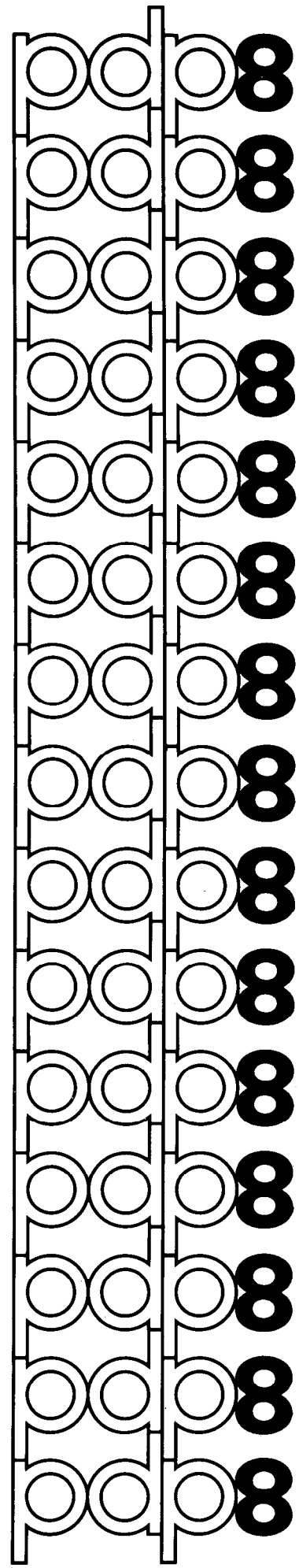
digital

4k disk monitor system



digital equipment corporation

DEC-08-ODSMA-A-D

4K DISK MONITOR SYSTEM

Order additional copies as directed on the Software Information page at the back of this document.

digital equipment corporation · maynard. massachusetts

2nd Printing (Rev) February 1974

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this manual.

The software described in this document is furnished to the purchaser under a license for use on a single computer system and can be copied (with inclusion of DIGITAL's copyright notice) only for use in such system, except as may otherwise be provided in writing by DIGITAL.

Digital Equipment Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by DIGITAL.

The HOW TO OBTAIN SOFTWARE INFORMATION page, located at the back of this document, explains the various services available to DIGITAL software users.

The postage prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

CDP	DIGITAL	INDAC	PS/8
COMPUTER LAB	DNC	KA10	QUICKPOINT
COMSYST	EDGRIN	LAB-8	RAD-8
COMTEX	EDUSYSTEM	LAB-8/e	RSTS
DDT	FLIP CHIP	LAB-K	RSX
DEC	FOCAL	OMNIBUS	RTM
DECCOMM	GLC-8	OS/8	RT-11
DECTAPE	IDAC	PDP	SABR
DIBOL	IDACS	PHA	TYPESET 8
			UNIBUS

PREFACE

This printing of the 4K Disk Monitor System Reference Manual updates the Disk Monitor System and Reference Manual (DEC-D8-SDAB-D) and incorporates Chapter 8 of Introduction to Programming 1970, and Change Notice (DEC-D8-SDAB-DN).

CONTENTS

TABLES

ILLUSTRATIONS

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 controls the flow of programs through the 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 allowing full access to the disk (or DECtape) - referred to as the SYSTEM DEVICE - for storage and retrieval of 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, PDP-8/I, 8L, also runs on 8/E, 8/F, 8/M.

4K of core

Teletype (Teletype is a registered trademark of Teletype Corporation)

3-Cycle Data Break (Option required with PDP-8/S)

At least one DF32 Random Access DECdisk File or at least one RF08 Random Access DECdisk File or a TC01 Automatic Control with a TU55 or TU56 DECtape transport. The DECtape must have timing and mark tracks written on it prior to use.

NOTE

The system recognizes up to 32K of core, up to four disks (one Type DF32 and three Type DS32 or one Type RF08 and three Type RS08) up to eight DECtapes (TC01's only) and a high-speed papertape 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 4K Disk 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 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 the system device. Following this, the System Program Library is created 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 (see Appendix F). 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 a CTRL/C is typed (this is accomplished by holding down the CTRL key while typing C) 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 CTRL/C causing the system to enter Monitor mode, return to Monitor at location 7600, and respond with a dot (.). At this point, any Monitor command can be issued 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 (*).

2.2 BOOTSTRAPPING THE MONITOR

The following discussion assumes that a customized Monitor has been built and stored on the 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, and is the same for both DF32 and RF08.

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

Disk Location	Contents	Symbolic
		*200
0200	6603	DMAR
0201	6622	DFSC
0202	5201	JMP .-1
0203	5604	JMPI .+1
0204	7600	7600
7750	7576	*7750
7751	7576	7576
		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
0217	7577	M201,	-201
0220	7755	CA,	7755
0221	7754	WC,	7754
0222	0220	RF,	0220

2. 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 CTRL/C when in Monitor mode or when a system program (or any user program which includes coding to sense a CTRL/C) is running. Jumping to 7600 after loading has been performed generates a START instruction (ST=7600). Certain errors also cause a jump to this location.

Monitor START performs the following actions.

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

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

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

The Monitor can be restarted by typing the RUBOUT key. A Monitor restart performs the same actions as described above except for step 1. A common use for RUBOUT is to terminate a command string when a mistake is made. The command string is ignored, and Monitor responds as described in step 3. The user core image on the system device is not changed by RUBOUT (it is changed, however, by ↑C).

2.4 COMMAND STRINGS

Commands are typed in the form of command strings to direct the Monitor, or a system program, to perform some action. Command strings are simple in format and afford 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, some Monitor command, such as CALL or SAVE can be typed.

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 one output device name. In the case of disk or DECtape the filename to be assigned to the output data must also be specified (see Paragraph 2.4.1).

*IN- Requests one or more (up to five) input device names. For disk and DECtape, filenames of input files must also be specified (see Paragraph 2.4.1).

*OPT- Request 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. The Command Decoder is a system program (.CD.) which is saved on the system at build time. 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)

S0: System Device in RF08 System (lower half of Disk 0)

Sn: Additional 128K segments of RF08 disk (n = unit number 1 through 7. Unit 1 is the upper half of disk 0,; unit 2 is the lower half of disk 1; etc.)

R: High-speed paper tape equipment (reader or punch)

T: Low-speed paper tape equipment of the Teletype (reader or punch)

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 with the following exceptions. Although both printed and nonprinted keyboard characters are allowable, printed characters are recommended.

1. Imbedded spaces cannot appear in a filename (they are ignored). It should be noted here that the Monitor is given the filename EX C. One reason for this unconventional use of an imbedded blank is to protect the Monitor from accidental destruction (e.g., deletion via PIP).

2. A file name 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. The data structure of these files is described in Appendix B under "DATA Structure". The extensions are:

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. (AF below means Version AF.)

Example:

```
NAME  TYPE    BLK
AF
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.

, Used to separate device names, when more than one is given in a command string. The comma is also used to separate core references in a SAVE command string, when more than one continguous area of core is specified.

; 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.

- 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.

CTRL/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. Type CTRL/C by holding down the CTRL key and striking C. ↑C does not echo (does not print).

CTRL/P Typed in response to a ↑ typeout. Instructs the system to proceed with the next operation. Type CTRL/P by holding down the CTRL key and striking P. ↑P does not echo (does not print). CTRL/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).

RETURN key 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. RETURN does not echo.

RUBOUT key 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. For all the examples below, the system response (typeout) is underlined for clarity.

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 or *IN-S0:PRO2↵	Use the file PRO2 on the system device as the input file. The "S:" is used for a DF32 or DECtape system, "Sn:" is used for RF08 system, see Table B1, Appendix B.
*IN-S:TST1,R:↵or *IN-S3:TST1,R:↵	Use the file TST1 on the system device and one file from the high-speed paper tape reader as the input files. (Note: "S:" is used for DF32 or DECtape system, "Sn:" for RF08 system, see Table B1, Appendix B.)
*OUT-D5:SPEC↵	Write the output file on DECtape unit No. 5 and assign it the filename SPEC.
*OUT-T:↵	Punch the output on the Teletype (low-speed) paper tape punch.
*OPT-M	Select option M. An automatic carriage return occurs after user response to an Opt-request.

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 - DISK SYSTEM BINARY LOADER

The Disk System Binary Loader takes as input the binary coding produced by the PAL-D Assembler and loads it into core in executable form. This loader is a system program saved on disk at build time. It is called by the user in the same manner as any system program. It occupies locations 6200-7577 and has a starting address of 6200. When loading is completed, 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. 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 a field other than 0.

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

Examples

*IN-R: ↵	Input one tape from the high-speed paper tape reader.
*IN-R:,R:,R: ↵	Input three tapes from the high-speed paper tape reader.
*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 high-speed paper tape reader.
*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.
*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; this forces Loader to start loading into the specified field until a "field setting" is found in the input file or tape (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 225, field 3, after loading.

*ST=10000↵

Load into field 1.
Return to Monitor after loading into field 1.

Loader now prints 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.

↑↑

First up-arrow: Loader is ready to load. If paper tape input, put the tape in the reader. Type CTRL/P.

Second up-arrow: Loading Computer. Type CTRL/P again to bring in data temporarily stored on the disk by the Loader and transfer to the specified starting address. (Note: if Teletype paper tape equipment is used, type CTRL/P before turning on the reader.

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 CTRL/P.

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

After all files have been entered, type CTRL/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 CTRL/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 CTRL/P but PRESS CONT on the console and START the paper tape reader.

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

2.5.2 Binary Loader Error Messages

An illegal checksum error condition causes Loader to type

?

and return to Monitor after a CTRL/P or CTRL/C. Error messages for illegal filenames or devices are as specified in Paragraph 2.9.

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 the 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. When the system device is an RF08 disk, all SAVE's are executed on S0. System and user files can be stored on S1, S2, etc., only by using PIP.

2.6.1 SAVE Command Format

.SAVE filename {!/:} core-specifications,...; entry-point

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 system program (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 user program. 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 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

fnnnn(1)-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 the following 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.

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 5-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 and a block requirement count is kept. Typing the terminating carriage return (↵), to start the SAVE initiates a directory name search on the system device. 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).

NOTE

The Monitor head contains a HLT instruction at location 7606. This can be used as a start address for a SAVEd program when it is desirable not to have the program start automatically.

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 :) if

.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 printed and control is returned to Monitor. When the system device is an RF08 disk, system and user files can be called into core only if they reside on S0.

2.8 MONITOR START CONDITION

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

1. 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.

2. Type SAVE SCRATCH: 7000-7577; n) (n is the point in the system program to which control returns, e.g., 200 for FOCAL).

3. Type CALL SCRATCH. This restarts with core intact.

2.9 SYSTEM ERROR MESSAGES

As an input command string is being typed, Monitor recognizes any incorrect syntax and remembers it. When a carriage return is typed, 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.

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). All library programs have the capability of accessing disk units other than unit 0; or in DECtape systems any DECtape unit. Where disk is the system device, PIP is the only program which can access DECtape. Builder must be used to record a system directory on the DECtape before it can be accessed by PIP. 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 a reply must be typed. The queries are the same for all programs. The replies will vary, however, depending on the particulars of the program being loaded.

When loading a program into core, first check to see whether Monitor is in core. This is done by typing CTRL/C (hold down the CTRL key while typing the C key). The CTRL/C does not echo (print on the teleprinter). If Monitor is in core, it responds by printing a dot (.) at the left margin of the teleprinter paper. If a dot is not typed in response to CTRL/C, Monitor is not in core. Refer to Chapter 2 for information on building Monitor and putting it into core.

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

Any program on the disk may be saved by typing a SAVE command, 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 in response to the Monitor's dot, as described in Paragragh 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 dot) with the four characters designated as the name of that program, as explained in Paragraph 2.7.

3.1 PIP

Peripheral Interchange Program (PIP) performs general utility operations, such as listing the contents of specified directories, deleting unwanted files from the system device, transferring files between devices, and copying specified files.

PIP is designed for the PDP-8/I, -8/L, and -8 computers only and can access any RF08 disk unit or DECtape unit. Before PIP can copy onto a DECtape, the Disk/DECtape System Builder must be used to build Monitor on the DECtape (see Section A.3).

3.1.1 Loading and Saving

PIP is loaded into core as indicated in Appendix D. Core requirements and starting address of the Loader are also found in Appendix D.

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

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

When PIP is loaded and saved, the printout resembles:

```
.LOAD)
*IN-R:)
*
*ST=)
↑↑      (type CTRL/P)
.SAVE PIP ! 0-5177;1000)
.
```

3.1.2 Operating Procedures

Once PIP has been loaded into core and saved on the disk it can be called in response to the Monitor dot. If a dot is not present as the last system response, type CTRL/C. The printout appears as follows:

```
.PIP)
```

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

```
*OPT-
```

and waits for one of the following options.

OPTION	MEANING
L	List entire directory of device to be specified
D	Delete a file to be specified
M	Move copy of directory to write-locked area of disk, as explained below
P	Protect blocks 0-176 of disk 0
A or RETURN key	Copy ASCII file (destination and origin(s) to be specified)
B	Copy binary file (destination and origin(s) to be specified). PIP copies relocatable binary-coded tapes (8K SABR Assembler output) to and from paper tape only.
F	Copy FORTRAN binary file (destination and origin(s) to be specified)
U	Copy user file (file structured origin and destination to be specified).
S	Copy system file (file structured origin(s) and destination to be specified). The U and S commands may not have paper tape as the destination because the files are core images and have no defined paper tape format.

Type only the option character; Monitor responds automatically with a carriage return and line feed. The *OPT line is not terminated with the RETURN key because it is a meaningful option.

It is illegal to type any character other than one of those listed above. PIP ignores the request, types ? (question mark), and asks for another option character. For example,

```
*OPT-G     G is not a legal option character
?
*OPT-
```

Refer to Appendix F for a table of valid I/O devices for PIP.

3.1.2.1 L and D Options - The L option lists the entire directory of the system device or DECtape on which a directory exists, providing the number of unused blocks and a report on each file. For example,

```
.PIP )                          User calls PIP
*OPT-L                          list option of the
*IN-S: )                        system device directory
FB=126                          PIP types number of free (unused) blocks
                                remaining on specified device

NAME  TYPE        BLK
AF
PALD  .SYS  (0)   0037          AF represents the version
EDIT  .SYS  (0)   0016          number of the Monitor system
LOAD  .SYS  (0)   0011          being used, followed by
.CD.  .SYS  (0)   0007          filename and description;
PIP   .SYS  (0)   0025          e.g., PAL-D is a system
DDT   .ASCII      0052          program in field 0 and
FOO   .USER (0)   0001          occupies 37(8) blocks of
BAR   .SYS  (0)   0037          storage
```

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, and S are the legal options indicating ASCII, binary, FORTRAN binary (compiler output), user program, and system program, respectively. The option letter may be typed before PIP completes the file type printout.

If the reply S (indicating a system file) follows the D option, PIP replies:

```
REALLY?
```

PIP will not delete a system file unless

```
Y .    (meaning yes)
```

is typed in answer to the question. Any reply other than Y causes PIP to repeat the FILE TYPE request. When Y is typed, PIP responds with

```
*IN-
```

and waits for the device and filename of the system file to be deleted. The printout would appear as:

```
*OPT-D                          delete option speci-
*FILE TYPE (A,B,F,U,S)-S        fying system file,
REALLY?N                        user must reply with Y,
*FILE TYPE (A,B,F,U,S)-S        PIP repeats request,
REALLY?Y                        user replied correctly,
*IN-S:BAR )                     PIP needs device and filename,
*OPT-                           file is deleted and PIP asks
                                for the next option.
```

When the file has been properly identified and deleted, PIP returns to ask for another option. If filename BAR, in the example above, had not been on the specified device, PIP would have ignored the request and printed a ? before asking for another option. For example,

*IN-S:BAR)	BAR is not the name of a
?	file on the specified
*OPT-	device

NOTE

The user should not try to delete the system files .CD. or LOAD.

When the reply to *FILE TYPE (A,B,F,U,S)- is other than S, PIP asks

*IN-

and waits for the device and filename of the file to be deleted. The printout would appear as:

*OPT-D	delete option specifying
*FILE TYPE (A,B,F,U,S)-B	a binary file;
*IN-S:BAR)	device and file names;
*OPT-	file is deleted, PIP asks for the next option.

If the wrong option is specified, type the RETURN key in response to PIP's request for data. PIP ignores the request, prints a ?, and requests another option. Example:

*OPT-L	user typed L by mistake
*IN-)	
?	
*OPT-	now specify another option

3.1.2.2 M and P Options - Options M and P, in conjunction with the hardware write-lock switch, are used to protect the lower 16K of the disk while using the system software. Either the system device or a DECtape unit numbered 0-7 can be specified. Since only input is requested, the action specified by the option is performed solely on the device specified. For instance, it is not possible to use the M option to move the system directory to another device.

The M option moves a copy of the first directory block (the first 25(10) filenames), block 177, of the device specified to block 3 of the same device. It also moves a copy of the first SAM (storage allocation map) block, block 200, of that device to block 4 of that device. For example, to move a copy of the system file directory, the printout is:

*OPT-M	move option specifying
*IN-S:)	the system device
*OPT-	PIP asks for another option

The directory which is moved should be one which does not contain files likely to be deleted from the working directory after the move.

The P option searches the first SAM block, block 200, for free or unused blocks in the lower half of the first disk. All unused blocks are marked as being used by Monitor, thus the lower half of the disk appears to have no unused space--it is protected. The write-lock switch on the disk control unit can be activated and Monitor will not attempt to write on the protected portion. If all blocks in the lower half of the first disk are already used, the P option does nothing. This option functions independently of the M option. Unless a copy of the true directory has been previously moved, there is no way (short of rebuilding the disk) to recover the space used by the P option. The printout would look as follows:

```
*OPT-P                  protect option specifying
*IN-S:)                 the system device
*OPT-                   PIP asks for another option
```

Files .SYM and .DDT should not be in the protected area of the disk. They are scratch files used by DDT-D and PAL-D during their operation and require output to the disk. (See PAL-D DISK ASSEMBLER, DEC-D8-ASAB-D, and Section 3.5.1 of this manual.)

Some typical uses for the M and P options are:

a. M, to save a specific disk (or DECtape) status.

b. M and P, to set write-lock switch and to operate protected.

3.1.2.3 A, B, F, U, and S Options - Options A, B, F, U, and S are used to transfer (copy) files from one device to another. When any of these five options is requested PIP responds with

```
*OUT-
```

and waits for input of the destination (output file or device), and, if the destination is disk or DECtape, the name of the file. For example,

```
*OPT-A                  copy an ASCII file
*OUT-S;ASCI)            specifying the destination and
                        file name
```

Only one destination is legal; if more than one is specified, PIP ignores the response, prints the error message E, and returns control to Monitor. For example,

```
*OPT-A                  copy an ASCII file
*OUT-S:ASCI,E           PIP recognizes the comma, which is used
.                       to separate file and device names,
                        control returns to Monitor
```

NOTE

The L and D options return to PIP's option request (*OPT-) when an illegal response is entered, and all other options return control to Monitor.

PIP indicates acceptance of the destination by responding with *, carriage return/line feed, and *IN-, and waits for the input specification; that is, from where the input is to originate. An attempt to specify more than one input to any but the A option causes PIP to ignore the response, type the error message E, and return control to Monitor. For example

```
*OPT-F              copy a FORTRAN file
*OUT-S:FORT)        specifying system device and filename
*                   PIP accepts destination
*IN-S:,E            input to system device, comma is
                    used to separate device names
.                   control returns to Monitor
```

If the input file specified to any option is not found on the specified device, PIP prints an I in place of the * and returns control to Monitor. For example:

```
*IN-S:FIL2)         the file does exist; when CTRL/P is typed,
*↑                  copying begins
*IN-S;FIL3)
I                   the file does not exist
.                   control returns to Monitor
```

When input is from DECtape there is a noticeable pause between the time the input device is specified and the time ↑ is printed--Monitor is searching the DECtape directory. Always wait for the ↑ before typing CTRL/P. If CTRL/P is typed prematurely, PIP returns to the option mode and ignores the current request.

If the B option is specified (copy a binary file), but the filename specified appears as an ASCII file, it is not acceptable; and PIP prints an I and control returns to Monitor. Use the L option to check the file directory for file type.

The A option allows any combination of up to 11 ASCII input files to be merged into one output file in the order specified by the input list. Generalized subroutines can be written as separate files to do often repeated operations and combined these with each specialized program BEFORE assembly thereby eliminating the need to rewrite such operations for each program. PIP acknowledges each legal input file by printing an *.

A paper tape file should not be merged with a disk or DECtape file when the output is going onto disk or DECtape--an illegal character may be inserted between the two files. Instead, copy each paper tape file onto disk or DECtape, and THEN merge the disk or DECtape files into one file. PIP merges ASCII files only (see examples at end of this Section).

The B, F, U, and S options are all formatted in the same fashion, and each requires the destination and origin of the file to be copied. The following example copies the system file PIP from disk to DECtape 3, using filename PIPX.

```
*OPT-S
*OUT-D3:PIPX)
*                    PIP acknowledges each legal file by
*IN-S:PIP)           printing an * for each file
*↑                   User types CTRL/P after every ↑
*OPT-
```

PIP copies only one segment (a section of paper tape delimited by leader/trailer code) of a binary paper tape onto disk or DECtape. If a multi-section binary tape is copied onto the disk using PIP, only the first section of the tape is read onto the disk even though the entire tape passes through the reader. Therefore, to copy all of a multi-section binary tape onto the disk or DECtape, each section must be copied as a separate file; for example, by physically cutting the tape after each section and then copying each section separately.

PIP, however, copies a multi-section binary tape to the paper tape punch.

A summary of the copy features is presented below.

Option		Number of Input Files	Disk	DECtape	High Speed Reader/Punch	Teletype
ASCII	A	11	Yes	Yes	Yes	Yes
Binary	B	1	Yes	Yes	Yes	Yes
FORTRAN Binary	F	1	Yes	Yes	Yes	Yes
User	U	1	Yes	Yes	No	No
System	S	1	Yes	Yes	No	No

Examples:

```
.PIP)

*OPT-L
*IN-S:)
FB=0121

NAME  TYPE      BLK
AF
PALD  .SYS  (0) 0037
EDIT  .SYS  (0) 0010
LOAD  .SYS  (0) 0011
.CD.  .SYS  (0) 0007
PIP   .SYS  (0) 0025
DDT   .ASCII    0052
FOO   .USER(0)  0001
BAR   .SYS  (0) 0037
```

User calls PIP
and requests the list option
of the system device directory
PIP types number of free (unused)
blocks remaining on specified device

followed by filename and description;
e.g., PAL-D IS A SYSTEM PROGRAM
in field 0 and occupies 37(8) blocks
of storage

```
*OPT-D                          User requests the delete option
*FILE TYPE (A,B,F,U,S)-U        and specifies type of file, U(user)
*IN-S:FOO)                      and device and filename; file is
                                deleted

*OPT-D                          User requests the delete option
*FILE TYPE(A,B,F,U,S)-S         and specifies type of file, S
REALLY?Y                        (system) (PIP double checks); Y is the
                                only meaningful answer

*IN-S:BAR )                     User specifes file and filename; file
                                is deleted

*OPT-L                          User request list option
*IN-S: )                        and system device directory,
FB=0161                         Note increase of 40(8) free blocks (see
                                above)

NAME TYPE      BLK
AF
PALD .SYS (0) 0037
EDIT .SYS (0) 0016
LOAD .SYS (0) 0011
.CD. .SYS (0) 0007
PIP  .SYS (0) 0025
DDT  .ASCII   0052
*OPT-D                          User requests delete option
*FILE TYPE (A,B,F,U,S)-S
REALLY?N                        Y is only response for deletion of
*FILE TYPE (A,B,F,U,S)-S        a system file; other responses
REALLY?W                        cause PIP repeat the file type
*FILE TYPE (A,B,F,U,S)-S        request
REALLY?Y
*IN-S:EX C )                    Even if user responds to REALLY?
?                               with Y, PIP does not delete the
                                Monitor file

*OPT-D
*FILE TYPE (A,B,F,U,S)-U
*IN-S:NONE )                    PIP knows NONE is not an existing
?                               user filename on the system device
*OPT-D                          and responds with ?
*FILE TYPE (A,B,F,U,S)-A        User requests ASCII file option
*IN-S:EDIT )                    PIP also knows when the filename
?                               and file type don't match; EDIT is
*OPT-D                          a system program
*FILE TYPE (A,B,F,U,S)-B
*IN-S:EDIT )
?
*OPT-
```

Merge into an ASCII paper tape on the high speed punch, one tape from the reader, one tape from the Teletype, one file from disk called SRC, and one file from DECtape 7 called SRC1.

```
*OPT-A
*OUT-R:)
*
*IN-R:,T:,S:SRC,D7:SRC1)
*
*
*
*↑↑↑↑                          (TYPE CTRL/P after each file)
*OPT-
```

Copy the system file PIP from disk to DECtape using filename PIPX.

```
*OPT-S
OUT-D3:PIPX)
*
*IN-S:PIP)
*↑                             (TYPE CTRL/P)
*OPT-
```

3.1.3 Using PIP in an RF08 System

With an RF08 disk as system device, inputs and outputs are specified by S0:,S1:,etc.

Examples:

```
*OPT-S
*OUT-S1:LLDR)
*
*IN-S0:LLDR)
*
```

3.2 EDITOR

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

Editor operates either in command or text mode. In command mode, each typed input is interpreted as a command to perform a certain operation. 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 (Chapter 5 of Introduction to Programming) with the following exceptions.

1. CTRL Commands:

CTRL/P	During output, progress stops and control returns to command mode.
CTRL/C	Always returns control to Monitor.

2. Commands:

P	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 transfer the remaining input to the output 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	Mode	Functions
(RETURN)	Text	Enter the line in the text buffer.
	Command	Execute the command.
RUBOUT	Text	Delete from right to left one character for each rubout typed (is not in effect during a READ command).
	Command	Delete entire command.
FORM FEED	Text	End of input, return to command mode.
	Command	List the next line.
CTRL/U	Text	Deletes the entire line of text currently being entered and performs a carriage return/line feed. Line counter is unchanged.
	Command	Cancels commands just typed and performs a carriage return/line feed. * is displayed.
. (period)	Command	Current line counter used as argument alone or in combination with + or - and a number.

(Continued on next page)

Table 3-1 (Cont.)
Special Key Functions

Key	Mode	Functions
/ (slash)	Command	Value equal to number of last line in buffer and used as argument.
LINE FEED	Text	Used in SEARCH command to insert a carriage return/line feed combination into the line being searched.
	Command	List the next line.
ESCape	Command	List the next line.
< (left angle bracket)	Command	List the previous line.
> (right angle bracket)	Command	List the next line.
= (equal sign)	Command	Used in conjunction with . and / to obtain their value (.=0027).
:(colon)	Command	Lower case character, same function as =.
CTRL/TAB	Text	On output, is interpreted as 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. If the low-speed reader is used, Editor pauses when a form feed is encountered so the reader can be turned off. Type any character to return to command mode.
APPEND	A↵	Append incoming text from Teletype to any already in the buffer until a form feed is encountered.
LIST	L↵ nL↵ m,nL↵	List the entire buffer. List the line n. List lines m through n.

(Continued on next page)

Table 3-2 (Cont.)
Summary of Editor Commands

Command	Format(s)	Meaning
OUTPUT (PUNCH)	P↵	Output the entire contents of the buffer followed by a form feed and return to command mode. When input is from the low-speed reader and output is to the low-speed punch, a P command causes Editor to pause, to allow time to turn on the punch. Any character may be typed to begin punching. When the specified amount has been punched, the Editor pauses so the punch can be turned off. Type any character to return to command mode. Using a P and then an E, without typing a K in between, causes the buffer to be written twice.
	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. When the input is from the low-speed reader, Editor pauses at the end of an N command so that the reader may be turned off. Type any character to return to command mode with text in the buffer. If the tape runs out before the N command is completed, Editor prints a ? and returns to command mode with an empty buffer. The ? indicates that Editor tried to read past the physical end of the input tape. When output is to the low-speed punch, an N command causes Editor to pause before the first punch operation only. If the N command is completed before the entire tape is processed, the Editor pauses after the specified number of N's have been processed. At this point, both the reader and the punch should be shut off. Type any character to return to command mode.
	nN↵	Repeat the above sequence n times.
KILL	K↵	Kill the buffer.

(Continued on next page)

Table 3-2 (Cont.)
Summary of Editor Commands

Command	Format(s)	Meaning
DELETE	nD↵	Delete line n. Editor does not use the core locations made available by line deletion until an N or K command is given.
	m,nD↵	Delete lines m through n.
INSERT	I↵	Insert before line 1 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 beginning with a tag.
	nG↵	Get and list the next line which begins with a tag. Start search with line n.
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 the remaining input to the output file) then output two form-feed characters signifying an end-of file; legal only for output to the system device. If the low-speed reader is used, the E command causes Editor to pause until the low-speed reader is turned on. After the tape is read and processed the Editor closes the file and returns to Monitor control. If there is no additional tape to be read, typing any character causes the file to be closed normally.

NOTE

If the low speed reader runs out of tape, or if the reader is turned off before a form feed is read, the Editor prints a ? and returns to command mode. The buffer contains material already read. Text entries should be terminated with the FORM FEED character. The text buffer has capacity for 50 to 60 lines of text or 3,600 characters; and the source should be limited to this number of lines or characters. This ensures that no buffer overflow will occur.

If a tape is prepared off-line, care must be taken with respect to form feeds. If input is from the low-speed reader, the Editor ignores the next character after the form feed. For this reason, AT LEAST one blank space should be punched after a form feed in an off-line tape. (Using Editor prepared tapes eliminates this problem.)

Editor prints an error message consisting of a question mark whenever nonexistent information is requested or an inconsistent or incorrect format is used to type a command. The question mark is followed by a carriage return/line feed and the command is ignored.

Editor performs a failsafe operation to preclude the loss of data when the output of text to the disk exceeds the space available. The failsafe technique causes the message FULL to be printed to indicate the buffer is filled. The failsafe operation continues as follows:

1. The E and N commands are disabled. A ? is printed when either is attempted.

2. A P command causes Editor to attempt to save the text by punching it on the high-speed punch. If there is no high-speed punch and a P command is issued the Editor loops, waiting for the high-speed punch flag. At this point the program should be stopped and restarted at location 0177, putting Editor into command mode with the buffer still intact. The buffer can be recovered by using the L command with the low-speed punch turned on.

NOTE

Before the punched output is reloaded into the buffer, the first and last lines of the buffer should be inspected. If the punch was turned on beforehand, the first line is L.RETURN; and the last line is * because Editor returned to command mode. These lines should be deleted.

3. All standard editing functions are available after the failsafe technique has occurred.

If the computer halts at location 2330, a system error has occurred while reading from the disk. Run disk maintenance tests to determine cause of error.

3.2.1 Loading and Saving

Editor is loaded into core from punched paper tape by the Binary Loader. When Editor is in core, it occupies locations shown in Appendix D.

To load Editor into core, call LOAD, using Monitor, and reply 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 as indicated in Appendix D.

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

A sample printout for loading and saving Editor is:

```
.LOAD)
*IN-R:)
*
*ST=7600)
↑↑
.SAVE EDIT!0-3377;2600)(See Appendix D.)
.
```

3.2.2 Operating Procedures

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

```
EDIT)
```

is typed.

Editor is now in core and responds by typing

```
*OUT-
```

Type one of the following output devices: (T:) for low-speed reader/punch; (R:) for high-speed reader/punch; (S:name) for output to the system device on a file called NAME immediately after OUT-. (E must be typed to properly close the output file.) If the specified device is not valid, that is, not declared when building Monitor, Editor responds with an error message (see Paragraph 2.9) and returns control to Monitor. Refer to Appendix F for a table of valid I/O devices for EDIT.

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

```
*
*IN-
```

Specify the input device by typing T: , R: , or S:name or in the same manner as when replying to *OUT-, above.

The Editor responds with

```
*OPT-
```

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, 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.

RETURN — None of the above options; assume conversion of two or more blanks to tabs, and not D.

Editor responds to one of the above options 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 symbolic program can now be read into core with the R command or typed into core with the A command (see Table 3-2).

Examples:

```
.LOAD)                          Call Loader using Monitor
*IN-R:)                         Input to be from high-speed reader
*                               Input device valid
*ST= )                          Return to Monitor after loading
↑↑                              Editor is loaded
.SAVE EDIT!0-3377;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, (Don't use dynamic deletion
                                mode or file will be lost.)
*
*IN-S:SRC1)                     Input from filename SRC1
*                               File name 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      (CTRL/P was typed here, stopped listing of buffer)
*/L
$
.                               (CTRL/C was typed here)
```

The Disk Editor does not always require specific input or output. For example, when creating a file from the keyboard the following is valid:

```
.EDIT)
*OUT-S:FILE)
*
*IN-)
*
*OPT-)
*A)
 TEXT MAY BE KEYED-IN DIRECTLY FROM THE
 KEYBOARD......
```

When lines are deleted the value of the current line counter is not set to the number of the line preceding the deletion. The counter is set to the number of the line deleted, or in the case of multiple deletion, it is set to the number of the line first deleted.

Example:

```
*.L)
            TAD(-1
*.=0024
*.D
*.=0024
*2,5L)
            TAD (-2
            DCA CNT
            TAD TEMP
            JMP .-1
*2,5D)
*.=0002
```

3.3 4K PAL-D DISK ASSEMBLER

The Program Assembly Language for the Disk system (PAL-D), is the symbolic assembly program designed primarily for the 4K PDP-8 family of computers with disk or DECtape. (8K PAL-D is an expanded version of 4K PAL-D and both binary tape and documentation are available from DECUS (Digital Equipment Corporation User's Society) upon request.)

PAL-D is compatible with PAL III except in the way it recognizes Memory Reference Instructions, and with MACRO-8 in respect to the following features: Boolean operators, linkage generation, literals, and a text facility. It does not have user-defined macros, floating point constants, or double precision numbers. The reader should review the 4K ASSEMBLER MANUAL (DEC-08-LAS4A-A-D) before proceeding.

The 4K PAL-D Assembler performs many useful functions, making machine language programmming 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.

Pseudo-operators (pseudo-ops) may be used 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	Punch all literals (a FIELD mark is punched followed by an origin at location 200). To assemble code at location 400 in field 1 write: FIELD 1 *400
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 ASCII code trimmed to six bits.
$	End of symbolic program, terminate current pass.
PAUSE	End-of-file or paper tape, suspend processing, proceed to next file or paper tape and resume processing.
EXPUNGE	Erase permanent symbol table, except pseudo-ops.
FIXTAB	Append to permanent symbol table all symbols defined before the FIXTAB.
I	Indirect Addressing.
Z	Page zero reference.

COGNITION OF MEMORY REFERENCE INSTRUCTIONS

e PAL-D Disk Assembler uses a different criterion for recognizing mory reference instructions than does PAL III.

ven a line of code such as:

```
SYMB1  SYMB2
```

where neither SYMB1 nor SYMB2 is an assembler pseudo-op, PAL-D will treat this line as a memory reference instruction only if SYMB1 is a permanent symbol and SYMB2 is not a permanent symbol. Note that the category "permanent symbol" includes user-defined symbols that have been added to the PAL-D permanent symbol table via the FIXTAB pseudo-op.

All other combinations of permanent and non-permanent symbols for SYMB1 and SYMB2 will cause the line to be treated as a microinstruction. The values of SYMB1 and SYMB2 will be inclusively ORed together and this resultant value will be generated as code for the line. Because of these considerations, users should be cautious in their use of FIXTAB pseudo-op.

Note especially that the floating point instructions (FADD, FSUB..... FPUT) must be defined at the beginning of a program and must be followed by a FIXTAB pseudo-op, as follows:

```
FADD=1000
FSUB=2000
.
.
FPUT=6000
FIXTAB
```

Refer to Appendix G for a complete list of all the permanent symbols contained in PAL-D.

4K PAL-D PROGRAM PREPARATION AND ASSEMBLER OUTPUT

The information on PAL III Program Preparation and Assembler Output is applicable to PAL-D with the following exceptions:

The symbol table is punched after the listing and is preceded and followed by a small amount of leader/trailer (200) code. The symbol table tape will be punched whether or not a listing is requested, and will appear either on the terminal punch or on the high-speed punch output, depending upon the device being used.

It is possible to terminate any pass of the assembly by typing a CTRL/P on the console terminal. CTRL/P causes PAL-D to go on to the next pass of the assembly. Assembly can be terminated at any time by typing a CTRL/C.

During the listing pass note that blank lines will remain in the listing and the form feed (214 ASCII) character is ignored.

The 4K PAL-D symbol table has room for 161 symbols in core (about 6 to 8 pages as an average). That number can be expanded as explained in paragraph 3.3.1.

Following pass 2, the binary output can be loaded into core by the Disk Monitor System Binary Loader.

3.3.1 Loading and Saving

PAL-D is loaded into core from punched paper tape by the Binary Loader. Appendix D illustrates the locations occupied when PAL-D is in core.

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

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

A sample printout for loading and saving PAL-D is: (see Paragraph 2.5).

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

The user's core resident symbol table can hold 160(decimal) user-defined symbols under the Disk Monitor System. This may be expanded by saving on the system device a user file named .SYM which can be used by PAL-D to store extra symbols. Each user-defined symbol occupies four words. The symbol table can be expanded by 128(decimal) or 200(octal) locations (one core page) by saving a file with the following statement.

```
.SAVE .SYM:0-177;0)                    (192 user symbols)
```

If a larger symbol table area is needed, simply specify additional pages, where each page saved provides storage for 32 additional symbols. For example:

```
.SAVE .SYM:0-377;0)                    (224 user symbols)
```

will save two core pages, and

```
.SAVE .SYM:0-1777;0)                   (416 user symbols)
```

will save eight core pages for symbol storage.

3.3.2 Operating Procedures

Monitor is used to transfer PAL-D from the system device to core. Begin by typing

```
.PALD)
```

PAL-D responds with a request for the output device

```
*OUT-)
```

Select the output device by specifying one of the following.

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

Refer to Appendix F for a table of valid I/O devices.

PAL-D then responds with:

```
*IN-
```

and waits for specification of the input device(s). Up to five input devices may be specified (for example, R:, T:, R:, R:, T:), but in this example input from the high-speed reader was selected.

```
R:)
```

If the devices in the parenthetical example above had been specified, 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 requests the third-pass listing option by printing

```
*OPT-
```

Type one of the following.

T	meaning listing and symbols are to be produced on the terminal
R	meaning listing and symbols are to be produced on the high-speed punch
RETURN key	meaning no third pass desired, symbols are printed on the terminal

any other character means no third pass desired.

Note that blank lines remain in the listing and that form-feed (214) which could not be treated properly, because of limited core, is ignored in the present version.

The entire printout might appear as follows.

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

When the high-speed punch is selected as a listing device, the alphabetic symbol table produced at the end of pass 3 is also produced on the high-speed punch.

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 prints an up-arrow (↑) on the terminal and waits for a CTRL/P to continue with the assembly. When using the low-speed reader on input i.e., (IN-T:), a CTRL/P must be typed BEFORE turning the reader on. CTRL/P terminates any pass and automatically begins the next one. (CTRL/P does not echo.)

When using the disk as an output device, the compiled binary is ready to be loaded for execution following pass 2.

The symbol table will be output at the end of pass 2 if the third pass has not been requested. If pass 3 is requested, PAL-D will follow the assembly listing with the user's symbol table in alphabetical order (in addition to the assembled binary output).

Type CTRL/C to terminate assembly and return control to Monitor at any time. The Symbol Table is punched after the listing and is preceded and followed by 38 frames of leader/trailer (200) code. The symbol tape is punched whether or not a listing is requested, and appears either on the terminal or high-speed punch. 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

```
nn          xxxx
```

where nn is a two-letter code which specifies the type of error, and xxxx 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 in Table 3-4.

Table 3-4
PAL-D Error Messages

Error Code	Explanation
BE	Two PAL-D internal tables have overlapped. This situation can usually be corrected by decreasing the level of literal nesting or the number of current page literals used prior to this point on the page.
DE	System device error. An error was detected when trying to read or write the system device; after three failures, control is returned to the Monitor.
DF	System device full. The capacity of the system device has been exceeded; assembly is terminated and control is returned to the Monitor.
IC	Illegal character. An illegal character was encountered other than in a comment or TEXT field; the character is ignored and the assembly continues.
ID	Illegal redefinition of a symbol. An attempt was made to give a previously defined symbol a new value by means other than an equal sign; the symbol was not redefined.
IE	Illegal equal sign. An equal sign was used in the wrong context. Examples: TAD A+=B A+B=C The expression to the left of the equal sign is not a single symbol.
II	Illegal indirect address. An off-page reference was made; a link could not be generated because the indirect bit was already set. Example: *200 TAD I A : PAGE A, 7240

(Continued on next page)

Table 3-4 (Cont.)
PAL-D Error Messages

Error Code	Explanation
ND	The program terminator, $, is missing (with TSS/8 only).
PE	Current nonzero page exceeded. An attempt was made to: 1. Override a literal with an instruction, or 2. Override an instruction with a literal; this can be corrected by a. Decreasing the number of literals on the page, or b. Decreasing the number of instructions on the page.
PH	Phase error. PAL-D has received input files in an incorrect order; program terminator ($) is either missing or misplaced. Assembly is terminated and control is returned to the Monitor.
SE	Symbol table exceeded. Assembly is terminated and control is returned to the Monitor; the symbol table may be expanded to contain up to 1184 user symbols by saving a file named .SYM on the system device. (Disk Monitor System only.)
US	Undefined symbol. A symbol has been processed during pass 2 that was not defined before the end of pass 1.
ZE	Page zero exceeded. Same as PE except with reference to page 0.

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
-

*ST= )                     Return to Monitor after loading
```

↑↑	PAL-D is loaded
.SAVE PALD!0-7577;6200)	PAL-D is saved on disk (see Appendix D)
.PALD)	Call PAL-D
*OUT-S:BIN)	Output to filename BIN on system device
*	Filename and system device valid for output
*IN-R:)	Input from high-speed reader
*	Reader is valid input device
*OPT-R	Output listing and symbols on high-speed punch
↑↑↑	CTRL/P should be typed after inserting source tape in reader for each pass. (If both input and output are to system device, no ↑'s are typed.)
.LOAD)	Assembly is finished; control returns to Monitor; user called the Loader.
*IN-S:BIN)	Input from filename BIN on system device
*	Filename and system device valid for input
*ST=7606)	Transfer control to the HALT in the Monitor after loading the user program
↑↑	CTRL/P typed two times in response to each ↑
.	User program is loaded; the computer halts with user program in core

3.4 FORTRAN-D

FORmula TRANslation for the Disk System (FORTRAN-D), is a modified version of PDP-8 4K FORTRAN designed for PDP-8 computers with disk or DECtape units and 4K words of core memory.

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. Appropriate commands can be typed at the terminal 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:

1. Arithmetic statements define numerical calculations.

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

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

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

For more detailed information on FORTRAN-D refer to DEC Library document 4K FORTRAN, DEC-08-AFDO-D.

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

Table 3-5
Summary of FORTRAN Statements

Statement and Form	Explanation
Arithmetic Statements	
v=e	v is a variable (possibly subscripted); e is an expression.
Control Statements	
GO TO n	n is a statement number.
GO TO (n(1),n(2),...,n(n)),i	n(1),...,n(n)) are statement numbers; i is a nonsubscripted 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 the statement number of a CONTINUE; i is an integer variable; k(1),k(2),k(3) are integers or nonsubscripted integer variables. K(3) if omitted is assumed to be 1.
CONTINUE	Proceed.
PAUSE	Temporarily suspend execution.
PAUSE n	n is a decimal address; subroutine execution will commence at the octal equivalent of n.
STOP	Terminate execution.
END	Terminate compilation; last statement in program.
Specification statements	
DIMENSION v(1)(n(1)),v(2)(n(2)),...,v(n)(n(n))	v(1),...,v(n) are variable names; n(1),...,n(n) are integers.
DEFINE device	Device is DISK or TAPE, system I/O device.
FORMAT (s(1),s(2),...,s(n))	s is a data field specification. (I, E, F and A are accepted by FORTRAN-D.)

(Continued on next page)

Table 3-5 (Cont.)
Summary of FORTRAN Statements

Statement and Form	Explanation
COMMENT	Designated by C as first character on line.
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:

Format	Meaning
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.

1. The READ and WRITE statements perform sequential input and output either on paper tape or on the system device.

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

3. 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.

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

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

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

7. When the ACCEPT statement is used, the rubout character deletes the preceding input as shown in the following examples.

Typed and Corrected	Read
Integer Numbers:	
128R1028	+1028
128R-28	-28
-128R128	+128

'R' indicates a rubout.

Floating-point numbers:	
2R42	+42.0
+2.R42	+42.0
-2.0R2.0	+2.0
42R-42.2	-42.2
20E6R5	+2.0 x 10(6)
2.0E-6R5	+2.0 x 10(5)

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

   The device assignments for FORTRAN-D READ and WRITE statements are:

Device Code	Device
1	Terminal
2	High-speed reader/punch
3	System device

For example:

```
READ 2,f,list
```

reads from the high-speed reader.

The following examples show how the READ and WRITE statements might be used in a typical FORTRAN program. (The example programs which use the system file must be assembled then run with FOSCL as shown in Example 4, section 3.4.2.2. Specify the input data file and output data file.)

```
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 ASCII 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 3
        IDEV=3
6       SUMX=0
        SUMY=0
        DO 10 I=1,100
        READ IDEV,20,X(I),Y(I)
        WRITE 2,20,X(I),Y(I)
        SUMX=SUMX+X(I)
        SUMY=SUMY+Y(I)
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,
        1//"TYPE 0 TO STOP, 1 TO CONTINUE")
40      FORMAT (I)
        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 PDP-8 4K FORTRAN compiler in the following ways.

1. It uses the disk or DECtape unit during its operation.
2. It compiles programs which have been stored on the system devices or programs which have been prepared on punched paper tape.
3. It generates a FORTRAN binary output file either on the system devices or on punched paper tape.
4. Significant improvements have been employed with the READ and WRITE statements.
5. Input and output devices are determined using the Command Decoder.
6. It is possible to terminate compilation at any time by typing CTRL/C, thus returning control to Monitor.
7. Within certain restrictions, a program compiled on a system device may be executed immediately by typing CTRL/P after compilation of the program.
8. A statement number need not be delimited by a semicolon, unless the user wishes it to be employed for appearance.
9. Statements without preceding numbers must be preceded by a space, a tab, or a semicolon.

3.4.1.1 Loading the FORTRAN Compiler - The FORTRAN compiler loader (FORT) is loaded into core from punched paper tape in one pass through the Binary Loader. FORT may be saved on the system device as described in Appendix D.

The main portion of the FORTRAN compiler (.FT.) is loaded into core with the Binary Loader. .FT. may be saved on the system device by the same method as FORT. When the compiler (both FORT and .FT.) is loaded and saved on the system device, the entire procedure generates the following printout:

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

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

.FORT)

is typed in. In response, the Command Decoder then prints

*OUT-

and waits for one of the following to be specified:

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

Refer to Appendix F for a table of valid I/O devices for FORTRAN-D.

Command Decoder responds with an * when it recognizes a valid output device, and then prints

*IN-

and waits for 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 prints 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 prints ↑ when it is ready to receive the tape for compilation.

Type CTRL/P, to initiate compilation. At the end of compilation the compiler prints any error diagnostics necessary, a carriage return/line feed, and ↑.

The process described above produces the following printout.

```
.FORT)
*OUT-R:)
*
IN-R:)
*
↑
↑          (CTRL/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 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 prints a four-digit error code and returns control to the Monitor. In the second case, the compiler prints SYS followed by a four-digit error code. Type CTRL/C to return control to the Monitor.

A complete list of the system error messages and their explanations is shown in Table 3-6.

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

(Continued on next page)

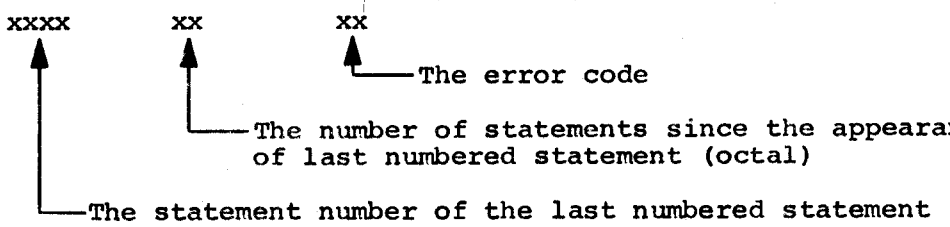
Table 3-6 (Cont.)
Compiler Systems Diagnostics

Error Code	Explanation
1521	Same as above
1626	Same as above
1726	WRITE error during SAM block search
3100	Illegal operator on compiler stack or machine malfunction.
3417	Pre-precedence error. (Compiler error or machine malfunction.)
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 or a machine malfunction.
6223	Error while loading .FT.
6226	Same as above
6257	Same as above
6407	Illegal overlay request or a machine malfunction.
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

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  CTRL/C typed; 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:



For example, during compilation of the statements

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

the error message

```
10   1    11
```

would be printed, indicating that an error exists in a statement which occurs 1 statement (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 compilations error messages and their explanations.

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
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 preceded 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. It helps create and debug user 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:

1. The limits of the interpretive code.
2. A list of variable names and their corresponding locations (the symbol table).
3. 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: )
*
*ST= )
↑↑
.SAVE STBL!600-777;600 )     (See Appendix D.)
.
```

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

When Symbolprint is called into core, it prints the interpretive code limits, symbol table, statement number table, carriage return/line feed, and ↑. Type CTRL/P to execute a user program or CTRL/C to return to Monitor.

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
*
↑                    (CTRL/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
↑                    (CTRL/C typed here.)
.
```

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.

1. It loads and executes programs which have been compiled and saved on the system device or programs which have been compiled on paper tape.

2. 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.

3. FOSL recognizes READ and WRITE statements which may read and write data in ASCII format on either the low-speed paper tape reader/punch, the high-speed paper tape reader/punch, or the system device.

4. The execution of a FORTRAN program may be interrupted at any time by typing CTRL/C; control returns to Monitor.

3.4.2.1 Loading the FORTRAN Operating System - The FORTRAN Operating System Loader (FOSL) is loaded into core in one pass. It may be saved on the system device as described under SAVE Command Format.

The operating system interpretive and arithmetic package (.OS.) is loaded into core in one pass through the Loader. .OS. may be saved on the system device by the same methods as FOSL. When the FORTRAN operating system (both FOSL and .OS.) is loaded and saved on the system device, the entire procedure will generate the following printout:

```
.LOAD )
*IN-R: )
*
*ST= )
↑↑
.SAVE FOSL!0-1777;200 )    (See Appendix D.)
.LOAD )
*IN-R: )
*
*ST= )
↑↑
.SAVE.OS.!0-5177;0 )       (See Appendix D.)
.
```

The Loader occupies core locations 0-1777 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 CTRL/P immediately after compiling a FORTRAN program onto the system device, or by typing FOSL immediately after Monitor prints a dot.

```
.FOSL↵
```

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
S:name↵      for system device input
```

FOSL prints * when it recognizes a valid input device and prints

```
*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 prints

```
*OUT-
```

Specify the desired output filename (if any) by typing S:name↵ (name is the name of the file). FOSL again prints

```
*IN-
```

Respond with S:name↵ (where name is the filename of the data file), and the RETURN key.

If the FORTRAN program is on paper tape, Loader prints ↑ when it is ready to begin loading. Type CTRL/P and the tape loads.

When the FORTRAN program or file is loaded, FOSL prints

```
*READY
```

Place data tapes in the appropriate reader and type CTRL/P to begin executing the program. (If the low-speed reader is used, turn the reader ON AFTER typing CTRL/P.)

When a STOP or END statement is executed, or when an end-of-file is read on the system device, the operating system prints ! and control returns 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: )
*
OPT- )
*
↑↑
ERROR 01         (CTRL/P typed here)
*READY
↑                (Program execution begins here)
```

Example 3

```
.FORT )
*OUT-S:SMSQ )
*                Compile
*IN-S:SMSQ )     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. This error will always occur when a PAL routine is assembled with a FORTRAN program. In this case the operator decided to attempt to execute the program in spite of the checksum error.

The following method for saving FORTRAN-D object programs enables the user to call his program with the FORTRAN-D operating system as a Monitor system program.

1. Compile the FORTRAN-D source program as usual.

2. Load and run the compiler output under FOSL.

3. At the end of a successful run, control will return to the Monitor. The user should then call FOSL to reload the compiler output as if to run the program again.

4. When FOSL has loaded the program (and subroutines, if any) it types

   READY
   ↑

   Type CTRL/C to return to the Monitor.

5. A core image of the user's program in FOSL may now be saved on disk as a system program by typing

   SAVE NAME!0-7577;5043↵

   Note that the starting address must be 5043.

6. When the Monitor has returned control to the user, the core image may be called at any time as a system program. On starting, the program types

   READY
   ↑

7. CTRL/P initiates execution. When execution is complete the program types "!" and returns to the Monitor.

Users who run FORTRAN-D with binary subroutines will find this method especially time saving. Execution times of tested FORTRAN programs can be increased by changing location 212 of .OS. from 4664 to 7000.

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

Table 3-8 lists the operating system error messages. The message indicates .OS. a system device read error.

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 or machine malfunction
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 or machine malfunction
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
77	Stack overflow error
	Errors 76 and 77 may be caused by source program or loading error; to correct do the following

(Continued on next page)

Table 3-8 (Cont.)
Operating System Diagnostics

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

When an error occurs, execution stops and the operating system waits for a CTRL/P or CTRL/C.

3.4.2.4 Debugging Aid (Diagnose) - Diagnose is a basic system program used to debug FORTRAN programs. 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.

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

2. It prints a message indicating the contents of the current location counter (CLC).

3. If the counter stack is nonempty, it prints the contents of that stack.

4. If location zero is nonzero, it prints 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 D. A sample printout is:

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

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

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

The use of Diagnose is demonstrated by the example of the following test program which contains a large amount of arithmetic calculations. (In programs 1 and 2, examples of system output are not underlined.)

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.+((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,'
        1,'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    Symbol Table
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
```

Symbol Table (Cont'd.)

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

Statement Number Table

EXAMPLE 1(A)

```
      ↑
      *READY
      ↑
      PDP-8 4-K FORTRAN TEST
       0.255323E+1    -0.825572E+1
      !
      .DIAG
      CURRENT LOCATION COUNTER AT 6247
```

EXAMPLE 1(B)

```
      .FOSL
      *IN-S:BIN
      *
      *OPT-
      *READY
      ↑
      PDP-8 4-K FORTRAN TEST
      ERROR 05            (CTRL/C typed here)
      .DIAG
      CURRENT LOCATION COUNTER AT 5407
```

EXAMPLE 1(C)

```
.FOSL
*IN-S:BIN
*
*OPT-
*READY
↑
PDP-8 4-K FORTRAN TEST     (CTRL/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. Refer to the statement number table (top of page 3-29), and note 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. Refer to the statement number table again and 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 CTRL/C was typed. 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. Refer to the statement number table and note 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. As shown in the statement number table, which is typed afterwards, 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
                    (CTRL/C typed here)
.DIAG
OPERAND STACK OVERFLOW
CURRENT LOCATION COUNTER AT 5231
.FORT
*OUT-
*
*IN-S:SRC
*
↑
                    (CTRL/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
*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
*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
*ST= )                           Return to Monitor after loading
↑↑                               Operating system loader is loaded
.SAVE FOSL!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
*ST= )                           Return to Monitor after loading
↑↑                               Interpretive and arithmetic package is
                                 loaded
.SAVE .OS.!0-5177;0 )            and saved on the system device
.LOAD )                          Call Loader
*IN-R: )                         Input to be from high-speed reader
*                                Input device is valid
*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 )                     ASCII input to be from system device
*                                Input device is valid
↑                                CTRL/C was typed. 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        Statement number table (typed by
                        Symbolprint)
0300        5771
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 (CTRL/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 (CTRL/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      (CTRL/C typed here)
.
```

3.5 DDT-D

Dynamic Debugging Technique for the Disk/DECtape System (DDT-D) provides on-line checking, testing, and altering of object programs through the terminal keyboard. On-line debugging allows program checking at the computer, control of program execution, and insertion of corrections or changes while the program is running.

When using DDT-D, it is important to have a listing of the program and its symbols which can be updated as corrections and changes are made to the program. Variables and tags may be referred to by their symbolic names or by their octal values.

Refer to Chapter 5 of Introduction to Programming for information on the operation of DDT-D.

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

DDT Proper	A slightly modified version of DDT-8 (Low), occupies core location 200-4577 and the three breakpoint locations.
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.
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.

1. Three breakpoints (as opposed to only one in DDT-8)
2. No punching (program may be output on the system device)
3. No switch options (user direction is via keyboard)
4. No halts (continues when user types CTRL/P)

Variations in commands from DDT-8 follow. ([represents the ALT MODE character)

[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
CTRL/P	Continue (DDT prints ↑ to indicate that it is waiting for CTRL/P)
CTRL/C	Restore user core image and return to Monitor

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.

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

The following subroutines have been added.

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.
ADDMOD	Updates real or virtual core.
DDTB	Updates symbol table pointer, gets value of breakpoint and its contents, prints breakpoint number and a hyphen (-) if a breakpoint, and goes to TRAP or prints nothing and goes to START if breakpoint number = 0.
STOSYM	Updates DDT proper symbol area (DDT proper must be on unprotected disk).
READS and SYMIO	Input-output routines for disk; a failure in either prints S and goes to start of DDT.

The following subroutines have been modified as indicated.

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

The following subroutines have been removed.

PUNWOR
FSTPUN
FUN
PUNCHK
PUNLDR
WHICH
CHKSUM

From the terminal keyboard, the user program can be automatically stopped 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, Type the absolute address or symbolic tag of the location where the program is to stop, the ALT MODE key, the B key, and then the breakpoint number. For example,

3400[B1 (absolute address, ALTMODE, B,1)

HERE[B2 (symbolic tag, ALTMODE, B, 2)

Locations 3, 4, and 5 on page zero are used as the breakpoint locations. The breakpoint locations can be reset to any three contiguous locations on page zero by reassembly of the furnished DDT Driver source Changing the breakpoint locations is done by setting BRKCEL=n, where n is the lowest of the three locations desired.

The following symbols represent certain registers in DDT-D whose contents are available to the user by typing:

[A Accumulator storage (at breakpoints)

[Y Link storage (at breakpoints)

[M Mask used in search

[L Lower limit of search

[U Upper limit of search

Table 3-9 lists the DDT-D commands.

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.
RETURN key	Make modifications, if any.
LINE FEED key	Make modifications, if any, and print the contents of the next sequential location.
=	Print last quantity as an octal integer.
. (period)	Current location.
← (left arrow)	Delete the line currently being typed.
[S	Set 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.
[X	Clear symbol table.

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 (See Appendix D) and uses three locations on page 0 for the breakpoint locations. The binary tape of the DDT driver uses locations 3, 4, 5 and 7200-7577. After loading DDT proper, reserve on the system device a user core image file name .SYM, which should also be assigned to core locations 200-4577. PAL-D also uses SYM to store additional symbol table entries.

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 these two files must be reassigned.

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
*ST= ↵	Return to Monitor after loading
↑↑	DDT proper is loaded
.SAVE .DDT:200-4577;0↵	Saved as a user program (See Appendix D.)
.SAVE .SYM:200-4577;0↵	User core image file also saved as a user program. (See Appendix D.)
.LOAD ↵	Call Loader using Monitor
*IN-R: ↵	Input to be from high-speed reader
*	Loader found input device valid
*ST=7000↵	Transfer to once-only code after loading
↑↑	Driver is loader
S:<.DDT>,S:<.USER>	DDT Loader expects names of 2 input files saved above for use by driver (See Appendix D.)
*IN-S:.DDT,S:.SYM ↵	Inputs to be from DDT proper and user core image files
*	Loader found input files valid (an asterisk is printed for each valid file (device))
*	

.SAVE DDT!7200-7577;7200)	Saved as a system program (See Appendix D.)
.	

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

1. User file too large
2. System device read error
3. No Command Decoder found

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 type DDT in response to Monitor's dot as shown below.

```
.DDT
```

DDT-D can now be used in debugging the program, directing execution and making modifications as described above and in Chapter 6 of Introduction to Programming.

A brief example of using DDT-D follows:

Example:

.LOAD)	Call Loader
*IN-R:)	Input to be from high-speed reader
*	Loader found input device valid
ST=)	Return to Monitor after loading
↑↑	DDT proper is loaded
.SAVE .DDT:200-4577;0)	DDT proper is saved on disk (See Appendix D.)
.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
ST=7000)	Transfer to once-only code after loading
↑↑	Driver is loaded
S:<.DDT>,S:<.SYM>	DDT Loader expects names of 2 input files saved above for use by driver

3.6 DISK MONITOR SYSTEM RESTORE

This program restores a protected portion of the Disk Monitor System after it has been destroyed, or removes all but a protected group of programs from the disk.

RESTORE may be used only in a DISK MONITOR SYSTEM built on a DF32 or RF08 Disk with Disk Monitor version "AF". The program is supplied as an ASCII source on paper tape, and must be assembled using PAL-D. Assembly of the source as it is yields a version of the program suitable for a single 32K disk unit. By making minor changes to the source, a version suitable for a 2, 3, or 4-disk unit configuration, or for the RF08/RS08 with 1 to 4 disk units can be created.

Due to repacking of system tables on INDAC system do not use or run RESTORE or option M of PIP on INDAC systems.

RESTORE

1. reads a fresh image of the Monitor into core 7600-7777.
2. recreates the first Directory Name block and first storage Allocation Map block by using the DN and SAM Backup blocks.
3. clears all other DN and existing SAM blocks.

If RESTORE has been saved on the disk before a crash occurs, it can be called into core and executed by a simple bootstrap which can easily be toggled in. For this purpose RESTORE needs to be saved on the disk in a known block; preferably, it should be saved as the first program on the disk (after the Monitor, Loader, and Command Decoder). Normally RESTORE is used in conjunction with the P (Protect) and M (MOVE DN and SAM to Backup blocks) options of PIP, and with the write-lock switches of the unit.

If the user has 2, 3, or 4 unit DF32 disk he should edit, using the PDP-8 Editor, the RESTORE source tape which is supplied in order to produce a version of the source suitable to his configuration. The original version is suitable only for the user having a single unit disk. The editing required consists of removing slashes which precede certain additional lines of code which are already in the source. The lines which should be edited are clearly indicated in the source itself.

3.6.1 Assemble and Save RESTORE

Assemble the proper version of the RESTORE source tape using the PAL-D Assembler to produce a binary paper tape of RESTORE.

Using the Disk Monitor System Builder build a new system on disk.

Now load the RESTORE binary paper tape into core in the usual fashion.

Save RESTORE on the disk as follows:

```
SAVE REST!200;200
```

*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 D.)
.DDT ↵	Call DDT using Monitor
3400/AND 0007 IAC ↵ 3401/AND JMP 3400↵	Examine contents of location 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
.	CTRL/C was typed here

NOTE

User Symbol Table extends from location 400 to 1574. This gives a total of 160 decimal symbols and may be cleared by typing [X.

3.6.2 Operating Procedures

Load and save PIP in the usual manner.

Load and save other programs to be placed in the protected system. These programs should not require a total of more than 123 (octal) disk blocks. This keeps all programs to be protected in the lower half of disk unit 0.

Duplicate the directory with the M-option of PIP as follows:

```
*OPT-M
*IN-S)
```

This copies the existing state of the first Directory Name block and the first Storage Allocation MAP block into the DN and SAM backup area. This backup area is in the lower half of disk unit 0. At this point Directory Name blocks 2 and 3 and any existing additional blocks are still clear.

Protect the monitor with the P-option (see section 3.1.2) of PIP as follows:

```
.PIP)
*OPT-P
*IN-S:)
```

This marks all unused blocks in the lower half of disk unit 0 as restricted, so this area may be write locked.

Although the M and P options are generally used together as shown in the above example, either may be used separately or they may be used in reverse order. The important facts to remember are:

1. The M option must always be executed before the RESTORE program can be run.

2. The P option must always be executed before the disk monitor can run with the lower 16K of the disk write-locked.

3. That portion of the disk reserved by the P option can be recovered only by using the RESTORE program and only when the M option was executed before the P option.

4. If the user file .SYM is to be used by the PAL-D assembler or by DDT with the lower 16K of the disk write-locked, it must be created (with the SAVE command) after the P option has been executed.

Write-lock the lower half of disk unit 0 by putting Disk Unit 0 Switch ON and the Lower 16K write-lock switch ON.

At any future time when the system is still running, all programs except the protected programs and data can be deleted from the disk by typing

```
.REST
```

When the Monitor responds with a carriage return/line feed and dot (.), the protected system has been restored.

3.6.3 Bootstrap Sequence

If at any time the unlocked portion of the disk including the DN and SAM blocks is destroyed, the protected system may be restored by toggling in the following bootstrap:

```
0171/5427    (Disk address of disk block 26 where RESTORE will
             reside if it is the first program saved after the
             system is built. The formula is BLKNO*201(8)+1)

0172/1171    (TAD disk address)

0173/6603    (DMAR disk read)

0174/6622    (DFSC skip on completion)

0175/5174    (JMP .-1)

0176/6621    (DFSE skip if no error)

0177/7402    (HLT on error)

7750/7600    (Data Break Word Count)

7751/0177    (Data Break Current Address)
```

START the bootstrap at 0172. This bootstrap reads RESTORE into core and automatically executes it.

NOTE

RESTORE halts on any disk error.

APPENDIX A

SYSTEM GENERATION

This appendix describes the creation of a Disk System (Disk Monitor and system programs) on an empty disk.

The steps involved in system generation are as follows.

1. Toggling in the Readin Mode (RIM) Loader.
2. Loading the Binary (BIN) Loader.
3. Loading and executing Disk System Builder to create Monitor.
4. Loading and saving any system programs.

A.1 TOGGLING IN THE 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 Chapter 6 and Appendix E-1 of Introduction to Programming.

A.2 LOADING THE 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 System Builder. The procedure for loading BIN is as follows.

1. Check that the RIM Loader is in core.

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

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

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

5. Press START. Tape should begin reading (if it does not, check that the SING INST and SING STEP switches are off 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.

6. After paper tape reads in, Press STOP on console and the BIN Loader will be in core.

A detailed description of BIN and its use can be found in Appendix E-1 of Introduction to Programming.

A.3 LOADING AND EXECUTING DISK SYSTEM BUILDER

Next, the Disk System Builder program, in binary format on paper tape, is loaded by the Binary Loader. System Builder (DEC-D8-SBAF) is compatible only with versions DEC-D8-PDAD and DEC-D8-PDZE of the Peripheral Interchange Program. Loading procedures are as follows.

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

2. 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.

   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 Teletype to line, and set reader to START.

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

4. 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).

5. 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 and the REMOTE switch to REMOTE.

NOTE

To rebuild the monitor without destroying previously stored programs on the system device make the following patch.

LOC	OLD	NEW
375	0400	7600

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

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

BUILDER Dialogue	Explanation
*SIZE OF CORE 8	User enters size of core K in (4,8,12,16,20,24,28, or 32).
*HIGH SPEED PAPER TAPE? Y	User answers Y(yes) or N(no).
*RF08? N	User answers Y(yes) or N(no). If Y, next question is omitted.
*DF32? Y	User answers Y or N. If N, the Builder assumes the user wishes to build a DECtape system on unit 0. In this case the next 2 questions are omitted.
*NUMBER OF DISK UNITS? 2	User types number of physical disk units on his machine 1,2,3 or 4 in addition to disk. Multiples of 32K for DF32, 256K for RF08 units.
*DECTAPE? Y	User types Y if he has DECtape, (in addition to his disk) otherwise N. Monitor creation is 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:

1. If disk, start over at Paragraph A.2; there may be a hardware problem.

2. If DECtape, try a new DECtape and start at Paragraph A.2. or, rewrite the timing and mark tracks and start at Paragraph 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. Figure A-1 illustrates the Binary Loader procedures with the low - speed reader. An example of the saving procedure 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.
*ST=7600)	Return to Monitor after loading.
↑↑	After each up-arrow typeout, user types CTRL/P to continue (also must press CONTinue on console if Teletype reader is being used).
.SAVE PIP!0-5177; 1000)	Save 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 D.

.

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

Binary Loader Procedures (Low Speed, or ASR, Reader)

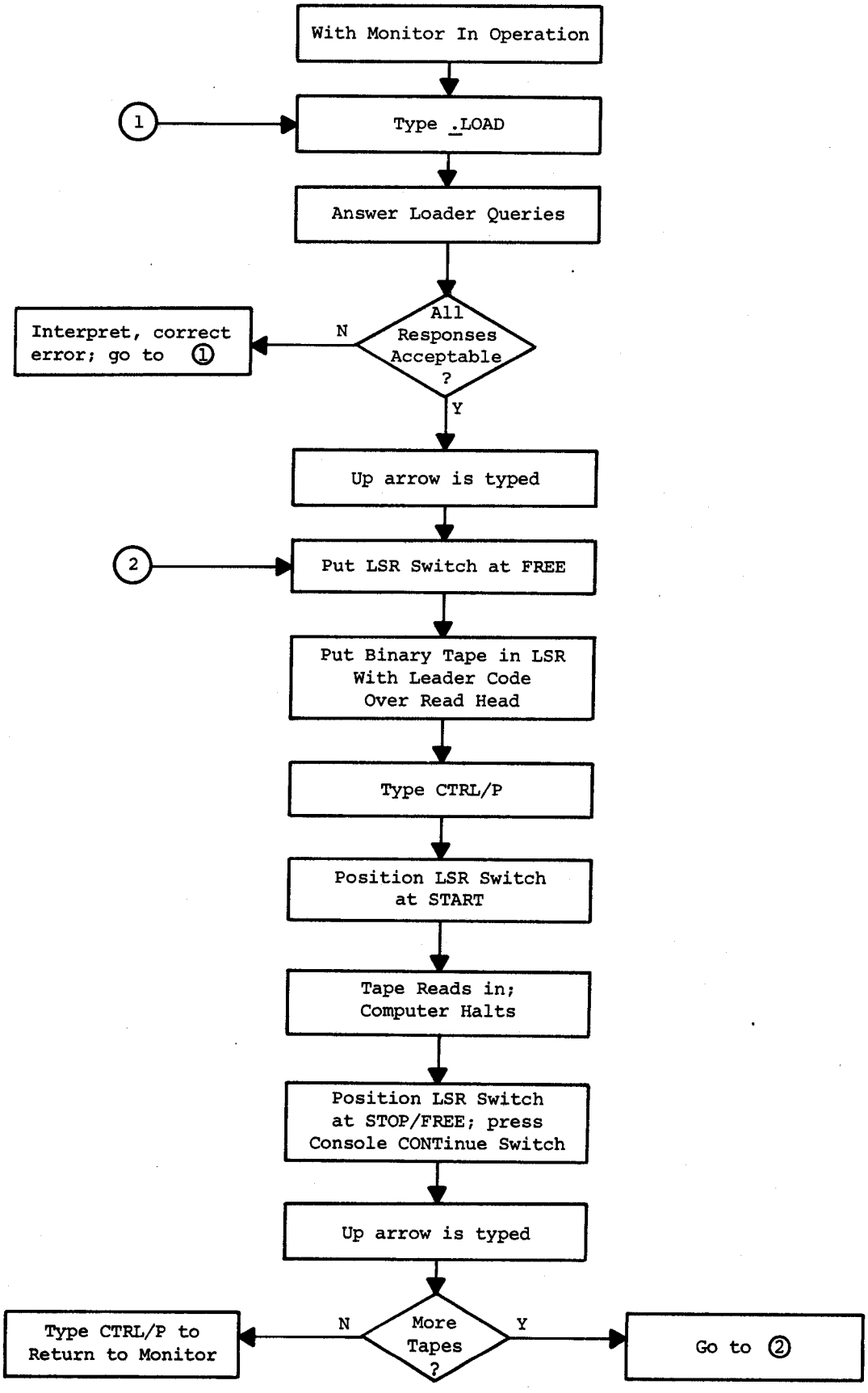


Figure A-1 Disk Loader/Paper Tape Flowchart

APPENDIX B

SYSTEM FORMATS

This appendix contains the following information.

1. 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

2. Data Structure

   Source File (ASCII)

   Binary File (BINARY, FTC BIN)

   Saved Files (SYS, USER)

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

4. Monitor Core Usage Diagrams

5. Monitor Flow Chart (Figure B-11)

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.

An RF/RS08 disk unit contains 256K 13-bit (12 bits-plus-parity) words of storage capacity. Each physical unit is divided into two "logical units", each equal in size to four DF/DS32 disk units. Table B-1 lists the capacities of the various devices which may be part of the system.

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. Each logical unit has its own directory, utilizing three Directory Name (DN) blocks and four Storage Allocation Map (SAM) blocks.

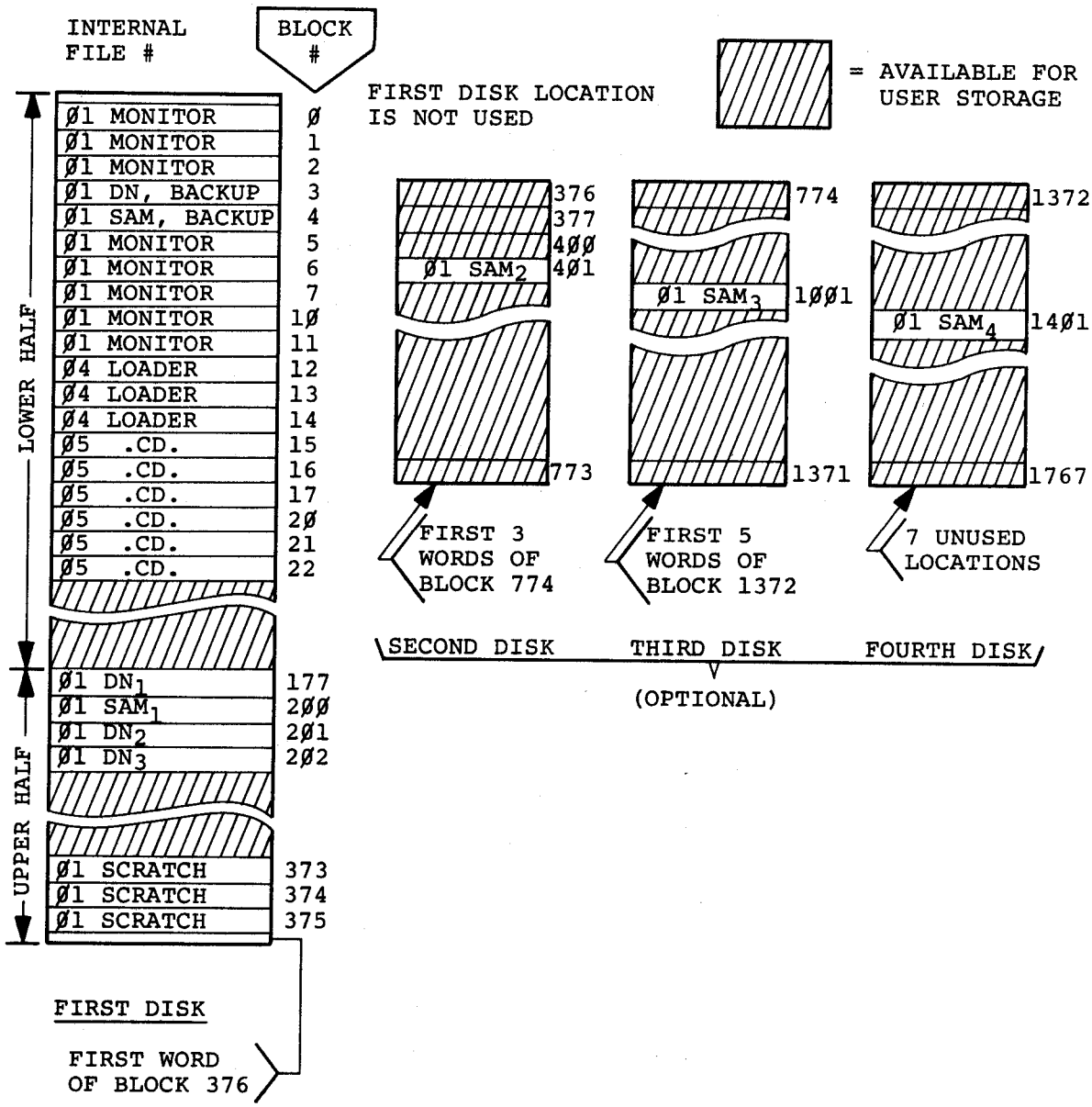


Figure B-1 Disk Storage Layout in the Disk Monitor System

NOTE

The odd accumulations at the end of each disk are due to the fact that dividing the disk into 129-word blocks leaves a remainder of two words. On the first disk only, the first location is unused. This allows the write lock boundary to fall between two blocks.

Blocks 1770-1777 do not exist. Do not attempt to use them.

In systems with less than four disks, SAM block entries corresponding to nonexistent blocks are set to 01 (the Monitor file number).

All blocks which are available but unassigned are given an internal file number of 0.

There may be 60(decimal) user files in the total system.

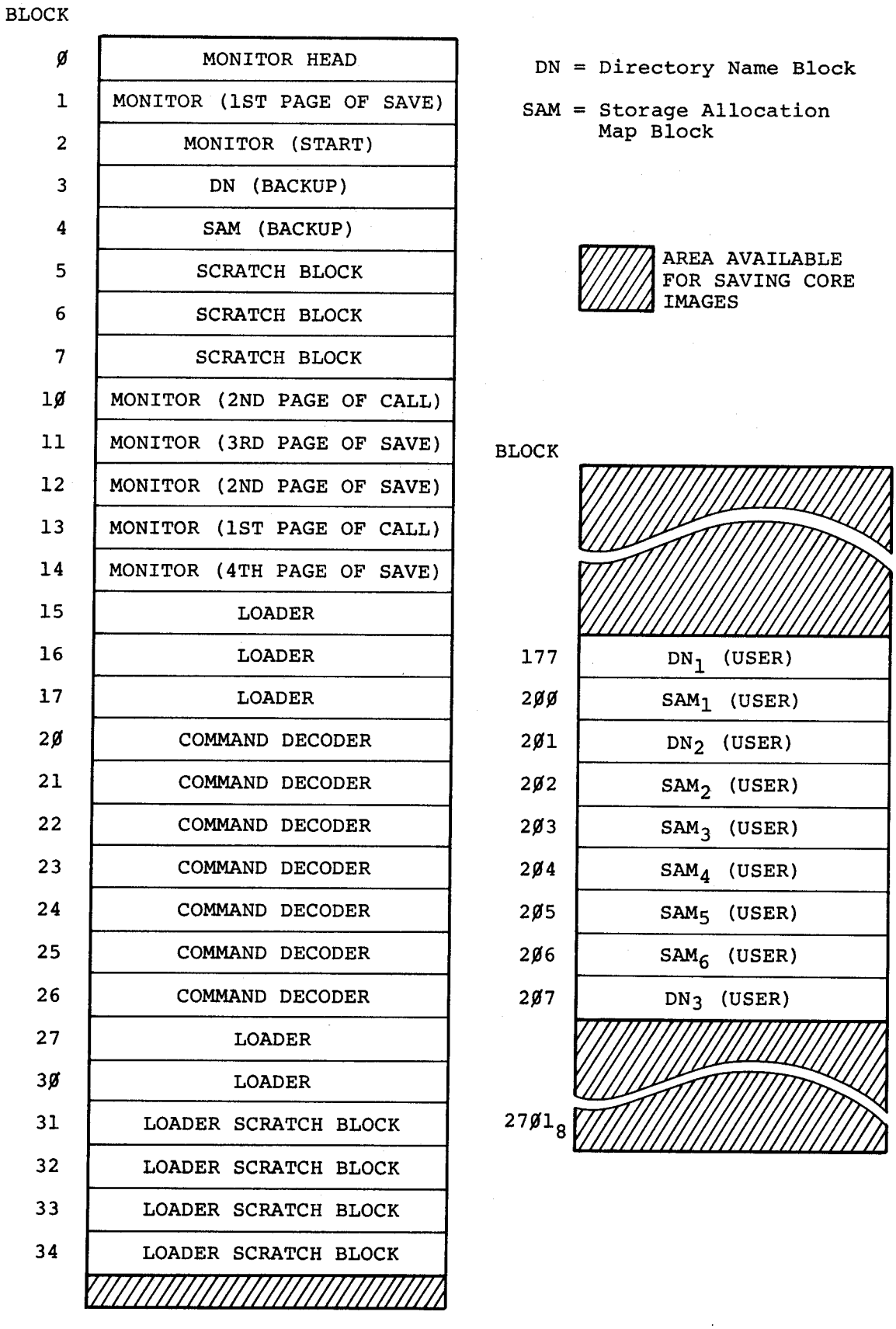


Figure B-2 DECtape Storage Layout (Revised)

Table B-1
System Device and Core Capacities

Unit	Words	Highest Page (Block) Number (1st Page = 0)	
1 DISK(DF32)	32,768	(375(8))	
2 DISKS(DF32)	65,536	(773(8))	
3 DISKS(DF32)	93,303	(1371(8))	
4 DISKS(DF32)	131,072	(1767(8))	
1 DECTAPE	190,146	(2701(8))	
1 RF08 DISK	262,144	S0:1767(8)	S1:1767(8)
2 RF08 DISKS	524,288	S2:1767(8)	S3:1767(8)
3 RF08 DISKS	786,432	S4:1767(8)	S5:1767(8)
4 RF08 DISKS	1,048,432	S6:1767(8)	S7:1767(8)

NOTE

Each RF08 disk is divided into two 128K logical units, Each logical unit is structured as if it were four DF32's.

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.

DN1 - Contains entries for internal file numbers 01 through 31(octal)(25(decimal)) and a link to DN2.

DN2 - Contains entries for internal file numbers 32 through 62(octal)(50(decimal)) and a link to DN3.

DN3 - Contains entries for internal file numbers 63 through 77(octal)(63(decimal)) 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(octal)). 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.

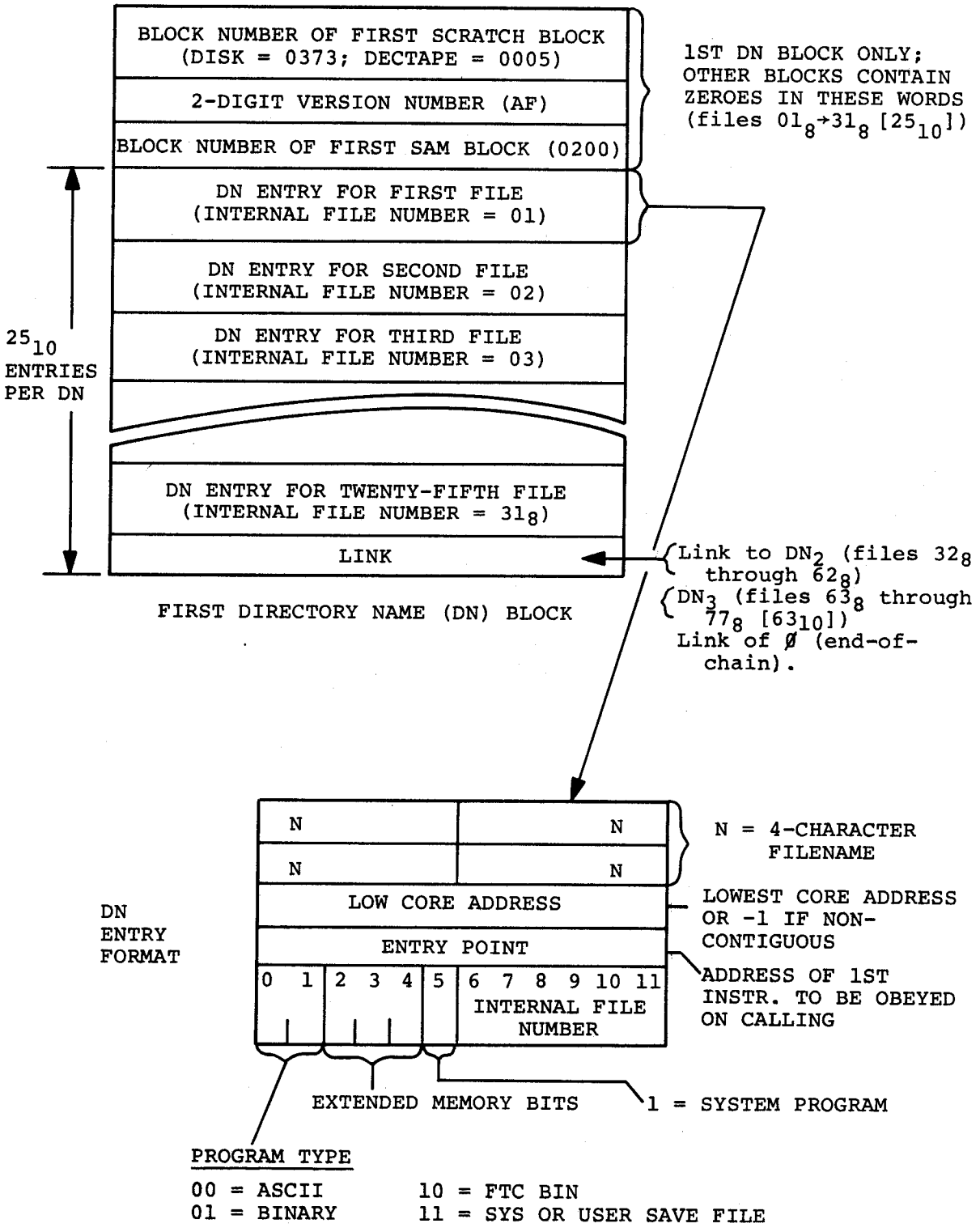


Figure B-3 Directory Name (DN) Block Format

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(octal)-block area. (See Figure B-4.)

SAM(1) contains the map for blocks 0 through 377(octal) and a link to SAM(2).

SAM(2) contains the map for blocks 400 through 777(octal) and a link to SAM(3).

SAM(3) contains the map for blocks 1000 through 1377(octal) and a link to SAM(4).

SAM(4) contains the map for blocks 1400 through 1777(octal) and either and end-of-chain link of 0 (if disk) or a link to SAM(5) (if DECtape).

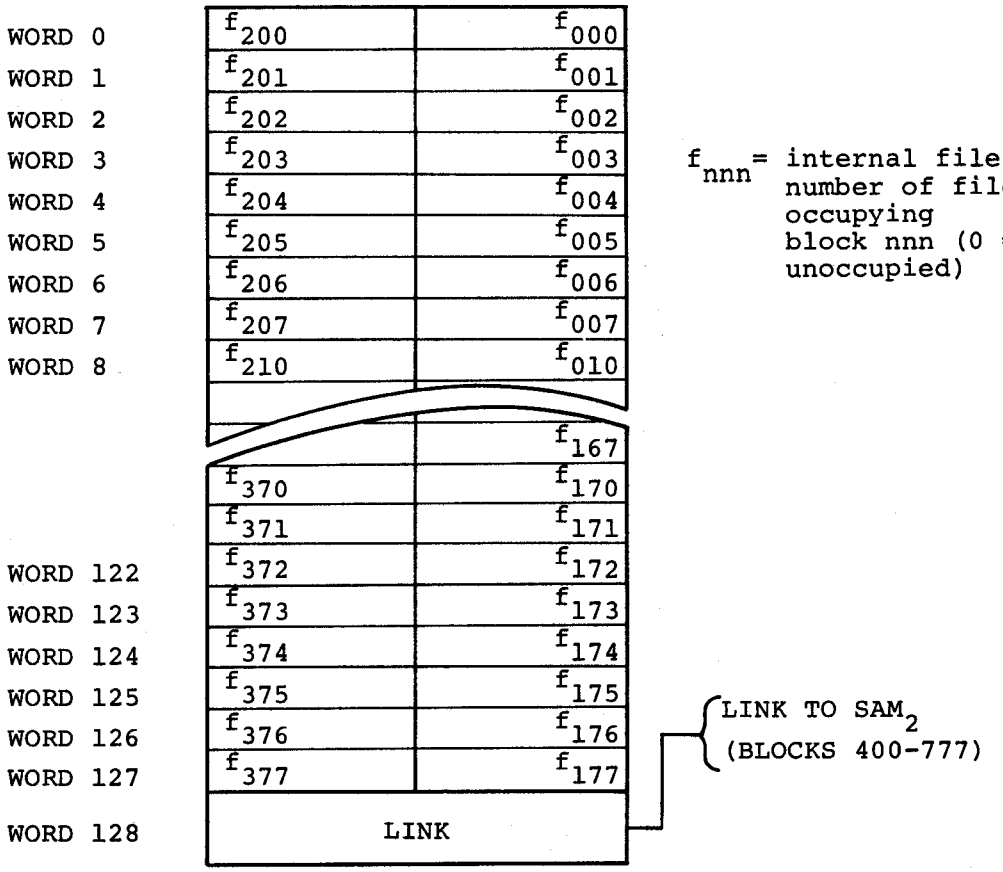
The next two SAM blocks are present only if a DECtape is the system device.

SAM(5) contains the map for blocks 2000 through 2377(octal) and a link to SAM(6).

SAM(6) contains the map for blocks 2400 through 2701(octal) 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. On an RF08 disk each pseudo-unit (S0-S7) has its own set of three DN blocks and four SAM blocks.

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



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

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 as 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 is contained within the rightmost eight bits of word 1, frame 2 is contained within the rightmost eight bits of word 2, and frame 3 is contained within 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
```

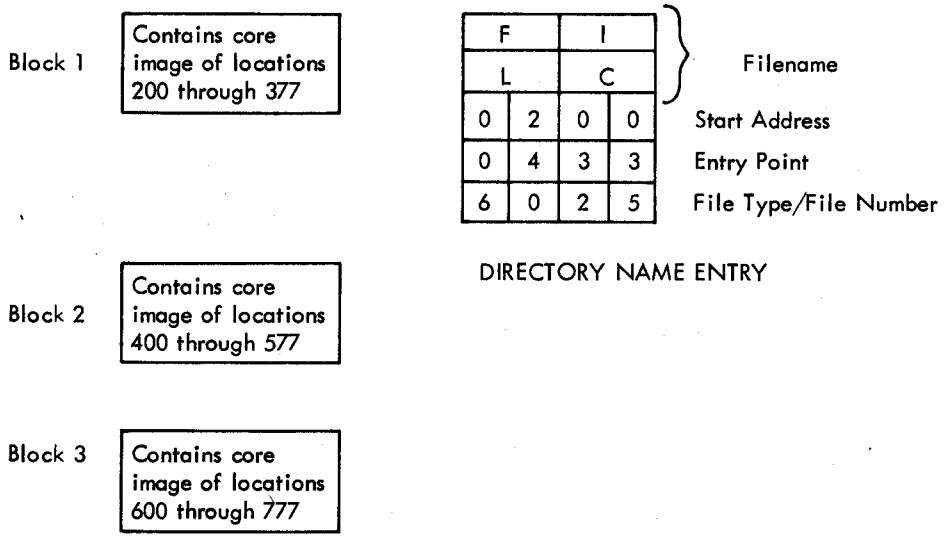


Figure B-5
Contiguous-Page Save File Format

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 stored 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.

```
SAVE FILN: 0,400,1000;433
```

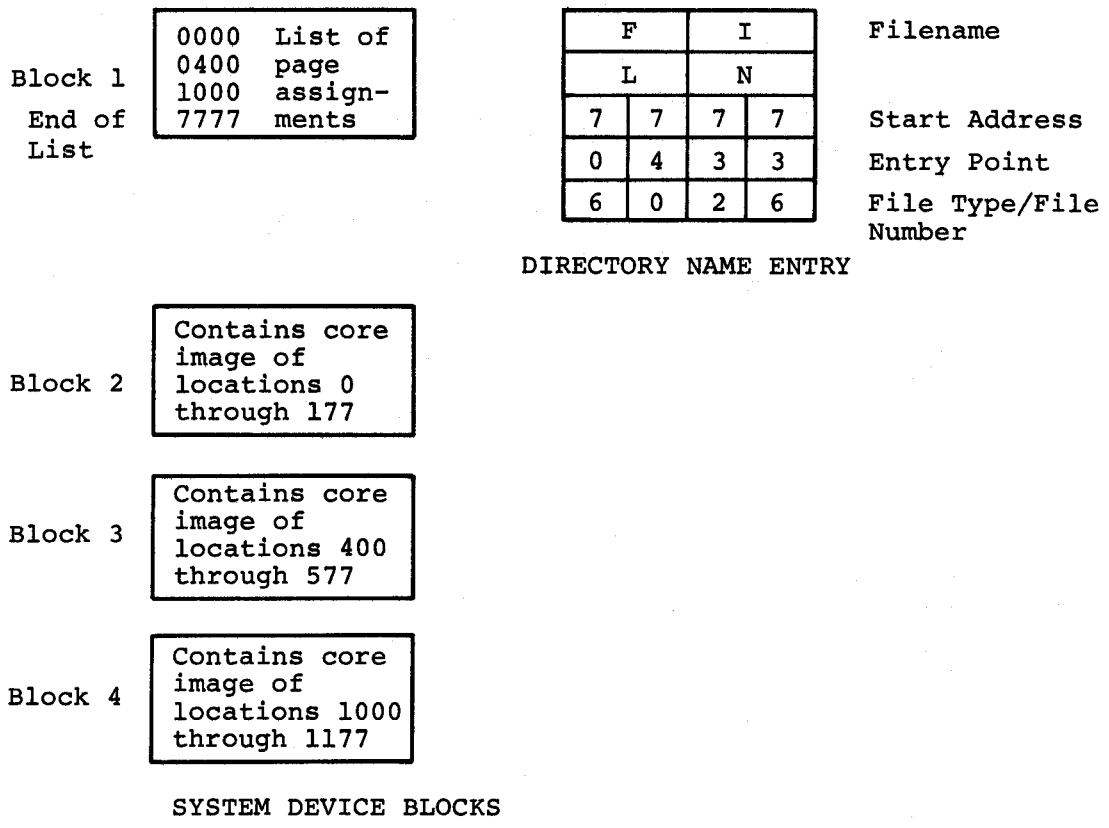


Figure B-6
Noncontiguous-Page Save Format

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=0121                    121 free blocks remain

PALD.SYS   (0)   0030
EDIT.SYS   (0)   0016
LOAD.SYS   (0)   0011
.CD..SYS   (0)   0007
PIP .SYS   (0)   0025
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
```

Number of blocks used

Field number

Extension name

Filename

Figure B-7
Sample PIP Directory Listing

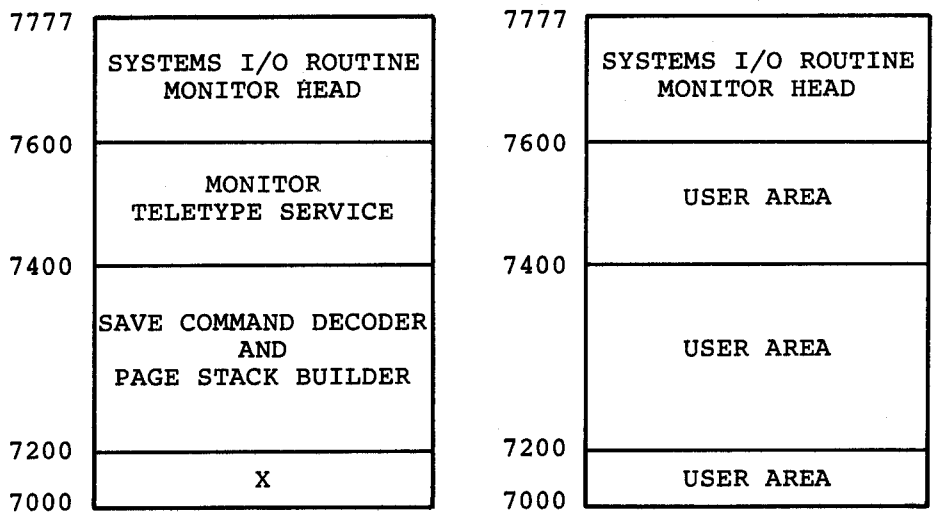
B.4 MONITOR CORE USAGE DIAGRAMS

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)

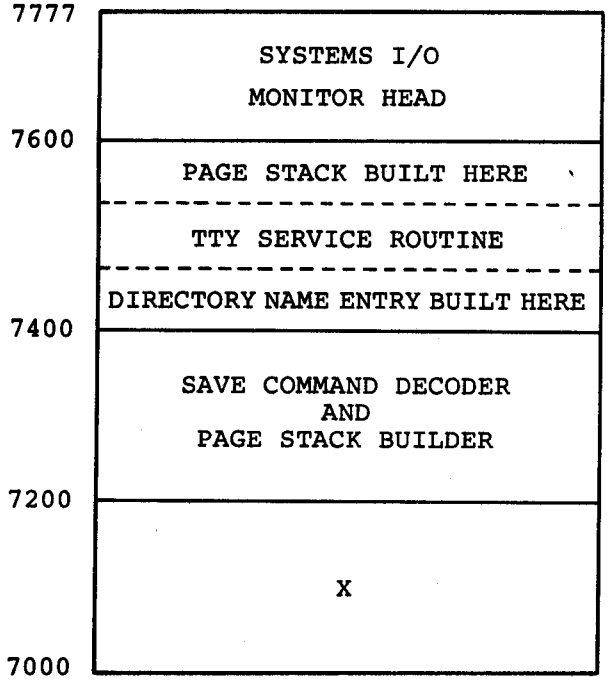
c. CALL Command Processing (Figure B-10)



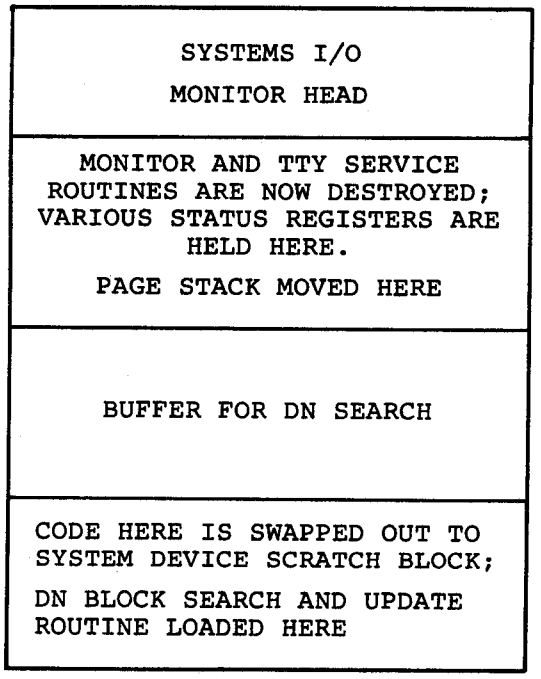
(a) Monitor-Time Core Usage (b) User-Time Core Usage

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

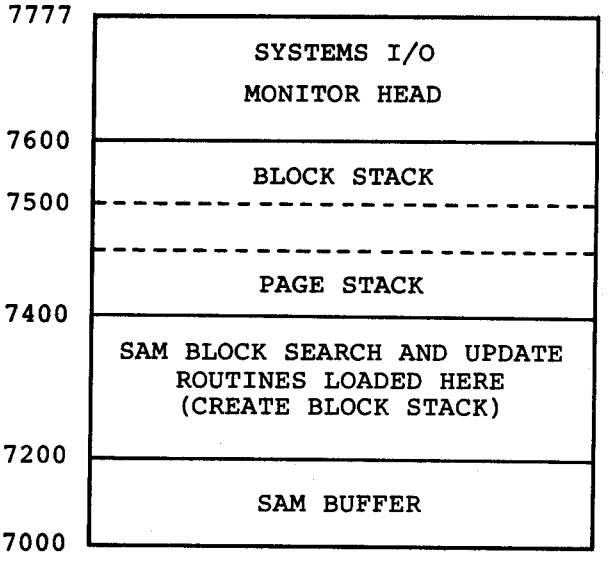
.SAVE filename:core-specifications,...:entry-point



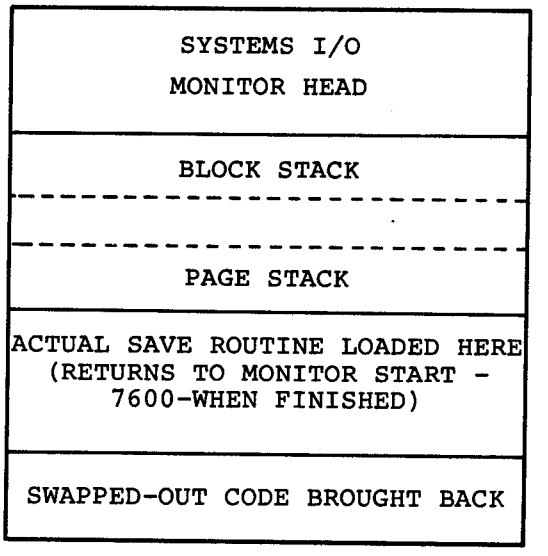
(a) "SAVE filename:" Processing



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



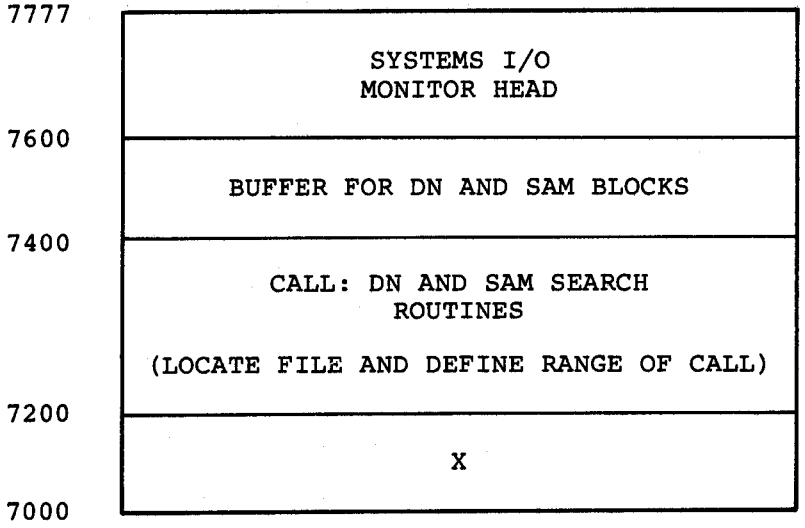
(c) SAM Search



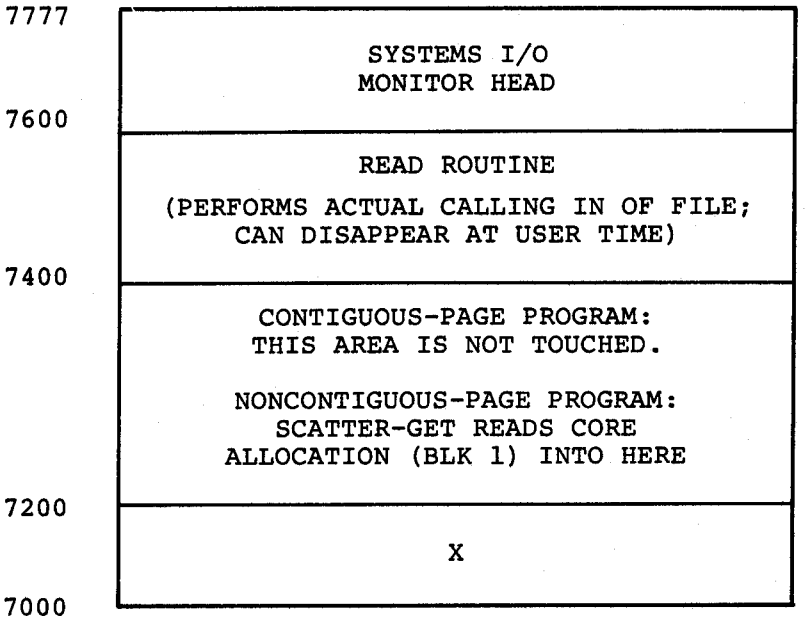
(d) Actual Save Time

Figure B-9 Core Usage During SAVE Command Execution

.CALL filename) or .filename)



(a) "CALL filename" Processing



(b) Actual CALL Time

Figure B-10 Core Usage During CALL Command Execution

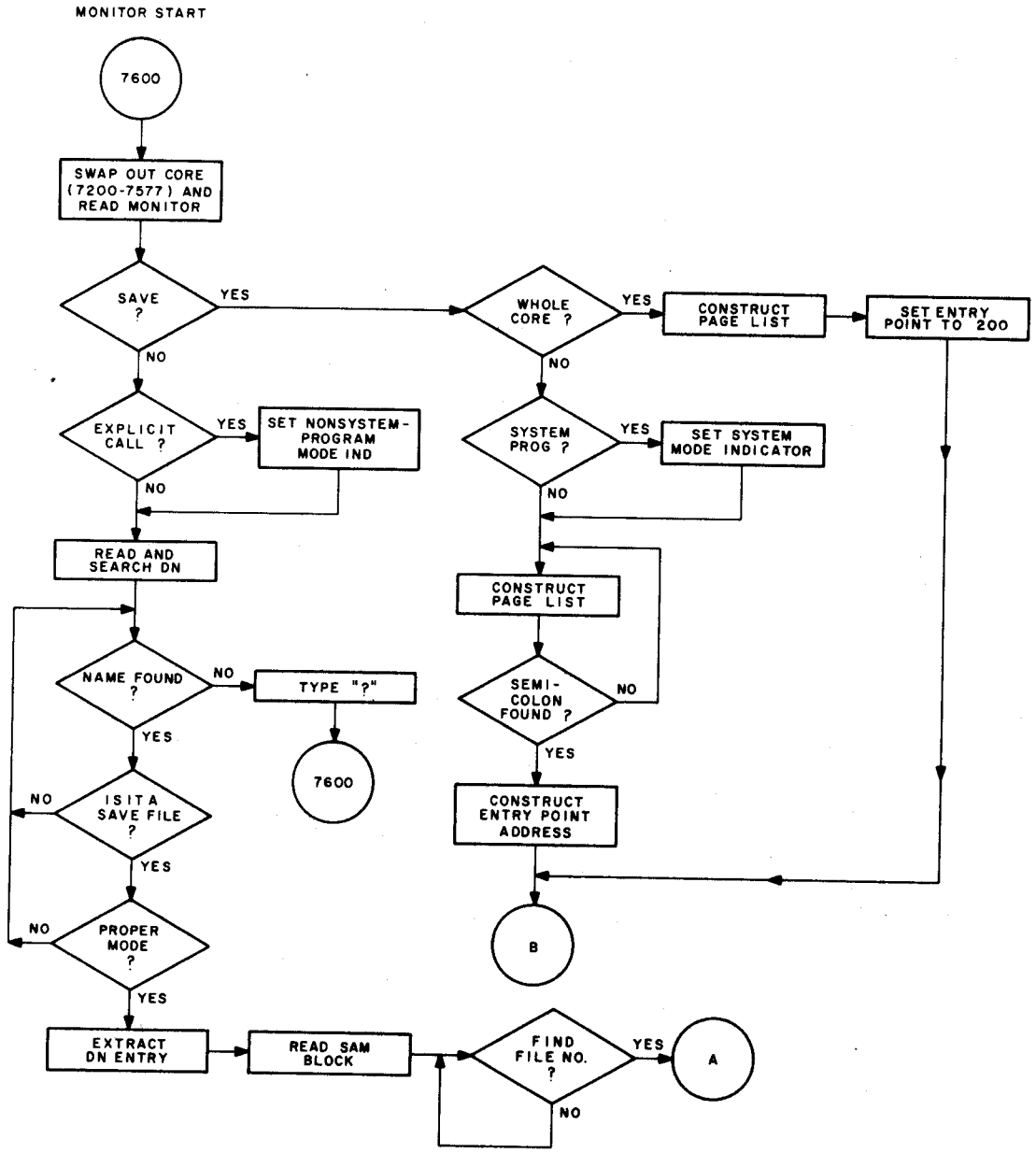


Figure B-11 Monitor Flow Chart (Part 1)

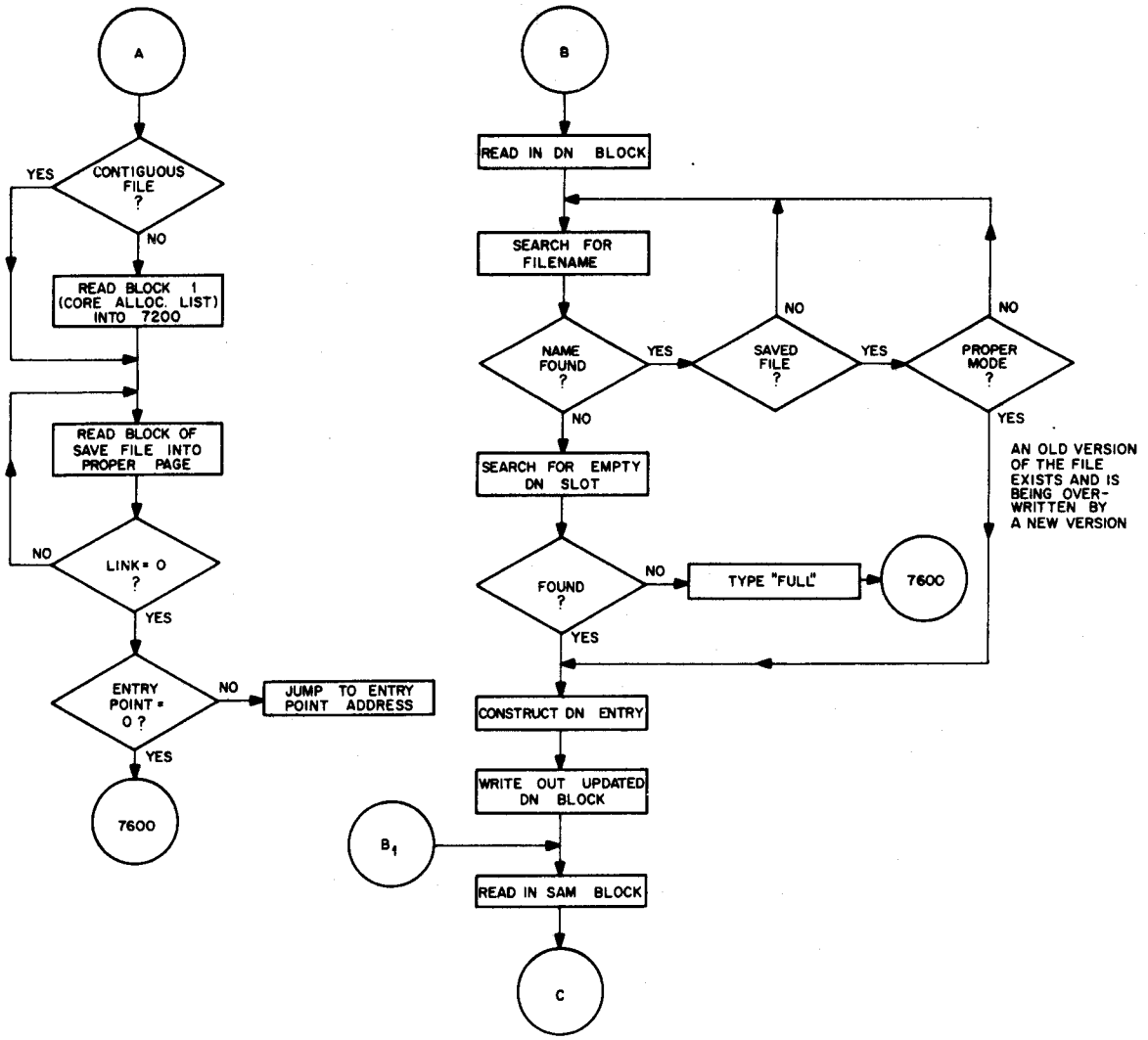


Figure B-11 Monitor Flow Chart (Part 2)

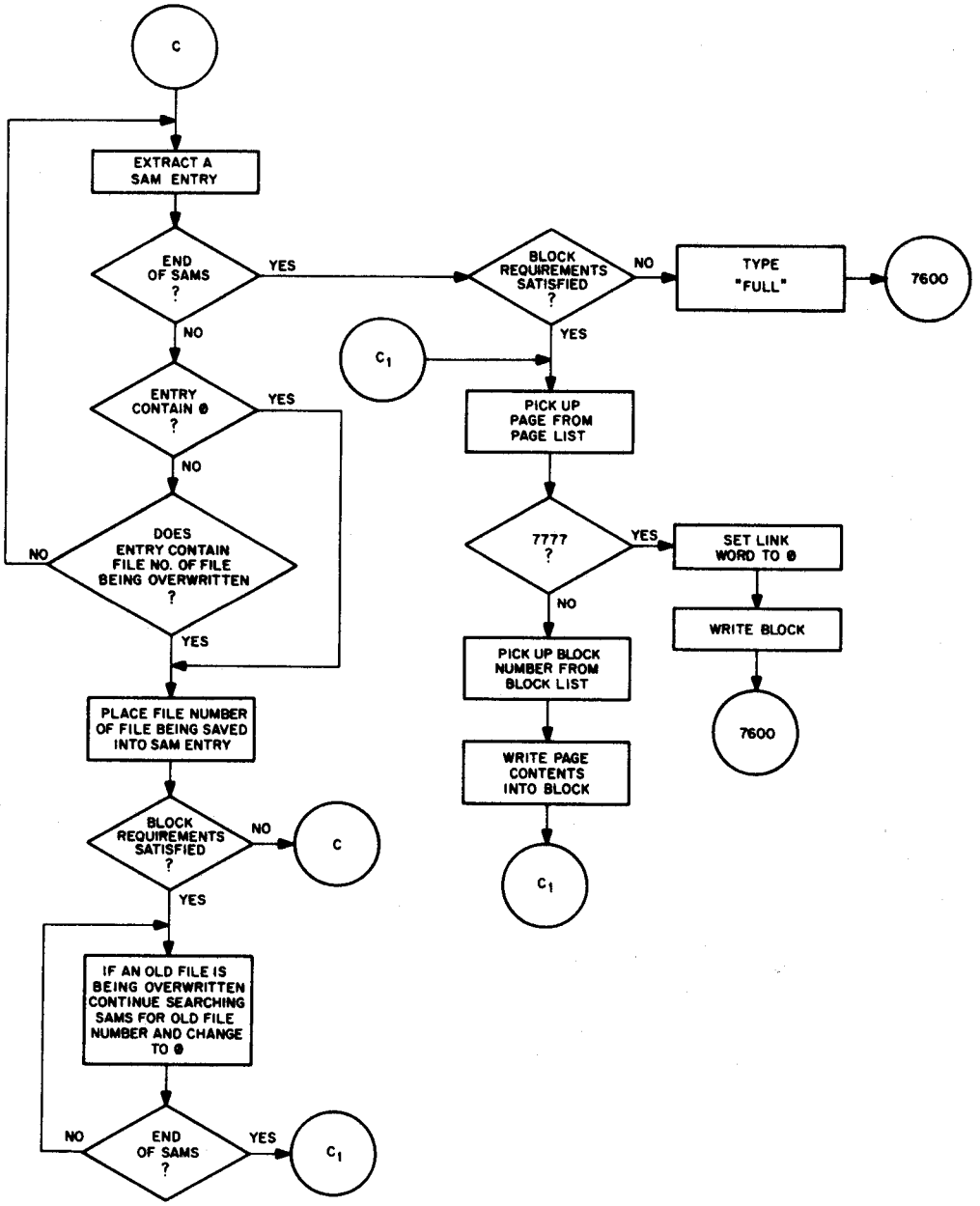


Figure B-11 Monitor Flow Chart (Part 3)

APPENDIX C

COMMAND DECODER

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

Command Decoder user four pages of core (see Figure C-1) and is called in by a system program as follows.

1. The internal file number of Command Decoder (filename=.CD.) is obtained. (05)

2. The starting block of the Command Decoder file is obtained. 15-Disk, 20-DECtape

3. 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.

4. 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 shown in Table C-1.

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.

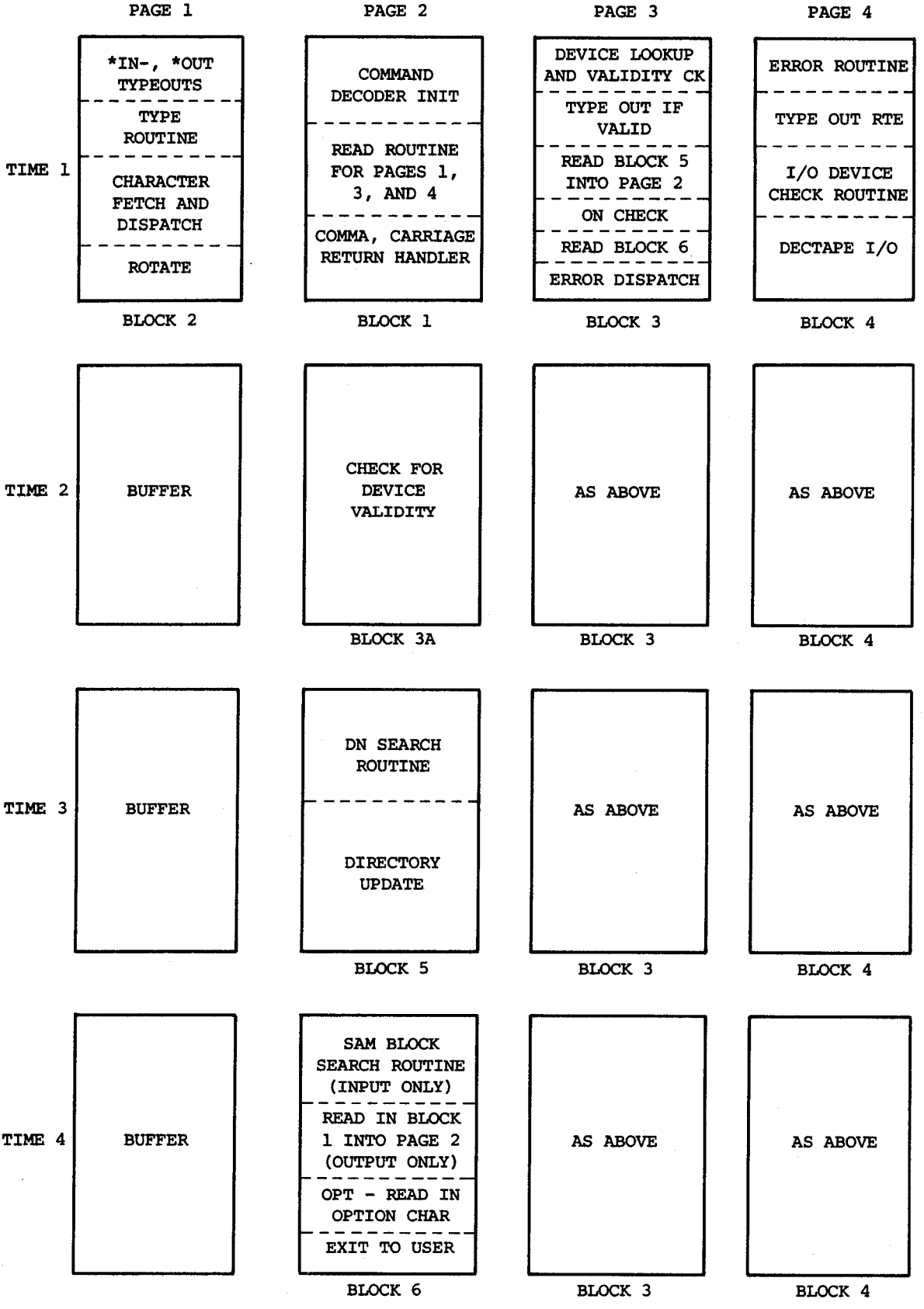


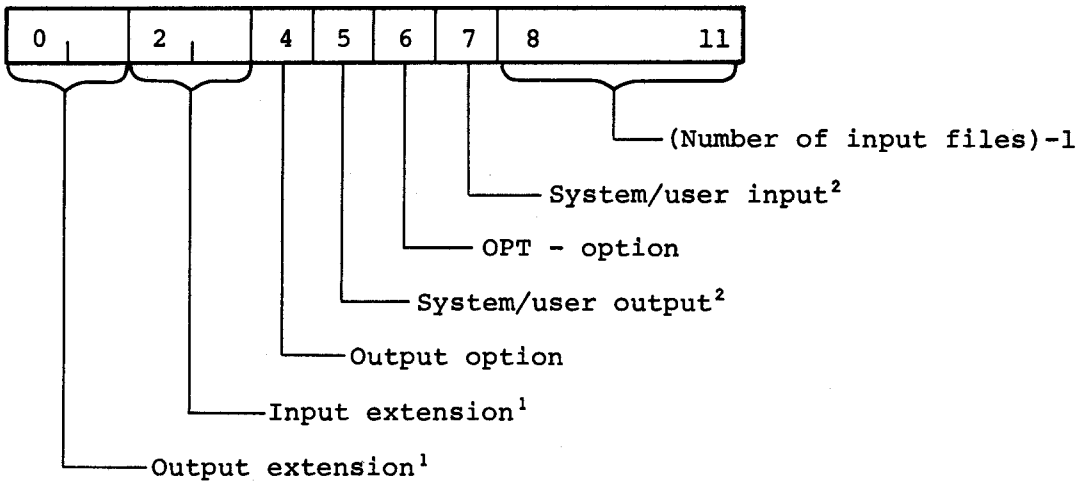
Figure C-1 Command Decoder Core Usage (Revised)

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. (Refer to Figure C-2.)

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).
Bits 2 and 3	Contain the input file extension.
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.



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

[2]1 = System, 0 = User

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

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
```

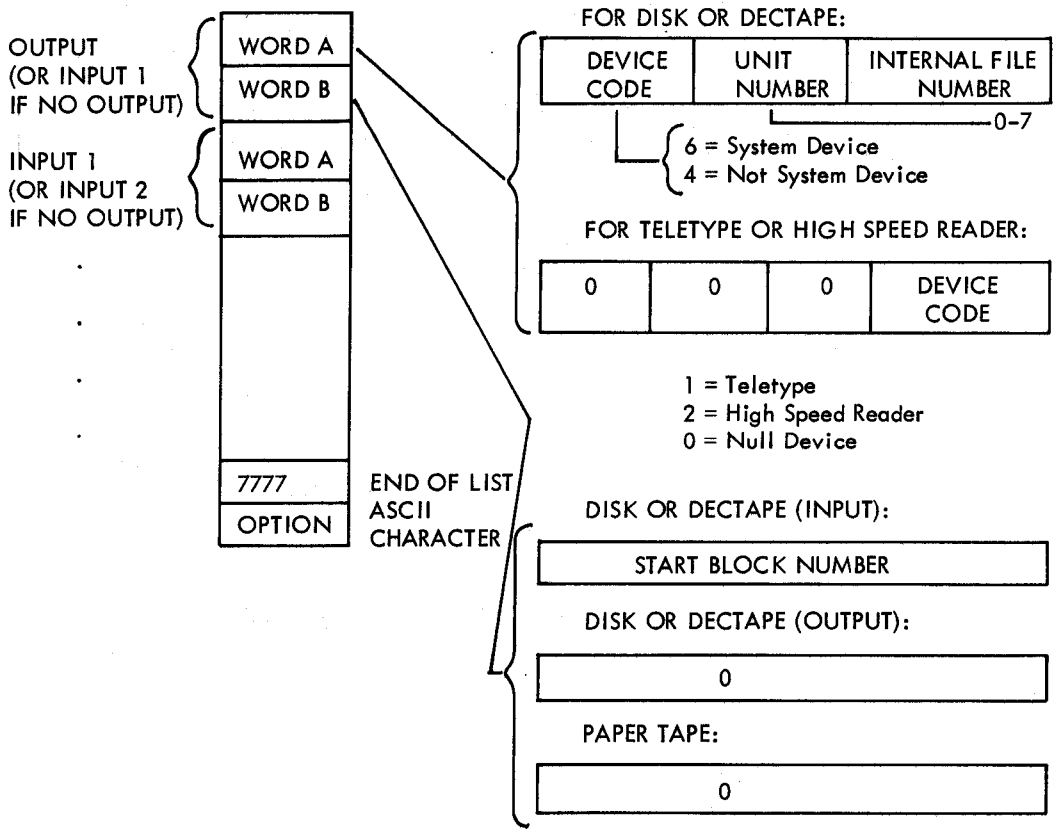


Figure C-2 Output List Produced by Command Decoder

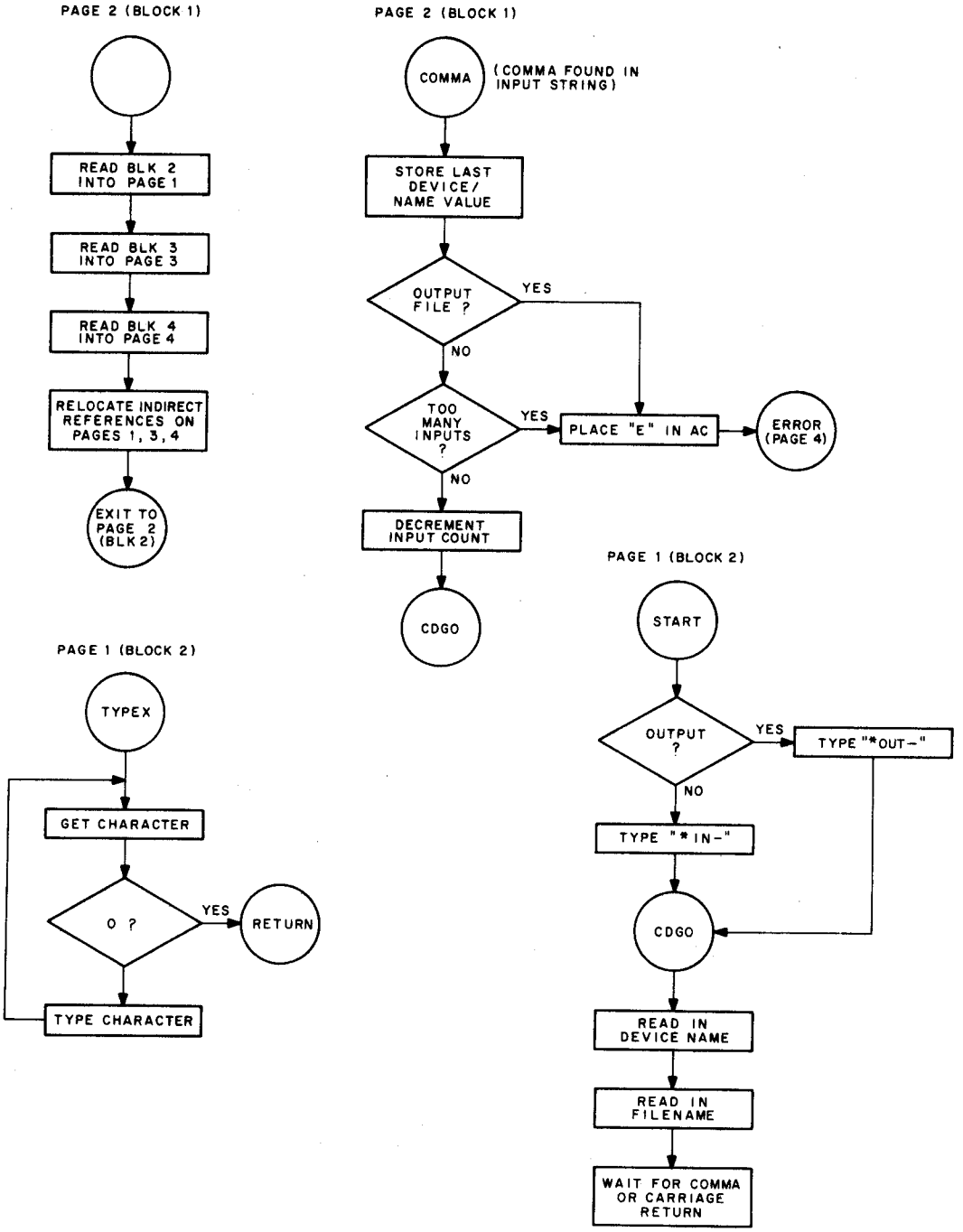


Figure C-3 Command Decoder Flow Chart (Part 1)

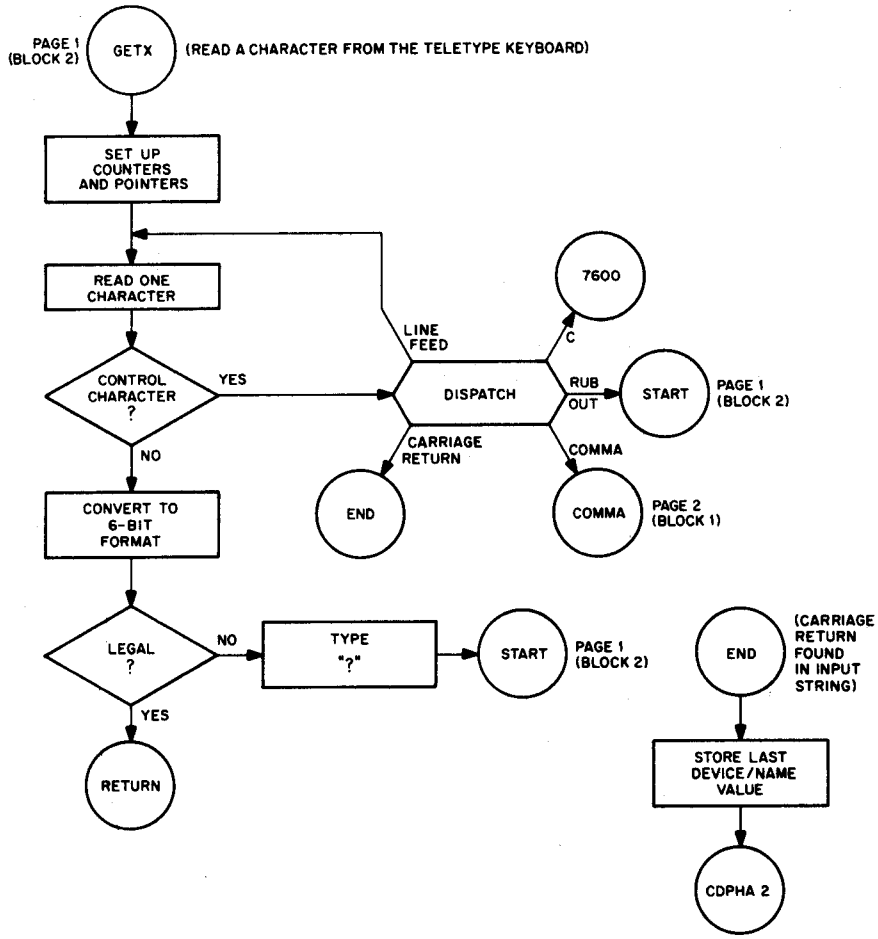


Figure C-3 Command Decoder Flow Chart (Part 2)

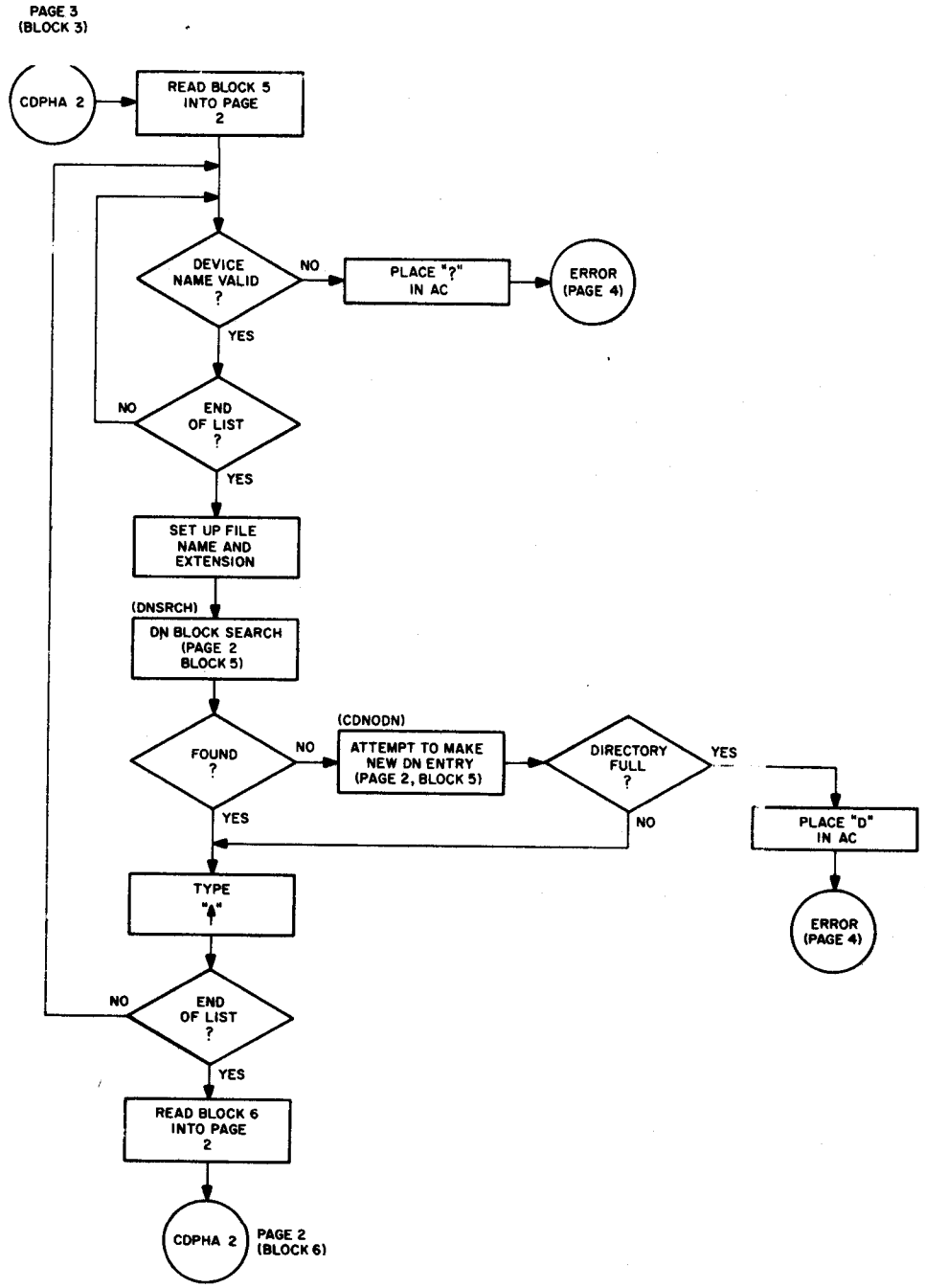


Figure C-3 Command Decoder Flow Chart (Part 3)

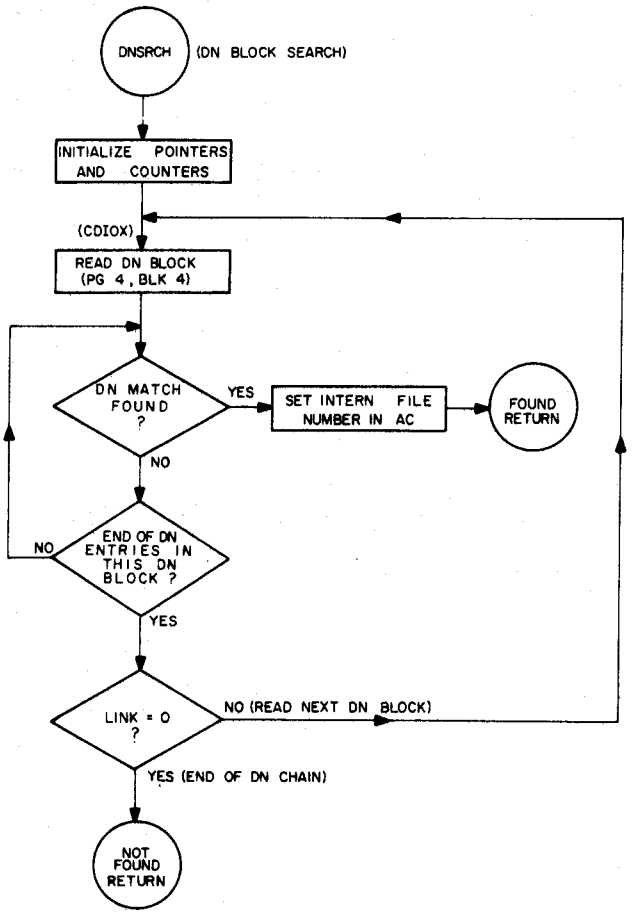


Figure C-3 Command Decoder Flow Chart (Part 4)

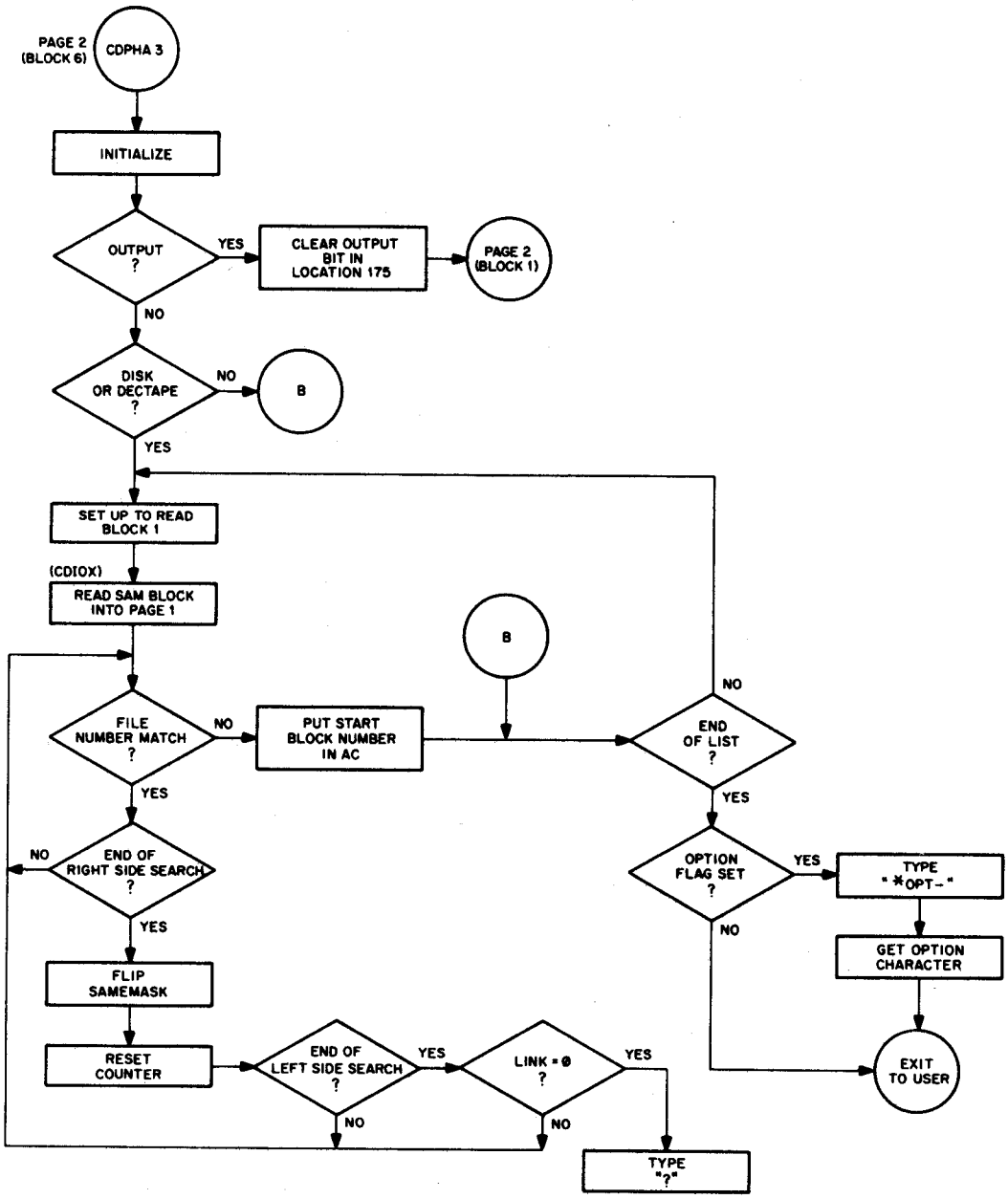


Figure C-3 Command Decoder Flow Chart (Part 5)

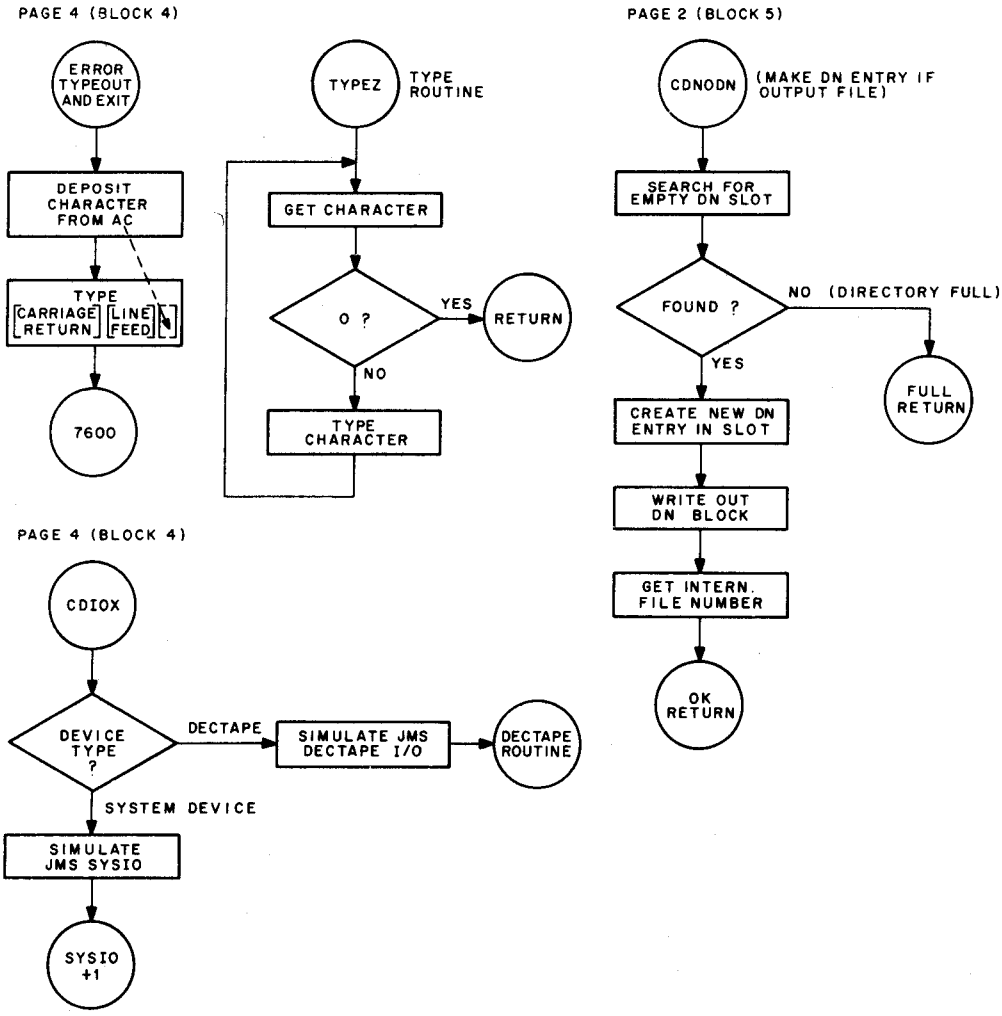


Figure C-3 Command Decoder Flow Chart (Part 6)

APPENDIX D

SYSTEM PROGRAMS

D.1 LOADING PARAMETERS FOR SYSTEM PROGRAMS

Name	Core Limits	Entry Point
PIP	0-5177	1000
EDIT	0-3377	2600
PALD	0-7577	6200
FORT	0-1777	200
.FT.	200-7377	--
STBL	600-777	600
FOSL	0-1777	200
.OS.	0-5177	--
DIAG	200-1177	200
.DDT	200-4577	--
.SYM	200-4577	--
DDT	7200-7577	7200
REST	200-377	200

D.2 SAVE STATISTICS

Name	Save Command String
PIP	SAVE PIP!0-5177;1000
EDIT	SAVE EDIT!0-3377;2600
PALD	SAVE PALD!0-7577;6200
FORT	SAVE FORT!0-1777;200
.FT.	SAVE .FT.!200-7377;0
STBL	SAVE STBL!600;600

Name	Save Command String
FOSL	SAVE FOSL!0-1777;200
.OS.	SAVE .OS.!0-5177;0
DIAG	SAVE DIAG!200-1177;200
REST	SAVE REST!200-377;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 E

I/O PROGRAMMING

E.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).

1. 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.

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

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

E.2 CALLING SEQUENCE FOR BASIC I/O ROUTINE

The basic I/O routine (see Paragraph E.1.1.) is called as shown in Figure E-1. It is called in two ways, as determined by bit 2 of the function word.

1. 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 is 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
```

2. 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).

Calling Sequence	Explanation
JMS I SYSIO	Location SYSIO points to I/O
BLOCK	Block to be accessed
CORE	Low-order core address
LINK	Filled by READ, used by WRITE
ERROR	Error return here
RETURN	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 or RF08 disk
	Bits 6-8	memory field
	Bits 9-11	function: READ = 3; WRITE = 5

Figure E-1 Calling Sequence of System Routine

E.3 GENERALIZED DISK/DECTAPE I/O ROUTINE

The user or system programs may use the generalized I/O routine in Appendix C. The calling sequence to this routine is the same as that used by the basic I/O routine, except for certain restrictions explained below.

The Command Decoder must be called as shown in Examples 1 and 2 in Section C.2. The entry point for the generalized I/O routine is the first location of the Command Decoder plus 603(octal) locations, i.e., in the previously mentioned examples the entry point would be at location 2603.

The generalized I/O routine uses location 0171 on page zero. This location must contain the address which points to the I/O list created by the Command Decoder. If the JMS instruction which calls the routine is at an even numbered location, location 0171 must point to Word B (refer to Appendix C) of a Command Decoder list entry. If the JMS is at an odd numbered location, location 0171 must point to Word A of a Command Decoder list entry. The contents of Word A then determines whether the DECtape or system device will be used.

The contents of the function word in the calling sequence determine whether information is to be read from or written on the selected device and DECtape unit (if applicable) as shown in Figure E-1.

The following examples illustrate the use of the generalized I/O routine. It is assumed that the Command Decoder was called and placed in locations 2000-2777, and that its list begins at location 3000.

```
3000    4312    (Word A) Output device - DECtape #3,
3001    0000    (Word B) internal file #12
3002    6007    (Word A) Input - system device, internal
3003    0065    (Word B) file #7, starting at block 65
3004    7777
3005    0215    Option - carriage return was typed

300     4777    JMS I (2603
301     0305    FUNCTION WORD
302     0160    BLOCK NR
303     7000    BUFFER ADDRESS
304     0161    LINK NR
305     7402    ERROR HALT
171     3001
377     2603    ENTRY POINT
```

The above code writes the contents of locations 7000-7177 onto block 160 of DECtape unit 3 and writes link word of 161.

```
501     4777    JMS I (2603)
502     0003    FUNCTION
503     0065    BLOCK NR
504     7200    BUFFER ADDRESS
505     0000    LINK NR
506     7402    ERROR HALT
171     3002
577     2603    ENTRY POINT
```

The above code reads the contents of block 65 of the system device into locations 7200-7377 and places the link to the next block of this file in location 505.

NOTE

There are two versions of the disk Monitor head (Block 0) for the DF32. One (Version A) is used for all 4K, non-8/S systems and the other (Version B) for everything else. The distinguishing difference between the two is the fact that Version A does only one disk read for every block transfer (i.e., - one disk revolution), compared with two disk reads per block with Version B (one to read the block, one to fetch the link word into memory field 0). This is necessary because of the technique used to fetch the link word in Version A (switching DATA-BREAK pointers on the fly) which would not be used in Version B (the PDP-8/S is too slow and altering the extended memory register on the fly in an 8K machine may lead to transmission errors).

APPENDIX F

VALID I/O DEVICES

The Table below illustrates the valid I/O Device Combinations for the three system devices (DF32 Disk, RF08 Disk, and the DECtape Systems).

Table F-1

Valid IN/OUT Devices for PAL-D, EDIT

(Star indicates valid device designation)

I/O DEV / SYS DEVICE	T:	R:	S:name	SØ:-name	S1:S7:name	DØ:name	D1:-D7:name
DF32	*	*	*	N/A	N/A	N/A	N/A
RF08	*	*	* Same as SØ	*	*	N/A	N/A
DECTAPE	*	*	* Same as DØ	N/A	N/A	*	*

Table F-2

Valid FORTRAN-D INPUT/OUTPUT

	T:	R:	S:	SØ:	S1:-S7:	DØ:	D1:-D7:
DF32	*	*	*	N/A	N/A	N/A	N/A
RF08	*	*	*	*	N/A	N/A	N/A
TAPE	*	*	*	N/A	N/A	*	N/A

Table F-3

VALID I/O DEVICES FOR PIP

I/O DEV / SYS DEV	T:	R:	S:name	SØ:name	S1:-S7:name	DØ:name	D1:-D7:name
DF32	*	*	*	N/A	N/A	*	*
RFØ8	*	*	*[1]	*	*	*	*
DECTAPE	*	*	*	N/A	N/A	*	*

[1]The designation "S:" may not be used in place of "SØ:" when deleting files and listing directories. This is a temporary restriction.

APPENDIX G

PERMANENT SYMBOL TABLES

The following are the most commonly used elements of the PDP-8 instruction set. For that reason they are found in the permanent symbol table within the assemblers. These instructions are already defined within the computer. For additional information on these instructions and for a description of the symbols used when programming other, optional, I/O devices, see the SMALL COMPUTER HANDBOOK, available from the DEC Software Distribution Center.

Times shown below are representative of the PDP-8/E.

G.1 INSTRUCTION CODES

Mnemonic	Code	Operation	Time (usec.)
Memory Reference Instructions			
AND	0000	Logical AND	2.6
TAD	1000	Two's complement add	2.6
ISZ	2000	Increment and skip if zero	2.6
DCA	3000	Deposit and clear AC	2.6
JMS	4000	Jump to subroutine	2.6
JMP	5000	Jump	1.2

Mnemonic	Code	Operation	Sequence
Group 1 Operate Microinstructions (1 cycle) (1 cycle is equal to 1.2 microseconds.)			
OPR	7000	Same as NOP	–
NOP	7000	No operation	–
IAC	7001	Increment AC	3
RAL	7004	Rotate AC and link left one	4
RTL	7006	Rotate AC and link left two	4
RAR	7010	Rotate AC and link right one	4
RTR	7012	Rotate AC and link right two	4
CML	7020	Complemented link	2
CMA	7040	Complement AC	2
CLL	7100	Clear link	1
CLA	7200	Clear AC	1

Mnemonic	Code	Operation	Sequence
Group 2 Operate Microinstructions (1 cycle)			
HLT	7402	Halts the computer	3
OSR	7404	Inclusive OR SR with AC	3
SKP	7410	Skip unconditionally	1
SNL	7420	Skip on nonzero link	1
SZL	7430	Skip on zero link	1
SZA	7440	Skip on zero AC	1
SNA	7450	Skip on nonzero AC	1
SMA	7500	Skip on minus AC	1
SPA	7510	Skip on positive AC (zero is positive)	1
Combined Operate Microinstructions			
CIA	7041	Complement and increment AC	2,3
STL	7120	Set link to 1	1,2
GLK	7204	Get link (put link in AC, bit 11)	1,4
STA	7240	Set AC to 1	2
LAS	7604	Load AC with SR	2,3
Internal IOT Microinstructions			
IOT	6000		
ION	6001	Turn interrupt processor on	
IOF	6002	Disable interrupt processor	
Keyboard/Reader (1 cycle)			
KSF	6031	Skip on keyboard/reader flag	
KRB	6036	Clear AC, read keyboard buffer (dynamic), clear keyboard flags	
Teleprinter/Punch (1 cycle)			
TSF	6041	Skip on teleprinter/punch flag	
TCF	6042	Clear teleprinter/punch flag	
TPC	6044	Load teleprinter/punch and print	
TLS	6046	Load teleprinter/punch, print, and clear teleprinter/punch flag	
High Speed Reader -- Type PR8/E (1 cycle)			
RSF	6011	Skip on reader flag	
RRB	6012	Read reader buffer and clear reader flag	
RFC	6014	Clear flag and buffer and fetch character	
High Speed Punch -- Type PP8/E (1 cycle)			
PSF	6021	Skip on punch flag	
PCF	6022	Clear flag and buffer	
PPC	6024	Load punch buffer and punch character	
PLS	6026	Clear flag and buffer, load buffer and punch character	

Mnemonic	Code	Operation	Time (usec.)
DECtape Transport Type TU5 and DECtape Control Type TC08			
DTRA	6761	Read status register A	2.6
DTCA	6762	Clear status register A	2.6
DTXA	6764	Load status register A	2.6
DTSF	6771	Skip if error flag is 1 or if DECtape control flag is 1	2.6
DTRB	6772	Read status register B	2.6
DTLB	6774	Load status register B	2.6
Disk File and Control, Type DF32D			
DCMA	6601	Clear disk memory address register, and interrupt flags	2.6
DMAR	6603	Load disk memory address register and read	3.6
DMAW	6605	Load disk memory address register and write	3.6
DCEA	6611	Clear disk extended address register and memory address extension	2.6
DSAC	6612	Skip on address confirmed flag	2.6
DEAL	6615	Load disk extended address and memory address extension	3.6
DEAC	6616	Read disk extended address register	3.6
DFSE	6621	Skip on zero error flag	2.6
DFSC	6622	Skip on data completion flag	2.6
DMAC	6626	Read disk memory address register	3.6
Memory Extension Control, Type MC8/E (1 cycle)			
CDF	62N1	Change to data field N	
CIF	62N2	Change to instruction field N	
RDF	6214	Read data field	
RIF	6224	Read instruction field	
RIB	6234	Read interrupt buffer	
RMF	6244	Restore memory field	

G.2 PSEUDO-OPERATORS

The following is a list of the 4K PAL-D assembler pseudo-ops. The first section consists of those pseudo-ops which have counterparts in the other assemblers.

```
DECIMAL
OCTAL
FIELD
PAUSE
I
Z
$
EXPUNGE
FIXTAB
PAGE
=
*

XLIST
TEXT
```

INDEX

HOW TO OBTAIN SOFTWARE INFORMATION

SOFTWARE NEWSLETTERS, MAILING LIST

The Software Communications Group, located at corporate headquarters in Maynard, publishes newsletters and Software Performance Summaries (SPS) for the various Digital products. Newsletters are published monthly, and contain announcements of new and revised software, programming notes, software problems and solutions, and documentation corrections. Software Performance Summaries are a collection of existing problems and solutions for a given software system, and are published periodically. For information on the distribution of these documents and how to get on the software newsletter mailing list, write to:

Software Communications
P. O. Box F
Maynard, Massachusetts 01754

SOFTWARE PROBLEMS

Questions or problems relating to Digital's software should be reported to a Software Support Specialist. A specialist is located in each Digital Sales Office in the United States. In Europe, software problem reporting centers are in the following cities.

Reading, England	Milan, Italy
Paris, France	Solna, Sweden
The Hague, Holland	Geneva, Switzerland
Tel Aviv, Israel	Munich, West Germany

Software Problem Report (SPR) forms are available from the specialists or from the Software Distribution Centers cited below.

PROGRAMS AND MANUALS

Software and manuals should be ordered by title and order number. In the United States, send orders to the nearest distribution center.

Digital Equipment Corporation	Digital Equipment Corporation
Software Distribution Center	Software Distribution Center
146 Main Street	1400 Terra Bella
Maynard, Massachusetts 01754	Mountain View, California 94043

Outside of the United States, orders should be directed to the nearest Digital Field Sales Office or representative.

USERS SOCIETY

DECUS, Digital Equipment Computer Users Society, maintains a user exchange center for user-written programs and technical application information. A catalog of existing programs is available. The society publishes a periodical, DECUSCOPE, and holds technical seminars in the United States, Canada, Europe, and Australia. For information on the society and membership application forms, write to:

DECUS	DECUS
Digital Equipment Corporation	Digital Equipment, S.A.
146 Main Street	81 Route de l'Aire
Maynard, Massachusetts 01754	1211 Geneva 26
	Switzerland

READER'S COMMENTS

NOTE: This form is for document comments only. Problems with software should be reported on a Software Problem Report (SPR) form (see the HOW TO OBTAIN SOFTWARE INFORMATION page).

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

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

If you do not require a written reply, please check here. ☐

---------- Fold Here ----------

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

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

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

Postage will be paid by:

digital

Software Communications
P. O. Box F
Maynard, Massachusetts 01754