PDP-8/I

digital

DISK MONITOR SYSTEM

PDP-8/I
DISK MONITOR SYSTEM
Programmer's Reference Manual

For additional copies specify Order No. DEC-D8-SDAA-D to Program Library, Digital Equipment Corporation, Maynard, Mass. Price: $2.75

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

1st Printing April 1968
2nd Printing June 1968

CONTENTS

CONTENTS (Cont)

ILLUSTRATIONS

HOW TO OBTAIN REVISIONS AND CORRECTIONS

Notification of changes and revisions to currently available Digital software and of new software manuals is available from the DEC Program Library for the PDP-5, 8, 8/S, 8/I, LINC-8, the PDP-4, 7, and 9 is currently published in DECUSCOPE, the magazine of the Digital Equipment Computer User's Society (DECUS). This information appears in a section of DECUSCOPE called "Digital Small Computer News".

Revised software products and documents are shipped only after the Program Library receives a specific request from a user.

DECUSCOPE is distributed periodically to both DECUS members and to non-members who request it. If you are not now receiving this information, you are urged to return the request form below so that your name will be placed on the mailing list.

To: Decus Office,
Digital Equipment Corporation,
Maynard, Mass. 01754

Please send DECUS installation membership information.

Please send DECUS individual membership information.

Please add my name to the DECUSCOPE non-member mailing list.

Name ____________________

Company ____________________

Address ____________________

____
(Zip Code)

CHAPTER 1
INTRODUCTION

The PDP-8 Disk/DECtape Monitor System is designed for any PDP-8 computer having at least one DECdisk or one DECtape. This system consists of a keyboard-oriented Monitor, which enables the user to efficiently control the flow of programs through his PDP-8, and a comprehensive software package, which includes a FORTRAN Compiler, Program Assembly Language (PAL-D), Edit program (Editor), Peripheral Interchange Program (PIP), and Dynamic Debugging Technique (DDT-D) program. Also provided is a program (Builder) for generating a customized monitor according to the user's particular machine configuration (amount of core, number of disks or DECtapes, etc.).

The system is modular and open ended, permitting the user to construct the software required in his environment, and allows the user full access to his disk (or DECtape) — referred to as the _system device_ — for storage and retrieval of his programs. By typing appropriate commands to the Monitor, the user can _load_ a program (construct it from one or more units of binary coding previously punched out on paper tape or written on the disk by the Assembler, and assign it core), _save_ it (write it out, with an assigned starting address, on the system device), and later _call_ it (read it back into core from the system device) for execution.

1.1 EQUIPMENT REQUIREMENTS

The minimum equipment requirements of the PDP-8 Disk/DECtape Monitor System are as follows.

A basic PDP-8, PDP-8/S, or PDP-8/I

- 4K of core
- Teletype
- 3-Cycle Data Break (Option required with PDP-8/S)
- At least one DF32 Random Access DECdisk File or a TC01 Automatic Control with a TU55 DECtape transport. The DECtape must have timing and mark tracks written on it prior to use.

NOTE

The system will recognize up to 32K of core, up to four disks (1 Type DF32 and 3 Type DS32's), up to eight DECtapes (TC01's only) and a high-speed paper-tape reader.

CHAPTER 2
MONITOR OPERATION

This chapter contains a discussion of the operation of the Monitor. Succeeding chapters contain descriptions and operating procedures for the system programs.

2.1 GENERAL DESCRIPTION

The PDP-8 Disk/DECtape Monitor System permits the user to control the flow of programs through his computer and takes full advantage of the extended memory capabilities of disk or DECtape. In addition to the Monitor, the system also contains a library of system programs. Together, they provide the user with the capabilities of compiling, assembling, editing, loading, saving, calling, and debugging his own programs.

2.1.1 Monitor Residence

Monitor, as well as system and user programs, is stored on and retrieved from the user's system device. To obtain a working Monitor, the user must first build his own customized version, via the easy-to-use dialogue technique of the System Builder program and store this version on his system device. Following this, the user then creates his System Program Library on the system device. Both of these procedures are described in Appendix A.

In core, the resident part of Monitor (called head of monitor) resides in the top page (locations 7600 through 7777) of field 0. The starting address of Monitor is 7600; 7642 is entry address to the system I/O routine, which performs all reading and writing on the system device.[1] Nonresident portions of Monitor, such as those routines which perform SAVEs and CALLs, are automatically called in as needed, and in core, they share the area from location 7000 through 7577. (These portions disappear after use, leaving this area for the user.)

Specific diagrams showing the allocation of the system, both on the system device and in core, are given in Appendix B.

2.1.2 System Modes

At any point in time, the system is running in one of two modes: Monitor mode or user mode.

Monitor mode is entered (1) whenever the Monitor is started (see Paragraph 2.2) or (2) when CTRL/C (↑C) is typed while running any system program. Monitor mode is signalled by the Monitor typeout of a dot (•). At both Monitor and system program time, Monitor is able to sense a ↑C typein, causing the system to enter Monitor mode, return to Monitor at location 7600, and respond with a dot (•) typeout. At this point, the user can issue any Monitor command via the Teletype keyboard.

User mode is present whenever the system is executing a system or user program. System programs signal user mode by responding with an asterisk (*) typeout.

[1] See appendix F

2.2 BOOTSTRAPPING THE MONITOR

The following discussion assumes that the user has built a customized Monitor and has stored it on his system device, according to the procedure described in Appendix A.

The bootstrapping of Monitor into core is necessary only when the resident Monitor area (locations 7600 through 7777) has been cleared or its contents otherwise destroyed. System Builder leaves the resident portion of Monitor in core after building. Turning the computer off and subsequently turning it on again does not normally destroy the contents of core.

The bootstrap procedure is as follows.

a. Toggle in one of the following bootstrap routines, depending upon the type of system device.

Disk Location	Contents	Symbolic
0200	6603	DMAR
0201	6622	DFSC
0202	5201	JMP .-1
0203	5604	JMP I .+1
0204	7600	7600
7750	7576	
7751	7576	

DECtape Location	Contents		Symbolic
			*200
0200	7600	BEG,	7600
0201	1216		TAD MVB
0202	4210		JMS DO
0203	1217		TAD M201
0204	3620		DCA I CA
0205	1222		TAD RF
0206	4210		JMS DO
0207	5600		JMP I BEG
0210	0000	DO,	0000
0211	6766		DTXA DTCA
0212	3621		DCA I WC
0213	6771		DTSF
0214	5213		JMP .-1
0215	5610		JMP I DO
0216	0600	MVB,	0600

DECtape Location	Contents	Symbolic	
0217	7577	M201,	-201
0220	7755	CA,	7755
0221	7754	WC,	7754
0222	0220	RF,	0220

b. After toggling in one of the above bootstrap routines, set the switches to 200 and press LOAD ADDress and START. Monitor should respond with a dot (·) after it has been brought into core.

2.3 STARTING THE MONITOR

Monitor start is at location 7600. A jump to this location can be made by either (1) stopping the machine, setting the switches to 7600, and pressing LOAD ADDress and START, or (2) typing ↑C when in Monitor mode or when a system program (or any user program which includes coding to sense a ↑C typein) is running.[1]

Monitor start performs the following actions.

a. Saves the coding from location 7200 through 7577 in the first two scratch blocks on the system device.

b. Reads blocks 1 and 2 (containing the rest of Monitor) from the system device into these locations.

c. Transfers control to Monitor, which responds with a carriage return, line feed, and a dot.

A monitor restart can be performed by typing RUBOUT to Monitor. A Monitor restart performs the same actions as described above except for Subparagraph a. A common use for RUBOUT is to terminate a command string when the operator has discovered that he has made a mistake. The command string is ignored, and Monitor responds as described in Subparagraph c. The user core image on the system device is not changed by RUBOUT (it is changed, however, by ↑C).

2.4 COMMAND STRINGS

The user types commands in the form of command strings to direct Monitor, or a system program, to perform some action. Command strings are simple in format and afford the user an easy means of communicating with the system.

Monitor indicates its readiness to accept a command string by typing a dot, and at this point, the user can type some Monitor command, such as CALL or SAVE.

[1] A start instruction (ST=7600) is issued when running Loader causes a jump to 7600 after loading has been performed. Certain errors also cause a jump to this location.

System programs indicate their readiness to receive information by typing either an asterisk or a query. The most common queries are as follows.

*OUT-	Requests that the user specify one output device name. In the case of disk or DECtape the filename to be assigned to the output data must also be specified.[1]
*IN-	Requests that the user specify one or more input device names. For disk and DECtape, filenames of input files must also be specified.[1]
*OPT-	Requests that the user specify one option or switch, entered as a single alphanumeric character; see Chapter 3 for options available in each system program.

This communication between the system and the user is handled by a portion of Monitor known as the Command Decoder.[2] Command Decoder is called into core by the system when needed and occupies any four contiguous pages of core. A description of its core allocation and calling procedure, plus a flow chart, is given in Appendix C. Error messages produced by Command Decoder are listed in Paragraph 2.8. Messages unique to individual system programs are given in Chapter 3.

2.4.1 Command String Format

Command strings are composed of a few basic elements and follow certain rules of punctuation. Their basic elements are as follows.

a. Device names
b. Filenames
c. Punctuation
d. Special characters

Each of these elements is described in the following paragraphs.

2.4.1.1 Device Names - Device names permitted in command strings are as follows.

Dn:	DECtape unit, if both disk and DECtape are present in the system (n = unit number, 0 through 7)
S:	System device (disk or DECtape unit 0)
R:	High-speed paper tape equipment (reader or punch)
T:	Low-speed paper tape equipment on the Teletype (reader or punch)

[1] Device names and filenames are explained in Paragraph 2.4.1.

[2] Command Decoder is a system program (.CD.) which is saved on the system device at build time.

2.4.1.2 Filenames - Filenames are limited to four characters in length and can be composed of any combination of alphanumeric characters or special characters[1] with the following exceptions.

a. Imbedded spaces cannot appear in a filename (they are ignored).[2] However, trailing spaces are permitted.

b. A filename cannot be one of the following words or symbols.

CALL SAVE ! , ; :

Extensions to the filenames specified by the user are automatically appended by the system. They are used internally by the system and cannot be referred to or modified by the user.[3]

SYS (n)	Saved system program file in core bank n.
USER (n)	Saved user program file in core bank n.
ASCII	Source language program file (input to PAL-D Assembler or FORTRAN Compiler).
BINARY	Binary program file (output from PAL-D Assembler).
FTC BIN	Interpretive binary file (output from FORTRAN Compiler).

Filenames (and extensions) are meaningful only for file structured devices (disk and DECtape). If they are specified for other devices, they are ignored. Both the filename and extension name appear on directory listings produced by the list feature in PIP.[4]

Example:

```
NAME   TYPE      BLK
8D
PIP  . SYS (0)   0015
SRC1. ASCII      0007
BIN  . BINARY    0001
SRC1.USER(0)     0001
```

2.4.1.3 Punctuation - Punctuation within command strings is as follows.

, (comma)	Used to separate device names, when more than one is given in a command string. The [redacted]e is also used to separate core references in a SAVE command string, when more than one contiguous area of core is specified. comma
;	Precedes the entry point specification in a SAVE command.
:	Terminates each device name. The colon is also used following the filename in a SAVE command to indicate that the file is to be saved as a user program.

[1] Although both printing and nonprinting keyboard characters are allowable, printing characters are recommended.

[2] Note that Monitor is given the filename EX C; one reason for this unconventional use of an imbedded blank is to protect Monitor from accidental destruction by the user (e.g., deletion via PIP).

[3] The data structure of these files is described in Appendix B under "Data Structure."

[4] "8D" in example means VERSION 8, change D.

-	Separates the beginning and ending addresses of a contiguous core area specification in a SAVE command.
!	Follows the filename in a SAVE command when a file is to be saved as a system program.

2.4.1.4 Special Characters - Special characters are used as described below.

↑C	If given while the system is in Monitor mode or a system program is running, control is returned to Monitor start (location 7600). Monitor responds with a dot. ↑C is typed by holding down the CTRL key and striking C. ↑C does not echo (does not print).
↑P	Typed in response to a ↑ typeout. Instructs the system to proceed with the next operation.[1] ↑P is typed by holding down the CTRL key and striking P. ↑P does not echo (does not print).
↲	Carriage return terminates current command string input. When typed alone, in response to a system query, it indicates that the user does not desire to specify the item (e.g., device name) requested.
RUBOUT	Causes the current command string to be ignored, and the system returns to the beginning of the command string and is ready to receive a new command. RUBOUT does not echo.

2.4.2 Examples of Command Strings

These examples illustrate the elements and rules explained above. Samples of both Monitor commands and system program commands are given.[2]

Monitor Commands:

. CALL PRG1 ↲	Call the user program file, PRG1, from the system device into core for execution.
. SAVE PALD! 0-7577; 6200 ↲	Save a program, previously loaded by Loader into locations 0 through 7577 of core, on the system device as a system program (!). Assign a starting address of 6200 and a filename, PALD.

System Program Commands:

*IN-S:PRO2 ↲	Use the file PRO2 on the system device as the input file.
*IN-S:TST1,R: ↲	Use the file TST1 on the system device and one file from the high-speed paper tape reader as the input files.
*OUT-D5:SPEC ↲	Write the output file on DECtape unit No. 5 and assign it the filename SPEC.

[1] ↑P can also be used to prematurely terminate certain operations while in progress (e.g., the typing out of a file directory by the list option in PIP).

[2] In all examples, system response (typeout) is underlined for clarity.

*OUT-T: ↲	Punch the output on the Teletype paper tape punch.
*OPT-M	Select option M.[1]

Spaces in command strings are ignored. Thus, both examples below are equally correct and perform the same function.

```
.SAVE PALD ! 0-7577; 6200 ↲
.SAVEPALD!0-7577; 6200 ↲
```

2.5 LOADING PROGRAMS - BINARY LOADER[2]

Binary Loader takes as input the binary coding produced by the PAL-D Assembler and loads it into core in executable form. When loading is completed, Binary Loader "disappears" after first entering the loaded program at the starting address typed by the user just prior to loading (see Paragraph 2.5.1). Loader accepts input from the system device or paper tape.

Loader requires one pass for any program which does not load above location 6777 (field 0). Loader uses core from location 167 through 177 and 6000 through 7577, and the resident portion of Monitor occupies the remainder of field 0. One-pass loading reads input files only once.

Two passes are required for all other programs (i.e., programs loading above 6777). In two-pass loading, programs can be loaded in all of field 0, except locations 7600 through 7777.[3] Two-pass loading requires that input paper tapes be read through the reader twice.

2.5.1 Binary Loader Operating Procedures

.LOAD ↲	Direct Monitor to bring Binary Loader from the system device into core for execution.
*IN-	Loader requests source of input(s). Type one or more device names, separated by commas. If an input device is a file-structured device, include filename(s). Up to five files can be specified.[4]

Examples

*IN-R: ↲	Input one tape from the paper tape reader.[5]
*IN-R:,R:,R: ↲	Input three tapes from the paper tape reader.[5]

[1] An automatic carriage return occurs after user response to an OPT-request.

[2] Binary Loader is a system program saved on the disk at build time. It is called by the user in the same manner as any system program. It occupies locations 7000-7577 and has a starting address of 7000.

[3] In 8K and larger systems, Loader sets up locations 7574 through 7577 to perform a start in fields other than 0. It is the user's responsibility to protect these locations if he wants to start in other than field 0.

[4] An E or I error message (see Table 2-1) may appear following the entry of an IN command.

[5] Regardless of whether R: or T: is used to specify paper tape input, the high-speed equipment is used if it was indicated as present in the system at System Builder time, otherwise the Teletype equipment is used. This convention is unique to Binary Loader.

*IN-S:INPT↲	Input the file INPT from the system device.
*IN-S:BIN2,R:↲	Input the file BIN2 from the system device and one tape from the paper tape reader.[1]
*IN-S:BIN1,S:BIN2↲	Input the files BIN1 and BIN2 from the system device.
*	If device(s) are valid and filenames (if any) are actually found on the system device, Loader responds with one asterisk for each correct input.
*OPT-	Loader requests mode desired (one-pass or two-pass).

Examples

*OPT-1	One pass loading desired; no programs are loaded above location 6777.
*OPT-2 (or anything else)	Two-pass loading desired; programs can be loaded above location 6777.
*ST=	Loader requests the starting address to which control is to be transferred when loading is completed. The address is typed in the form

fnnnn

where

f = field number[2] (omitted if field 0),

and

nnnn = location within field

Examples

*ST=↲ *ST=7600↲ *ST=0↲	Load into field 0. Return to Monitor after loading.
*ST=30225↲	Load into field 3. Jump to location 255, field 3, after loading.
*ST=10000↲	Load into field 1. Return to Monitor after loading into field 1.

Loader now types a series of up-arrows, one at a time, as explained below.

Following each up-arrow typeout, the user is required to perform one or more actions.

[1] Regardless of whether R: or T: is used to specify paper tape input, the high-speed equipment is used if it was indicated as present in the system at System Builder time, otherwise the Teletype equipment is used. This convention is unique to Binary Loader.

[2] The f-digit forces Loader to start loading into the specified field until a "field bit" is found in the input.

↑↑↑↑

First up-arrow: Loader is ready to load. If paper tape input, put the tape in the reader. Type ↑P.[1]

Second up-arrow: End of pass 1. If operating in one-pass mode, type ↑P to jump to previously specified starting address.

If operating in two-pass mode, type ↑P.

The next two up-arrows appear only if operating in two-pass mode.

Third up-arrow: Reload paper tape input for pass 2. Type ↑P.

Fourth up-arrow: End of pass 2. Type ↑P to jump to previously specified starting address.

Multiple Input Files

An up-arrow is typed out as the processing of each input file is completed. If paper tape input, insert the next file in the reader and type ↑P.

Repeat the above step until all files given in response to the *IN- request have been processed.

If in two-pass mode, each tape must be entered twice, in the order

T1,T2,T3,....T1,T2,T3,....

After all files have been entered the required number of times, type ↑P to jump to the previously specified starting address.

NOTE

After each input paper tape is read, the high-speed paper tape version of Loader loops until the user types ↑P to continue. However, the low-speed paper tape version halts. Thus, when using the Teletype paper tape equipment for input, the user need not type ↑P but press CONT on the console and start the paper tape reader.

At this point, Binary Loader disappears and control is transferred to the previously specified starting address.

A flow chart of Binary Loader can be found in Appendix D.

2.5.2 Binary Loader Error Messages

An illegal checksum error condition causes Loader to type

?

and return to Monitor after the user types ↑P or ↑C. Error messages for illegal filenames or devices are as specified in Paragraph 2.8.

[1] If Teletype paper tape equipment is used, type ↑P before turning on the reader.

2.6 SAVING PROGRAMS (SAVE COMMAND)

The SAVE command enables the user to write core images of system or user programs from core onto his system device for subsequent call-in (CALL) and execution. For example, a program which has been loaded by Binary Loader can be stored on the system device by the SAVE command. Or, a previously saved program which has been called in and modified by DDT can be stored in its updated version on the system device, overlaying the old version if desired.

Core images can be saved in units of one or more pages, each page occupying one block on the system device. If a core specification (see below) addresses only a portion of a page, the entire page is written out. For example, the core specification 45-150 is treated as though it were 0-177. Core areas to be saved may be contiguous or noncontiguous as desired by the user. Up to 32_{10} core specifications, in any combination of monotonically increasing single-page or multiple-page requests, can be entered in a single SAVE command.

2.6.1 SAVE Command Format

$$.\mathrm{SAVE}\quad \mathrm{filename}\begin{Bmatrix} ! \\ : \end{Bmatrix}\mathrm{core\text{-}specifications,\ldots;\ entry\text{-}point} \downarrow$$

SAVE	Directs Monitor to call in the nonresident SAVE routine.
filename	The filename (program name) to be assigned to the file on the systems device. This name will be used to call the file later when the user wants to read in and execute the program. Restrictions on the formation of filenames can be found in Paragraph 2.4.1.2. Any previously saved program with the same "filename" and having the same extension will be automatically overwritten.
! or :	! is typed immediately after the filename of a file if the user desires to save it as a <u>system program</u> (e.g., PIP). A program saved in this manner can be called in by simply typing its name to Monitor (the word CALL is not required). .filename ↲ An extension name of .SYS is automatically appended to the filename. : is typed immediately after the filename of a file if the user desires to save it as a <u>user program</u>. A program saved in this manner can be called in and executed later via the CALL command. .CALL filename ↲ An extension name of .USER is automatically appended to the filename.
core-specifications	Up to 32 core specifications can be entered in a single SAVE command. Each core specification is separated from the following one by a comma. The last core specification in the series is followed by a semicolon. Addresses are expressed in octal.

Single-page core specification

fnnnn

where

f = field number (can be omitted if field 0).

nnnn = any location within the page which the user desires to save.

Examples		
	0	Saves page 0 (locations 0 through 177) of field 0.
	3570	Saves [redacted] (locations 3400 through 3577) of field 0.
	30100	Saves page 0 (locations 0 through 177) of field 3.

Multiple-page core specification

When a user wishes to save a core area of several contiguous pages, he can type a multiple-page core specification in the format

$\text{fnnnn}_1\text{-nnnn}_2$

where

f = field number (can be omitted if field 0).

nnnn_1 = any location within the first page of the series of contiguous pages to be saved.

nnnn_2 = any location within the last page of the series of contiguous pages to be saved.

The following rules apply.

a. The beginning address of a multiple-page request must be smaller than the ending address (nnnn_1 must be smaller than nnnn_2).

b. Both addresses must be in the same field.

c. The field number (f) must be within the range of your system; however, no check for the validity of this number is performed at SAVE time.

Examples

0-7577	Saves all of field 0.
10000-7777	Saves all of field 1. Note that this is the same as typing 10000-17777 See below for explanation of how the field number (5th significant digit to the left of the decimal point) is "remembered."

30425-745	Saves locations 400 through 777 (pages 3 and 4) of field 3.

NOTE

Only one field can be saved by each SAVE command. If multiple fields are to be saved, a separate SAVE command must be given for each.

entry-point The entry point of the saved program, in the format

Fnnnn (see explanation above)

An entry point of 0 causes a return to Monitor at CALL time, regardless of the field into which the program was saved.

NOTE

The last nonzero field number encountered in a SAVE command string is remembered and prefixed to all other addresses in the command string. (Remember: only one field can be referred to in each command string.)

Example: The following entries are identical in meaning.

```
SAVE PRGA: 10000-10777, 11400, 1600-17777; 10200
SAVE PRGA: 30000-777, 51400, 26000-7777; 10200
SAVE PRGA: 10000-777, 1400, 6000-7777; 200
SAVE PRGA: 0-777, 1400, 6000-7777; 10200
```

In each of these examples, all addresses are treated as being in field 1, because the last five-digit entry seen contained a most significant digit 1.

2.6.2 SAVE Command Processing

A list of the required pages is constructed from the information typed by the user and a block requirement count is kept. When the user types the terminating carriage return (↵), allowing the SAVE process to begin, a directory name search on the system device is initiated. If a file having the same name as the filename in the SAVE command is found, it is replaced by the file now being saved. If no such file is found, a new file is created. Next, a storage availability search finds a sufficient number of available blocks on the system device to satisfy the block requirement count. (See above.) These block numbers are stored in a corresponding block list; the blocks are then filled with the contents of the pages to be saved. When the SAVE process is completed, control returns to Monitor (7600).

2.7 CALLING A PROGRAM (CALL COMMAND)

Once a file has been loaded and saved, it can be called into core as desired. There are two types of CALL command strings: one for system programs and the other for user programs.

The CALL command string format for system programs (programs saved by a SAVE command string in which the filename was followed by a !) is

```
.filename ↵
```

where filename is the same as the one used in the SAVE command string which saved it.

The CALL command string format for user programs (programs saved by a SAVE command string in which the filename was followed by a :) is

.CALL filename ↲

When a program is called, a directory name search is performed on the system device. Associated with the directory entry is the entry point of the program and information concerning file protection and memory extension. If the appropriate directory name entry is found and the file has the proper extension (.SYS or .USER), calling proceeds. If not, the calling process is terminated, ? is typed and control is returned to Monitor.

2.9 SYSTEM ERROR MESSAGES

As an input command string is being typed, Monitor recognizes any incorrect syntax and remembers it. When the user types a carriage return, Monitor responds with a ? to indicate invalid input.

Error messages output by Command Decoder are given in Table 2-1.

Table 2-1
System Error Messages

Message	Meaning
?	Illegal syntax or miscellaneous error condition
D	Directory on the systems device is full
E	Too many inputs or outputs were entered
I	No such inputs
S	System I/O failure

Local errors in each system program are given in Chapter 3.

Monitor time read or write errors cause a halt to occur. Persistence of this condition indicates a hardware failure, as the system I/O routine attempts to read or write three times before halting.

2.8 MONITOR START CONDITION

When a CTRL/C or start at 7600_8 has just occurred, the Monitor saves three pages of old core on the disk scratch blocks. To recover this core perform the following:

a. Do not type a second CTRL/C, as the user core memory from 7000-7577 (the locations removed from core) are still on the disk.

b. Type "SAVE SCRATCH: 7000-7577; n↲" (n is the point in the System Program to which you would like to return, e.g., 200 for FOCAL).

c. Type "CALL SCRATCH↲". This restarts you with your core intact.

CHAPTER 3
SYSTEM PROGRAM LIBRARY

The Monitor System's library of programs presently consists of the Peripheral Interchange Program (PIP), Disk System Editor (Editor), PAL-D Disk Assembler (PAL-D), 4K Disk FORTRAN (FORTRAN-D), and Dynamic Debugging Technique for Disk (DDT-D), and this list is destined to lengthen with time. A section of this chapter is devoted to each program in the library.

To load a program using the Monitor System, the Loader makes certain queries to which the user must type a reply. The queries are the same for all programs. The user's replies will vary, however, depending on the particulars of the program being loaded.

When loading a program into core, the user should first check to see whether Monitor is in core. This is done by typing ↑C (CTRL key and then the C key). The ↑C will not echo (not print on the teleprinter). If Monitor is in core, it will respond by typing a period (•) at the left margin of the teleprinter paper. If a period is not typed in response to ↑C, Monitor is not in core. Therefore, the user should refer to Chapter 2 of this manual for information on building Monitor and putting it into core.

The library system includes the Binary Loader (LOAD) which is automatically saved on the disk at build time. (For Loader operating procedures see Paragraph 2.5.)

The user may save any program on the disk by responding to the last period typed by Monitor with the word SAVE, a four character name of the program, the type of program (user or system), whether it's a one or more page save, and the location of its starting address, as is thoroughly described in Paragraph 2.6.

After each program is saved on the system device, it may be called (i.e., transferred from the disk into core) merely by responding to Monitor (to a period) with the four characters designated as the name of that program, as explained in Paragraph 2.7.

3.1 PIP

PIP (Peripheral Interchange Program) performs general utility operations, such as listing the contents of specified directories, deleting unwanted files from the system device, and transferring files between devices, and copying specified files. PIP enables the user to do any of the above operations merely by typing commands from the teleprinter keyboard.

3.1.1 Loading and Saving

PIP is loaded into core as indicated in Appendix E. Core requirements, starting address, and number of passes through the Binary Loader (hereafter frequently referred to merely as Loader) are also found in Appendix E.

To load PIP into core, the user calls LOAD, using Monitor, and replies to the system responses as explained in Chapter 2.

When in core, PIP may be saved on the system device as a system device by Monitor, as indicated in Appendix E. (See Paragraph 2.6.1 for a detailed description of the SAVE format.)

When loading and saving PIP, the printout will take the following format. (See Appendix E for precise core limits.)

```
.LOAD ↲
*IN-R: ↲
*
*OPT-1 ↲
*ST= ↲
↑↑
.SAVE PIP! 0-3177;1000 ↲

. ↲
```

3.1.2 Operating Procedures

PIP has now been loaded into core and saved on the disk. To use PIP, the user must call PIP via Monitor which can be done only in response to a period. If a period is not present as the last system response, the user must type ↑C, which should cause Monitor to type the needed period. The printout should appear as follows:

```
.PIP ↲
```

which transfers PIP from the disk into core. PIP now responds with

```
*OPT-
```

and waits for the user to select and specify one of the following:[2]

L	List entire system directory
B	Copy a binary file
D	Delete a file to be specified
F	Copy a FORTRAN binary file
M	Move directory to safe disk
P	Protect disk 1 (blocks 0-176)
R	Restore directory from safe disk
S	Copy a system file[1]
U	Copy a user file[1]
↲ or A	Copy a USA SCII file

If the user selects an option using any character other than one of those listed above, the option is recognized as illegal; PIP ignores the request, types ? (question mark), and asks for another option character. The output appears as follows:

```
*OPT-G ↲
?
*OPT-
```

[1] User system files may not be copied onto paper tape.

[2] Only L and D are in version on tape DEC-B8-PDAA-PB

When the user specifies the D (delete a file) option, PIP responds with

*FILE TYPE(A,B,F,U,S)-

where A,B,F,U,S are the legal options from which the user may choose. If the user's reply is S (system file), PIP asks

REALLY?

PIP will not delete a system file unless the user answers by typing

Y ↲ (meaning yes)

Any other answer causes PIP to repeat the file type request.

If the file requested for deletion is Monitor (EX C), PIP will not delete it even if the user answers to REALLY? with a Y. Instead, the request is ignored, and Monitor types a ?.

When PIP is satisfied with the user's reply to the file type request, it will type

*IN-

When PIP recognizes a legal option character other than D, it will type *IN-, as shown in the example below.

The user should now specify the input device[1] as explained at the beginning of this chapter and in Paragraph 2.4.

3.1.3 Examples

```
.PIP ↲                             User calls PIP
*OPT-L ↲                           and requests that it list option
*IN-S: ↲                           of the system device directory
FB=2426                            PIP types number of free (unused) blocks remaining
                                   on specified device
NAME  TYPE      BLK
8D    .                            followed by file name and description; e.g., PAL-D
PALD  . SYS  (0)  0037             is a system program in field 0 and occupies 37
EDIT  . SYS  (0)  0015             blocks of storage                              8
LOAD  . SYS  (0)  0003
.CD.  . SYS  (0)  0006
PIP   . SYS  (0)  0015
DDT   . ASCII     0062
FOO   . USER (0)  0001
BAR   . SYS  (0)  0037
*OPT-D ↲                           User requests delete option
*FILE TYPE (A,B,F,U,S)-U ↲         and specifies type of file, U (user)
```

[1] Options requiring input ask only for an input device; those requiring both input and output request a device for each.

*IN-S:FOO ↵	and device and file name
*OPT-D ↵	User requests delete option
*FILE TYPE(A,B,F,U,S)-S ↵ REALLY?Y ↵	and specifies type of file, S (system) (PIP double checks); Y is the only meaningful answer
*IN-S:BAR ↵	User specifies file and filename
*OPT-L ↵	User requests list option
*IN-S: ↵	and system device directory
FB=2466	Note increase of 40_8 free blocks (see above)
NAME TYPE BLK	
8D	
PALD .SYS (0) 0037	
EDIT .SYS (0) 0015	
LOAD .SYS (0) 0003	
.CD. .SYS (0) 0006	
PIP .SYS (0) 0015	Note removal of two deleted files
DDT .ASCII 0062	
*OPT-D ↵	User requests delete option
*FILE TYPE (A,B,F,U,S)-S ↵	
REALLY?N ↵ *FILE TYPE(A,B,F,U,S)-S ↵ REALLY?W ↵	Y is only response for deletion of a system file; other responses cause PIP to repeat the file type request
*FILE TYPE(A,B,F,U,S)-S ↵ REALLY?Y ↵	
*IN-S: EX C ↵ ? *OPT-D ↵	Even if user responds to REALLY? with Y, PIP will not delete the Monitor file
*FILE TYPE(A,B,F,U,S)-S ↵ REALLY?Y ↵	
*IN-S:EX C ↵ ? *OPT-D ↵	
*FILE TYPE(A,B,F,U,S)-U ↵	
*IN-S:NONE ↵ ? *OPT-D ↵	PIP knows NONE is not an existing user file name on the system device and indicates by typing ?
*FILE TYPE (A,B,F,U,S)-A ↵	User requests USA SCII file option
*IN-S:EDIT ↵ ? *OPT-D ↵	PIP also knows when the file name and file type don't match; EDIT is a system program
*FILE TYPE (A,B,F,U,S)-B ↵	
*IN-S:EDIT ↵ ? *OPT-	

3.2 EDITOR

Editor (Disk System Editor) enables the user to generate and edit symbolic programs on-line from the teleprinter keyboard. The symbolic program may be either printed on the teleprinter, punched on paper tape using the high- or low-speed punch, or saved on the system device as a user program.

Editor operates either in command or text mode. In command mode, all typed input is interpreted as a command instructing Editor to perform a certain operation or to allow the user to perform an operation on the text stored in the buffer. In text mode, all typed input is interpreted as text to replace, to be inserted into, or to be appended to the contents of the text buffer.

The command language of the Disk System Editor is identical to that of the PDP-8 Symbolic Editor (DEC-08-ESAB-D) but with the following exceptions.

a. Special characters:

↑P	During output, progress stops and control is returned to command mode.
↑C	Always returns control to Monitor.

b. Commands:

P	Proceed, and output entire contents of the buffer followed by a form feed and return to command mode.
nP	Output line n, followed by a form feed, return to command mode.
m,nP	Output lines m through n, followed by a form feed, return to command mode.
F	Illegal command
E	Process entire file (perform enough NEXT commands to fill the file) and create an end-of-file indicator (legal only for output to the system device).

Certain keys have special operating functions. These keys and their associated functions are listed in Table 3-1.

Table 3-1
Special Key Functions

Key	Functions
↲ (carriage return)	Text mode: Enter the line in the text buffer. Command mode: Execute the command.
← (back arrow)	Text mode: Cancel the entire line of text and continue typing on same line. Command mode: cancel command.
\ (rubout)	Text mode: Delete from right to left one character for each rubout typed (is not in effect during a READ command). Command mode: Delete entire command.
FORM FEED	Text mode: End of input, return to command mode.

Table 3-1 (Cont)
Special Key Functions

Key	Functions
. (period)	Command mode: Current line counter used as argument alone or in combination with + or - and a number.
/ (slash)	Command mode: Value equal to number of last line in buffer and used as argument.
↓ (line feed)	Text mode: Used in SEARCH command to insert a carriage return/line feed combination into the line being searched. Command mode: List the next line.
ALT MODE	Command mode: List the next line.
ESCape	Command mode: List the next line.
< (left angle bracket)	Command mode: List the previous line.
= (equal sign)	Command mode: Used in conjunction with . and / to obtain their value (. = 27).
: (colon)	Command mode: Lower case character, same function as = .
→| (tabulation)	Text mode: On output, is interpreted as [illegible] a tab/rubout combination.

Table 3-2 is a summary of Editor commands.

Table 3-2
Summary of Editor Commands

Command	Format(s)	Meaning
READ	R ↵	Read incoming text and append to buffer until a form feed is encountered.
APPEND	A ↵	Append incoming text to any already in the buffer until a form feed is encountered.
LIST	L ↵	List the entire buffer.
	nL ↵	List the line n.
	m,nL ↵	List lines m through n.
PROCEED	P ↵	Proceed and output the entire contents of the buffer and return to command mode.
	nP ↵	Output line n, followed by a form feed.
	m,nP ↵	Output lines m through n, followed by a form feed.
TRAILER	T ↵	Punch four inches of trailer.
NEXT	N ↵	Punch the entire buffer and a form feed; kill the buffer and read next page.
	nN ↵	Repeat the above sequence n times.
KILL	K ↵	Kill the buffer.
DELETE	nD ↵	Delete line n.
	m,nD ↵	Delete lines m through n.

Table 3-2 (Cont)
Summary of Editor Commands

Command	Format(s)	Meaning
INSERT	I ↵	Insert before line one all text until a form feed is encountered.
	nI ↵	Insert before line n until a form feed is encountered.
CHANGE	nC ↵	Delete line n and replace it with any number of lines from the keyboard until a form feed is encountered.
	m,nC ↵	Delete lines m through n, replace from keyboard as above until form feed is encountered.
MOVE	m,n$kM ↵	Move and insert lines m through n before line k.
GET	G ↵	Get and list the next line beginningwith a tag.
	nG ↵	Get and list the next line after line n which begins with a tag.
SEARCH	S ↵	Search the entire buffer for the character specified (but not echoed) after the carriage return; allow modification when found.
	nS ↵	Search line n, as above, allow modification.
	m,nS ↵	Search lines m through n, allow modification.
END FILE	E ↵	Process the entire file (perform enough NEXT commands to pass over the entire file) and create an end-of-file indication; legal only for output to the system device. If the low-speed paper tape reader is used for input while performing an E command, the paper tape reader will eventually run out ot tape, and at this point typing a form feed will allow the command to be completed.

Editor will print an error message consisting of a question mark whenever the user requests nonexistent information or uses an inconsistent or incorrect format in typing a command. The question mark will be followed by a carriage return/line feed and the command will be ignored.

3.2.1 Loading and Saving

Editor is loaded into core from punched paper tape in one pass using the Loader. When in core, it occupies locations shown in Appendix E.

To load Editor into core, the user calls LOAD, using Monitor, and replies to the system responses as explained at the beginning of this chapter and in Paragraph 2.5.

When in core, Editor may be saved on the system device as a system program by Monitor when the user types the command indicated in Appendix E.

(See Paragraph 2.6.1 for detained description of the SAVE format.)

When loading and saving Editor, the printout should appear approximately as follows.

```
.LOAD ↵
*IN-R: ↵
*
*OPT-1 ↵
*ST=7600 ↵
↑↑
.SAVE EDIT!0-3177; 2600 ↵          (See Appendix E.)
.
```

3.2.2 Operating Procedures

Editor is transferred from the system device into core by Monitor when the user types

EDIT ↵

Editor is now in core and responds by typing

*OUT-

The user selects one of the following output devices: (T:) for low-speed reader/punch; (R:) for high-speed reader/punch; (S:name) for output to the systems device on a file called name and types his choice immediately after OUT-. If the specified device is not valid, that is, not declared when building Monitor, Editor will respond with an error message (see Paragraph 2.8) and return control to Monitor. Thus the user must call EDIT and respond to *OUT- with a valid device.

When Editor recognizes a valid device, it responds with *↵ (asterisk, carriage return/line feed) and *IN-, as shown below.

```
*
*IN-
```

The user now specifies the input device by typing T:↵ , R:↵ , or S:name↵ or↵ in the same manner as when replying to *OUT-, above.[1]

The Editor responds with

*OPT-

asking the user to specify one of the following options.

B ↵	Preserve blanks. Editor normally replaces multiple blanks (spaces) with tabs, resulting in considerable saving of space on the system device.
D ↵	Enter dynamic deletion mode if input is from the system device. As the file is read, it is deleted from the system device, thus allowing space for output if desired. (File name remains on the directory but without any assigned blocks.)
C ↵	Combine the functions of B and D options.
↵	None of the above options; assume conversion of two or more blanks to tabs, and not D.

[1]With a System Device output, the user must type E ↵ to properly close the output file.

After the user has specified one of the options listed above, Editor responds with a carriage return/line feed and asterisk. The entire printout might appear as follows.

```
.EDIT↲
*OUT-R:↲
*
*IN-T:↲
*
*OPT-B↲
*
*
```

The appearance of the last asterisk in the example above indicates that Editor is ready to accept and operate on the user's symbolic program.

The user may now load the symbolic program into core by using the procedures described in Paragraph 2.5.

3.2.3 Example

```
.LOAD ↲                        Call Loader using Monitor
*IN-R: ↲                       Input to be from high-speed reader
*                              Input device valid
*OPT-1 ↲                       One-pass load
*ST= ↲                         Return to Monitor after loading
↑↑                             Editor is loaded
.SAVE EDIT!0-3177;2600 ↲       and saved on the system device
.EDIT ↲                        Call Editor using Monitor
*OUT-S:SRC1 ↲                  Output to be on system device, file named SRC1
*
*IN-R:  ↲                      Input to be from high-speed reader
*                              Input device valid
*OPT- ↲                        No blanks, no dynamic deletion mode
*R ↲                           Read incoming text
*E ↲                           Process entire file
.EDIT↲                         Call Editor using Monitor
*OUT- ↲                        No output desired
*
*IN-S:SRC1 ↲                   Input from filename SRC1
*                              Filename valid
*OPT-                          No option desired
*R ↲                           Read incoming text
*L ↲                           List the entire buffer
*7400                  /STARTING ADDRESS OF PROGRAM
ODUM,     CLA
          OSR          /GET LOWER LIMIT
          DCA LOCK
          HLT
          OSR          /GET UPPER LIMIT
          CMA                  (↑P was typed here, stopped listing of buffer)
*/L
$
.                              (↑C was typed here)
```

3.3 PAL-D DISK ASSEMBLER

PAL-D, the acronym for Program Assembly Language for the Disk system, is the symbolic assembly program designed primarily for the 4K PDP-8 family of computers with disk or DECtape.

The PAL-D Assembler performs many useful functions, making machine language programming easier, faster, and more efficient. Basically, the Assembler processes the user's source program statements by translating mnemonic operation codes into the binary codes needed in machine instructions, relating symbols to numeric values, assigning absolute core addresses for program instructions and data, and preparing an output listing of the program which includes notification of any errors detected during the assembly process.

The user may use pseudo-operators (pseudo-ops) to direct PAL-D to perform certain tasks or to interpret subsequent coding in a certain manner. Instead of generating instructions or data, pseudo-ops direct the Assembler on how to proceed with the assembly. Pseudo-ops are maintained in the Assembler's permanent symbol table.

The following is a summary of PAL-D's pseudo-ops.

Table 3-3
PAL-D Pseudo-Operators

Pseudo-Operator	Explanation
PAGE	Set current location counter to first location on next page.
PAGE n	Set current location counter to first location on page n.
FIELD n	Load subsequent data in field n.
DECIMAL	Interpret subsequent integers as decimal.
OCTAL	Interpret subsequent integers as octal.
XLIST	Data enclosed is not to appear on third pass listing.
TEXT	Input text strings in USASCII code trimmed to six bits.
$	End of program, terminate current pass.
PAUSE	End of file, terminate processing, proceed to next file.
EXPUNGE	Erase symbol table, except pseudo-ops.
FIXTAB	Append to symbol table.

The Assembler is thoroughly documented in PAL-D Disk Assembler Programming Manual (Doc. No. DEC-D8-ASAA-D).

3.3.1 Loading and Saving

PAL-D is loaded into core from punched paper tape in two passes using Loader. When in core, it occupies locations, as shown in Appendix E.

To load PAL-D into core, the user calls LOAD using Monitor and replies to the system responses as explained at the beginning of this chapter.

When in core, PAL-D may be saved on the system device as a system program by Monitor as described in Appendix E. (See Paragraph 2.6.1 for a detailed description of the SAVE format.)

When loading and saving PAL-D, the printout should appear approximately as shown below. (See Paragraph 2.5.)

```
.LOAD ↵
*IN-R:↵
*
*OPT-2 ↵
ST=7600↵
↑↑↑↑
.SAVE PALD!0-7577; 6200↵

.
```

3.3.2 Operating Procedures

PAL-D is transferred from the system device to core using Monitor. The user begins by typing

```
.PALD ↵
```

PAL-D responds with a request for the output device by typing

```
*OUT-
```

The user selects the output device by specifying one of the following.

T: ↵	for the low-speed punch
R: ↵	for the high-speed punch
S:name ↵	for output to the system device as a file called name

PAL-D then responds with

```
*IN-
```

and waits for the user to select the input device(s). Up to five input devices may be specified (for example, R:, T:, R:, R:, T:↵), but in this example the user selected

R: ↵	input from the high-speed reader

If the user had specified the devices in the parenthetical example above, PAL-D would have typed an asterisk for each input device that it found valid.

When PAL-D is satisfied that the input device is valid (i.e., the device does exist or the file is present on the file-structured device), it will request the third-pass listing option by typing

```
*OPT-
```

The user types one of the following.

T ↲	meaning listing and symbols are to be produced on the teleprinter
R ↲	meaning listing and symbols are to be produced on the high-speed reader/punch
↲	meaning no third pass desired

(any other character means no third pass desired)

The entire printout might appear as follows.

```
.PALD↲
*OUT-T:↲
*
*IN-R:↲
*
*OPT-T↲
```

PAL-D is now ready to proceed with the assembly, pausing only when user intervention is required (i.e., placing a new paper tape in the reader, turning off the punch, etc.). On these occasions, PAL-D will type an up-arrow (↑) on the teleprinter and wait for the user to type ↑P, indicating that the user is ready to continue with the assembly.

Assembly may be terminated and control may be returned to Monitor at any time by typing ↑C. When assembly is complete, control is automatically returned to Monitor.

PAL-D makes many error checks as it processes source language statements. When an error is detected the Assembler prints an error message. The format of the error messages is

ERROR CODE ADDRESS

where ERROR CODE is a two-letter code which specifies the type of error, and ADDRESS is either the absolute octal address where the error occurred or the address of the error relative to the last symbolic tag (if there was one) on the current page.

PAL-D's error messages are listed and explained below.

Table 3-4
PAL-D Error Messages

Error Code	Explanation
BE	Two PAL-D internal tables have overlapped.
DE	System device error
DF	System device full
IC	Illegal character
ID	Illegal redefinition of a symbol
IE	Illegal equal sign
II	Illegal indirect address
PE	Current nonzero page exceeded
PH	Phase error
SE	Symbol table exceeded
US	Undefined symbol.
ZE	Page zero exceeded

3.3.3 Examples

The following example shows the entire process covered in this section.

.LOAD ↲	Call Loader
*IN-R: ↲	Input to be from high-speed reader
*	Loader found input device valid
*OPT-2 ↲	Two-pass load
*ST= ↲	Return to Monitor after loading
↑↑↑↑	PAL-D is loaded
.SAVE PALD!0-7577;6200 ↲	PAL-D is saved on disk (see Appendix E)
.PALD ↲	Call PAL-D
*OUT-S:BIN ↲	Output to filename BIN on system device
*	filename and system device are valid
*IN-S:SRC1 ↲	Input to filename SRC1 from system device
*	Filename and system device are valid
*OPT-R ↲	Output listing and symbols on high-speed reader/punch
.LOAD ↲	Call Loader
*IN-S:BIN ↲	Innut to filename BIN from system device
*	Filename and system device are valid
*OPT-2 ↲	PAL-D generates [redacted] binary code from sources program in two passes
*ST= ↲	Return to Monitor after assembly
↑↑↑↑	Source program is translated into [redacted] binary code, followed by the output of the listing and symbols on the high-speed punch

3.4 FORTRAN-D

FORTRAN-D (FORmula TRANslation for the Disk System), is an expanded version of standard PDP-8 FORTRAN designed for PDP-8 computers with disk or DECtape units.

FORTRAN-D contains a compiler and an operating system. The FORTRAN compiler is used to convert a source program into an object program. The FORTRAN operating system is used to execute the object program.

This version of FORTRAN is designed to facilitate user/system communication by typing appropriate commands from the teleprinter keyboard, eliminating the need to toggle input using the switch registers.

FORTRAN statements specify the computations required to carry out the processes of the FORTRAN program. There are four types of statements provided for by the FORTRAN language:

a. Arithmetic statements define a numerical calculation.

b. Control statements determine the sequence of operation in the program.

c. Specification statements define the properties of variables, functions, and arrays appearing in the source program. They also enable the user to control storage allocation.

d. Input-output statements are used to transmit information between the computer and related input-output devices.

A summary of the FORTRAN statements is given in Table 3-5.

Table 3-5
Summary of FORTRAN Statements

Statement and form	Explanation
1. Arithmetic Statements	
v = e	v is a variable (possibly subscripted); e is an expression.
2. Control Statements	
GO TO n	n is a statement number.
GO TO $(n_1,n_2,\ldots n_n),i$	$n_1,\ldots n_n$ are statement numbers; i is a non-subscripted integer variable.
IF (e) n_1,n_2,n_3	e is an expression; n_1,n_2,n_3 are statement numbers.
DO n $i=k_1,k_2,k_3$	n is a statement number of a CONTINUE; i is an integer variable; k_1,k_2,k_3 are integers or nonsubscripted integer variables.
CONTINUE	Proceed
PAUSE	Temporarily suspend execution.
PAUSE n	n is an address; subroutine execution will commence at n.
STOP	Terminate execution.
END	Terminate compilation; last statement in program.
3. Specification statements	
DIMENSION $v_1(n_1),v_2(n_2,\ldots v_n(n_n)$	$v_1,\ldots v_n$ are variable names; $n_1,\ldots n_n$ are integers.
DEFINE device	Device is DISK or TAPE, system I/O device.
FORMAT $(s_1,s_2,\ldots s_n)$	s is a data field specification.
COMMENT	Designated by C as first character on line.
4. Input-Output Statements	
ACCEPT f,list	f is a FORMAT statement number; list is a list of variables.
TYPE f,list	f is a FORMAT statement number; list is a list of variables.
READ u,f,list	u is an integer, representing device from which data is to be read. f is a FORMAT statement number; list is a list of variables.
WRITE u,f,list	u is an integer, representing device onto which data will be written. f is a FORMAT statement number; list is a list of variables.

The following functions are allowed:

SQTF(x)	square root of x
SINF(x)	sine of x
COSF(x)	cosine of x
ATNF(x)	arctangent of x (in radians)
EXPF(x)	exponential of x
LOGF(x)	logarithm of x
ABSF(x)	absolute value of x

Certain input-output statements have special characteristics when used with disk or DECtape units.

a. The READ and WRITE statements disable the user from performing sequential input and output either on paper tape or on the system device.

b. A DEFINE statement must precede the first executable statement in any program by using the system device to input or output data.

c. When the operating system is called, the input or output filename must be specified by using the S option if data is to be read from or written on the system device.

d. When a READ statement is used with the teleprinter, the statement differs from the ACCEPT statement in that the data being read is not echoed on the printer.

e. A WRITE statement used with the teleprinter differs from a TYPE statement in that it always terminates by typing a carrige return-line feed.

f. The READ and WRITE statements allow the user to input and output data on either the teleprinter, the high-speed reader/punch, or the system device.

g. When the ACCEPT statement is used, the rubout character deletes the previous number as shown in the following examples.

Typed and Corrected	Read
Integer Numbers:	
128 1028	+1028
128 -28	-128
-128 128	+128
Floating-point numbers:	
2 42	+42.0
+2. 42	+42.0
-2.0 2.0	+2.0
42 -42.2	-42.2
20E6 5	$+2.0 \times 10^5$
2.0E-6 5	$+2.0 \times 10^5$

h. When the READ statement is used, the rubout character is completely ignored.

The following examples show how the READ and WRITE statements might be used in a typical FORTRAN program.

```
C       EXAMPLE PROGRAM TO READ COORDINATE PAIRS
C       FROM THE TELETYPE AND STORE THEM ON
C       THE SYSTEM DEVICE
        DEFINE DISK
        TYPE 100
100     FORMAT ("ENTER THE NUMBER OF COORDINATE PAIRS"/)
        ACCEPT 10,N
10      FORMAT (I)
        TYPE 102
102     FORMAT ("NOW ENTER THE COORDINATES"/)
        DO 20 I=1,N
        ACCEPT 30,X,Y
        WRITE 3,30,X,Y
20      CONTINUE
        STOP
30      FORMAT (E,E)
        END
```

Several READ and WRITE statements may occur within a single DO loop and may refer to different devices. The data is written in USA SCII format regardless of the device used. The following program demonstrates how information previously stored on the disk might be read, processed, and punched using the high-speed punch.

```
C       FORTRAN EXAMPLE PROGRAM
        DEFINE DISK
        DIMENSION X(100),Y(100)
C       READ DATA FROM THE DISK DEVICE NR3
        IDEV=3
6       SUMX=0
        SUMY=0
        DO 10 I=1,100
        READ IDEV,20,X(1),Y(1)
        WRITE 2,20,X(1),Y(1)
        SUMX=SUMX+X(1)
        SUMY=SUMY+Y(1)
10      CONTINUE
        TYPE 30,SUMX,SUMY
        ACCEPT 40,J
        IF (J) 12,12,6
12      STOP
20      FORMAT (E,E)
30      FORMAT ("SUM OF X VALUES = ",E," SUM OF Y VALUES = ",E,"
        //"TYPE 0 TO STOP, 1 TO CONTINUE")
40      FORMAT (1)
        END
```

3.4.1 Compiler

The compiler consists of a loader (FORT) and the main portion of the compiler (.FT.). This version of the compiler differs from the standard PDP-8 4K FORTRAN compiler in the following ways.

a. It uses the disk or DECtape unit during its operation.

b. It will compile programs which have been stored on the system devices or programs which have been prepared on punched paper tape.

c. It will generate a FORTRAN binary output file either on the system devices or on punched paper tape.

d. Significant improvements have been employed with the READ and WRITE statements.

e. Input and output devices are determined using the Command Decoder

f. It is possible to terminate compilation at any time by typing ↑C, thus returning control to Monitor.

g. Within certain restrictions, a program compiled on a system device may be executed immediately when the user types ↑P after compilation of the program.

h. Statement numbers need not be delimited by a semicolon, unless the user wishes them to be employed for appearance.

i. Statements without preceding numbers must be preceded by a space, a tab, or a semicolon.

3.4.1.1 Loading the FORTRAN Compiler -- To load the compiler, the following steps must be performed.

a. Load the compiler loader (FORT) into core using Loader in one pass and save it on the system device as shown in Appendix E.

b. Load the compiler (.FT.) into core using Loader in two passes and save it on the sys tem device as shown in Appendix E. The compiler is now loaded and saved on the system device and is ready for use. The entire procedure will generate the following printout.

```
.LOAD ↵
*IN-R: ↵
*
*OPT-1 ↵
*ST=7600 ↵
↑↑
.SAVE FORT!0-1777;  200↵   (See Appendix E.)
.LOAD ↵
*IN-R: ↵
*
*OPT-2 ↵
*ST=7600 ↵
↑↑↑↑
.SAVE .FT.!200-7377; ↵  (See Appendix E.)
.
```

The loader occupies core locations 0-1777 with a starting address at 200. The compiler occupies core locations 200-7377, its starting address is not specified since the loader (not the user) calls .FT. when needed.

3.4.1.2 Operating Procedures -- The FORTRAN compiler is transferred from the system device into core when the user responds to Monitor's period with FORT, as shown below.

```
.FORT ↵
```

Command Decoder then types

```
*OUT-
```

and waits for the user to specify one of the following:

T: ↵	Output on low-speed punch/printer
R: ↵	Output on high-speed punch
S:name ↵	Output on system device and assign name
↵	No output desired

Command Decoder will respond with an * when it recognizes a valid output device, and then types

*IN-

and waits for the user to specify one of the following:

T: ↲	Input to be from low-speed reader
R: ↲	Input to be from high-speed reader
S:name ↲	Input to be from system device file named

Command Decoder will type an * when it recognizes a valid input device.

The compiler now assumes control, and if the program to be compiled is on paper tape, the compiler types ↑ when it is ready to receive the tape for compilation.

When the user is ready to read in his program he should type ↑ P, which initiates compilation. At the end of compilation the compiler will type any error diagnostics necessary, a carriage return/line feed, and ↑.

The process described above would produce the following printout.

```
.FORT ↲
*OUT-R: ↲
*
*IN-R: ↲
*
↑
↑                  (↑C typed here;  compilation finished)
```

3.4.1.3 Compiler Diagnostics -- Certain errors can make it impossible for the compiler to proceed in the normal manner. These are referred to as system errors. They may be caused by improperly loading the compiler, by not having an END statement on a source file, by a machine malfunction, or for various other reasons.

There are two types of system errors: those which occur before the compiler has been loaded into core, and those which occur after the compiler has been loaded into core. In the first case, the compiler will type a four-digit error code and return control to the Monitor. In the second case, the compiler will type SYS followed by a four-digit error code. At this point the operator must type ↑C to return control to the Monitor.

Table 3-6 lists the system error messages.

Table 3-6
Compiler Systems Diagnostics

Error Code	Explanation
0227	Could not find Command Decoder on system device
0231	Same as above
0326	Could not find .FT. on system device
0330	Same as above
1425	READ error during directory or SAM block search
1521	Same as above
1626	Same as above
1726	WRITE error during SAM block search
3100	Illegal operator on compiler stack[1]
3417	Pre-precedence error[1]
4737	No input device or invalid input device specified
6141	Attempt to execute a program not compiled onto the system device
6145	Could not find FOSL on system device; if the error occurs, it may be necessary to reload FORT and FOSL.
6207	READ error while loading FOSL
6211	Error while doing SAM block manipulation[1]
6223	Error while loading .FT.
6226	Same as above
6257	Same as above
6407	Illegal overlay request[1]
6416	Same as above
6467	System device READ error
6724	No END statement on source device
6746	Same as above
7114	Same as above
7136	READ error on system device source file
7150	System device full
7173	WRITE error on system device output file

[1] Error may be due to a compiler error or a machine malfunction.

The example below illustrates the appearance of the error codes.

```
.FORT ↵
0227                  Command Decoder not on system device
.FORT ↵
*OUT- ↵
*
*IN-S:name ↵
SYS 6141 <↑C>         No output file specified
```

Error messages for errors which occur during compilation of a program are typed out upon completion of the compilation. These errors are referred to as compilation errors and take the form:

```
XXXX      XX      XX
 |         |       └── The error code
 |         |
 |         └── The number of statements since the appearance
 |             of last numbered statement (octal)
 |
 └── The statement number of the last numbered statement
```

For example, during compilation of the statements

```
          :
10     A=I (J+1)
       B=A*(B+SINF(THTA)
          :
```

the error message

```
10  11  11
```

would be printed, indicating that an error exists in a statement which occurs 11 statements (octal) after the appearance of statement 10. The message corresponding to error code 11 shows that the number of left and right parentheses in the statement is not equal. The statement is examined and corrected, then compilation is resumed.

Table 3-7 lists the compilation error

Table 3-7
Compiler Compilation Diagnostics

Error Code	Explanation
00	Mixed mode arithmetic expression
01	Missing variable or constant in arithmetic expression
03	Comma was found in an arithmetic expression
04	Too many operators in this expression
05	Function argument is in fixed-point mode
06	Floating-point variable used as a subscript
07	Too many variable names in this program
10	Program too large, core storage exceeded

Table 3-7 (Cont)
Compiler Compilation Diagnostics

Error Code	Explanation
11	Unbalanced right and left parentheses
12	Illegal character found in this statement
13	Compiler could not identify this statement
14	More than one statement with same statement number
15	Subscripted variable did not appear in a DIMENSION statement
16	Statement too long to process
17	Floating-point operand should have been fixed-point
20	Undefined statement number
21	Too many numbered statements in this program
22	Too many parentheses in this statement
23	Too many statements have been referenced before they appear in the program
25	DEFINE statement was proceeded by some executable statement
26	Statement does not begin with a space, tab, C, or number

3.4.1.4 Debugging Aid (Symbolprint) -- Symbolprint is a program which may be used with the FORTRAN compiler. Its purpose is to help the user create and debug his FORTRAN programs by providing certain information about the compiler-generated interpretive code. Symbolprint may be used only immediately after a program has been compiled by using the Disk/DECtape FORTRAN compiler.

Symbolprint provides the following information:

a. The limits of the interpretive code.
b. A list of variable names and their corresponding locations (the symbol table).
c. A list of statement numbers and their corresponding locations (the statement number table).

Symbolprint is loaded into core from punched paper tape and may be saved on the system device approximately as shown below (see Paragraph 2.5).

```
.LOAD ↲
*IN-R: ↲
*
*OPT-1 ↲
*ST= ↲
↑↑
.SAVE STBL!600-777;600 ↲    (See Appendix E.)
.
```

When in core, Symbolprint occupies locations 600-777 with its starting address at location 600.

When symbolprint is called into core, it types the interpretive code limits, symbol table, statement number table, carriage return/line feed, and ↑. At this point the user may execute his program by typing ↑P, or he may return to Monitor by typing ↑C.

In the following example, a program named SRC is compiled with no output specified. Symbolprint is then used as shown above.

```
.FORT ↵
*OUT- ↵
*
*IN-S:SRC ↵
*
↑                       (↑C typed here)
.STBL↵
     6154  7565
N  7576
I  7575                 (Symbol Table)
X  7571
Y  7566
0100  6033
0010  6060
0102  6066              (Statement Number Table)
0020  6145
0030  6147
↑<↑C>
.
```

In the example above, location 6154 is the highest location used for interpretive code and location 7565 is the lowest location used for data, indicating that the part of core between 6145 and 7565 is unused. Interpretive code starts at location 600 if a DEFINE statement appears in the program; otherwise, the code starts at location 5200.

3.4.2 Operating System

The FORTRAN operating system consists of a loader (FOSL) and the interpreter and arithmetic subroutine package (.OS.). This version of FOSL differs from the paper tape FORTRAN operating system in the following ways.

a. It will load and execute programs which have been compiled and saved on the system device or programs which have been compiled on paper tape.

b. FOSL may be called directly by the compiler when a program has been compiled and saved on the system device. This is referred to as compile-and-go mode.

c. FOSL is able to recognize READ and WRITE statements which may read and write data in USA SCII format on either the low-speed paper tape reader/punch, the high-speed paper tape reader/punch, or the system device.

d. The execution of a FORTRAN program may be interrupted by the user at any time by typing ↑C; control will be returned to Monitor.

3.4.2.1 Loading the FORTRAN Operating System -- To load the operating system, the following steps are performed.

a. Load the operating system loader (FOSL) using Loader in one pass and save it on the system device as shown in Appendix E.

b. Load the operating system interpretive and arithmetic package (.OS.) by using Loader in one pass and save it on the system device as shown in Appendix E. The FORTRAN operating system is now loaded and ready for use. The loading process will generate the following printout.

```
.LOAD ↲
*IN-R: ↲
*
*OPT-1 ↲
*ST= ↲
↑↑                   1577
.SAVE FOSL!0-[illegible];200 ↲      (See Appendix E.)
.LOAD ↲
*IN-R: ↲
*
*OPT-1 ↲
*ST= ↲
↑↑↑↑                 5177
.SAVE .OS.!0-[illegible];0 ↲        (See Appendix E.)
.
```

The loader occupies core locations 0-1577 with its starting address at 200. The arithmetic and subroutine package occupies core locations 0-5177; its starting address is not specified since the loader (not the user) calls .OS. when needed.

3.4.2.2 Operating Procedures -- The FORTRAN operating system may be transferred from the system device into core in one of two ways: by typing ↑P immediately after compiling a FORTRAN program onto the system device, or by typing FOSL immediately after Monitor types a period.

If the operating system is called from Monitor, specify the desired input device by typing T:↲ for low-speed reader, R:↲ for high-speed reader, or S:name↲ for system device input. FOSL will type * when it recognizes a valid input device.

FOSL will type *OPT-. If input or output is to be to or from the system device, type S. Any other character indicates that the system device is not to be used. However, if the S option is used, FOSL will type *OUT-. The user should now specify the desired output filename (if any) by typing S:name↲ (name is the name of the file). FOSL will ask for the input filename by typing *IN-. The user should respond with S: and the name of the file, followed by a carriage return.

If the FORTRAN program is on paper tape, Loader will type ↑ when it is ready to begin loading. When the user is ready to load his program, he types ↑P and the tape will begin loading.

When the FORTRAN program or file is loaded, FOSL will type *READY, followed by a carriage return/line feed and ↑. Place data tapes in the appropriate reader and type ↑P to begin executing the program. (If the low-speed reader is used, turn the reader ON after typing ↑P.)

When a STOP or END statement is executed, or when an end-of-file is read on the system device, the operating system will type ! and return control to the Monitor.

The following examples show how the FORTRAN operating system may be used.

Example 1

```
.FOSL ↵
*IN-S:FBIN ↵
*
*OPT- ↵
*READY
 ↑
                        (Program execution occurs here)
```

Example 2

```
!
.
.FOSL ↵
*IN-R:,R: ↵
*
*
↑↑
ERROR 01                (↑P typed here)
*READY
↑
                        (Program execution begins here)
```

Example 3

```
.FORT ↵
*OUT-S:SMSQ ↵
*
*IN-S:SMSQ ↵            Compile
*                         and
↑                         Go
*READY ↵
↑
                        (Program execution begins here)
```

Example 4

```
.FOSL ↵
*IN-S:BIN ↵
*
*OPT-S
*OUT-S:DAT2 ↵
*
*IN-S:DAT1 ↵
*
*READY
↑
                        (Program execution begins here)
```

In example 2 a checksum error was detected on the second input tape. In this case the operator decided to attempt to execute the program in spite of the checksum error.

3.4.2.3 Operating System Diagnostics -- When an error occurs during program execution, the operating system will type ERROR followed by a two-digit error code number which will indicate the cause of the error. Depending on the nature of the error, it may be possible to continue program execution by typing ↑P or it may be necessary to return to the Monitor by typing ↑C.

The following is a list of the operating system error messages.

Table 3-8
Operating System Diagnostics

Error Code	Explanation
01	Checksum error on FORTRAN binary input
02	Illegal origin or data address on FORTRAN binary input
04	System device input-output error[1]
05	High-speed reader error
06	Illegal FORTRAN binary input device
11	Zero divide error
12	Floating-point input data conversion error
13	Illegal op code
14	System device input-output error[1]
15	Non-FORMAT statement used as a FORMAT
16	Illegal FORMAT specification
17	Floating-point number larger than 2048
20	Square root of a negative number
21	Exponential negative number
22	Logarithm of a number less than or equal to zero
40	Illegal device code used in READ or WRITE statement
41	System device full, could not complete a WRITE statement
76	Stack underflow error[2]
77	Stack overflow error[2]

[1] May be caused by machine malfunction or operating system error.

[2] May be caused by source program or loading error; to correct, do the following in descending order.

a. Use Diagnose to determine where the error occurred.
b. Recompile the source program.
c. Examine source program (in particular the arithmetic statements and subscripted variables).

When an error occurs, execution will stop and the operating system will wait for the user to type ↑P or ↑C.

3.4.2.4 Debugging Aid (Diagnose) -- Diagnose is a basic system program whose purpose is to help the user debug his FORTRAN program. It is intended to be used in conjunction with the PDP-8 4K FORTRAN Operating System and revised FORTRAN Symbolprint. Diagnose provides the following information.

a. If stack overflow or underflow has occurred, it will type a message indicating which of the five run-time stacks caused the error.

b. It will type a message indicating the contents of the current location counter (CLC).

c. If the counter stack is nonempty, it will type the contents of that stack.

d. If location zero is nonzero, it will type the contents of that location (minus one), indicating the point at which some FORTRAN systems error occurred.

Diagnose is loaded into core from punched paper tape and may be saved on the system device as shown in Appendix E.

```
.LOAD↵
*IN-R:↵
*
*OPT-1↵
*ST= ↵
↑↑
.SAVE DIAG!200-1177;200↵     (See Appendix E.)
.
```

When in core, Diagnose occupies locations 200-1177 with its starting address at location 200.

Diagnose is called by typing the letters DIAG to the Monitor. It may be used any time the FORTRAN 4K Operating system is in core. (If it is called any other time, the information typed will be meaningless.)

The use of Diagnose is demonstrated by the example of the following test program which contains a large amount of arithmetic calculations.

Program 1:

```
*L
C       FORTST
C       PDP-8 ADVANCED SOFTWARE
C       FORTRAN TEST 1/2/68
        DIMENSION ADIFE(6),AFAC(3),APIPE(6),IMRCD(3),PP(27)
        ,ACPRI(3)
        TYPE 1
1       FORMAT("PDP-8 4-K FORTRAN TEST"/)
        ASPVA=.60
        APIPE(1)=12.09
        APIPE(3)=6.66
        APIPE(4)=5.
        APIPE(5)=5.0
        IMRCD(1)=30
        IMRCD(2)=30
        ADIFE(1)=47.
        ADIFE(2)=47.
        ADIFE(4)=508.
        ADIFE(5)=3857048.
        AF=37.96
```

```
          SC=3.1416
          AMEAS=9.02
          FSUBB=10.0
          ASUVA=100.98
          DO 200 I=1,27
          READ 2,199,PP(I)
199       FORMAT(E)
200       CONTINUE
          AGAST=38
          INORU=2
25        BSPVA=(1./ASPVA)**.5
          DO 550 JCB=1,INORU
          AVEDE=IMRCD(JCB)
          BE=APIPE(JCB+3)/APIPE(JCB)
          IF(BE-.75)471,472,472
472       AK=.731
          GO TO 16
471       AG=.075
          DO 100 IE=1,27
          AG=AG+.025
          IF(AG-BE) 100,100,110
100       CONTINUE
110       TOTA=PP(IE)
          TOTB=PP(IE-1)
          SC=.025-(AG-BE)
          WRITE 2,991,TOTA,TOTB,SC,AG,BE,IE
991       FORMAT(/" 1",E,E,E,E/" ",E,I)
          IF(TOTA-TOTB)120,120,130
120       AK=TOTA
          GO TO 16
130       AK=TOTB+(SC *(TOTA-TOTB))/.025
16        FRD=830.-5000.*BE+9000.*BE**2-4200.*BE**3+(530./APIPE(JCB)**.5)
          BMEAS=AMEAS+14.4
          FR=1.+((F RD/(12835.*AK))/((BMEAS*AVEDE)**.5))
          XSUB2=AVEDE/(27.7*BMEAS)
          YTTA=(XSUB2+1.)**.5
          YTTB=.35*BE**4.+41.
          YTTC=XSUB2/(1.3*YTTA)
          YSUB2=YTTA-YTTB*YTTC
          ACPRI(JCB)=YSUB2*FR*1.0177*FSUBB
110       TOTA+PP(IE)
          TOTB=PP(IE-1)
          SC=.025-(AG-BE)
          WRITE 2,991,TOTA,TOTB,SC,AG,BE,IE
991       FORMAT(/" 1",E,E,E,E/" ",E,I)
          IF(TOTA=TOTB)120,120,130
120       AK=TOTA
          GO TO 16
130       AK=TOTB+(SC*(TOTA-TOTB))/.025
16        FRD=830.-5000.*BE+9000.*BE**2-4200.*BE**3+(530./APIPE(JCB)**.5)
          BMEAS=AMEAS+14.4
          FR=1.+((FRD/(12835.*AK))/((BMEAS*AVEDE)**.5))
          XSUB2=AVEDE/(27.7*BMEAS)
          YTTA=(XSUB2+1.)**.5
          YTTB=.35*BE**4.+41.
          YTTC=XSUB2/(1.3*YTTA)
          YSUB2=YTTA-YTTB*YTTC
          ACPRI(JCB)=YSUB2*FR*1.0177*FSUBB
          AFAC(JCB)=ADIFE(JCB)*BSPVA
```

```
            WRITE 2,992,AK,FRD,AMEAS,BMEAS,FR,XSUB2,YTTA,YTTB,'
            YTTC,YSUB2,ACPRI(JCB),JCB
992         FORMAT(/" 2",E,E,E,E)
550         CONTINUE
            AFTF=(520./(460.+AGAST))**.5
            AFPV=(1.+(ASUVA*AMEAS)/((AGAST+460.)**3.825))**.5
            FLOW=0
            RATE=0
            DO 38 I=1,INORU
            AMWP=(ADIFE(1)*AMEAS)**.5/1000.
            RATE=RATE+ACPRI(I)
            FLOW=FLOW+AFTF*(AFAC(I)*AFPV*AMWP)
38          CONTINUE
            WRITE 2,993,AFTF,AFPV,AMWP,FLOW,RATE
            TYPE 14, FLOW,RATE
14          FORMAT(E,E/)
            STOP
993         FORMAT(/E,E,E,E)
            END
```

```
.STBL

  6360      6756

ADIF        7555
AFAC        7544
APIP        7522
IMRC        7517
PP          7376
ACPR        7365
ASPV        7362
AF          7310
SC          7302
AMEA        7274
FSUB        7266
ASUV        7260
I           7254
AGAS        7246
INOR        7244
BSPV        7240
JCB         7231
AVED        7225
BE          7222
AK          7213
AG          7205
IE          7201
TOTA        7171
TOTB        7166
FRD         7153
BMEA        7124
FR          7116
XSUB        7102
YTTA        7074
YTTB        7063
YTTC        7047
YSUB        7041
AFTF        7032
AFPV        7016
FLOW        6777
RATE        6773
AMWP        6766
```

$\updownarrow$ Symbol Table

```
0001        5203
0199        5411
0200        5414
0025        5426
0472        5507
0471        5515
0100        5547
0110        5550
0991        5615    Statement Number Table
0120        5650
0130        5656
0016        5676
0992        6147
0550        6162
0038        6323
0014        6342
0993        6350
```

Example 1(a)

```
↑
*READY
↑
PDP-8 4-K FORTRAN TEST
 0.255323E+1     -0.825572E+1
!
.DIAG

CURRENT LOCATION COUNTER AT 6347
```

Example 1(b)

```
.FOSL
*IN-S:BIN
*
*READY
↑
PDP-8 4-K FORTRAN TEST

ERROR 05                (↑C typed here)
.DIAG

CURRENT LOCATION COUNTER AT 5407
```

Example 1(c)

```
.FOSL
*IN-S:BIN
*
*READY
↑
PDP-8 4-K FORTRAN TEST          (↑C typed here)

.DIAG

CURRENT LOCATION COUNTER AT 4404

COUNTER STACK...
4733
4716
4673
6024
```

In example 1(a), the program was run to completion after which Diagnose was called. Diagnose indicated that the current location counter contained 6347. By referring to the statement number table (top of page 3-29, we can see that the CLC was pointing to an address just above statement 993 (address 6350), verifying that the program terminated normally at that point.

In example 1(b), program execution was attempted without paper tape in the high-speed reader. After observing the error diagnostic 05, Diagnose was called, indicating that CLC=5407. Again referring to the statement number table, we note that the address 5407 must refer to a statement just before statement number 199 which is indeed the READ statement at which the error occurred.

In example 1(c), program execution was arbitrarily stopped when the user typed ↑C. It should be noted that in this case the CLC contained a 4404 which is outside the user's interpretive code area. In such cases it is necessary to refer to the counter stack in order to determine where the program interruption occurred. The last address on the counter stack points to location 6024, and by again referring to the statement number table we can determine that the program was interrupted at some point between statements 16 and 992.

Program 2 is a FORTRAN program in which a missing operator appears on the 6th line. When program execution is attempted a stack overflow (error 77) occurs. Diagnose indicates that the operand stack has overflowed, which suggests some noncompiler detected error in the source program. By referring to the statement number table, which is typed afterwards, we note that the CLC points just before statement 10, which happens to be the source of the error. It should be pointed out, however, that when stack overflow or underflow occurs the CLC will not always point to the source of the error. It may be necessary to examine the entire program for errors of this type.

Program 2:

```
.EDIT
*OUT-S:SRC
*
*IN-
*
*OPT-B
*I
C          FORTRAN TEST
           B=1
           C=2
           D=3
           DO 10 I=1,160
           A=B(C+D)
10         CONTINUE
           TYPE 20, A
20         FORMAT(E)
           STOP
           END

*E

.FORT
*OUT-S:BIN
*
*IN-S:SRC
```

```
*
↑
*READY
↑
ERROR 77
                              (↑C typed here)
.DIAG
OPERAND STACK OVERFLOW
CURRENT LOCATION COUNTER AT 5231
.FORT
*OUT-
*
*IN-S:SRC
*
↑
                              (↑C typed here)
.STBL
  5251        7555
B            7574
C            7570
D            7564
I            7562
A            7555
0010         5237
0020         5244
↑

.
```

When Diagnose finishes typing the appropriate information control returns to the Monitor since it is impossible to resume FORTRAN program execution.

3.4.3 Examples

```
.LOAD↲                          Call Loader
*IN-R:↲                         Input to be from high-speed reader
*                               Input device is valid
*OPT-1↲                         One-pass load
*ST=↲                           Return to Monitor after loading
↑↑                              Loader-driver is loaded
.SAVE FORT!0-1777;200↲           and saved on the system device
.LOAD↲                          Call Loader
*IN-R:↲                         Input to be from high-speed reader
*                               Input device is valid
*OPT-2↲                         Two-pass load
*ST=↲                           Return to Monitor after loading
↑↑↑↑                            Compiler is loaded
.SAVE .FT.!200-7377;0↲           and saved on the system device
.LOAD↲                          Call Loader
*IN-R:↲                         Input to be from high-speed reader
*                               Input device is valid
*OPT-1↲                         One-pass load
*ST=↲                  6577     Return to Monitor after loading
↑↑                              Operating system loader is loaded
.SAVE FOSL!0-[illegible];200↲    and saved on the system device
```

```
.LOAD↵                          Call Loader
*IN-R:↵                         Input to be from high-speed reader
*                               Input device is valid
*OPT-1↵                         One-pass load
*ST=↵                           Return to Monitor after loading
↑↑                              Interpretive and arithmetic package is loaded
.SAVE .OS.!0-4777;0↵             and saved on the system device

.LOAD↵                          Call Loader
*IN-R:↵                         Input to be from high-speed reader
*                               Input device is valid
*OPT-1↵                         One-pass load
*ST=↵                           Return to Monitor after loading
↑↑                              Symbolprint is loaded
.SAVE STBL!600-777;600↵          and saved on the system device

.EDIT↵                          Call Editor
*OUT-S:FORT↵                    Output to be on system device
*                               Output device is valid
*IN-R:↵                         Input to be from high-speed reader
*                               Input device is valid
*OPT-B↵                         Leave blanks (spaces) unchanged
*E                              Write the program on the system device
                                 then write an end-of-file

.FORT↵                          Call FORTRAN compiler
*OUT-S:FORT↵                    FORTRAN binary output to be on system device
*                               Output device is valid
*IN-S:FORT↵                     USA ASCII input to be from system device
*                               Input device is valid
↑<↑C>↵                          Compilation is finished, return to Monitor

.STBL↵                          Call FORTRAN Symbolprint

  6177        7565              Core between 6200 and 7564 is unused

M             7576
A             7573
B             7570              Symbol table (typed by Symbolprint)
ANS           7565

0001          5200
0002          5257
0003          5413
0004          5570
0005          5717
0006          5754
0009          5760
0100          5763
0200          5766
0300          5771              Statement number table (typed by Symbolprint)
0400          5774
0500          5777
1000          6027
2000          6040
3000          6051
4000          6062
1500          6071
0008          6077
0007          6123
```

```
↑                           Symbolprint is finished, load operating system
                             and interpretive code
*READY ↲                    Operating system and interpretive code are loaded
↑                           Execute the program
THIS IS A DEMONSTRATION OF PDP FORTRAN.
THIS PROGRAM WAS COMPILED IN ONE PASS      (↑C typed here)
.FOSL ↲                     Call operating system and loader
*IN-S:FORT↲                 FORTRAN binary input is on system device
*                           Input device is valid
*OPT- ↲                     No input or output to be done on system device
                             during program execution
*READY ↲                    Operating system and interpretive code have been
                             loaded
↑                           Begin program execution
THIS IS A DEMONSTRATION OF PDP FORT       (↑C typed here)
.FORT↲                      Call FORTRAN compiler
*OUT-S:FORT ↲               Output to be on the system device
*                           Output device is valid
*IN-S:FORT↲                 Input to be from the system device
*                           Input device is valid
↑                           Compilation is finished, loading operating system
                             and interpretive code
*READY ↲                    Operating system and interpretive code are loaded
↑                           Begin program execution
THIS IS A DEMONSTRATION OF                 (↑C typed here)

.
```

3.5 DDT-D

DDT-D (Dynamic Debugging Technique for the Disk/DECtape System) is used for on-line checking, testing, and altering object programs by typing from the teleprinter keyboard. When debugging on-line, the user checks his program at the computer, controlling its execution, and making corrections or changes to his program while it is running on the computer.

When using DDT-D, the user should have a listing of his program and its symbols so that he can update the program listing as corrections and changes are made to his program. The user may refer to variables and tags by their symbolic names or by their octal values.

DDT-D operates as described in DDT Programming Manual, DDT-8 (Doc. No. DEC-08-CDDA-D), except where that manual differs from this one, in which case this manual has precedence.

DDT-D can be considered as being in three sections.

a. DDT Proper	~~Self-explanatory; occupies core locations 200-4577 and the three breakpoint locations,~~
b. Driver	Resident in core with its origin set above DDT proper (above 4577); it is a two-page program plus a one-page once-only program, and it contains breakpoint insertion and removal logic, overlay routines, continuation iteration count and control, and breakpoint list.

A slightly modified version of DDT-8 (low) occupying core locations 200-4577 + 3 breakpoint locations

c.	User Core Image File	Occupies same storage area as DDT proper and is used for swapping DDT proper and the user program to and from the system device.

DDT-D is an expanded version of DDT-8 with the following exceptions.

a. Three breakpoints (as opposed to only one in DDT-8)

b. No punching (program may be output on the system device)

c. No switch options (user direction is via keyboard)

d. No halts (continues when user types ↑ P)

Variations in commands[1] follows.

a.	[O, [S, [Y, [L, [M	Are temporary modifications to their respective constants; are reset at every entrance to DDT-D from a [G or [C
b.	[P	Continue (Monitor types ↑ to indicate that it is waiting for ↑P)
c.	[C	Restore user core image and return to Monitor
d.	n[Bk	Set breakpoint; where n is the address of the break, [B is the breakpoint command, and k is 1, 2, or 3

NOTE

If user tries to set two breakpoints at the same address, a ? is typed and no action occurs.

e.	n[B, [T, a;b[P, [E	Have been removed
f.	[R	Is switch independent

The following subroutines have been added.

a.	ADDCHK	Finds word to be examined and puts it in WORD 2; remembers if last virtual word referenced was in same buffer as present virtual word and reads only if required.
b.	ADDMOD	Updates real or virtual core.
c.	DDTB	Updates symbol table pointer, gets value of breakpoint and its contents, types breakpoint number and a - (hyphen) if a breakpoint, and goes to TRAP or types nothing and goes to START if breakpoint number = 0.
d.	STOSYM	Updates DDT proper symbol area (DDT proper must be on unprotected disk).
e.	READS and SYMIO	Input-output routines for disk; a failure in either types S and goes to start of DDT.

[1] The ALT MODE key precedes each command character and is echoed as [.

The following subroutines have been modified as indicated.

a.	REDTAB	Assumes user wants to add to existing symbol table; user must type [X to clear the symbol table.
b.	FINIS	Does not halt, instead, it waits for the user to type ↑P.
c.	CHANGE	Allows lookup of values to change limit of search and search mask.
d.	TODDT	Handles breakpoint insertion; transferred to DDT driver.
e.	TRAP	Breakpoint handler; transferred to DDT driver.

The following subroutines have been removed.

a. PUNWOR
b. FSTPUN
c. FUN
d. PUNCHK
e. PUNLDR
f. WHICH
g. CHKSUM

From the teleprinter keyboard, the user can automatically stop his program at up to three strategic points by setting breakpoints, which may be set before the debugging run is started or during another breakpoint stop. To set a breakpoint, the user types the absolute address or symbolic tag of the location where he wants his program to stop, the ALT MODE key, the B key, and then the breakpoint number. For example,

3400[B1	(absolute address, ALT MODE, B, 1)
HERE[B2	(symbolic tag, ALT MODE, B 2)

Locations 3, 4, and 5 on page zero are used as the breakpoint locations. The user may, however, reset the breakpoint locations to any three contiguous locations on page zero by setting BRKCEL=n, where n is the lowest of the three locations desired. When the user sets his breakpoints, DDT-D remembers the locations set with BRKCEL=n.

The following symbols are the address tags of certain registers in DDT-D whose contents are available to the user.

a.	A	Accumulator storage (at breakpoints)
b.	Y	Link storage (at breakpoints)
c.	M	Mask used in search
d.	L	Lower limit of search
e.	U	Upper limit of search

Table 3-9 lists the DDT-D commands available to the user.

Table 3-9
DDT-D Commands

Character	Action
(space)	Separation character
+	Arithmetic plus
-	Arithmetic minus
/	Location examination character; when it follows the address of a location, it causes the contents of that location to be printed
↵ (carriage return)	Make modifications, if any
↑ (line feed)	Make modifications, if any, and print the contents of the next sequential location
=	Type last quantity as an octal integer
. (period)	Current location
← (left arrow)	Delete the line currently being typed
[S	Sets DDT-D to type out in symbolic mode
[O	Sets DDT-D to type out in octal mode
n[W	Word search for all occurrences masked with C([M) of the expression n
k[Bn	Insert breakpoint n at location k (n = 1, 2, or 3)
[Bn	Remove breakpoint n (n = 1, 2, or 3)
n[C	Continue n times automatically; if n is absent, it is assumed to be 1
k[G	Go to location k
[R	Append symbol table into external symbol table or define symbols on line

3.5.1 Loading and Saving

DDT-D is loaded into core from punched paper tape. The tape is in two sections. The first section contains DDT proper which loads in one pass, occupies core locations 200-4577 (Appendix E) and uses 3 locations [redacted] (page 0) as the breakpoint locations.[2] After loading DDT proper, the user should reserve on the system device a user core image file name .SYM, which should also be assigned to core locations 200-4577.[1]

The next section of DDT (the driver) loads in two passes and occupies two pages in core with its origin anywhere above DDT proper, that is, anywhere above location 4577. The driver is resident in core. For setup, it uses five more pages: one for once-only code plus four for Command Decoder. Command Decoder expects two inputs to be assigned as files to be used by the driver. These files are assigned only once unless the system is changed or destroyed, in which case the user must reassign these two files.

[1] .SYM is used also by PAL-D to store additional symbol table entries.

[2] Bin tape of DDT driver used locations 3-5, 7200-7577

The sections of DDT are loaded and saved as described below.

```
.LOAD ↲                          Call Loader using Monitor
*IN-R: ↲                         Input to be from high-speed reader
*                                Loader found input device valid
*OPT-1 ↲                         DDT proper loads in one pass
*ST= ↲                           Return to Monitor after loading
↑↑                   4577        DDT proper is loaded
.SAVE .DDT:200-4[illegible];0 ↲           Saved as a user program      (See Appendix E.)
.SAVE .SYM:200-4577;0 ↲          User core image file also    (See Appendix E.)
                                  saved as a user program
.LOAD ↲                          Call Loader using Monitor
*IN-R: ↲                         Input to be from high-speed reader
*                                Loader found input device valid
*OPT-2 ↲                         Driver loads in two passes
*ST=7000 ↲                       Transfer to once-only code after loading
↑↑↑↑                   USER      Driver is loaded
*IN-S:<.DDT>,S:<[illegible]> ↲           Assign 2 input files for use by
                                  driver                      (See Appendix E.)
*IN-S:.DDT,S:.SYM ↲              Inputs to be from DDT proper and user
                                  core image files
*                                Loader found input files valid (an
*                                 asterisk for each valid file (device)
.SAVE DDT!7200-7577;7200 ↲       Saved as a system program    (See Appendix E.)

.
```

The error message DDT? is typed whenever an error is encountered while loading DDT-D. Errors may be caused by the following.

a. User file too large

b. System device read error

c. No Command Decoder

3.5.2 Operating Procedures

DDT-D is now saved on the system device. The user must now load into core the program to be debugged. This is done as described in Paragraph 2.5.

When the program to be debugged is in core the user types DDT↲ in response to Monitor's period as shown below.

```
.DDT
```

The user may now use DDT-D in debugging this program, directing execution and making modifications to his program as described above and in the DDT-8 programming manual.

A brief example of using DDT-D is shown in Paragraph 3.5.3.

3.5.3 Example

```
.LOAD ↲                          Call Loader
*IN-R: ↲                         Input to be from high-speed reader
*                                Loader found input device valid
*OPT-1 ↲                         One-pass load
ST= ↲                            Return to Monitor after loading
```

```
     ↑↑                                  DDT proper is loaded
     .SAVE .DDT:200-4577; 0 ↲           DDT proper is saved on disk     (See Appendix E.)
     .SAVE .SYM:200-4577;0 ↲            User code image file also saved
     .LOAD ↲                            Call Loader
     *IN-R: ↲                           Input to be from high-speed reader
     *                                  Loader found input device valid
     *OPT-2 ↲                           Two-pass load
     ST=7000 ↲                          Transfer to once-only code after loading
     ↑↑↑↑                               Driver is loaded
*IN- S:<.DDT>,S:<.SYM> ↲                Loader expects 2 input files for use by driver
     *IN-S:.DDT,S:.SYM ↲                Inputs from DDT proper and user code image file

     *
     *                                  Loader found both input files valid
     .SAVE DDT!7200-7577;7200 ↲         Driver is saved on disk          (See Appendix E.)
     .DDT ↲                             Call DDT using Monitor
     3400/AND 0007 IAC ↲                Examine contents of location
     3401/AND JMP 3400 ↲                  3400 and 3401
     3400[B1 ↲                          Set breakpoint No. 1 at location 3400
     3400[G ↲                           Start execution at location 3400
     1-3400)0000                        Location 3400 contains 0000
     [C ↲                               Continue
     1-3400)0001                        Location 3400 now contains 0001
     700[C ↲                            Pass through location 3400 700 times
     1-3400)0701                        Location 3400 now contains 0701
     .                                  ↑C was typed here
```

APPENDIX A
SYSTEM GENERATION

This appendix describes the creation of a Disk/DECtape System (Disk/DECtape Monitor and system programs) on an empty disk or DECtape (if DECtape, it must have timing and mark tracks previously written on it).

The steps involved in system generation are as follows.

a. Toggling in the Readin Mode (RIM) Loader.
b. Loading the Binary (BIN) Loader
c. Loading and executing Disk/DECtape System Builder to create Monitor.
d. Loading and saving any system programs.

A.1 TOGGLING IN THE READIN MODE (RIM) LOADER

The Readin Mode (RIM) Loader is a short program which loads any program in RIM format on paper tape into core. Although the RIM Loader has various uses, its sole purpose in the System Building process is to load the Binary Loader.

There are two versions of the RIM Loader, one for loading programs from the high-speed paper tape reader and the other for loading from the Teletype paper tape reader.

High-Speed Reader		Teletype Reader	
Location	Instruction	Location	Instruction
7756	6014	7756	6032
7757	6011	7757	6031
7760	5357	7760	5357
7761	6016	7761	6036
7762	7106	7762	7106
7763	7006	7763	7006
7764	7510	7764	7510
7765	5374	7765	5357
7766	7006	7766	7006
7767	6011	7767	6031
7770	5367	7770	5367
7771	6016	7771	6034
7772	7420	7772	7420
7773	3776	7773	3776
7774	3376	7774	3376
7775	5357	7775	5356
7776	0000	7776	0000

A detailed description of the toggling and checking procedures for the RIM Loader can be found in the PDP-8 Console Manual (Doc. No. DEC-08-NGCA-D). Acomplete discussion of the RIM Loader is contained in the PDP-8 Readin Mode Loader Program writeup (Doc. No. Digital-8-1-U).

A.2 LOADING THE BINARY (BIN) LOADER

The Binary (BIN) Loader loads any program in binary format on paper tape into core. Its purpose in the System Building process is to load the Disk/DECtape System Builder. The procedure for loading BIN is as follows.

a. Check that the RIM Loader is in core.

b. Place the paper tape containing BIN in the paper tape reader (high-speed or Teletype, according to version of RIM).

c. If Teletype reader is to be used, turn it on.

d. Place the address 7756 into the SWITCH REGISTER and press LOAD ADD.

e. Press START. Tape should begin reading (if it does not, check that the SING INST and SING STEP switches are down and that the reader is on line). (Note: The Teletype reader is always on line.) If the Teletype begins to print, flip Teletype switch from LOCAL to LINE and back up the tape to the leader/trailer.

f. After paper tape reads in, wait until only bit 0 of the accumulator is on. Press STOP on console. If the high-speed reader is used, a 7402 (HLT) appears in the accumulator, and the tape stops over the leader/trailer (200 code).

A detailed description of BIN and its use can be found in the PDP-8 Console Manual and PDP-8 Binary Loader Program writeup (Doc. No. Digital-8-2-U).

A.3 LOADING AND EXECUTING DISK/DECTAPE SYSTEM BUILDER

Next, the Disk/DECtape System Builder program, in binary format on paper tape, is loaded by the Binary Loader. Loading procedures are as follows.

a. Place the address 7777 (starting address of BIN) into the SWITCH REGISTER. Press LOAD ADD.

b. If the high-speed paper tape reader is to be used, put down (or set to 0) bit 0 of the SWITCH REGISTER, place the System Builder tape in the reader, and turn the reader on.

If the Teletype reader is to be used, leave up bit 0 of the SWITCH REGISTER, place the System Builder tape in the reader, put the reader on line, and press reader START.

c. Press START on the console. Tape should read in.

d. When tape has been read, the accumulator should contain all zeroes (if not, the program has loaded incorrectly; begin the loading procedure from the beginning).

e. Turn off WRITE PROTECT on the disk (if present). Otherwise, mount a DECtape reel on one of your DECtape units, set the unit selector to 8, and set the WRITE switch to WRITE.

f. To begin System Builder execution, place the address 0200 into the SWITCH REGISTER, press LOAD ADD, and then START.

g. As the following questions are typed out, answer them according to your machine configuration.

*TYPE SIZE OF CORE (IN K)↵	
*8↵	User enters core size of his machine (4, 8, 12, 16, 20, 24, 28, or 32).
*HIGH SPEED PAPER TAPE?	
*YES↵	User answers YES or NO.
*PDP-8/S?	
*NO↵	User answers YES or NO.
*DISC?	
*YES↵	User answers YES or NO.
*TYPE NUMBER OF DISC UNITS	
*1↵	User types number of disk units on his machine.
*TAPE?	
*YES↵	User types YES if he has DECtape, NO if he does not
•	~~A maximum delay of one minute occurs here while~~ Monitor creation is ~~being~~ completed, the resident portion is moved to the appropriate core area (7600 through 7777), and the nonresident portions are written on the system device.

NOTE

> If specified as present, the disk is automatically selected as the system device; if not, DECtape unit 8 is selected.

.	Monitor is loaded and ready. If the response

WRITE ERROR

occurs:

a. If disk, start over at ~~Step (a)~~ Para A.2; there may be a hardware problem.

b. If DECtape, try a new DECtape and start at Step (e). Or, rewrite the timing and mark tracks and start ~~at Step (a)~~. Para A.2

A.4 LOADING AND SAVING SYSTEM PROGRAMS

Binary Loader is one of the nonresident portions of Monitor and is used to load system and user programs into core. It is fully described in Chapter 2. An example follows.

.LOAD↵	Calls Binary Loader from the system device.
*IN-R:↵	Input device is paper tape reader (high-speed reader if specified as present at System Builder time; otherwise Teletype reader).
*	Device is valid.
*OPT-1	One-pass loading mode selected.
ST=7600↵	Return to Monitor after loading.
	After each up-arrow typeout, user types ↑P to continue (also must press CONTinue on console if Teletype reader is being used).
.SAVE PIP! 0-3177; 1000↵	Saves program (in this case, PIP) on system device. Note that a ! must follow name of system program. The SAVE command is explained in Chapter 2. The SAVE command program is given in Appendix E.
.	

Repeat the procedure above for each system program to be saved.

APPENDIX B
SYSTEM FORMATS

This appendix contains the following information.

a. System Device Layouts

   Disk Storage Layout
   DECtape Storage Layout
   Directory Name (DN) Block Format
   Storage Allocation Map (SAM) Block Format
   Table of System Device and Core Capacities

b. Data Structure

   Source File (ASCII)
   Binary File (BINARY, FTC BIN)
   Saved Files (SYS, USER)

c. PIP Listing of System Device Map (for Disk)

d. Monitor Core Usage Diagrams

B.1 SYSTEM DEVICE LAYOUTS

Figures B-1 and B-2 illustrate the layout of the system device for both disk and DECtape. Note that, although the layouts differ in arrangement, they are logically equivalent.

A relatively sophisticated file structure is used for all automatic retrieval of storage by the system. Two special types of blocks are required: Directory Name (DN) Blocks, and Storage Allocation Map (SAM) Blocks.

B.1.1 Directory Name (DN) Blocks

The format of a Directory Name Block is illustrated in Figure B-3. Each file has an entry in one of the three DN blocks on the system device.

DN_1 – Contains entries for internal file numbers 01 through 31_8 (25_{10}) and a link to DN_2.

DN_2 – Contains entries for internal file numbers 32 through 62_8 (50_{10}) and a link to DN_3.

DN_3 – Contains entries for internal file numbers 63 through 77_8 (63_{10}) and an end-of-chain link of 0.

Thus, the system device can contain up to 63 files. Each file entry contains the filename, start address, entry point address, file type, and an internal file number (1 through 77_8). When a file is to be added on the system device, an entry for the file is created in the first open entry slot found in the DN blocks. When a file is deleted, its DN entry is cleared and the slot is made available for some other file.

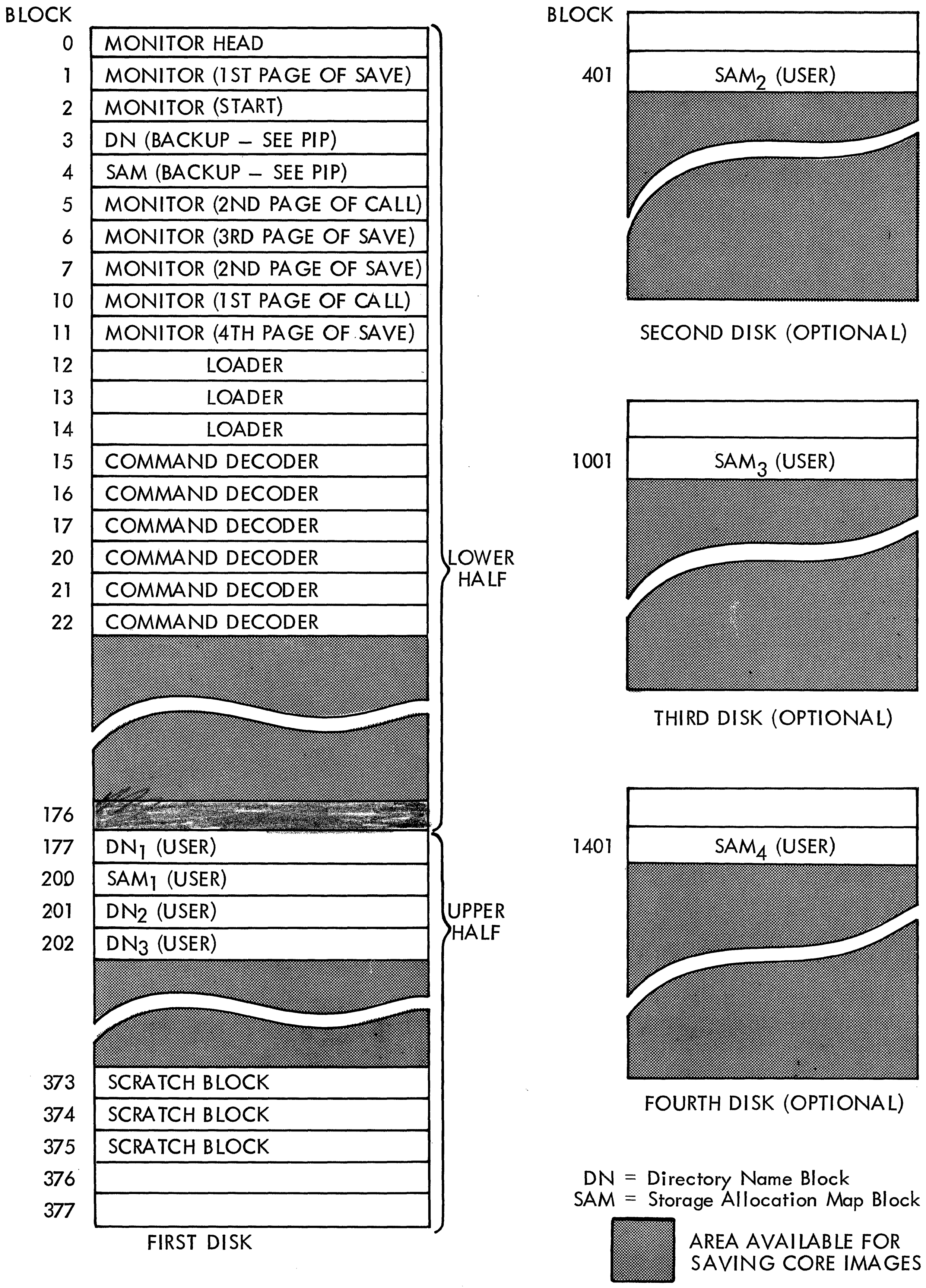


Figure B-1 Disk Storage Layout

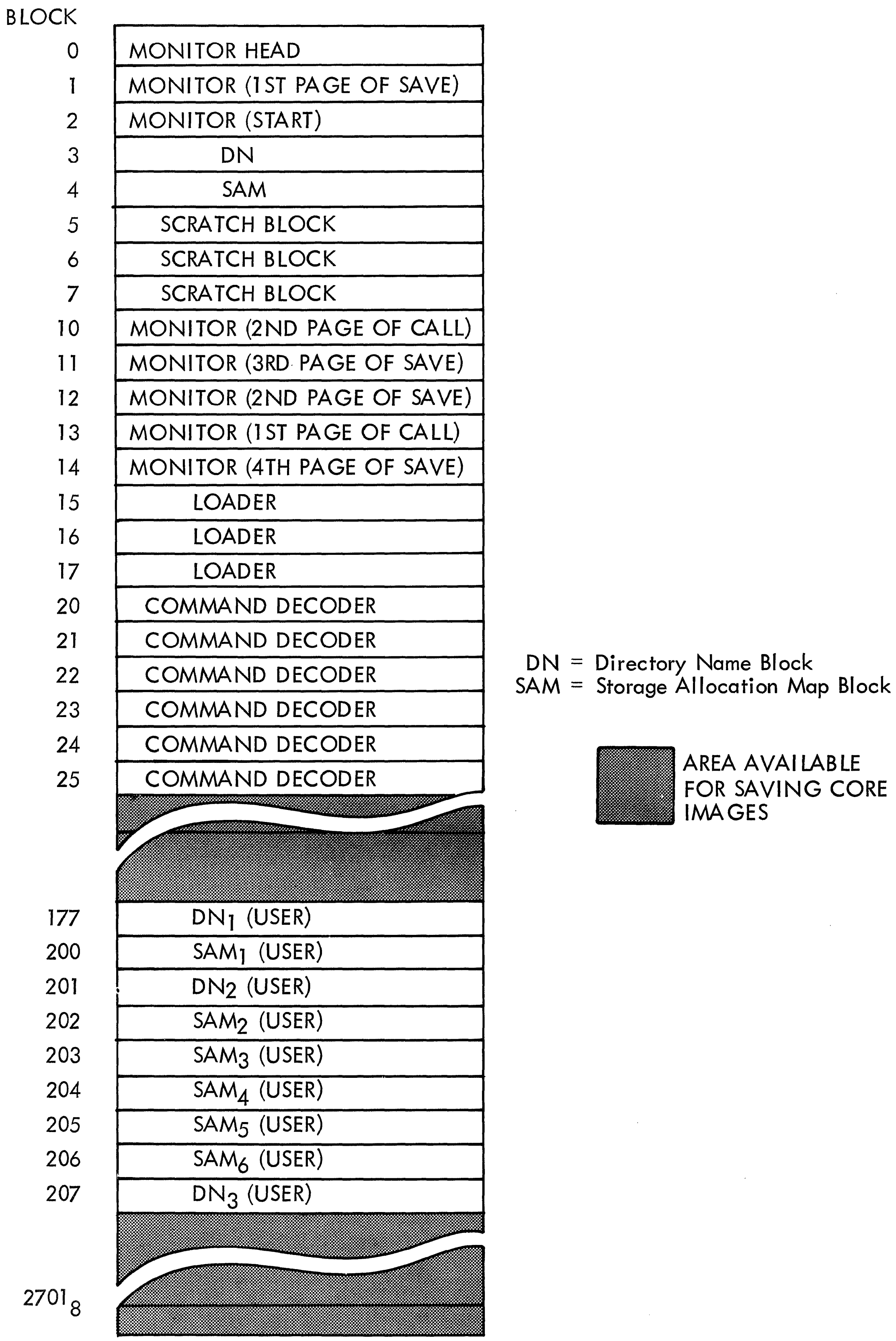
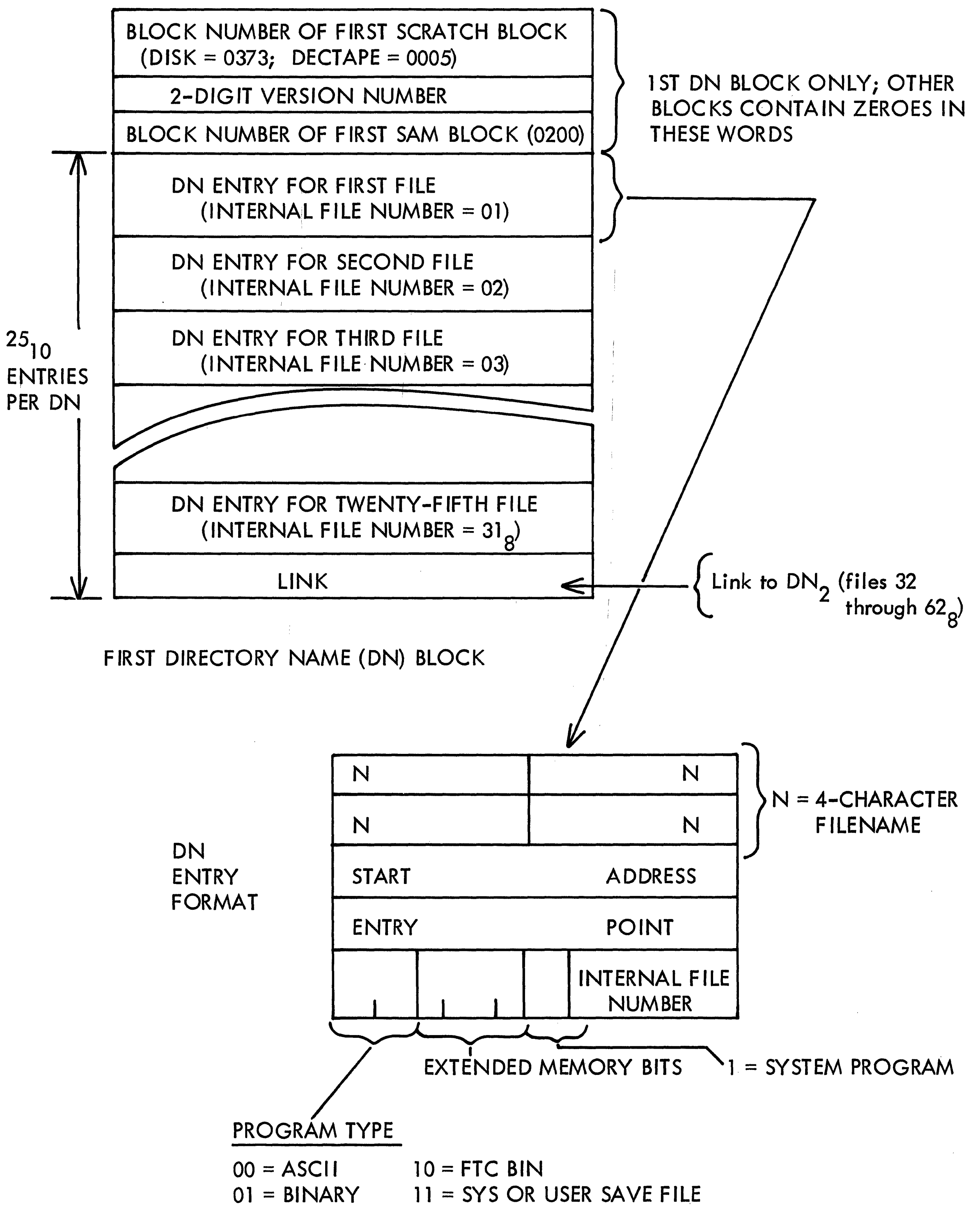


Figure B-2 DECtape Storage Layout



BLOCK NUMBER OF FIRST SCRATCH BLOCK (DISK = 0373; DECTAPE = 0005)
2-DIGIT VERSION NUMBER
BLOCK NUMBER OF FIRST SAM BLOCK (0200)
1ST DN BLOCK ONLY; OTHER BLOCKS CONTAIN ZEROES IN THESE WORDS
DN ENTRY FOR FIRST FILE (INTERNAL FILE NUMBER = 01)
DN ENTRY FOR SECOND FILE (INTERNAL FILE NUMBER = 02)
DN ENTRY FOR THIRD FILE (INTERNAL FILE NUMBER = 03)
DN ENTRY FOR TWENTY-FIFTH FILE (INTERNAL FILE NUMBER = 31_8)
LINK
25_{10} ENTRIES PER DN
Link to DN_2 (files 32 through 62_8)
FIRST DIRECTORY NAME (DN) BLOCK
DN ENTRY FORMAT
N
N
N
N
N = 4-CHARACTER FILENAME
START ADDRESS
ENTRY POINT
INTERNAL FILE NUMBER
EXTENDED MEMORY BITS
1 = SYSTEM PROGRAM
PROGRAM TYPE
00 = ASCII
01 = BINARY
10 = FTC BIN
11 = SYS OR USER SAVE FILE

Figure B-3 Directory Name (DN) Block Format

B.1.2 Storage Allocation Map (SAM) Blocks

SAM blocks contain a record of which files are occupying which blocks on the system device. Each SAM block contains a record of a 377_8-block area. (See Figure B-4.)

SAM_1 contains the map for blocks 0 through 377_8 and a link to SAM_2.

SAM_2 contains the map for blocks 400 through 777_8 and a link to SAM_3.

SAM_3 contains the map for blocks 1000 through 1377_8 and a link to SAM_4.

SAM_4 contains the map for blocks 1400 through 1777_8 and either an end-of-chain link of 0 (if disk) or a link to SAM_5 (if DECtape).

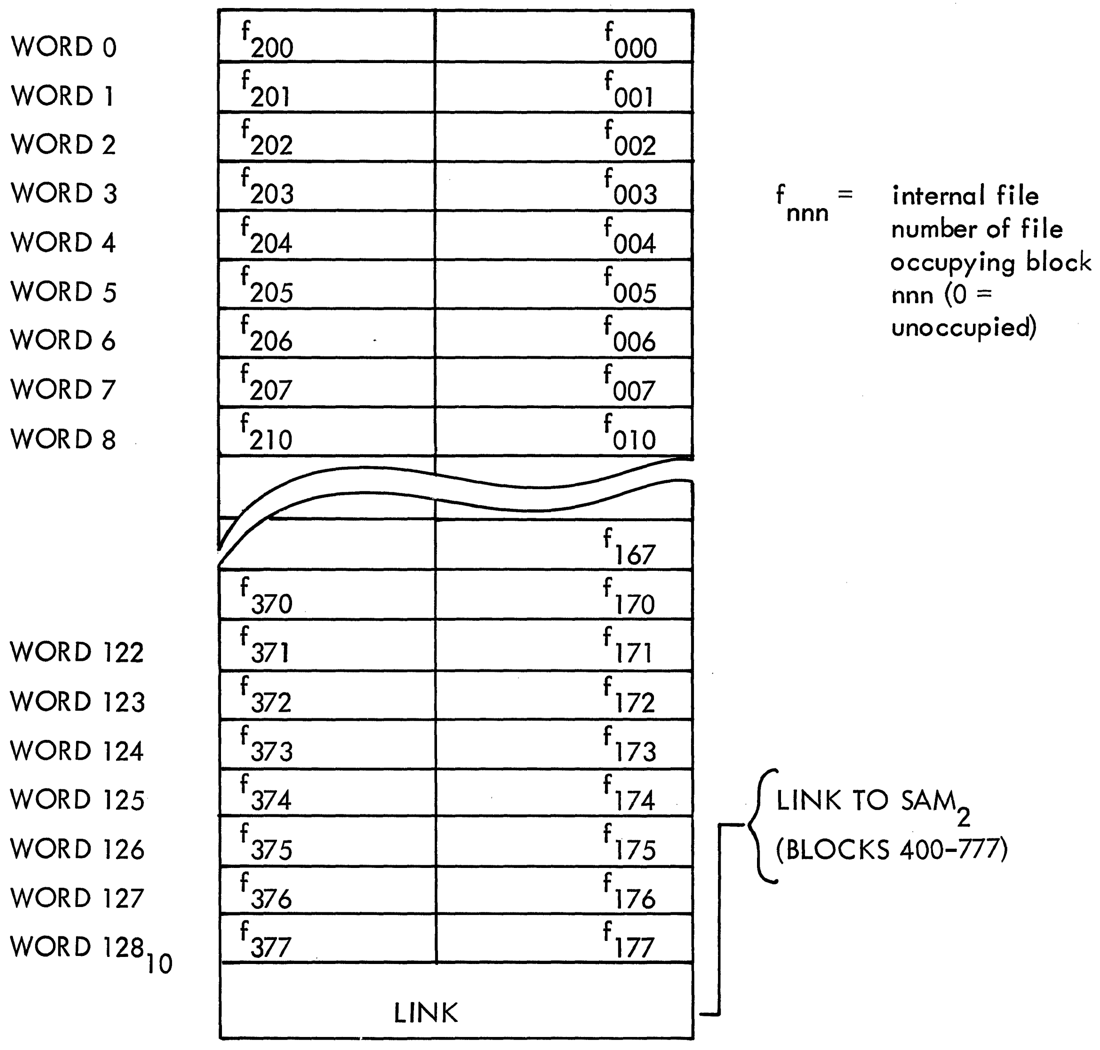
The next two SAM blocks are present only if a DECtape is the system device.

SAM_5 contains the map for blocks 2000 through 2377_8 and a link to SAM_6.

SAM_6 contains the map for blocks 2400 through 2701_8 and an end-of-chain link of 0.

On disk, one SAM block is present for each disk unit (up to four allowed) and each SAM block resides on the disk which it maps (SAM_1 on the first disk, SAM_2 on the second disk, etc.). When a file is to be added, a search is made through the SAM blocks for an entry containing 0 (block is unoccupied), the internal file number of the file is placed in that entry (and in as many other unoccupied entries as are needed for the file), and the storage block linking is adjusted. When a file is deleted, all SAM block entries containing the file's internal file number are set to 0. The block number of the beginning block of the SAM chain (200) is stored in the third word of the first DN block.

SPECIAL INTERNAL FILE NUMBERS: 01 = ALL MONITOR, DN, SAM, AND SCRATCH BLOCKS
04 = LOADER BLOCKS
05 = COMMAND DECODER BLOCKS



WORD 0 f_{200} f_{000}
WORD 1 f_{201} f_{001}
WORD 2 f_{202} f_{002}
WORD 3 f_{203} f_{003}
WORD 4 f_{204} f_{004}
WORD 5 f_{205} f_{005}
WORD 6 f_{206} f_{006}
WORD 7 f_{207} f_{007}
WORD 8 f_{210} f_{010}
f_{167}
f_{370} f_{170}
WORD 122 f_{371} f_{171}
WORD 123 f_{372} f_{172}
WORD 124 f_{373} f_{173}
WORD 125 f_{374} f_{174}
WORD 126 f_{375} f_{175}
WORD 127 f_{376} f_{176}
WORD 128_{10} f_{377} f_{177}
LINK
f_{nnn} = internal file number of file occupying block nnn (0 = unoccupied)
LINK TO SAM_2 (BLOCKS 400-777)

STORAGE ALLOCATION MAP (SAM)

EXAMPLE

FILE #1 - BLOCKS 0, 1, 2
FILE #3 - BLOCKS 5, 6, 11
FILE #4 - BLOCK 10
FILE #13 - BLOCKS 201, 202, 206, 207
FILE #15 - BLOCKS 200, 203, 205, 210
UNUSED - BLOCKS 3, 4, 7, 204, 211

15	01	0
13	01	
13	01	2
15	00	
00	00	4
15	03	
13	03	6
13	00	
15	04	10
00	03	
		12

Figure B-4 Storage Allocation Map (SAM) Block Format

Table B-1
System Device and Core Capacities

Unit	Words	Highest Page (Block) Number (1st Page = 0)
1 DISK	32,768	255_{10} (375_8)
2 DISKS	65,536	511_{10} (773_8)
3 DISKS	98,304	767_{10} (1377_8)
4 DISKS	131,072	1023_{10} (1777_8)
1 DECTAPE	667,520	5215_{10} (2701_8)
4K CORE	4,096	31_{10} (37_8)
8K CORE	8,192	63_{10} (77_8)
12K CORE	12,288	95_{10} (137_8)
16K CORE	16,384	127_{10} (177_8)
20K CORE	20,480	159_{10} (237_8)
24K CORE	24,576	191_{10} (277_8)
28K CORE	28,672	233_{10} (337_8)
32K CORE	32,768	255_{10} (377_8)

B.2 DATA STRUCTURE

The data structure of each type of program file is described in the following paragraphs.

B.2.1 Source File (ASCII) Data Structure

All characters are stored in 6-bit ASCII code, two words per three paper tape frames as described below. All nonprinting characters (200 through 237 and 340 through 377) have their two most significant bits dropped and a 77 prefixed to them. (The one exception to this rule is RUBOUT, 377, which is nonexistent.) All printing characters are trimmed to six bits, except for ? (277), which is packed as 7777.

B.2.2 Binary File (BINARY, FTC BIN) Data Structure

All binary (BINARY) and FORTRAN binary (FTC BIN) files are stored as two words per three paper tape frames. Frame 1 contains the rightmost eight bits of word 1, frame 2 contains the rightmost eight bits of word 2, and frame 3 contains the leftmost four bits of words 1 and 2 (the most significant bits of frame 3 are those of word 2).

Example:

Paper tape	Meaning	Disk(Octal)	Disk (Binary)
200	Leader	5600	1011 10000000
102	ORG	0502	0001 01000010
033	Second half of ORG word		

This procedure is repeated until a trailer code is found.

B.2.3 Saved File (SYS, USER) Data Structure

Saved files are stored on the system device as an integral number of pages and each page occupies one disk or DECtape block. Storage conventions differ between saved files of contiguous pages of core and those of noncontiguous pages.

Contiguous Pages

All system device blocks contain core images (Figure B-5). The Start Address word in the Directory Name (DN) entry for the file is set to the starting page address.

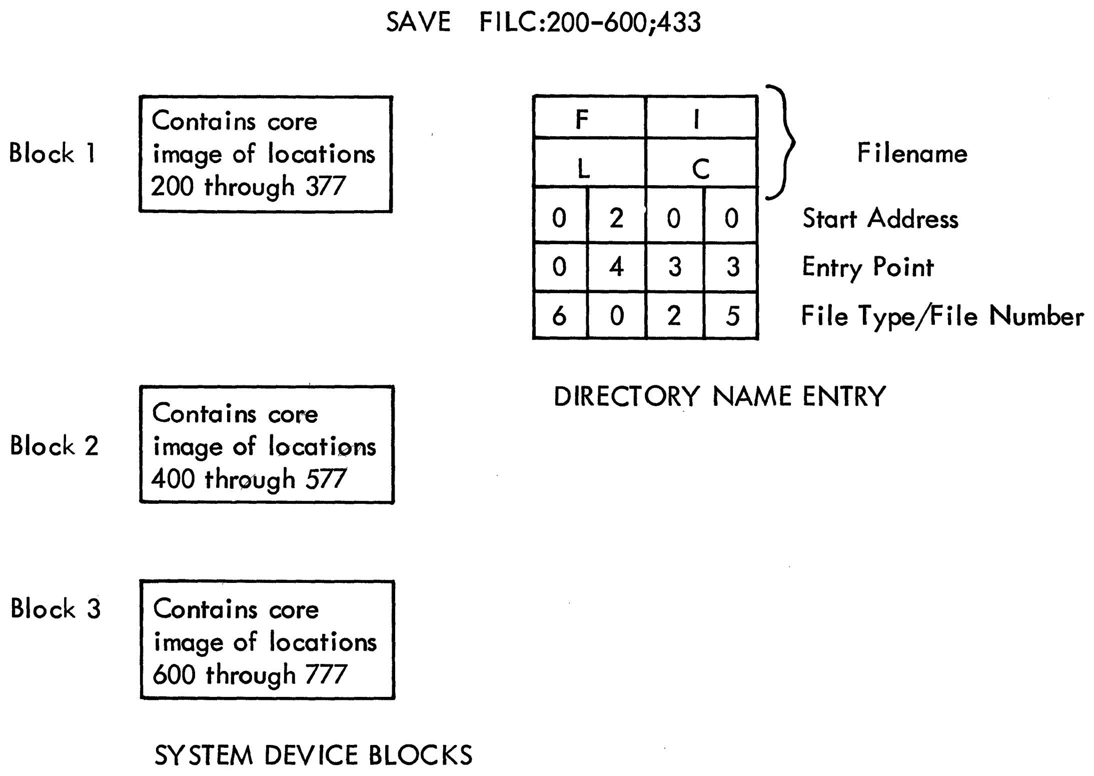

SAVE FILC:200-600;433
Block 1
Contains core image of locations 200 through 377
Block 2
Contains core image of locations 400 through 577
Block 3
Contains core image of locations 600 through 777
SYSTEM DEVICE BLOCKS
F I
L C
Filename
0 2 0 0 Start Address
0 4 3 3 Entry Point
6 0 2 5 File Type/File Number
DIRECTORY NAME ENTRY

Figure B-5 Contiguous-Page Save File Format

Noncontiguous Pages

The first system device block of a saved file composed of noncontiguous pages of core contains a list of core page assignments and the core images sotred in subsequent blocks. The last entry in this list is set to 7777 (Figure B-6). The Start Address word in the Directory Name entry for the file is set to 7777 to indicate that the first block does not contain a core image but a page assignment listing.

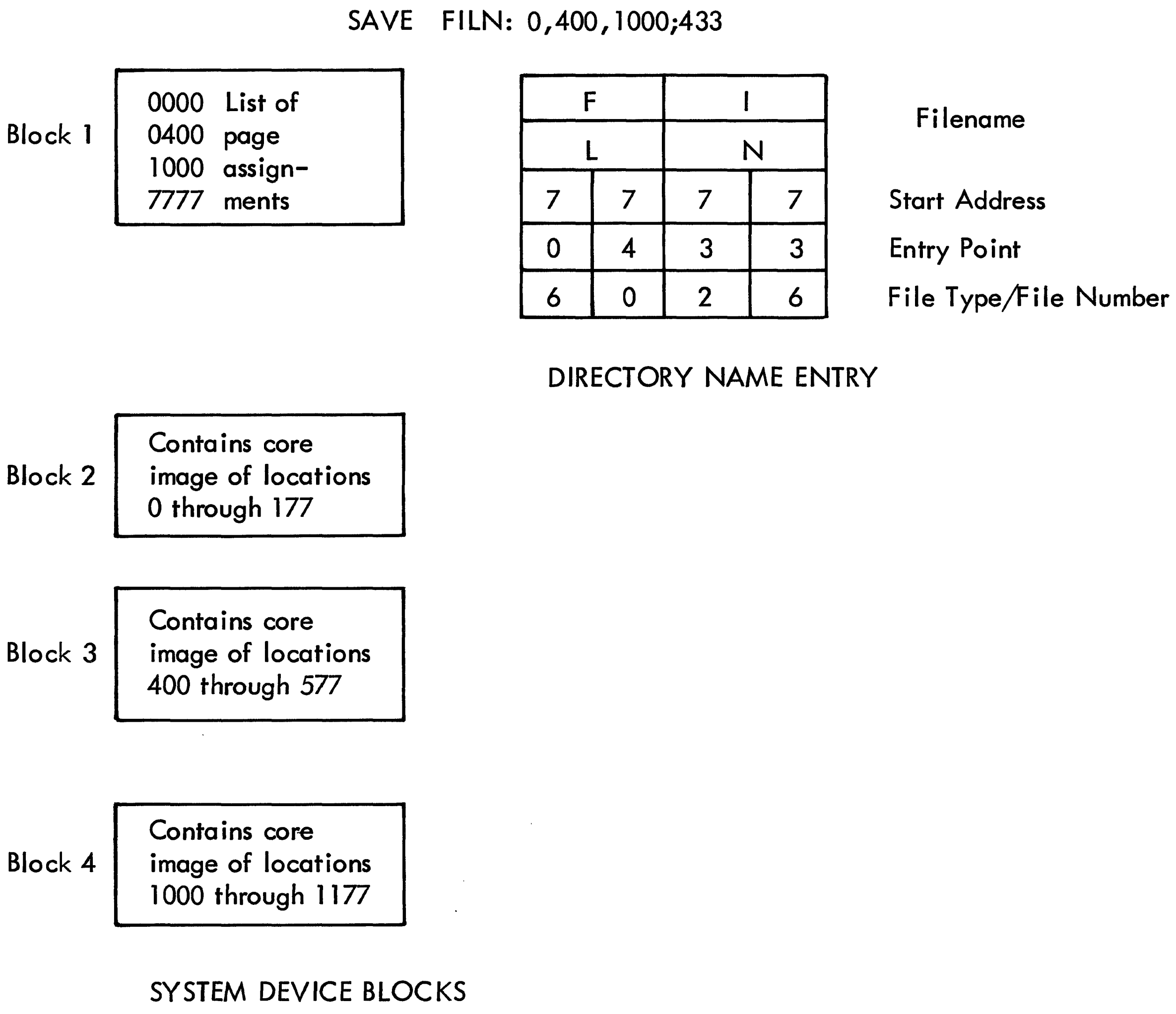


Figure B-6 Noncontiguous-Page Save File Format

B.3 PIP DIRECTORY LISTING

A directory listing of the system device can be obtained by running PIP (Figure B-7). A sample output is given below.

```
.PIP
*OPT-L
*IN-S:
FB=0110                        110 free blocks remain



8D
PALD.SYS  (0)  0030
EDIT.SYS  (0)  0015
LOAD.SYS  (0)  0003
.CD..SYS  (0)  0006
PIP .SYS  (0)  0020
FORT.SYS  (0)  0010
.FT..SYS  (0)  0035
.OS..SYS  (0)  0024
FOSL.SYS  (0)  0006
STBL.SYS  (0)  0001
.DDT.USER (0)  0022
.USR.USER (0)  0023
DDT .SYS  (0)  0002
```

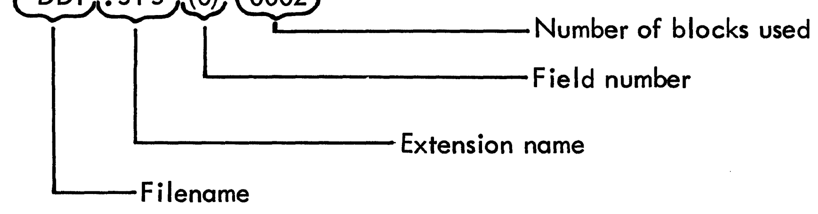


Figure B-7 Sample PIP Directory Listing

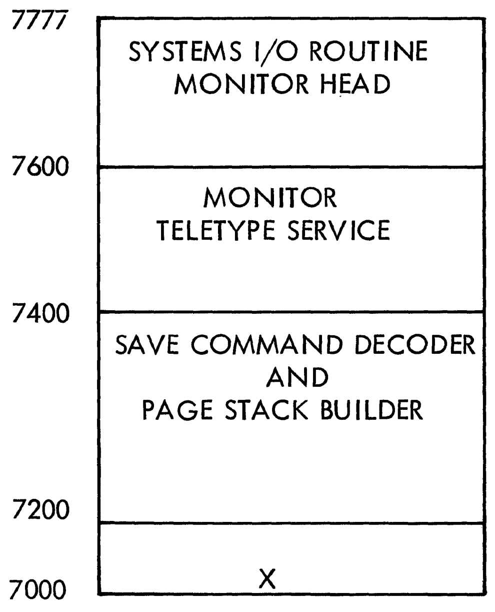
B.4 MONITOR CORE USAGE DIAGRAMS

The following illustrations show Monitor usage of locations 7000 through 7777 at

a. Monitor Time and User Time (Figure B-8)

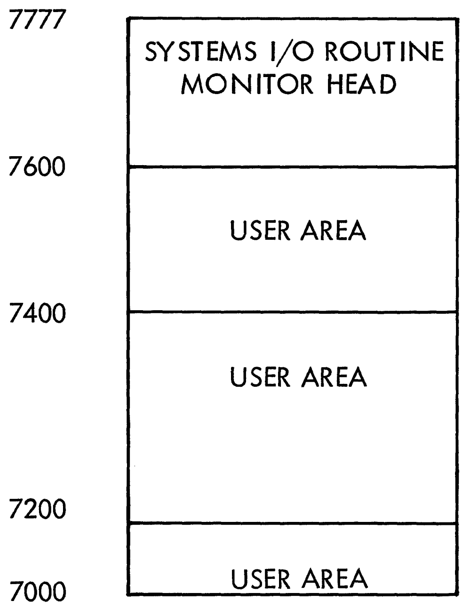
b. SAVE Command Processing (Figure B-9)

c. CALL Command Processing (Figure B-10)



7777
SYSTEMS I/O ROUTINE MONITOR HEAD
7600
MONITOR TELETYPE SERVICE
7400
SAVE COMMAND DECODER AND PAGE STACK BUILDER
7200
X
7000

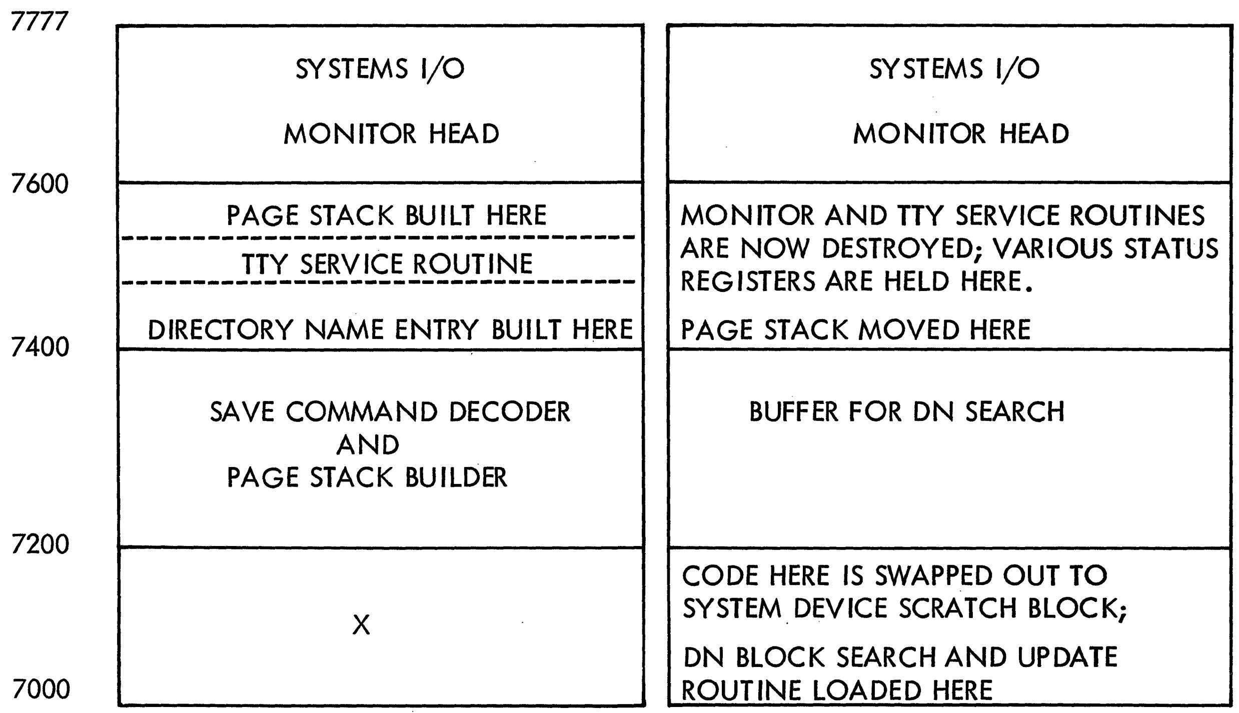
(a) Monitor-Time Core Usage



(b) User-Time Core Usage

Figure B-8 Monitor-Time vs User-Time Core Usage

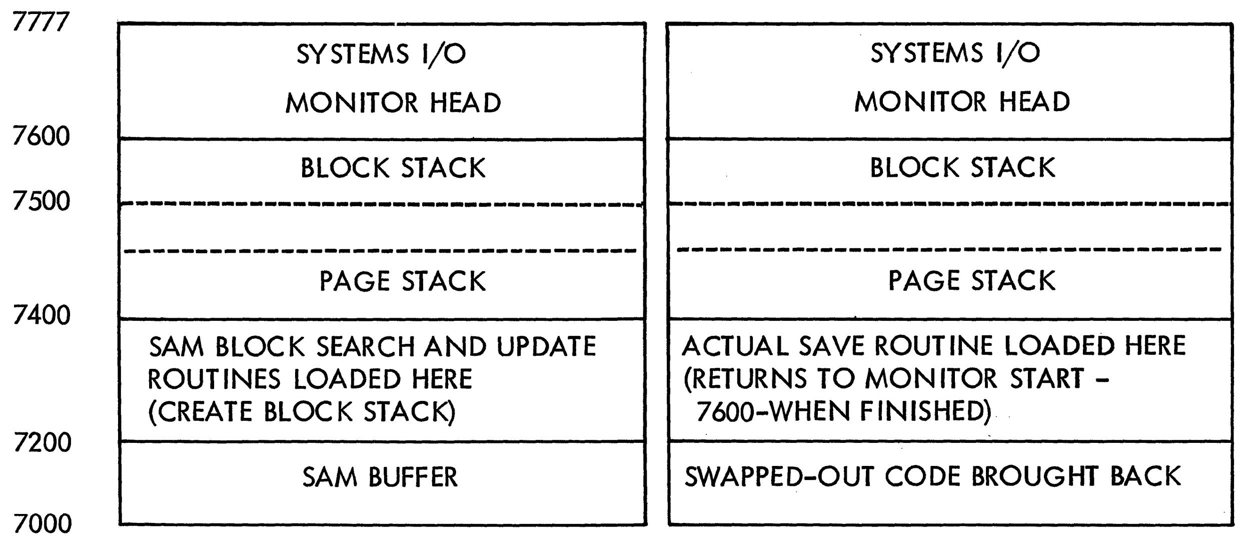
.SAVE filename:core-specifications, . . . ;entry-point



7777
7600
7400
7200
7000
SYSTEMS I/O
MONITOR HEAD
PAGE STACK BUILT HERE
TTY SERVICE ROUTINE
DIRECTORY NAME ENTRY BUILT HERE
SAVE COMMAND DECODER AND PAGE STACK BUILDER
X
SYSTEMS I/O
MONITOR HEAD
MONITOR AND TTY SERVICE ROUTINES ARE NOW DESTROYED; VARIOUS STATUS REGISTERS ARE HELD HERE.
PAGE STACK MOVED HERE
BUFFER FOR DN SEARCH
CODE HERE IS SWAPPED OUT TO SYSTEM DEVICE SCRATCH BLOCK;
DN BLOCK SEARCH AND UPDATE ROUTINE LOADED HERE

(a) "SAVE filename:" Processing

(b) "Core-specifications, . . ;entry-point" Processing



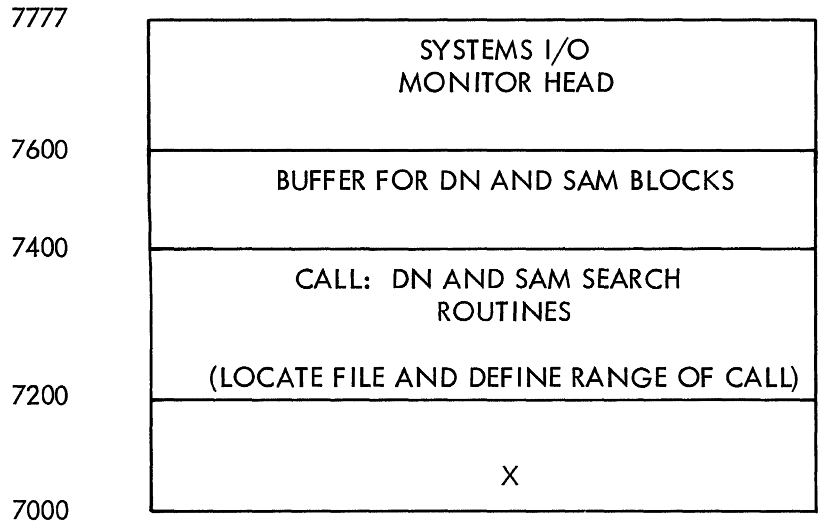
7777
7600
7500
7400
7200
7000
SYSTEMS I/O
MONITOR HEAD
BLOCK STACK
PAGE STACK
SAM BLOCK SEARCH AND UPDATE ROUTINES LOADED HERE (CREATE BLOCK STACK)
SAM BUFFER
SYSTEMS I/O
MONITOR HEAD
BLOCK STACK
PAGE STACK
ACTUAL SAVE ROUTINE LOADED HERE (RETURNS TO MONITOR START - 7600-WHEN FINISHED)
SWAPPED-OUT CODE BROUGHT BACK

(c) SAM Search

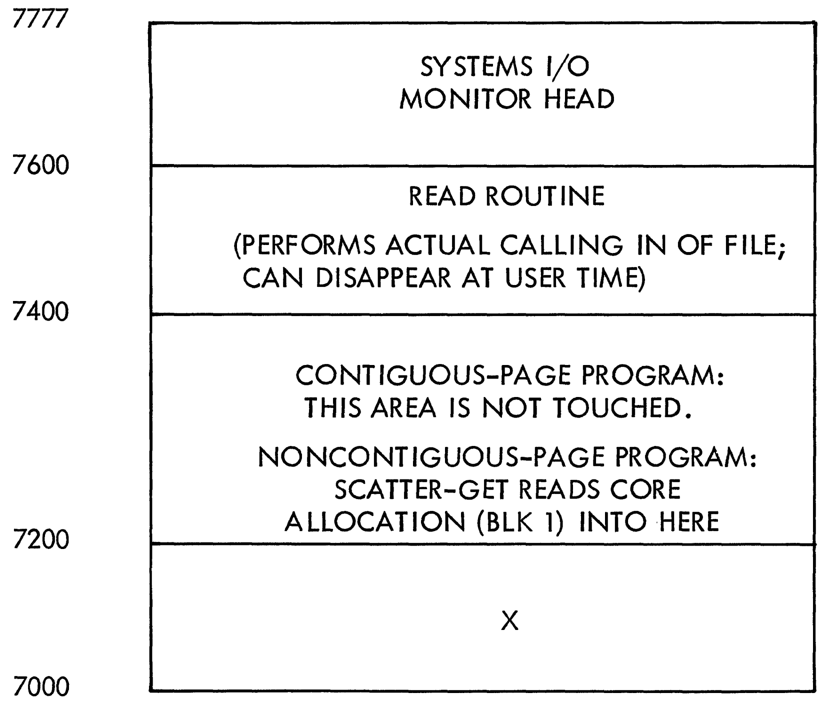
(d) Actual Save Time

Figure B-9 Core Usage During SAVE Command Execution

.CALL filename↲ or .filename↲


7777
SYSTEMS I/O MONITOR HEAD
7600
BUFFER FOR DN AND SAM BLOCKS
7400
CALL: DN AND SAM SEARCH ROUTINES
(LOCATE FILE AND DEFINE RANGE OF CALL)
7200
X
7000

(a) "CALL filename" Processing


7777
SYSTEMS I/O MONITOR HEAD
7600
READ ROUTINE
(PERFORMS ACTUAL CALLING IN OF FILE; CAN DISAPPEAR AT USER TIME)
7400
CONTIGUOUS-PAGE PROGRAM: THIS AREA IS NOT TOUCHED.
NONCONTIGUOUS-PAGE PROGRAM: SCATTER-GET READS CORE ALLOCATION (BLK 1) INTO HERE
7200
X
7000

(b) Actual CALL Time

Figure B-10 Core Usage During CALL Command Execution

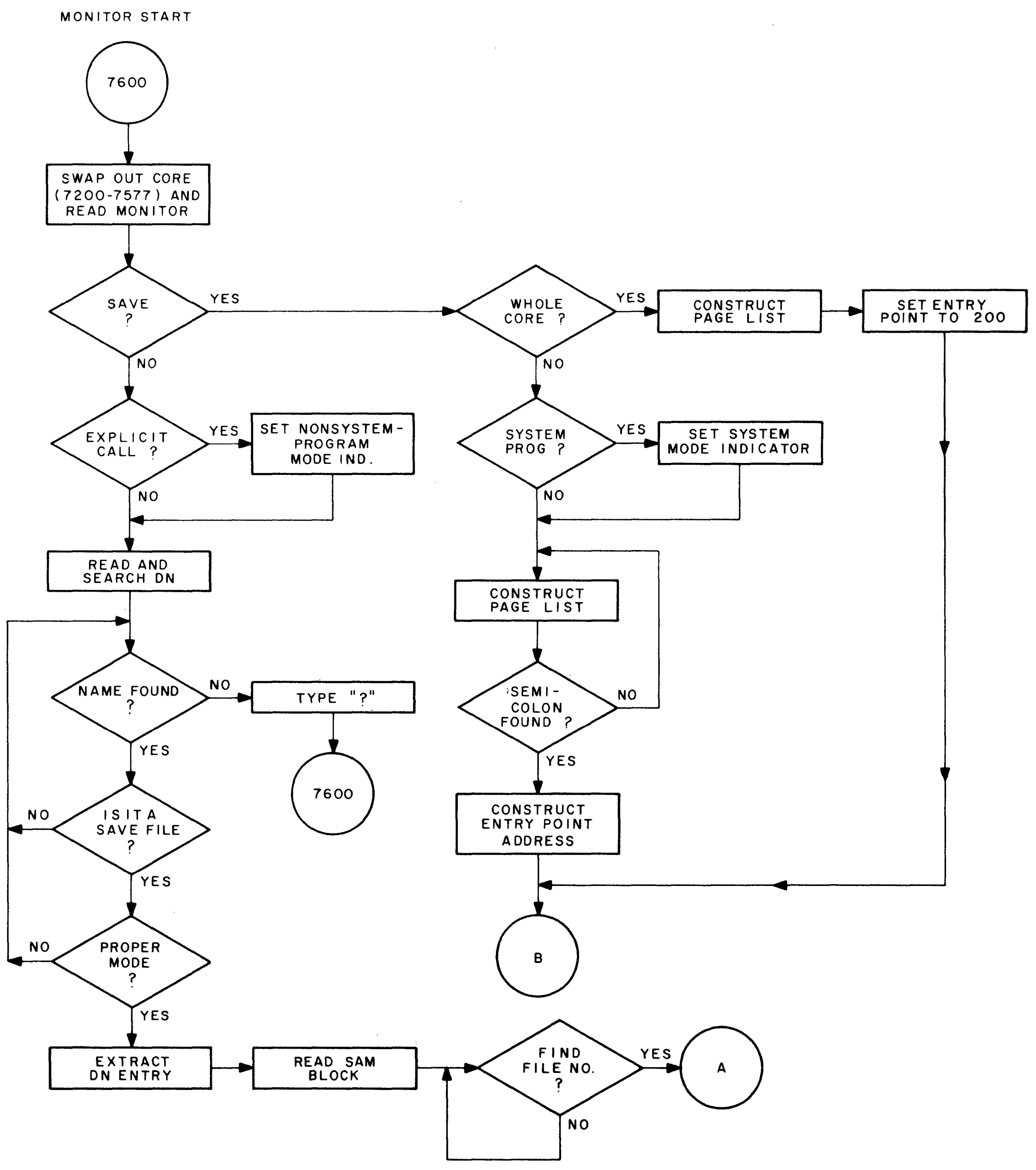


Figure B-11 Monitor Flow Chart (Part 1)

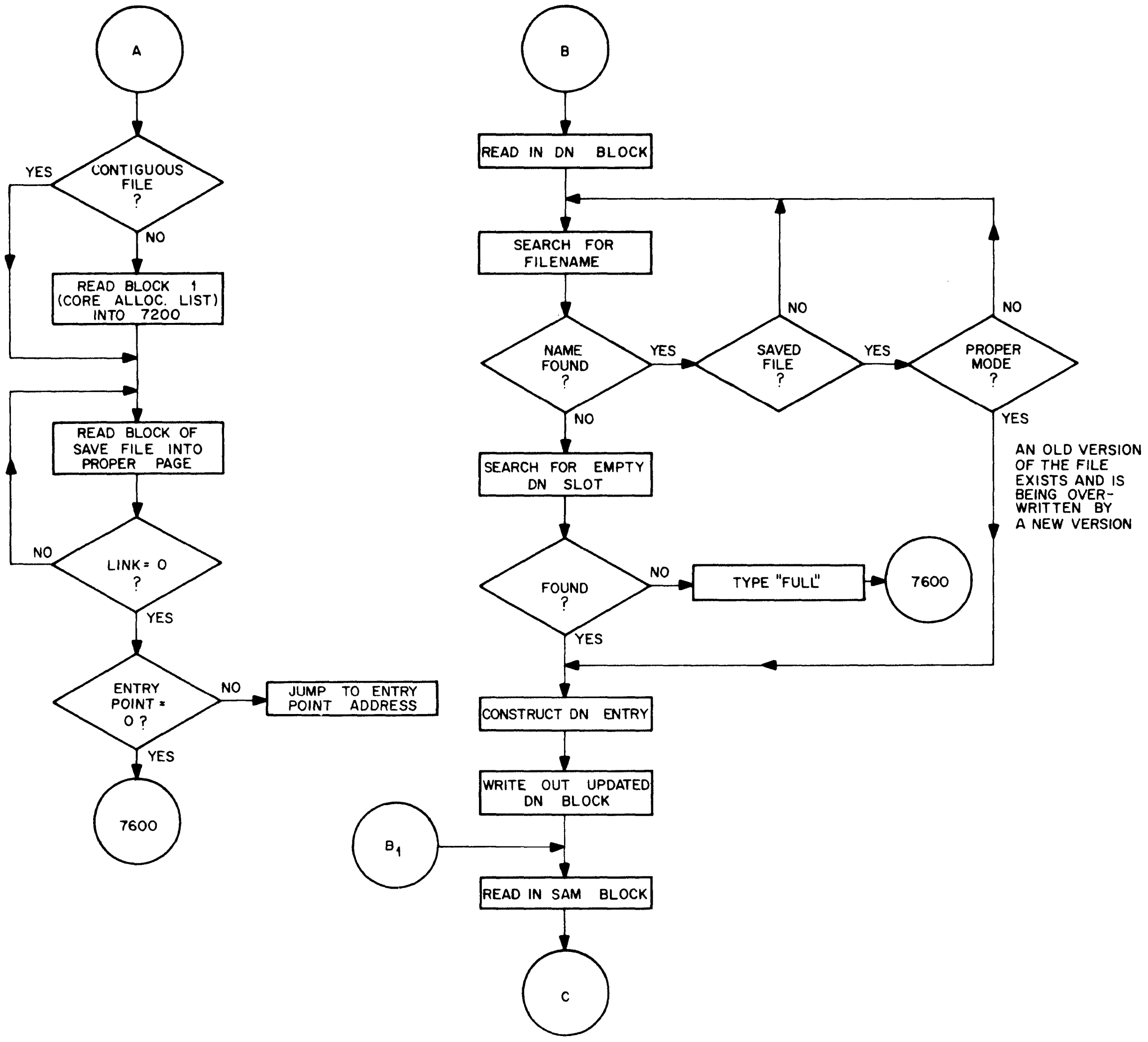


Figure B-11 Monitor Flow Chart (Part 2)

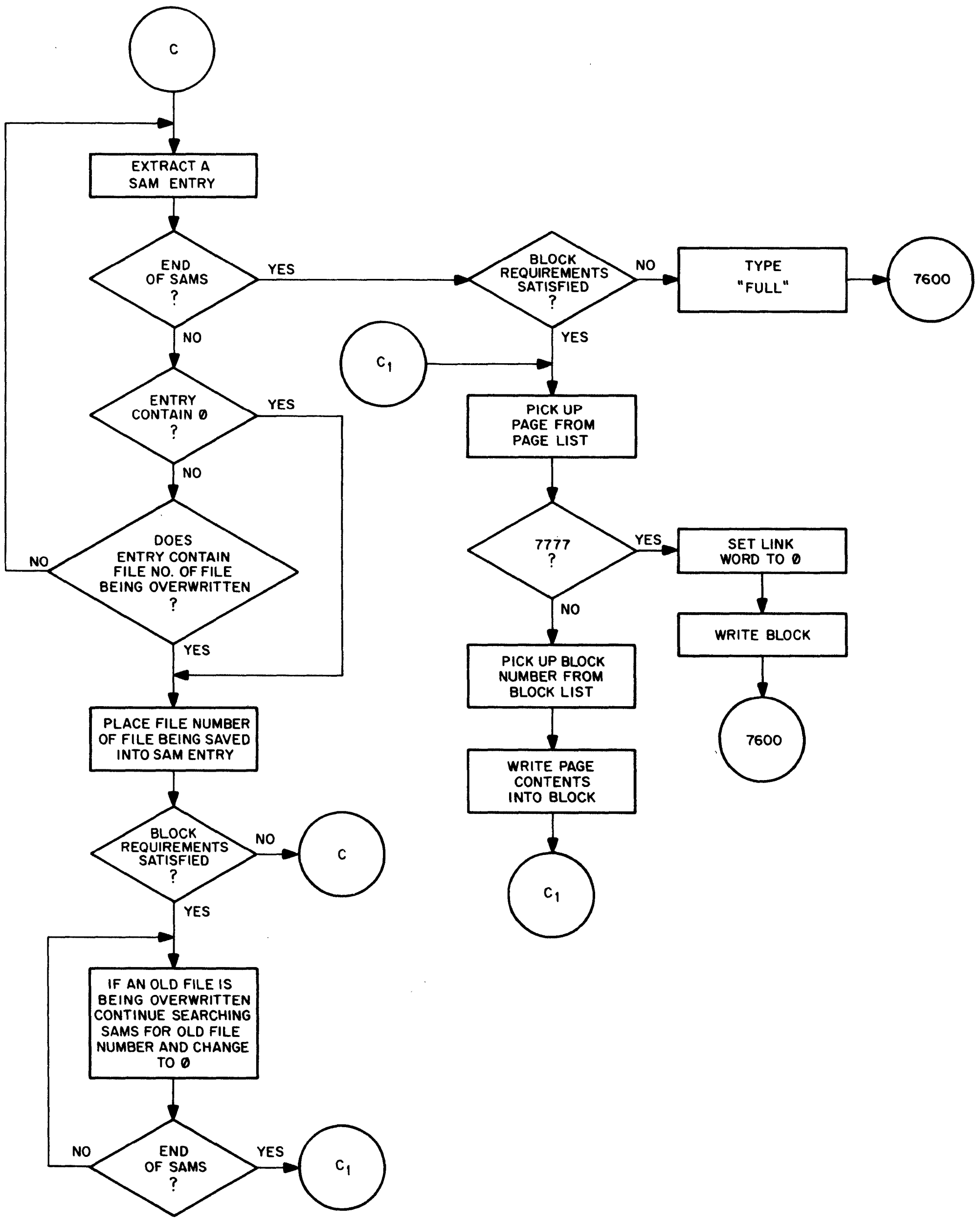

C
EXTRACT A SAM ENTRY
END OF SAMS ?
YES
NO
ENTRY CONTAIN Ø ?
YES
NO
DOES ENTRY CONTAIN FILE NO. OF FILE BEING OVERWRITTEN ?
NO
YES
PLACE FILE NUMBER OF FILE BEING SAVED INTO SAM ENTRY
BLOCK REQUIREMENTS SATISFIED ?
NO
C
YES
IF AN OLD FILE IS BEING OVERWRITTEN CONTINUE SEARCHING SAMS FOR OLD FILE NUMBER AND CHANGE TO Ø
END OF SAMS ?
NO
YES
C1
BLOCK REQUIREMENTS SATISFIED ?
NO
TYPE "FULL"
7600
YES
C1
PICK UP PAGE FROM PAGE LIST
7777 ?
YES
SET LINK WORD TO Ø
WRITE BLOCK
7600
NO
PICK UP BLOCK NUMBER FROM BLOCK LIST
WRITE PAGE CONTENTS INTO BLOCK
C1

Figure B-11 Monitor Flow Chart (Part 3)

APPENDIX C
COMMAND DECODER

Command Decoder is a general-purpose program used by all system programs to read in and interpret command strings entered by the user via his Teletype keyboard. Command Decoder is generated and stored on the system device by System Builder.

Command Decoder uses four pages of core (see Figure C-2) and is called in by a system program in the following way.

a. The internal file number of Command Decoder (filename = .CD.) is obtained.

b. The starting block of the Command Decoder file is obtained.

c. This block is then read into the second of the four pages to be used by Command Decoder. Command Decoder is position-independent and can be read into any four contiguous pages of core between locations 200 and 7577 inclusive.

d. Command Decoder is then entered by jumping to the second location of page 2 (the first location is an error return).

C.1 LOCATIONS USED BY COMMAND DECODER

Locations 167 through 177, page 0, are used as follows.

Table C-1
Page 0 Locations Used by Command Decoder

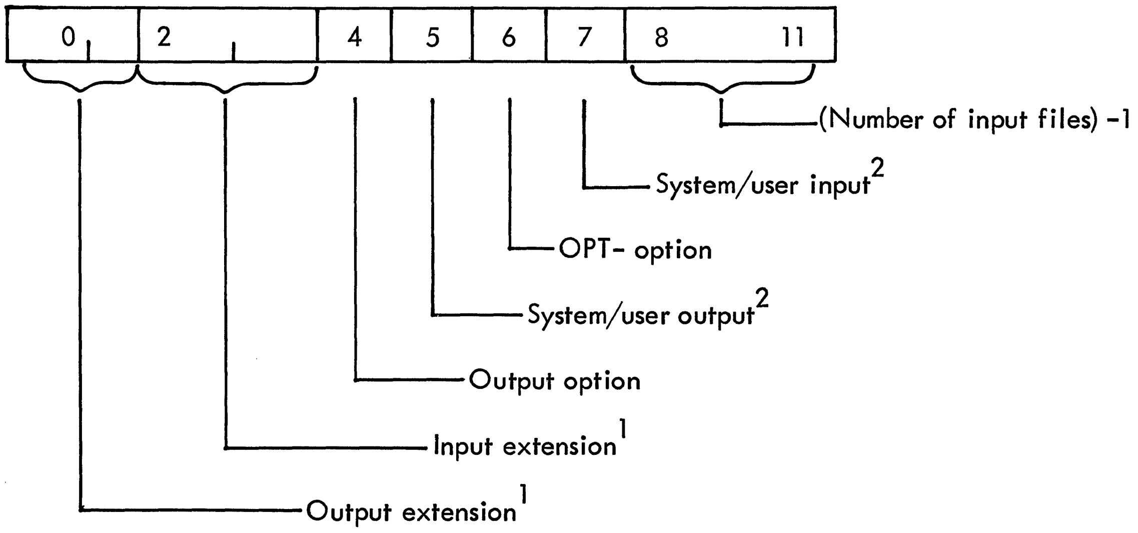
Location	Purpose
167	Preloaded with 7777 if input and output filenames and extension names are different.
170	Scratch location.
171	Scratch location.
172	Points to the first block of Command Decoder.
173	Scratch location.
174	Points to the output list. Information concerning each device request is placed in this list by Command Decoder.
175	Contains the option bits. This location is not left in its original state upon exit from Command Decoder
176	Scratch location.
177	Contains the address of the return from Command Decoder.

C.2 INPUT AND OUTPUT REQUIREMENTS FOR COMMAND DECODER

Location 174 (CDPTRP), the output list pointer, must point to a block of code, the length of which must be 3*n+1, where n is the total number of device requests expected. For example, a program with one output file plus three input files requires 13 locations. (See Figure C-1.)

The option bit location (175) is constructed as follows.

Bits 0 and 1	Contain output file extension code (or input, if no output is requested).[1]
Bits 2 and 3	Contain the input file extension code.[1]
Bit 4	1 = Output file is expected (Command Decoder will type *OUT- query (in addition to *IN-)).
Bit 5	1 = Saved output file is a system program (bit 5 of word 4 in DN entry is set to 1).
Bit 6	1 = Option is available (Command Decoder will type *OPT-).
Bit 7	1 = Saved input file is a system program (bit 5 of word 4 in DN entry is checked for a 1).
Bits 8-11	(Total number of input files allowed) -1.



This option word must be set up by the system program before calling Command Decoder.

[1]Extension codes:
00 = ASCII
01 = BINARY
10 = FTC BIN
11 = Saved file (USER, SYS)

[2]1 = System, 0 = User

The first block of the Command Decoder is read into the second of the four blocks into which it is to run. In the following examples, assume Command Decoder is to be run in locations 2000-2777; that you have already loaded FBLK with the first block number of the Command Decoder; output list is in 3000; return is at 203 and you are looking for user file output, system file input, no *OPT- is desired, and three input files are allowed.

Example 1:

```
              *1700
        TAD   (203
        DCA   177              /RETURN
        TAD   (7622            /111 110 010 010 = BITS
        DCA   175
        TAD   (3000
        DCA   174              /POINTER TO LIST
        TAD   FBLK
        DCA   172              /BLOCK 1 OF .CD. (DISK)
        CMA
        DCA   167
        JMS I (7642
        3                      /READ
FBLK,   0                      /BLOCK 1 OF COMMAND DECODER
        2200                   /INTO LOCATION 2200
        0                      /LINK
        HLT                    /BAD READ
        JMP I .+1
        2201                   /ENTER .CD.
```

Example 2:

```
        *2200
        TAD   (203
        DCA   177              /RETURN
        TAD   (7622            /111 110 010 010 = BITS
        DCA   175
        TAD   (3000
        DCA   174              /POINTER TO LIST
        TAD   FBLK
        DCA   172              /BLOCK 1 OF .CD.
        CMA
        DCA   167
        JMS I (7642
        1003                   /READ AND RETURN THRU
FBLK,   0                      /ADDRESS IN ERROR RETURN
        2200                   /IF ERROR, OR ERROR RETURN
        0                      /+1 IF CORRECT RETURN
        2200                   /NOTE THIS CODE IS OVER-WRITTEN
```

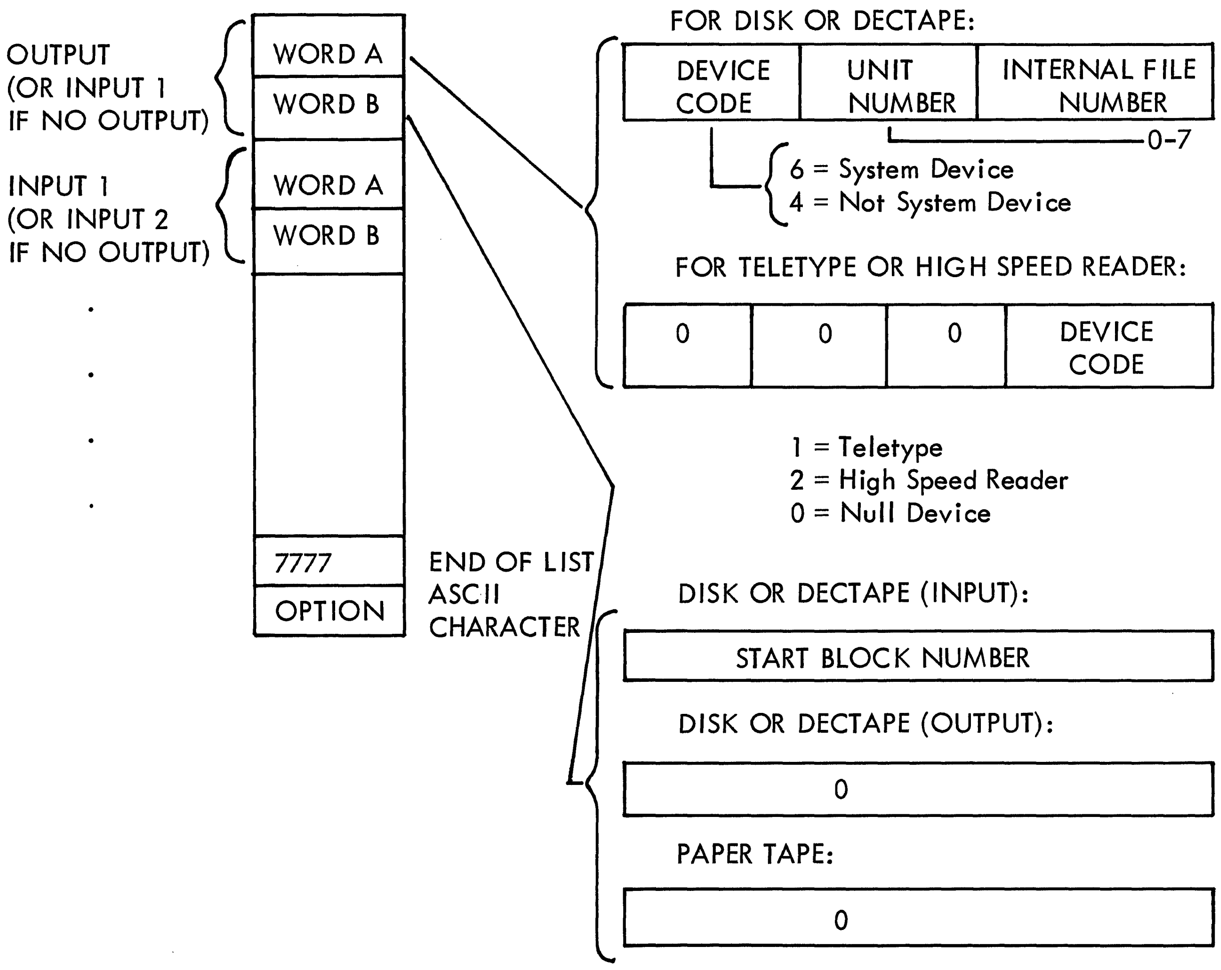


Figure C-1 Output List Produced by Command Decoder

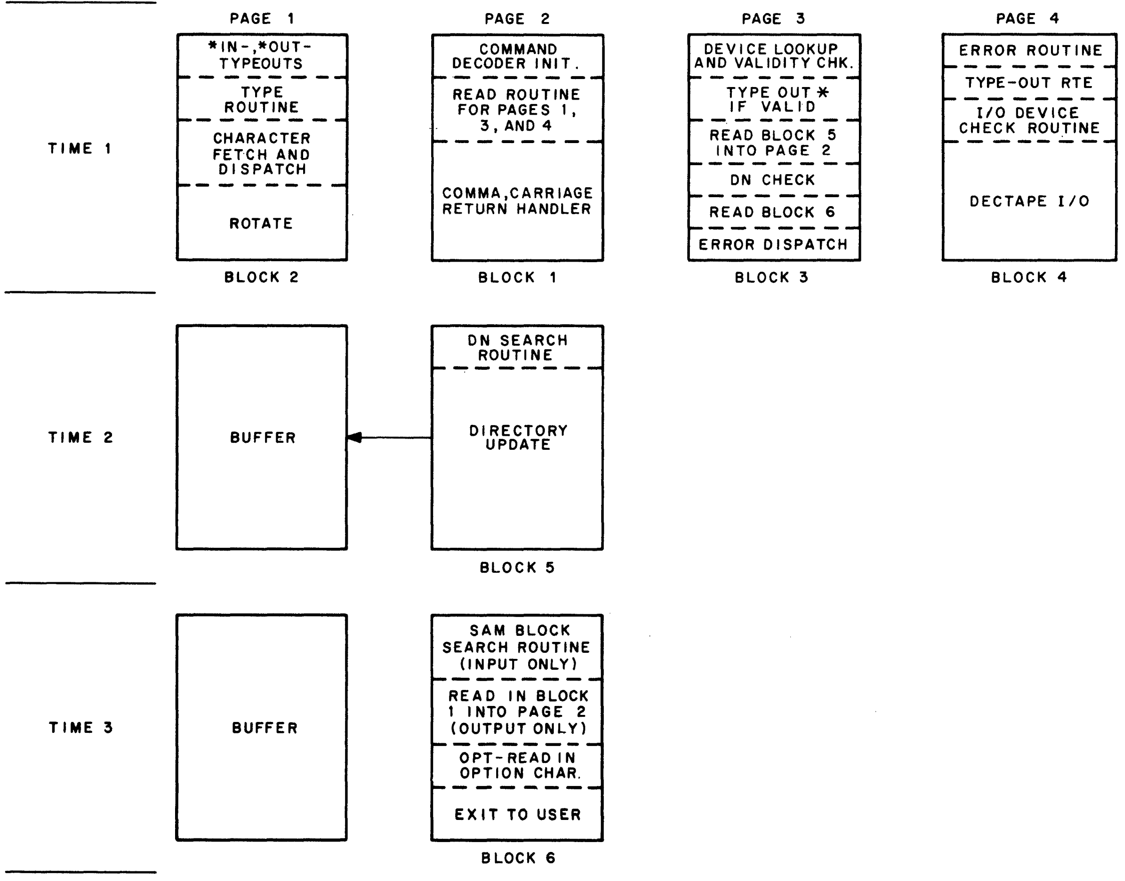


Figure C-2 Command Decoder Core Usage

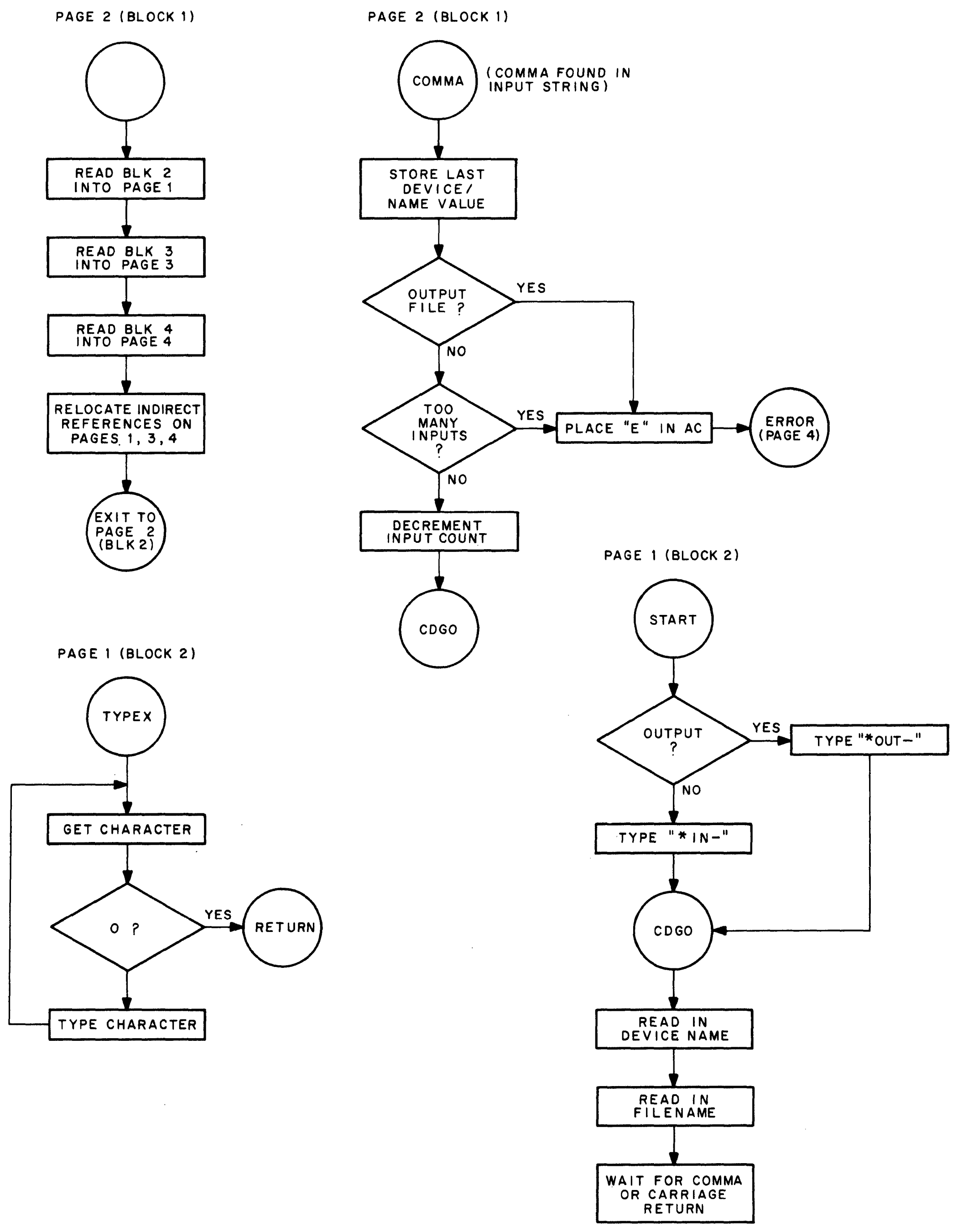


Figure C-3 Command Decoder Flow Chart (Part 1)

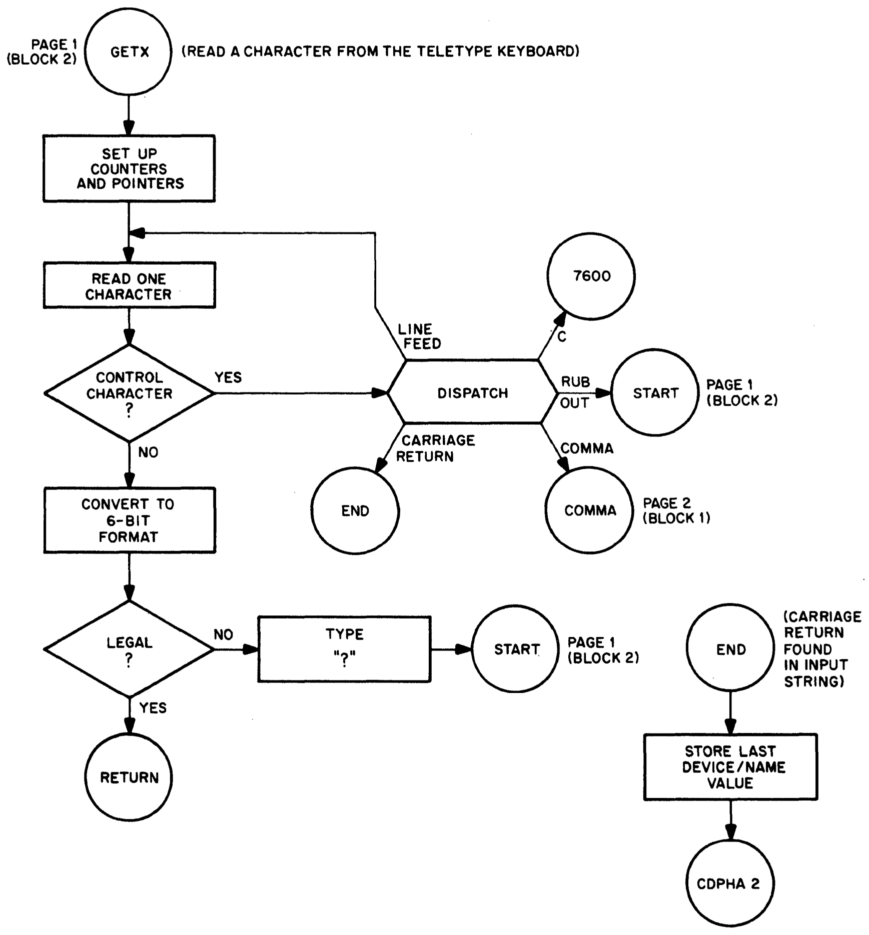


Figure C-3 Command Decoder Flow Chart (Part 2)

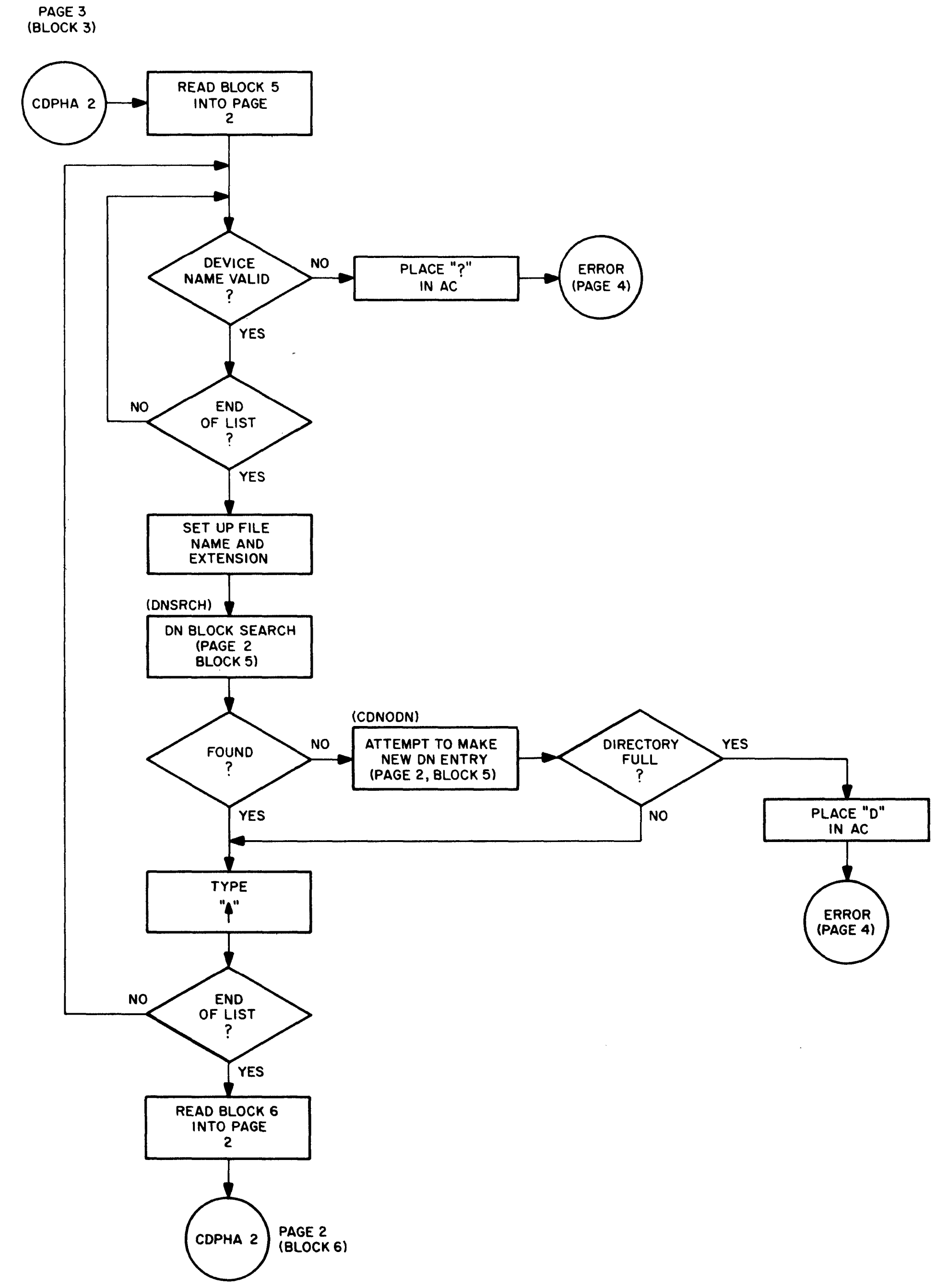


Figure C-3 Command Decoder Flow Chart (Part 3)

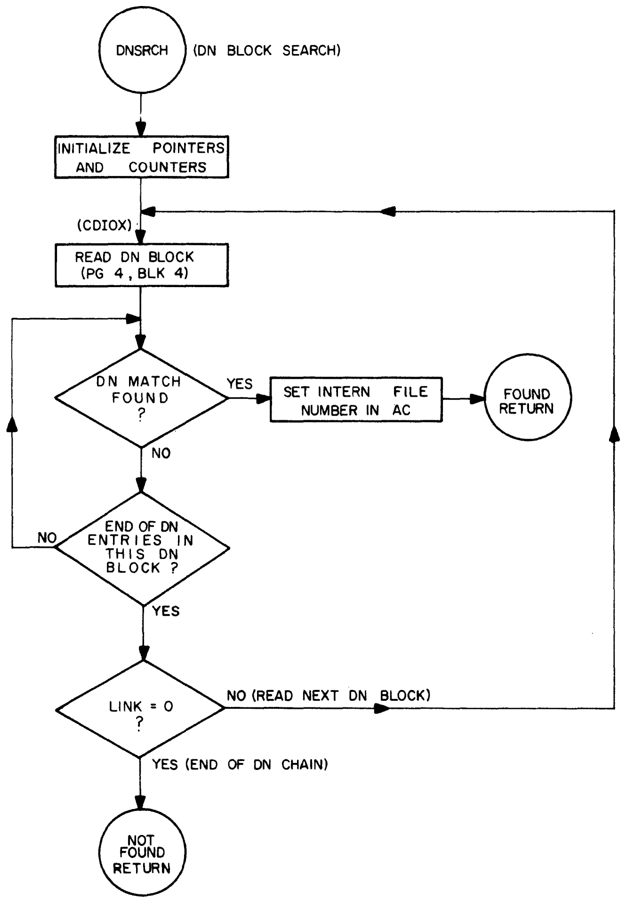


Figure C-3 Command Decoder Flow Chart (Part 4)

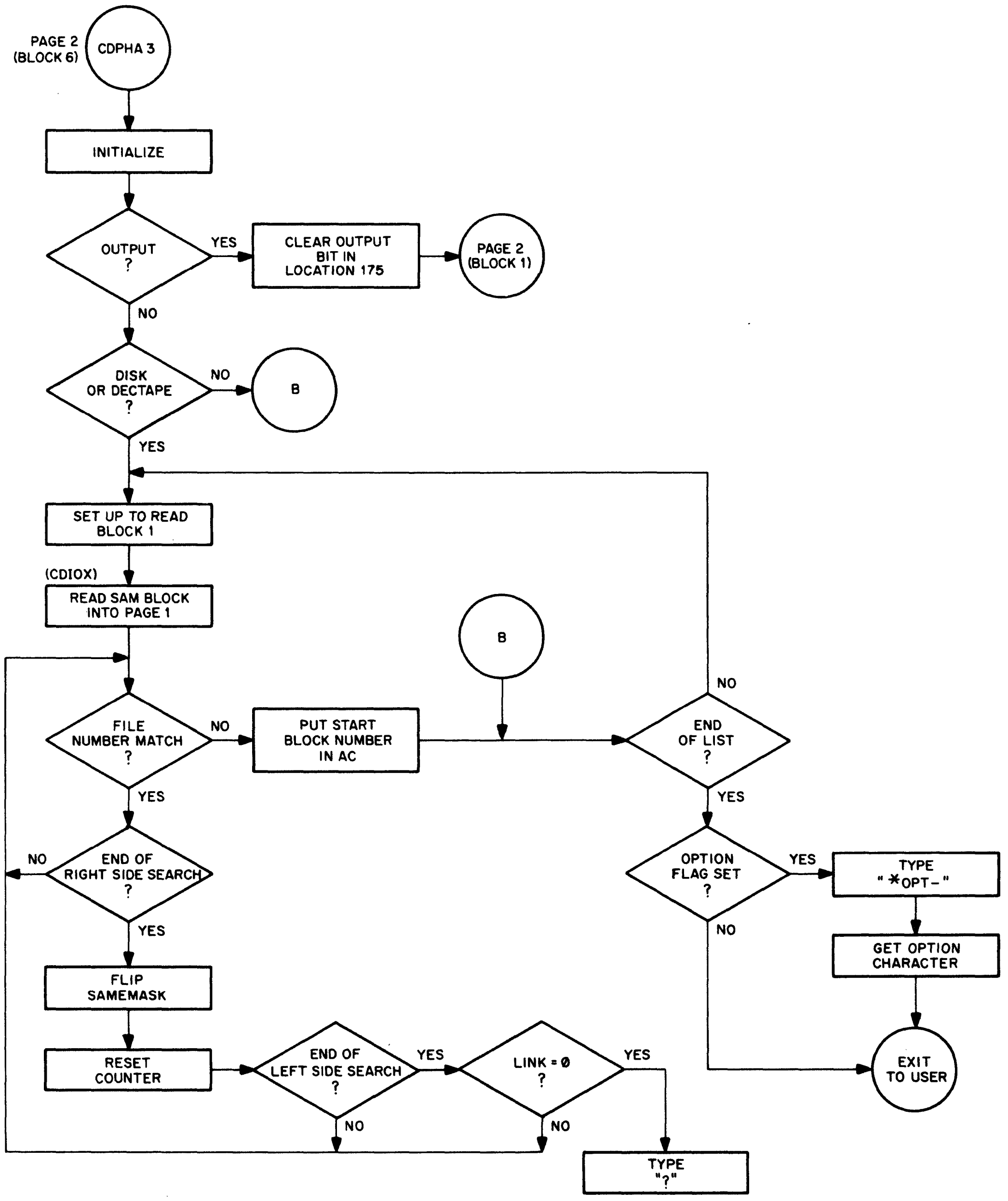


Figure C-3 Command Decoder Flow Chart (Part 5)

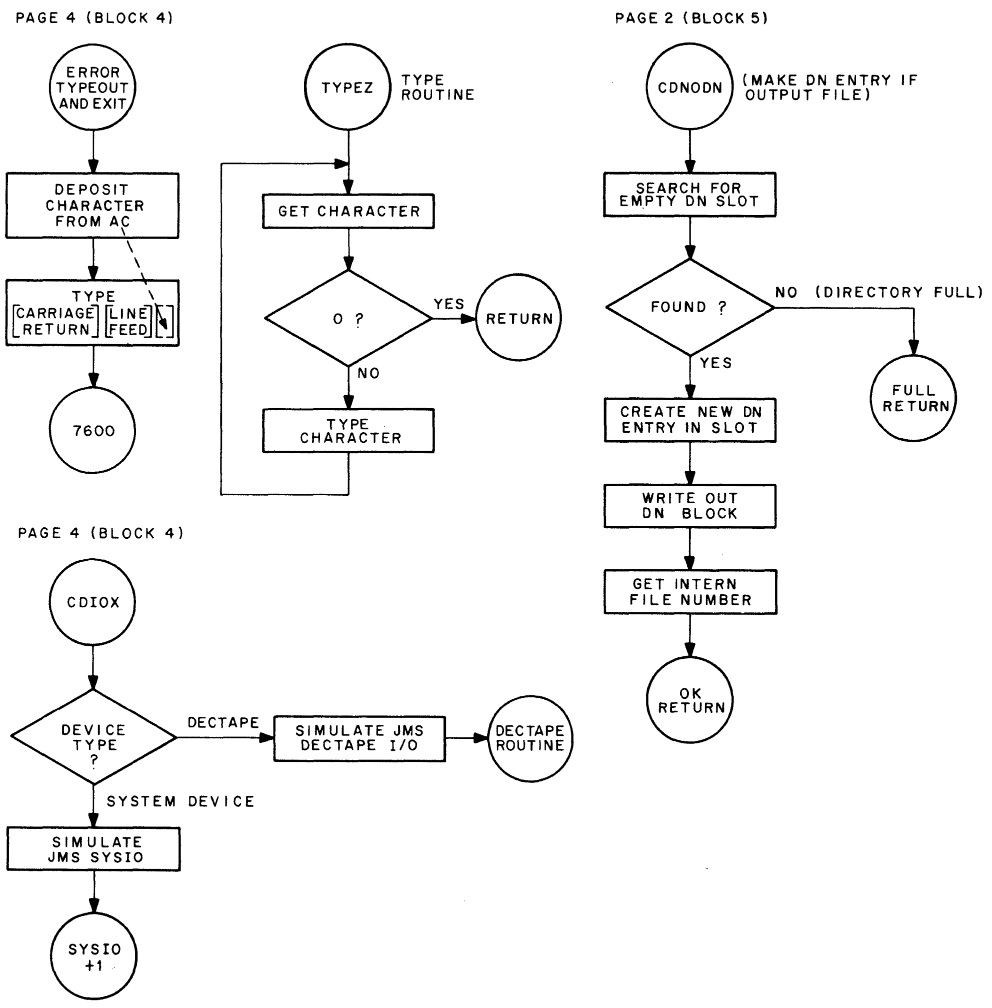


Figure C-3 Command Decoder Flow Chart (Part 6)

APPENDIX D
BINARY LOADER

Binary Loader loads binary output from Assembler into one or more fields in core in executable form. It operates in either 1-pass or 2-pass mode (all input files must be read in once for each pass). A field bit indicator, which determines the field into which loading occurs, is initially set equal to the field bit of the address typed in response to the ST= typeout. This indicator can be changed during loading by the occurrence, in any input file, of a FIELD word (generated by the PAL-D pseudo-op FIELD).

In 1-pass mode, Binary Loader can load core from locations 0 through 6777 in field 0 and all of fields 1 through 7. In 2-pass mode, it can load core from 0 through 7577 in field 0 and all of fields 1 through 7. Two-pass loading, then, is required when any of the input files require that coding be loaded into locations 7000 through 7577 in field 0; the reason for this is that Loader occupies these positions and cannot load the information over itself. To handle this situation, 2-pass loading operates as described in the following paragraphs.

PASS 1

All input files are read to find those portions of coding residing in the area from 7000 through 7577. Such coding is loaded into locations 6000 through 6577 instead. All other coding is bypassed. At the end of Pass 1, the contents of locations 6000 through 6577 are written into three scratch blocks on the system device.

PASS 2

Normal loading is performed, just as in the single pass of 1-pass loading, except that coding to be loaded in the 7000-7577 area is ignored. At the end of Pass 2, the contents of the three scratch blocks written during Pass 1 are read into locations 7000 through 7577. A jump is then made to the ST= address.

The ST= address has a double significance.

a. It initially sets the field bit indicator for loading[1].

b. It specifies the address (either in the loaded program or Monitor) to which control is to be transferred after loading.

[1]In 8 through 32K systems it is the user's responsibility to specify existing bank settings. In 4K systems, a 5-digit specification is illegal.

Examples

ST=10000	Begin loading in field 1 and jump to Monitor start (7600) after loading.
ST=31015	Begin loading in field 3 and jump to location 1015, field 3, after loading.
ST=27600	Begin loading in field 2 and jump to location 7600, field 2, after loading.

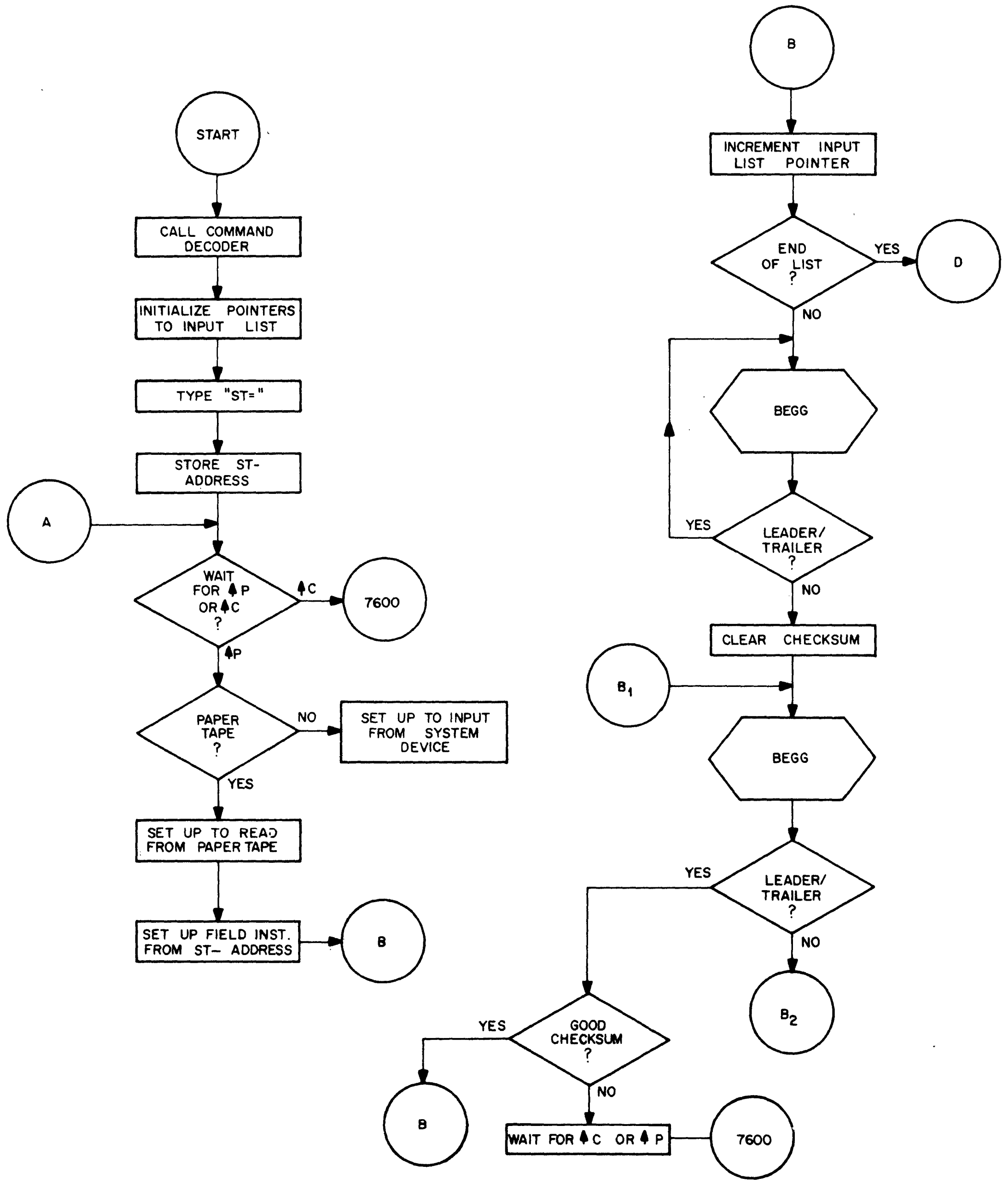


Figure D-1 Binary Loader Flow Chart (Part 1)

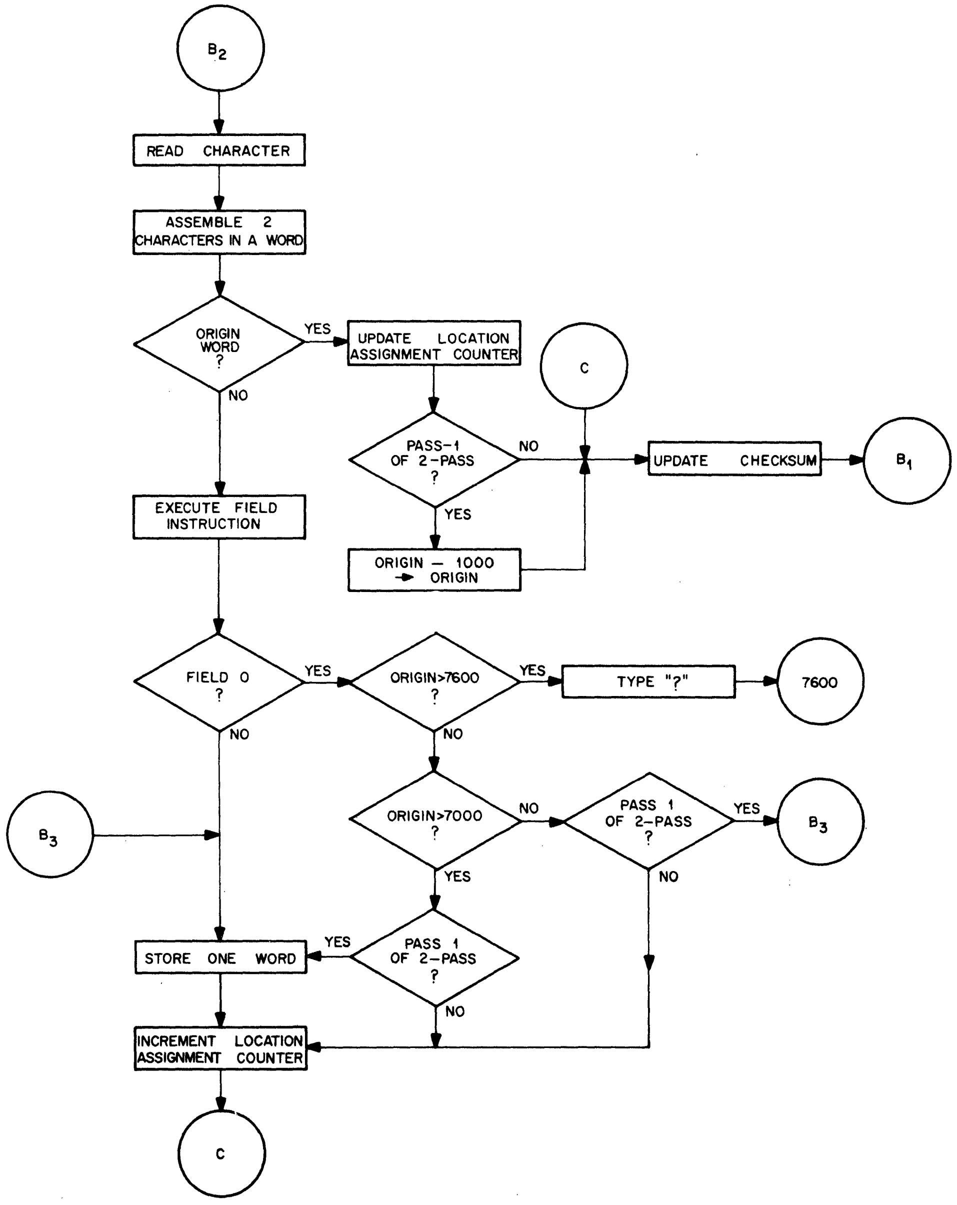


Figure D-1 Binary Loader Flow Chart (Part 2)

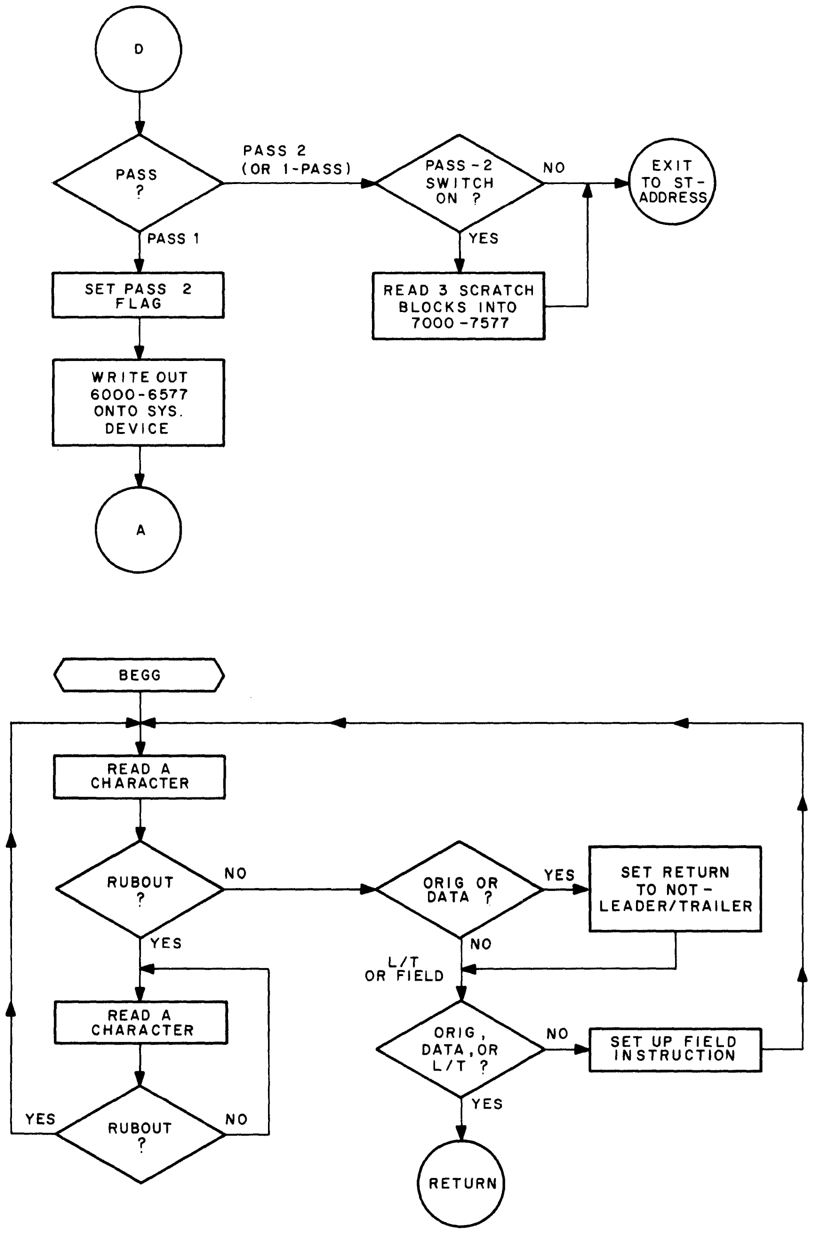


Figure D-1 Binary Loader Flow Chart (Part 3)

APPENDIX E
SYSTEM PROGRAMS

E.1 LOADING STATISTICS

Name	Core Limits	Entry Point	Pass
PIP	0-177, 1000-3177	1000	1
EDIT	0-3177	2600	1
PALD	0-3377, 3600-4377, 4600, 5200, 6200-6577, 7000-7577	6200	2
FORT	0-1777	200	1
.FT.	200-7377	--	2
STBL	600-777	600	1
FOSL	0-1577	200	1
.OS.	0-5177	--	1
DIAG	200-1177	200	1
.DDT	200-4577	--	1
.SYM	200-4577	--	-
DDT	7200-7577	7200	2

E.2 SAVE STATISTICS

PIP	SAVE PIP!0,1000-4777;1000
EDIT	SAVE EDIT!0-3177;2600
PALD	~~SAVE PALD!0-3377,3600-4377,4600,5200,6200-6577, 7000-7577;6200~~ SAVE PALD!0-7577;6200
FORT	SAVE FORT!0-1777;200
.FT.	SAVE .FT.!200-7377;0
STBL	SAVE STBL!600;600
FOSL	SAVE FOSL!0-1577;200
.OS.	SAVE .OS.!0-5177;0
DIAG	SAVE DIAG!200-1177;200

.DDT	SAVE .DDT!200-4577;0	
.SYM	SAVE .SYM!200-4577;0	
DDT	SAVE DDT!7200-7577;0	(User may assemble anywhere above location 4577)

APPENDIX F
I/O PROGRAMMING

F.1 GENERAL

The modular concept of input-output (I/O) handling of the disk system provides for easy maintenance and programming. The system device I/O is found in the following places (all I/O routines must be in field 0).

a. Top page of field 0 (location 7642) which is the I/O routine used by all system programs for normal I/O. A copy of this page is on block 0 of the system device. Block 0 of each DECtape is the DECtape I/O routine.

b. Interrupt versions of disk and DECtape routines are found in PIP.

c. Paper tape I/O is handled by individual programs.

F.2 CALLING SEQUENCE FOR BASIC I/O ROUTINE

The basic I/O routine (see Para. F.1.a.) is called as shown in Figure F-1. It is called in two ways, as determined by bit 2 of the function word.

a. Normal -- The I/O routine returns to JMS +6 (normal) or JMS +5 (error). For example, the following routine would read consecutive blocks from a file on the system device. The routine is initialized by putting the first block number of the desired file into location LINK. If an attempt is made to read past the last block of the file, an exit will be made to a routine called ENDFIL.

```
GETBLK,   0
          TAD LINK          /GET LINK FROM LAST READ
          SNA               /IS THIS END OF FILE?
          JMP I (ENDFIL     /YES
          DCA BLOK
          JMS I (7642       /CALL DISK I/O ROUTINE
          3                 /FUNCTION - READ
BLOK,     0
          BUFFAD            /BUFFER ADDRESS
LINK,     0
          JMP I (ERROR      /ERROR RETURN
          JMP I GETBLK
```

b. Indirect -- The I/O routine returns to the 12-bit address in the error return word +1 (normal or the 12-bit address in the error (ERROR)). An example of the indirect routine is given on page C-2a of this manual.

Calling Sequence	Explanation
JMS I SYSIO	Location SYSIO points to I/O
FUNCT	Function word*
BLOCK	Block to be accessed
CORE	Low-order core address
LINK	Filled by WRITE, used by READ
ERROR	Error return here
.	Normal return here

* Function word:	Bits 0-1	unused
	Bit 2	= 0, normal return
		= 1, indirect return at end of read/ write to address +1 in error return
	Bits 3-5	unit no. if DECtape
	Bits 6-8	memory field
	Bits 9-11	function: READ = 3; WRITE = 5

Figure F-1. Calling Sequence of System Routine

PDP-8/I DISK MONITOR SYSTEM

PROGRAMMER'S REFERENCE MANUAL

DEC-D8-SDAA-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 and the DEC products described.

Please comment on this publication. For example, in your judgment, is it complete, accurate, well-organized, well-written, usable, etc? ____________________

Did you find this manual easy to use? ____________________

What is the most serious fault in this manual? ____________________

What single feature did you like best in this manual? ____________________

Did you find errors in this manual? Please describe. ____________________

Please describe your position. ____________________

Name ____________________ Organization ____________________

Street ____________________ State ____________________ Zip ____________________

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 Quality Control
Building 12
146 Main Street
Maynard, Mass. 01754

digital

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

PRINTED IN U.S.A.