that of MPOOL+ XMONITOR continues.

***FCFU > = CFU - BKGD HAS NO CFU

The value for FCFU was equal or greater than that specified for CFU. XMONITOR continues.

***ANS NOT SPECIFIED-ANSPROT IGNORED

The keyword ANSPORT was specified but not ANS. XMONITOR continues.

***AVGSER OUT OF RANGE – DEFAULT (1) USED

The value specified for AVGSER is invalid and the default is used. The legal range is 1 to 63 unless no private packs are defined on the system in which instance the minimum may be 0. XMONITOR continues.

***BIG9 SPECIFIED – SIG9 ALSO INCLUDED

The option BIG9 has been specified but not SIG9. SIG9 is also included.XMONITOR continues.

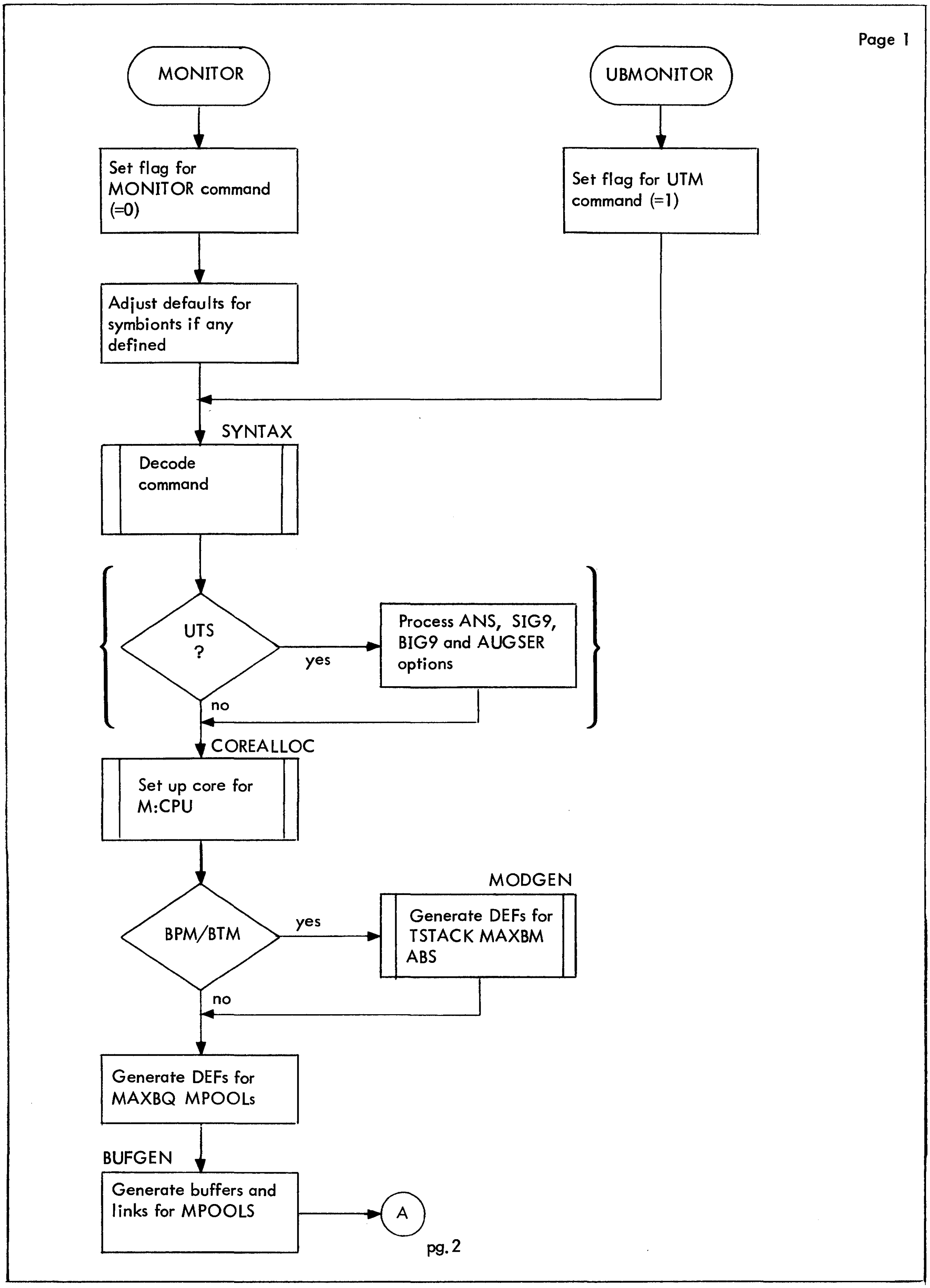

Page 1
MONITOR
UBMONITOR
Set flag for MONITOR command (=0)
Set flag for UTM command (=1)
Adjust defaults for symbionts if any defined
SYNTAX
Decode command
UTS ?
yes
Process ANS, SIG9, BIG9 and AUGSER options
no
COREALLOC
Set up core for M:CPU
MODGEN
BPM/BTM
yes
Generate DEFs for TSTACK MAXBM ABS
no
Generate DEFs for MAXBQ MPOOLs
BUFGEN
Generate buffers and links for MPOOLS
A
pg. 2

Figure 2-11. Flow Diagram of XMONITOR

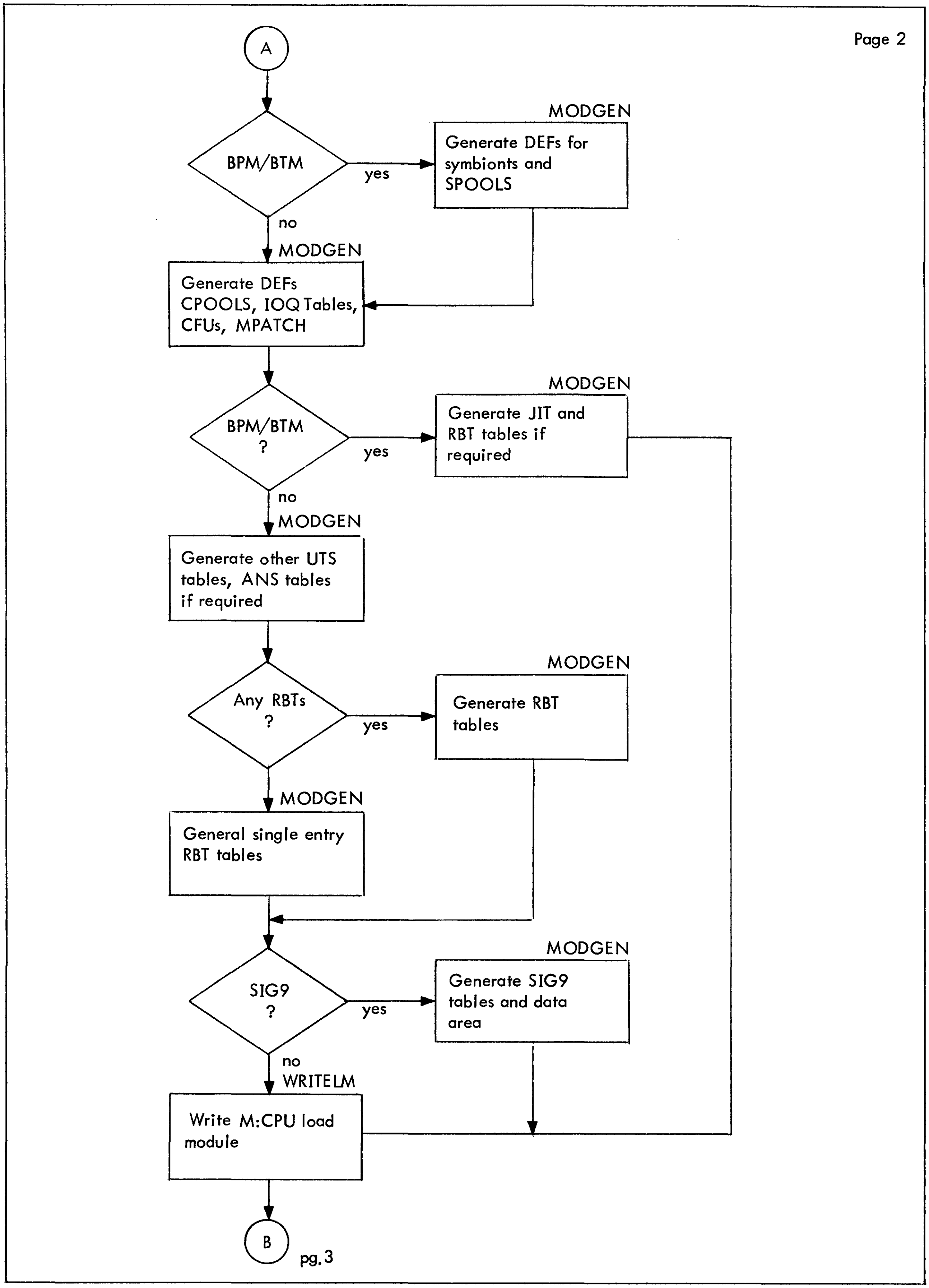

Page 2
A
BPM/BTM
yes
MODGEN
Generate DEFs for symbionts and SPOOLS
no
MODGEN
Generate DEFs CPOOLS, IOQ Tables, CFUs, MPATCH
BPM/BTM ?
yes
MODGEN
Generate JIT and RBT tables if required
no
MODGEN
Generate other UTS tables, ANS tables if required
Any RBTs ?
yes
MODGEN
Generate RBT tables
MODGEN
General single entry RBT tables
SIG9 ?
yes
MODGEN
Generate SIG9 tables and data area
no
WRITELM
Write M:CPU load module
B
pg. 3

Figure 2-11. Flow Diagram of XMONITOR (cont.)

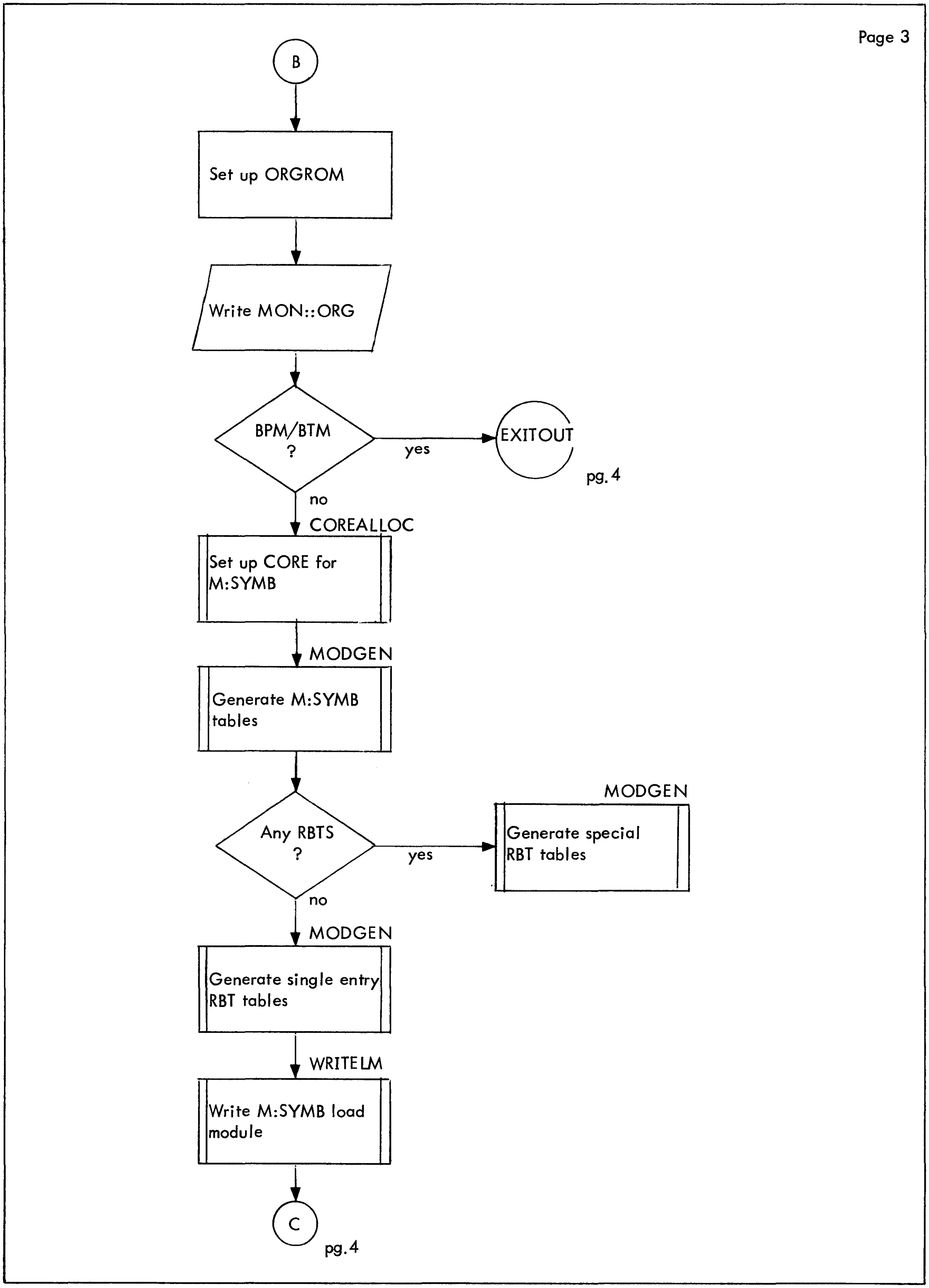


Figure 2-11. Flow Diagram of XMONITOR (cont.)

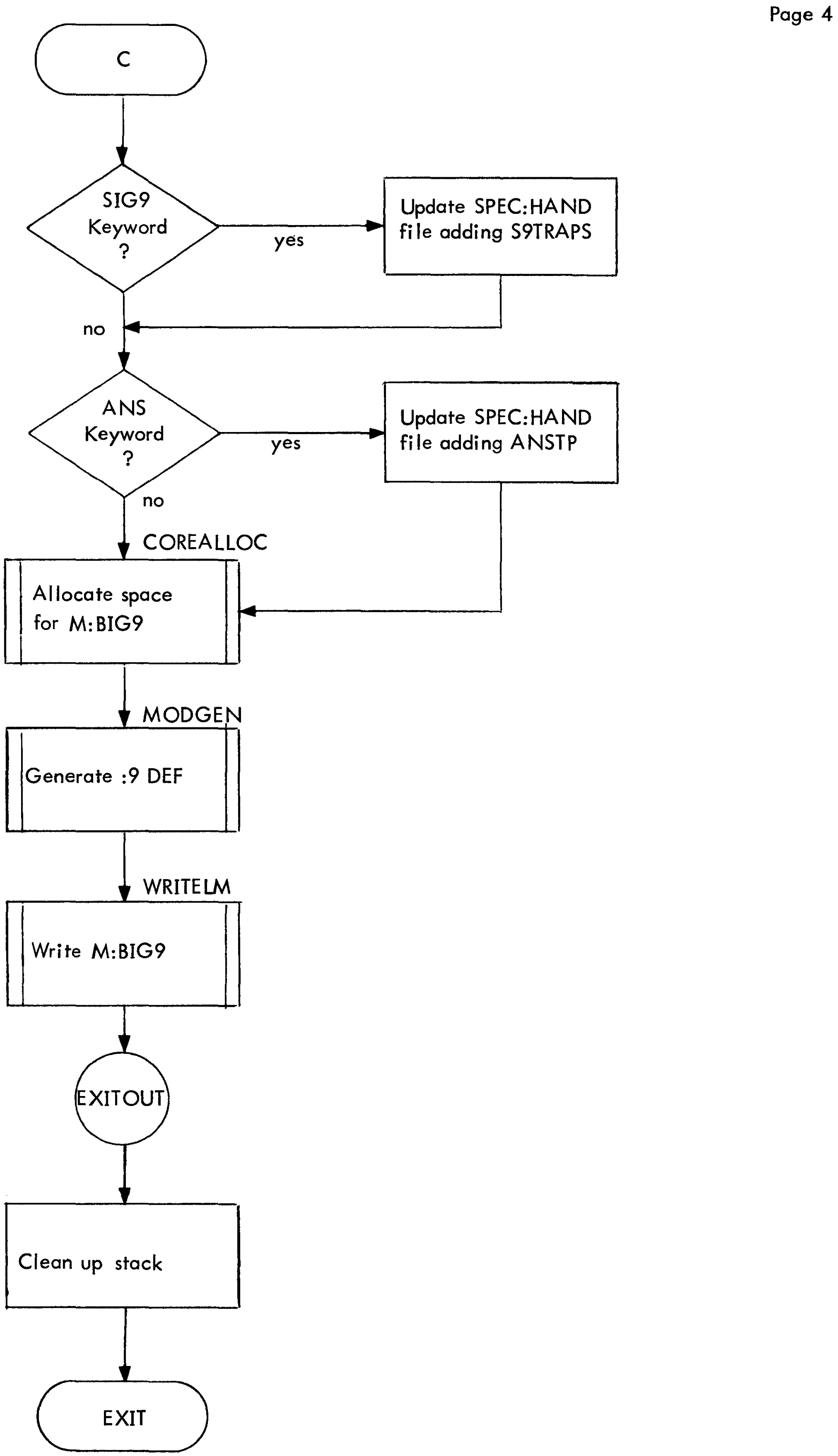


Figure 2-11. Flow Diagram of XMONITOR (cont.)

2.5 XLIMIT

2.5.1 Purpose

To process DLIMIT, BLIMIT, OLIMIT or ELIMIT PASS2 control commands, creating the M:DLIMIT, M:BLIMIT, M:OLIMIT or M:ELIMIT load module.

2.5.2 Usage

B	DLIMIT	(DLIMIT command)
B	UBBLIMIT	(BLIMIT command)
B	UBOLIMIT	(OLIMIT command)
B	UBELIMIT	(ELIMIT command)

2.5.3 Input

Control card (:DLIMIT, :BLIMIT, :OLIMIT, or :ELIMIT) image.

2.5.4 Output

M:DLIMIT load module - BPM/BTM (Table 2-10)

M:OLIMIT/M:BLIMIT/M:ELIMIT load modules - UTS (Table 2-11)

Table 2-10. Contents of M:DLIMIT

Label	Entry Size (Wds)	Size	Contents/Value (in terms of Keywords)	
DLIMTBL	—	—	Base address of M:DLIMIT	
TIMELIM	1	16	TIME	Default limit table, indexed by priority
LOLIM	1	16	LO	
POLIM	1	16	PO	
DOLM	1	16	DO	
UOLIM	1	16	UO	
SCTLIM	1	16	SCRATCH	
FPOOLIM[1]	1	16	FPOOL	
PSTLIM	1	16	PSTORE	
TSTLIM	1	16	TSTORE	
IPOOLIM[2]	1	16	IPOOLIM	

#1. Left half of each FPOOLIM entry contains buffer size (512 words)

#2. Left half of each IPOOLIM entry contains buffer size (256 words)

Table 2-11. Contents of M:OLIMIT/M:BLIMIT/M:ELIMIT

Label[1.]	Size	Contents (in terms of Keywords)
SL:XTIME	1	TIME[2.]
SL:XLO	1	LO
SL:XPO	1	PO
SL:XDO	1	DO
SL:XUO	1	UO
SL:XT[3.]	1	TAPES
SL:XFP[3.]	1	FPOOL
SL:XPS	1	PSTORE
SL:XTS	1	TSTORE
SL:XIP[3.]	1	IPOOL
SL:XC[3.]	1	CORE
SL:XF[3.]	1	FILES
SL:XSP[3.]	1	SP

1. X in label is replaced by 'O' or 'B' or 'E'
2. SL:OTIME is always zero.
3. Not generated in M:ELIMIT

2.5.5 Interaction

SYNTAX	is used to convert control card to stack data blocks.
COREALLOC	is used to allocate dynamic data pages.
MODGEN	is used to generate DEFs.
WRITELM	is used to write the output module.
READSTRG	is exit location.

2.5.6 Data Bases

Initially:

UKWD and DKWD are overlapping keyword tables for SYNTAX. UKWD is used for BLIMIT and OLIMIT and ELIMIT; DKWD for DLIMIT.

DSYS	is the DLIMIT virgin stack data block for SYNTAX (containing default values).
OSYS	is the OLIMIT virgin stack data block.
BSYS	is the BLIMIT virgin stack data block.
ESYS	is the ELIMIT virgin stack data block.

DNAMES, ONAMES, BNAMES and ENAMES are the portions of code (interlaced with TEXTCs) which are fed to MODGEN to generate the corresponding load modules.

After SYNTAX:

Stack data block (now in the stack and pointed to by R5) for OLIMIT, BLIMIT and ELIMIT contains an image of the output module data record.

For DLIMIT, R5 points to a number of data blocks equal to one more than the number of PRTY options encountered in the :DLIMIT command. Each block contains a word for each kind of limit (UO, PO, etc.), and a word for the PRTY value (priority). PRTY is a type 4 keyword with a default of -1, so the first block has a PRTY value of -1 and positive values (either default or command - specified) for all limits. Every other block has positive values only for those limits command - specified for the block's PRTY value. If the PRTY value in the block is negative, it is less than -1 and indicates that the command - specified value was too large or had been previously specified.

2.5.7 Description

Each entry point sets a flag (0, 1 or 2) in R4 and branches to LIMIT. LIMIT sets input pointers for SYNTAX and BALs to it. Upon return, the values specified for SCRATCH and SP (UTS only) are examined to determine if they are equal to or less than the total number of tapes units and private disc packs specified via :DEVICE commands. If not, error routines are entered that change the value to the maximum number specified and print an appropriate message and return. For the :ELIMIT command, all parameters are checked to determine that no value specified exceeds the maximum permissible value. Any error detected causes a message to be generated and the default value substituted. A BAL to COREALLOC allocates core. The address of the proper name table (DNAMES, ONAMES, BNAMES or ENAMES) is put in R10 and a branch to MODGEN generates the required DEFs. Then, for BLIMIT OLIMIT and ELIMIT a branch to BUILD moves the stack data block to the output module data record and drops into WRITE, which points R14 to the proper filename, BALs to WRITELM, cleans data block(s) out of the stack, and exits to READSTRG. For DLIMIT, BUILD is replaced by a routine which moves values from the first data block to the module data record for all priorities, and then for every other data block with non-negative PRTY values, moves non-negative limit values to the data record only for that priority. Then, a branch to WRITE finishes up and exits.

**SCRATCH TAPES > TOTAL ON SYSTEM --
*XXXX*USED FOR XXXXXXXPRIORITY

For the designated priority, the number of tapes specified for SCRATCH exceeds the total numbers of tape drives (9T and 7T) defined via :DEVICE commands. XLIMIT substitutes the *XXXX* value and continues. (BPM/BTM only)

**TAPES > TOTAL ON SYSTEM --
XXXX USED
**DISC PACKS > TOTAL ON SYSTEM --
XXXX USED

The number of Tapes/Disc Pack (SP) specified exceeds the total number of tape drives (9T and 7T) disc packs defined via :DEVICE commands. XLIMIT substitutes the *XXXX* value and continues. (UTS only)

**XXXXXXXX INVALID–
XXXX USED

The parameter (XXXXXXXX) on the :ELIMIT command is in error and the value (XXXX) has been substituted

2.5.8 Flow Chart

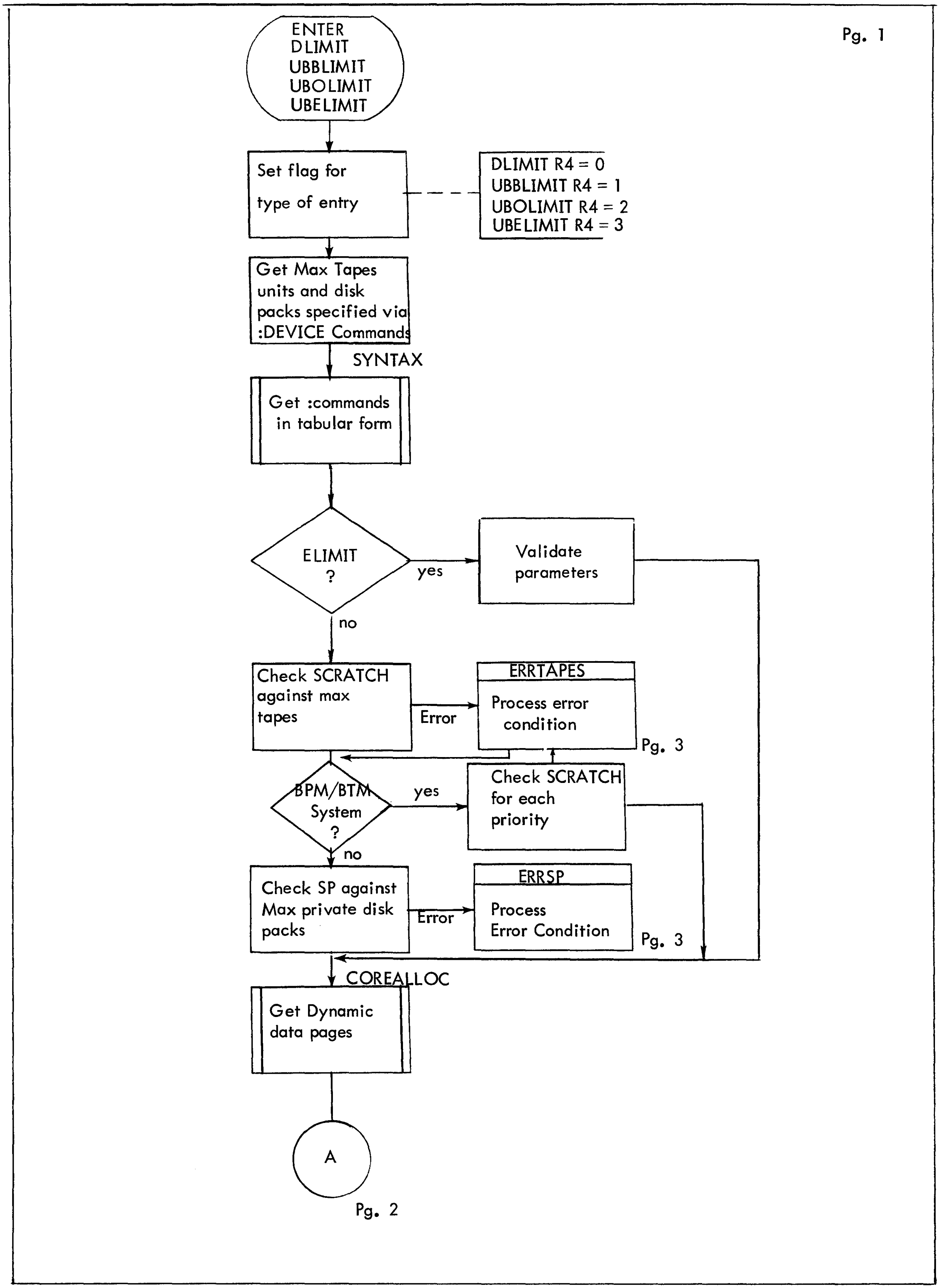


Figure 2-12. Flow Diagram of XLIMIT

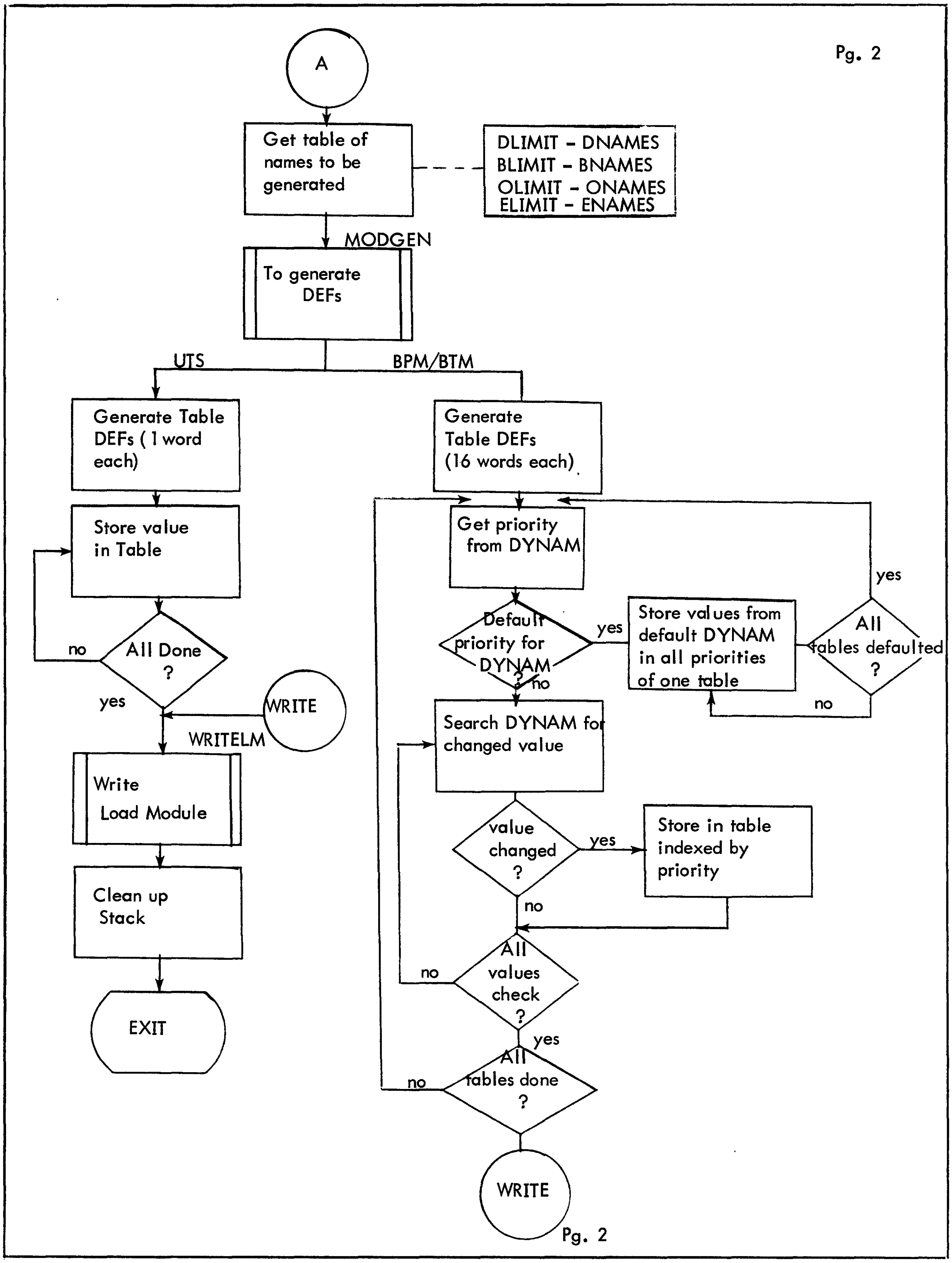


Figure 2-12. Flow Diagram of XLIMIT (Cont.)

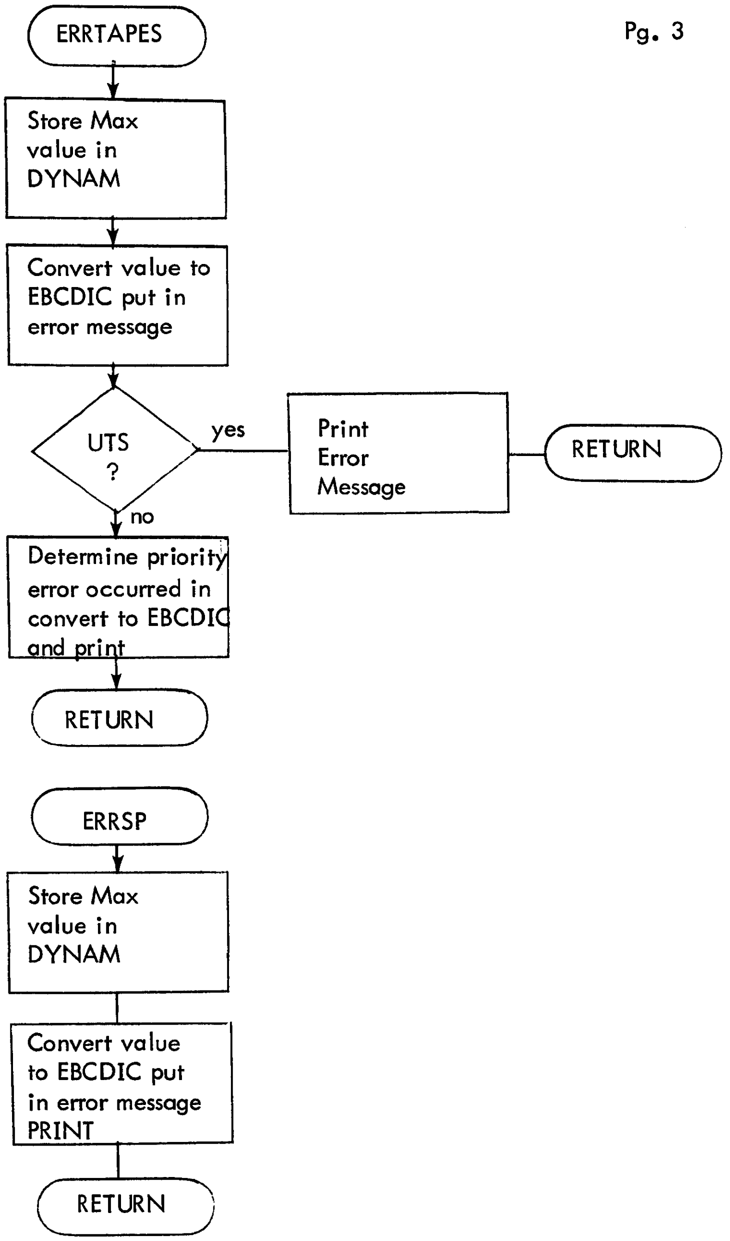

ERRTAPES
Pg. 3
Store Max value in DYNAM
Convert value to EBCDIC put in error message
UTS ?
yes
Print Error Message
RETURN
no
Determine priority error occurred in convert to EBCDIC and print
RETURN
ERRSP
Store Max value in DYNAM
Convert value to EBCDIC put in error message PRINT
RETURN

Figure 2-12. Flow Diagram of XLIMIT (Cont.)

2.6 ABS

2.6.1 Purpose

To process the PASS2 :ABS command and generate the load module M:ABS. This load module defines which processor root segments are to be located in the absolute area on the system random access device for a BPM/BTM target system only.

2.6.2 Usage

```
        B    ABS
             With    R7 pointing to the control card PLIST
                     R0 pointing to the temp stack pointer
                     R3 pointing to PASS2 stack data
                     R3 and R7 are saved
                     Return is to READSTRG in P2CCI
```

2.6.3 Input

Control card (:ABS) image

2.6.4 Output

M:ABS load module (Table 2-12)

Table 2-12. M:ABS Load Module

Label	Size	Contents or Value
ABSGOSZ	Value DEF	SIZE option on :ABS
ABSPROC	4*(number of processors+1)+ entries for names of processors in TEXTC	:L file and names of processors specified and space for control information.

2.6.5 Subroutines

CHARSCAN	(used to check a specific character for legal syntax)
DECSCAN	(used to scan a field containing a decimal value)
NAMSCAN	(used to scan a field containing a name)
QUOTSCAN	(used to scan a field containing a keyword)
MODIFY	(used to generate the M:ABS load module)
PRINTMSG OUTLLERR	(display error information)

2.6.6 Description

The ABS processor is entered when the PASS2 type is BPM (i.e., BPM/BTM target system) and :ABS is encountered by P2CCI. When entered, ABS obtains and initializes four pages of core work area (Table 2-13) to be used in generating the M:ABS load module. The syntax for the :ABS control command is:

:ABS [,size]   [(processor[, S])] [, (processor [, S])] . .

The "size" field is obtained and identifies, in decimal, the number of words desired for the absolute read/write scratch area on the system random access device for the target system. This scratch area will be used by CCI, LOADER, LOCCT, PASS3 for transmitting the loader overlay control command tables (LOCCTs) between processors in a BPM/BTM base system. This area is also usable by any other processor or user if so desired. If the "size" parameter is null or less than 1024, the value 1024 is used.

ABS then obtains the parenthetical expression. If the processor "name" is syntactically legal (1-15 alphanumeric characters 1 of which must be alpha), it is checked to see if it has already been encountered or is ":L", if so, it is ignored and ABS continues processing the command. If the name is unique, it is entered into an intermediate table (Table 2-14) and the next field is scanned. If no character exists, then ABS continues with the next parenthetical expression. If the field contains information, the contents is checked for the letter "S" and if it is not, it is assumed to be. Therefore, a flag is set in the intermediate table indicating the presence of the "S" field. ABS then continues processing the command.

When this processing is completed ABS generates the M:ABS load module. The RELDICT.00 (Table 2-13) is initialized to all Es. The contents of SECT.00 (See Table 2-14) is complete once the syntax analysis is accomplished. The intermediate table thus generated becomes SECT.00. The ABS processor continues by generating an external definition (ABSPROC) defining the base address of SECT.00. The value obtained from the "size" option is used to generate the value DEF ABSGOSZ.

When completed, the generated load module (M:ABS) is written and ABS releases its work area and temp stack area and returns to P2CCI at READSTRG.

Table 2-13 ABS Work Area

SECT.00 (Intermediate Table)	Low Core
RELDICT.00 1/8 Size of SECT.00	4 Pages Work area
RFDFSTK 1/2 of Remaining Work Area	Remaining Work Area
EXPRSTK 1/2 of Remaining Work Area	High Core

Table 2-14. ABSPROC Table

Word 0	02	:	L	---	Default Entry
1	00	00	00	00	
2	00	00	00	00	
3	00	00	00	00	
4	00	1 0 0 0	00 0	00	
5	#C	C_1	C_2	---	n-th Entry
⋮					
n	---	Cn	---	---	
n1	P0	SECT0DA			
n2	P1	SECT1DA			
n3	SECT0SZ		SECT1SZ		
n4	P2	S L T 0	STRTAD		
⋮					Subsequent Entries
m					

Table values set by ABS processor:

where

#C =number of characters (C_n) in name (TEXTC type name).

$C_1, C_2, \ldots, C_n$ = characters in name.

S = 1 Save load modules of "name" after it has been made absolute, (i.e., the parenthetical field on the ABS command specified "S").

S = 0 Release load module of "name" after it has been made absolute unless it contains an overlay structure.

L = 1 Last item in ABSPROC table.

L = 0 Not last item in ABSPROC table.

Table values not set by ABS processor but eventually set by the bootstrap procedure

P0 =page address of SECT.00 (protection type 0) when loaded into core.

P1 =page address of SECT.01 (protection type 1) when loaded into core.

P2 =page address of SECT.10 (protection type 2) which indicates the page address plus one of the end of SECT.01.

SECT0DA =disc address of the SECT.00 information.

SECT1DA =disc address of the SECT.01 information.

SECT0SZ =size of SECT.00 (words).

SECT1SZ =size of SECT.01 (words).

STRTAD =start address of the absolute processor.

T = 0 absolute processor contains a task control block (TCB).

T = 1 absolute processor does not contain a task control block (TCB).

2.6.7 ABS Messages

":L" NAME ILLEGAL OR NAME ALREADY DEFINED
A field has specified a processor name which is either ":L" or has already been specified. The ABS processor skips to next field and continues processing.

NO PAGES AVAILABLE
Not enough core available for work area. The abort message is displayed and the ABS processor returns to the Monitor.

'ABS' ABORTED
Displayed in conjunction with other catastrophic error messages. ABS returns to the Monitor.

'(' EXPECTED BUT NOT FOUND
A parenthetical field is expected but not found. Also could imply no absolute processors are desired. ABS continues by generating the load module "M:ABS".

NO FIELDS ON CC
For information purposes only. Implies no size field was specified. ABS continues by displaying " '(' EXPECTED BUT NOT FOUND" message.

INVALID PROCESSOR NAME
The processor name is not alphanumeric. ABS skips to next field and continues processing.

'S' EXPECTED BUT NOT FOUND
**'S' ASSUMED
This message appears if there is a field specification following the processor name which should be the key value "S", but instead is an unknown character string. The value "S" is assumed and ABS continues.

')' EXPECTED BUT NOT FOUND
The syntax requires a ")", and the character found is unknown. ABS continues to next parenthetical field.

SYNTAX ERROR
A terminator is encountered and is unknown or misplaced. ABS continues to next parenthetical field.

PROCESSOR NAME > 11
CHARACTERS
Self-explanatory ABS continues to next parenthetical field.

INVALID SIZE OR SIZE MISSING,
DEFAULT TAKEN
The size option is either <1024 or is not specified. The value 1024 is used and ABS continues.

LOAD MODULE GEN.
UNSUCCESSFUL
The number of processor names specified causes the intermediate table to overflow the available work area. ABS displays abort messages and returns to the Monitor.

2.6.8 Internal Routines

ABS
main entry, initializer and controls
Register 3 = address in PASS2 temp stack of information.
Register 7 = address in PASS2 temp stack of PLISTs for processing control command.

ABS0
process next parenthetical field.

CHEKNAME
check processor name against previous names. Name cannot be defined more than once and cannot be ":L".
Register 1 = length of new name.
Register 15 = address of new name.
Register 6 = address of end of absolute processor name table.

FINDEOC
search for end of control command.

FINDRPAR
search for ")" and start processing with next "(".

ABSOUT
Control command processing finished, now generate "M:ABS" load module.
Register 6 = address of next available entry in work area.
Register 9 = base address of work area.

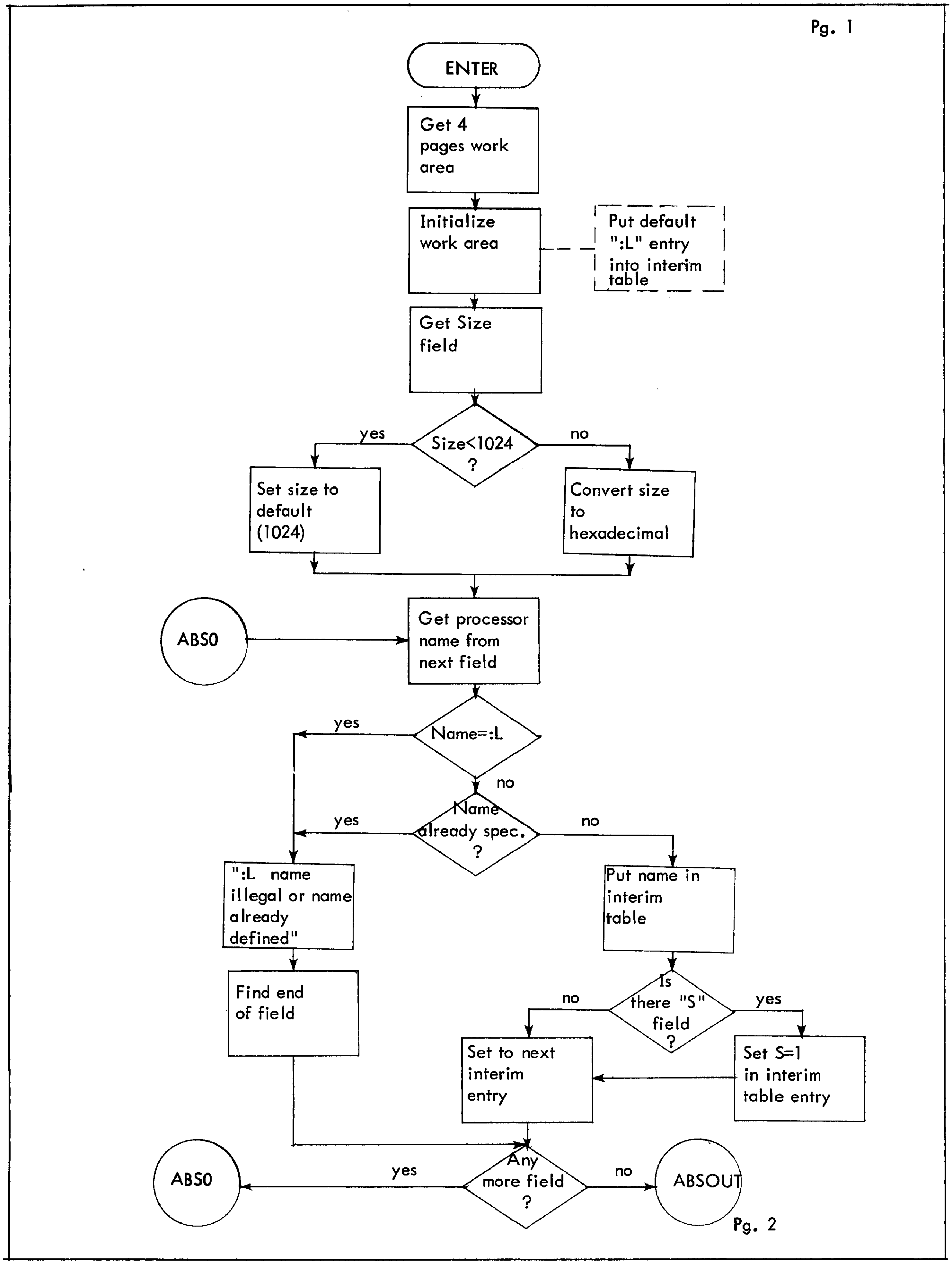
WRITE
Write load module to "M:ABS" file.
Register 12 = buffer address.
Register 13 = buffer size (bytes).
Register 14 = key address (load module elements key).

DECCNV convert decimal size to hexadecimal equivalent.

Input: Register 2 = number of characters.
Register 1 = address of character string.

Output: Register 3 = converted value.
CC1 = 0 = converted value.
CC1 = 1 = conversion cannot be completed.

2.6.9 Flow Chart



Pg. 1
ENTER
Get 4 pages work area
Initialize work area
Put default ":L" entry into interim table
Get Size field
Size<1024 ?
yes
no
Set size to default (1024)
Convert size to hexadecimal
ABSO
Get processor name from next field
Name=:L
yes
no
Name already spec. ?
yes
no
":L name illegal or name already defined"
Put name in interim table
Find end of field
Is there "S" field ?
no
yes
Set to next interim entry
Set S=1 in interim table entry
ABSO
yes
Any more field ?
no
ABSOUT
Pg. 2

Figure 2-13. Flow Diagram of ABS

Pg. 2

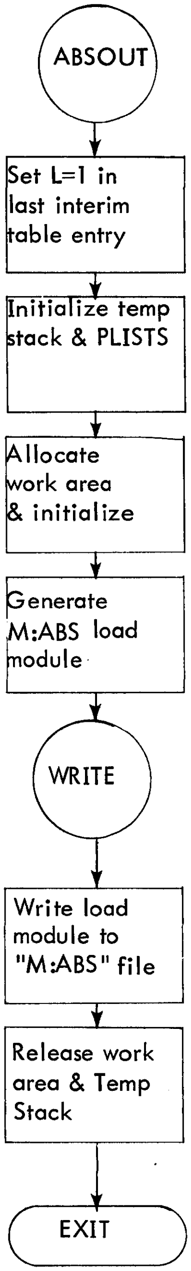

ABSOUT
Set L=1 in last interim table entry
Initialize temp stack & PLISTS
Allocate work area & initialize
Generate M:ABS load module
WRITE
Write load module to "M:ABS" file
Release work area & Temp Stack
EXIT

Figure 2-13. Flow Diagram of ABS (Cont.)

2.7 BTM

2.7.1 Purpose

To process BTM PASS2 control command, creating M:BTM load module for BTM systems, and to include the name BTMSTAT in SPEC:HAND if the BTM performance monitor routine is to be included in the target system.

2.7.2 Usage

```
B          BTM with  R0 = temp stack pointer
                     R3 = PASS2 stack data pointer
                     R7 = control card PLIST
```

Return is to READSTRG with R0 and R3 intact.

2.7.3 Input

:BTM control command image

2.7.4 Output

M:BTM load module (Table 2-15).

Table 2-15. M:BTM Load Module Contents

Label	Entry Size (Wds)	Length	Contents or value (in terms of BTM Keyword values)
NUMUSERS	Value	-	NUMUSERS
USERSIZE	Value	-	USERSIZE
NUMSYSTS	Value	-	NUMSYSTS
BPMQTM	Value	-	BPMQTM/2 (i.e., converted to clock ticks)
BTMQTM	Value	-	BTMQTM/2 (i.e., converted to clock ticks)
BTMQTM2	Value	-	BTMQTM2/2 (i.e., converted to clock ticks)
IBUFSIZE	Value	-	IBUFSIZE
OBUFSIZE	Value	-	OBUFSIZE
COCIINT1	Value	-	IINT
COCIGRP1	Value	-	Interrupt group of IINT
COCIIBT1	Value	-	Write direct bit setting for IINT
COCOINT1	Value	-	OINT
COCOGRP1	Value	-	Interrupt group of OINT

Table 2-15. M:BTM Load Module Contents (Cont.)

Label	Entry Size (Wds)	Length	Contents or Value (in terms of BTM Keyword values)
COCOBIT1	Value	-	Write direct bit setting for OINT
CLK3INT	Value	-	BTMINIT
BTMGL	Value	-	Interrupt group for BTMINIT
BTMIBIT	Value	-	Write direct bit setting for BTMINIT
BTMPM	Value	-	=0 if BTMPM Keyword not present =1 if BTMPM Keyword present
USERPGS[1]	Value	-	# of pages computed from USERSIZE +1
PMNQINTS[1]	Value	-	26
PMMISC[1]	Value	-	58
ACTFLAG	1/4	NUMUSERS	0
ACTPASSD	1/4	NUMUSERS	0
ACTTYPE	1/4	NUMUSERS	0
TTYFLAG	1/4	NUMUSERS	0
INCOUNT	1/4	NUMUSERS	0
OUTCOUNT	1/4	NUMUSERS	0
LITNEXT	1/4	NUMUSERS	0
RDROFF	1/4	NUMUSERS	0
ITABPSN	1/4	NUMUSERS	0
OTABPSN	1/4	NUMUSERS	0
INBUF	1/4	NUMUSERS* IBUFSIZE	0
OUTBUF	1/4	NUMUSERS* OBUFSIZE	0
COCBUF1	1/2	NUMUSERS*2 + 2	0
INPNTI	1	NUMUSERS	0
INPNTR	1	NUMUSERS	0
INBFEND	1	NUMUSERS+1	0
OUTPNTI	1	NUMUSERS	0
OUTPNTR	1	NUMUSERS	0
OUTBFEND	1	NUMUSERS+1	0
TABSTOPS	2	NUMUSERS	X'FF000000', X'00000000'
ACTCOND	1/4	NUMUSERS	0
HASITBYT	1/4	NUMUSERS	0
00SIZ	1/4	NUMUSERS*2	X'01'
01SIZ	1/4	NUMUSERS*2	0
DYPG	1/4	NUMUSERS*2	0
UNUSD	1/4	NUMUSERS*2	0

[1] If BTMPM=1

Table 2-15. M:BTM Load Module Contents (Cont.)

Label	Entry Size (Wds)	Length	Contents or Value (in terms of BTM Keyword values).
COMPG	1/4	NUMUSERS*2	0
PROGLEVL	1/4	NUMUSERS	0
BREAKC	1/4	NUMUSERS	0
USRTABLE	1	NUMUSERS	0
LOGNAME	2	NUMUSERS	0
PSDLEV0P	2	NUMUSERS	0
PSDLEV0C	2	NUMUSERS	0
PSDLEV1P	2	NUMUSERS	0
PSDLEV1C	2	NUMUSERS	0
PSDLEV2P	2	NUMUSERS	0
PSDLEV2C	2	NUMUSERS	0
RESTCMND	2	NUMSYSTEMS+8	See Insert Following
CMNDOBEY	1	NUMSYSTEMS+8	" " "
COMMANDS	1/2	NUMSYSTEMS+8	" " "
SYSBEG	1/2	NUMSYSTEMS	0
SYSCOUNT	1/4	NUMSYSTEMS	0
SYSTABLE	1	NUMSYSTEMS	0
SWPLST	2	120 if SWAPPER = 720X 15 if SWAPPER= 7232/7212/7242 (Swapping device passed from UBCHAN)	0
QFREQ[1]	1	PMNQINTS (26)	0
RSPFR[1]	1	PMNQINTS (26)	0
DATPGS4[1]	1	USERPGS	0
QUANTBGN[1]	1	NUMUSERS	0
PPPGS[1]	1	USERPGS	0
LASTIM[1]	1	NUMUSERS	0
QTMSAV[1]	1	NUMUSERS	0
SUBSQTM[1]	1	NUMSYSTEMS	0
SUBSTSK[1]	1	NUMSYSTEMS	0
SUBSESQ[1]	2	NUMSYSTEMS	0
PMREQS[1]	2	PMMISC (58)	0

[1] If BTMPM=1

Insert :

RESTCMND - contents

First N entries (Doubleword) where N=NUMSYSTS are 0.

Last eight entries (Doubleword) in TEXTC format are

SIGN
E
SSAGE
OCEED
STORE
VE
BS
EKATMEM

COMMANDS - contents

First N entries (halfword) where N=NUMSYSTS are 0.

Last eight entries (halfword) in TEXTC format are

AS
BY
ME
PR
RE
SA
TA
PE

CMNDOBEY - contents

First N entries (word) where N=NUMSYSTS are

X'68000000' + STSUBSYS (REFed)

Last eight entries (word) are

X'68000000'+STASSIGN (REFed)
X'68000000'+STEXIT (REFed)
X'68000000'+STMESS (REFed)
X'68000000'+STPROCED (REFed)
X'68000000'+RESTEXC (REFed)
X'68000000'+SAVEXC (REFed)
X'68000000'+TABEXC (REFed)
X'68000000'+STLOOK (REFed)

SPEC:HAND Data File

If BTMPM=1

Then BTMSTAT name is added to SPEC:HAND data file so that PASS3 will include the module in the HANDLERS file in M:MON.

2.7.5 Interaction

SYNTAX to convert command to stack data block

COREALLOC to allocate dynamic memory

MODGEN to generate DEFs, REFs

WRITELM to write M:BTM load module

2.7.6 Data Bases

KWD is the Keyword table for SYNTAX

DYNAM is the virgin stack data block for SYNTAX, containing the defaults for each Keyword.

CMNTBL1 is the table used in generating the COMMANDS entry in M:BTM.

CMNTBL2 is the table used in generating the RESTCMND entry in M:BTM.

RANGE is the table used to verify that the values specified with the Keywords are within a valid range. This is required as the defaults are greater than the minimum permissible values and SYNTAX cannot do both limit checking and default setting under these conditions.

VALERR - a table of error message addresses and default values used in conjunction with RANGE. The index value of the parameter in RANGE isused to obtain the appropriate default and error message.

2.7.7 Description

After determination of the type of swapping device to be used on the target system, a BAL to SYNTAX puts command information and defaults (except for USERSIZE) from DYNAM into the stack, pointed to by R5 and returns. The parameter USERSIZE is defaulted if the Keyword has not been specified. All parameters are then checked against the permissible values in RANGE. If any is out of range an error routine is entered that stores the appropriate default in the stack and prints a message indicating the parameter in error and the value used and then returns.

Upon completion of this checking, a BAL to COREALLOC causes memory to be allocated. A BAL to MODGEN begins the generation of the value DEFs, followed by the location DEFs and where applicable the REFs. The value DEFs and location DEFs for the BTM Performance Monitor are by-passed if the module is not to be included in the target system.

Upon completion of this generation, a BAL to WRITELM causes M:BTM load module to be written. Then, if the BTM Performance Monitor is required, the SPEC:HAND data file is read into that area of core previously occupied by the Data Base which is no longer needed. The name BTMSTAT is added to the file, and the number of entries in SPEC:HAND incremented by 1. Then the file is written and closed. Any difficulty encountered causes BTMSTAT not to be included and an error message is printed indicating this has occurred.

The stack generated by SYNTAX is then cleaned up and BTM exits to READSTRG in P2CCI.

2.7.8 BTM Messages

***TROUBLE WITH SPEC:HAND-
BTMSTAT NOT INCLUDED

In attempting to open the file SPEC:HAND to add the name of the file BTMSTAT an error or abnormal condition was encountered. BTM continues.

***NUMUSERS ERROR - DEFAULT (8) USED
***USERSIZE ERROR - DEFAULT (16384) USED
***NUMSYSTS ERROR - DEFAULT (12) USED
***BPMQTM ERROR - DEFAULT (200) USED
***BTMQTM ERROR - DEFAULT (800) USED
***BTMQTM2 ERROR - DEFAULT (800) USED
***IBUFSIZE ERROR - DEFAULT (100) USED
***OBUFSIZE ERROR - DEFAULT (100) USED
***IINT ERROR - DEFAULT (60) USED
***OINT ERROR - DEFAULT (61) USED
***BTMINIT ERROR - DEFAULT (5A) USED

The values used for the specified parameter is in error. The given default is used. BTM continues.

2.7.9 Flow Chart

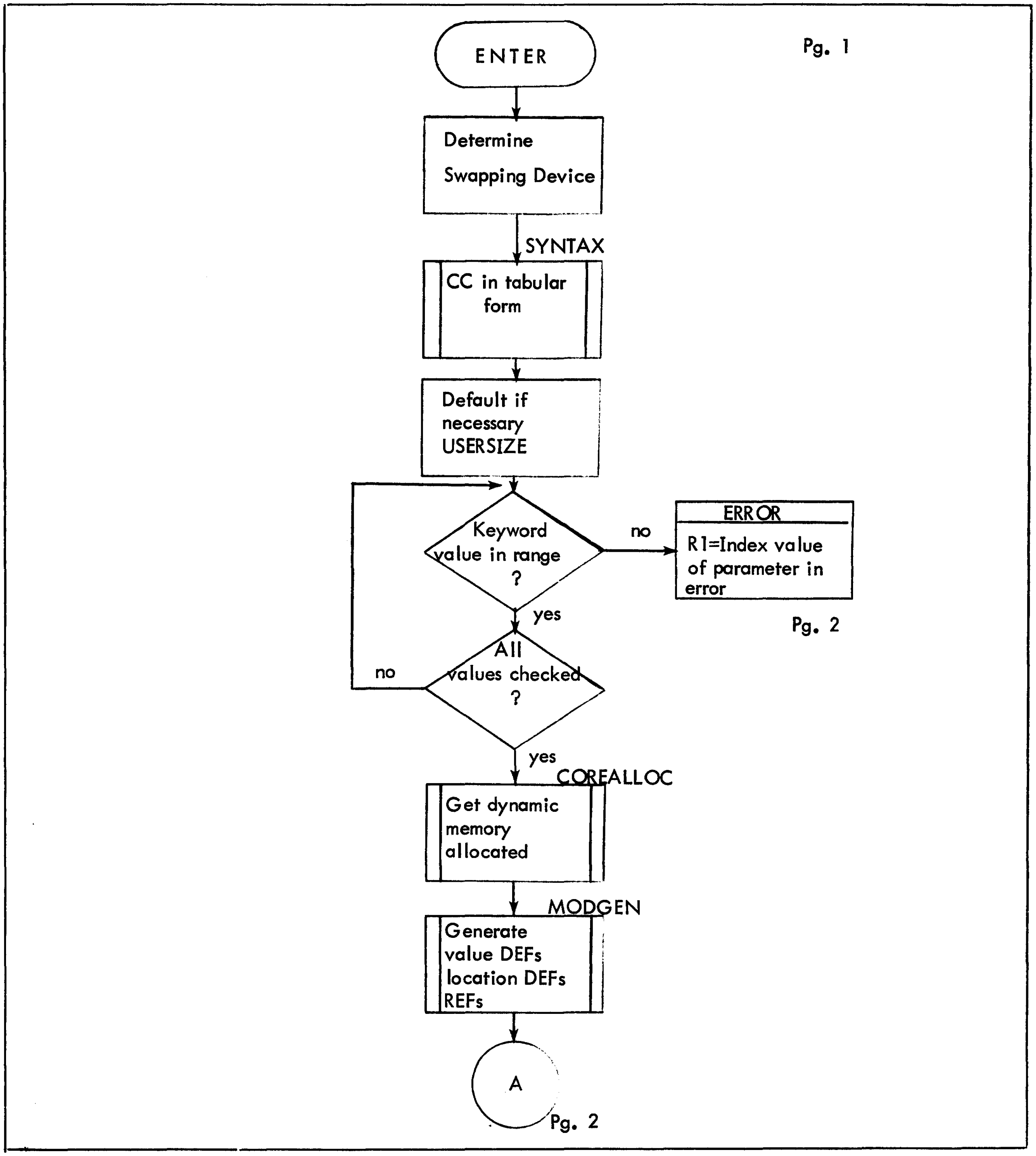


Figure 2-14. Flow Diagram of BTM

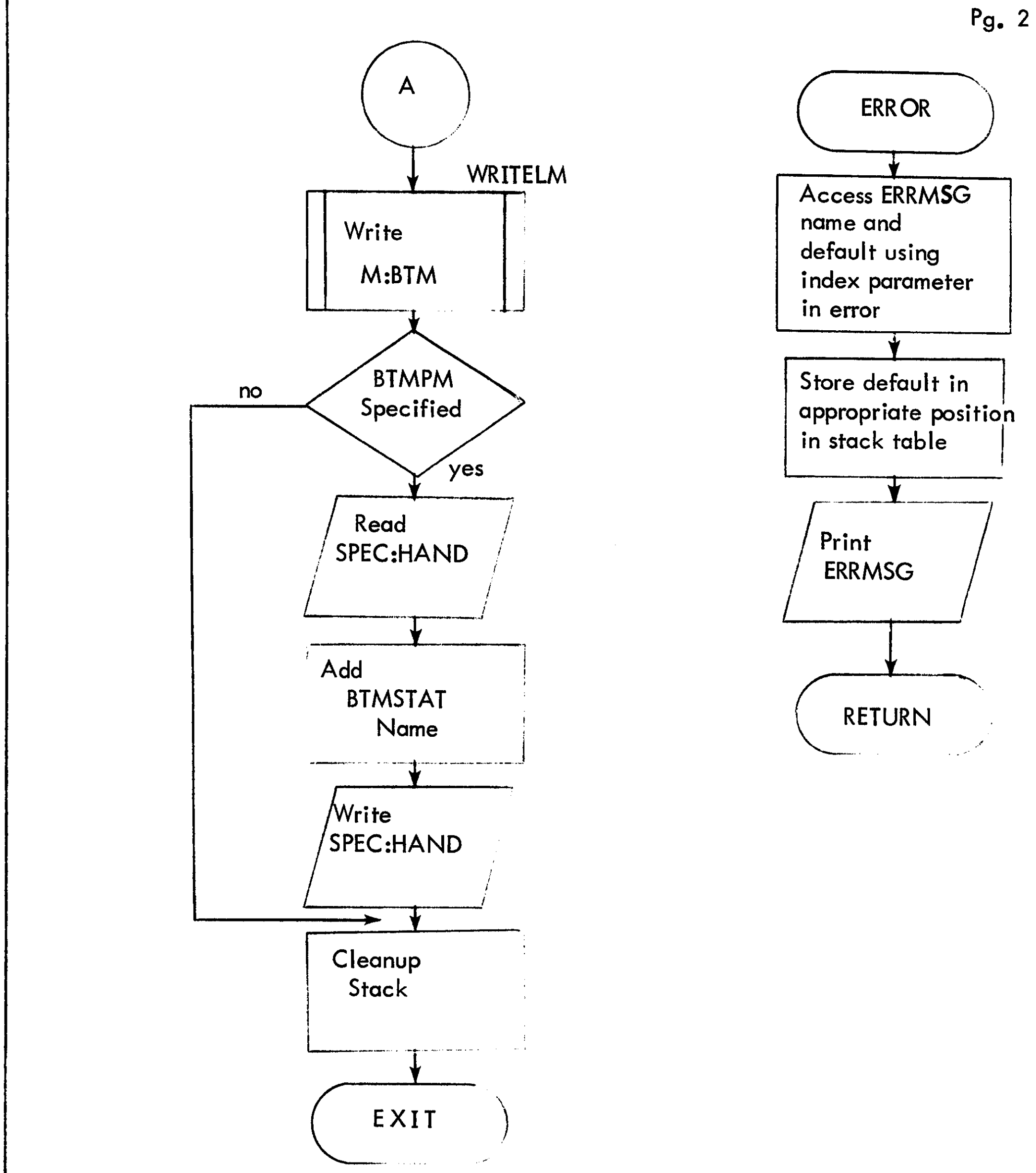


Figure 2-14. Flow Diagram of BTM (Cont.)

2.8 P2COC

2.8.1 Purpose

To process the :COC PASS2 control command generating the M:COC load module and updating the SPEC:HAND file to include requested translate tables.

2.8.2 Usage

```
    B      COC
           With:  R7 pointing to control card PLIST
                  R0 pointing to temp stack pointer
                  R3 pointing to PASS2 stack data
                  R0 and R3 saved
                  Return is to READSTRG in P2CCI.
```

2.8.3 Input

Control card (:COC)

COCS - the relative address in PASS2 Stack where a halfword table of COC device addresses begins if any have been defined via :DEVICE commands.

2.8.4 Output

SPEC:HAND

The names of any standard requested translate tables are added.

M:COC (Table 2-16).

Table 2-16. Contents of M:COC

Label	Entry Size (Wds)	Length	Contents/Value (In terms of Keywords)
LCOC	Value	-	Number of COC-1
COD:LPC	2	COC	1st entry word 0=0, word 1=LINES-1 for each subsequent entry. Word 0=1 more than previous entry's word 1. Word 1=LINES-1 (for the corresponding COC) added to previous entry's word 1
CO:IN0	10	(COC)-6	First entry-4 words only 0,0 COCIP (PREF), X'10000000'. All other entries - 10 words 0,0, $+2, X'11000000', STW,5 $+5 LI,5 COCO Interrupt mask bit WD,5 X'1700'+COCO group level LW,5 $+2 LPSD,11 $+8 DATA 0 Note: Each entry is pointed to by CO:XPSDI
Out Interrupt (no name)	6	COC	Each 6 word entry is 0,0 $+2, X'170000N0' where N is the entry number LI,3 N-1 B COCOP (PREF) 1st entry only all other entries here branch to this location Note: Each entry is pointed to by CO:OUT and CO:XPSDO
I/O Command (no name)	4	COC	Each entry has I/O command Read; DATA CHAIN into RING buffer, 4* RING bytes; TIC DA($-2),0 Note: Each entry pointed to by CO:CMND
COH:DN	1/2	COC	DEVICE address
COH:II	1/2	COC	INput interrupt address
COH:ILI	1/2	COC	INput interrupt level bit
CO:WDAEI	1	COC	Each entry = WD,5 X'1200+IN level group
COH:IO	1/2	COC	OUTput interrupt address
COH:ILO	1/2	COC	OUTput interrupt level bit
CO:WDAEO	1	COC	Each entry=WD,5 X'1200'+OUT level group
CO:STAT	1	COC	WD,0 X'3ON0' where n = index in table
CO:OUTRS	1	COC	RD,7 X'3ON0'
CO:RCVON	1	COC	WD,7 X'3ON1'
CO:RCVDOFF	1	COC	WD,7 X'3ON3'
CO:TRNDOFF	1	COC	WD,7 X'3ON7'
CO:XDATA	1	COC	WD,6 X'3ON5'
CO:XSTOP	1	COC	WD,7 X'3ONE'

Table 2-16. Contents of M:COC (Cont.)

Label	Entry Size (Wds)	Length	Contents/Value (In terms of Keywords)
CO:RINGE	1	COC	Each entry contains word address of end of RING buffer
CO:LST	1	COC	Each entry= 4*RING$_n$ buffer size
CO:OUT	1	COC	Each entry= word address of OUT interrupt routine
CO:CMND	1	COC	Each entry= Doubleword address of I/O command
CO:XPSDI	1	COC	Each entry= XPSD,8 CO:INO entry
CO:XPSDO	1	COC	Each entry= XPSD,8 Out interrupt routine
COB:RBS	1/4	COC	Each entry=4*RING$_n$ buffer size
LNOL	Value	-	Total LINES for all COCs
COCOC	1/4	LNOL	0
LB:UN	1/4	LNOL	0
RSZ	1/4	LNOL	0
MODE2	1/4	LNOL	For 2741 model type entry = X'30' For other model types entry=X'20'
MODE	1/4	LNOL	For 2741 model type entry = X'08' For other model types entry=X'88'
COCTERM	1/4	LNOL	For 2741 model type entry = 0 For other model types = 3
MODE3	1/4	LNOL	0
ARSZ	1/4	LNOL	0
CPOS	1/4	LNOL	1
CPI	1/4	LNOL	0
BUFCNT	1/4	LNOL	0
TL	1/2	LNOL	X'8000'
COCOI	1/2	LNOL	0
COCOR	1/2	LNOL	0
COCII	1/2	LNOL	0
COCIR	1/2	LNOL	0
EOMTIME	1/2	LNOL	0
COD:HWL	2	COC	For each doubleword entry – given the total # of lines defined in a COC, right-justified, those lines that are HARDWIREd have the corresponding bit set. All other bits are 0. Ex total lines = 8, HARDWIRE = 0,1,2 Entry = 00000000, 000000E0
Ring buffers (no name)			Each buffer has all bits set and is RING words long.

Table 2-16. Contents of M:COC (Cont.)

Label	Entry Size (Wds)	Length	Contents/Value (In terms of Keywords)
COCBUF	4		Each buffer's first word has the word displacement of the next from COCBUF except the last, which has 0.
COCNB	Value	-	number of 4 word buffers
HRBA	Value	-	displacement of the last buffer from COCBUF
COCHPB	1	1	4 (head of buffer pool)

2.8.5 Interaction

M:OPEN, M:READ, M:WRITE, M:CLOSE are used to update SPEC:HAND

SYNTAX is used to decode the control command.

COREALLOC is used to allocate dynamic memory

MODGEN is used to generate M:COC

WRITELM is used to write M:COC

2.8.6 Data Bases

KWDTBL is the Keyword table for SYNTAX

DYNAM is the virgin stack data block

2.8.7 Subroutines

ERRLIST outputs an error message after inserting the COC number into it. (BAL,R11 with COC number in R15, message address in R14).

WDLG returns in R12 an interrupt group number, in R13 the interrupt level bit for the interrupt at the address in R12 on entry. (BAL,R14)

COCGEN is similar to MODGEN in that if interpretively executes code, but its link register is R11. All MODGEN-type code encountered is skipped except relocation dictionary changes, for which MODGEN is used to effect the changes.

2.8.8 Description

SYNTAX decodes the control card, producing a number of stack data blocks equal to the requested number of COCs. P2COC first checks that IN and OUT have been specified for every COC and that every IN is less than the corresponding OUT, and that no IN or OUT is the same as any other IN or OUT. If not, P2COC returns. All the LINES options are added up, all BUFFERS options are added, and all RING, BUFFERS, and LINES options are validated.

All DEVICE options are checked for presence and correspondence with some :DEVICE MENDD (from COCS).

Then if translate tables were requested, the SPEC:HAND file is updated appropriately. The total size of the data record is calculated and COREALLOC is called to set it up.

Then MODGEN is entered and the value DEF for LCOC and all tables of COC length are generated. The same code (from LGEN to LINETBLS) is then executed under the control of the COCGEN routine. This causes changes to be made to the relocation dictionary but bypasses the generation of the DEFs. The different link register also allows selective branches to take place. When the COC tables are completed, MODGEN is reentered to generate the line tables. WRITELM then creates the M:COC file and P2COC exits to READSTRG.

2.8.9 P2COC Messages

***COCX -- INTERRUPT LEVEL CONFLICT - COC ABORTED

The IN, or OUT parameter for COCX were in conflict with either previously defined levels or IN was greater than OUT or IN or OUT was undefined. P2COC restores the temp stack and exits.

***COCX -- LINES > 64 - DEFAULT TAKEN

Greater than 64 LINES were specified per COC device. P2COC defaults the value and continues.

***COCX -- WARNING: BUFFERS < 3XLINES

The value specified for BUFFER was less than 3 lines the number of LINES specified. P2COC issues this warning and continues.

***COCX -- RING INADEQUATE - DEFAULT TAKEN

The value specified for RING was too small for the number of LINES specified. P2COC defaults the value (2 bytes / line for the first 30 lines and 1 byte/line above 30, divided by 4) and continues.

***COCX -- DEVICE OPTION MISSING - COC ABORTED

For the given COC, no DEVICE was specified. P2COC restores the temp stack and exits.

***COCX -- DEVICE NOT DEFINED - COC ABORTED

For the given COC, the DEVICE option specifies a device (NDD) that was not defined via :DEVICE commands. P2COC restores the temp stack and exits.

2.8.10 Flow Chart

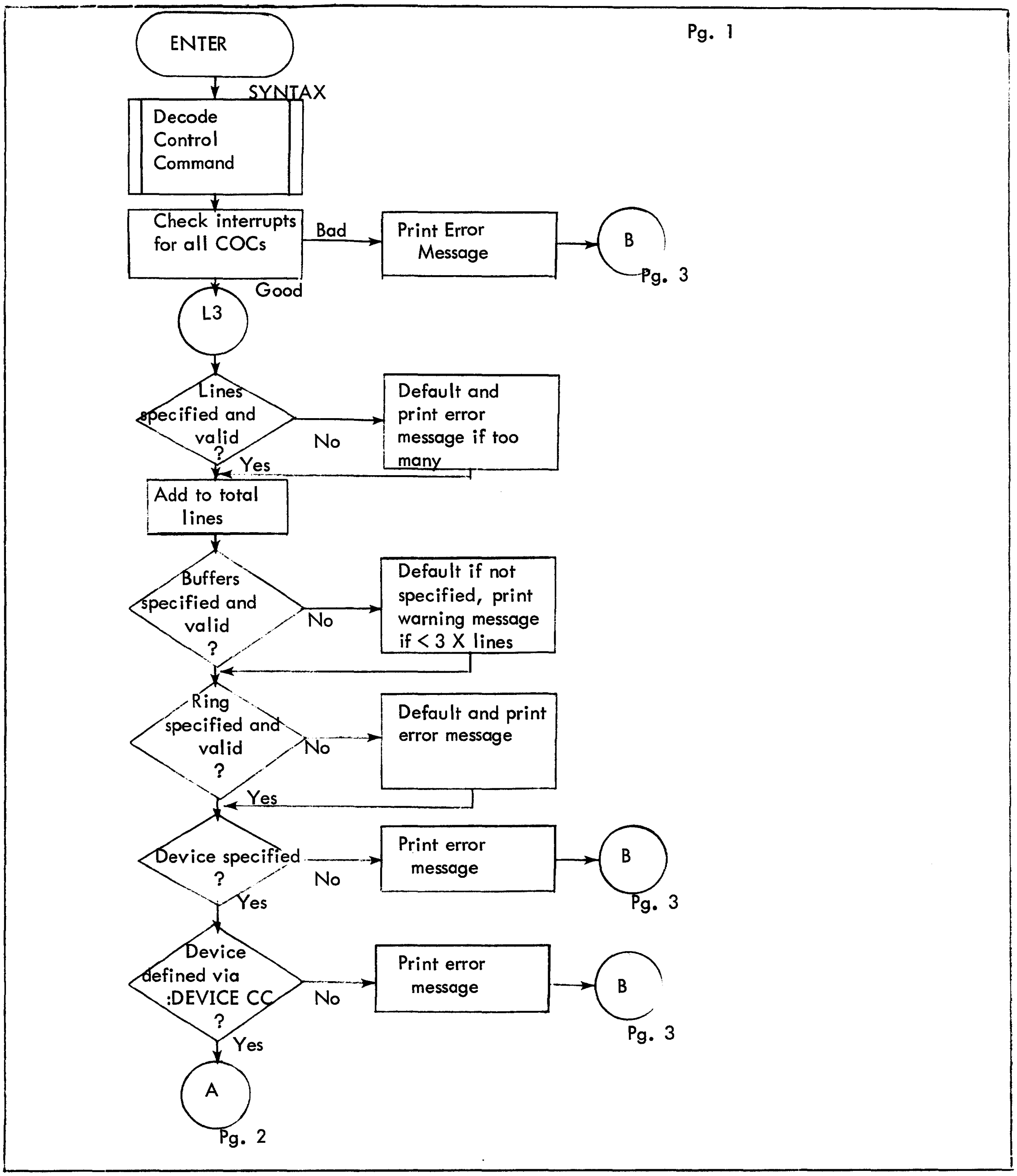

Pg. 1
ENTER
SYNTAX
Decode Control Command
Check interrupts for all COCs
Bad
Print Error Message
B
Pg. 3
Good
L3
Lines specified and valid ?
No
Default and print error message if too many
Yes
Add to total lines
Buffers specified and valid ?
No
Default if not specified, print warning message if < 3 X lines
Ring specified and valid ?
No
Default and print error message
Yes
Device specified ?
No
Print error message
B
Pg. 3
Yes
Device defined via :DEVICE CC ?
No
Print error message
B
Pg. 3
Yes
A
Pg. 2

Figure 2-15. Flow Diagram of P2COC

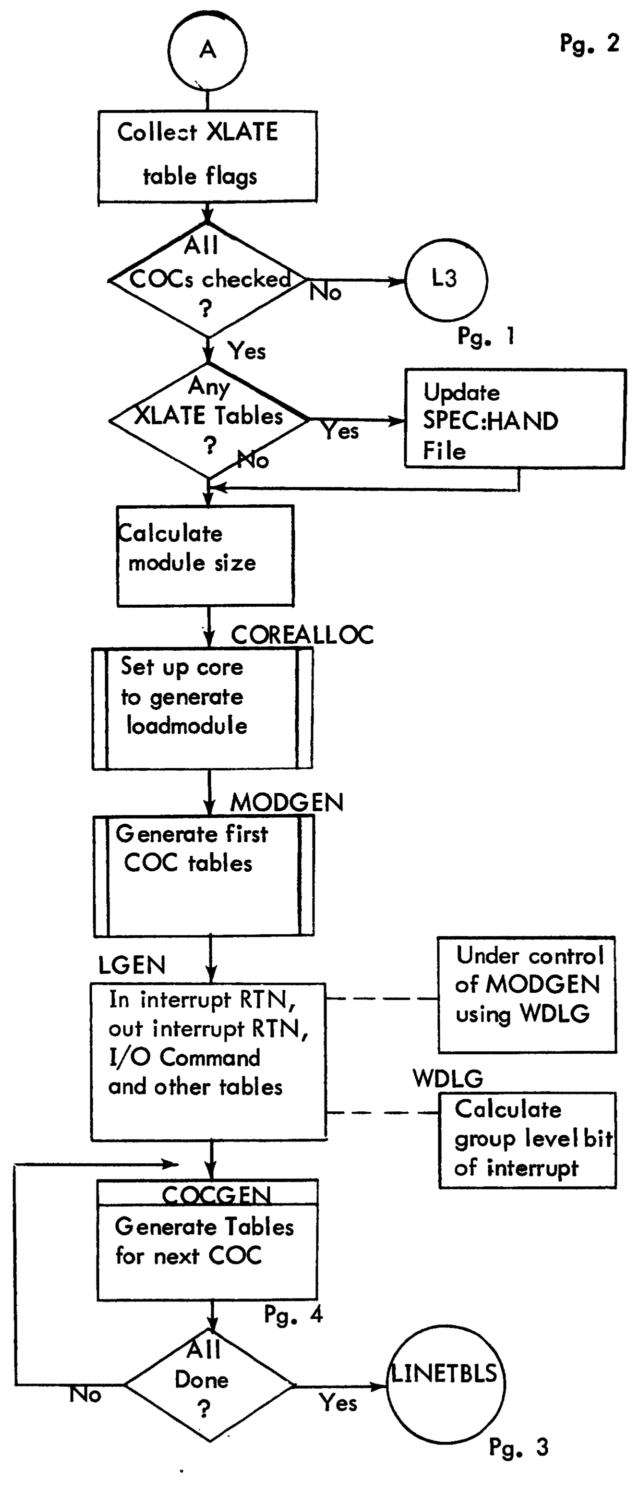

A
Pg. 2
Collect XLATE table flags
All COCs checked ?
No
L3
Pg. 1
Yes
Any XLATE Tables ?
Yes
Update SPEC:HAND File
No
Calculate module size
COREALLOC
Set up core to generate loadmodule
MODGEN
Generate first COC tables
LGEN
In interrupt RTN, out interrupt RTN, I/O Command and other tables
Under control of MODGEN using WDLG
WDLG
Calculate group level bit of interrupt
COCGEN
Generate Tables for next COC
Pg. 4
All Done ?
No
Yes
LINETBLS
Pg. 3

Figure 2-15. Flow Diagram of P2COC (Cont.)

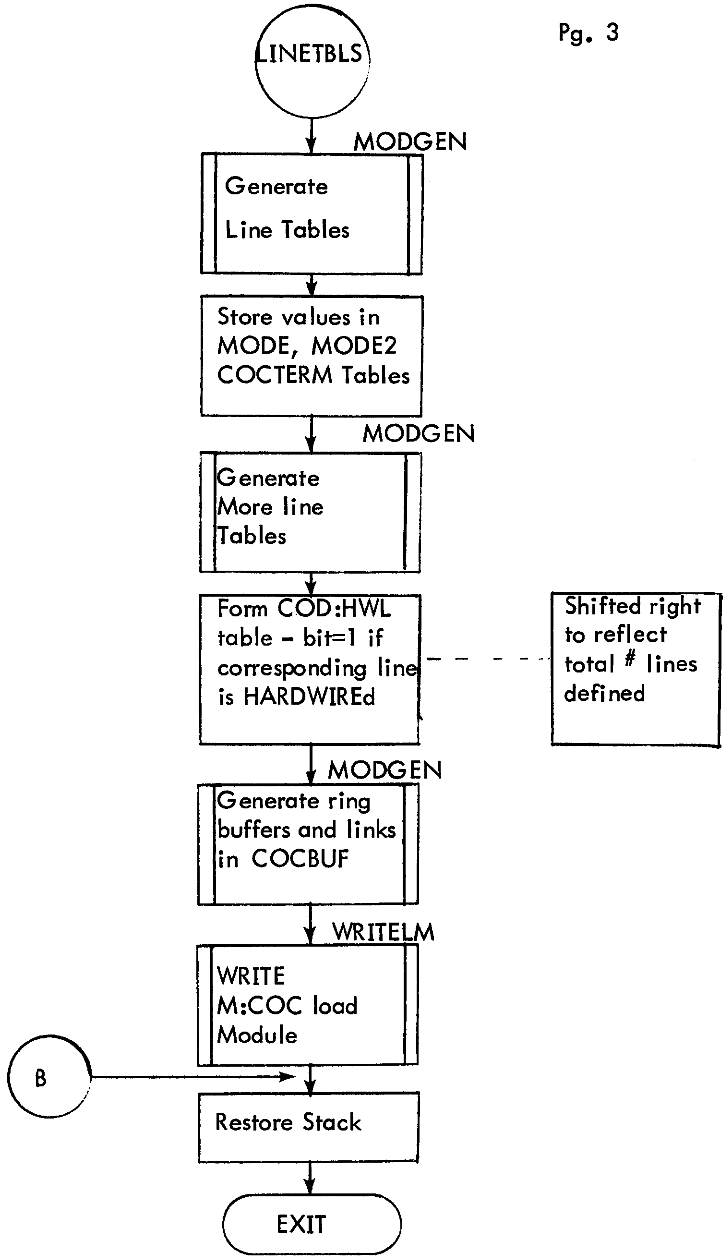

LINETBLS
Pg. 3
MODGEN
Generate Line Tables
Store values in MODE, MODE2 COCTERM Tables
MODGEN
Generate More line Tables
Form COD:HWL table - bit=1 if corresponding line is HARDWIREd
Shifted right to reflect total # lines defined
MODGEN
Generate ring buffers and links in COCBUF
WRITELM
WRITE M:COC load Module
B
Restore Stack
EXIT

Figure 2-15. Flow Diagram of P2COC (Cont.)

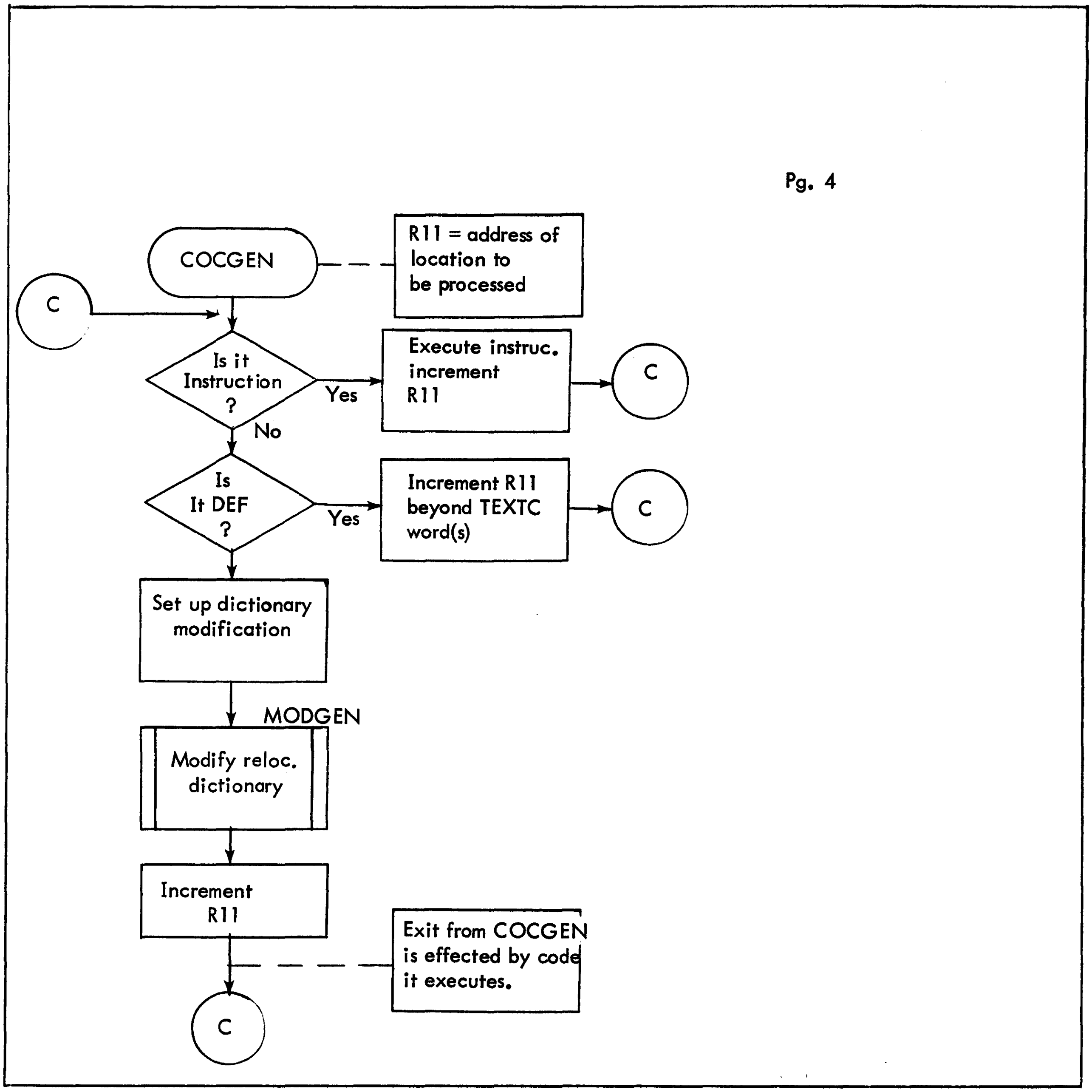

Pg. 4
COCGEN
R11 = address of location to be processed
C
Is it Instruction ?
Yes
Execute instruc. increment R11
C
No
Is It DEF ?
Yes
Increment R11 beyond TEXTC word(s)
C
Set up dictionary modification
MODGEN
Modify reloc. dictionary
Increment R11
Exit from COCGEN is effected by code it executes.
C

Figure 2-15. Flow Diagram of P2COC (Cont.)

2.9 IMC

2.9.1 Purpose

To generate the M:IMC SYSGEN load module for UTS systems only.

2.9.2 Usage

B IMC with R0 = temp stack pointer

R3 = PASS2 stack data pointer

R7 = Control card PLIST pointer

Return is to READSTRG in P2CCI with R0 and R3 intact.

2.9.3 Input

:IMC control command image.

BIG9FLG flag set by XMONITOR if target system > 128K (i.e., BIG9 option specified, see 2.4.8).

2.9.4 Output

M:IMC load module (Table 2-17)

Table 2-17. M:IMC Load Module Contents

Label	Entry Size (wds)	Length	Contents/Value (in terms of IMC Keyword values)
SMUIS	Value	-	MAXG+MAXB+MAXOL-1
MING	Value	-	4
MAXG	Value	-	MAXG
SMBUIS	Value	-	MAXB
SL:THRS	Value	-	THRESHOLD
SL:BKUP	Value	-	1 if BACKUPALL specified 0 if not
SL:EX	Value	-	EXPIRE (converted to hours)
SL:MEX	Value	-	MAXEXPIRE (converted to hours)
S:GUAIS	1	1	MAXG
S:BUAIS	1	1	MAXB
SL:TB	1	1	BLOCK
SL:UB	1	1	UNBLOCK
SL:QUAN	1	1	QUANTA/2
SL:QMIN	1	1	MINQUAN/2
SL:SQUAN	1	1	MINTIME/2
SL:BB	1	1	PERCENT
SL:IOC	1	1	IOCORE
SL:IOPC	1	1	IOPASSCOUNT

Table 2-17. M:IMC Load Module Contents (Cont.)

Label	Entry Size (wds)	Length	Contents/Value (in terms of IMC Keyword values)
SL:OLTO	1	1	LOGTIME
SL:OITO	1	1	INTIME
S:QUAIS	1	1	MAXOL
SL:PI	1	1	PI
SL:9T	1	7	Word 0 = T9TAPE
			1 = 0
			2 = B9TAPE
			3 = 0
			4 = 09TAPE
			5 = 0
			6 = 0
SL:7T	1	7	Word 0 = T7TAPE
			1 = 0
			2 = B7TAPE
			3 = 0
			4 = O7TAPE
			5 = 0
			6 = 0
SL:SP	1	7	Word 0 = TDISC
			1 = 0
			2 = BDIC
			3 = 0
			4 = ODISC
			5 = 0
			6 = 0
SL:C	1	7	Word 0 = X'7FFFFFFF'
			1 = 0
			2 = TBCORE
			3 = 0
			4 = TOCORE
			5 = 0
			6 = X'C'
SL:ONCB	1	1	COCBUF
SL:CORE	1	1	0
SL:OXMF	1	1	6

Table 2-17. M:IMC Load Module Contents (Cont.)

Label	Entry Size (wds)	Length	Contents/Value (in terms of IMC Keyword values)
SL:BXMF	1	1	6
SL:OIMF	1	1	3
SL:BIMF	1	1	3
U:MISC	1	SMUIS+1	0
UH:FLG	1/2	SMUIS+1	0
UH:JIT	1/2	SMUIS+1	0
UH:AJIT	1/2	SMUIS+1	0
UH:FLG2	1/2	SMUIS+1	0
UH:TS	1/2	SMUIS+1	0
U:JIT[1]	1/4 or 1/2	SMUIS+1	0
UB:PCT	1/4	SMUIS+1	0
UB:SWAPI	1/4	SMUIS+1	0
UB:MF	1/4	SMUIS+1	0
UB:US	1/4	SMUIS+1	0
UB:FL	1/4	SMUIS+1	Beginning with UB:FL+MING entry each byte has 1+ its byte index from UB:FL except the last, which is 0
UB:BL	1/4	SMUIS+1	Beginning with UB:BL+MIG+1 entry, each byte has -1 +its byte index from UB:BL
UB:APR	1/4	SMUIS+1	0
UB:APO	1/4	SMUIS+1	0
UB:ASP	1/4	SMUIS+1	0
UB:ACP	1/4	SMUIS+1	0
UB:DB	1/4	SMUIS+1	0
UB:OV	1/4	SMUIS+1	0
S:UCYL[2]	1	1	UCYL
UB:C#[2]	1/4	SMUIS+1	Entry 0 = 0 Entry 1 = highest PSA cylin. # Entry 2 = 0 Entry 3 = highest PSA cylin. # - UCYL Entry 4 to MING = UCYL less than preceding entry

[1] Entry size is 1/2 word when target system > 128K on Sigma 9. Otherwise, entry size is 1/4 word. Determining factor is BIG9 option on UTM command (see 2.4).

[2] Generated only for no-RAD target systems.

Table 2-17. M:IMC Load Module Contents (Cont.)

Label	Entry Size (wds)	Length	Contents/Value (in terms of IMC Keyword values)
S:GJOBTBL	2	MAXG+5	Entry 0 = 0 Entry 1 = KEYIN (TEXTC) 2 = ALLOCAT (TEXTC) 3 = RBBAT (TEXTC) 4 = GHOST1 (TEXTC) 5 = MAXG = 0
SB:GJOBUN	1/4	MAXG+5	0

2.9.5 Interaction

SYNTAX to convert command to stack data block.

COREALLOC to allocate dynamic memory.

MODGEN to generate DEFs.

WRITELM to write output module.

READSTRG EXIT

2.9.6 Data Bases

KWDTBL is the keyword table for SYNTAX.

DYNAM is the virgin stack data block for SYNTAX, containing default or limits for each keyword.

2.9.7 Description

Upon entry, the total number of DP, 7T, and 9T units defined via :DEVICE is obtained (DEVS, R3) and the defaults for total, batch and on line values are computed and stored in DYNAM.

A BAL to SYNTAX puts command information and defaults from DYNAM into the stack, pointed to by R5. If the option UCYL has been used and the target system is not a no-RAD system, the option is ignored and an error message produced. If the option is used for a no-RAD system, then it must be 1 or 2 or the default 1 is used and an error message generated. If the command specified UNBLOCK greater than BLOCK, BLOCK is used. QUANTA and MINQUAN are converted from milliseconds to clock pulses.

MINTIME is converted from milliseconds to clock pulses and compared to the QUANTA value. If MINTIME is less than QUANTA, an error message is produced and the value set equal to QUANTA. MAXG value is compared with 3 and if less, is defaulted to 8 and produces an error message. If MAXG+MAXB+MAXOL is greater than 255, a message is generated indicating IMC processing has been terminated and control returned to P2CCI.

COCBUF value is then compared with 255 and if greater is set to 255 and an error message is generated. The values for EXPIRE and MAXEXPIRE are converted to hours and compared (EXPIRE must be ≤ MAXEXPIRE) unless -1 specified indicating never for expiration or retention period. The values specified for DP, 7T, and 9T in terms of total, batch and online are compared with the total defined via :DEVICE commands. Should any value exceed the total, an appropriate message is produced indicating the parameter in error and the value to be used. Then, COREALLOC is used to allocate memory. A BAL to MODGEN begins generation of DEFs after the data have been moved from the stack to the data record. A short loop to generate UB:BL and UB:FL exits from MODGEN with its BDRs, so another BAL to MODGEN is necessary to finish generating DEFs. Then, a BAL to WRITELM writes the output module and IMC exits to READSTRG in P2CCI.

2.9.8 IMC Messages

***UNBLOCK > BLOCK - SET EQUAL TO BLOCK ***MINTIME > QUANTA -- QUANTA VALUE USED ***LOGTIME OUT OF RANGE -- DEFAULT (3) USED ***INTIME OUT OF RANGE -- DEFAULT (15) USED ***EXPIRE OR MAXEXPIRE OUT OF RANGE -- 999 DAYS, 23 HOURS USED ***UCYL VALUE INVALID -- DEFAULT (1) USED	The value used for the specified parameter is in error. The designated value is used. IMC continues.
***MAXEXPIRE < EXPIRE -- EXPIRE VALUE USED ***COC BUFFERS > 255 -- 255 USED ***MAXG < 3 -- DEFAULT (8) USED ***MAXG > 255 -- DEFAULT (8) USED	The value used for the specified parameter is in error. The designated value is used. IMC continues.
***XXXXONLINE EXCEEDS TOTAL ON SYSTEM -- *XXXX*USED ***XXXXBATCH EXCEEDS TOTAL ON SYSTEM -- *XXXX*USED ***XXXXTOTAL EXCEEDS TOTAL ON SYSTEM -- *XXXX*USED ***USERS > 255 - IMC ABORTED	The 9T, 7T or disc pack value for online batch or total parameter exceeds the total number of device units specified via :DEVICE commands. The *XXXX* value is used. IMC continues. The sum of MAXG + MAXOL+MAXB exceeds 255. IMC restores the temp stack and exits to READSTRG in P2CCI.
***SWAPPER NOT DP -- UCYL IGNORED	UCYL has been specified for a target system in which the PSA is not defined on a disc pack. IMC ignores the option and continues processing.

2.9.9 Flow Chart

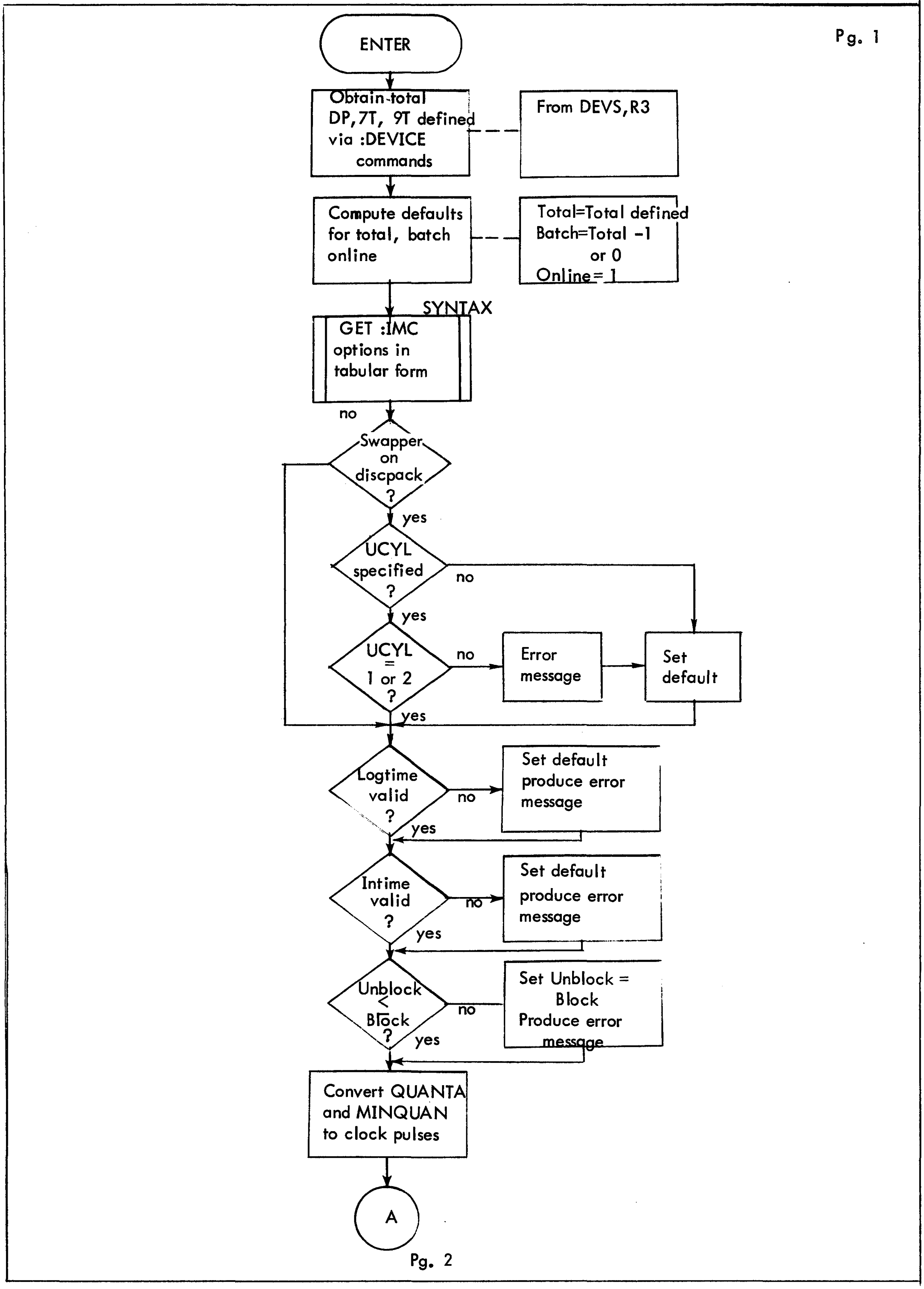

Pg. 1
ENTER
Obtain total DP,7T, 9T defined via :DEVICE commands
From DEVS,R3
Compute defaults for total, batch online
Total=Total defined
Batch=Total -1 or 0
Online= 1
SYNTAX
GET :IMC options in tabular form
no
Swapper on discpack ?
yes
UCYL specified ?
no
yes
UCYL = 1 or 2 ?
no
Error message
Set default
yes
Logtime valid ?
no
Set default produce error message
yes
Intime valid ?
no
Set default produce error message
yes
Unblock < Block ?
no
Set Unblock = Block Produce error message
yes
Convert QUANTA and MINQUAN to clock pulses
A
Pg. 2

Figure 2-16. Flow Diagram of IMC

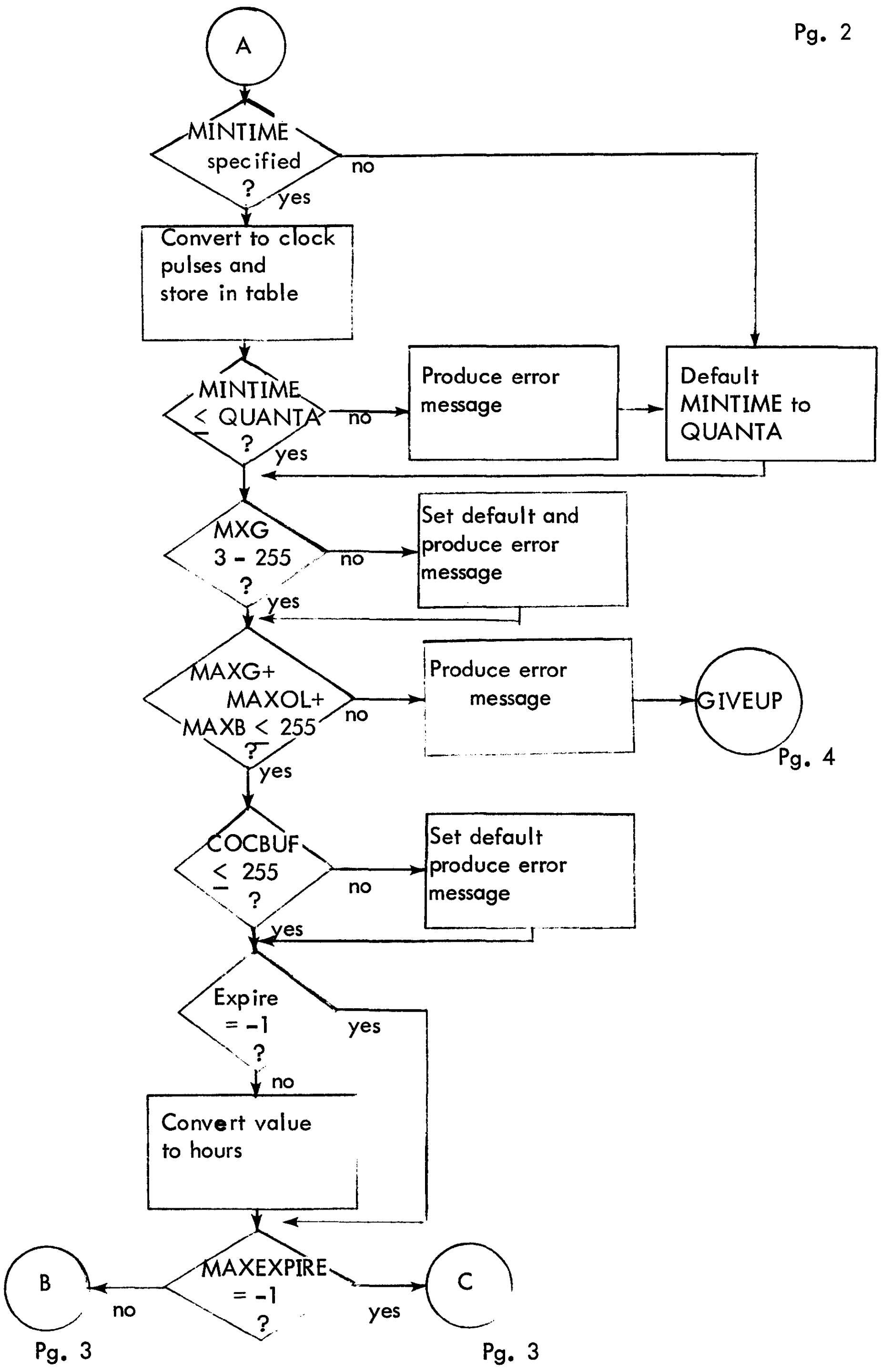


Figure 2-16. Flow Diagram of IMC (Cont.)

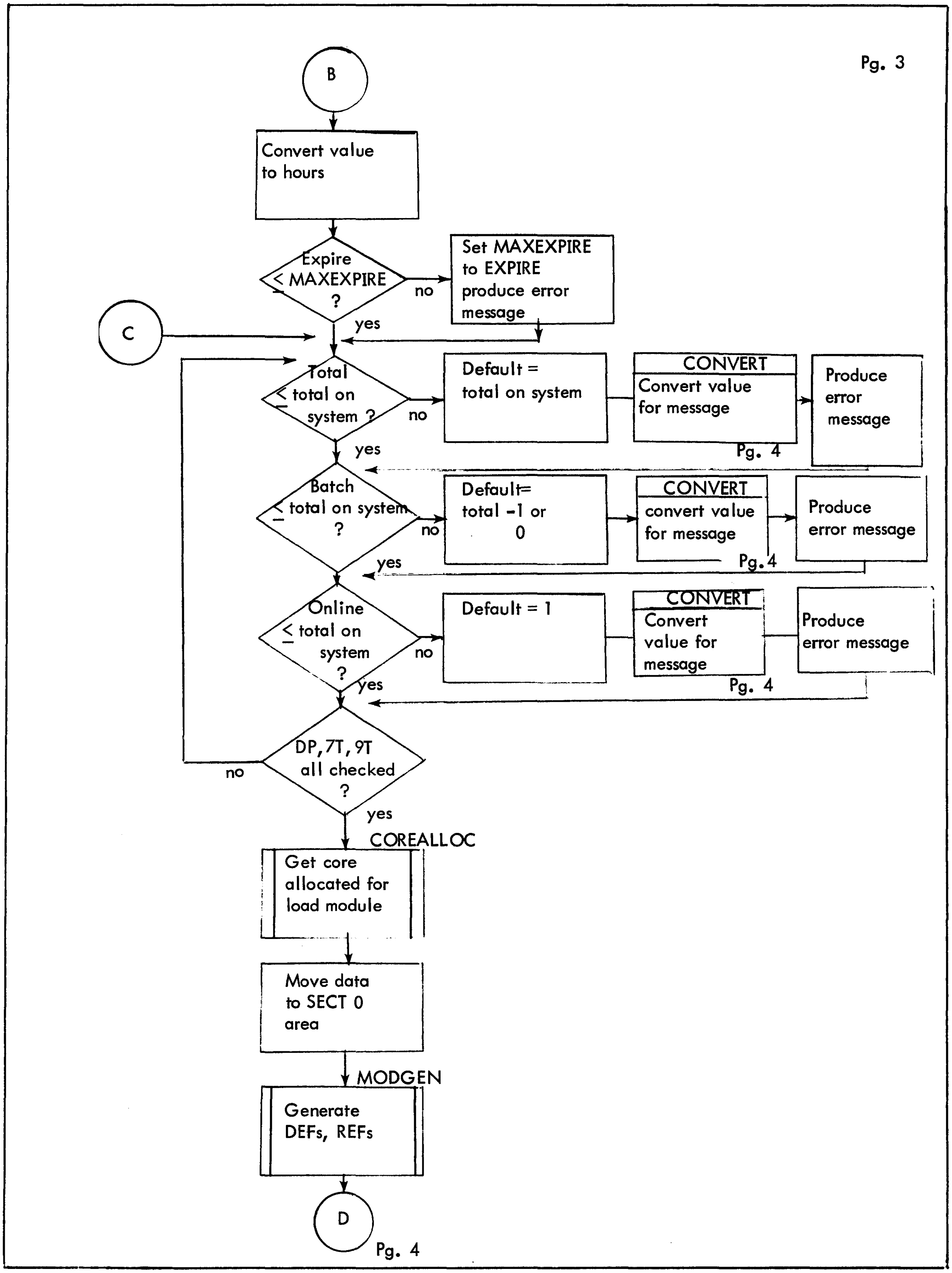


Figure 2-16. Flow Diagram of IMC (Cont.)

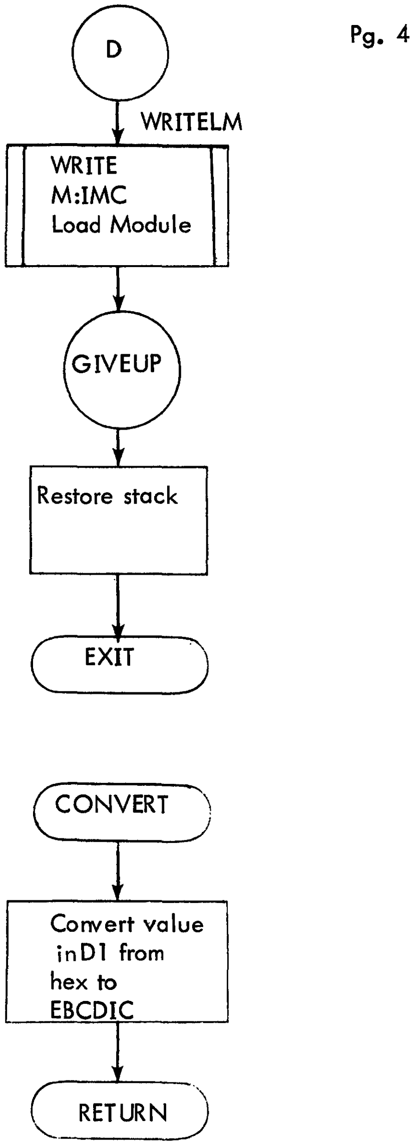


Figure 2-16. Flow Diagram of IMC (Cont.)

2.10 SPROCS

2.10.1 Purpose

To process SPROCS PASS2 control commands, creating the M:SPROCS load module for UTS systems only.

2.10.2 Usage

```
    B         SPROCS
              with: R7 pointing to control card PLIST
                    R0 pointing to temp stack pointer
                    R0 and R3 saved
                    Return is to READSTRG in P2CCI.
```

2.10.3 Input

Control card (:SPROCS)

TREEOO is the word in the TREE built by COREALLOC containing the size and address of the SECT 00.

BIG9FLG flag set by XMONITOR if target system >128K (i.e., BIG9 option specified; see 2.4.8).

2.10.4 Output

M:SPROCS module (Table 2-18).

Table 2-18. M:SPROCS Load Module Contents

Label	Entry Size (Wds)	Length	Contents/Value
PPROCS	Value	-	= 1 + # of Monitor overlays required (UTM Keyword and defaults) + #of shared processors required (NAMES + default)+ MOSPACE+ PSPACE+POSPACE+number of processors overlays required (decimal options+defaults)
MAXOVLY	Value	-	1 + # of Monitor overlays + MOSPACE+first processor index
PTEL	Value	-	Index of TEL (first default processor) (PTEL=MAXOVLY)
PCCI	Value	-	Index of CCI (second default processor)
PDEL	Value	-	Index of DELTA (seventh default processor)

Table 2-18. M:SPROCS Load Module Contents (cont.)

Label	Entry Size (Wrds)	Length	Contents/Value
BGNPMPRC	Value	-	PDEL-1
ENDPMPRC	Value	-	10
PNAMEND	Value	-	PPROCS less processor overlays and POSPACE
SPSIZE	Value	-	Second PSPACE option. This value, in NO-RAD systems must be ≤ # of granules per physical cylinder or the latter value is substituted.
P:NAME	2	PPROCS	First doubleword=0. Others = TEXTC names for monitor overlays, MOSPACE entries with TEXTC M:DUMLM, TEXTC names for shared processors, PSPACE entries with TEXTC M:DUMLM, and the rest = 0. Names are ordered with defaults first.
P:NAMEND	1	1	Is the address of last non zero entry+2 (i.e., address of 1st zero entry).
NXTPOVLY	1	1	0
PH:PDA	1/2	PPROCS	0
PB:LNK	1/4	PPROCS	0
PX:HPP[t]	1/4 or 1/2	PPROCS	0
PX:TPP[t]	1/4 or 1/2	PPROCS	0
PB:PSZ	1/4	PPROCS	0
PB:REP	1/4	PPROCS	0
PB:UC	1/4	PPROCS	0
PB:PVA	1/4	PPROCS	0
PB:C#[tt]	1/4	PPROCS	0
PB:DC#[tt]	1/4	PPROCS	0
P:SA	1	PNAMEND	Bits 8 to 31 of each word = 0. Bits 0 - 7 represent the flag options associated with the processor whose name is in the corresponding entry in P:NAME. Monitor overlays and M:DUMLM have zero flags Bit 0 = J flag Bit 1 = S flag Bit 2 = D flag (implies S)

[t] Entry size is 1/2 word when target system > 128K on Sigma 9. Otherwise, entry size is 1/4 word. Determining factor is BIG9 keyword on :UTM command (see 2.4).

[tt] These tables are generated only for NO-RAD systems.

Table 2-18. M:SPROCS Load Module Contents (cont.)

Label	Entry Size (Wrds)	Length	Contents/Value
			Bit 3 = P flag (implies S) Bit 4 = M flag Bit 5 = T flag Bit 6 = B flag Bit 7 = G flag Bits 5,6,7 = C flag
P:AC	2	PNAMEND	First entry = X'7FFFFFFF', -1; other entry = 0
P:TCB	1	PNAMEND	0
PH:DDA	1/2	PNAMEND	0
PB:DSZ	1/4	PNAMEND	0
PB:DCBSZ	1/4	PNAMEND	0
PB:HVA	1/4	PNAMEND	0
PBT:LOCK	Bits	PNAMEND	0

2.10.5 Interaction

COREALLOC	is used to set up memory for load module generation
SYNTAX	is used to decode the control command
MODGEN	is used to generate DEFs
WRITELM	is used to write the M:SPROCS module
READSTRG	is exit location

2.10.6 Data Bases

KWDTBLO is a two part input table for SYNTAX. The first part is a set of SYNTAX control halfwords that permits SYNTAX to process the non-standard format of the :SPROCS card (see SYNTAX Chapter 6.8). The second part is a normal SYNTAX Keyword table.

DYNAM is the virgin stack data block for SYNTAX, containing two doubleword table pointers and four normal keyword entries.

FLGS is a byte table of valid flag characters

FLAGS is a word table corresponding to FLGS containing an internal representation of each flag.

STDOLY is a two-word-entry table of default monitor overlay names in TEXTC form.

STDPROC is a two-plus-n-word-entry table of default processor names. Each entry has the name in TEXTC format in the first two words, followed variously by any number of words containing either a binary number (indicating a number of overlays) or up to four TEXT flag characters (left-adjusted, blank-filled).

2.10.7 Subroutine

SQUEEZE moves a string of words in memory from one location (starting at the address in R2) to another (starting at the address in R12). As each word in moved, its old location is zeroed and R2 and R12 are incremented. If a single zero word is encountered, it is not moved (R2 is incremented, but R12 is not). When two consecutive zero words are encountered, SQUEEZE returns *R11 with R2 pointing to the first zero word, R12 pointing to where it was moved to, and R13 zero.

2.10.8 Description

SPROCS begins, unconventionally, by using COREALLOC to obtain some work space. 400 words are reserved for REFDEF stack and EXPRESSION stack, and the remainder of available core for the data record and its relocation dictionary. Then preparation is made to use SYNTAX to decode the control command. The data record (from TREE00 is divided in half. The end address of each half is put in the table upper limit word of the corresponding table pointer in DYNAM. STDOLY is moved to the first half, and the address of the word after the last word moved is put in the table pointer in DYNAM. STDPROC is moved to the second half, and its table pointer set up similarly. Then SYNTAX decodes the card image. The result in memory is the same as before except that more entries may have been added to the two tables and the table pointers updated accordingly (now in the copy of DYNAM in the stack). New entries are in the same format as the default entries (see DATA BASES). (Section 2.10.6) If a no-RAD system is being generated, SPROCS, upon return from SYNTAX, validates the second value specified on the PSPACE option. This value representing the size reserved for one space processor, must be less than or equal to the number of granules per physical cylinder on the disc pack containing the PSA area. See 2.2.7 for definition of this number. Should the PSPACE value be invalid, the previously computed number is substituted and an error message produced. The term S:CYLSZ in the message refers to this completed value. SPROCS then adds MOSPACE M:DUMLM entries to the first table. The SQUEEZE subroutine is used to attach the second table to the end of the first one and remove any extraneous zero words from it. If SQUEEZE did not reach the end of the second table (because of two consecutive zero words in the middle of it) it is reentered at the skip-zero-words point until it finishes the table. Then PSPACE M:DUMLM entries are appended to the end of the (now only one) table. The table now contains no zero words.

Every word in the table is one of four types:

1. First word of a TEXTC name (byte 0 between 1 and 7).
2. Second word of a TEXTC name (follows first word always).
3. Binary number representing number of overlays (byte 0 = 0).
4. Flag word in TEXT form (byte 0 greater than 7).

The name count is now initialized to one, the overlay count to POSPACE and the NAME loop is entered. NAME looks at all but type 2 words in the table. Name count is incremented for each type 1 word. The next word (type 2) is skipped and an internal flag word is initialized to zero. Words are then examined until the next type 1 is encountered. Type 3 words are added to the overlay count. Each byte of a type word is converted via FLGS and FLAGS to an internal representation and OR'ed into the internal flag word (R15) unless it is not valid or is P when the name is not :PO through :P9, in which case it is ignored with an explanatory message. If the P flag is validly encountered, the name (previous type 1 and 2 words) is changed from :Pn to :Pnn. The internal flag word is stored in the word following the previous type 2 word. When the next type 1 word or zero is encountered, SQUEEZE is called to slide it and what follows down to the internal flag word, unless no flags were encountered, in which case the SQUEEZE is to the type 2 word.

When NAME is finished with the table, it consists of only two- or three-word entries. The first two words are TEXTC name, and the third, if it exists, is a flag word, in internal form. Name count has PNAMEND in it, and overlay count has PPROCS-PNAMEND. At this point MODGEN is entered and the tables P:NAME through P:SA are DEF'ed. The entry size of two tables, PX:HPP and PX:TPP depends on whether or not the BIG9 option was specified on the :UTM command. This information is passed via a DEF'ed location in P2CCI stack. If BIG9 has been specified, the entry size is 1/2 word, otherwise, these tables have byte size entries. Tables, PB:C# and PB:DC# are only generated if the target system is to be of the NO-RAD type. Now that the core location of P:SA is determined, the flag words are converted to P:SA form and put in corresponding entries to P:SA. As they are moved, SQUEEZE squeezes them out of P:NAME, resulting in a doubleword table of TEXTC names. There can be no overlays of the end of the two or three word entry and P:SA because there are at least 4 byte tables between P:NAME and P:SA.

Then the rest of the module is generated, WRITELM writes it to the M:SPROCS file, and SPROCS exits.

2.10.9 SPROCS Messages

***INSUFFICIENT SPACE -- SPROCS ABORTED

There is insufficient room in the table area for processor overlays. SPROCS restores the tempstack and exits.

***ILLEGAL FLAG f FOR XXXXXXXX -
FLAG IGNORED

The flag "f" is not an S, J, P, D, M, T, B, G or C or is a P when XXXXXXXX is not Pi. Only "f" is ignored, even when it occurs in a multiflag option field. SPROCS continues.

****PSPACE SIZE >S:CYLSZ--S:CYLSZ USED

For a no-RAD target system, the second value on the PSPACE option is not ≤ to the number of granules per physical cylinder (S:CYLSZ) for the disc pack on which the PSA area has been defined. SPROCS substitutes the correct value and continues.

2.10.10 Flow Charts

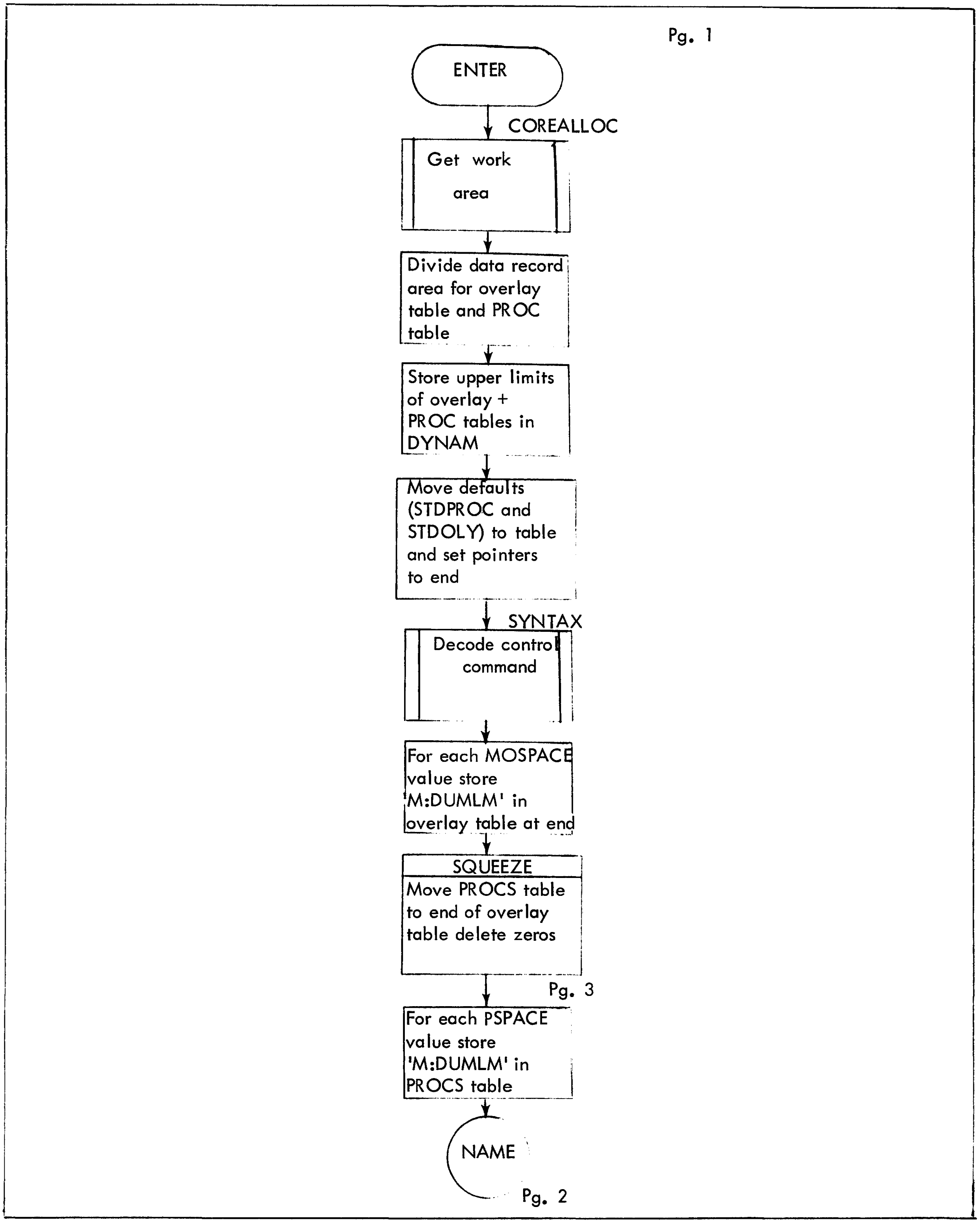
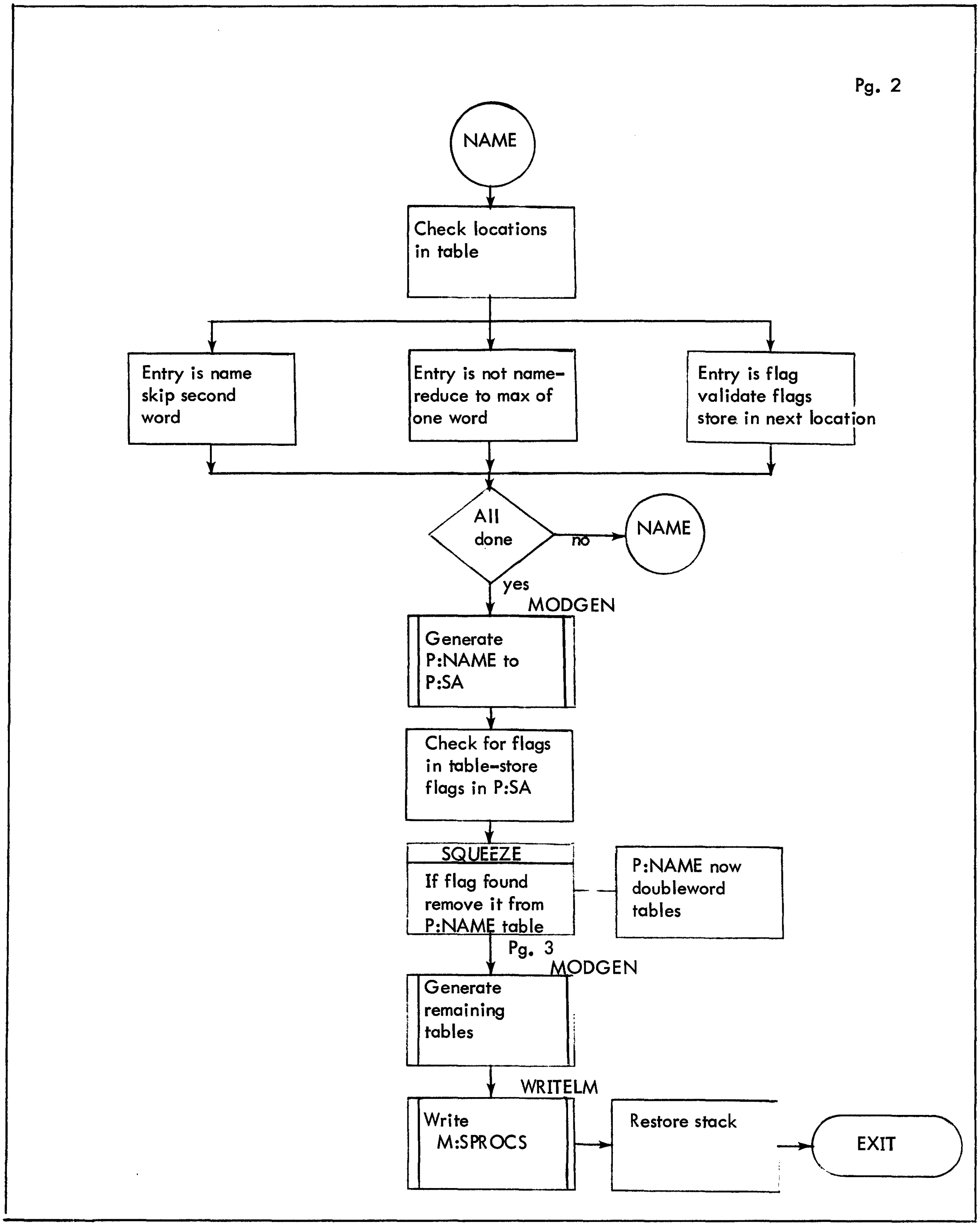

Pg. 1
ENTER
COREALLOC
Get work area
Divide data record area for overlay table and PROC table
Store upper limits of overlay + PROC tables in DYNAM
Move defaults (STDPROC and STDOLY) to table and set pointers to end
SYNTAX
Decode control command
For each MOSPACE value store 'M:DUMLM' in overlay table at end
SQUEEZE
Move PROCS table to end of overlay table delete zeros
Pg. 3
For each PSPACE value store 'M:DUMLM' in PROCS table
NAME
Pg. 2

Figure 2-17. Flow Diagram of SPROCS



Pg. 2
NAME
Check locations in table
Entry is name skip second word
Entry is not name-reduce to max of one word
Entry is flag validate flags store in next location
All done
no
NAME
yes
MODGEN
Generate P:NAME to P:SA
Check for flags in table-store flags in P:SA
SQUEEZE
If flag found remove it from P:NAME table
P:NAME now doubleword tables
Pg. 3
MODGEN
Generate remaining tables
WRITELM
Write M:SPROCS
Restore stack
EXIT

Figure 2-17. Flow Diagram of SPROCS (Cont.)

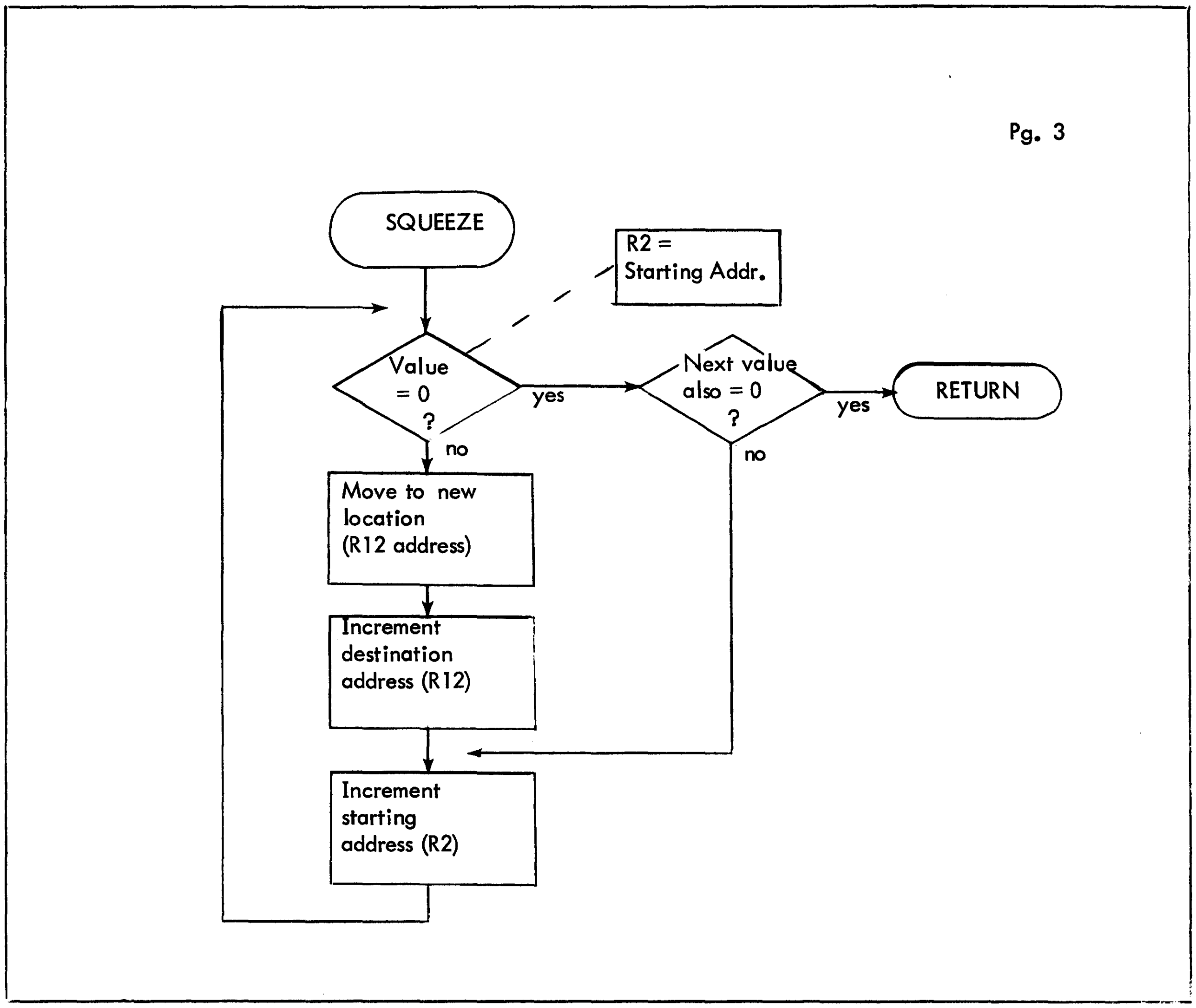


Figure 2–17. Flow Diagram of SPROCS (Cont.)

2.11 FRGD

2.11.1 Purpose

To process the :FRGD and :INTLB commands and to generate the load module M:FRGD which defines the foreground characteristics for BPM/BTM systems only.

2.11.2 Usage

```
    B       FRGD
            With   R7 pointing to the control card PLIST
                   R0 pointing to the temp stack pointer
                   R3 pointing to the PASS2 stack data
    Return is to READSTRG (if :INTLB was processed)
                READOK (if no :INTLB encountered)
                   R3 and R7 are equivalent to that upon entry.
```

2.11.3 Input

Control Cards (:FRGD and optional :INTLB) images

2.11.4 Output

M:FRGD load module (Table 2-19).

N.B:

The column entitled "Internal Control" refers to the FRGD processor internal control routine that generates the tables or values and apply to all subsequent tables up to the next entry in the column.

Table 2-19. M:FRGD Load Module Contents

Label	Entry Size (Wrds)	Length	Contents or Value	Internal Control
NFRGD	Value	-	NFRGD	DNFRGD
FPDESIZE	Value	-	12	"
FPDTLINK	Value	-	0	"
FPDTCF	Value	-	1	"
FPDTNAME	Value	-	2	"
FPDTP1	Value	-	4	"
BAFPDTP1	Value	-	16	"
FPDTSA	Value	-	4	"
FPDTP2	Value	-	5	"

Table 2-19. M:FRGD Load Module Contents (cont.)

Label	Entry Size (Wrds)	Length	Contents or Value	Internal Control
BAFPDTP2	Value	-	20	DNFRGD
FPDTTCB	Value	-	5	"
FPDTTF	Value	-	6	"
BAFPDTTF	Value	-	24	"
FPDTTA	Value	-	6	"
FPDTDA1	Value	-	7	"
FPDTDA2	Value	-	8	"
FPDTCSS0	Value	-	9	"
FPDTCSS1	Value	-	10	"
FPDTCSS2	Value	-	11	"
FPDTH1	1	1	Address of FPDT	"
FPDTH2	1	1	0	"
FPDTT	1	1	0	"
FPDTIAC	1	1	0	"
FPDT	12	NFRGD	1st word=FLINK to next entry other words = 0	"
FPDTEND	1	1	(is end of FPDT+1)	"
NINT	Value	-	NINT	DNINT
FIDTLINK	Value	-	0	"
FIDTPDA	Value	-	0	"
FIDTCF	Value	-	1	"
FIDTIL	Value	-	2	"
BAFIDTIL	Value	-	8	"
FIDTGL	Value	-	2	"
FIDTWD	Value	-	3	"
FIDTPSM	Value	-	4	"
FIDTBAL	Value	-	5	"
FIDTLPSD	Value	-	6	"
FIDTXPSD	Value	-	8	"
FIDTMCLK	Value	-	3	"
FIDTMCSA	Value	-	4	"
FIDESIZE	Value	-	12	"
MCLKH	1	1	0	"
MCLKT	1	1	0	"
FIDTH	1	1	address of FIDT	"

Table 2-19. M:FRGD Load Module Contents (cont.)

Label	Entry Size (Wrds)	Length	Contents of Value	Internal Control
FIDTIAC	1	1	0	DNINT
FIDT	12	NINT	1st word = FLINK to next entry Other words = 0	"
FIDTEND	1	1	(is end of FIDT+1)	"
CLOCKS	2	3	0	"
NCTQE	Value	-	CTQ	DCTQ
CTQESIZE	Value	-	4	"
CTQLINK	Value	-	0	"
CTQINT	Value	-	1	"
CTQNAME	Value	-	2	"
CTQH1	1	1	address of CTQ	"
CTQH2	1	1	0	"
CTQT	1	1	0	"
CTQ	4	CTQ	1st word = FLINK to next entry Other words = 0	"
INTSPGMT	4	INTS	See table 2-21 for contents	INTS
INTSPGMTEND	1	1	(is end of INTSPGMT+1)	"
RJIT	-	164	See Insert Below	DRJIT
FCOMLL	1	1	0	"
FCOMUL	1	1	0	"
RESDFLL	1	1	0	"
RESDFUL	1	1	0	"
FBUFLL	1	1	0	"
CTINT	Value	-	CT	DCT
FDFRSIZE	Value	-	Interrupt group of CT	"
CTWDA	1	1	X'6D201200'+level of CT interrupt (i.e., write direct to arm, enable CT)	"
CTIDE	1	2	0	"
CTGL	1	1	Group level indicator of CT	"
CTWD	1	5	1st word = X'6D201700' level of CT interrupt (i.e., write direct to trigger CT) other words = 0	"
CTPSD	1	6	Words 0 - 1 = 0 Words 2 =X'40000000'+CNTASK (REF)	"

Table 2-19. M:FRGD Load Module Contents (cont.)

Label	Entry Size (Wrds)	Length	Contents of Value	Internal Control
			Word 3 = X'07000000'	
			Word 4-5 = 0	
CTXPSD	1	1	XPSD,8 CTPSD	DCT
CTINTENV	1	1	X'80000000'+CTIDE	"
CURPDA	1	1	0	"
CURINT	1	1	0	"
BKPSD	1	2	0	"
CTQHC	1	1	0	"
CTQTD	1	1	0	"
CTDFRFLG	1	1	0	"
FDFRFLAG	1	1	0	"
FDFRINTS	1	FDFRSIZ +1	0	"
NFIPOOL	Value	-	FIPOOL	DOTHERS
NFFPOOL	Value	-	FFPOOL	"
FIBUFSIZE	Value	-	256	"
FFBUFSIZE	Value	-	512	"
FCOMSIZ	1	1	FCOM*512	"
RESDFSIZ	1	1	RESDF*512	"
RESDFSIZP	1	1	RESDF*512	"
IBUFTBL	1	2*FIPOOL	0	"
FBUFTBL	1	2*FFPOOL	0	"
STOPTBL	1/4	#tape defined on system	0	"
INTLBSIZ	Value	-	# of INTLB Entries	DINTLB
INTLB1	1/2	# of INTLB	See Table 2-20 for contents	"
INTLB2	1/2	# of INTLB	See Table 2-20 for contents	"

RJIT Contents

Word 0 = 0

Words 1-2 = :SYSRT (TEXT)

Words 3-70 = 0

Words 71 = $+4

Words 72-73 = M:OC (TEXTC)

Word 74 = $+2

Word 75 = 0

Word 76 = X'00200003'

Word 77 = X'00060002'

Words 78 - 163 = 0

2.11.5 Subroutines Used

CHARSCAN	(used to check a specific character for syntax)
DECSCAN	(used to obtain a field whose value is decimal/hexadecimal).
HEXSCAN	
QUOTSCAN	(used to scan a field containing a Keyword)
NAMSCAN	(used to scan a field containing a name)
MODIFY	(used to generate the load module)
PRINTMSG	(display error information)

2.11.6 Special Restriction

The option INTS must be contained wholly on one physical card image.

2.11.7 Description

2.11.7.1 Overview

The FRGD processor is entered when P2CCI encounters the :FRGD command. The processor initializes its temp stack area by moving the DYNAM table into the stack. It then obtains 100 pages of core for use as its work area to generate the load module M:FRGD.

The initial syntax analysis of the :FRGD command is performed in the GETKEY routine which processes a parenthetical expression up to the first comma and then determines what the Keyword is and enters an appropriate routine which in turn processes the remainder of the expression and returns. This continues until the entire command is processed. Then the FRGD portion of the load module is generated. When completed, the next control command is read and if it is :INTLB it is processed by the GETOPLB and OPLBENT routines. GETOPLB syntactically checks the parenthetical expression and OPLBENT saves the "label" and "loc" in intermediate tables (See Table 2-20). The load module is then written and FRGD releases its work area restores the temp stack and returns to P2CCI at READSTRG.

When the FRGD processor obtains the next control command and it is not :INTLB, the load module M:FRGD is written, and the work area released, the stack restored and control is returned to P2CCI at READOK.

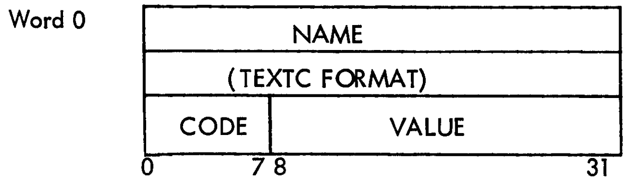
When FRGD requests the next control command and it is a Monitor control command, P2CCI will enter FRGD at FRGDLMX and the remainderof the FRGD procedure is as described for no :INTLB encountered.

2.11.7.2 Details

When the GETKEY routine is entered, it obtains a Keyword and determines through the FRGDOPT table which subroutine to enter for further processing. For all Keywords except "INTS" the subroutine obtains the value (i.e., value, address, or size in the parenthetical expression), converts it to binary and checks if the value is within range. The value is saved in the temp stack for use in generating the load module. If the

Keyword is INTS, the subroutine entered generates an intermediate table (see Table 2-20) containing the information from all expressions within the INTS option. When the complete parenthetical expression has been processed, GETKEY returns and the next expression is analyzed.

The LMFRGD routine allocates the work area for the M:FRGD load module and generates the data. The procedure used when generating the necessary SECT.00 information is controlled by various tables within the FRGD processor. The routine PROCDEF interrogates a given table and performs the necessary function according to Figure 2-18:



where

NAME a TEXTC formatted name of a value, table or data word which is to be externally defined.

CODE = 0 NAME is to be equated to VALUE

= 1 NAME is to be equated to the address of a data word or table where VALUE is the number of words in table

= 2 NAME is to be equated to the value found in the temp stack pointed to by the displacement VALUE.

= 3 NAME is to be equated to the address of a table or series of tables to be generated. This code indicates that each table or series of tables will be generated according to pre-described algorithms

CODE = 4 NAME is to be equated to the value pointed to by VALUE multiplied by 4.

Figure 2-18. PROCDEF Table

The internal tables controlling the generation of the ":FRGD" command's portion of the "M:FRGD" load module consist of:

DNFRGD	for the NFRGD option
DNINT	if NINT ≠ 0
DCTQ	if CTQ ≠ 0
DINTS	if the INTS option was specified
DRJIT	always used
DCT	for the CT option, where (X'60'≤CT≤X'13F')
DOTHERS	always used.

When an internal controlling table is being processed, each entry's code is interrogated. An appropriate routine is then entered to generate the necessary load module information.

When the code is 0, the routine GENT0, is entered and it sets up the PLIST required by the MODIFY routine for the generation of an external definition whose value is a constant.

When the code is 1, the routine GENT1 is entered and it sets up the PLIST required by the MODIFY routine for the generation of an external definition whose value is the address in the load module of a data word. The data word may be a one-word table or it may be the first word of a multi-word table. Each word is set to zero.

When the code is 2, the routine GENT2 is entered and it sets up the PLIST required by the MODIFY routine for the generation of an external definition whose value is an constant.

When the code is 3, the routine GENT3 is entered. The address of the current internal table's entry is used to search another internal table (vix., INVECT), for a match. When found, the relative position of the match in the INVECT table is used as an index into the OUTVECT table. The OUTVECT entry identifies the entry point to a special subroutine which generates the desired data, tables, external definitions or references for a given parameter. These special subroutines are referred to as "Special Compile Processors".

When the code is 4, the routine GENT4 is entered and it sets up the PLIST required by the MODIFY routine for the generation of an external definition whose value is an expression which is evaluated into a constant. The expression is a constant or value multiplied by four (e.g., a byte displacement into a table).

FRGD proceeds to finish creation of SECT.00 by generating the information concerning :INTLB command. The internal table controlling this portion of the load module generation is DINTLB.

Table 2-20. :INTLB INTERMEDIATE TABLES

OP	0 0	OPL_1
	OPL_2	- - - -
	- - - - -	OPL_n

LC	0 0	LOC_1
	LOC_2	- - - -
	- - - -	LOC_n

where

OP is the op-label table pointed to by INTLBOP

LC is the location table pointed to by INTLBLOC

Table 2-21. INTS INTERMEDIATE TABLE

IN	#E	NAV			Word 0
	UNUSED				1
(INTSPGMT)	#C	C_i	C_2	C_3	2
	- -	- -	- -	- -	3
	- -	- -	C_n	-	4
	FIMAX	PAGES	SIZE		5
					.
					.
					.
					.

where

IN = the INTS table pointed to by INTSAREA

#E=number of entries in table

NAV = next available address in table

#C = number of characters in name

C_i, C_2, ... C_n = characters in name

FIMAX = FIMAX value from INTS expression
PAGES = Pages value from INTS expression
SIZE = Size value from INTS expression
INTSPGMT = Starting address of the table DEF for M:FRGD

2.11.8 FRGD Messages

***DELIMITER ERROR — Invalid delimiter encountered. FRGD continues at next parenthetical expression.

***UNKNOWN KEYWORD — The keyword in a parenthetical expression is unknown. FRGD continues at next parenthetical expression.

***DELIMITER ERROR, PROCESSOR ABORTED — The end of ":FRGD" or ":INTLB" command cannot be found. FRGD returns to PASS2 control.

***INVALID DECIMAL VALUE
***INVALID HEXADECIMAL VALUE — The value field is expected to be either decimal or hexadecimal. FRGD continues at next parenthetical expression.

***VALUE ERROR, DEFAULT TAKEN — The value specified is less than the default or is less than a previously defined value of the same type. FRGD continues at next parenthetical expression.

***NAME INVALID or > 11 CHAR. OR > 2 CHAR. — The name is not alphanumeric (at least one alpha), is greater than 11 characters in "INTS" option, or is greater than two characters in INTLB command's options. FRGD continues at next parenthetical expression.

***SIZE/PAGES/FIMAX VALUE INVALID — The size, page, or fimax value after the INTS option is not a valid decimal number. FRGD continues at next parenthetical expression.

***NOT ENOUGH CORE AVAILABLE TO GEN LM, PROC. ABORTED — The work area for FRGD has been used and no more is available. FRGD returns to PASS2 control.

***GEN. OF LM UNSUCCESSFUL — There is not enough work area available. FRGD returns to PASS2 control.

***CT FIELD NOT = >60 OR =< 13F, PROC. ABORTED — The address field for the CT option is too small or too large, or there is no CT option specified FRGD returns to PASS2 control.

***NFRGD FIELD MISSING OR INVALID, PROC. ABORTED — The value field for the NFRGD option is invalid or missing. FRGD returns to PASS2 control.

2.11.9 Flags Used by FRGD

VNFRGD = NFRGD field's value

VCT = CT field's value (address)

VFCOM = FCOM field's value (size)

VRESDF = RESDF field's value (size)

VFIPOOL = FIPOOL field's value

VFFPOOL = FFPOOL field's value

VNINT = NINT field's value

VCTQ = CTQ field's value

LMAREA = base address in work area (following INTS option's information).

WORKSIZE = number of pages in work area.

INTSAREA = base address of INTS option's information and also entire work area.

SAVER1 = control command type which follows ":FRGD" command:
= first four characters of next command if not ":INTLB" command.
= -1 if next command is a Monitor system command (i.e., end of file to PASS2).
= 0 if ":INTLB" command was found; Initially, SAVER1=-1

#INTLB = number of entries in intermediate table generated as a result of the ":INTLB" command

INTLBOP = address of OP table

INTLBLOC = address of LC table

#AVTENT = obtained from the REF AVRTBLGTH # tapes drives defined on system

2.11.10 Internal Routines

FRGD main entry, initializer, and controller
Register 3 = address of PASS2 information in temp stack
Register 7 = address in temp stack of command processing PLISTs.

FRGDOP process FRGD command's parenthetical expressions.

INTLBOPC process INTLB command s parenthetical expressions.

FRGDLMX there is no INTLB command, but instead, there is a Monitor system control command (i.e., end of file to PASS2).

GETKEY check for "(", get keyword, and check for "," in next parenthetical expression.

PNFRGD set conditions for processing value for NFRGD option.

PCT set conditions for processing value for CT option.

PFCOM set conditions for processing value for FCOM option.

PRESDF set conditions for processing value for RESDF option.

PFIPOOL set conditions for processing value for FIPOOL option.

PFFPOOL set conditions for processing value for FFPOOL option.

PNINT set conditions for processing value for NINT option.

PCTQ set conditions for processing value for CTQ option.

COMRET set conditions for value in decimal. All other values, above, are hexadecimal.

COMRETA obtain value.

PINTS process INTS option.

GETVAL obtain value, convert it to binary, check if value is valid.
(i.e., use default or use value), and check for ")".
Register 13 = 0 value in hexadecimal
= 1 value in decimal
Register 14 = address in temp stack where value is to be saved.

GETOPLB check for "(", get oplabel, check for "," get location value, and validate it, and check for ")" in :INTLB command's next parenthetical expression.
Register 2 = address in work area of next available word for interim tables.

OPLBENT saves oplabel and location value in next available entry in interim tables for :INTLB command.
Register 12 = hexadecimal location
Register 13 = oplabel in TEXT format
Register 2 = address of next available word in work area

LMINT this routine takes the interim tables generated by OPLBENT and adds them to SECT.00 portion of the M:FRGD load module.

CNVDEC convert EBCDIC decimal to hexadecimal.
exit: Register 12 = converted value

CNVHEX convert EBCDIC hexadecimal to hexadecimal
exit: Register 12 = converted value.

LMFRGD allocate work area for the M:FRGD load module. Request the generation of load module information according to the internal control tables.
Register 3 = address in temp stack of PASS2 information
Register 6 = address in temp stack of FRGD processor information
Register 7 = address in temp stack of PLIST used in processing control command
Register 15 = Register 3 = temporary save of Register 3.

WRITLM write M:FRGD load module to M:FRGD file.

WRITETM do actual write for M:FRGD load module parts.
Register 8 = buffer address
Register 9 = buffer size (bytes)
Register 10 = address of key (load module elements key)

PROCDEF interrogate each internal control table's entry, and give control to appropriate processor according to entry's code.

Register 2 = address in work area to generate tables.
Register 5 = address of next entry in internal control table.
Register 15 = address of end of current internal control table.

GENT0 processes Code = 0 type internal control table entries.
Register 2 = address in work area of next available word.
Register 5 = address of current internal control table entry.
Register 7 = address of MASTER PLIST for MODIFY.

GENT1 processes Code = 1 type internal control table entries.
Register 2 = address in work area of next available word.
Register 5 = address of current internal control table entry.
Register 7 = address of MASTER PLIST for MODIFY.

GENT2 processes Code = 2 type internal control table entries.
Register 2 = address in work area of next available word.
Register 5 = address of current internal control table entry.
Register 7 = address of MASTER PLIST for MODIFY.

GENT3 processes Code = 3 type internal control table entries by searching INVECT table for a corresponding address equivalent to the current internal control table entry, and then enter appropriate routine as indicated by the same relative entry in OUTVECT table.
Register 2 = address in work area of next available word.
Register 5 = address of current internal control table entry
Register 7 = address of MASTER PLIST for MODIFY

GENT4 processes Code = 4 type internal control table entries.
Register 2 = address in work area of next available word.
Register 5 = address of current internal control table entry
Register 7 = address of MASTER PLIST for MODIFY

VALU obtain the value from the internal control table entry whose code is 0, 1, or 2.
Register 13 = 0
    and code = 0 value is in current entry
    and code = 2 value in current entry is index in temp stack where value is found.
Register 13 $\neq$ 0 value is relative to the base address of SECT.00 in load module.
    exit: Register 14 = actual value.

MODF set up MASTER PLIST and sub-PLISTs with desired information (DEF, EXPR, or DICT) and then go to MODIFY routine.
Register 4 = VALUE$_1$, if applicable
Register 5 = address of NAME$_1$
Register 7 = address of MASTER PLIST for MODIFY
Register 12 = relative address in temp stack of desired sub-PLIST (DEF, EXPR, or DICT).

Register 13 = for DEF, DICT = NAME$_2$
for EXPR = address of NAME$_2$
Register 14 = VALUE$_2$
Register 15 = RELDICT.00 code, if applicable

The following is a list of the routines that generate the DEF,RELDICT.00 changes and the data in SECT. for the appropriate tables.

CFPDTH1
CFPDT
CFIDTH
CFIDT
CCLOCKS
CCTQH1
CCTQ
CINTSPGMT
CRJIT
CCTWDA
CCTIDE
CCTGL
CCTWD
CCTPSD
CCTXPSD
CCINTENV
CBKPSD
CFCOMSIZ
CRESDFSIZ
CRESDFSIZP
CINTLB1
CINTLBSIZ
CINTLB2
DOIBTB
DOFBTB
DOSPTB
CFDFR

Special error routines include: DELY, DEL, COMERR, KEY, FINDRPAR, DELX, FRGDEXIT, EOCCSCAN, DEC, HEX, DEFAULT, NAMY, NAM, SIZPAG, NOROOM, NOROOMX, COMABORT, MOD, CTERR, NFRGDERR

2.11.11 Flow Chart

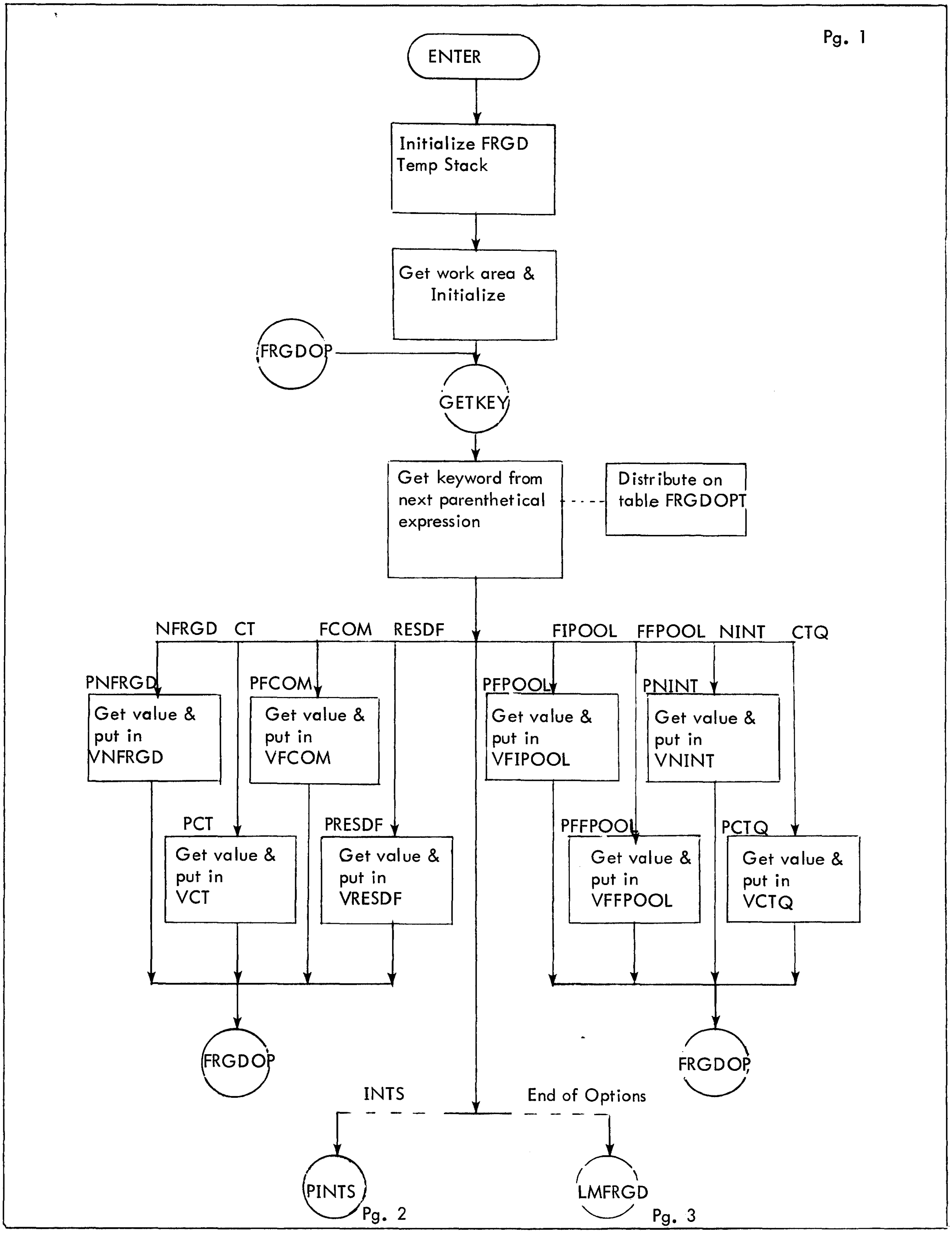


Figure 2-19. Flow Diagram of FRGD

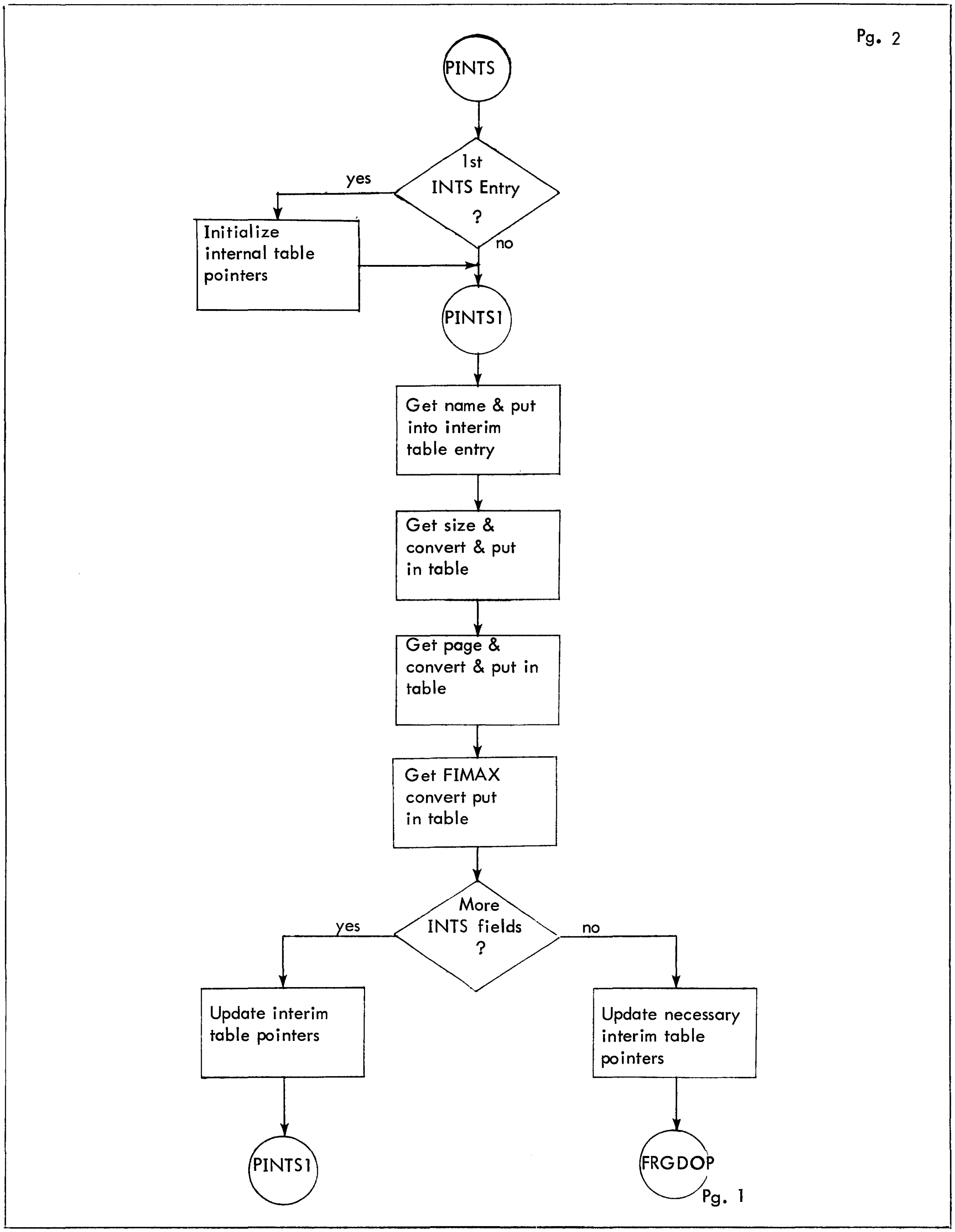

Pg. 2
PINTS
1st INTS Entry ?
yes
no
Initialize internal table pointers
PINTS1
Get name & put into interim table entry
Get size & convert & put in table
Get page & convert & put in table
Get FIMAX convert put in table
More INTS fields ?
yes
no
Update interim table pointers
Update necessary interim table pointers
PINTS1
FRGDOP
Pg. 1

Figure 2-19. Flow Diagram of FRGD (Cont.)

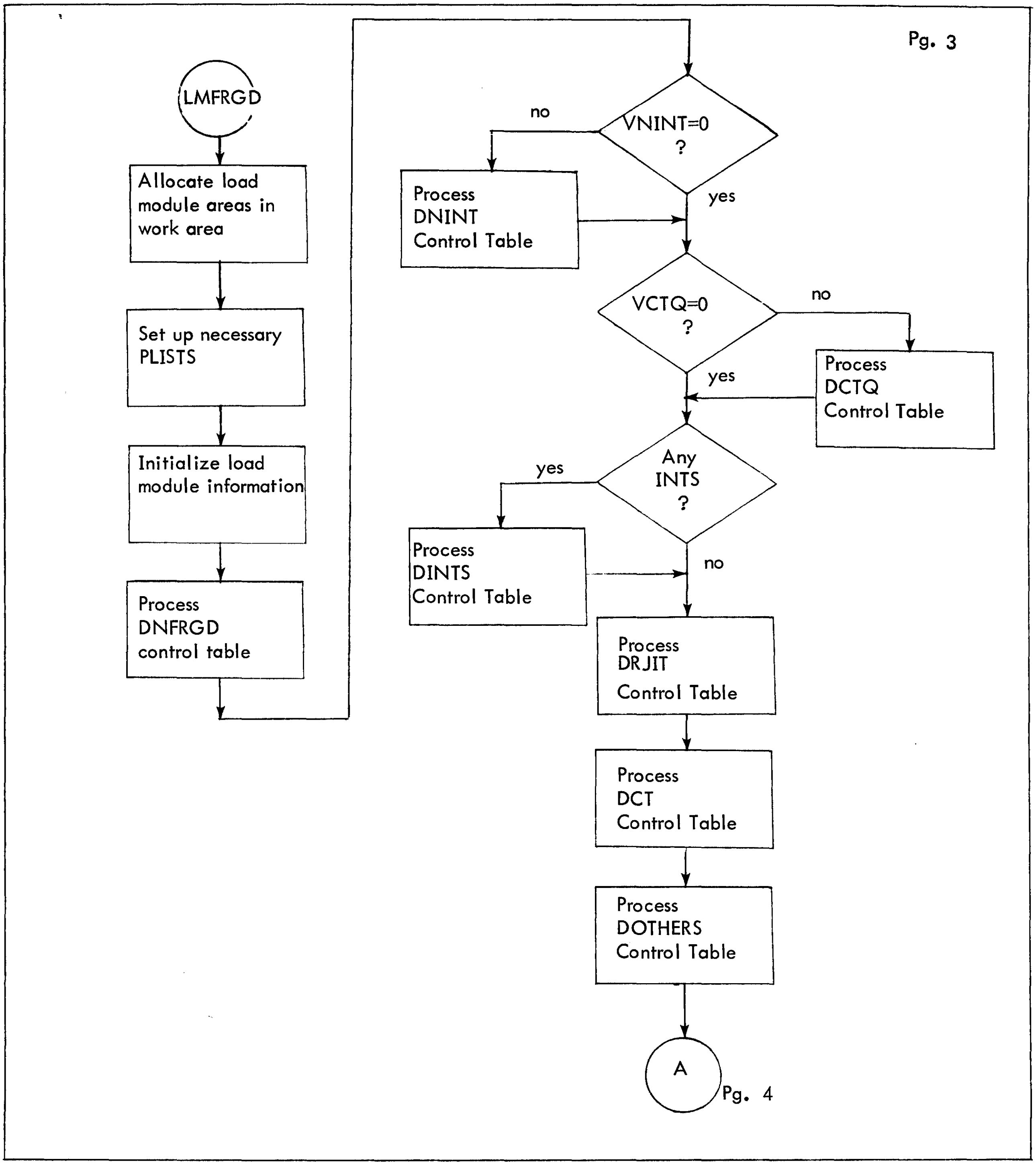

Pg. 3
LMFRGD
Allocate load module areas in work area
Set up necessary PLISTS
Initialize load module information
Process DNFRGD control table
VNINT=0 ?
no
yes
Process DNINT Control Table
VCTQ=0 ?
no
yes
Process DCTQ Control Table
Any INTS ?
yes
no
Process DINTS Control Table
Process DRJIT Control Table
Process DCT Control Table
Process DOTHERS Control Table
A
Pg. 4

Figure 2-19. Flow Diagram of FRGD (Cont.)

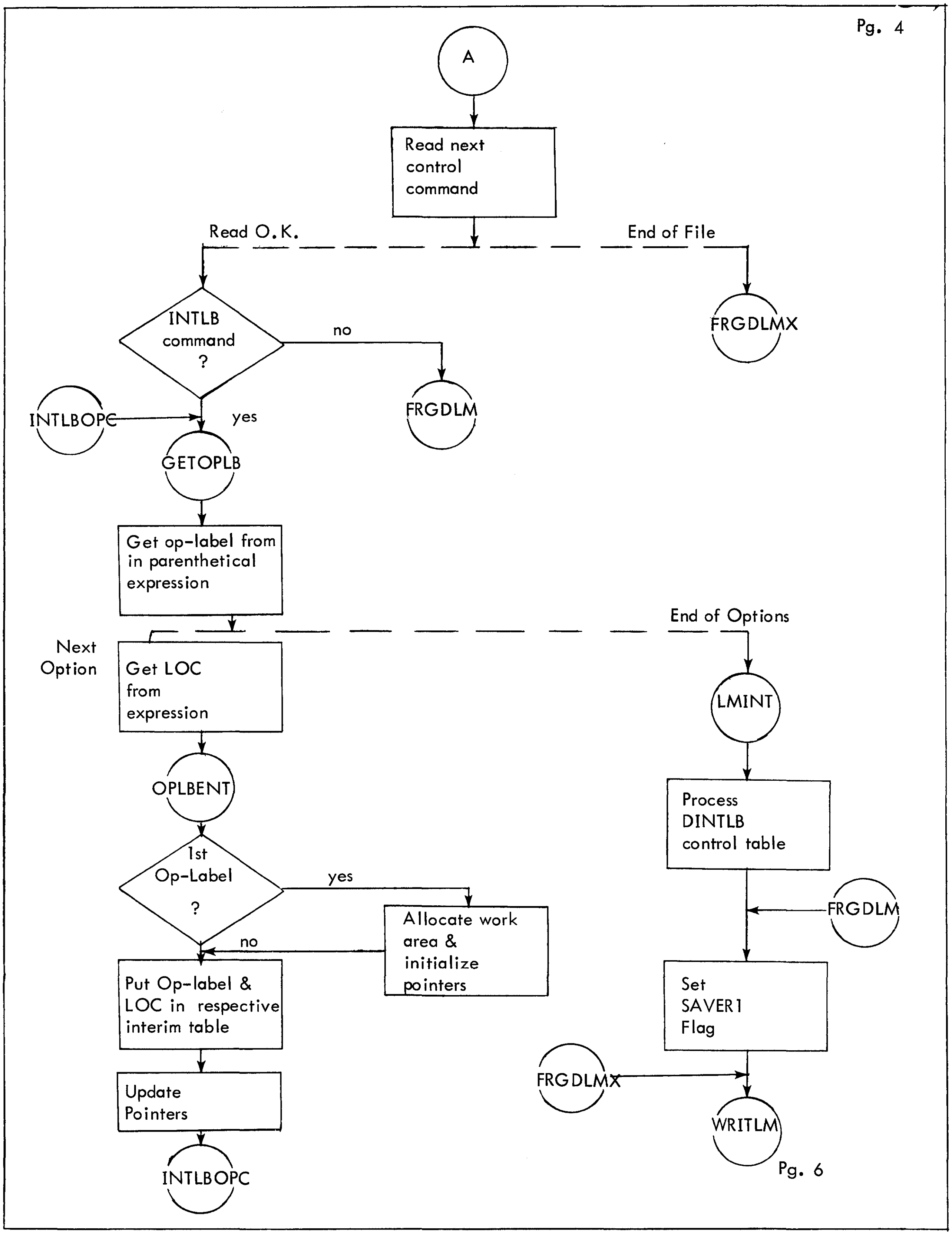


Figure 2-19. Flow Diagram of FRGD (Cont.)

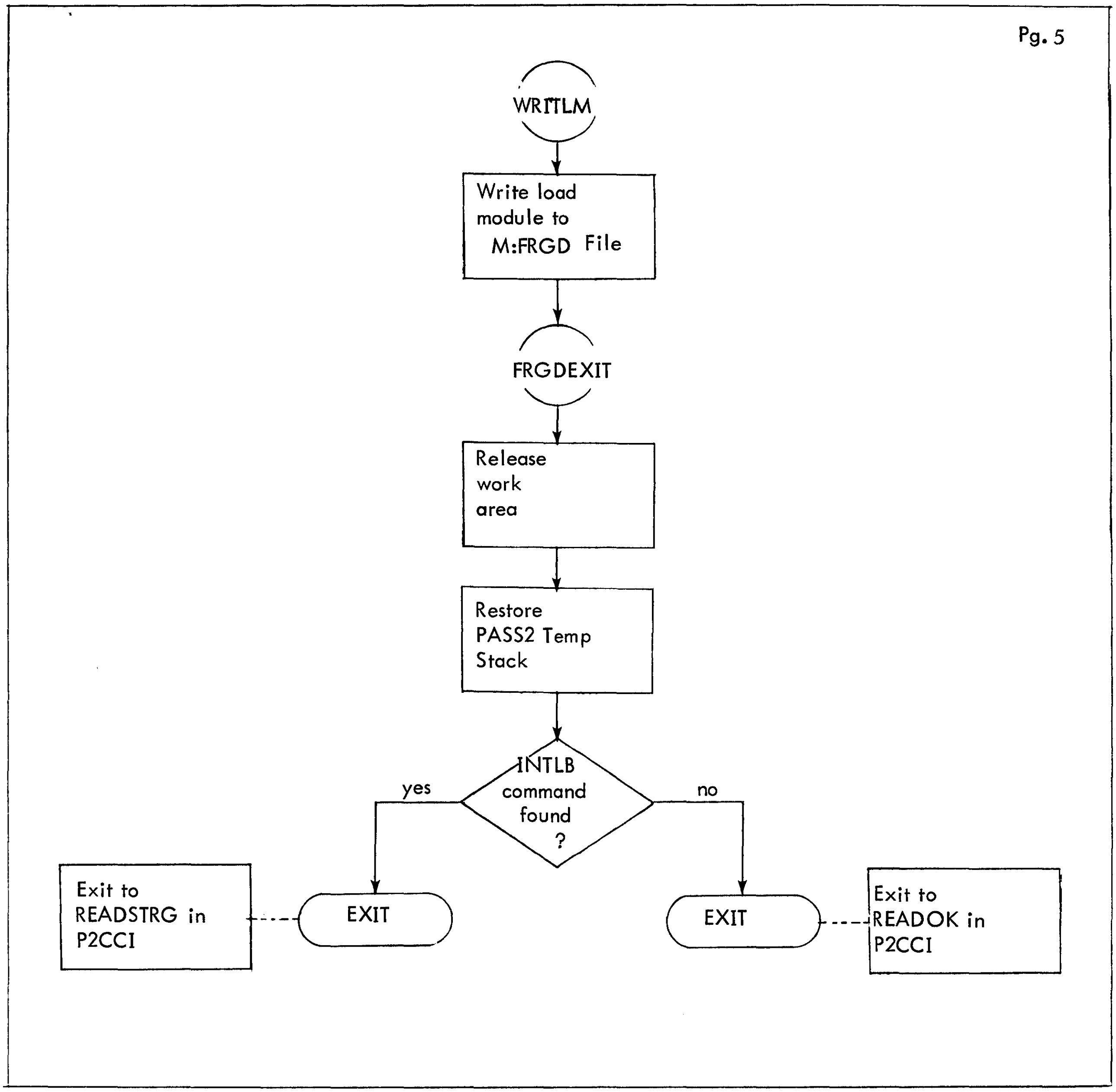


Figure 2-19. Flow Diagram of FRGD (Cont.)

2.12 XPART

2.12.1 Purpose

To process the :PART[ITION]command and generate the load module M:PART in which is defined the permissible number of partitions and the resources available for each for UTS systems only.

2.12.2 Usage

```
B        UTXPART
```

With:

R3 pointing to PASS2 stack data

R7 pointing to control card PLIST

R0 pointing to temp stack pointer

R3 and R7 are saved

Return is to READSTRG in P2CCI

2.12.3 Input

Control card (:PART) image

2.12.4 Output

Display of PASS2 control information to LL device

M:PART load module (Table 2-22)

Table 2-22 M:PART Load Module

Label	Entry Size (Wrds)	Length	Contents or Value in terms of PART Keywords
LPART	Value	-	#PART defined
PL:LK	1	1	0
PL:CHG	1	1	0
PLD:ACT	2	LPART+1	0
PLB:USR	1/4	LPART+1	0
PLH:CUR	1/2	LPART+1	0
PLH:SID	1/2	LPART+1	0
PLH:TOL	1/2	LPART+1	0

PLH:FLG	1/2	LPART+1	Bit 0 of each entry for HOLD/SWAP
			if SWAP = 0 if HOLD = 1
			Bit 15 of each entry for LOCK/UNLOCK
			if LOCK=1 if UNLOCK = 0

PLH:TL	1/2	LPART+1	Lower TIME limit
PLH:TU	1/2	LPART+1	Upper TIME limit
PLH:QN	1/2	LPART+1	QUAN
PL:MIN	1	LPART+1	Lower limits Bits 0 - 7 = DP Bits 8 - 15 = 7T Bits 16 - 23 = 9T Bits 24 - 31 = CORE
PL:MAX	1	LPART+1	Upper limits Bits 0 - 7 = DP Bits 8 - 15 = 7T Bits 16 - 23 = 9T Bits 24 - 31 = CORE

2.12.5 Subroutines and Definitions

1. Subroutines

SYNTAX	used to convert control card to stack data blocks
COREALLOC	used to allocate dynamic data pages
MODGEN	used to generate DEFs
WRITELM	used to write output module

2. Definitions

DEVS,R3 A core location in P2CCI basic dynamic data area containing the total number of private disk packs, 7-Track tape drives and 9-Track tape drives defined via :DEVICE commands.

0 7	8 15	16 23	
#DP	#7T	#9T	

2.12.6 Data Base

The XPART module contains two tables for use by the SYNTAX routine.

1. KEYWORD TABLE

   Contains an entry for each valid keyword on the :PART card. This entry consists of the TEXT form of the keyword followed by a word containing an operation identifier and the displacement into the temp stack table.

2. SKELETON TEMP STACK TABLE

   Each entry in the table is a single word in length and contains the default value associated with the parameter.

   Note, the entries are order dependent because of the method subsequently used to error check the parameters.

2.12.7 Description

Upon entry the processor accesses the DEVS,R3 cell and stores the total DP,7T, and 9T on the system permissible in a table of maximum value. This method is required in order to allow zero to be a legal maximum value for peripheral parameters, (i.e., 7T,9T,DP).

The processor then BALS to the SYNTAX routine which converts the control command image into managable tempstack tables. All keywords encountered by SYNTAX prior to the first (PART,value) are incorporated into the first table to be used for all partitions as their default parameters. The first PART options causes SYNTAX to set the system defaults for all unchanged parameters in the first table and initiate a new table. Subsequent PARTs only cause new tables to be started. Upon return R5 contains the starting address of the generated tables which is saved for subsequent cleanup.

The processor then BALS to COREALLOC, specifying an unknown word size for the REF/DEF stack and data record. Upon return R7 contains the address of the MODIFY PLIST; SR1 contains the first address of the allocated data record area; SR2, the address of the REF/DEF stack area.

The highest partition number specified (providing that $\geq$ 3 or $\leq$ 16) is used as the total number of partitions desired (LPART) and all tables are LPART + 1 in length. If the highest PART specified is less than 3, then 3 is used as the default.

The processor then BALs to MODGEN and causes the value DEFs, location DEFs, and requisite tables to be generated for M:PART. Prior to storing values into the tables, the registers are saved and the processor checks each specified value that the maximum and minimum are within range, to determine the maximum is greater equal than the mimimum and core is the range 0 to 64, and that there is no conflict between the LOCK and UNLOCK, SWAP and HOLD paramters. If an error is detected, the processor puts a code in R3 and BALs on D2 to an internal subroutine (ERROR) that determines what error occured in which partition, and formulates an appropriate error message identifying the error, the partition and the value(s) used. The corrected value is stored in the skeleton table upon return from the routine. Note, if the mimimum is out of range, or if maximum is less than mimimum, zero is used for the minimum. If the maximum value is out of range, the SYSGEN defaults are used for both values.

The defaults are then stored in all tables to which are added any parameters that have been specified for a given partition.

The registers are restored and the processor BALs to WRITELM with the address of the file name to be created in D3. Upon return, the tempstack is cleaned up, releasing all the area obtained by SYNTAX for the tempstack tables and the processor exits to READSTRG in P2CCI.

2.12.8 XPART Messages

```
**ERROR-PARAMETER*XXXX*
  IN XXXXXXXX MAX < MIN--
  0 USED FOR MIN
**ERROR-PARAMETER*CORE*
  IN XXXXXXXX MIN > 64--
  0 USED FOR MIN
**ERROR-PARAMETER*CORE*
  IN XXXXXXXX MAX>64--
  64 USED FOR MAX
```

The specified parameter in the designated partition or default (INXXXXXXXX) is in error. The value used is indicated in the message. XPART continues.

*ERROR-PARAMETER*XXXX*
IN XXXXXXX MAX INVALID--
SYSGEN DEFAULTS USED
**ERROR-PARAMETER*XXXX*
IN XXXXXXX MIN INVALID--
0 USED FOR MIN

**ERROR-PARAMETER*XXXX*
IN XXXXXXX MAX & MIN INVALID --
SYSGEN DEFAULTS USED

**CONFLICT IN HOLD IN XXXXXXX -
PARTITION NOT HELD
**CONFLICT IN LOCK IN XXXXXXX -
PARTITION NOT LOCKED
**PART 0 NOT ALLOWED --
SPECIFIED RESOURCES IGNORED

The specified parameter (*XXXX*) in the designated partition or default (IN XXXXXXX) is in error. The value used is indicated in the message. XPART continues.

2.12.9 Flow Chart

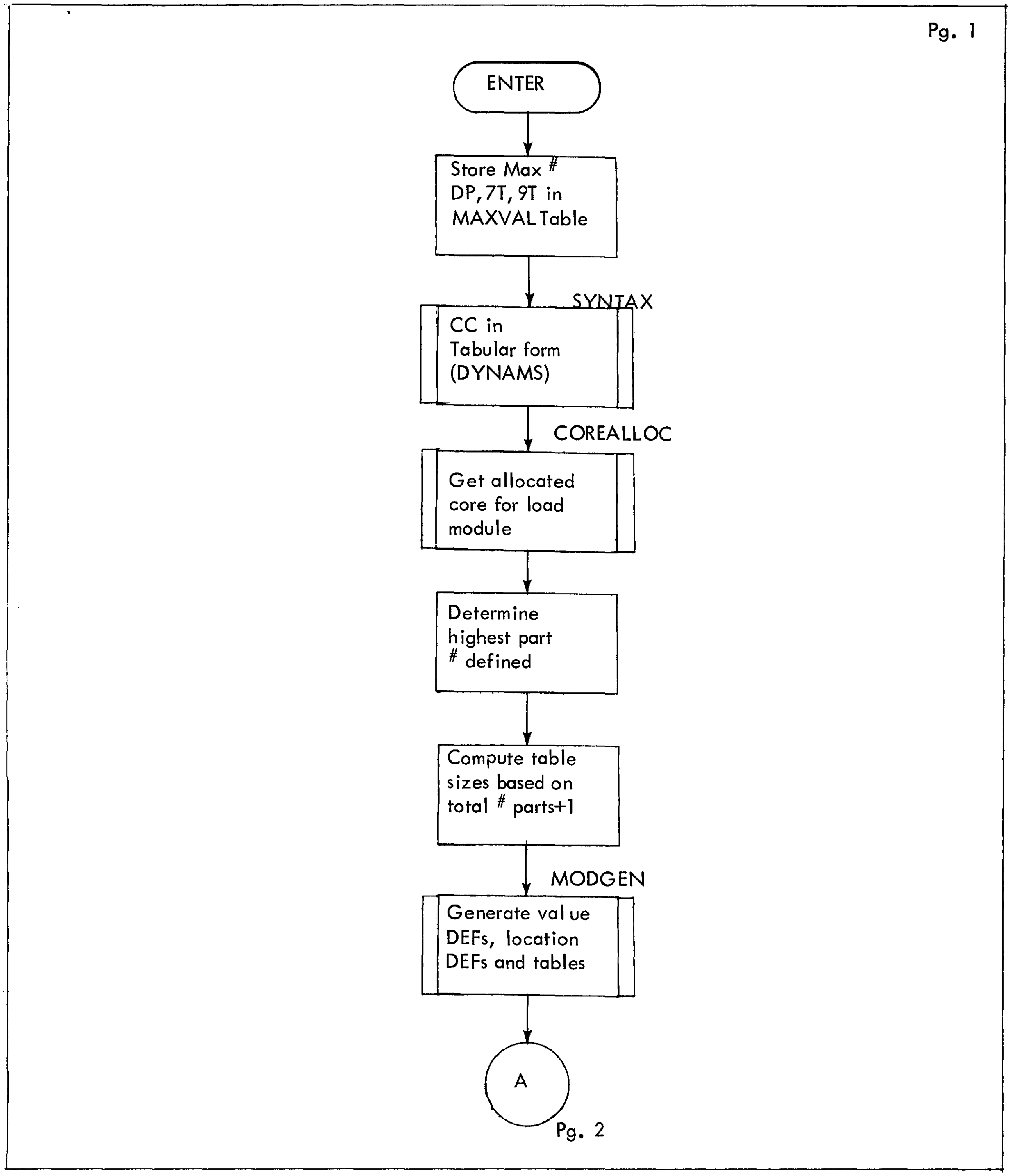


Figure 2-20. Flow Diagram of XPART

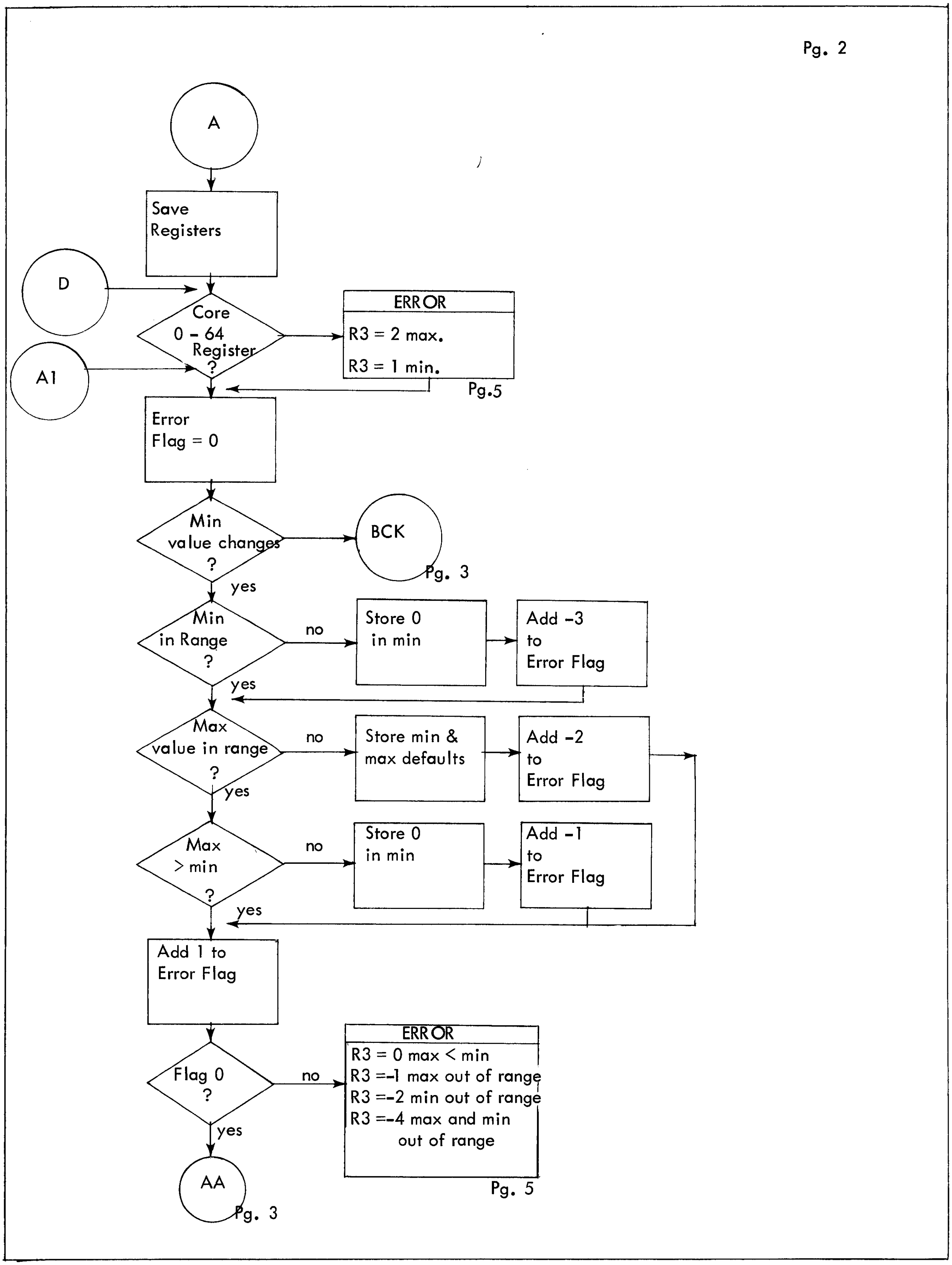


Figure 2-20. Flow Diagram of XPART (Cont.)

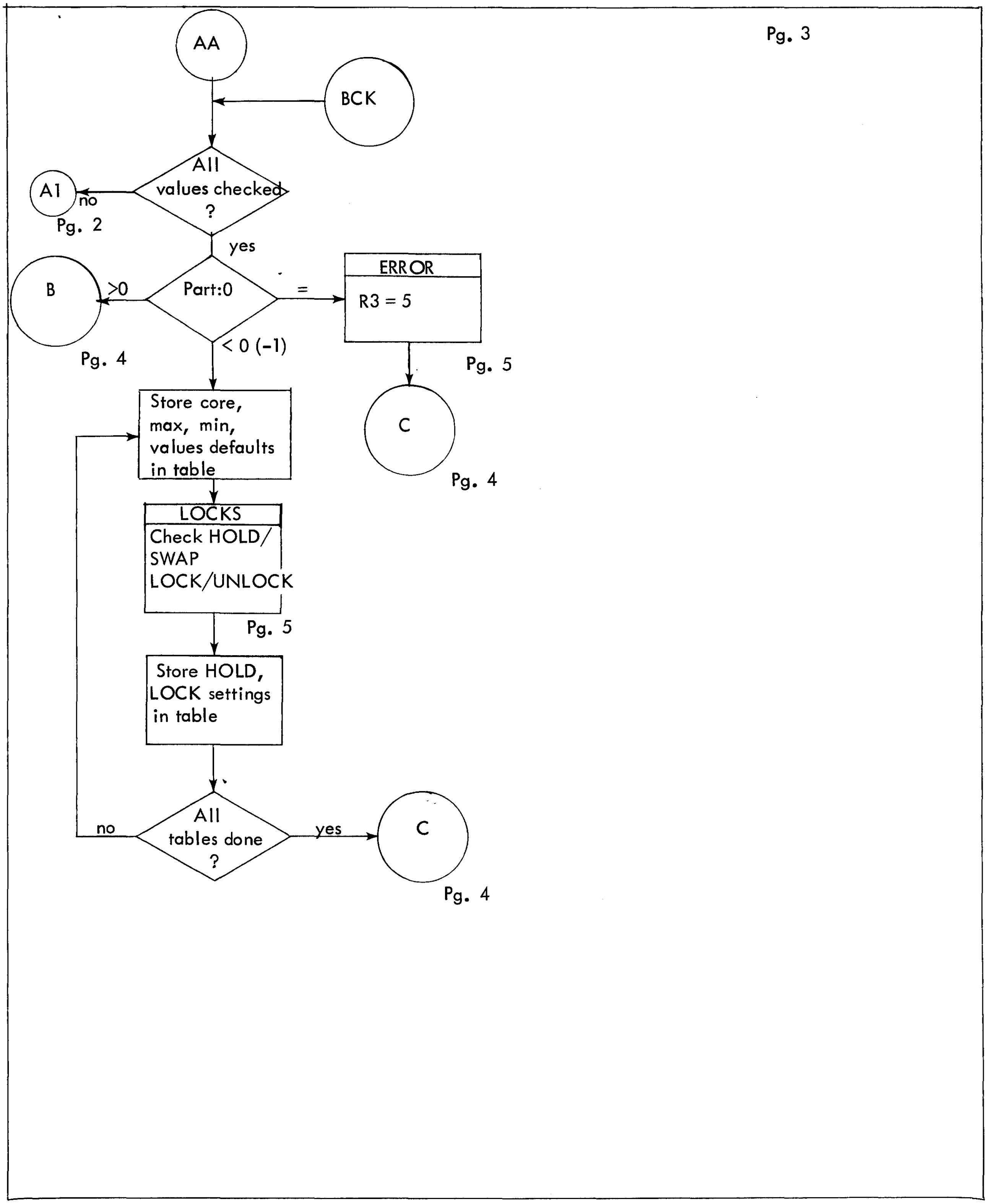


Figure 2-20. Flow Diagram of XPART (Cont.)

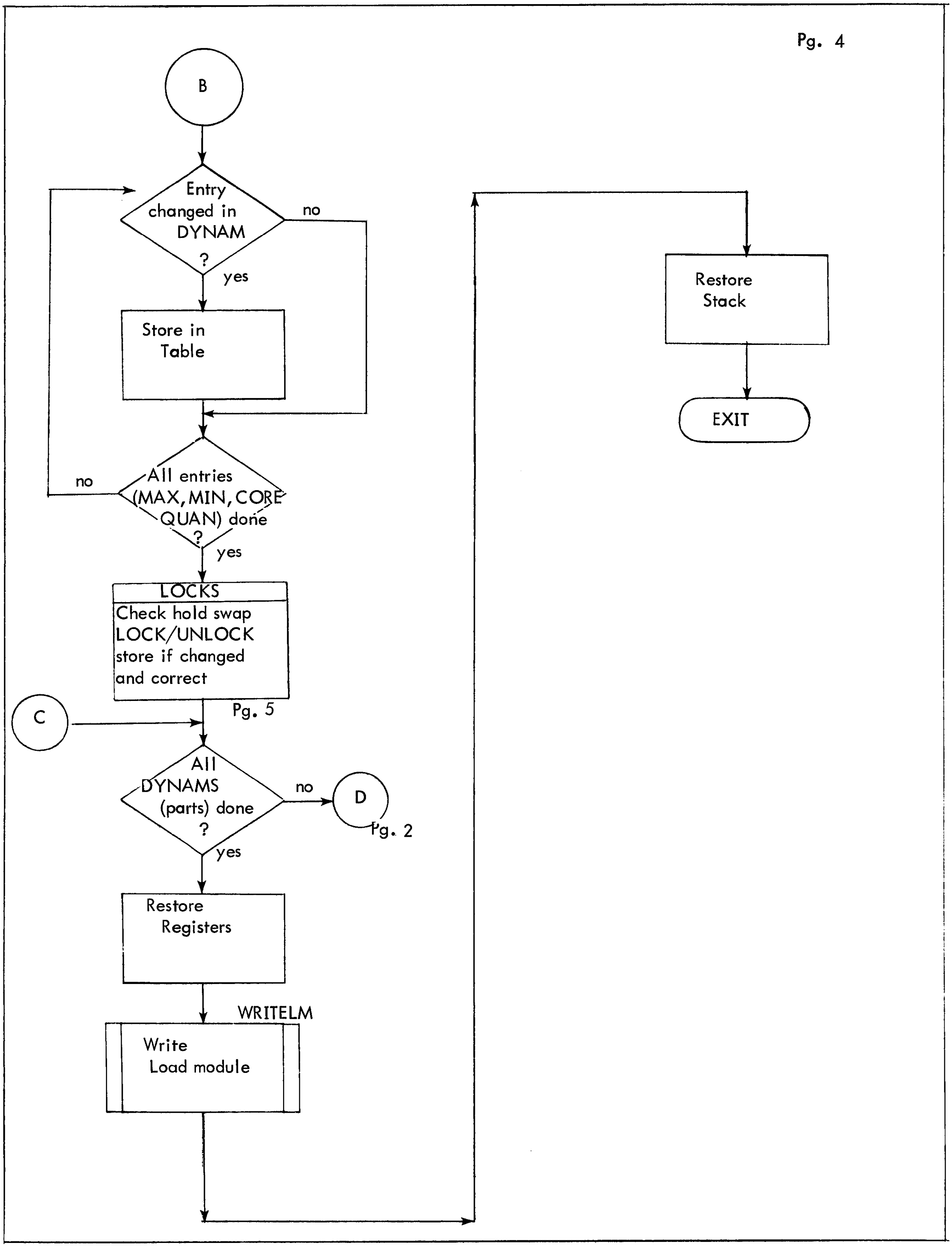


Figure 2-20. Flow Diagram of XPART (Cont.)

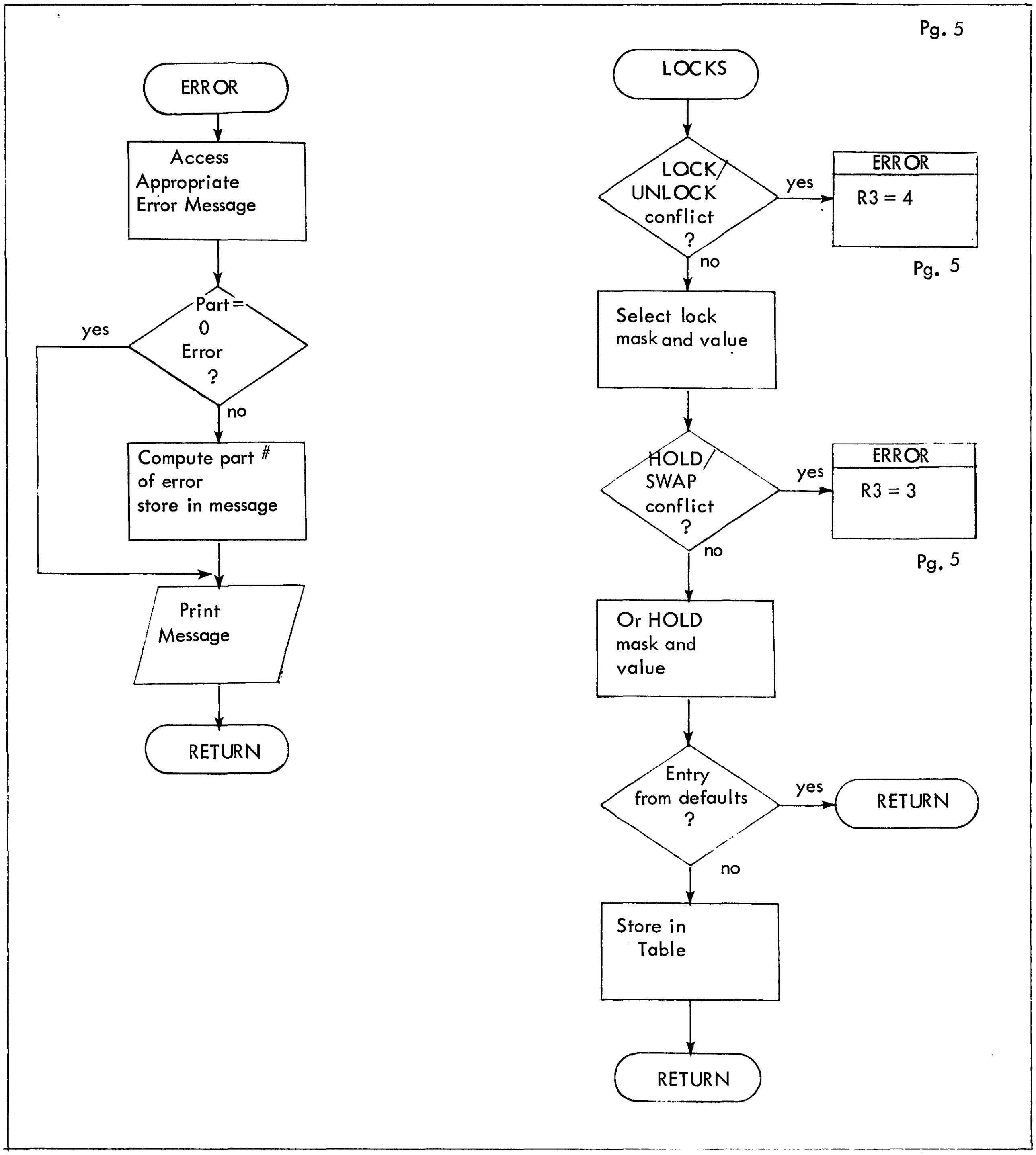


Figure 2-20. Flow Diagram of XPART (Cont.)

3.0 PASS3.

3.1 PURPOSE.

To provide the communication with the system LOADER necessary to load a specific Monitor, processor, or library. PASS3 provides for automatic biasing and, optionally, the releasing of random access device files.

3.2 CALLING SEQUENCE.

Monitor Control Command: !PASS3

3.3 INPUT.

PASS3 control commands from SI device.

:name

where

name is the name of a LOCCT (see LOCCT processor Chapter 5) to be used by the system LOADER in loading an element.

Comment commands from SI device.

Files containing LOCCTs from random access device.

M:MON load module, if target system is BPM/BTM, to determine background lower limit.

SPEC:HAND file when LOCCT is for M:MON. .

I/O handlers named in SPEC:HAND file plus BASHANDL file for basic I/O handler set when LOCCT is for M:MON.

3.4 OUTPUT.

Display of PASS3 control information to LL device.

HANDLERS file containing all necessary I/O handlers when LOCCT is for M:MON.

LOCCT to absolute read/write scratch area on system's random access device (BPM/BTM base system) or to core common storage (UTS base system).

ROOT Load Module generated if LOCCT is for M:MON and target system is BPM/BTM. See description (Chapter 3.7.2) for detailed discussion of contents of various tables. See Table 3.1 for contents of ROOT load module.

Table 3.1 ROOT LOAD MODULE CONTENTS

LABEL	ENTRY SIZE	LENGTH	CONTENTS OR VALUE
MAXLEV	Value	-	Max. levels in TREE structure
TBBASE	1	MAXLEV	0
COVLSEG	1	MAXLEV	segment number (starting with X'3F' and incremented by 1 for subsequent entries)
OSTACK	1	18	stack pointer doubleword in 1st 2 words followed by 16-word stack.
MAXSEG	Value	-	Number of segments in TREE structure (number of TREE entries - 1)
STTB	2	MAXSEG	0
XSEG*	Value	-	segment number
YSEG*	Value	-	segment number

* where the names in the TREE entries are substituted for X and the names of ROMS found in OLDSEGS tables are substituted for Y.

3.5 BASE REGISTERS.

Register 7 = address of control command PLISTs, I/O PLISTs, and data in temp stack.

Register 6 = address in temp stack of positions of "ROOT" load module.

3.6 SUBROUTINES USED.

NXACTCHR (obtain next character from control command).

NAMSCAN (used to scan a field which contains a name).

CHARSCAN (used to check a specific character, such as a terminator, for legal syntax).

HEXSCAN (used to scan a field which contains a hexadecimal value).

QUOTSCAN (used to scan a field which contains a key word, e.g., SAVE).

System Processor, LOADER (PASS3 does a M:LINK call to this processor to perform a load function. The LOADER must be in the :SYS account).

MODIFY (used when generating the "ROOT" load module).

3.7 DESCRIPTION.

3.7.1 Overview.

PASS3 is entered when the monitor control command !PASS3 is encountered. Upon entry, PASS3 obtains 4 pages of core as an initial work area. PASS3 then processes its own control command. A parameter is required (BPM or UTS) identifying the target system type. The presence of an optional parameter (MON or ALL) will cause PASS3 to abort should M:MON or any module be unable to be loaded successfully.

PASS3 then initializes its work area to zero and proceeds to read a :command and processes it according to the following options and syntax:

:name (option ,option ..)

The "name" is used to form a file-name which will identify a file containing a LOCCT (see Chapters 5 and 6). The "name" is syntactically checked and must be from one to ten alphanumeric characters of which one character must be alpha.

PASS3 proceeds to obtain the file containing the LOCCT referenced by "name". This file is made up of individual records of binary card image format. PASS3 reads each record, checks its sequence number, checks its check-sum, and reforms the LOCCT in one continuous record in the work area.

The remainder of PASS3's control command is then processed.

The options which may be encountered are as follows:

BIAS=value specifies that the load bias in the LOCCT identified by "name" is to be changed to the specified hexadecimal "value". The "value" cannot be greater than X'1FFFF'. The "value" is also converted to the next higher page boundary, if not already at a page boundary. If a bias offset is specified (see "BIAS=+offset"), the "offset" is added to the bias "value".

BIAS=+offset specifies that a hexadecimal "offset", converted to next higher page boundary, if not at one already, is to be added to the specified bias "value" (see "BIAS=value"). If no BIAS=value has been specified, the offset is added to the lower limit of the background area (BKGRDLL). This background lower limit is determined from the M:MON load module in the current account. However, if M:MON is not present, the offset value has no effect and the original LOCCT bias is unchanged. The logical interaction of these bias values is illustrated in the following table:

PASS3 processes these two options. However, they have no effect on a LOCCT's bias for a UTS SYSTEM. The original bias in the LOCCT is assumed to be correct.

Table 3-2. BIAS RESULTS

BIAS=value	BIAS=+offset	M:MON	Resulting BIAS
unspecified	unspecified	absent	LOCCT unchanged
unspecified	unspecified	present	BKGRDLL
unspecified	specified	absent	LOCCT unchanged
unspecified	specified	present	BKGRDLL+offset
specified	unspecified	absent	value
specified	unspecified	present	value
specified	specified	absent	value + offset
specified	specified	present	value + offset

DELETE specifies that when the system LOADER has completed the loading of the module (load module) "name", all element files comprising this module, as well as its LOCCT, are to be deleted. However, this does not include those files specifically named by the SAVE option (see SAVE option). The deleted files must be in the current account and must not be protected by a password.

SAVE (name [, name' . . .) specifies that the named elements ("name") are not to be deleted (see DELETE option). All element names in the LOCCT from the current account not SAVEd are deleted. The presence of this option automatically implies a DELETE option.

If, during the loading of a module, the system LOADER encounters an error, the DELETE/SAVE feature is ignored. That is, no element files are deleted, as the LOADER has not been successful in loading them.

If the SAVE option is encountered, PASS3 obtains enough core to build an internal table of "name$_i$" entries (Table 3-3). This table of names is used when PASS3 performs the DELETE for all element files named in the ROM Table entries in the LOCCT. This option is necessary when certain element files are referenced by several LOCCTs.

Table 3-3. Table of SAVE Names

					LOWCORE
				LW1	
				NAB	
INIT	#C	C_1	C_2	C_3	
	- -	C_N	#C	C_1	
	C_2	. . .	. . .	C_N	
BANAB ↓				↓	
BALW ↓				↓	

where

LW1 = byte address of the last available word plus one in the Table work area, (i.e., BA (BALW+1))

NAB = byte address of next available byte in table's work area (i.e., BA (BANAB)), NAB initially contains the byte address of INIT.

#C = number of characters in name (TEXTC format)

C_1, C_2, . . . , C_N = characters in name

When the control command has been processed, PASS3 checks the load module name in the LOCCT, which exists in the work area, to see if it is "M:MON" (i.e., this LOCCT is for a Monitor). If it is, and the target system is BPM/BTM, PASS3 generates a "ROOT" load module. This module describes the Monitor's overlay structure as identified by the TREE entries in the current LOCCT.

The "ROOT" module is ultimately loaded with the Monitor and controls the Monitor's run-time overlay structure. If the current LOCCT's load module name is "M:MON", and the target system is UTS, the "ROOT" module is not generated.

If the current LOCCT's load module name is not "M:MON", PASS3 determines if a previous Monitor has been loaded, and if so, what is its background lower limit. This is accomplished by obtaining the keyed record TREE from the load module M:MON in the current account. If the M:MON module does not exist, there is no background lower limit. However, if the TREE structure is available, the end of the longest Monitor overlay path is determined and is used as the background lower limit. Note RCUR2, RCURSYMB, TSDBTA and CLS are ignored in determining the longest overlay path.

If the target system is UTS, all further bias analysis ceases, and PASS3 assumes that the bias in the LOCCT is correct. However, if the target system is BPM/BTM, the resulting bias in the LOCCT is calculated according to Table 3-1.

PASS3 then determines where to save the LOCCT for the System LOADER's use. If the base system is BPM/BTM, the LOCCT is written to the absolute read/write scratch area on the system's random access device. But, if the base system is UTS, the LOCCT is saved in common core storage.

When the LOCCT's load module name is "M:MON", PASS3 proceeds to generate the "HANDLERS" file. This file contains all of the necessary peripheral I/O handlers as defined by the user on PASS2 ":DEVICE" control commands. The required handler names are found in the keyed file "SPEC:HAND". The format of this table of names is described in Chapter 6. PASS3 obtains the contents of the "BASHANDL" file and writes it to the "HANDLERS" file. The "BASHANDL" file, by default, contains the handlers for: typewriter (handler name = KBTIO), card reader (handler name = CRDIN), line printer (handler names = PRTOUT and PRTOUTL), RAD (handler name = DISCIO).

The handler names are then obtained from the "SPEC:HAND" file. Any name which is already a part of "BASHANDL" is ignored. If a name is not a part of "BASHANDL", it is used as the name of a handler file which must be merged into the new "HANDLERS" file. If a specific name is encountered a second or subsequent time in the "SPEC:HAND" file, it is ignored. Any name encountered for which there is no handler file, causes PASS3 to discontinue the processing of this control command and, therefore, to continue to its next command.

When all LOCCT and "HANDLERS" file processing is completed, PASS3 performs a Monitor M:LINK call to the system LOADER which is found in the :SYS account, and must be a load module file.

When the System LOADER has completed its task, it performs a Monitor M:LDTRC call requesting a return to the calling processor, namely, PASS3. The LOADER returns a flag which indicates whether or not the load function was successful. If not successful (i.e., flag set), PASS3 proceeds to read and process the next control command, therefore, ignoring any DELETE/SAVE option requested. If the load was successful (i.e., flag reset), and no DELETE/SAVE option was specified, PASS3 proceeds by reading and processing the next control command.

However, if the load was successful, and the DELETE/SAVE option was requested, PASS3 attempts to release each file named in the LOCCT ROM entries, and the actual LOCCT used in loading the module. Each ROM entry is checked to see if its account field corresponds to the running account. If not, the entry is ignored. Each entry is then checked to determine if it was listed as a SAVE file. If it was, the named file is saved. When PASS3 has completed the delete phase, it proceeds to read and process the next control command.

3.7.2 Generation of ROOT Load Module.

When the LOCCT is for M:MON, after obtaining background lower limit for a BPM/BTM target system, the GENROOT routine is entered to generate the "ROOT" load module.

First, the dynamic data is moved into the stack and maximum core obtained and initialized for the load module generation.

The TREE table in the LOCCT is searched to determine the number of levels in the tree structure. The number of levels starts at 1 and is incremented by 1 for each non-zero OVERLAY LINK encountered in the TREE. The number of levels cannot exceed 5.

The value DEF MAXLEV is generated followed by the table TBBASE, a word table of MAXLEV length, each entry containing zero. COVLSEG is then generated, a word table of MAXLEV length. PASS3 stores X'3F' in the first word and increments this by 1 for each subsequent entry. OSTACK is then generated. This is an 18 word table, containing a stack pointer doubleword in the first two words, followed by a 16-word stack.

The number of segments in the tree structure (i.e., the number of TREE entrys in the LOCCT minus one) is used for the value DEF MAXSEG and to build the table S1TB. PASS3 allocates a doubleword entry initialized to zero for each segment.

Each TREE table entry is processed, starting with the root segment. The segment number designation starts with the value X'3F'. As each successive TREE entry is processed, the segment number is incremented by one, the least significant six bits being saved. That is, the segment number which follows X'3F' would be X'00'. The TREE entries are processed serially, and not according to overlay structure.

As an entry is processed, a segment name is formed by appending the characters "SEG" to the name in the TREE entry. That is, if a TREE entry's name is "A", the result will be "ASEG". This segment name is then used to create an external definition as follows:

```
    nameSEG     EQU     segment#                    < an absolute value>
```

where

"name" is the name from the TREE entry (e.g., A).

segment# is the current segment number value (e.g., X'3F').

After generating this def, the first ROM POINTER (in the current TREE entry) is used to obtain the Relocatable Object Module (ROM) names from the ROM entries in the LOCCT.

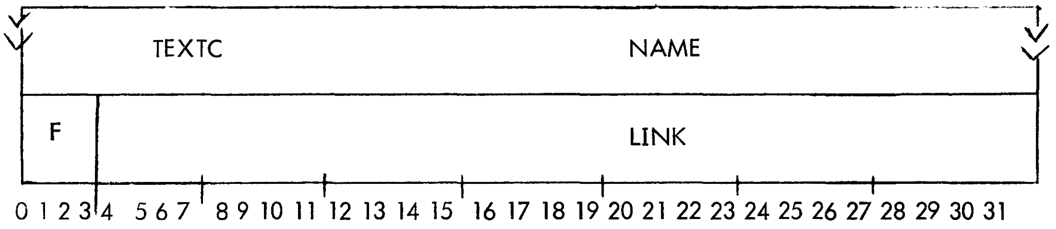
The first ROM name is ignored, as it is the same as the name in the current TREE table entry. However, each subsequent name in the ROM table, up to and including the name whose FLAG (see ROM table description in Chapter 6) is X'00', will be processed. The segment number value (e.g., X'3F') remains the same throughout the processing of these ROM names. Each ROM name identifies an element file needed to complete the segment identified in the current TREE entry. As a ROM name is processed, it is used to generate an external definition only if it is found in the table of special segment names (Table 3-4). Otherwise, the ROM name is ignored. If the ROM name is found, a segment name is formed by appending the characters "SEG" to the ROM name. That is, if a ROM name is "B", the result will be "BSEG". The external definition would then be:

```
    BSEG     EQU     segment#                        < an absolute value >
```

When all TREE table entries have been processed, the generated load module will be written to a file called "ROOT".

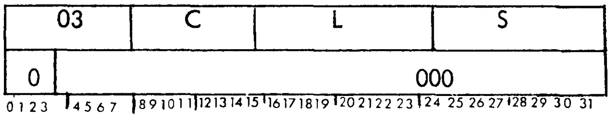
The following table controls the generation of external definitions using the ROM names. The format of a table entry is as follows:

TABLE 3-4. PASS3 "OLDSEGS" TABLE FORMAT



Each entry contains a name (TEXTC form), for which a segment number must be assigned. When a ROM name from a ROM entry (LOCCT) is found in this table, the characters "SEG" are appended to this TEXTC name, and an external definition is generated. Note that the linked entries each contain a duplicate entry for the very first name. The reason is that the first entry of a chain of linked entries is ignored, yet, it must be defined.

Example 3-1.



If ROM name in ROM entry is "CLS", and this name is found in the table, then the following def is generated:

```
CLSSEG    EQU    segment#
```

The "segment#" will be the same as that used when defining the segment number for the name in the TREE table entry.

LINK ≠ 0 is the address of another table entry (TEXTC name) which is to be defined.

If F = 8, then the current entry's name is defined, and the LINK address is used to obtain the next entry's name which is to be defined. This continues until LINK = 0.

If F = 0, then the current entry's name is ignored and the LINK address is used to obtain an entry's name which is to be defined. This condition continues for only one level, and not until LINK = 0.

LINK = 0 the current table entry is not linked to other entrys.

Example: If ROM name in ROM entry is "RDF", and this name is found in this table, then, the following defs are generated when F = 8, and LINK ≠ 0 or LINK = 0:

Address = 100	03	R	D	F	RDFSEG EQU segment#
	8			102	RLBTSEB EQU segment#
102	04	R	L	B	TRUNCSEG EQU segment#
	T				
	8			105	
105	05	T	R	U	
	N	C			
	8			000	

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

If the ROM name in ROM entry is "M:16", and this name is found in this table, then the following def is generated when F = 0, and LINK ≠ 0:

Address = 200	02	I	S		

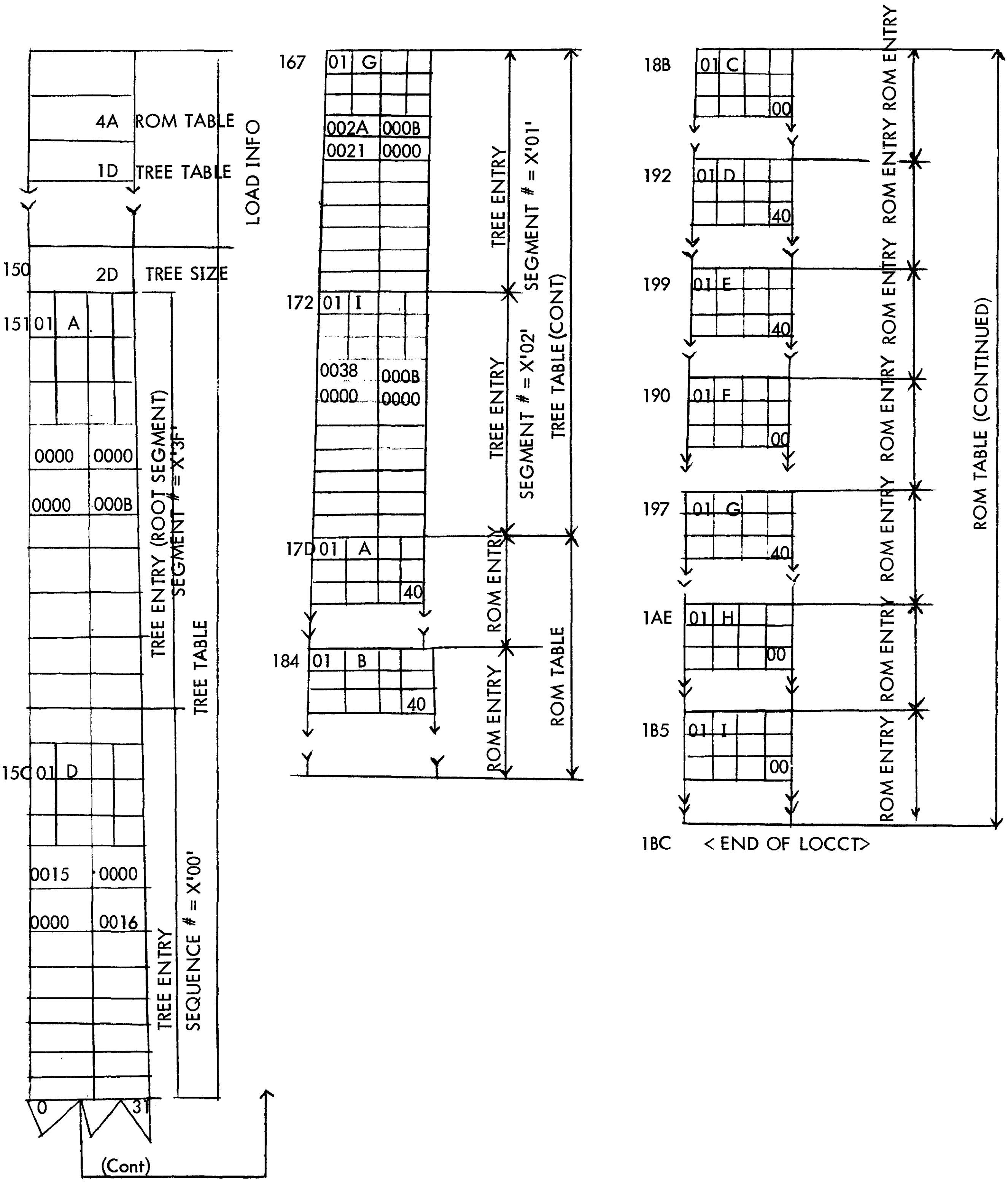
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

Address = 300	04	M	:	1	ISSEG EQU segment#
	6				
	0			200	

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

The following example shows pictorially what has just been described (to conserve space, the element file names used are not those specified in PASS3's "OLDSEGS" table, although the philosophy is the same). The LOCCT would look like:

EXAMPLE 3-2. DIAGRAM OF LOCCT.



Assume that PASS3's "OLDSEGS" Table contains the following entries:

=200	01	A		
	0	000		
202	01	B		
	8	204		
204	03	P	O	S
	8	206		
206	04	A	R	D
	L			
	8	000		
209	01	D		
	0	300		
20B	03	R	D	F
	8	20D		
20D	04	R	B	L
	T			
	8	210		
210	05	T	R	U
	N	C		
	8	000		
213	01	G		
	0	000		
215	01	I		
	0	302		
300	04	M	L	I
	N	K		
302	02	O	S	

The resultant external definitions are:

From TREE entry for "A"

```
ASEG   EQU X'3F'
```

From ROM entry for "B"

```
BSEG      EQU    X'3F'
POSSEG EQU      X'3F'
ARDLSEG  EQU  X'3F'
```

From TREE entry for "D"

```
DSEG      EQU     X'00'
```

From TREE entry for "G"

```
GSEG      EQU     X'01'
```

From TREE entry for "I"

```
ISEG      EQU     X'02'
```

From ROM entry for " I"

```
OSSEG   EQU   X'02'
```

3.8 PASS3 MESSAGES.

The error or information messages which may appear during a PASS3 consists of the following:

####PASS3--IN-CONTROL####

The PASS3 processor has been entered. For information only.

####PASS3--COMPLETED####

PASS3 has completed its function and exits to the Monitor. For information only.

**I/O ERR/ABN ON M:SI=xxxx

An error/abnormal code (xxxx), has been encountered when reading the SI device. If there is a background lower limit, it is displayed. PASS3 displays completed message and exits to the Monitor.

**OPEN M:EI ERR/ABN=xxxx(LOCCT)

An error/abnormal code (xxxx), has been encountered when attempting to obtain a referenced LOCCT file. PASS3 continues to next control command.

**WRITE ABS ERR/ABN=xxxx(LOCCT)

An error/abnormal code (xxxx), has been encountered when attempting to write the LOCCT to the absolute read/write scratch area on the random access device (BPM/BTM base system only). PASS3 continues to the next control command.

**READ M:EI ERR/ABN=xxxx(LOCCT)

An error/abnormal code (xxxx), has been encountered when attempting to read a LOCCT file. PASS3 continues to next control command.

[†]**OPEN M:MON ERR/ABN=xxxx

An error/abnormal code (xxxx), was encountered when attempting to open the M:MON load module file, and the error/abnormal was other than file does not exist (code=03). PASS3 displays background lower limit, if available, displays completed message, and then exits to Monitor.

[†]**READ M:MON ERR/ABN=xxxx

An error/abnormal code (xxxx), was encountered when attempting to read the keyed file TREE within the M:MON load module file. PASS3 displays background lower limit, if available, displays completed message, and then exits to Monitor.

**CANNOT OPEN/RELEASE

An attempt is made to release (through DELETE/SAVE option) an element file, but the file does not exist, or is from an account other than the running account or the file has a password. If the file does not exist, it may be because it is referenced more than once and the first reference has already released the item. This message is followed by the name of the element file. PASS3 continues.

**CC ERROR, NO ':' IN COLUMN-1

PASS3 control commands require a ":" in character position one. PASS3 continues to next control command.

**CC ID INVALID

The name identifying a LOCCT is not a valid alphanumeric character string. PASS3 continues to next control command.

**ID SIZE> 10 OR = 0 CHARACTERS

The name identifying a LOCCT does not exist or is longer than ten characters. PASS3 continues to next control command.

[†] Note: When one of these messages appears, PASS3 is in the process of determining a new background lower limit. That is, a "M:MON" load module does exist in current account, but cannot be accessed.

**DELIMITER NOT (), =OR SYNTAX BAD

One of the specified delimiters is expected but not found or the delimiter is recognized but is syntactically incorrect. PASS3 continues to next control command.

**KEYWORD NOT BIAS/DELETE/SAVE

The keyword is invalid. PASS3 continues to next control command.

**KEYWORD SAVE ALREADY USED

The SAVE option can be defined only one time. PASS3 continues to next command.

**NAME INVALID

A SAVE option "name" is not alphanumeric. PASS3 skips to next "name", if one exists, or to next field.

**BIAS NOT HEXADECIMAL VALUE OR
TOO LARGE VALUE

The BIAS "value" is not valid. It is either not hexadecimal or is greater than X'1FFFF'.
PASS3 continues to next control command.

**BIN. CARD INVALID TYPE, SEQ. #xxxx

A LOCCT file contains an image, sequence #xxxx, with an invalid ID type (i.e., not a X'3E' or X'1E'). PASS3 continues to next control command.

**BIN. CARD SEQUENCE ERR, SEQ. #xxxx

A LOCCT contains an image, with an invalid SEQ field. The sequence #xxxx is the number which the image should have been. PASS3 continues to next control command.

*CHECKSUM ERROR, SEQ. # xxxx

A LOCCT file contains an image, sequence # xxxx with a checksum error. PASS3 continues to next control command.

****SPECIFIED BIAS< BKGRDLL

The bias specified is less than the calculated background lower limit. This is a warning message, and appears when target system is BPM/BTM only.

****BIAS USED WILL BE xxxx

The bias used (xxxxx) is that which appears in the LOCCT. For information only.

****M:MON BKGRDLL IS xxxx

When PASS3 has completed its function, and the target system is BPM/BTM, and an "M:MON" load module has been built, its background lower limit, xxxxx, is displayed. PASS3 displays completed message and exits to Monitor.

*****LM 'ROOT' CANNOT BE GENERATED

PASS3 cannot generate the "ROOT" module (BPM/BTM target system). This results from PASS3 not having enough core for work area. PASS3 exits to the Monitor.

-----PASS3 ABORTED

This message is displayed in conjunction with other messages and implies an abort condition.

*****M:MON TREE STRUCTURE > 5 LEVELS

A Monitor overlay structure containing more than five levels (BPM/BTM target system) is illegal. A level implies the root segment as well as each overlay area with the tree structure. PASS3 displays abort and completed messages and exits to Monitor.

*****PASS3 TYPE UNKNOWN

The system control command, "!PASS3", contains a parameter which is neither "BPM" nor "UTS". PASS3 does an error exit to Monitor.

***OPEN/READ NNNNNNNNN
ERR/ABN = xxxx

An I/O error/abnormal code = xxxx, was encountered when attempting to form the "HANDLERS" file. The name "NNNNNNNNN" is initially "BASHANDL" and then is changed to the handler name which PASS3 is attempting to obtain and merge into the "HANDLERS" file. PASS3 continues to next control command.

****OPEN/READ SPEC:HAND FILE
ERR/ABN = xxxx

An I/O error/abnormal code = xxxx, was encountered when attempting to obtain the "SPEC:HAND file generated by PASS2. PASS3 continues to next control command.

***UNKNOWN TYPE - XXM USED

The type field of !PASS3 was not specified. System type under which doing SYSGEN is used, (XXM).

M:MON NOT SUCCESSFULLY LOADED

The MON or ALL option has been specified on the !PASS3 command and M:MON either cannot be loaded or has not been loaded. PASS3 aborts.

MODULE NOT SUCCESSFULLY LOADED

The ALL option was specified on the !PASS3 command and the Loader found errors loading a processor. PASS3 aborts.

OPTION NOT 'MON' OR 'ALL' -
NONE ASSUMED

The option field on the !PASS3 command is invalid. PASS3 continues.

3.9 PASS3 PROCESSOR FLAGS.

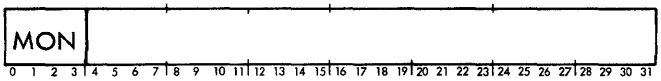
TYPEFLG

Designates the type of PASS3 specified:

=0 BPM
=2 UTS

LOC. X'2B' in Monitor

This location contains the information which identifies what type of base Monitor is in control (e.g., BPM/BTM or UTS). The format is:



where
MON = 4 BPM
= 5 BPM/BTM
= 6 UTS

MMONTYPE

A flag indicating the base Monitor type:

=0 non-UTS.
≠0 UTS.

M:MONFLG

A flag which indicates that the current LOCCT generates a M:MON (Monitor) load module. This flag and the MMONTYPE flag control whether or not a "ROOT" module is to be generated.

=0 LOCCT not for M:MON.
≠0 LOCCT is for M:MON.

MONFLG

A flag set the same as M:MONFLG, except it controls the generation of the HANDLERS file.

=0 No HANDLERS file generation.
≠0 generate HANDLERS file.

BKGDRLL

A cell containing the target system Monitor's background lower limit, if one exists (BPM/BTM target system only). If none exists, it is = 0.

BIAS

A cell containing the bias (BIAS = value) defined on PASS3's control command, and used only if BPM/BTM target system:

= -1 no bias defined.
> 0 the bias as defined (page boundary).

BIASADD

A cell containing the bias offset (BIAS = +value) defined on PASS3's control command, and used only if BPM/BTM target system:

=0 no bias offset defined.
>0 the bias offset as defined (page boundary).

LOADFLG

This flag indicates the System LOADER's success or failure in loading a module. The LOADER return to PASS3 with register 15 set = 0, if successful, or ≠ if failure. This value is saved in LOADFLG, and indicates whether or not the DELETE/SAVE options are to be honored:

= 0 yes, DELETE/SAVE options to be honored.
≠ 0 no, do not perform DELETE/SAVE.

DELETE

This flag indicates that the DELETE and/or SAVE option was encountered on the PASS3 control command. A SAVE option implies a DELETE:

= 0 no DELETE and no SAVE encountered.
≠ 0 a DELETE and/or SAVE was encountered.

SAVE

This flag is set when a SAVE option is encountered. The value in SAVE is the base address of the work area obtained for the SAVE "names":

= 0 no SAVE option.
> 0 the base address of the work area.

3.10 INTERNAL ROUTINES.

PASS3

main entry, initialize processor, and general controller.

PASS3NXT

get next control command.

PASS3LCT

obtain information from LOCCT file and re-form the records into continuous LOCCT.

PASS3CHK

control command has been processed, put various values into LOCCT table.

GENHAN

determine if HANDLERS file is to be generated for M:MON load module, and then do a M:LINK call to the System LOADER.

PASS3PAR

process all parameters on the current control command.

NXTNAM

get next name after SAVE option and enter it into the work area table.

GETPAGE

get more work area for SAVE option if needed.

PASS3DEL

process DELETE option.

PASS3BIS

process BIAS option.

SAVINCOM

get common storage and save LOCCT for System LOADER.
Register 15 = buffer address; Register 13 = LOCCT size in bytes.

GTMONTRE

obtain M:MON's TREE structure from M:MON load module in current account (if one exists) and determine the background lower limit.

GENHANDL

generate HANDLERS file from list of handler names found in SPEC:HAND file.

CPYHNDL

copy a handler to HANDLERS file. Register 9 = buffer address.

ROMDELET

delete element files which are named in the current LOCCT ROM entries. Register 13 = LOCCT size in bytes; Register 15 = base address of LOCCT.

CHKNAM

check LOCCT element file name against table of SAVE names to determine if delete is desired. If found, do not delete (condition code one = 1 if found, = 0 if not found). Register 1 = address of name to be checked; Register 4 = address of table of names.

READCC

read next control command. Register 12 = buffer address.

READCONT

continuation command requested.

LISTCONT

display control command from character subroutines.

LISTCC

display control command.

EOCCSCAN

search for end of control command.

GENROOT

generate "ROOT" load module from M:MON's TREE structure in M:MON load module in current account. Register 13 = address of LOCCT table minus one.

WRITM/WRITROOT

write "ROOT" load module to "ROOT" file. Register 1 = buffer address; Register 2 = buffer size in bytes; Register 3 = address of key (load module element's key).

APNDSEG

append "SEG" to a given segment name. INPUT: Register 4 = address of segment name OUTPUT: Registers 8-10 = segment name with appended "SEG".

EIA/EIE

desired LOCCT file not available.

RSPHER/RSPHAB/OSPHER/OSPHAB/HANDLER/ ONHER/ONHAB/ABSHER/OBSHAB

The HANDLERS file cannot be generated. Either a handler file, the BASHANDL file, or the SPEC:HAND file, is not available.

CONV

convert error/abnormal code to EBCDIC.

GENDEF

build DEF PLIST for MODIFY routine to add external definition. Register 1 = address of NAME$_1$. Register 2 = address of NAME$_2$. Register 2 = 0, no NAME$_2$. Register 3 = VALUE$_1$.

GENDICT

built DICT PLIST for MODIFY routine to change RELDICT.00. Register 1 = address of NAME$_1$. Register 2 = VALUE$_1$. Register 3 = resolution code for RELDICT.00.

Special error routines include:

RBSHER, RBSHAB, CE, CA, CEA0, CEA0X, EIAE, EOA, RE, RA, OTME, OTMA, OTMAE1, RTME, RTMA, DE, DA, NO:, IDERR, IDSIZE, DEL, COMEXIT, KEYWRD, KEYWRDX, DUPKEYWD, NAM, HEX, TYP, COMERR, SEQ, SUM, MODERR, BADM:MON, TYPERR, COMNER.

3.11. PASS3 PROCESSOR FLOWCHARTS

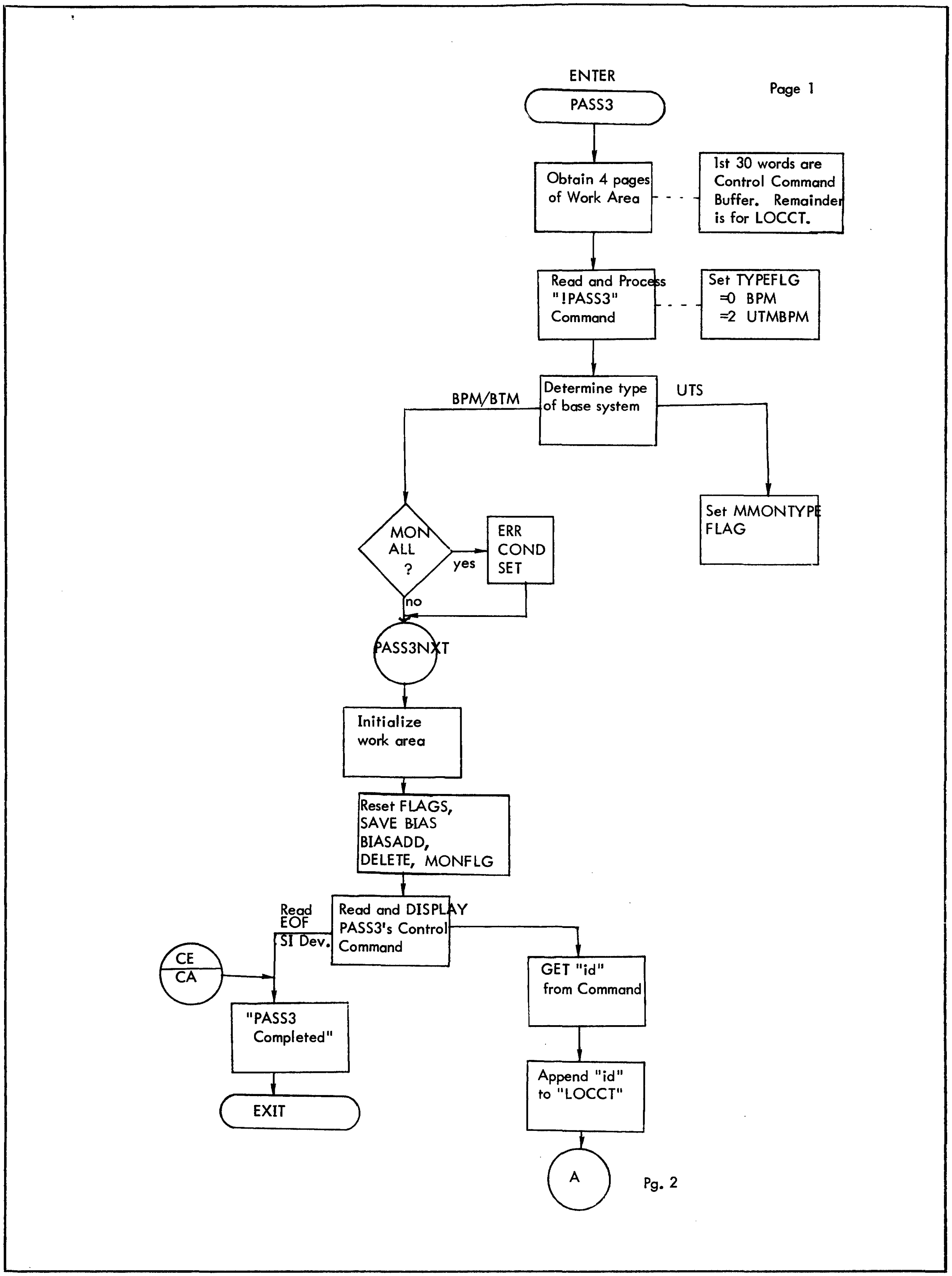

ENTER
PASS3
Page 1
Obtain 4 pages of Work Area
1st 30 words are Control Command Buffer. Remainder is for LOCCT.
Read and Process "!PASS3" Command
Set TYPEFLG =0 BPM =2 UTMBPM
Determine type of base system
BPM/BTM
UTS
Set MMONTYPE FLAG
MON ALL ?
yes
no
ERR COND SET
PASS3NXT
Initialize work area
Reset FLAGS, SAVE BIAS BIASADD, DELETE, MONFLG
Read and DISPLAY PASS3's Control Command
Read EOF SI Dev.
CE
CA
"PASS3 Completed"
EXIT
GET "id" from Command
Append "id" to "LOCCT"
A
Pg. 2

Figure 3-1. Flow Diagram of PASS3

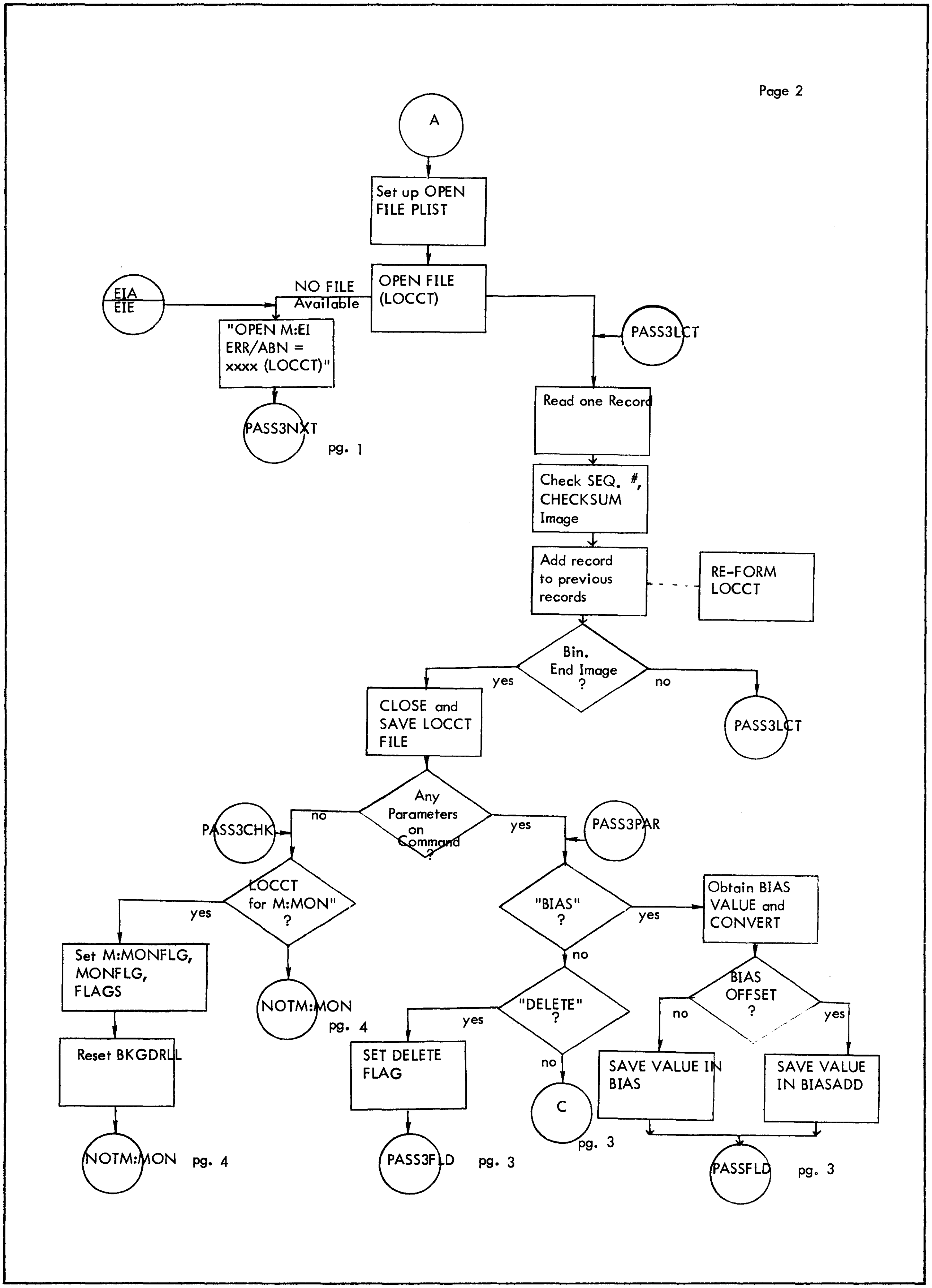


Figure 3-1. Flow Diagram of PASS3 (CONT)

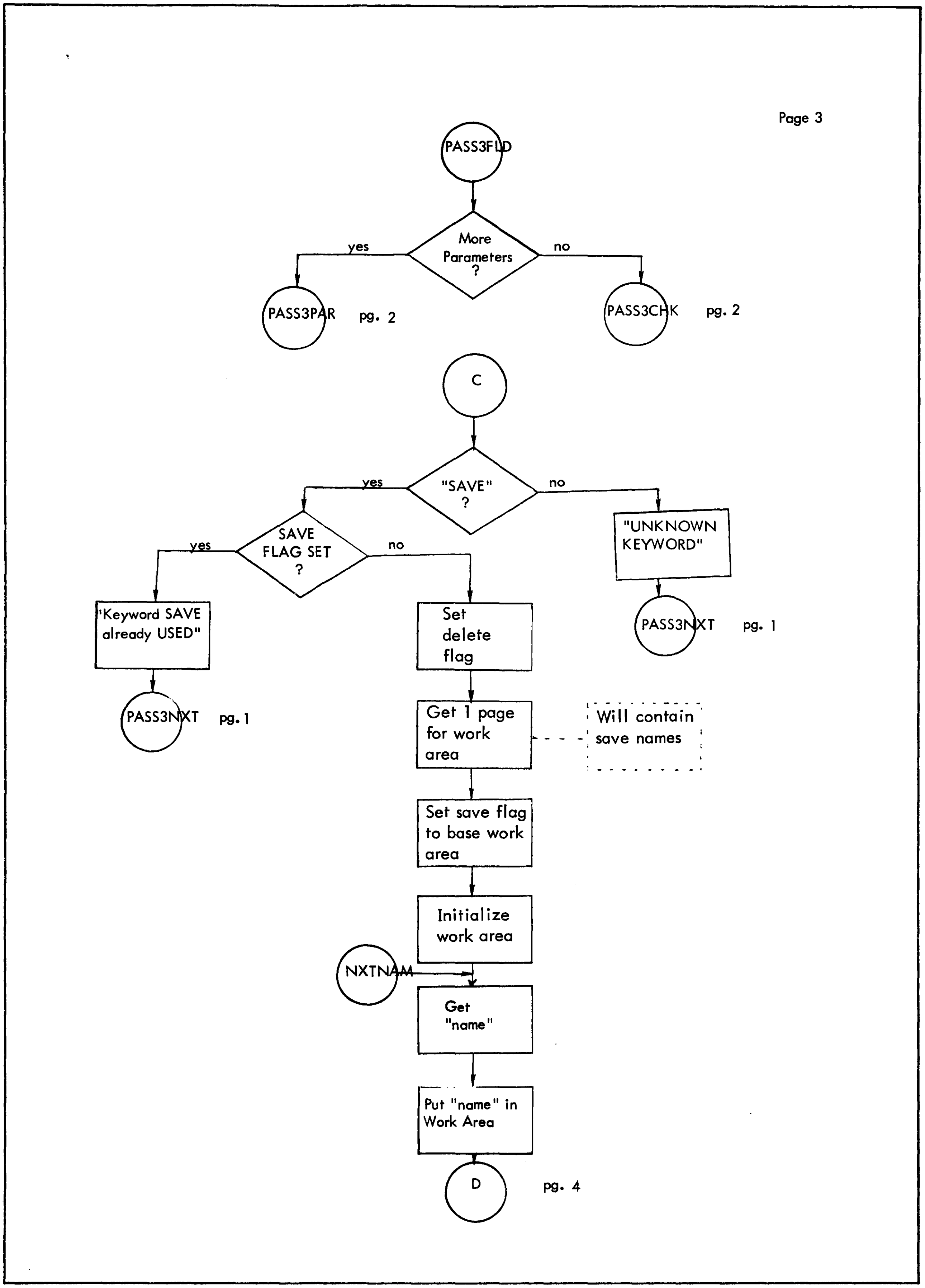


Figure 3-1. Flow Diagram of PASS3 (CONT)

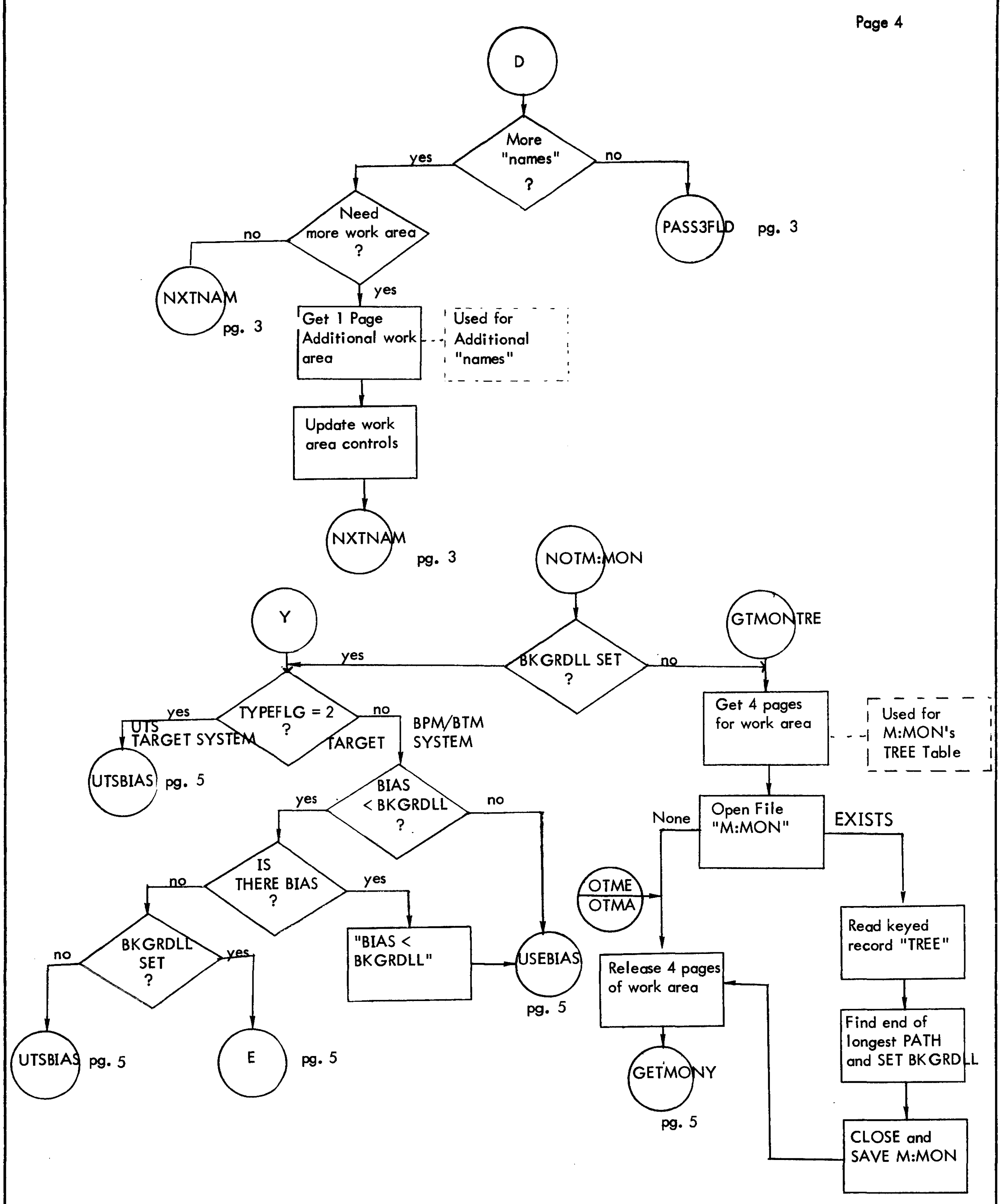


Figure 3-1. Flow Diagram of PASS3 (CONT)

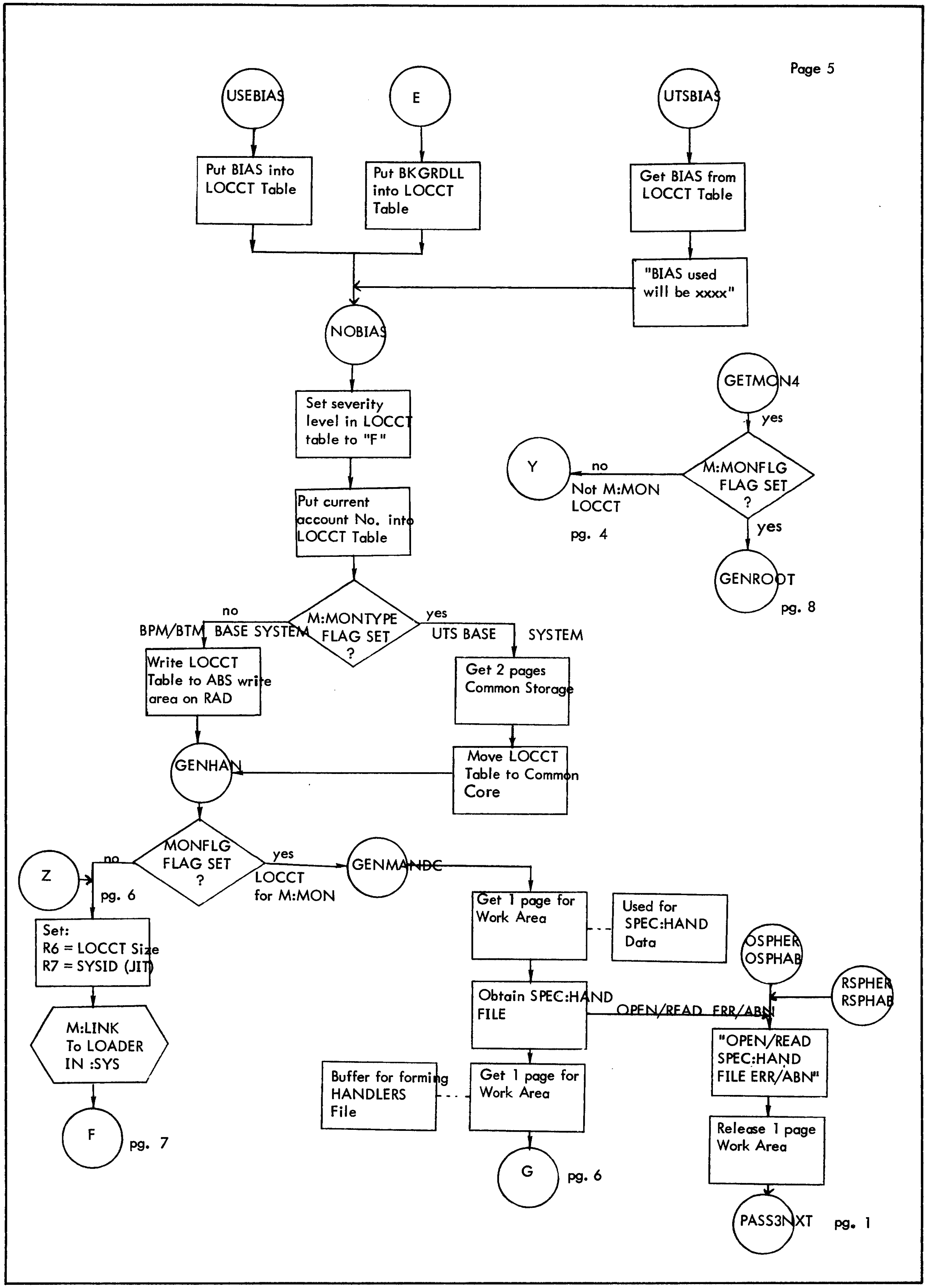


Figure 3-1. Flow Diagram of PASS3 (CONT)

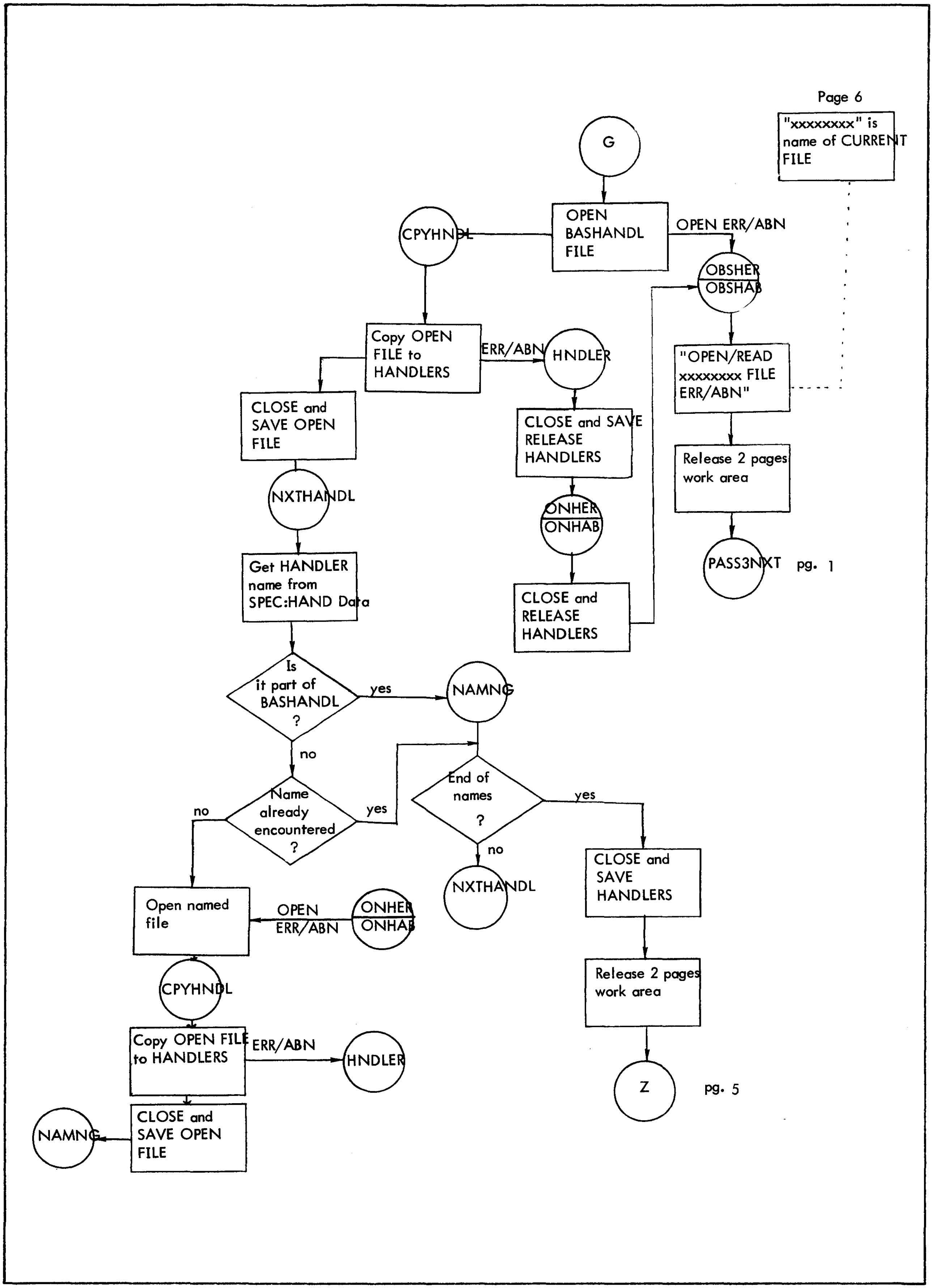


Figure 3-1. Flow Diagram of PASS3 (CONT)

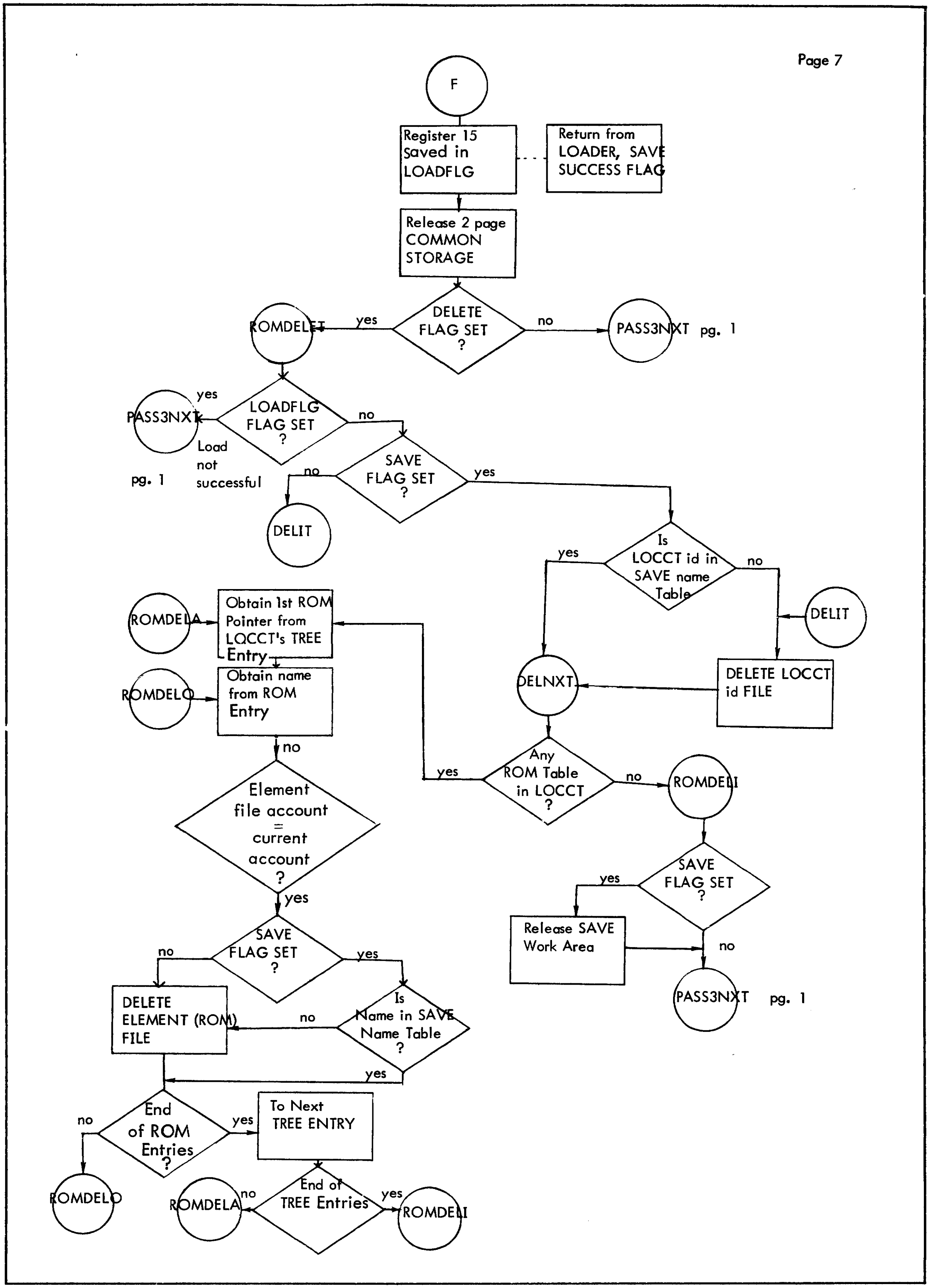


Figure 3-1. Flow Diagram of PASS3 (CONT)

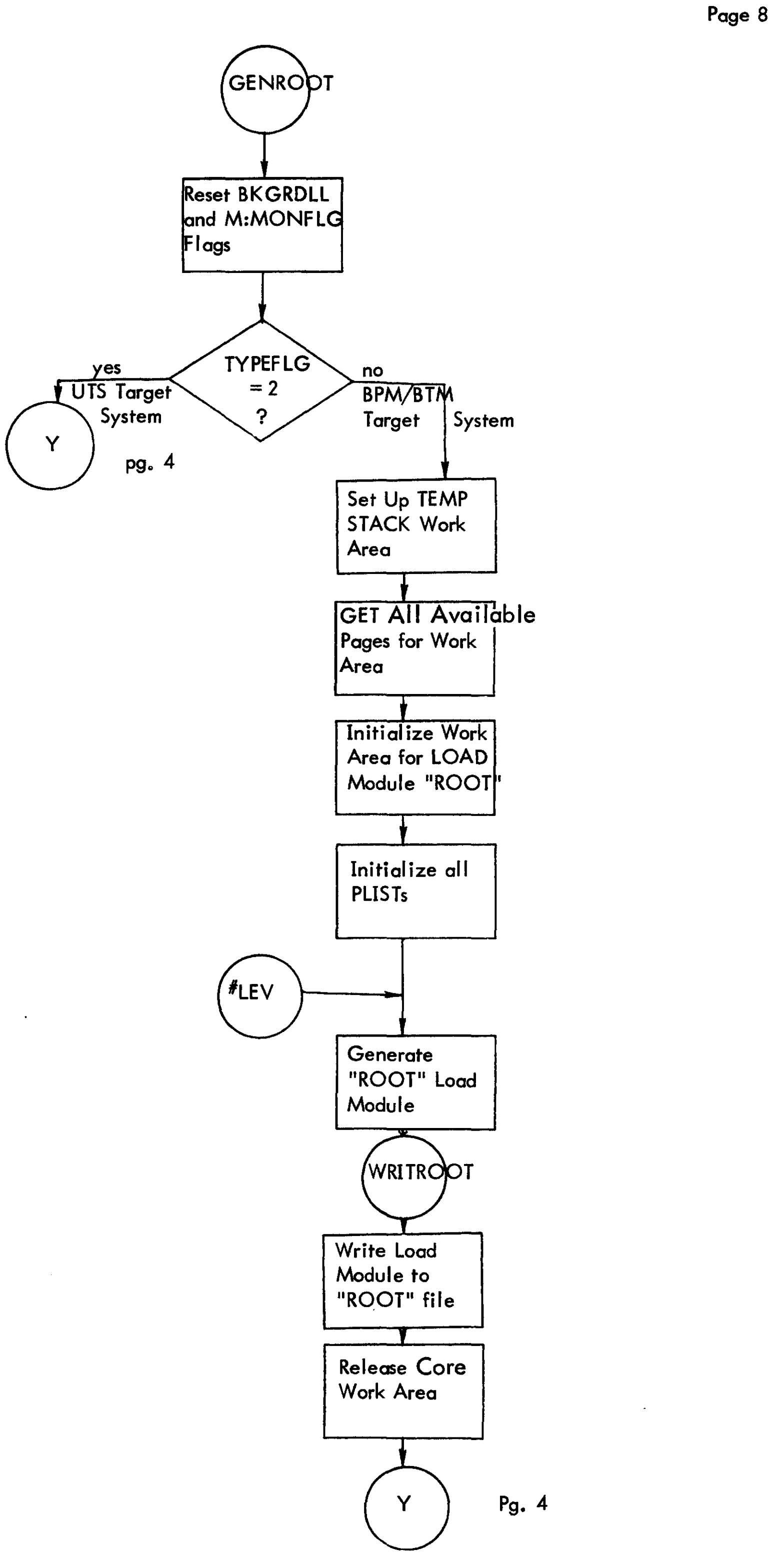

Page 8
GENROOT
Reset BKGRDLL and M:MONFLG Flags
TYPEFLG = 2 ?
yes
UTS Target System
Y
pg. 4
no
BPM/BTM Target System
Set Up TEMP STACK Work Area
GET All Available Pages for Work Area
Initialize Work Area for LOAD Module "ROOT"
Initialize all PLISTs
#LEV
Generate "ROOT" Load Module
WRITROOT
Write Load Module to "ROOT" file
Release Core Work Area
Y
Pg. 4

Figure 3-1. Flow Diagram of PASS3 (CONT)

4.0 DEF.

Note This Chapter discusses DEF for UTS (D00 release) and pre-H00 releases of BPM/BTM. DEF for BPM/BTM (H00 releases) is described in Appendix C.

4.1 PURPOSE.

To generate one or more target system tapes (PO tape) or BO tapes which maybe used as master BI tapes for subsequent SYSGENs.

4.2 CALLING SEQUENCE.

Monitor control command

!DEF . . . . . .

4.3 INPUT.

DEF control commands from the SI device.

!DEF (from C device)
:WRITE
:INCLUDE
:IGNORE
:DELETE
END
Files from random access device.
Comment commands

4.4 OUTPUT.

Display of DEF control information to LL device.

PO tape
BO tape

4.5 DATA BASE AND REGISTERS.

R7 = address in temp stack of control command PLISTS
R6 = address in temp stack of data and I/O PLISTS

IGSTRT/IGEND - Pointer to START/END of IGNORE table
INCLSTRT/INCLEND - Pointer to START/END of INCLUDE table
Open FPTs

OPNTMSQN - Open disc to file
OPNPO - Open PO or BO (via DCB whose address is in R5)
OPNTM - Open disc to next file
OPNSYN - Open tape for Synonymous file
OPNPOLST - Open tape for LASTLM file

POIGS Table - Automatic IGNOREs for PO tape
POINCLS Table - Automatic INCLUDEs for UTS PO tape
BOINCLS Table - Automatic INCLUDEs for UTS BO tape
BBOINCLS Table - Automatic INCLUDEs for BPM/BTM BO tape

4.6 SUBROUTINES.

BPMBT (write BPM/BTM system to unlabeled portion of BO/PO tape) (See Appendix A for description)
UTMBPMBT (write UTS system to unlabeled portion of BO/PO tape) (See Appendix B for description)
NAMSCAN (to scan any field containing a name)
CHARSCAN (to check a specific character for legitimate syntax)
CHSTSCAN (to obtain a character-string field)
NXACTCHR (to get next active character from input record)
HEXSCAN (to scan for a hexadecimal number)

DECSCAN (to scan for a decimal number)
QUOTSCAN (to compare a quote constant with a character string)
GETCHST (to obtain the next character string)

4.7 CONTROL COMMANDS.

Upon entry DEF requires a parameter on the !DEF command that identifies the SYSGEN system for which tapes are being created. This parameter may be either BPM, BTM or UTS. If none is specified then the currently running operating system is used to determine the system type. If the parameter is invalid DEF prints an indicator message and aborts. For UTS, an optional second parameter is a version number. There is no syntax analysis made on the field so any set of characters is accepted. However, only the first three characters are retained as a general practice the format of the character string is:

```
    L0D
```

where

```
    L = an alphabetic character
    0 = the digit zero
    D = a numeric digit (0-9)
```

The format then for the DEF control command is:

```
    !DEF ⎧UTS⎫[,L0D]
         ⎨BPM⎬
         ⎩BTM⎭
```

The type and composition of the tape(s) DEF creates is a function of the control commands read by DEF. If the !DEF is immediately followed by a monitor control command, one PO tape is created by default. The function of the END command is to cause DEF to exit since an EOF on reading M:SI causes one PO tape to be generated unless the last command processed was :WRITE. To write a BO tape and/or include, ignore or delete files for either tape type, :Commands are required. These commands have the following format:

```
    :INCLUDE   (name1, name2 . . .)
    IGNORE     (nameA, nameB . . .)
    :DELETE
    :WRITE  ⎧BO⎫ [,SN]
            ⎩PO⎭
```

All commands preceeding the :WRITE apply to that tape being created and may appear in any order. The :WRITE is required for BO tapes as PO is generated if the type parameter is null or illegal. The optional SN field permits a generalized assignment of PO/BO to (DEVICE, 9T) prior to calling DEF. The processor itself stores the particular SN into the DCB.

The type of files that may be specified or are affected by the other commands depend on the type of tape being generated. Table 4-1 summarizes this information.

Table 4-1. File Types

:COMMAND	BO	PO
:INCLUDE	Keyed Files	Consecutive Files
:IGNORE	Consecutive Jobs	Keyed Files
:DELETE	BOTH	BOTH

4.8 DESCRIPTION.

Upon entry DEF initializes its dynamic data area and processes the !DEF command. One page of core is obtained for storing file names into the tables pointed to by IGSTRT and INCLSTRT.

DEF then reads it :Commands and branches to the appropriate routine to process them. For :INCLUDE and :IGNORE, this involves syntax checking the names (≤ 15 characters); determining if room exists for the entry (if not, obtaining an additional page of core); storing the name in the appropriate table; and exiting to read another command. For :DELETE, a flag (DELETEF) is set before exiting to read another command. When an abnormal return (EOF) is made from reading SI for commands, ENDFLG is set and if the WRTFLG is non-zero indicating the last command was :WRITE then the routine is entered to clear up the stack and exit. If WRTFLG is zero, then the routine to write a PO tape by default is entered.

For the :WRITE command, entry is made to the initial routine that determines which type of tape is being generated. From here, a branch is made to either the PO or BO routines.

For PO tapes, after processing the optional SN field, the names of files that are to be automatically ignored (i.e., LASTLM and SPEC:HAND) are linked to the end of the IGNORE table. If the system being created is UTS, then the names of files to be automatically included are linked to the end of the INCLUDE table. These files are listed in Table 4-2. The appropriate routine to write the unlabelled portion of the tape is segloaded and entered. See Appendices A and B for a description of these routines. Upon return, ten additional pages of core are obtained and the common routine (CCA) to generate the remainder of either type of tape is entered.

For BO tapes, after processing the optional SN field, the appropriate routine to write the unlabelled portion of the tape is segloaded and entered. Upon return, for BPM/BTM systems, the files to be automatically included (see Table 4-2), are linked to the end of the INCLUDE table, ten additional pages of core are obtained and exit is made to CCA. If the system is UTS, the file, M:SPROCS in :SYS account, is opened, ten pages of core are obtained and the file is read into the newly acquired area and linked to the end of the INCLUDE table. M:SPROCS contains the names of the monitor overlays and shared processors but only the overlays are added to the INCLUDE table. The names of the other files to be automatically included are linked to the end of the added monitor overlay names and exit is made to CCA. For UTS the automatic INCLUDES reflect earlier releases of the system.

Upon entry, the common routine (CCA) begins by processing the INCLUDE list. This involves obtaining the name of a file, storing it in the open FPT (OPNTMSQN) for M:TM to the disc and then opening the file, using the FPARAM option. The file, thus opened, is checked first if it is a synonymous file in which case special handling is required, namely its parent name must be added before writing it to the tape. Note the parent file must occur before the synonymous file or the latter is lost.

Then the organization of the file is determined. If the tape being generated is PO then only consecutive files are processed, if BO then only keyed files. The other types are automatically handled later.

The PO/BO tape is then opened and the file is read into core and written to tape until an EOF is encountered at which point the DCB is closed. This routine is repeated until all the names in the INCLUDE table have been processed. When this processing is completed and the files thus written to the tape have been listed on the LL device, the next phase of DEF is entered.

If a BO tape is being generated, a null file, LASTLM, is written to the tape. Subsequently, or if a PO tape is being created, the FPT for open-next to the disc (OPNTM) is opened and file parameters obtained. If the file organization is consecutive (BO)/keyed (PO), the IGNORE table is searched to determine if it is listed there. If the DELETEF is set, then the file is deleted when M:TM is closed. If the file is not be be IGNOREd, then it is read into core and written to the tape. This procedure is repeated until all files in the current account have been processed.

If on opening-next-file, an abnormal return is made indicating the file is a synonymous file, its name is stored in a new INCLUDE table whose location is pointed by INCLSTRT and a flag (SYNFLG) is incremented, thus maintaining a running total of the number of synonymous files found.

When an abnormal return is made indicating an end of all files on open-next, if the tape being created is BO, it is immediately closed, rewound, and saved. If a PO tape is being generated, SYNFLG is tested. If non-zero, a second pass is made through the INCLUDE routines. If or when SYNFLG is zero, the null file, LASTLM, is written to the tape which is closed, rewound, and saved.

The pages of core acquired thus far are released. If ENDFLG is not set, the flags and counters are zeroed to prepare for the generation of another tape. If ENDFLG is set, DEF exits.

Table 4-2. Automatic INCLUDES

PO Tape *	BO Tape **	
UTS	UTS†	BPM/BTM
BPM	XDELTA	FMGE
UTS	LOGON	PASS1
SIG7FDP	TEL	ERRMSG
:BLIB	SUPER	:DIC
FLIBMODE	DEFCOM	:LIB
SIGMET	SYMCOM	M:C
M:CDCB	JIT0	M:OC
M:OCDCB	JIT1	M:BI
M:BIDCB	JIT2	M:CI
M:CIDCB	JIT3	M:SI
M:SIDCB	JIT6	M:EI
M:EIDCB	ANLZ	M:BO
M:BODCB	ERRMSG	M:CO
M:CODCB	GHOST1	M:SO
M:SODCB	RECOVER	M:PO
M:PODCB	M:SPROCS	M:GO
M:GODCB	M:MON	M:LO
M:LODCB	PCL	M:DO
M:DODCB	CCI	M:EO
M:EODCB	LOADER	M:LL
M:LLDCB	PASS2	M:CK
M:SLDCB	LOCCT	M:SL
M:ALDCB	PASS3	M:AL
M:LIDCB	DEF	M:LI
	Plus Monitor overlays from M:SPROCS	M:MON
		PCL
		CCI
		LOADER
		PASS2
		LOCCT
		PASS3
		DEF

* From Current Account

** From :SYS Account

†Reflects earlier releases of UTS

4.9 DEF MESSAGES.

::::SYSGEN DEF IN CONTROL::::	Commentary at beginning of execution.
::::DEF COMPLETED::::	Commentary at end of execution.
**CC TYPE UNKNOWN ****GET NEXT CC	Error in :Command. DEF reads next command.
**SYNTAX ERROR, NO '('	Error in syntax. DEF reads next command.
**DELIMITER MUST BE ',' OR ')'	Invalid terminator on :Command. DEF reads next command.
**NAME INVALID OR > 15 CHAR. LONG	DEF searches for next parameter.
****NOT ENOUGH CORE AVAILABLE *****SYSGEN DEF ABORTED	Work area too small. DEF exits.
****WRITING PO BY DEFAULT	Either no tape type specified or parameter invalid on :WRITE.
***ILLEGAL INCLUDE - WILL BE COPIED LATER	On the :INCLUDE command a keyed file (for PO) or a consecutive file (for BO) has been specified. The file name is printed above this message. DEF continues.
***DEF TYPE UNKNOWN	System type field of !DEF command has been specified but is invalid. DEF exits.
***TYPE UNKNOWN - xx M used	System type field of !DEF missing. DEF defaults to currently running system type (xx).
**NO ':' in column-1	Command in error. DEF reads next command.
****TROUBLE WITH M:SPROCS	In attempting to open M:SPROCS in creating a BO tape for UTS system, difficulty encountered. DEF releases the tape and if ENDFLG set, exits.
***CANNOT WRITE TAPE	
***CANNOT OPEN OUTPUT DEVICE	In attempting to open tape (BO/PO), abnormal condition occurs. DEF releases tape and if ENDFLG set, exits.
. . . PO TAPE CONTENTS . . .	These are subtitles that are followed by a list of the appropriate files.
. . . BO TAPE CONTENTS . . .	
INCLUDE ITEMS	
OTHER ITEMS	
********INCLUDE ITEMS NOT FOUND	

4.10 INTERNAL ROUTINES.

DEF	Main entry, initialize processor dynamic data area
READFRST	Process DEF command.
INIT	Initialization of pointers.
DEFRDCC	Read :Command for DEF, and branch to appropriate routine or set DELETEF.
DEFINCL	Process :INCLUDE.
DEFIG	Process :IGNORE.
DEFWRITE	Initial processing of :WRITE.
DFWRTPO	:WRITE processing for PO.
DFWRTBO	:WRITE processing for BO.
DEFTABLR	Processing name options on :INCLUDE or :IGNORE.
PAGER	Get a page of core and zero it out.

READCC	Reads :Commands for DEF. Register 12 = CC Buffer address
LISTCC	Display commands on LL device. Register 12 = CC Buffer address
GETRITEMON	Obtain appropriate WRITEMON overlay according to system type (UTS or BPM/BTM).
EOCCSCAN	Find end of current control command.
CCA	Generate PO/BO tape.
NXTINCL	Obtain next INCLUDE file name.
RDWRITEM	Read and write file.
SYNINCL	Process synonymous file includes.
NOINCL	End of includes, begin generating remainder of tape.
NXTFILE	Obtain next file on disc.
IGNOR1	Search ignore table for match.
ISSPEC	Delete file if required.
CLSDSK	Close file.
RDWRITE	Read file and write to tape.
ALLDONE	Releases pages acquired, if ENDFLG set, exits.
NXTTPE	Zeroes flags and counters, restores FPTs to original state, returns to INIT via PAGER.

Error and abnormal return routines.

LSTWRT	EOF on reading M:SI.
RTMAINCL	EOF on reading INCLUDE files.
OTMAINCL	Abnormal return on opening of INCLUDE file.
RTMA	EOF on reading M:TM file.
OTMA	EOF on open next of M:TM or synonymous file found for open-next.
OPOA	Cannot open BO/PO Tape.

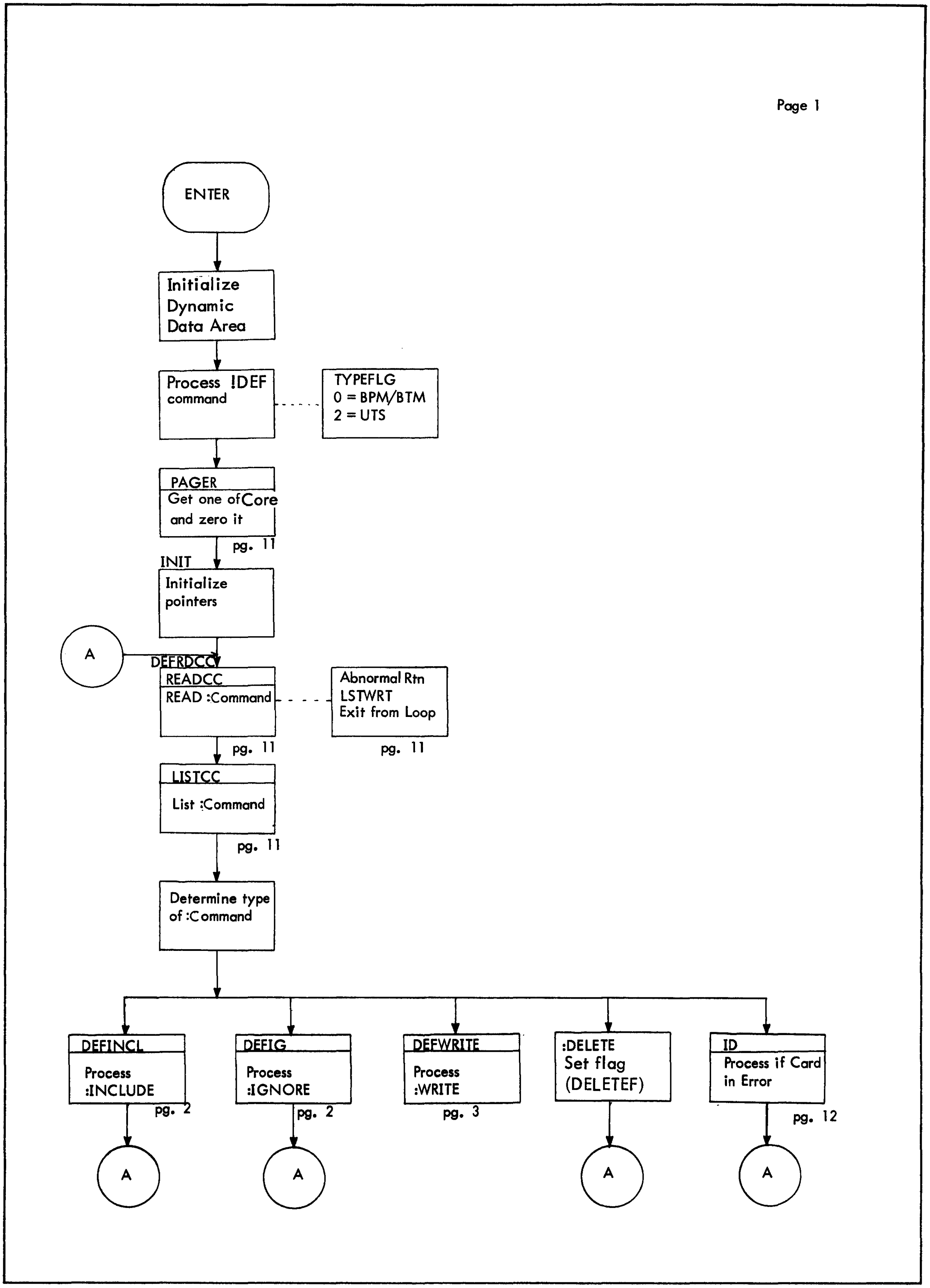


Figure 4-1. Flow Diagram of DEF

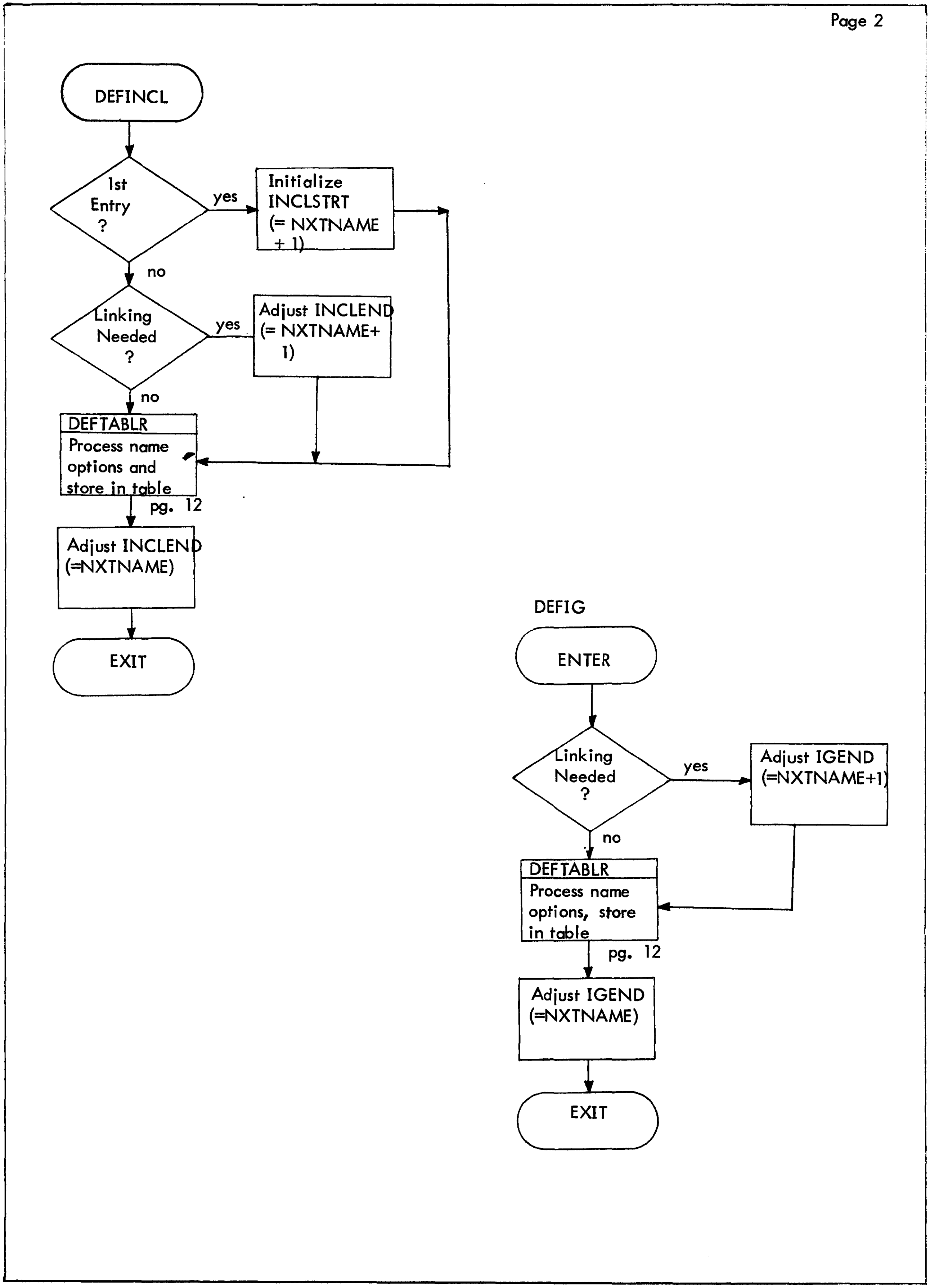

Page 2
DEFINCL
1st Entry ?
yes
Initialize INCLSTRT (= NXTNAME + 1)
no
Linking Needed ?
yes
Adjust INCLEND (= NXTNAME+ 1)
no
DEFTABLR
Process name options and store in table
pg. 12
Adjust INCLEND (=NXTNAME)
EXIT
DEFIG
ENTER
Linking Needed ?
yes
Adjust IGEND (=NXTNAME+1)
no
DEFTABLR
Process name options, store in table
pg. 12
Adjust IGEND (=NXTNAME)
EXIT

Figure 4-1. Flow Diagram of DEF (CONT)

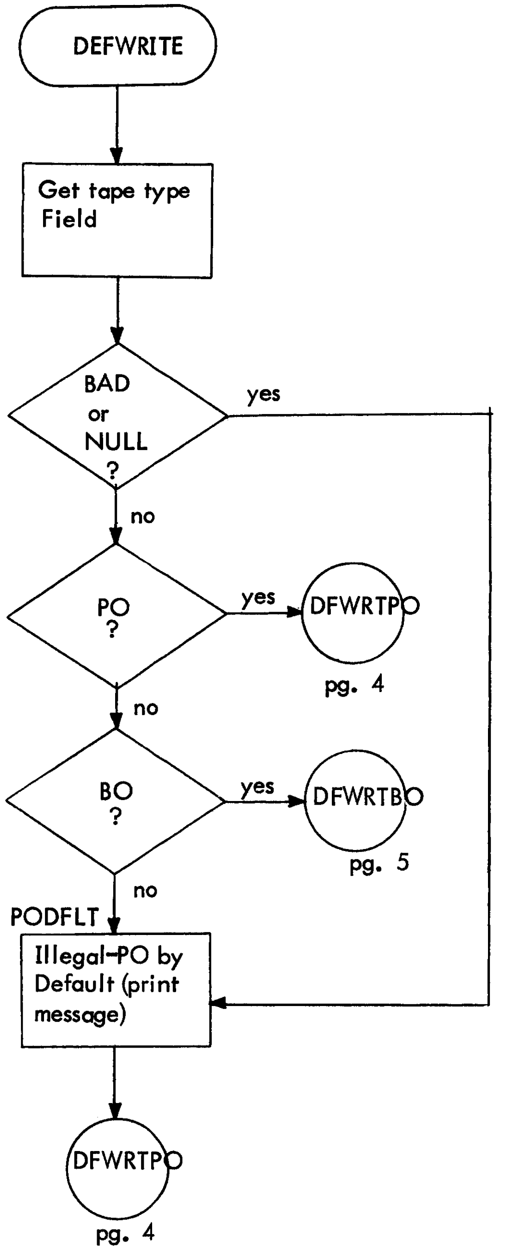


Figure 4-1. Flow Diagram of DEF (CONT)

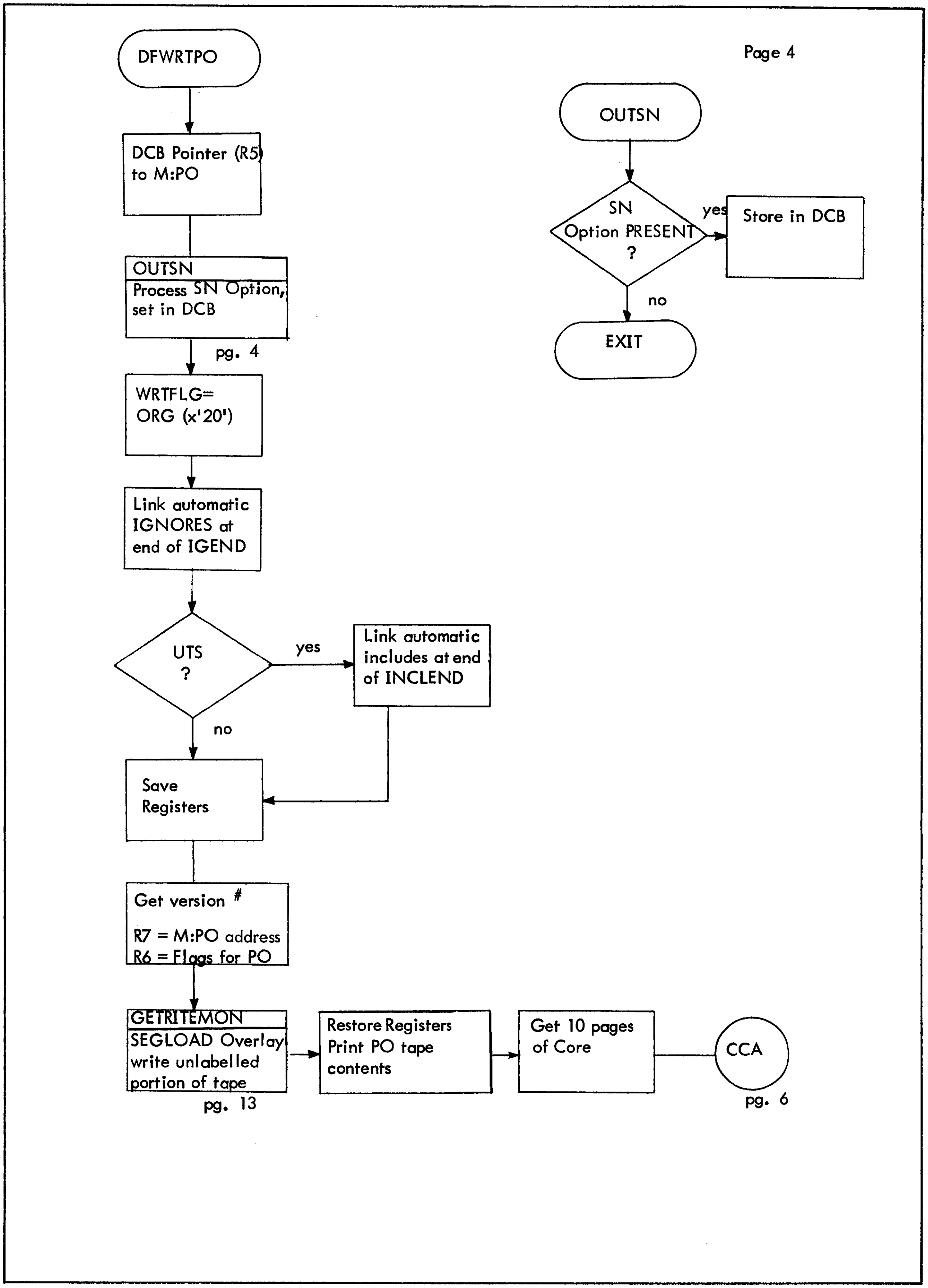


Figure 4-1. Flow Diagram of DEF (CONT)

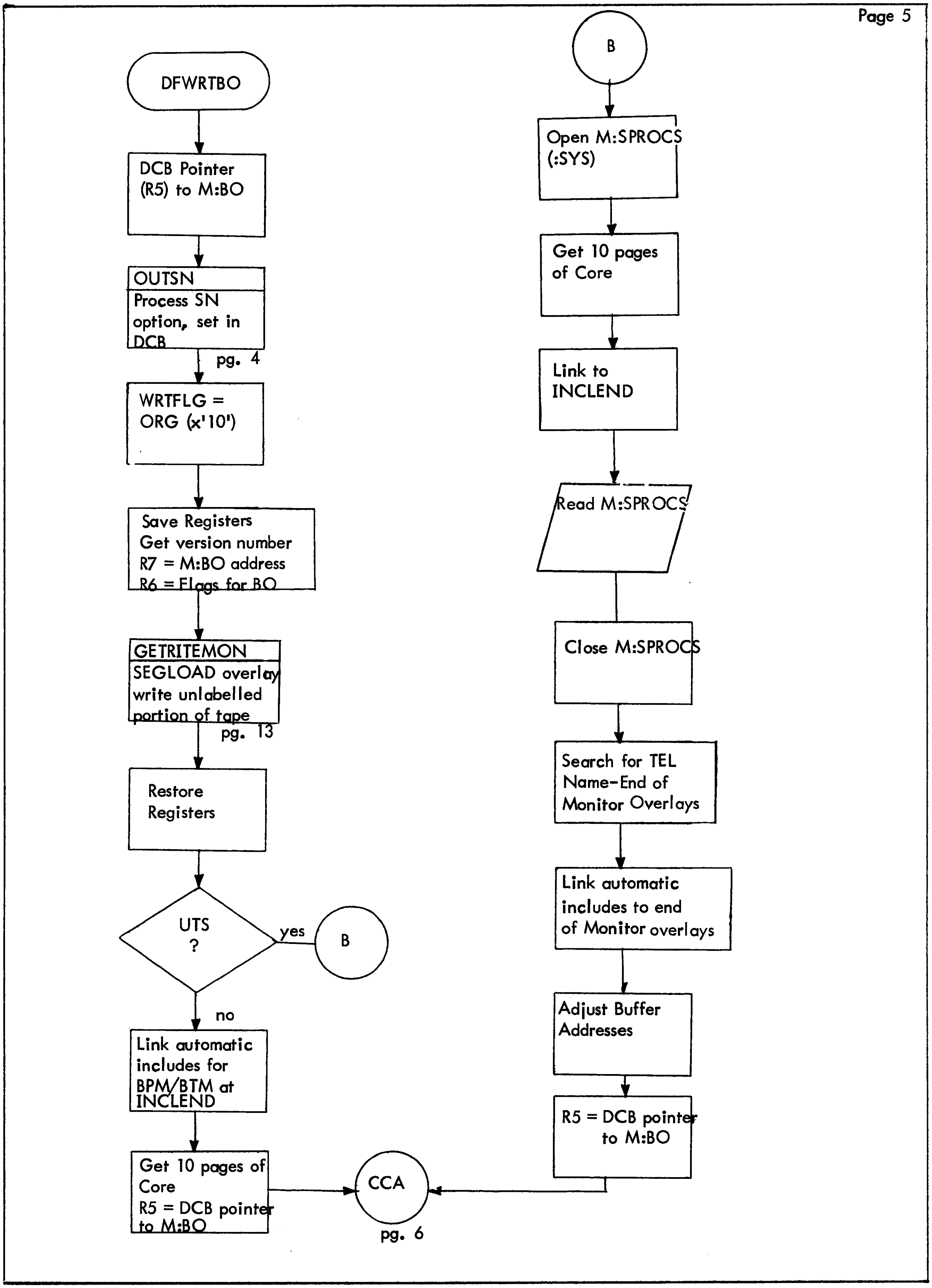


Figure 4-1. Flow Diagram of DEF (CONT)

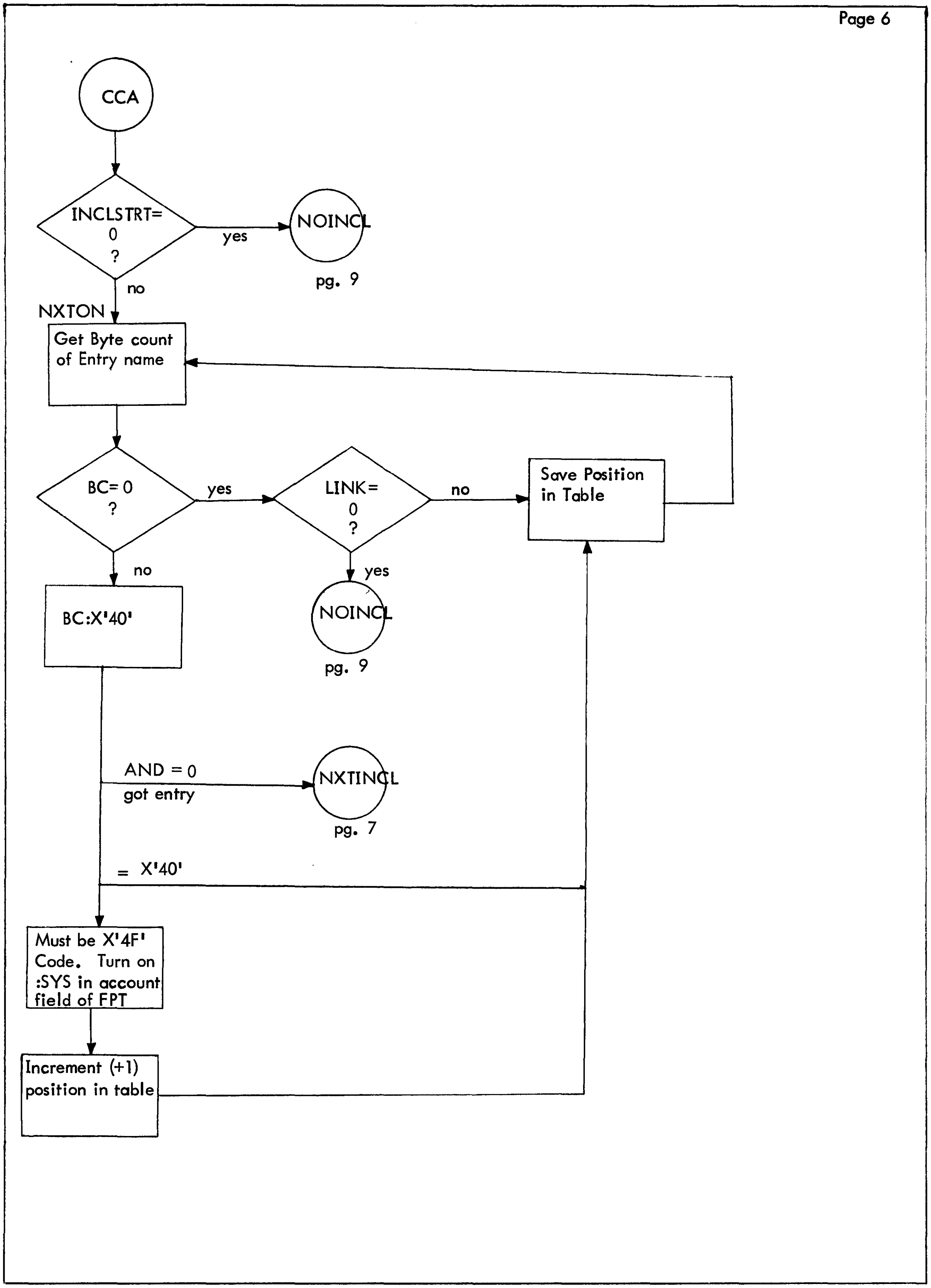


Figure 4-1. Flow Diagram of DEF (CONT)

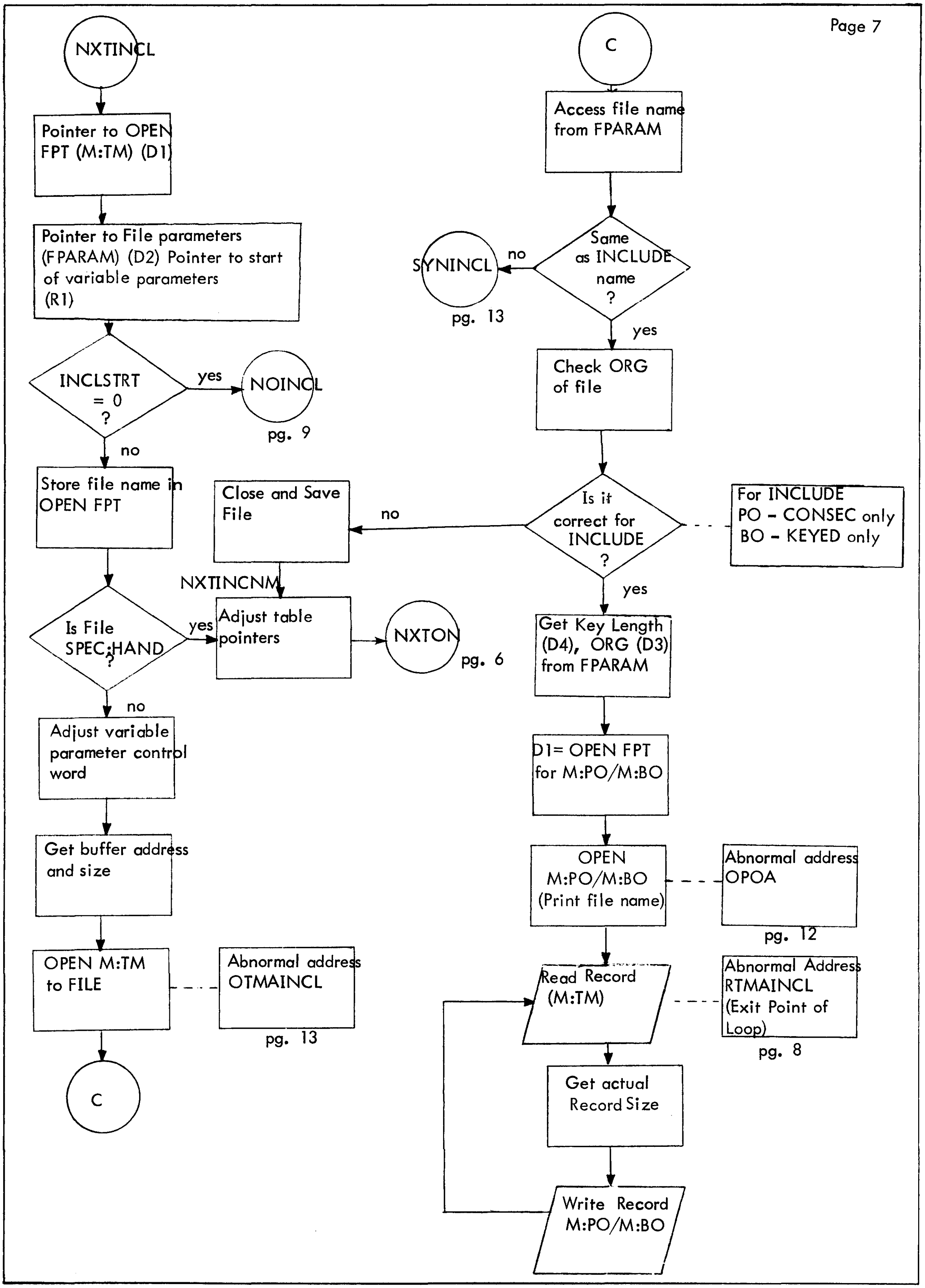

Page 7
NXTINCL
Pointer to OPEN FPT (M:TM) (D1)
Pointer to File parameters (FPARAM) (D2) Pointer to start of variable parameters (R1)
INCLSTRT = 0 ?
yes
NOINCL
pg. 9
no
Store file name in OPEN FPT
Is File SPEC;HAND ?
yes
no
Close and Save File
NXTINCNM
Adjust table pointers
NXTON
pg. 6
Adjust variable parameter control word
Get buffer address and size
OPEN M:TM to FILE
Abnormal address OTMAINCL
pg. 13
C
C
Access file name from FPARAM
Same as INCLUDE name ?
no
SYNINCL
pg. 13
yes
Check ORG of file
Is it correct for INCLUDE ?
no
For INCLUDE PO - CONSEC only BO - KEYED only
yes
Get Key Length (D4), ORG (D3) from FPARAM
D1= OPEN FPT for M:PO/M:BO
OPEN M:PO/M:BO (Print file name)
Abnormal address OPOA
pg. 12
Read Record (M:TM)
Abnormal Address RTMAINCL (Exit Point of Loop)
pg. 8
Get actual Record Size
Write Record M:PO/M:BO

Figure 4-1. Flow Diagram of DEF (CONT)

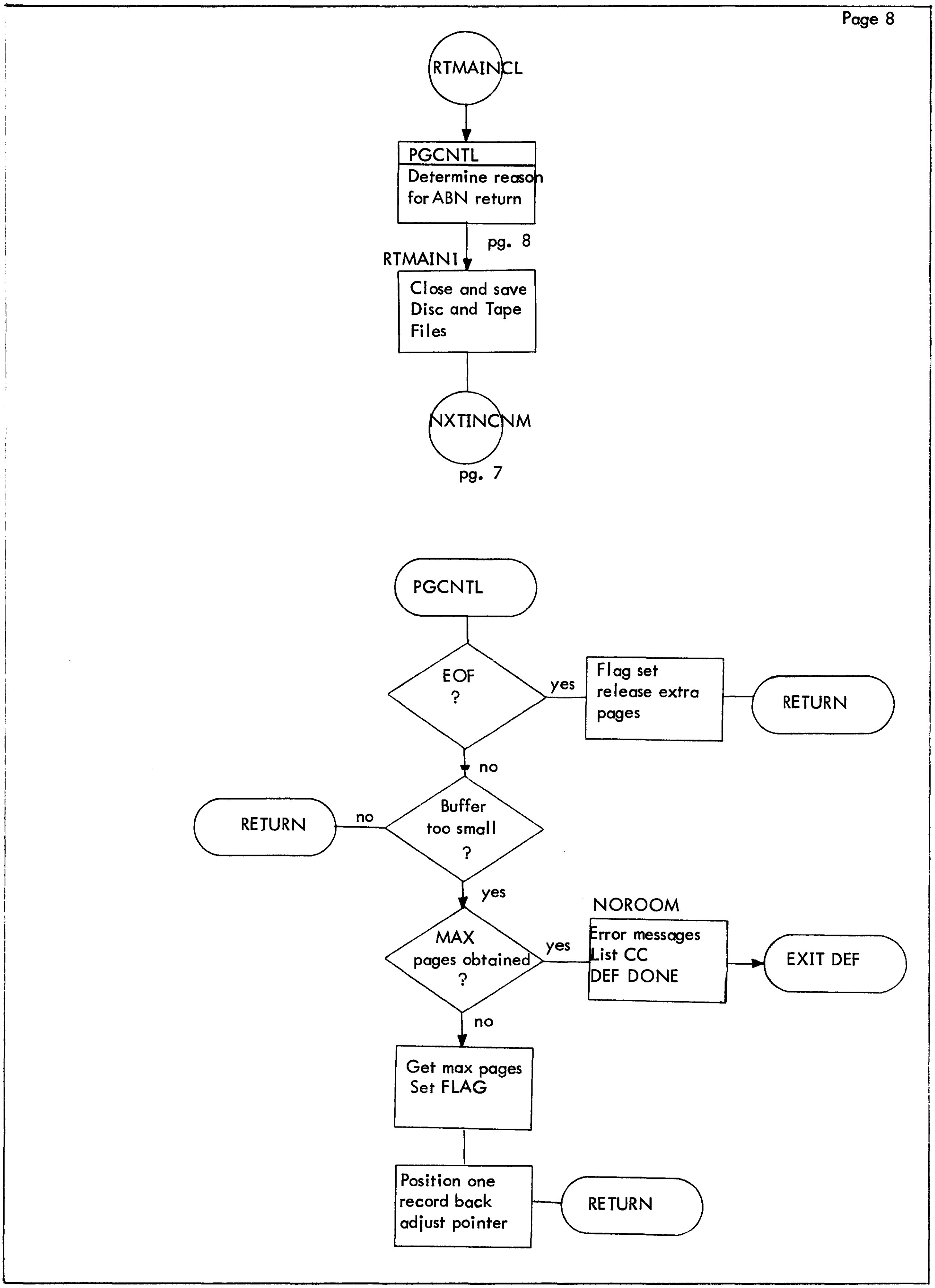

Page 8
RTMAINCL
PGCNTL
Determine reason for ABN return
pg. 8
RTMAIN1
Close and save Disc and Tape Files
NXTINCNM
pg. 7
PGCNTL
EOF ?
yes
Flag set release extra pages
RETURN
no
Buffer too small ?
no
RETURN
yes
NOROOM
MAX pages obtained ?
yes
Error messages List CC DEF DONE
EXIT DEF
no
Get max pages Set FLAG
Position one record back adjust pointer
RETURN

Figure 4-1. Flow Diagram of DEF (CONT)

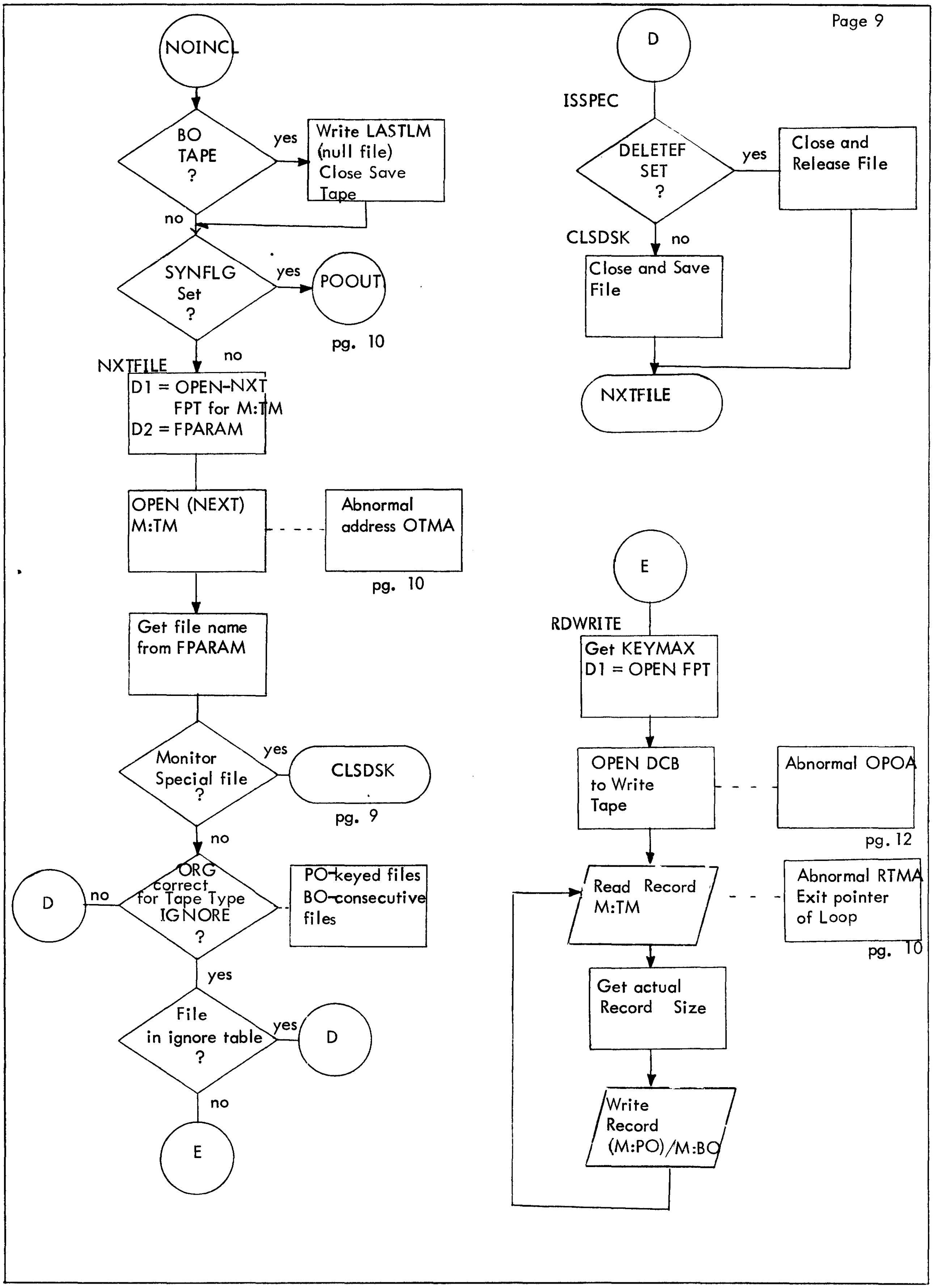


Figure 4-1. Flow Diagram of DEF (CONT)

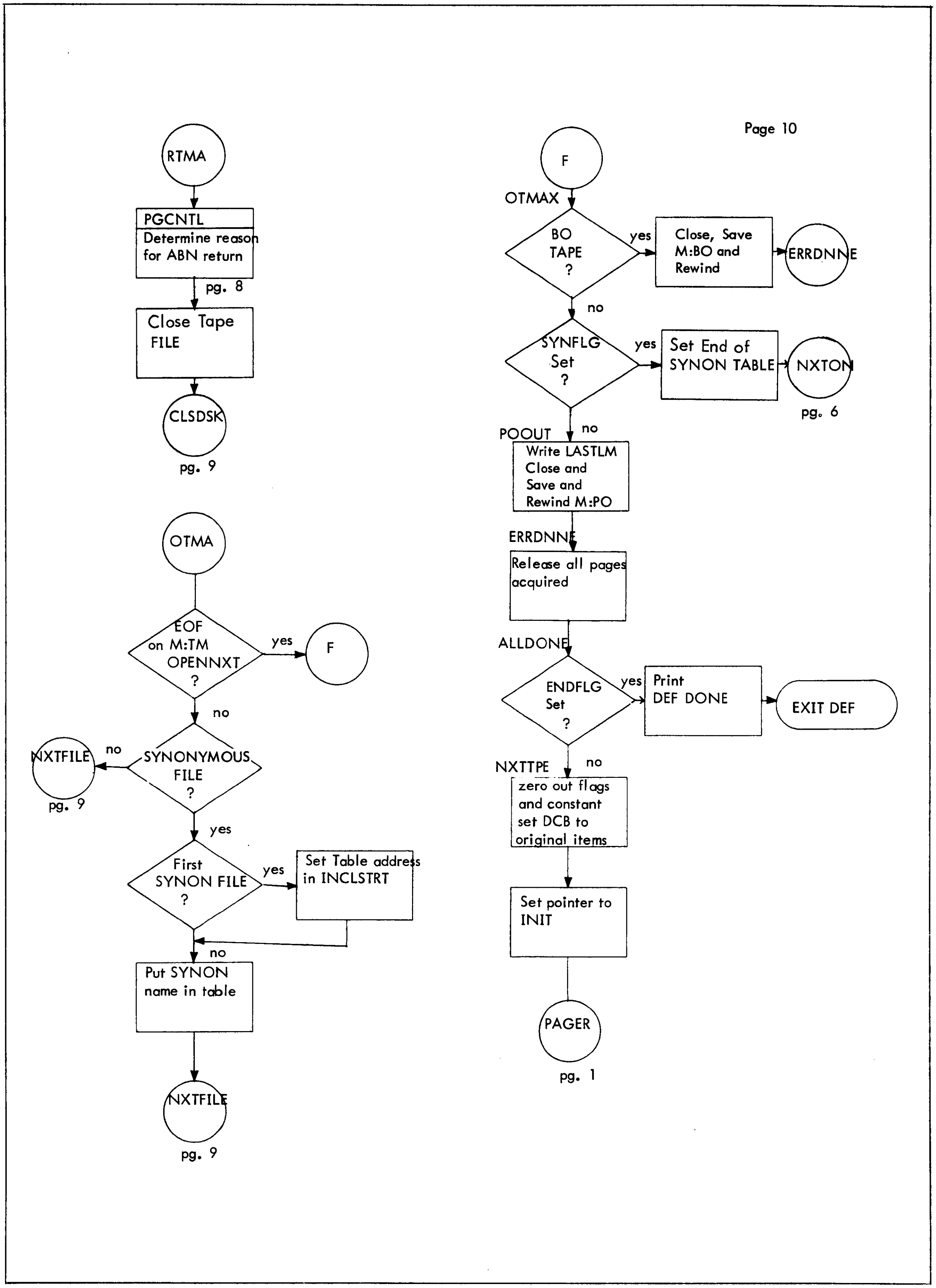


Figure 4-1. Flow Diagram of DEF (CONT)

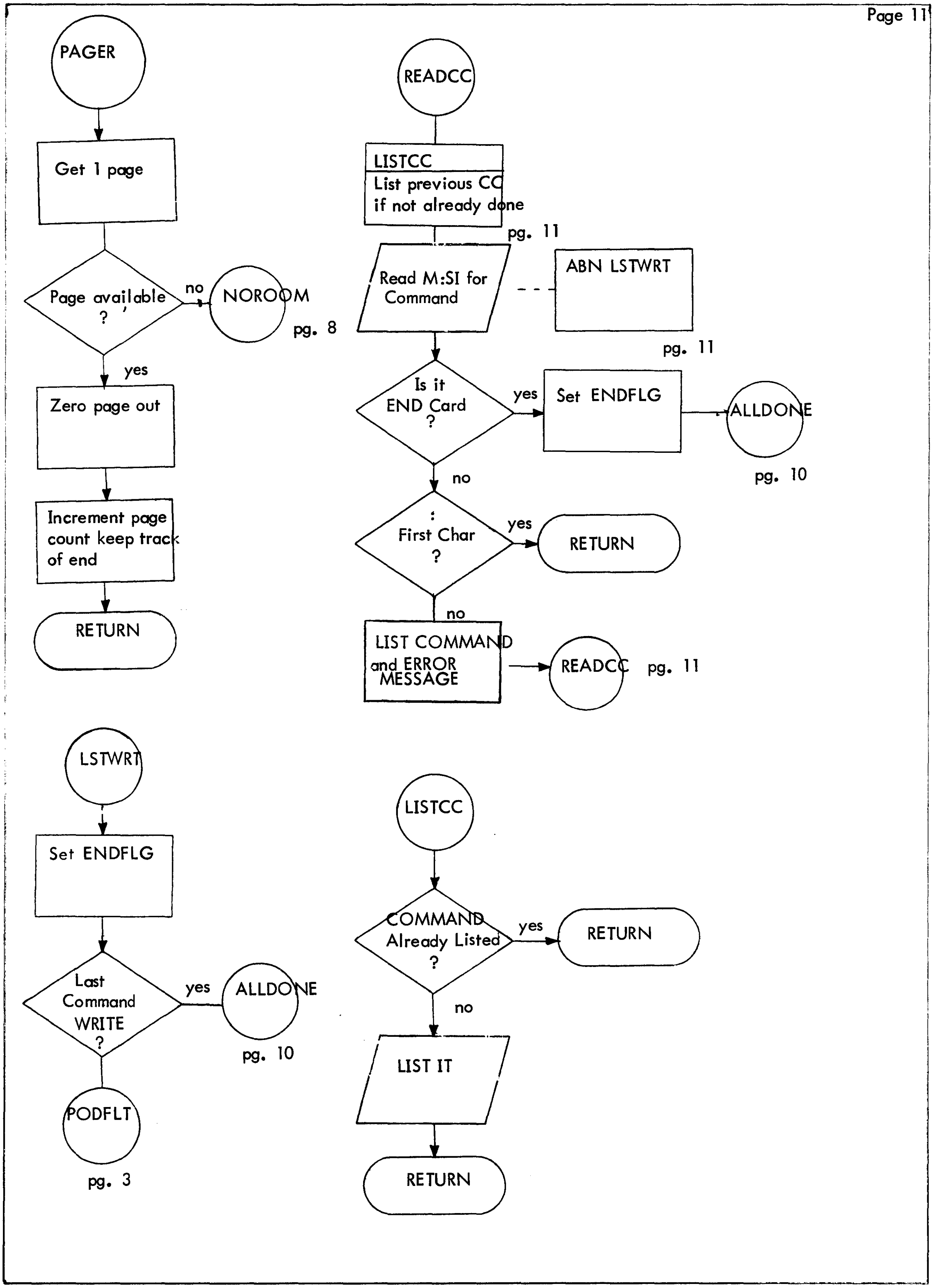


Figure 4-1. Flow Diagram of DEF (CONT)

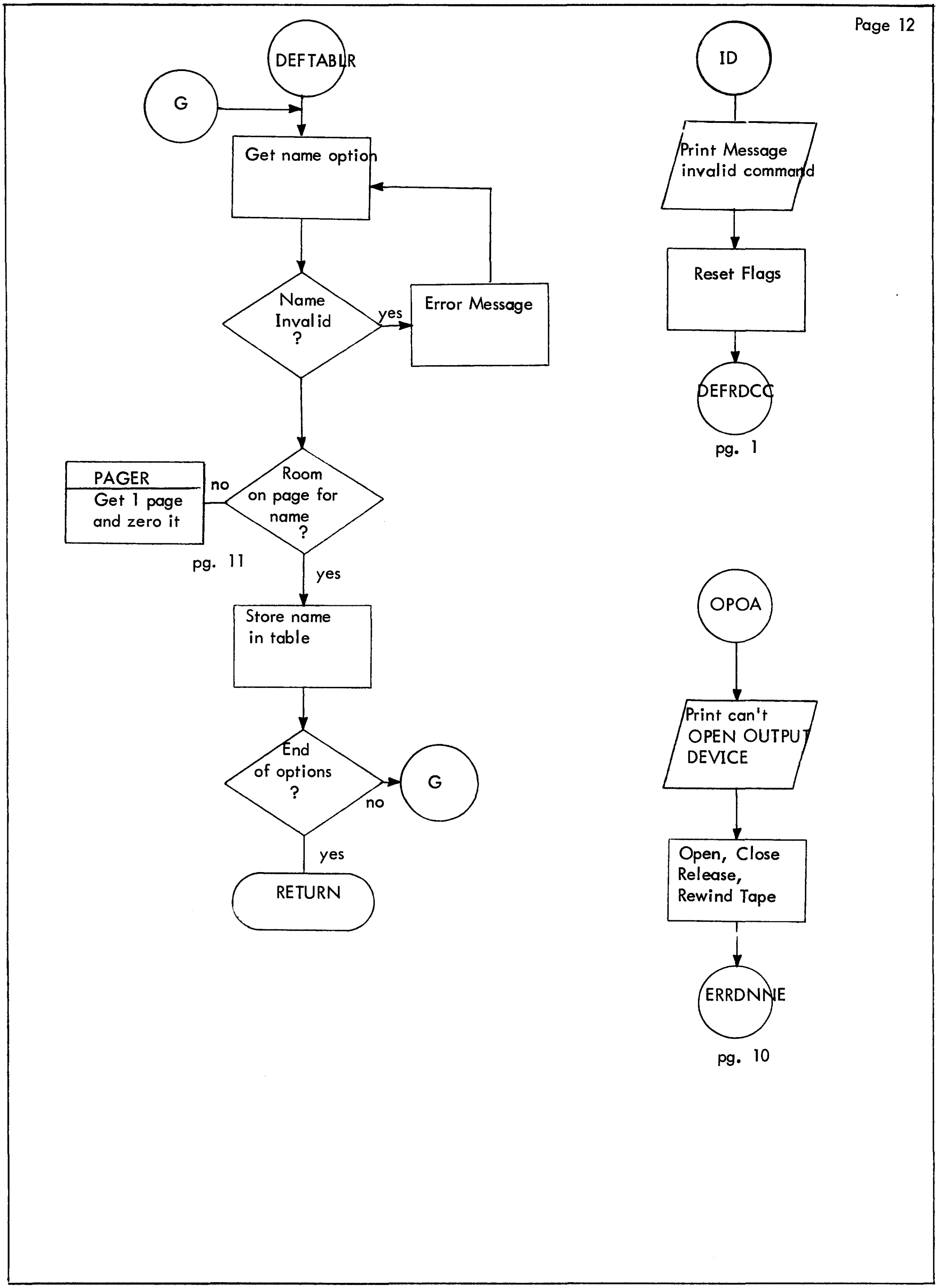


Figure 4-1. Flow Diagram of DEF (CONT)

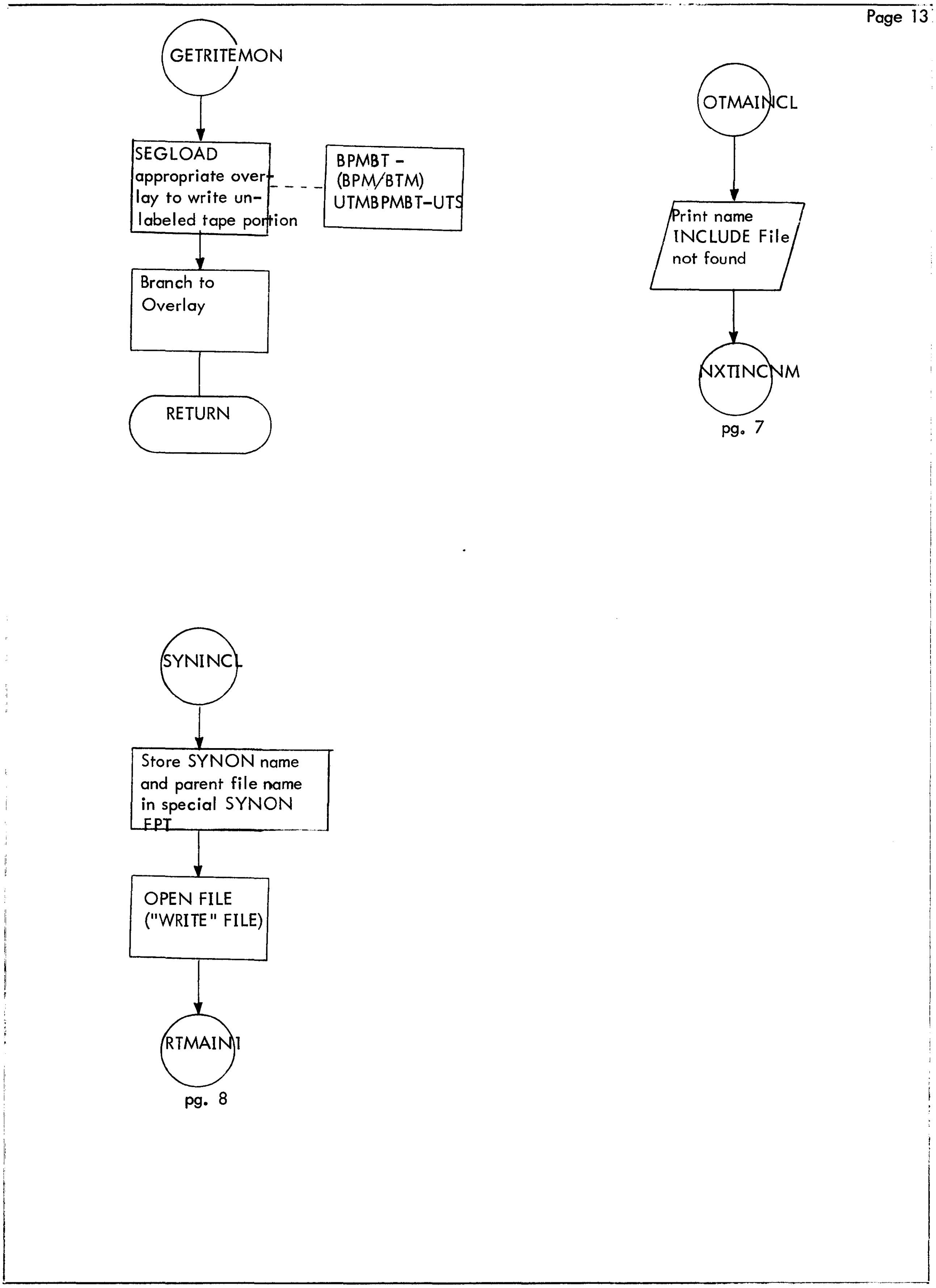

Page 13
GETRITEMON
SEGLOAD appropriate overlay to write unlabeled tape portion
BPMBT - (BPM/BTM) UTMBPMBT-UTS
Branch to Overlay
RETURN
OTMAINCL
Print name INCLUDE File not found
NXTINCNM
pg. 7
SYNINCL
Store SYNON name and parent file name in special SYNON FPT
OPEN FILE ("WRITE" FILE)
RTMAIN1
pg. 8

Figure 4-1. Flow Diagram of DEF (CONT)

5.0 LOCCT

5.1 PURPOSE

To generate a permanent file containing the Loader Overlay Control Command Table (LOCCT) information used by SYSGEN PASS3 and the system Loader when loading a specified element (e.g., M:MON, PCL, EDIT,).

5.2 CALLING SEQUENCE

The following Monitor control command sequence:

```
        !LOCCT  (LMN,X),.....
        !TREE.........          < optional >
        !DATA
        :LOCCT    X
```

5.3 INPUT

LOCCT control command from C device (80 characters maximum per physical image):

```
        :LOCCT
```

LOCCT from core common storage (UTS Base System) or absolute read/write scratch area on system's random access device (BPM/BTM Base System).

5.4 OUTPUT

A display of LOCCT control information to LL device. A permanent file to random access device containing LOCCT table. A copy of permanent file information to the PO device.

5.5 BASE REGISTERS

Register 7 = address in temp stack of control command PLISTs, and I/O PLISTs.

5.6 SUBROUTINES

NAMSCAN (used to obtain the name from the LOCCT control command).

5.7 DESCRIPTION

The LOCCT processor is entered from the system's Control Command Interpreter (CCI) as a result of a "!LOCCT" Monitor control command being encountered. The "!LOCCT" command replaces the "!LOAD" or "!OVERLAY" command, although the information on the "!LOCCT" command is identical to that of a LOAD or OVERLAY command. CCI processes the "!LOCCT" command and the optional "!TREE" command and terminates on the "!DATA" command. The resultant output is a LOCCT (see Chapter 6) which contains all of the information from the "!LOCCT" and "!TREE" commands in a compressed format. If the base system is BPM/BTM, the LOCCT is saved, temporarily, in the absolute read/write scratch area on the system random access device. However, if the base system is UTS, the LOCCT is saved in the common area of core.

CCI then enters the LOCCT processor. LOCCT proceeds to read its own control command, ":LOCCT". The syntax for this command is

```
        :LOCCT    name
```

This control command must immediately follow the "!DATA" control command. The "name" field is optional, however, if present, it is used to form a file-name which is the name of the permanent file containing the LOCCT. The "name" is syntactically checked and must be from one to ten alphanumeric characters of which one character must be alpha.

This control command cannot be continued to other physical images. Unlike other SYSGEN processors, a comment control command (i.e., a control command with an asterisk in character position one), is not recognized. However, comments may be added to the ":LOCCT" control command by preceding the comment with a peri od.

The file-name formed from the "name" option is accomplished by appending the "name" to the characters "LOCCT".

If the "name" field is not specified, a previous assignment must have defined the file-name. This is accomplished with the Monitor control command "!ASSIGN". The assignment is to the M:EO DCB, and must provide a file-name which is equivalent to that which would have been generated by the LOCCT processor, i.e., the name must be appended to the characters "LOCCT".

If an Assign command is specified, and the ":LOCCT" command specifies a "name", the "name" on the ":LOCCT" command supersedes the assignment.

The LOCCT processor continues by obtaining the LOCCT from either the absolute read/write scratch area or the system random access device (if the base system is BPM/BTM), or from the common area of core storage (if the base system is UTS). The LOCCT is interrogated to determine if it is legal. That is, all element file names (Relocatable Object Module names) are checked to make sure that they do not reference labeled tape for their inputs. All element file references must be to a random access device. The LOCCT processor then forms binary card image type records (Table 5-1) from the LOCCT. Each record contains a binary type identification, a hexadecimal sequence number, a byte checksum, a byte containing the size of the record (in bytes), and 84 bytes of LOCCT table information. The last eight columns (or 32 bytes) remain unused. Each record is then written in the file "LOCCT name" and is also output the PO device.

When the LOCCTs are being generated, it is recommended that they be built in the same account in which SYSGEN PASS3 executes. In a LOCCT, each element file reference contains a corresponding account number in which that file is to be found. If these account numbers vary, and the files and accounts do not exist when SYSGEN PASS3 eventually uses the LOCCTs, then the load phase will not be successful.

5.8 LOCCT MESSAGES

The error messages which may appear on the LL device while generating a LOCCT are as follows:

***UNKNOWN CC OR CONTINUATION ILLEGAL

The LOCCT control command ":LOCCT" contains an invalid character string for the characters assumed to be "LOCCT", or the command requests a continuation command (i.e., the ":LOCCT" command contains a semicolon prior to the "name" field). LOCCT displays abort message and exits to the Monitor.

LOCCT PROCESSOR ABORTED

This message is displayed in conjunction with other LOCCT processor messages.

***NAME INVALID

The ":LOCCT" command's "name" is not a legal alphanumeric name. LOCCT displays abort message and exits to the Monitor.

***CANNOT GENERATE LOCCT WITH ROMS ON LABELED TAPE

A load item or element file (Relocatable Object Module - ROM) is to be found on labeled tape and is invalid. LOCCT displays abort message and exits to the Monitor.

***ROM TABLE END CANNOT BE FOUND

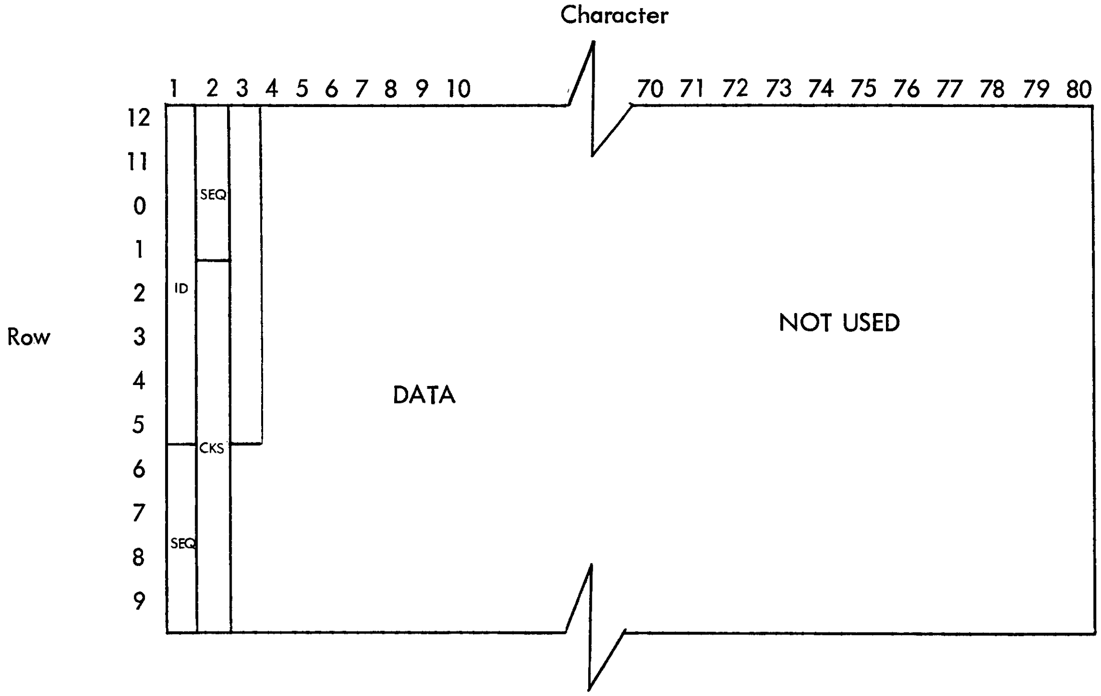
The LOCCT, as generated by CCI (System's Control Command Interpreter), contains invalid or unrecognized Relocatable Object Module (ROM) information, i.e., the end cannot be found. LOCCT displays abort message and exits to the Monitor.

***NAME > 10 CHARACTERS

The "name" contains more than ten characters. LOCCT displays abort message and exits to the Monitor.

***I/O ERR/ABN FOR READ C = XXXX
***I/O ERR/ABN FOR WRITE EO=XXXX
***I/O ERR/ABN FOR WRITE PO=XXXX
***OPEN EO ERR/ABN = XXXX
***ABS READ ERR/ABN = XXXX < for BPM/BTM only >

One of the above messages appears when an I/O error or abnormal condition is encountered on the C, EO, or PO devices. Under a BPM/BTM base system, the ABS READ message may appear. The value XXXX is the I/O error/abnormal code. LOCCT displays abort message and exits to the Monitor.



ID = X'3E' binary card code
X'1E' binary end card code

SEQ = two-digit hexadecimal sequence number

CKS = byte checksum of card image

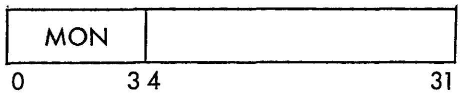
SIZ = number of useful bytes in card image, including the control word in columns 1-3. However, the size will never include columns 73-80, but, will include all of the DATA. If the useful data ends prior to column 72, then the size will not include those columns (or bytes) prior to column 72.

Figure 5-1. LOCCT Record Format

5.9 LOCCT PROCESSOR FLAG

LOC X'2B' in Monitor — This location contains the information which identifies what type of base Monitor is in control (e.g., BPM/BTM or UTS).

The format is:



where

MON = 4 BPM
5 BPM/BTM
6 UTS

5.10 INTERNAL ROUTINES

LOCCT — main entry, initialize processor, and general controller

GENFILE — generate a permanent file for LOCCT

LOCCT1 — form next record from LOCCT information

GETCOM — get original LOCCT from common storage
Register 13 = buffer address where LOCCT is to end up
Register 14 = buffer size in bytes

FINDEND — find end of LOCCT for purposes of generating a permanent copy
Register 13 = base address of work area
Register 7 = size of LOCCT in words

FINDENDX — check for valid ROM table in LOCCT
Register 2 = relative address of ROM table in LOCCT

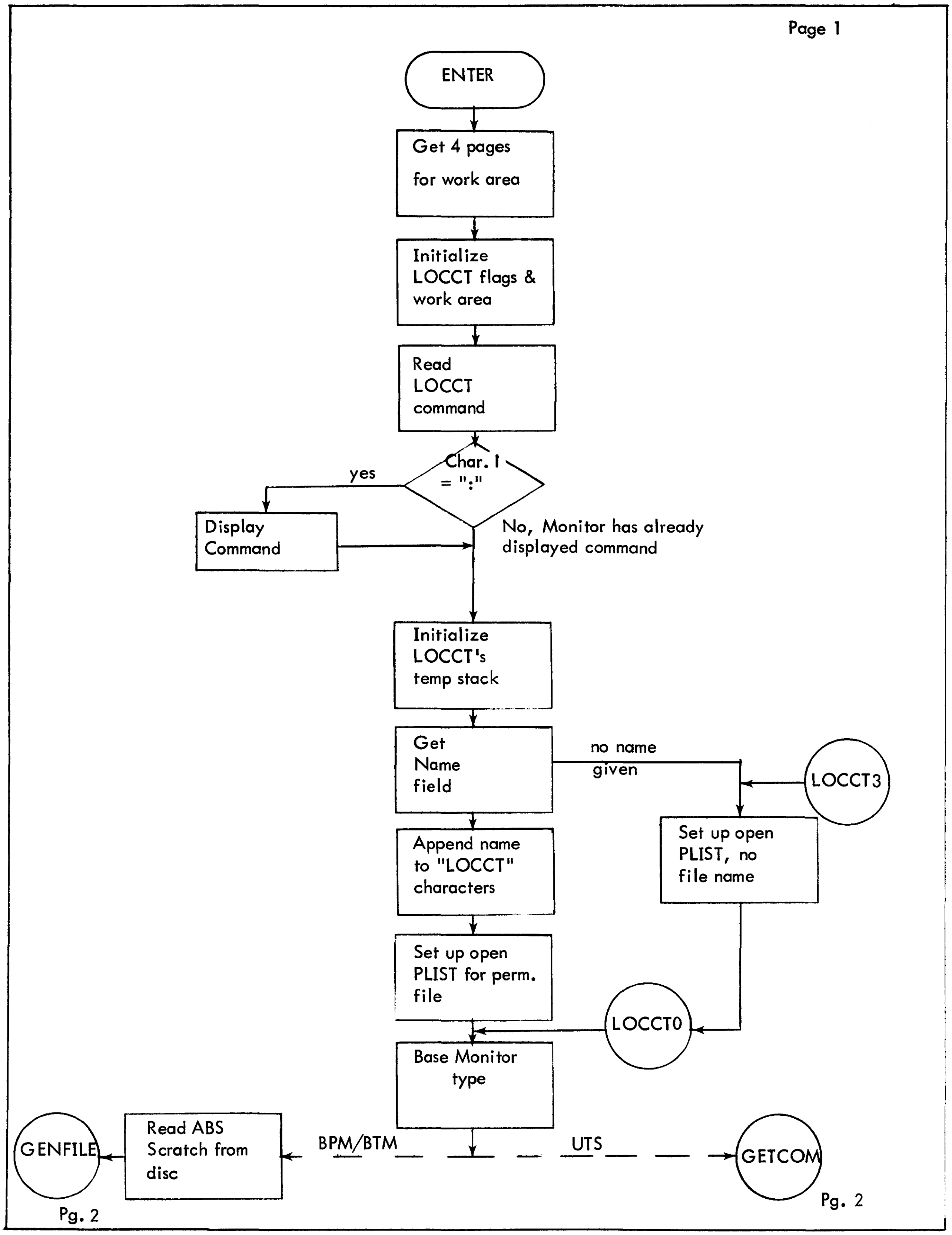
FINDROMX — obtain from TREE table (in LOCCT) the next ROM table information pointer.
Register 1 = address of LOCCT

CONV — convert error/abnormal code to EBCDIC

Input: Register 10 = error/abnormal code, (bits 0-7)
Output: Register 4 = converted code in EBCDIC.

Special error routines include: EO, ECOMMON, E1, E2, E3, E4, CE, CA, WE, WA, PE, PA, OE, OA, RE, RA

5.11 LOCCT Flow Charts



Page 1
ENTER
Get 4 pages for work area
Initialize LOCCT flags & work area
Read LOCCT command
Char. 1 = ":"
yes
Display Command
No, Monitor has already displayed command
Initialize LOCCT's temp stack
Get Name field
no name given
LOCCT3
Append name to "LOCCT" characters
Set up open PLIST, no file name
Set up open PLIST for perm. file
LOCCT0
Base Monitor type
BPM/BTM
UTS
Read ABS Scratch from disc
GENFILE
Pg. 2
GETCOM
Pg. 2

Figure 5-2. Flow Diagram of LOCCT

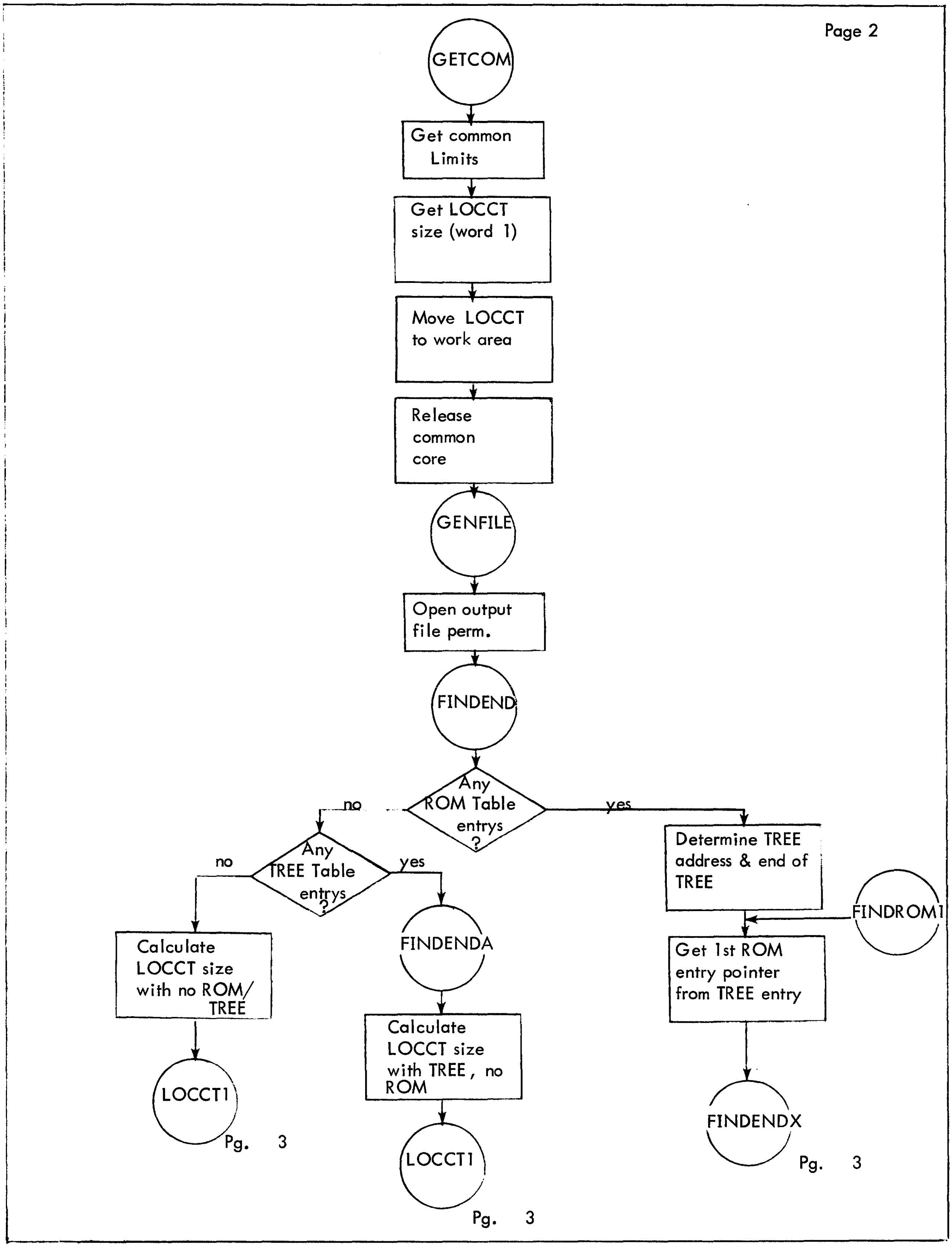


Figure 5-2. Flow Diagram of LOCCT (Cont.)

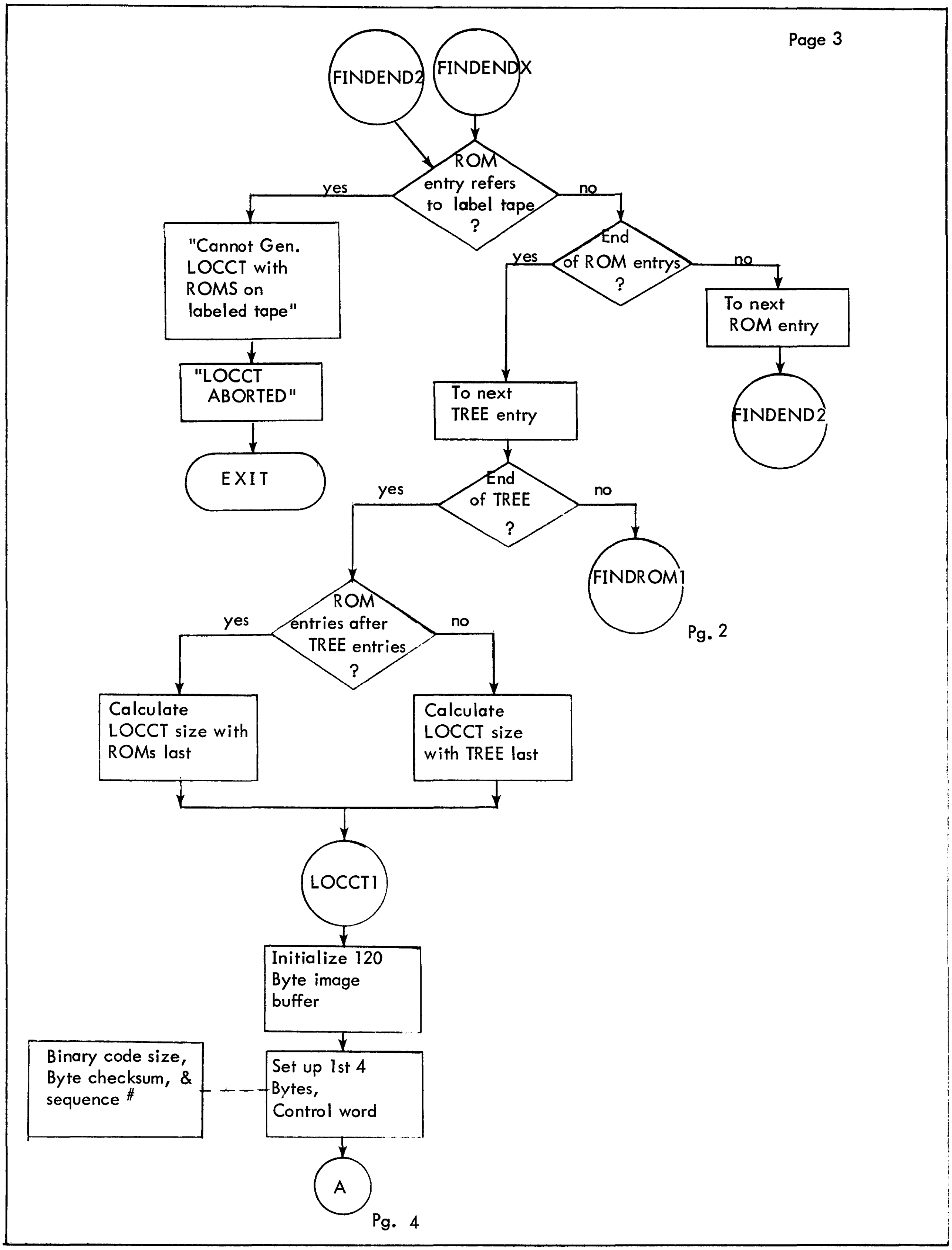

Page 3
FINDEND2
FINDENDX
ROM entry refers to label tape ?
yes
no
"Cannot Gen. LOCCT with ROMS on labeled tape"
"LOCCT ABORTED"
EXIT
End of ROM entrys ?
yes
no
To next ROM entry
FINDEND2
To next TREE entry
End of TREE ?
yes
no
FINDROM1
Pg. 2
ROM entries after TREE entries ?
yes
no
Calculate LOCCT size with ROMs last
Calculate LOCCT size with TREE last
LOCCT1
Initialize 120 Byte image buffer
Binary code size, Byte checksum, & sequence #
Set up 1st 4 Bytes, Control word
A
Pg. 4

Figure 5-2. Flow Diagram of LOCCT (Cont.)

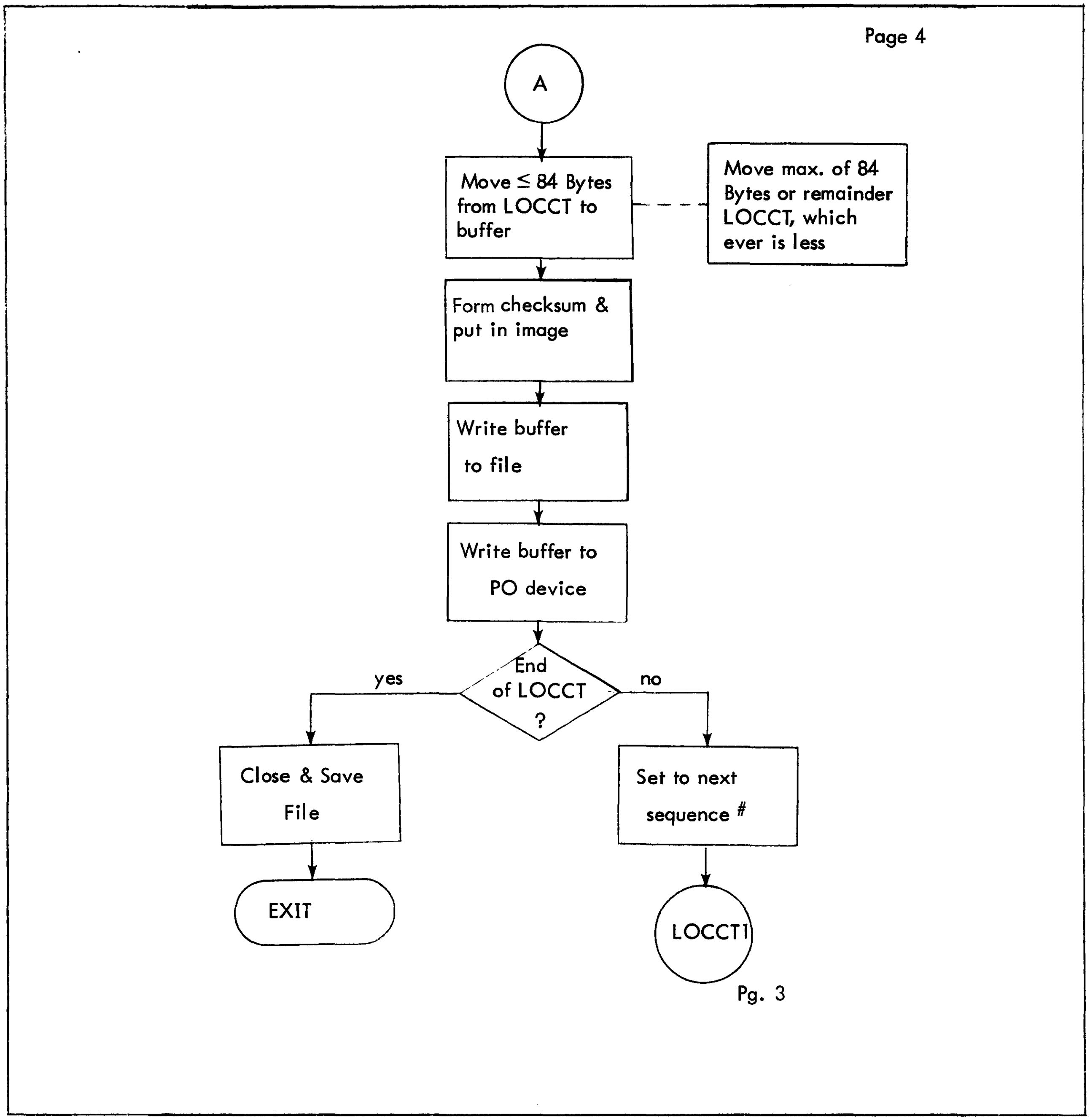
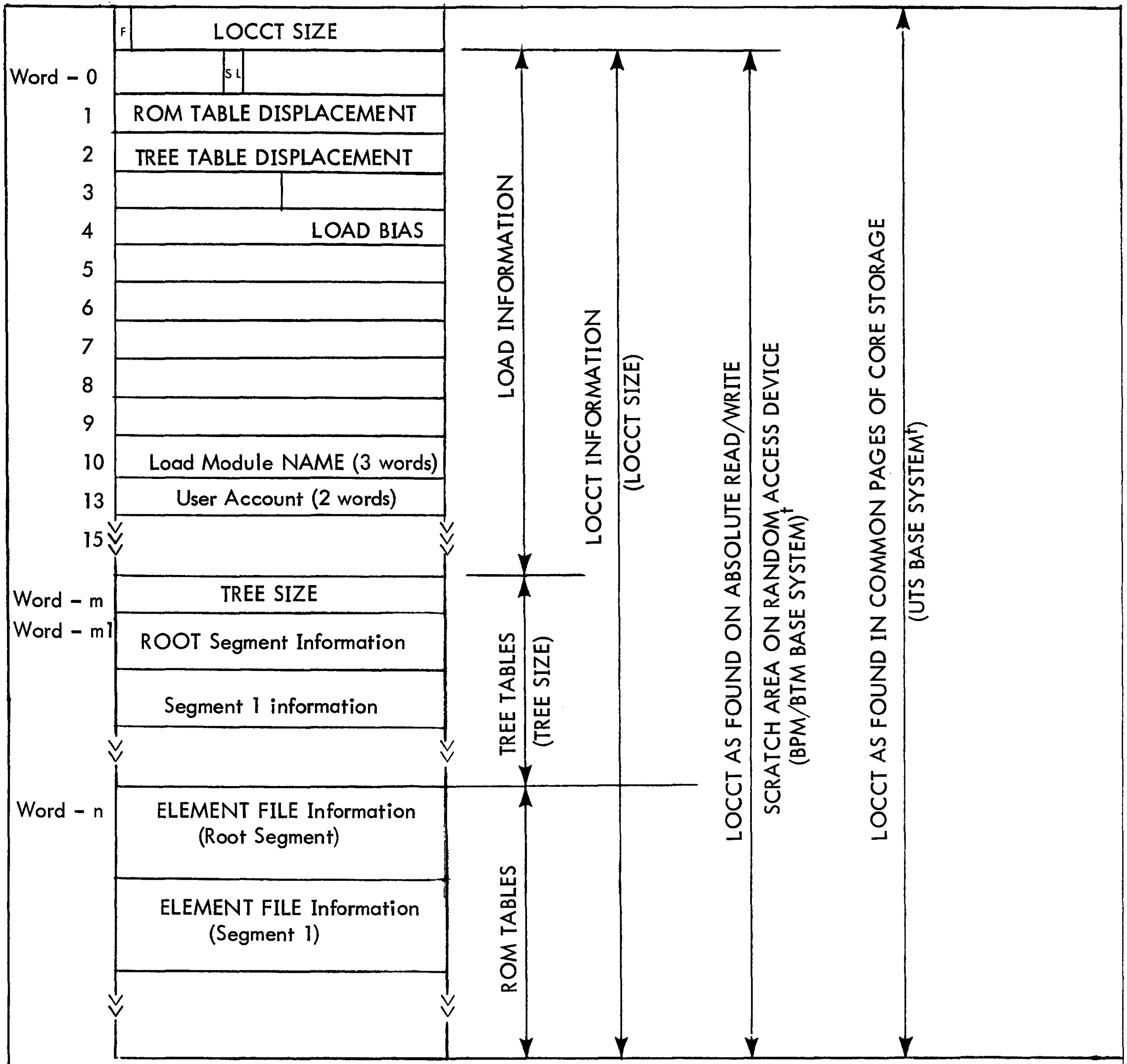

Page 4
A
Move ≤ 84 Bytes from LOCCT to buffer
Move max. of 84 Bytes or remainder LOCCT, which ever is less
Form checksum & put in image
Write buffer to file
Write buffer to PO device
End of LOCCT ?
yes
no
Close & Save File
EXIT
Set to next sequence #
LOCCT1
Pg. 3

Figure 5-2. Flow Diagram of LOCCT (Cont.)

6.0 TABLES, FILES, SUBROUTINES

6.1 SYSGEN LOADER OVERLAY CONTROL COMMAND TABLE (LOCCT)


F
LOCCT SIZE
Word – 0
S L
1
ROM TABLE DISPLACEMENT
2
TREE TABLE DISPLACEMENT
3
4
LOAD BIAS
5
6
7
8
9
10
Load Module NAME (3 words)
13
User Account (2 words)
15
Word – m
TREE SIZE
Word – m1
ROOT Segment Information
Segment 1 information
Word – n
ELEMENT FILE Information (Root Segment)
ELEMENT FILE Information (Segment 1)
LOAD INFORMATION
TREE TABLES (TREE SIZE)
ROM TABLES
LOCCT INFORMATION (LOCCT SIZE)
LOCCT AS FOUND ON ABSOLUTE READ/WRITE
SCRATCH AREA ON RANDOM ACCESS DEVICE (BPM/BTM BASE SYSTEM)†
LOCCT AS FOUND IN COMMON PAGES OF CORE STORAGE (UTS BASE SYSTEM†)

where

ROM TABLE DISPLACEMENT the number of words from word – 0 to word n.

TREE TABLE DISPLACEMENT the number of words from word – 0 to word – m1.

LOAD BIAS the bias at which to load this element.

LOAD MODULE NAME the name given to this element.

USER ACCOUNT the account in which this LOCCT is being generated. PASS3 forces this to the current account.

F a flag which identifies this LOCCT (if in common storage only, i.e., UTS Base System) as coming from PASS3 or the or the Systems Control Command Interpreter (CCI). If CCI, F=0 (Bit 0); if PASS3, F = 1 (Bit 0).

SL the load severity level indicator. PASS3 forces this to be "F", the highest severity level possible (Bits 8–11).

† BASE SYSTEM refers to the monitor type which is in execution.

Figure 6-1. LOCCT Format

Word		
0	SEGMENT NAME (TEXTC FORM)	
1		
2		
3	1st ROMPOINTER	BACK LINK
4	FORWARD LINK	OVERLAY LINK
5	00 SIZE	00 LOCATION
6	RF/DF SIZE	RF/DF LOCATION
7	01 SIZE	01 LOCATION
8	EXPRESSION SIZE	EXPRESSION LOCATION
9	10 SIZE	10 LOCATION
10		

where

1st ROM POINTER the number of words from word-n to first ROM* name (or Element File Information block) in the LOCCT.

BACK, OVERLAY, AND FORWARD LINKS the displacements from word-m1 to Segment Information for next overlay, in the overlay structure.

00 SIZE AND 00 LOCATION the load modules' protection type 0 size (doublewords) and base location (doubleword address).

01 SIZE AND 01 LOCATION same as for 00, except it is for load modules' protection type 1.

10 SIZE AND 10 LOCATION same for 00, except it is for load modules' protection type 2.

RF/DF SIZE AND RF/DF LOCATION the External DEF/REF table's size (words) and base location (doubleword address) for this load module.

EXPRESSION SIZE AND EXPRESSION the Expression (External DEF/REF) table's size (words) and base location (doubleword address) for this load module.

NOTE: *ROM name implies Relocatable Object Module name.

Figure 6-2. TREE Entry Format

The Element (ROM) Table's entry has the following format:

Word		
0	ELEMENT FILE NAME (TEXT FORM)	
1		
2		FLAG
3	ACCOUNT OF ELEMENT FILE	
4		
5	PASSWORD	
6		

where

FLAG is a flag identifying the last Element File Name within this Table and identifying the element file name as being found on labeled tape, i.e., X'4x' = not end of Element File names within this table.

X'0x' = end of Element File names within this table.

x = 2 = Element File name found on labeled tape.

Figure 6-3. ROM Table Entry

6.2 SYSGEN LOAD MODULE COMPONENTS

KEY = "HEAD"

Word 0	TYPE	00	FF	SIZE	BPM/BTM HEAD	UTS HEAD
1	A T		STADD			
2			MODBIAS			
3	LOCB/SIZU		LOCU			
4						
5	RFDFU		#W			
6						
7						
8						
9						
10						
11						

where

TYPE = X'81' library load module (used by SYSGEN).
= X'80' load module.

SIZE = X'18' HEAD size (bytes) for BPM/BTM target system.
= X'30' HEAD size (bytes) for UTS target system.

*A = 1 load module is in absolute form, i.e., no relocation dictionary(s) (RELDICT).
= 0 load module is relocatable.

*T = 1 no TCB
= 0 TCB is present.

*STADD Load module's entry point or start address.

#W number words in TREE.

MODBIAS load module's bias, doubleword address.

LOCB SECT.00 base location, doubleword address for BPM/BTM module.
LOCU SECT.00 base location, doubleword address for UTS module.
SIZEU SECT.00 size, in doublewords, for UTS module.
RFDFU REF/DEF STACK (RFDFSTK) size, in words, for UTS module.

*For SYSGEN - generated load modules: A=0, T=1, and STADD=0.

Figure 6-4. HEAD Record Format

KEY="TREE"

Word				
0				#W
1		SEGMENT NAME		
2		(TEXTC FORM)		
3				
4				
5				
6	00 SIZE		00 LOCATION	
7	RF/DF SIZE		RF/DF LOCATION	
8				
9	EXPRESSION SIZE		EXPRESSION LOCATION	
10				
11				

where

#W number words in TREE.

SEGMENT NAME name of load module root segment.

00 SIZE SECT.00 size in doublewords

00 LOCATION SECT.00 base address, as a doubleword address.

RF/DF SIZE RFDFSTK size in words

RF/DF LOCATION RFDFSTK base address, as a doubleword address.

EXPRESSION SIZE EXPRSTK size in words.

EXPRESSION LOCATION EXPRSTK base address, as a doubleword address.

Figure 6-5. TREE Record Format

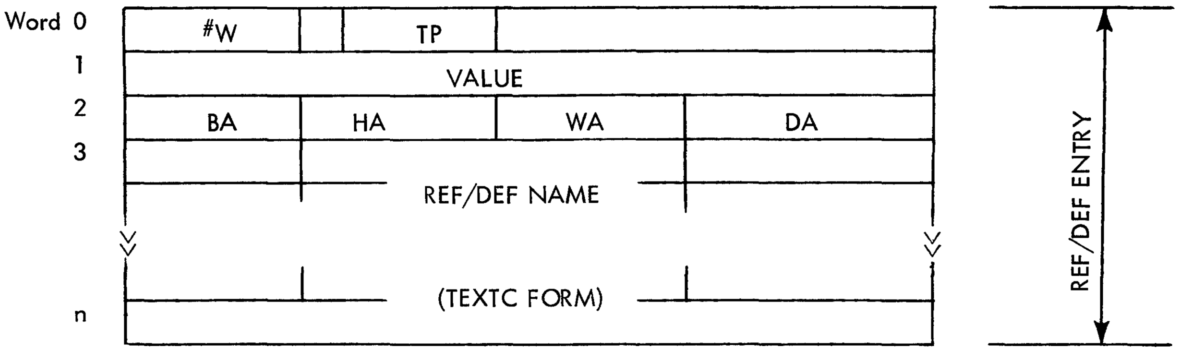
The KEY names used to identify the remainder of a load module are formed by appending to the SEGMENT NAME a one-byte digit which identifies a component as follows:

digit = 00 segments RFDFSTK
= 01 segments EXPRSTK
= 02 RELDICT.00 (relocation dictionary for SECT.00)
= 03 SECT.00 (SYSGEN - generated tables, data)

Therefore, for a segment name of "ROOT" (refer to PASS3), the corresponding components will be:

05D9D6D6E300	implies	"ROOT$_{00}$"	=RFDFSTK
05D9D6D6E301	implies	"ROOT$_{01}$"	=EXPRSTK
05D9D6D6E302	implies	"ROOT$_{02}$"	=RELDICT.00
05D9D6D6E303	implies	"ROOT$_{03}$"	=SECT.00

RFDFSTK (Generated by call to MODIFY subroutine, 6.5)



where

#W number of words in REF/DEF entry (n + 1).

TP = 0 external definition.

= 1 secondary external reference.

= 2 primary external reference.

= 3 dummy section.

= 4 or 6 control section.

= 5 forward reference (null if #W = 1).

VALUE actual value assigned to an external definition. 0, when name is external reference.

BA = 1 VALUE is byte resolution.

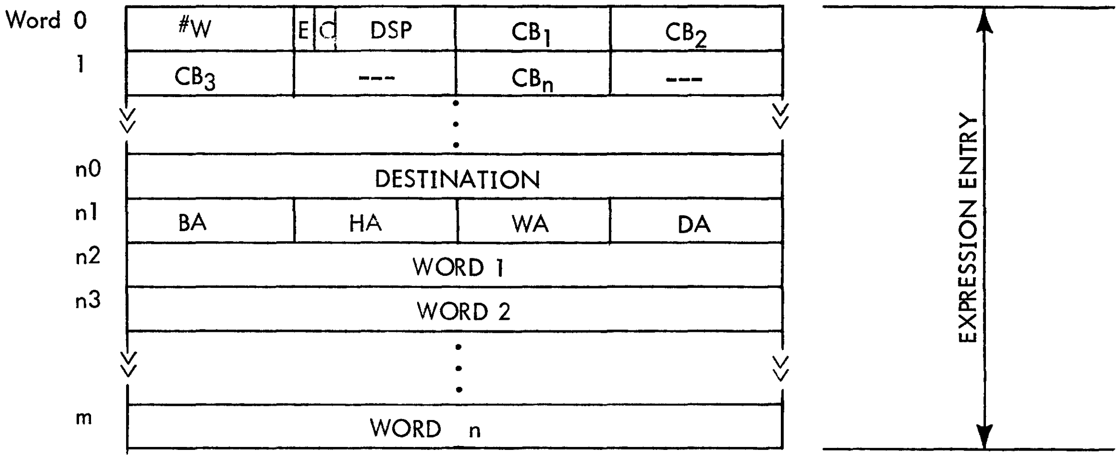
HA = 1 VALUE is halfword resolution.

WA = 1 VALUE is word resolution.

DA = 1 VALUE is doubleword resolution.

Figure 6-6. RFDFSTK Format

EXPRSTK (Generated by call to MODIFY Subroutine 6.5)



where

#W number of words in expression entry (m + 1).

DSP the number of words to word - n2.

CB_1, CB_2, ...,CB_n expression evaluation control bytes.

Figure 6-7 EXPRSTK Format

The expression control byte values recognized by SYSGEN include:

CB_i = 0	null
= 1	add constant from next word ($word_i$) in word list in EXPRSTK.
= 2	end of expression
= X'20' to X'23'	add declaration according to pointer given in text word ($word_i$) of word list with resolution of bits 6-7 of CB.
= X'28' to X'2B'	subtract declaration similar to X'20' to X'23'.
= X'30' to X'33'	change expression resolution to bits 6-7 of CB.
= X'34' to X'37'	add the ASECT doubleword address zero to the expression at the resolution given.
= X'38' to X'3B'	subtract the ASECT similar to X'34' to X'37'.

The CB_i digit 's format for X'20' throught X'2B' is

00 10 S0 rr	

where

S = 0	add
= 1	subtract
rr = 0	byte resolution
= 1	halfword resolution
= 2	word resolution
= 3	doubleword resolution

The CB_i digit's format for X'30' through X'33' is

00	11	00	rr

where

rr = resolution code as previously described.

The CB_i digit's format for X'34' through X'37' is

00	11	01	rr

where

rr = resolution code as previously described.

The CB_i digit's format for X'38' through X'3B' is

00	11	10	rr

where

rr = resolution code as previously described.

word0, word1, . . . wordn corresponds to control bytes (CB_i) whose values are 1 or X'20' through X'2B', and are either constants or displacement pointers to the RFDFSTK. The format is the same as for DESTINATION.

BA=HA=WA=DA 0 (resolution word)

DESTINATION a displacement pointer to the RFDFSTK.

C= 0 destination is to RFDFSTK.

= 1 DESTINATION is a core expression with the format

FLDSZ	TBP	ADDR

where

FLDSZ the destinations (ADDR) field size (in bits) where ultimate value is to be set.

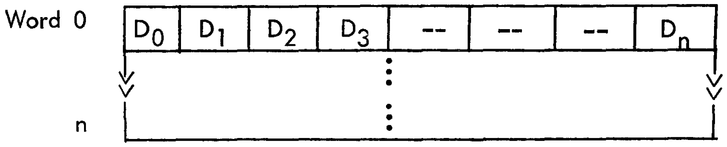
TBP terminal bit position of value in the destination's field.

ADDR word address in SECT.00 of DESTINATION.

E =0 expression not evaluated.

=1 expression has been evaluated.

RELDICT.00 (Generated by SYSGEN, modified by call to MODIFY Subroutine, 6.5)



where

$D_0, D_1, \ldots, D_n$ =a four-bit digit which describes how to relocate a data word in SECT.00.

D_0 implies relative word - 0 in SECT.00

D_1 implies relative word - 1 in SECT.00

⋮

D_n implies relative word - n in SECT.00

Figure 6-8 RELDICT Format

Table 6-1. Relocation Digit Interpretation

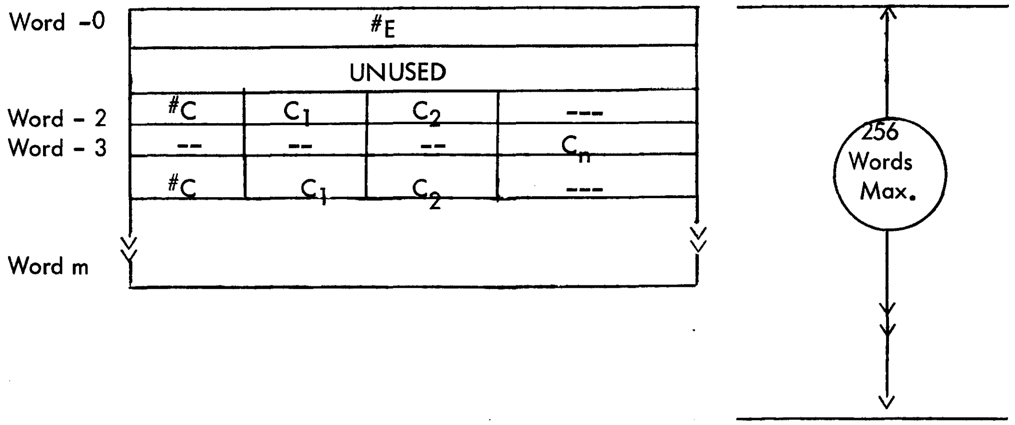
Relocation Digit (D_i)	Part of Word to Relocate	Resolution	Relocate With Request for What Bias
0	Address	Byte	Module Bias
1	Address	Half Word	Module Bias
2	Address	Word	Module Bias
3	Address	Doubleword	Module Bias
A	Both Halves	Doubleword	Module Bias
E	Absolute	-------	-------

SYSGEN uses the above relocation digits with their corresponding implications and results.

SECT.00 (Generated by SYSGEN)

This area contains the actual data, tables, and reserve area which are generated by SYSGEN for a given control command.

6.3 SPEC:HAND FILE CONTENTS



where

#E = number of entries in table.

UNUSED = this word is unused.

#C = number of characters (C_n) in name (TEXTC format).

$C_1, C_2, . . . C_n$ = characters in handler name, (from one to seven characters only). Each name is on a doubleword boundary.

Figure 6-9. SPEC:HAND File Format

6.4 SYSGEN MODIFY SUBROUTINE PARAMETER LISTS (PLISTS)

The SYSGEN MODIFY subroutine requires parameter lists (PLISTS) which describe what is to be accomplished. This procedure is useful only when changing, adding or defining items in a load module. The master PLIST refers to sub-PLISTS which will be referred to as change description tables, (pointed to by Register 7).

Word 0	I	ADDRCDT
1	SUBR	
2	V	TREEAD
3	RFDFAD	EXPRAD
4	RELDICT0AD	SECT0AD
5	RELDICT1AD	SECT1AD
6	RELDICT2AD	SECT2AD
7	RFDFUL	EXPRUL

where

ADDRCDT word address of a change description table or sub-PLIST.

I = 1 a change description table is not supposed to cause expansion of the RFDFSTK or EXPRSTK. Therefore, the EXPR type of change description table is not permitted, and "name 2" of the DEF type change description table must be defined.

I = 0 a change description table may cause an expansion of the RFDFSTK or EXPRSTK. Therefore, it is assumed that the area between the given RFDFSTK and EXPRSTK lines is available for that expansion.

SUBR = 0 no action taken.

SUBR ≠ 0 the address of a subroutine to be entered under the following circumstances. Any call on the MODIFY subroutine may result in a number of core locations in SECT. 0, SECT. 1, or SECT. 2 being modified. If it is so desired that MODIFY not make these changes, but, instead, inform the caller of the changes, then this subroutine is entered. The subroutine is entered via: Set Register 1 = core location to which change applies; Set Register 2 = value to store; Set Register 3 = mask for BAL, 11 SUBR.

V =00 RELDICT. 00, 01, 10 are available.

=01 RELDICT. 00, 01, 10 are not available, (i.e., the load module is an absolute module).

TREEAD =word address of the load module TREE table.

RFDFAD =double word address of RFDFSTK.

EXPRAD =double word address of EXPRSTK.

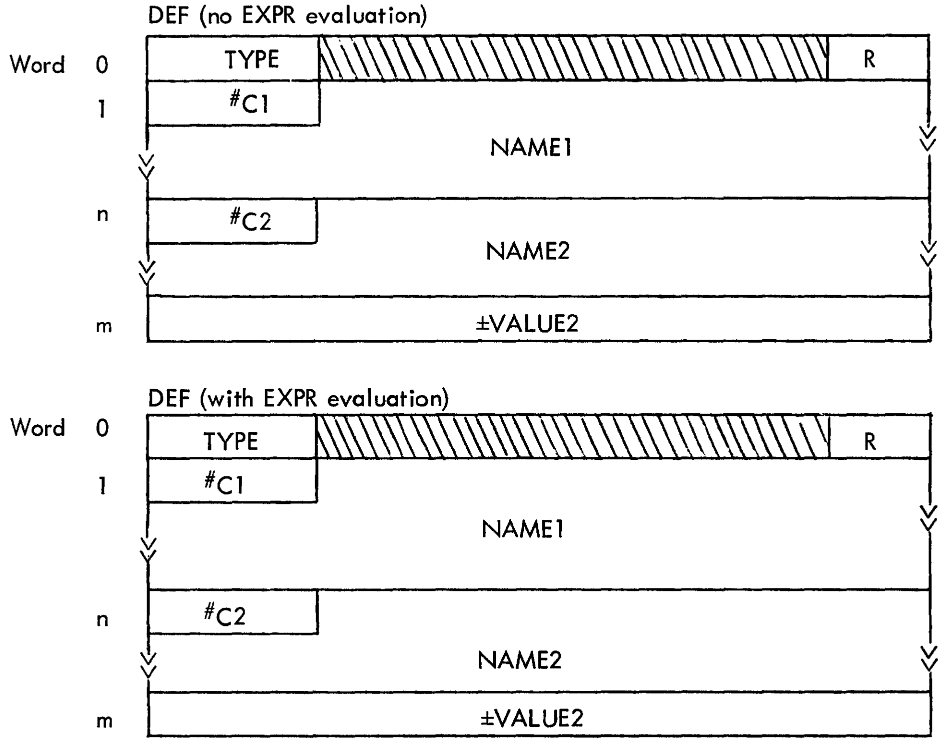
RFDFUL =double word address of RFDFSTK upper limit, (i.e., last useable doubleword).

EXPRUL =double word address of EXPRSTK upper limit, (i.e., last useable doubleword).

RELDICT0AD, RELDICT1AD, RELDICT2AD =doubleword address of the relocation dictionaries for SECT. 00, SECT. 01, SECT. 10 respectively.

SECT0AD, SECT1AD, SECT2AD =doubleword address of the SECT. 00, SECT. 01, SECT. 10 data areas respectively.

Figure 6-10. MASTER PLIST Format



where

R =the resolution of NAME2.*

#C1 =number characters in NAME1.

NAME1 =the name of the external definition.

#C2 =number of characters in NAME2. If = 0, no NAME2, thus, NAME1 is absolute.

NAME2 =the name of an external definition or reference upon which NAME1's value depends.

±VALUE2 =a positive or negative displacement from NAME2, or the absolute value to which NAME1 is equated (if #C2 = 0).

TYPE = 01 definition with no expression

TYPE = 04 definition with expression

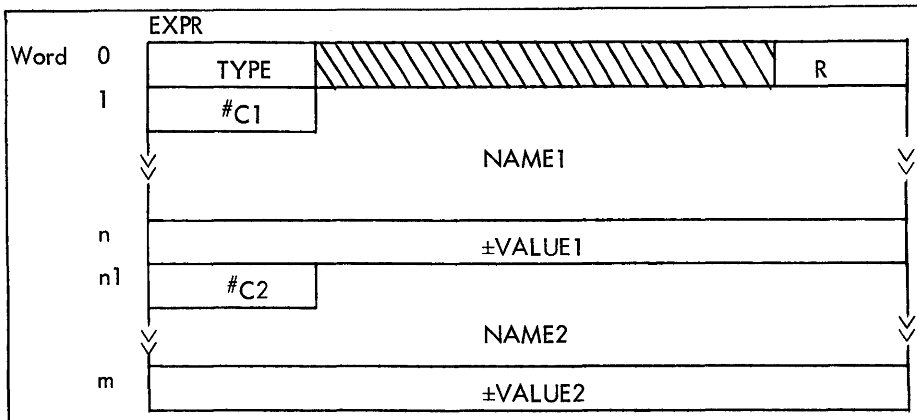
*The resolution codes ("R") consist of: 0 = byte address, 1 = halfword address, 2 = word address, 3 = doubleword address, and 4 = same resolution as NAME1.

Figure 6-11. CHANGE DESCRIPTION TABLE (SUBPLIST) Format for DEF

These two change description tables both generate an external definition. In the first case TYPE = 01, no EXPRSTK entry is made which evaluates the definition. However, in the second case TYPE = 04, an EXPRSTK entry is made which generates an actual value for a DEF.

The interpretation of these tables is:

```
NAME1    EQU    resolution (NAME2) ±VALUE2
NAME1    EQU    resolution (NAME2)
NAME1    EQU    ±VALUE2
```



where

R =the resolution of NAME2.

#C1 =number of characters in NAME1.

NAME1 =the name of an external definition for the destination.

±VALUE1 =a positive or negative displacement from NAME1 for the destination.

#C2 =number of characters in NAME2.

NAME2 =the name of an external definition or reference for which a value is to be evaluated.

±VALUE2 =a positive or negative added used to modify NAME2's value.

TYPE = 00

Figure 6-12. CHANGE DESCRIPTION TABLE (SUBPLIST) Format for REF

The interpretation of this table is :

Destination NAME1±VALUE1 will be set with the resolution (NAME2)±VALUE2 when NAME2 is defined.

The format of VALUE1 is:

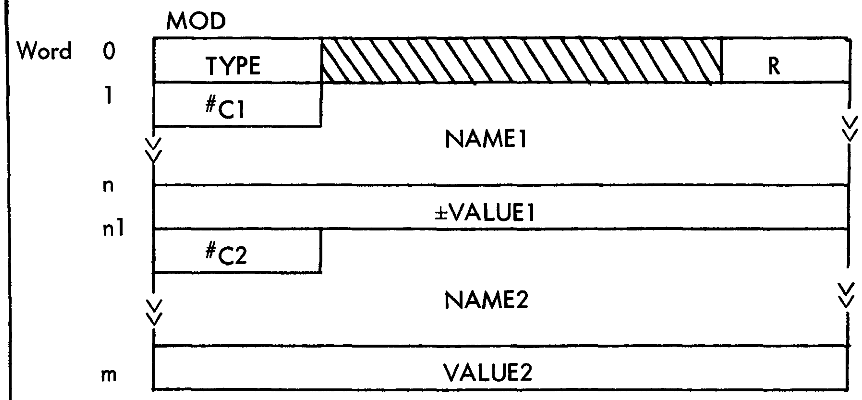
FLDSZ	TBP	ADDR

where

FLDSZ =the destination (NAME1±ADDR) field size (in bits) where ultimate value is to be set.

TBP =terminal bit position of value in the destination field.

ADDR =relative word address in SECT.00,01,10 of destination (relative to NAME1).



where

R =the resolution of NAME2.

#C1 =number of characters in NAME1.

NAME1 =the name of an external definition for the destination.

±VALUE1 =a positive or negative displacement from NAME1 for the destination.

#C2 =the number of characters in NAME2.

NAME2 =the name of an external definition used in relocating VALUE2.

VALUE2 =value to be put into cell NAME1±VALUE1.

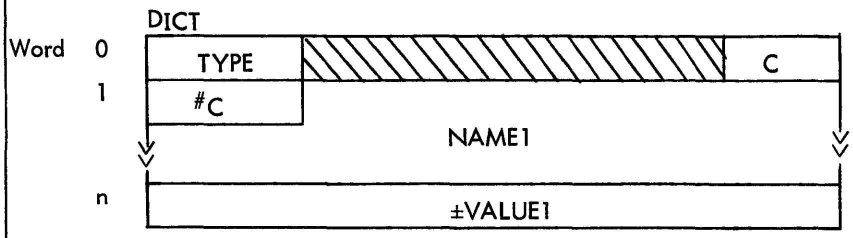
TYPE = 02

Figure 6-13. Change Description Table (SUBPLIST) Format for Sect. Modification

The interpretation of this table is:

Destination NAME±VALUE1 will be replaced with the ±VALUE2 +resolution (NAME2) (i.e., if resolution is byte, then bits 13-31 in VALUE2 will be relocated as a byte address according to the defined value of NAME2).

The effective address, NAME1±VALUE1, within the load module, determines whether the modification is to SECT.00,01, or 10.



where

C =relocation code to be used to modify the relocation dictionary

#C =number of an external definition.

±VALUE1 =a positive or negative displacement from NAME1.

TYPE = 03

Figure 6-14. Change Description Table (SUBPLIST) Format for RELDICT. Modification

The interpretation of this table is:

Change the code in the Relocation Dictionary, whose relative position corresponds to the relative address {(NAME1±VALUE1) - module base address }, to the code "C".

The effective address, NAME1±VALUE1, within the load module, determines whether the modification is to RELDICT.00, 01, or 10

6.5 MODIFY

6.5.1 Purpose

To accept parameters via parameter lists and to generate external definition and reference entries in a load module's RFDFSTK and EXPRSTK, respectively, and to modify a load module's relocation dictionary (i.e., RELDICT.00, 01, 10) or data section (i.e., SECT.00,01,10).

6.5.2 Calling Sequence

BAL,SR4 MODIFY

6.5.3 Input

Register 7 = word address of MASTER PLIST

6.5.4 Output

Condition code one (CC1) set if generation of external definition or reference or modification of relocation dictionary or data section is unsuccessful, or the change description table TYPE code is unknown.

Condition code one (CC1) reset if generation or modification is successful.

6.5.5 Subroutines

LOC - Evaluate Location

INPUT: Register 13 = address of $NAME_1$ in Sub-PLIST minus 1.

CALL: BAL,11 LOC

RETURN: Register 13 = address of $NAME_2$ in Sub-PLIST minus 1.
Register 4 = core address of designated location, i.e., $NAME_1 \pm VALUE_1$.
Register 0 = Section number of SECT.00,01, 10 in load module where core address is located.

SECTION - Compute Section Number

INPUT: Register 13 = address of $VALUE_i$ in Sub-PLIST.
Register 4 = core address for which Section Number is to be obtained.

CALL: BAL,11 SECTION

RETURN: Register 0 = Section Number of SECT.00,01 or 10 in load module where core address is located.

RSEARCH - RFDFSTK Search

INPUT: Register 13 = address in Sub-PLIST of $NAME_i$.

CALL: BAL,11 RSEARCH

RETURN+0: $NAME_i$ was not found in RFDFSTK.
Register 13 = same as input.
Register 3 = address of next available RFDFSTK entry
Register 12 = index to next available RFDFSTK entry.

RETURN+1: $NAME_i$ was found in RFDFSTK.
Register 13 = same as input.
Register 3 = address of name in RFDFSTK.
Register 12 = index to next available RFDFSTK entry.

ADJUST - Adjust Value

INPUT: Register 14 = value from RFDFSTK entry to be adjusted to desired resolution.

Register 15 = Resolution word from RFDFSTK entry,

BA	HA	WA	DA

Register 10 = Resolution desired from Sub-PLIST.

CALL: BAL,11 ADJUST

RETURN: Register 14 = value from RFDFSTK entry adjusted to desired resolution.

Register 15 = resolution word adjusted to indicate desired resolution.

Register 10 = resolution code of resolved value in Register 14.

resolution code BA = 0, HA = 1, WA = 2, DA = 3

EVAL - Evaluate Expression

INPUT: Register 13 = address of NAME$_i$ in Sub-PLIST minus 1.

Register 10 = desired Resolution code.

CALL: BAL,11 EVAL

RETURN+0: NAME$_i$ not found in RFDFSTK.

Registers 10 and 13 are the same as for RETURN+1.

Register 3 = address of next available RFDFSTK entry.

Register 12 = index to next available RFDFSTK entry.

RETURN+1: NAME$_i$ was found in RFDFSTK as a Ref.

Register 13 = address of NAME$_i$ in Sub-PLIST

Register 10 = resolution code of desired value.

Register 3 = address of RFDFSTK entry.

Register 12 = index to RFDFSTK entry.

RETURN+2: Expression for NAME$_i$ ±VALUE$_i$ evaluated.

Register 13 = address of next NAME$_i$ in Sub-PLIST minus 1.

Register 10 = resolution code of final value in Register 14.

Register 12 = index to RFDFSTK entry.

= -1 no NAME$_i$.

Register 15 = resolution word of final value in Register 14.

Register 14 = evaluated expression value, NAME$_i$ ±VALUE$_i$.

EVALR - Evaluate Expression

INPUT: Same as EVAL, except,

Register 10 = desired resolution code from Sub-PLIST.

CALL: BAL,11 EVALR

RETURNS: Same as EVAL.

$\left.\begin{array}{l}\text{RADD}\\ \text{DADD}\end{array}\right\}$ - Add RFDFSTK item (REF/DEF)

INPUT: Register 3 = address of next available RFDFSTK entry.
Register 13 = address of NAME_i in Sub-PLIST.

CALL: BAL, 11 RADD

CALL: BAL, 11 DADD

RETURN: Register 3 = address in RFDFSTK of current entry.
Register 12 = index to RFDFSTK entry.
Register 13 = address of NAME_i in sub-PLIST.

EADD - Add EXPRSTK item

INPUT: Register 9 = code = 0 Expression for a DEF.
code = 4 Expression for a REF, i.e., core destination.
Register 8 = value of $\text{NAME}_i \pm \text{VALUE}_i$.
Register 13 = address of NAME_i in Sub-PLIST.
Register 12 = Index to next available entry in RFDFSTK if Register 10 is < 4.
Register 10 = Resolution code for a REF expression.

CALL: BAL, 11 EADD

CHCORE - Change Core Location or EXPRSTK

INPUT: Register 4 = Destination information word, i.e., address field size and terminal bit position of address field and destination address.

FLDSZ	TBP	ADD
1 7	8 15	16 31

Register 0 = Section number of SECT. 00, 01, 10 in load module where destination address is found.
Register 14 = the value to be stored in destination.
Register 15 = mask for masking value into destination.
Register 10 = resolution code, 0 = BA, 1 = HA, 2 = WA, 3 = DA, 4 = NONE, for EXP or MOD type Sub-PLIST, or the resolution code plus bit 0 set = 1 if DICT type Sub-PLIST.

CALL: BAL, 11 CHCORE

EDEST - EXPRSTK Destination Address

INPUT: Register 4 = desired destination address for searching purposes.

CALL: BAL, 11 EDEST

RETURN: All EXPRSTK items having a destination (DEST) address equivalent to the desired destination address in Register 4, are removed from EXPRSTK by setting Bit 8 in word 0 of EXPRSTK entry to 1.

EFIX - Fix EXPRSTK for Unsatisfied REF

CALL: BAL,11 EFIX

RINDEX

INPUT: Register 13 = RFDFSTK entry index

CALL: BAL,11 RINDEX

RETURN: Register 4 = RFDFSTK entry address

6.5.6 Description

The MODIFY routine consits of various subroutines which interpret input parameter lists (PLISTS), evaluate their designated expression, (e.g., $NAME_1 \pm VALUE_1$), generate RFDFSTK (external reference and definition stack) EXPRSTK (expression stack defining a ref/def expression value) entries, and modify core, RFDFSTK, relocation dictionary, EXPRSTK values.

When an expression (EXPR) PLIST is encountered, NAME ± VALUE is evaluated and is the destination address used when the expression $NAME_2 \pm VALUE_2$ is satisfied (i.e., evaluated). If there is no $NAME_1$, $VALUE_1$ then becomes the destination. The expression, $NAME_2 \pm VALUE_2$, is then interrogated. If $NAME_2$ is not in the RFDFSTK, it is added to it as a ref type and an EXPRSTK entry is made which expresses the evaluation algorithm for $NAME_2$. If $NAME_2$ is in the RFDFSTK and is a ref type, an EXPRSTK entry is made which expresses the evaluation algorithm for $NAME_2$. However, if $NAME_2$ is in the RFDFSTK and is a def type, the expression, $NAME_2 \pm VALUE_2$, is evaluated. The resulting value is then used to replace the field in the destination address which is masked according to the destination resolution. That is, if the destination resolution is a byte value, then the evaluation of $NAME_2 \pm VALUE_2$ is converted to a byte value and replaces the designated field in the destination. Each entry in the EXPRSTK is then checked to see if its destination address is the same as the destination address of $NAME_1 \pm VALUE_1$, and if so, the EXPRSTK entry is removed by setting flag (E = 1) indicating that this expression has been evaluated. If there is no $NAME_2$ (i.e., only a $\pm VALUE_2$), then the result is the same as if there was a $NAME_2$ defined in the RFDFSTK with value 0.

When a definition (DEF,TYPE = 01) PLIST is encountered, $NAME_1$ is entered into the RFDFSTK as a reference type unless it is already in the RFDFSTK. The expression, $NAME_2 + VALUE_2$ is then interrogated. If $NAME_2$ is not in the RFDFSTK, it is added to it as a ref type and an EXPRSTK entry is made which expresses the evaluation algorithm for $NAME_2$. However, if $NAME_2$ is in the RFDFSTK and is a def type, the expression, $NAME_2 \pm VALUE_2$, is evaluated. The result is then put into the RFDFSTK entry and the entry is flagged as a def. When $NAME_2$ is a reference, then the RFDSTK entry remains as a ref type. The EXPRSTK is then searched to find all entries which reference this definition. Each satisfied reference is then evaluated. The resulting value is then used to replace the field identified by the destination and is masked according to the destination resolution. That is, if the destination resolution is a double word value, then the resulting value is converted to a double word value and replaces the designated field in the destination. If there is no $NAME_2$ (i.e., only a ±$VALUE_2$) then the result is the same as if there was a $NAME_2$ with value 0.

When a definition (DEF, TYPE = 04) PLIST is encountered, NAME is entered into the RFDFSTK as a definition type (unlike that for a definition of TYPE = 0) unless it is already in the RFDFSTK. The expression, $NAME_2 \pm VALUE_2$ is then interrogated. If $NAME_2$ is not in the RFDFSTK, it is added to it as a ref type and an EXPRSTK entry is made which expresses the evaluation algorithm for $NAME_2$. If $NAME_2$ is in the RFDFSTK and is a ref type, an EXPRSTK entry is made which expresses the evaluation algorithm for $NAME_2$. However, if $NAME_2$ is in the RFDFSTK as a def type or if there is no $NAME_2$ (i.e., only a $\pm VALUE_2$), the expression, $NAME_2 \pm VALUE_2$, is evaluated. The result is then put into the RFDFSTK entry for $NAME_1$.

When a core modification (MOD) PLIST is encountered, $NAME_1 \pm VALUE_1$ is evaluated and is the destination address word when the expression $NAME_2 \pm VALUE_2$ is satisfied (i.e., evaluated). If there is no $NAME_1$, $VALUE_1$ then becomes the destination. The expression, $NAME_2 \pm VALUE_2$, is then interrogated. If there is a $NAME_2$, it must have been previously defined with complete evaluation. If there is no $NAME_2$, (i.e., only a $\pm VALUE_2$), then the result of the expression evaluation is the same as if there was a $NAME_2$ whose value is known as 0. Once the expression, $NAME_2 \pm VALUE_2$, has been evaluated, the entire 32 bit value replaces the contents of the destination address. The EXPRSTK is then checked to see if any of its entries contain a destination address equivalent to the destination address, $NAME_1 \pm VALUE_1$, and if so, each such EXPRSTK entry is removed by setting a flag (E = 1) indicating that this expression has been evaluated.

When a relocation dictionary modification (DICT) PLIST is encountered, $NAME_1 \pm VALUE_1$ is evaluated and represents the destination address for which a relocation dictionary modification is desired. This type of PLIST will normally accompany a MOD PLIST. That is, if a core modification is made, its corresponding relocation dictionary entry may also require a resolution code change. The address $NAME_1 \pm VALUE_1$ relative to the section base, becomes the relative digit position (4 bits per digit) in the section relocation dictionary.

When MODIFY finishes processing a PLIST successfully, condition code one (CC1) is reset (i.e., CC1 = 0) and a return is made to the caller at the BAL plus one. However, if MODIFY finds any error condition, it sets condition code one (i.e., CC1 = 1) and a return is made to the caller at the BAL plus one.

The following conditions will cause an error return (i.e., CC1 = 1);

1. TYPE> 4 (sub-PLIST type invalid).
2. No NAME$_1$ in DEF PLIST (TYPE = 01).
3. No NAME$_1$ in DEF PLIST (TYPE = 04).
4. NAME$_2$ in MOD PLIST is not defined or is a reference.
5. NAME$_1$ in EXPR, MOD, or DICT PLIST is not defined or is a reference.
6. The expression value (NAME$_1$ $\pm$ VALUE$_1$) does not fit into any of the sections of a load module, (i.e., SECT.00,01, or 10).
7. The MASTER PLIST indicates (I = 1) that the RFDFSTK and EXPRSTK cannot be expanded and a DEF (TYPE = 01 or 04) or EXPR PLIST is encountered which would require expansion of either the RFDFSTK or EXPRSTK or both.
8. An EXPR, or MOD PLIST requests a core modification whose field size (FLDSZ in sub-PLIST) is either greater than 32 bits or overlaps a word boundary.

6.5.7 Flowcharts

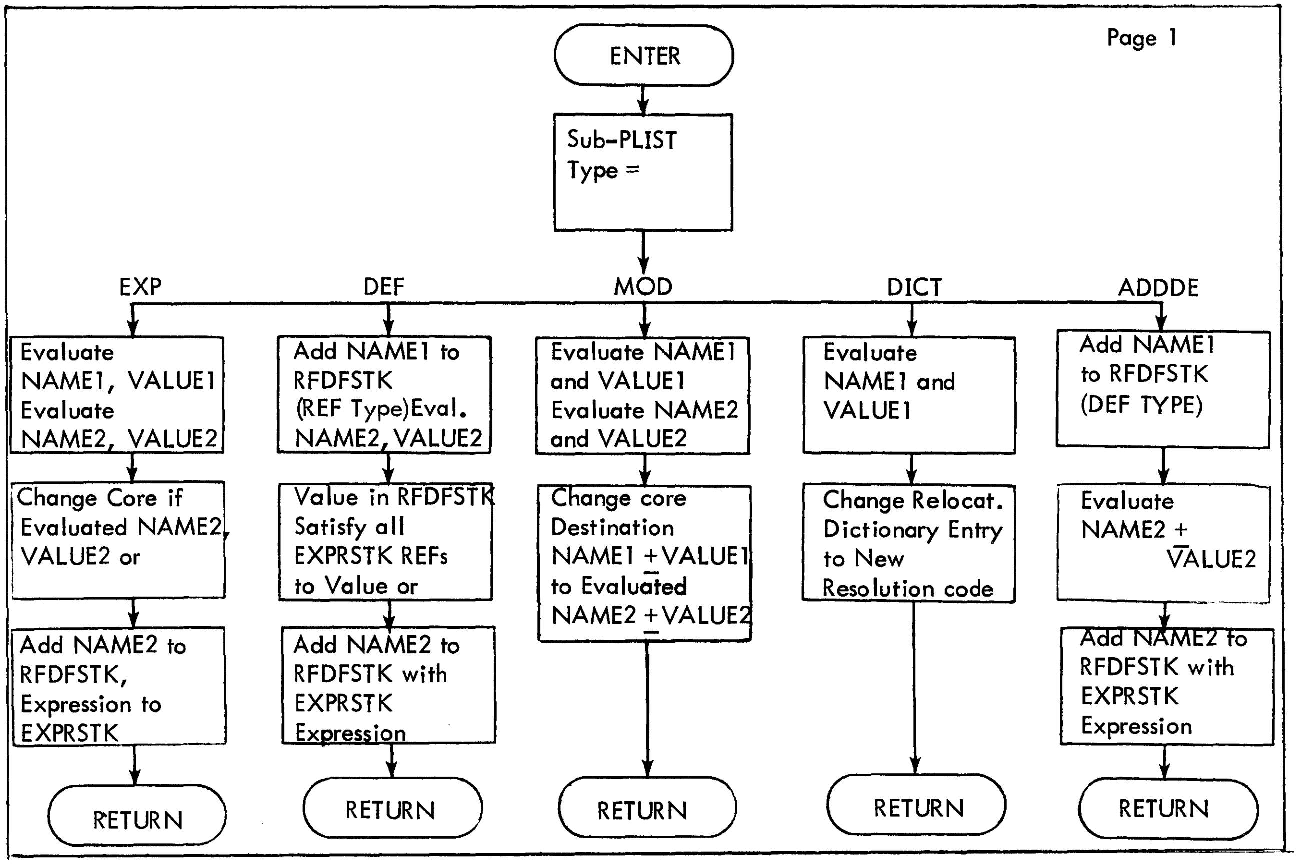

Page 1
ENTER
Sub-PLIST Type =
EXP
DEF
MOD
DICT
ADDDE
Evaluate NAME1, VALUE1 Evaluate NAME2, VALUE2
Add NAME1 to RFDFSTK (REF Type)Eval. NAME2, VALUE2
Evaluate NAME1 and VALUE1 Evaluate NAME2 and VALUE2
Evaluate NAME1 and VALUE1
Add NAME1 to RFDFSTK (DEF TYPE)
Change Core if Evaluated NAME2, VALUE2 or
Value in RFDFSTK Satisfy all EXPRSTK REFs to Value or
Change core Destination NAME1 ±VALUE1 to Evaluated NAME2 ±VALUE2
Change Relocat. Dictionary Entry to New Resolution code
Evaluate NAME2 + VALUE2
Add NAME2 to RFDFSTK, Expression to EXPRSTK
Add NAME2 to RFDFSTK with EXPRSTK Expression
Add NAME2 to RFDFSTK with EXPRSTK Expression
RETURN
RETURN
RETURN
RETURN
RETURN

Figure 6-15. Flow Diagram of MODIFY
(General Flow)

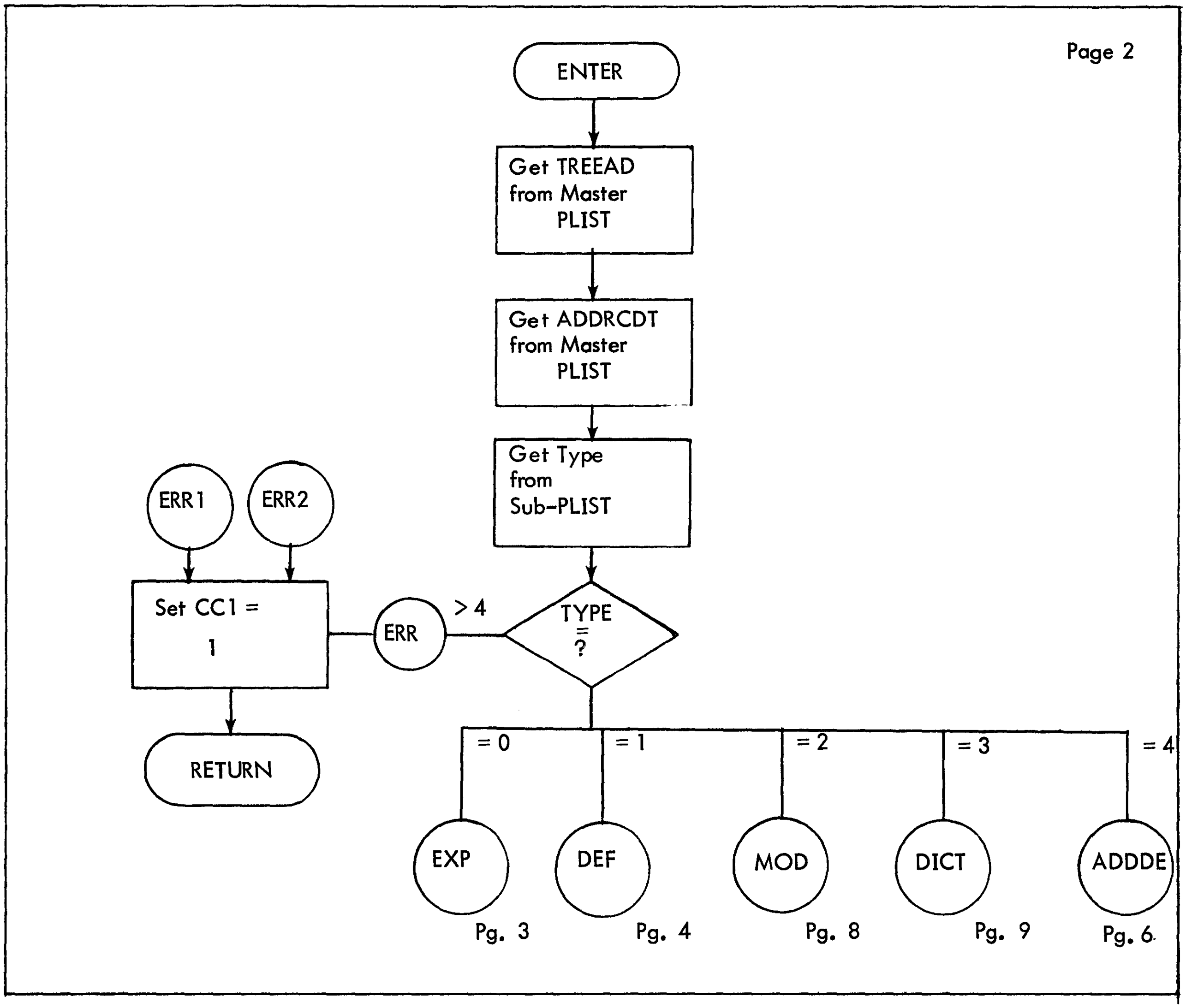


Figure 6-15. Flow Diagram of MODIFY (Cont.)
(Entry Point)

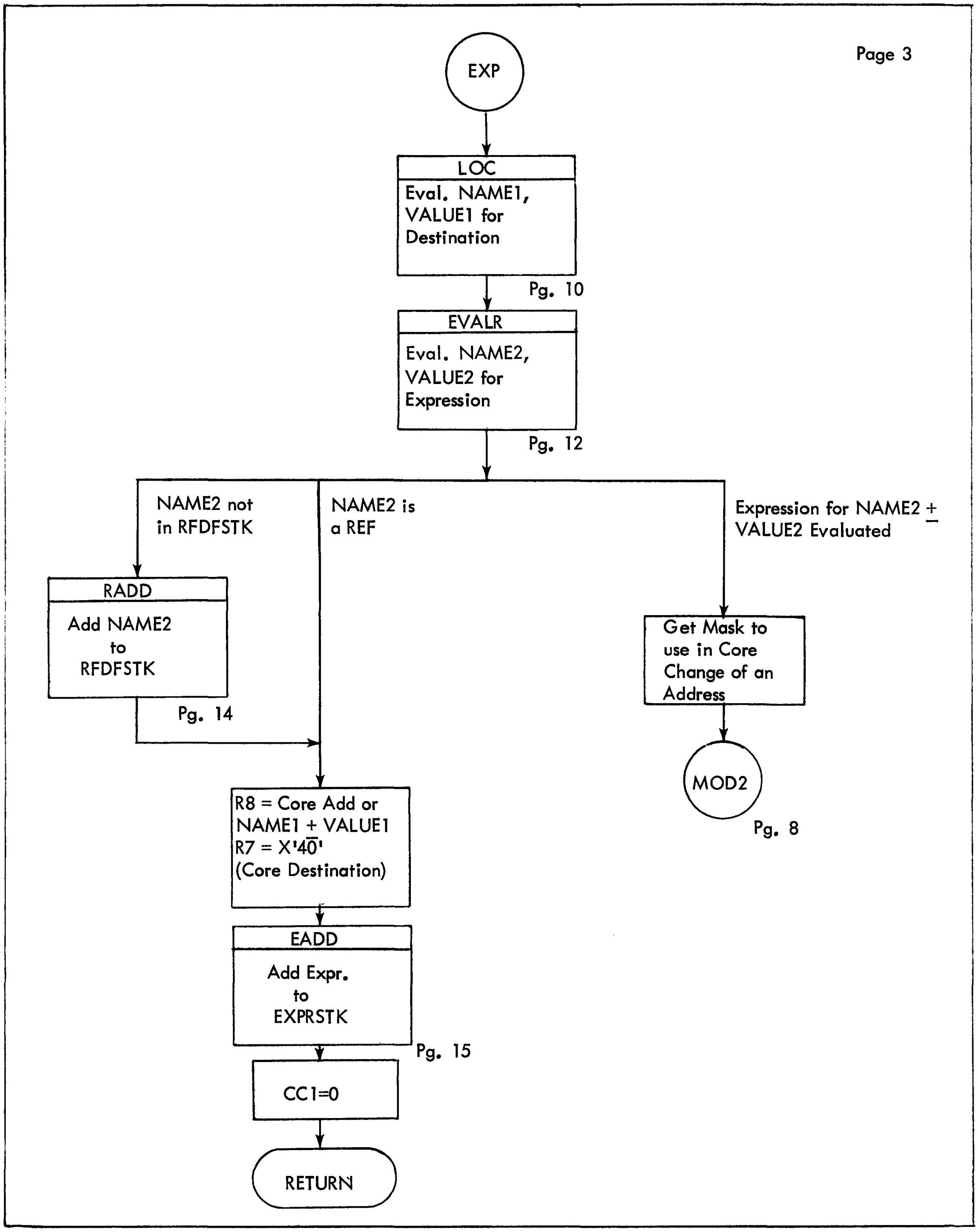

Page 3
EXP
LOC
Eval. NAME1, VALUE1 for Destination
Pg. 10
EVALR
Eval. NAME2, VALUE2 for Expression
Pg. 12
NAME2 not in RFDFSTK
NAME2 is a REF
Expression for NAME2 ± VALUE2 Evaluated
RADD
Add NAME2 to RFDFSTK
Pg. 14
Get Mask to use in Core Change of an Address
MOD2
Pg. 8
R8 = Core Add or NAME1 + VALUE1
R7 = X'4̄0'
(Core Destination)
EADD
Add Expr. to EXPRSTK
Pg. 15
CC1=0
RETURN

Figure 6-15. Flow Diagram of MODIFY (Cont.)
(Process EXPR SubPLIST)

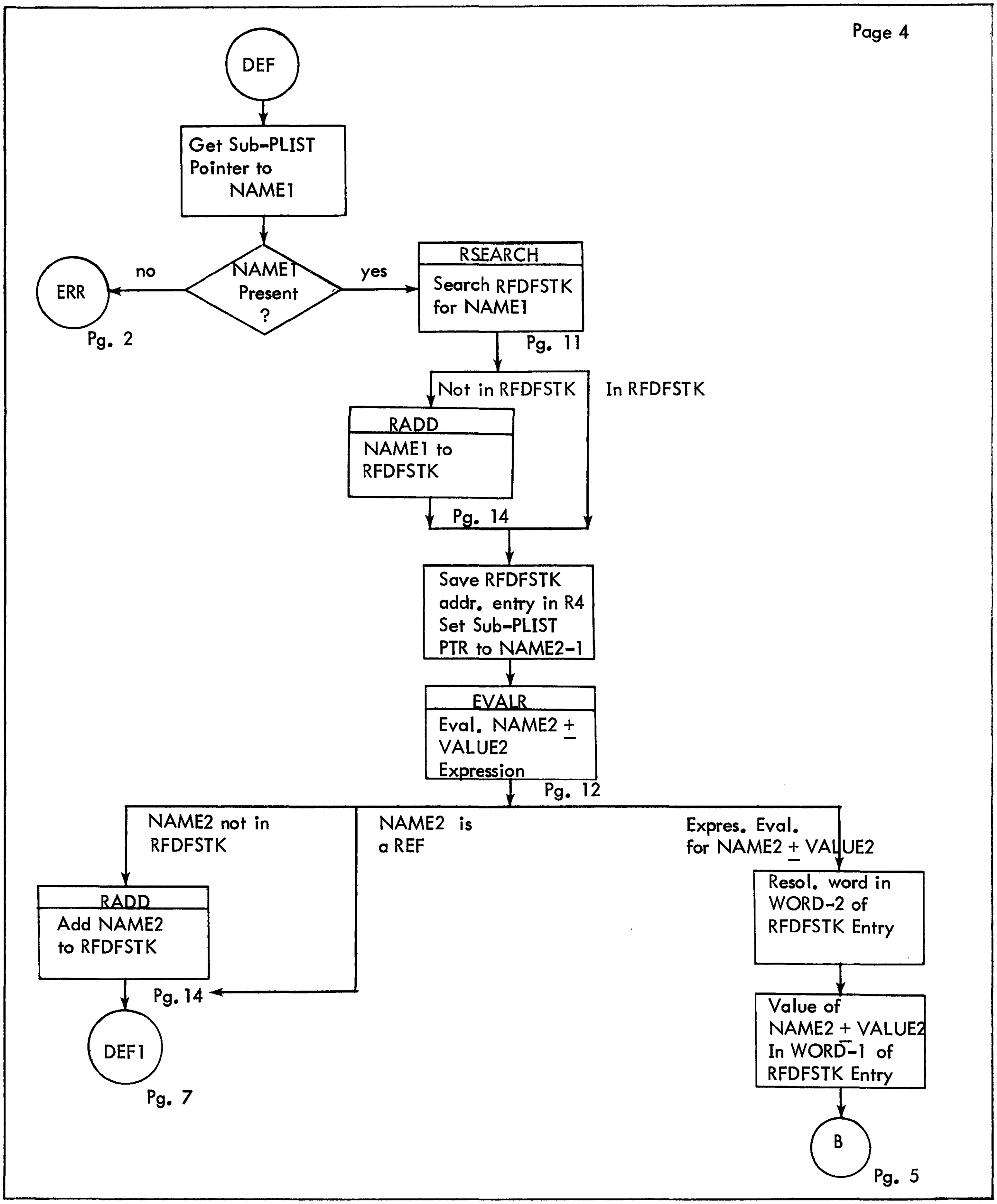

Page 4
DEF
Get Sub-PLIST Pointer to NAME1
NAME1 Present ?
no
ERR
Pg. 2
yes
RSEARCH
Search RFDFSTK for NAME1
Pg. 11
Not in RFDFSTK
In RFDFSTK
RADD
NAME1 to RFDFSTK
Pg. 14
Save RFDFSTK addr. entry in R4 Set Sub-PLIST PTR to NAME2-1
EVALR
Eval. NAME2 ± VALUE2 Expression
Pg. 12
NAME2 not in RFDFSTK
NAME2 is a REF
Expres. Eval. for NAME2 ± VALUE2
RADD
Add NAME2 to RFDFSTK
Pg. 14
DEF1
Pg. 7
Resol. word in WORD-2 of RFDFSTK Entry
Value of NAME2 ± VALUE2 In WORD-1 of RFDFSTK Entry
B
Pg. 5

Figure 6-15. Flow Diagram of MODIFY (Cont.)
(Process DEF SubPLIST - No Expression of DEF)

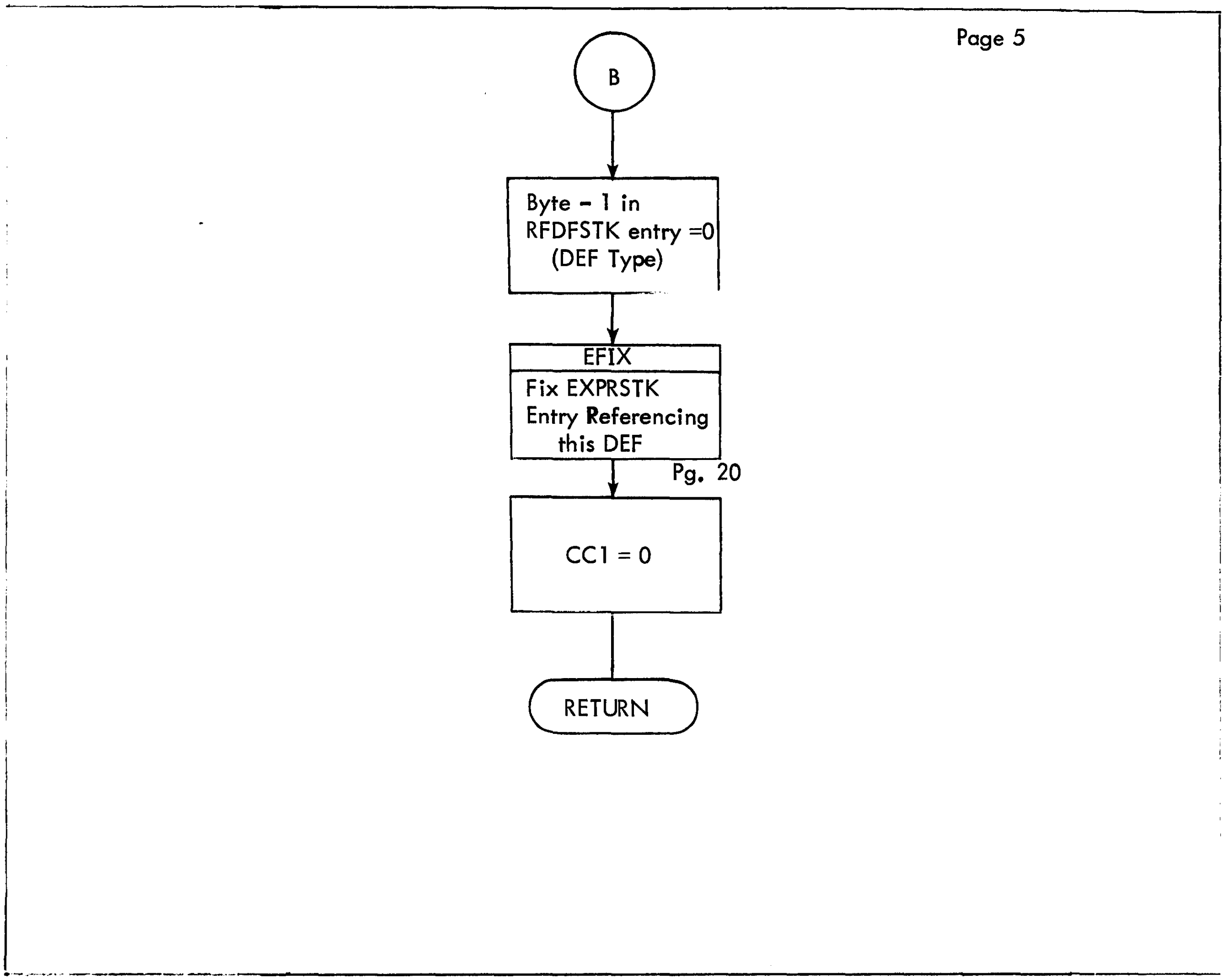
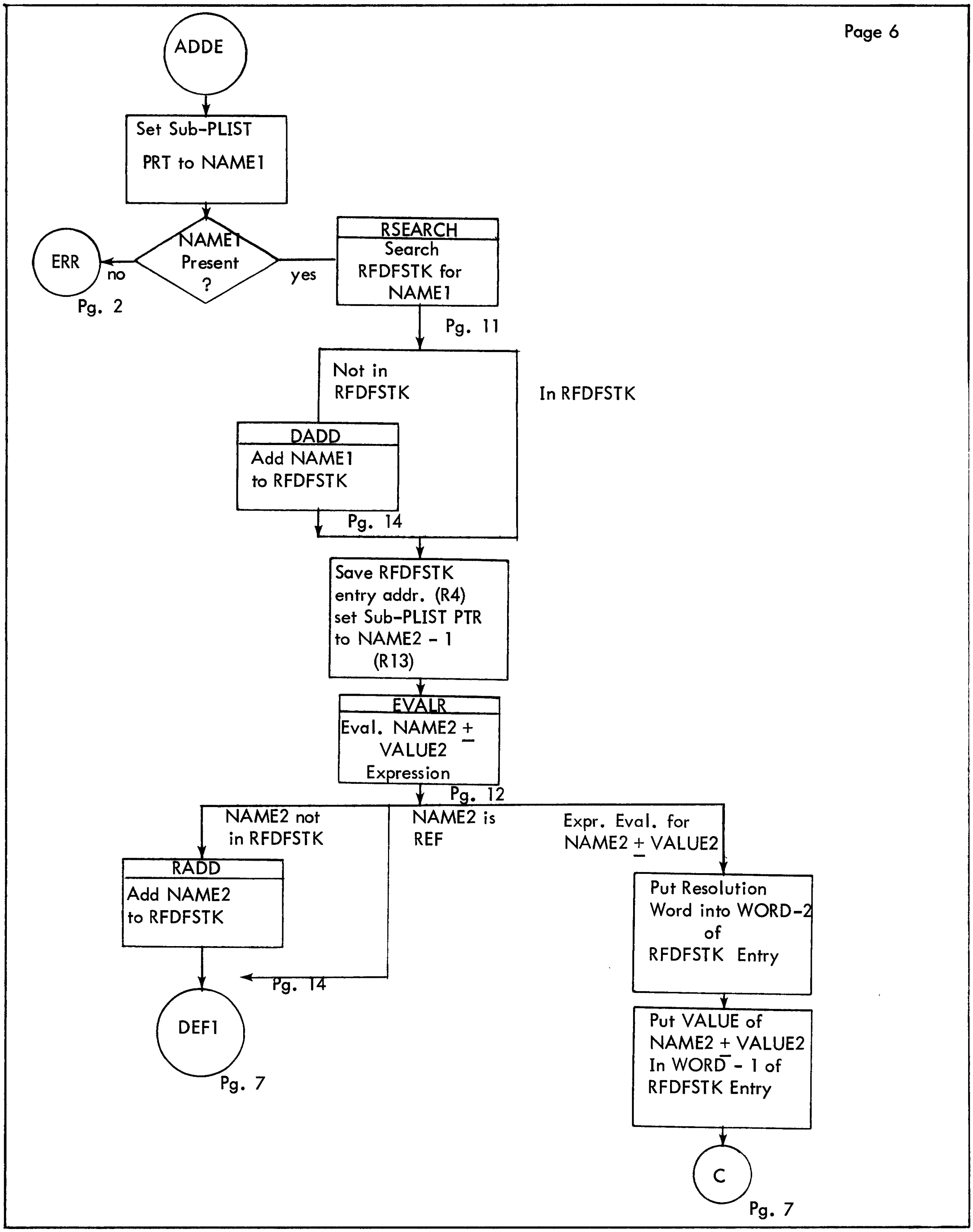

Page 5
B
Byte - 1 in RFDFSTK entry =0 (DEF Type)
EFIX
Fix EXPRSTK Entry Referencing this DEF
Pg. 20
CC1 = 0
RETURN

Figure 6-15. Flow Diagram of MODIFY (Cont.)
(Process DEF SubPLIST - No Expression of DEF)



Page 6
ADDE
Set Sub-PLIST PRT to NAME1
NAME1 Present ?
no
ERR
Pg. 2
yes
RSEARCH
Search RFDFSTK for NAME1
Pg. 11
Not in RFDFSTK
In RFDFSTK
DADD
Add NAME1 to RFDFSTK
Pg. 14
Save RFDFSTK entry addr. (R4) set Sub-PLIST PTR to NAME2 - 1 (R13)
EVALR
Eval. NAME2 ± VALUE2 Expression
Pg. 12
NAME2 not in RFDFSTK
NAME2 is REF
Expr. Eval. for NAME2 ± VALUE2
RADD
Add NAME2 to RFDFSTK
Pg. 14
DEF1
Pg. 7
Put Resolution Word into WORD-2 of RFDFSTK Entry
Put VALUE of NAME2 + VALUE2 In WORD - 1 of RFDFSTK Entry
C
Pg. 7

Figure 6-15. Flow Diagram of MODIFY (Cont.)
(Process DEF SubPLIST, Build Expression of DEF)

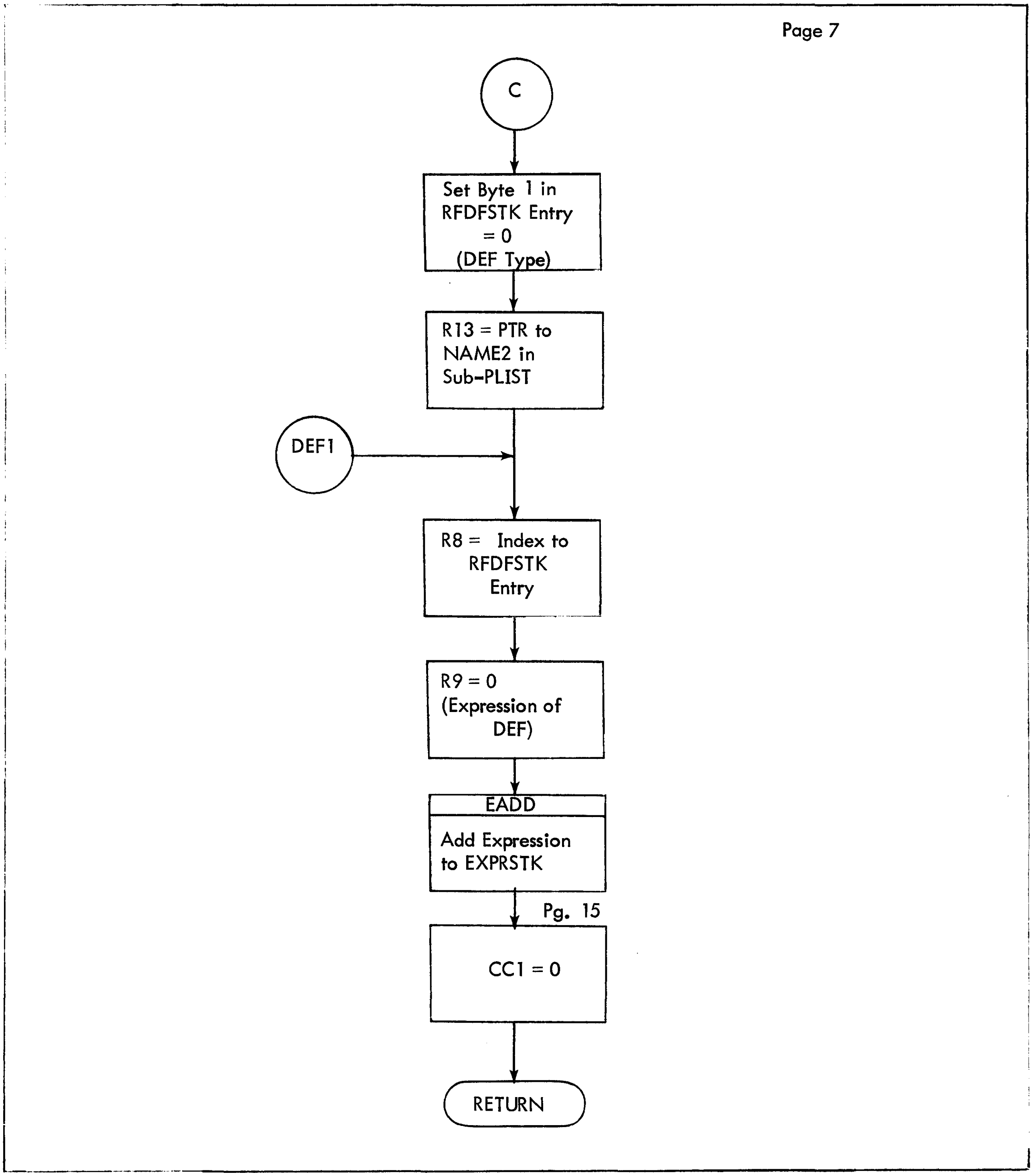

Page 7
C
Set Byte 1 in RFDFSTK Entry = 0 (DEF Type)
R13 = PTR to NAME2 in Sub-PLIST
DEF1
R8 = Index to RFDFSTK Entry
R9 = 0 (Expression of DEF)
EADD
Add Expression to EXPRSTK
Pg. 15
CC1 = 0
RETURN

Figure 6-15. Flow Diagram of MODIFY (Cont.)
(Process DEF SubPLIST, Build Expression of DEF)

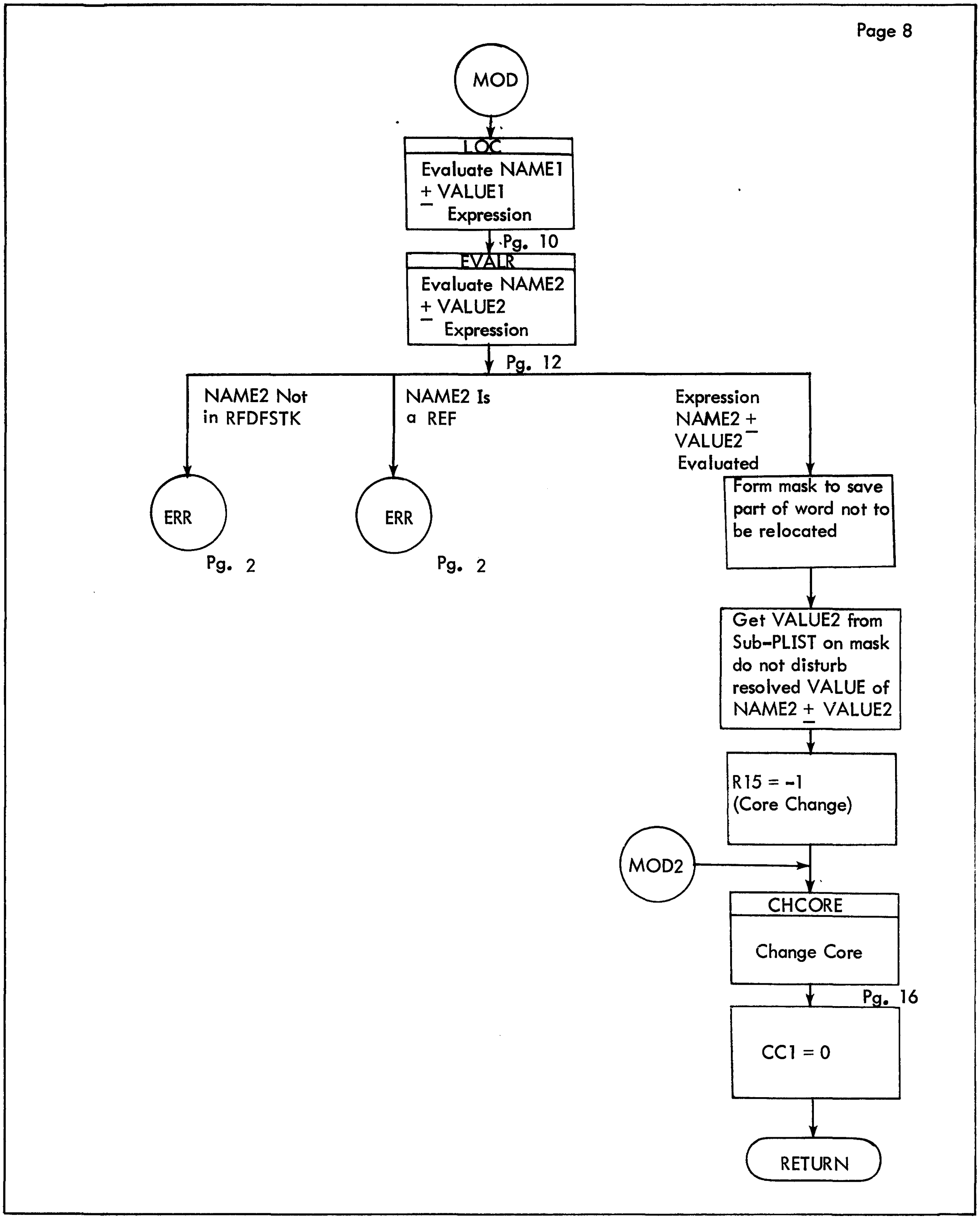

Page 8
MOD
LOC
Evaluate NAME1 ± VALUE1 Expression
Pg. 10
EVALR
Evaluate NAME2 ± VALUE2 Expression
Pg. 12
NAME2 Not in RFDFSTK
NAME2 Is a REF
Expression NAME2 ± VALUE2 Evaluated
ERR
Pg. 2
ERR
Pg. 2
Form mask to save part of word not to be relocated
Get VALUE2 from Sub-PLIST on mask do not disturb resolved VALUE of NAME2 ± VALUE2
R15 = -1 (Core Change)
MOD2
CHCORE
Change Core
Pg. 16
CC1 = 0
RETURN

Figure 6-15. Flow Diagram of MODIFY (Cont.)
(Process MOD SubPLIST)

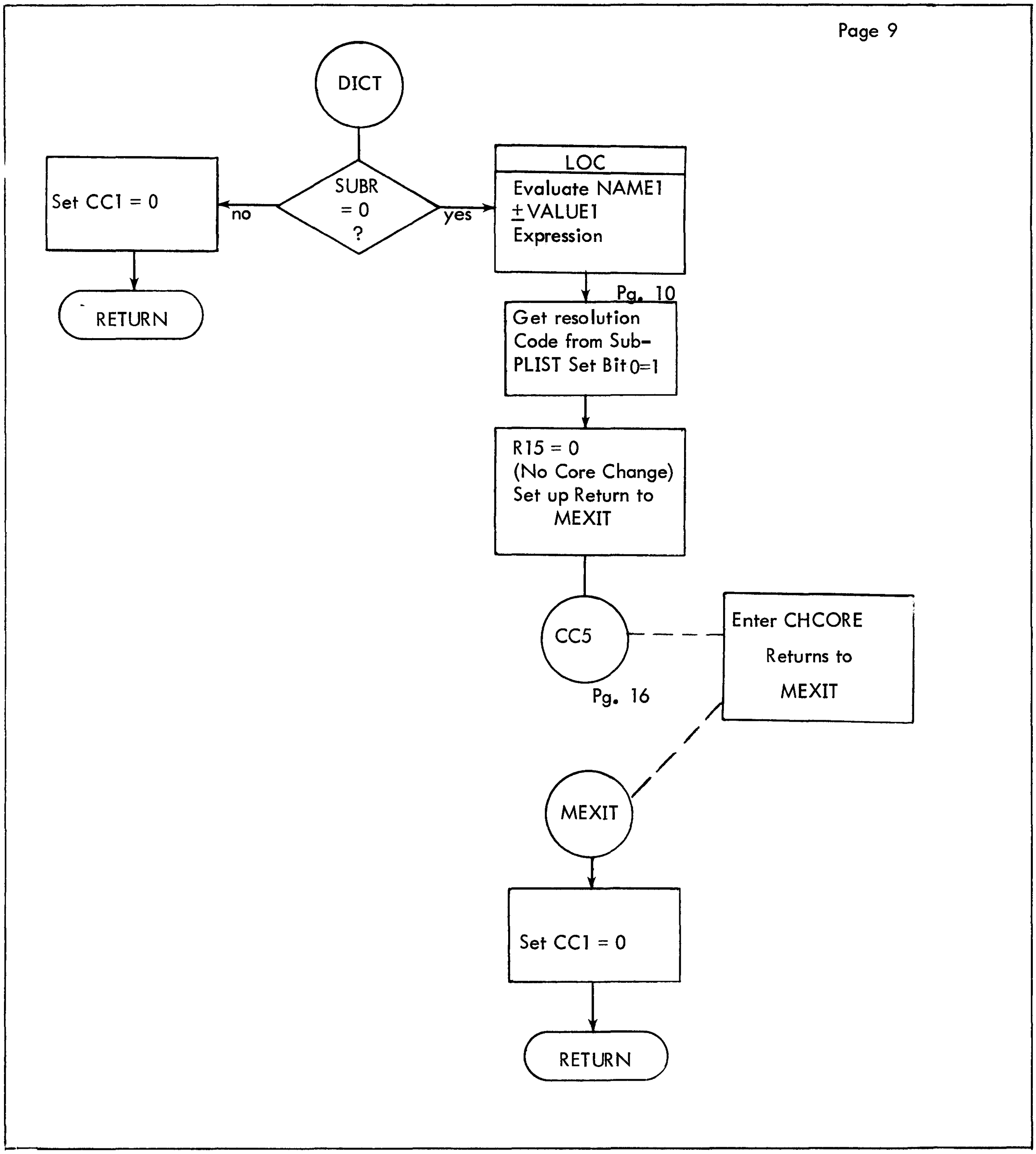


Figure 6-15. Flow Diagram of MODIFY (Cont.)
(Process DICT SubPLIST)

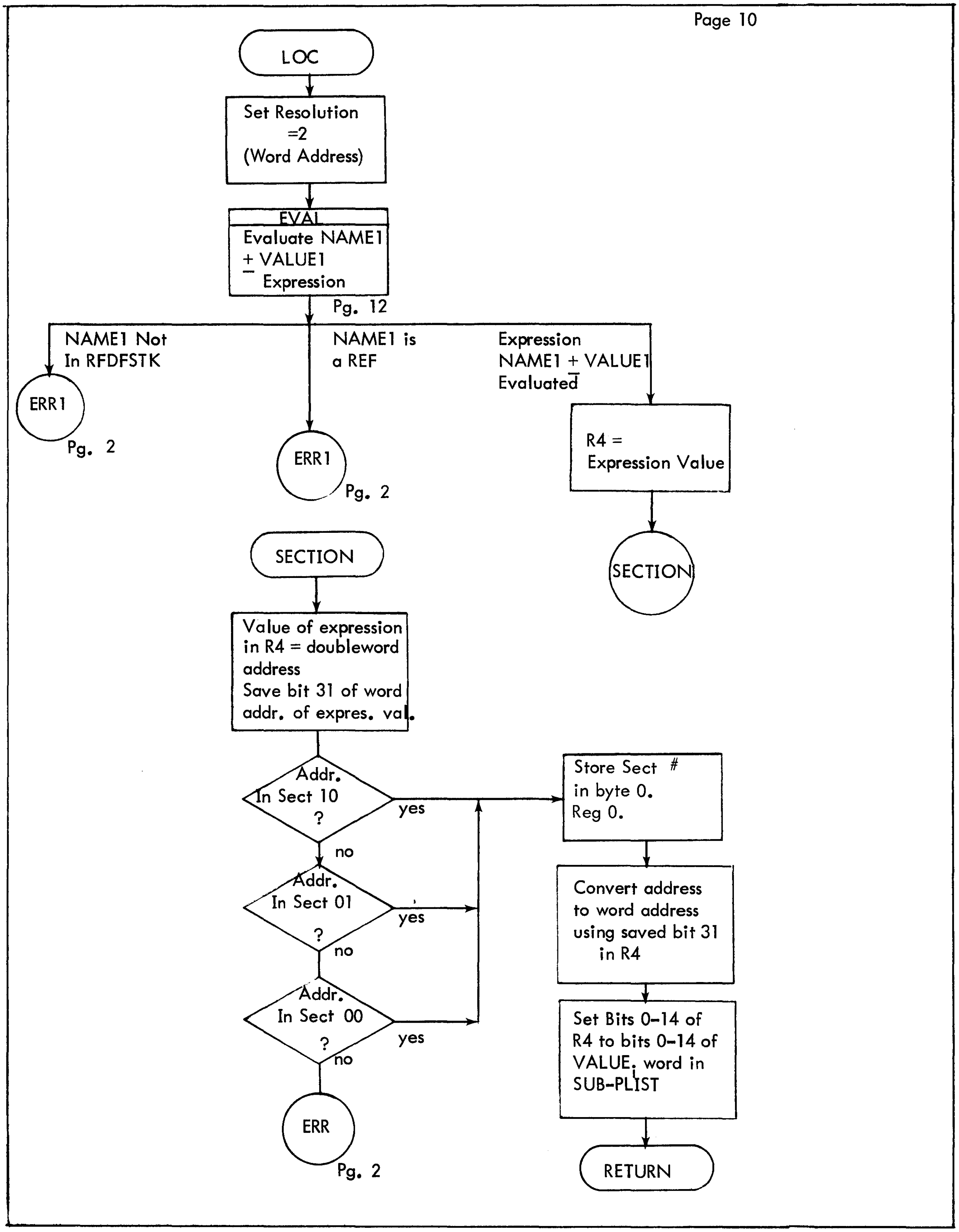
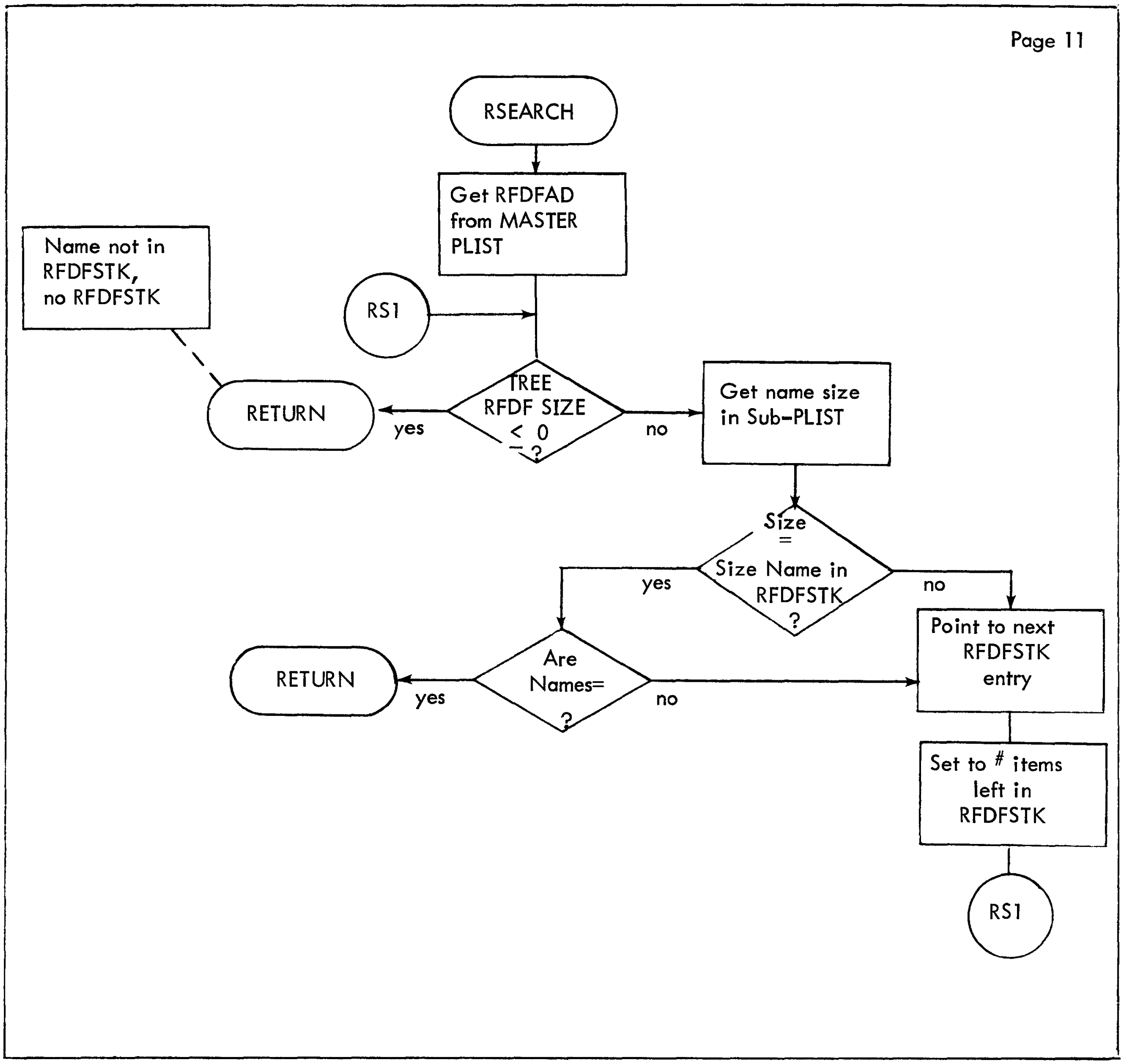

Page 10
LOC
Set Resolution =2 (Word Address)
EVAL
Evaluate NAME1 ± VALUE1 Expression
Pg. 12
NAME1 Not In RFDFSTK
ERR1
Pg. 2
NAME1 is a REF
ERR1
Pg. 2
Expression NAME1 ± VALUE1 Evaluated
R4 = Expression Value
SECTION
SECTION
Value of expression in R4 = doubleword address
Save bit 31 of word addr. of expres. val.
Addr. In Sect 10 ?
yes
no
Addr. In Sect 01 ?
yes
no
Addr. In Sect 00 ?
yes
no
ERR
Pg. 2
Store Sect # in byte 0. Reg 0.
Convert address to word address using saved bit 31 in R4
Set Bits 0-14 of R4 to bits 0-14 of VALUE word in SUB-PLIST
RETURN

Figure 6-15. Flow Diagram of MODIFY (Cont.)
(Subroutines)



Page 11
RSEARCH
Get RFDFAD from MASTER PLIST
Name not in RFDFSTK, no RFDFSTK
RS1
RETURN
yes
TREE RFDF SIZE ≤ 0 ?
no
Get name size in Sub-PLIST
Size = Size Name in RFDFSTK ?
yes
no
Are Names= ?
RETURN
yes
no
Point to next RFDFSTK entry
Set to # items left in RFDFSTK
RS1

Figure 6-15. Flow Diagram of MODIFY (Cont.)
(Subroutines)

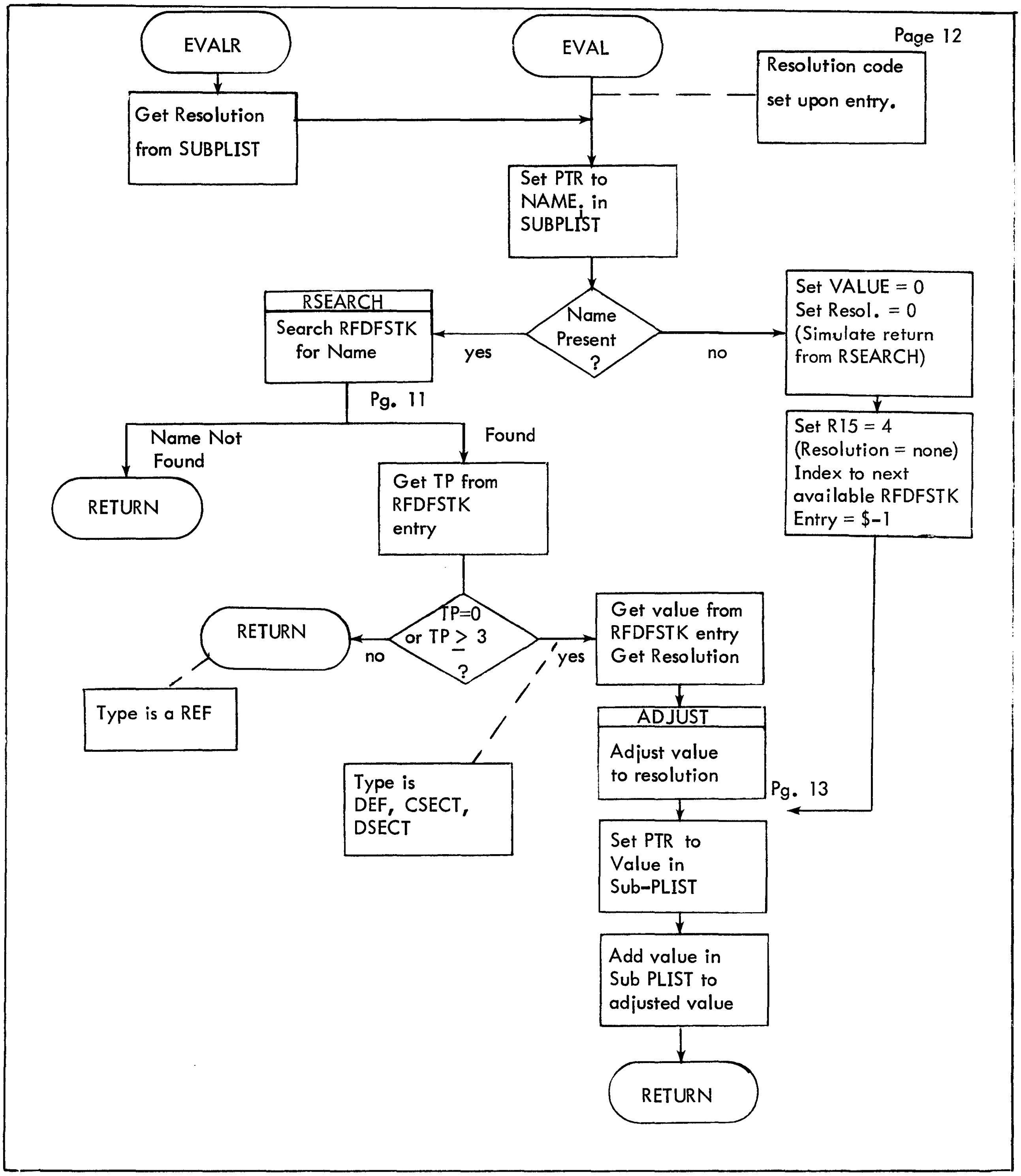

Page 12
EVALR
EVAL
Resolution code set upon entry.
Get Resolution from SUBPLIST
Set PTR to NAME, in SUBPLIST
RSEARCH
Search RFDFSTK for Name
yes
Name Present ?
no
Set VALUE = 0
Set Resol. = 0
(Simulate return from RSEARCH)
Pg. 11
Name Not Found
Found
RETURN
Get TP from RFDFSTK entry
Set R15 = 4
(Resolution = none)
Index to next available RFDFSTK Entry = $-1
RETURN
TP=0 or TP ≥ 3 ?
no
yes
Get value from RFDFSTK entry
Get Resolution
Type is a REF
ADJUST
Adjust value to resolution
Pg. 13
Type is DEF, CSECT, DSECT
Set PTR to Value in Sub-PLIST
Add value in Sub PLIST to adjusted value
RETURN

Figure 6-15. Flow Diagram of MODIFY (Cont.)
(Subroutines)

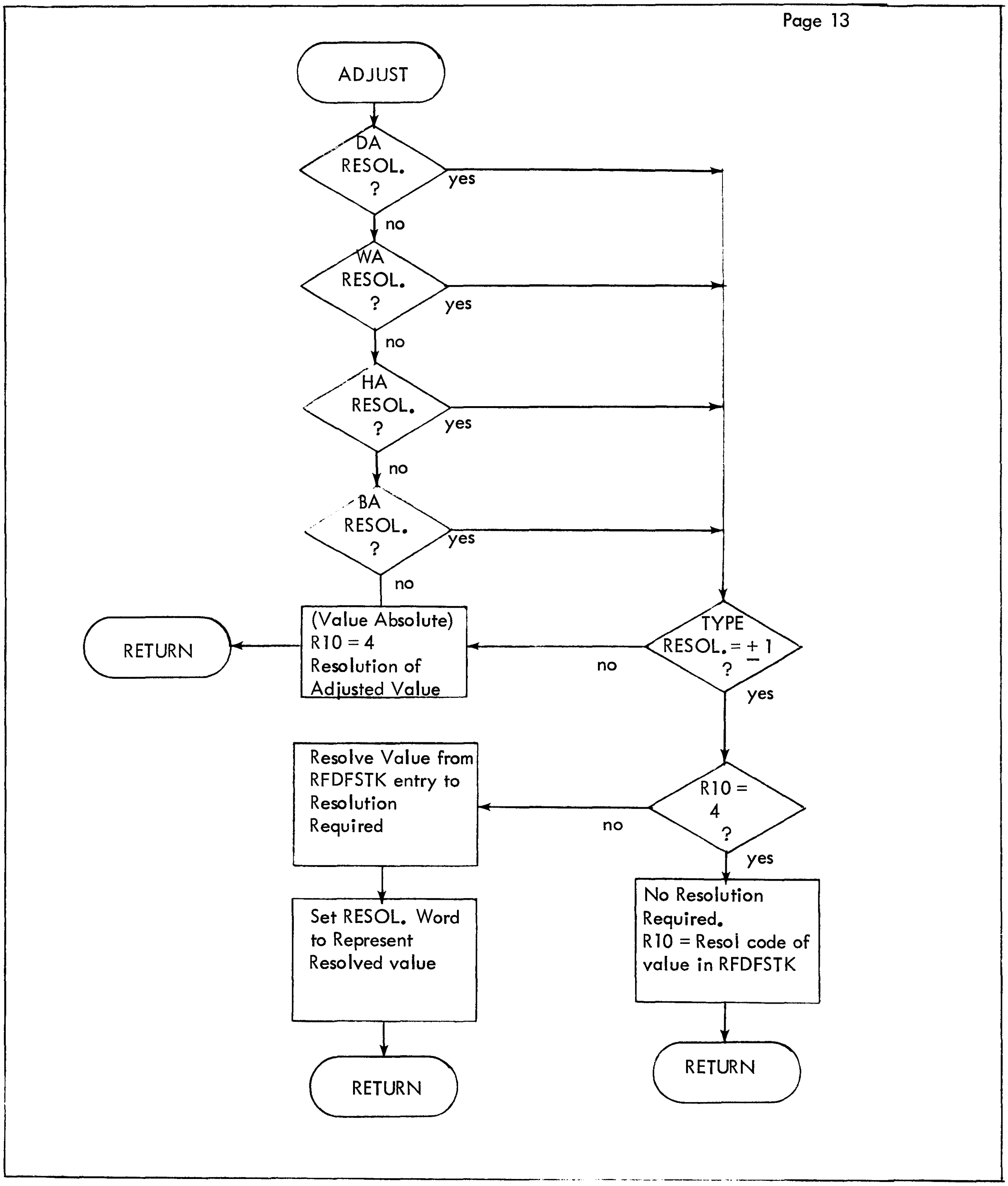

Page 13
ADJUST
DA RESOL. ?
yes
no
WA RESOL. ?
yes
no
HA RESOL. ?
yes
no
BA RESOL. ?
yes
no
(Value Absolute) R10 = 4 Resolution of Adjusted Value
RETURN
TYPE RESOL. = ± 1 ?
no
yes
R10 = 4 ?
no
yes
Resolve Value from RFDFSTK entry to Resolution Required
Set RESOL. Word to Represent Resolved value
RETURN
No Resolution Required. R10 = Resol code of value in RFDFSTK
RETURN

Figure 6-15. Flow Diagram of MODIFY (Cont.)
(Subroutines)

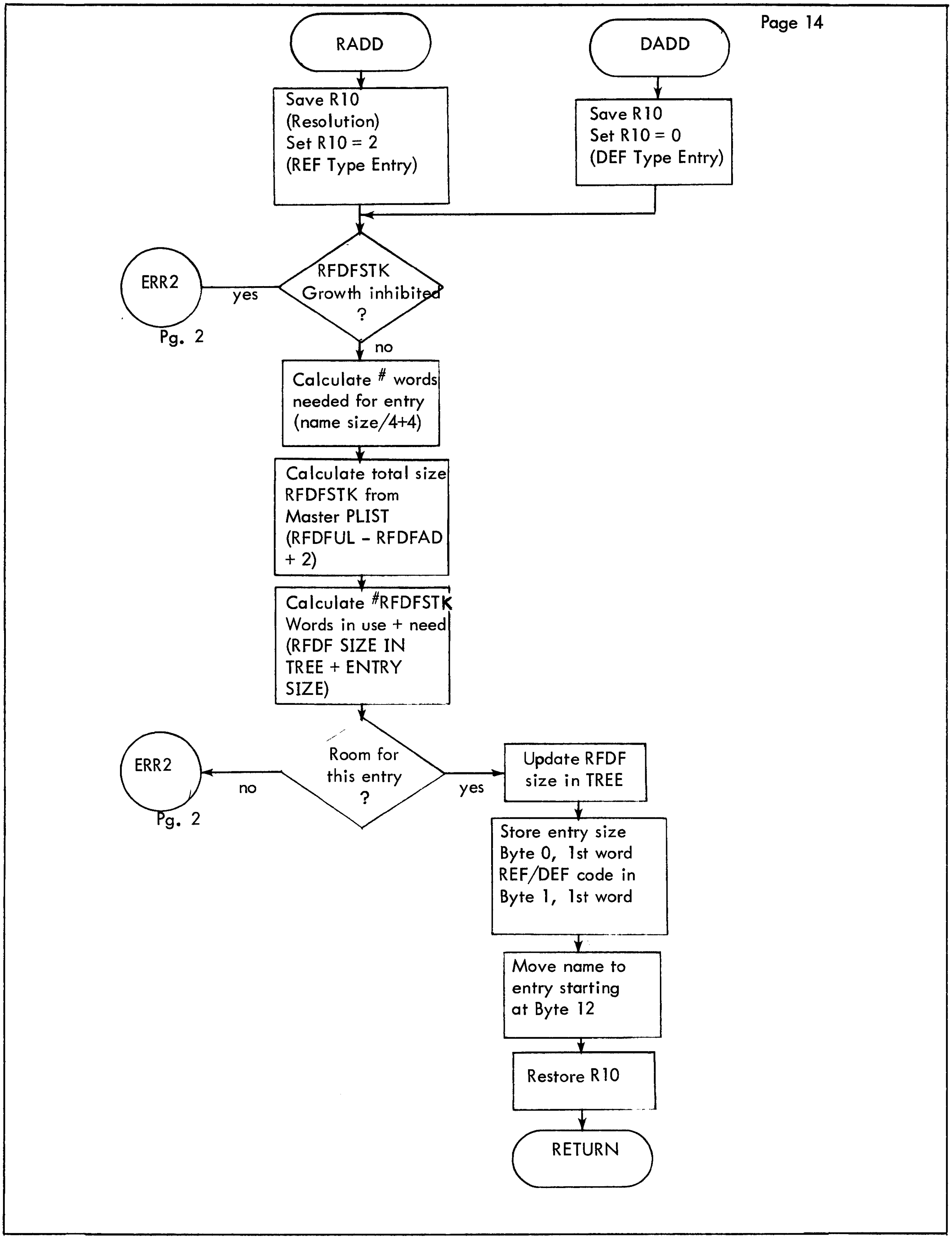


Figure 6-15. Flow Diagram of MODIFY (Cont.)
(Subroutines)

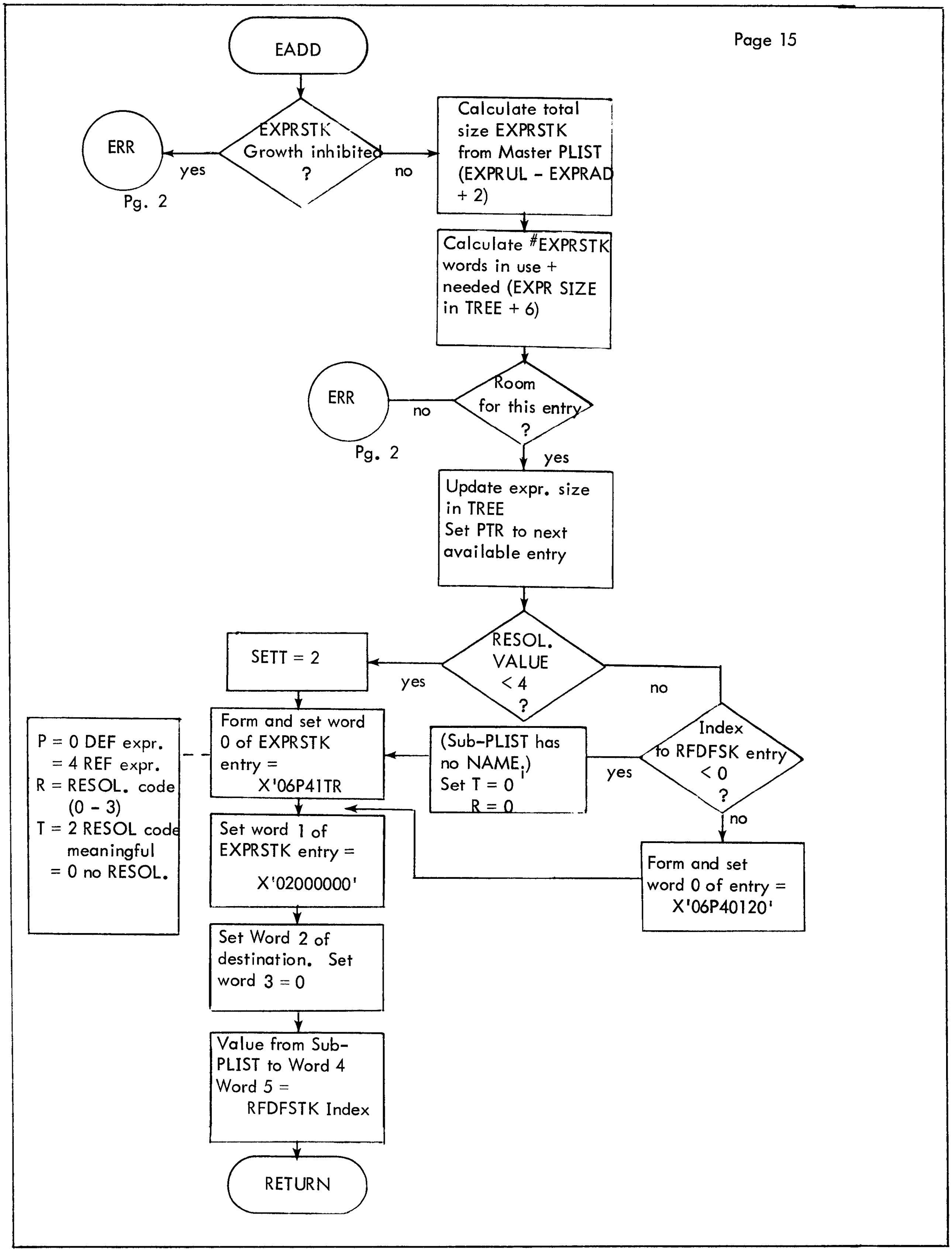

Page 15
EADD
EXPRSTK Growth inhibited ?
yes
no
ERR
Pg. 2
Calculate total size EXPRSTK from Master PLIST (EXPRUL - EXPRAD + 2)
Calculate #EXPRSTK words in use + needed (EXPR SIZE in TREE + 6)
Room for this entry ?
no
yes
ERR
Pg. 2
Update expr. size in TREE Set PTR to next available entry
RESOL. VALUE < 4 ?
yes
no
SETT = 2
Index to RFDFSK entry < 0 ?
yes
no
(Sub-PLIST has no NAME.) Set T = 0 R = 0
Form and set word 0 of EXPRSTK entry = X'06P41TR
P = 0 DEF expr. = 4 REF expr. R = RESOL. code (0 - 3) T = 2 RESOL code meaningful = 0 no RESOL.
Set word 1 of EXPRSTK entry = X'02000000'
Form and set word 0 of entry = X'06P40120'
Set Word 2 of destination. Set word 3 = 0
Value from Sub-PLIST to Word 4 Word 5 = RFDFSTK Index
RETURN

Figure 6-15. Flow Diagram of MODIFY (Cont.)
(Subroutines)

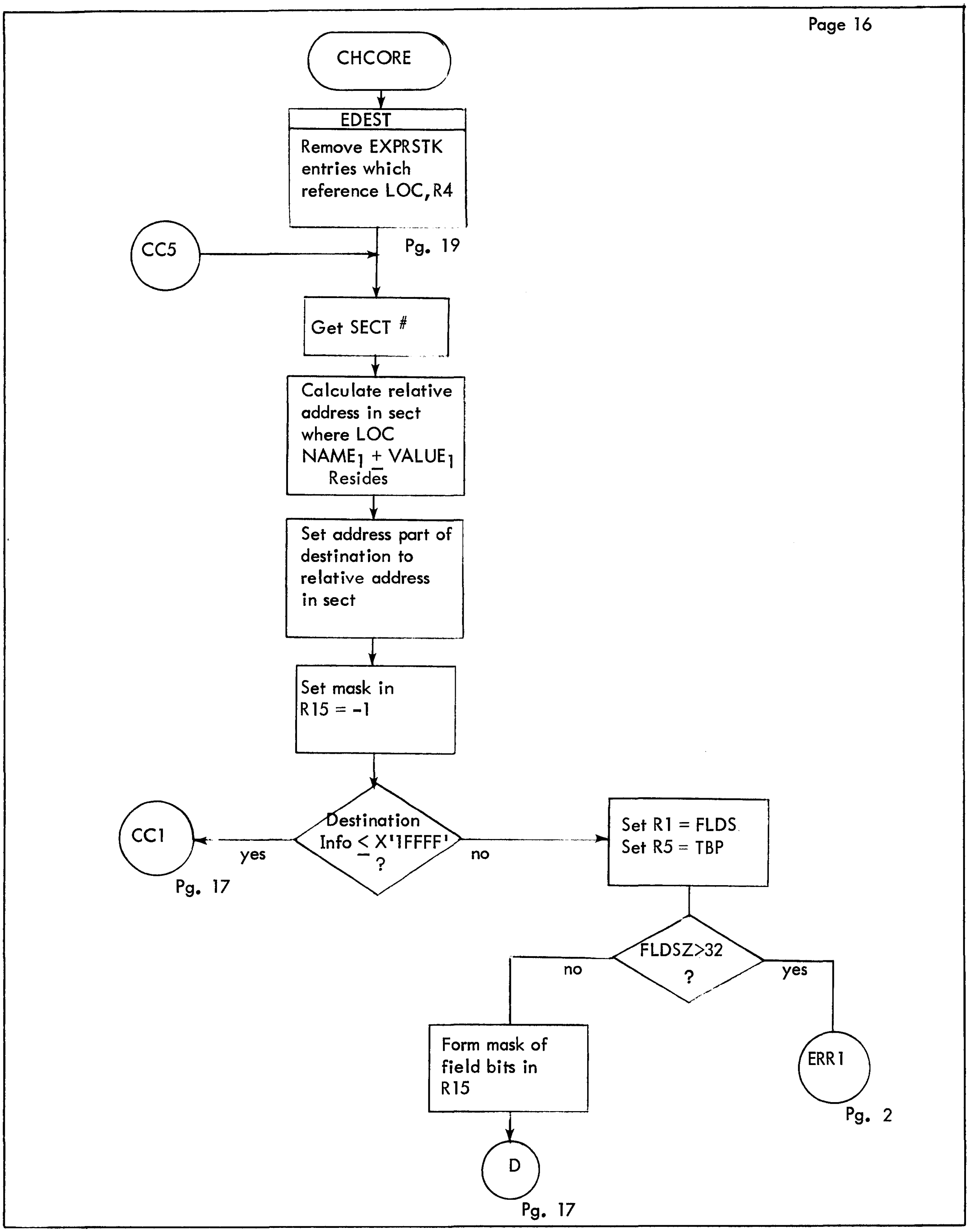

Page 16
CHCORE
EDEST
Remove EXPRSTK entries which reference LOC,R4
CC5
Pg. 19
Get SECT #
Calculate relative address in sect where LOC NAME$_1$ ± VALUE$_1$ Resides
Set address part of destination to relative address in sect
Set mask in R15 = -1
Destination Info ≤ X'1FFFF' ?
yes
CC1
Pg. 17
no
Set R1 = FLDS
Set R5 = TBP
FLDSZ>32 ?
no
yes
Form mask of field bits in R15
ERR1
Pg. 2
D
Pg. 17

Figure 6-15 Flow Diagram of MODIFY (Cont.)
(Subroutines)

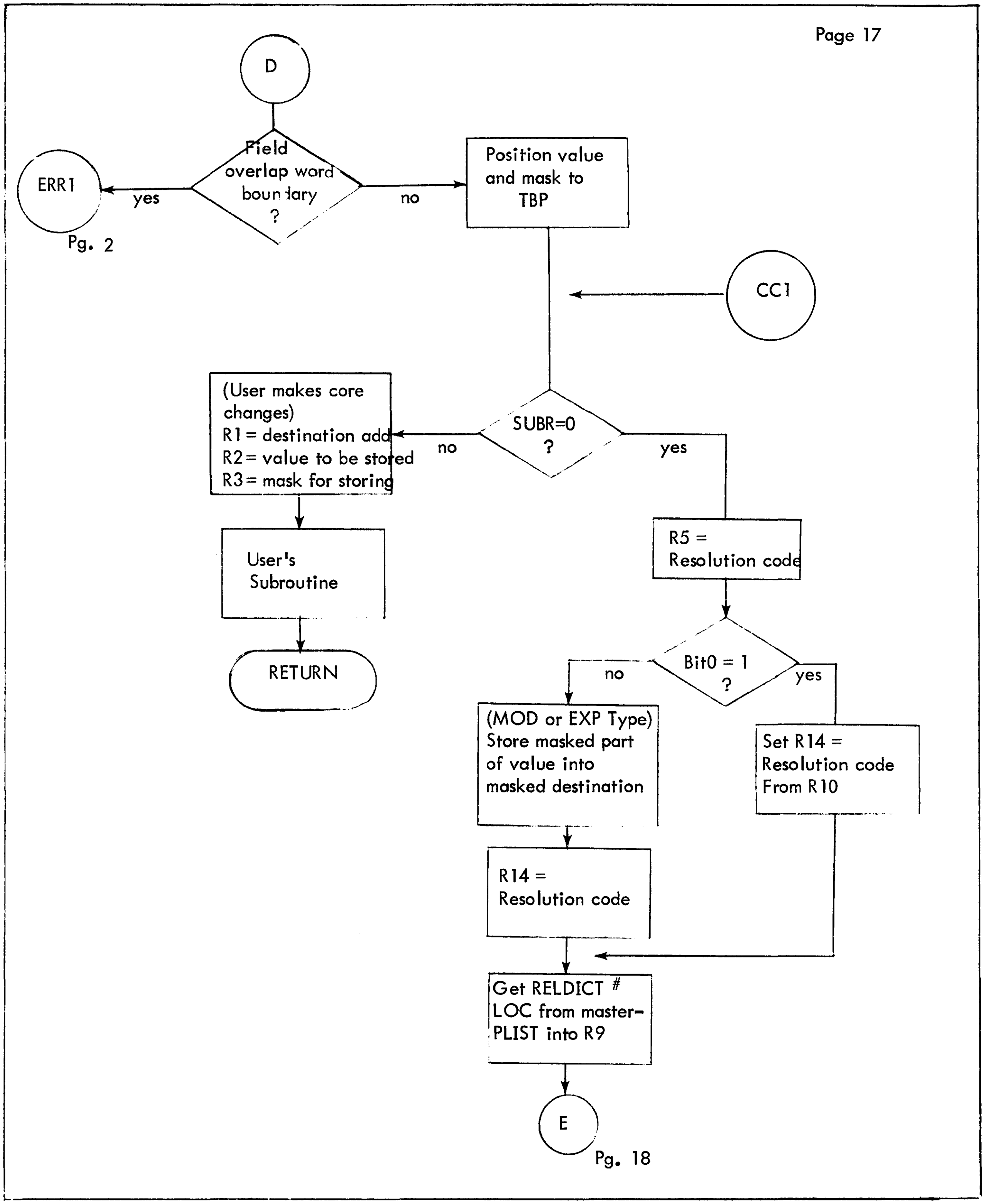

Page 17
D
Field overlap word boundary ?
yes
ERR1
Pg. 2
no
Position value and mask to TBP
CC1
SUBR=0 ?
no
yes
(User makes core changes)
R1 = destination add
R2 = value to be stored
R3 = mask for storing
User's Subroutine
RETURN
R5 = Resolution code
Bit0 = 1 ?
no
yes
(MOD or EXP Type) Store masked part of value into masked destination
Set R14 = Resolution code From R10
R14 = Resolution code
Get RELDICT # LOC from master-PLIST into R9
E
Pg. 18

Figure 6-15. Flow Diagram of MODIFY (Cont.)
(Subroutines)

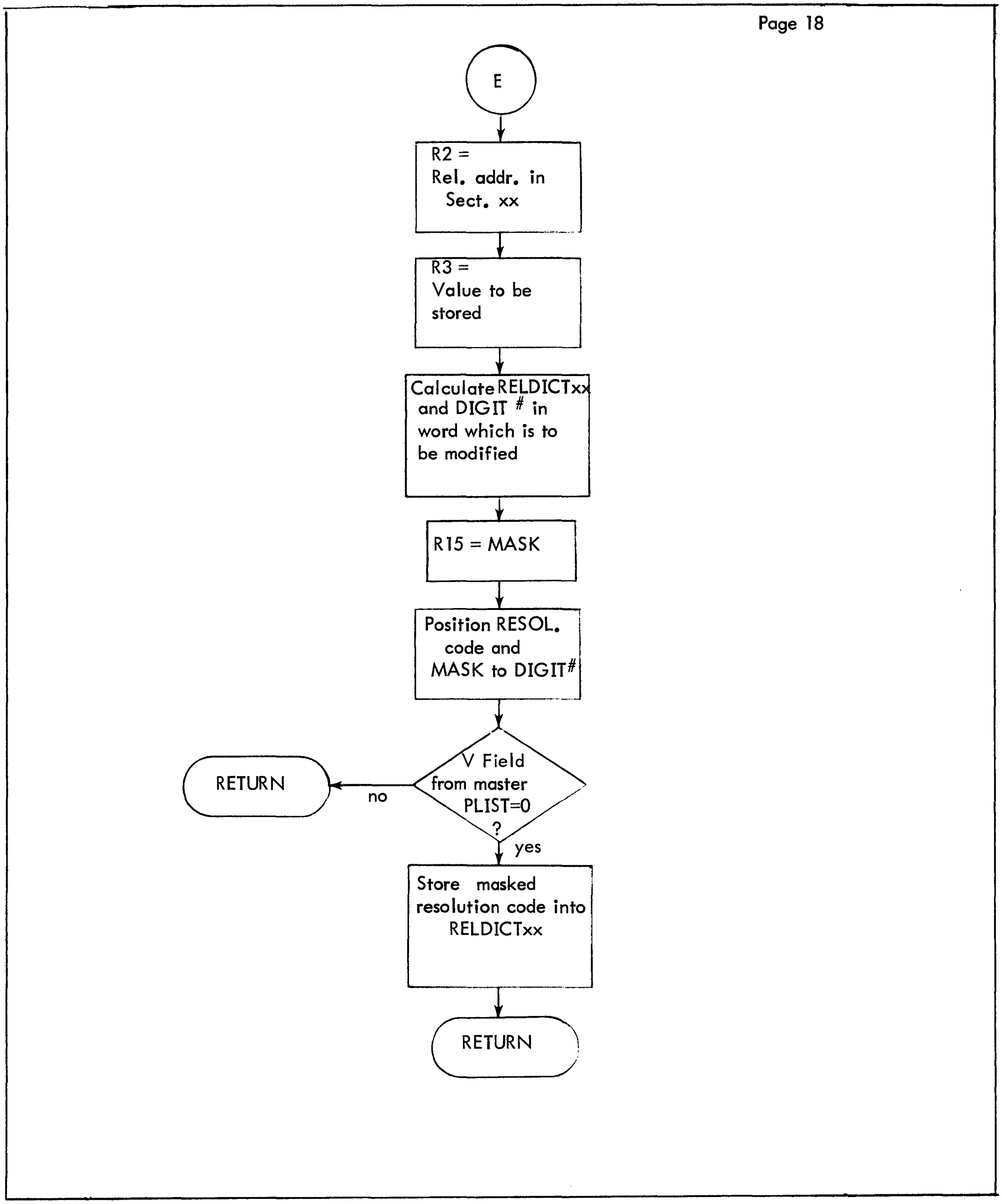


Figure 6-15. Flow Diagram of MODIFY (Cont.)
(Subroutines)

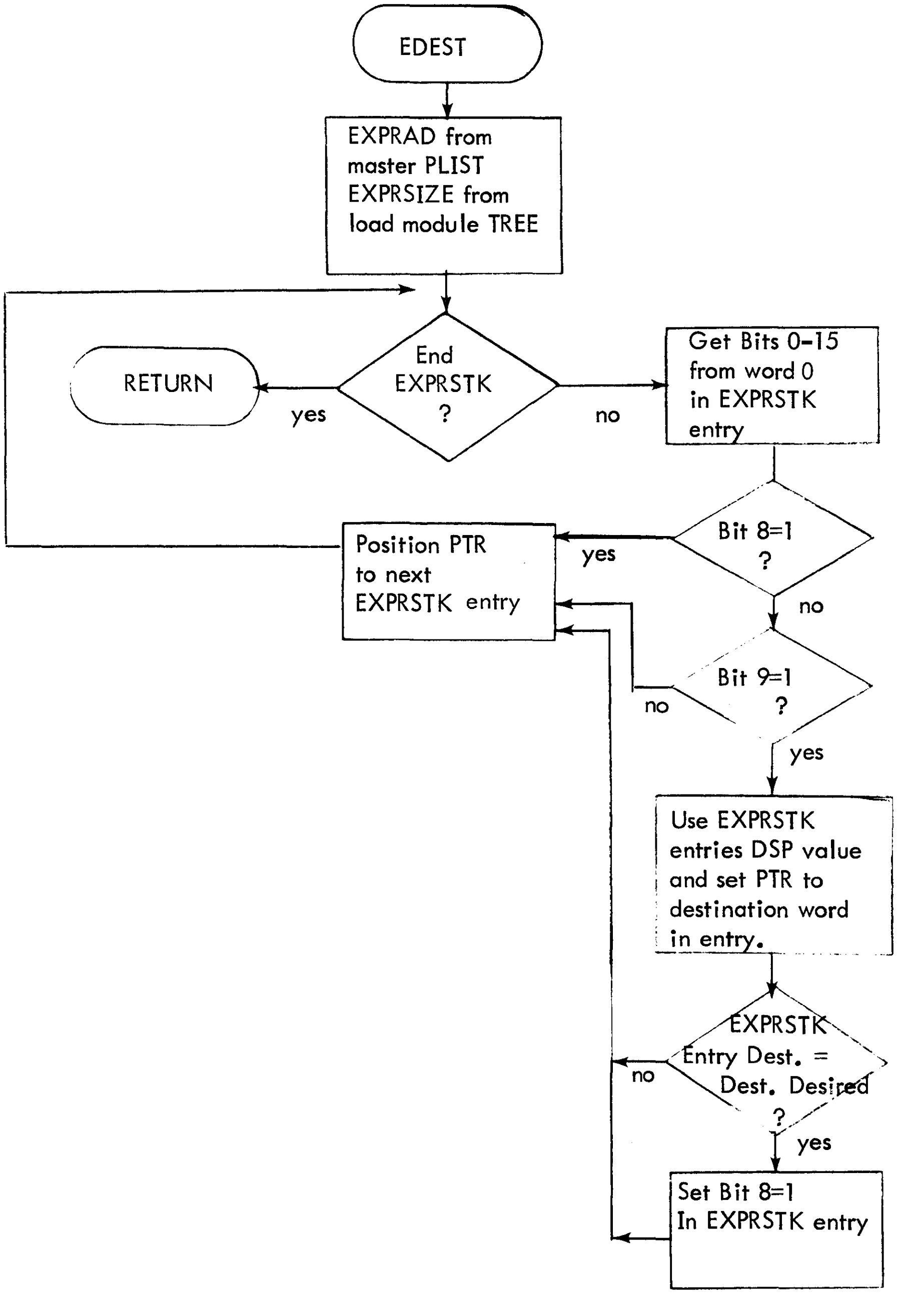

EDEST
EXPRAD from master PLIST EXPRSIZE from load module TREE
End EXPRSTK ?
yes
RETURN
no
Get Bits 0-15 from word 0 in EXPRSTK entry
Bit 8=1 ?
yes
Position PTR to next EXPRSTK entry
no
Bit 9=1 ?
no
yes
Use EXPRSTK entries DSP value and set PTR to destination word in entry.
EXPRSTK Entry Dest. = Dest. Desired ?
no
yes
Set Bit 8=1 In EXPRSTK entry

Figure 6-15. Flow Diagram of MODIFY (Cont.)
(Subroutines)

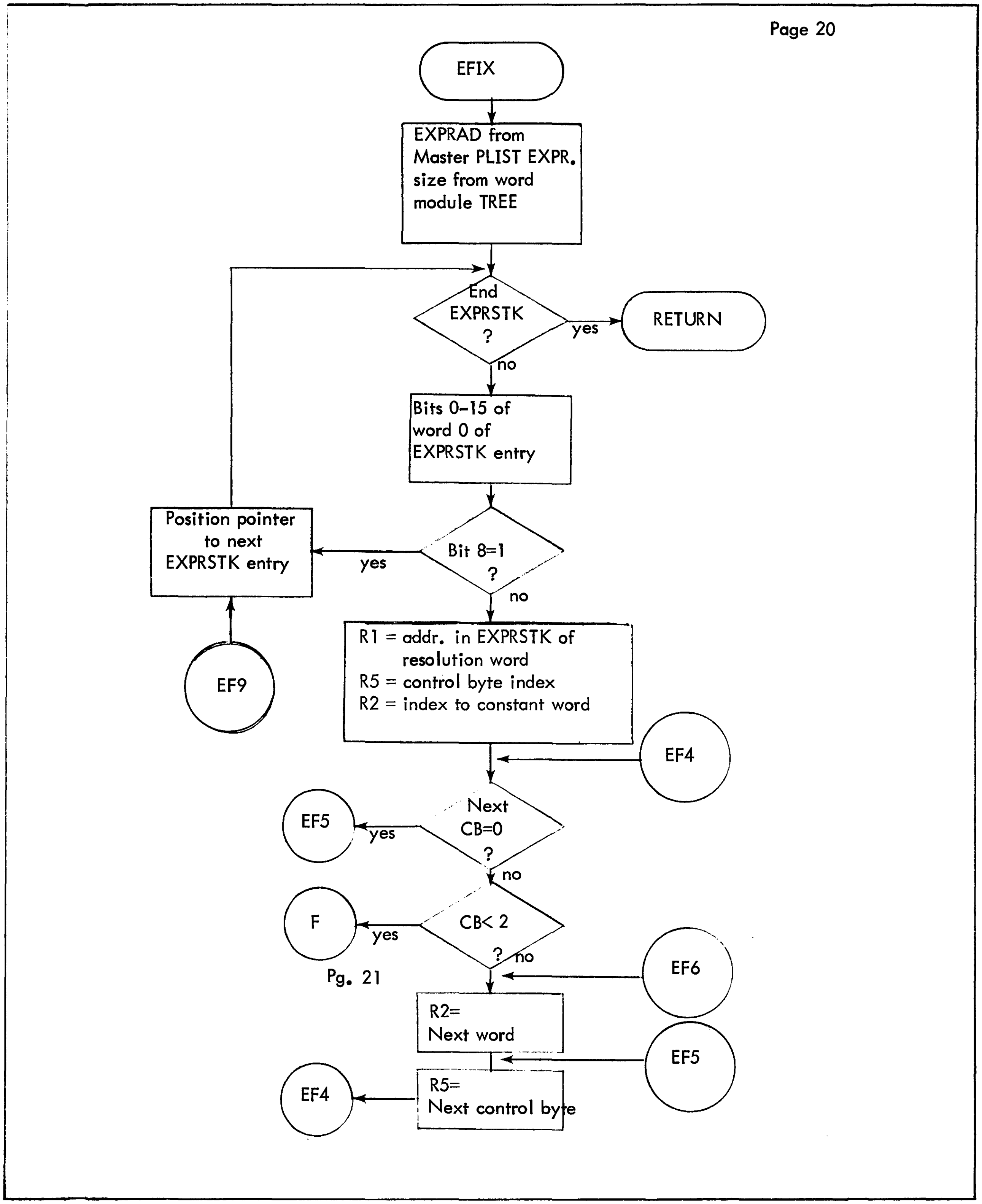

Page 20
EFIX
EXPRAD from Master PLIST EXPR. size from word module TREE
End EXPRSTK ?
yes
RETURN
no
Bits 0-15 of word 0 of EXPRSTK entry
Position pointer to next EXPRSTK entry
yes
Bit 8=1 ?
no
EF9
R1 = addr. in EXPRSTK of resolution word
R5 = control byte index
R2 = index to constant word
EF4
EF5
yes
Next CB=0 ?
no
F
yes
CB< 2 ?
no
Pg. 21
EF6
R2= Next word
EF5
EF4
R5= Next control byte

Figure 6-15. Flow Diagram of MODIFY (Cont.)
(Subroutines)

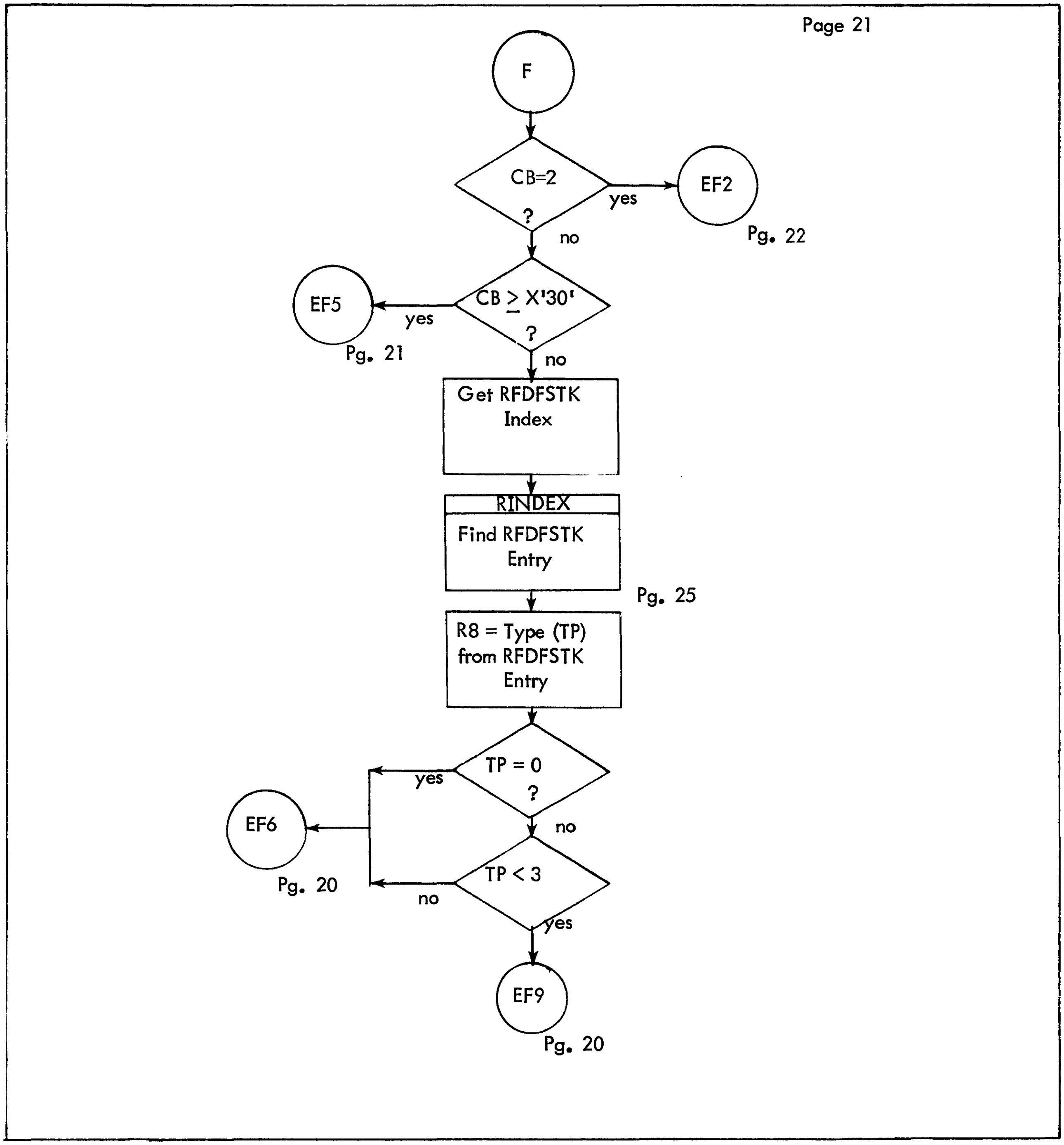


Figure 6-15. Flow Diagram of MODIFY (Cont.)
(Subroutines)

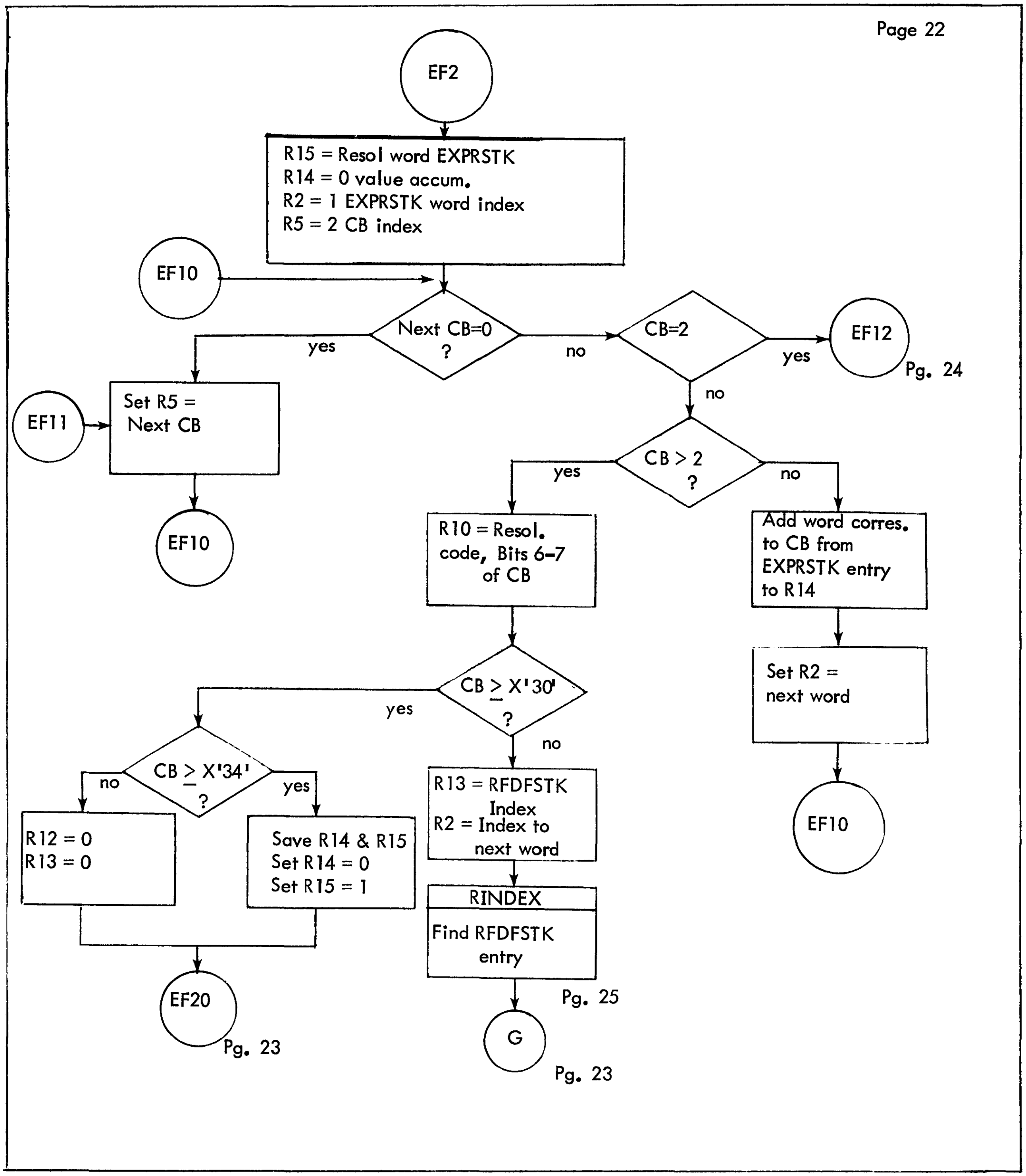


Figure 6-15. Flow Diagram of MODIFY (Cont.)
(Subroutines)

Page 23

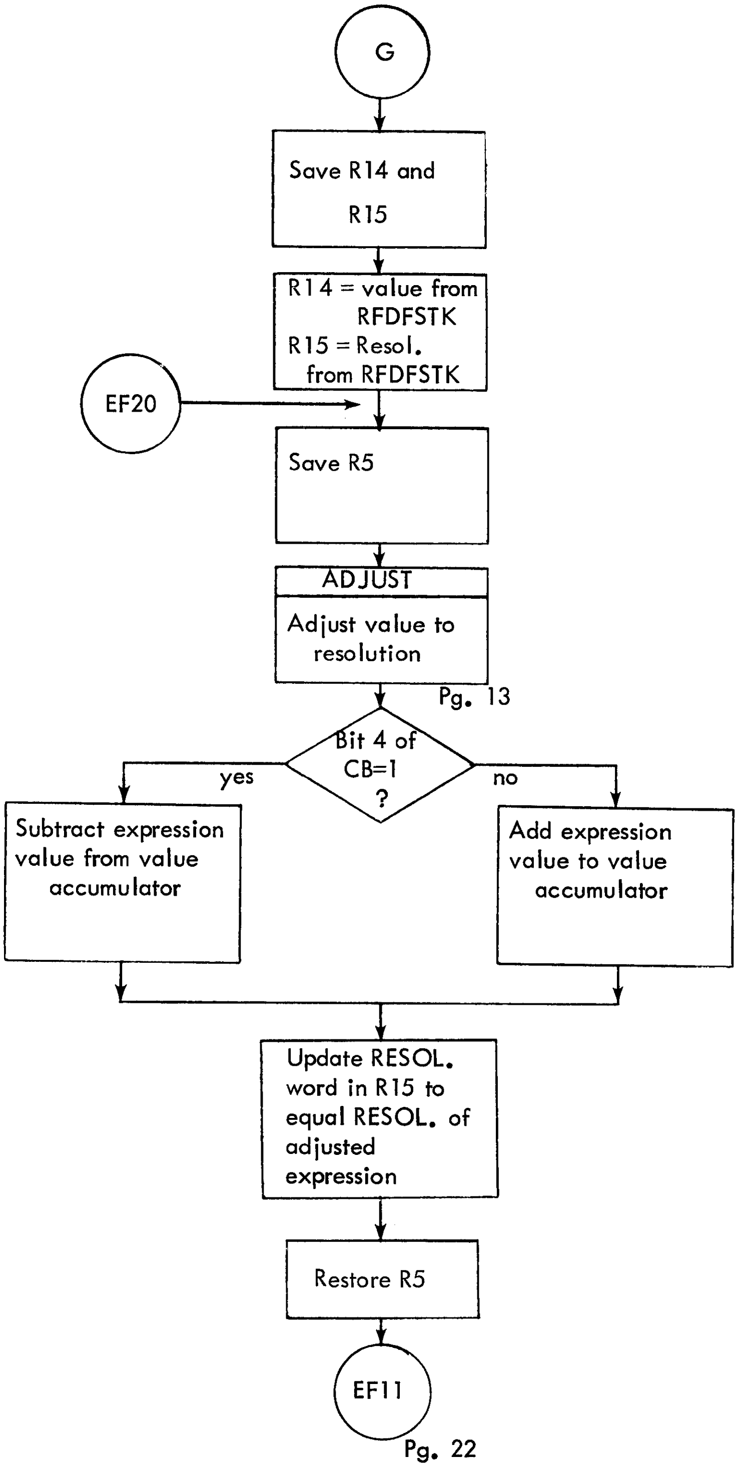

G
Save R14 and R15
R14 = value from RFDFSTK
R15 = Resol. from RFDFSTK
EF20
Save R5
ADJUST
Adjust value to resolution
Pg. 13
Bit 4 of CB=1 ?
yes
no
Subtract expression value from value accumulator
Add expression value to value accumulator
Update RESOL. word in R15 to equal RESOL. of adjusted expression
Restore R5
EF11
Pg. 22

Figure 6-15. Flow Diagram of MODIFY (Cont.)
(Subroutines)

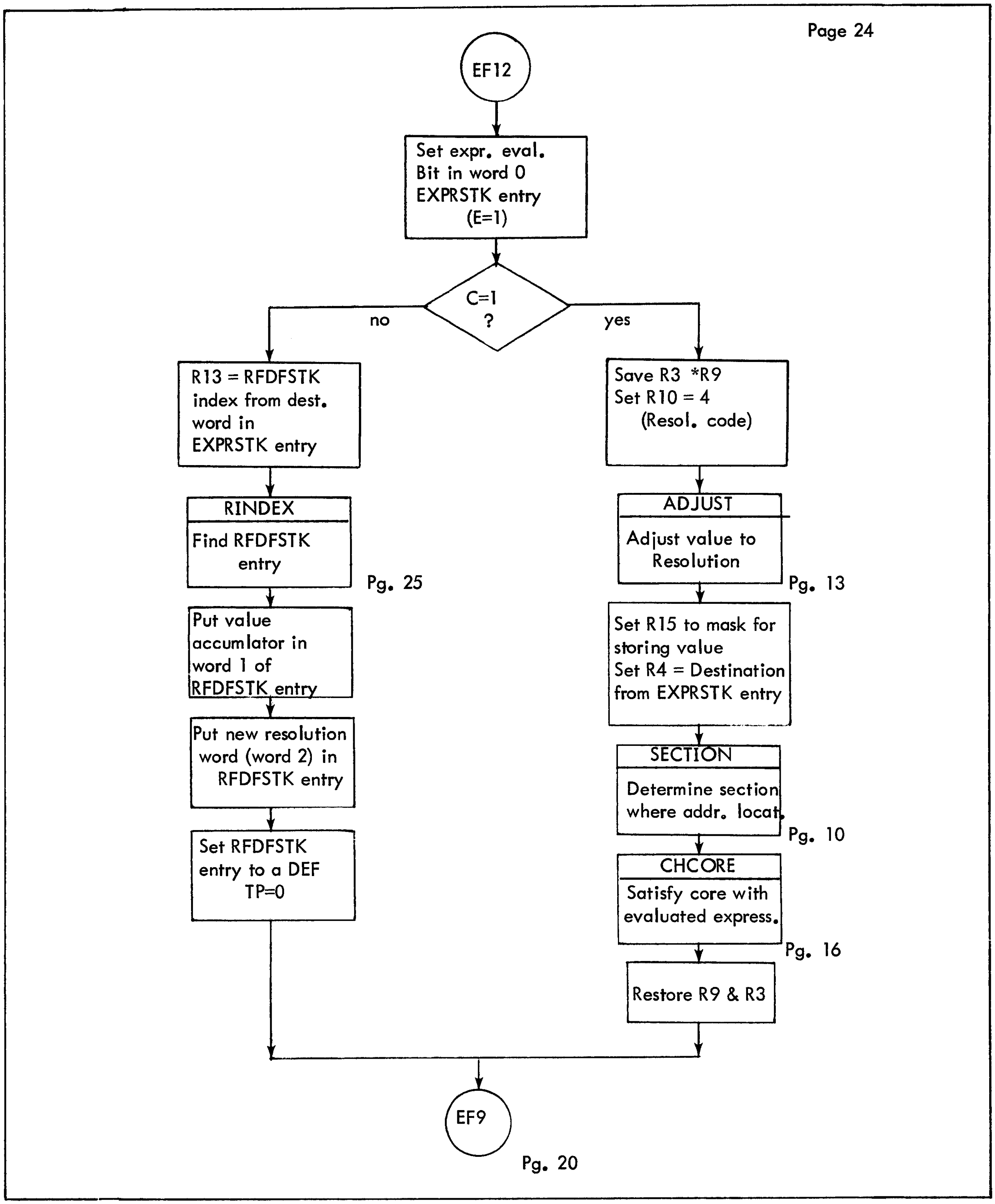

Page 24
EF12
Set expr. eval. Bit in word 0 EXPRSTK entry (E=1)
C=1 ?
no
yes
R13 = RFDFSTK index from dest. word in EXPRSTK entry
RINDEX
Find RFDFSTK entry
Pg. 25
Put value accumlator in word 1 of RFDFSTK entry
Put new resolution word (word 2) in RFDFSTK entry
Set RFDFSTK entry to a DEF TP=0
Save R3 *R9
Set R10 = 4 (Resol. code)
ADJUST
Adjust value to Resolution
Pg. 13
Set R15 to mask for storing value
Set R4 = Destination from EXPRSTK entry
SECTION
Determine section where addr. locat.
Pg. 10
CHCORE
Satisfy core with evaluated express.
Pg. 16
Restore R9 & R3
EF9
Pg. 20

Figure 6-15. Flow Diagram of MODIFY (Cont.)
(Subroutines)

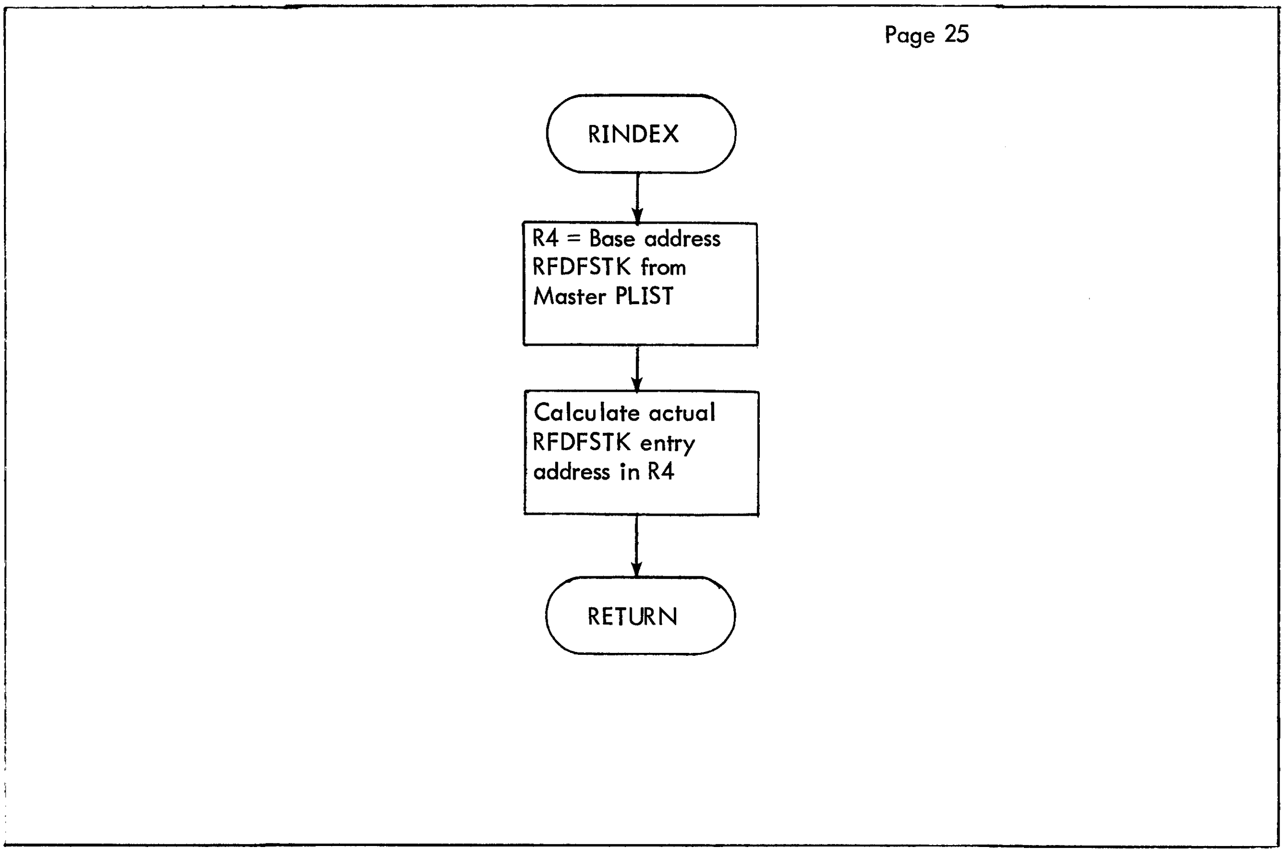

Page 25
RINDEX
R4 = Base address RFDFSTK from Master PLIST
Calculate actual RFDFSTK entry address in R4
RETURN

Figure 6-15. Flow Diagram of MODIFY (Cont.)
(Subroutines)

6.6 SYSGEN CHARACTER Routines Parameter List (PLIST)

The SYSGEN character subroutines (NXACTCHR, NAMSCAN, CHARSCAN, HEXSCAN, QUOTSCAN, DECSCAN, CHSTSCAN, and GETCHST) require a parameter list (PLIST) which controls certain aspects of control command processing. The PLIST is as follows:

PLIST (pointed to by Register 7)

Word - 0	#D	CLD
1	CNTC	CONTR
2		OUTR
3		CCP
4	FLAGS	CBUF
5		CSL
6		PCCP
7		
8		
9	CHARACTER	
10	STRING	
11	BUFFER	
12	(≤ 35 CHARACTERS)	
13		
14		
15		

CLD = byte address of list (byte table) of delimiters (terminators).

#D = number of delimiters in list.

CNTC = continue scan at this relative character position in continuation image, i.e., 1 implies character position 2.

CONTR = word address of read routine which is to read a continuation image. The subroutine must be supplied by the user.

OUTR = word address of output routine which will display current image when a semicolon, period, new line, or end of image (80 characters maximum) is encountered. This is optional but, if specified, the subroutine must be supplied by user.

CCP = relative character position in image of current character, i.e., 11 implies current character position 12.

CBUF = word address of buffer containing current images.

FLAGS = special indicators which control the scan function.

BA	BO	BF	▨	
0	1	2	3	7

BA = 0 Blank character not active character, i.e., ignore it and get next character

= 1 Blank character is an active character, i.e., it is not a field delimiter unless specified in delimiter list.

BO = 0 do not blank out input image

= 1 when a character is obtained from input image, replace its position in image with a blank character.

BF = 0 character string buffer is empty
= 1 character string buffer is full, i.e., no further scan of input image is needed, therefore, use what is in buffer.

CSL = number of characters in current character string in character string buffer.

PCCP = relative character position in input image of the first character in the character string now in character string buffer i.e., 20 implies current character position 21.

CHARACTER STRING BUFFER = a nine word (36 bytes) buffer which contains the characters of a field after the field has been scanned. There can be from one to thirty-six characters.

Figure 6-16. Character String PLIST Format

6.7 CHARACTER ROUTINES

6.7.1 NXACTCHR (SYSGEN Next Active Character Retrieve)

6.7.1.1 Purpose

To obtain a character from the input image, check it for a delimiter, then return to caller.

6.7.1.2 Calling Sequence and Input

Set REGISTER 7 = address of parameter list

Set REGISTER 8 = current character from input buffer or zero if no character exists (i.e., get next character).

```
          BAL,11    NXACTCHR
```

6.7.1.3 Output

Register 8 = current character from input buffer.

Condition Code one (CC1) set if current character is delimiter, or reset if not a delimiter.

6.7.1.4 Subroutines Used

Users display image routine, if desired.

Users read continuation image routine.

6.7.1.5 Description

This routine obtains the next active character from the input buffer (as specified by the PLIST) if Register 8 is zero. If Register 8 is non-zero, it then is assumed to contain the next active character. If the next active character is a semicolon, the users specified output subroutine is entered, if one is specified, and then the users read continuation image routine is entered. The first character (represented by CNTC in PLIST) is then obtained. If the next active character is a period, new line, or end of buffer (i.e., special delimiters), then the input buffer end has been found. If the next active character is other than those already described, the delimiter list (represented by #D and CLD in PLIST) is searched to determine if the current character is a delimiter. If not a delimiter, then condition code one (CC1) is set to zero and NXACTCHR returns to caller.

However, if the current character is a delimiter, or the input buffer end has been found, CC1 is set to one and NXACTCHR returns to caller.

6.7.1.6 NXACTCHR Flags and Counters

CCP = current character position, in input image, relative to beginning of image, i.e., CCP = 3 implies current character is character number 4.

= 80, implies end of buffer has been found.

FLAGS = BO = 1 obtain next character, and replace its position with a blank.
= 0 just obtain next character.
BA = 1 if next character is a blank, accept it as a non-delimeter unless specified as delimiter in delimiter list.
= 0 if next character is a blank, ignore it.

CLD = byte address of delimiter list.

#D = number of delimiters in list.

OUTR = address of output subroutines.
= 0 unspecified

CONTR= address of read continuation image subroutine.

CNTC = continue scan on continuation image at this character position (relative to beginning of image, i.e., CNTC = 1 implies character position 2).

CBUF = input buffer address.

6.7.1.7 Flowchart

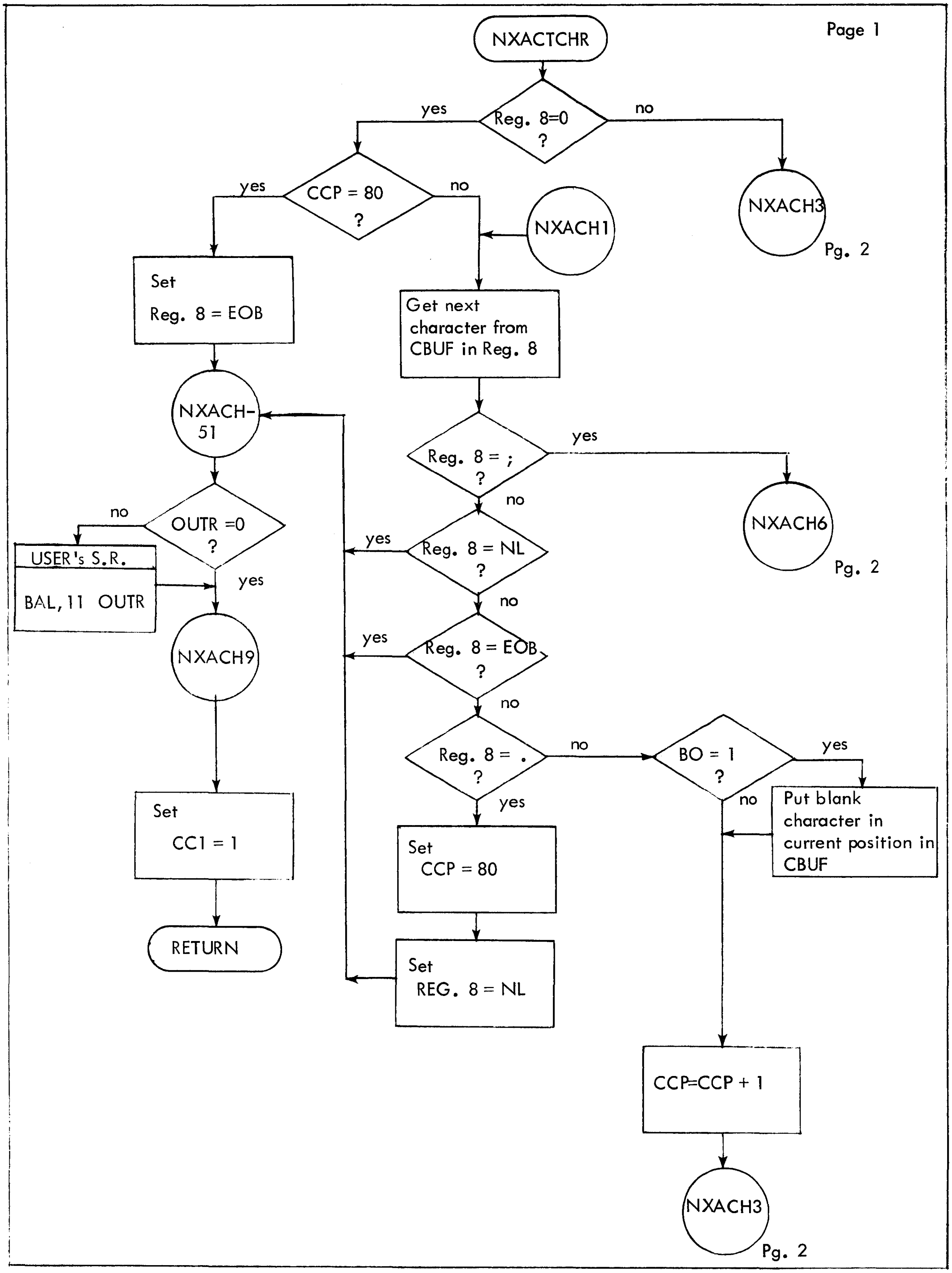


Figure 6-17. Flow Diagram of NXACTCHR

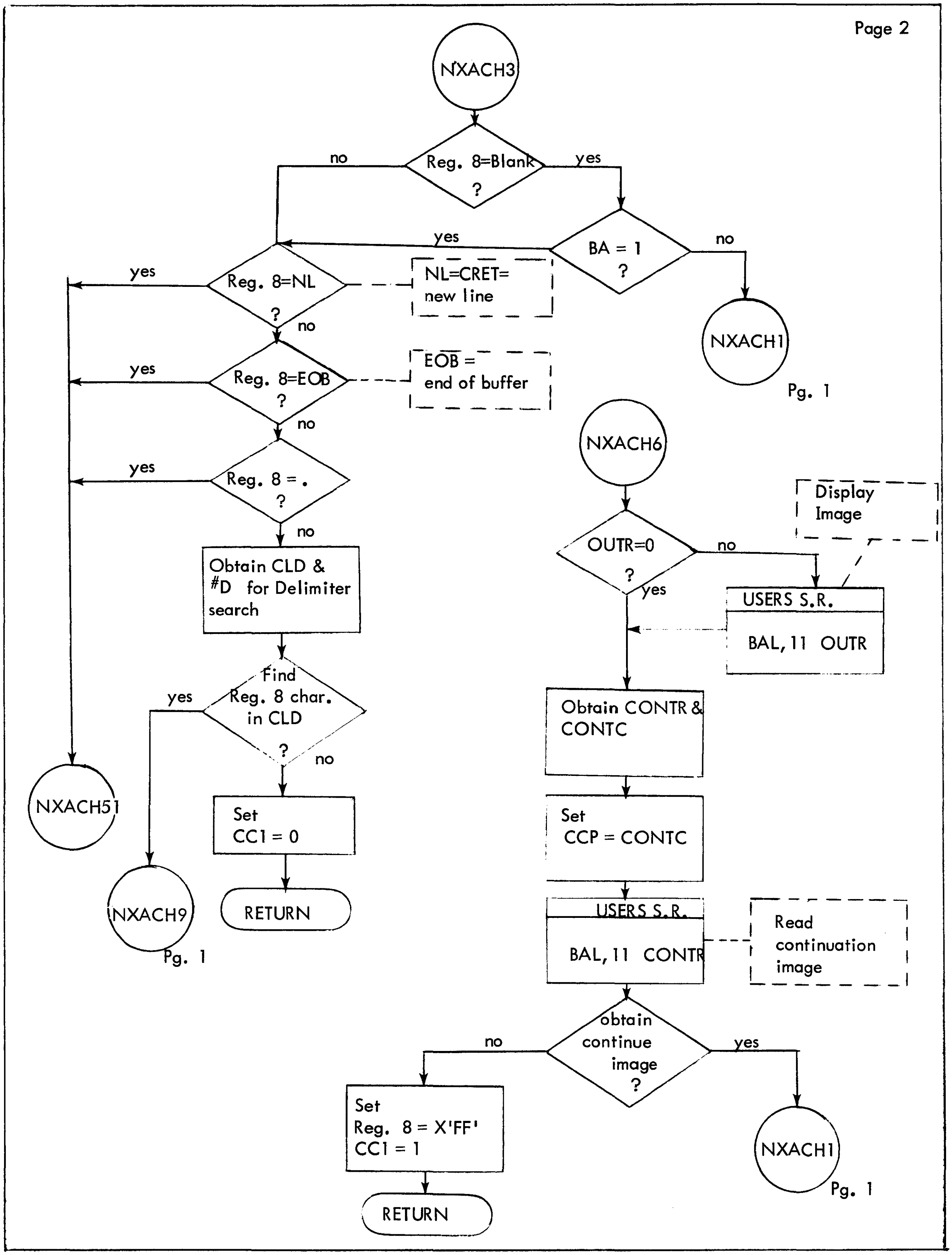


Figure 6-17. Flow Diagram of NXACTCHR (Cont.)

6.7.2 NAMSCAN (SYSGEN Get next field and check for name)

6.7.2.1 Purpose

To obtain next field from input buffer and check if it is an alphanumeric character string of which at least one character is alpha.

6.7.2.2 Calling Sequence

Set REGISTER 7 = addressof parameter list.

Set REGISTER 8 = current character from input buffer or zero. If zero, start name string with next character . If non-zero, start name with this character, then get next character.

```
        BAL,11    NAMSCAN
```

6.7.2.3 Input

See Calling Sequence.

Character string in PLIST's Character String Buffer.

Register 2 = byte address displacement of Character String Buffer in PLIST (from GETCHST).

6.7.2.4 Output

Register 8 = current character from input buffer.

Condition Code one (CC1) set if name not legal.

Condition Code one (CC1) reset if name is legal.

6.7.2.5 Subroutines Used

GETCHST (Get next character string into character string buffer).

6.7.2.6 Description

This routine requests the next string of characters from the input buffer to be put into the character string buffer. Each character is then checked for legality (i.e., alphanumeric) and the entire string of characters must contain at least one alpha character. If the name is legal, condition code one (CC1) is set to zero. However, if the name is illegal, CC1 is set to one. NAMSCAN then returns to the caller.

6.7.2.7 Flow Chart

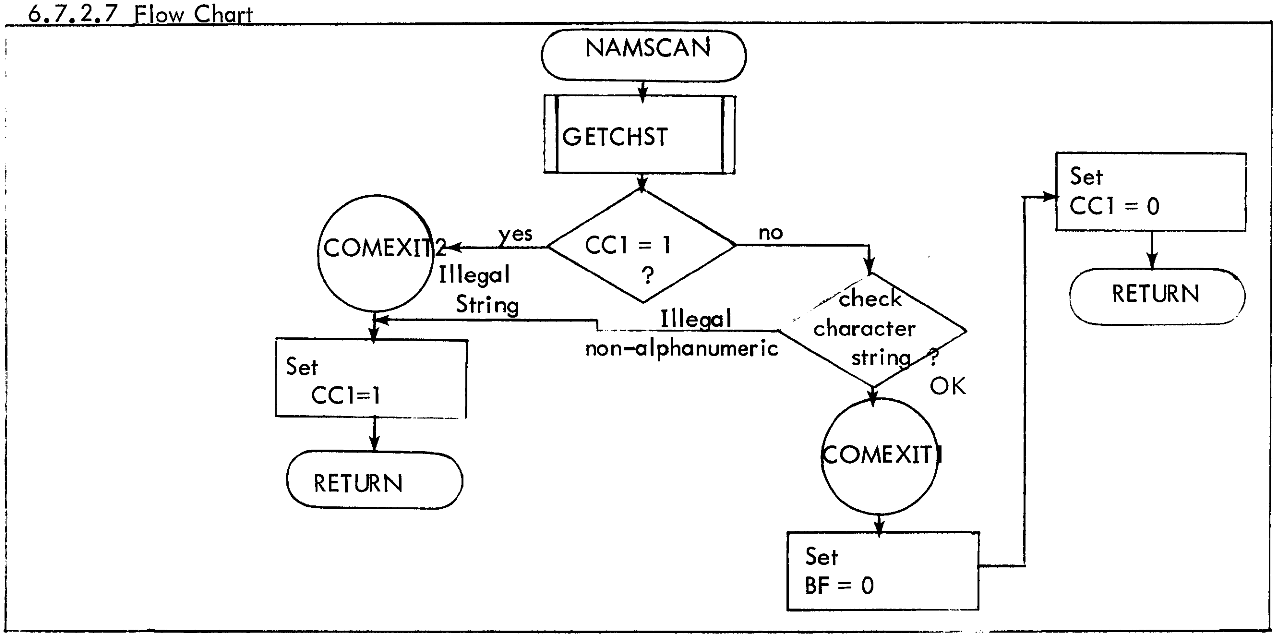


Figure 6-18. Flow Diagram of NAMSCAN

6.7.3 CHARSCAN (SYSGEN check current or next character for a specific character)

6.7.3.1 Purpose

To obtain next character from input buffer, or use current character if one exists, and check it for a specific character.

6.7.3.2 Calling Sequence and Input

Set REGISTER 7 = address of parameter list (PLIST)

Set REGISTER 8 = current character from input buffer or zero. If zero, get next character from input buffer.
If non-zero, the next character is already in Register 8.

Set REGISTER 9 = the specific character which is being checked for.

```
        BAL,11    CHARSCAN
```

6.7.3.3 Output

Register 8 = 0 if character check does compare.

Condition code one (CC1) reset if character check does compare.

Register 8 = current character if check does not compare.

Condition Code one (CC1) set if check does not compare.

6.7.3.4 Subroutines Used

NXACTCHR (Get next character from input buffer).

6.7.3.5 Description

This routine obtains the next character from the input buffer if Register 8 equals zero. If Register 8 is non-zero, the next character is assumed to be in the register 8. The character is compared to that in Register 9. If they compare, Register 8 is set to zero and CC1 is reset. If they do not compare, Register 8 is not modified and CC1 is set. CHARSCAN then returns the caller.

6.7.3.6 Flow Chart

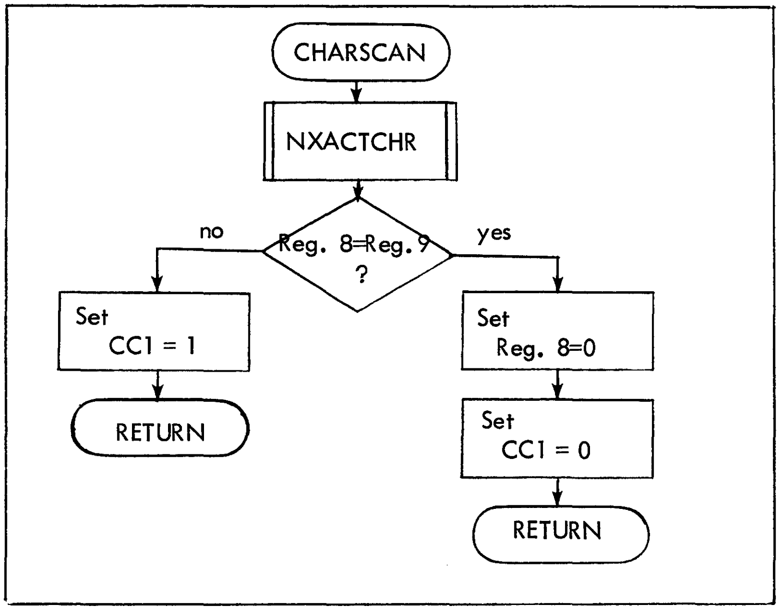


Figure 6-19. Flow Diagram of CHARSCAN

6.7.4 HEXSCAN (SYSGEN get next field and check for hexadecimal value)

6.7.4.1 Purpose

To obtain next field from input buffer and check if it is a hexadecimal value. If it is hexadecimal, then convert it from EBCDIC hexadecimal to hexadecimal.

6.7.4.2 Calling Sequence

Set REGISTER 7 = address of parameter list

Set REGISTER 8 = current character from input buffer or zero. If zero, start string with next character. If non-zero, start string with this character, then get next character.

```
        BAL,11    HEXSCAN
```

6.7.4.3 Input

Character string in PLIST's character string buffer.

Register 2 = byte address displacement of character string buffer in PLIST (from GETCHST).

6.7.4.4 Output

Register 8 = current character from input buffer.

Condition Code one (CCI) reset if value is hexadecimal.

Condition Code one (CC1) set if value is illegal.

Register 12 and last word in character string buffer = converted value (EBCDIC HEX to HEX).

6.7.4.5 Subroutines Used

GETCHST (Get next character string into character string buffer).

6.7.4.6 Description

This routine requests the next string of characters from the input buffer to be put into the character string buffer. Each character is then checked for legality (i.e., EBCDIC hexadecimal). If the character string is legal, it is converted and saved in Register 12 and in the last word of the character string buffer in the PLIST. HEXSCAN sets CC1 = 1 if illegal string or CC1 = 0 if legal, and returns to caller.

6.7.4.7 Flow Chart

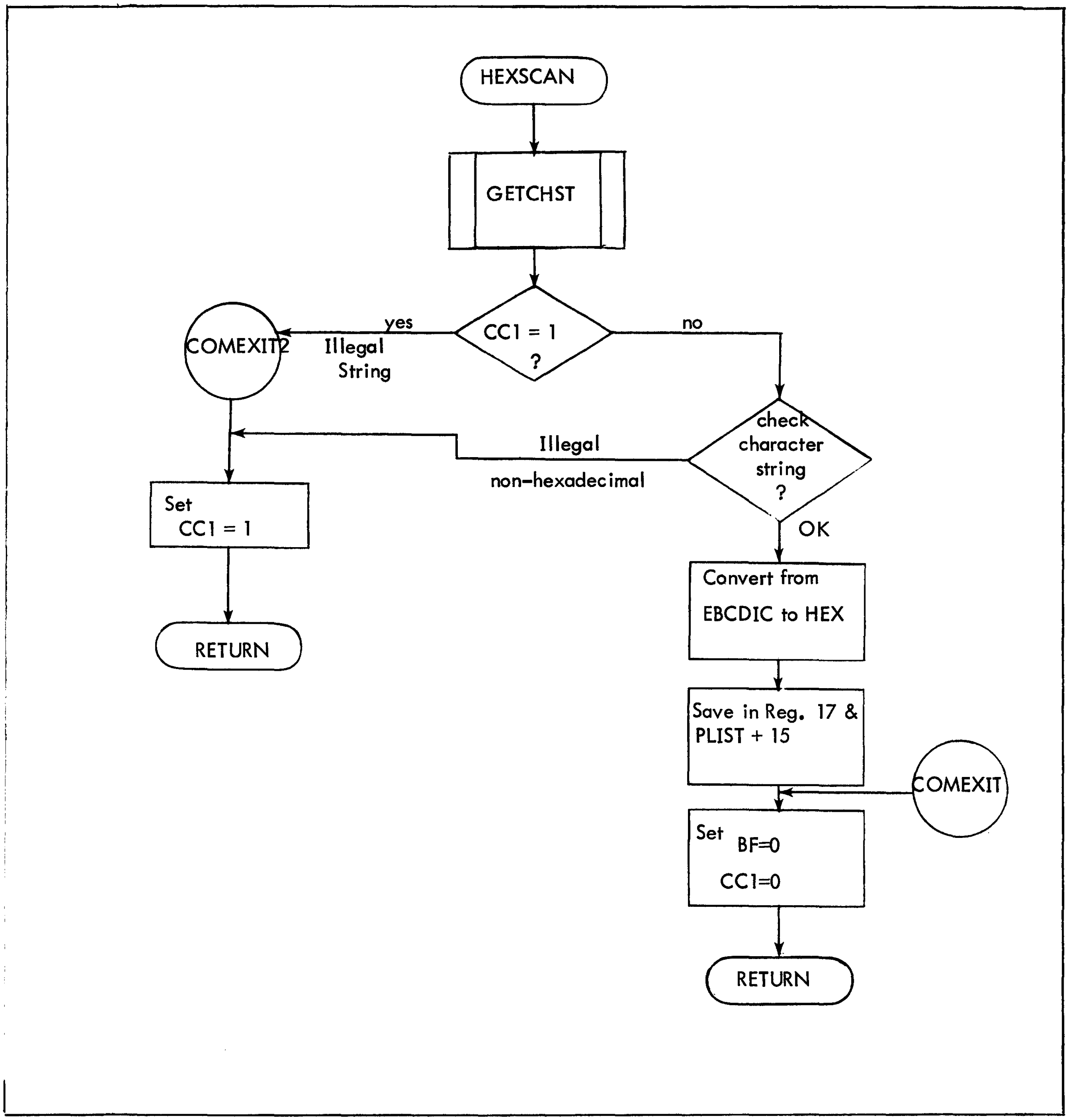


Figure 6-20. Flow Diagram of HEXSCAN

6.7.5 QUOTSCAN (SYSGEN get next field and check for specific character string)

6.7.5.1 Purpose

To obtain next field from input buffer and check the character string for a specific quote constant.

6.7.5.2 Calling Sequence

Set REGISTER 7 = address of parameter list

Set REGISTER 8 = current character from input buffer or zero. If zero, start string with this character, then get next character.

Set REGISTER 9 = address of a quote constant in TEXTC format.

```
        BAL,11    QUOTSCAN
```

6.7.5.3 Input

Character string in PLIST's character string buffer.

Register 2 = byte address displacement of character string buffer in PLIST (from GETCHST).

6.7.5.4 Output

Register 8 = current character from input buffer.

Condition Code one (CC1) set if comparison fails.

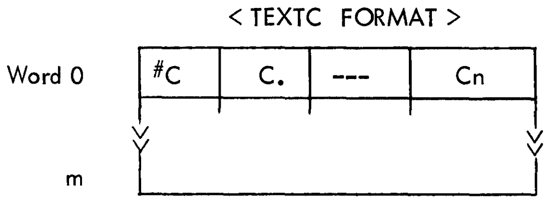
Condition Code one (CC1) reset if comparison is ok.

6.7.5.5 Subroutines Used

GETCHST (Get next character string into character string buffer).

6.7.5.6 Description

This routine requests the next string of characters from the input buffer to be put into the character string buffer. The size of the field and each character in the buffer is compared to the quote constant pointed to by register 9. If they compare, CC1 is set to zero. If they do not compare, CC1 is set to one. QUOTSCAN then returns to the caller.



#C = number of characters in quote constant.

C_1, ---, C_n = characters in quote constant.

Figure 6-21. Quote Constant

6.7.5.7 Flow Chart

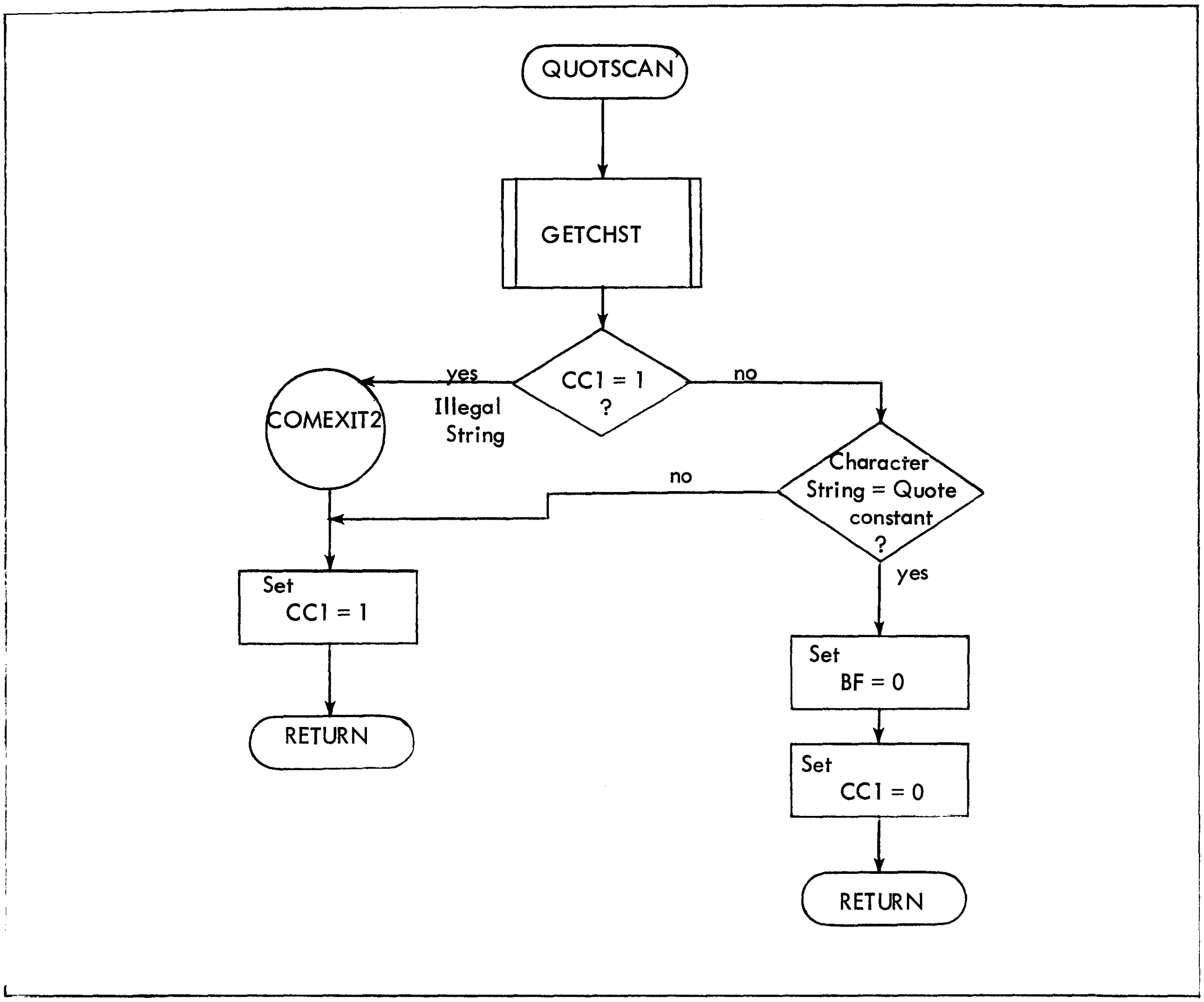


Figure 6-22. Flow Diagram of QUOTSCAN

6.7.6 DECSCAN (SYSGEN Get next field and check for decimal value).

6.7.6.1 Purpose

To obtain next field from input buffer and check if it is a decimal value. If it is decimal, then convert it to binary.

6.7.6.2 Calling Sequence

Set REGISTER 7 = address of parameter list

Set REGISTER 8 = current character from input buffer or zero. If zero, start string with next character. If non-zero, start string with this character, then get next character.

```
        BAL,11    DECSCAN
```

6.7.6.3 Input

Character string in PLIST's character string buffer.

Rggister 2 = byte address displacement of character string buffer in PLIST (from GETCHST).

6.7.6.4 Output

Register 8 = current character from input buffer.

Condition Code one (CC1) reset if value is decimal.

Condition Code one (CC1) set if value not decimal.

Register 12 and last word in character string buffer = converted value.

6.7.6.5 Subroutines Used

GETCHST (Get next character string into character string buffer).

6.7.6.6 Description

This routine requests the next string of characters from the input to be put into the character string buffer. Each character is then checked for legality (i.e., EBCDIC decimal). If the character string is legal it is converted and saved in Register 12 and in the last word of the character string buffer in the PLIST. DECSCAN sets CC1=1 if illegal string or CC1=0 if legal, and then returns to caller. Legal EBCDIC decimal characters include: 0 through 9

6.7.6.7 Flow Chart

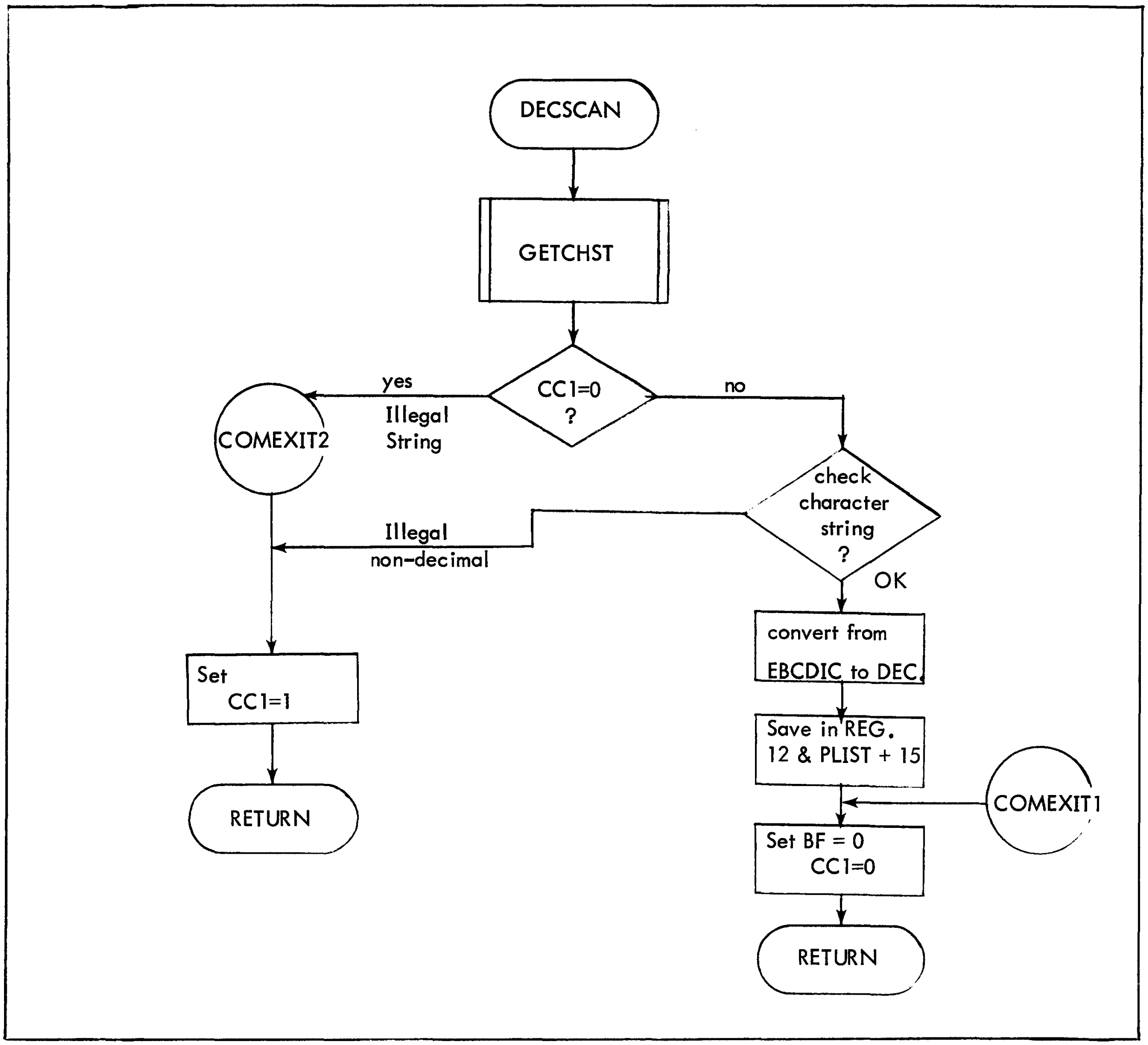


Figure 6-23. Flow Diagram of DECSCAN

6.7.7 CHSTSCAN (SYSGEN Get next field)

6.7.7.1 Purpose

To obtain next field from input buffer.

6.7.7.2 Calling Sequence

Set REGISTER 7 = address of parameter list

Set REGISTER 8 = current character from input buffer or zero. If zero, start string with next character.
If non-zero, start string with this character, then get next character.

```
        BAL,11   CHSTSCAN
```

6.7.7.3 Input

See Calling Sequence

Next character from input buffer.

6.7.7.4 Output

Register 8 = current character from input buffer.

Condition Code one (CC1) set if string not legal.

Condition Code one (CC1) reset if string is legal.

A character string in the character string buffer.

6.7.7.5 Subroutines Used

NXACTCHR (Get next character from input buffer).

6.7.7.6 Description

This routine sets the character string buffer to blanks and then requests the next character from the input buffer. If it is a delimiter, and no characters have been obtained, condition code (CC1) is set and CHSTSCAN returns to caller. If the next character is not a delimiter, it is saved in the character string buffer in the PLIST, and if the character obtained is the first character in a string, the current character position count is set into the PLIST. When the character obtained is a delimiter and a string has been found, condition code one (CC1) is reset and CHSTSCAN returns to caller. If the number of characters in a string exceeds thirty-five, condition code one (CC1) is set and CHSTSCAN returns to caller. In each case, CHSTSCAN puts the character string length into the PLIST.

6.7.7.7 CHSTSCAN Flags and Counters

BA = 0 Set initially to ignore leading blanks.
= 1 Set after first non-blank, non-delimiter character is obtained, indicating that a blank character is no longer ignored..

PCCP = relative position in input buffer of first non-blank character in character string. That is, if PCCP = 2, first character is in position 3.

CCP = relative position in input buffer of current character. If CCP = 3, then character position is 4.

CSL = the number of characters in string.

6.7.7.8 Flow Chart

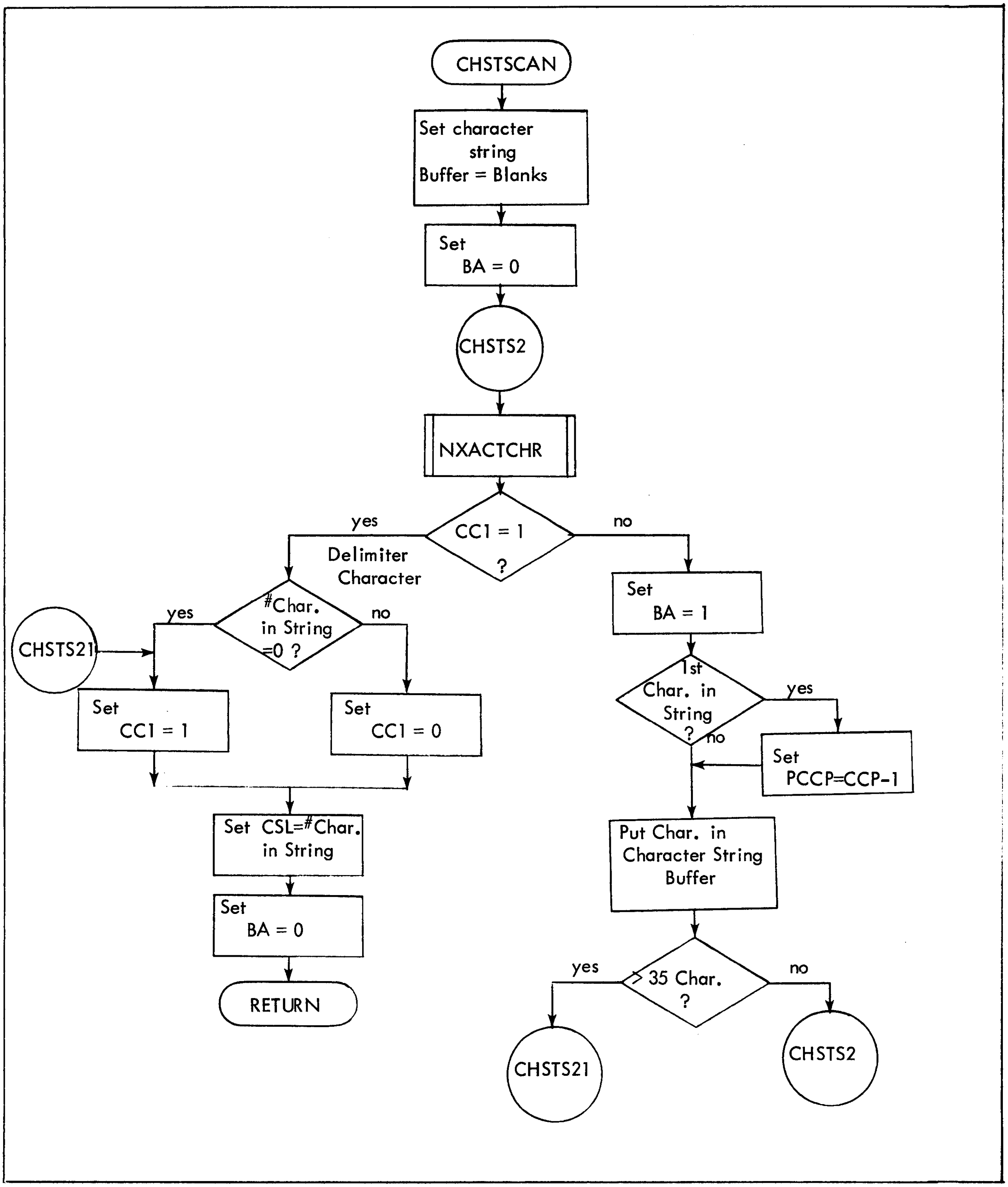


Figure 6-24. Flow Diagram of CHSTSCAN

6.7.8 GETCHST

6.7.8.1 Purpose

To check if a character string alreedy exists in the Character String Buffer, and if not, obtain the next field from the input buffer.

6.7.8.2 Calling Sequence and Input

Set Register 7 = address of parameter list

Set Register 8 = current character from input buffer or zero.

If zero, start string with next character if Character String Buffer is empty.

If not zero, start string with this character, then get next character providing Character String Buffer is empty.

```
    BAL,11  GETCHST
```

6.7.8.3 Output

Register 1 = character string length (in bytes)

Register 2 = byte address displacement of Character String Buffer in PLIST.

Condition code one (CC1) set or reset depending upon character string legality.

set if not legal

reset if legal

6.7.8.4 Subroutines Used

CHSTSCAN (Get next character string from input image).

6.7.8.5 Description

This routine is entered by the character subroutines NAMSCAN, HEXSCAN, DECSCAN, and QUOTSCAN to obtain the next character string from the input image. When a string is obtained, the BF flag in the PLIST is set to one which indicates that the Character String Buffer contains a character string. The BF flag is reset only when one of the character subroutines (as named) finds that the character string satisfies its requirements (i.e., the string is legal). If a string is not legal, the BF flag remains set such that when another one of the character subroutines (as named) requests a character string, it will receive the string which is currently in the Character String Buffer. Therefore, a field which may be either a name or decimal value, may be scanned by both the NAMSCAN and DECSCAN subroutines to determine which type it is. The GETCHST routine sets condition code one (CC1) according to the character strings legality and then exits to the caller.

6.7.8.6 Flags and Counters

BF	=	0 Character String Buffer in PLIST is empty.
	=	1 Character String Buffer in PLIST contains a character string.
CSL	=	number of characters in character string.

6.7.8.7 Flow Chart

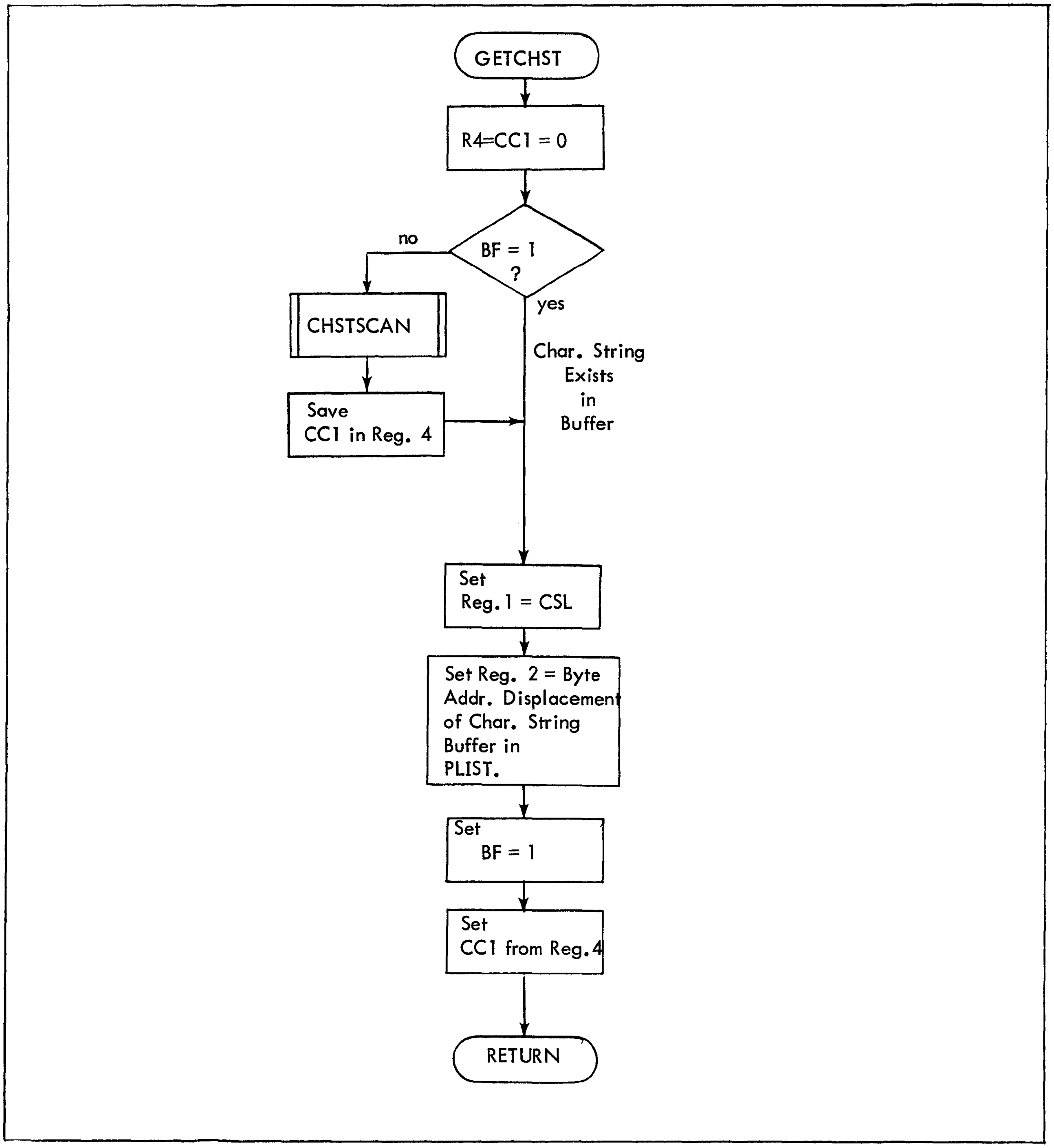


Figure 6-25. Flow Diagram of GETCHST

6.8 SYNTAX ROUTINE IN P2CCI (PASS2)

6.8.1 Purpose

SYNTAX is a PASS2 subroutine which converts control command images into manageable temp stack tables. It can analyze commands of arbitrary format, detect any syntax errors, and reject (with notification to the user) any syntactically correct information which is not acceptable to the calling program.

6.8.2 Usage

SYNTAX is called via a BAL,11 to SYNTAX. Registers 0, 1, 2, 3, 4, and 7 are assumed to contain:

(0) word address of TEMPSTACK doubleword

(1) word size of the TEMPSTACK table to be generated (see OUTPUT)

(2) word address of a skeleton TEMPSTACK table (see INPUT)

(3) base word address of P2CCI dynamic data

$(4)_{0-14}$ word size of the keyword table to be used (actual keyword portion)

$(4)_{15-31}$ word address of the keyword table (see INPUT).

(7) word address of character routine parameter list connected with the command image.

All registers are saved except (5), which upon return contains the word address of the generated. TEMPSTACK table. SYNTAX returns to BAL+1 when it has scanned and interpreted all of the command image (including continuation records).

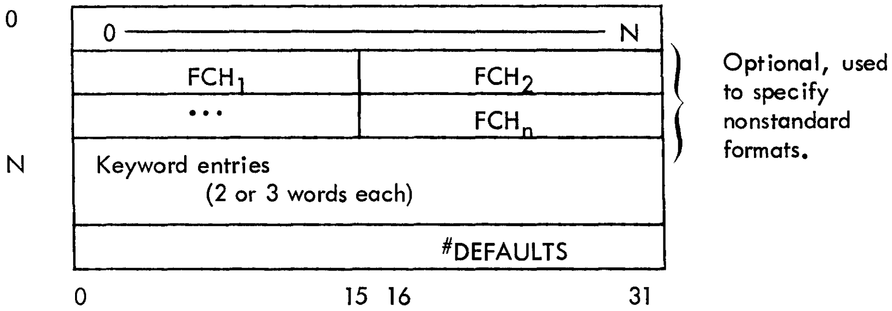
6.8.3 Input-Output

6.8.3.1 Keyword-Format Table

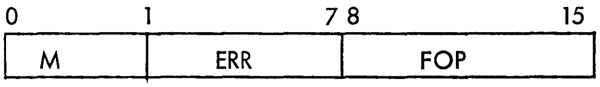
The keyword table serves two purposes. It defines valid keywords for the command and what action is to be taken when they are encountered. It also defines the format of the command if it is not standard, i.e., options separated by commas of the form:

(Keyword, value ,value ....)

The keyword table format is as follows:



FCH (Format Control Halfword):



Where:

FOP   is a syntax operation (see table of FOP's below).

ERR   is the number of the format control halfword to be used next if FOP is unsuccessful.

M   if set, implies that an error message be produced if FOP is unsuccessful.

Keyword Entry:

<table>
<tr><td>0</td><td>1   7</td><td>8   15</td><td>16   23</td><td>24   31</td></tr>
<tr><td colspan="2">C_1</td><td>C_2</td><td>C_3</td><td>C_4</td></tr>
<tr><td colspan="2">C_5</td><td>C_6</td><td>C_7</td><td>C_8</td></tr>
<tr><td>0</td><td>KOP</td><td>KF</td><td colspan="2">VDISP</td></tr>
</table>

Keyword characters (blank-filled) 2nd word optional, C_1 and $C_5 \geq$ X'80'.

KOP   is a key operation (see table of KOPs below)

KF   is usually zero (see FLAG KOP)

VDISP   is the displacement in the TEMPSTACK table of the word associated with this keyword.

#DEFAULTS   is the number of words in TEMPSTACK table that should be replaced by their defaults if the command did not specify values for them.

Currently implemented values and meanings of FOP:

VALUE	NAME	MEANING
0	NOP	Used to set error return (WDTBL, DWTBL)
1	GOTO	Set POINTER to ERR (M must be zero)
2	LEFT	Next character must be left parenthesis
3	FLEFT	Search for left parenthesis or end of command
4	RIGHT	Next character must be right parenthesis
5	FRIGHT	Search for right parenthesis or end of command
6	COMMA	Next must be comma
7	FCOMMA	Search for comma or end of command
8	INTEROPT	Next must be right parenthesis followed by end of command or comma, left parenthesis
9	KWD	Next string must be a valid keyword
10	PROCKWD	Determined by KOP, KF, and VDISP for the particular keyword
11	ANTXT	Next string must be alphanumeric, convert it to TEXT form
12	ANTXTC	Next string must be alphanumeric, convert it to TEXTC form
13	DEC	Next string must be decimal, convert it to binary
14	HEX	Next string must be hexadecimal, convert it to binary.

VALUE	NAME	MEANING
15	CNVTXTC	Convert current string to TEXTC form
16	CNVDEC	Convert current string from decimal to binary
17	CNVHEX	Convert current string from hexadecimal to binary
18	WDTBL	Store current value (output of 11 through 17) in a word table whose next available address is contained in the TEMPSTACK table displaced by ERR and whose last address is in the word preceding that. Error return is that of the previous control halfword.
19	DWTBL	Same as WDTBL except two-word entries are made.
20	GETSTRG	GET NEXT CHARACTER STRING.

By way of example, the standard format control table:

Displacement	M	ERR	FOP
0	0	0	FLEFT
1	1	0	KWD
2	1	0	PROCKWD
3	1	0	INTEROPT
4	0	1	GO TO

Currently implemented values and meanings of KOP:

VALUE	MEANING
0	If KF nonzero, OR's KF into byte displaced by VDISP from TEMPSTACK table. If KF zero, sets defaults and initiates a new TEMPSTACK table. No comma or value is expected for these keywords.
1	One decimal value expected.
2	One hexadecimal value expected.
3	One device address (NDD) value expected.
4	Same as 0 if KF zero, except one hexadecimal value is read,put into the new table, and defaults are set only for the first table.
5	Same as 4, except expects one decimal value
8	The value of VDISP represents a displacement into the table of FOP's. Effectively this performs a keyword GOTO function.
X'40'	Stores the flag KF in a byte table (at VDISP into TEMPSTACK table) indexed by each decimal value following the keyword. The byte size of the table must be in VDISP-1 into TEMPSTACK table.

NOTE: When the KOP is 1-5, then the KF field may be used to represent the number of values expected to follow the keyword. The entries at VDISP into the temp stack table are arranged to conform with the order of the values following the keyword.

6.8.3.2 SKELETON TABLE

The skeleton table is copied intact into the TEMPSTACK table at SYNTAX initialization. Its format is arbitrary, but SYNTAX expects format in most cases.

If FOP is WDTBL or DWTBL, the format is as is described in their descriptions.

If FOP is PROCKWD and KOP is not 0 or X'40' (whose skeleton (TEMPSTACK) table formats are described above), then the word at VDISP into TEMPSTACK table has three parts:

Bit 0 — is a flag. When set it implies that no value has been stored into the word by SYNTAX (the associated keyword either has not been encountered or the value following it was in error). If this bit is set when end of command is reached and VDISP is less than #DFLT, then SYNTAX replaces the word with its default.

Bits 16-31 — is the upper limit of acceptable values. If it is zero, no limit checking is done.

Bits 1-15 — is the default value and, if the upper limit is nonzero, is also the lower limit of acceptable values. (Signed arithmetic value).

Note that this implies that SYNTAX cannot do both limit checking and default setting if the lower limit is not the same as the default.

6.8.4 Interaction

SYNTAX is currently used by PASS2 modules: XMONITOR, XLIMIT, IMC, P2COC, SPROCS, XPART, and BTM. SPROCS and XMONITOR provide their own format table.

SYNTAX uses PASS2's character scanning subroutines to scan the command, P2CCI's LISTIT to print the command, and P2CCI's OUTLLERR to print error position indicators.

6.8.5 Errors

*** SYNTAX ERROR - '(' EXPECTED

Self-explanatory. Occurs on LEFT or INTEROPT FOP.

*** SYNTAX ERROR - ')' EXPECTED

Self-explanatory. Occurs on RIGHT or INTEROPT FOP.

*** SYNTAX ERROR - ',' EXPECTED

Self-explanatory. Occurs on COMMA, INTEROPT, or PROCKWD FOP.

*** INVALID, UNKNOWN, or DUPLICATE KEYWORD

Self-explanatory. Occurs on KWD or PROCKWD FOP.

*** INVALID ALPHANUMERIC STRING

Self-explanatory. Occurs on ANTXT or ANTXTC FOP.

*** ILLEGAL TYPE OR SIZE

A non-keyword string does not conform to the restraints imposed on it.
Occurs on FOP's 10 through 19.

*** TOO MANY VALUES

A table has been filled. Occurs on WDTBL or DWTBL FOP.

*** ERROR IN PROCESSOR-JOB ABORTED

A FOP or KOP has not been implemented.

*** INVALID CHARACTER STRING

Error has occurred in getting character string, occurs on FOP 20.

6.8.6 Description

SYNTAX initializes itself by copying the skeleton table to the TEMPSTACK, and finding the keyword table and the appropriate format table. It then proceeds to perform the operations contained in the format table, putting valid information from the command into the TEMPSTACK table and producing error messages for invalid information. When it reaches the end of the command, it finds #DEFAULTS at the end of the keyword table, sets the appropriate number of defaults, restores the registers, and returns.

6.8.7 Flow Chart

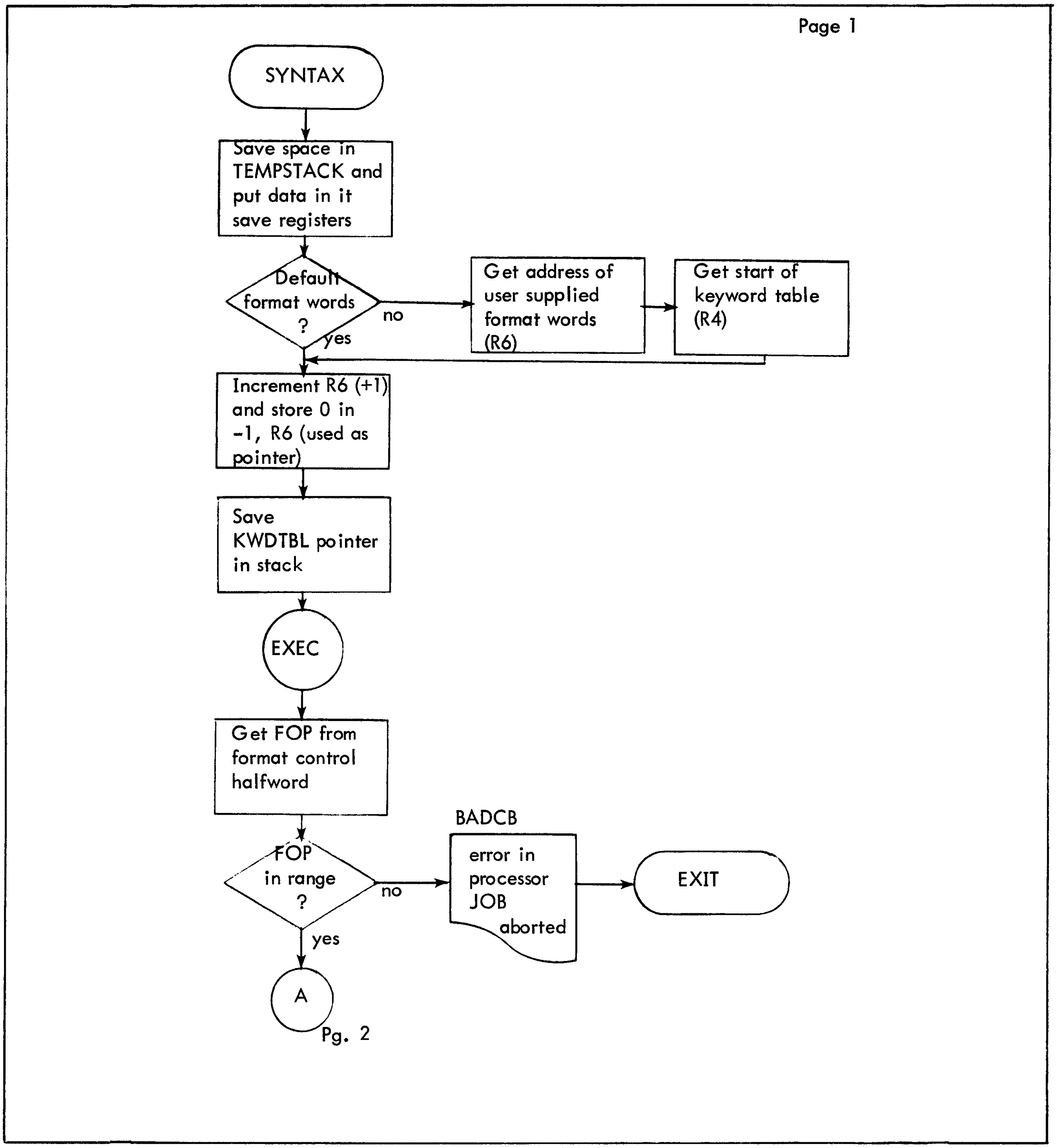
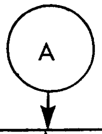

Page 1
SYNTAX
Save space in TEMPSTACK and put data in it save registers
Default format words ?
no
yes
Get address of user supplied format words (R6)
Get start of keyword table (R4)
Increment R6 (+1) and store 0 in -1, R6 (used as pointer)
Save KWDTBL pointer in stack
EXEC
Get FOP from format control halfword
FOP in range ?
no
yes
BADCB
error in processor JOB aborted
EXIT
A
Pg. 2

Figure 6-26. Flow Diagram of SYNTAX

Page 2



FOP=	GO TO	
0	EX1	Pg. 3
1	EX2	Pg. 3
2	LEFT	Pg. 4
3	FLEFT	Pg. 4
4	RIGHT	Pg. 4
5	FRIGHT	Pg. 4
6	COMMA	Pg. 4
7	FCOMMA	Pg. 4
8	INTEROPT	Pg. 8
9	KWD	Pg. 8
10	PROCKWD	Pg. 9
11	ANTXT	Pg. 5
12	ANTXTC	Pg. 5
13	DEC	Pg. 5
14	HEX	Pg. 5
15	CNVTXTC	Pg. 6
16	CNVDEC	Pg. 6
17	CNVHEX	Pg. 6
18	WDTBL	Pg. 7
19	DWTBL	Pg. 7
20	GETSTRG	Pg. 14

Figure 6-26 Flow Diagram of SYNTAX (Cont.)

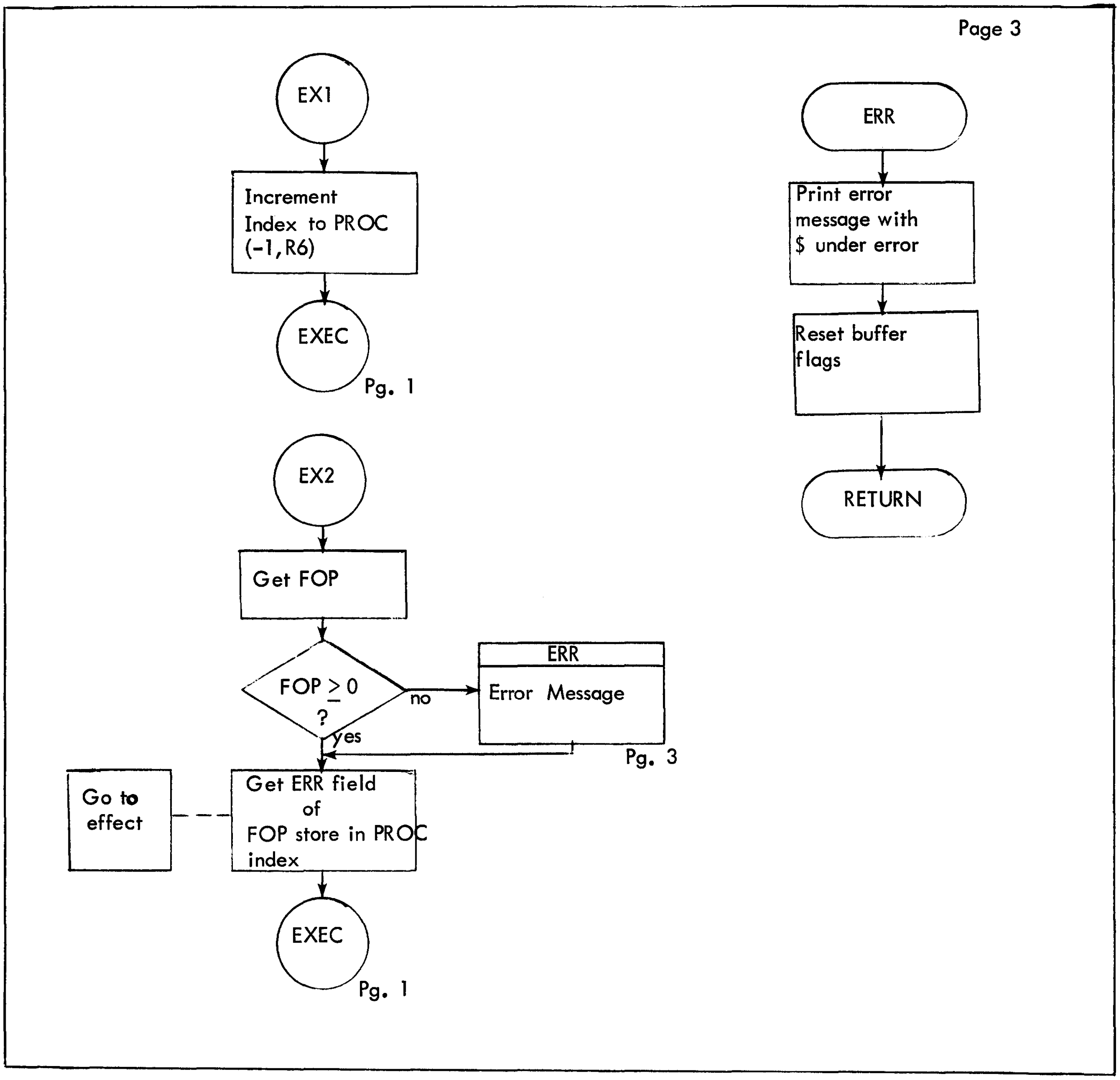


Figure 6-26 Flow Diagram of SYNTAX (Cont.)

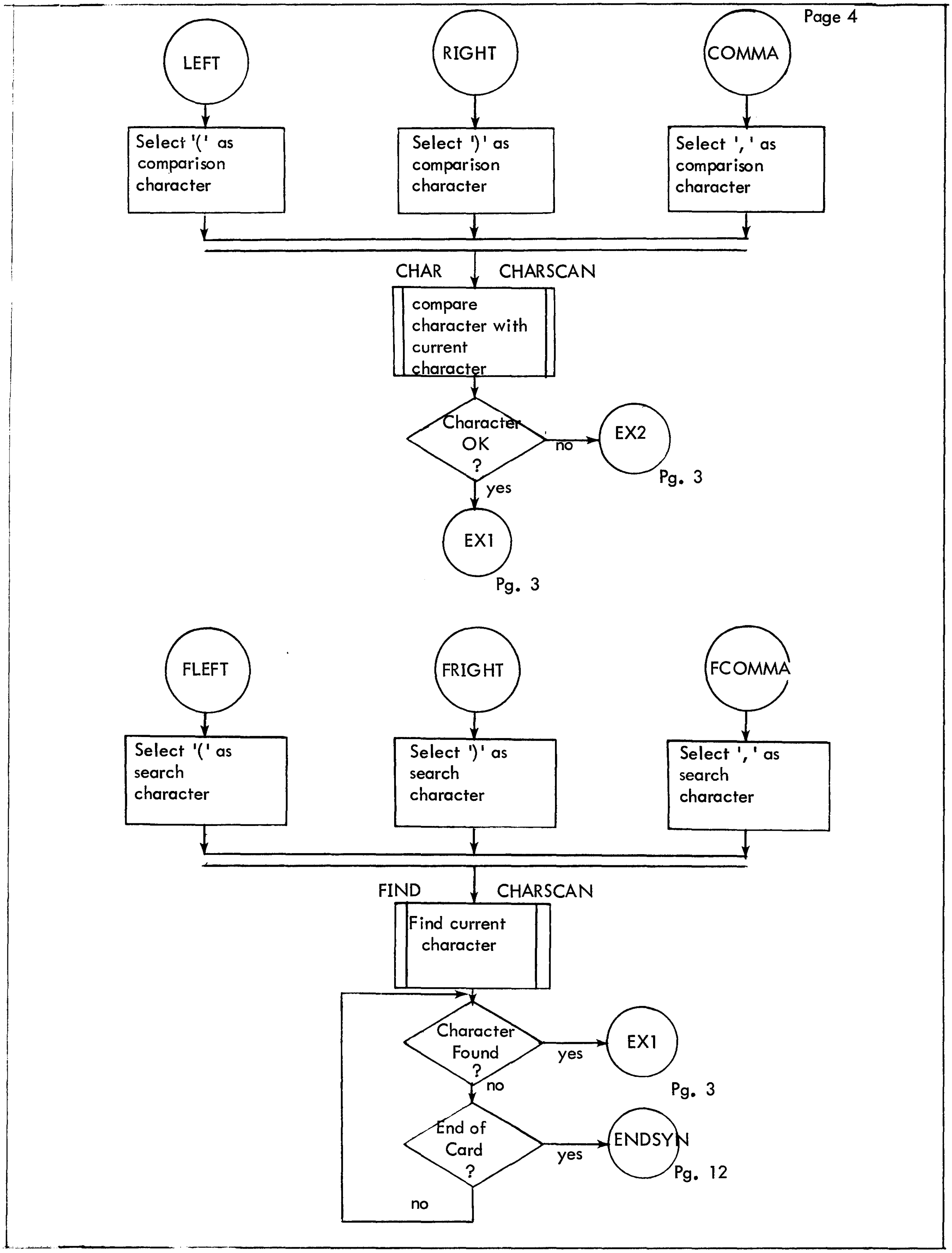

Page 4
LEFT
RIGHT
COMMA
Select '(' as comparison character
Select ')' as comparison character
Select ',' as comparison character
CHAR
CHARSCAN
compare character with current character
Character OK ?
no
EX2
Pg. 3
yes
EX1
Pg. 3
FLEFT
FRIGHT
FCOMMA
Select '(' as search character
Select ')' as search character
Select ',' as search character
FIND
CHARSCAN
Find current character
Character Found ?
yes
EX1
Pg. 3
no
End of Card ?
yes
ENDSYN
Pg. 12
no

Figure 6-26 Flow Diagram of SYNTAX (Cont.)

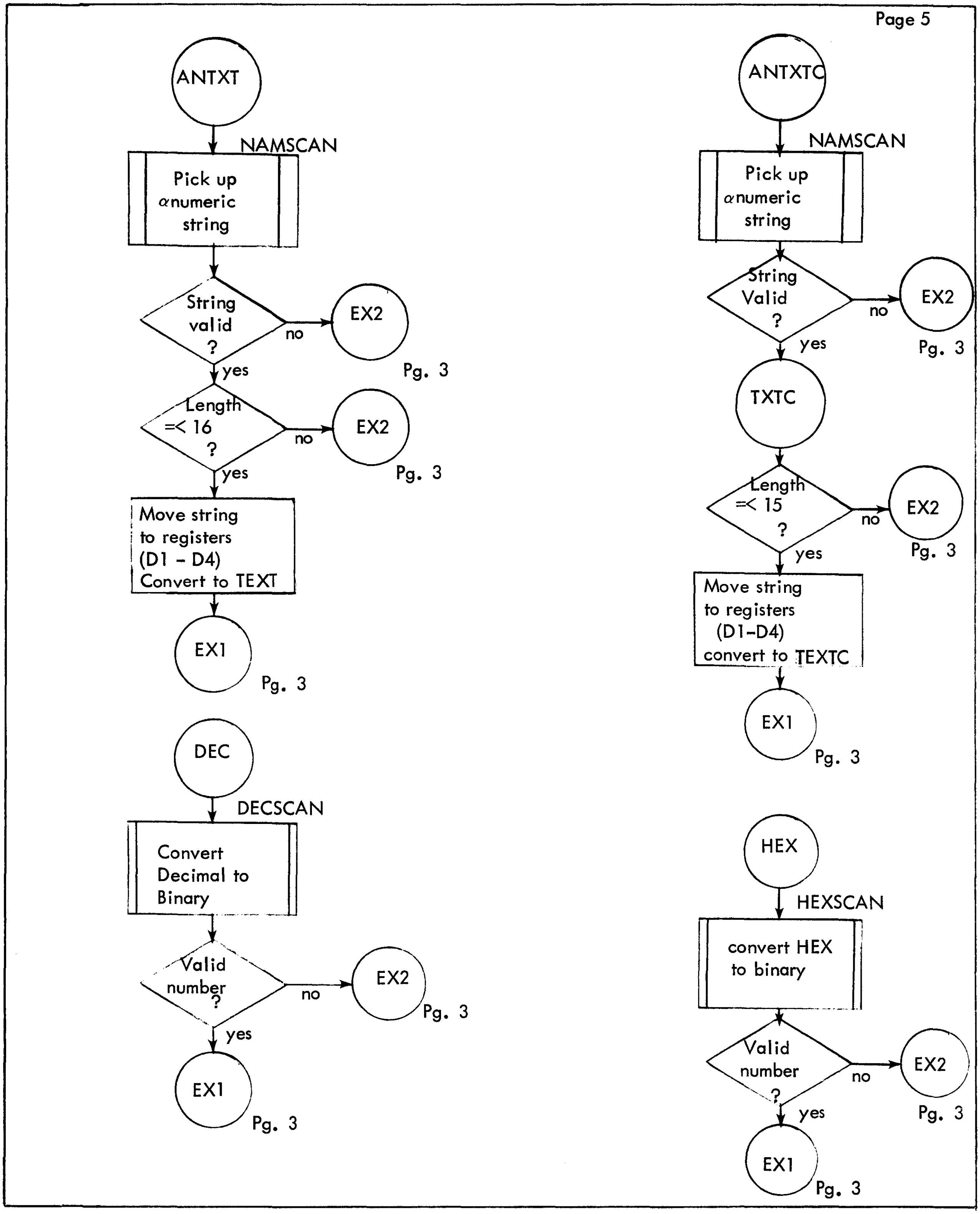


Figure 6-26 Flow Diagram of SYNTAX (Cont.)

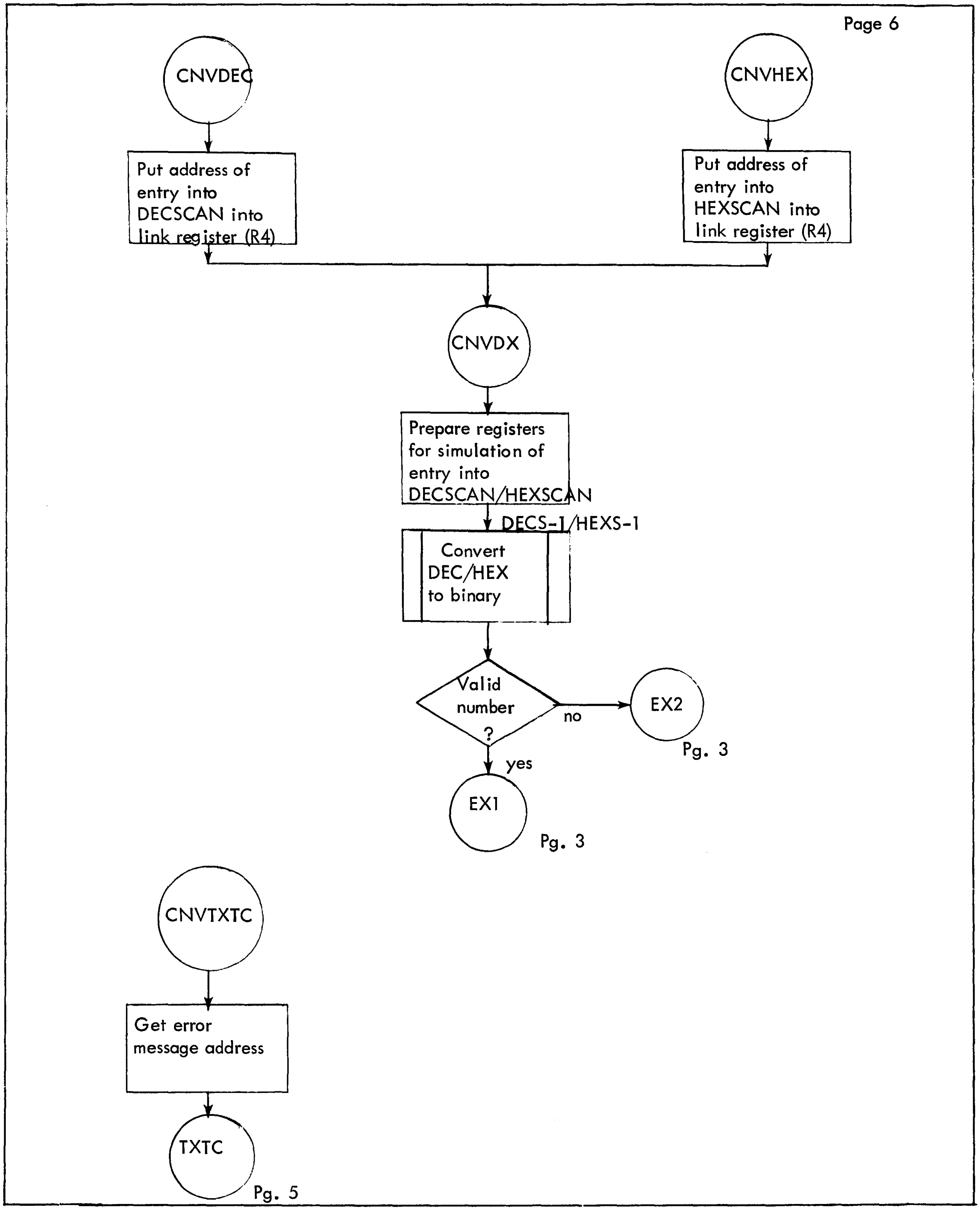

Page 6
CNVDEC
CNVHEX
Put address of entry into DECSCAN into link register (R4)
Put address of entry into HEXSCAN into link register (R4)
CNVDX
Prepare registers for simulation of entry into DECSCAN/HEXSCAN
DECS-1/HEXS-1
Convert DEC/HEX to binary
Valid number ?
no
EX2
Pg. 3
yes
EX1
Pg. 3
CNVTXTC
Get error message address
TXTC
Pg. 5

Figure 6-26 Flow Diagram of SYNTAX (Cont.)

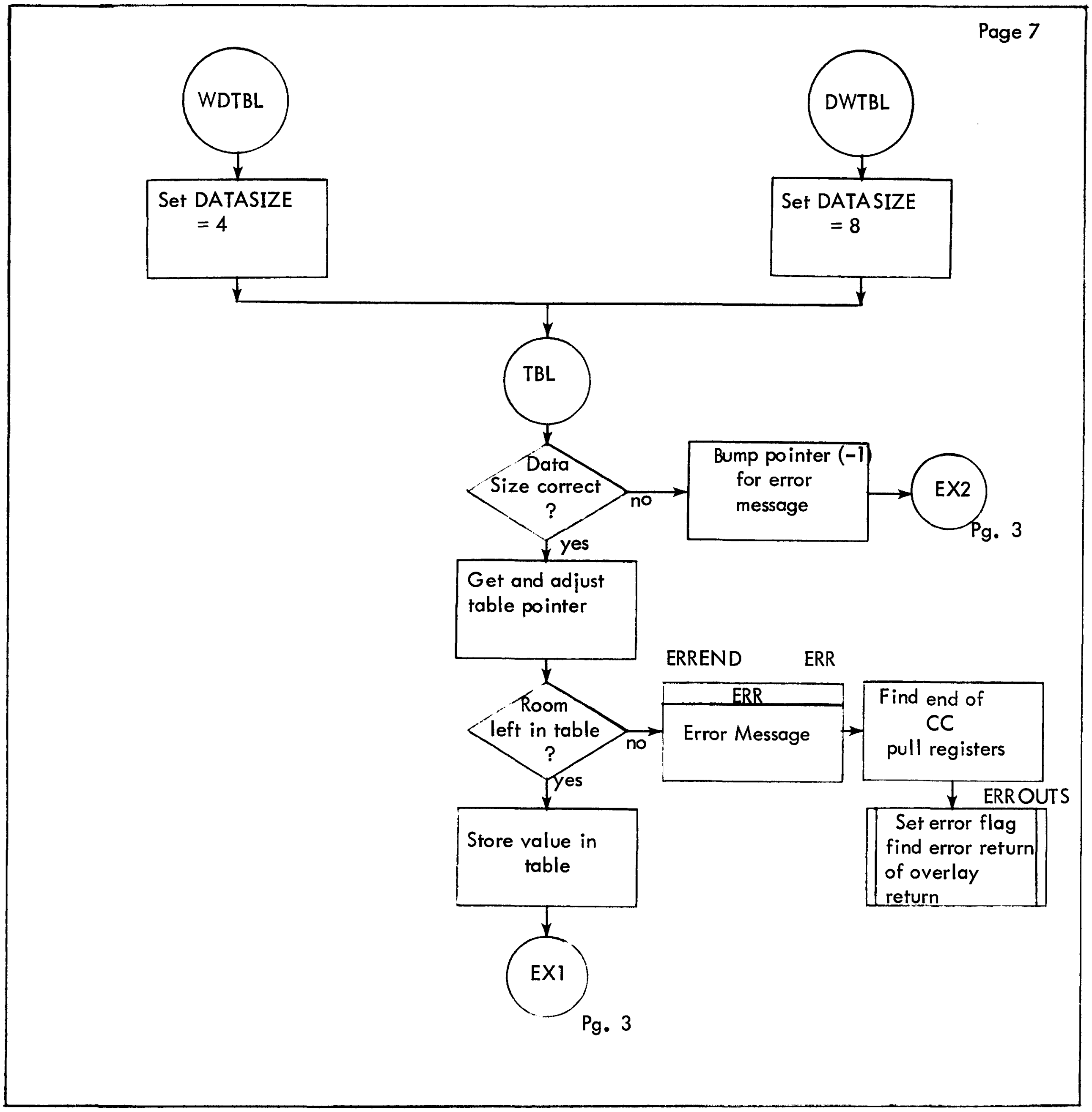


Figure 6-26 Flow Diagram of SYNTAX (Cont.)

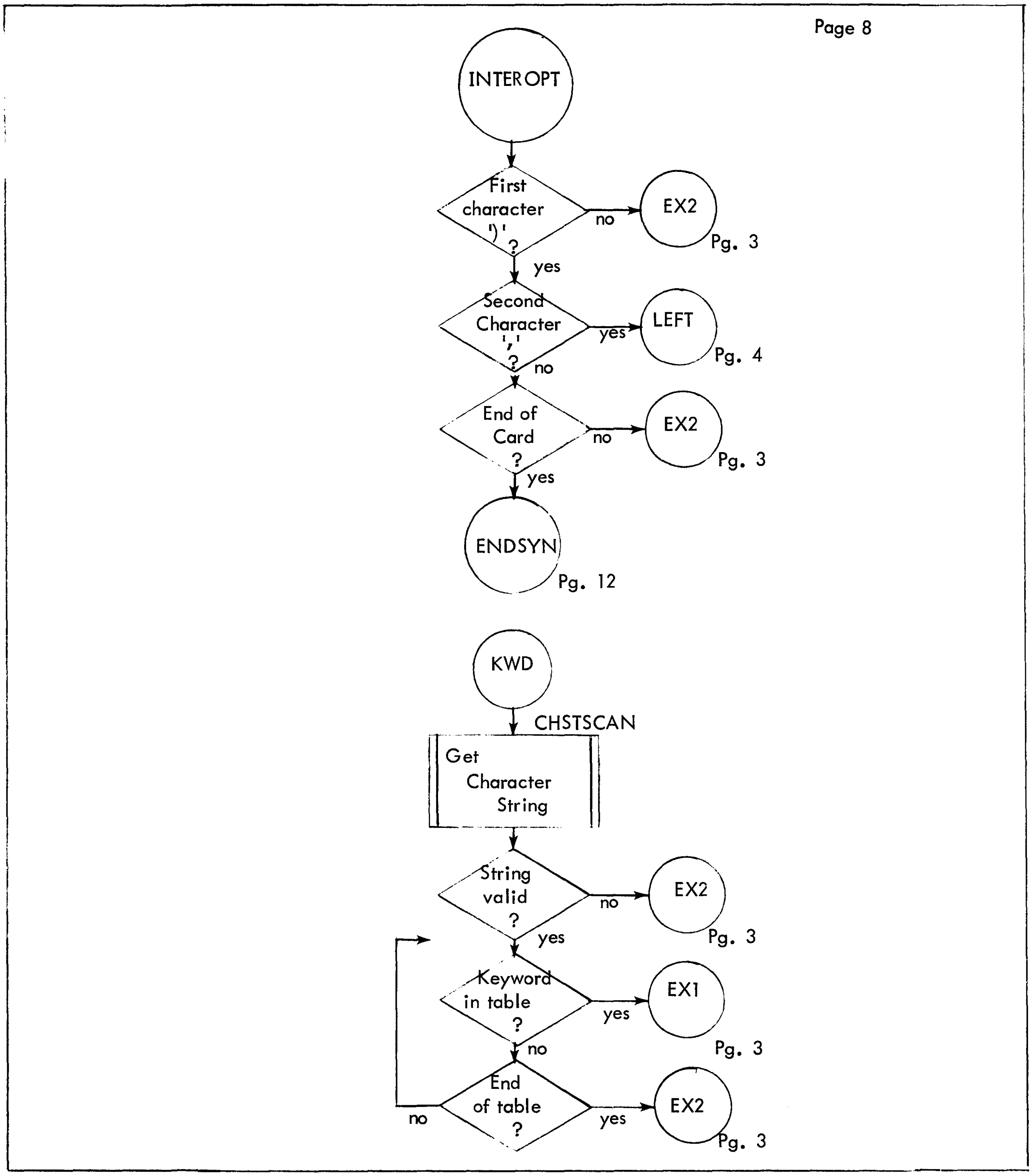


Figure 6-26 Flow Diagram of SYNTAX (Cont.)

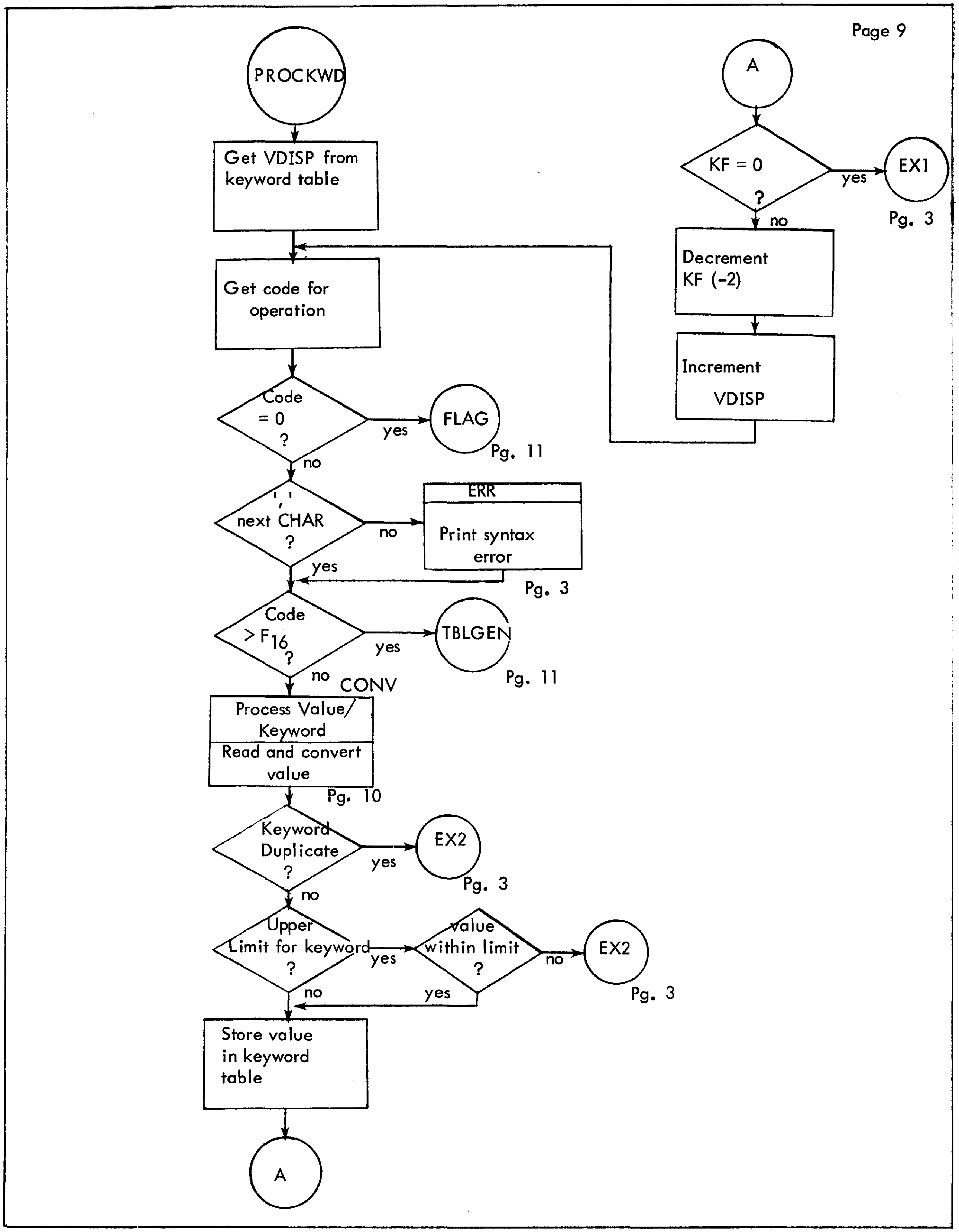
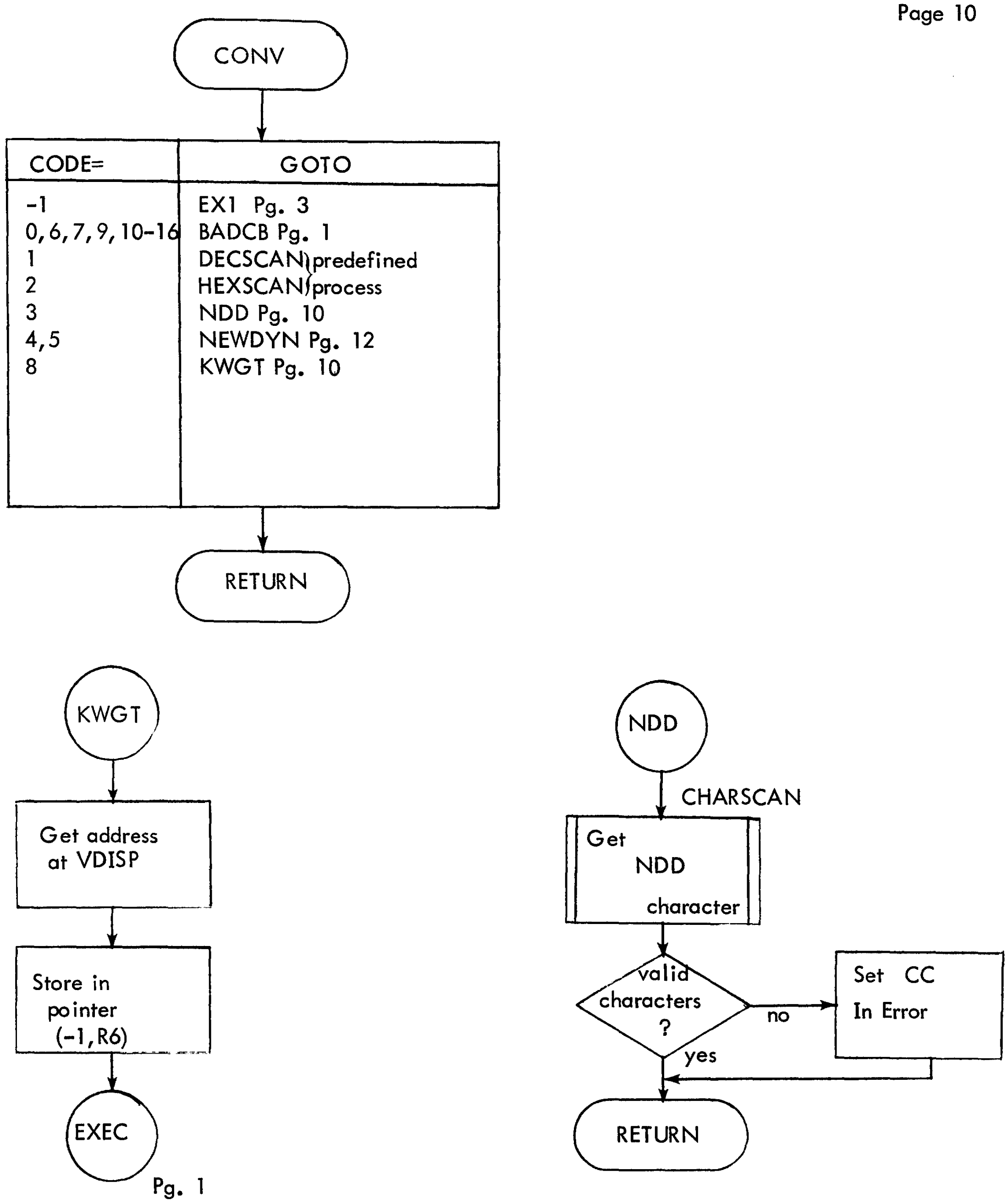


Figure 6-26 Flow Diagram of SYNTAX (Cont.)

Page 10



CODE=	GOTO
-1	EX1 Pg. 3
0,6,7,9,10-16	BADCB Pg. 1
1	DECSCAN} predefined
2	HEXSCAN} process
3	NDD Pg. 10
4,5	NEWDYN Pg. 12
8	KWGT Pg. 10

Figure 6-26 Flow Diagram of SYNTAX (Cont.)

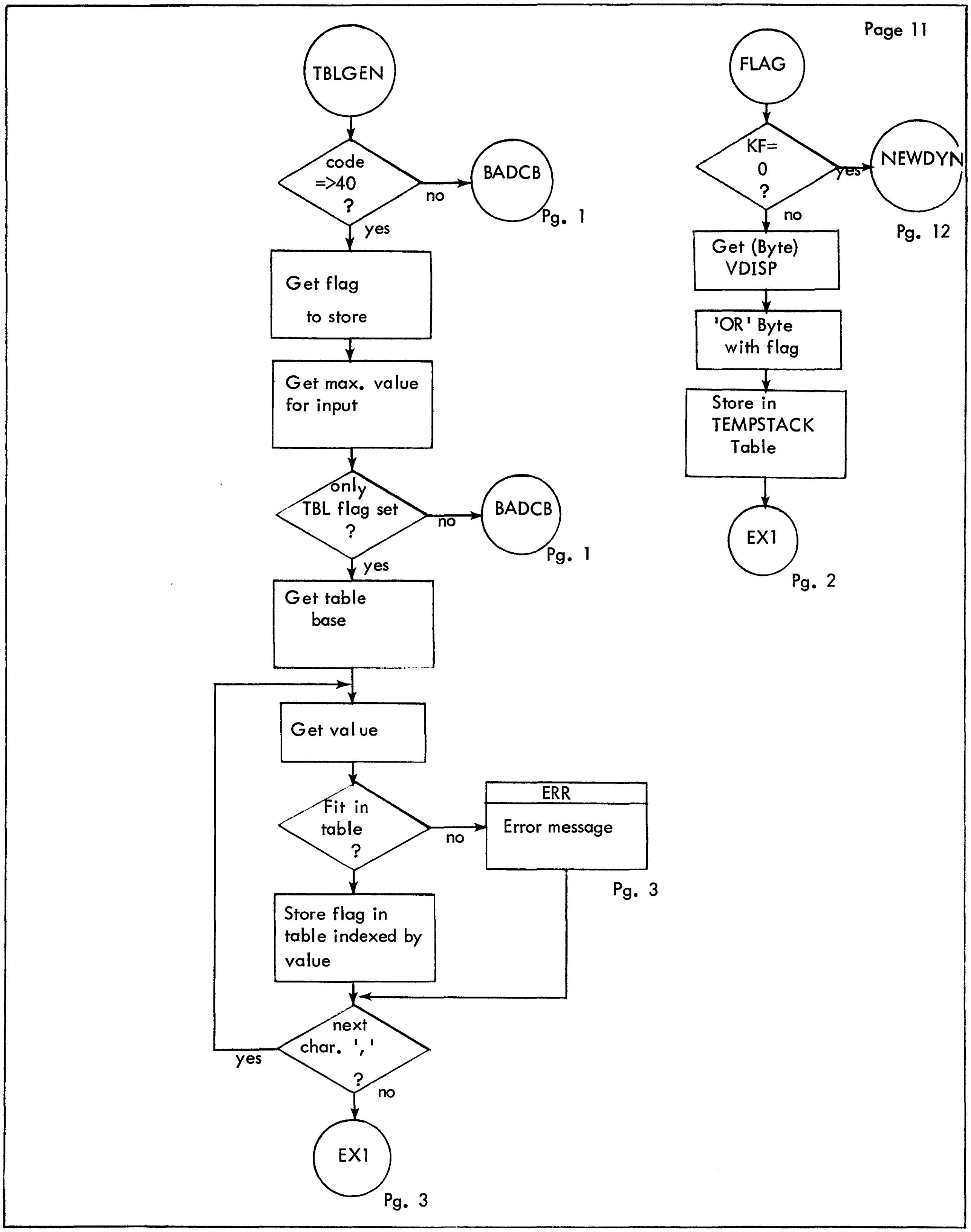


Figure 6-26 Flow Diagram of SYNTAX (Cont.)

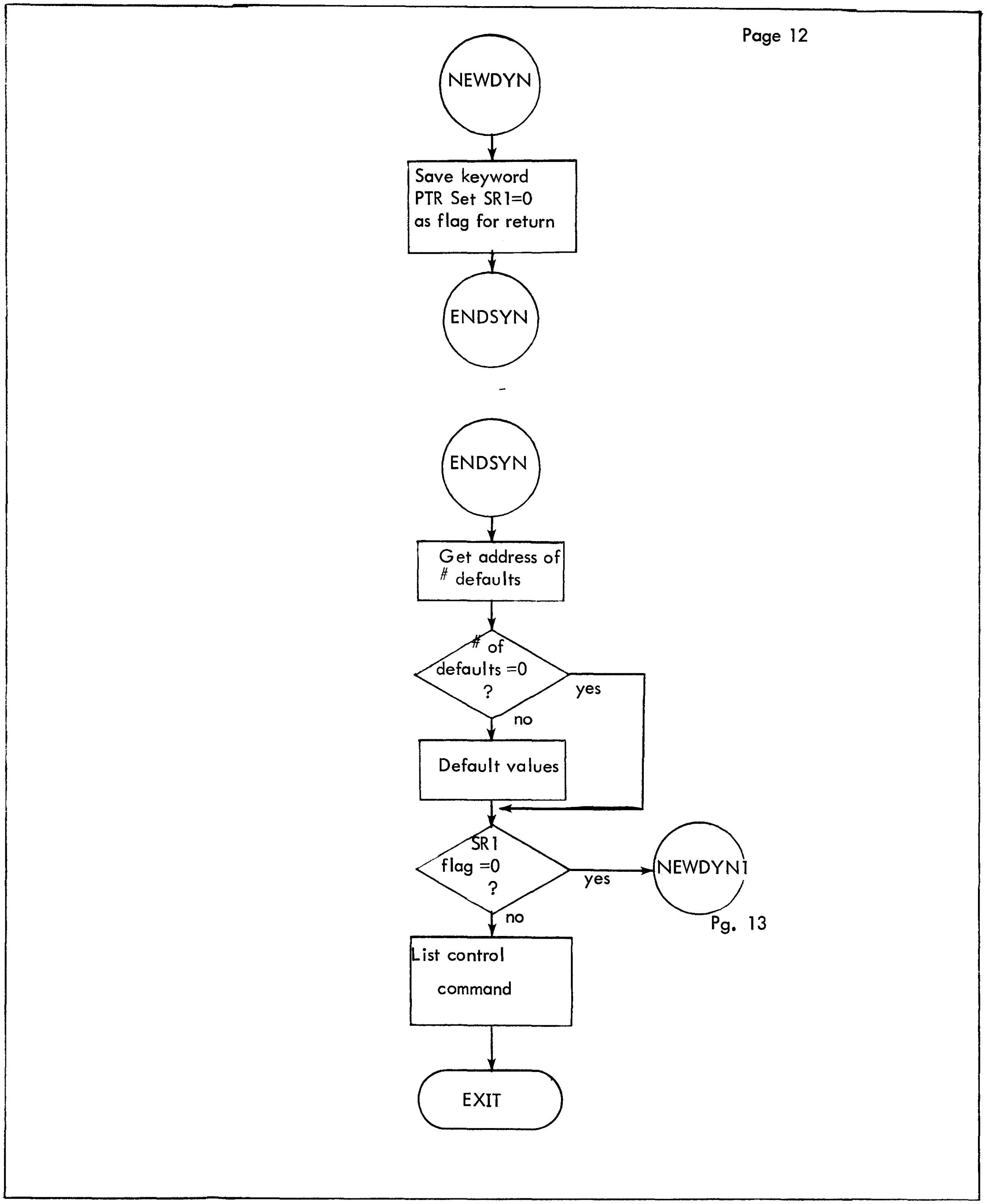


Figure 6-26 Flow Diagram of SYNTAX (Cont.)

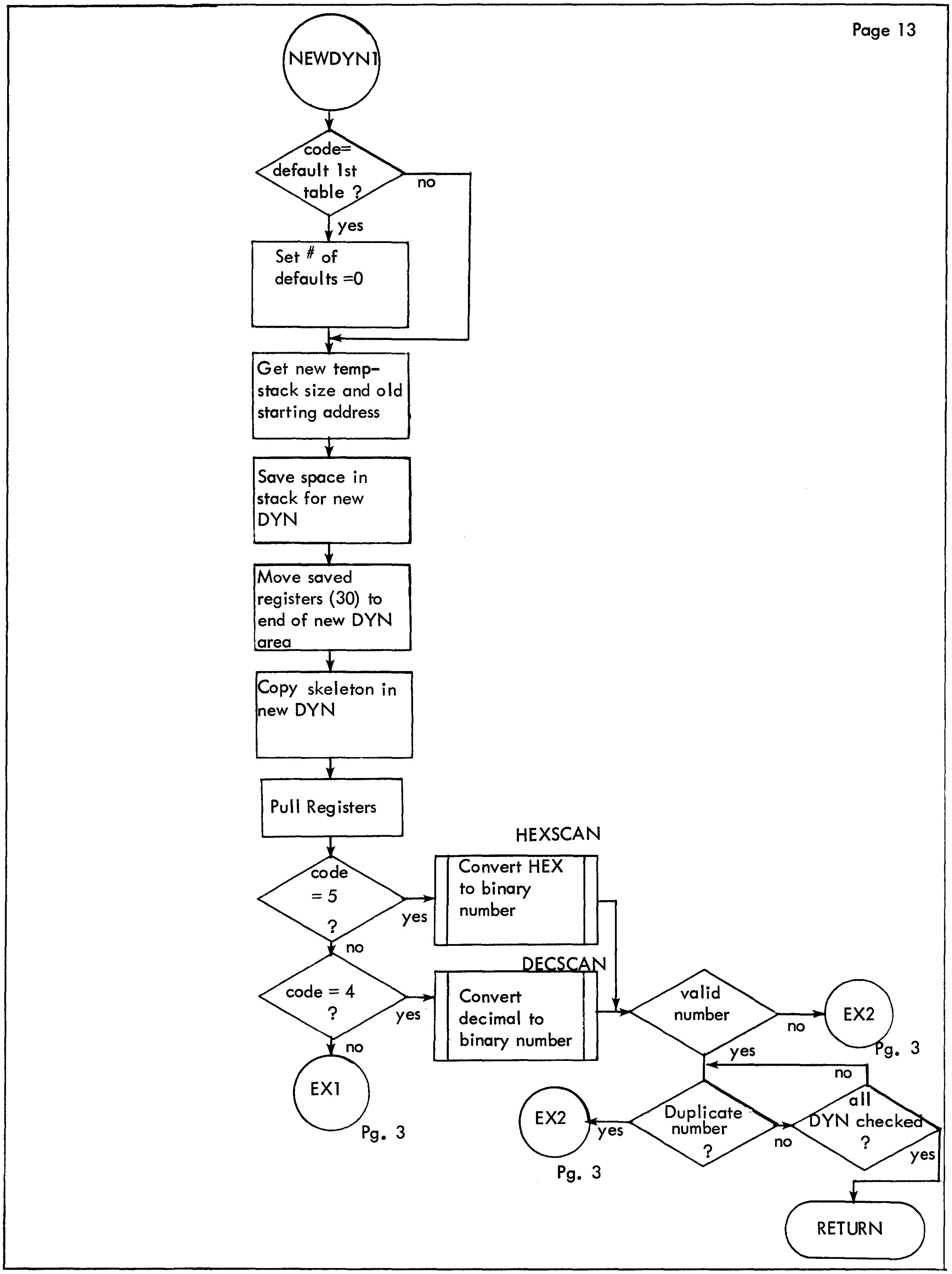


Figure 6-26. Flow Diagram of SYNTAX (Cont.)

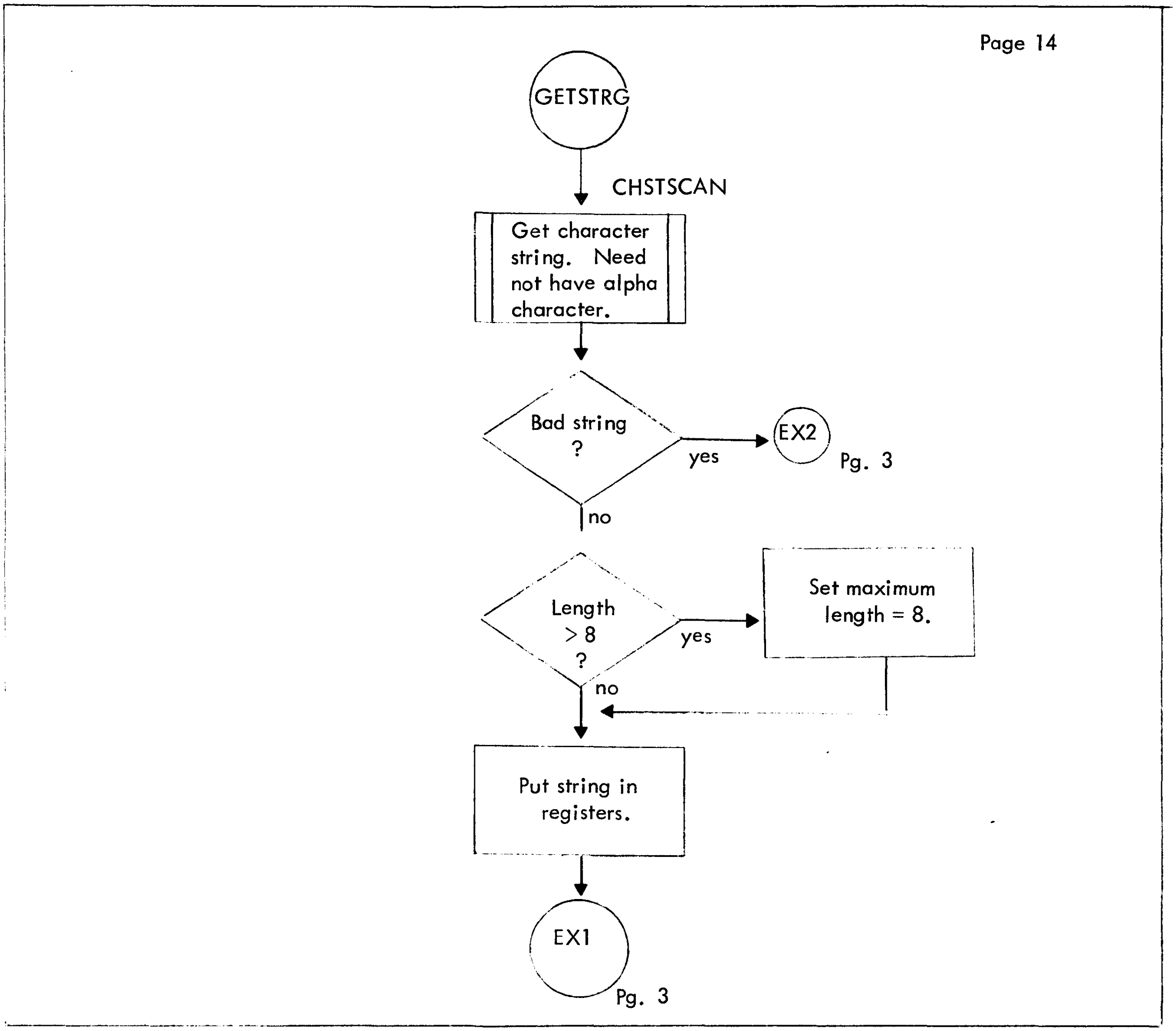


Figure 6-26. Flow Diagram of SYNTAX (Cont.)

6.9 MODGEN ROUTINE IN P2CCI (PASS2)

6.9.1 Purpose

To facilitate REFDEF stack, expression stack, and relocation dictionary changes (functions performed by the MODIFY module) for PASS2 module builders whose memory has been allocated by COREALLOC.

6.9.2 Usage

BAL, 10 MODGEN

MODGEN interpretively executes all code following the BAL, saving condition code and all registers (except R10, which is used to save the condition codes and/or the instruction address). Any successful branch instruction (BIR, BDR, BCS, or BCR) constitutes a return from MODGEN (except when used as a return to BAL+1 from a subroutine whose BAL was executed by MODGEN). Any instruction for which bits 0 to 3 are all reset is not executed but constitutes input data for MODGEN functions (See INPUT). A zero byte-0 error address word (see COREALLOC) must be present.

6.9.3 Input

All MODGEN specific input is in TEXTC format (maximum 15 characters) as follows:

Byte 0

n + 1	C_1	C_2	• • •
C_n	N	--	--

The type of operation is determined by N which is an EBCDIC decimal digit (X'F0' through X'F9')

N	OPERATION	MEANING OF $C..C_n$	COMMENT
0	Create value DEF	Name of DEF	Value in R12
1	Create location DEF	Name of DEF	Address (in data record) in R8
2	Change relocation dictionary	n=1, C_1=new relocation code	Address (in data) in R8
3	Create PREF for address field	C_1 through C_n-1 = name of REF; C_n = resolution	Address (in data) in R8

6.9.4 Interaction

MODIFY is used to make the modifications.

6.9.5 Errors

If a MODGEN operation (other than a value type DEF) is requested with the address in R8 larger than the highest valid address in the data record, the message

***INADEQUATE CORE SPACE - SKIP TO NEXT CC

is printed, R8 is set to minus one (as a flag for WRITELM), memory is released, and the word address in R10 is decremented until it points to a byte which is zero, which word is used for an error return address.

An identical sequence occurs if MODIFY reports an error except that the message is

***MODIFY ERROR - SKIP TO NEXT CC

If an illegal operation code is encountered (N not EBCDIC decimal or not implemented), MODGEN errors the job step with the message.

***ERROR IN PROCESSOR - JOB ABORTED

6.9.6 Description

MODGEN examines the word pointed to by R10. If any of bits 0-3 are set, the word is assumed to be an instruction and the condition codes are loaded from R10, the word is executed, the new condition codes stored in R10, and R10 is incremented. Otherwise, the operation code is picked up from the end of the TEXTC and a corresponding subroutine generates a change description table and performs the operation through MODIFY. R10 is incremented to point to the word after the TEXTC. This process is repeated until a successful branch instruction is encountered, since its execution transfers control out of MODGEN.

6.9.7 Flow Charts

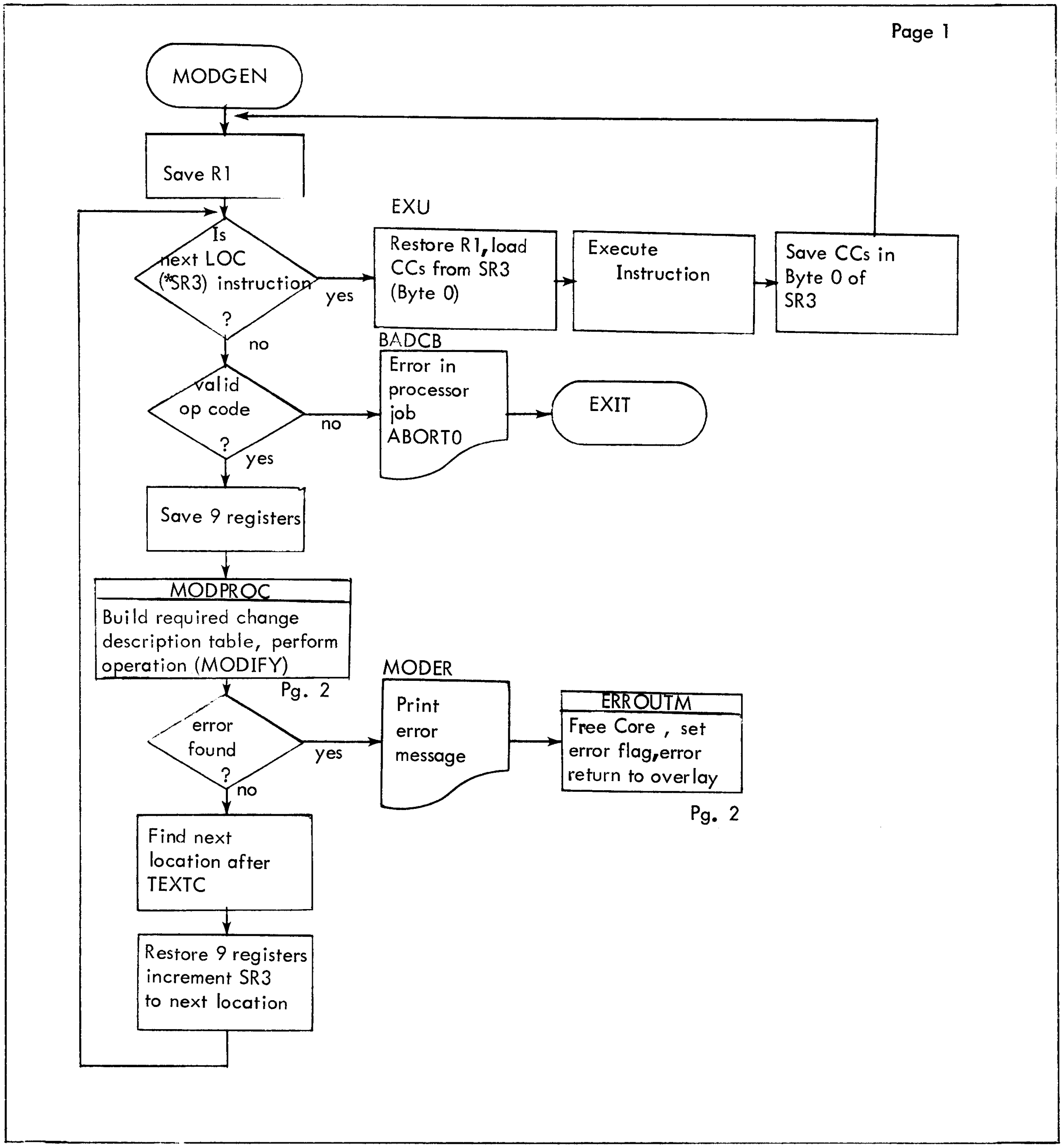
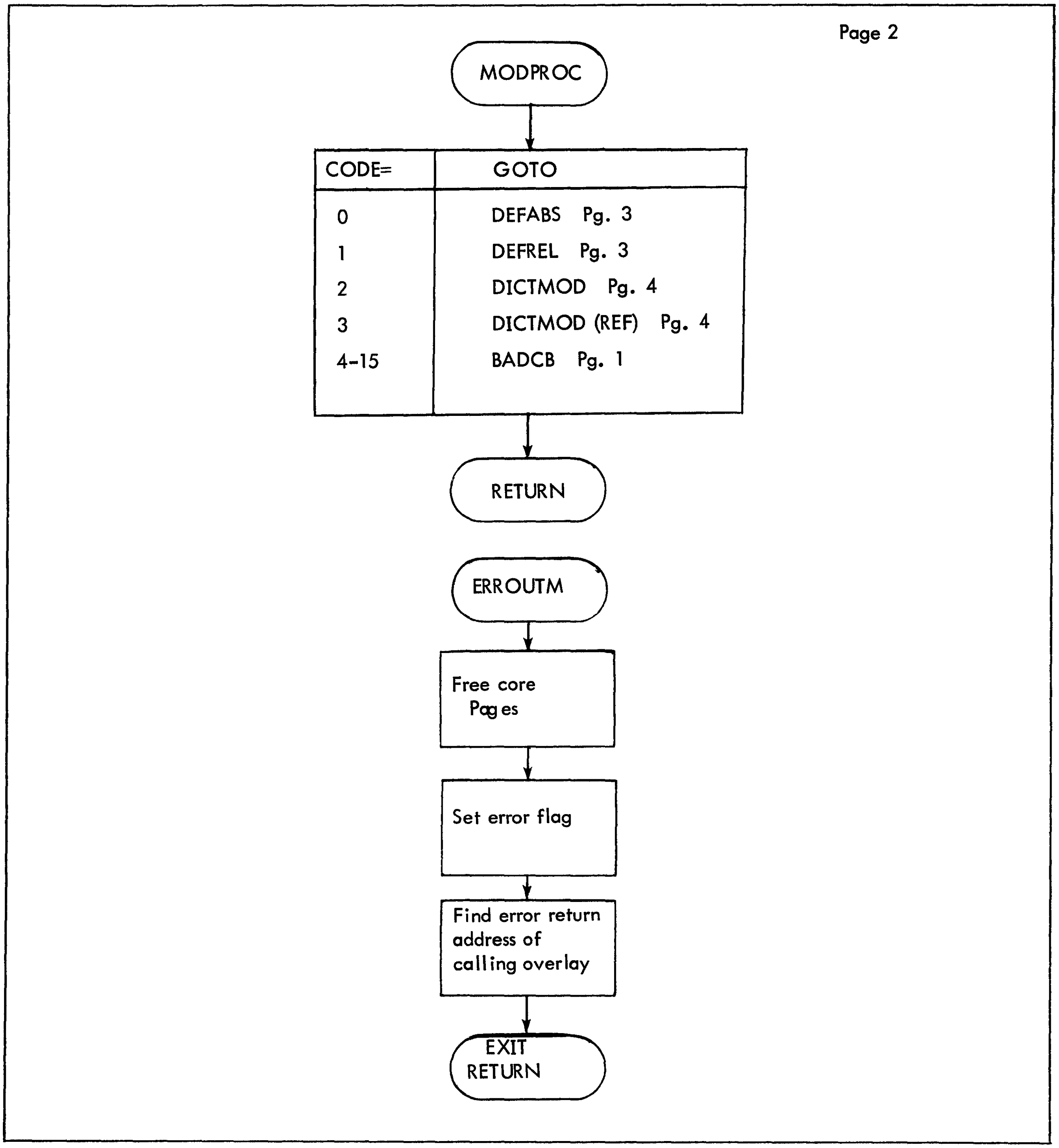

Page 1
MODGEN
Save R1
EXU
Is next LOC (*SR3) instruction ?
yes
no
Restore R1, load CCs from SR3 (Byte 0)
Execute Instruction
Save CCs in Byte 0 of SR3
BADCB
Error in processor job ABORTO
EXIT
valid op code ?
no
yes
Save 9 registers
MODPROC
Build required change description table, perform operation (MODIFY)
Pg. 2
MODER
error found ?
yes
no
Print error message
ERROUTM
Free Core , set error flag, error return to overlay
Pg. 2
Find next location after TEXTC
Restore 9 registers increment SR3 to next location

Figure 6-27 Flow Diagram of MODGEN



CODE=	GOTO
0	DEFABS Pg. 3
1	DEFREL Pg. 3
2	DICTMOD Pg. 4
3	DICTMOD (REF) Pg. 4
4-15	BADCB Pg. 1

Figure 6-27 Flow Diagram of MODGEN (Cont.)

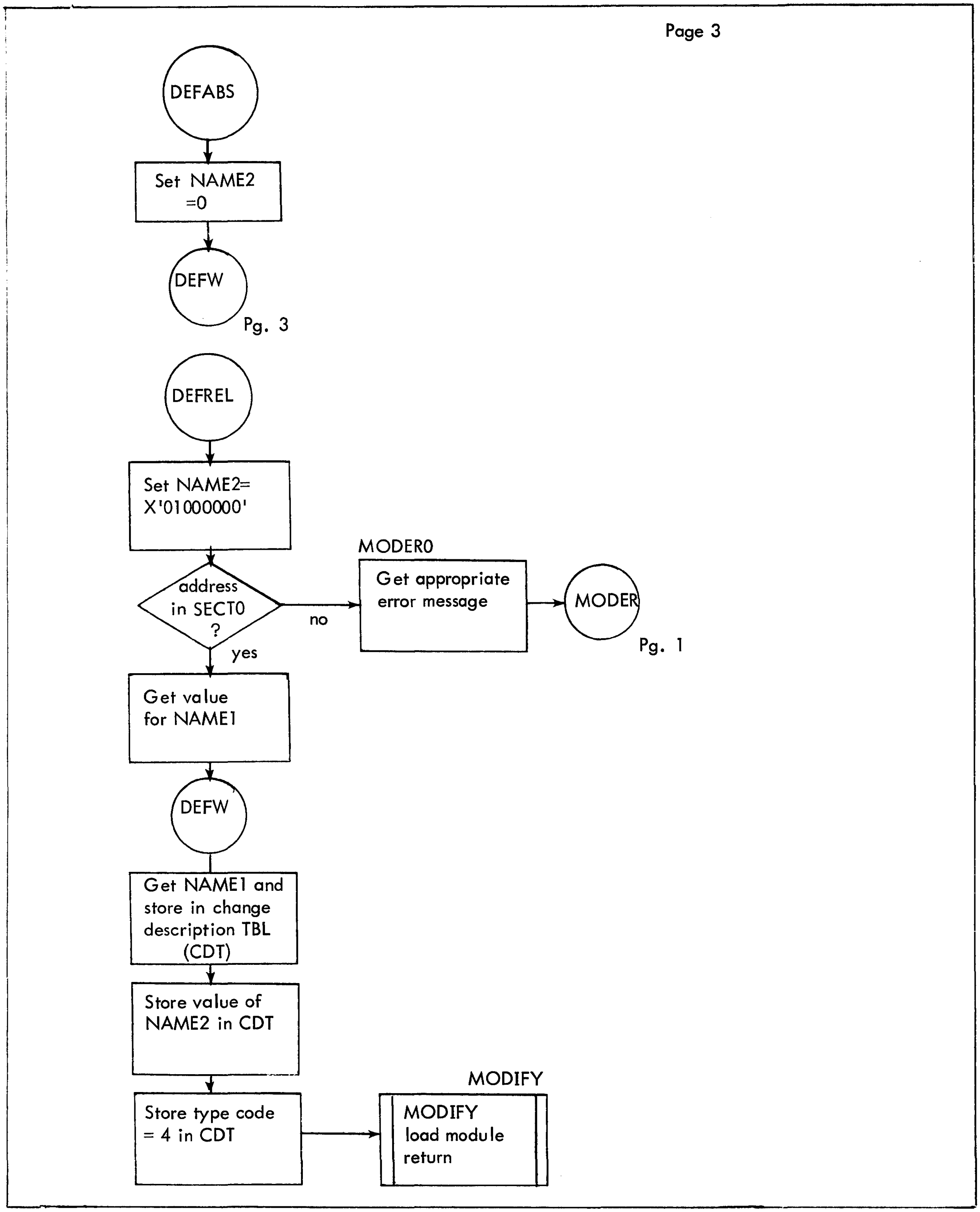

Page 3
DEFABS
Set NAME2 =0
DEFW
Pg. 3
DEFREL
Set NAME2= X'01000000'
address in SECT0 ?
no
yes
MODER0
Get appropriate error message
MODER
Pg. 1
Get value for NAME1
DEFW
Get NAME1 and store in change description TBL (CDT)
Store value of NAME2 in CDT
Store type code = 4 in CDT
MODIFY
MODIFY load module return

Figure 6-27 Flow Diagram of MODGEN (Cont.)

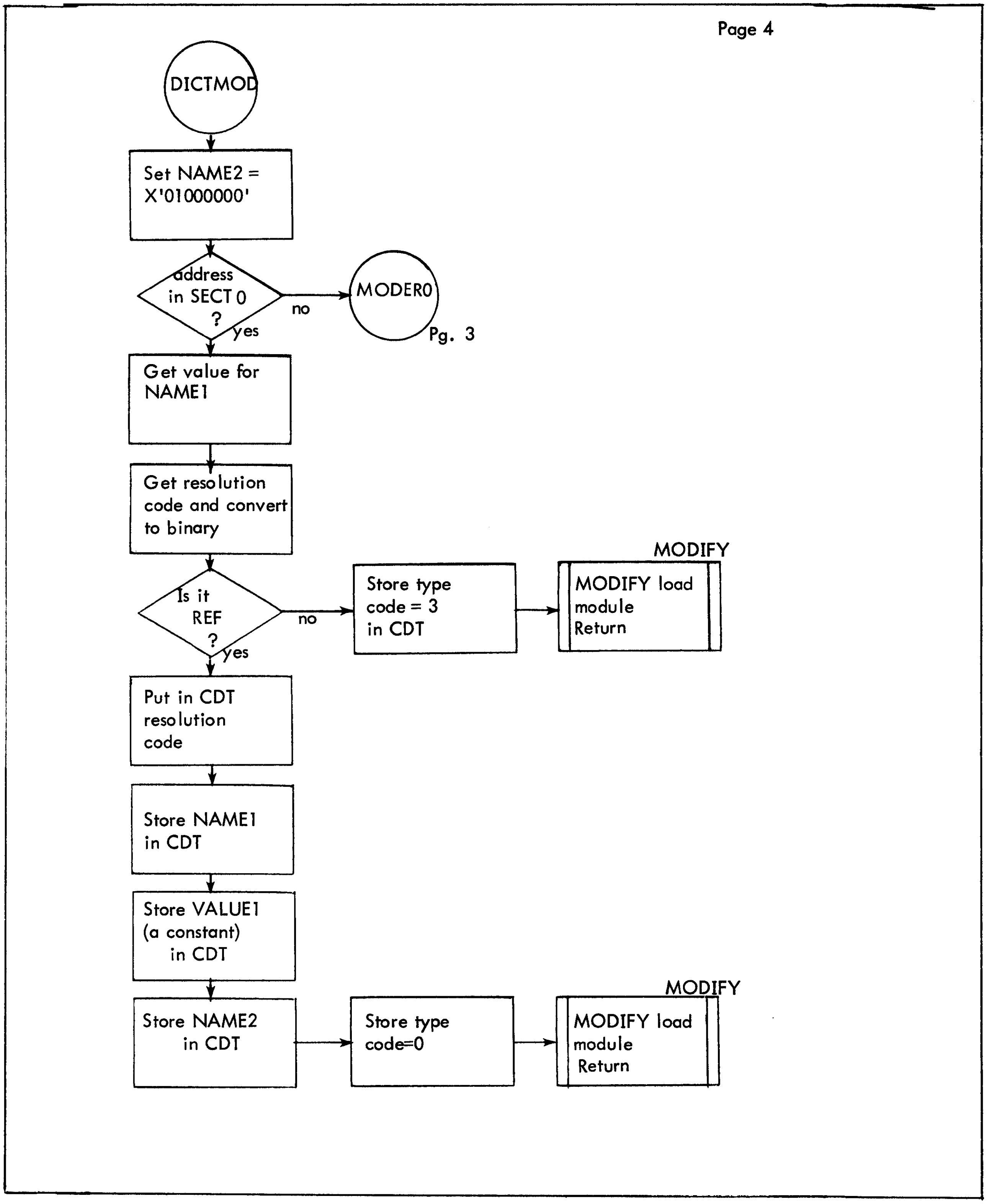

Page 4
DICTMOD
Set NAME2 = X'01000000'
address in SECT 0 ?
no
MODER0
Pg. 3
yes
Get value for NAME1
Get resolution code and convert to binary
Is it REF ?
no
Store type code = 3 in CDT
MODIFY
MODIFY load module Return
yes
Put in CDT resolution code
Store NAME1 in CDT
Store VALUE1 (a constant) in CDT
Store NAME2 in CDT
Store type code=0
MODIFY
MODIFY load module Return

Figure 6-27 Flow Diagram of MODGEN (Cont.)

6.10 COREALLOC ROUTINE IN P2CCI (PASS2)

6.10.1 Purpose

To obtain and allocate memory for building SYSGEN library load modules.

6.10.2 Usage

BAL, 11

(0) = Temp stack pointer address

(3) = P2CCI dynamic data address

(12) = Desired word size of REFDEF stack or -1, if unknown

(13) = Desired word size of data record or -1, if unknown

Upon exit:

(7) = Address MODIFY PLIST

(8) = Address of data record

(9) = Address of REFDEF stack.

4 and 14 are destroyed.

6.10.3 Output

Allocation of Memory:

◄———— Z words ————►

REFDEF Stack X words One 4-word entry Rest zero-filled	EXPRESSION Stack X words Zero-filled	Data Record Relocation Dictionary 'E' filled	Data Record 8 times Rel. Dict. size. Y words zero-filled	Unused W words

Higher addresses ——► Upper limit of memory

If neither R12 nor R13 is minus one:	X=(R12), Y=(R13), W=Z-2X-9Y/8
If only R12 is minus one:	Y=(R13), X=(Z-9Y/8)/2, W=0
If only R13 is minus one:	X=(R12), Y=(Z-2X) *8/9, W=0
If both R12 and R13 are minus one:	X=Z/3, Y=8Z/27, W=0

The entry in REFDEF defines the data base address.

In the Temp stack, COREALLOC builds a MODIFY PLIST, a ten-word Change Description Table area, a HEAD record, a TREE record, and an M:OPEN FPT, and one word (MAX00 for the word address of the end of the data). Each of these has appropriate address and size fields filled in.

6.10.4 Input

The highest addressed word before the BAL to COREALLOC which contains a zero in byte-0. This word must contain the address of code which cleans up if sufficient memory not available and contains a BAL to WRITELM which releases stack space used by COREALLOC.

6.10.5 Interaction

M:GP, M:FP to obtain and release memory.

6.10.6 Errors

If available memory is insufficient, the message:

    ***INADEQUATE CORE SPACE-SKIP TO NEXT CC

is printed, (R8) is set to minus one as a flag for WRITELM, and a return is made to the address described under INPUT.

6.10.7 Description

COREALLOC first sets up R7 and moves a blank data area (PLIST, HEAD, TREE) to the stack. All available memory is obtained. The sizes of the various sections are calculated according to the formulas under OUTPUT. If enough memory is available, all the appropriate slots in PLIST, HEAD, and TREE are filled in, the memory is set up, and COREALLOC returns.

6.10.8 Flow Chart

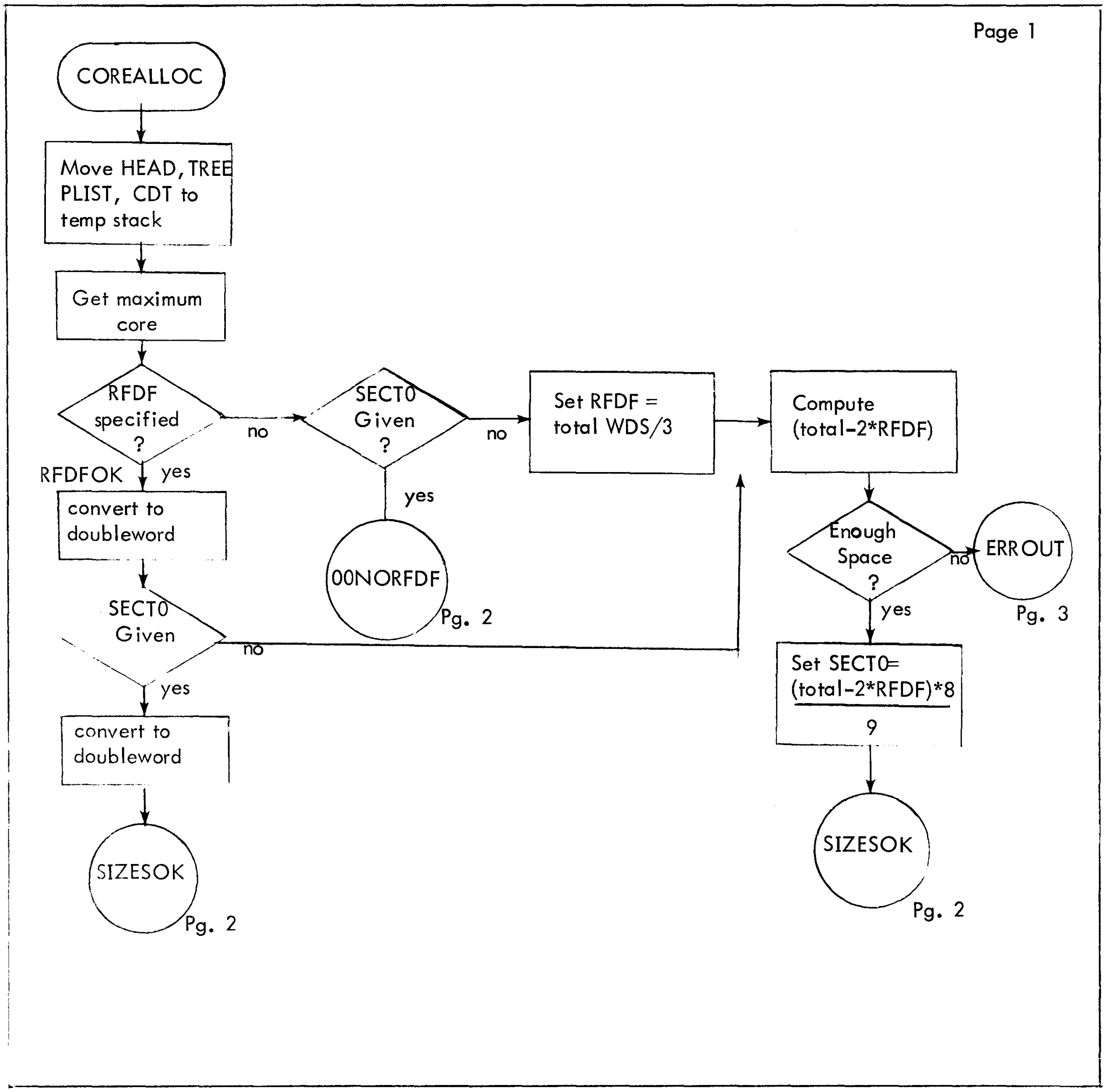


Figure 6-28. Flow Diagram of COREALLOC

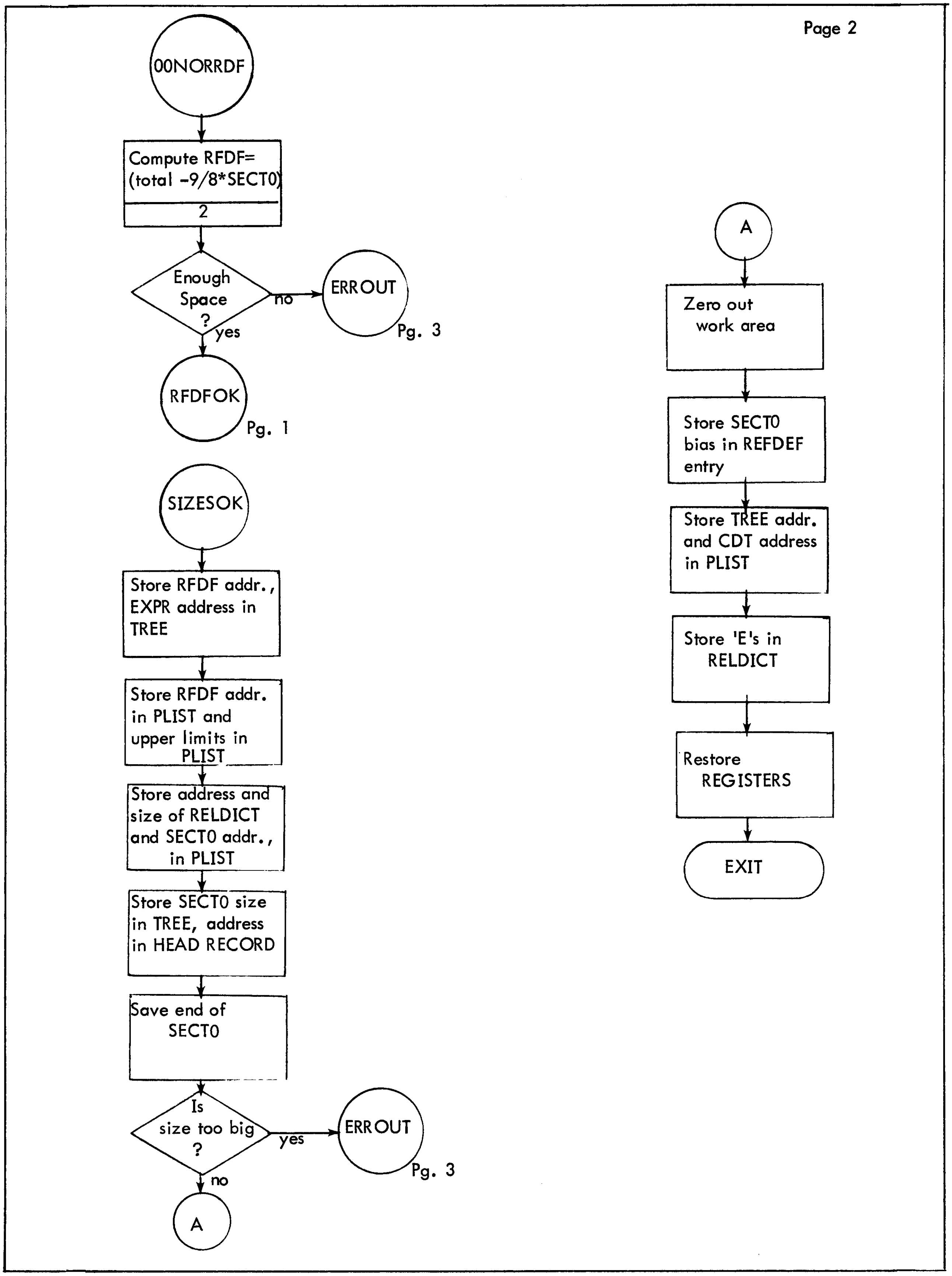


Figure 6-28. Flow Diagram of COREALLOC (Cont.)

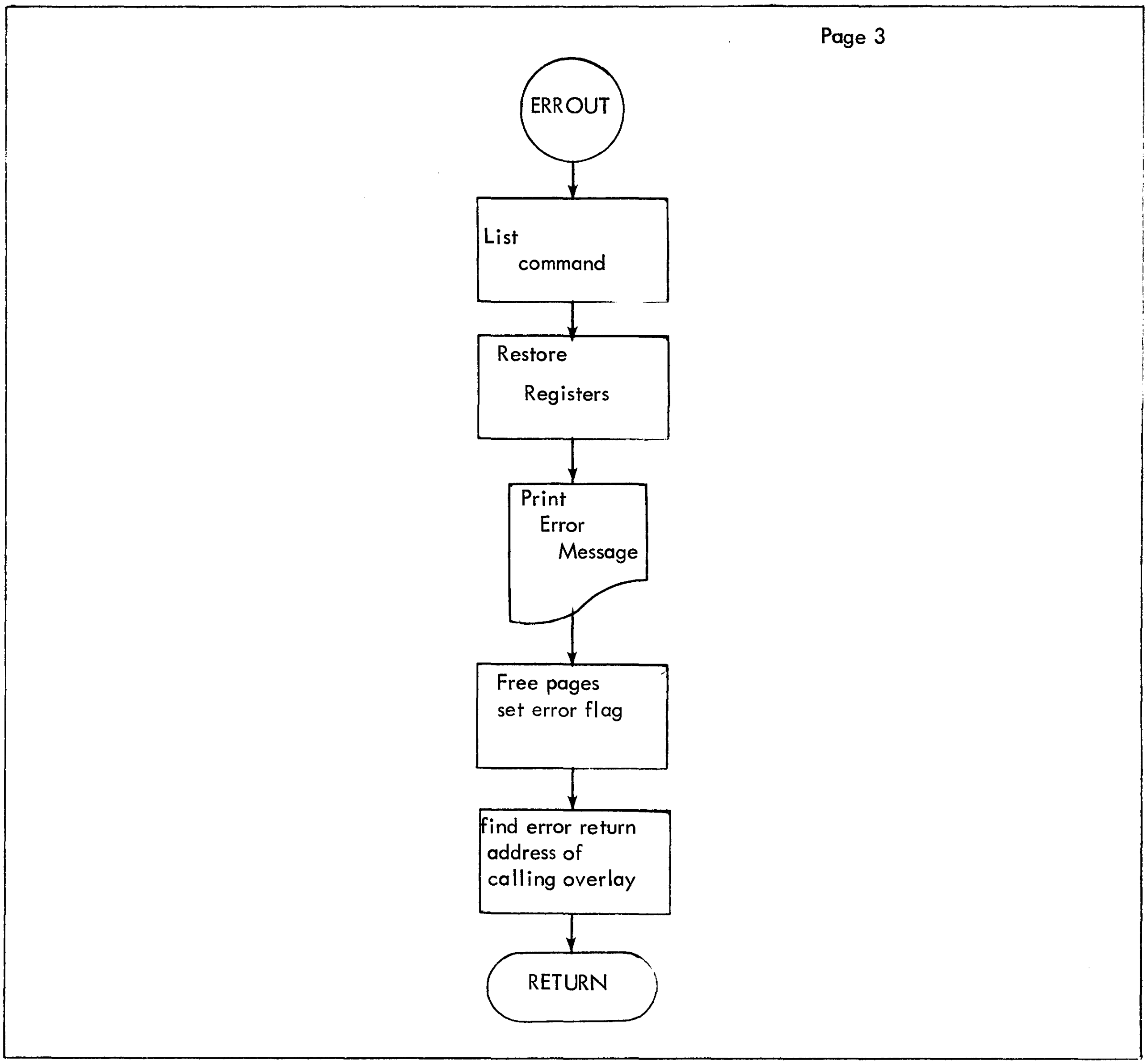

Page 3
ERROUT
List command
Restore Registers
Print Error Message
Free pages set error flag
find error return address of calling overlay
RETURN

Figure 6-28 Flow Diagram of COREALLOC (Cont.)

6.11 WRITELM ROUTINE IN P2CCI (PASS2)

6.11.1 Purpose

To write a SYSGEN library load module file whose memory has been allocated by COREALLOC.

6.11.2 Usage

BAL, 11

(0) = Temp stack pointer address

(7) = Modify PLIST address (also used to find HEAD, TREE, and M:OPEN FPT)

(8) = Actual end of data record or minus one to skip the write for errors.

(14) = Address of TEXTC file name.

All registers saved except 12, which is used to reduce the temp stack to pre-COREALLOC status.

6.11.3 Interaction

M:OPEN, M:WRITE, M:CLOSE to create the file. M:FP to release memory obtained by COREALLOC.

6.11.4 Description

If R8 contains minus one, the stack is pulled to pre-COREALLOC status and WRITELM returns. If not, the actual data record size is put into the TREE and the HEAD is adjusted according to the running monitor type The first entry in the REFDEF stack (put there by COREALLOC) is replaced with a CSECT entry, the filename (from R14) is put in the TREE, and the OPEN FPT and the file is opened in the output mode. Then the HEAD, TREE, REFDEF stack, Expression stack, relocation dictionary, and data are written to the file, which is then closed and saved. Memory is released, the stack is pulled, and WRITELM returns.

6.11.5 Flow Chart

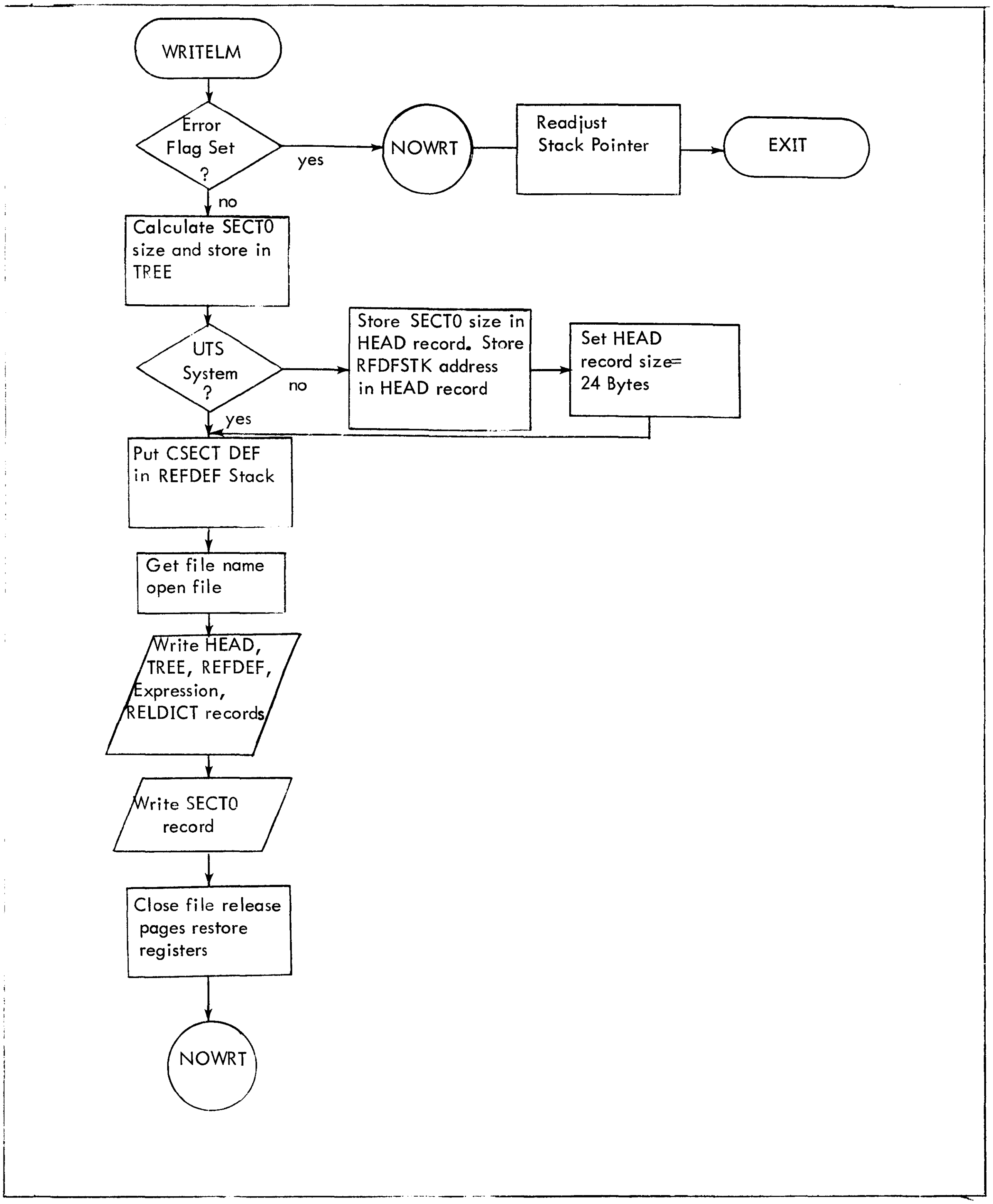

WRITELM
Error Flag Set ?
yes
NOWRT
Readjust Stack Pointer
EXIT
no
Calculate SECT0 size and store in TREE
UTS System ?
no
Store SECT0 size in HEAD record. Store RFDFSTK address in HEAD record
Set HEAD record size= 24 Bytes
yes
Put CSECT DEF in REFDEF Stack
Get file name open file
Write HEAD, TREE, REFDEF, Expression, RELDICT records
Write SECT0 record
Close file release pages restore registers
NOWRT

Figure 6-29 Flow Diagram of WRITELM

APPENDIX A

A.0 BPMBT – DEF OVERLAY

A.1 PURPOSE

To write a bootable BPM monitor to either 9-track tape, 7-track tape or disk pack.

A.2 CALLING SEQUENCE

BAL instruction from DEF

BAL, 12   BPMBT

A.3 INPUT

R6 = address of the parameter list control word to open the input DCB to read the monitor

R7 = address of DCB for writing BO/PO tape/disk

R0 = push down stack pointer address

A.4 OUTPUT

Error messages to LL device bootable monitor

A.5 CORE USAGE (not to scale)

	BPMBT OVERLAY					WORK AREA		
DEF Processor Root	WRITEMON (Procedure)	Mag Tape Mini-Boot	CDWs for DISKLOAD	Boot Subroutine	System Device Boot Routine	To Read MON Head & Tree (1 Page)	To Read MON Root	
							SGMT Names and #'s (1 Page)	To Read Monitor SEGMESTS

A.6 OVERVIEW

A.6.1 Description

BPMBT consists of two sections; both sections are contained in one ROM. Section I is the portion of BPMBT that is executed as part of the DEF processor to write a bootable monitor. This section of BPMBT is executed under the BPM operating system. Section II contains bootstrap routines and the Boot Subroutine. This section of BPMBT is written to the boot device along with the bootable monitor by Section I. The coding in this section is executed in the master mode at boot time in order to bring up the BPM system.

A.6.2 Module Organization – BPMPT

I WRITEMON (writes a bootable monitor)

II BOOTMON (routines to boot the monitor)

- A. Magnetic Tape Bootstrap
- B. CDWs for Disk Pack Boot Deck
- C. Boot Subroutine
  - 1. RDROOTTP (boots monitor from tape)
  - 2. RDROOTDP (boots monitor from disk pack)

3. WRTROOT (writes monitor to system device)
4. WRTLOOP (reads overlays and writes them to system device)
5. Various BOOTMON Subroutines
6. System Device Bootstrap

Only the WRITEMON portion (Section I) of BPMBT is documented in detail here. For details concerning the BOOTMON portion (Section II of BPMBT) refer to the BPM Technical Manual, 90 15 28, Chapter 3.

A.7 DESCRIPTION

Upon entry, WRITEMON issues an M:GP to get all of core to use as a work area. WRITEMON then opens M:TM DCB to the keyed file M:MON. The account specified is determined by the parameter list control word which is passed to WRITEMON by DEF. If creating a BO device this is the :SYS account; if PO, the current account. After M:MON has been opened, WRITEMON reads the keyed record 'HEAD' to get the start address. This start address has been defined by the END INITIAL Metasymbol directive in the module M:TABLES. The stack address is stored in the Boot-Subroutine to be used as an entry point to the Monitor initialization routine after the Monitor root has been read into core by the Boot-Subroutine.

Next the keyed records 'TREE' and MON::ORG are read. WRITEMON then accesses the size of the Monitor root from the Tree Table and stores the byte size of the root in the System Device Boot Routine. This address also is used as the buffer address to read the Monitor Tree Table at boot time and is stored within the Boot Subroutine in the CDW used to read the tree.

The Monitor root is to be written out in 2048 byte segments on either tape or disk pack. To avoid the increased possibility of tape read errors caused by reading an extremely short record the last segment is always written using a minimum of 40 bytes. The Boot Subroutine must know how many of these segments to expect and the size of the last segment. So WRITEMON makes these calculations at this point and stores the result in the Boot Subroutine.

During the initiai boot process the Boot Subroutine will be read (either by the mag-tape mini boot or DISKLOAD) into core location TOPRT + 1 page. Again, using the size of the Monitor root as accessed from the tree table, WRITEMON calculates this address and stores it into the CDWs used to read the Boot Subroutine in both the mag-tape mini-boot and the CDWs set up for DISKLOAD. This address is also stored at all entry points to the Boot Subroutine so that it may be accessed and used as a base register by the various routines within the Boot Subroutine.

A.8 BPMBT SUBROUTINES

A.8.1 WRSEG

A.8.1.1 Purpose

To write the Monitor Root and Overlay Segments to the output device.

A.8.1.2 Calling Sequence

```
BAL,11    WRSEG
```

A.8.1.3 Input

Reg. 4 = Number of bytes to be written
Reg. 14 = Address of buffer

A.8.1.4 Description

The WRSEG routine writes out the buffer in 2048 byte segments to the output tape or pack. Following each write operation the cell CURBLOCK is updated. This cell is used for each subsequent write operation to indicate the BLOCK value when the output device is pack.

A.8.2 DISPSEG

A.8.2.1 Purpose

To build the Segment Names and Numbers Record and list it on the LL Device.

A.8.2.2 Calling Sequence

```
BAL, 11     DISPSEG
```

A.8.2.3 Input

Reg. 1 = Segment Number (binary)

Reg. 2 = Address of Segment Name

Reg. 13 = Address of work area to build the record

A.8.2.4 Description

If the overlay number contained in R1 is '3F', DISPSEG puts out a standard segment message for the Root. Otherwise, the value in R1 is converted to EBCDIC and stores it along with the overlay name in a message work area and prints this message.

In any case the message is moved to the work area allocated to building the Segment Names and Number Record. The first word in this work area is used to indicate the displacement into the work area where the next segment message is to be stored. Each message is 6 words long.

Next WRITEMON determines if a pack or tape is being created. If pack, WRITEMON calculates the required size of BOOTFILE by searching through the Monitor Tree Table and accumulating the total number of granules required for the Monitor Root and all Monitor Overlays. Added to this are 6 granules which is a fixed requirement for the Boot Subroutine, Monitor Tree Table and Segment Names and Numbers Record. This value is then used as the RSTORE value to open random file 'BOOTFILE' on the BO/PO pack. After the file has been opened WRITEMON accesses the FDA from the File CFU to determine if BOOTFILE is the first data file on a private pack that has been initialized at 2 granules per cylinder. If not the DEF processor is aborted. Otherwise, the Boot Subroutine, along with the CDWs required to boot that routine, is written to the first 4 granules in BOOTFILE.

If a tape is to be written, the tape DCB is opened to device. Then the Mag-Tape Mini-boot, followed by the Boot Subroutine, is written to the tape. The FPTs used to write the Monitor Root and Overlays, the Monitor Tree Table and the Segment Names and Numbers Record are then modified to delete the BLOCK parameter. This enables the same FPTs to be used to write to both tape and pack. From this point on, processing is the same for both tape and pack output.

WRITEMON next BALs to the WRSEG subroutine to write out the Monitor Root in 2048 byte segments. Following this the Monitor Tree Table is written.

WRITEMON then begins the process of reading the overlay segments and writing these segments, again in 2048 byte records, to the output device. First, a page is reserved in the work area to build the Segment Names and Numbers Record. WRITEMON scans the Monitor Tree Table to locate each successive overlay. For each overlay a BAL is made to the DISPSEG subroutine to build the Segment Names and Numbers Record and list it on the LL Device. WRITEMON then obtains the segment name and size from the tree table and reads the Monitor Overlay into the work area. The overlay name is moved to the three words preceding the overlay to be written out along with the overlay itself. The Boot Subroutine will use this information to compare against the Monitor Tree Table as a check at boot time. WRITEMON BALs to the WRSEG subroutine to write out each Monitor Overlay in 2048 byte segments.

After all overlays have been written to the output device, the Segment Names and Numbers Record is written, the input M:TM DCB is closed, the output tape/pack DCB is closed, the work area is released and control is returned to the calling routine.

A.9 WRITEMON MESSAGES

$$$$SEGMENT #'s FOR PATCHING MONITOR MONITOR $$$$	This is a title message that precedes the list of segment numbers.
xxxxxxxxxxxx = SEG. #nnnn	This message identifies the segment number (nnnn) for each segment (xxxxxxxxxxxx) as the absolute bootable Monitor is written for the PO tape.
*****CANNOT OBTAIN 'M:MON' FROM CURRENT ACCOUNT	The M:MON load module cannot be obtained from the current account. The processor is aborted.
*****CANNOT OBTAIN 'M:MON' FROM ':SYS' ACCOUNT ---------PROCESSOR ABORTED (WRITEMON)	The M:MON load module cannot be obtained from the :SYS account. The processor is aborted.
*****CANNOT READ KEYED RECORD 'HEAD' FROM IN M:MON *****CANNOT READ KEYED RECORD 'MON::ORG' IN M:MON *****CANNOT READ KEYED RECORD 'TREE' IN M:MON *****CANNOT READ KEYED RECORD 'xxxxxxxxxxxx' IN M:MON	One of these messages appears if a part of the M:MON load module cannot be obtained. 'xxxxxxxxxxxx' is the name of a M:MON segment. The processor is aborted.
*****CANNOT USE PACK - REINITIALIZE WITH CYL SZ=2	The PO or BO pack must not contain any files prior to the DEF. Processor is aborted.
*****SPACE OVERFLOW ON SEGMENT NAMES RECORD	The Monitor Tree Table is too large for segment name information to be contained in one page. The processor is aborted.
---------PROCESSOR ABORTED (WRITEMON)	Information message to indicate the BPMBT overlay has caused the abort of the DEF processor.

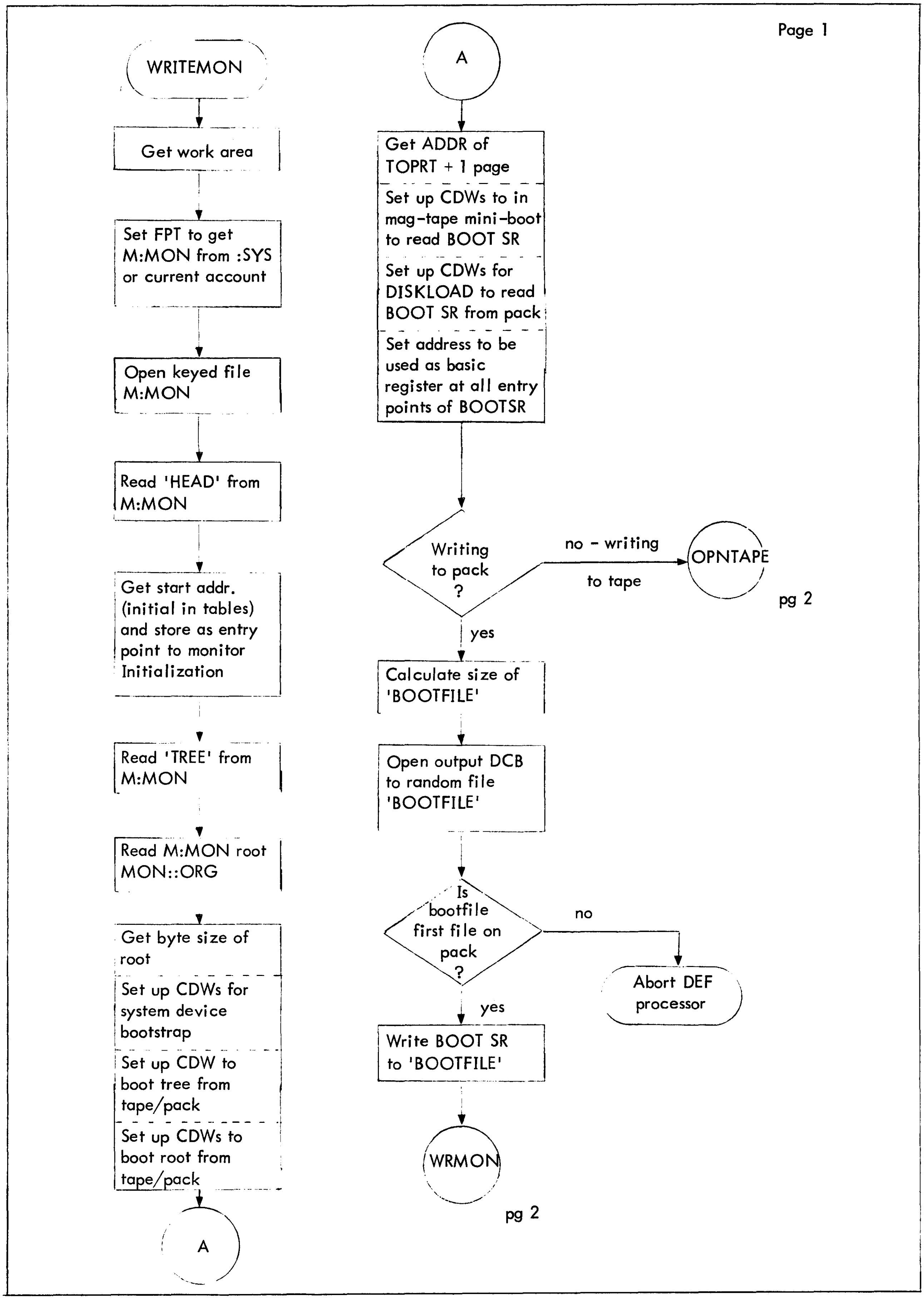

Page 1
WRITEMON
Get work area
Set FPT to get M:MON from :SYS or current account
Open keyed file M:MON
Read 'HEAD' from M:MON
Get start addr. (initial in tables) and store as entry point to monitor Initialization
Read 'TREE' from M:MON
Read M:MON root MON::ORG
Get byte size of root
Set up CDWs for system device bootstrap
Set up CDW to boot tree from tape/pack
Set up CDWs to boot root from tape/pack
A
A
Get ADDR of TOPRT + 1 page
Set up CDWs to in mag-tape mini-boot to read BOOT SR
Set up CDWs for DISKLOAD to read BOOT SR from pack
Set address to be used as basic register at all entry points of BOOTSR
Writing to pack ?
no - writing to tape
OPNTAPE
pg 2
yes
Calculate size of 'BOOTFILE'
Open output DCB to random file 'BOOTFILE'
Is bootfile first file on pack ?
no
Abort DEF processor
yes
Write BOOT SR to 'BOOTFILE'
WRMON
pg 2

Figure A-1. Flow Diagram of BPMBT

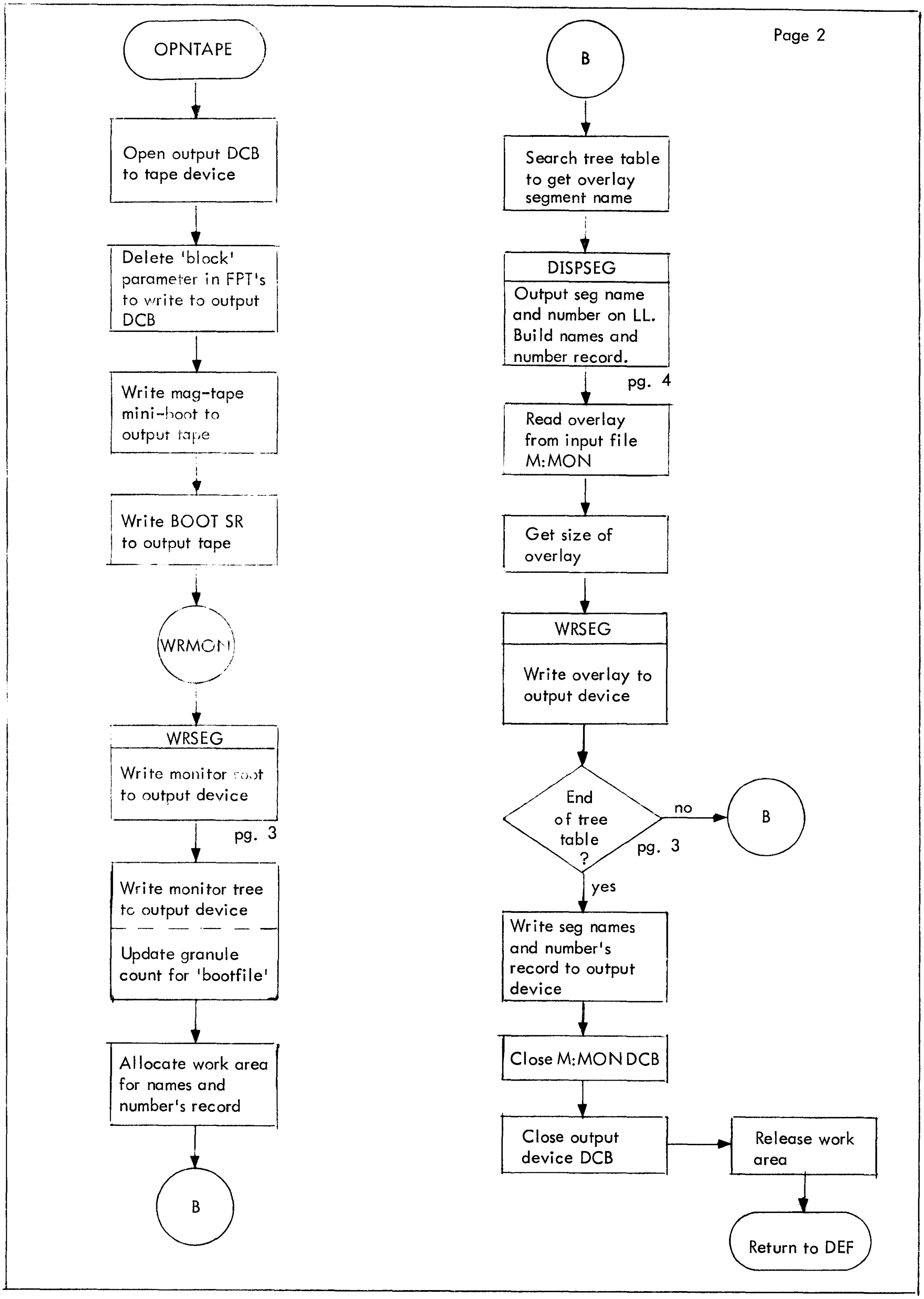

Page 2
OPNTAPE
Open output DCB to tape device
Delete 'block' parameter in FPT's to write to output DCB
Write mag-tape mini-boot to output tape
Write BOOT SR to output tape
WRMON
WRSEG
Write monitor root to output device
pg. 3
Write monitor tree to output device
Update granule count for 'bootfile'
Allocate work area for names and number's record
B
B
Search tree table to get overlay segment name
DISPSEG
Output seg name and number on LL. Build names and number record.
pg. 4
Read overlay from input file M:MON
Get size of overlay
WRSEG
Write overlay to output device
End of tree table ?
pg. 3
no
B
yes
Write seg names and number's record to output device
Close M:MON DCB
Close output device DCB
Release work area
Return to DEF

Figure A-1. Flow Diagram of BPMBT (Cont.)

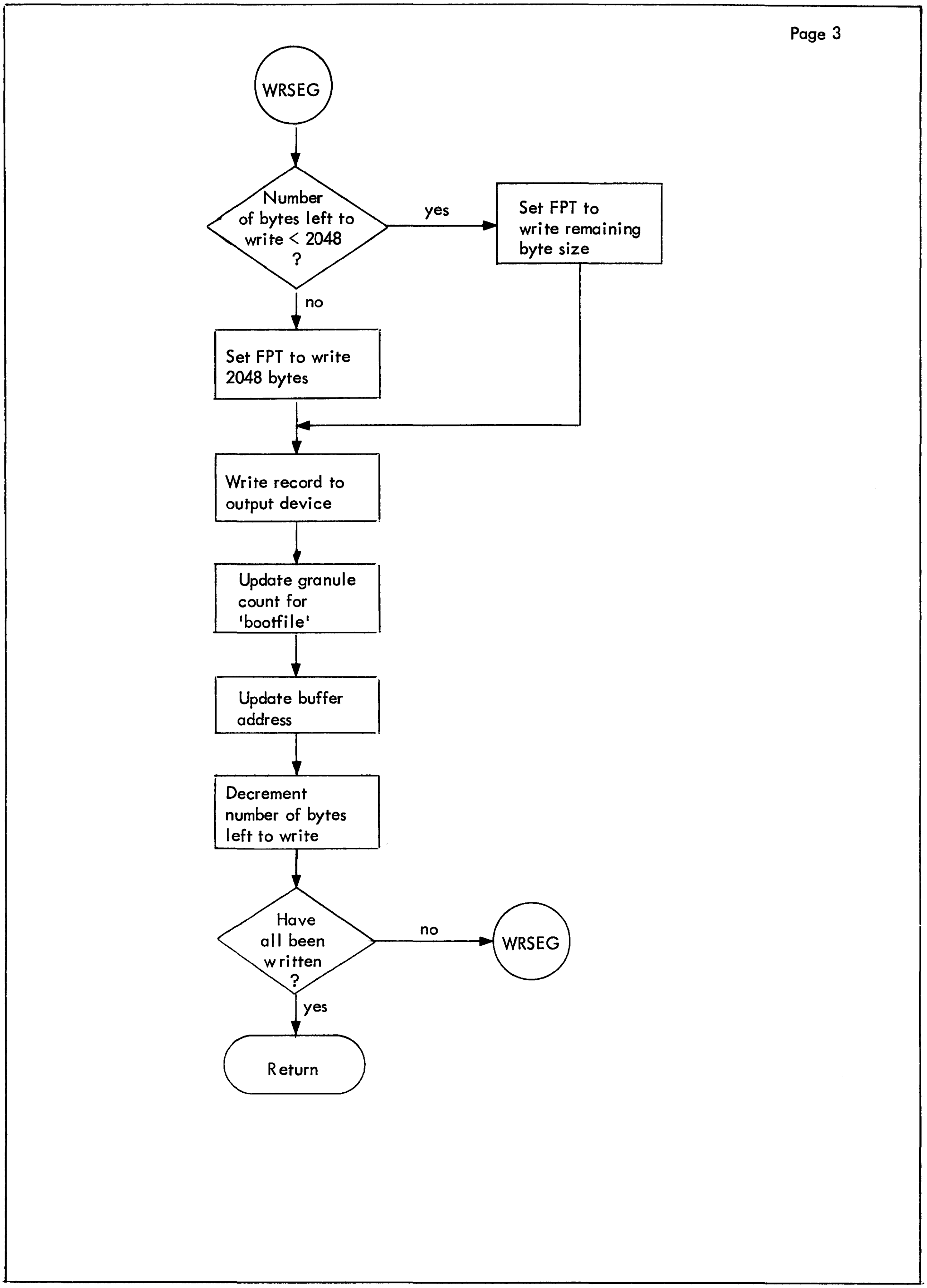

Page 3
WRSEG
Number of bytes left to write < 2048 ?
yes
Set FPT to write remaining byte size
no
Set FPT to write 2048 bytes
Write record to output device
Update granule count for 'bootfile'
Update buffer address
Decrement number of bytes left to write
Have all been written ?
no
WRSEG
yes
Return

Figure A-1. Flow Diagram of BPMBT (Cont.)

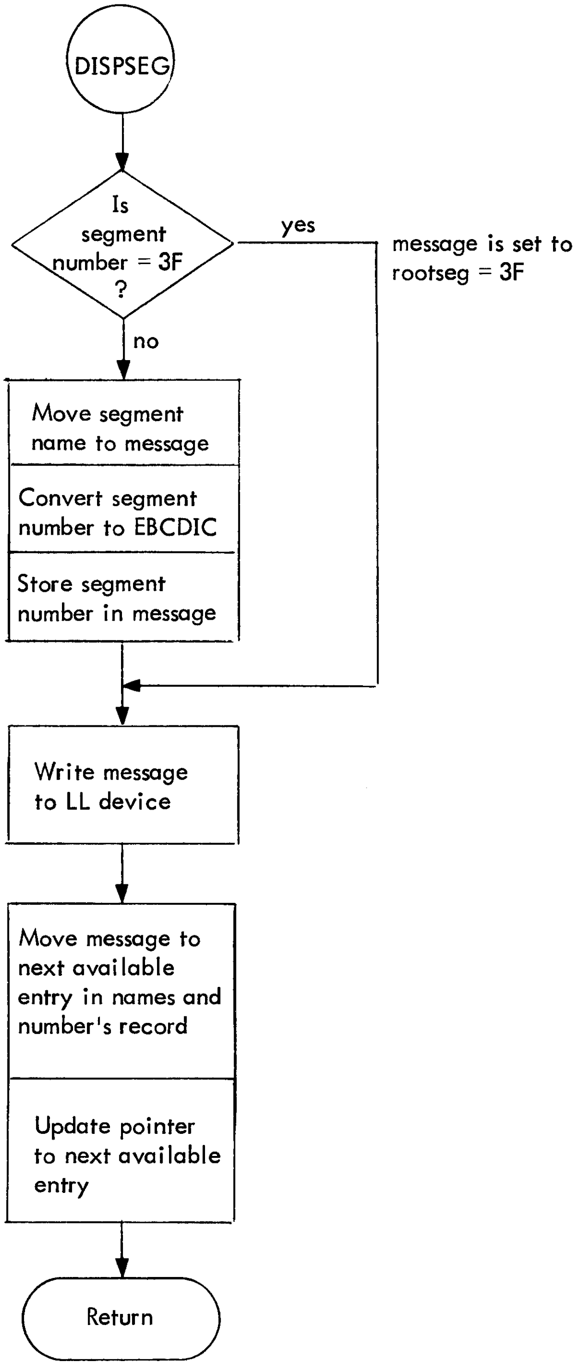

DISPSEG
Is segment number = 3F ?
yes
message is set to rootseg = 3F
no
Move segment name to message
Convert segment number to EBCDIC
Store segment number in message
Write message to LL device
Move message to next available entry in names and number's record
Update pointer to next available entry
Return

Figure A-1. Flow Diagram of BPMBT (Cont.)

APPENDIX B

B.0 UTMBPMBT – DEF OVERLAY

B.1 PURPOSE

To write the bootable portion of a UTS system tape.

B.2 CALLING SEQUENCE

```
BAL,11    UTMBPMWRITEMON
          (6) = X'01010202' for input from current account (DEF)
          (7) = output tape DCB address
          (8) = version number (3 EBCDIC characters, left-adjusted)
          (0) = temp stack pointer address
          All registers destroyed.
```

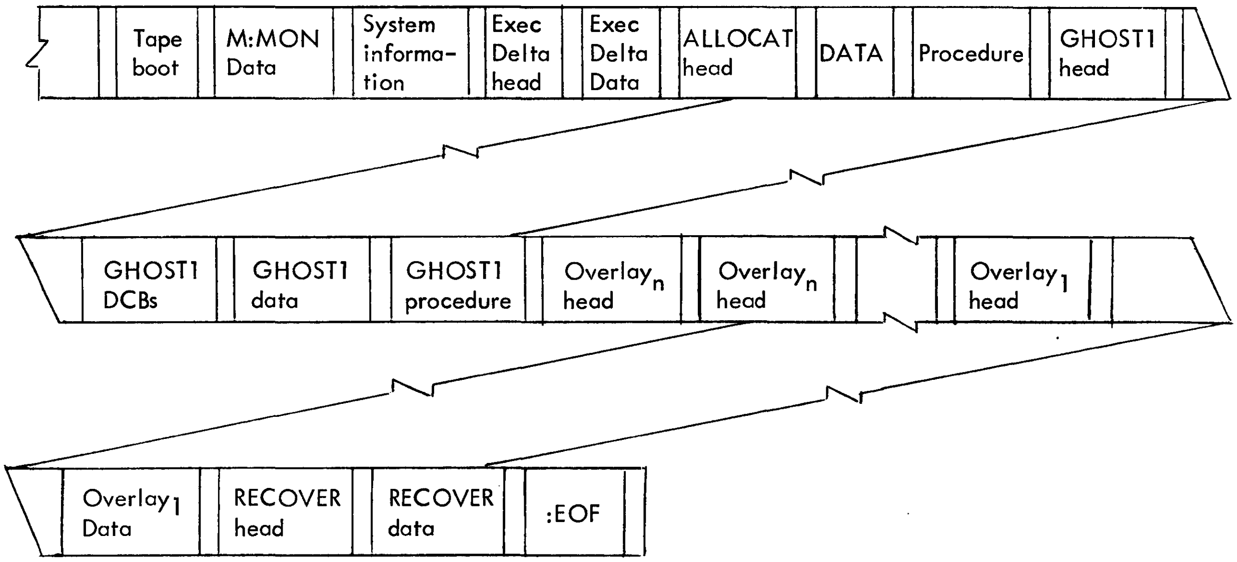
B.3 INPUT

Load module files for M:MON, XDELTA, GHOST1, ALLOCAT RECOVER, M:SPROCS, and all Monitor overlays (according to M:SPROCS) in the current account (for DEF usage).

B.4 OUTPUT

Error messages to the LL Device

Bootable monitor (see Figures B-1 and B-2)



NOTE:

Head Head portion of load module

Data Protection type 0 portion of load module

DCBs Protection type 2 portion of load module

Procedure Protection type 1 portion of load module

OVERLAY$_i$ M:MON overlays (shared processor type) as described in M:SPROCS module (except M:DUMLMs)

Figure B-1. Output Tape Format

VD	NL

byte + 3 bytes

```
Blank line
*****
---UTS---
SYSTEM GENERATED ON:

hh:mm     MM     DD     YY

VERSION NO. IS:  XXXb̸
*****

PATCH SEGMENT NUMBERS:

      00     =     M:MON
      SN     =     SSSSSS
       .     .       .
       .     .       .
       .     .       .
       .     .       .
       .     .       .
       .     .       .
******
Blank line
```

Six words each line. First character of each line must be X'40'.

The message contains the time/data (hh:mm, MM DD YY) of when the tape was generated, and the segment numbers (SN) assigned to each segment (SSSSSS) for absolute patching purposes.

VD = word displacement to version number (XXXb̸)

NL = number of six-word lines of text

Figure B-2. System Information Format

DESCRIPTION

Initialization consists of saving the version number (in R8) in the canned information record, obtaining all available dynamic data pages, and putting the appropriate file name control word (in R6) into M:OPEN FPTs for M:MON, M:SPROCS, and monitor overlays, GHOST1 and RECOVER. Then the output tape DCB is opened and set to the unformatted mode. The M:MON HEAD record is used to put the start address into the tape bootstrap. The data size from the HEAD is used to set up the bootstrap, which is then written to tape. It is followed by M:MON data in 2048-byte records. M:TIME puts the date and time into the canned information record. The M:SPROCS data record is then used to construct a doubleword table of file names for transfer to tape, since this record starts with P:NAME (doubleword table of TEXTC format monitor overlay names). This table is then used to construct the segment number portion of the monitor information record, which is then written in a 2048-byte to tape and line-by-line to the LL device. Then the loop, NXTULBL, is entered which picks names out of the file name table, and copies appropriate protions of the file to tape. The HEAD record is copied unless the file is a JIT. Then the data protection type is copied unless the file is GHOST1 or ALLOCAT, in which case DCB (for GHOST1 only), data, and procedure are copied, in that order. All pieces of information (including the HEAD record) are copied in 2048-byte segments. When RECOVER has been copied, those pages that were obtained are freed, the tape DCB is CLOSEd (to write an :EOF record), and a return is made to the caller via R11.

An abort via M:MERC with X'80' merged into the error code occurs if any module cannot be obtained.

APPENDIX C

C. 0 DEF (H00 BPM/BTM VERSION)

This Appendix describes DEF released as the H00 version for BPM/BTM systems. This version is based on the processor described in Chapter 4 but has been significantly enhanced to permit creation (for BPM/BTM only) of PO/BO disk packs in addition to PO/BO tapes.

C. 1 PURPOSE

To generate one or more target system tapes (PO tapes) or disk packs (PO packs) or BO tapes or disk packs which maybe used as master BI tapes/packs for subsequent SYSGENs.

C. 2 CALLING SEQUENCE

Monitor control command

!DEF....

C. 3 INPUT

DEF control commands from the SI device.

!DEF (from C device)

:WRITE

:INCLUDE

:IGNORE

:DELETE

END

Files from random access device comment commands

C. 4 OUTPUT

Display of DEF control information to LL device.

PO tape

PO disk pack

BO tape

BO disk pack

C. 5 DATA BASE AND REGISTERS

R7 = address in temp stack of control command PLISTS

R6 = address in temp stack of data and I/O PLISTS

IGSTRT/IGEND – Pointer to Start/End of IGNORE table

INCLSTRT/INCLEND – Pointer to Start/End of INCLUDE table

LSTLMST – Pointer to Start of intermediate table for LASTLM for BO disc pack

LSTLMBUF – Pointer to Start of compact LASTLM record to be written for BO disc pack

OPEN FPTs

OPNTMSQN – Open disc to file

OPNPOBO – Open PO/BO (via DCB whose address is in R5). Originally set up to write to tape, dynamically changed if writing to disc pack.

OPNTM – Open disc to next file.

OPNSYN – Open for Synonomous files originally set up to write to tape, changed dynamically if writing to disc pack.

OPNPOBOLST – Open to write null LASTLM used for PO/BO tapes and PO disc pack set up for tapes, changed if disc pack.

OPNDPLST – Open to write non-null LASTLM to BO disc pack.

POIGS/BOIGS Tables – Automatic IGNOREs for BO/PO tapes and disc packs.

POINCLS Table – Automatic INCLUDEs for UTS PO tape.

BOINCLS Table – Automatic INCLUDEs for UTS BO tape.

BBOINCLS Table – Automatic INCLUDEs for BPM/BTM BO tape/disc pack.

C.6 SUBROUTINES

BPMBT – to write BPM/BTM system to unlabeled portion of BO/PO tape or to random file 'BOOTFILE' on BO/PO disc pack. See Appendix A for description.

UTMBPMBT – to write UTS system to unlabeled portion of BO/PO tape. See Appendix B for description.

NAMSCAN – to scan any field containing a name.

CHARSCAN – to check a specific character for legitimate syntax.

CHSTSCAN – to obtain a character string field.

NXACTCHR – to get next active character from input record.

HEXSCAN – to scan for a hexadecimal character.

DECSCAN – to scan for a decimal character.

QUOTSCAN – to compare a quote constant with a character string.

GETCHST – to obtain the next character string.

C.7 CONTROL COMMANDS

Upon entry DEF requires a parameter on the !DEF command that identifies the SYSGEN system for which tapes/disc packs are being created. This parameter maybe either BPM, BTM, or UTS. If none is specified, then the currently running operating system is used to determine the system type. If the parameter is invalid, then DEF prints an indicative message and aborts. For UTS, an optional second parameter is a version number. There is no syntax analysis made on the field, therefore any set of characters is accepted. However, only the first three characters are retained.

The type and composition of the tape(s)/disc packs DEF creates is a function of the control commands read by DEF. If the !DEF is immediately followed by a monitor control command, one PO tape is created by default. The function of the END command is to cause DEF to exit since an EOF on reading M:SI causes one PO tape to be generated unless the last command processed was :WRITE. To write a BO tape/PO or BO disc pack and/or include, ignore or delete files for any tape or pack type, :Commands are required. These commands have the following format:

:INCLUDE (name1, name2...)

:IGNORE (nameA, nameB...)

:DELETE

:WRITE $\begin{Bmatrix}\text{BO}\\ \text{PO}\\ \text{BODP}\\ \text{PODP}\end{Bmatrix}$,[SN]

All commands preceding the :WRITE apply to that tape/disk pack being created and may appear in any order. The :WRITE is required for BO tapes and packs and PO packs as a PO tape is generated if the type parameter is null or illegal. The optional SN field permits a generalized assignment of PO/BO to (DEVICE, $\genfrac{}{}{0pt}{}{9T}{DP}$ prior to calling DEF. The processor itself stores the particular SN into the DCB.

The type of files that may be specified or are affected by the other commands depend on the type of tape/disc pack being generated. Table C-1 summarizes this information.

Table C-1. File Types

:COMMAND	BO tape/pack	PO tape/pack
:INCLUDE	Keyed files	Consecutive files
:IGNORE	Consecutive files	Keyed files
:DELETE	BOTH	BOTH

C.8 DESCRIPTION

Upon entry DEF initializes its dynamic data area and processes the !DEF command. One page of core is obtained for storing file names into the tables pointed to by IGSTRT and INCLSTRT.

DEF then reads it :Commands and branches to the appropriate routine to process them. For :INCLUDE and :IGNORE, this involves syntax checking the names ($\leq$ 15 characters); determining if room exists for the entry (if not, obtaining an additional page of core); storing the name in the appropriate table; and exiting to read another command. For :DELETE, a flag (DELETEF) is set before exiting to read another command. When an abnormal return (EOF) is made from reading SI for commands, ENDFLG is set and if the WRTFLG is non-zero indicating the last command was :WRITE then the routine is entered to clear up the stack and exit. If WRTFLG is zero, then the routine to write a PO tape by default is entered.

For the :WRITE command, entry is made to the initial routine that determines which type of tape or disc pack is being generated. From here, a branch is made to either the PO or BO tape/pack routines.

For PO and BO packs for BPM/BTM systems only, the initial routines alter the fpts to reflect file type operations and change the file access parameter in the fpt for synonymous files to update mode. From these routines DEF then enters the main processing routines for PO and BO tapes.

For PO tapes, after processing the optional SN field, the names of files that are to be automatically ignored (i.e., LASTLM and SPEC:HAND) are linked to the end of the IGNORE table. If the system being created is UTS, then the names of files to be automatically included are linked to the end of the INCLUDE table. These files are listed in Table C-2. The appropriate routine to write the unlabeled portion of the tape is segloaded and entered. See Appendixes A and B for a description of these routines. Upon return, ten additional pages of core are obtained and the common routine (CCA) to generate the remainder of either type of tape is entered.

For BO tapes, after processing the optional SN field, the appropriate routine to write the unlabeled portion of the tape is segloaded and entered. Upon return, for BPM/BTM systems, the files to be automatically included (see Table C-2), are linked to the end of the INCLUDE table, ten additional pages of core are obtained and exit is made to CCA. If the system is UTS, the file, M:SPROCS in :SYS account, is opened, ten pages of core are obtained and the file is read into the newly acquired area and linked to the end of the INCLUDE table. M:SPROCS contains the names of the monitor overlays and shared processors but only the overlays are added to the INCLUDE table. The names of the other files to be automatically included are linked to the end of the added monitor overlay names and exit is made to CCA.

Upon entry, the common routine (CCA) first determines if a disc pack is being created, if not then VOLINIT is removed from the list of automatic INCLUDEs. If a BO disc pack is being generated then the start of the INCLUDE table is stored as the start of the list of files for the non-null LASTLM file. The INCLUDE list is then processed. This involves obtaining the name of a file, storing it in the open FPT (OPNTMSQN) for M:TM to the disc and then opening the file, using the FPARAM option. The file, thus opened, is checked first if it is a synonymous file in which case special handling is required, namely its parent name must be added before writing it to the tape. Note the parent file must occur before the synonymous file or the latter is lost.

Then the organization of the file is determined. If PO, then only consecutive files are processed, if BO, then only keyed files. The other types are automatically handled later. If a file for a disc pack is invalid then a link is set over the entry thus removing it from possible subsequent processing in BO pack situations.

The PO/BO tape disc pack is then opened and the file is read into core and wirtten to tape/pack until an EOF is encountered at which point the DCB is closed. This routine is repeated until all the names in the INCLUDE table have been processed. When this processing is completed and the files thus written to the tape/pack have been listed on the LL device, the next phase of DEF is entered.

If a BO pack is being generated then the routine WRLSTLM is entered to build a LASTLM file of one record containing the names of all the INCLUDEd files and terminated by a final word of zero. The files so named are to go into the :SYS account at boottime. On BO tapes, the file LASTLM is null and its position on the tape serves as a termination of the files for :SYS. However, on BO packs, because the files on a disc pack are accessed through the alphabetized file directory, LASTLM is used as a mini-directory for the :SYS file.

If a BO tape is being generated, a null file, LASTLM, is written to the tape. Subsequently, or if a PO tape is being created, the FPT for open-next to the disc (OPNTM) is opened and file parameters obtained. If the file organization is consecutive (BO)/keyed (PO), the IGNORE table is searched to determine if it is listed there. If the DELETEF is set, the file is deleted when M:TM is closed. If the file is not to be IGNOREd, it is then read into core and written to tape/disc pack. This procedure is repeated until all files in the current account have been processed.

If on opening-next-file, an abnormal return is made indicating the file is a synonymous file, its name is stored in a new INCLUDE table whose location is pointed by INCLSTRT and a flag (SYNFLG) is incremented, thus maintaining a running total of the number of synonymous files found.

When an abnormal return is made indicating an end of all files on open-next, if the tape/pack being created is BO, it is immediately closed, rewound, and saved. If a PO tape/pack is being generated, SYNFLG is tested. If non-zero, a second pass is made through the INCLUDE routines. If or when SYNFLG is zero, the null file, LASTLM, is written to the tape/pack which is closed, rewound, and saved.

Note: When writing Synonymous files to disc pack, the DCB is opened in the update mode. If the parent file is not there, then the fpt is set to open out and the file is subsequently written out.

The pages of core acquired thus far are released. If ENDFLG is not set, the flags and counters are zeroed to prepare for the generation of another tape. If ENDFLG is set, DEF exits.

Table C-2. Automatic INCLUDEs

PO Tape/Pack*	BO Tape/Pack**	
UTS	UTS	BPM/BTM
BPM	XDELTA	FMGE
UTS	LOGON	PASS1
SIG7FDP	TEL	ERRMSG
:BLIB	SUPER	:DIC
FLIBMODE	DEFCOM	:LIB
SIGMET	SYMCOM	M:C
M:CDCB	JIT0	M:OC
M:OCDCB	JIT1	M:BI
M:BIDCB	JIT2	M:CI
M:CIDCB	JIT3	M:SI
M:SIDCB	JIT6	M:EI
M:EIDCB	ANLZ	M:BO
M:BODCB	ERRMSG	M:CO
M:CODCB	GHOST1	M:SO
M:SODCB	RECOVER	M:PO
M:PODCB	M:SPROCS	M:GO
M:GODCB	M:MON	M:LO
M:LODCB	PCL	M:DO
M:DODCB	CCI	M:EO
M:EODCB	LOADER	M:LL
M:LLDCB	PASS2	M:CK
M:SLDCB	LOCCT	M:SL
M:ALDCB	PASS3	M:AL
M:LIDCB	DEF	M:LI
	Plus Monitor overlays	M:MON
	from M:SPROCS	PCL
		CCI
		LOADER
		PASS2
		LOCCT
		PASS3
		DEF
		VOLINIT***

*From Current Account

**From :SYS Account

***For BO pack only

C.9 DEF MESSAGES

::::SYSGEN DEF IN CONTROL::::	Commentary at beginning of execution.
::::DEF COMPLETED::::	Commentary at end of execution.
**CC TYPE UNKNOWN ****GET NEXT CC	Error in Command. DEF reads next command.
**SYNTAX ERROR, NO '('	Error in syntax. DEF reads next command.
**DELIMITER MUST BE ',' OR ')'	Invalid terminator on :Command. DEF reads next command.
**NAME INVALID OR > 15 CHAR. LONG	DEF searches for next parameter.
****NOT ENOUGH CORE AVAILABLE *****SYSGEN DEF ABORTED	Work area too small. DEF exits.
***WRITING PO TAPE BY DEFAULT	Either no type specified or parameter invalid on :WRITE.
***ILLEGAL INCLUDE - WILL BE COPIED LATER	On the :INCLUDE command a keyed file (for PO) or a consecutive file (for BO) has been specified. The file name is printed above this message. DEF continues.
***DEF TYPE UNKNOWN	System type field of !DEF command has been specified but is invalid. DEF exits.
***TYPE UNKNOWN - xx M used	System type field of !DEF missing. DEF defaults to currently running system type (xx).
**NO ':' in column-1	Command in error. DEF reads next command.
****TROUBLE WITH M:SPROCS ***CANNOT WRITE TAPE	In attempting to open M:SPROCS in creating a BO tape for UTS system, difficulty encountered. DEF releases the tape and if ENDFLG set, exits.
*****CANNOT WRITE DP	DEF is unable to write to the disc pack. It releases the pack and proceeds to the next command.
***I/O ERR/ABN = xx/xx	An I/O error/abnormal condition has been detected by DEF and is not expected.
****TROUBLE WRITING LASTLM FOR BO DP	An error or abnormal return has been made when opening or writing LASTLM for BO packs.
***CANNOT OPEN OUTPUT DEVICE	In attempting to open tape/pack (BO/PO), an abnormal condition occurs. DEF releases tape pack and if ENDFLG set, exits.
... PO TAPE CONTENTS... ... PO DP CONTENTS... ... BO TAPE CONTENTS... ... BO DP CONTENTS... ***INCLUDE ITEMS*** ***OTHER ITEMS*** ********INCLUDE ITEMS NOT FOUND	These are subtitles that are followed by a list of the appropriate files.

C. 10 INTERNAL ROUTINES

DEF	Main entry, initialize processor dynamic data area.
READFRST	Process DEF command.
INIT	Initialization of pointers.
DEFRDCC	Read :Command for DEF, and branch to appropriate routine or set DELETEF.
DEFINCL	Process :INCLUDE.
DEFIG	Process :IGNORE.
DEFWRITE	Initial processing of :WRITE.
DFPODP DFBODP	Initial processing of PO/BO disc pack for :WRITE commands. Sets DPFLG and changes FPT to reflect file type operations, changing mode of synonymous file fpt to update. Exits to DFWRTPO/DFWRTBO.
DFWRTPO	:WRITE processing for PO.
DFWRTBO	:WRITE processing for BO.
DEFTABLR	Processing name options on :INCLUDE or :IGNORE.
PAGER	Get a page of core and zero it out.
READCC	Reads :commands for DEF. Register 12 = CC Buffer address
LISTCC	Display commands on LL device. Register 12 = CC Buffer address
GETRITEMON	Obtain appropriate WRITEMON overlay according to system type (UTS or BPM/BTM).
EOCCSCAN	Find end of current control command.
CCA	Generate PO/BO tape/disc pack.
NXTINCL	Obtain next INCLUDE file name.
RDWRITEM	Read and write file.
SYNINCL	Process synonymous file includes.
NOINCL	End of includes, begin generating remainder of tape/disc pack.
WRLSTLM	Builds non-null LASTLM record for BO disc pack after completion of processing of :INCLUDEs.
NXTFILE	Obtain next file on disc.
IGNOR1	Search ignore table for match.
ISSPEC	Delete file if required.
CLSDSK	Close file.

RDWRITE	Read file and write to tape/disc pack.
ALLDONE	Releases pages acquired, if ENDFLG set, exits.
NXTTPEDP	Zeroes flags and counters, restores FPTs to original state, returns to INIT via PAGER.
Error and abnormal return routines.	
LSTWRT	EOF on reading M:SI.
RTMAINCL	EOF on reading INCLUDE files.
OTMAINCL	Abnormal return on opening of INCLUDE file.
RTMA	EOF on reading M:TM file.
OTMA	EOF on open next of M:TM or synonymous file found for open-next.
OPOA	Cannot open BO/PO Tape/disc pack.
DPABN	Abnormal condition for opening writing disc pack.
SYNERR	Abnormal/error conditions for writing synonymous files to disc pack.

C. 11 FLOWCHARTS FOR DEF

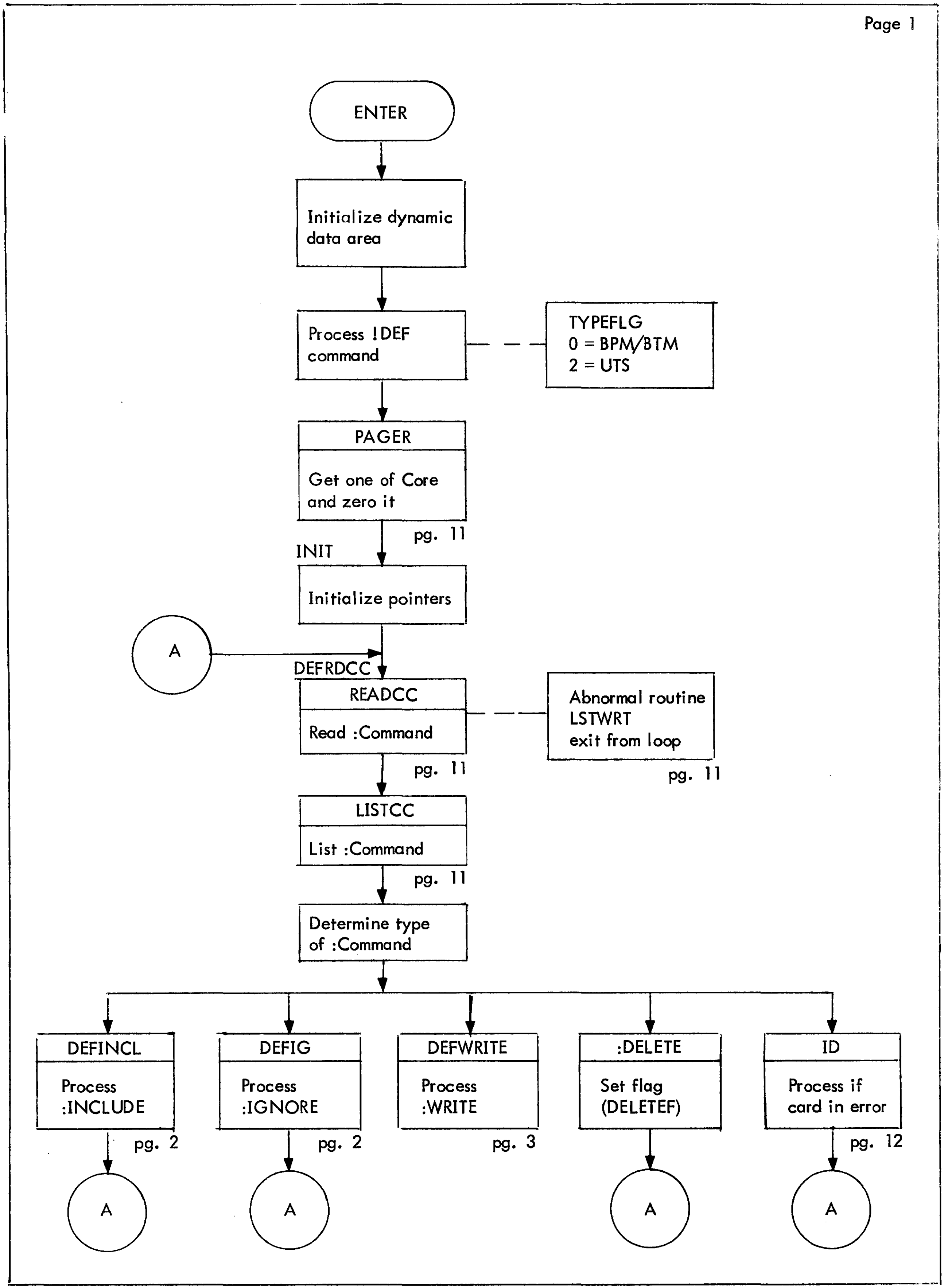


Figure C-1. Flow Diagram of DEF

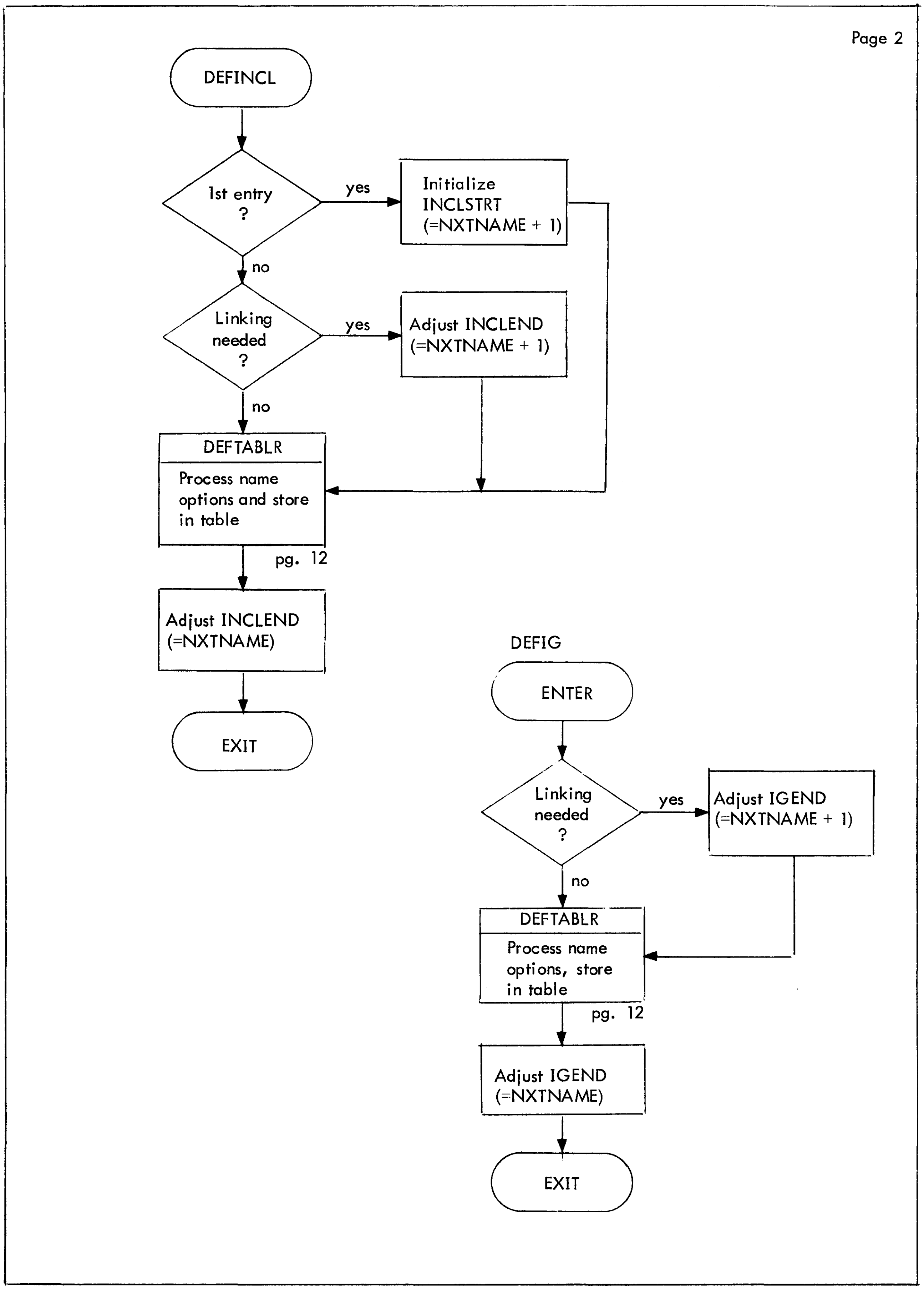

Page 2
DEFINCL
1st entry ?
yes
Initialize INCLSTRT (=NXTNAME + 1)
no
Linking needed ?
yes
Adjust INCLEND (=NXTNAME + 1)
no
DEFTABLR
Process name options and store in table
pg. 12
Adjust INCLEND (=NXTNAME)
EXIT
DEFIG
ENTER
Linking needed ?
yes
Adjust IGEND (=NXTNAME + 1)
no
DEFTABLR
Process name options, store in table
pg. 12
Adjust IGEND (=NXTNAME)
EXIT

Figure C-1. Flow Diagram of DEF (Cont.)

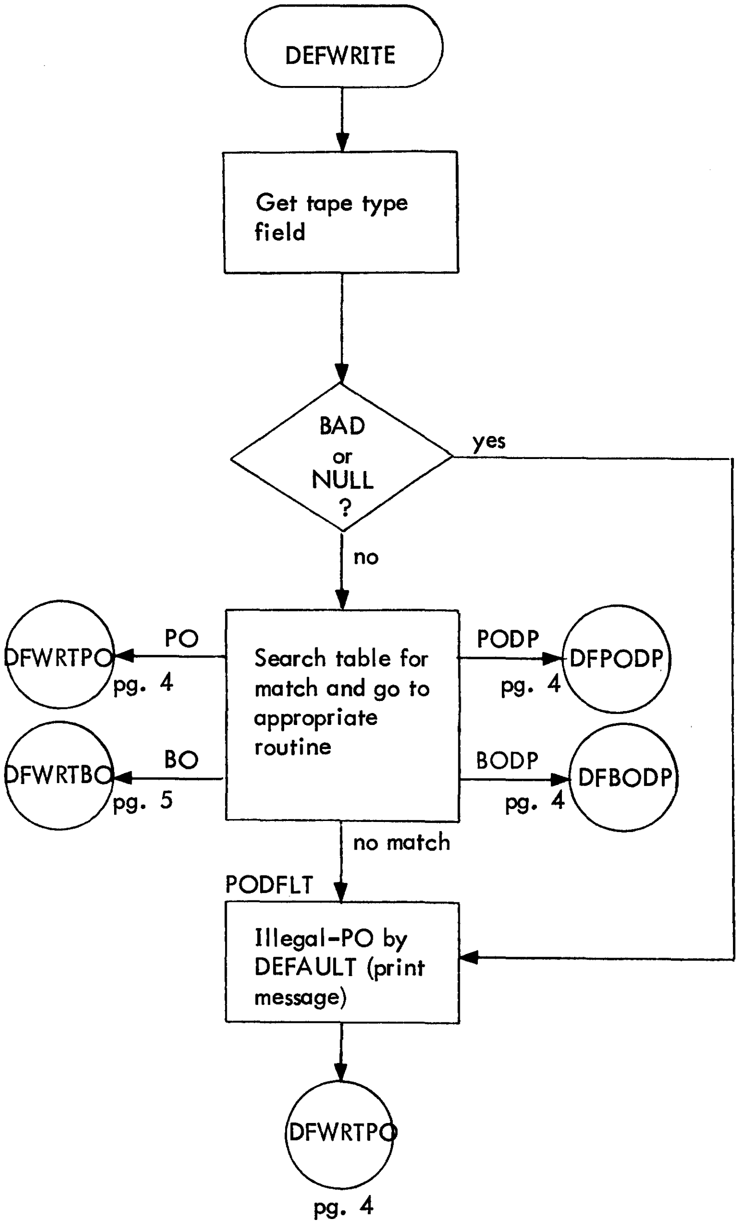


Figure C-1. Flow Diagram of DEF (Cont.)

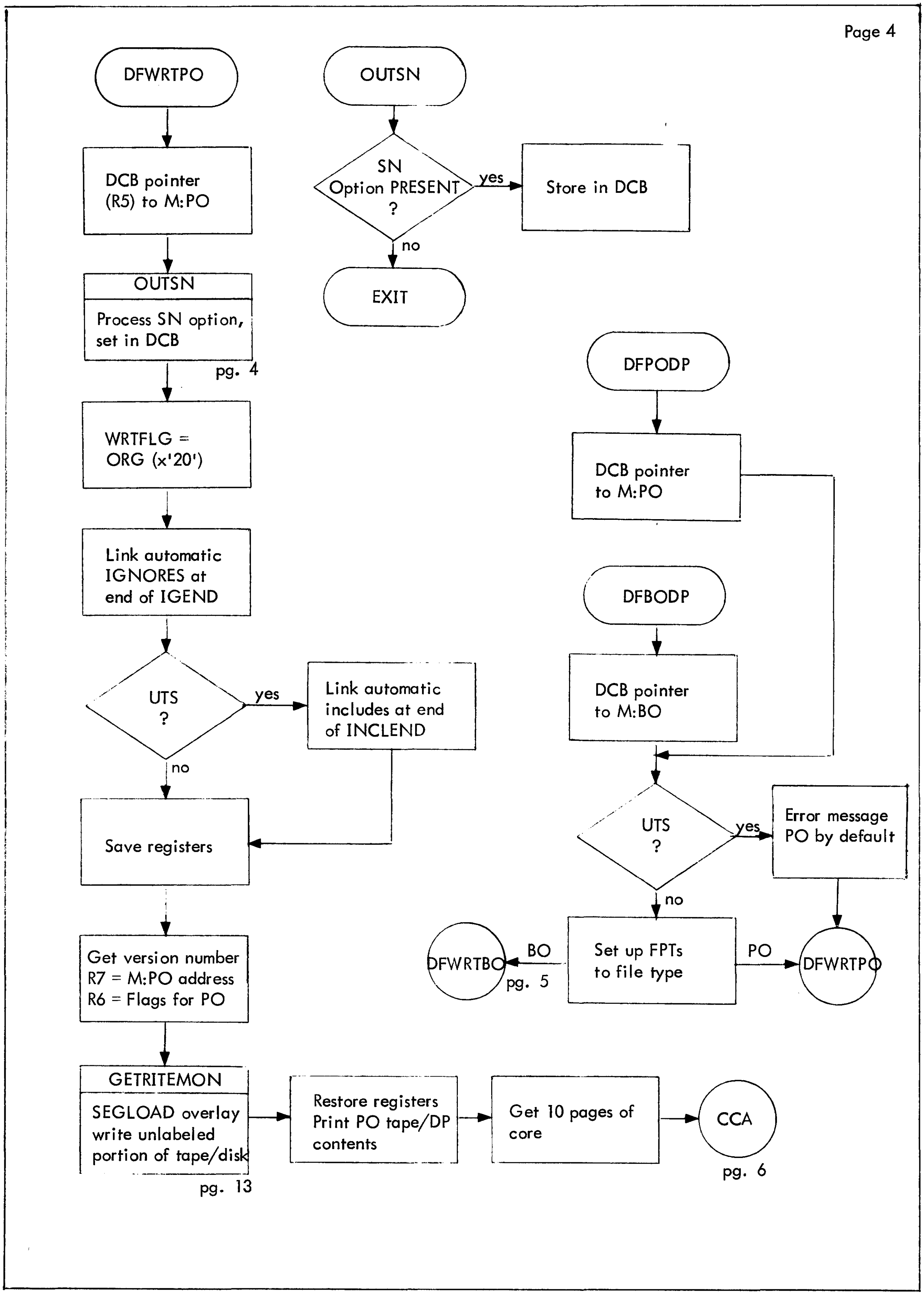


Figure C-1. Flow Diagram of DEF (Cont.)

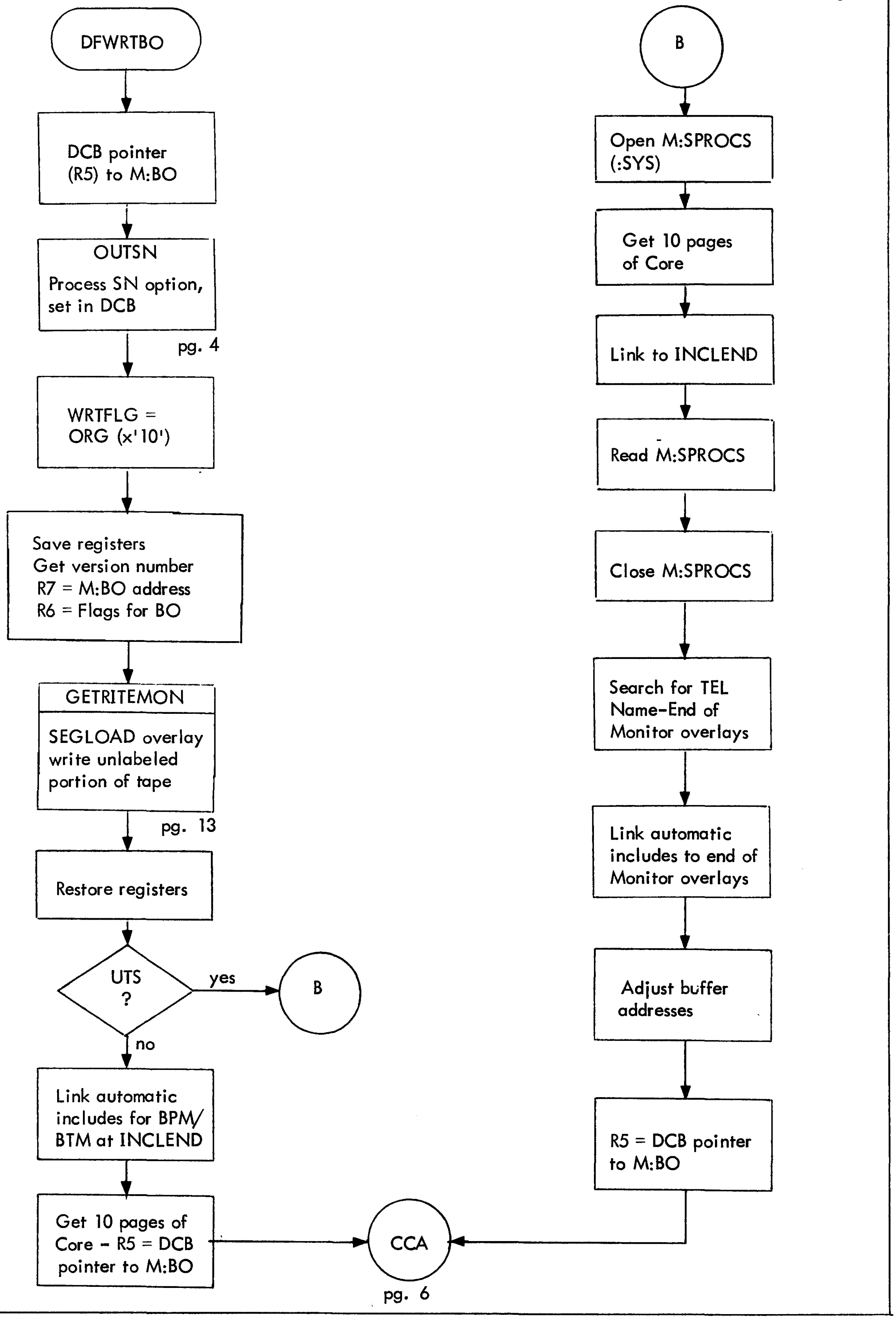


Figure C-1. Flow Diagram of DEF (Cont.)

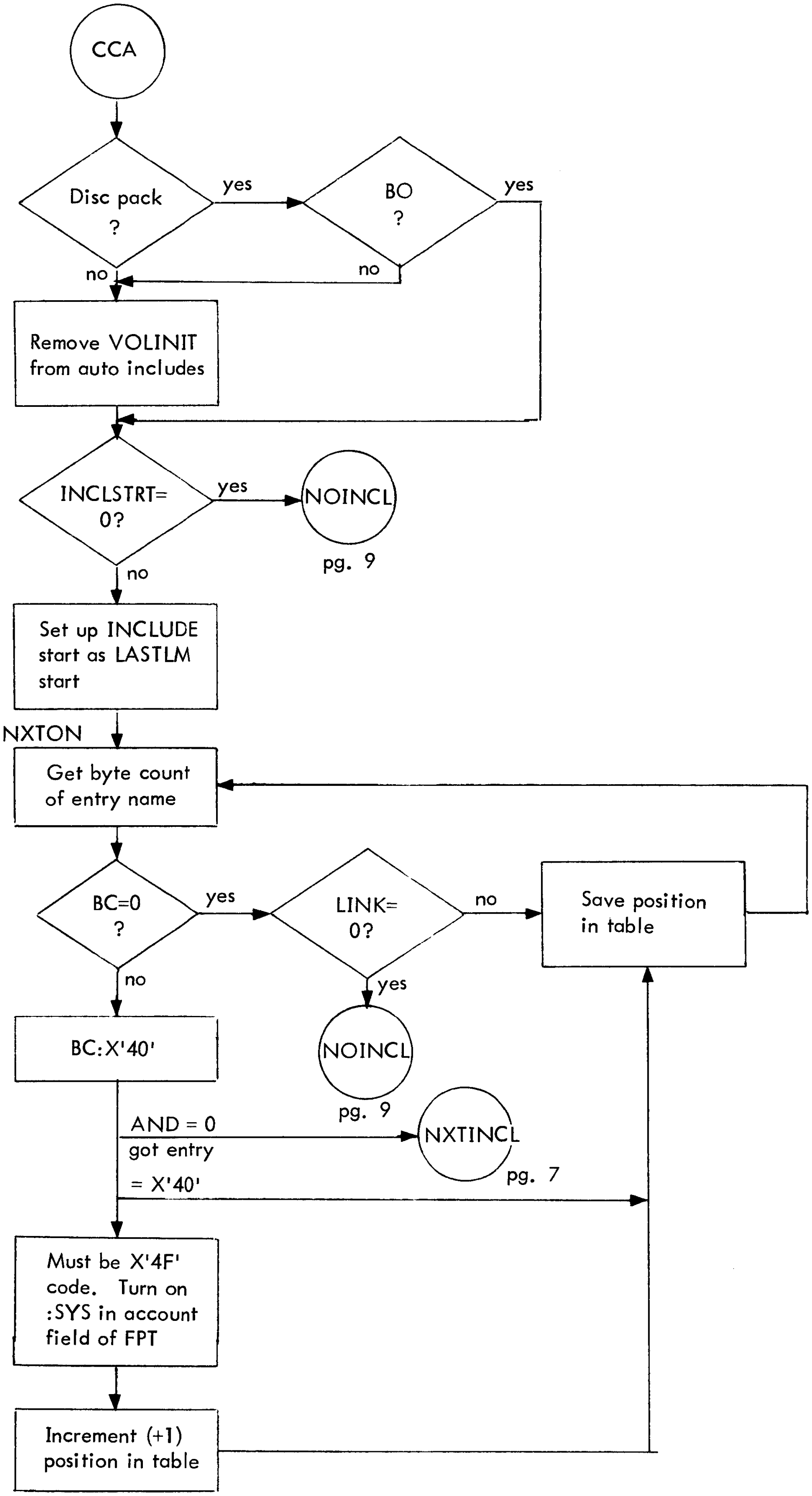
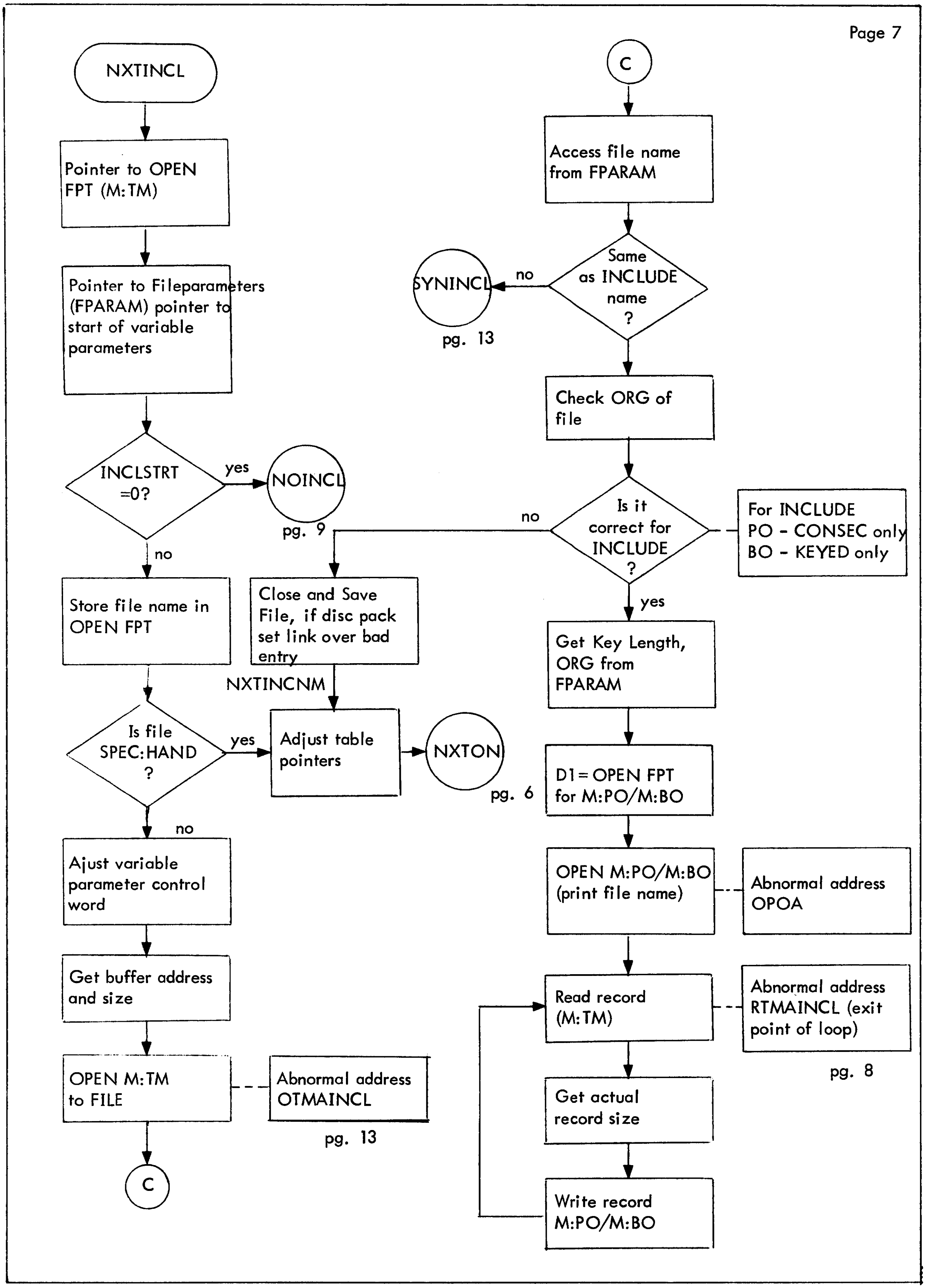

CCA
Disc pack ?
yes
BO ?
yes
no
no
Remove VOLINIT from auto includes
INCLSTRT= 0?
yes
NOINCL
pg. 9
no
Set up INCLUDE start as LASTLM start
NXTON
Get byte count of entry name
BC=0 ?
yes
LINK= 0?
no
Save position in table
yes
no
NOINCL
pg. 9
BC:X'40'
AND = 0 got entry
NXTINCL
pg. 7
= X'40'
Must be X'4F' code. Turn on :SYS in account field of FPT
Increment (+1) position in table

Figure C-1. Flow Diagram of DEF (Cont.)



Page 7
NXTINCL
Pointer to OPEN FPT (M:TM)
Pointer to Fileparameters (FPARAM) pointer to start of variable parameters
INCLSTRT =0?
yes
NOINCL
pg. 9
no
Store file name in OPEN FPT
Close and Save File, if disc pack set link over bad entry
NXTINCNM
Is file SPEC:HAND ?
yes
Adjust table pointers
NXTON
pg. 6
no
Ajust variable parameter control word
Get buffer address and size
OPEN M:TM to FILE
Abnormal address OTMAINCL
pg. 13
C
C
Access file name from FPARAM
Same as INCLUDE name ?
no
SYNINCL
pg. 13
Check ORG of file
Is it correct for INCLUDE ?
no
For INCLUDE PO - CONSEC only BO - KEYED only
yes
Get Key Length, ORG from FPARAM
D1=OPEN FPT for M:PO/M:BO
OPEN M:PO/M:BO (print file name)
Abnormal address OPOA
Read record (M:TM)
Abnormal address RTMAINCL (exit point of loop)
pg. 8
Get actual record size
Write record M:PO/M:BO

Figure C-1. Flow Diagram of DEF (Cont.)

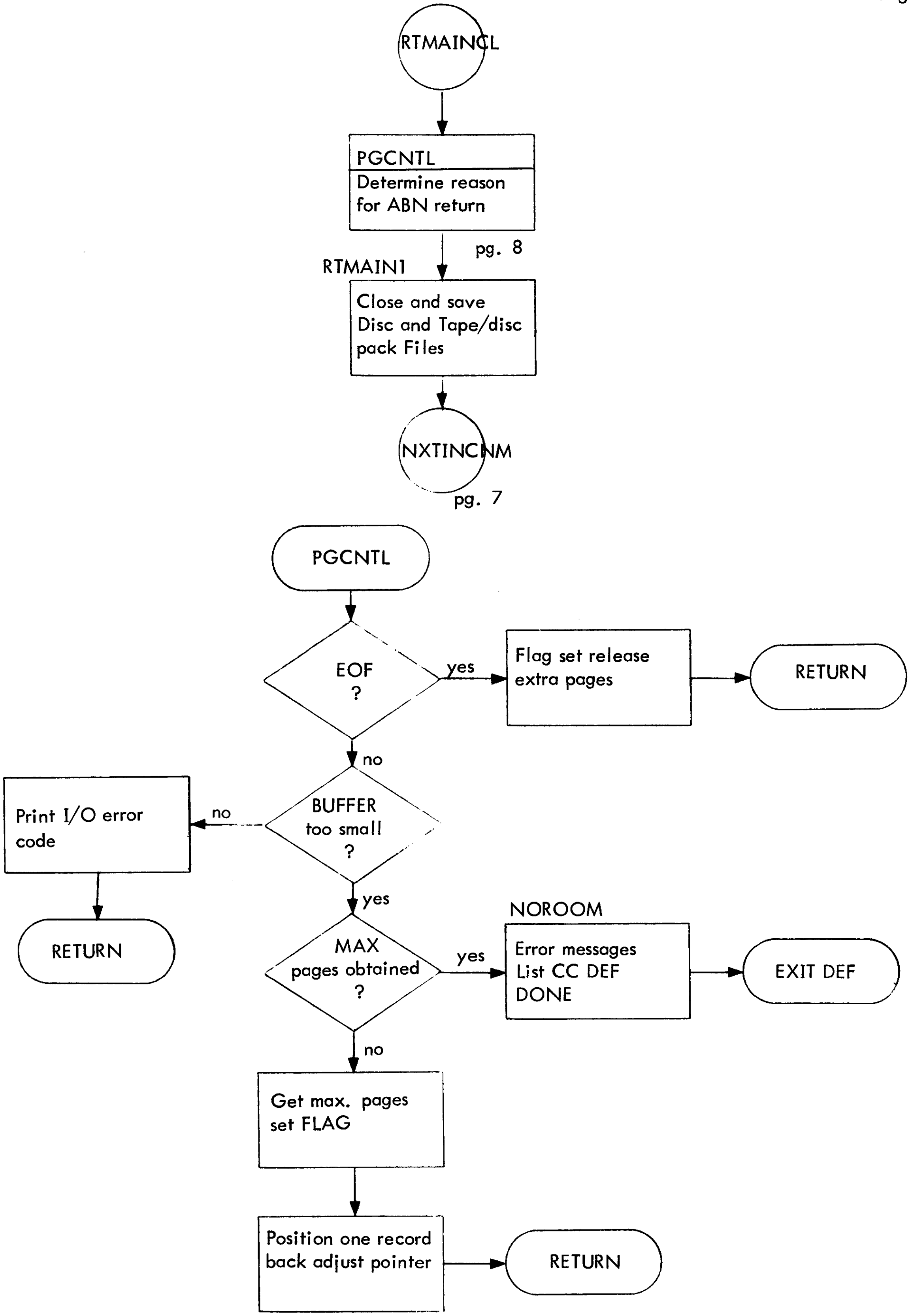


Figure C-1. Flow Diagram of DEF (Cont.)

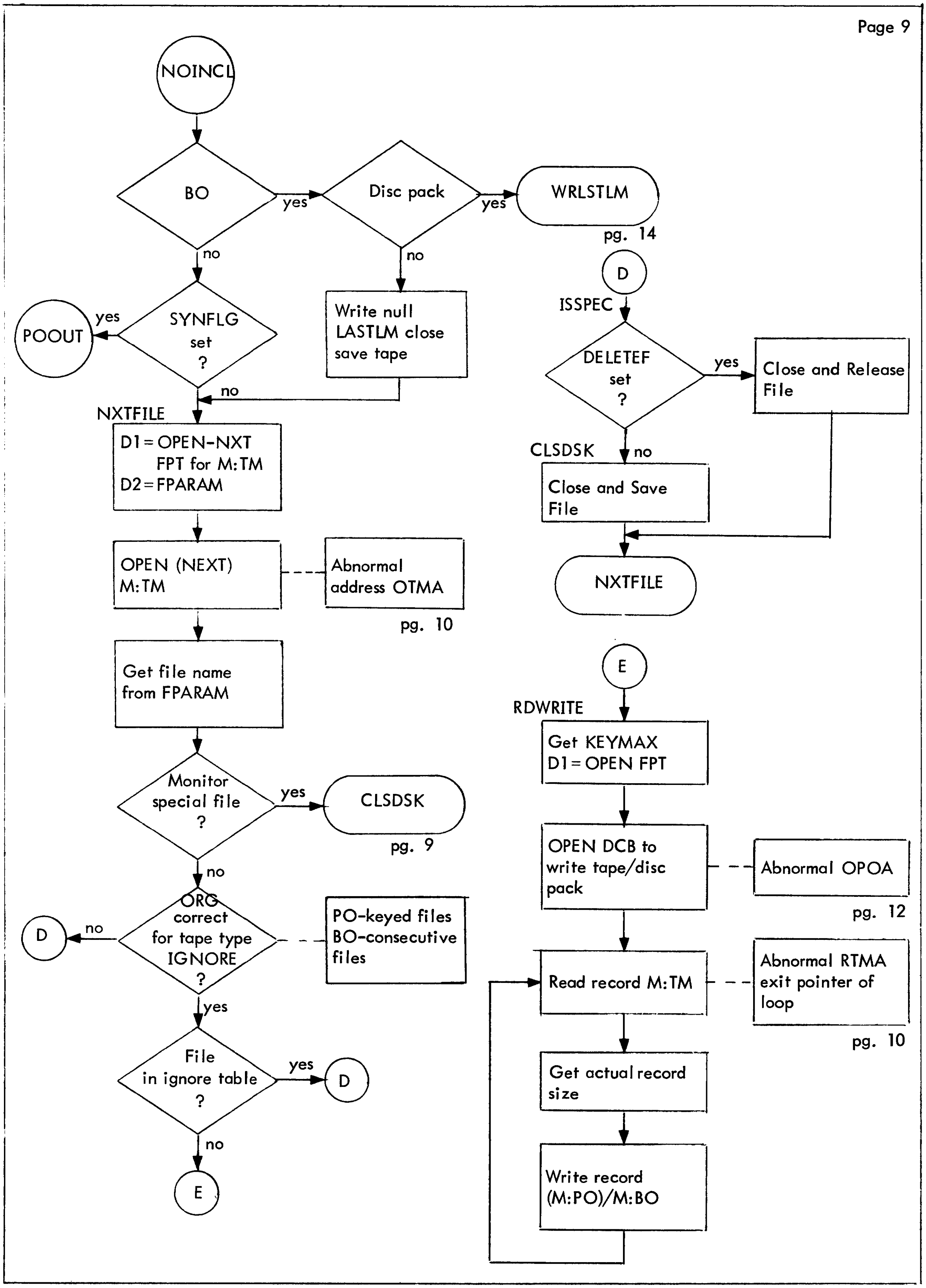


Figure C-1. Flow Diagram of DEF (Cont.)

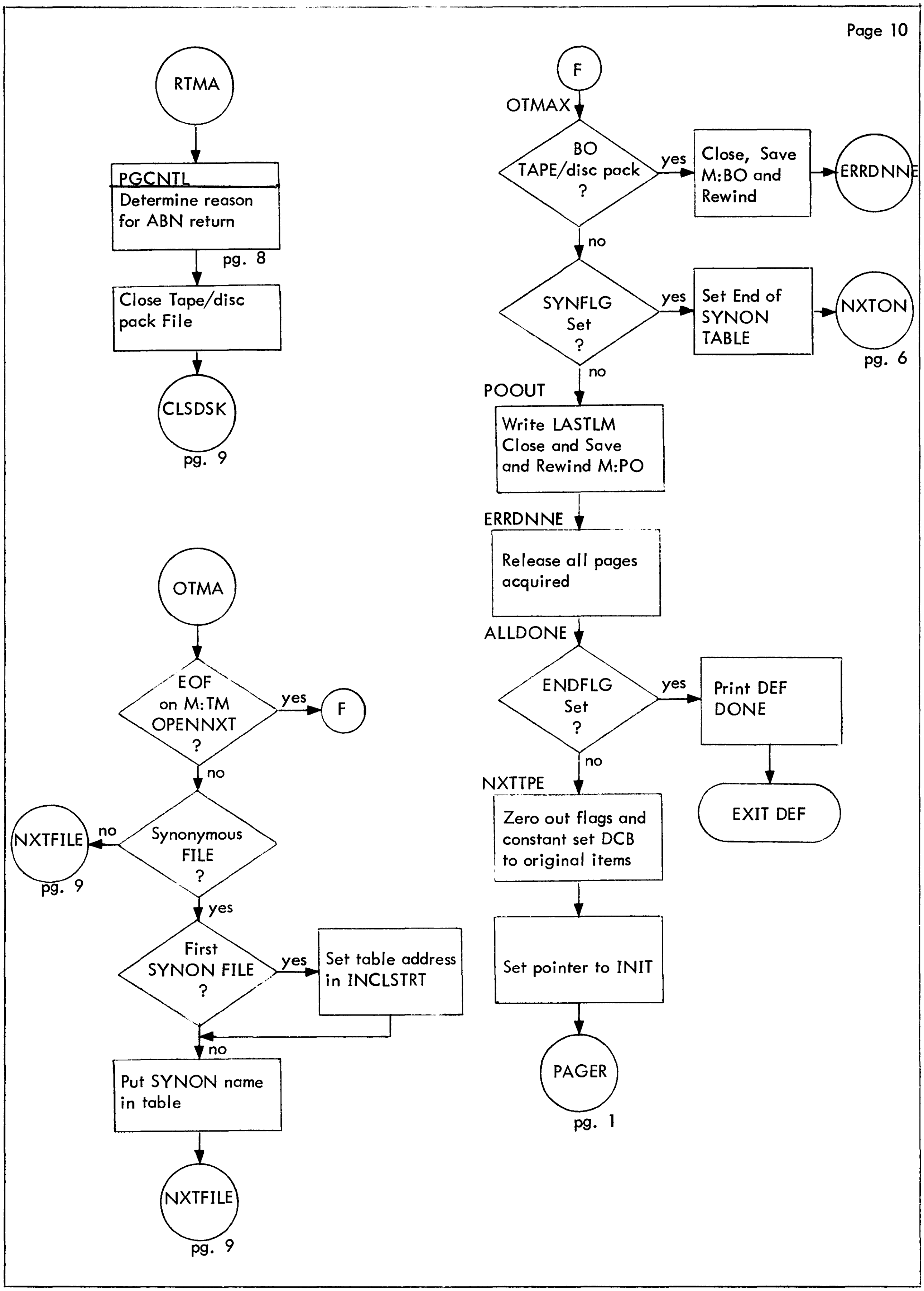


Figure C-1. Flow Diagram of DEF (Cont.)

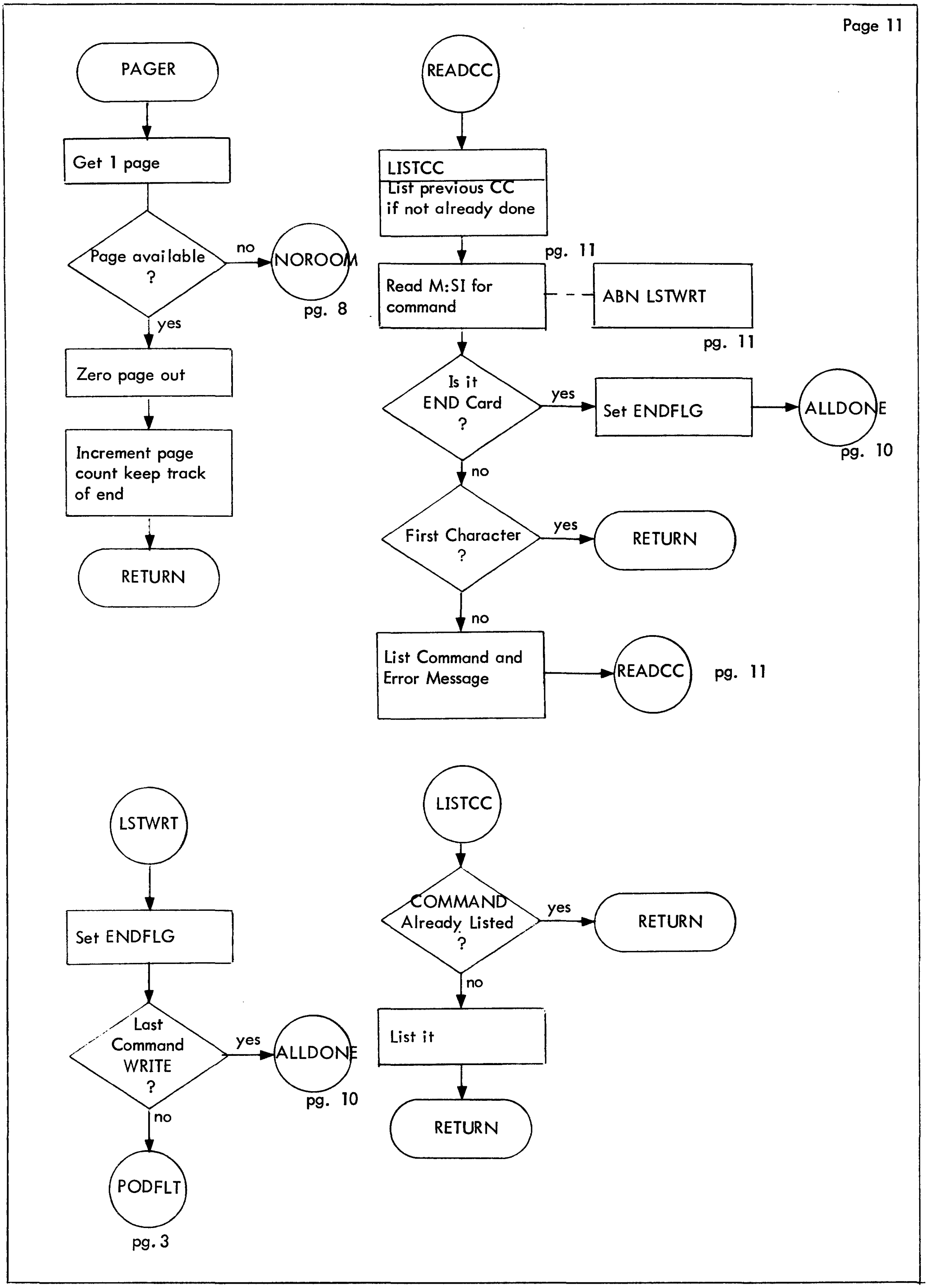


Figure C-1. Flow Diagram of DEF (Cont.)

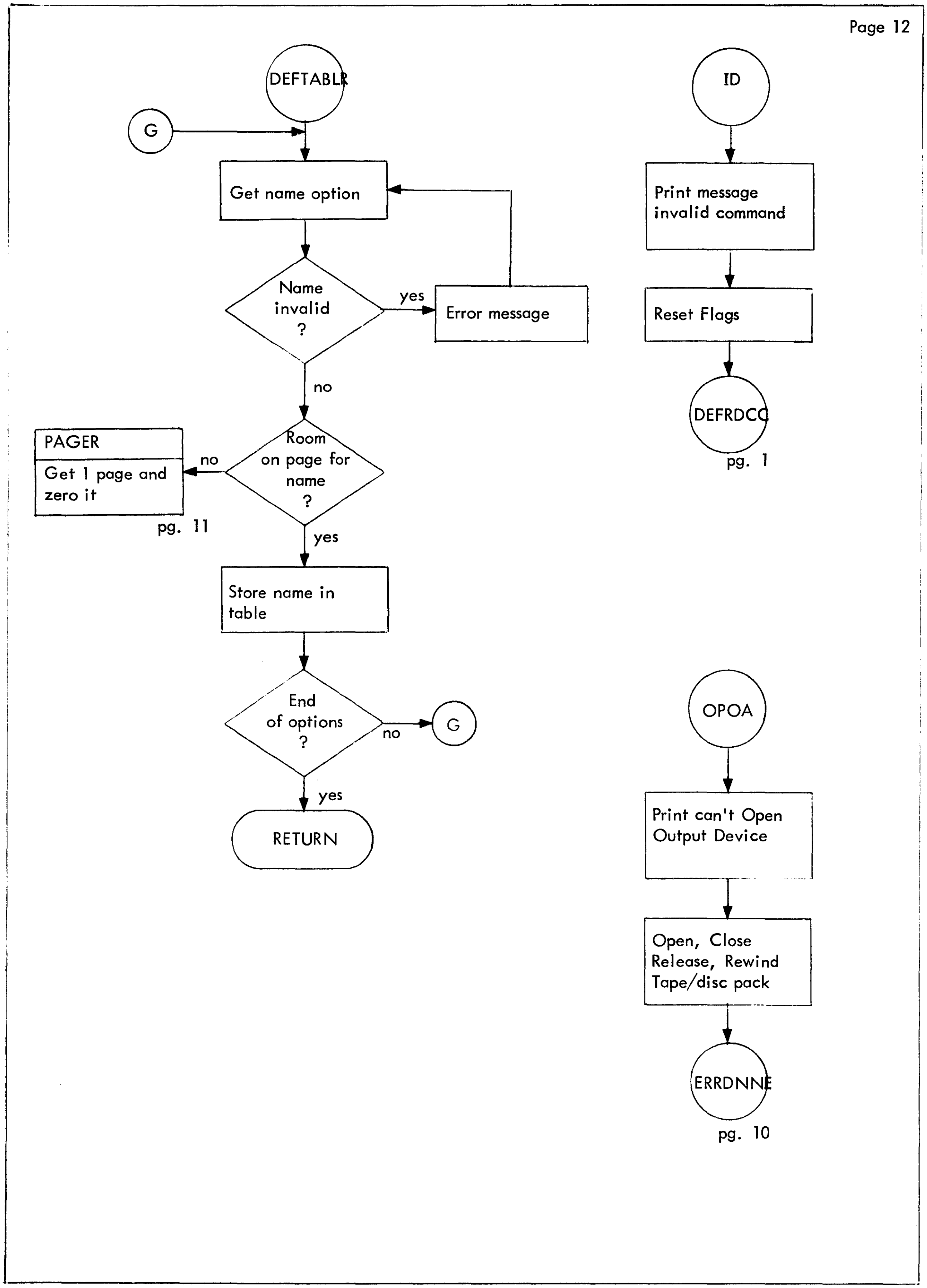


Figure C-1. Flow Diagram of DEF (Cont.)

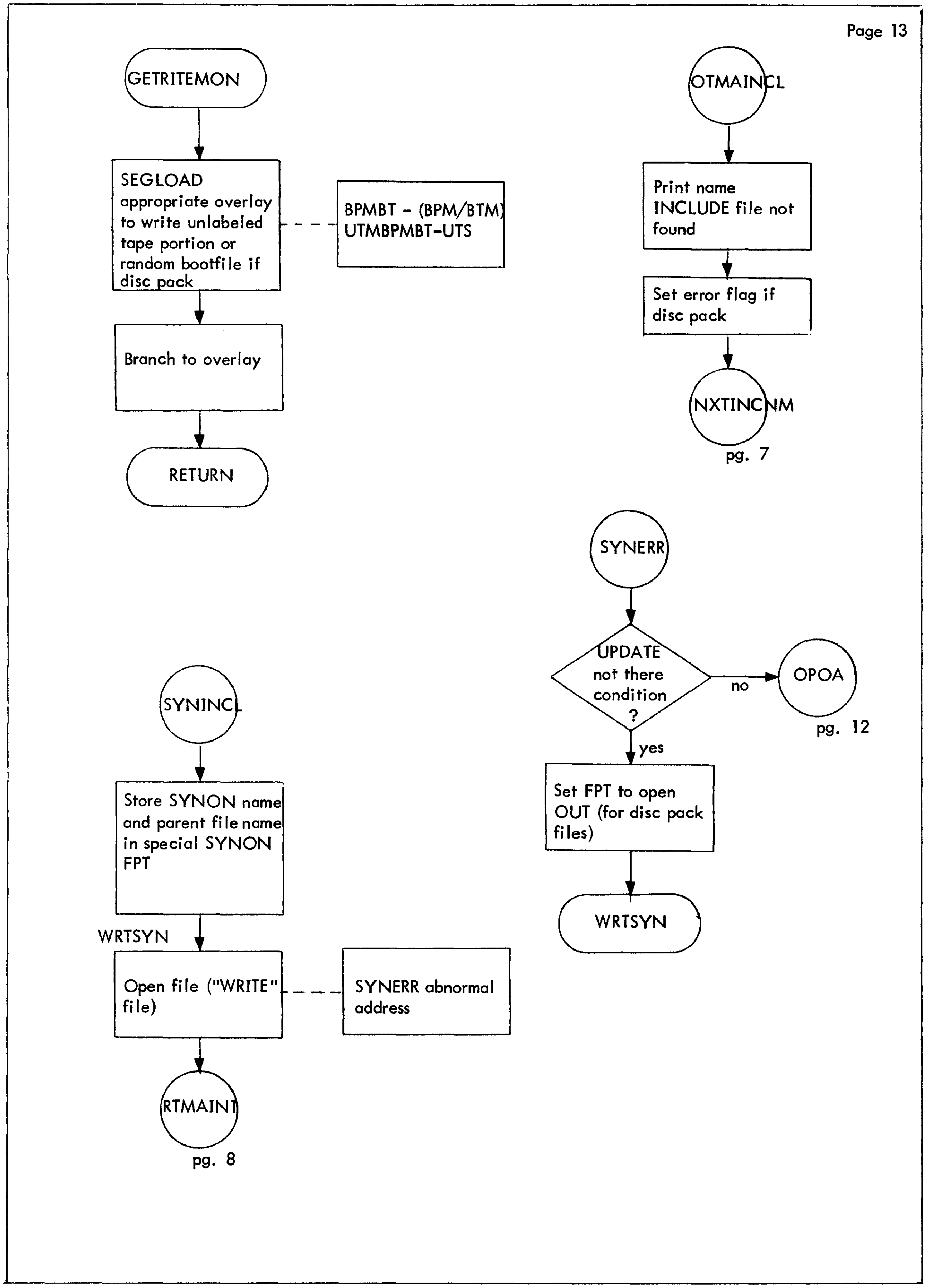

Page 13
GETRITEMON
SEGLOAD appropriate overlay to write unlabeled tape portion or random bootfile if disc pack
BPMBT - (BPM/BTM) UTMBPMBT-UTS
Branch to overlay
RETURN
OTMAINCL
Print name INCLUDE file not found
Set error flag if disc pack
NXTINCNM
pg. 7
SYNERR
UPDATE not there condition ?
no
OPOA
pg. 12
yes
Set FPT to open OUT (for disc pack files)
WRTSYN
SYNINCL
Store SYNON name and parent file name in special SYNON FPT
WRTSYN
Open file ("WRITE" file)
SYNERR abnormal address
RTMAIN1
pg. 8

Figure C-1. Flow Diagram of DEF (Cont.)

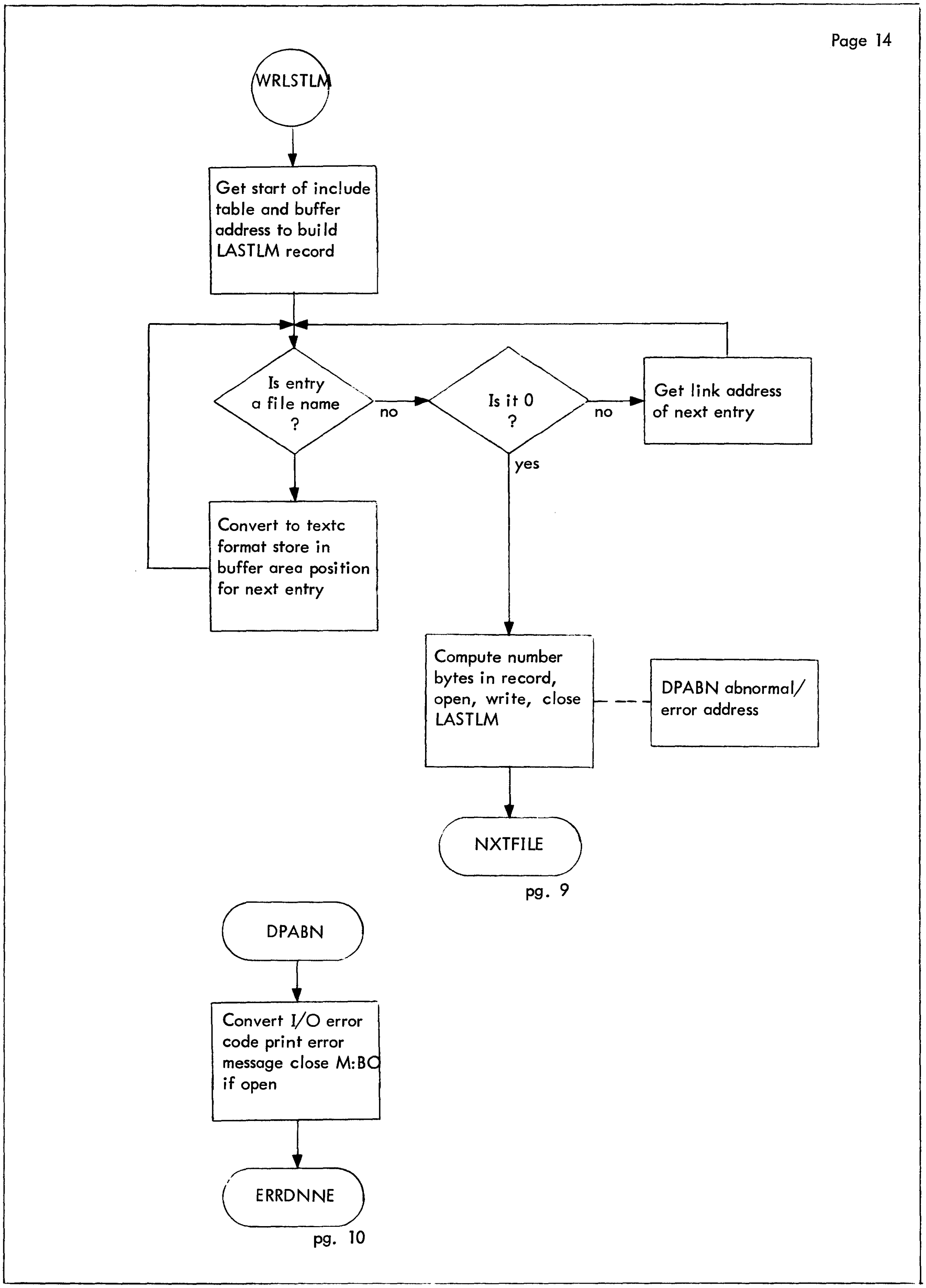

Page 14
WRLSTLM
Get start of include table and buffer address to build LASTLM record
Is entry a file name ?
no
Is it 0 ?
no
Get link address of next entry
yes
Convert to textc format store in buffer area position for next entry
Compute number bytes in record, open, write, close LASTLM
DPABN abnormal/ error address
NXTFILE
pg. 9
DPABN
Convert I/O error code print error message close M:BO if open
ERRDNNE
pg. 10

Figure C-1. Flow Diagram of DEF (Cont.)

APPENDIX D

D.0 BPM/BTM MODULE NAMES

For BPM/BTM systems only, the following is a list of the ROM names, CL labels and internal names of the modules comprising the SYSGEN processors. This information is available in LISTFILE on CI tape or BO pack releases of the system.

LMN	ROM	CL Label	Internal Names	Sect. of manual where described
PASS2	P2CCI	CN704896	M:SYSCCI2	2.1
	UBCHAN	CN704897	M:SYSDVLB2	2.2
	SDEVICE	CN704893	M:SYSSDEV2	2.3
	XMONITOR	CN704868	M:MONITOR2	2.4
	XLIMIT	CN704957	M:DLIMIT2	2.5
	ABS	CN705536	M:SYSABS2	2.6
	BTM	CN705418	BTM:CCI	2.7
	P2COC	CN706164	M:P2COC	2.8
	IMC	CN706165	M:IMC	2.9
	SPROCS	CN706163	M:SPROCS	2.10
	FRGD	CN705538	M:SYSFRGD2	2.11
	XPART	CN706293	M:XPART	2.12
	MODIFY	CN704898	M:SYSMOD	6.5
PASS3	PASS3ROM	CN705539	M:PASS3ROM	3.0
DEF	DEFROM	CN704876	M:TMTOPO	4.0, C.0
	BPMBT	CN704875	M:WRITEMON	A
	UTMBPMBT	CN706166	M:UTMBPMBT	B
LOCCT	LOCCTROM	CN705540	M:LOCCTROM	5.0

Xerox Corporation
701 South Aviation Boulevard
El Segundo, California 90245

XEROX

Reader Comment Form

We would appreciate your comments and suggestions for improving this publication.

Publication No.	Rev. Letter	Title	Current Date

How did you use this publication?

□ Learning □ Installing □ Sales
□ Reference □ Maintaining □ Operating

Is the material presented effectively?

□ Fully Covered □ Well Illustrated □ Well Organized □ Clear

What is your overall rating of this publication?

□ Very Good □ Fair □ Very Poor
□ Good □ Poor

What is your occupation?

Your other comments may be entered here. Please be specific and give page, column, and line number references where applicable. To report errors, Please use the Xerox Software Improvement or Difficulty Report (1188) instead of this form.

Your Name & Return Address

Thank You For Your Interest. (fold & fasten as shown on back, no postage needed if mailed in U.S.A.)

Staple Staple

Fold

First Class
Permit No. 229
El Segundo,
California

BUSINESS REPLY MAIL

No postage stamp necessary if mailed in the United States

Postage will be paid by

Xerox Corporation
701 South Aviation Boulevard
El Segundo, California 90245

Attn: Programming Publications

Fold

XEROX

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