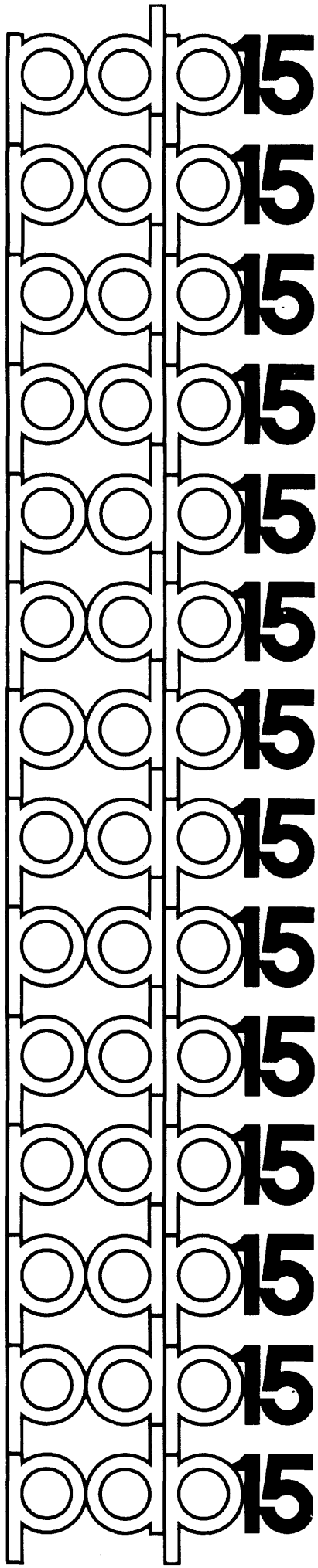
digital

dos

system manual

- system architecture
- internal operation
- maintenance

digital equipment corporation



DEC-15-ODFFA-A-D

DOS-15

SYSTEM MANUAL

FOR ADDITIONAL COPIES OF THIS MANUAL, ORDER THE NUMBER ABOVE FROM THE PROGRAM LIBRARY, DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS 01754

PRICE $10.00

First Printing, January 1972
Second Printing, July 1972

The material in this document is for information purposes and is subject to change without notice.

CONTENTS

PREFACE

This manual was written for customer systems programmers, DEC Software Specialists, and internal maintenance programmers. Readers must be familiar with the DOS User's Manual, DEC-15-MRDA-D. In addition, chapter 8 requires familiarity with the BOSS Reference Manual, DEC-15-GUDA-D.

CHAPTER 1

DOS OPERATION

The System Manager must use DOSSAV in order to load DOS-15 for the first time. The DOS System Generator manual, DEC-15-YWZB-DN12, describes DOSSAV operation in its appendix. After successful DOSSAV operation, the System Manager should load the Bootstrap into the highest bank. (This tells DOS how many banks it can use.) The Bootstrap loads the System Loader, which in turn loads the Nonresident Monitor. In order to ensure a working system, the System Manager should place the DOS-15 Checkout Package tape (RF.CHK, DEC-15-CIDA-PA, for RF DECdisk systems, or RP.CHK, DEC-15-CTAA-PA, for RPØ2 Disk Pack systems) into the Paper Tape Reader, and type BATCH PR). Operating instructions for the Checkout Package, and the tape itself, are distributed as part of the DOS-15 system.

Once the system has been checked out, the System Manager should use DOSGEN, the DOS System Generator program, to tailor the system to his needs. As mentioned in the System Generator manual, a complete tailoring of the system may also involve use of PATCH, PIP, and UPDATE.

Commands to the Nonresident Monitor allow temporary modification of the system, in order to suit the needs of a particular program. The Nonresident Monitor modifies the system by changing information in the .SCOM Table. The System Loader examines the .SCOM Table, along with three disk-resident information blocks, SYSBLK, COMBLK and SGNBLK, and carries out all operations necessary to fulfill the operator's commands. The System Loader "builds" the Resident Monitor by relocating and linking those routines indicated by the .SCOM table as needed by the next core load. The Resident Monitor then retains general control over the system.

CHAPTER 2

THE RESIDENT MONITOR

2.1 INTRODUCTION

The Resident Monitor gets its name because it seems resident to the user. Strictly speaking, however, the only part of the system that is always resident is the Bootstrap. There are two parts of the system that are refreshed only after manual Bootstrap loads and restarts: .SCOM and the Resident Monitor Patch Area. Every time an operator or program changes certain key system parameters, the system will build a new Resident Monitor from blocks stored on the system device.

The Resident Monitor is the interface between the operator, and the active devices on one hand, and the program which is running (the Nonresident Monitor), on the other. The Resident Monitor always contains the following routines and tables:

Chapter 5
- .DAT
- .UFDT
- .SCOM

This Chapter
- The CAL Handler, which routes all System and I/O Macro calls
- The Startup routine after using the Bootstrap
- .MED, the Monitor's standard error routine
- The Expanded Error Processor, for more flexibility with error messages
- Handlers for the following error conditions:
  - Nonexistent Memory
  - Memory Protect
  - Interrupt-Memory Parity
  - Power-Fail
  - Software API not set up
- The Monitor's TRAN routine (different from I/O .TRAN's)
- A clock handler
- The .GTBUF and GVBUF processor
- The CTRL Q processor
- The .USER processor
- The .OVRLA processor
- TTA.
- The Resident Monitor's Patch Area

In addition, the user can request the system to retain certain other routines in a resident Monitor status:

- The CTRL X Feature, including a driver for the VT-15
- The Paper Tape or Card Reader Handler for Batch
- The Resident Batch code

BOSS-15 also has resident routines, which are covered in Chapter 8.

2.2 THE CAL HANDLER

The CAL instruction transfers control to register 21, bank Ø, and loads register 2Ø with the address of the next instruction after the CAL. All DOS I/O and system macros take the form of a CAL instruction (possibly with some code in the low-order bits), and the next sequential register contains a dispatch code. Some macros require more information in succeeding registers. Figure 2-1, Resident Monitor CAL Handler, illustrates the operation of that portion of the Resident Monitor. The CAL Handler does only minimal error checking -- for legal function code, and for legal .DAT slot. Aside from that, and ensuring the clock is turned on, the CAL Handler is only a dispatcher to other routines.

2.3 IOPS ERROR HANDLER, AND THE EXPANDED ERROR PROCESSOR

2.3.1 .MED

There are two error processors in the Resident Monitor: .MED and the Expanded Error Processor. Figure 2-2 illustrates those routines. Figure 2-3 shows two subroutines used by the error routines. .MED (location 3, bank Ø) processes IOPS errors from all device handlers except the disk handlers, and CDB., MTF., TTA., and LPA. Calls to .MED should take the following form, if not IOPS 4:

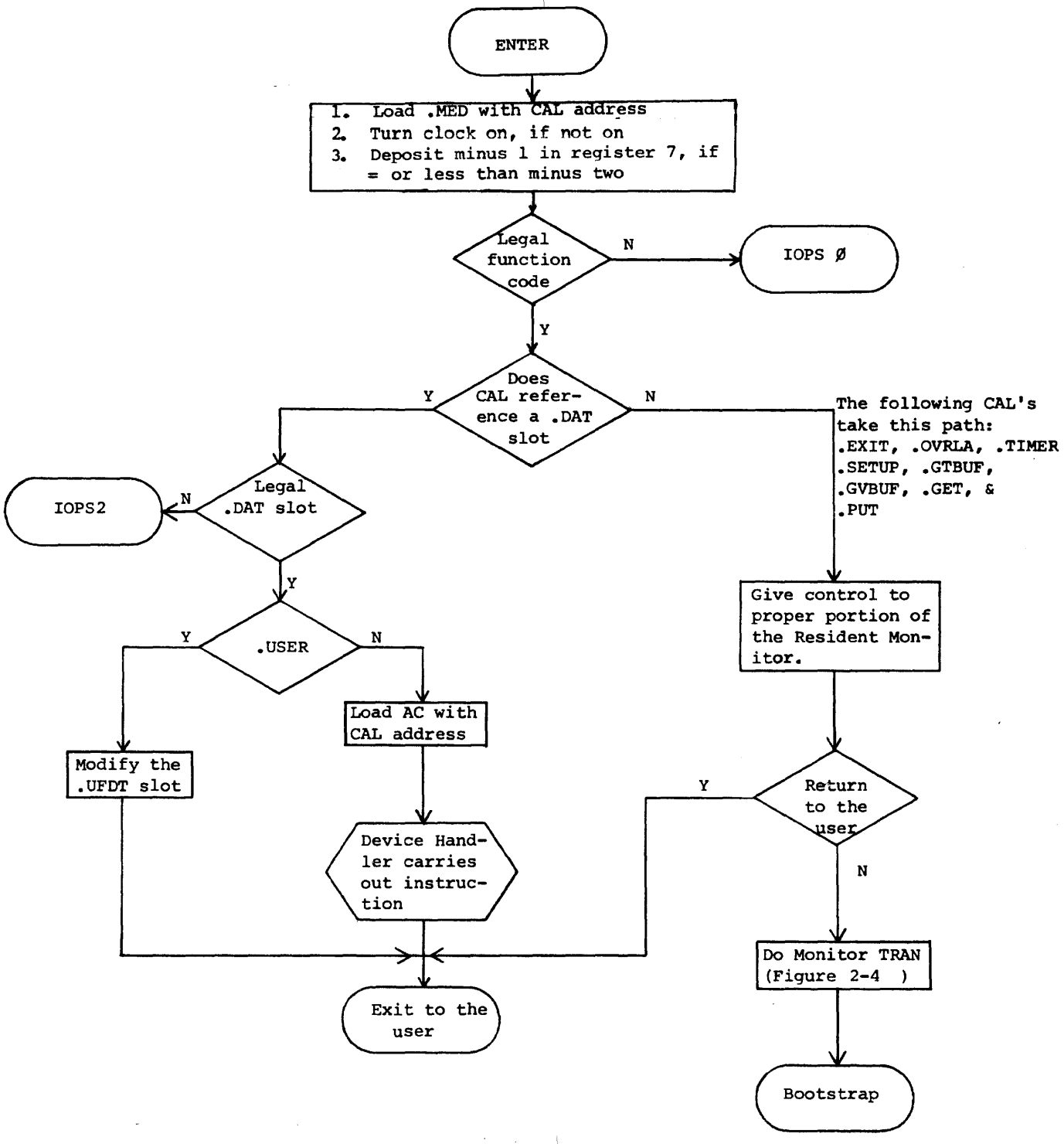
```
LAC   INFO   /ARGUMENT OF ERROR
DAC*  (.MED  /ADDRESS OF CAL IS ALREADY IN .MED,
             /IF DESIRED
LAW   N      /N IS ERROR CODE Ø≥N≥777.  AC MUST BE NEGATIVE.
JMP*  (.MED+1
```

IOPS 4 messages may take the following form:

```
LAC  (4      /AC MUST BE POSITIVE
JMS* (.MED
```

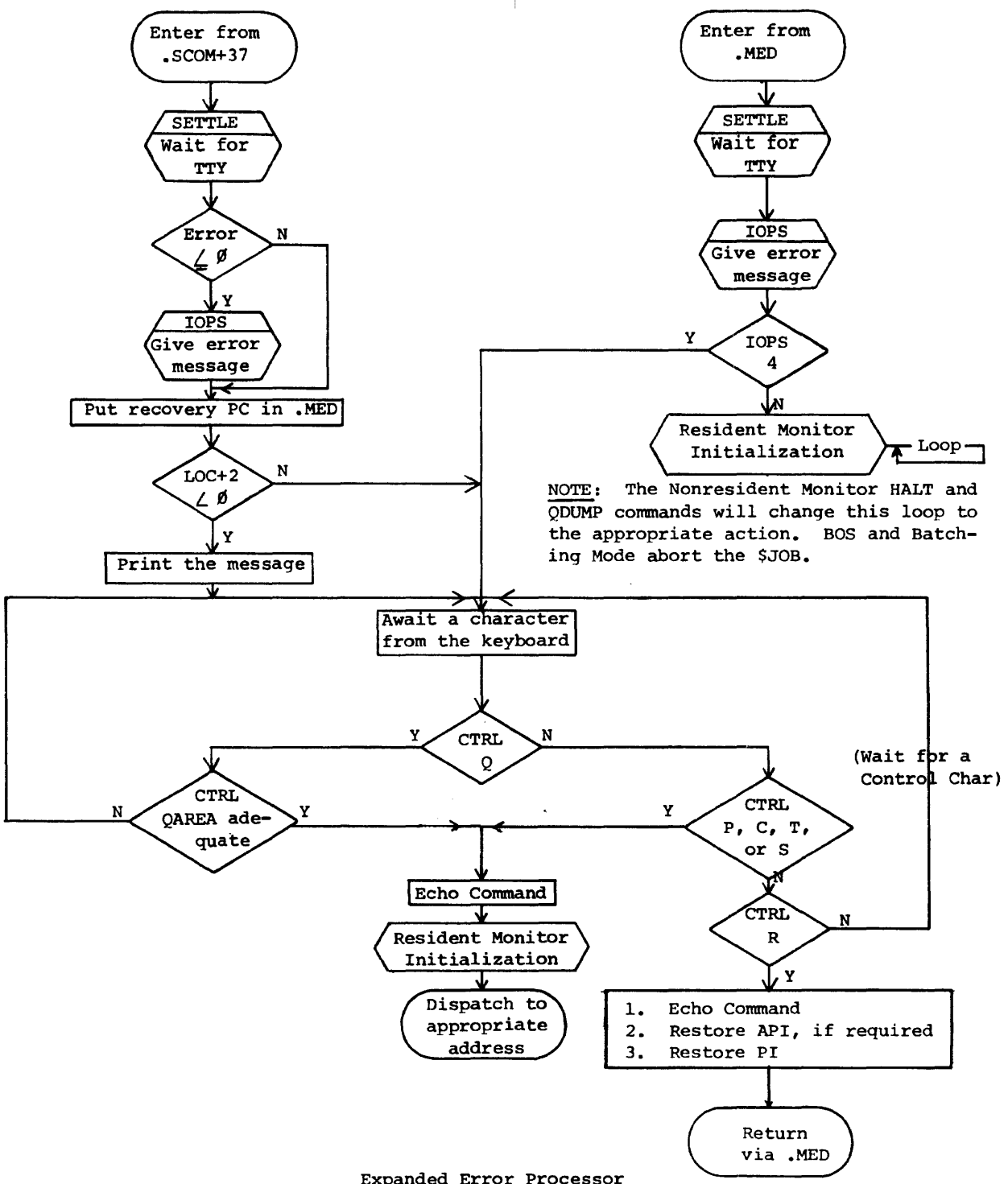
.MED+1 contains a JMP to the Monitor Error Diagnostic Routine. The above calls to .MED will cause the following printouts:

```
IOPSN (contents of .MED)
IOPS4
```



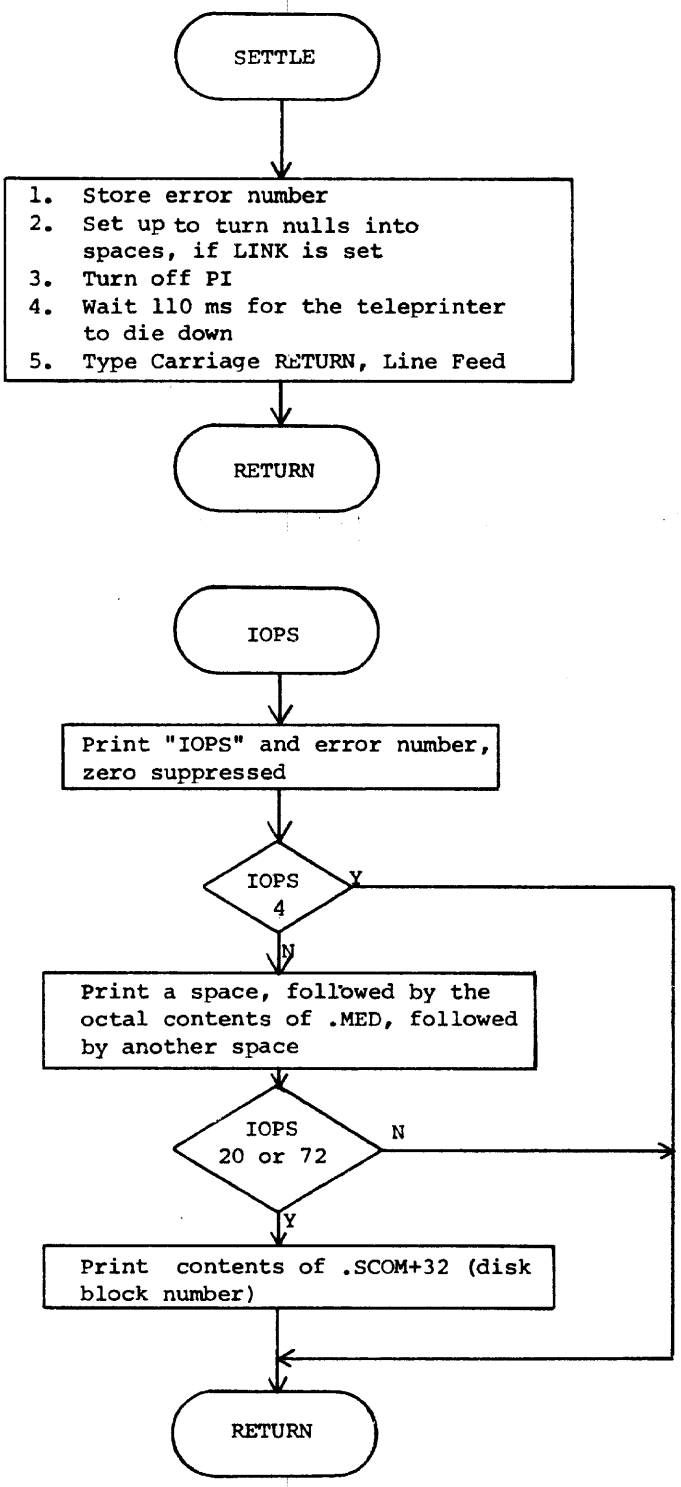
Resident Monitor CAL Handler

Figure 2-1



Expanded Error Processor
and
Monitor Error Diagnostic Routine
.MED

Figure 2-2



Resident Monitor Subroutines

Figure 2-3

2.3.2 The Expanded Error Processor

The disk handlers (except the Bootstrap), CDB., MTF., TTA., and LPA. use the Expanded Error Processor. Each error message is "potentially" recoverable by typing CTRL R. That is, the Resident Monitor always returns control to the caller upon a CTRL R. It is up to the caller to respond accordingly. All handlers supplied with the system simply repeat the error message if the error is unrecoverable.

The Expanded Error Processor gives the capability of printing additional information after the standard IOPS message. As with .MED, the AC must contain the error number (Ø≤number≤777) in bits 9-17. Control must be passed, however, via JMS* (.SCOM+37, not JMP* (.MED+1.

The following information pertains to the message: LOC+2 must contain the two's complement of the number of message words to be typed after the standard "IOPSNN nnnnnn" message. If the number is zero or positive, no message will be printed. If the LINK is set, nulls will be printed as spaces. If the LINK is zero, nulls will be ignored. If the AC is positive on calling the expanded error facility, only the special message will be printed. The "IOPS" part will be omitted. The message itself must be packed in .SIXBT.

The following are examples of use of the Expanded Error Processor:

Example a:

```
        UNREC  LAC    STATUS          /STATUS
               DAC*   (.MED           /CAL ADDRESS IS NOW OVERWRITTEN
                                      /BY CONTENTS OF STATUS REGISTER
               STL                    /TURN NULLS INTO SPACES
               LAW    ERRNUM          /≤ERRNUM<1ØØØ
               JMS*   (.SCOM+37
               JMP    UNREC           /THIS IS AN UNRECOVERABLE ERROR.
                                      /JMP .-1 WILL NOT DO -- EXPANDED
                                      /ERROR PROCESSOR CHANGES THE
                                      /CONTENTS OF .MED.
               LAW    -1
               .SIXBT 'DKA'
        UNITNO Ø
        .SIXBT 'FIL'
        .SIXBT 'E'
        .SIXBT 'SRC'
```

The printout from that code will be as follows:

IOPS777 nnnnnn DKA FILE SRC

where nnnnnn is the contents of .MED, and equals the Status Register B, and ERRNUM was 777.

Example b:

```
PARITY  LAW    61
        STL                 /TURNS NULLS INTO SPACES
        JMS*   (.SCOM+37
        JMP    RETRY        /THIS IS A RECOVERABLE ERROR
        LAW    -1
        .SIXBT 'DTA'
```

The printout from that code will be as follows:

```
IOPS61 nnnnnn DTA
```

where nnnnnn is the contents of .MED, the address of the last CAL, deposited by the CAL Handler.

2.4 THE SYSTEM BOOTSTRAP

The System Bootstrap is nothing more than a disk driver. It may load the System Loader and Resident Monitor from Hardware Readin, or manual restart. All other Bootstrap operations result from the use of the Monitor TRAN routine. The Monitor TRAN routine sets up the Bootstrap to read or write any block or set of contiguous blocks from the disk to or from any location in core. Before calling the Bootstrap, the Monitor TRAN does a .WAIT to all .DAT slots in the Mass Storage Busy Table, clears all flags, turns off the VT if it was on, and allows the clock to tick positive, so that it will keep time but not interrupt. After the Bootstrap has finished, it calls the Monitor Initialization Routine, which updates the clock and turns on the VT, if necessary.

The Monitor TRAN Routine requires the following parameter table:

```
PARADD  LOC+Ø   BLKNUM          /FIRST BLOCK NUMBER
        LOC+1   FIRSTA-1        /FIRST ADDRESS OF BUFFER, MINUS ONE
        LOC+2   -SIZE           /# OF WORDS TO BE TRANSFERRED IN 2'S COM
        LOC+3   START           /STARTING ADDRESS AFTER DISK I/O
                                /COMPLETION
```

The following code illustrates the use of the Monitor TRAN:

```
        UNIT=1ØØØØØ             /MONITOR TRAN WILL USE UNIT ONE[1]
        .
        .
        .SCOM=1ØØ
        .
        .
        .
        LAC    (PARADD          /MONITOR TRAN REQUIRES ADDRESS OF
        XOR     UNIT            /PARAMETER TABLE IN BITS 3-17 AND
                                /UNIT NUMBER IN BITS Ø-2 OF AC
        STL                     /NONZERO LINK GIVES TRAN OUT
        JMP*    (.SCOM+55       /.SCOM+55 IS USER ENTRY POINT FOR
                                /MONITOR TRAN
```

See also paragraph 5.7.

[1]DECdisk TRANs ignore unit number, use block number.

.OVRLA, .EXIT, and manual Q dumps all use the Monitor TRAN routine. Figure 2-4, .OVRLA, .EXIT and CTRL Q, illustrates their operation, and also the Monitor TRAN.

For the RF DECdisk, the user can reference a specific platter just by identifying the block number he wants. That is, the block numbers to not automatically go to zero at the beginning of every platter. The block numbers and platter relationships are shown below:

TABLE 2-1

RF Platter-Block Number Correspondence

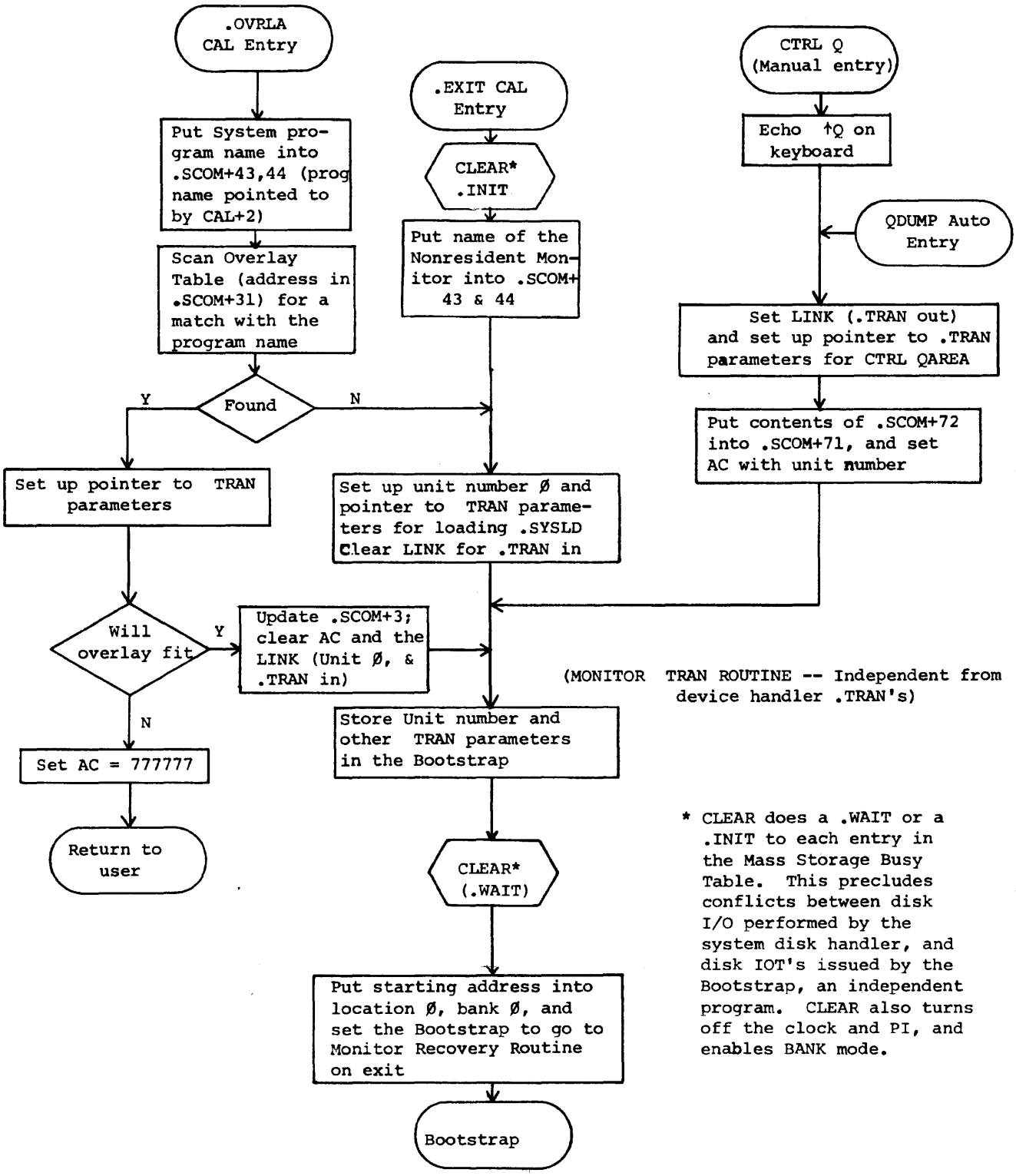
Platter Number	Block Number
Ø	Ø-1777
1	2ØØØ-3777
2	4ØØØ-5777
3	6ØØØ-7777
4	1ØØØØ-11777
5	12ØØØ-13777
6	14ØØØ-15777
7	16ØØØ-17777
(All numbers are in octal)	

2.5 SYSTEM I/O INITIALIZATION

There are two routines that do DOS I/O initialization: the startup routine after Bootstrap manual loads and restarts, and the startup routine performed after Monitor TRAN's and after a CTRL C, P, T or S for an error. The startup routine after Bootstrap loads is described in Figure 4-1, The System Loader Interface Routine. Figure 2-5, Resident Monitor Initialization, describes the other routine.

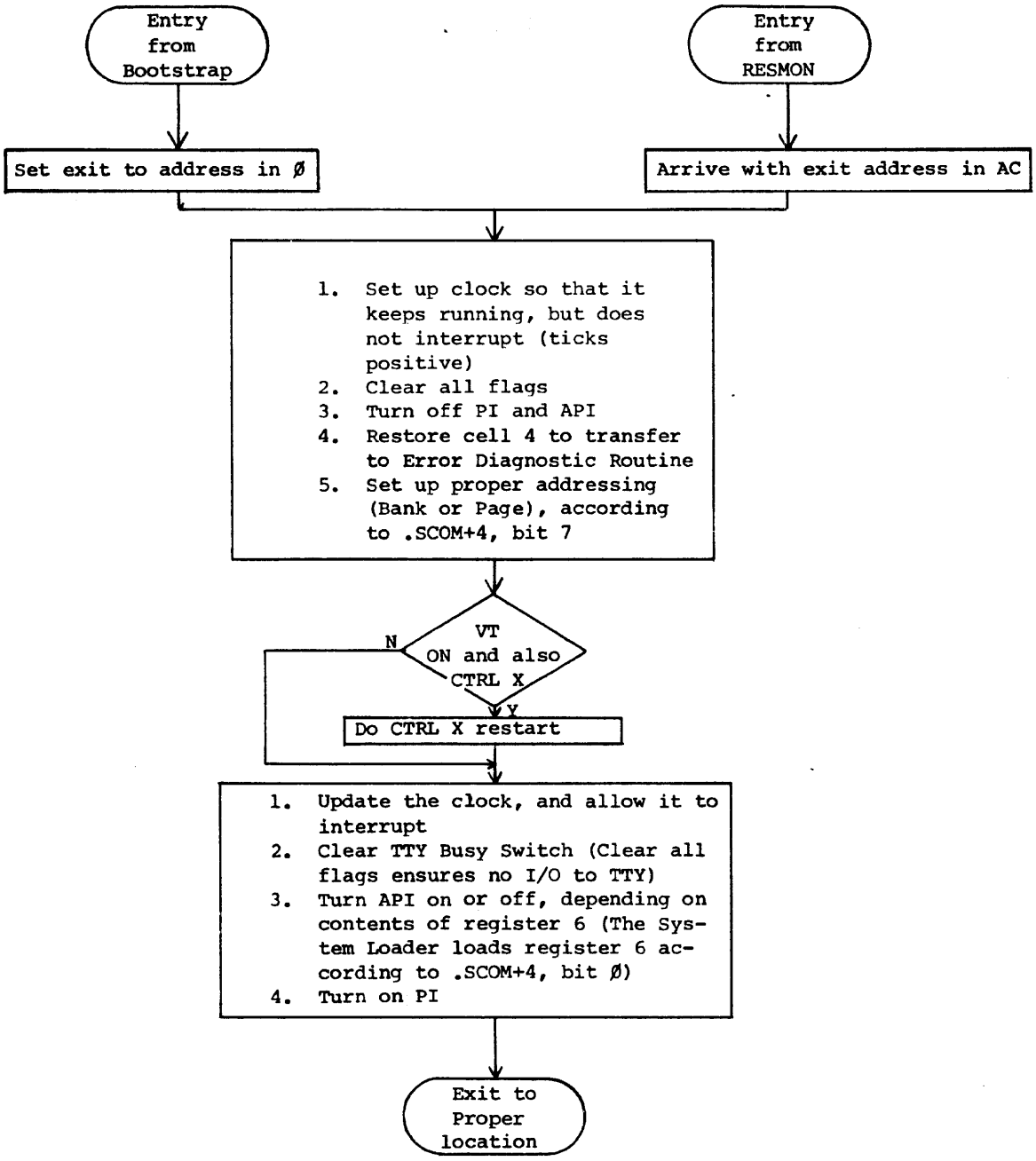
2.6 RESIDENT MONITOR TIMING FEATURES

Figure 2-6, The Resident Monitor Clock Routine, describes the Resident Monitor's time functions. There are three places in DOS which start or try to update the clock -- (1) the first-time initialization after manual Bootstrap loads and restarts, (2) the Resident Monitor Initialization, and (3) the CAL Handler. The following .SCOM registers contain timing information:



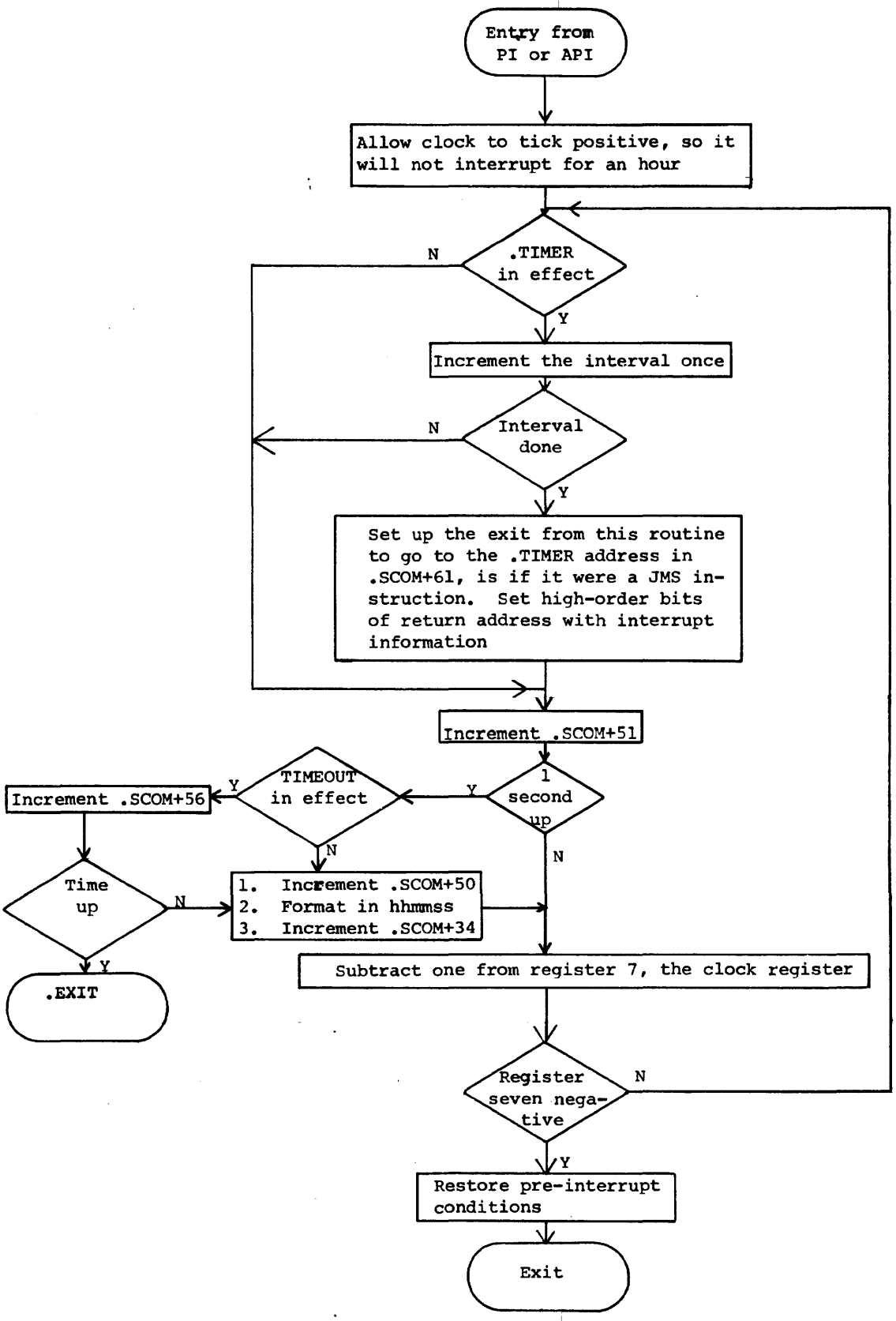
.OVRLA, .EXIT and CTRL Q

Figure 2-4



Resident Monitor Initialization

Figure 2-5



Note: The Clock Routine will use PI if API is busy, or down.

The Resident Monitor Clock Routine

Figure 2-6

.SCOM+5Ø	Time of day, in hhmmss (six bits each)
.SCOM+51	Elapsed time, in ticks
.SCOM+56	Time limit, in seconds (zero, if no limit)
.SCOM+6Ø	Time left for .TIMER interrupt (zero, if .TIMER not in effect)
.SCOM+61	Address of .TIMER user interrupt routine
.SCOM+73	Number of ticks left in the next second
.SCOM+74	Line frequency, in ticks per second

2.6.1 Clock Operation

The Nonresident Monitor's TIME command changes or senses .SCOM+5Ø. .SCOM+51 is not used by any system program. The clock handler simply increments it upon each clock tick. User programs may deposit a known quantity into .SCOM+51, in order to time events. The Nonresident Monitor deposits the argument for a TIMEST command into .SCOM+56. If .SCOM+56 is nonzero, the Resident Monitor will issue an ISZ .SCOM+56 command each second, until it reaches zero. At such a time, the Resident Monitor will perform a .EXIT. MICLOG, LOGIN, and LOGOUT clear **.SCOM+56.**

2.6.2 .TIMER

.TIMER allows users to schedule routines for a specified time from "now". These routines may return to the interrupted code, if the programmer desires. .TIMER users should take care that the time-dependent code follows certain rules:

a. When a programmer does not wish to reset the .TIMER mechanism, but wishes to return to the interrupted program, his code should look like this:

```
C           Ø           /C+1 REACHED VIA JMS
     DAC    SAVEAC      /MUST NOT USE NON-REENTRANT CODE
     .      .           /POSSIBLY USED BY THE INTERRUPTED
     .      .           /PROGRAM.  (INCLUDES THE CAL IN-
     .      .           /STRUCTION)
     LAC    C           /RESTORE THE LINK
     RAL
     LAC    SAVEAC      /RESTORE THE AC
XIT  JMP*   C
```

b. When the programmer does wish to reset the .TIMER mechanism, and return to the interrupted code, his routine should look like this:

```
        .SCOM=1ØØ
        CLON=7ØØØ44
        CLOF=7ØØØØ4
        INTRVL=-1ØØ      /THIS ROUTINE WILL RUN EVERY 1ØØ(8)+
        .                /TICKS
        .
C       Ø
        DAC   SAVEAC
        .
        .
        .
        LAC   ADDRES     /RETURN TO THE NEXT ROUTINE
        DAC*  (.SCOM+61
        CLOF             /TURN THE CLOCK OFF TO ENSURE NO
                         /REENTRANCE BEFORE .TIMER RESET AND
                         /RETURN
        LAC   INTRVL     /DESIRED INTERVAL IN TWO'S COMPLEMENT
        DAC*  (.SCOM+6Ø
        LAC   C          /RESTORE THE LINK
        RAL
        LAC   SAVEAC     /RESTORE THE AC
        CLON             /TURN THE CLOCK BACK ON (AFTER NEXT
                         /INSTRUCTION)
        JMP*  C
```

c. When a programmer does not wish to return to the interrupted program, he need not save the AC, and he may use the CAL instruction. He should beware of using I/O buffers that may still be modified by a handler's interrupt section. In many cases, a .INIT to an active .DAT slot will terminate I/O. Teleprinter I/O should be terminated by the following:

```
XCT*  (.SCOM+35
```

The user should program a delay of at least 11Ø milliseconds after such an instruction, before he attempts teleprinter I/O.

Note: The interrupt routine will run at the level of the interrupted code, with the same addressing mode and memory protect status. Thus, no debreak and restore is required.

2.7 THE RESIDENT MONITOR PATCH AREA

There are two types of patch area taken from the space allocated at assembly time:

1. That allocated by using PATCH
2. That allocated when answering the Patch Area question in system generation

Patch area one is the place for permanent changes to the Resident Monitor. It is always refreshed when the System Loader comes into core. Patch area two is only refreshed on manual Bootstrap loads and restarts. The second area would be appropriate for communication between successive programs loaded by the System Loader. This area should be used because the System Loader refreshes all of core, except the Bootstrap, .SCOM, the CTRL X buffer, and the patch area two.

The combined size is limited by the current assembly at $47ØØ_8$. Both areas can be initialized, using PATCH. The important dividing line between area one and area two is register 1Ø1 (.SCOM+1) of RESMON. The way to allocate more space in part one is to increase the value of register 1Ø1. The way to change the area in part two is to use DOSGEN. The second part will start at the address in register 1Ø1. The upper bound of the second area will be the sum of the contents of register 1Ø1, and the number specified to DOSGEN.

2.8 CONTROL CHARACTERS

CTRL C, P, R, S, and T are all special characters that interrupt the current program and transfer control. The Resident Monitor ignores CTRL R except after IOPS 4 and any call to the Expanded Error Processor. CTRL S always transfers control to the address in .SCOM+6. In the case of core-image system programs and EXECUTE, a CTRL S will transfer to register zero, and result in an IOPS 3. The Linking Loader places the starting address of the first load module into .SCOM+6.

A .INIT macro to the teleprinter handler will change the address of either CTRL C, P or T. The Resident Monitor is always initialized to

perform a .EXIT after CTRL C, and ignore CTRL P and T. DDT uses CTRL T, and CTRL P is ordinarily used by programs for restarts. MACRO-15 expands .INIT to change the CTRL P address. If the programmer expands .INIT without the aid of the assembler, a 1Ø in bits zero and one of LOC+2 will change the address of CTRL T. A Ø1 in those bits will change the address of CTRL C. It should be obvious that special care should be taken with CTRL C. In addition, modifications to the CTRL T address should not be made when debugging with DDT. There are cases, however, when such modifications are desirable. In particular, all zeroes in LOC+2 (2-17) will cause the teleprinter handler to ignore CTRL C, P, or T. This address might be used when sensitive code is being executed, as in DOSGEN. The following .INIT expansion will cause the Resident Monitor to ignore CTRL C:

```
CAL-2&777
1
2ØØØØØ
```

CHAPTER 3

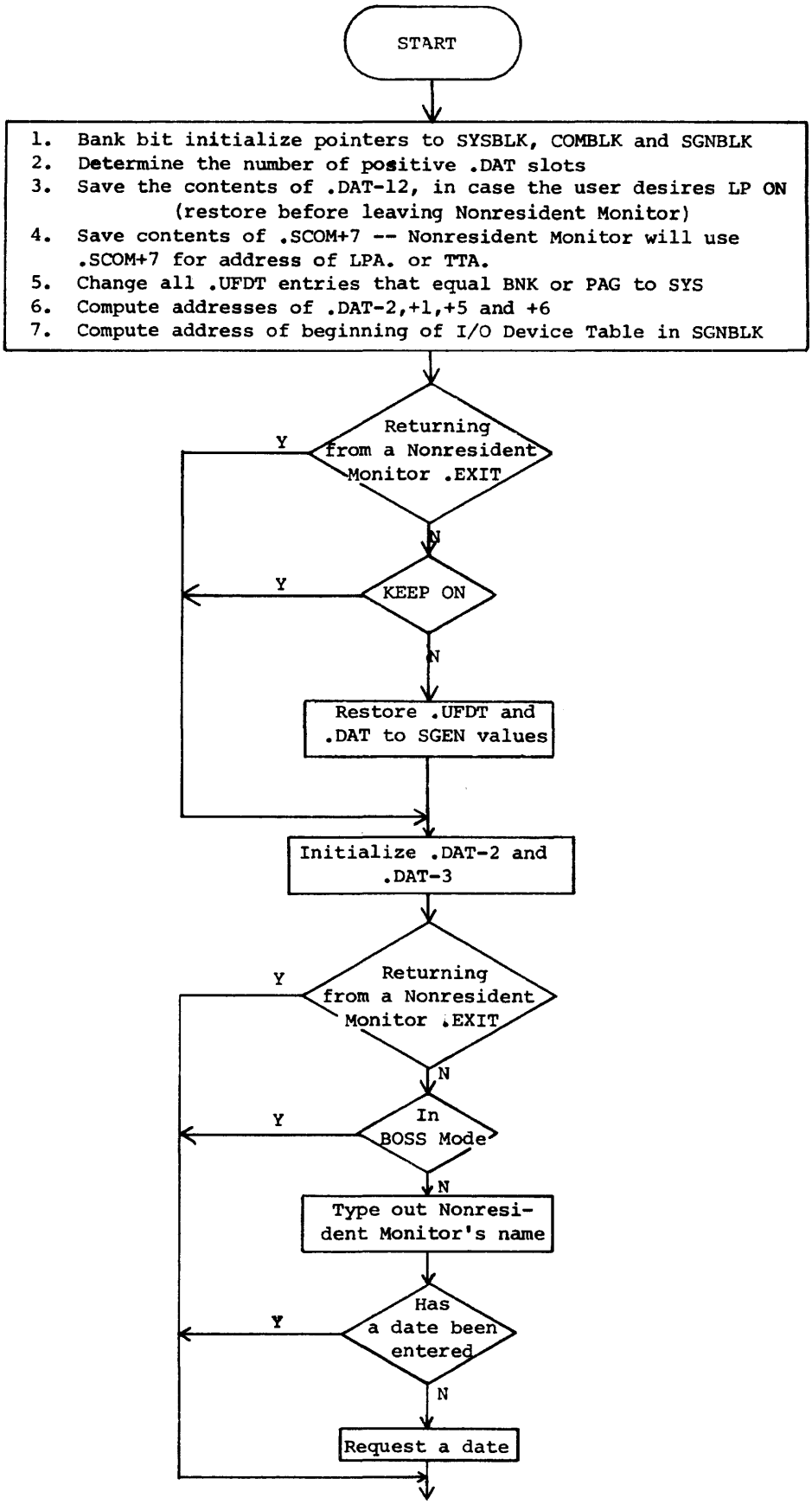
THE NONRESIDENT MONITOR

3.1 INTRODUCTION

The System Loader brings the Nonresident Monitor into core after a hardware readin, a manual restart, a CTRL C, or a .EXIT. The RCOM Table, SGNBLK, SYSBLK and COMBLK are always coresident with the Nonresident Monitor. This gives the Nonresident Monitor access to all important system parameters.

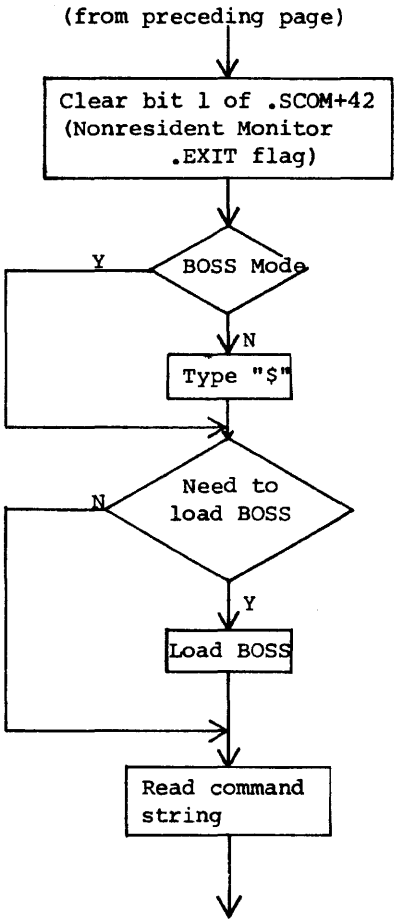
The Nonresident Monitor announces its presence by typing DOS-15 VNA on the teleprinter. It remains in core until the operator requests another system program, or until the operator's command implies a refreshed configuration of the Resident Monitor is necessary.

The Nonresident Monitor's actions are limited to (1) decoding commands, (2) manipulating or examining bits and registers in .SCOM, .DAT, .UFDT, SYSBLK, COMBLK, and SGNBLK, and (3) calling the System Loader, when necessary. The Nonresident Monitor has only one entry, which starts an initialization section. Figure 3-1, Nonresident Monitor Initialization, describes that logic. Every time the System Loader brings in the Nonresident Monitor, it passes control to the initialization section. After initialization, and after all commands that do not require the System Loader, the Nonresident Monitor types a $, and awaits an input line, terminated by a Carriage RETURN or an ALT MODE. It then examines the first six characters (or those up to the first blank) and tries to find an entry in the Nonresident Monitor's Command Table. If a match is found, control passes to the appropriate routine, and thence to the next command, or the System Loader. If the typed command does not correspond to an entry in the command table, the Nonresident Monitor temporarily assumes the operator wishes a new core-image system program, and checks COMBLK for a corresponding entry. If there is no corresponding entry in COMBLK, the Nonresident Monitor will type an error message, and await the next command. If COMBLK contains a matching entry, the Nonresident Monitor composes a .OVRLA, and passes control to the System Loader via that .OVRLA.



Nonresident Monitor Initialization

Figure 3-1



(Continue to Command Decoder)

Nonresident Monitor Initialization
(continued)

Figure 3-1 (Cont.)

3.2 COMMANDS TO THE NONRESIDENT MONITOR

This paragraph discusses legal commands listed in the Nonresident Monitor's Command Table. Table 3-I, Effects and Exits for Nonresident Monitor Commands, describes all commands that do not request a new program.

There are five entries in the Command Table that load relocatable system programs. They are DDT, EXECUTE, GLOAD and LOAD. The Nonresident Monitor treats these commands separately, because SYSBLK does not list them. All information necessary for loading these programs resides in the Nonresident Monitor itself.

3.3 CONSIDERATIONS FOR ADDITIONS TO THE NONRESIDENT MONITOR

Programmers should not attempt to add commands to the Nonresident Monitor unless they have access to a copy of the source code. The source code may be purchased from Digital Equipment Corporation, 146 Main Street, Maynard, Massachusetts, under one of the order numbers listed in the footnote. They should then use the EDITOR program to put in the indicated changes, and reassemble.

New additions to the Nonresident Monitor require the following actions:

1. Update the Nonresident Monitor's Command Table.
   The Command Table is in two parts:
   a) The .SIXBT names of the commands
   b) The corresponding transfer vector
2. Write the code for the command.
3. Consider the kind of exit the command will take:
   a) Commands that end with a request for a new command should end with JMP KLCOM
   b) Commands that re-configure the Nonresident Monitor should end with JMP NRMEX1.

DECtape DEC-15-SRDA-U1

Magtape Unavailable

Table 3-I

Effects and Exits
for Nonresident Monitor Commands*

COMMAND	MODIFIER	ACTION TAKEN	EXIT
API	ON OFF	Set bit Ø of .SCOM+4. Clear bit Ø of .SCOM+4.	.EXIT .EXIT
ASSIGN	handler (and/or) UIC	Check whether handler is available. If yes, load .DAT slot with proper handler code. (The proper loader will load the handler, and insert its starting address into the .DAT slot. Load proper slot via a .USER	Next Command Next Command
BANK	ON OFF	Set bit 11 of .SCOM+4. Clear bit 11 of .SCOM+4.	Next Command
BATCH	PR CD	Set bit Ø and clear bit 2 in location 1777 of the Bootstrap's bank. If bit 2 of .SCOM+33 is set (i.e., if VT is ON) and bit 17 of .SCOM+33 is set (i.e., CTRL X is set for VT), set bit 1 of .SCOM+33 in order to tell the Resident Monitor Initialization to start up CTRL X. Set bits Ø and 2 of location 1777 of the Bootstrap's bank, and set bit 1 of .SCOM+33 as with BATCH PR.	.EXIT .EXIT
BUFFS	number	Put number indicated into .SCOM+26, and set Nonresident Monitor Initialization to leave .SCOM+26 alone.	Next Command
CHANNEL	7 9	Clear bit 13 of .SCOM+4. Set bit 13 of .SCOM+4	Next Command
DATE	date no date	Enter date into .SCOM+47. Print date from .SCOM+47.	Next Command

* This table assumes error-free input.

Table 3-I (cont.)

Effects and Exits
for Nonresident Monitor Commands

COMMAND	MODIFIER	ACTION TAKEN	EXIT
GET GETP GETS GETT		Set Section 3.4.	
HALF	ON OFF	Set bit Ø of .SCOM+33. Clear bits Ø and 1 of .SCOM+33.	.EXIT .EXIT
HALT		If not in BOSS-15 mode, put a HLT instruction (instead of a JMP) into the exit from non-IOPS 4 errors to .MED. If in BOSS mode, do nothing.	Next Command
INSTRUCT	none ERRORS	Print INSALL Table. Print INSERR Table.	Next Command
KEEP	ON OFF	Set bit 16 of .SCOM+42. Clear bit 16 of .SCOM+42. Initialize to SGEN default values all entries in .DAT and .UFDT, except change SCR default values to current UIC.	Next Command
LOG		Output five spaces after Carriage RETURNs. After ALT MODE, go to next command.	Next Command (after ALT MODE)
LOGIN	uic	Redefine current UIC (.SCOM+41). Clear bit Ø of .SCOM+42, reset variable system parameters to SGEN default values zero .SCOM+56.	.EXIT
LOGOUT		Set current UIC to SCR. Set .UFDT entries to SGEN default parameters. Deposit zero into .SCOM+42 and 56.	.EXIT
LOGW		For BOSS-15, print message. In all cases, after a Carriage RETURN, output five spaces. After ALT MODE, type four bells ↑P, and await CTRL P. After CTRL P, go to next command.	Next Command (after ALT MODE)

Table 3-I (cont.)

Effects and Exits
for Nonresident Monitor Commands

COMMAND	MODIFIER	ACTION TAKEN	EXIT
LP	ON OFF	Set bit 3 of .SCOM+42. Clear bit 3 of .SCOM+42.	.EXIT .EXIT
MICLOG	mic	Check key with SGNBLK. If correct, set bit Ø of .SCOM+42, make "SYS" the current UIC, and zero .SCOM+56. If incorrect, ignore command.	Next Command
PAGE	ON OFF	Clear bit 11 of .SCOM+4. Set bit 11 of .SCOM+4.	Next Command
PROTECT	n	If n is between Ø and 7, inclusive, enter it into .SCOM+54.	Next Command
PUT		See Section 3.4.	
QDUMP		Enter MANSAV, the address of the manual CTRL Q, into the exit from non-IOPS 4 errors to .MED.	Next
REQUEST	none USER prog	Print the current assignments for .DAT and .UFDT. Print the current assignments for all positive .DAT and .UFDT slots. Print required .DAT and .UFDT slots, and the assignments and use for each.	Next Command
.SCOM		Print the information for the current system.	Next Command
TIME	time none	Enter time into .SCOM+5Ø. Print time from .SCOM+5Ø.	Next Command
VT	ON OFF	Set bit 2 of .SCOM+33. Clear bits 1, 2, and 17 of .SCOM+33. Execute 7Ø3Ø44.	.EXIT
X4K	ON OFF	Enter 4ØØØØØ into .SCOM+2Ø. Deposit zero into .SCOM+2Ø.	Next Command
33TTY	ON OFF	Clear bit 2 of .SCOM+4. Set bit 2 of .SCOM+4.	.EXIT

4. After assembly, the programmer must call PATCH, in order to make his relocatable binary program absolute. Commands to PATCH should be as follows:

```
>DOS15
>READR 16Ø77 DOSNRM BIN
```

16Ø77 indicates the highest location the new monitor can occupy. (SYSBLK begins at 161ØØ.) DOSNRM BIN happens to be the file name used by program development. The programmer may, of course, substitute his own file name. More information may be found in the PATCH manual -- DEC-15-YWZB-DN5.

3.4 QFILE

QFILE is a system program that allows users to (1) store core images in named files, and (2) retrieve such core images for examination via DUMP (or possibly for a slow, core-swapping capability). QFILE implements the following Resident Monitor system macros and Nonresident Monitor commands:

.GET, GET, GETP, GETS, GETT, .PUT and PUT

Users can not obtain QFILE by typing its name to the Nonresident Monitor. The Resident Monitor will load QFILE as part of its response to the commands and macros listed above.

PUT creates a file that contains the data in the CTRL QAREA; .PUT creates a file from the current core image. GET, GETP, GETS, GETT and .GET all overlay core with the contents of the QAREA or file. (The different commands specify different startup locations.) In addition to the above capabilities, the Resident Monitor provides the capability of overlaying core with the contents of the CTRL Q area. The following instructions show how to use that routine:

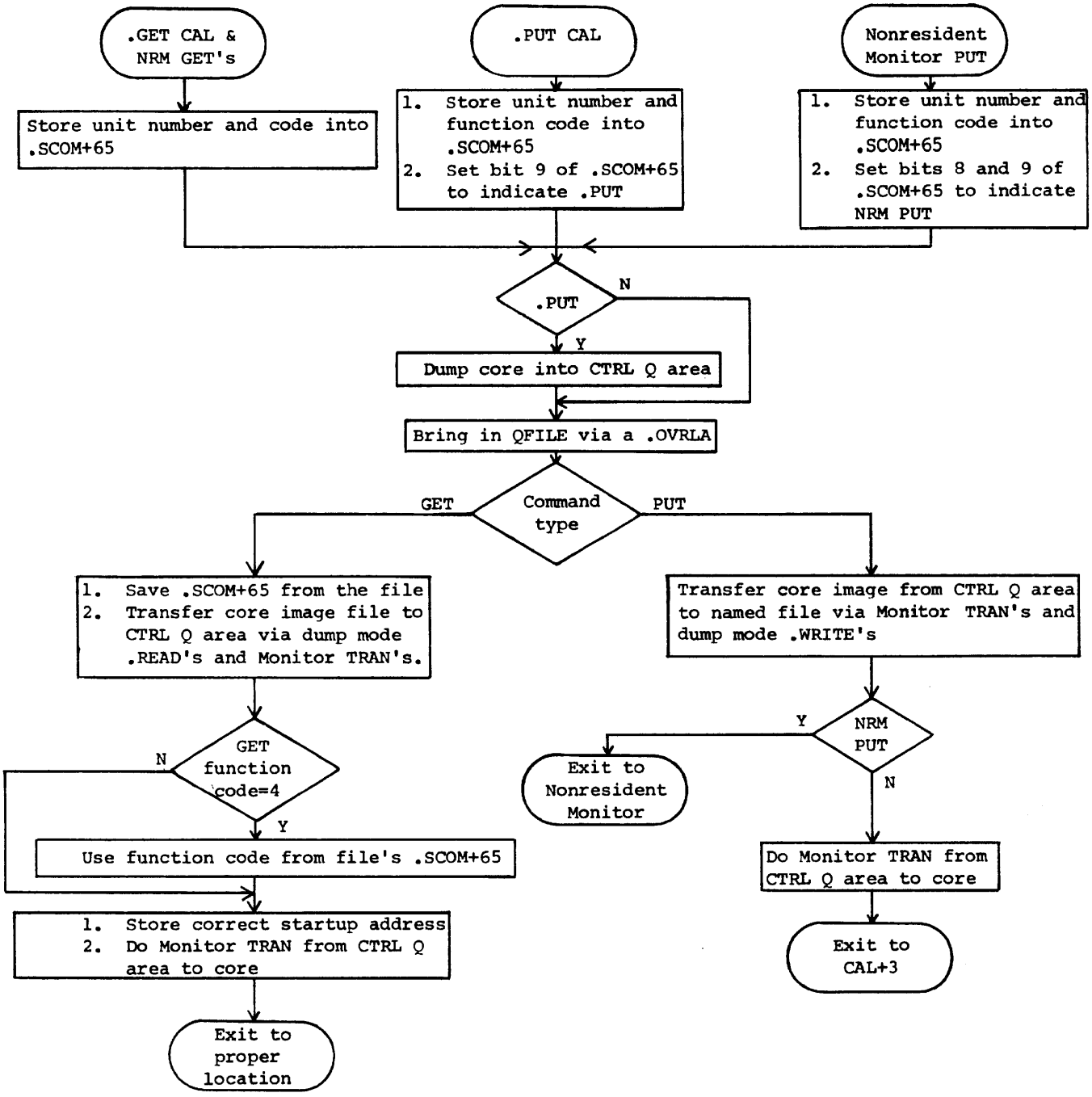
```
UNITNO=4ØØØØØ           /UNIT FOUR
.SCOM=1ØØ
.
.
LAC     START           /STARTING ADDRESS AFTER THE CTRL Q
                        /GET
XOR     UNITNO          /UNIT NUMBER IN HIGH-ORDER THREE BITS
JMP*    (.SCOM+64       /ADDRESS OF CTRL Q GET ROUTINE
```

Figure 3-2, QFILE, and Implementation of GET and PUT Logic, shows the information flow associated with QFILE. QFILE uses the following registers:

.SCOM+7,1Ø & 11	.SIXBT Filename and Extension
.SCOM+65	Command parameters, packed as follows: Bits Ø-2 Device unit number Bit 8 NRM PUT, when set Bit 9 PUT logic, when set Bits 15-17 Function Code
.SCOM+66-71	CTRL Q Area parameters
.DAT-14	File must be on the device assigned to this .DAT slot.

NOTE

All GET and .GET operations change all of core, except registers Ø through 4 of bank zero.



Note: This chart assumes error free input.

QFILE, and Implementation of GET and PUT Logic

Figure 3-2

CHAPTER 4

THE SYSTEM LOADER AND THE LINKING LOADER

The System Loader is the third major part of the DOS-15 Monitor. The other two are the Resident and Nonresident parts. The Resident and Nonresident Monitors communicate with the System Loader by manipulating certain .SCOM registers. When commands to either part imply a new configuration is needed, that part sets up the appropriate .SCOM registers, and passes control to the System Bootstrap via the Monitor TRAN routine. The Bootstrap then loads the System Loader into high core, and gives it control.

The System Loader examines the .SCOM registers, and loads a fresh copy of the Resident Monitor, including any features that the user wishes to be resident, such as the CTRL X feature. It will also load the desired system program and all handlers required by the new configuration. In addition, it will allocate all required buffers. The Nonresident Monitor is treated like any other core-image system program.

The System Loader never loads user programs. It only loads core-image system programs, the Linking Loader and Execute. The latter two load user programs.

The System Loader uses two device handlers to interface with the disk: the System Bootstrap, and the System Loader Disk Handler (DKL.). DKL. arrives in core along with SYSBLK, COMBLK and SGNBLK, as well as the loader itself. The Bootstrap loads core image programs only. The DKL. takes care of relocatable programs and any handlers loaded by the System Loader. Those include all handlers for core-image system programs, the Linking Loader's own handlers, and any needed by the Execute file. The Linking Loader loads some handlers needed by user programs it links.

There are two parts to the System Loader: the System Loader Interface, and the System Loader proper (.SYSLD). Figure 4-1 describes the System Loader Interface. Figure 4-2 describes the System Loader Proper, and Figure 4-3 describes the Linking Loader.

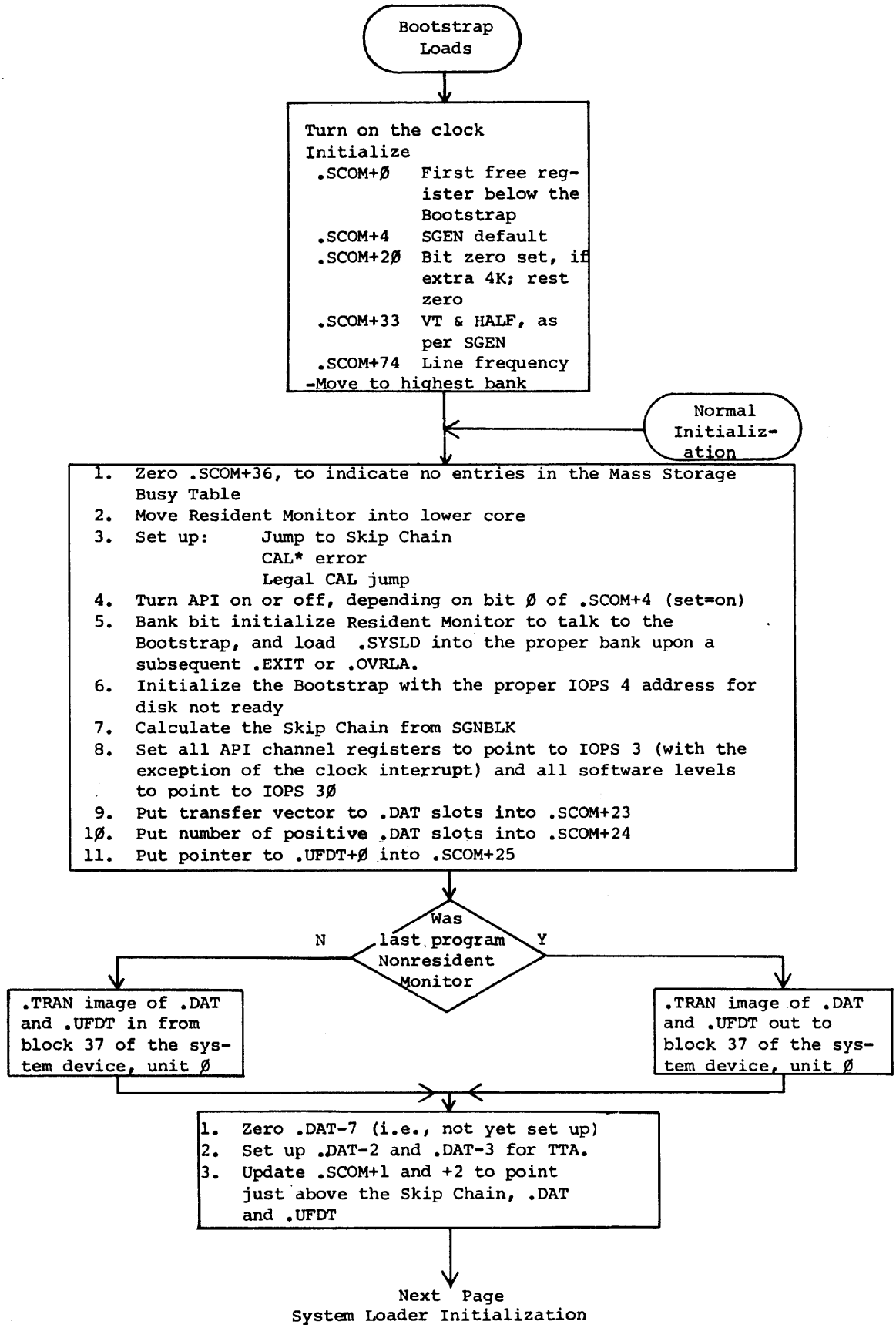

Bootstrap Loads
Turn on the clock
Initialize
.SCOM+Ø First free register below the Bootstrap
.SCOM+4 SGEN default
.SCOM+2Ø Bit zero set, if extra 4K; rest zero
.SCOM+33 VT & HALF, as per SGEN
.SCOM+74 Line frequency
-Move to highest bank
Normal Initialization
1. Zero .SCOM+36, to indicate no entries in the Mass Storage Busy Table
2. Move Resident Monitor into lower core
3. Set up: Jump to Skip Chain
CAL* error
Legal CAL jump
4. Turn API on or off, depending on bit Ø of .SCOM+4 (set=on)
5. Bank bit initialize Resident Monitor to talk to the Bootstrap, and load .SYSLD into the proper bank upon a subsequent .EXIT or .OVRLA.
6. Initialize the Bootstrap with the proper IOPS 4 address for disk not ready
7. Calculate the Skip Chain from SGNBLK
8. Set all API channel registers to point to IOPS 3 (with the exception of the clock interrupt) and all software levels to point to IOPS 3Ø
9. Put transfer vector to .DAT slots into .SCOM+23
1Ø. Put number of positive .DAT slots into .SCOM+24
11. Put pointer to .UFDT+Ø into .SCOM+25
Was last program Nonresident Monitor
N
Y
.TRAN image of .DAT and .UFDT in from block 37 of the system device, unit Ø
.TRAN image of .DAT and .UFDT out to block 37 of the system device, unit Ø
1. Zero .DAT-7 (i.e., not yet set up)
2. Set up .DAT-2 and .DAT-3 for TTA.
3. Update .SCOM+1 and +2 to point just above the Skip Chain, .DAT and .UFDT
Next Page
System Loader Initialization

Figure 4-1

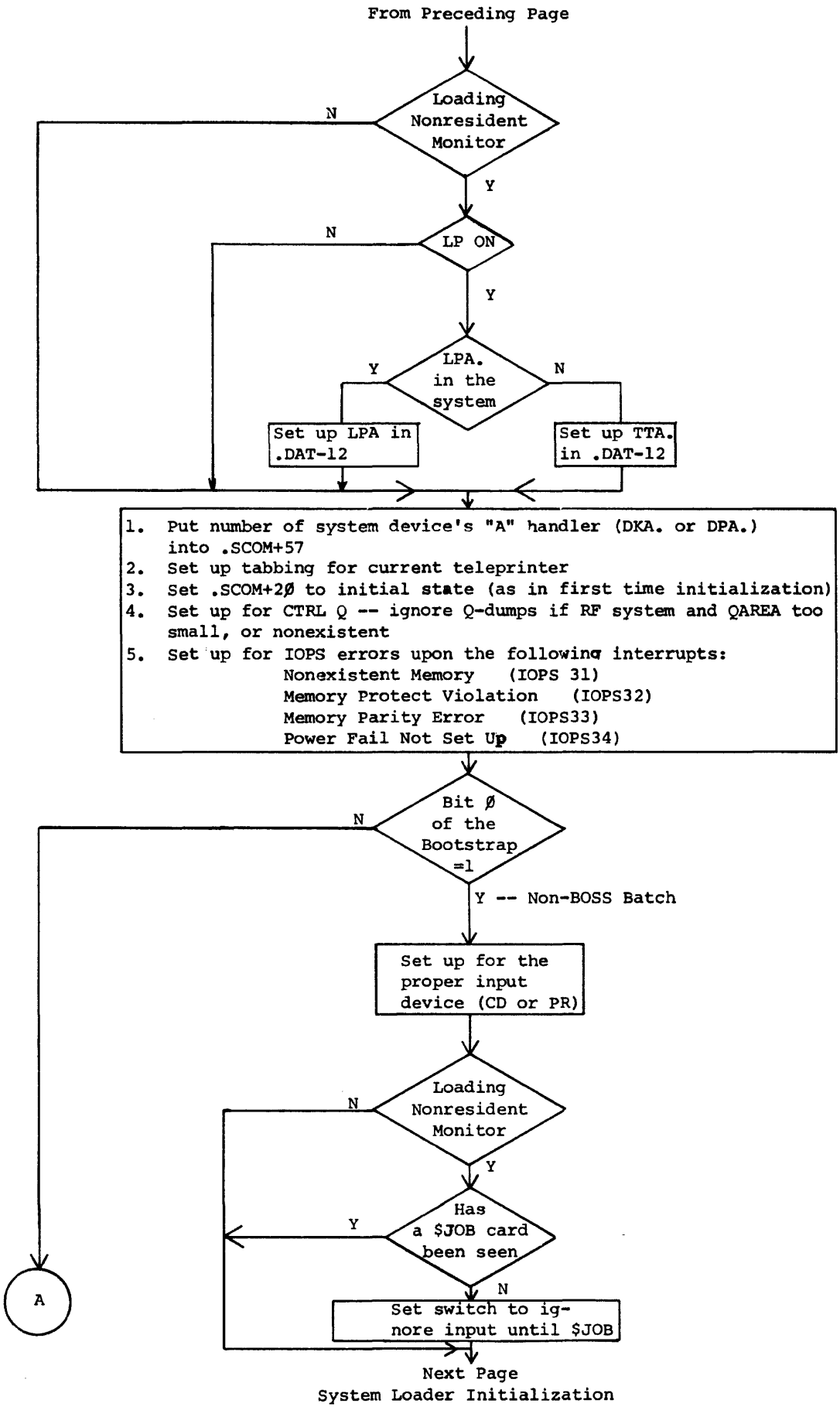


Figure 4-1 (Cont.)

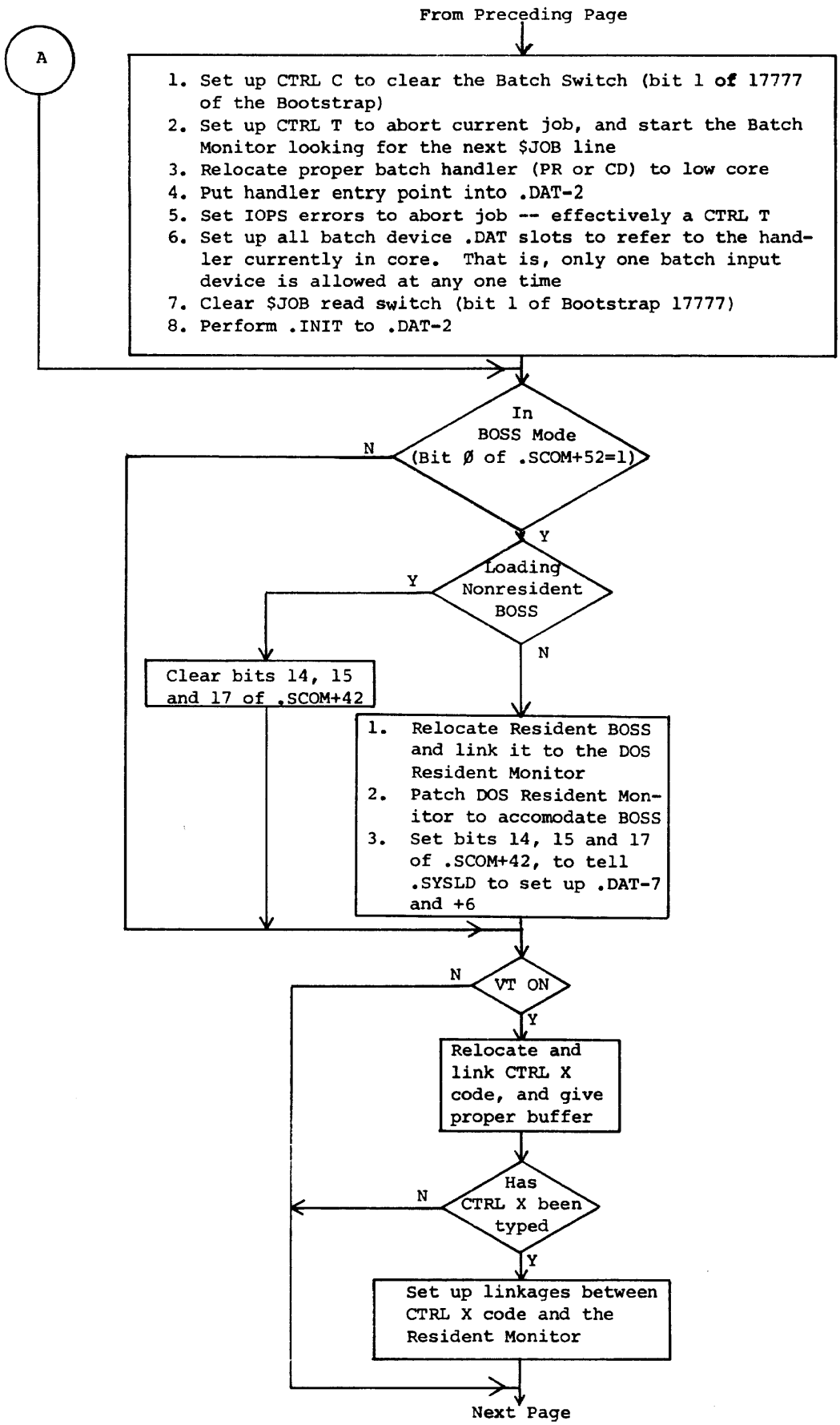


Figure 4-1 (Cont.)

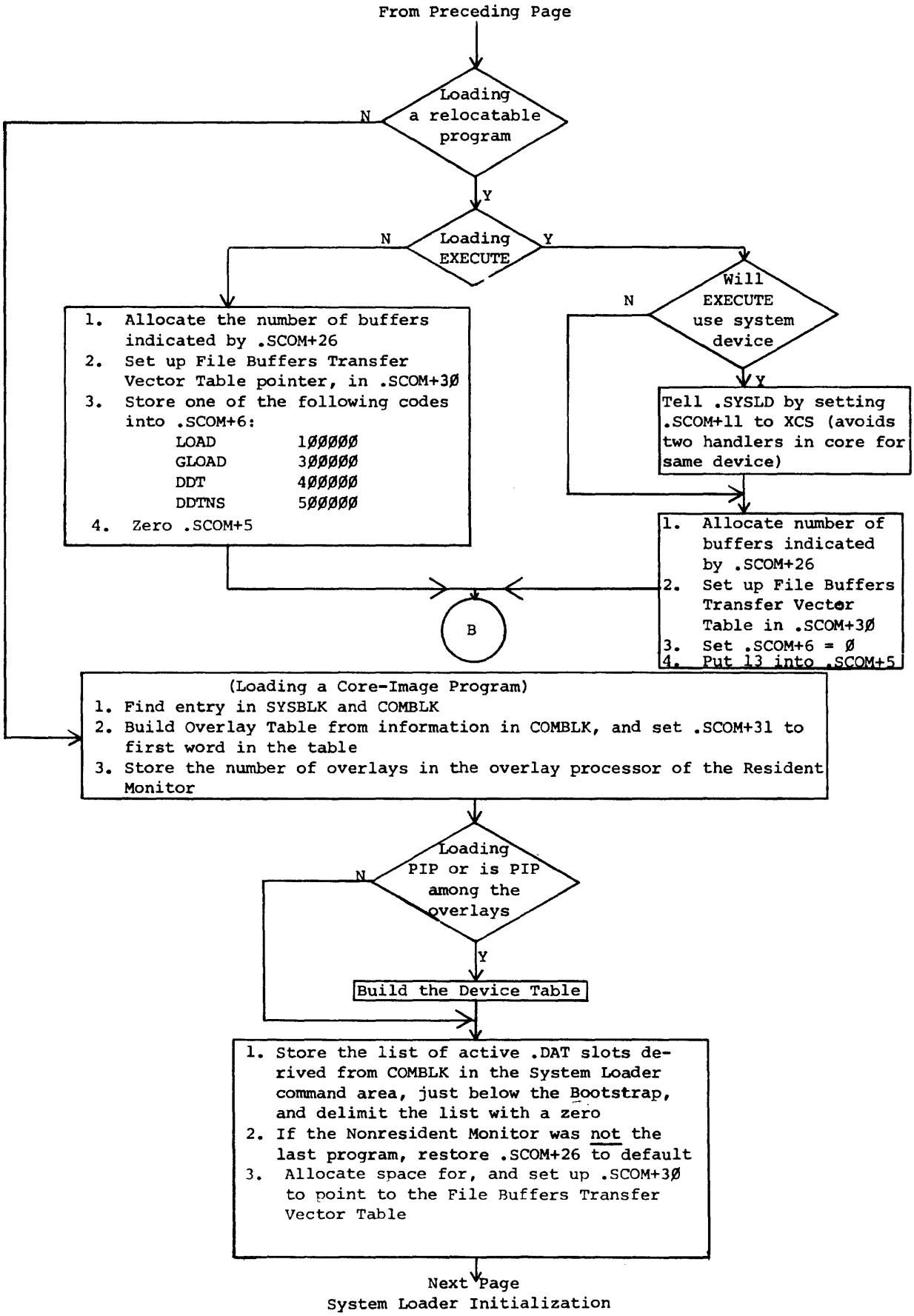
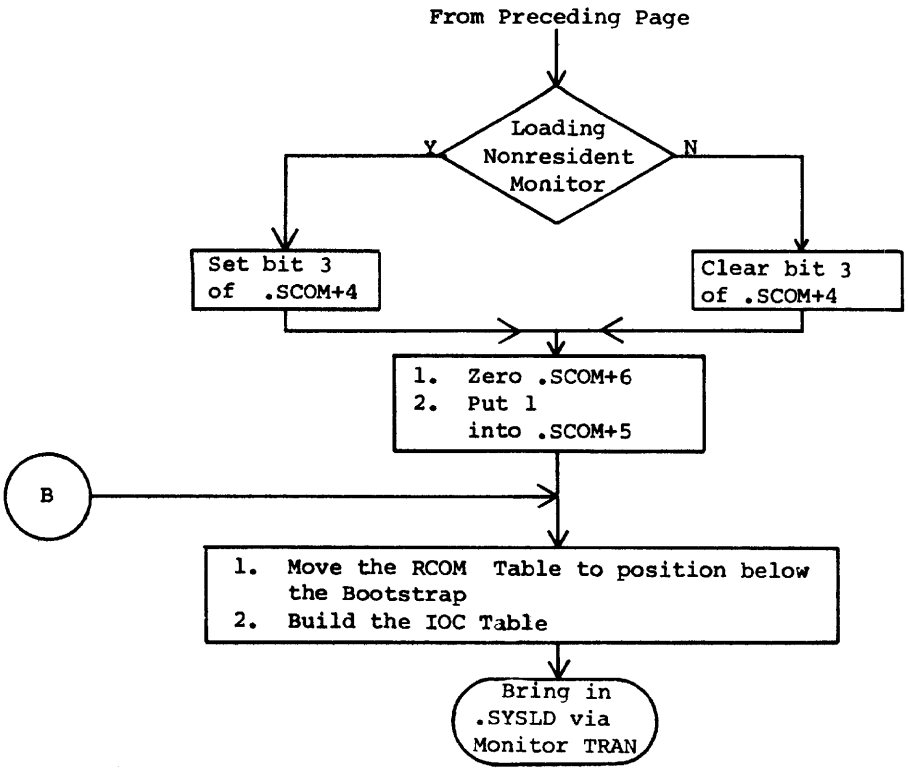


Figure 4-1 (Cont.)



System Loader Initialization

Figure 4-1 (Cont.)

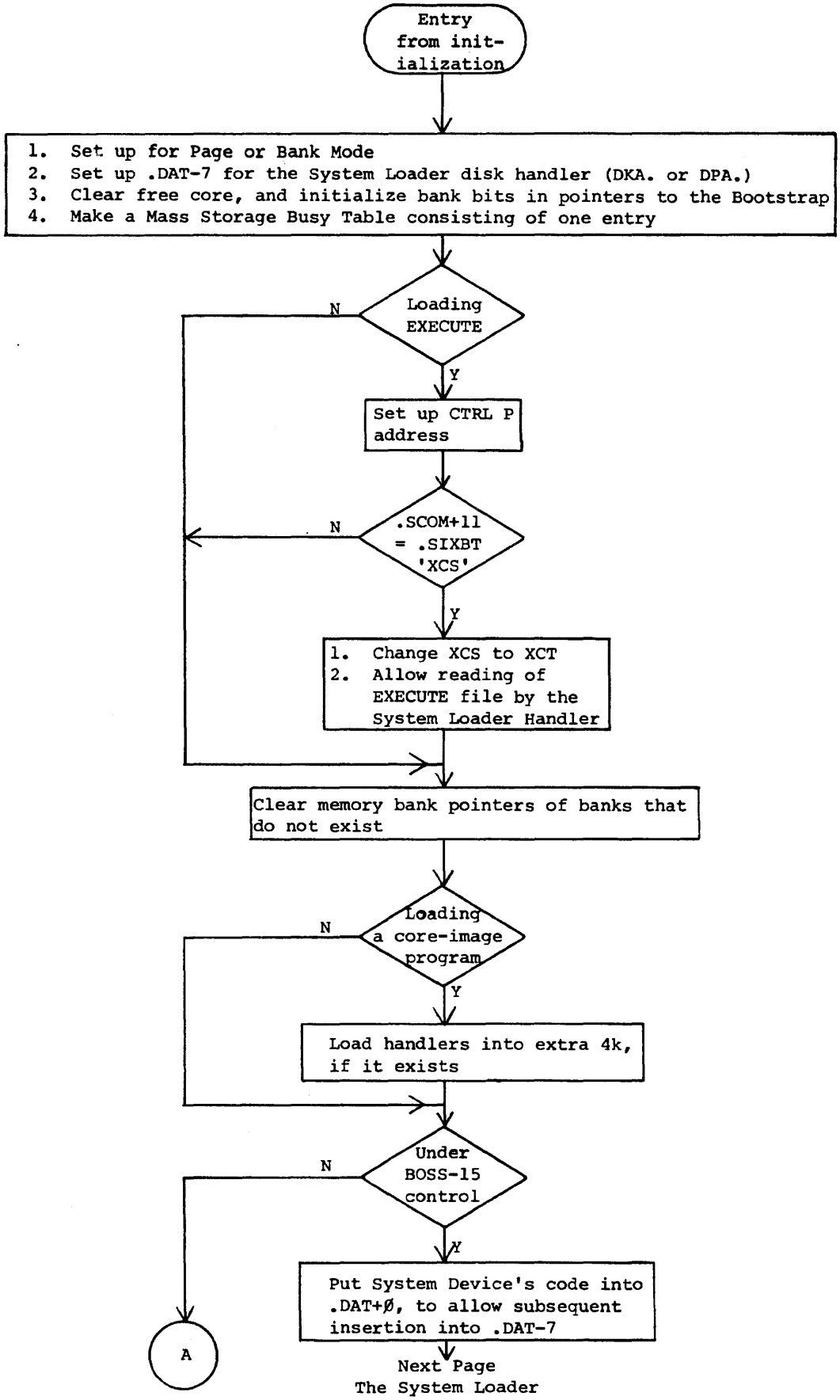
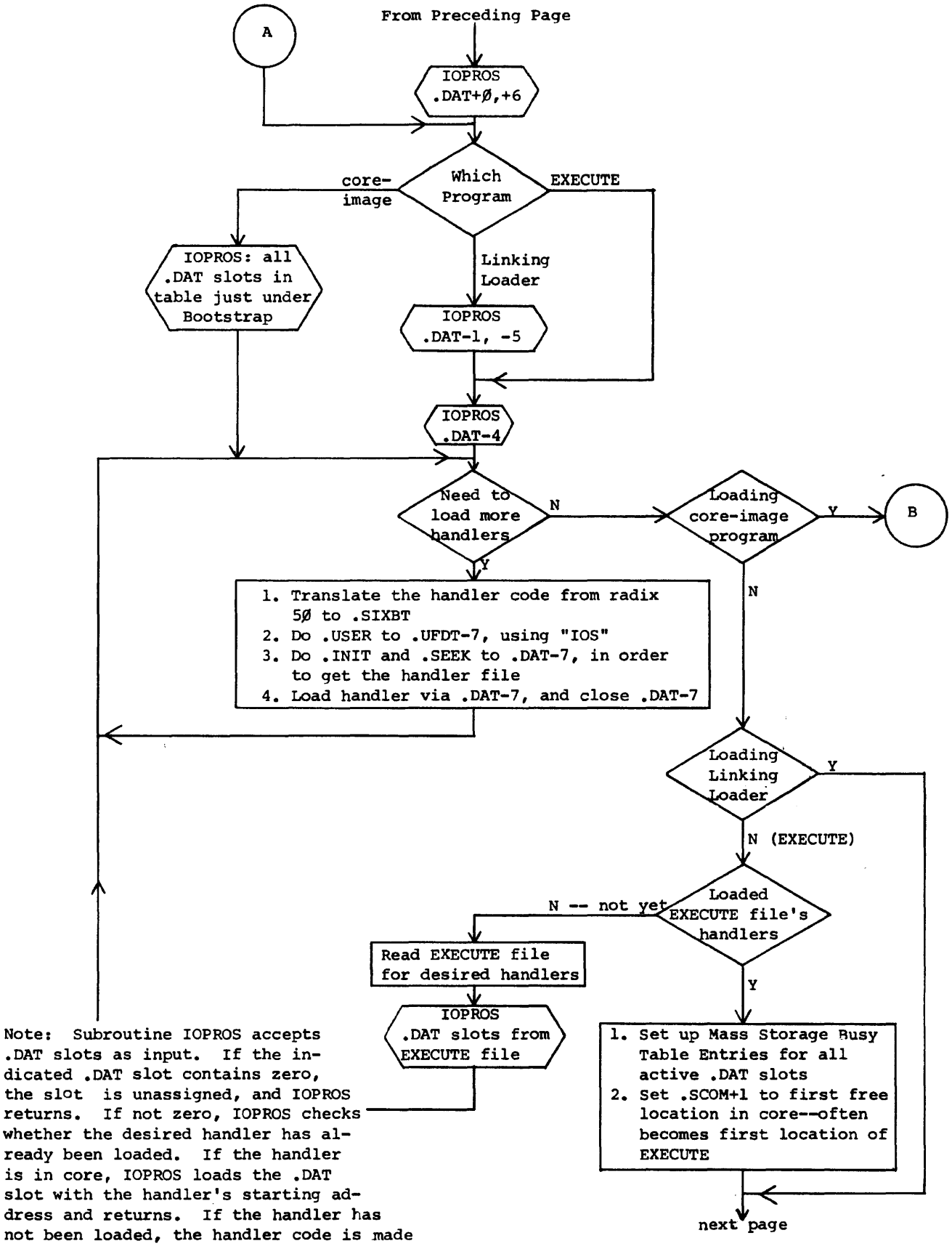

Entry from init-ialization
1. Set up for Page or Bank Mode
2. Set up .DAT-7 for the System Loader disk handler (DKA. or DPA.)
3. Clear free core, and initialize bank bits in pointers to the Bootstrap
4. Make a Mass Storage Busy Table consisting of one entry
Loading EXECUTE
N
Y
Set up CTRL P address
.SCOM+11 = .SIXBT 'XCS'
N
Y
1. Change XCS to XCT
2. Allow reading of EXECUTE file by the System Loader Handler
Clear memory bank pointers of banks that do not exist
Loading a core-image program
N
Y
Load handlers into extra 4k, if it exists
Under BOSS-15 control
N
Y
Put System Device's code into .DAT+Ø, to allow subsequent insertion into .DAT-7
A
Next Page
The System Loader

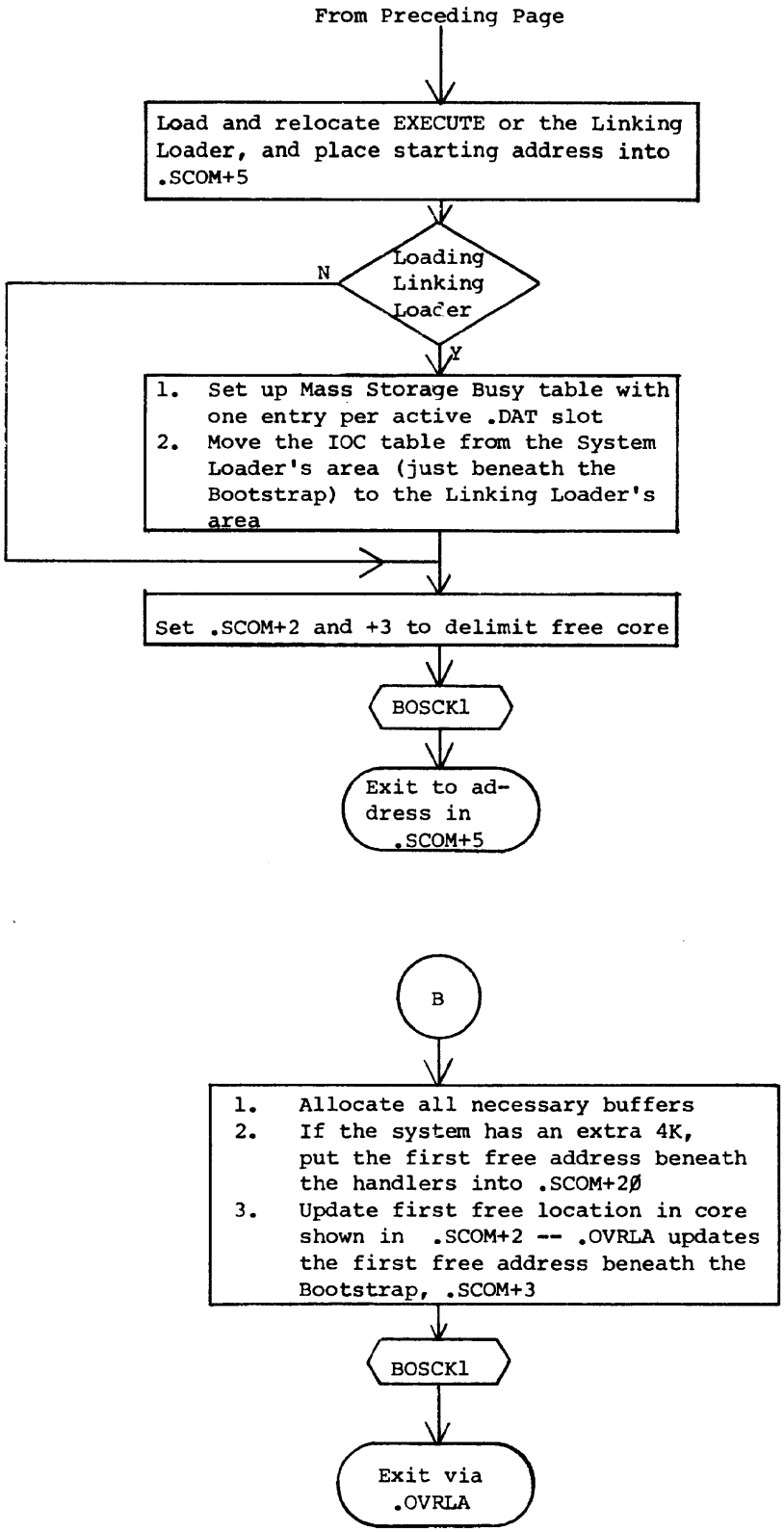
Figure 4-2



Note: Subroutine IOPROS accepts .DAT slots as input. If the indicated .DAT slot contains zero, the slot is unassigned, and IOPROS returns. If not zero, IOPROS checks whether the desired handler has already been loaded. If the handler is in core, IOPROS loads the .DAT slot with the handler's starting address and returns. If the handler has not been loaded, the handler code is made an unresolved .GLOBL, to be satisfied by the loop that follows immediately.

The System Loader

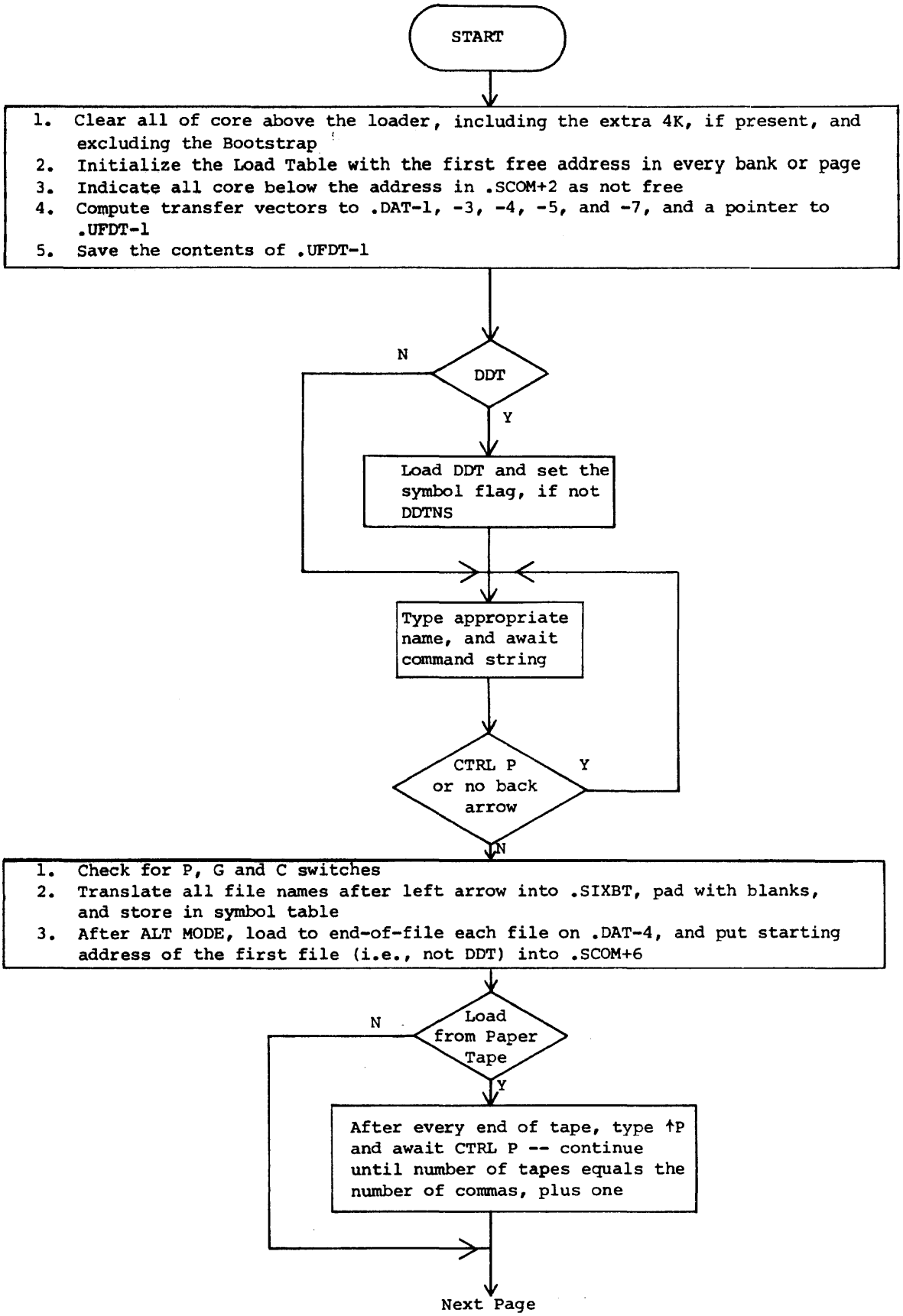
Figure 4-2 (Cont.)



Note: Subroutine BOSCK1 does the following, if loading a program under BOSS-15: (1) .USER to .UFDT-7, (2) .SEEK to .DAT-7 for PRCFIL PRC.

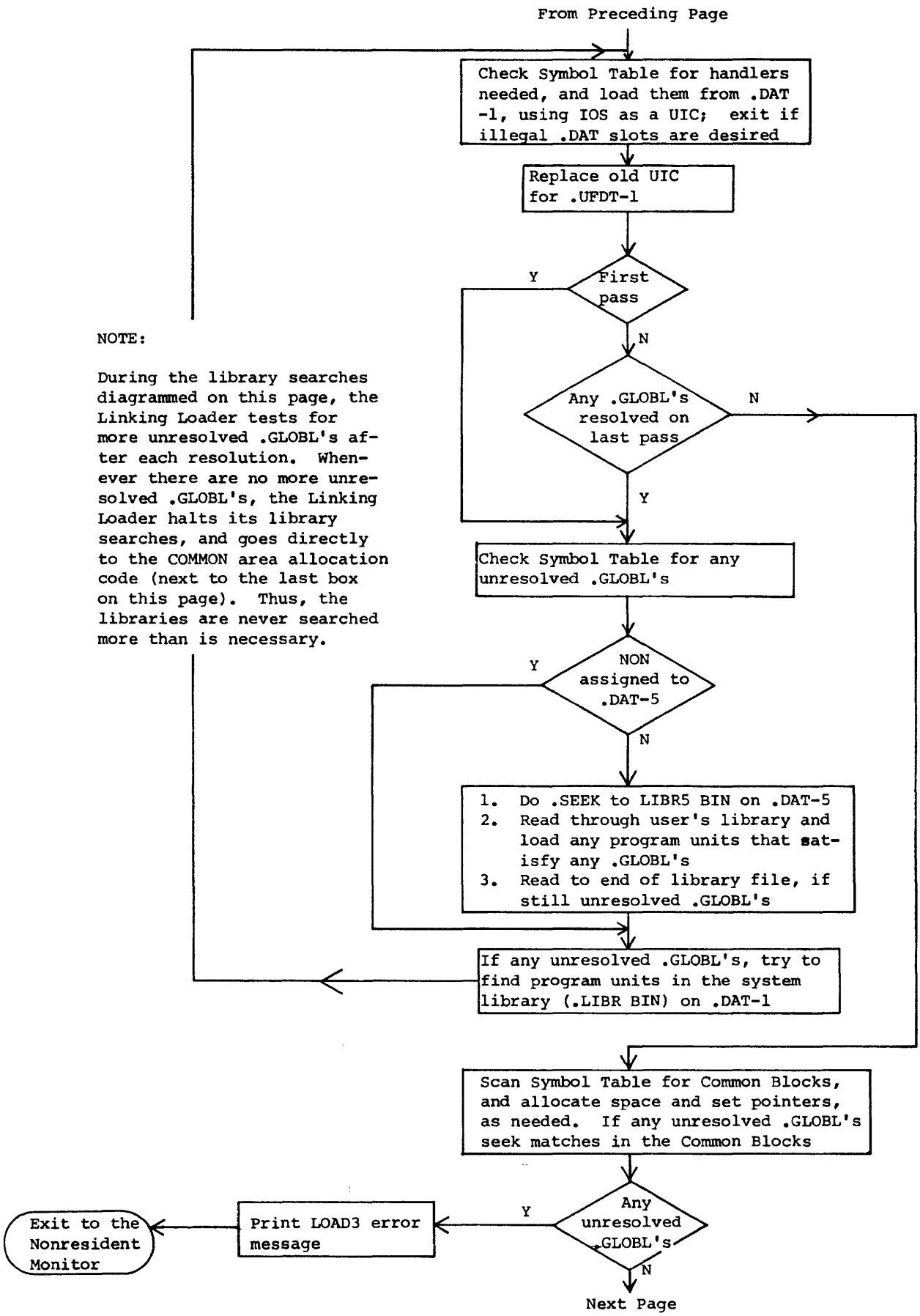
The System Loader

Figure 4-2 (Cont.)



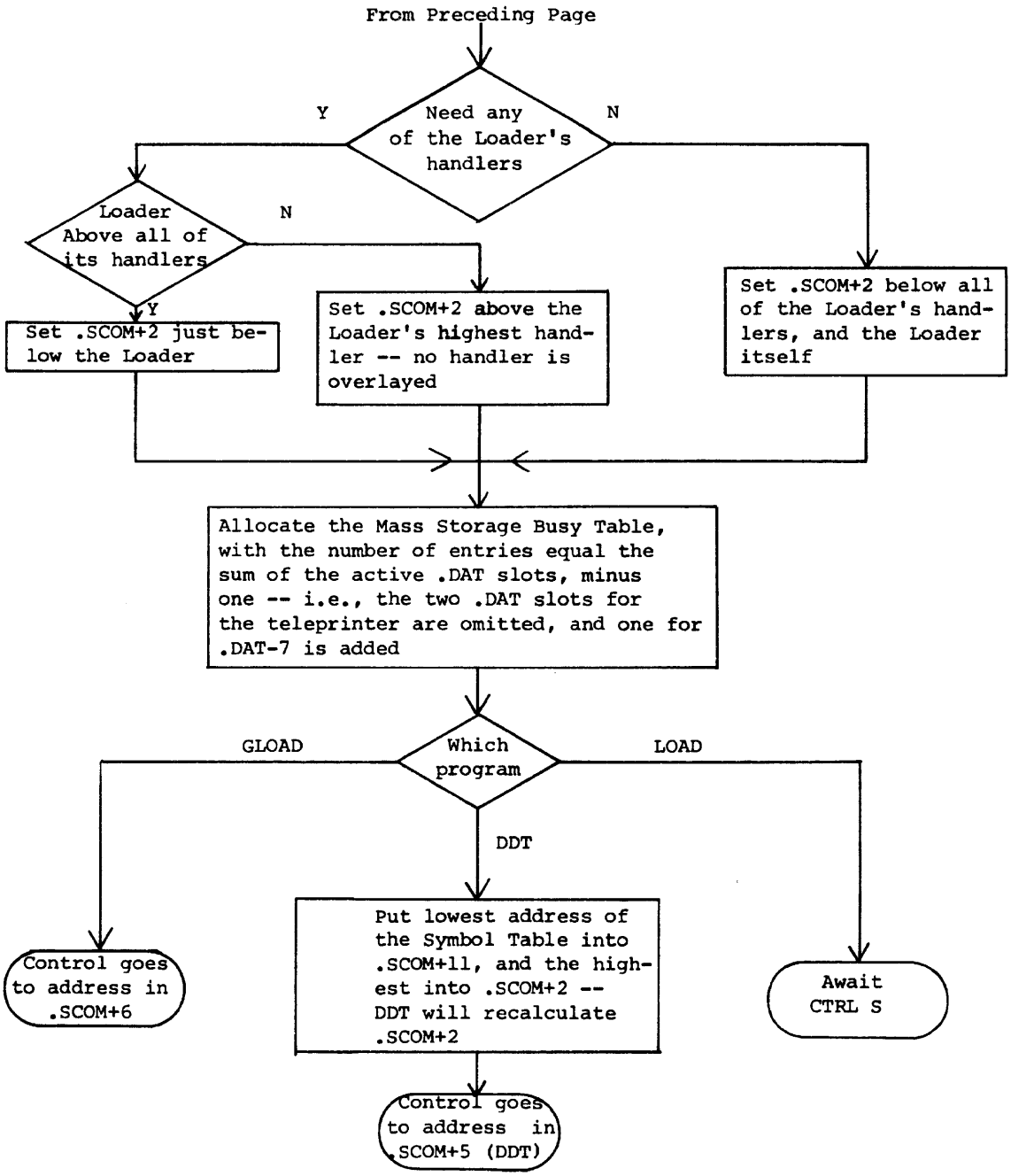
The Linking Loader

Figure 4-3



The Linking Loader

Figure 4-3 (Cont.)



The Linking Loader

Figure 4-3 (Cont.)

4.1 MANUAL BOOTSTRAP LOADS AND RESTARTS

Manual Bootstrap loads and restarts bring blocks Ø-36 of the system device into the lowest bank. These blocks contains the Resident Monitor, the System Loader Interface Routine, and SYSBLK, COMBLK and SGNBLK. Figure 4-4 illustrates the core load after manual Bootstrap loads and restarts. The Interface sets up .SCOM+Ø, 4, 2Ø, 27, 33, 54 and 74 from SGNBLK values determined at system generation time, and then transfers the whole core image of the Interface to the Bootstrap's bank. (DOS requires 16K, because this bank must be different from bank Ø.) At all other times, the Bootstrap loads the System Loader into its own bank. This preserves the image of .SCOM, part two of the Resident Monitor patch area, and the CTRL X buffer.

4.2 LOADING SYSTEM PROGRAMS

The System Loader Interface Routine gets control in the highest bank, either by a transfer from the lowest bank, or by load from the Bootstrap. After setting up for the System Loader Proper (.SYSLD), according to the program to be loaded and the settings of certain SCOM registers, the Interface Routine brings it in as a complete overlay. Figure 4-5 illustrates the core configuration of the Interface when it is in the highest bank. (The addresses provided are for a 16K system.) The System Loader loads handlers from the lowest part of free core up, with the exception that the extra 4K is filled first, if it exists. Core image system programs are usually loaded just beneath the Bootstrap (always in the highest bank). Such core images must be wholely within the top bank of core, and above register 17 of that bank. Figure 4-6 illustrates the core maps for system programs.

Whenever the Linking Loader is loaded (LOAD, GLOAD, DDT, and DDTNS), the System Loader loads all handlers for .DAT slots -1, -4, and -5, and then loads the Linking Loader itself. (DDT is loaded by the Linking Loader.) Figure 4-7 illustrates the core maps for the Linking Loader.

For EXECUTE, the System Loader loads EXECUTE's handler, and reads the EXECUTE file, in order to determine the active .DAT slots. The System Loader then loads all the handlers required, and sets up the .DAT slots. Figure 4-8 illustrates core maps for EXECUTE.

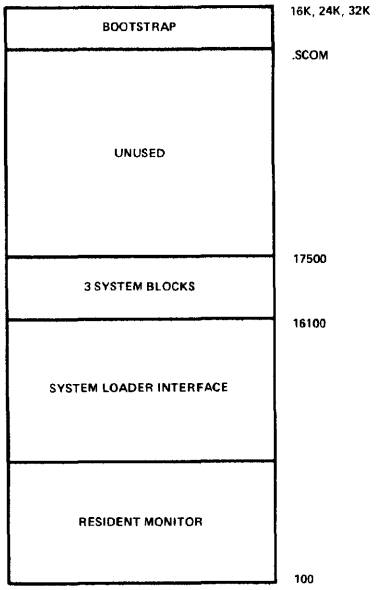


Figure 4-4
Bootstrap Load

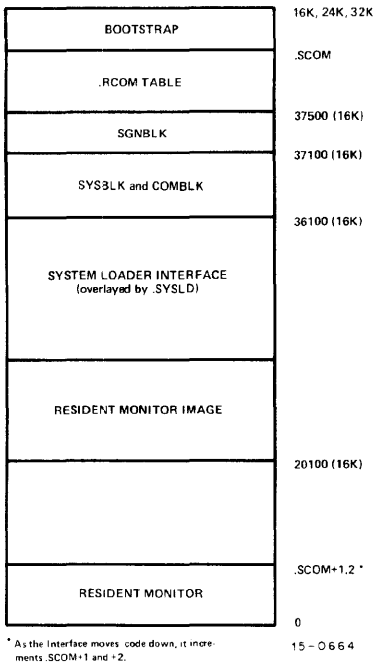


Figure 4-5
Standard
Interface Load

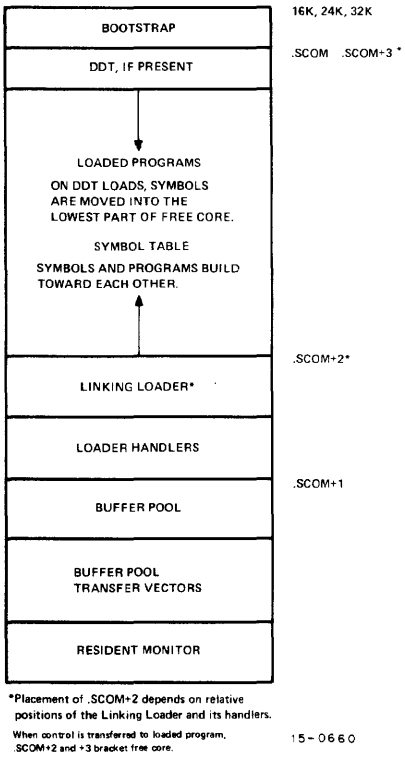


Figure 4-7
Linking Loader

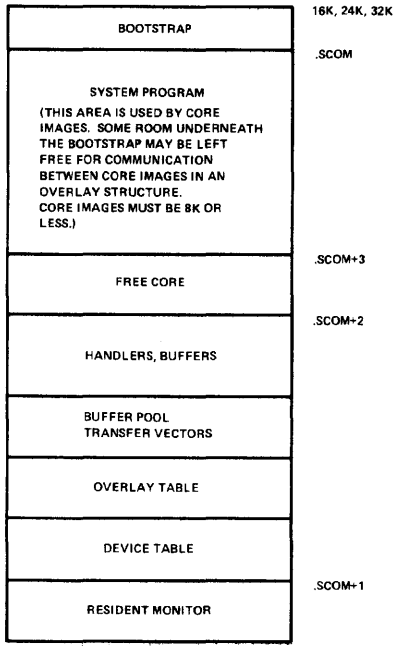


Figure 4-6
System
Program Load

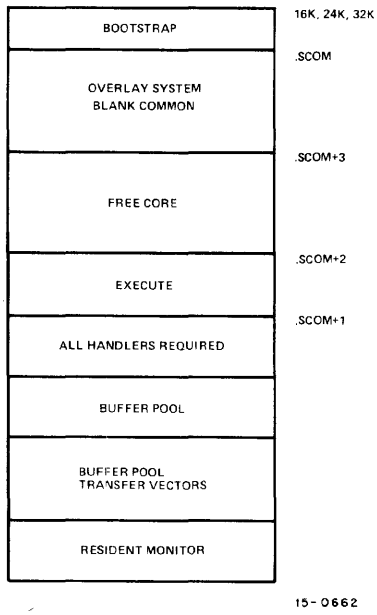


Figure 4-8
Execute

BOSS-15 Mode operation requires the system "A" handler be assigned to .DAT-7. This requires a sleight of hand on the part of the System Loader, which needs the "L" handler on .DAT-7. It therefore loads the "A" handler as if it were assigned to .DAT+Ø, and transfers the set up .DAT slot Ø contents to .DAT-7 before transferring control to the program being loaded. .DAT+Ø is then restored to its original status.

4.4 TABLES AND INFORMATION BLOCKS USED AND BUILT BY LOADERS

The System Loader uses SYSBLK, COMBLK, SGNBLK, block 37 of the system device, .SCOM, the RCOM Table, the IOC Table, the Device Table, the Mass Storage Busy Table, the File Buffers Transfer Vector Table, the Overlay Table, .DAT, .UFDT and three bits in the Bootstrap. Tables 4-I, 4-II and 4-III describe how the Loaders use these blocks and tables.

4.5 .DAT SLOT MANIPULATION BY THE SYSTEM LOADER

The System Loader maintains the .DAT slot device handler assignments as they were the last time the Nonresident Monitor was in core. The Loader saves the .DAT and .UFDT on the system device whenever the Nonresident Monitor was the last program in core. Thereafter, the Loader refreshes .DAT and .UFDT from the image on the disk. If KEEP is off, the Nonresident Monitor's initialization routine restores the .DAT and .UFDT to default values.

When loading core-image system programs, the System Loader determines the active .DAT slots by examining COMBLK. When loading EXECUT, the System Loader sets up .DAT-4, and any active slots indicated by the Execute file itself. When loading the Linking Loader, the System Loader sets up .DAT-1, -4, and -5. The Linking Loader will set up other other active .DAT slots according to the .IODEV commands in the assembly of the program units being loaded.

Both the System Loader and the Linking Loader set up .DAT slots in this manner: (In the following procedure, "loader" refers to either one.)

Table 4-I

Tables and Blocks
Used by the Loaders

NAME	USE	LOCATION
SYSBLK	The System Loader obtains Monitor TRAN parameters from SYSBLK when it builds	165ØØ of .SYSLD's bank
COMBLK	Indicates number of buffers required, the active .DAT slots, and the names	171ØØ down, in .SYSLD's bank
SGNBLK	Default settings for .SCOM registers, number of words per buffer, size of Resident Monitor's patch area (part two), Skip Chain, .DAT and .UFDT default contents, and handler information.	161ØØ of .SYSLD's bank
Block 37 of the System Device	Image of .DAT and .UFDT, when last program was loaded (excluding the Nonresident Monitor).	
.SCOM Table	See Table 4-II.	1ØØ of 1st bank
RCOM Table	Moved for use by the Nonresident Monitor.	175ØØ of the highest bank
IOC Table	Built by Interface Routine for .SYSLD itself.	Just beneath the System Loader
Device Table	Built by Interface Routine if loading PIP, or if PIP is among the overlays listed in COMBLK	Just above .SCOM+1
Mass Storage Busy Table	Built by the System Loader itself.	Pointed to by .SCOM+62
File Buffers Transfer Vector Table	Allocated by the Interface Routine, and initialized by it for non-core Image programs. System Loader proper initializes for core-image programs.	Pointed to by .SCOM+3Ø
Overlay Table	Built by the Interface Routine	Pointed to by .SCOM+31
.DAT and .UFDT	Image stored and restored from block 37 of the System Device. The System Loader loads all handlers for core-image programs and EXECUTE Files, and sets up the appropriate .DAT slots. The System Loader also loads handlers assigned to .DAT-1, -4, and -5 when loading the Linking Loader, and .DAT-7 and +6 for BOSS-15.	Pointed to by .SCOM+23 and .SCOM+25
BOOTSTRAP	Bits Ø, 1, and 2 of location 17777 in the Bootstrap's bank used for Batch (non-BOSS) information.	

Table 4-II

.SCOM REGISTERS USED BY THE SYSTEM LOADER

.SCOM+	Description of Use by the System Loader
Ø	Set in first-time initialization routine. Used to locate the System Loader Command Area, which is just below the Bootstrap.
1	System Loader Interface routine updates this indication of the first free register above the Resident Monitor each time it moves a piece down to low core.
2	The Interface and .SYSLD itself continually update this indication of the first free location as they move code and build tables.
3	Updated as with .SCOM+2. Last free location in core.
4	First Time Initialization routine sets this register according to a SGNBLK parameter. Refer to Table 4-III.
5	Interface Routine stores code of program to be loaded into .SCOM+5. .SYSLD uses .SCOM+5 for starting address when loading EXECUT or LOAD. The .OVRLA routine loads .SCOM+5 with starting address of the Monitor Recovery Routine. The Bootstrap transfers to the address in .SCOM+5 after all its operations.
6	Interface Routine stores codes for DDT, DDTNS, LOAD and GLOAD into .SCOM+6. For other programs, the Interface Routine zeroes .SCOM+6.
7	.SYSLD saves contents of .DAT-1 in .SCOM+7, when loading the Linking Loader. When loading EXECUT, .SCOM+7 contains the first three characters of the Execute file's name. Contains .DAT-12 when loading Nonresident Monitor.
1Ø	.SYSLD saves contents of .DAT-4 in SCOM+1Ø, when loading the Linking Loader. When loading EXECUT, .SCOM+1Ø contains the second three characters of the Execute file's name.
11	.SYSLD saves contents of .DAT-5 in .SCOM+11, when loading the Linking Loader. When loading EXECUT, .SCOM+11 contains the extension of the Execute file's name. (The Interface routine sets .SCOM+11 to XCS, telling .SYSLD that EXECUT will be using the system device. .SYSLD then restores .SCOM+11 to XCT.)
12-15	The Interface routine initializes these transfer vectors for API software levels to point to SERR, an error routine that will produce an IOPS3Ø.
16, 17	Unaffected.

Table 4-II (Cont'd)

.SCOM+	Description of Use by the System Loader
2Ø	Bit zero set in first time initialization, if system contains an extra 4K. If the system does contain an extra 4K, the System Loader will load handlers in that page -- from the bottom up -- when loading a core-image program. Whenever there is an extra 4K, the System Loader will update bits 3-17 with the address of the first free cell in the extra 4K.
21	Unaffected.
22	Unaffected.
23	The Interface Routine refreshes this pointer to .DAT.
24	The Interface Routine refreshes this indication of the number of positive .DAT slots.
25	The Interface Routine refreshes this pointer to .UFDT+Ø.
26	When the Nonresident Monitor was the last program, the System Loader allocates the number of buffers indicated by the contents of .SCOM+26. If the Nonresident Monitor was not the last program, the System Loader restores .SCOM+26 to the default value if program to be loaded is core image. Otherwise, untouched.
27	The first time initialization routine sets this indication of the number of words per file buffer.
3Ø	The Initialization Routine loads this pointer to the File Buffer Transfer Vector Table.
31	When loading a core-image program, the Interface Routine loads .SCOM+31 with the pointer to the Overlay Table, or with zero, if there is none.
32	Unaffected.
33	See Interface Routine table, to determine how that routine reacts to the bits in .SCOM+33.
34, 35	Unaffected.
36	System Loader loads with the number of active .DAT slots assigned to the system device.
37-42	Unaffected.
43, 44	Contains name of the program to be loaded.
45-56	Unaffected.
57	System Loader loads with the number of entries in the Mass Storage Busy Table.
6Ø, 61	Unaffected.
62	System Loader loads with the address of the first entry in the Mass Storage Busy Table.
63-	Unaffected.

Table 4-III

Use of .SCOM+4 by the System Loader

Bit	
0	If set, place "API ON" constant into ØØØØØ6. If clear, place "API OFF" constant in same register.
1	Ignored.
2	If set, change the Resident Monitor so it will tab with the KSR 35/37 tabbing mechanism.
3	Loader will set this bit, if loading the Nonresident Monitor; clear it otherwise.
4-6	Ignored.
7	Loader sets this bit if bit 11 is cleared, and loading the Linking Loader or Execute. Otherwise clear.
8	Sets or clears, after comparing current core size (known by location of Bootstrap, and status of bit Ø, .SCOM+2Ø) with SGNBLK parameter. Also, modifies Resident Monitor to give IOPS77 after attempts to use CTRL Q.
9, 10	Ignored
11	Indicates whether to clear or set bit 7, when loading Linking Loader or Execute.
12-17	Ignored

1. Each .DAT slot will contain a handler number -- either the system default, or one inserted via an ASSIGN command to the Nonresident Monitor. This handler number is the relative location of the handler name in the IOC Table, which the Interface Routine builds. (The IOC Table contains handler names in Radix 5Ø.)

2. For each active .DAT slot, the loader uses the handler number in that slot to find the name in the IOC table, and converts the name to .SIXBT.

3. If the handler is already in core, the loader simply inserts the starting address of the handler into the .DAT slot.

4. If the handler is not yet in core, the loader does a .SEEK to IOS for the handler, reads it into core, relocates it, and places the starting address of the handler into the .DAT slot.

The System Loader always sets up .DAT-2 and -3. (It reserves .DAT-7 for its own use.) When not in non-BOSS Batch Mode, -2 is assigned to TTA. In non-BOSS Batch Mode, the batch input device goes to -2. If loading the Nonresident Monitor and bit three of .SCOM+42 is set, the System Loader will set up .DAT-12 for the LPA, if it is in the system, or else for TTA. If in BOSS mode, the Nonresident Monitor assigns LPA. to .DAT+6, and the System Loader assigns .DAT-7 to the system device "A" handler. The System Loader then ensures that both handlers are in core. The Resident BOSS set up routine subsequently routes all .DAT slots connected to TTA. to Resident BOSS.

4.6 BUFFER ALLOCATION BY THE SYSTEM LOADER

The System Loader allocates space for buffers equal to the contents of .SCOM+26 times the contents of .SCOM+27. The first time initialization routine sets .SCOM+27 to the standard number of locations per buffer. Before the Nonresident Monitor does an .OVRLA to a software system program, it checks whether a BUFFS command has been issued. If so, it leaves .SCOM+26 as is. If not, it uses the default number of buffers for that program, as shown in SYSBLK.

CHAPTER 5

SYSTEM INFORMATION BLOCKS AND TABLES

5.1 CORE-RESIDENT NON-REFRESHED REGISTERS

The .SCOM table, the Bootstrap and the resident Patch Area are the only registers not refreshed by the System Loader. Table 5-I describes the .SCOM Table.

5.2 DISK-RESIDENT UNCHANGING BLOCKS: SYSBLK, COMBLK AND SGNBLK

SYSBLK, COMBLK and SGNBLK occupy blocks 34, 35, and 36 (octal) on the system device (unit zero). SYSBLK and COMBLK (blocks 34 and 35) contain the parameters for loading all core image system programs. SGNBLK contains all the other information needed to run DOS. All three arrive in core along with the Resident Monitor and the System Loader Interface, and start at location 161ØØ of the highest bank. The Nonresident Monitor and System Loader use them, and DOSGEN and PATCH modify them, when necessary.

5.2.1 SYSBLK

SYSBLK contains the parameters required for implementation of .OVRLA to any system program, or any of the system program overlays.

The order of entries in SYSBLK is unimportant, except for the first three permanent entries: RESMON, .SYSLD, and ↑QAREA. The first word of SYSBLK contains the block address (the unrelocated address) of the first free word after itself. Figure 5-1 describes SYSBLK.

5.2.2 COMBLK

COMBLK contains information the System Loader and the Nonresident Monitor need to remember about the current core-image system programs. The last location in COMBLK (that is, location 377 of block 35) contains the block address of the first entry in COMBLK. The remainder of COMBLK consists of variable-length entries associated with the system programs. The Nonresident Monitor searches COMBLK when it finds no match for a typed command in its own Command Table. Figure 5-1 illustrates the organization of COMBLK. The System Generator adds

TABLE 5-I

.SCOM Registers

REGISTER	BIT	MEANING
Ø		First register below the Bootstrap (set by the System Loader Interface)
1		First register above the Resident Monitor (set by the System Loader Interface)
2		Lowest free register available for storage (set by the System Loader or the Linking Loader)
3		Highest free register available for storage (set by the System Loader, the Linking Loader or DDT)
4		Initialized from SGNBLK values by the "first time" section of the System Loader Interface Routine, and by the LOGIN, LOGOUT and MICLOG logic of the Nonresident Monitor; modified by the Nonresident Monitor, unless otherwise indicated.
	Ø = 1	API is available.
	1 = 1	EAE is available (always set)
	2 = 1	Teleprinter is Model 35 or 37
	3 = 1	Nonresident Monitor is in core
	4,5	Reserved
	6 = 1	9-Channel Magnetic Tape System
	7 = 1	Page Mode Operation
	8 = 1	QAREA inadequate for current core size (set by the System Loader Interface Routine)
	9 = 1	DOS disk file structure (always set)
	1Ø = 1	RBØ9 disk is system device.
	11 = 1	Bank Mode System
	12,13	Line Printer Line Size: ØØ No Line Printer Ø1 8Ø Characters 1Ø 12Ø Characters 11 132 Characters
	14 = 1	Background/Foreground System (always clear)
	15- 17	Drum size (ignored -- DOS does not support drum)

TABLE 5-I (Cont'd)

REGISTER	BIT	MEANING
5		Core Image System Program starting address.
6	Ø = 1	DDT in core.
	1 = 1	GLOAD
	2 = 1	DDTNS
	3-17	User program starting address.
7-11		When using the Linking Loader, .SCOM+7, 10 and 11 contain the handler numbers for handlers needed by the Linking Loader in .DAT -1, -4, and -5 respectively. When using EXECUTE, 7-11 contain the .SIXBT representation of the name and extension of the Execute File. When using QFILE (for implementation of .GET, .PUT and the Nonresident Monitor GET and PUT commands), 7-11 contain the .SIXBT representation of the name and extension of the core image file.
12		API Level 4 service routine entry point
13		API Level 5 "
14		API Level 6 "
15		API Level 7 "
16		Program Counter on Keyboard Interrupts.
17		AC on Keyboard Interrupts.
2Ø	Ø = 1	2ØK or 28K system.
	3-17	First free address in top page.
21		Magtape Status Register.
22		Reserved for Magtape Handler.
23		Pointer to .DAT+Ø.
24		Number of positive .DAT slots.
25		Pointer to .UFDT+Ø.
26		Number of buffers.
27		Number of words per buffer.
3Ø		Pointer to Buffer Transfer Vector Table.

TABLE 5-I (Cont'd)

REGISTERS	BIT	MEANING
31		Pointer to first entry in the Overlay Table (zero, if none).
32		Bad block number on IOPS 2Ø and 72.
33		CTRL X status register.
	Ø = 1	HALF ON
	1 = 1	Display Buffer already set up.
	2 = 1	VT ON
	17 = 1	If VT ON, display mode is on.
34		If in BOSS mode, elapsed time in seconds.
35		Instruction to clear TT Busy Switch.
36		Number of Entries in the Mass Storage Busy Table.
37		Entry point for Expanded Error Processor.
4Ø		JMP to Expanded Error Processor.
41		The logged-in UIC.
42		Bit Register.
	Ø = 1	MICLOG successful.
	1 = 1	.EXIT from Nonresident Monitor.
	2 = 1	.OVRLA from Nonresident Monitor.
	3 = 1	LP ON -- LPA to .DAT-12 when loading Nonresident Monitor.
	4 = 1	Dump core on calls to .MED (except IOPS 4).
	5 = 1	Halt on calls to .MED (except IOPS 4).
	6-13	Unused.
	14 = 1	Set up .DAT+6 (used by BOSS Mode).
	15 = 1	Load System Device Handler into .DAT-7.
	16 = 1	KEEP ON.
	17 = 1	BOSS Mode.
43,44		.SIXBT Representation of the name of the core image system program to be loaded (if any).
45,46		.SIXBT Representation of the name of the Non-resident Monitor

TABLE 5-I (Cont'd)

REGISTERS	BIT	MEANING
47		Date (MMDDYY)
5Ø		Time (HHMMSS)
51		Elasped time, in ticks.
52		BOSS Bit Register
	Ø = 1	BOSS15 Mode.
	1 = 1	Control Card Read by user, 5/7 ASCII image saved in first block of NRBOSS.
	2 = 1	Resident BOSS reached "EOF" on run time file (RTF).
	3 = 1	User exceeded time estimate.
	4 = 1	I/O CAL to go to TTY.
	5 = 1	Terminal IOPS error by user.
	6 = 1	QDUMP to be given to user on IOPS errors.
	7 = 1	Operator abort (Control T).
	8 = 1	Job active.
	9 = 1	Exit from BOSS15 Mode.
	1Ø = 1	User tried to do a .PUT. Core will be dumped and a listing given on LP.
	11 = 1	User tried to do a .GET.
	12	Not defined.
	13	Not defined.
	14-16	.SYSLD error number.
	17 = 1	Job abort.
53		Reserved for CTRL X code.
54		Default Protection Code.
55		Entry to Monitor TRAN routine.
56		Two's complement of time limit, in seconds (zero, if no limit).
57		System Device Code, for use by the Linking Loader.
6Ø		Number of ticks until clock interrupt specified in last .TIMER (zero, if .TIMER not in use).

TABLE 5-I (Cont'd)

REGISTER	BIT	MEANING
61		.TIMER address.
62		Address of the first word in the Mass Storage Busy Table.
63		Number of words per Mass Storage Busy Table Entry.
64		JMP to CTRL Q GET routine.
65		QFILE Communication Register.
66		First Block of the CTRL Q Area.
67		Starting Address minus one of the CTRL Q Area.
7Ø		Two's complement of number of word in Qdump
71		Starting Address after DUMP or GET.
72		Starting Address after CTRL Q.
73		Two's complement of the number of ticks left in the next second.
74		Two's complement of the line frequency.
75		Number of RTF Lines (for BOSS Mode).
76-1Ø5		Unused.

	Word #		Value	Description
	Ø		ØØØnnn	Pointer to first free word after SYSBLK (There is one set of seven words/core image program.)
	.			
	.			
	7N+1		.SIXBT	Name of System Program or overlay
	7N+2		.SIXBT	
S	7N+3		nnnnnn	Number of first block on system device occupied by this program or overlay.
Y	7N+4		ØØØØnn	Number of blocks occupied by this program or overlay
S				
B	7N+5		addres	Thirteen-bit first address for this program or overlay
L	7N+6		Ønnnnn	Program size
K	7N+7		addres	Thirteen-bit starting address for this program or overlay
	.		.	.
	.		.	.
	.		.	.
			(free area)	
	5ØØ	prog1	ØØØØ1Ø	Number of words in this entry (in this case, 1Ø)
	5Ø1	prog1	.SIXBT	Name of this system program (left-justified and zero-filled)
	5Ø2	prog1	.SIXBT	
	5Ø3	prog1	.SIXBT	Name of an overlay (left-justified and zero-filled) -- overlays are optional
	5Ø4	prog1	.SIXBT	
C	5Ø5	prog1	ØØØØØ2	Number of buffers required by this system program (Bits Ø-6 = Ø means the end of any overlay names. This is why program and overlay names must be left-justified.)
O				
M	5Ø6	prog1	.DAT&777	Active .DAT slot
B	5Ø7	prog1	.DAT&777	Active .DAT slot (Note: 777777 for a .DAT slot means all positive .DAT slots.)
L	51Ø	prog2	ØØØØØ5	Number of words for this entry (in this case, 5)
K	511	prog2	.SIXBT	Name of this system program
	512	prog2	.SIXBT	
	513	prog2	ØØØØØ1	Number of buffers required by this program (Note that his program has no overlays.)
	514	prog2	.DAT&777	.DAT slot for this program
	.		.	.
	.		.	.
	.		.	.
	777		ØØØ5ØØ	Pointer to first word in COMBLK (equals count from first word in SYSBLK). The two contiguous blocks on the system device that hold SYSBLK and COMBLK are treated by the system as one large block. In this case, COMBLK happens to start at location 5ØØ of the two blocks combined.

Figure 5-1

SYSBLK and COMBLK

names of core-image system programs by making them the new first entry. In this way, SYSBLK and COMBLK build toward the center.

5.2.3 SGNBLK

SGNBLK (block 36 on the system device) contains all the system parameters not directly associated with core-image system programs. The bulk of SGNBLK is concerned with I/O (.DAT slots, .UFDT slots, Skip Chain Order, Handlers, and skip IOT codes and mnemonics). The first few registers hold such important system information as the system device, .SCOM+4 contents, and so on. The very first word in SGNBLK points to the block address of the first free word after SGNBLK. The next entry is an offset word indicating the total length (including itself) of the miscellaneous system parameter table to follow. This table includes the size of the .DAT and the size of the skip chain. The end of the handler and skip IOT table is the first free entry of the block.

The .DAT slot table corresponds to the legal range of .DAT slots, with the maximum negative set to -15 and the maximum positive set to a number not to exceed 77_8. The .DAT slots are in the form in which they appear when the Nonresident Monitor is in core. That is, the unit number is in bits Ø-2, and the number of the handler right-justified in bits 3-17. The handler number for the first handler in the Device Handler-Skip IOT Table is zero, for the pseudo-handler NON. TTA. is one, and so on. The constant 1ØØØØØ indicates a fixed or illegal .DAT slot (such as -2, -3, and Ø). DOSGEN will not modify such slots.

The .UFD Table is in one-to-one correspondence with the .DAT slot Table. An entry of .SIXBT 'UIC' indicates that the logged in UIC is to be substituted for the name UIC in the table. An entry of .SIXBT 'SYS' indicates BNK or PAG is to be substituted, in accordance with the current addressing mode. Otherwise, the contents of each location will be the .SIXBT representation of the corresponding .UFD slot.

The Skip Chain Table lists the system skip IOT's in order. A negative skip (one that skips on "off", not "on") is represented in one's complement. Not all skips in the handler Skip IOT Table (described below) need to be included in the Skip Chain Table.

The Device Handler/Skip IOT Table contains all the handler names and skip IOT numbers and mnemonics for each I/O device identified to the system. Every such device has an entry in the table. A handler name must be exactly three characters in length, with the last character not an octal digit. The device code for a device is exactly two characters. The first two characters of each handler name for a device must be the device code. This fact is essential for understanding the format of a device entry, since the device code is never stored as such in an entry, but is inferred from the device handler name. The typical entry for a device is the following:

1. The first words of an entry contain the handler names for a device in .SIXBT. Each handler name is different, and the end of the list of handlers is determined by a word with zeros in bits Ø-5 (the first character position).

2. The word that terminated the list of handler names contains the number of skip IOT's for the device. For each skip IOT, there are three words in the table: two for the skip mnemonic and one for the actual code.

The next device entry follows the last skip for the previous device. Handlers may be entered without any skips, but no devices may be entered without at least one handler name. Figure 5-2 illustrates the organization of SGNBLK. Appendix D of SGEN-DOS Utility Programs, DEC-15-YWZB-DN12, lists SGNBLK, SYSBLK and COMBLK, as they are supplied by Digital Equipment Corporation.

5.3 DISK-RESIDENT CHANGING BLOCKS

The System Loader uses block 37 of the system device to store an image of .DAT and .UFDT. Other disk-resident changing blocks are the storage Allocation Table and the Bad Allocation Table. These tables are described in Chapter 6.

5.4 TEMPORARY TABLES BUILT FROM DISK-RESIDENT TABLES

5.4.1 The Overlay Table

The System Loader builds the Overlay Table from the entries in SYSBLK referenced by a core-image system program's entry in COMBLK. That is, the Overlay Table contains an entry for the system program itself, and one for each of its overlays. Figure 5-3 illustrates the format of an entry in the Overlay Table. The first entry in the Overlay Table is

Location	Value	Description
Ø	ØØØnnn	Pointer to first free entry in SGNBLK
1	ØØØØ15	Number of miscellaneous parameters
2	ØØØnnn	Size of .DAT plus size of .UFDT = (number of positive .DAT slots+16)*2. (Initial value is 2Ø positive .DAT slots.)
3	ØØØnnn	Number of skips in Skip Chain
4	Ø413ØØ Ø42ØØØ	System device code in .SIXBT
5	nnnnnn	Original contents of .SCOM+4
6	nnnnnn	Original contents of .SCOM+2Ø
7	nnnnnn	Number of words per buffer (.SCOM+27)
1Ø	nnnnnn	Default number of buffers (.SCOM+26)
11	.SIXBT	Monitor Identification Code
12	nnnnnn	Information on VT and CTRL X (.SCOM+33)
13	ØØØØØn	Default files protection code (.SCOM+54)
14	ØØnnnn	Size of the Resident Monitor Patch Area
15	7777nn	Minus the number of clock ticks in a second (-74 for 6Ø hz, -62 for 5Ø hz.)
16 . . 53	ØØØnnn ØØØnnn	Device assignments for the .DAT (made by handler numbers). (Termination at 53 assumes 2Ø positive slots.)
54 . . 111	.SIXBT .SIXBT	UIC assignments for the .UFDT. (Termination at 111 assumes 2Ø positive slots.)
112 . . 137	nnnnnn nnnnnn	Skip Chain Table (Negative skips in one's complement.) (Termination at 137 assumes 22 skips in chain.)
14Ø 312	.SIXBT . . . .SIXBT .SIXBT . . .SIXBT ØØØØØ3 nnnnnn nnnnnn nnnnnn .SIXBT ØØØØØ1 nnnnnn . . . SGNBLK ends at 312, in the DOS-15 system distributed by Digital Equipment Corporation.	The last part of the SGNBLK is the Device Handler-Skip IOT Table. Each entry starts with the .SIXBT representations of all handlers for a particular device. (First two characters equal device code, for all handlers.) Zeros in the first six bits of a word indicate the end of the handler names, and says that the rest of the word contains the number of skips for this entry's device. The skip IOT's follow immediately. As above, one's complement skips indicate negative skips. Note, however, the confusing fact that a one's complement of a skip IOT is a positive number. Thus, 7Ønnn complemented is Ø7nnnn.

Figure 5-2

SGNBLK

pointed to by .SCOM+31. .SCOM+31 will contain zero, if there are no entries in the Overlay Table. This will occur during Linking Loader or EXECUTE loads.

.OVRLA is the only Monitor function that looks at the Overlay Table. If the .OVRLA processor finds a match to the .OVRLA argument in the Overlay Table, it uses the parameters listed in the table to bring it in via a Monitor TRAN. Note that this bypasses the System Loader, and does not change the handler load. Thus, the overlay must use only those .DAT slots required by the original program, the one listed in COMBLK.

If the .OVRLA processor does not find a match in the Overlay Table, it calls in the System Loader, which searches COMBLK for the requested program. This type of overlay request does not require that .DAT slot assignments be the same. On the other hand, the System Loader refreshes all of core except .SCOM, etc. Thus, communication between overlays is more difficult. The resident patch area, however, can be used for this purpose.

5.4.2 The Device Table

The Device Table is built by the System Loader interface whenever PIP is being loaded, or when PIP is listed in COMBLK among the overlays for a program. It is located just above the register pointed to by .SCOM+1, and has an entry for each positive .DAT slot. If a slot has an assigned device, the low-order twelve bits of the corresponding entry in the Device Table will contain the device's code, in .SIXBT. Bit 3 is set when the slot is busy. If no device is assigned to a slot, the corresponding entry in the Device Table will contain zero.

5.4.3 The Input/Output Communication (IOC) Table

The System Loader Interface builds the IOC Table and locates it just below the first register of the System Loader. It contains an entry for each handler in the system, in the order that they appear in SGNBLK. The entries themselves contain the handler name in Radix 50. The System Loader and the Linking Loader use the handler number supplied by the Nonresident Monitor to index down the IOC Table. They use the contents of the entry for a .SEEK to the IOS UIC.

5.4.4 The Device Assignment Table (.DAT)

The Device Assignment Table makes the association between logical and physical devices. The Monitor knows its location by the contents of .SCOM+23, which points to the zeroth entry in the Table. Specific slots are found by indexing on the contents of .SCOM+23. The number of negative slots is fixed at 15_8. The number of positive slots is specified by .SCOM+24, and may be any positive number less than 100_8. It is specified at system generation time.

The Nonresident Monitor places the handler number in the low order bits and the unit number in the high order bits. It derives the handler number from SGNBLK. As mentioned above, the System Loader and the Linking Loader subsequently use the IOC Table to determine the handler name. After either loader has loaded and relocated a handler, it places the handler's starting address in all .DAT slots that reference that handler. The unit number remains in the high-order three bits. Slots with no handler (NON) contain zero. Active .DAT slots are designated by COMBLK, for core-image system programs, and by .IODEV pseudo-ops for the Linking Loader and EXECUTE.

5.4.5 The User File Directory Table (.UFDT)

.UFDT+Ø is offset from .DAT+Ø (pointed to by .SCOM+23) by the sum of the positive and negative .DAT slots. Each .DAT slot has a corresponding .UFDT slot. UIC's in the .UFDT are packed in .SIXBT. The address of .UFDT+Ø is stored in .SCOM+25.

5.4.6 The Skip Chain

Register 1 of Bank Ø contains a jump to the beginning of the Skip Chain. The Skip Chain is defined during System Generation, is located in SGNBLK, and is rebuilt every time the System Loader is called in. The System Generator Manual (DEC-15-YWZB-DN12) describes considerations for constructing the Skip Chain.

5.5 TEMPORARY TABLES BUILT FROM SCRATCH

5.5.1 File Buffer Transfer Vector Table

The System Loader allocates space for the buffer pool, and creates the File Buffer Transfer Vector Table. .SCOM+30 points to the first entry

in the table, and the number of entries is specified by .SCOM+26. Each entry in the table contains the address of a buffer, or its one's complement. Negative addresses indicate a busy buffer. Since references to buffers must be indirect anyway, buffers are allocated without regard to bank boundaries.

5.5.2 The RCOM Table

The Nonresident Monitor requires certain information about the Resident Monitor that does not warrant reserving additional .SCOM registers. The System Loader therefore puts this information into the RCOM table, whenever it is loading the Nonresident Monitor. The RCOM Table starts at register 175ØØ of the highest bank.

5.5.3 The Mass Storage Busy Table

Entries in this table are allocated by the System Loader or the Linking Loader. The Mass Storage Busy Table is pointed to by .SCOM+62. .SCOM+63 contains the number of words per entry in the table, and .SCOM+36 contains the current number of entries. Generally speaking, there are as many entries in the Busy Table as there are active .DAT slots, although the disk handlers are the only ones that currently refer to the Busy Table.

The .INIT command to a disk handler establishes a Busy Table entry. The .CLOSE command (or the Rewind .MTAPE command) deletes the corresponding entry. Figure 5-4 illustrates a typical Busy Table Entry.

The first word of an active entry in the Busy Table contains the .DAT slot in bits 9-17. The disk handlers save information about the UFD current for this .DAT slot in the Mass Storage Busy Table. They save information about the file current to the .DAT slot (if any) in the buffer pointed to by word 1 of the Busy Table Entry. More information on the disk handlers and file structure is contained in Chapter 6.

5.6 RESERVED WORD LOCATIONS

Word locations Ø through 77 are dedicated systems locations and cannot be employed by the user. The contents of these locations are described in Table 5-5.

Word #	Contents
N,N+1	.SIXBT name of Overlay
N+2	First block number
N+3	First address, minus 1
N+4	Size, in two's complement
N+5	Fifteen-bit starting address

Table 5-3 Overlay Table

Word #	Contents
N	Device Type$_{\emptyset-2}$, Unit Number$_{3-5}$, Write Check$_{6}$,.DAT$_{9-17}$
N+1	Buffer Address, or $\emptyset$, if none allocated
N+2	Three-character UIC
N+3	First UFD block for this UIC
N+4	UFD Entry size for files in this UFD

Table 5-4 Mass Storage Busy Table Entry

ADDRESS	USE
$\emptyset$	Stores the contents of the extended PC, link, extend mode status, and memory protect status during a program interrupt
1	EEM (for PDP-9 compatibility)
2	JMP to Skip Chain
3	.MED, entry to Monitor Error Diagnostic routine
4	JMP to error handler
5	Stores system type (Bank or Page) indicator during Teletype interrupts
6	Used for API ON/OFF indicator
7	Stores real time clock count
10-17	Autoindex registers
20	Stores the contents of the extended PC, link, mode status, and memory protect status on a CAL instruction.
21	JMP to CAL handler
22-37	Seven pairs of word counter-current address registers for use with 3-cycle I/O device data channels.
40-77	Store unique entry instructions for each of 32_{10} automatic priority interrupt channels

Table 5-5 Reserved Address Locations

5.7 BOOTSTRAP NON-BOSS BATCH BITS

The high-order three bits of word 17777 in the Bootstrap are reserved for the Monitor, and have the following meanings:

Bit Ø 1 = In non-BOSS Batch Mode
       Ø = Not in non-BOSS Batch

Bit 1 1 = $JOB ASCII line or card just read by batch device
       Ø = Last line or card not $JOB

Bit 2 1 = Batch device is card reader
       2 = Batch device is paper tape reader

CHAPTER 6

FILE STRUCTURES

6.1 DECTAPE FILE ORGANIZATION

DECtape can be treated either as a directoried or non-directoried device.

6.1.1 Non-Directoried DECtape

A DECtape is said to be non-directoried when it is treated as magnetic tape by issuing the .MTAPE commands: REWIND, BACKSPACE, followed by .READ or .WRITE. No directory of identifying information of any kind is recorded on the tape. A block of data (255_{10} word maximum), exactly as presented by the user program, is transferred into the handler buffer and recorded at each .WRITE command. A .CLOSE terminates recording with a software end-of-file record consisting of two words: 001005, 776773

Because braking on DECtape allows for tape roll, staggered recording of blocks is employed in DOS to avoid constant turnaround or time-consuming back and forth motion of physically sequential block recording. When recorded as a non-directoried DECtape, block Ø is the first block recorded in the forward direction. Thereafter, every fifth block is recorded until the end of the tape is reached, at which time recording, also staggered, begins in the reverse direction. Five passes over the tape are required to record all 1100_8 blocks.

6.1.2 Directoried DECtape

Just as a REWIND or BACKSPACE command declares a DECtape to be non-directoried, a .SEEK or .ENTER implies that a DECtape is to be considered directoried. A directory listing of any such DECtape is available via the (L)ist command in PIP. A fresh directory may be recorded via the N or S switch in PIP.

The directory of all DECtapes except system tapes occupies all $4ØØ_8$ words of block $1ØØ_8$. It is divided into two sections: (1) a 40_8 word Directory Bit Map and (2) a 340_8 word Directory Entry Section.

The Directory Bit Map defines block availability. One bit is allocated for each DECtape block (1100_8 bits = 40_8 words). When set to 1, the bit indicates that the DECtape block is occupied and may not be used to record new information.

The Directory Entry Section provides for a maximum of 56_{10} files on a DECtape. Each file on the DECtape has a four-word entry. Each entry includes the three-word file name and extension, a pointer to the first DECtape block of the file, and a file active or present bit. Figure 6-1 illustrates the DECtape directory.

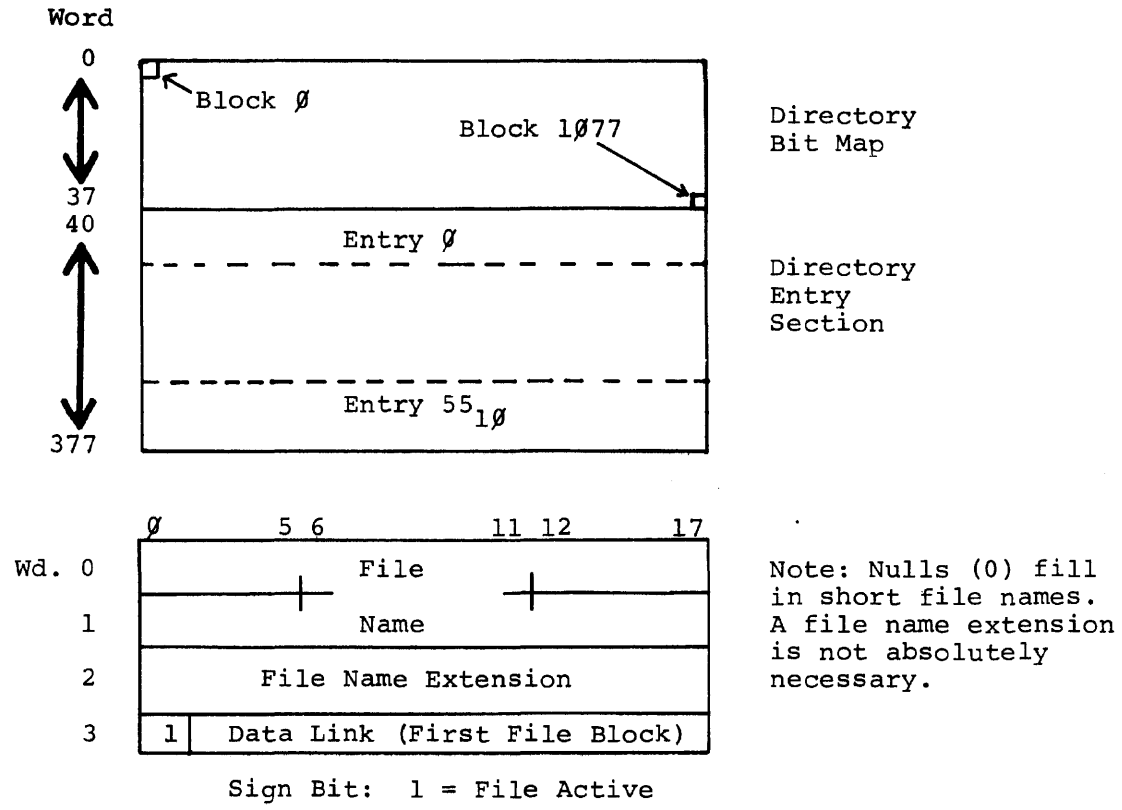


Figure 6-1

DECtape Directory

Additional file information is stored in blocks 71 through 77 of every directoried DECtape. These are the File Bit Map Blocks. For each file in the directory, a 40_8-word File Bit Map is reserved in block 71 through 77. The bit maps are contiguous, and the N^{th} file uses the

N^{th} bit map. Each block is divided into eight File Bit Map Blocks. A File Bit Map specifies the blocks occupied by that particular file and provides a rapid, convenient method to perform DECtape storage retrieval for deleted or replaced files. Note that a file is never deleted until the new one of the same name is completely recorded on the .CLOSE of the new file. When a fresh directory is written on DECtape, blocks 71 through 100 are always indicated in the Directory Bit Map as occupied. Figure 6-2 illustrates DECtape file bit maps.

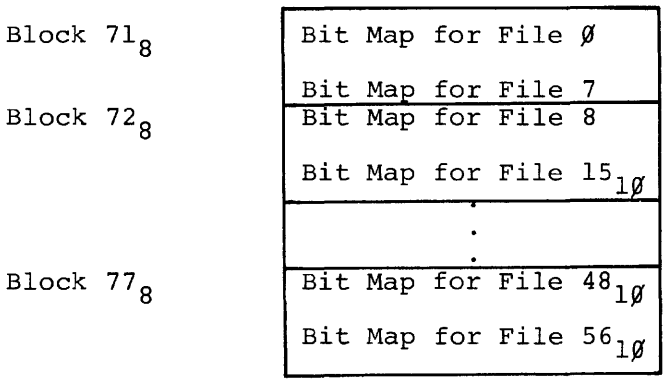


Figure 6-2

DECtape File Bit Map Blocks

Staggered recording (at least every fifth block) is used on directoried DECtapes, where the first block to be recorded is determined by examination of the Directory Bit Map for a free block. The first block is always recorded in the forward direction; thereafter, free blocks are chosen which are at least five beyond the last one recorded. The last word of each data block recorded contains a data link or pointer to the next block in the file. When turnaround is necessary, recording proceeds in the same manner in the opposite direction. When reading, turnaround is determined by examining the data link. If reading has been in the forward direction, and the data link is smaller than the last block read, turnaround is required. If reverse, a block number greater than the last block read implies turnaround.

A software end-of-file record (001005, 776773) terminates every file. The data link of the final block is 777777.

Data organization for each I/O medium is a function of the data modes. On directoried DECtape there are two forms in which data is recorded: (1) packed lines - IOPS ASCII, IOPS Binary, Image Alphanumeric, and Image Binary, and (2) dump mode data - Dump Mode.

In IOPS or Image Modes, each line (including header) is packed into the DECtape buffer. In IOPS Binary, a 2's complement checksum is computed and stored in the second word of the header. When a .WRITE which will exceed the remaining buffer capacity is encountered, the buffer is output, after which the new record is placed in the empty buffer. No record may exceed 254_{10} words, including header, because of the data link and even word requirement of the header word pair count. An end-of-file is recorded on a .CLOSE. It is packed in the same manner as any other line.

In Dump Mode, the word count is always taken from the I/O macro. If a word count is specified which is greater than 255_{10} (note that space for the data link must be allowed for again), the DECtape handler will transfer 255_{10} word increments into the DECtape buffer and from there to DECtape. If some number of words less than 255_{10} remain as the final element of the Dump Mode .WRITE, they will be stored in the DECtape buffer, which will then be filled on the next .WRITE, or with an EOF if the next command is .CLOSE. DECtape storage is thus optimized in Dump Mode since data is stored back-to-back. See Appendix A.

6.2 MAGNETIC TAPE

DOS provides for industry-compatible magnetic tape as either a directoried or non-directoried medium. The magnetic tape handlers communicate with a single TC-59D Tape Control Unit (TCU). Up to eight magnetic tape transports may be associated with one TCU; these may include any combination of transports TU-10A or B and TU-30A or B.

There are a number of major differences between magnetic tape and DECtape or Disk; these differences affect the operation of the device handlers. Magnetic tape is well suited for handling data records of variable length. Such records, however, must be treated in serial fashion. The physical position of any record may be defined only in relation to the preceding record. Three techniques available in I/O operations to block-addressable devices are not honored by the magnetic tape handlers:

a. The user cannot specify physical block numbers for transfer. In processing I/O requests that have block numbers in their argument lists (i.e., .TRAN) the handler ignores the block-number specification.

b. The only area open for output transfers in the directoried environment is that following the logical end of tape.

c. Only a single file may be open for transfers (either input or output) at any time on a single physical unit.

6.2.1 Non-directoried Data Recording (MTF)

MTF is intended to satisfy the requirements of the FORTRAN programmer while still providing the assembly language programmer maximum freedom on the design of his tape format. MTF writes out a record to the tape each time the main program issues a .WRITE. The length of the record is always two times the word pair count in the header word pair. FIOPS records are always as long as the buffer size returned on a .INIT (up to 256_{10} words). MTF returns a standard buffer size of 377_8, after a .INIT. The FORTRAN user may dynamically change this size, however, via the following instructions.

```
(FORTRAN STATEMENTS)                (MACRO STATEMENTS)
.                                   .TITLE SETMTB
.                                   .GLOBL .DA, MTBSIZ, SETMTB
.                        SETMTB     Ø
CALL SETMTB (ISIZE)                 JMS*    .DA
.                                   JMP     START
.                        BUFSIZ     Ø
                         START      LAC*    BUFSIZ   (any buffer size)
                                    DAC*    MTBSIZ
                                    JMP*    SETMTB
                                    .END
```

6.2.2 Directoried Data Recording (MTA., MTC.)

The programmer can make the fullest possible use of those features peculiar to magnetic tape by using MTF. On the other hand, MTF does not offer the powerful file-manipulation facilities available in the system. Directoried I/O allows device independence, and extensive use of the storage medium with a minimum of effort.

Every block recorded by MTA. (with the exception of end-of-file markers, which are hardware-recorded) includes a two-word Block Control Pair and not more than 255_{10} words of data. The data will contain the records from one or more .WRITE's.

The Block Control Pair serves three functions: it specifies the character of the block (label, data, etc.), provides a word count for the block, and gives an 18-bit block checksum. The Block Control Pair has the following format:

Word 1:

Bits 0 through 5: Block Identifier (BI). This 6-bit byte specifies the block type. Values of BI may range from 0 to 77_8. Current legal values of BI, for all user files, are as follows:

BI Value	Block Type Specified
00	User-File Header Label
10	User-File Trailer Label
20	User-File Data Block

Bits 6 through 17: Block Word Count (BWC). This 12-bit byte holds the 2's complement of the total number of words in the block (including the Block Control Pair). Legal values of BWC range from -3 to -401_8.

Word 2:

Bits 0 through 16: Block Checksum. The Block Checksum is the full-word, unsigned, 2's complement sum of all the data words in the block and word 1 of the Block Control Pair.

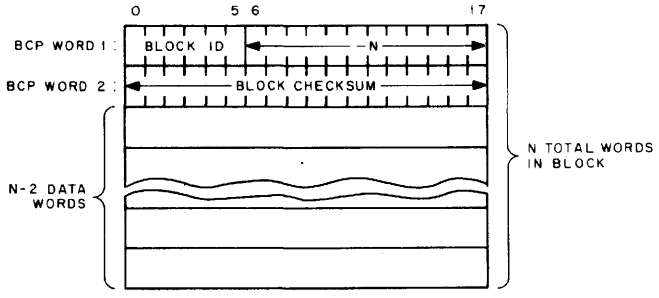


Figure 6-3

Block Format, File-Structured Mode

One of the main file functions of MTA. and MTC. is that of identifying and locating referenced files. This is carried out by two means: first, names of files recorded are stored in a file directory at the beginning of the tape; and second, file names are contained in the file's header and trailer labels.

6.2.2.1 Magnetic Tape File Directory

The directory, a single-block file (and the only unlabeled file on any file-structured tape), consists of the first recorded data block on the tape. It is a 257_{10} word block with the following characteristics:

a. Block Control Pair (words 1 and 2)

   Word 1

   Block Identifier = 74_8 = File Directory Data Block
   Block Word Count = -401_8 = 7377_8.

   Word 2:

   Block Checksum: As described above.

b. Active File Count (Word 3, Bits 9 through 17) 9-bit one's complement count of the active file names present in the File Name Entry Section (described below).

c. Total File Count (Word 3, Bits 0 through 8) 9-bit one's complement count of all files recorded on the tape, including both active and inactive files, but not the file directory block.

d. File Accessibility Map (Words 4 through 17): The File Accessibility Map is an array of 252_{10} contiguous bits beginning at bit 0 of word 4 and ending as bit 17 of word 17. Each of the bits in the Accessibility Map refers to a single file recorded on tape. The bits are assigned relative to the zeroth file recorded; that is, bit 0 of word 4 refers to the first file recorded; bit 1, word 4, to the second file recorded; bit 0, word 6, to the 37_{10}^{th} file recorded; and so on, for a possible total of 252_{10} files physically present.

   A file is only accessible for reading if its bit in the Accessibility Map is set to one. A file is made inaccessible for reading (corresponding bit = 0) by a .DLETE of the file, by a .CLOSE (output) of another file of the same name, or by a .CLEAR. A file is made accessible for reading (corresponding bit = 1) by a .CLOSE (output) of that file. Operations other than those specified above have no effect on the File Accessibility Map.

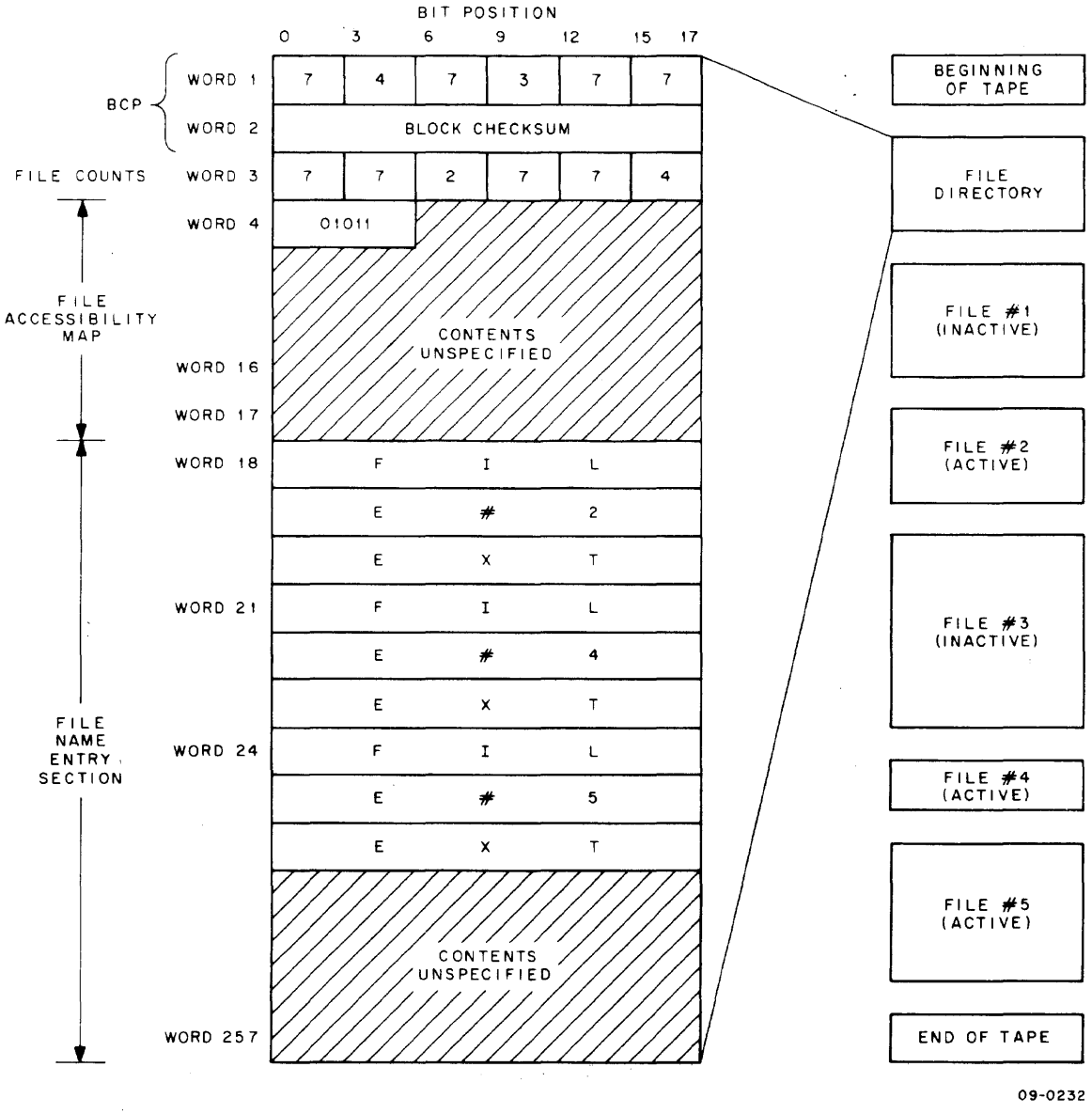


Figure 6-4a. Format of the File Directory Data Block, showing relationship of active and inactive files to file name entries and to Accessibility Map

Figure 6-4b. Format of file-structured tape, showing directory block and data files.

e. File Name Entry Section (Words 18 through 257): The File Name Entry Section, beginning at word 18 of the directory block, includes successive 3-word file name entries for a possible maximum of 80 entries. Each accessible file on the tape has an entry in this section. Entries consist of the current name and extension of the referenced file in .SIXBT (left-adjusted and, if necessary, zero-filled).

The position of a file name entry relative to the beginning of the section reflects the position of its accessibility bit in the map. That bit, in turn, defines the position of the referenced file on tape with respect to other (active or inactive) files physically present. Only active file names appear in the entry section, and accessibility bits for all inactive files on the tape are always set to zero; accessibility bits for all active files are set to one.

To locate a file on the tape having a name that occupies the second entry group in the File Name Entry Section, the handler must (a) scan the Accessibility Map for the second appearance of a 1-bit, then (b) determine that bit's location relative to the start of the map. That location specifies the position of the referenced file relative to the beginning of the tape. The interaction of the File Name Entry Section and the Accessibility Map are shown in Figure 6-4.

6.2.2.2 User-File Labels

Associated with each file on tape is one label, the header label. It precedes the first data block of the file. Each label is 27_{10} words in length. Label format is shown in Figure 6-5.

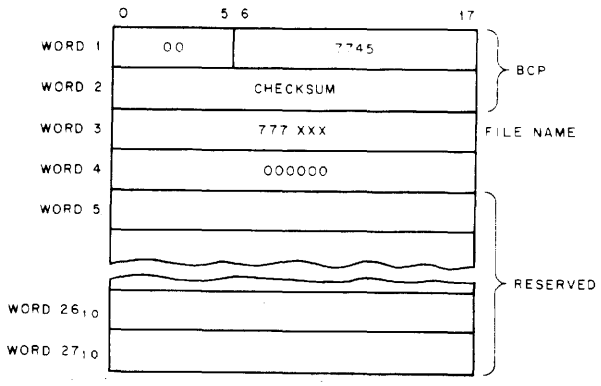


Figure 6-5

User-File Header Label Format

6.2.2.3 File-Names in Labels

The handler will supply the contents of the file-name fields (Word 3) in labels. These are used only for control purposes during the execution of .SEEK's. The name consists simply of the two's complement of the position of the recorded file's bit in the Accessibility Map; the "name" of the first file on tape is 777777, that of the third file is 777775, and so on. A unique name is thus provided for each file physically present on the tape. Since there may be a maximum of 252_{10} files present, legal file-name values lie in the range 777777 to 777404.

6.2.3 Continuous Operation

Under certain circumstances, it is possible to perform successive I/O transfers without incurring the shut-down delay that normally takes place between blocks. The handler stacks transfer requests, and thus ensures continued tape motion, under the following conditions:

a. The I/O request must be received by the CAL handler before a previously-initiated I/O transfer has been completed.

b. The unit number must be identical to that of the previously initiated I/O transfer.

c. The I/O request must be one of those listed below to ensure successful completion. The handler in processing requests in continuous mode depends on receiving control at the CAL level in order to respond to I/O errors. The functions for which continuous operation is attempted include only the following:

1.	.MTAPE	3.	.WRITE
2.	.READ	4.	.TRAN

d. With MTA, more than one logical record may be in a physical block, so tape motion may stop if fewer successive .READ's or .WRITE's are issued than there are records in a block.

e. The previously-requested transfer must be completed without error. In general, successive error-free READ's (WRITE's) to the same transport will achieve non-stop operation. The following examples illustrate this principle.

Example 1: Successful Continued Operation

```
SLOT = 1
INPUT = 0
BLOKNO = 0
READ1          .TRAN SLOT, INPUT, BLOCKNO, BUFF1, 257
READ2          .TRAN SLOT, INPUT, BLOCKNO, BUFF2, 257
RETURN         JMP READ1
```

The program segment in Example 1 will most probably keep the referenced transport (.DAT slot 1) up to speed. The probability decreases as more time elapses between READ1 and READ2, and between READ2 and RETURN. Each .TRAN request causes an implicit .WAIT until its operation is completed.

Example 2: Unsuccessful Continued Operation

```
SLOT = 1
INPUT = 0
BLOKNO = 0
READ            .TRAN SLOT, INPUT, BLOKNO, BUFF, 257
STOP            .WAIT SLOT
RETURN          JMP READ
```

The program segment in Example 2 will not keep the tape moving because the .WAIT at location STOP prevents control from returning to location READ until the transfer first initiated at READ has been completed.

Example 3: Unsuccessful Continued Operation

```
SLOT1 = 1
SLOT2 = 2
INPUT = 0
BLOKNO = 0
READ1           .TRAN SLOT1, INPUT, BLOKNO, BUFF1, 257
READ2           .TRAN SLOT2, INPUT, BLOKNO, BUFF2, 257
RETURN          JMP READ1
```

This program segment will not provide non-stop operation because of the differing unit specification at READ1 and READ2.

6.2.4 Storage Retrieval on File-Structured Magnetic Tape

The use of a file accessibility map as well as block identifiers in Magtape file directories makes it almost impossible to retrieve the area of a deleted file from a magnetic tape. The execution of the deletion command (i.e., .DLETE) removes the name of the object file from the file directory, and clears the corresponding bit in the File Accessibility Map.

The only circumstance under which a file area may be easily retrieved is when the deleted file is also the last file physically on the tape. Under these conditions, the handler can retrieve the area occupied by the deleted file when the next .ENTER - .WRITE - .CLOSE sequence is executed. Users may also copy the active files to another device, renew the directory, and recopy the files.

6.3 DISK FILE STRUCTURE

6.3.1 Introduction

The DOS-15 disk file structure is in some ways analogous to DECtape file structure. Ordinarily, each disk user has a directory which points to named files, just as each DECtape has a directory. The DECtape has only one directory, but the disk has as many directories as users have cared to establish. A single user's disk directory might correspond to a single DECtape directory. A single disk file's size is also limited only by the available space, as is true with DECtape. Whereas DECtape directories may only reference a maximum of 56_{10} files, however, the number of files associated with any one directory on the disk is limited only by the available disk space.

The DECtape directory is in a known location -- at block 100. Since the disk may have a variable number of directories, the Monitor must know how to find each user's directory. It therefore maintains a Master File Directory (MFD) at a known location[1], and the Master File Directory points to each User File Directory (UFD). DOS-15 allows only those users who know the Master Identification Code to have access to any protected UFD's within the MFD. Figure 6-6 illustrates the MFD. Appendix B is a flowchart of the Disk "A" Handlers.

6.3.2 User Identification Codes (UIC)

The Monitor finds User File Directories by seeking associated User Identification Codes (UIC's), which are all listed in the Master File Directory. The UIC is a three-character code that is necessary for all non-.TRAN I/O to the disk. .TRAN macros use no directory references. A programmer may operate under as many UIC's as he wishes, provided all are unique and none is reserved[2]. He may establish a new User File Directory by (1) logging in his new UIC to the Monitor via the LOGIN command, (2) calling PIP, and (3) issuing an N DK command. This establishes a new User File Directory, or refreshes (wipes clean) an old directory under that UIC. (.ENTER will also create a new MFD entry and/or a UFD, if none exists.) Figure 6-7, User File Directory, illustrates the organization of a UFD.

---

[1]On the RF disk, the first block of the MFD is located at block 1777. On the RP disk, the first block of the MFD is located at block 47Ø4Ø.

[2]The following are reserved UIC's: @@@, ???, PAG, BNK, SYS, IOS, CTP.

Word #	Contents	Description
Ø	777777	Dummy UIC used by system.
1	nnnnnn	Bad Allocation Table's first block number, or 777777, if there is none.
2	nnnnnn	SYSBLK's first block number, or -1, if there is none.
3	$4_{Ø-2}$+blknum	MFD entry size in bits Ø-2, plus the block number of the first submap
⋮	⋮	⋮
4N	.SIXBT	UIC for this UFD
4N+1	nnnnnn	Block number for the first block of this UFD or 777777, if no UFD exists (as after PIP's N␣DK⏎)
4N+2	$P_{Ø}$+M	Protection code in bit Ø, plus the UFD entry size for each file
4N+3	spare	Unused at this writing
⋮	⋮	⋮
376	nnnnnn	Pointer to previous MFD block, or 777777 if none.
377	nnnnnn	Pointer to next MFD block, or 777777 if none.

Figure 6-6

Master File Directory

Word #	Contents	Description
⋮	⋮	⋮
8N	.SIXBT	Name of this file
8N+1	.SIXBT	and its
8N+2	.SIXBT	extension
8N+3	$T_{Ø}$+blknum	Truncation code in bit Ø, plus the number of the first block of the file
8N+4	nnnnnn	Number of blocks in this file
8N+5	ribptr	Pointer to the first block of the Retrieval Information Block
8N+6	$P_{Ø-1}$+ribwrd	Protection code in bits Ø-1, plus the first word in ribptr used by the RIB-- if the last block of the file has room for the RIB, the handlers will put it there, and load word 8N+6 accordingly.
8N+7	crdate	Date of file's creation -mmddyy (yy modulo 7Ø)
⋮	⋮	⋮
376	nnnnnn	Pointer to previous block, or 777777 if none
377	nnnnnn	Pointer to next UFD block, or 777777 if none

Figure 6-7

User File Directory

6.3.3 Organization of Specific Files on Disk

The Disk Handlers write out files in almost the same way that a DECtape handler does. Disk file blocks, however, have a forward and backward link. (Non-dump records are therefore limited to lengths of 254_{10} words.) Further, upon receipt of a .CLOSE I/O macro, the disk handlers fill out a Retrieval Information Block (RIB). The RIB performs the same functions as the file bitmap on DECtape, and also associates the logical sequence of blocks in the file with the physical locations of the blocks on the disk. The disk handler uses the RIB to implement .RTRAN commands and to delete files. Figure 6-8, The Retrieval Information Block, illustrates a RIB.

After a user has created a disk file he can access logical records sequentially via .READ commands, just as with DECtape files. He can also access physical blocks of that file by referencing relative block numbers in the .RTRAN command. (The .RTRAN commands require the file be opened with the .RAND command.)

6.3.4 Buffers

The handlers break buffers from the pool into three parts: (1) File Information (about 40_8 words)*, (2) the Block List -- addresses of pre-allocated blocks (between 4 and 253_{10} addresses, inclusive), and (3) data buffer (256_{10} words). Figure 6-9, Disk Buffer, illustrates the breakdown of disk buffers.

6.3.4.1 Commands That Obtain And/or Return Buffers

The following commands obtain buffers from the pool, and return them immediately after execution:

.DLETE
.RENAM
.CLEAR

The following commands obtain a buffer from the pool and do not return it until a subsequent .CLOSE is performed:

.FSTAT
.ENTER
.SEEK
.RAND

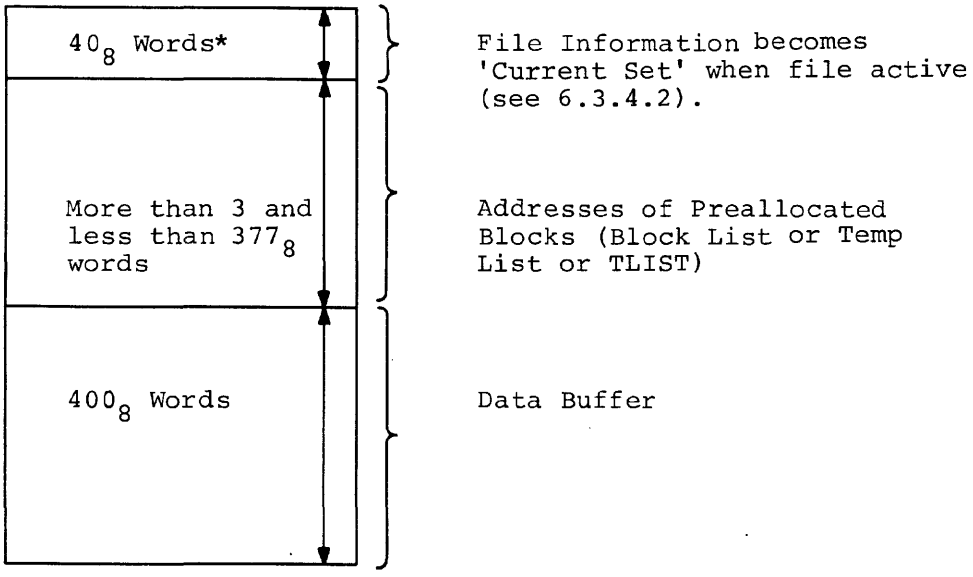
*This number is determined by assembly parameters.

Word #	Contents	Description
Ø*	nnnnnn	Total number of blocks described by this physical block.
1	nnnnnn	First data block pointer.
2	nnnnnn	Second data block pointer.
3	nnnnnn	Third data block pointer.
.	.	.
.	.	.
.	.	.
376	nnnnnn	Pointer to previous RIB block or -1 if no previous RIB block.
377	nnnnnn	Pointer to next RIB block or -1 if no next RIB block.

* Zeroth word of the RIB may not be zeroth word of physical block. This occurs whenever the entire RIB will fit in the last data block of the file.

Figure 6-8

Retrieval Information Block



*This is not a fixed number. It is different for RP and RF.

Figure 6-9

Disk Buffer

The following commands return a buffer to the pool, if any was allocated.

.INIT
.CLOSE
.MTAPE (rewind)

6.3.4.2 The Current Set

The handlers retain information about the last file and .DAT slot processed in an internal storage area. This area is called the "Current Set", and is swapped back to the file's buffer whenever a command to a different file is used. Thus,

.WRITE to .DAT slot A
.WRITE to .DAT slot B

will swap the Current Set, but...

.WRITE to .DAT slot A
.TRAN to .DAT slot A
.WRITE to .DAT slot A

will not swap the Current Set.

6.3.5 Pre-allocation

The handlers pre-allocate blocks on the disk upon all .ENTER commands, and whenever sufficient .WRITE commands have been issued to use up the pre-allocated blocks. The number of pre-allocated blocks will be the minimum of the number of free blocks on the device and the number of address slots available in the Temp List (block list).

When the handlers pre-allocate blocks, they fill out the bit maps, and immediately fill out the RIB and write it out in one of the pre-allocated blocks.

Upon a .CLOSE command, the handlers give back unused blocks, and rewrite the RIB.

The number of blocks in the Block List depends on the size of the buffer, which is determined at system generation by setting the buffer size. The larger the Block List, the faster will be output. Smaller Block Lists may give more efficient allocation of core and disk space. Smaller buffers save core. Further, the number of pre-allocated blocks may affect concurrently opened files on a disk that is tight for space. Thus, if the Block List is sixty entries long, and there are forty blocks left on the disk, a .ENTER to .DAT slot will pre-allocate all forty, leaving none for any subsequent .ENTER's to different .DAT slots.

IOPS 70 will occur when there are less than four free blocks on the disk when a handler tries to pre-allocate blocks.

6.3.6 Storage Allocation Tables (SAT's)

The disk handlers use a Storage Allocation Table, in order to distinguish between allocated and free blocks. If more than one physical block is required, the individual blocks are called Submaps.

Unlike DECtape, the Storage Allocation Table is never held in core. When the handlers wish to preallocate some blocks, they read in the required Submap, and write out the updated one.

Storage Allocation blocks use the following format:

WORD Ø	Total blocks on the disk
WORD 1	Number of blocks described by this Submap
WORD 2	Number of blocks occupied in this Submap
WORD 3	First word of the bit map (eighteen blocks per word)
.	
.	
.	
WORD 376	Pointer to previous Submap (or 777777)
WORD 377	Pointer to next Submap (or 777777)

The bit maps refer to blocks in numerical order. Thus, bit Ø of word three of a Submap will refer to block N, bit 1 will refer to block N+1, and so on. The block is free if the corresponding bit equals Ø. Starting and ending block numbers for all Submaps are retained in the handlers. The first Submap, however, starts with block zero.

6.3.7 Bad Allocation Tables (BAT's)

Occasionally, a particular block on the disk will not record data correctly. In such instances, the handlers should be prevented from using the bad blocks. Accordingly, PIP maintains a Bad Allocation Table. Whenever a user updates that table, PIP will set the appropriate bit in the Storage Allocation Table. The block is thus made unavailable. Refer to PIP manual (DEC-15-vWZB-DN13) for more information.

CHAPTER 7

WRITING NEW I/O DEVICE HANDLERS

This chapter contains information essential for writing new I/O device handlers to work in DOS.

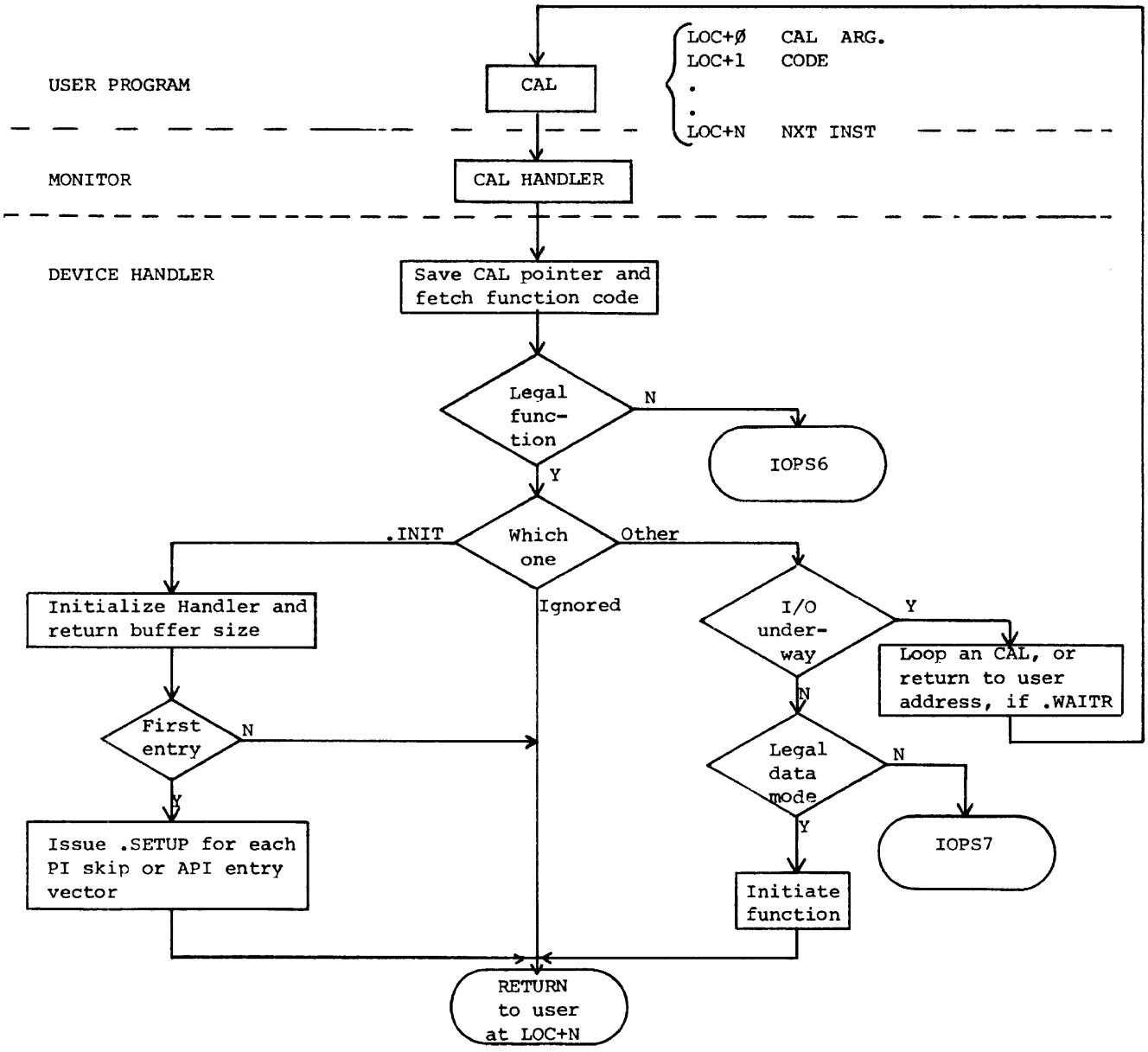
7.1 I/O DEVICE HANDLERS, AN INTRODUCTION

All communications between user programs and I/O device handlers are made via CAL instructions followed by an argument list. The CAL Handler in the Monitor (Figure 2-1) performs preliminary setups, checks the CAL calling sequence, and transfers control via a JMP* instruction to the entry point of the device handler. When the control transfer occurs (see Figures 7-1 and 7-2), the AC contains the address of the CAL in bits 3 through 17 and bits 0, 1, and 2 indicate the status of the Link, Bank/Page mode, and Memory Protect, respectively, at the time of the CAL. Note that the content of the AC at the time of the CAL is not preserved when control is returned to the user.

On machines that have an API, the execution of a CAL instruction automatically raises the priority to the highest software level (level 4). Control passes to the handler while it is still at level 4, allowing the handler to complete its non-reentrant procedures before debreaking (DBK) from level 4. This permits the handler to receive reentrant calls from software levels higher than the priority of the program that contained this call. Device handlers which do <u>not</u> contain reentrant procedures (including all handlers supplied with DOS) may avoid system failure caused by inadvertent reentries by remaining at level 4 until control is returned to the user.

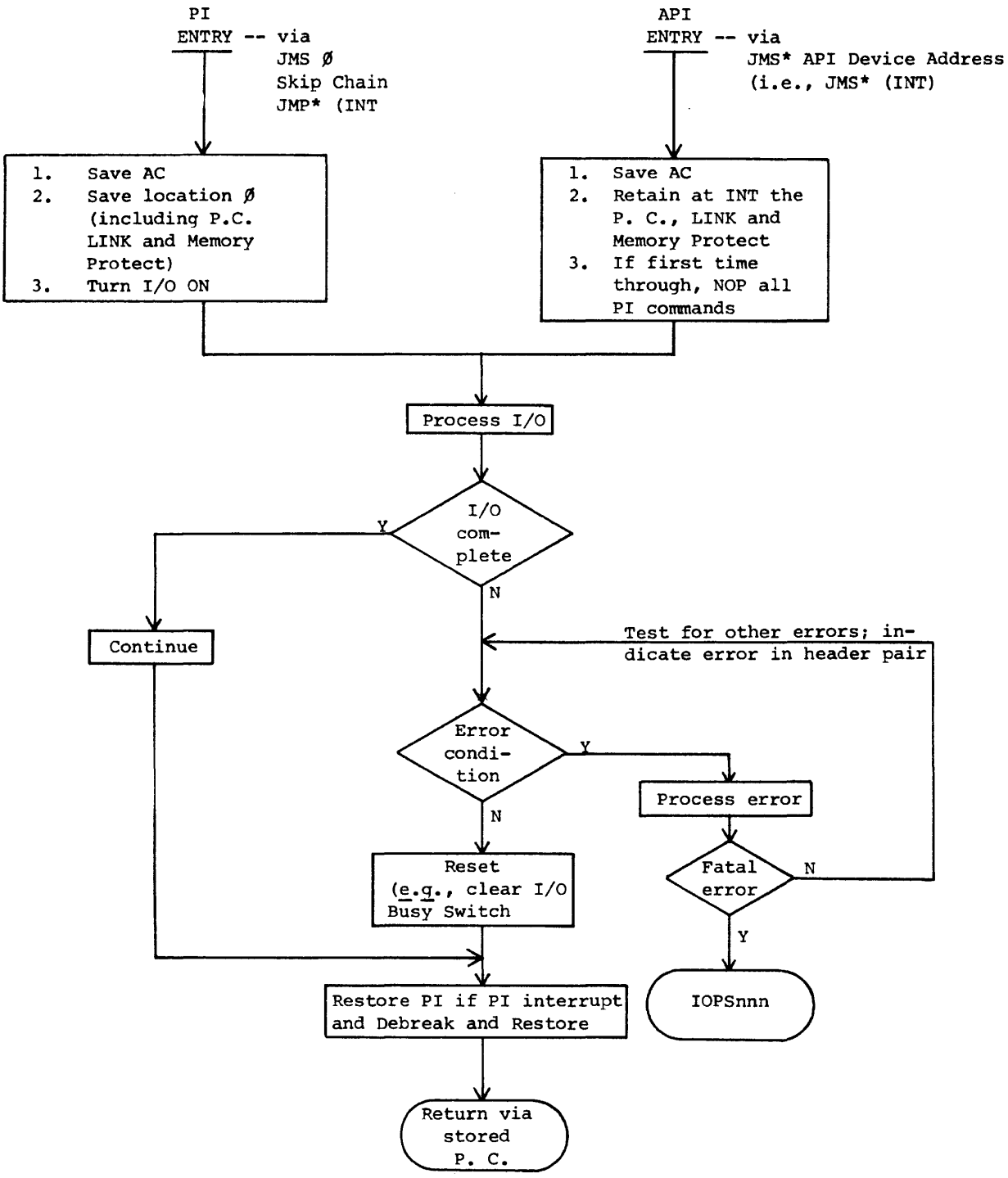
If the non-reentrant method is used, the debreak and restore (DBR) instruction should be executed just prior to the JMP* which returns control to the user, allowing debreak from level 4 and restoring the conditions of the Link, Bank/Page mode, and Memory Protect. Any IOT's issued at the CAL level (level 4 if API present, mainstream if no API) should be executed immediately before the

```
DBR
JMP*
```



CAL Entry to Device Handler

Figure 7-1



PI and API Entries to Device Handlers

Figure 7-2

exit sequence in order to ensure that the exit takes place before the interrupt from the issued IOT occurs.

The CAL instruction must not be used at any level (API or PIC) that might interrupt a CAL. A CAL at such a level will destroy the content of location 00020 for the previous CAL.

Care must also be taken when executing CALs at level 4. For example, a routine that is CALed from level 4 must know that if a debreak (DBR or DBK) is issued, control will return to the calling program (which had been at level 4) at a level lower than level 4.

7.1.1 Setting Up the Skip Chain and API (Hardware) Channel Registers

When the Monitor is loaded, the Program Interrupt (PI) Skip Chain and the Automatic Priority Interrupt (API) channels are set up to handle the TTY keyboard and printer and clock interrupts only. The Skip Chain contains the other skip IOT instructions, but indirect jumps to an error routine result if a skip occurs, as follows:

```
SKPDTA              /Skip if DECtape flag.
SKP
JMP* INT1           /INT1 contains error address.
SKP LPT             /Skip if line printer flag.
SKP
JMP* INT2           /INT2 contains error address.
SKPTTI              /Skip if teleprinter flag.
SKP
JMP TELINT          /To teleprinter interrupt handler
    .
    .
    .
```

All unused API channels, memory protect, memory parity, and powerfail, also contain JMP's to the error address.

When a device handler is called for the first time in a core load, it must call a Monitor routine (.SETUP) to set up its skip(s) in the Skip Chain, or its API channel, prior to performing any I/O functions.

The calling sequence is as follows:

```
CAL N           /N = API channel register 40 through 77 (see User's
                     Handbook Vol. 1, for standard channel assign-
                     ments),
                /0 if device not connected to API.
16              /.SETUP function code.
SKP IOT         /Skip IOT for this device.
DEVINT          /Address of interrupt handler.
 (normal return)
```

7.1.2 Handling the Interrupt

DEVINT exists in the device handler in the following format to allow for either API or PI interrupts.

```
ONLY1    LAC     (NOP       /LEAVE PI ALONE, WHEN API IS RUNNING
         DAC     DEVION     /THESE REGISTERS
         DAC     DEVIOF     /ARE AVAILABLE
         DAC     IGNRPI     /THIS IS ONCE ONLY CODE
         JMP     COMMON
DEVPIC   DAC     DEVAC      /SAVE AC
         LAC*    (Ø
         DAC     DEVOUT     /SAVE PC, LINK, ADDRESSING MODE AND
                            /MEMORY PROTECT
         JMP     COMMON
DEVINT   JMP     DEVPIC     /PI ENTRY
         DAC     DEVAC      /API ENTRY; SAVE AC
         LAC     DEVINT
         DAC     DEVOUT     /SAVE PC, LINK, ADDRESSING MODE AND
IGNRPI   JMP     ONLY1      /API IS OPERATING, SO LEAVE PI ALONE.
                            /PI INTERRUPTS ARE NOT POSSIBLE, BE-
                            /CAUSE .SETUP EFFECTIVELY NOP'S PI
                            /SKIPS.
COMMON   DEVCF              /CLEAR DEVICE DONE FLAG.
DEVION   ION                /PI ALLOWS INTERRUPTS; API DOES A NOP.
         .
         .
         .
DEVIOF   IOF                /API DOES NOP; PI TURNS IO OFF TO ENSURE
                            /NON-REENTRANCE AFTER ISSUING IOT'S.
         DEVIOT
         .
         .
         .
/DISMISS ROUTINE
         .
         .
         .
         LAC     DEVAC      /RESTORE AC.
DVSWCH   ION                /ION OR NOP.
         DBR                /DEBREAK AND RESTORE CONDITIONS
         JMP*    DEVOUT     /OF LINK, ADDRESSING MODE AND MEMORY
                            /PROTECT.
```

If the Index, Autoincrement, or EAE registers are used by the I/O device handler, it is necessary to save and restore them.

.SETUP allows either API or PI, but not both for a single device. The System Generator Manual gives the method for incorporating new handlers and associated Skip Chain entries into the Monitor.

7.2 API SOFTWARE LEVEL HANDLERS, An Introduction

The information presented in the following paragraphs assumes that the reader is familiar with the system input/output considerations described in the PDP-15 User's Handbook Vol. 1.

7.2.1 Setting Up API Software Level Channel Registers

When the Monitor is loaded, the API software-level channel registers (40 through 43) are initialized to

```
JMS*      .SCOM+12        /LEVEL 4
JMS*      .SCOM+13        /LEVEL 5
JMS*      .SCOM+14        /LEVEL 6
JMS*      .SCOM+15        /LEVEL 7
```

where .SCOM is equal to absolute location 000100 and .SCOM+12 through .SCOM+15 (000112 through 000115) each contains the address of an error routine.

Therefore, prior to requesting any interrupt at these software priority levels, the user must modify the contents of the .SCOM registers so that they point to the entry point of the user's software level handlers.

Example:

```
.SCOM=100
          LAC       (LV5INT          / set level 5 entry.
          DAC*      (.SCOM+13
           .
           .
           .
```

LV5INT exists in the user's area in the following format:

```
LV5INT    0                  /PC,LINK,BANK/PAGE MODE,MEM.PROT.
          DAC      SAV4AC    /SAVE AC
/SAVE INDEX, AUTOINCREMENT AND EAE REGISTERS, IF LEVEL 5
/ROUTINES USE THEM AND LOWER LEVEL ROUTINES ALSO USE THEM.
/SAVE MQ AND STEP COUNTER, IF SYSTEM HAS EAE AND IT IS USED
/AT DIFFERENT LEVELS.
          .
          .
/RESTORE SAVED REGISTERS.
DBR                          /DEBREAK FROM LEVEL 5 AND RESTORE
JMP*     LV5INT              /L, BANK/PAGE MODE, MEM. PROT.
```

7.2.2 Queueing

High priority/high data rate/short access routines cannot perform complex calculations based on unusual conditions without holding off further data input. To perform the calculations, the high priority program segment must initiate a lower priority (interruptable) segment to perform the calculations. Since many data handling routines would generally be requesting calculations, there will exist a queue of calculation jobs waiting to run at the software level. Each data handling routine must add its job request to the appropriate queue (taking care to raise the API priority level as high as the highest level that manipulates the queue before adding the request) and issue an interrupt request (ISA) at the corresponding software priority level. The general flow chart, Figure 7-4, depicts the structure of a software handler involved with queued requests.

Care must be taken about which routines are called when a software level request is honored; that is, if a called routine is "open" (started but not completed) at a lower level, it must be reentrant, or errors will results.

NOTE

The DOS hardware I/O device handlers do not contain reentrant procedures and must not be reentered from higher levels.

Resident handlers for Power Fail, Memory Parity, nonexistent memory violation, and Memory Protect violation have been incorporated into the system and effect an IOPS error message if the condition is detected. The user can, via a .SETUP, tie his own handler to these skip IOT or API channel registers.

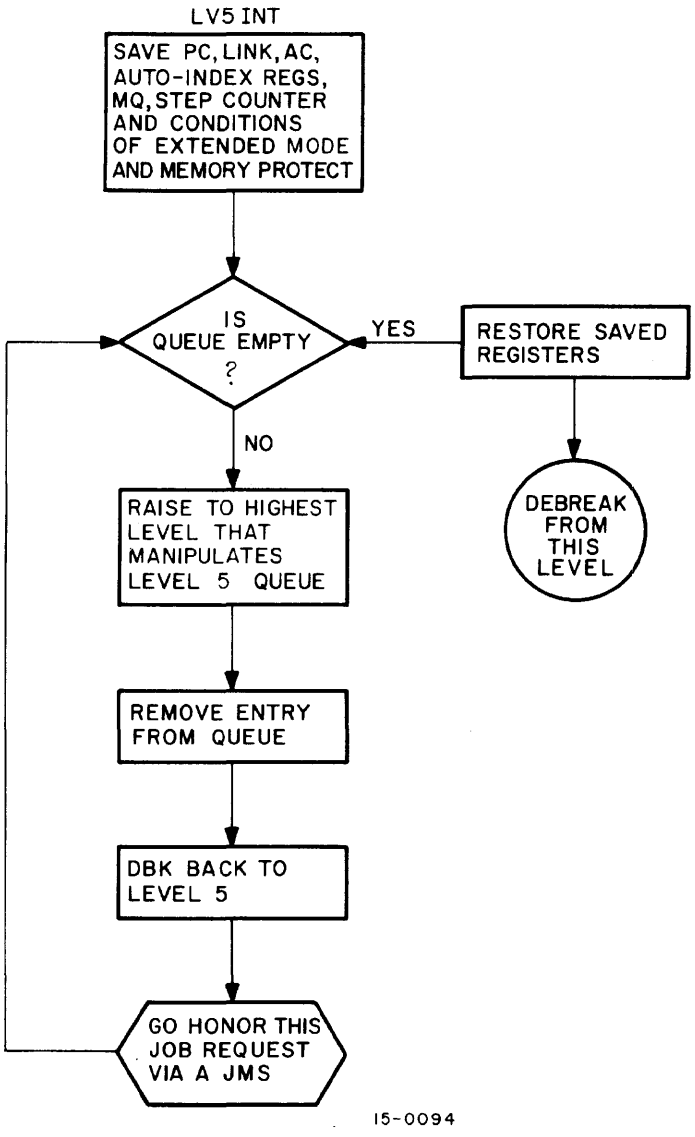


Figure 7-4

Structure of API Software Level Handler

7.3 WRITING SPECIAL I/O DEVICE HANDLERS

This section contains information prepared specifically to aid those users who plan to write their own special I/O device handlers for DOS.

DOS is designed to enable users to incorporate their own device handlers. Precautions should be taken when writing the handler however, to ensure compatibility with the Monitor.

Here is a summary of handler operation. The handler is entered via a JMP* from the Monitor as a result of a CAL instruction. The contents of the AC contain the address of the CAL in bits 3 through 17. Bit 0 contains the Link, bit 1 contains the Bank/Page Mode status, and bit 2 contains the Memory Protect status. The previous contents of the AC and Link are lost.

In order to show the steps required in writing an I/O device handler, a complete handler (Example B) was developed with the aid of a skeleton handler (Example A). In addition, Appendices A and B are complete flowcharts of the DTA and DKA handlers. The skeleton handler is a non-reentrant type (discussed briefly at the beginning of this chapter) and uses the Debreak and Restore Instruction (DBR) to leave the handler at software priority level 4 or at a hardware level for interrupt servicing (if API), and restore the status of the Link, Bank/Page Mode, and Memory Protect. Example A is referenced by part numbers to illustrate the development of Example B, a finished Analog-to-Digital Converter (ADC) I/O Handler. The ADC handler shown in Example B was written for the Type AFØ1B Analog to Digital Converter. This handler is used to read data from the ADC and store it in the user's I/O buffer.

The reader, while looking at the skeleton of a specialized handler as shown in Example A, should make the following decisions about his own handler. (The decisions made in this case are in reference to developing the ADC handler):

a. Services that are required of the handler (flags, receiving or sending of data, etc.) - By looking at the ADC IOT's shown in the Reference Manual, it can be seen that there are three IOT instructions to be implemented. These instructions are: Skip if Converter Flag Set, Select and Convert, and Read Converter Buffer.

The only service the ADC handler performs is that of receiving data and storing it in user specified areas. This handler will have a standard 256-word buffer.

b. Data Modes used (for example, IOPS ASCII, etc.) - Since there is only one format of input from the Type AF01B ADC, mode specification is unnecessary in Example C.

c. Which I/O macros are needed for the handler's specific use; that is, .INIT, .CLOSE, .READ, etc. For an ADC, the user would be concerned with four of the macros.

(1) .INIT would be used to set up the associated API channel register or the interrupt skip IOT sequence in the Program Interrupt Skip Chain. This is done by a CAL (N) as shown in Part III of Example A, where (N) is the channel address.

(2) .READ is used to transfer data from the ADC. When the .READ macro is issued, the ADC handler will initiate reading of the specified number of data words and then return control to the user. The analog input data received is in its raw form. It is up to the programmer to convert the data to a usable format.

(3) .WAIT detects the availability of the user's buffer area and ensures that the I/O transfer is completed. It would be used to ensure a complete transfer before processing the requested data.

(4) .WAITR detects the availability of the user's buffer area as in (3) above. If the buffer is not available, control is returned to a user specified address, which allows other processing to continue.

d. Implementation of the API or PIC interrupt service routine - Example A shows an API or PIC interrupt service routine that handles interrupts, processes the data and initiates new data requests to fully satisfy the .READ macro request. Note that the routines in Example A will operate with or without API. Example B uses the routines exactly as they are shown in Example A.

During the actual writing of Example B, consideration was given to the implementation of the I/O macros in the new handler in one of the following ways:

(1) Execute the function in a manner appropriate to the given device as discussed in(c). .INIT, .READ, .WAIT, and .WAITR were implemented into the ADC handler (Example B) under the subroutine names ADINIT, ADREAD, ADWAIT (.WAIT and .WAITR).

Wait for completion of previous I/O. (Example B shows the setting of the ADUND switch in the ADREAD subroutine to indicate I/O underway.)

(2) Ignore the function if meaningless to the device. See Example B (.FSTAT results in JMP ADIGN2) in the dispatch table DSPCH. For ignored macros, the return address must be incremented in some cases, depending upon the number of arguments following the CAL (see Chapter 3).

(3) Issue an error message in the case where it is not possible to perform the I/O function - (An example would be trying to execute a .ENTER on the paper tape reader.) In Example B, the handler jumps to DVERR6 which returns to the Monitor with a standard error code in the AC.

After the handler has been written and assembled, the Monitor must then be modified to recognize the new handler. This is accomplished by the use of the System Generator Program (DOSGEN) described in the DEC-15-YWZB-DN12 manual.

When the system generation is complete, the PIP program (refer to DEC-15-YWZA-DN13) must be used to add the new handler to the IOS UFD. At this time, the user is ready to use his specialized device handler in the DOS-15 system.

7.3.1 Discussion of Example A by Parts

Part 1 Stores CAL pointer and argument pointer, and picks up function code from argument string.

Part 2 By getting proper function code in Part 1 and adding a JMP DSPCH, the CAL function is dispatched to the proper routine.

Part 3 This is the .SETUP CAL used to set up the PI skip chain or the API channel register.

Part 4 Shows the API and PI handlers. It is suggested these be used as shown.

Part 5 This area reserved for processing interrupt and performing any additional I/O.

Part 6 Interrupt dismiss routine.

Part 7 Increments argument pointer in bypassing arguments of ignored macro CAL's.

7.3.2 Example A, Skeleton I/O Device Handler

```
/CAL ENTRY ROUTINE
        .GLOBL DEV,                 /MUST BE OF FORM AAA,
,MED=3                              /,MED (MONITOR ERROR DIAGNOSTIC)
DEV,    DAC     DVCALP              /SAVE CAL POINTER
        DAC     DVARGP              /AND ARGUMENT POINTER
        ISZ     DVARGP              /POINTS TO FUNCTION CODE
        LAC*    DVARGP              /GET CODE
        AND     (77777              /REMOVE UNIT NO IF APPLICABLE
        ISZ     DVARGP              /POINTS TO CAL+2
        TAD     (JMP DSPCH
        DAC     DSPCH               /DISPATCH WITH
DSPCH   XX                          /MODIFIED JUMP
        JMP     DVINIT              /1 = ,INIT
        JMP     DVFSAT              /2 = ,FSTAT, ,DLETE, ,RENAM
        JMP     DVSEEK              /3 = ,SEEK
        JMP     DVENTR              /4 = ,ENTER
        JMP     DVCLER              /5 = ,CLEAR
        JMP     DVCLOS              /6 = ,CLOSE
        JMP     DVMTAP              /7 = ,MTAPE
        JMP     DVREAD              /10 = ,READ
        JMP     DVWRTE              /11 = ,WRITE
        JMP     DVWAIT              /12 = ,WAIT
        JMP     DVTRAN              /13 = ,TRAN

/ILLEGAL FUNCTIONS IN ABOVE TABLE CODED AS:
/       JMP     DVERR6

/FUNCTION CODE ERROR
DVERR6  LAW     6                   /ERROR CODE 6
        JMP*    (,MED+1             /TO MONITOR

/DATA MODE ERROR
DVERR7  LAW     7                   /ERROR CODE 7
        JMP*    (,MED+1             /TO MONITOR

/DEVICE NOT READY
DVERR4  LAC     (RETURN             /RETURN (ADDRESS IN HANDLER)
                                    /TO RETURN TO WHEN NOT READY
                                    /CONDITION HAS BEEN REMOVED

        DAC*    (,MED
        LAC     (4                  /ERROR CODE 4
        JMP*    (,MED+1             /TO MONITOR

/I/O UNDERWAY LOOP
DVBUSY  DBR                         /BREAK FROM LEVEL 4
        JMP*    DVCALP              /LOOP ON CAL

/NORMAL RETURN FROM CAL
DVCK    DBR                         /BREAK FROM LEVLE 4
        JMP*    DVARGP              /RETURN AFTER CAL AND
                                    /ARGUMENT STRING

/THE DVINIT ROUTINE MUST INCLUDE
/A ,SETUP CALLING SEQUENCE FOR
```

```
/EACH FLAG CONNECTED TO API
/AND/OR PI A(AT SGEN TIME),
/THE SETUP CALLING SEQUENCE IS:

DVINIT  CAL     N               /N = API CHANNEL REGISTER
                                /(40 -77), N = 0 IF NOT CONNECTED
                                /TO API

        16                      /IOPS FUNCTION CODE
        SKPIOT                  /SKIP IOT TO TEST THE FLAG
        DBVINT                  /ADDRESS OF INTERRUPT
                                /HANDLER (PI OR API)

/THIS SPACE MAY BE USED FOR I/O SUBROUTINES

/INTERRUPT HANDLER FOR API OR PI
ONLY1   LAC     (NOP
        DAC     DEVION
        DAC     DEVIOF
        DAC     DVSWCH
        DAC     IGNRPI
        JMP     COMMON
DVPIC   DAC     DEVAC           /SAVE AC
        LAC*    (0              /SAVE:PC, LINK, BANK/PAGE MODE
        DAC     DVOUT           /AND MEMORY PROTECT
        JMP     COMMON
DVINT   JMP     DEVPIC          /PI ENTRY
        DAC     DEVAC           /API ENTRY: SAVE AC
        LAC     DEVINT          /SAVE: PC, LINK, BANK/PAGE MODE
        DAC     DEVOUT          /MEMORY PROTECT
IGNRPI  JMP     ONLY1           /LEAVE PI ALONE
COMMON  DEVCF                   /ENABLE PI OR NOP
DEVION  ION                     /ENABLE PI OR NOP

/THIS AI  THE AREA DEVOTED TO PROCESSING INTERRUPT AND
/PERFORMING ANY ADDITIONAL I/O DESIRED,

DEVIOF  IOF                     /DISABLE PI OR NOP
        DEVIOT                  /DISMISSAL BEFORE INTERRUPT
                                /FROM THIS IOT OCCURS

/INTERRUPT HANDLER DISMISS ROUTE
DVDISM  LAC     DEVAC           /RESTORE AC
DVSWCH  ION                     /ION OR NOP
        DBR                     /DEBREAK AND RESTORE
        JMP*    DEVOUT          /LINK, BANK/PAGE MODE, MEMORY
                                /PROTECT

/IF THE HANDLER USES THE AUTOINCREMENT , INDEX
/OR EAE REGISTERS, THEIR CONTENTS
/SHOULD BE SAVED AND RESTORED, FUNCTIONS
/POSSIBLY IGNORED SHOULD CONTAIN
/PROPER INDEXING TO BYPASS
/CAL ARGUMENT STRING
/
/CODE TO BYPASS IGNORED FUNCTIONS
/



DVIGN2  ISZ     DVARGP          /BYPASS FILE POINTER
        JMP     DVCK
```

7.3.3 Example B. Special Device Handler for AF01B A/D Converter

```
PAGE   1      R       001

    1                                 /ADC HANDLER
    2                                 /
    3                 701301 A        ADSF=701301       /SKIP IF CONVERSION FLAG IS SET
    4                 701304 A        ADSC=701304       /SELECT AND CONVERT (ADC FLAG IS CLEARED
    5                                                   /AND A CONVERSION IS INITIALISED)
    6                 701312 A        ADRB=701312       /READ CONVERTER BUFFER INTO AC AND CLEAR FLAG
    7                                 /
    8                                       .GLOBL  ADC,
    9                 440000 A        IDX=ISZ
   10                 000003 A        .MED=3            /MED (MONITOR ERROR DIAGNOSTIC)
   11                                 /
   12          00000 R 040150 R       ADC,    DAC       ADCALP  /SAVE CAL POINTER
   13          00001 R 040151 R               DAC       ADARGP  /AND ARGUMENT POINTER
   14          00002 R 440151 R               IDX       ADARGP  /POINTS TO FUNCTION CODE
   15          00003 R 220151 R               LAC*      ADARGP  /GET CODE
   16          00004 R 440151 R               IDX       ADARGP  /POINTS TO CAL + 2
   17          00005 R 340154 R               TAD       (JMP DSPCH
   18          00006 R 040007 R               DAC       DSPCH   /DISPATCH WITH
   19          00007 R 740040 A       DSPCH   XX                /MODIFIED JUMP
   20          00010 R 600027 R               JMP       ADINIT  /1=,INIT
   21          00011 R 600074 R               JMP       ADIGN2  /2=,FSTAT,,DLETE,,RENAM
   22          00012 R 600074 R               JMP       ADIGN2  /3=,SEEK
   23          00013 R 600023 R               JMP       ADERR6  /4=,ENTER
   24          00014 R 600023 R               JMP       ADERR6  /5=,CLEAR
   25          00015 R 600075 R               JMP       ADIGN1  /6=,CLOSE
   26          00016 R 600075 R               JMP       ADIGN1  /7=,MTAPE
   27          00017 R 600051 R               JMP       ADREAD  /10=,READ
   28          00020 R 600023 R               JMP       ADERR6  /11=,WRITE
   29          00021 R 600044 R               JMP       ADWAIT  /12=,WAIT
   30          00022 R 600023 R               JMP       ADERR6  /13=,TRAN
   31                                 /
   32                                 /ILLEGAL FUNCTIONS IN ABOVE TABLE CODED AS
   33                                 /       JMP       ADERR6
   34                                         .EJECT
```

```
                                /
                                /FUNCTION CODE ERROR
                                /
00023 R 760006 A    ADERR6  LAW     6       /ERROR CODE 6
00024 R 620155 R            JMP*    (,MED+1 /TO MONITOR
                            /DATA MODE ERROR
00025 R 760007 A    ADERR7  LAW     7       /ERROR CODE 7
00026 R 620155 R            JMP*    (,MED+1 /TO MONITOR
                            /THE ADINT ROUTINE MUST INCLUDE A ,SETUP
                            /FOR EACH FLAG ASSOCIATED WITH THE DEVICE
                            /
00027 R 440151 R    ADINIT  IDX     ADARGP  /IDX TO RETURN BUFF SIZE
                            .DEC
00030 R 200156 R            LAC     (256    /STANDARD BUFFER SIZE (DECIMAL)
                            .OCT
00031 R 060151 R            DAC*    ADARGP  /RETURN IT TO USER
00032 R 440151 R            IDX     ADARGP
00033 R 000057 A    ADCMOD  CAL     57      /57=API CHANNEL
00034 R 000016 A    ADCKSM  16              /,SETUP IOPS FUNCTION CODE
00035 R 701301 A    ADCBP   ADSF            /ADC SKIP IOT
00036 R 000111 R    ADLBHP  ADCINT          /ADDR. OF INTERRUPT
00037 R 200041 R    ADUND   LAC     .+2     /SET-UP ONCE ONLY
00040 R 040033 R    ADWC    DAC     ADCMOD  /SKIP SET-UP CODE IF MORE
00041 R 600042 R    ADWPCT  JMP     ADSTOP  /,INITS ARE DONE
                            /
                            /STOP ADC ROUTINE CLEARS I/O UNDERWAY SWITCH
                            /
00042 R 140037 R    ADSTOP  DZM     ADUND
00043 R 600075 R            JMP     ADIGN1  /RETURN
                            /
                            /THE PREVIOUS TAGS IN THE CAL AREA ARE USED FOR
                            /STORAGE DURING THE ACTUAL ,READ FUNCTION
                            /
                            /ADCKSM IS FOR STORING THE CHECKSUM
                            /ADCBP IS THE CURRENT BUFFER POINTER
                            /ADLBHP IS THE LINE BUFFER HEADER POINTER
                            /ADUND IS FOR DEVICE UNDERWAY SWITCH
                            /ADWC IS USED AS THE COUNTER
                            /ADWPCT IS USED TO STORE CURRENT WORD COUNT
                            /
                                    .EJECT
```

```
                00044 R 200037 R    ADWAIT  LAC     ADUND
                00045 R 741200 A            SNA
                00046 R 600075 R            JMP     ADIGN1
                                    /I/O UNDERWAY LOOP
                00047 R 703344 A    ADBUSY  DBR
                00050 R 620150 R            JMP*    ADCALP
                                    /
                                    /
                00051 R 200037 R    ADREAD  LAC     ADUND   /CHECK TO SEE IF I/O IS UNDERWAY
                00052 R 740201 A            SZA!CMA         /IF NOT SET IT WITH -1
                00053 R 600047 R            JMP     ADBUSY  /IT WAS SET,GO BACK TO CAL
                00054 R 040037 R            DAC     ADUND   /SET IT
                00055 R 220150 R            LAC*    ADCALP  /LOOK AT MODE
                00056 R 500157 R            AND     (7000   /BITS 6-8 ONLY
                00057 R 740200 A            SZA             /IOPS BINARY?
                00060 R 600025 R            JMP     ADERR7  /NO, ERROR
                00061 R 220151 R            LAC*    ADARGP  /GET LINE BUFFER HEADER POINTER
                00062 R 040035 R            DAC     ADCBP   /STORE IT
                00063 R 040036 R            DAC     ADLBHP  /ALSO STORE IT FOR LATER HEADER
                00064 R 440151 R            IDX     ADARGP  /INCREMENT ARG. POINTER
                00065 R 220151 R            LAC*    ADARGP  /GET -L,B,W,C(2'S COMP)
                00066 R 040040 R            DAC     ADWC    /STORE IT IN WORD COUNTER
                00067 R 140041 R            DZM     ADWPCT  /ZERO WORD COUNT REG.
                00070 R 140034 R            DZM     ADCKSM  /ZERO CHECKSUM REG.
                00071 R 440035 R            IDX     ADCBP   /GET PAST HEADER PAIR
                00072 R 440035 R            IDX     ADCBP   /NOW POINTING AT BEGINNING OF
                                                            /BUFFER
                00073 R 701304 A            ADSC            /START UP DEVICE
                00074 R 440151 R    ADIGN2  IDX     ADARGP  /INCR, FOR EXIT
                00075 R 703344 A    ADIGN1  DBR             /BREAK FROM LEVEL 4
                00076 R 620151 R            JMP*    ADARGP  /RETURN AFTER CAL
                                    /INTERRUPT HANDLER FOR API OR PIC
                                    /
                00077 R 200160 R    ONLY1   LAC     (NOP
                00100 R 040117 R            DAC     ADCION
                00101 R 040142 R            DAC     ADCONT
                00102 R 040145 R            DAC     ADSWCH
                00103 R 040115 R            DAC     IGNRPI
                00104 R 600116 R            JMP     COMMON
                00105 R 040153 R    ADCPIC  DAC     ADCAC   /SAVE AC
                00106 R 220161 R            LAC*    (0)     /SAVE PC,LINK,EX, MODE
                00107 R 040152 R            DAC     ADCOUT  /MEM,PROT,
                                            .EJECT
```

```
119          00110 R 600116 R                JMP     COMMON
120          00111 R 600105 R    ADCINT      JMP     ADCPIC  /PIC ENTRY
121          00112 R 040153 R                DAC     ADCAC   /API ENTRY,SAVE AC
122          00113 R 200111 R                LAC     ADCINT  /SAVE PC,LINK,EX,MODE
123          00114 R 040152 R                DAC     ADCOUT  /MEM,PROT
124          00115 R 600077 R    IGNRPI      JMP     ONLY1
125          00116 R 701312 A    COMMON      ADRB            /READ CONVERTER BUFFER
126          00117 R 700042 A    ADCION      ION             /ENABLE PIC FOR OTHER DEVICES
127          00120 R 060035 R                DAC*    ADCBP   /STORE DATA IN USER BUFFER
128          00121 R 440035 R                IDX     ADCBP   /INC. BUFFER POINTER
129          00122 R 440041 R                IDX     ADWPCT  /INC. WORD PAIR COUNTER
130          00123 R 340034 R                TAD     ADCKSM  /ADD CHECKSUM
131          00124 R 040034 R                DAC     ADCKSM  /STORE IT
132          00125 R 440040 R                ISZ     ADWC    /IS I/O COMPLETE
133          00126 R 600142 R                JMP     ADCONT  /NO KEEP GOING
134          00127 R 200041 R                LAC     ADWPCT  /YES,COMPUTE WORD COUNT PAIR
135          00130 R 740030 A                IAC             /MAY BE ODD
136          00131 R 742030 A                SWHA            /TO TOP HALF
137          00132 R 740020 A                RAR             /MAKE WD. PRS.
138          00133 R 500162 R                AND     (377000 /8 BITS ONLY
139          00134 R 060036 R                DAC*    ADLBHP  /STORE IN HEADER #1
140          00135 R 440036 R                IDX     ADLBHP  /INC. TO STORE CKSUM
141          00136 R 340034 R                TAD     ADCKSM  /ADD WORD PAIR COUNT
142          00137 R 060036 R                DAC*    ADLBHP  /STORE IN HEADER #2
143          00140 R 140037 R                DZM     ADUND   /CLEAR DEVICE UNDERWAY
144          00141 R 600144 R                JMP     ADDISM  /EXIT
145          00142 R 700002 A    ADCONT      IOF             /DISABLE  PIC OR NOP
146          00143 R 701304 A                ADSC            /BEFORE INTERRUPT FROM THIS IOT OCCURS
147                                  /INTERRUPT HANDLER DISMISS RTE
148                                  /
149          00144 R 200153 R    ADDISM      LAC     ADCAC   /RESTORE AC
150                                          .EJECT
```

```
PAGE    5      R       001

 151          00145 R 700042 A     ADSWCH  ION                 /ION OR NOP
 152          00146 R 703344 A             DBR                 /DEBREAK AND RESTORE
 153          00147 R 620152 R             JMP*      ADCOUT    /LINK,EX,MODE,MEM,PROT
 154          00150 R 000000 A     ADCALP  0                   /AUD CAL POINTER
 155          00151 R 000000 A     ADARGP  0                   /AUD ARGUMENT POINTER
 156          00152 R 000000 A     ADCOUT  0                   /PC,L,FM,MP
 157          00153 R 000000 A     ADCAC   0                   /AC SAVED HERE
 158                              /
 159                  000000 A             ,END
              00154 R 600007 R *L
              00155 R 000004 A *L
              00156 R 000400 A *L
              00157 R 007000 A *L
              00160 R 740000 A *L
              00161 R 000000 A *L
              00162 R 377000 A *L
                  SIZE=00163      NO ERROR LINES
```

```
ADARGP  00151   13    14    15    16    46    50    51    92    95
                96   104   106   155*
ADBUSY  00247   80*   86
ADCAC   00153  115   121   149   157*
ADCALP  00150   12    81    88   154*
ADCBP   00735   54*   93   100   101   127   128
ADCINT  00111   55   120*  122
ADCION  00117  119   126*
ADCKSM  00234   53*   99   130   131   141
ADCMOD  00033   52*   57
ADCONT  00142  111   133   145*
ADCOUT  00152  117   123   153   156*
ADCPIC  00105  115*  120
ADC.    00000    8    12*
ADDISM  00144  144   149*
ADERR6  00023   23    24    28    30    38*
ADERR7  00025   41*   91
ADIGN1  00075   25    26    63    78   105*
ADIGN2  00074   21    22   104*
ADINIT  00027   20    46*
ADLBHP  00036   55*   94   139   140   142
ADRB    701312    6*  125
ADREAD  00051   27    84*
ADSC    701304    4*  103   146
ADSF    701301    3*   54
ADSTOP  00042   58    62*
ADSWCH  00145  112   151*
ADUND   00037   56*   62    76    84    87   143
ADWAIT  00044   29    76*
ADWC    00040   57*   97   132
ADWPCT  00041   58*   98   129   134
COMMON  00116  114   119   125*
DSPCH   00007   17    18    19*
IDX     440000    9*   14    16    46    51    95   100   101   104
                128   129   140
IGNRPI  00115  113   124*
ONLY1   00077  109*  124
.MED    000003   10*   39    42
```

CHAPTER 8

BOSS-15

BOSS enables DOS users with a card reader and a line printer to run jobs sequentially, with a minimum of operator intervention. BOSS supports a subset of the DOS system programs, and adds a line editor, its own resident and nonresident routines (called Resident BOSS and Nonresident BOSS), and the Procedure Files. Paragraph 8.1 describes Procedure Files. Figure 8-1 shows which monitor supports each system program.

The DOS programs run by BOSS are identical to those run by DOS. Exceptions are the Resident and Nonresident Monitors, which are explained later. BOSS expands the information on Control Cards into a series of commands in the format expected by the DOS system programs. Nonresident BOSS does this command expansion, and stores the expanded commands in a disk file, the Run Time File (RTF). Since DOS programs expect to communicate with an operator at a teleprinter, BOSS feeds the expanded commands to the programs via .DAT slots assigned to TTA. In BOSS mode, therefore, BOSS attaches .DAT-2 to the Run Time File, and directs most teleprinter output to the Line Printer. Programs can force I/O to the teleprinter by setting bit 4 in .SCOM+52, and proceding with macros directed to TTA.

Whenever bit Ø of .SCOM+52 is set, the System Loader Interface attaches the Resident BOSS code to the Resident Monitor. The main purposes of Resident BOSS are to (1) ensure that BOSS will retain control of the teleprinter, (2) feed commands to programs via the Run Time File, (3) properly route internal Monitor commands, such as .EXIT, .GET and .PUT, and (4) direct teleprinter output to the Line Printer. Figure 8-2 illustrates the connections between the DOS Resident Monitor and the BOSS Resident Monitor that accomplish these changes. Figure 8-3, the flowchart for Resident BOSS, further describes Resident BOSS.

Resident BOSS communicates with Nonresident BOSS by TRANing information to and from the first block of Nonresident BOSS. Nonresident BOSS gains control on all error conditions, such as IOPS, operator abort, Time Estimate exceeded, and after a BOSS15 command. Figure 8-4 is a flowchart of Nonresident BOSS.

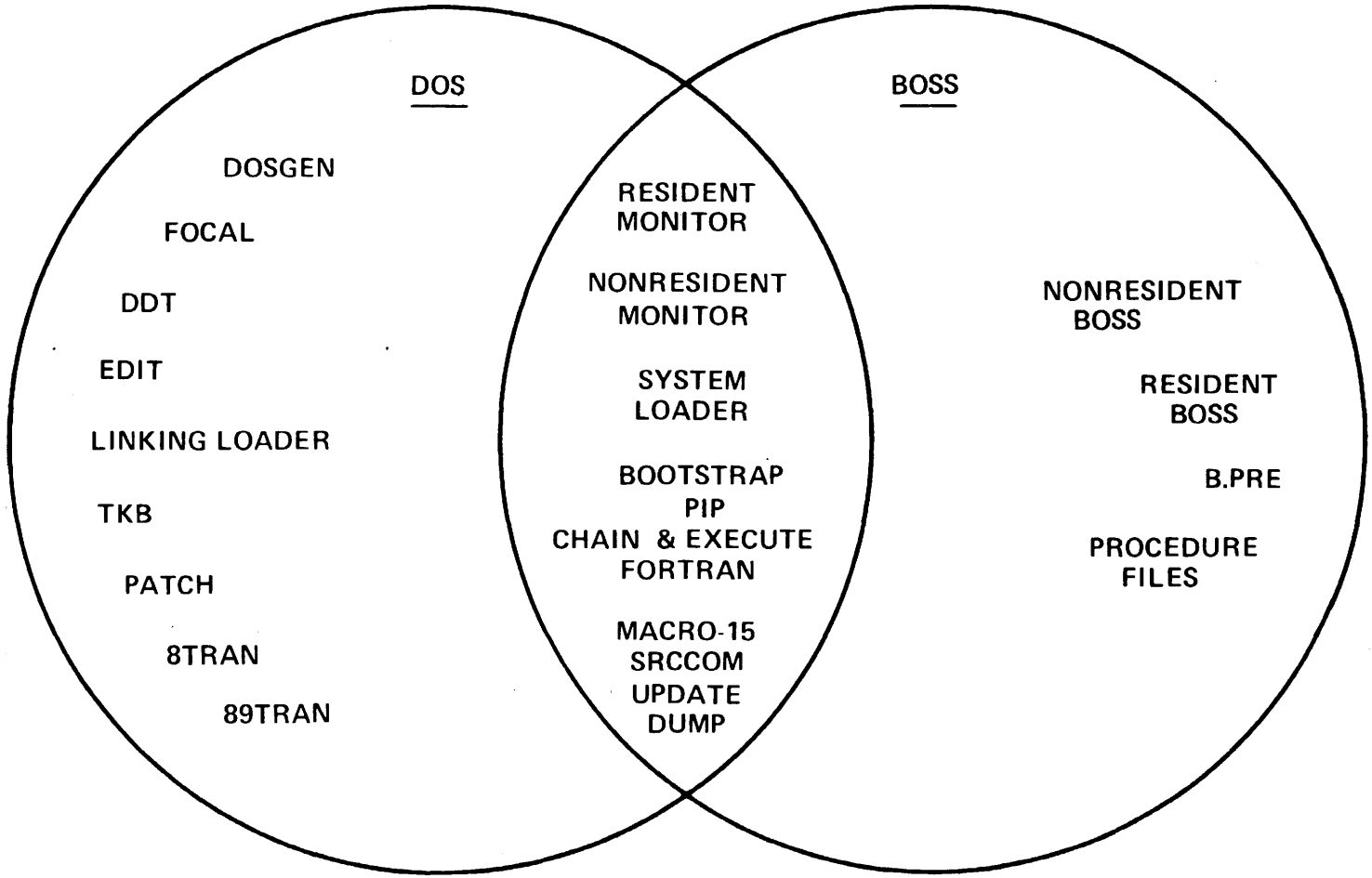


Figure 8-1, BOSS/DOS Intersection

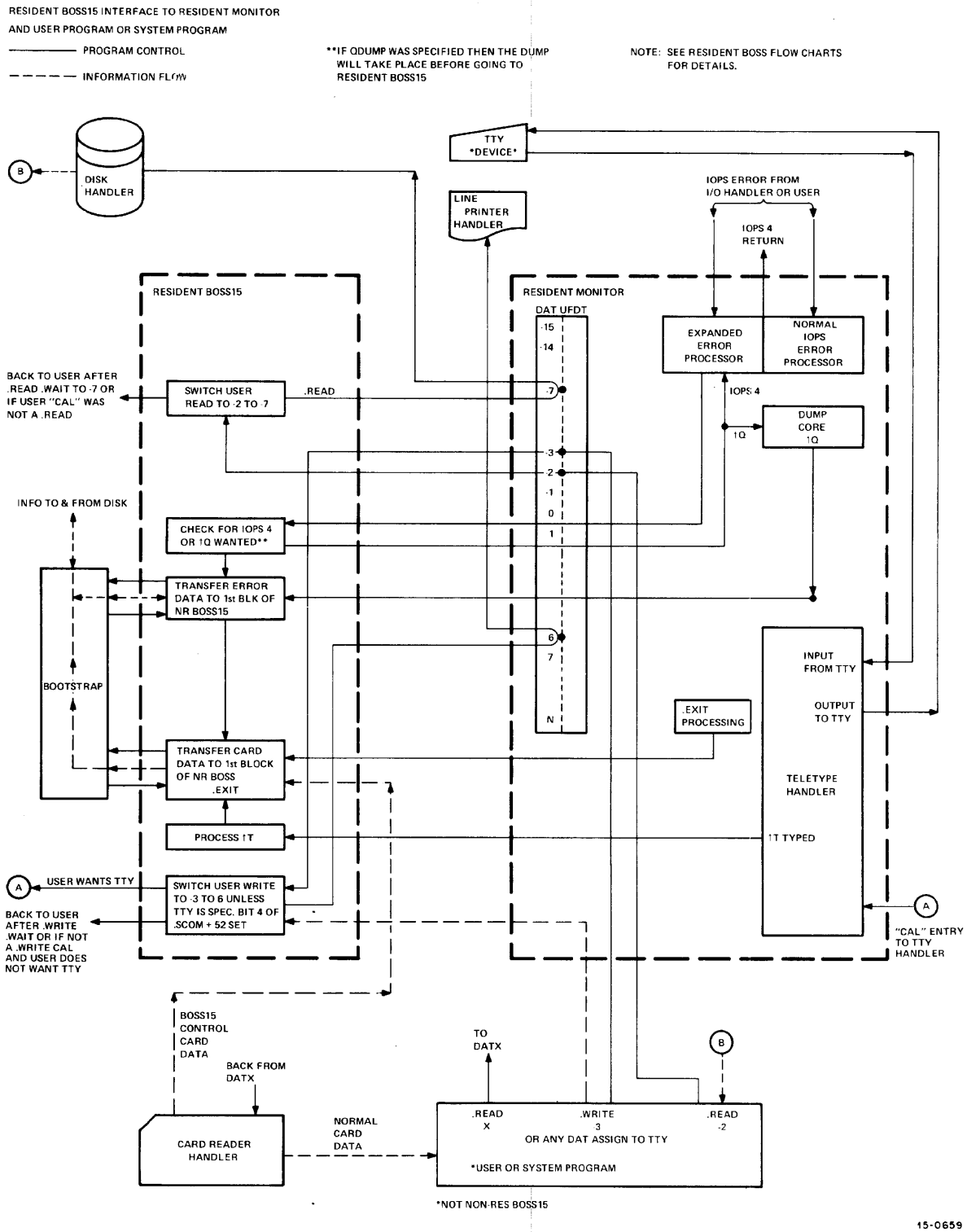
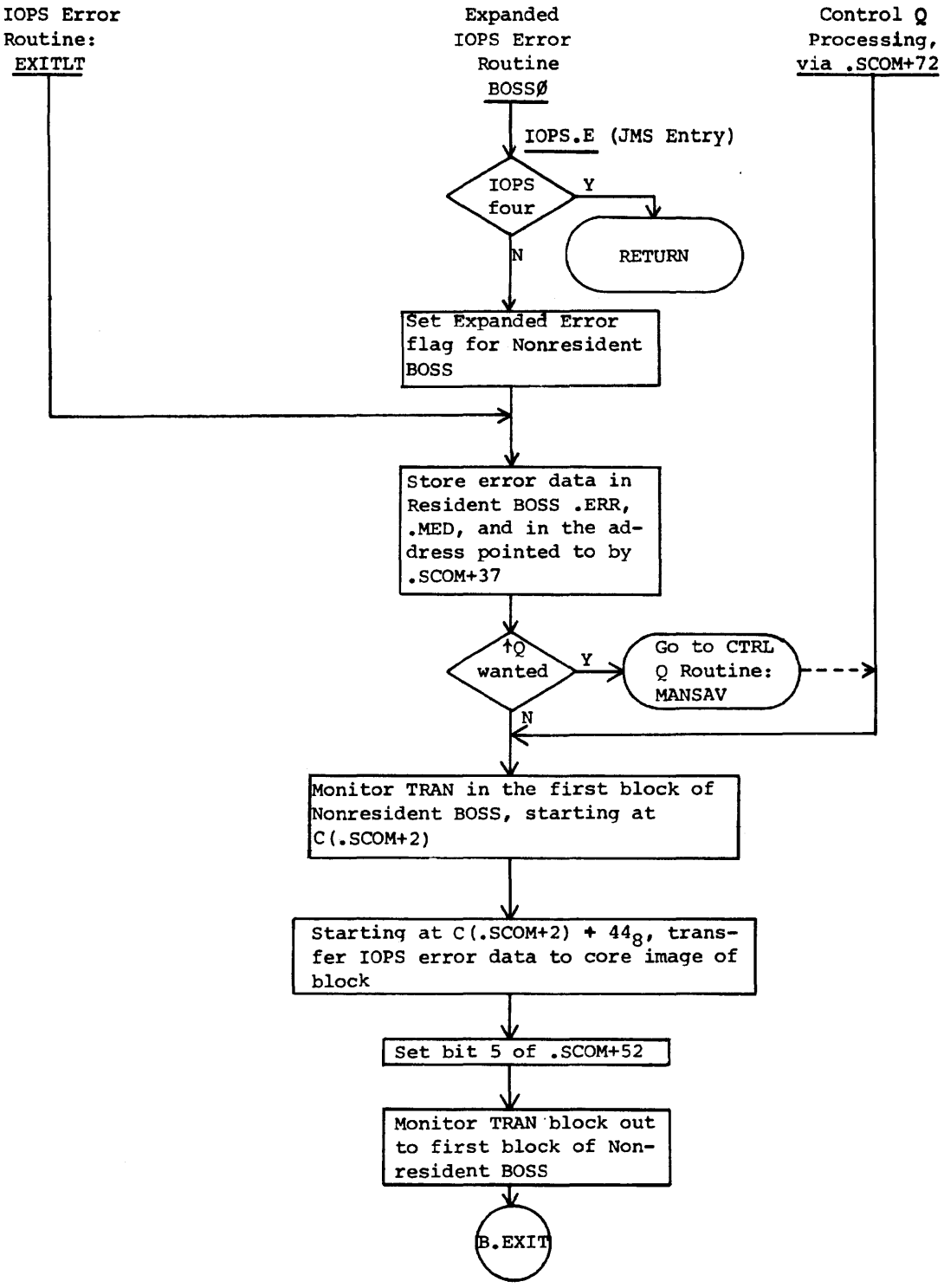

RESIDENT BOSS15 INTERFACE TO RESIDENT MONITOR
AND USER PROGRAM OR SYSTEM PROGRAM
PROGRAM CONTROL
INFORMATION FLOW
**IF QDUMP WAS SPECIFIED THEN THE DUMP WILL TAKE PLACE BEFORE GOING TO RESIDENT BOSS15
NOTE: SEE RESIDENT BOSS FLOW CHARTS FOR DETAILS.
B
DISK HANDLER
TTY *DEVICE*
LINE PRINTER HANDLER
IOPS ERROR FROM I/O HANDLER OR USER
IOPS 4 RETURN
RESIDENT BOSS15
RESIDENT MONITOR
DAT UFDT
-15
-14
-7
-3
-2
-1
0
1
6
7
N
EXPANDED ERROR PROCESSOR
NORMAL IOPS ERROR PROCESSOR
IOPS 4
DUMP CORE ^Q
^Q
BACK TO USER AFTER .READ .WAIT TO -7 OR IF USER "CAL" WAS NOT A .READ
SWITCH USER READ TO -2 TO -7
.READ
INFO TO & FROM DISK
CHECK FOR IOPS 4 OR ^Q WANTED**
TRANSFER ERROR DATA TO 1st BLK OF NR BOSS15
BOOTSTRAP
INPUT FROM TTY
OUTPUT TO TTY
.EXIT PROCESSING
TELETYPE HANDLER
TRANSFER CARD DATA TO 1st BLOCK OF NR BOSS .EXIT
PROCESS ^T
^T TYPED
A
USER WANTS TTY
SWITCH USER WRITE TO -3 TO 6 UNLESS TTY IS SPEC. BIT 4 OF .SCOM + 52 SET
BACK TO USER AFTER .WRITE .WAIT OR IF NOT A .WRITE CAL AND USER DOES NOT WANT TTY
"CAL" ENTRY TO TTY HANDLER
BOSS15 CONTROL CARD DATA
BACK FROM DATX
TO DATX
B
CARD READER HANDLER
NORMAL CARD DATA
.READ X
.WRITE -3
.READ -2
OR ANY DAT ASSIGN TO TTY
*USER OR SYSTEM PROGRAM
*NOT NON-RES BOSS15
15-0659

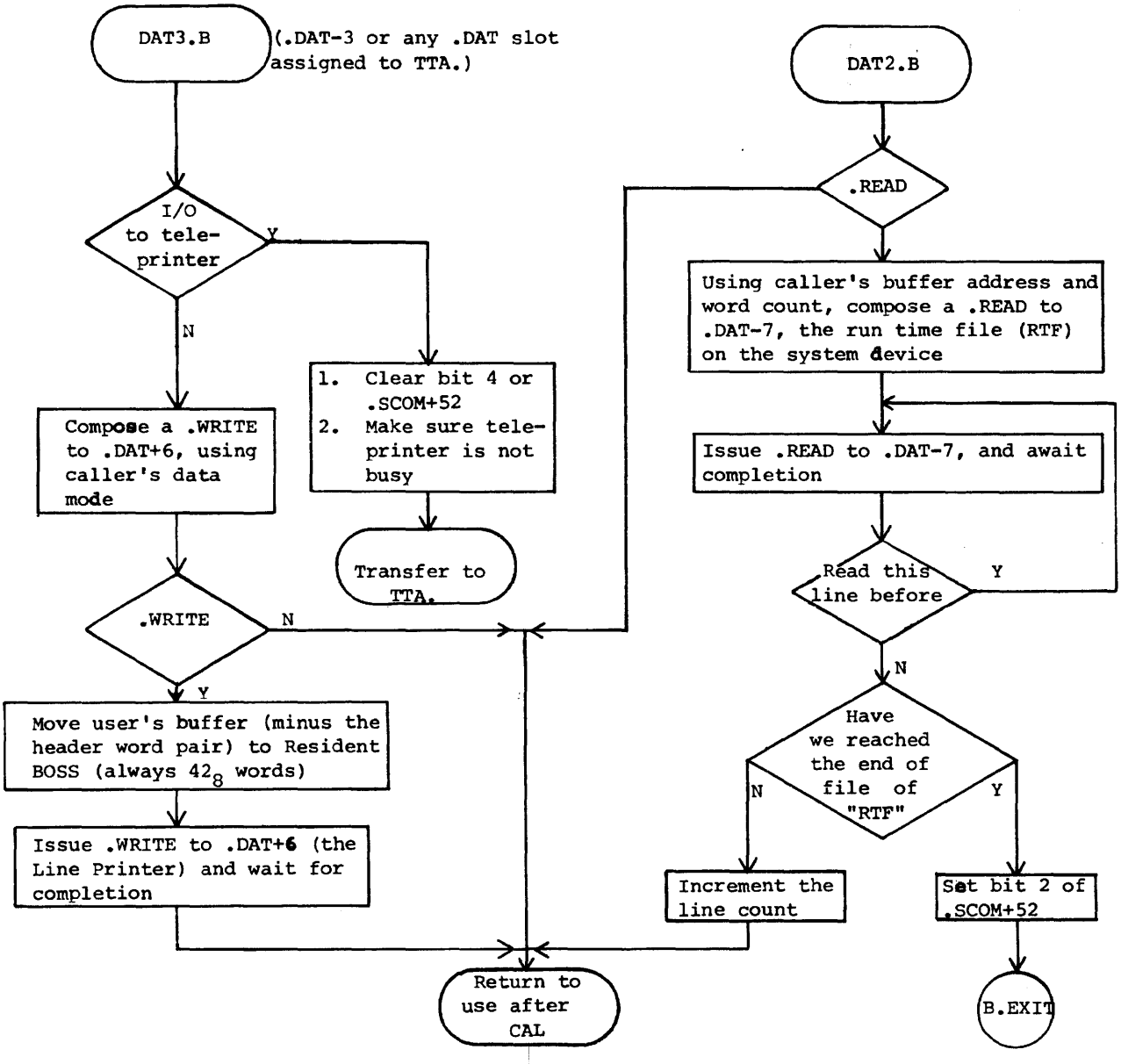
Figure 8-2



Points within the Resident Monitor which transfer control to Resident BOSS - 15

RESIDENT BOSS - 15

Figure 8-3

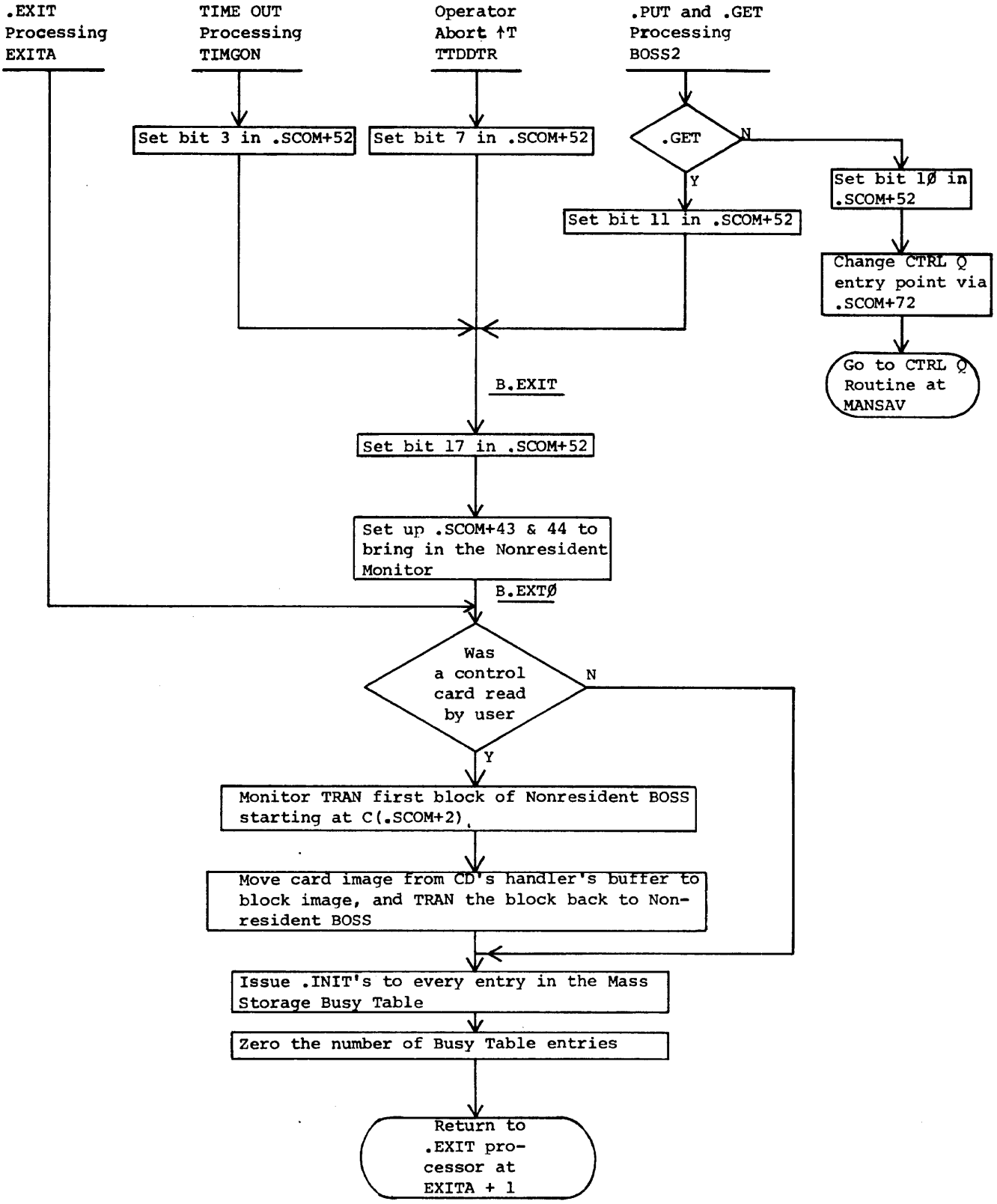


Processing for I/O Macros addressed to .DAT slots -2 or -3, or any slot assigned to TTA.

Points within the Resident Monitor which transfer control to Resident BOSS - 15

RESIDENT BOSS - 15

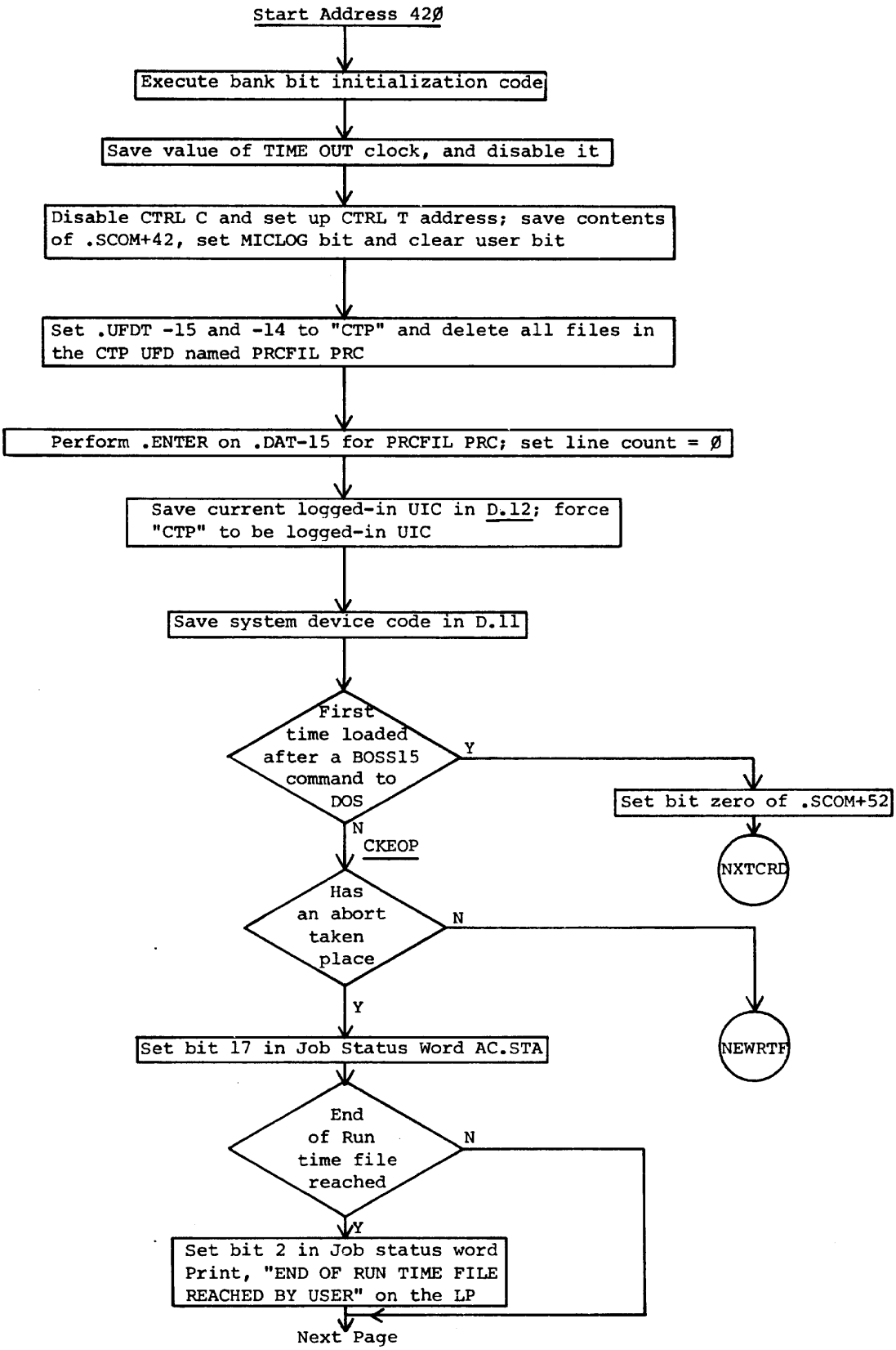
Figure 8-3 (cont.)



Points within the Resident Monitor which transfer control to Resident BOSS - 15

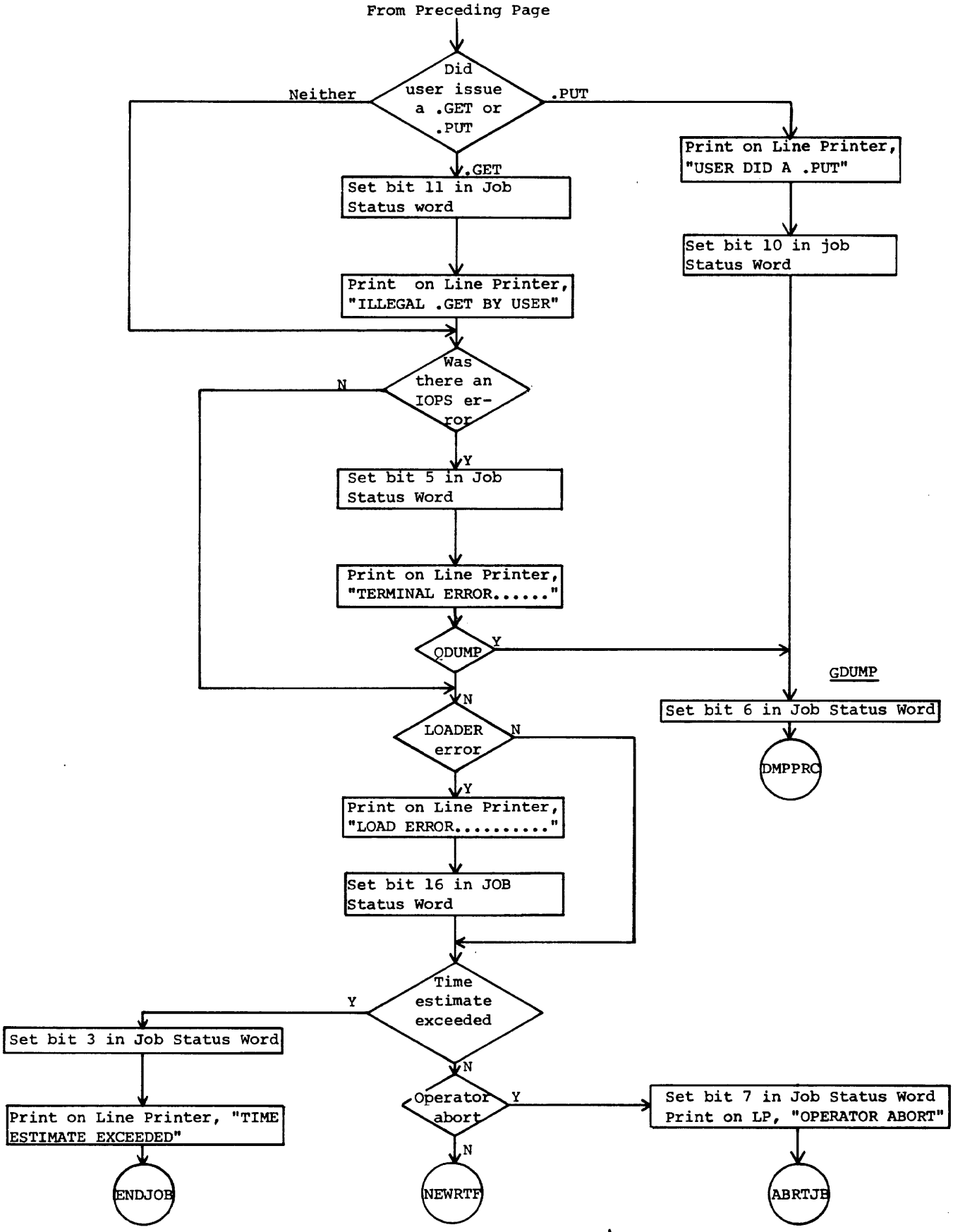
RESIDENT BOSS - 15

Figure 8-3 (cont.)



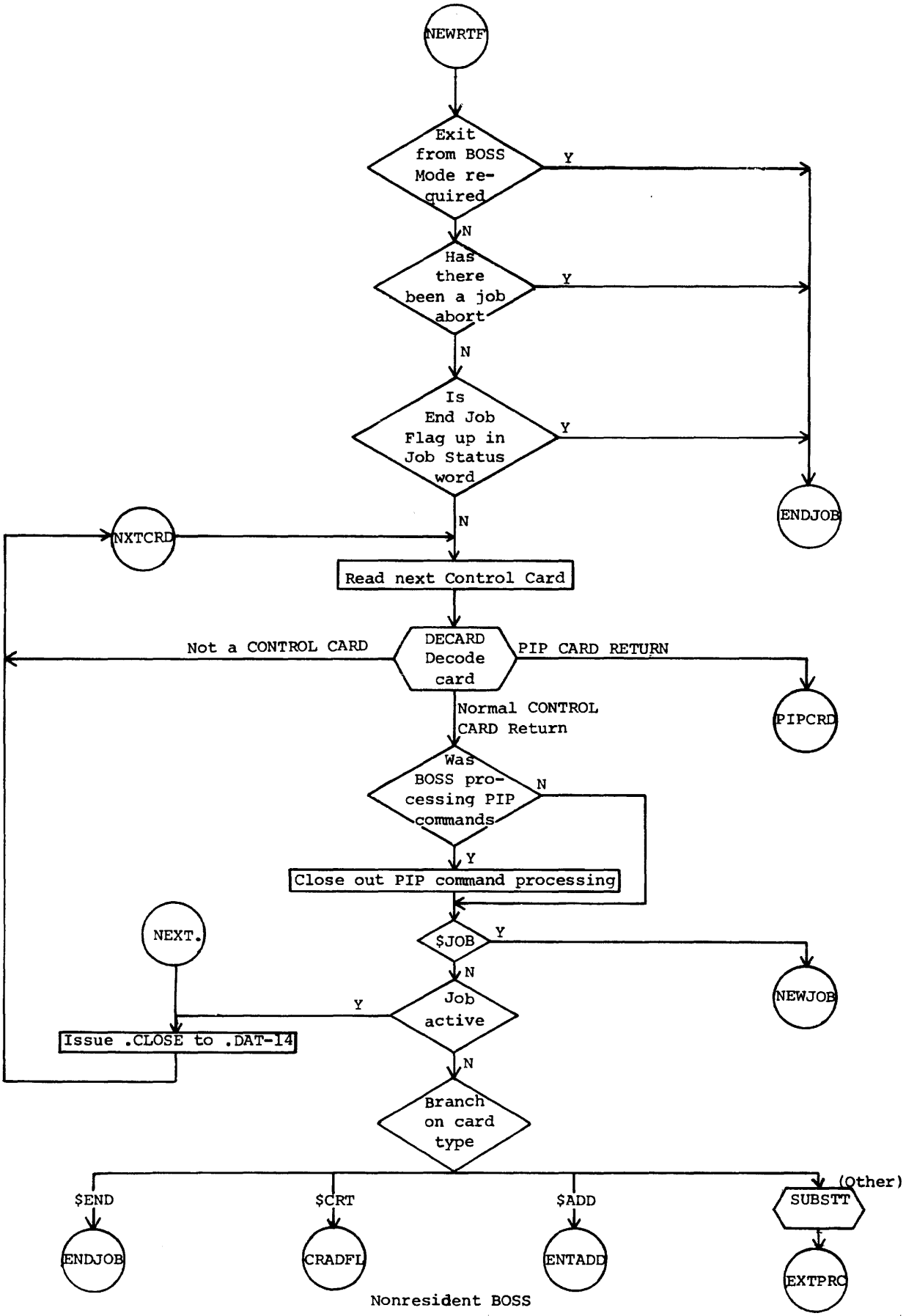
Nonresident BOSS

Figure 8-4



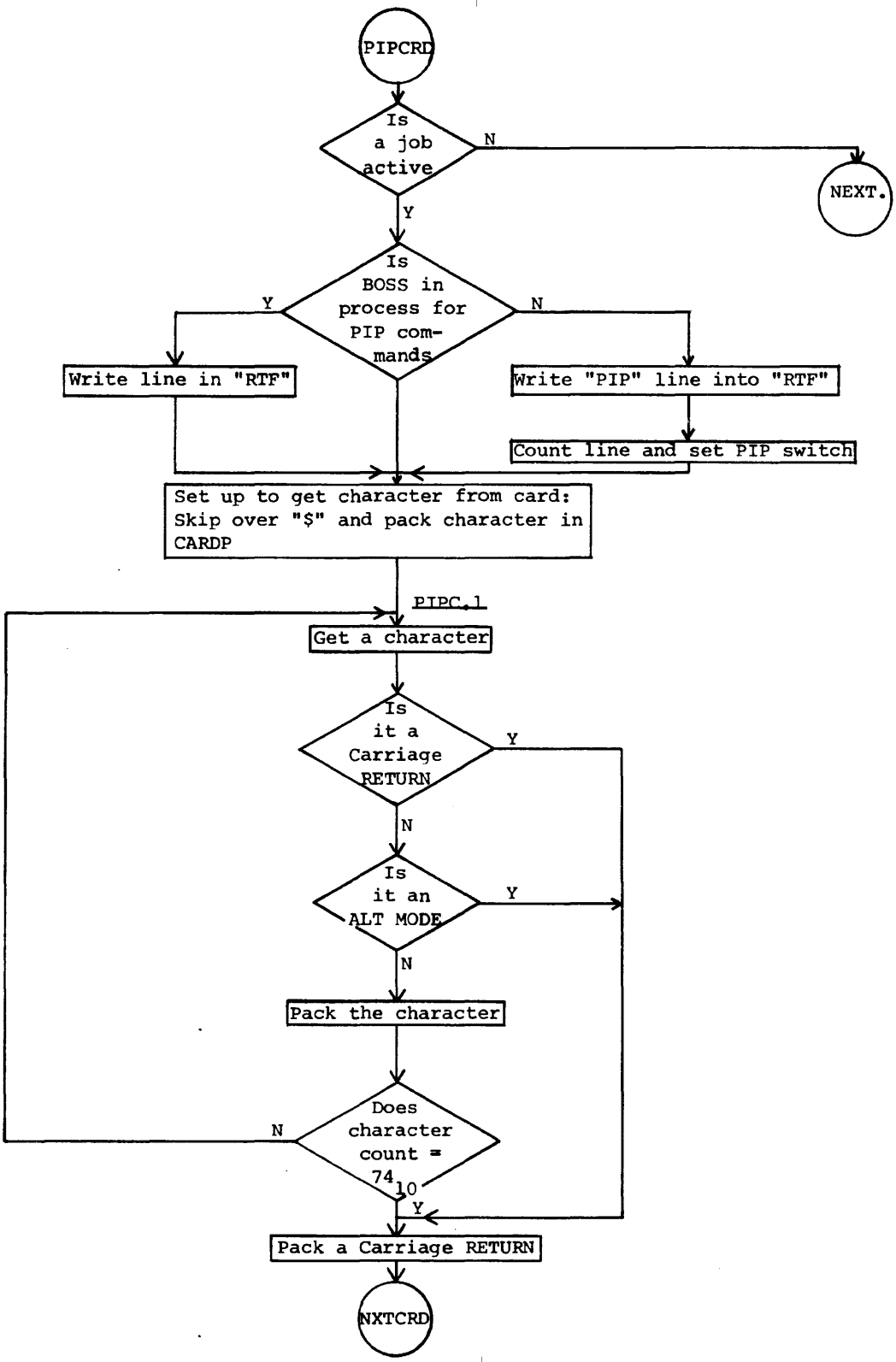
Nonresident BOSS

Figure 8-4 (cont.)



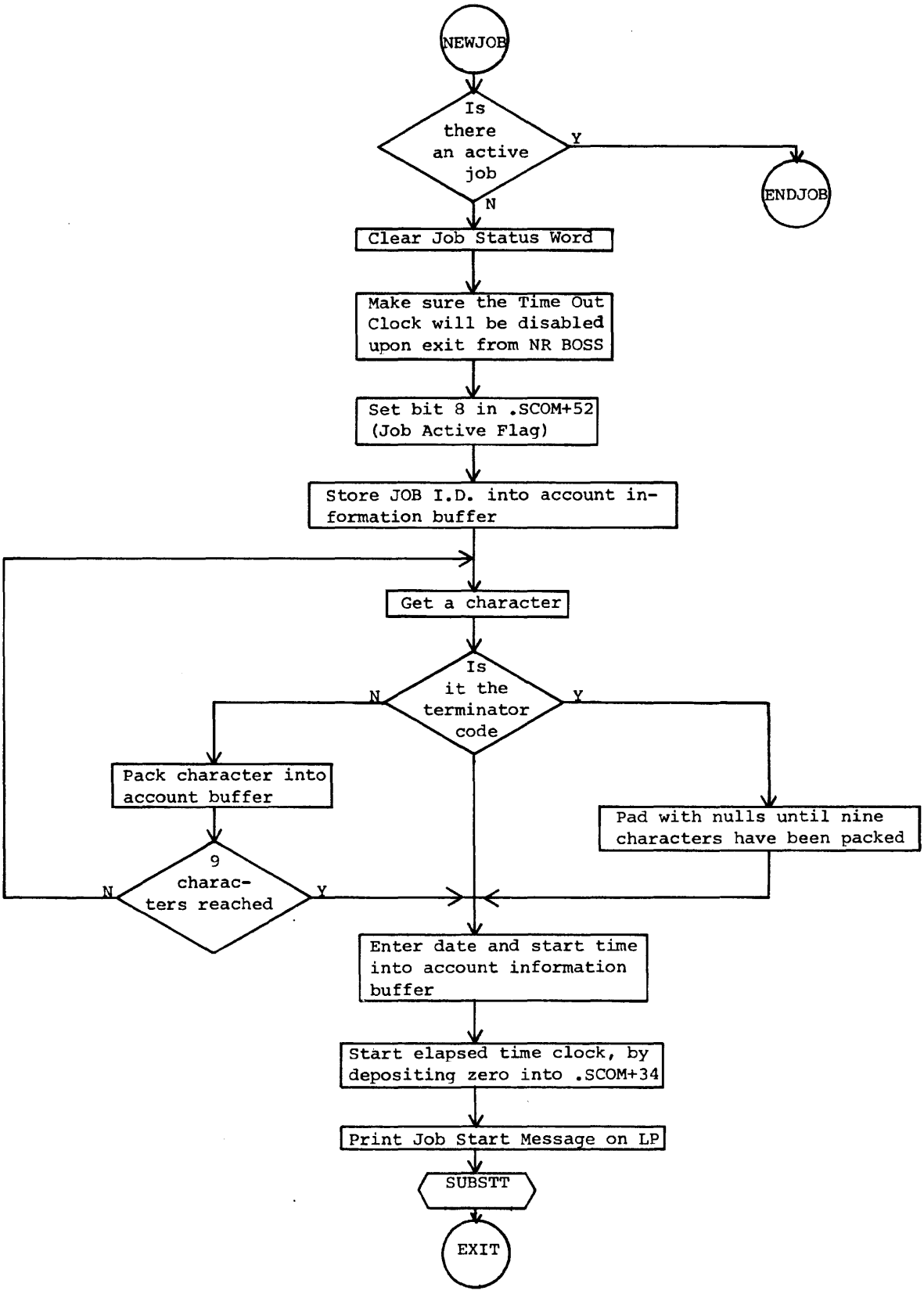
Nonresident BOSS

Figure 8-4 (cont.)



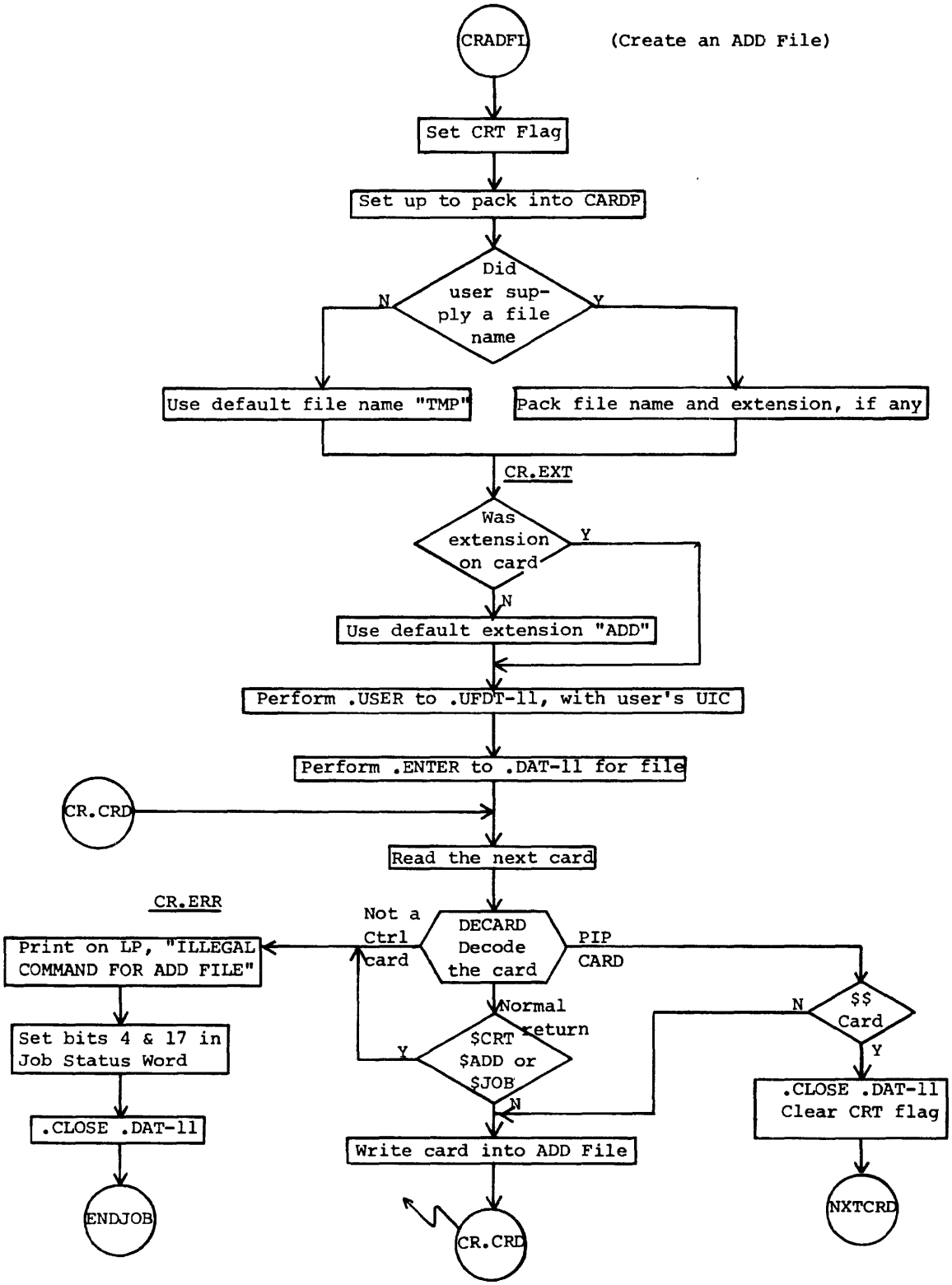
Nonresident BOSS

Figure 8-4 (cont.)



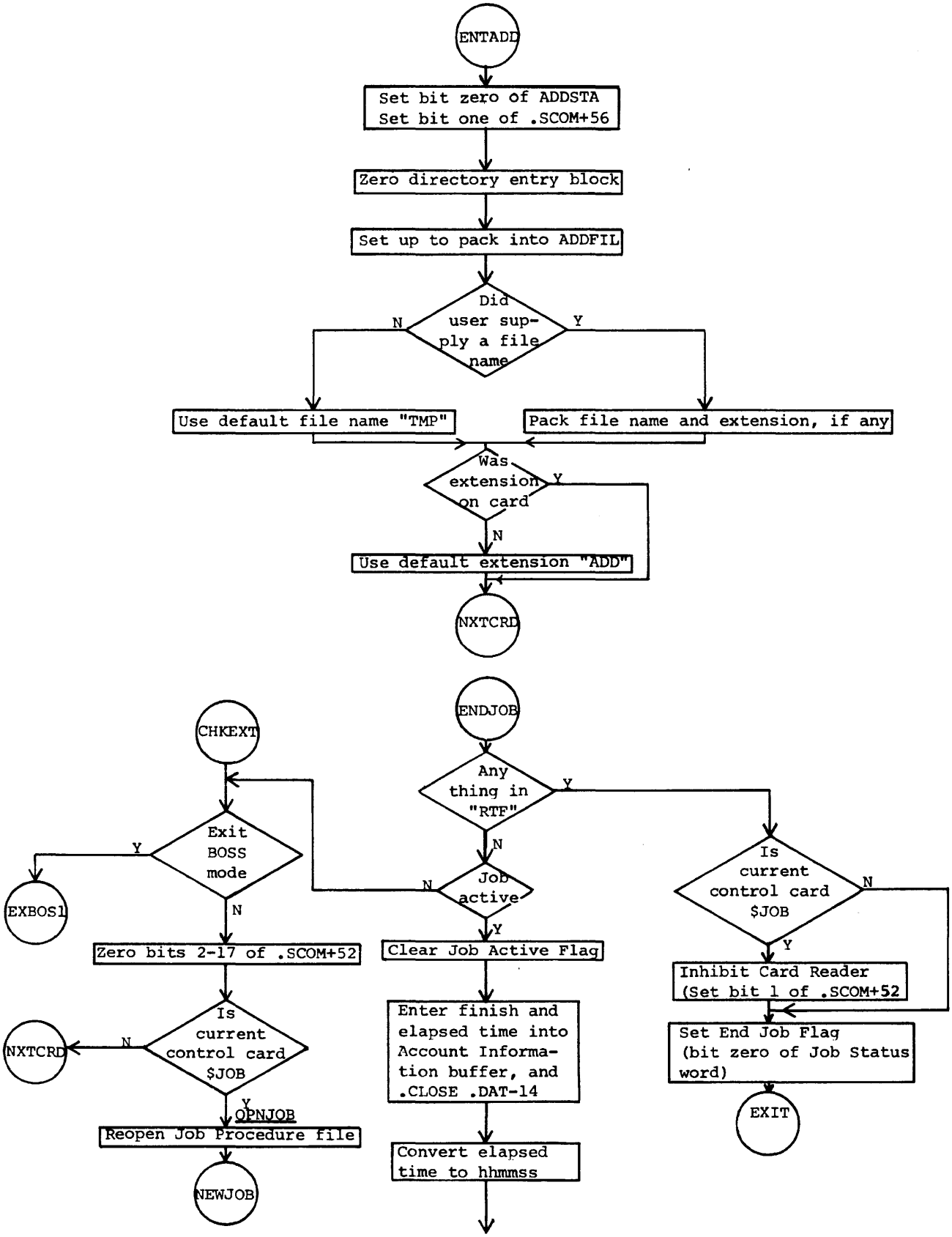
Nonresident BOSS

Figure 8-4 (cont.)



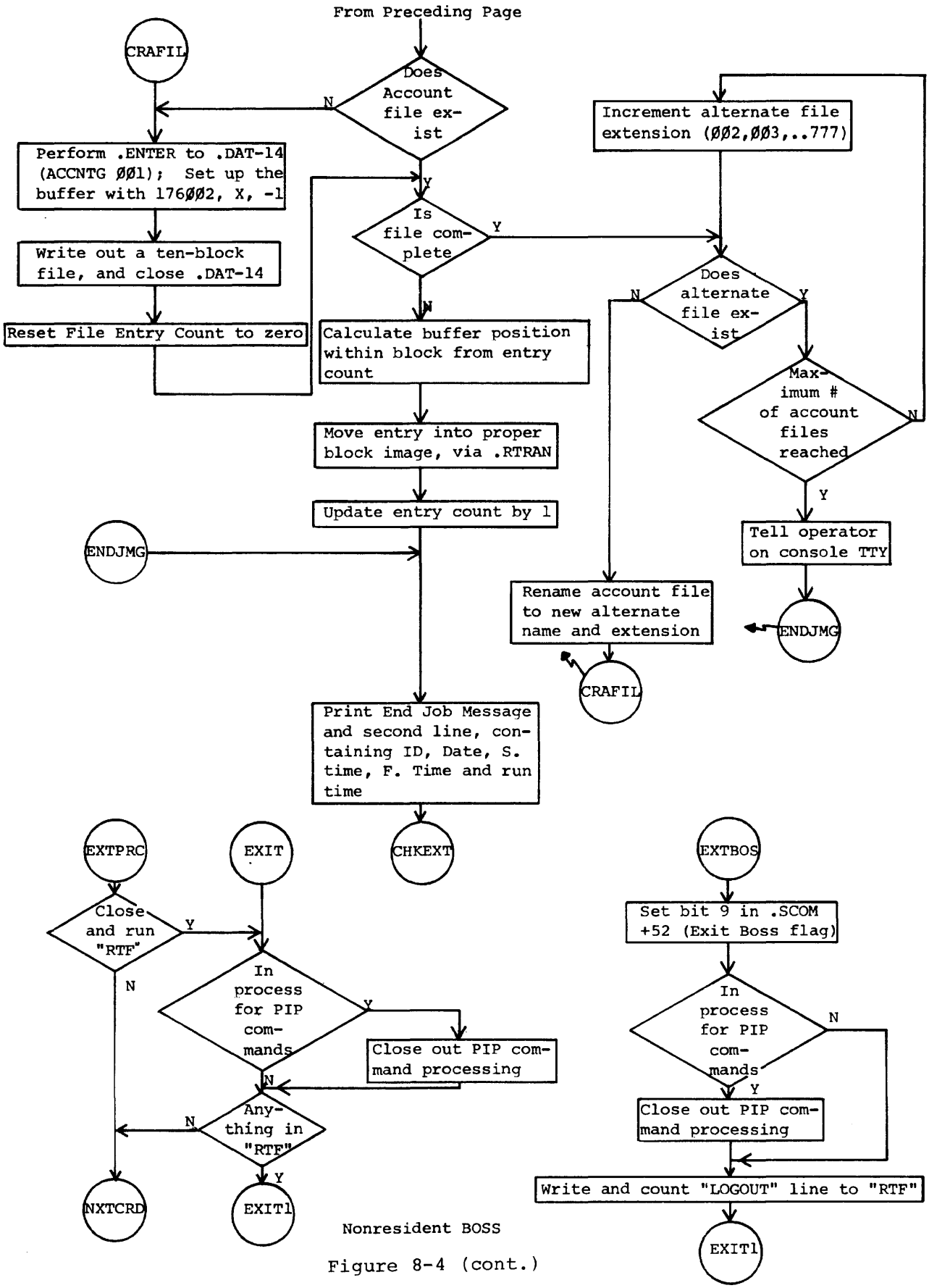
Nonresident BOSS

Figure 8-4 (cont.)



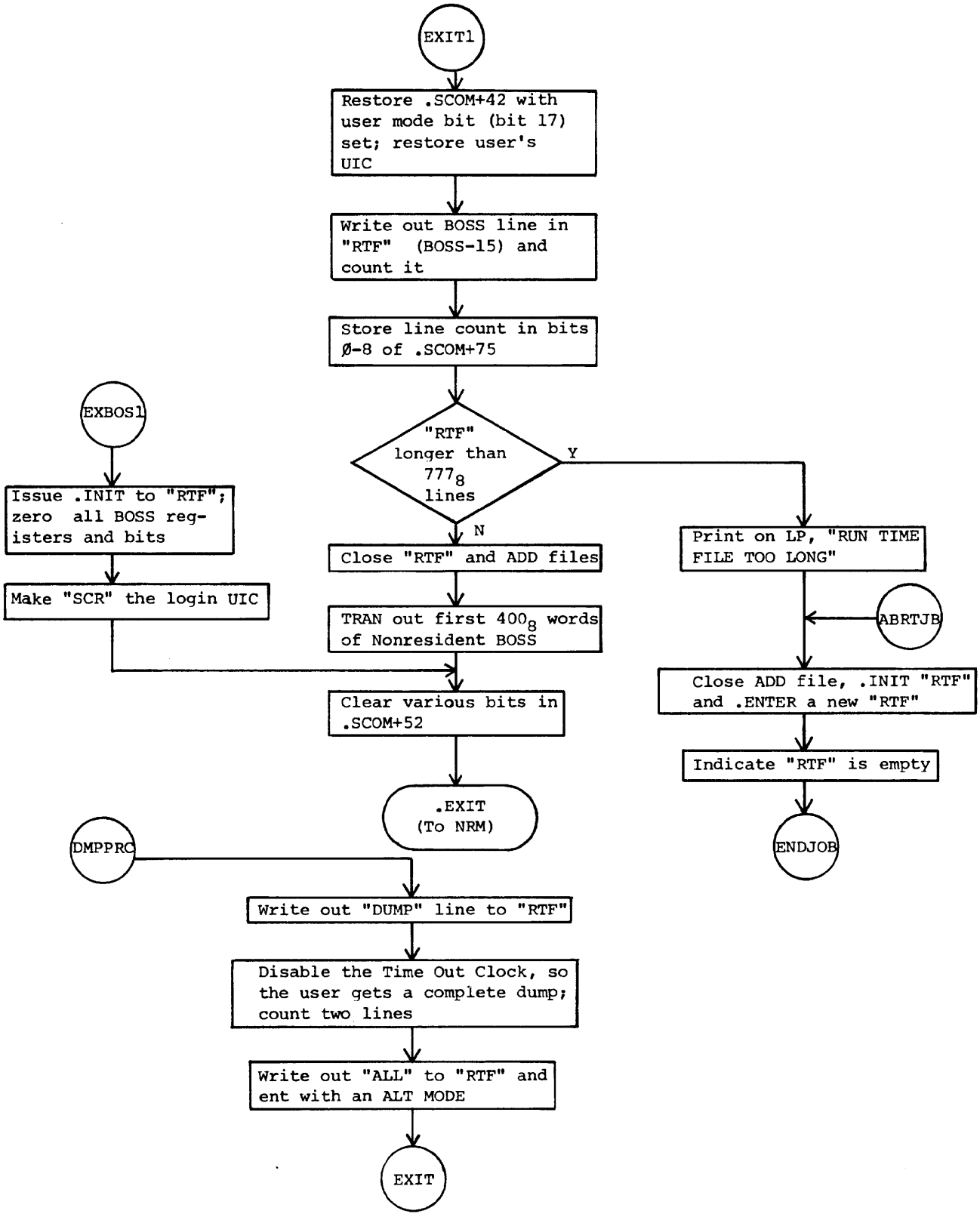
Next Page
Nonresident BOSS

Figure 8-4 (cont.)



Nonresident BOSS

Figure 8-4 (cont.)



Nonresident BOSS

Figure 8-4 (cont.)

8.1 PROCEDURE FILES

To each BOSS command there corresponds a disk-resident ASCII file, called a Procedure File. The Procedure File contains DOS commands. When DOS executes the commands in the Procedure File, it carries out the function specified by the BOSS control card. The DOS commands in the Procedure Files contain fields (for instance, a file name) that Nonresident BOSS fills in with text strings from the control card. These fields are called, "Variable Fields". Before executing the DOS commands contained in the Procedure File, therefore, all the variable fields have to be resolved. This process is very similar to a macro expansion, where (1) DOS is the assembly language, (2) the BOSS command name is the macro name, (3) the contents of the BOSS control card are the macro arguments, and (4) the Procedure File is the macro definition. The expanded DOS commands are put in a Disk File, called the "Run Time File (RTF)". The RTF can contain the expansion of one or more Procedure Files, up to 777_8 IOPS ASCII records.

BOSS expands Procedure Files strictly on a text string, character basis. It has no knowledge of the intrinsic function of each BOSS control card, except for $JOB, $END, $CRT, and $ADD ($END, $CRT, $ADD have no Procedure Files) Appendix C contains a listing of all standard Procedure Files.

8.1.1 Procedure File Format

In order to ensure successful expansion, all Procedure Files must follow a strict format. The first record of the Procedure File must be a control record, with parameter information. The first record may also contain comments, because BOSS interprets only pertinent information, and ignores the rest. The numbers Ø, 1, 2, 3, and 4 specify different options. All other characters are ignored. The option digits can appear in any order, and anywhere on the record. The option specified by each number is given below:

Ø - Expanded Substitution (default, if "3" not given explicitly)

This option specifies that the Procedure File is to be expanded according to the normal rules of substitution, which are given below.

1 - Open Ended File (default, if "2" not given explicitly)

This option instructs the Nonresident BOSS Monitor to leave the RTF open after expanding the current Procedure File. BOSS then searches for the next control card.

2 - Closed End File

This option instructs Non-resident BOSS to close the RTF after expanding the current Procedure File, and to execute the DOS commands in the RTF. Procedure Files corresponding to commands that may possibly be followed by "Data Cards" should be of Type 2.

3 - Direct Substitution

This option indicates the BOSS should not expand the Procedure File according to normal rules. Refer to paragraph 8.1.2 for information on Direct Substitution.

4 - Test Mode

This option indicates that BOSS should echo the Procedure File expansion on the Line Printer. This allows a check on the Procedure File.

The following combinations are illegal:

Ø and 3

1 and 2

If BOSS finds an illegal option combination, it will print,

ILLEGAL PROC FILE

and search for the next control card.

BOSS uses all other records in the Procedure File as macro definition records. Records after the first one are all Macro Definition Records. For each such record, a record will be written in the RTF. Each Macro Definition Record has the same format. Two types of fields are used: K-fields and V-fields. K-fields specify constant character strings that will be written into the RTF exactly as they appear in the Procedure File. V-fields specify variable character strings to be substituted from specified strings on the Control Cards. Each Macro Definition Line of a Procedure File can contain any number of K- and V-fields, in any combination. V-fields are delimited by @-signs. K-fields are delimited by adjacent V-fields, or the end or beginning

of the record. Since there are only two types of fields, only one need have delimiters. Two adjacent V-fields, however, require two adjacent @-signs.

K-fields

K-fields may be any string of legal IOPS ASCII characters, except the **@-sign.**

V-fields

A V-field has the following format*:

$$v = @\left\{\begin{matrix}A\emptyset n\\ U\emptyset n\\ Dnn\\ O\end{matrix}\right\}([\{\text{V-field}\atop\text{K-field}\}])@$$

The two @-signs delimit the field. The first part of the field (A, D, U or O) is a card-position identifier, and must be present. It identifies the position on the current Control Card of the character string to be substituted in the RTF. The legal combinations are:

```
AØØ,AØ1,....AØ9
UØØ,UØ1,....UØ9
DØØ,DØ1,....DØ9,D1Ø,...D17
O
```

With the exception of D1Ø through D17, each of the above position identifiers corresponds to a unique character string of the Control Card, according to the following scheme:

```
$CMD;O AØØ:DØØ(UØØ);AØ1:DØ1(UØ1);....;AØ9:DØ9(UØ9)
```

The D1Ø...D17 position identifiers do not correspond to character strings found on the Control Card, but rather to character strings defined by BOSS. Thus,

```
D1Ø - Unused
D11 - .SIXBT representation of the System Device Code
      ('DK'or 'DP')
D12 - Current Logged in UIC
D13 - .SIXBT representation of Carriage RETURN
D14 - .SIXBT representation of ALT MODE
D15 - Unused
D16 - Unused
D17 - Unused
```

---

* Standards for this format description are identical to those specified in Chapter 5 of the DOS-15 User's Manual, DEC-15-MRDA-D.

The parentheses in a V-field must be present. They are used to specify a default string. The default string is used in case the string on the Control Card specified by the position identifier is null. A set of parentheses must be included, even if the default string is null. The default string itself can be a variable, resulting in nested variables. Nesting has a theoretical limit of 2^{17} variable fields.

8.1.2 Direct Substitution

When processing a Direct Substitution Procedure File, BOSS places the fields on the Control Card into the RTF just as they stand with only leading spaces ignored. That is, BOSS does not necessarily expect to find file names, and so on, as with normal substitution. Fields on the Control Cards are separated by semi-colons (;), and are processed in a serial manner. The ampersand (&) is used for a special purpose. It causes the current record being composed for the RTF to be terminated with a Carriage RETURN, and written out, and a new record started. This is so that the limit of seventy-five characters per line will not be exceeded.

There are only two legal field types within the Procedure File. They are as follows:

1. AØØ through A99
2. DlØ through Dl7 (System Defined)

In making up Direct Substitution Procedure Files, the following rules must be followed:

1. The first line must contain a three (3). This declares the file to be direct substitution.
2. The "A" fields must appear in sequential order, starting at AØØ. Each "A" field can be used only once within the Procedure File.
3. The "D" fields can only be "DlØ" through "Dl7". They can be used any number of times, in any order.
4. Variable expressions must follow the standard V-field format, as in expanded substitution.

8.1.3 Example of Procedure File

The following example shows a typical Direct Substitution Procedure File, the Control Cards used to call it, and the resulting lines produced in the Run Time File.

Procedure File[1]- Map PRC

```
3 PROCEDURE FILE TO RUN CHAIN WITH NO OVERLAYS
CHAIN
@AØØ(TMPXCT)@@D14()@
@AØ1(SZ)@@D14()@
@AØ2(FILTMP)@@D14()@
@D14()@
```

Control Cards as They Appear

```
$MAP TEST1;SZ,VTC/ABC,DEF,NAM1,&NAM2,;
$*Ø1 NAM3,NAM4,NAM5/;TEST1,SUB1,SUB2,&;
$*Ø2 SUB3,SUB4,SUB5
```

Run Time File Lines

```
CHAIN )
TEST1 (ALT MODE)
SZ,VTC/ABC,DEF,NAM1,)
NAM2,NAM3,NAM4,NAM5/ (ALT MODE)
TEST1, SUB1, SUB2,)
SUB3,SUB4,SUB5 (ALT MODE)
(ALT MODE)
```

Note: D14=Altmode, <ALTMODE> is an Altmode, and <CR> is a Carriage Return.

8.2 BOSS-15 ACCOUNTING

BOSS has a very simple accounting mechanism. It keeps an account record for each job in a random access file in the CTP UFD. Hence, the file is protected, and can only be accessed after successful execution of a $MIC command.

The name of the accounting file is ACCNTG nnn. (The first has an extension of ØØ1.) Each file is ten physical blocks long, and contains enough information for 31Ø jobs, thirty-one per physical block. When BOSS fills up one file, it increments the extension, and starts a new one. Every time a job ends, BOSS checks whether ACCNTG ØØ1 exists. If it does not, BOSS creates one. If it does, BOSS checks whether it is full. If not full, BOSS makes a new entry; if full, BOSS

[1]Direct Substitution File

searches for the first unused extension number. If all extension numbers have been used (up to 999) BOSS prints this message to the operator on the teleprinter:

```
MAX NUMBER OF ACCOUNTING FILES REACHED
PLEASE PROCESS AND DELETE THEM
```

Every time the system manager processes an accounting file, therefore, he should delete the file.

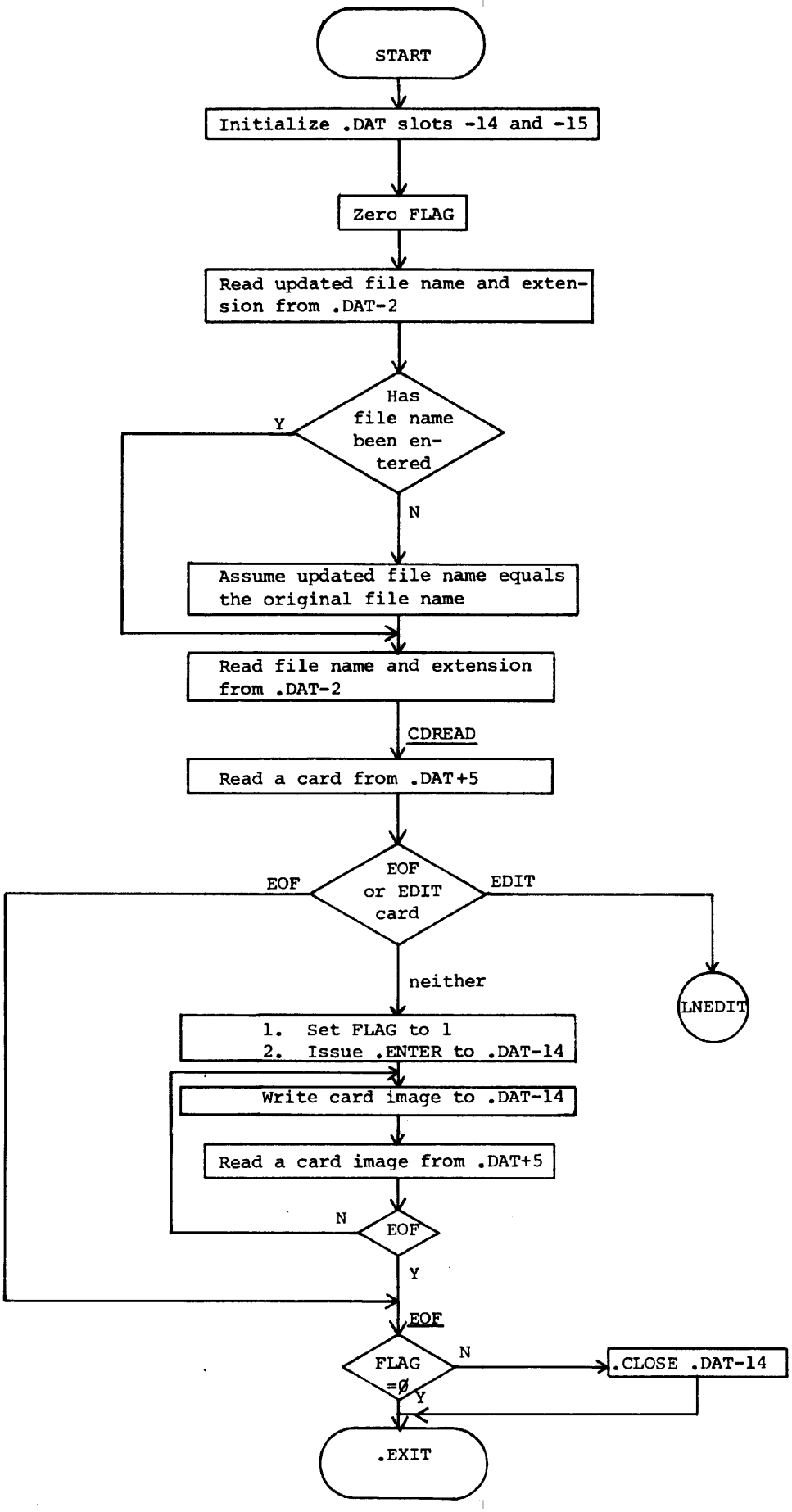
For each completed job, BOSS writes out an eight-word record to the accounting file. The records have the following format:

Word #	Content
1, 2, 3	Job I.D., in .SIXBT
4	Date, packed mmddyy
5	Start Time, in hhmmss
6	End Time, in hhmmss
7	Run Time, in hhmmss
8	Terminal Job Status Word

A word whose contents equal 777777_8 immediately follows the last job accounting record in each physical block of the accounting file.

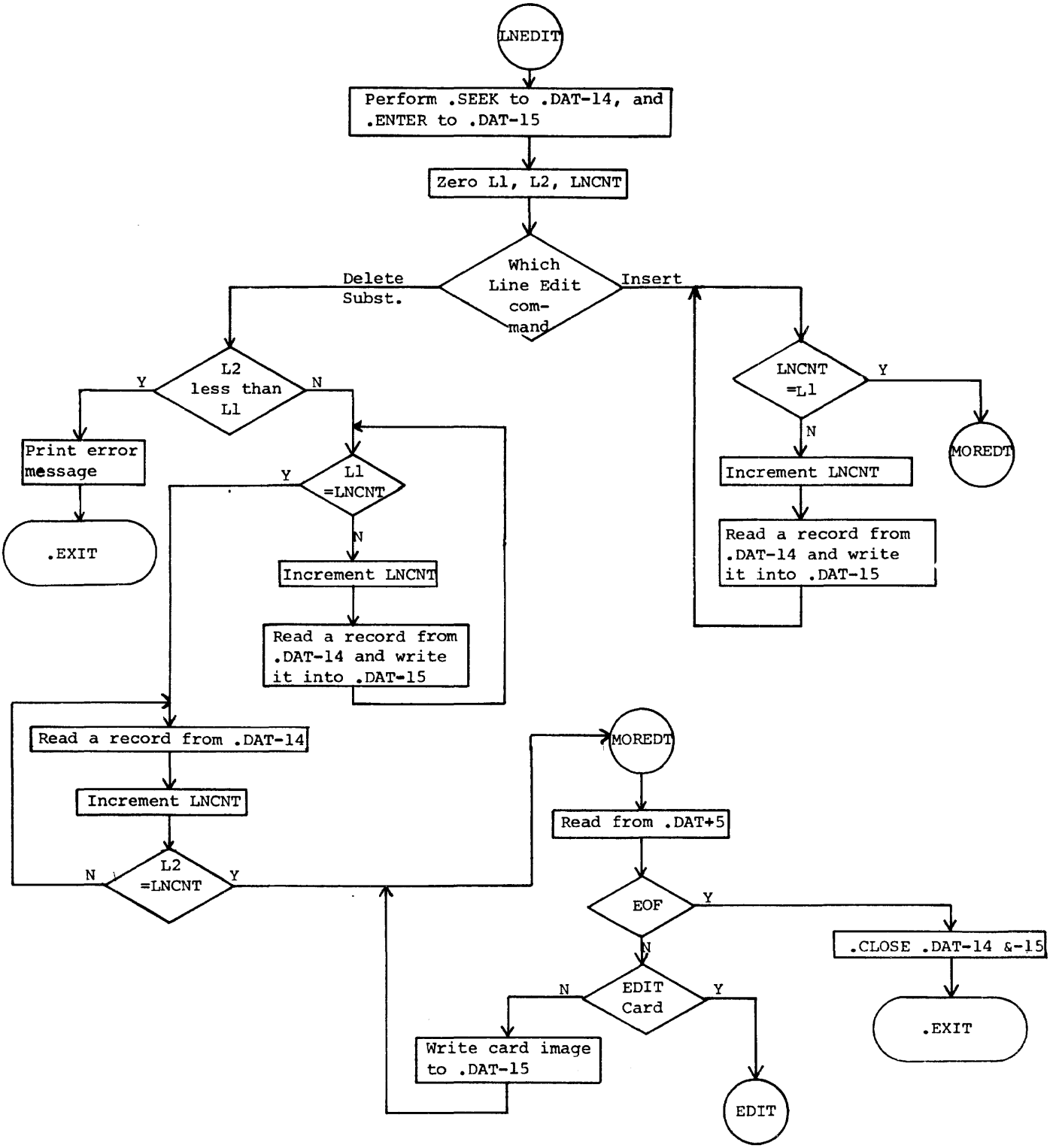
8.3 B.PRE

Figure 8-5 is a flowchart of B.PRE, the BOSS Line Editor.



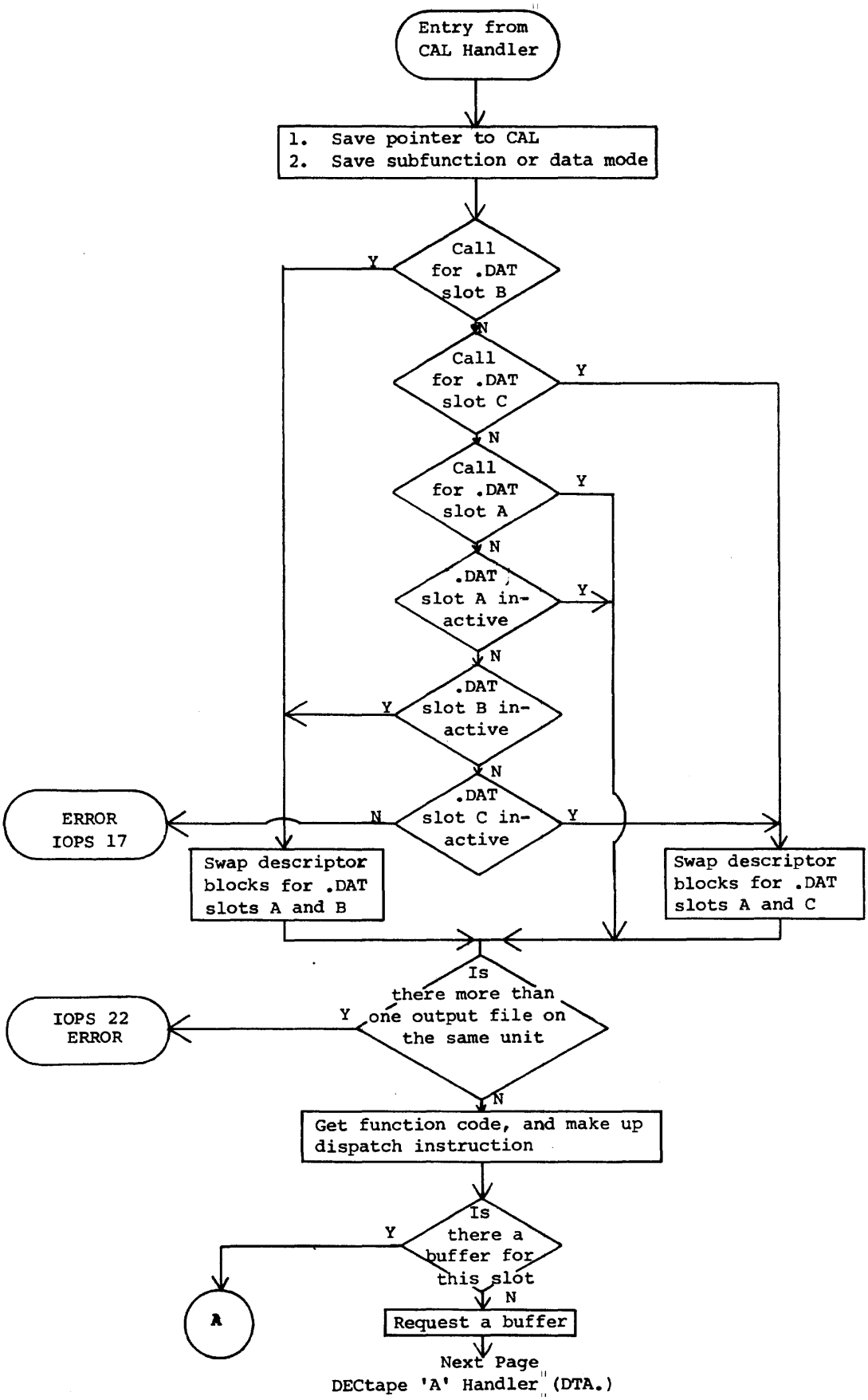
B.PRE

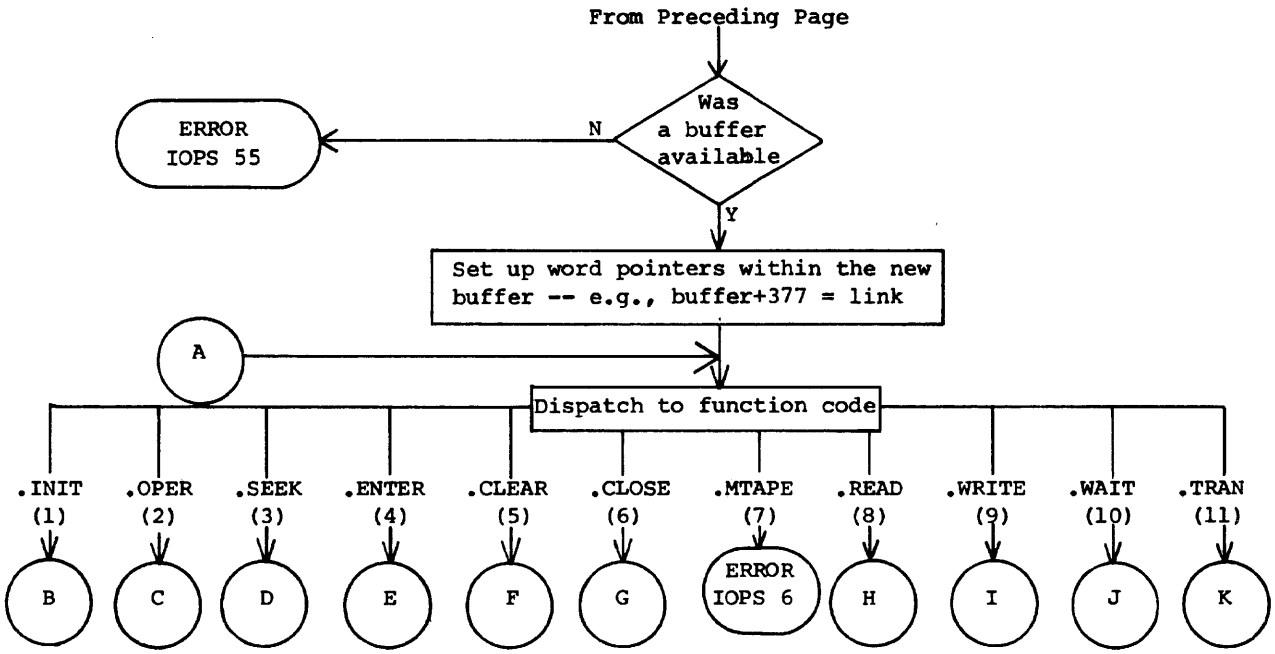
Figure 8-5

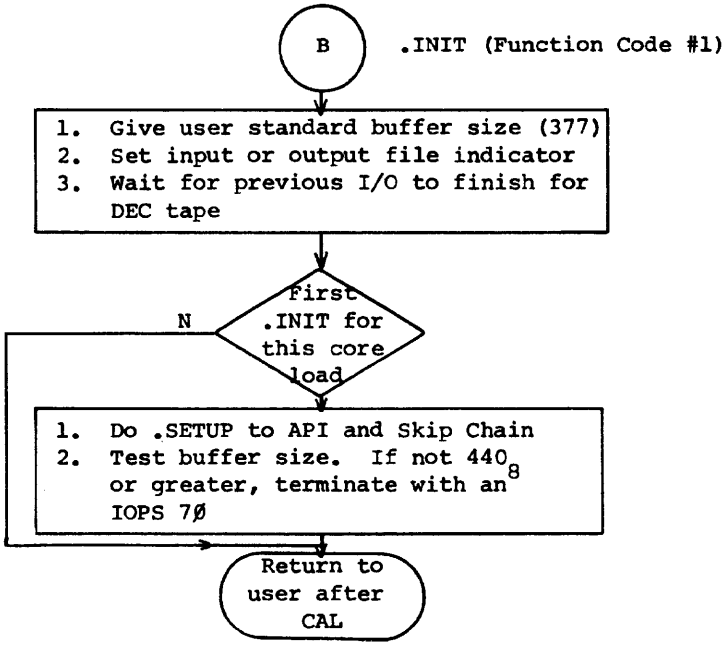


B.PRE

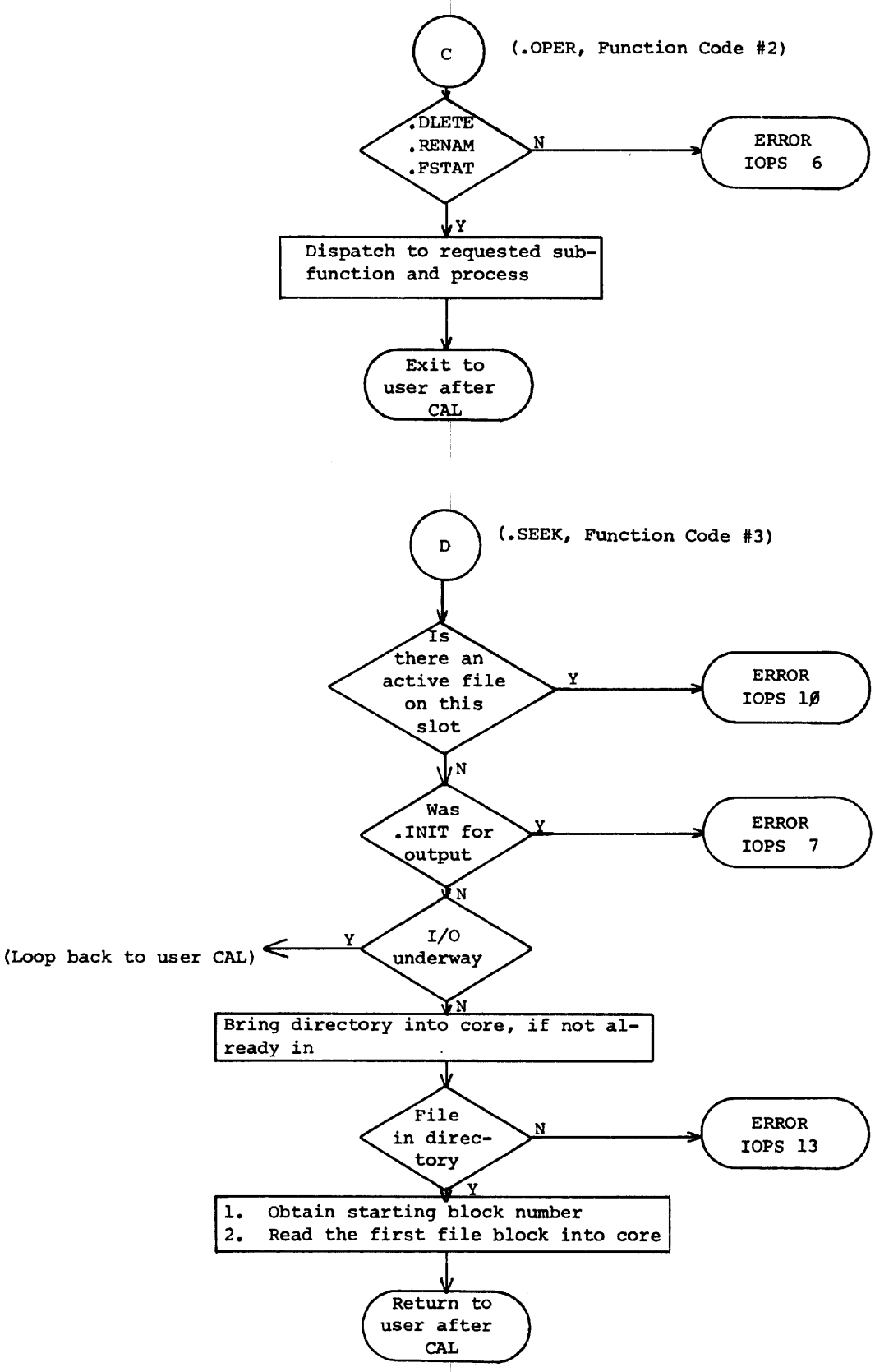
Figure 8-5 (cont.)


Entry from CAL Handler
1. Save pointer to CAL
2. Save subfunction or data mode
Call for .DAT slot B
Y
N
Call for .DAT slot C
Y
N
Call for .DAT slot A
Y
N
.DAT slot A in-active
Y
N
.DAT slot B in-active
Y
N
.DAT slot C in-active
N
Y
ERROR IOPS 17
Swap descriptor blocks for .DAT slots A and B
Swap descriptor blocks for .DAT slots A and C
Is there more than one output file on the same unit
Y
IOPS 22 ERROR
N
Get function code, and make up dispatch instruction
Is there a buffer for this slot
Y
N
A
Request a buffer
Next Page
DECtape 'A' Handler (DTA.)

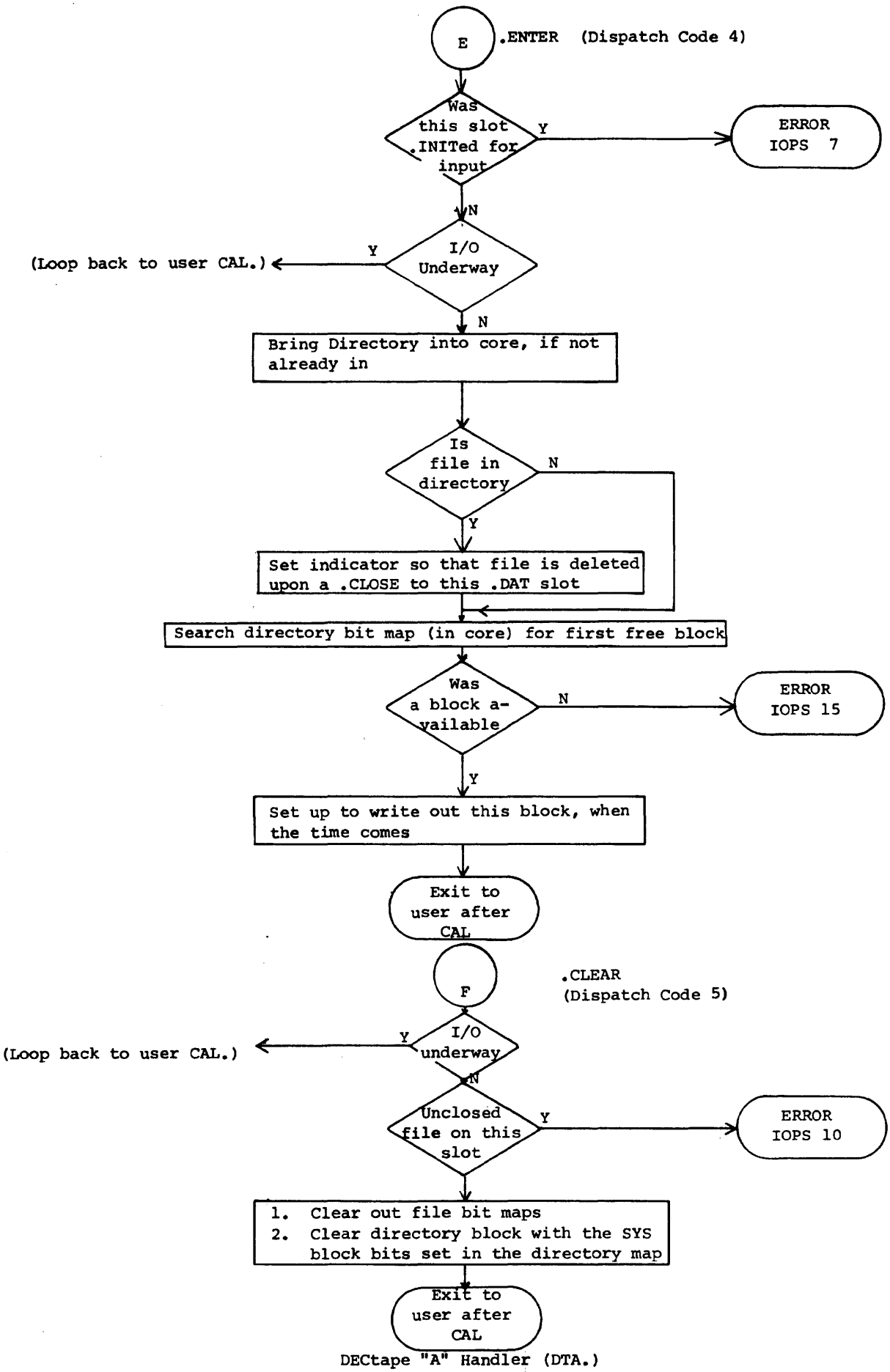


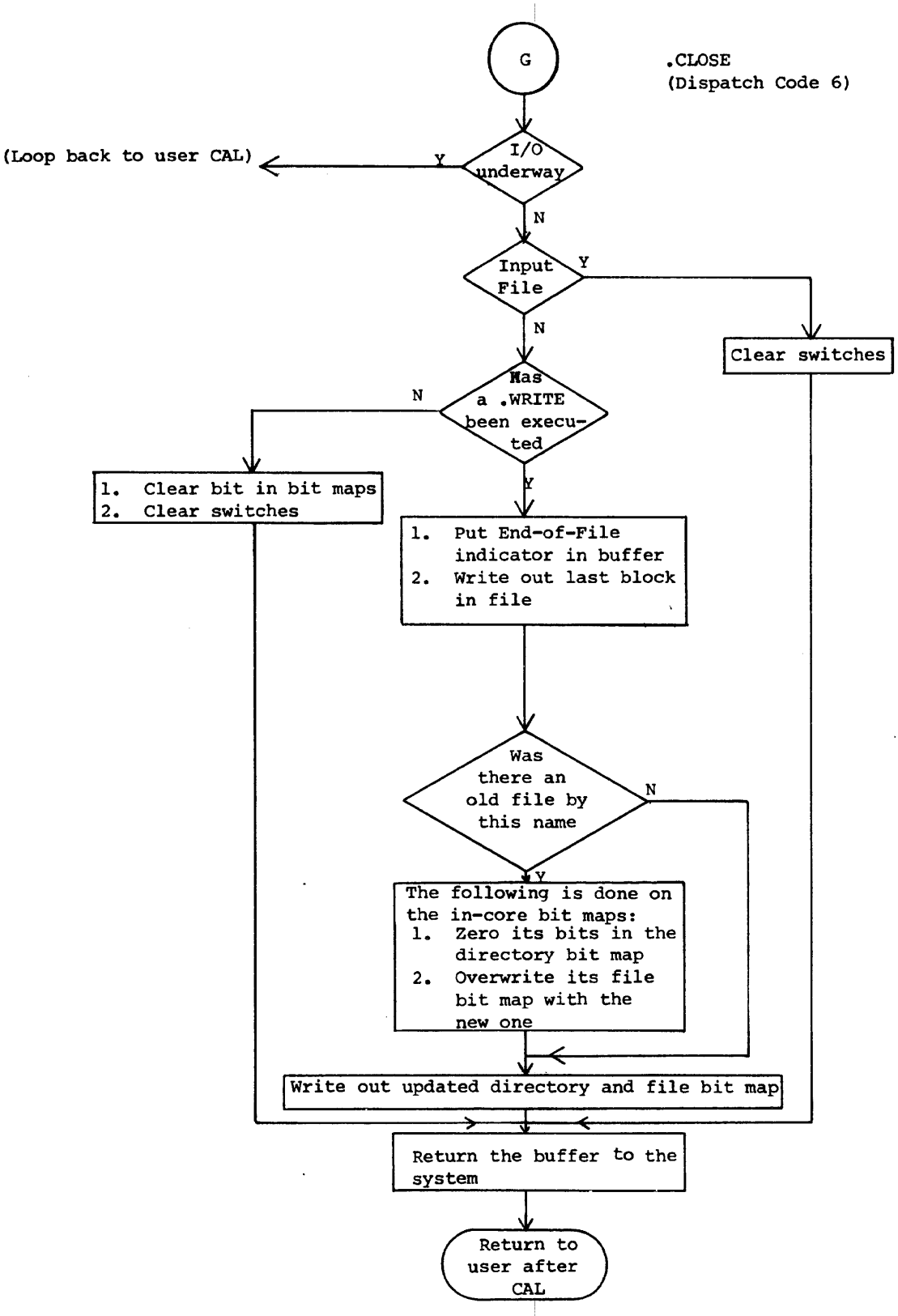


DECtape "A" Handler (DTA.)

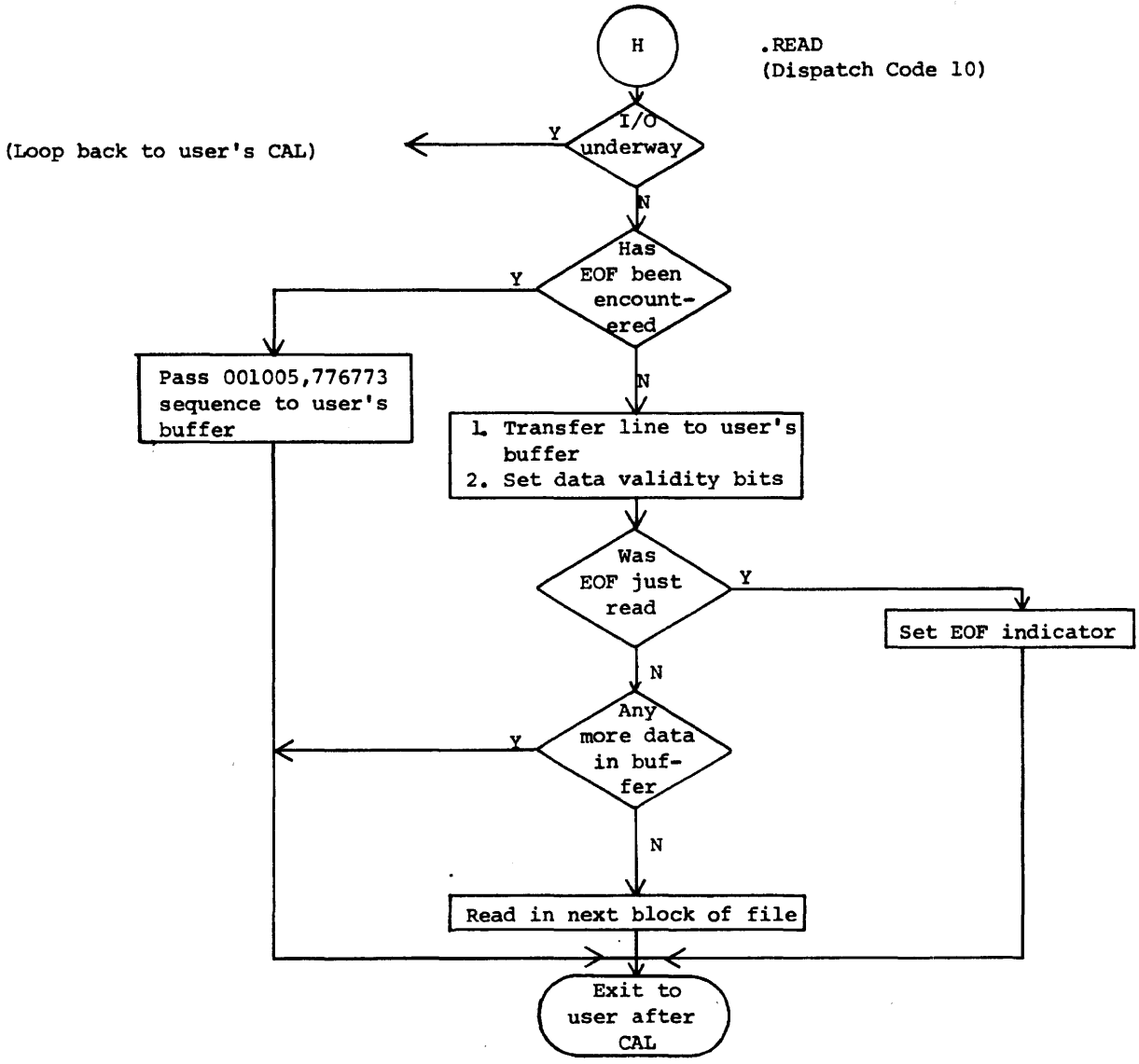


DECtape "A" Handler (DTA.)

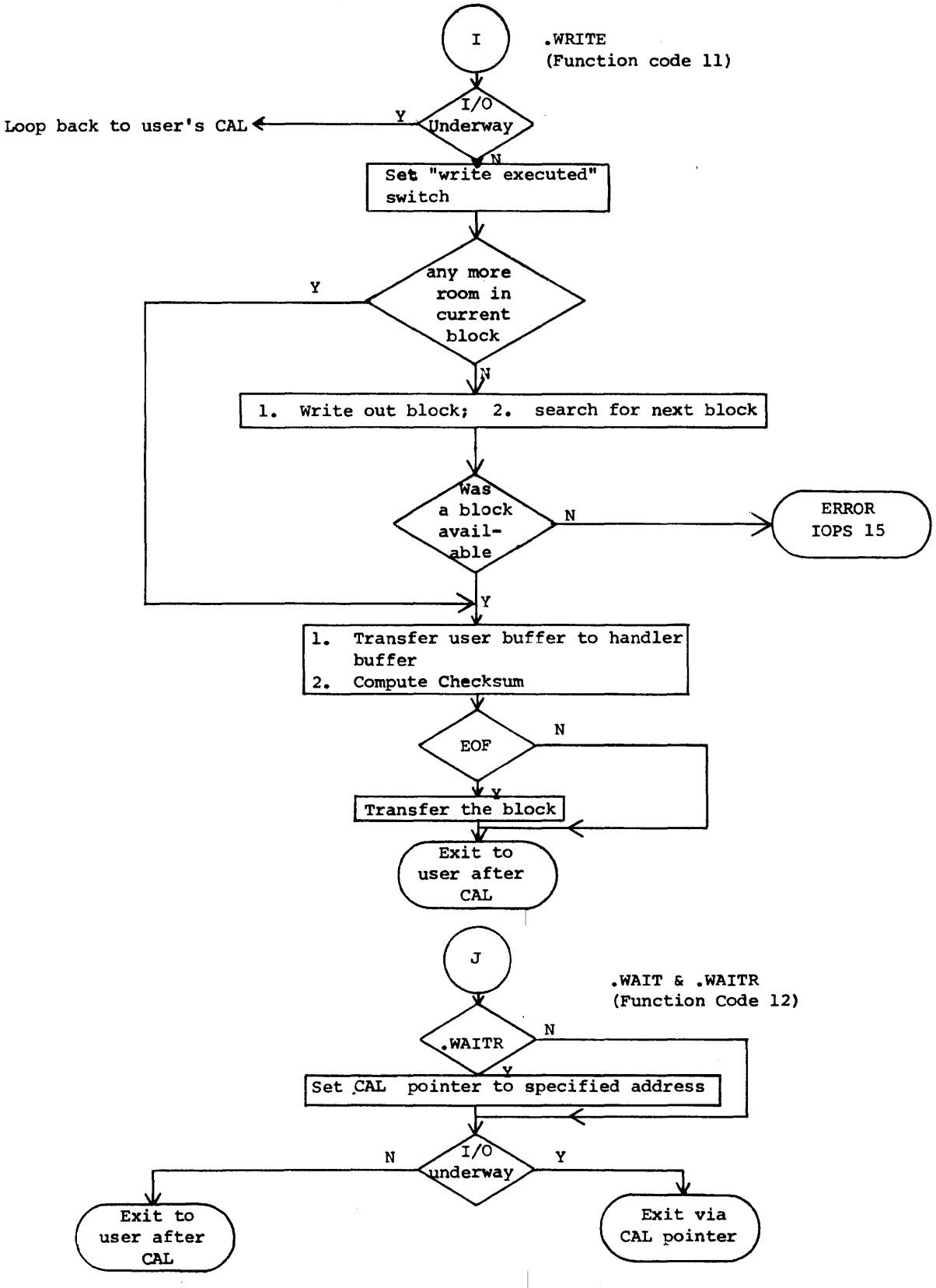

E
.ENTER (Dispatch Code 4)
Was this slot .INITed for input
Y
ERROR IOPS 7
N
I/O Underway
Y
(Loop back to user CAL.)
N
Bring Directory into core, if not already in
Is file in directory
N
Y
Set indicator so that file is deleted upon a .CLOSE to this .DAT slot
Search directory bit map (in core) for first free block
Was a block available
N
ERROR IOPS 15
Y
Set up to write out this block, when the time comes
Exit to user after CAL
F
.CLEAR
(Dispatch Code 5)
I/O underway
Y
(Loop back to user CAL.)
N
Unclosed file on this slot
Y
ERROR IOPS 10
1. Clear out file bit maps
2. Clear directory block with the SYS block bits set in the directory map
Exit to user after CAL



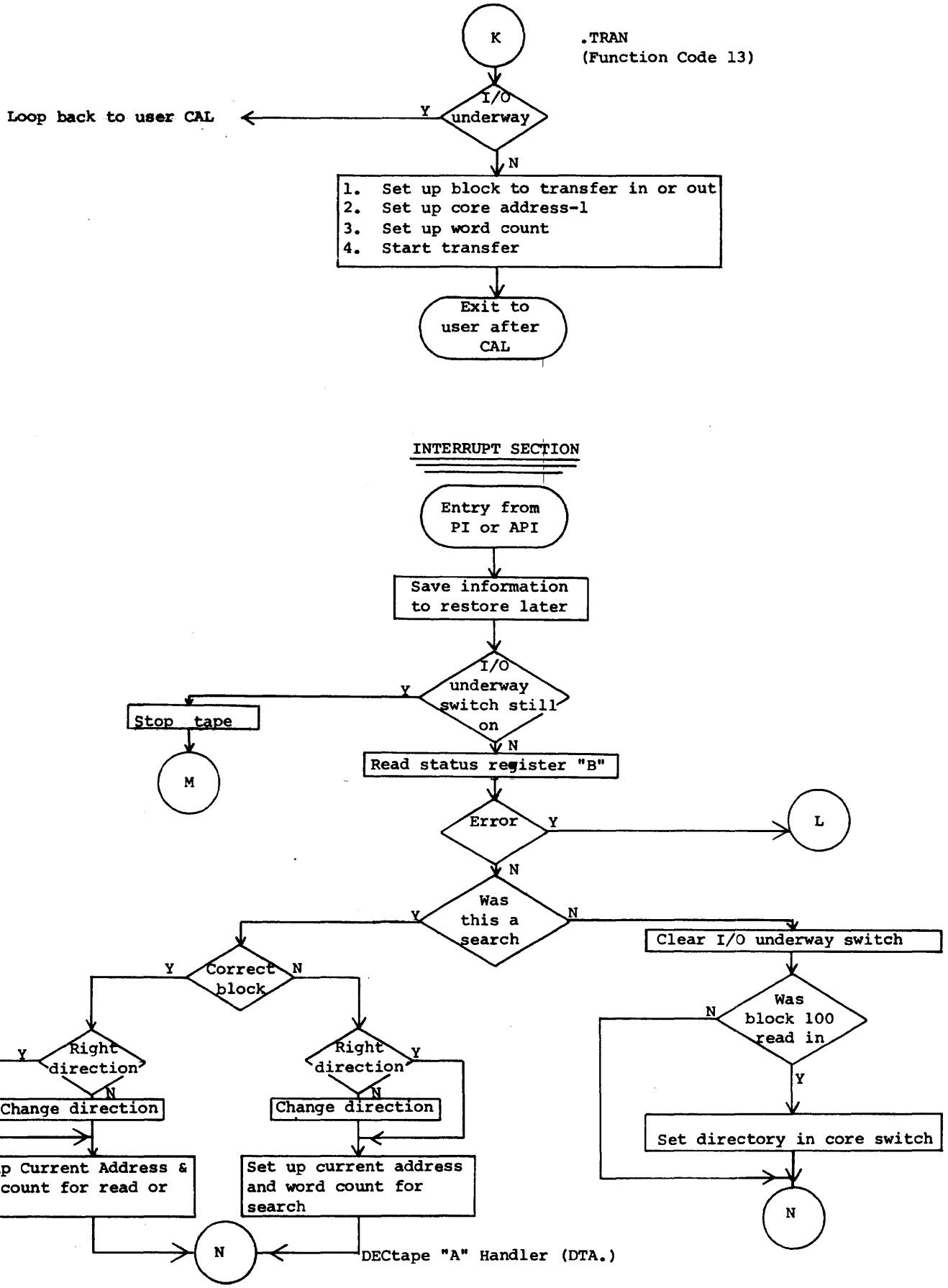
DECtape "A" Handler (DTA.)

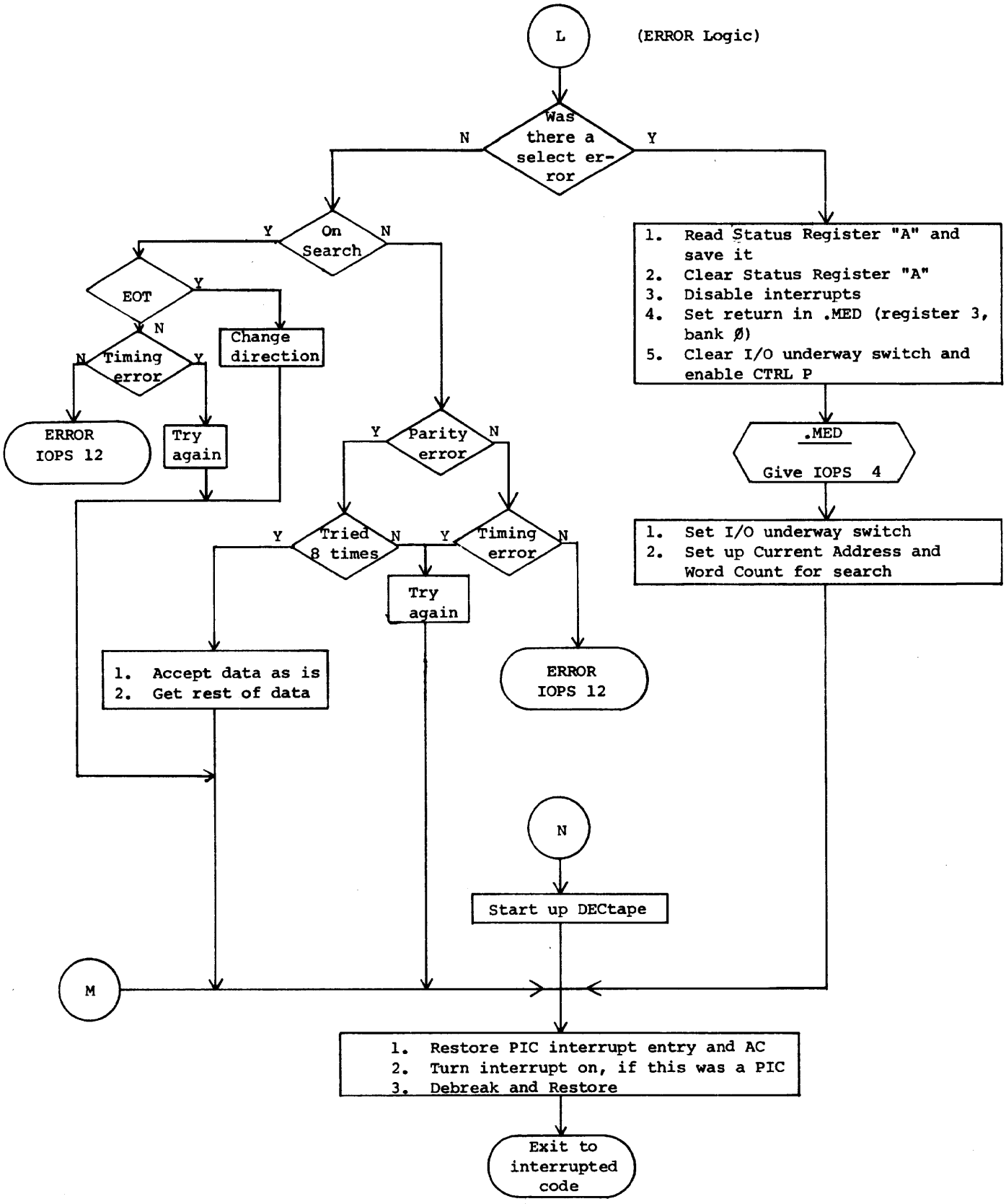


DECtape "A" Handler (DTA.)

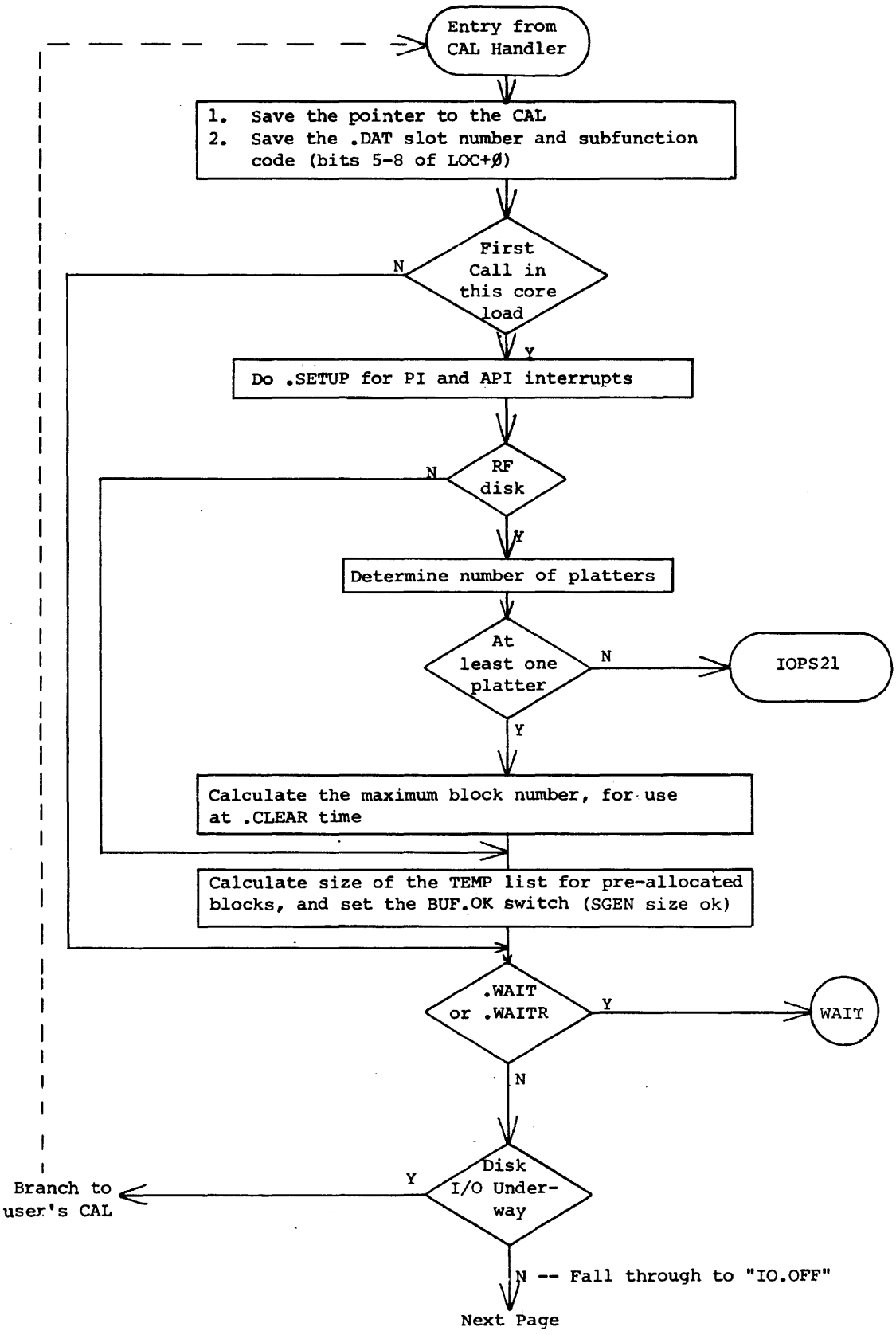


DECtape "A" Handler (DTA.)

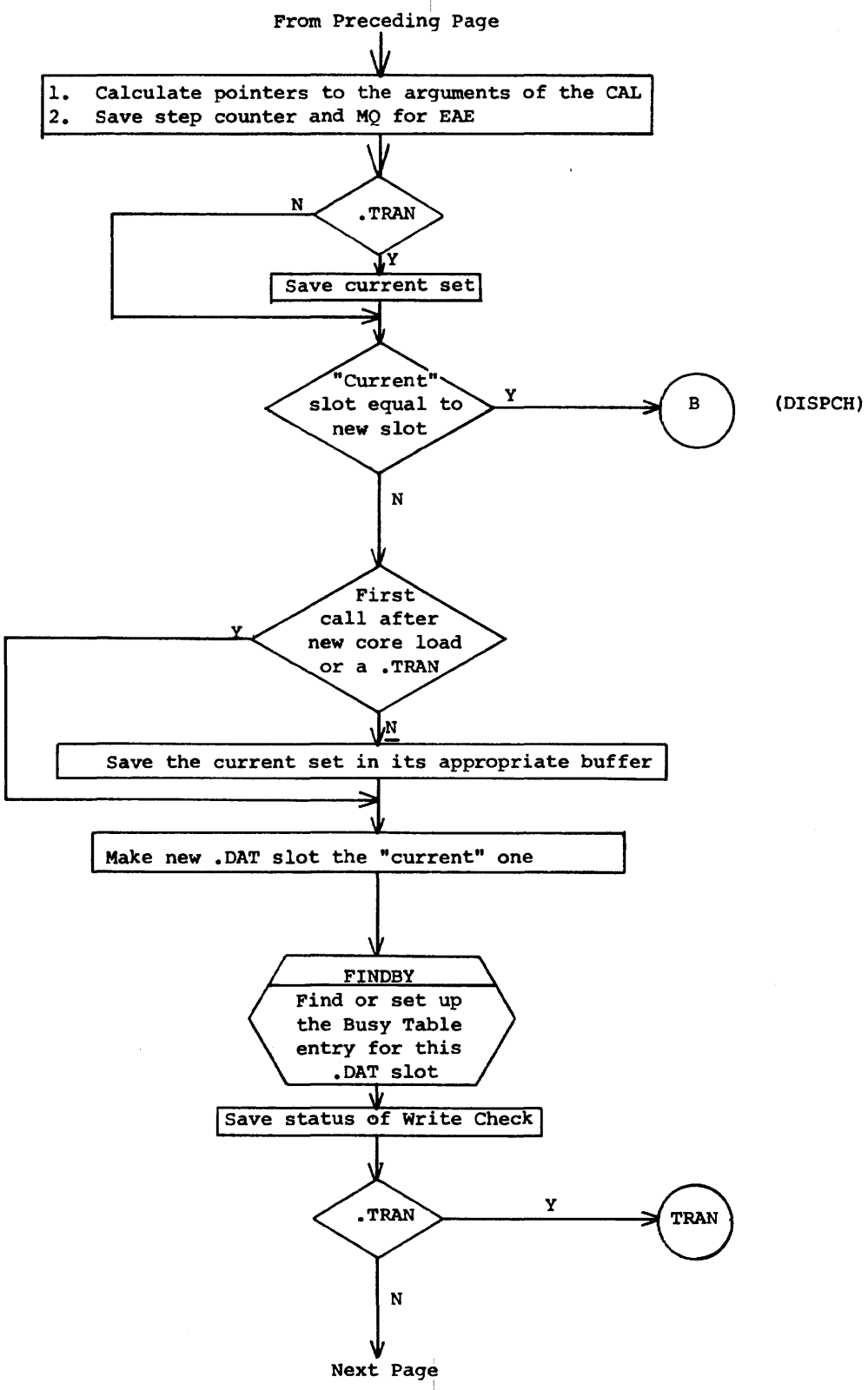

K
.TRAN
(Function Code 13)
I/O underway
Y
Loop back to user CAL
N
1. Set up block to transfer in or out
2. Set up core address-1
3. Set up word count
4. Start transfer
Exit to user after CAL
INTERRUPT SECTION
Entry from PI or API
Save information to restore later
I/O underway switch still on
Y
Stop tape
M
N
Read status register "B"
Error
Y
L
N
Was this a search
Y
N
Clear I/O underway switch
Correct block
Y
N
Was block 100 read in
N
Y
Right direction
Y
N
Change direction
Right direction
Y
N
Change direction
Set directory in core switch
Set up Current Address & word count for read or write
Set up current address and word count for search
N
N
DECtape "A" Handler (DTA.)



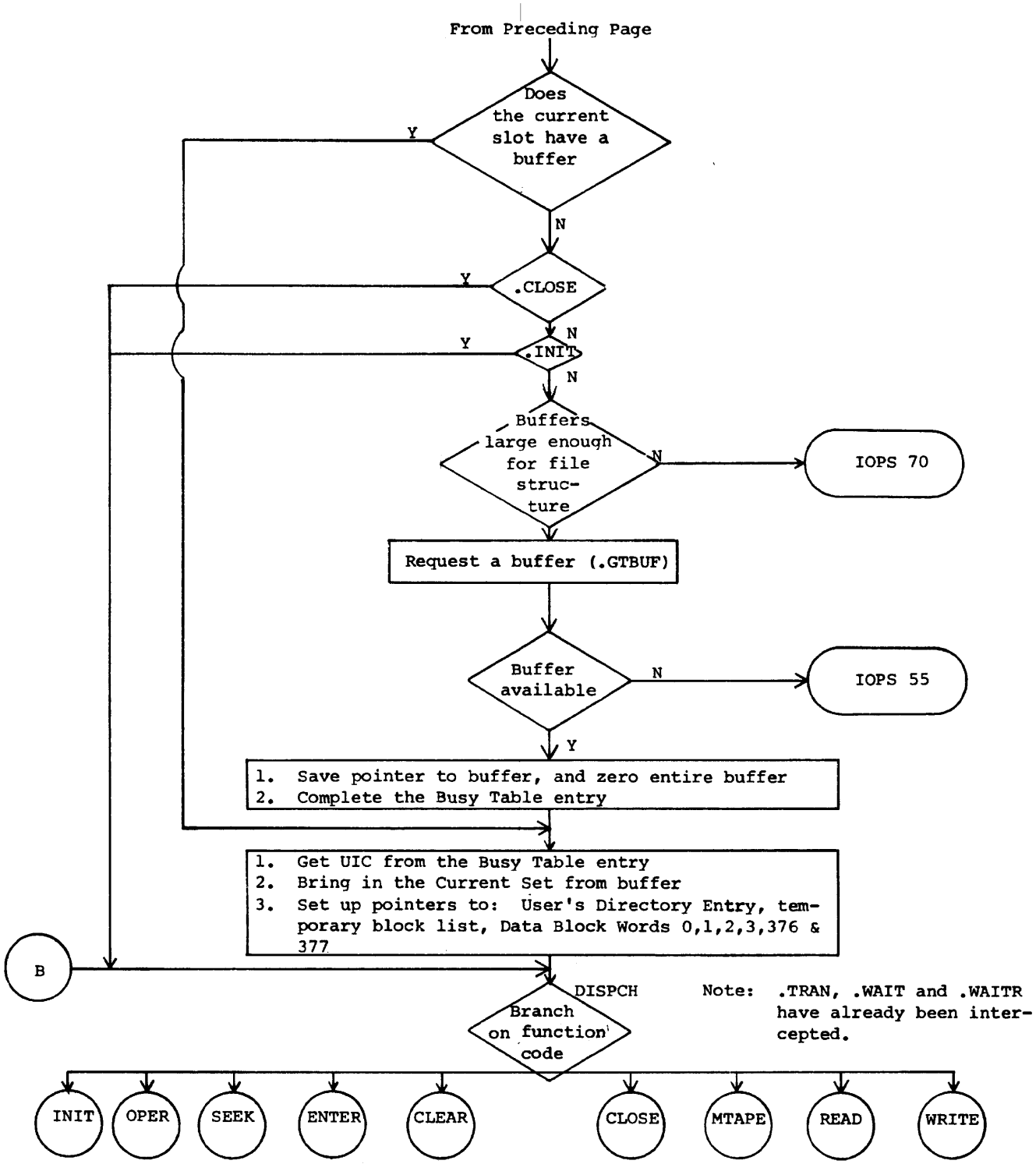
DECtape "A" Handler (DTA.)



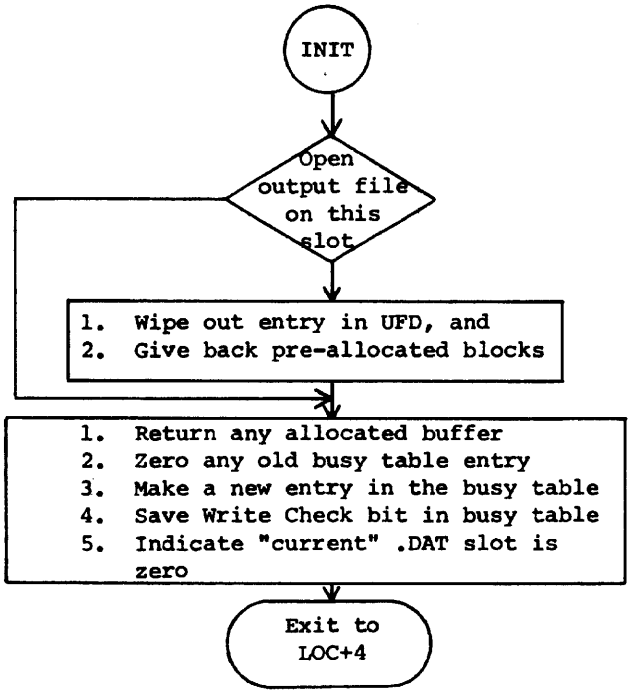
Disk "A" Handlers

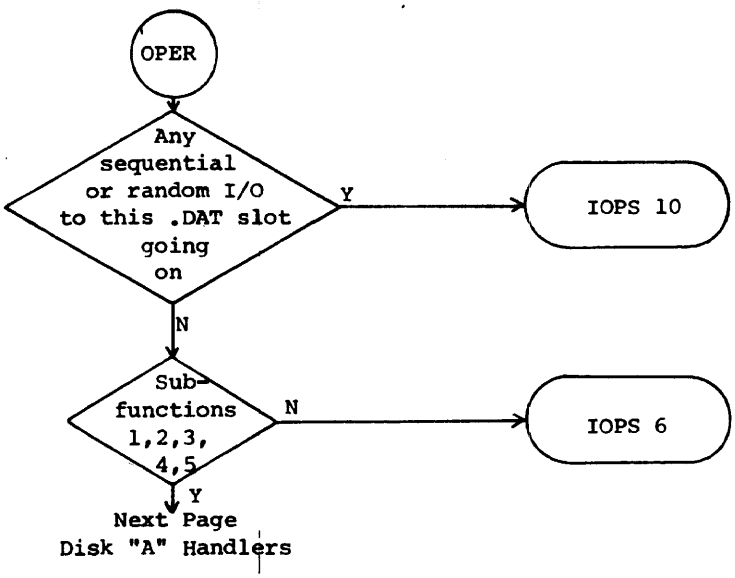


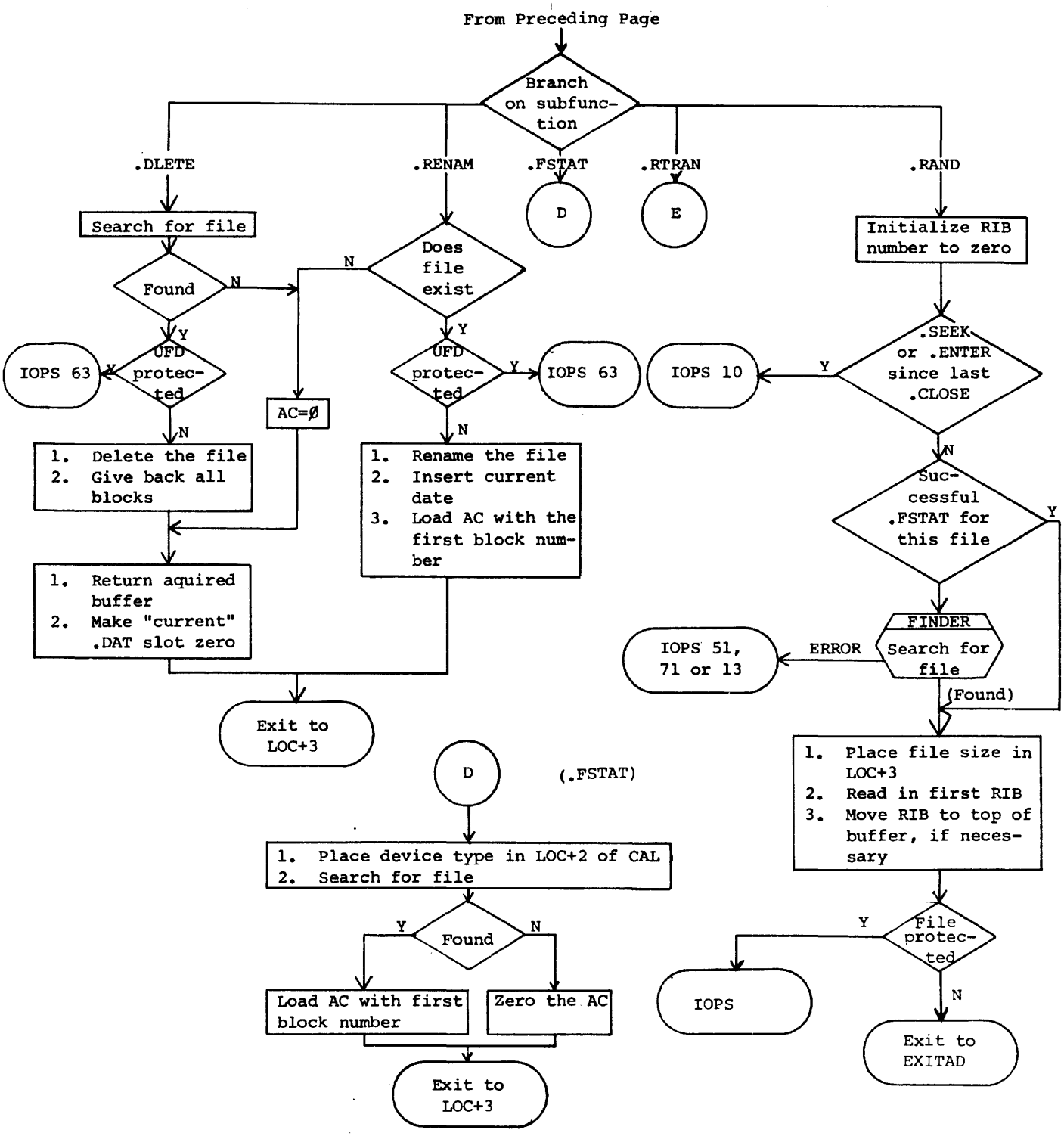
Disk "A" Handlers



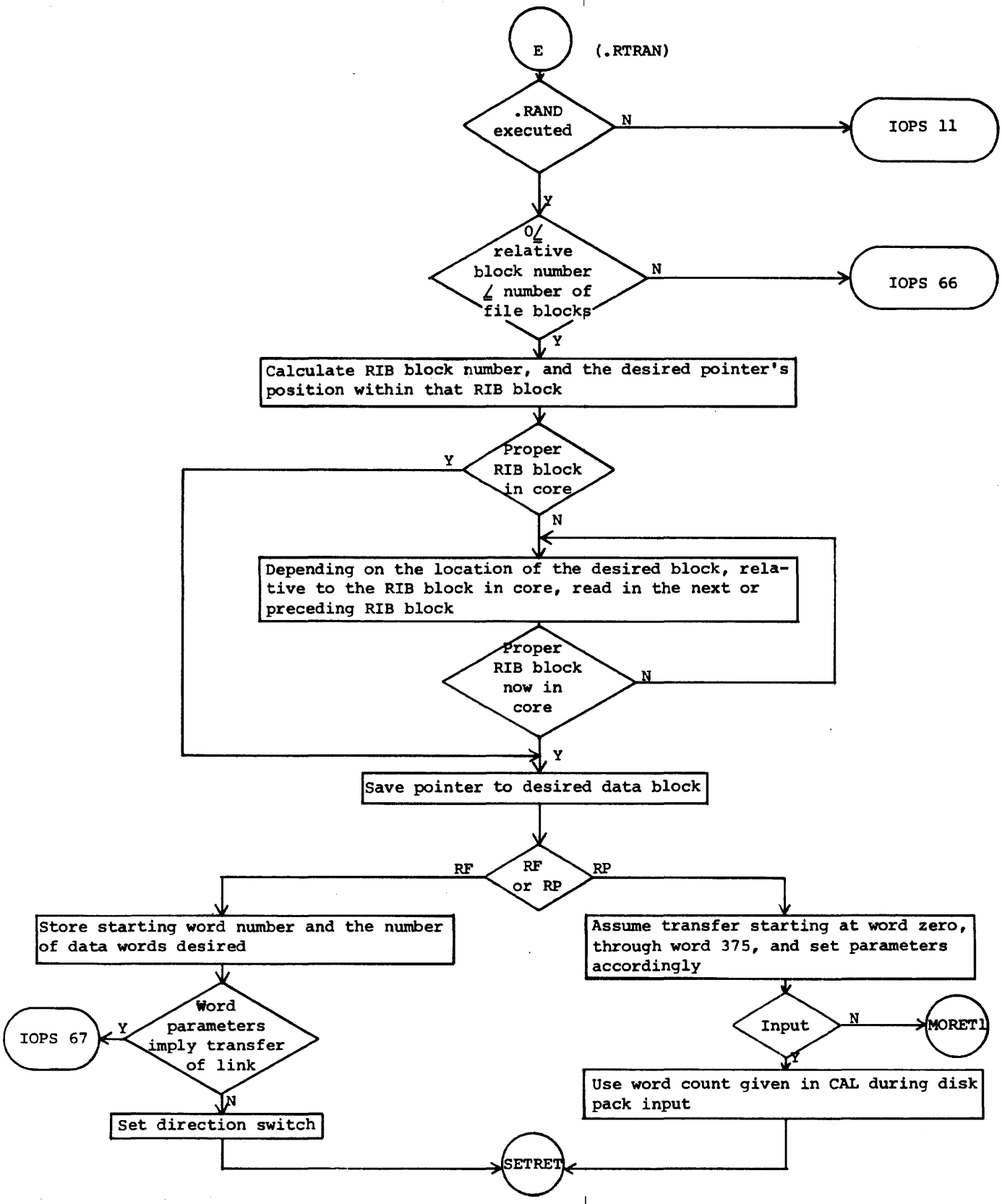
Disk "A" Handlers


INIT
Open output file on this slot
1. Wipe out entry in UFD, and
2. Give back pre-allocated blocks
1. Return any allocated buffer
2. Zero any old busy table entry
3. Make a new entry in the busy table
4. Save Write Check bit in busy table
5. Indicate "current" .DAT slot is zero
Exit to LOC+4

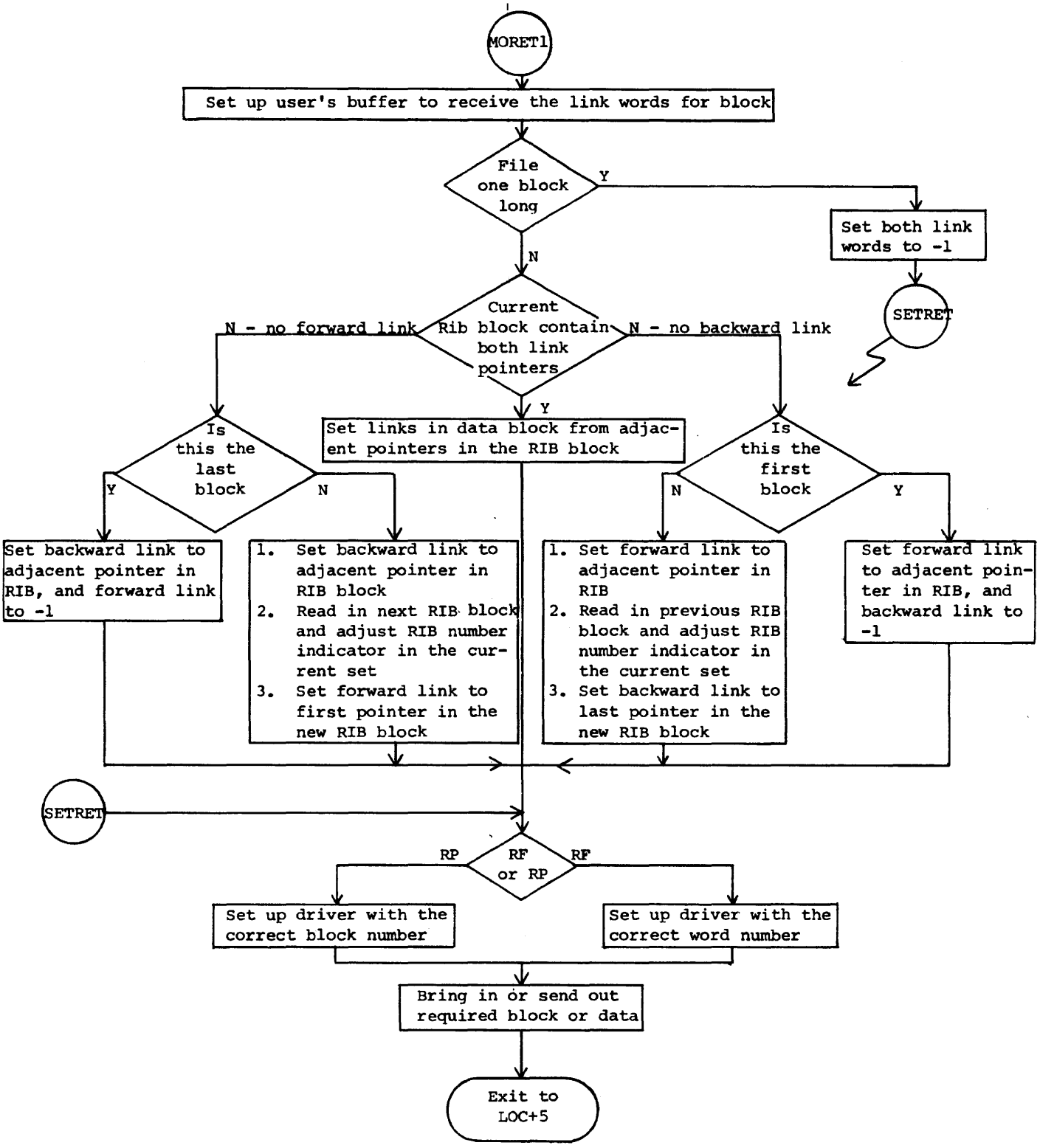

OPER
Any sequential or random I/O to this .DAT slot going on
Y
IOPS 10
N
Sub-functions 1,2,3, 4,5
N
IOPS 6
Y
Next Page
Disk "A" Handlers



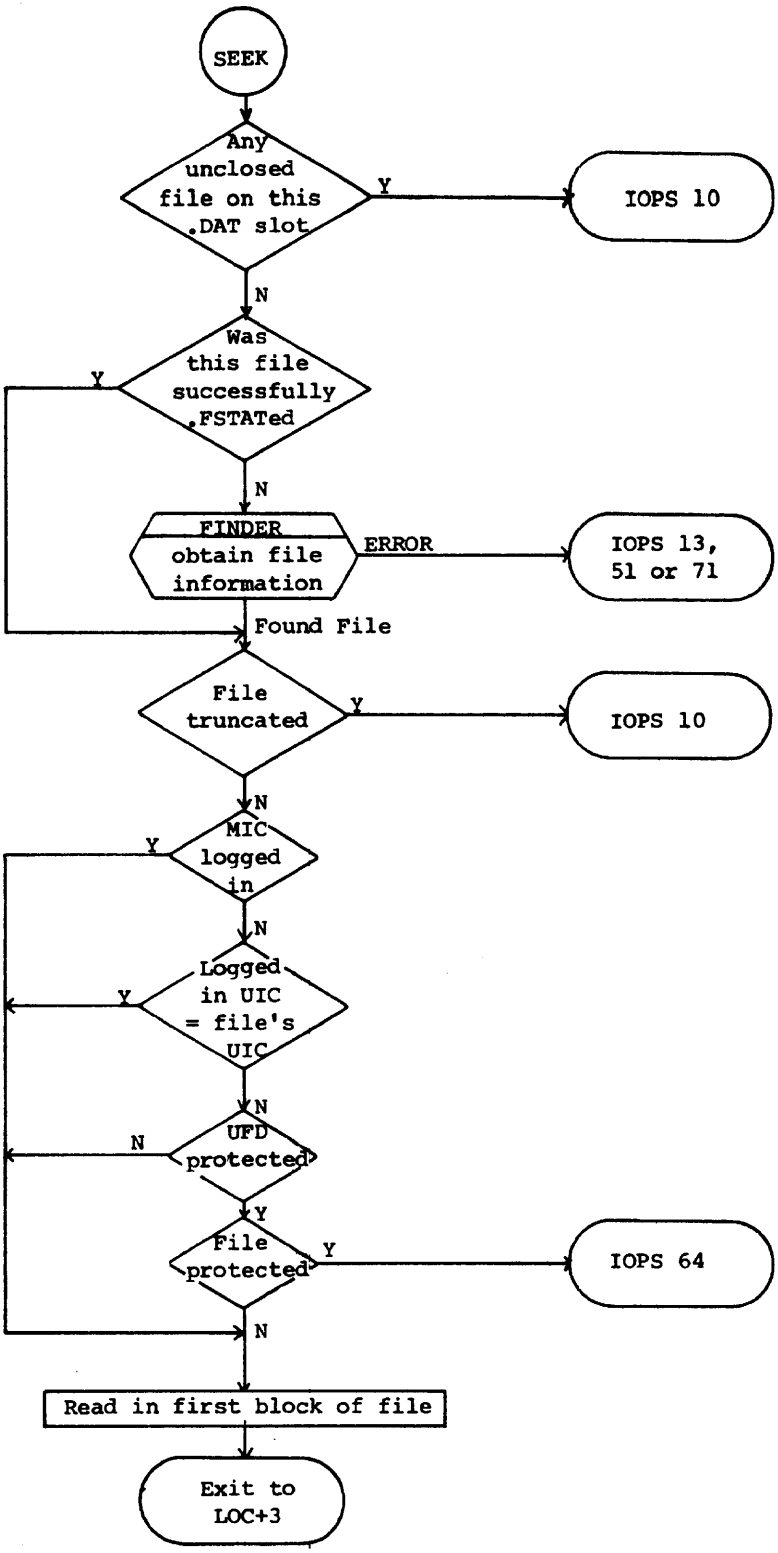
Disk "A" Handlers



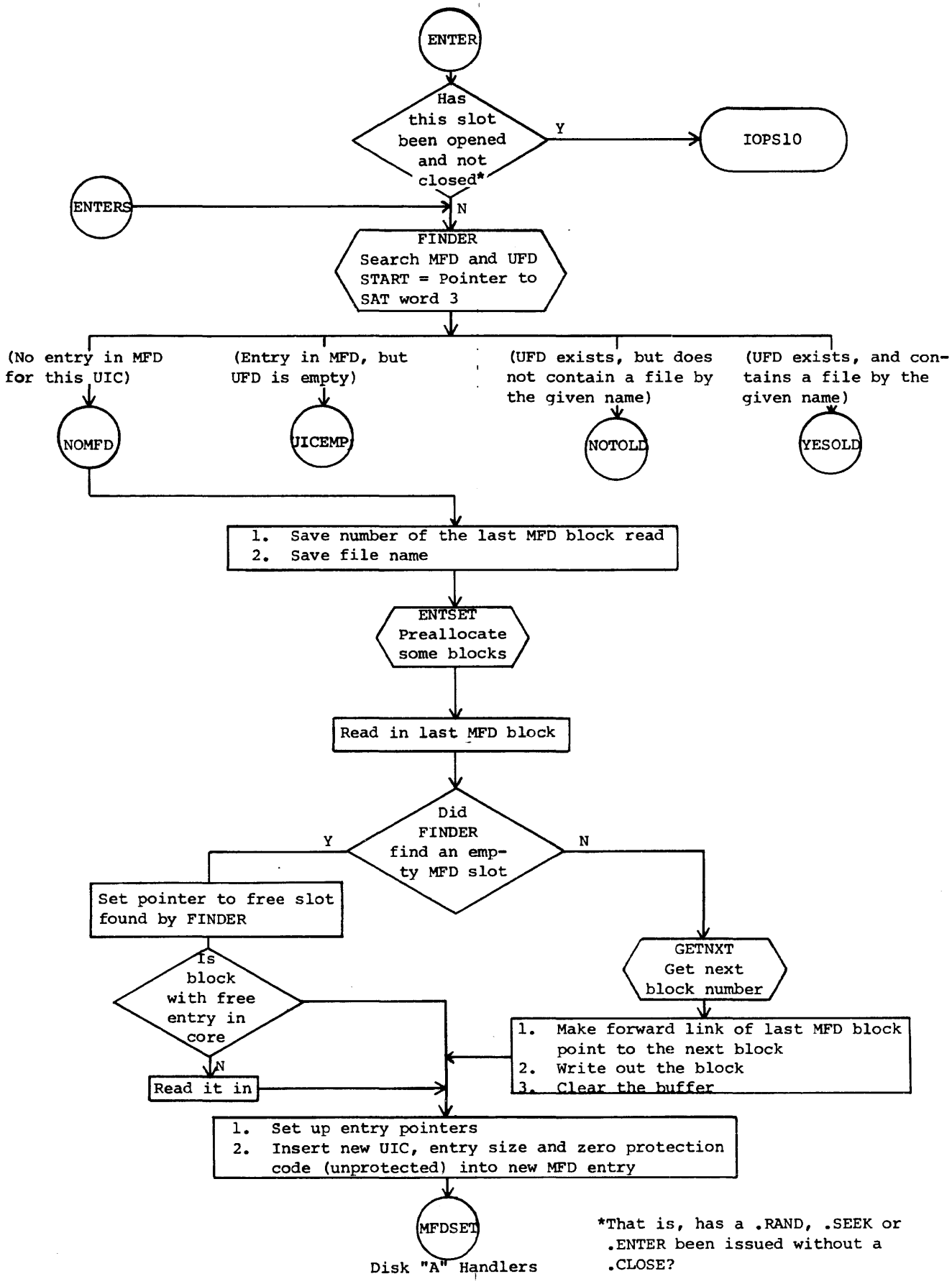
Disk "A" Handlers

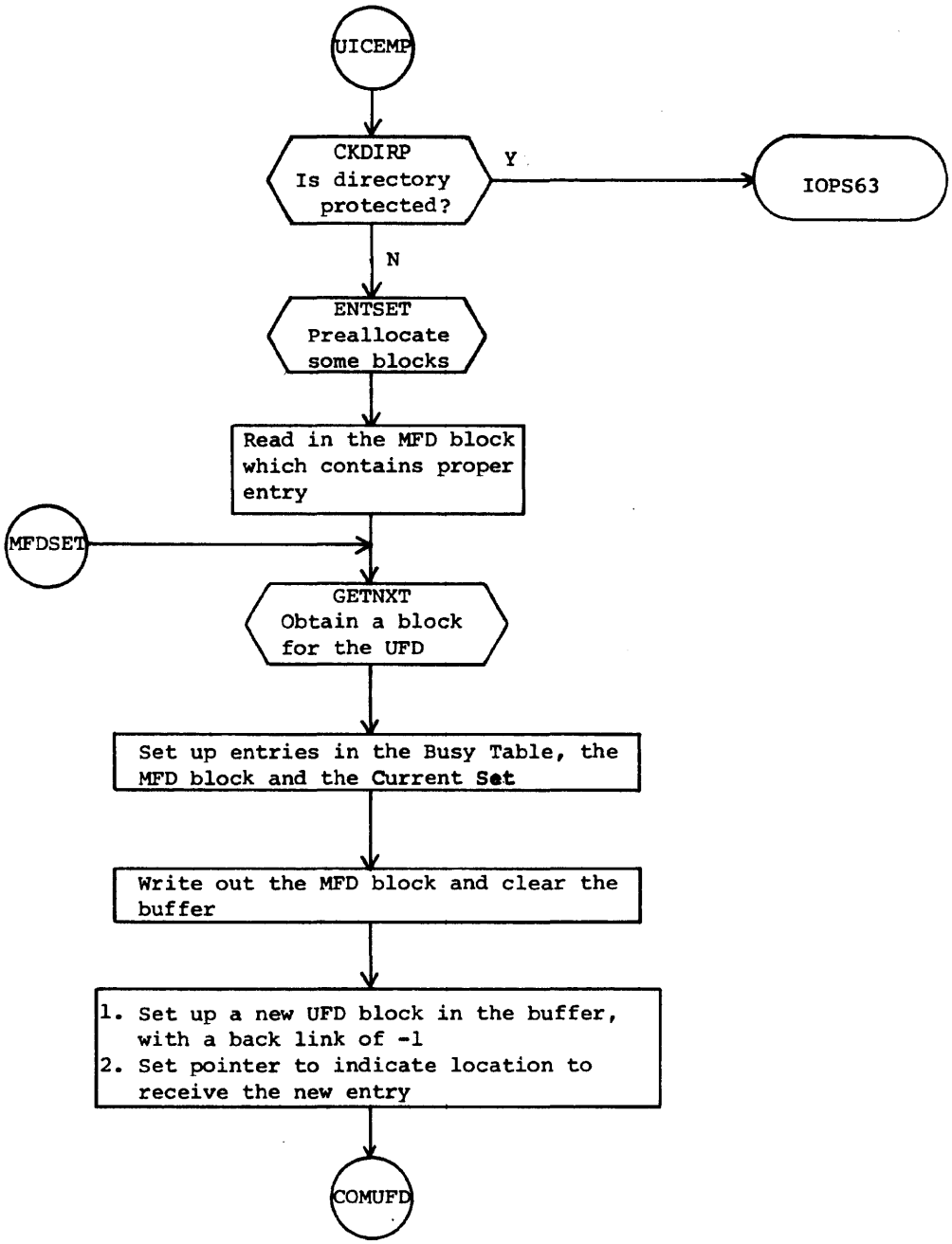


Disk "A" Handlers

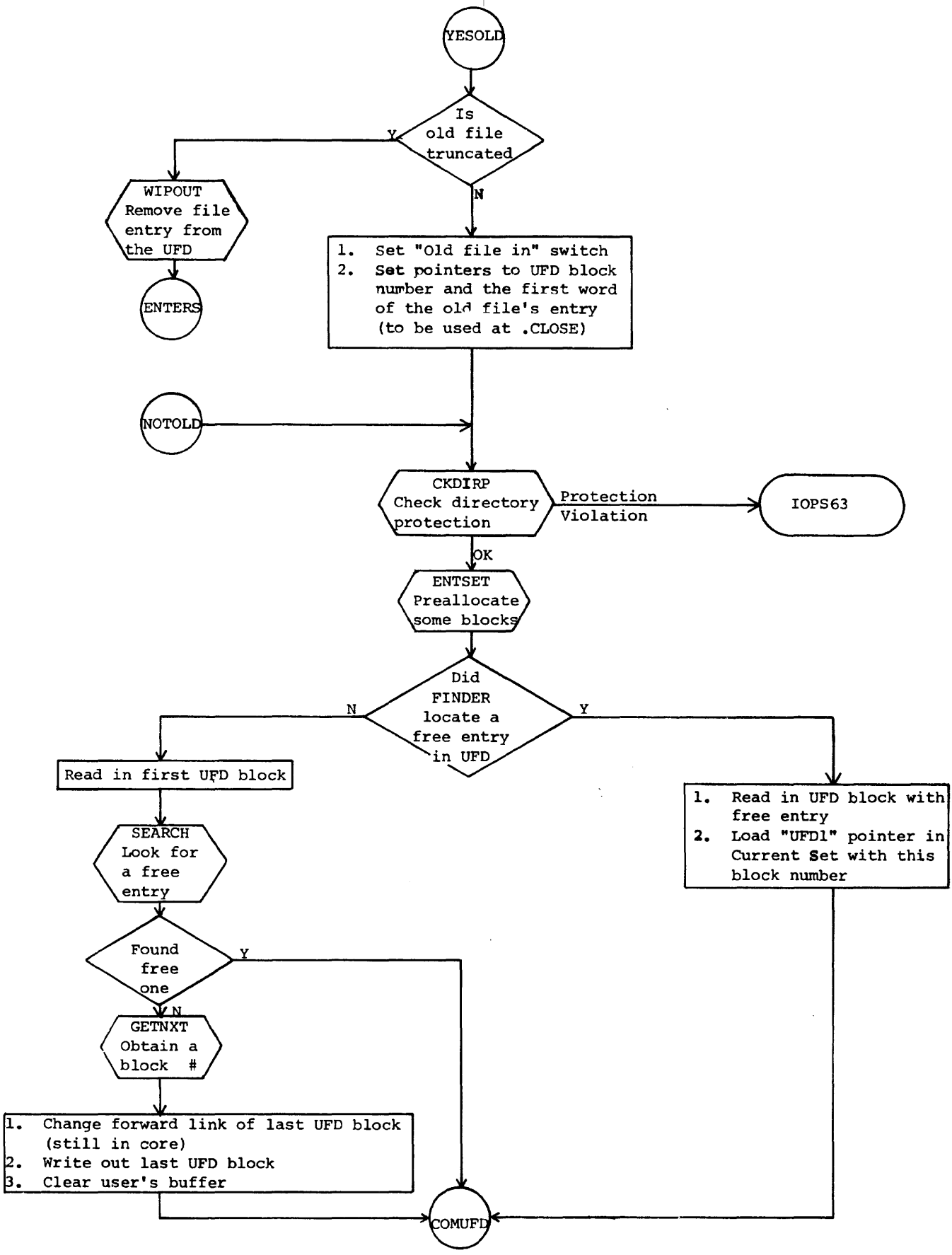


Disk "A" Handlers

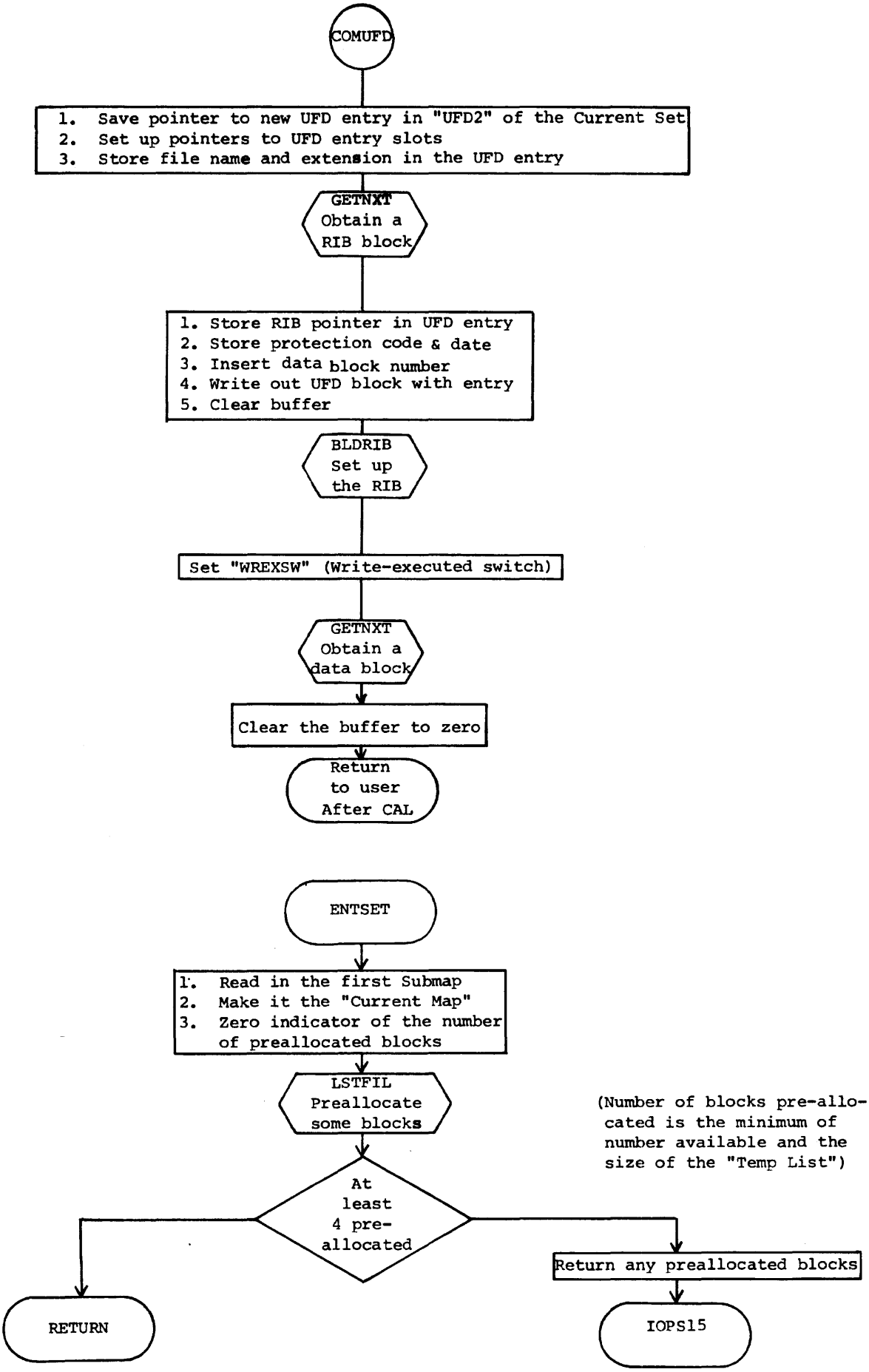

ENTER
Has this slot been opened and not closed*
Y
IOPS10
ENTERS
N
FINDER
Search MFD and UFD
START = Pointer to SAT word 3
(No entry in MFD for this UIC)
(Entry in MFD, but UFD is empty)
(UFD exists, but does not contain a file by the given name)
(UFD exists, and contains a file by the given name)
NOMFD
UICEMP
NOTOLD
YESOLD
1. Save number of the last MFD block read
2. Save file name
ENTSET
Preallocate some blocks
Read in last MFD block
Did FINDER find an empty MFD slot
Y
N
Set pointer to free slot found by FINDER
Is block with free entry in core
N
Read it in
GETNXT
Get next block number
1. Make forward link of last MFD block point to the next block
2. Write out the block
3. Clear the buffer
1. Set up entry pointers
2. Insert new UIC, entry size and zero protection code (unprotected) into new MFD entry
MFDSET
Disk "A" Handlers
*That is, has a .RAND, .SEEK or .ENTER been issued without a .CLOSE?



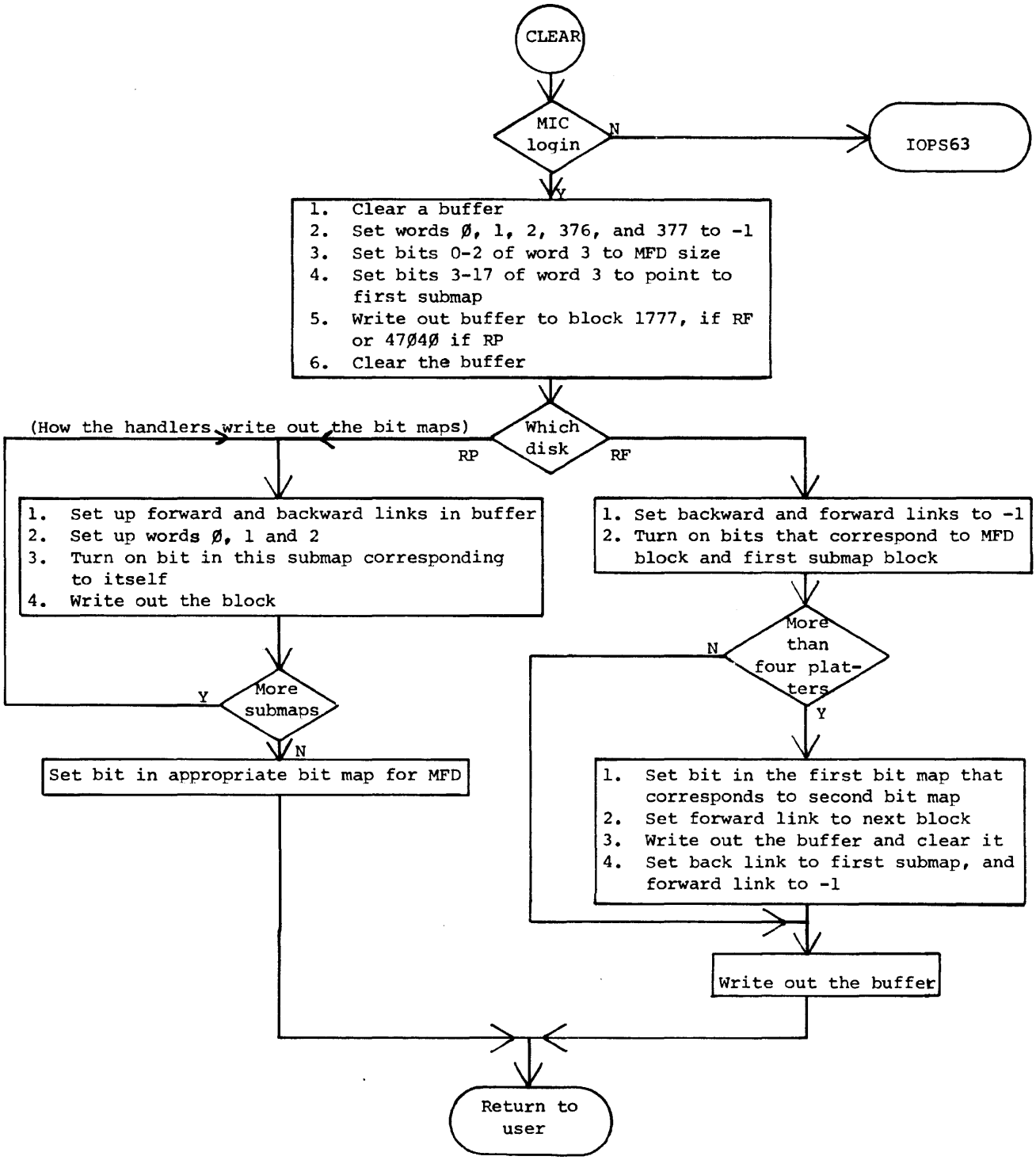
Disk "A" Handlers



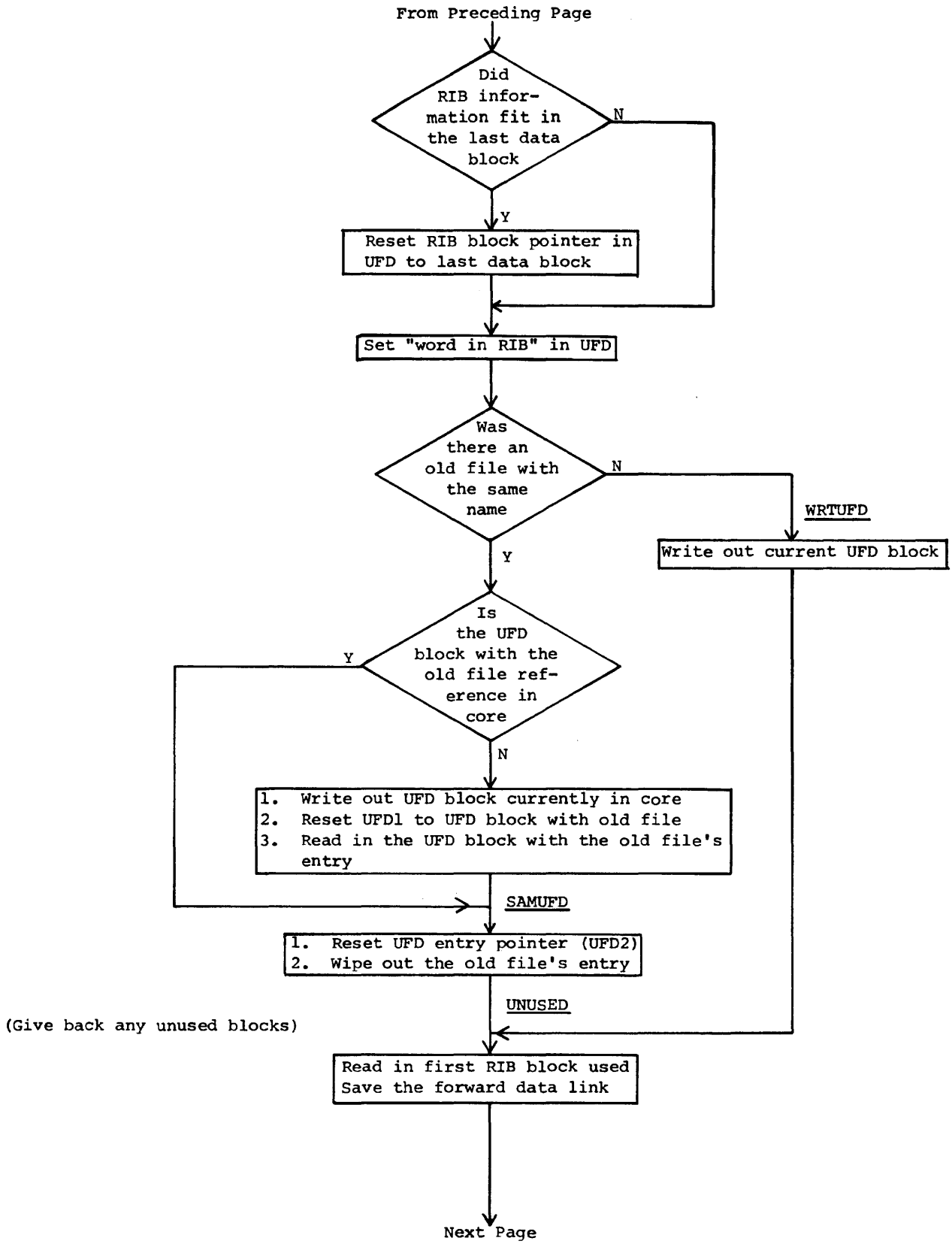
Disk "A" Handlers



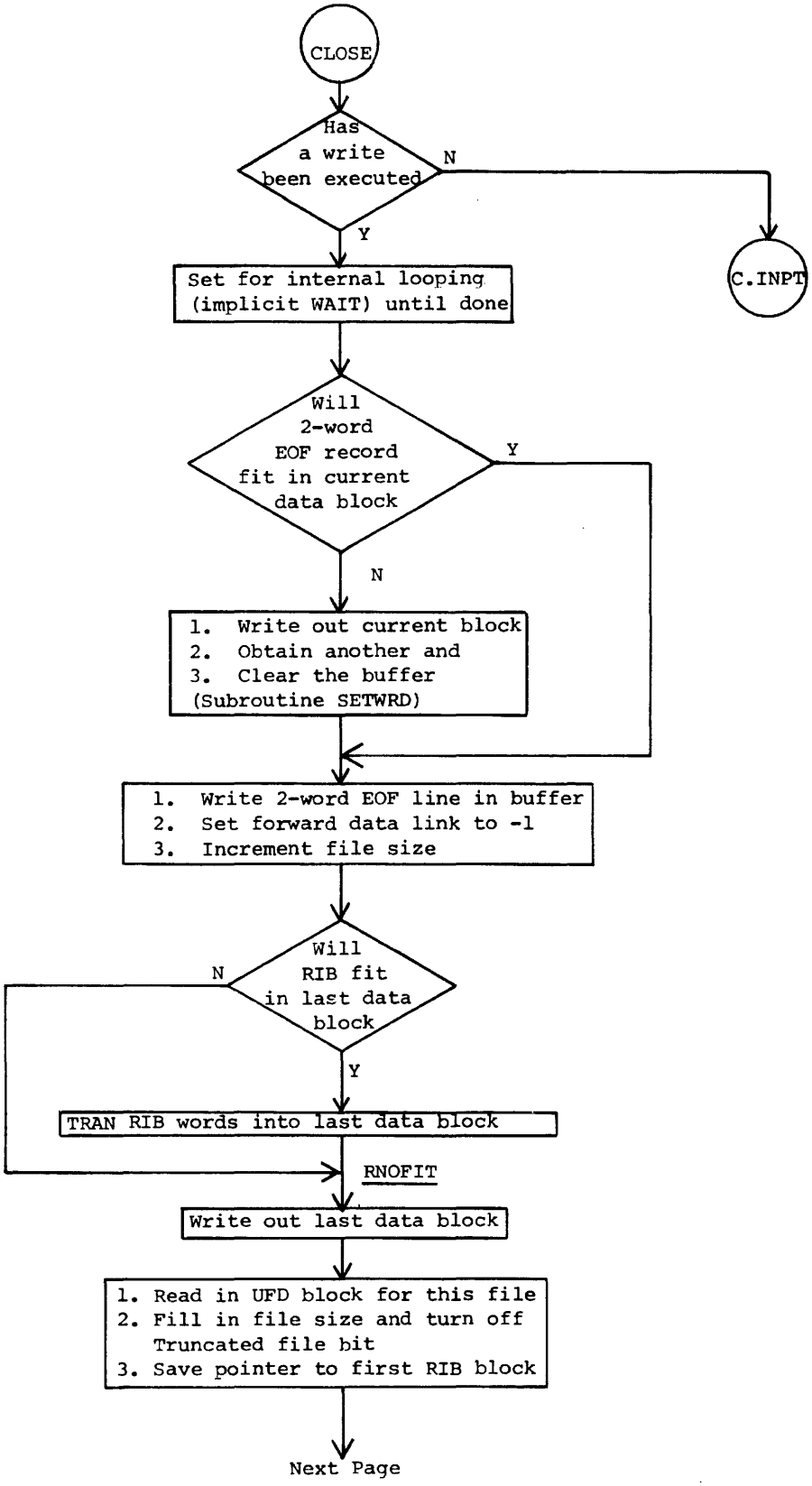
Disk "A" Handlers



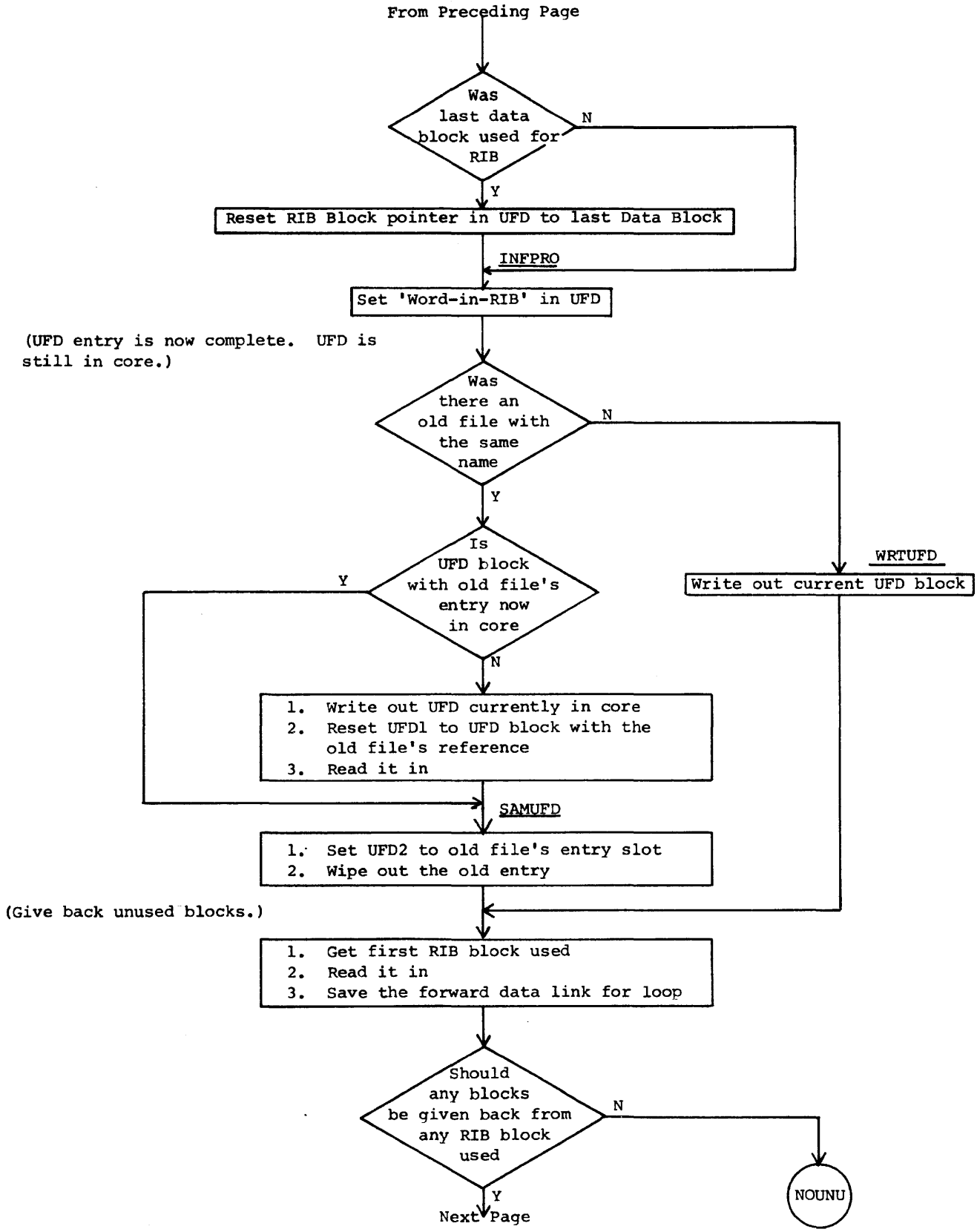
Disk "A" Handlers



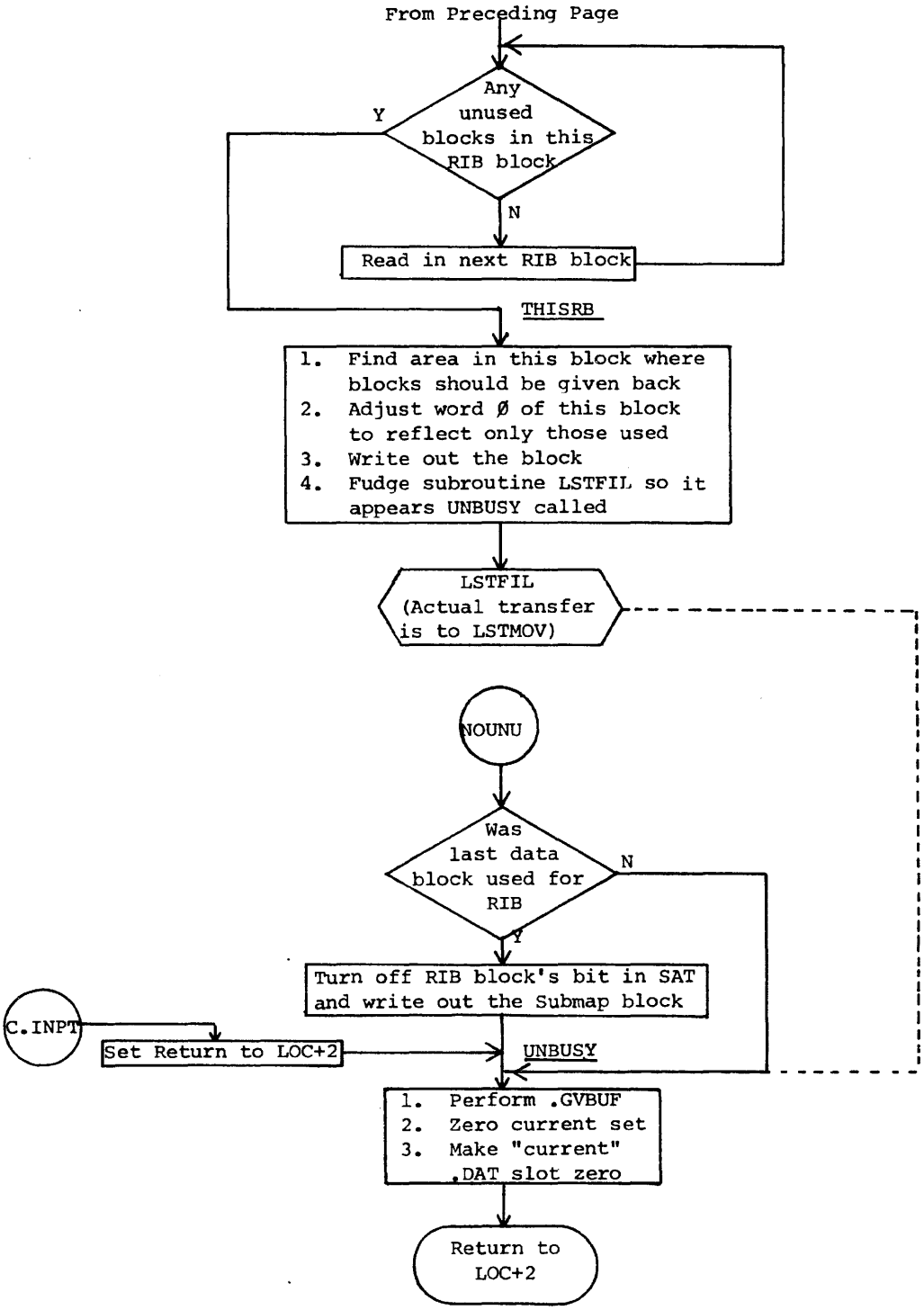
Disk "A" Handlers

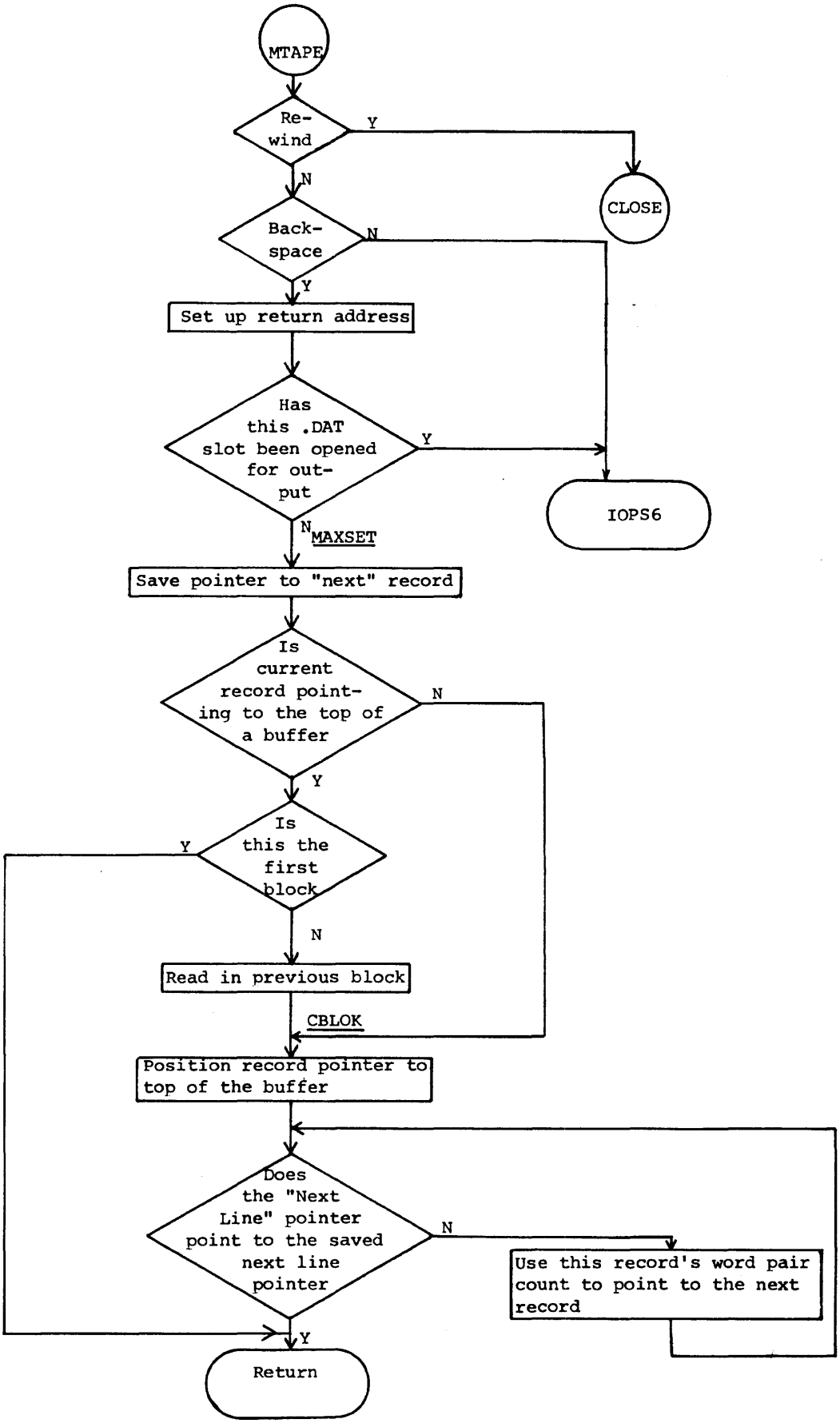


Disk "A" Handlers

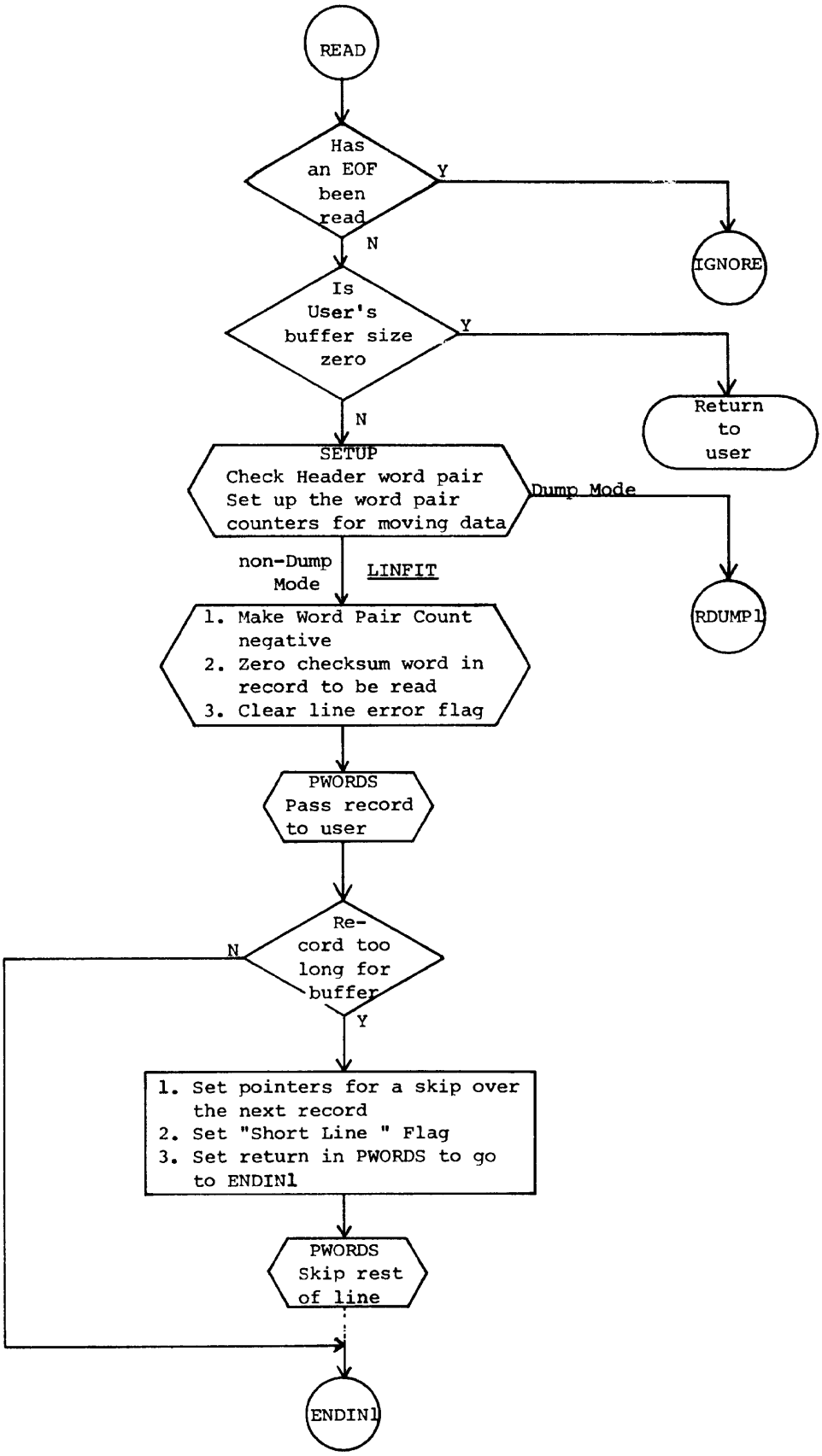


Disk "A" Handlers

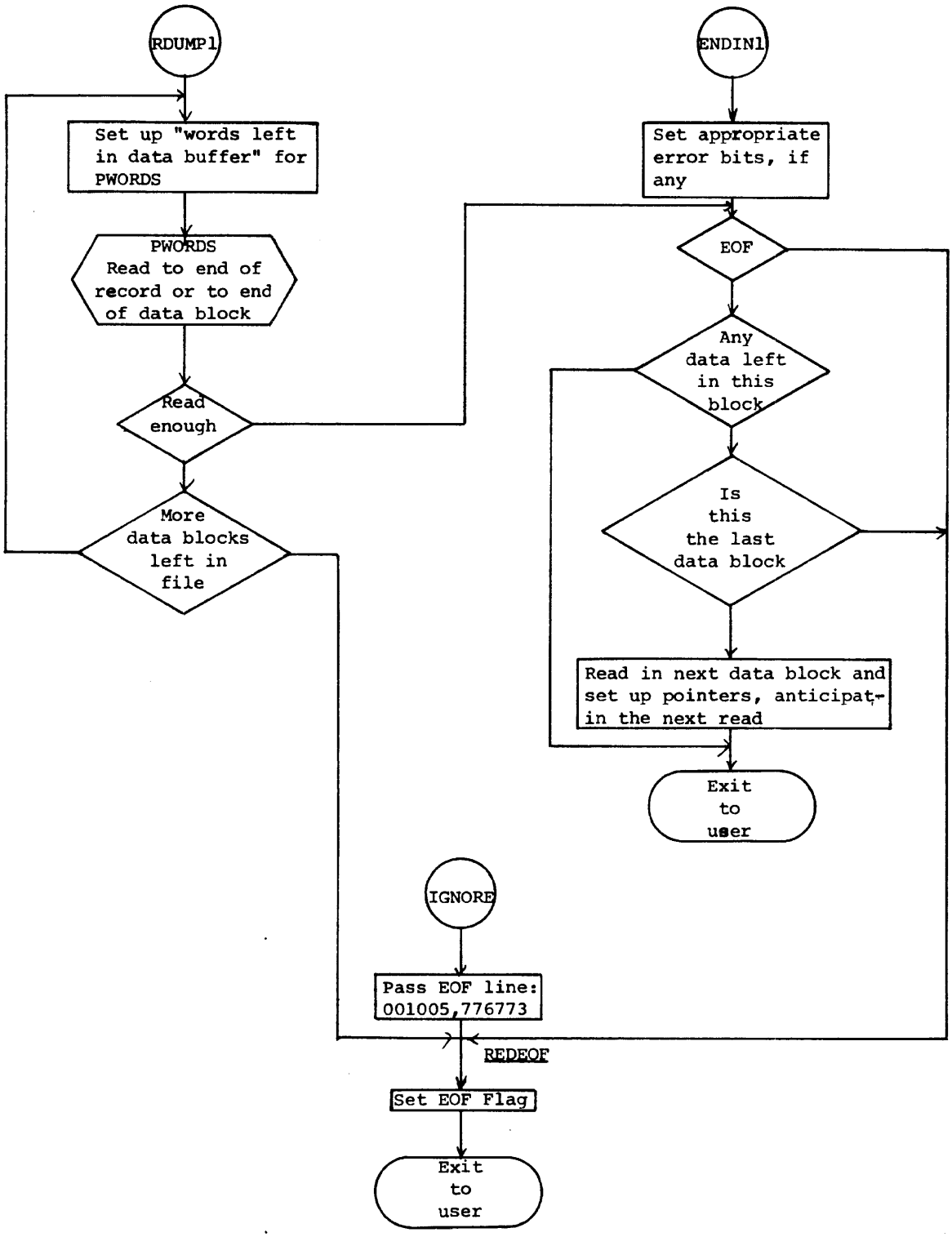

From Preceding Page
Any unused blocks in this RIB block
Y
N
Read in next RIB block
THISRB
1. Find area in this block where blocks should be given back
2. Adjust word Ø of this block to reflect only those used
3. Write out the block
4. Fudge subroutine LSTFIL so it appears UNBUSY called
LSTFIL (Actual transfer is to LSTMOV)
NOUNU
Was last data block used for RIB
N
Y
Turn off RIB block's bit in SAT and write out the Submap block
C.INPT
Set Return to LOC+2
UNBUSY
1. Perform .GVBUF
2. Zero current set
3. Make "current" .DAT slot zero
Return to LOC+2



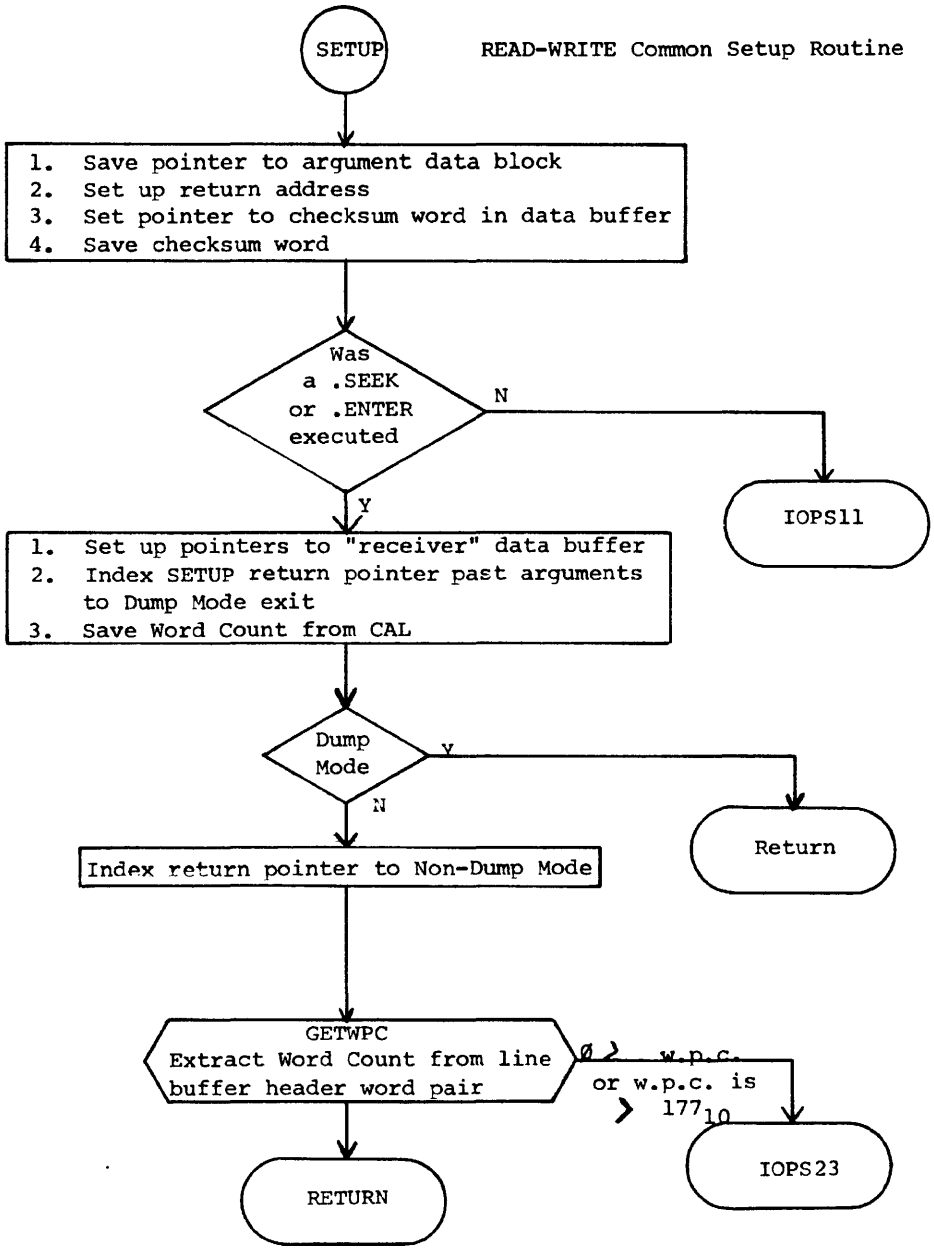
Disk "A" Handlers

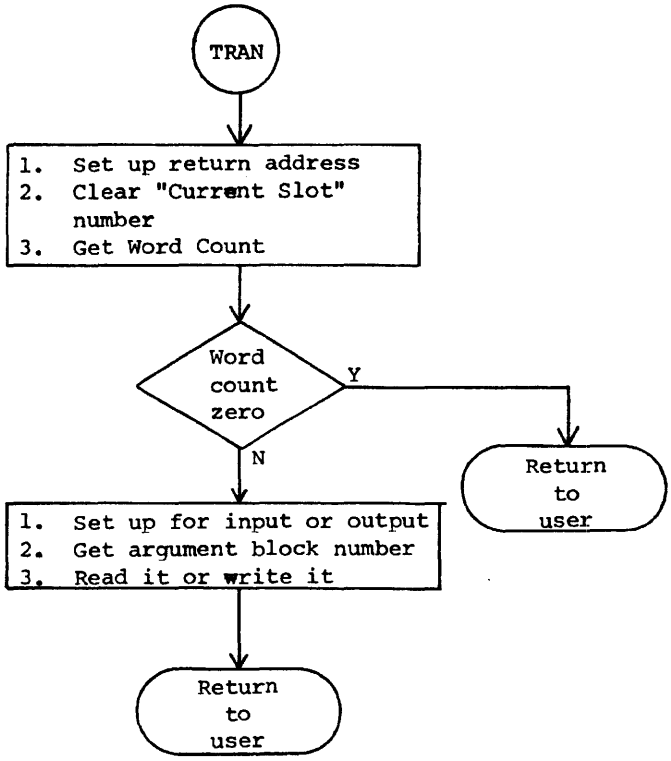


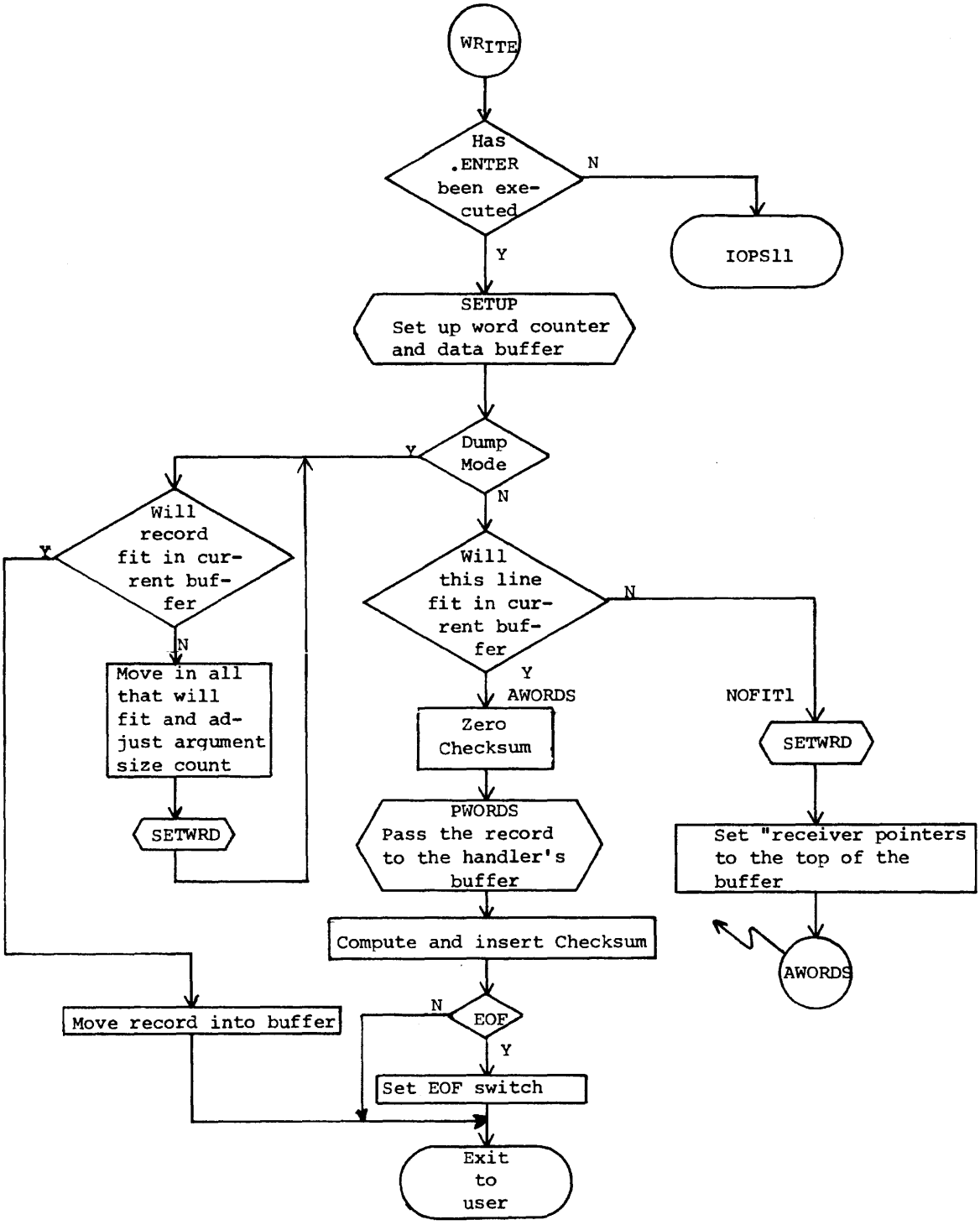
Disk "A" Handlers



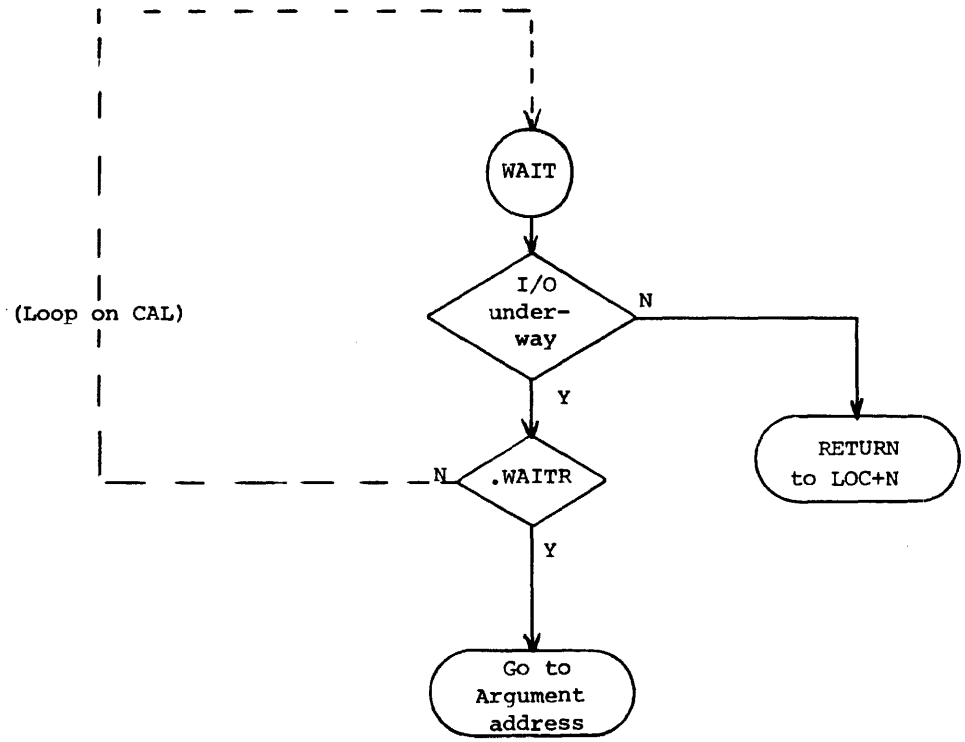
DISK "A" Handlers

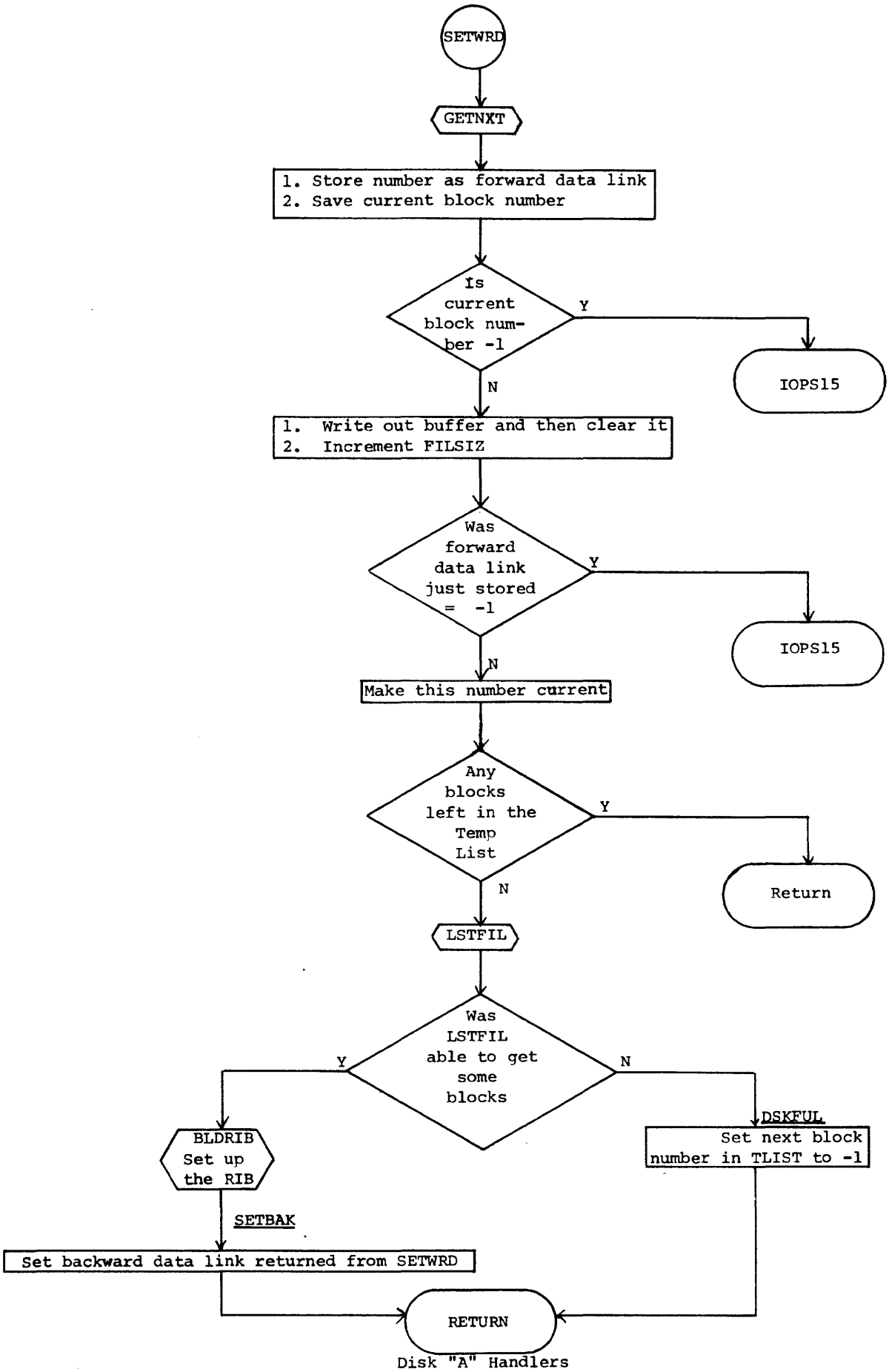

SETUP
READ-WRITE Common Setup Routine
1. Save pointer to argument data block
2. Set up return address
3. Set pointer to checksum word in data buffer
4. Save checksum word
Was a .SEEK or .ENTER executed
N
Y
IOPS11
1. Set up pointers to "receiver" data buffer
2. Index SETUP return pointer past arguments to Dump Mode exit
3. Save Word Count from CAL
Dump Mode
Y
N
Return
Index return pointer to Non-Dump Mode
GETWPC
Extract Word Count from line buffer header word pair
Ø > w.p.c.
or w.p.c. is > 177₁₀
IOPS23
RETURN

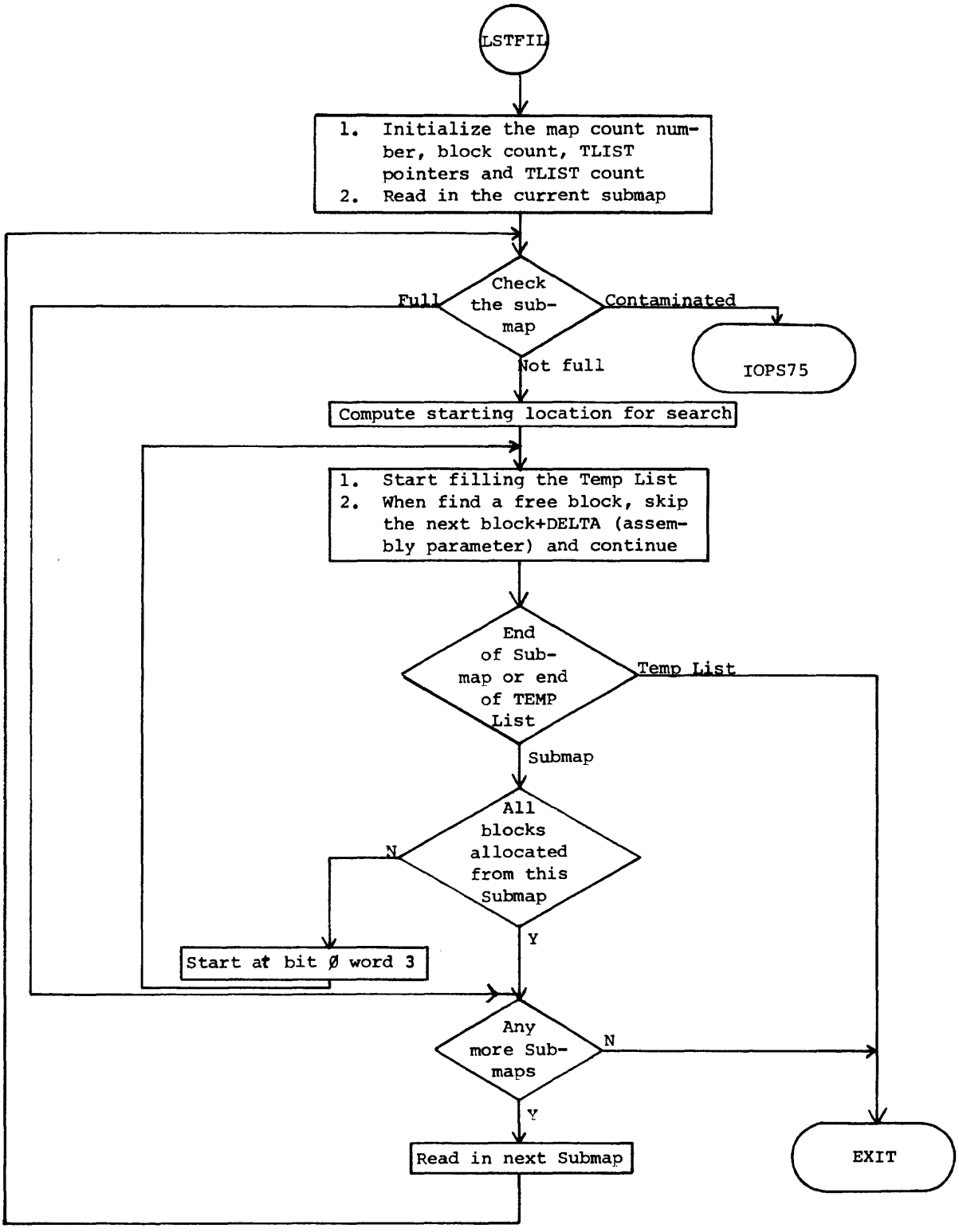

TRAN
1. Set up return address
2. Clear "Current Slot" number
3. Get Word Count
Word count zero
Y
N
Return to user
1. Set up for input or output
2. Get argument block number
3. Read it or write it
Return to user



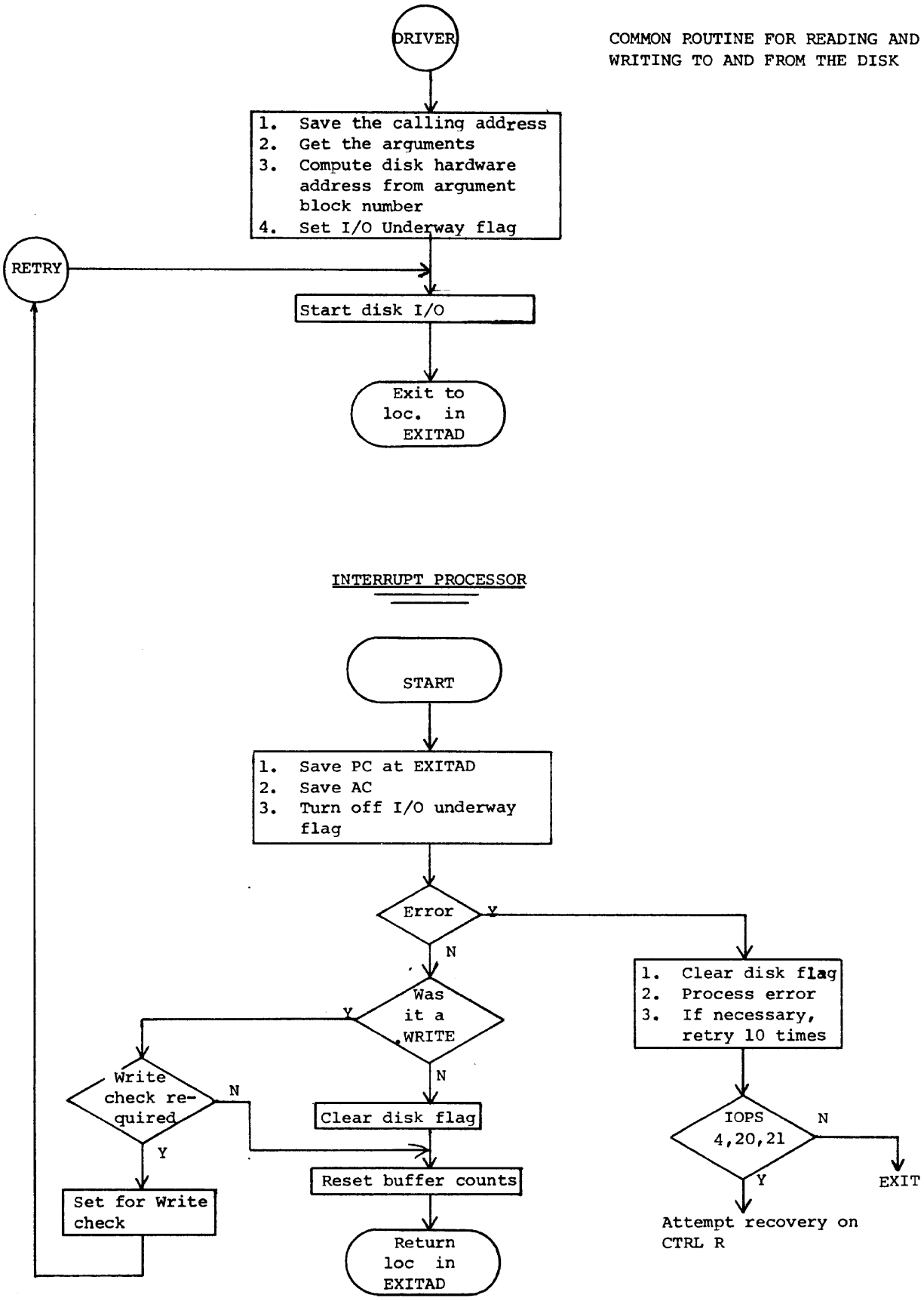
Disk "A" Handlers


(Loop on CAL)
WAIT
I/O under-way
N
Y
RETURN to LOC+N
N
.WAITR
Y
Go to Argument address


SETWRD
GETNXT
1. Store number as forward data link
2. Save current block number
Is current block number -1
Y
IOPS15
N
1. Write out buffer and then clear it
2. Increment FILSIZ
Was forward data link just stored = -1
Y
IOPS15
N
Make this number current
Any blocks left in the Temp List
Y
Return
N
LSTFIL
Was LSTFIL able to get some blocks
Y
N
BLDRIB
Set up the RIB
DSKFUL
Set next block number in TLIST to -1
SETBAK
Set backward data link returned from SETWRD
RETURN
Disk "A" Handlers



Disk "A" Handlers



Disk "A" Handlers

APPENDIX C

ASG

```
1 ASSIGN DEVICE UIC TO .DAT
A @D00(@D11()@)@ <@U00(@D12()@)@> @A00()@
```

ASM

```
2 MACRO AND LINE EDITOR
A @D00(@D11()@)@ <@U00(@D12()@)@> -14/@D03(@D11()@)@ <@U03(@D12()@)@> -15
B,PRE
@A00()@
@A03(@A00(FILTMP)@)@
A @D00(@D11()@)@ <@U00(@D12()@)@> -11/@D01(@D11()@)@ <@U01(@D12()@)@> -10
A @D02(@D11()@)@ <@U02(@D12()@)@> -14/@D03(@D11()@)@ <@U03(@D12()@)@> -13
A @D04(LP)@ <@U04(@D12()@)@> -12
MACRO
@O(BL)@←@A00(FILTMP)@@D14()@
```

BNK

```
2 BANK MODE OPERATION-ON
BANK ON
```

BUF

```
2 NUMBER OF BUFFERS
BUFFS @A00()@
```

CHN

```
1 SPECIFY 7 OR 9 TRACK MAGTAPE
C @A00()@
```

CMP

```
1 SOURCE COMPARE
A @D00(@D11()@)@ <@U00(@D12()@)@> -15/@D01(@D11()@)@ <@U01(@D12()@)@> -14
SRCCOM
@O()@←@A00()@/@A01()@@D14()@
```

DIR

```
1 LIST DIRECTORY
PIP
L LP←@A00(@D11()@)@ <@U00(@D12()@)@>@D14()@
```

```
 DLG

2 LOGOUT UIC
LOGOUT

 DMP

13 DUMP UTILITY - DIRECT SUB FILE
A @D00(@D11()@)@ <@U00(@D12()@)@> -14/@D01(LP)@ <@U01(@D12()@)@> -12
@A00(ALL)@@D14()@



DOS

  1,3, GENERAL PRC FILE FOR GIVING COMMAND STRINGS
  @A00(@D14()@)@

 FIL

 2 CREATE A FILE FROM CARDS/EDITOR
 A @D00(@D11()@)@ <@U00(@D12()@)@> -14
 A @D01(@D11()@)@ <@U01(@D12()@)@> -15
 B,PRE
 @A00(FILTMP)@
 @A01(@A00(FILTMP)@)@

 FOR

2 FORTRAN IV AND LINE EDITOR
A @D00(@D11()@)@ <@U00(@D12()@)@> -14/@D01(@D11()@)@ <@U01(@D12()@)@> -15
B,PRE
@A00()@
@A01(@A00(FILTMP)@)@
A @D00(@D11()@)@ <@U00(@D12()@)@> -11/@D01(@D11()@)@ <@U01(@D12()@)@> -13
A @D02(LP)@ <@U02(@D12()@)@> -12
F4
@O(BL)@←@A00(FILTMP)@@D14()@


 JOB

2 START NEW JOB
LOG JOB @A00()@ BEGIN @D14()@
T
LOGIN @A02(SCR)@
A NON 2,3,4,7,10,11,12,13,14,15,16,17,20/@D11()@ 1
PIP
N @D11()@ <SCR>@D14()@
@A03()@
KEEP @A04(OFF)@
TIMEST @A01(1)@:00
```

```
KEP

 1 RETAIN DEVICE ASSIGNMENTS
 KEEP @A00()@

LCM

 13 SUPPLEMENT TO LIB PRC-UPDATE ,LIBR
 @A00(CLOSE@D13()@)@ @A01(@D13()@)@ @A02()@


 LIB

1
A @D00(@D11()@)@ <@U00(@D12()@)@> -14
A @D01(@D00(@D11()@)@)@ <@U01(@U00(@D12()@)@)@> -15
A @D02(@D11()@)@ <@U02(@D12()@)@> -10
A @D03(LP)@ <@U03(@D12()@)@> -12
UPDATE
@D(LUS)@←@A00(,LIBR)@@D14()@


  LNK

13 DIRECT SUB FILE - BUILDS LINKS FOR EXECUTE FILE-USE WITH OVL PRC
@A00(@D14()@)@@D14()@


 LOG

2 LOGIN UIC
LOGIN @A00(SCR)@


LST

2 LIST CONTENTS OF FILE ON LINE PRINTER
PIP
T LP←@D00(@D11()@)@ <@U00(@D12()@)@> @A00(FILTMP)@ (A)@D14()@


 MAP

1;3 DIRECT SUB FILE FOR CHAIN OPTION AND RES CODE ONLY
CHAIN
@A00(TMPXCT)@@D14()@
@A01(SZ)@@D14()@
@A02(FILTMP)@@D14()@
@D14()@


MIC

2 LOGIN MIC UIC
MICLOG @A00()@
```

```
MNT

1 MOUNT TAPE# ON DRIVE #
LOGW MOUNT @D(D)@-TAPE# @A00()@ ON DRIVE# @A01()@ - WRITE @A02(LOCK)@


MSG

13 MESSAGE TO OPERATOR-DIRECT SUB FILE
LOG @A00()@


  MSW

13 MESSAGE TO OPERATOR W/WAIT-DIRECT SUB
LOGW @A00()@


NDR

  1 CREATE NEW DIRECTORY
  PIP
  N @A00(@D11()@)@ <@U00(@D12()@)@>@D14()@


 OVL

13 DIRECT SUB FILE - USE FOR BUILDING OVERLAYS(CHAIN)
CHAIN
@A00(TMPXCT)@@D14()@
@A01(SZ)@@D14()@
@A02(FILTMP)@@D14()@


PAG

 2 PAGE MODE OPERATION-ON
 PAGE ON


 PCD

2 SPECIFY PROTECTION CODE
P @A00(3)@


QDP

1 DUMP CORE ON TERMINAL ERRORS-NO ARGUMENTS
QDUMP


 XCT

2 EXECUTE
A @D00(@D11()@)@ <@U00(@D12()@)@> -4
E @A00(TMPXCT)@
```

HOW TO OBTAIN SOFTWARE INFORMATION

Announcements for new and revised software, as well as programming notes, software problems, and documentation corrections are published by Software Information Service in the following newsletters.

Digital Software News for the PDP-8 & PDP-12
Digital Software News for the PDP-11
Digital Software News for the PDP-9/15 Family

These newsletters contain information applicable to software available from Digital's Program Library, Articles in Digital Software News update the cumulative Software Performance Summary which is contained in each basic kit of system software for new computers. To assure that the monthly Digital Software News is sent to the appropriate software contact at your installation, please check with the Software Specialist or Sales Engineer at your nearest Digital office.

Questions or problems concerning Digital's Software should be reported to the Software Specialist. In cases where no Software Specialist is available, please send a Software Performance Report form with details of the problem to:

Software Information Service
Digital Equipment Corporation
146 Main Street, Bldg. 3-5
Maynard, Massachusetts 01754

These forms which are provided in the software kit should be fully filled out and accompanied by teletype output as well as listings or tapes of the user program to facilitate a complete investigation. An answer will be sent to the individual and appropriate topics of general interest will be printed in the newsletter.

Orders for new and revised software and manuals, additional Software Performance Report forms, and software price lists should be directed to the nearest Digital Field office or representative. U.S.A. customers may order directly from the Program Library in Maynard. When ordering, include the code number and a brief description of the software requested.

Digital Equipment Computer Users Society (DECUS) maintains a user library and publishes a catalog of programs as well as the DECUSCOPE magazine for its members and non-members who request it. For further information please write to:

DECUS
Digital Equipment Corporation
146 Main Street, Bldg. 3-5
Maynard, Massachusetts 01754

DEC-15-ODFFA-A-D

READER'S COMMENTS

Digital Equipment Corporation maintains a continuous effort to improve the quality and usefulness of its publications. To do this effectively we need user feedback -- your critical evaluation of this manual.

Please comment on this manual's completeness, accuracy, organization, usability and readability.

Did you find errors in this manual? If so, specify by page.

How can this manual be improved?

Other comments?

Please state your position. _____ Date: _____

Name: _____ Organization: _____

Street: _____ Department: _____

City: _____ State: _____ Zip or Country _____

— — — — — — — — — — — — — — — — — — Fold Here — — — — — — — — — — — — — — — — — —

— — — — — — — — — — — — Do Not Tear - Fold Here and Staple — — — — — — — — — — —

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Digital Equipment Corporation
Software Information Services
146 Main Street, Bldg. 3-5
Maynard, Massachusetts 01754