HOW TO OBTAIN REVISIONS AND CORRECTIONS

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

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

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

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

   Please send DECUS installation membership information.
   Please send DECUS individual membership information.
   Please add my name to the DECUSCOPE non-member mailing list.

   Name ___________________________
   Company _________________________
   Address _________________________
   ________________________________
   (Zip Code)
## CONTENTS

### CHAPTER 1
**INTRODUCTION**

1.1 Equipment Requirements  

### CHAPTER 2
**MONITOR OPERATION**

2.1 General Description  
2.1.1 Monitor Residence  
2.1.2 System Modes  
2.2 Bootstrapping the Monitor  
2.3 Starting the Monitor  
2.4 Command Strings  
2.4.1 Command String Format  
2.4.2 Examples of Command Strings  
2.5 Loading Programs - Binary Loader  
2.5.1 Binary Loader Operating Procedures  
2.5.2 Binary Loader Error Messages  
2.6 Saving Programs (Save Command)  
2.6.1 Save Command Format  
2.6.2 Save Command Processing  
2.7 Calling a Program (Call Command)  
2.8 System Error Messages  

### CHAPTER 3
**SYSTEM PROGRAM LIBRARY**

3.1 PIP  
3.1.1 Loading and Saving  
3.1.2 Operating Procedures  
3.1.3 Examples  
3.2 Editor  
3.2.1 Loading and Saving  
3.2.2 Operating Procedures  
3.2.3 Example  
3.3 PAL-D Disk Assembler
CONTENTS (Cont)

3.3.1 Loading and Saving 3-10
3.3.2 Operating Procedures 3-11
3.3.3 Examples 3-13
3.4 FORTRAN-D 3-13
3.4.1 Compiler 3-16
3.4.2 Operating System 3-22
3.4.3 Examples 3-31
3.5 DDT-D 3-33
3.5.1 Loading and Saving 3-36
3.5.2 Operating Procedures 3-37
3.5.3 Examples 3-37

APPENDIX A
SYSTEM GENERATION A-1

APPENDIX B
SYSTEM FORMATS B-1

APPENDIX C
COMMAND DECODER C-1

APPENDIX D
BINARY LOADER D-1

APPENDIX E
SYSTEM PROGRAMS E-1

TABLES

2-1 System Error Messages 2-13
3-1 Special Key Functions 3-5
3-2 Summary of Editor Commands 3-6
3-3 PAL-D Pseudo-Operators 3-10
3-4 PAL-D Error Messages 3-12
3-5 Summary of FORTRAN Statements 3-14
3-6 Compiler Systems Diagnostics 3-19
3-7 Compiler Compilation Diagnostics 3-20
3-8 Operating System Diagnostics 3-25
3-9 DDT-D Commands 3-36
ILLUSTRATIONS

B-1 Disk Storage Layout
B-2 DEClape Storage Layout
B-3 Directory Name (DN) Block Format
B-4 Storage Allocation Map (SAM) Block Format
B-5 Contiguous-Page Save File Format
B-6 Noncontiguous-Page Save File Format
B-7 Sample PIP Directory Listing
B-8 Monitor-Time vs User-Time Core Usage
B-9 Core Usage During SAVE Command Execution
B-10 Core Usage During CALL Command Execution
B-11 Monitor Flow Chart (Part 1)
B-11 Monitor Flow Chart (Part 2)
B-11 Monitor Flow Chart (Part 3)
C-1 Output List Produced by Command Decoder
C-2 Command Decoder Core Usage
C-3 Command Decoder Flow Chart (Part 1)
C-3 Command Decoder Flow Chart (Part 2)
C-3 Command Decoder Flow Chart (Part 3)
C-3 Command Decoder Flow Chart (Part 4)
C-3 Command Decoder Flow Chart (Part 5)
C-3 Command Decoder Flow Chart (Part 6)
D-1 Binary Loader Flow Chart (Part 1)
D-1 Binary Loader Flow Chart (Part 2)
D-1 Binary Loader Flow Chart (Part 3)
CHAPTER 1
INTRODUCTION

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

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

1.1 EQUIPMENT REQUIREMENTS

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

A basic PDP-8, PDP-8/S, or PDP-8/1
- 4K of core
- Teletype
- 3-Cycle Data Break (Option required with PDP-8/S)
- At least one DF32 Random Access DECdisk File or a TC01 Automatic Control with a TU55 DECTape transport. The DECTape must have timing and mark tracks written on it prior to use.

NOTE

The system will recognize up to 32K of core, up to four disks (1 Type DF32 and 3 Type D532's), up to eight DECTapes (TC01's only) and a high-speed paper-tape reader.
CHAPTER 2
MONITOR OPERATION

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

2.1 GENERAL DESCRIPTION

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

2.1.1 Monitor Residence

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

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

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

2.1.2 System Modes

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

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

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

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

The bootstrap procedure is as follows.

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

**Disk**

<table>
<thead>
<tr>
<th>Location</th>
<th>Contents</th>
<th>Symbolic</th>
</tr>
</thead>
<tbody>
<tr>
<td>0200</td>
<td>6603</td>
<td>DMAR</td>
</tr>
<tr>
<td>0201</td>
<td>6622</td>
<td>DFSC</td>
</tr>
<tr>
<td>0202</td>
<td>5201</td>
<td>JMP .-1</td>
</tr>
<tr>
<td>0203</td>
<td>5604</td>
<td>JMP l .+1</td>
</tr>
<tr>
<td>0204</td>
<td>7600</td>
<td>7600</td>
</tr>
</tbody>
</table>

**DecTape**

<table>
<thead>
<tr>
<th>Location</th>
<th>Contents</th>
<th>Symbolic</th>
</tr>
</thead>
<tbody>
<tr>
<td>0200</td>
<td>7600</td>
<td>BEG, 7600</td>
</tr>
<tr>
<td>0201</td>
<td>1216</td>
<td>TAD MVB</td>
</tr>
<tr>
<td>0202</td>
<td>4210</td>
<td>JMS DO</td>
</tr>
<tr>
<td>0203</td>
<td>1217</td>
<td>TAD M201</td>
</tr>
<tr>
<td>0204</td>
<td>3620</td>
<td>DCA I CA</td>
</tr>
<tr>
<td>0205</td>
<td>1222</td>
<td>TAD RF</td>
</tr>
<tr>
<td>0206</td>
<td>4210</td>
<td>JMS DO</td>
</tr>
<tr>
<td>0207</td>
<td>5600</td>
<td>JMP l BEG</td>
</tr>
<tr>
<td>0210</td>
<td>0000</td>
<td>DO, 0000</td>
</tr>
<tr>
<td>0211</td>
<td>6766</td>
<td>DTXA DTCA</td>
</tr>
<tr>
<td>0212</td>
<td>3621</td>
<td>DCA I WC</td>
</tr>
<tr>
<td>0213</td>
<td>6771</td>
<td>DTSF</td>
</tr>
<tr>
<td>0214</td>
<td>5213</td>
<td>JMP .-1</td>
</tr>
<tr>
<td>0215</td>
<td>5610</td>
<td>JMP l DO</td>
</tr>
<tr>
<td>0216</td>
<td>0600</td>
<td>MVB, 0600</td>
</tr>
</tbody>
</table>
b. After toggling in one of the above bootstrap routines, set the switches to 200 and press LOAD ADDdress 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 ADDdress and START, or (2) typing 1C when in Monitor mode or when a system program (or any user program which includes coding to sense a 1C typein) is running.¹

Monitor start performs the following actions.

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

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

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

A monitor restart can be performed by typing RUBOUT to Monitor. A Monitor restart performs the same actions as described above except for Subparagraph a. A common use for RUBOUT is to terminate a command string when the operator has discovered that he has made a mistake. The command string is ignored, and Monitor responds as described in Subparagraph c. The user core image on the system device is not changed by RUBOUT (it is changed, however, by 1C).

2.4 COMMAND STRINGS

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

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

---

¹ A start instruction (ST=7600) is issued when running Loader causes a jump to 7600 after loading has been performed. Certain errors also cause a jump to this location.
System programs indicate their readiness to receive information by typing either an asterisk or a query. The most common queries are as follows.

*OUT- Requests that the user specify one output device name. In the case of disk or DECtape the filename to be assigned to the output data must also be specified. ¹

*IN- Requests that the user specify one or more input device names. For disk and DECtape, filenames of input files must also be specified. ¹

*OPT- Requests that the user specify one option or switch, entered as a single alphanumeric character; see Chapter 3 for options available in each system program.

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

2.4.1 Command String Format

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

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

Each of these elements is described in the following paragraphs.

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

Dn: DECtape unit, if both disk and DECtape are present in the system (n = unit number, 0 through 7)

S: System device (disk or DECtape unit 0)

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

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

¹ Device names and filenames are explained in Paragraph 2.4.1.

² Command Decoder is a system program (.CD.) which is saved on the system device at build time.
2.4.1.2 Filenames - Filenames are limited to four characters in length and can be composed of any combination of alphanumeric characters or special characters with the following exceptions.

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

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

   CALL  SAVE

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

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.

Example:  

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>BLK</th>
</tr>
</thead>
<tbody>
<tr>
<td>8D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIP . SYS (0) 0015</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SRC1 . ASCII 0007</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIN . BINARY 0001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SRC1 . USER (0) 0001</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 apostrophe is also used to separate core references in a SAVE command string, when more than one contiguous 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.

---

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

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

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

4 "8D" in example means VERSION 8, change D.
Separates the beginning and ending addresses of a contiguous core area specification in a SAVE command.

! Follows the filename in a SAVE command when a file is to be saved as a system program.

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

`tC` 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. 
`tC` is typed by holding down the CTRL key and striking C. `tC` does not echo (does not print).

`tP` Typed in response to a `t` typeout. Instructs the system to proceed with the next operation. `tP` is typed by holding down the CTRL key and striking P. `tP` does not echo (does not print).

`J` Carriage return terminates current command string input. When typed alone, in response to a system query, it indicates that the user does not desire to specify the item (e.g., device name) requested.

RUBOUT Causes the current command string to be ignored, and the system returns to the beginning of the command string and is ready to receive a new command. RUBOUT does not echo.

2.4.2 Examples of Command Strings

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

Monitor Commands:

`- CALL PRG1 J` Call the user program file, PRG1, from the system device into core for execution.

`- SAVE PALD! 0-7577; 6200 J` 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 J` Use the file PRO2 on the system device as the input file.

`*IN-S:TST1,R: J` Use the file TST1 on the system device and one file from the high-speed paper tape reader as the input files.

`*OUT-D5:SPEC J` Write the output file on DECatape unit No. 5 and assign it the filename SPEC.

---

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

2 In all examples, system response (typeout) is underlined for clarity.
Punch the output on the Teletype paper tape punch. Select option M.  

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

```
SAVE PALD ! 0-7577; 6200
```

```
SAVE PALD !0-7577; 6200
```

2.5 LOADING PROGRAMS - BINARY LOADER

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

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

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

2.5.1 Binary Loader Operating Procedures

```
LOAD
```

Direct Monitor to bring Binary Loader from the system device into core for execution.

```
*IN-
```

Loader requests source of input(s). Type one or more device names, separated by commas. If an input device is a file-structured device, include filename(s).

Examples

```
*IN-R: 
```

Input one tape from the paper tape reader.

```
*IN-R; R: 
```

Input three tapes from the paper tape reader.

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

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

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

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

Regardless of whether R: or T: is used to specify paper tape input, the high-speed equipment is used if it was indicated as present in the system at System Builder time, otherwise the Teletype equipment is used. This convention is unique to Binary Loader.
*IN-S:INPT
*IN-S:BIN2,R:
*IN-S:BIN1,S:BIN2

Input the file INPT from the system device.
Input the file BIN2 from the system device and one
tape from the paper tape reader.
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 re­
sponds with one asterisk for each correct input.

Loader requests mode desired (one-pass or two-pass).

Examples

*OPT-

One pass loading desired; no programs are loaded
above location 6777.

*OPT-2
(or anything else)

Two-pass loading desired; programs can be loaded
above location 6777.

Loader requests the starting address to which control
is to be transferred when loading is completed. The
address is typed in the form

fnnn

where

f = field number 2 (omitted if field 0),
and

nunn = location within field

Examples

*ST=

Load into field 0.
Return to Monitor after loading.

*ST=7600
*ST=0

Load into field 3.
Jump to location 255, field 3, after loading.

*ST=30225
*ST=10000

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

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

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

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

2 The f-digit forces Loader to start loading into the specified field until a "field bit" is found in the
input.
First up-arrow: Loader is ready to load. If paper tape input, put the tape in the reader. Type tP.

Second up-arrow: End of pass 1. If operating in one-pass mode, type tP to jump to previously specified starting address.
If operating in two-pass mode, type tP.
The next two up-arrows appear only if operating in two-pass mode.

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

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

Multiple Input Files
An up-arrow is typed out as the processing of each input file is completed. If paper tape input, insert the next file in the reader and type tP.
Repeat the above step until all files given in response to the *IN- request have been processed.
If in two-pass mode, each tape must be entered twice, in the order

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

After all files have been entered the required number of times, type tP 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 tP 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 tP but press CONT on the console and start the paper tape reader.

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

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

2.5.2 Binary Loader Error Messages
An illegal checksum error condition causes Loader to type ?

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

1 If Teletype paper tape equipment is used, type tP before turning on the reader.
2.6 SAVING PROGRAMS (SAVE COMMAND)

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

Core images can be saved in units of one or more pages, each page occupying one block on the system device. If a core specification (see below) addresses only a portion of a page, the entire page is written out. For example, the core specification 45-150 is treated as though it were 0-177.

Core areas to be saved may be contiguous or noncontiguous as desired by the user. Up to 32,10 core specifications, in any combination of monotonically increasing single-page or multiple-page requests, can be entered in a single SAVE command.

2.6.1 SAVE Command Format

```
SAVE filename { !| or : | 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).

ункция

: 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.

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

\[ fnnn \]

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 the 15th page (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

\[ fnnn_1-nnnn_2 \]

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 below for explanation of how the field number (5th significant digit to the left of the decimal point) is "remembered."
NOTE

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

entry-point

The entry point of the saved program, in the format
Fnnnn (see explanation above)

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

NOTE

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

Example: The following entries are identical in meaning.
SAVE PRGA: 10000-10777, 11400, 1600-17777; 10200
SAVE PRGA: 30000-7777, 51400, 26000-7777; 10200
SAVE PRGA: 10000-7777, 1400, 6000-7777; 200
SAVE PRGA: 0-777, 1400, 6000-7777; 10200

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

2.6.2 SAVE Command Processing

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

2.7 CALLING A PROGRAM (CALL COMMAND)

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

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

```
.filename
```

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

2-12
The CALL command string format for user programs (programs saved by a SAVE command string in which the filename was followed by a :) is

```
CALL filename
```

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

2.8 SYSTEM ERROR MESSAGES

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

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

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

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

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

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

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

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

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

3.1 PIP

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

3.1.1 Loading and Saving

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

To load PIP into core, the user calls LOAD, using Monitor, and replies to the system responses as explained in Chapter 2.
When in core, PIP may be saved on the system device as a system device by Monitor, as indicated in Appendix E. (See Paragraph 2.6.1 for a detailed description of the SAVE format.)

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

\[
\begin{align*}
\text{LOAD } & \text{ } \\
\text{IN-R: } & \\
\text{* } & \\
\text{OPT-1 } & \\
\text{ST= } & \\
\text{SAVE PIP! 0-3177;1000 } & \\
\end{align*}
\]

3.1.2 Operating Procedures

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

\[
\begin{align*}
\text{LOAD } & \\
\text{IN-R: } & \\
\text{* } & \\
\text{OPT-1 } & \\
\text{ST= } & \\
\text{SAVE PIP! 0-3177;1000 } & \\
\end{align*}
\]

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

\[
\begin{align*}
\text{OPT- } & \\
\end{align*}
\]

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

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

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

\[
\begin{align*}
\text{OPT- } & \\
? & \\
\text{OPT- } & \\
\end{align*}
\]

\[\text{User system files may not be copied onto paper tape.}\]
Pages 3-1 through 3-4, attached, replace pages 3-1 through 3-4 of the PDP-8/I Disk Monitor System, DEC-D8-SDAA-D.
When in core, PIP may be saved on the system device as a system program by Monitor, as indicated in Appendix E. (See Paragraph 2.6.1 for a detailed description of the SAVE format.)

When loading and saving PIP, the printout will take approximately the following format:

```
.LOAD
*IN-R:2
*
*OPT-1
*ST=2
↑↑↓↓
.SAVE PIP! 0-5177;1000↓
```

3.1.2 Operating Procedures

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

```
.PIP↓
```

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

```
*OPT-↓
```

and waits for the user to select and specify one of the following options.
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 (See below)
P  Protect blocks 0-176 of disk 0
R  Restore the previously moved directory
A or \  Copy ASCII file (destination and origin(s) to be specified)
B  Copy binary file (destination and origin to be specified)
F  Copy FORTRAN binary file (destination and origin to be specified)
U  Copy user file (file structured origin and destination to be specified)
S  Copy system file (file structured origin and destination to be specified)

The user types only the option character, to which Monitor immediately responds with a carriage return and line feed. The user does not terminate the line with the RETURN key, it is a meaningful option.

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

\texttt{*OPT-G}
\texttt{?}
\texttt{*OPT-}

\textsuperscript{1} User and system files may not be copied onto paper tape as they are core images and have no defined paper tape format.
The L option lists the entire directory of the system device or DECTape on which a directory exists. For example,

```
.PIP
*OPT-L
*IN-S:
FB=2426
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>BLK</th>
</tr>
</thead>
<tbody>
<tr>
<td>PALD.SYS</td>
<td>(0)</td>
<td>0037</td>
</tr>
<tr>
<td>EDIT.SYS</td>
<td>(0)</td>
<td>0015</td>
</tr>
<tr>
<td>LOAD.SYS</td>
<td>(0)</td>
<td>0003</td>
</tr>
<tr>
<td>.CD..SYS</td>
<td>(0)</td>
<td>0006</td>
</tr>
<tr>
<td>PIP .SYS</td>
<td>(0)</td>
<td>0015</td>
</tr>
<tr>
<td>DDT .ASCII</td>
<td></td>
<td>0062</td>
</tr>
<tr>
<td>FOO .USER</td>
<td>(0)</td>
<td>0001</td>
</tr>
<tr>
<td>BAR .SYS</td>
<td>(0)</td>
<td>0037</td>
</tr>
</tbody>
</table>

User calls PIP 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 378 blocks of 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 from which the user may choose; indicating ASCII, binary, FORTRAN binary (compiler output), user (see Section 2.6.1), and system program (see Section 2.6.1), respectively.

If the user's reply is S \, indicating a system file, PIP asks

```
REALLY?
```
PIP will not delete a system file unless the user answers by typing
\[ Y] \] (meaning yes)
to the question. Any reply other than \[ Y] \ causes PIP to repeat the FILE TYPE request. When the user types \[ Y] \, PIP responds with

\[ *IN- \]

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

```
*OPT-D
*FILE TYPE(A,B,F,U,S)-S
REALLY?N
*FILE TYPE(A,B,F,U,S)-S
REALLY?Y
*IN-S:BAR
*OPT-
```

delete option specifying system file,
user must reply with \[ Y] \, PIP repeats request,
user replied correctly, PIP needs device & filename, 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 typed a ? before asking for another option. For example,

```
*IN-S:BAR
?
*OPT-
```

BAR is not the name of a file on the specified device.
The user should not try to delete the system files .CD. or LOAD.

Options M, P, and R, in conjunction with the hardware write-lock switch, allow the user to protect the lower 16K of his disk (1/2 of disk Ø for users with more than one disk) while using the system software. The user may specify either the system device or a DECTape unit numbered Ø-7. 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 will move a copy of the first directory block (the first 2510 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. If the user were to move a copy of the system file directory, the printout would appear as follows:

```
*OPT-M
*IN-S: move option specifying the
*OPT- the system device
```

PIP asks for another option

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 user may now activate the write-lock switch on
the disk control unit 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 will function independently of the M option. However, unless the user has previously moved a copy of the true directory which he can later restore, 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
*IN-S:
*OPT-
```

protect option specifying the system device

PIP asks for another option

The R option restores the copy of the directory name block (DN block 1) from block 3 back onto block 177 and the copy of the SAM block from block 4 back onto block 200. It then zeros all SAM blocks above the first one (if any) as well as directory name blocks 2 and 3. The R option will do nothing until a Move has been done on the specified device, so that a system may not be destroyed by inadvertently requesting the R option. The printout would look as follows:

```
*OPT-R
*IN-S:
*OPT-
```

restore option specifying the system device

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. Some typical uses for the M, P, and R options are:

1. M to save a specific disk (or DECTape) status and later R to effectively erase all scratch files created subsequent to the M, thus restoring the device to its status prior to the M.

3. L to determine the number of unused blocks and for a report on the status of the system device.

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-ASAA-D, and Section 3.5.1 of this manual.)

Options A, B, F, U, and S are used to transfer files from one device to another. When the user has requested any of these five options PIP responds with

*OUT-

and waits for the user to specify the destination or output file, and if the destination is disk or DECTape, the name of the file. For example,

*OPT-A
*OUT-S:ASCII

*OPT-A
*OUT-S:ASCII, E

*OPT-A
*OUT-S:ASCII

Only one destination is legal, and if the user specifies more than one, PIP will ignore the response, type the error message E, and return control to Monitor. For example,

NOTE: The L and D options return to PIP's option request (*OPT-) when the user responds illegally, and all other options return control to Monitor.
PIP indicates acceptance of the user's destination by responding with *, carriage return/line feed, and *IN-, and waits for the user to specify the input, that is, to state from where the input is to originate. An attempt to specify more than one input to any but the A option will cause PIP to ignore the response, type the error message E, and return control to Monitor. For example,

*OPT-F
*OUT-S:FORT
*IN-S:, E

copy a FORTRAN file option specifying system device and filename PIP accepts user's destination input to system device, comma is used to separate device names control returns to Monitor

The A option will allow any combination of up to 11 ASCII input files to be merged into one output file in the order specified by the input list. Therefore, the user may write generalized subroutines as separate files to do his often repeated operations and then, by combining these with each specialized program before assembly, eliminate the need to rewrite such operations for each program. PIP acknowledges each legal input file by printing an *. If, however, the input file specified to any option is not found on the specified device, PIP prints I in place of the * and returns to the Monitor. For example,

*IN-S:FIL2

the file does exist; when the user types CTRL/P, copying begins

*IN-S:FIL3

the file does not exist

If the user requests the B option indicating he wishes to copy a binary file but the filename he has specified appears
as an ASCII file, it is not acceptable, therefore, PIP prints an I control returns to Monitor. The user can ascertain file types by using the L option and checking the file directory.

A summary of the copy features of PIP is presented in the following table.

<table>
<thead>
<tr>
<th>Option</th>
<th>ASCII</th>
<th>Number of Input Files</th>
<th>Disk</th>
<th>DECtape</th>
<th>High-Speed Reader/Punch</th>
<th>Teletype</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>11</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Binary</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>User</td>
<td>1</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>System</td>
<td>S</td>
<td>1</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

3.1.3 Examples

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 378 blocks of storage.

User requests the delete option and specifies type of file, U(user) and device and filename; file is deleted.

User requests the delete option and specifies type of file, S (system) (PIP double checks); Y is the only meaningful answer.

User specifies file and filename; file is deleted.
User requests list option and system device directory
Note increase of 40 free blocks (see above)

Note removal of two deleted files

User requests delete option

Y is only response for deletion of a system file; other responses cause PIP to repeat the file type request

Even if user responds to REALLY? with Y, PIP will not delete the Monitor file

PIP knows NONE is not an existing user filename on the system device and indicates by typing ?
User requests ASCII file option
PIP also knows when the filename and file type don't match; EDIT is a system program

Merge into an ASCII file on disk "ASCI", one tape from the reader, one tape from the Teletype, one file from disk called SRC, and one file from DECtape 7 called SRC1.

(type CTRL/P after each file)
Copy the system file PIP from disk to DECTape 3 using filename PIPX.

*OPT-S
*OUT-D3:PIPX
*IN-S:PIP

(type CTRL/P)

Try to merge two binary files onto disk called BIN from paper tape.

*OPT-B
*OUT-S:BIN
*IN-R:,E

(list exceeded)

Try to copy an ASCII paper tape from high-speed reader, a non-existent file from DECTape 5, and a paper tape from Teletype to high-speed punch.

*OPT-A
*OUT-R:
*IN-R:,D5:FOO,T:

(R: accepted as legal)
(D5:FOO rejected, no such file on D5:)

3-4h
Editor (Disk System Editor) enables the user to generate and edit symbolic programs on-line from the teleprinter keyboard. The symbolic program may be either printed on the teleprinter, punched on paper tape using the high- or low-speed punch, or saved on the system device as a user program.

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

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

a. Special characters:

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

b. Commands:

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

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

<table>
<thead>
<tr>
<th>Key</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ (carriage return)</td>
<td>Text mode: Enter the line in the text buffer. Command mode: Execute the command.</td>
</tr>
<tr>
<td>- (back arrow)</td>
<td>Text mode: Cancel the entire line of text and continue typing on same line. Command mode: cancel command.</td>
</tr>
<tr>
<td>\ (rubout)</td>
<td>Text mode: Delete from right to left one character for each rubout typed (is not in effect during a READ command). Command mode: Delete entire command.</td>
</tr>
<tr>
<td>FORM FEED</td>
<td>Text mode: End of input, return to command mode.</td>
</tr>
</tbody>
</table>
Table 3-1 (Cont)
Special Key Functions

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

Table 3-2 is a summary of Editor commands.

Table 3-2
Summary of Editor Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Format(s)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ</td>
<td>R J</td>
<td>Read incoming text and append to buffer until a form feed is encountered.</td>
</tr>
<tr>
<td>APPEND</td>
<td>A J</td>
<td>Append incoming text to any already in the buffer until a form feed is encountered.</td>
</tr>
<tr>
<td>LIST</td>
<td>L J</td>
<td>List the entire buffer.</td>
</tr>
<tr>
<td></td>
<td>nL J</td>
<td>List the line n.</td>
</tr>
<tr>
<td></td>
<td>m,nL J</td>
<td>List lines m through n.</td>
</tr>
<tr>
<td>PROCEED</td>
<td>P J</td>
<td>Proceed and output the entire contents of the buffer and return to command mode.</td>
</tr>
<tr>
<td></td>
<td>nP J</td>
<td>Output line n, followed by a form feed.</td>
</tr>
<tr>
<td></td>
<td>m,nP J</td>
<td>Output lines m through n, followed by a form feed.</td>
</tr>
<tr>
<td>TRAILER</td>
<td>T J</td>
<td>Punch four inches of trailer.</td>
</tr>
<tr>
<td>NEXT</td>
<td>N J</td>
<td>Punch the entire buffer and a form feed; kill the buffer and read next page.</td>
</tr>
<tr>
<td></td>
<td>nN J</td>
<td>Repeat the above sequence n times.</td>
</tr>
<tr>
<td>KILL</td>
<td>K J</td>
<td>Kill the buffer.</td>
</tr>
<tr>
<td>DELETE</td>
<td>nD J</td>
<td>Delete line n.</td>
</tr>
<tr>
<td></td>
<td>m,nD J</td>
<td>Delete lines m through n.</td>
</tr>
</tbody>
</table>
### Table 3-2 (Cont)
#### Summary of Editor Commands

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

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

3.2.1 Loading and Saving

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

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

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

(See Paragraph 2.6.1 for detailed description of the SAVE format.)
When loading and saving Editor, the printout should appear approximately as follows.

```
LOAD
*IN-R: *
*OPT-1
*ST=7600

SAVE EDIT10-3177; 2600
```

(See Appendix E.)

3.2.2 Operating Procedures

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

```
EDIT
```

Editor is now in core and responds by typing

```
*OUT-
```

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

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

```
*IN-
```

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

```
*OPT-
```

asking the user to specify one of the following options.

- **B**: Preserve blanks. Editor normally replaces multiple blanks (spaces) with tabs, resulting in considerable saving of space on the system device.
- **D**: Enter dynamic deletion mode if input is from the system device. As the file is read, it is deleted from the system device, thus allowing space for output if desired. (File name remains on the directory but without any assigned blocks.)
- **C**: Combine the functions of B and D options.

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

---

1 With a System Device output, the user must type E to properly close the output file.
After the user has specified one of the options listed above, Editor responds with a carriage return/line feed and asterisk. The entire printout might appear as follows:

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

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

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

### 3.2.3 Example

```
LOAD
*IN-R:
*OPT-1
*ST=
SAVE EDIT10-3177;2600
EDIT
*OUT-S:SRC1
*IN-R:
*OPT-
*R
*E
EDIT
*OUT-
*IN-S:SRC1
*OPT-
*R
*L
7400
ODUM,
CLA
OSR
DCA LOCK
HLT
OSR
CMA
/L
S
-  

/*L was typed here, stopped listing of buffer
(C was typed here)
```
3.3 PAL-D DISK ASSEMBLER

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

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

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

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

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

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

3.3.1 Loading and Saving

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

To load PAL-D into core, the user calls LOAD using Monitor and replies to the system responses as explained at the beginning of this chapter.
When in core, PAL-D may be saved on the system device as a system program by Monitor as described in Appendix E. (See Paragraph 2.6.1 for a detailed description of the SAVE format.) When loading and saving PAL-D, the printout should appear approximately as shown below. (See Paragraph 2.5.)

\[
\text{LOAD} \downarrow \\
\text{*IN-R:} \downarrow \\
\text{*OPT-2} \downarrow \\
\text{ST=7600} \uparrow \\
\text{SAVE PALD}!0-7577; 6200 \downarrow \\
\]

3.3.2 Operating Procedures

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

\[ \text{PALD} \downarrow \]

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

\[ \text{*OUT-} \]

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

\begin{align*}
\text{T:} & \quad \text{for the low-speed punch} \\
\text{R:} & \quad \text{for the high-speed punch} \\
\text{S:name} & \quad \text{for output to the system device as a file called name} \\
\end{align*}

PAL-D then responds with

\[ \text{*IN-} \]

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

\[ \text{R:} \downarrow \]

input from the high-speed reader

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

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

\[ \text{*OPT-} \]
The user types one of the following.

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

(Any other character means no third pass desired)

The entire printout might appear as follows.

```
P A L-D J
+ *O U T - T: J
* *I N - R: J
* *O P T - T J
```

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

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

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

```
ERROR CODE   ADDRESS
```

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

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

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

Table 3-4
PAL-D Error Messages
3.3.3 Examples

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

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

3.4 FORTRAN-D

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

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

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

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

a. Arithmetic statements define a numerical calculation.

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

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

d. Input-output statements are used to transmit information between the computer and related input-output devices.
A summary of the FORTRAN statements is given in Table 3-5.

### Table 3-5
**Summary of FORTRAN Statements**

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

\[
\begin{align*}
\text{SQTF}(x) & \quad \text{square root of } x \\
\text{SINF}(x) & \quad \text{sine of } x \\
\text{COSF}(x) & \quad \text{cosine of } x \\
\text{ATNF}(x) & \quad \text{arctangent of } x \text{ (in radians)} \\
\text{EXPF}(x) & \quad \text{exponential of } x \\
\text{LOGF}(x) & \quad \text{logarithm of } x \\
\text{ABSF}(x) & \quad \text{absolute value of } x
\end{align*}
\]

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

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

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

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

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

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

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

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

<table>
<thead>
<tr>
<th>Typed and Corrected</th>
<th>Read</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Integer Numbers:</strong></td>
<td></td>
</tr>
<tr>
<td>128 1028</td>
<td>+1028</td>
</tr>
<tr>
<td>128 -28</td>
<td>-128</td>
</tr>
<tr>
<td>-128 128</td>
<td>+128</td>
</tr>
<tr>
<td><strong>Floating-point numbers:</strong></td>
<td></td>
</tr>
<tr>
<td>2 42</td>
<td>+42.0</td>
</tr>
<tr>
<td>+2.42</td>
<td>+42.0</td>
</tr>
<tr>
<td>-2.0 2.0</td>
<td>+2.0</td>
</tr>
<tr>
<td>42 -42.2</td>
<td>-42.2</td>
</tr>
<tr>
<td>20E6 5</td>
<td>+2.0 \times 10^5</td>
</tr>
<tr>
<td>2.0E-6 5</td>
<td>+2.0 \times 10^{-5}</td>
</tr>
</tbody>
</table>

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

The following examples show how the READ and WRITE statements might be used in a typical FORTRAN program.
C EXAMPLE PROGRAM TO READ COORDINATE PAIRS
C FROM THE TELETYPETE 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 (1)
TYPE 102
102 FORMAT ("NOW ENTER THE COORDINATES")
DO 20 I=1,N
ACCEPT 30,X,Y
WRITE 3,30,X,Y
20 CONTINUE
STOP
30 FORMAT (E,E)
END

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

C FORTRAN EXAMPLE PROGRAM
DEFINE DISK
DIMENSION X(100),Y(100)
C READ DATA FROM THE DISK DEVICE NR3
IDEv=3
6 SUMX=0
SUMY=0
DO 10 I=1,100
READ IDEV,20,X(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,
//"TYPE 0 TO STOP, 1 TO CONTINUE")
40 FORMAT (1)
END

3.4.1 Compiler

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

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

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

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

d. Significant improvements have been employed with the READ and WRITE statements.
e. Input and output devices are determined using the Command Decoder.

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

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

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

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

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

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

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

```
LOAD J
*IN-R: J
*OPT-1 J
*ST=7600 J

SAVE FOR10-1777; 200 J (See Appendix E.)
LOAD J
*IN-R: J
*OPT-2 J
*ST=7600 J

SAVE .FT.1200-7377; J (See Appendix E.)
```

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

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

```
_FORT J
```

Command Decoder then types

```
*OUT-
```

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

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

3-17
Command Decoder will respond with an * when it recognizes a valid output device, and then types

```
*IN-
```

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

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

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

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

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

The process described above would produce the following printout.

```
• FORT J
• *OUT-R: J
• *IN-R: J
• t
• t
```

(tC typed here; compilation finished)

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

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

Table 3-6 lists the system error messages.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0227</td>
<td>Could not find Command Decoder on system device</td>
</tr>
<tr>
<td>0231</td>
<td>Same as above</td>
</tr>
<tr>
<td>0326</td>
<td>Could not find .FT. on system device</td>
</tr>
<tr>
<td>0330</td>
<td>Same as above</td>
</tr>
<tr>
<td>1425</td>
<td>READ error during directory or SAM block search</td>
</tr>
<tr>
<td>1521</td>
<td>Same as above</td>
</tr>
<tr>
<td>1626</td>
<td>Same as above</td>
</tr>
<tr>
<td>1726</td>
<td>WRITE error during SAM block search</td>
</tr>
<tr>
<td>3100</td>
<td>Illegal operator on compiler stack 1</td>
</tr>
<tr>
<td>3417</td>
<td>Pre-precedence error 1</td>
</tr>
<tr>
<td>4737</td>
<td>No input device or invalid input device specified</td>
</tr>
<tr>
<td>6141</td>
<td>Attempt to execute a program not compiled onto the system device</td>
</tr>
<tr>
<td>6145</td>
<td>Could not find FOSL on system device; if the error occurs, it may be necessary to reload FORT and FOSL.</td>
</tr>
<tr>
<td>6207</td>
<td>READ error while loading FOSL</td>
</tr>
<tr>
<td>6211</td>
<td>Error while doing SAM block manipulation 1</td>
</tr>
<tr>
<td>6223</td>
<td>Error while loading .FT.</td>
</tr>
<tr>
<td>6226</td>
<td>Same as above</td>
</tr>
<tr>
<td>6257</td>
<td>Same as above</td>
</tr>
<tr>
<td>6407</td>
<td>Illegal overlay request 1</td>
</tr>
<tr>
<td>6416</td>
<td>Same as above</td>
</tr>
<tr>
<td>6467</td>
<td>System device READ error</td>
</tr>
<tr>
<td>6724</td>
<td>No END statement on source device</td>
</tr>
<tr>
<td>6746</td>
<td>Same as above</td>
</tr>
<tr>
<td>7114</td>
<td>Same as above</td>
</tr>
<tr>
<td>7136</td>
<td>READ error on system device source file</td>
</tr>
<tr>
<td>7150</td>
<td>System device full</td>
</tr>
<tr>
<td>7173</td>
<td>WRITE error on system device output file</td>
</tr>
</tbody>
</table>

1 Error may be due to a compiler error or a machine malfunction.
The example below illustrates the appearance of the error codes.

```
*FORT*
0227
*FORT*
*OUT-*
*IN-Sname >
SYS 6141 <C>
```

Command Decoder not on system device
No output file specified

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

```
XXXX XX XX
```

The error code
The number of statements since the appearance of last numbered statement (octal)
The statement number of the last numbered statement

For example, during compilation of the statements

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

the error message

```
10 11 11
```

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

Table 3-7 lists the compilation error

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

Table 3-7
Compiler Compilation Diagnostics
Table 3-7 (Cont)
Compiler Compilation Diagnostics

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

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

Symbolprint provides the following information:

a. The limits of the interpretive code.

b. A list of variable names and their corresponding locations (the symbol table).

c. A list of statement numbers and their corresponding locations (the statement number table).

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

```
LOAD
*IN-R:
*OPT-1
*ST=

SAVE STBL!600-777;600 d
```

(See Appendix E.)

When in core, Symbolprint occupies locations 600-777 with its starting address at location 600.
When symbolprint is called into core, it types the interpretive code limits, symbol table, statement number table, carriage return/line feed, and \texttt{t}. At this point the user may execute his program by typing \texttt{tP}, or he may return to Monitor by typing \texttt{tC}.

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

\begin{verbatim}
FORT 
*OUT- 
*IN-S:SRC 
T
STBL 
6154 7565
N 7576 
1 7575 
X 7571 
Y 7566 
0100 6033
0010 6060
0102 6066 
0020 6145
0030 6147
\texttt{<\texttt{tC}>}
\end{verbatim}

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

3.4.2 Operating System

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

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

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

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

d. The execution of a FORTRAN program may be interrupted by the user at any time by typing \texttt{tC}; control will be returned to Monitor.
3.4.2.1 Loading the FORTRAN Operating System -- To load the operating system, the following steps are performed.

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

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

```
LOAD %
*IN-R:* 
*OPT-1 *
*ST=* 

SAV FOSL!0-1377;200 (See Appendix E.)
LOAD %
*IN-R:* 
*OPT-1 *
*ST=* 

SAV .OS. !0-4777;0 (See Appendix E.) 
```

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

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

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

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

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

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

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

3-23
The following examples show how the FORTRAN operating system may be used.

Example 1

```
.FOSL
*IN-S:FBIN
*OPT-
*READY

1
```

(Program execution occurs here)

Example 2

```
1
.FOSL
*IN-R:, R:

ERROR 01
*READY
1
```

(t'P typed here)

Example 3

```
.FORT
*OUT-S:SMSQ
*IN-S:SMSQ

*READY

1
```

Program execution begins here

Example 4

```
.FOSL
*IN-S:BIN
*OPT-S
*OUT-S:DAT2
*IN-S:DAT1
*READY
1
```

(Program execution begins here)

In example 2 a checksum error was detected on the second input tape. In this case the operator decided to attempt to execute the program in spite of the checksum error.
3.4.2.3 Operating System Diagnostics -- When an error occurs during program execution, the operating system will type ERROR followed by a two-digit error code number which will indicate the cause of the error. Depending on the nature of the error, it may be possible to continue program execution by typing tP or it may be necessary to return to the Monitor by typing tC.

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

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

¹ May be caused by machine malfunction or operating system error.
² May be caused by source program or loading error; to correct, do the following in descending order.
   a. Use Diagnose to determine where the error occurred.
   b. Recompile the source program.
   c. Examine source program (in particular the arithmetic statements and subscripted variables).

When an error occurs, execution will stop and the operating system will wait for the user to type tP or tC.
3.4.2.4 Debugging Aid (Diagnose) -- Diagnose is a basic system program whose purpose is to help the user debug his FORTRAN program. It is intended to be used in conjunction with the PDP-8 4K FORTRAN Operating System and revised FORTRAN Symbolprint. Diagnose provides the following information.

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

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

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

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

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

```
LOAD 'IN -R; DIAG
OPT -1;
ST -
SAVE DIAG1200-1177;200
```

(See Appendix E.)

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

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

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

Program 1:
```
*L
C FORTST
C PDP-8 ADVANCED SOFTWARE
C FORTRAN TEST 1/2/68
DIMENSION ADIFE(6),AFAC(3),APIPE(6),IMRCD(3),PP(27),ACPRI(3)
TYPE 1
1 FORMAT("PDP-8 4-K FORTRAN TEST")
ASPVA = .60
APIPE(1) = 12.09
APIPE(3) = 6.66
APIPE(4) = 5.0
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
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
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, 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*.0177*FSUBB
110 TOTA + PP(IE)
TOTB = PP(IE - 1)
SC = .025 - (AG - BE)
WRITE 2, 991, TOTA, TOTB, SC, AG, BE, IE
991 FORMAT(" " 1", E, E, E/ "", E, 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*.0177*FSUBB
AFAC(JCB) = ADIFE(JCB)*BSPVA

3-27
WRITE 2,992,AK,FRD,AMEAS,BMEAS,FR,XSUB2,YTTA,YTTB,'YTTC,YSUB2,ACPRI(JCB),JCB
992 FORMAT(" 2",E,E,E,E)
550 CONTINUE
AFTF=(520./(460.+AGAST)**.5
AFPV=(1.+(ASUVA*AMEAS)/((AGAST+460.)**3.825)**.5
FLOW=O
RATE=O
DO 38 I=1,INORU
AMWP=(ADIFE(I)*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/E)
STOP
993 FORMAT(E,E,E,E)
END

*STBL

6360 6756
ADIF 7555
AFAC 7544
APIP 7522
IMRC 7517
PP 7376
ACPR 7365
ASPV 7362
AF 7310
SC 7302
AMEA 7274
FSUB 7266
ASUV 7260
I 7254
AGAS 7246
INOR 7244
BSPV 7240
JCB 7231
AVED 7225
BE 7222
AK 7213
AG 7205
IE 7201
TTA 7171
TOTB 7166
FRD 7153
BMEAS 7124
FR 7116
XSUB 7102
YTTA 7074
YTTB 7063
YTTC 7047
YSUB 7041
AFTF 7032
AFPV 7016
FLOW 6777
RATE 6773
AMWP 6766
Example 1 (a)

*READY
*READY
PDP-8 4-K FORTRAN TEST
0.255323E+1 -0.825572E+1

.DIAG
CURRENT LOCATION COUNTER AT 6347

Example 1 (b)

.FOSL
*IN-S:BIN
*READY
PDP-8 4-K FORTRAN TEST
ERROR 05 (tC typed here)
.DIAG
CURRENT LOCATION COUNTER AT 5407

Example 1 (c)

.FOSL
*IN-S:BIN
*READY
PDP-8 4-K FORTRAN TEST (tC typed here)
.DIAG
CURRENT LOCATION COUNTER AT 4404
COUNTER STACK...
4733
4716
4673
6024
In example 1(a), the program was run to completion after which Diagnose was called. Diagnose indicated that the current location counter contained 6347. By referring to the statement number table (top of page 3-29, we can see that the CLC was pointing to an address just above statement 993 (address 6350), verifying that the program terminated normally at that point.

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

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

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

Program 2:

```fortran
.C EDIT
*OUT-S:SRC
*
*IN-
*
*OPT-B
*

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

.DIAG
OPERAND STACK OVERFLOW
CURRENT LOCATION COUNTER AT 5231

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

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 /  
*IN-R: /  
*OPT-1 /  
*ST= J  
SAVE FORT!0-1777;200 J
```

Call Loader
Input to be from high-speed reader
Input device is valid
One-pass load
Return to Monitor after loading
Loader-driver is loaded
and saved on the system device

```
LOAD /  
*IN-R: /  
*OPT-2 /  
*ST= J  
SAVE .FT.,1200-7377;0 J
```

Call Loader
Input to be from high-speed reader
Input device is valid
Two-pass load
Return to Monitor after loading
Compiler is loaded
and saved on the system device

```
LOAD /  
*IN-R: /  
*OPT-1 /  
*ST= J  
SAVE FOSL!0-1377;200 J
```

Call Loader
Input to be from high-speed reader
Input device is valid
One-pass load
Return to Monitor after loading
Operating system loader is loaded
and saved on the system device
Call Loader
Input to be from high-speed reader
Input device is valid
One-pass load
Return to Monitor after loading
Interpretive and arithmetic package is loaded
and saved on the system device

Call Loader
Input to be from high-speed reader
Input device is valid
One-pass load
Return to Monitor after loading
Symbolprint is loaded
and saved on the system device

Call Editor
Output to be on system device
Output device is valid
Input to be from high-speed reader
Input device is valid
Leave blanks (spaces) unchanged
Write the program on the system device
then write an end-of-file

Call FORTRAN compiler
FORTRAN binary output to be on system device
Output device is valid
USA ASCII input to be from system device
Input device is valid
Compilation is finished, return to Monitor

Call FORTRAN Symbolprint
Core between 6200 and 7564 is unused

Symbol table (typed by Symbolprint)

Statement number table (typed by Symbolprint)

<table>
<thead>
<tr>
<th>M</th>
<th>A</th>
<th>B</th>
<th>ANS</th>
<th>0001</th>
<th>0002</th>
<th>0003</th>
<th>0004</th>
<th>0005</th>
<th>0006</th>
<th>0009</th>
<th>0100</th>
<th>0200</th>
<th>0300</th>
<th>0400</th>
<th>0500</th>
<th>1000</th>
<th>2000</th>
<th>3000</th>
<th>4000</th>
<th>1500</th>
<th>0008</th>
<th>0007</th>
<th>0006</th>
</tr>
</thead>
<tbody>
<tr>
<td>7576</td>
<td>7573</td>
<td>7570</td>
<td>7565</td>
<td>5200</td>
<td>5257</td>
<td>5413</td>
<td>5570</td>
<td>5717</td>
<td>5754</td>
<td>5760</td>
<td>5763</td>
<td>5766</td>
<td>5771</td>
<td>5774</td>
<td>5777</td>
<td>6027</td>
<td>6040</td>
<td>6051</td>
<td>6062</td>
<td>6071</td>
<td>6077</td>
<td>6123</td>
<td></td>
</tr>
</tbody>
</table>

6177 7565
Symbolprint is finished, load operating system and interpretive code
*READY ∙ Operating system and interpretive code are loaded
∙
THIS IS A DEMONSTRATION OF PDP FORTRAN.
THIS PROGRAM WAS COMPILED IN ONE PASS (∗C typed here)

-FOSL J Call operating system and loader
IN-S:FOR FORT Binary input is on system device
IN-S:FOR FORT Input device is valid
OPT- ∙ No input or output to be done on system device
during program execution
*READY J Operating system and interpretive code have been loaded
∙
BEGIN PROGRAM EXECUTION (∗C typed here)

THIS IS A DEMONSTRATION OF PDP FORTRAN (∗C typed here)

3.5 DDT-D

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

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

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

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

a. DDT Proper
   Self-explanatory; occupies core locations 200-4577 and the three breakpoint locations,

b. Driver
   Resident in core with its origin set above DDT proper (above 4577); it is a two-page program plus a one-page once-only program, and it contains breakpoint insertion and removal logic, overlay routines, continuation iteration count and control, and breakpoint list.
c. User Core Image File

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

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

a. Three breakpoints (as opposed to only one in DDT-8)
b. No punching (program may be output on the system device)
c. No switch options (user direction is via keyboard)
d. No halts (continues when user types t P)

Variations in commands1 follows.

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

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

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

The following subroutines have been added.

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

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

3-34
The following subroutines have been modified as indicated.

a. REDTAB  
Assumes user wants to add to existing symbol table; user must type \[ X \] to clear the symbol table.

b. FINIS  
Does not halt, instead, it waits for the user to type \[ +P \].

c. CHANGE  
Allows lookup of values to change limit of search and search mask.

d. TODDT  
Handles breakpoint insertion; transferred to DDT driver.

e. TRAP  
Breakpoint handler; transferred to DDT driver.

The following subroutines have been removed.

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

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

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

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

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

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

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

DDT-D Commands

<table>
<thead>
<tr>
<th>Character</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>(space)</td>
<td>Separation character</td>
</tr>
<tr>
<td>+</td>
<td>Arithmetic plus</td>
</tr>
<tr>
<td>-</td>
<td>Arithmetic minus</td>
</tr>
<tr>
<td>/</td>
<td>Location examination character; when it follows the address of a location, it causes the contents of that location to be printed</td>
</tr>
<tr>
<td>\ (carriage return)</td>
<td>Make modifications, if any</td>
</tr>
<tr>
<td>] (line feed)</td>
<td>Make modifications, if any, and print the contents of the next sequential location</td>
</tr>
<tr>
<td>=</td>
<td>Type last quantity as an octal integer</td>
</tr>
<tr>
<td>. (period)</td>
<td>Current location</td>
</tr>
<tr>
<td>← (left arrow)</td>
<td>Delete the line currently being typed</td>
</tr>
<tr>
<td>[S</td>
<td>Sets DDT-D to type out in symbolic mode</td>
</tr>
<tr>
<td>[O</td>
<td>Sets DDT-D to type out in octal mode</td>
</tr>
<tr>
<td>n\W</td>
<td>Word search for all occurrences masked with C[M) of the expression n</td>
</tr>
<tr>
<td>k\Bn</td>
<td>Insert breakpoint n at location k (n = 1, 2, or 3)</td>
</tr>
<tr>
<td>[Bn</td>
<td>Remove breakpoint n (n = 1, 2, or 3)</td>
</tr>
<tr>
<td>n\C</td>
<td>Continue n times automatically; if n is absent, it is assumed to be 1</td>
</tr>
<tr>
<td>k\G</td>
<td>Go to location k</td>
</tr>
<tr>
<td>[R</td>
<td>Append symbol table into external symbol table or define symbols on line</td>
</tr>
</tbody>
</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 (Appendix E) and uses locations 3, 4, and 5 (page 0) as the breakpoint locations. After loading DDT proper, the user should reserve on the system device a user core image file name .SYM, which should also be assigned to core locations 200-4577.¹

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

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

3-36
The sections of DDT are loaded and saved as described below.

• LOAD
  • IN-R: 
  • OPT-1
  • ST = 
  • SAVE .DDT:200-4500;0
  • SAVE .SYM:200-4577;0

• LOAD
  • IN-R: 
  • OPT-2
  • ST = 7000
  • IN-S:.<.DDT>,..SYM>
  • IN-S:.DDT,.SYM 
  • SAVE DDT!7200-7577;7200

The error message DDT? is typed whenever an error is encountered while loading DDT-D.

Errors may be caused by the following.

a. User file too large
b. System device read error
c. No Command Decoder

3.5.2 Operating Procedures

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

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

.DDT

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

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

3.5.3 Example

• LOAD
  • IN-R: 
  • OPT-1
  • ST = 
  • SAVE

Call Loader
Input to be from high-speed reader
Loader found input device valid
One-pass load
Return to Monitor after loading
DDT proper is loaded
DDT proper is saved on disk  (See Appendix E.)
User code image file also saved
Call Loader
Input to be from high-speed reader
Loader found input device valid
Two-pass load
Transfer to once-only code after loading
Driver is loaded
Loader expects 2 input files for use by driver
Inputs from DDT proper and user code image file

Loader found both input files valid
Driver is saved on disk  (See Appendix E.)
Call DDT using Monitor
Examine contents of location
3400 and 3401
Set breakpoint No. 1 at location 3400
Start execution at location 3400
Location 3400 contains 0000
Continue
Location 3400 now contains 0001
Pass through location 3400 700 times
Location 3400 now contains 0701
↑C was typed here
APPENDIX A
SYSTEM GENERATION

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

The steps involved in system generation are as follows.

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

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

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

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

<table>
<thead>
<tr>
<th>High-Speed Reader</th>
<th>Teletype Reader</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Location</strong></td>
<td><strong>Instruction</strong></td>
</tr>
<tr>
<td>7756</td>
<td>6014</td>
</tr>
<tr>
<td>7757</td>
<td>6011</td>
</tr>
<tr>
<td>7760</td>
<td>5357</td>
</tr>
<tr>
<td>7761</td>
<td>6016</td>
</tr>
<tr>
<td>7762</td>
<td>7106</td>
</tr>
<tr>
<td>7763</td>
<td>7006</td>
</tr>
<tr>
<td>7764</td>
<td>7510</td>
</tr>
<tr>
<td>7765</td>
<td>5374</td>
</tr>
<tr>
<td>7766</td>
<td>7006</td>
</tr>
<tr>
<td>7767</td>
<td>6011</td>
</tr>
<tr>
<td>7770</td>
<td>5367</td>
</tr>
<tr>
<td>7771</td>
<td>6016</td>
</tr>
<tr>
<td>7772</td>
<td>7420</td>
</tr>
<tr>
<td>7773</td>
<td>3776</td>
</tr>
<tr>
<td>7774</td>
<td>3376</td>
</tr>
<tr>
<td>7775</td>
<td>5357</td>
</tr>
<tr>
<td>7776</td>
<td>0000</td>
</tr>
</tbody>
</table>

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

A.2 LOADING THE BINARY (BIN) LOADER

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

a. Check that the RIM Loader is in core.

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

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

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

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

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

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

A.3 LOADING AND EXECUTING DISK/DECTAPE SYSTEM BUILDER

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

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

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

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

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

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

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

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

g. As the following questions are typed out, answer them according to your machine configuration.
*YES*
*TYPE SIZE OF CORE (IN K) 8*

*HIGH SPEED PAPER TAPE? YES
*PSP-B/S? NO
*DISC? YES
*TYPE NUMBER OF DISC UNITS 1
*TAPE? YES

User enters core size of his machine (4, 8, 12, 16, 20, 24, 28, or 32).

User answers YES or NO.

User answers YES or NO.

User answers YES or NO.

User types number of disk units on his machine.

User types YES if he has DECtape, NO if he does not.

A maximum delay of one minute occurs here while Monitor creation is being completed, the resident portion is moved to the appropriate core area (7600 through 7777), and the nonresident portions are written on the system device.

**NOTE**

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

Monitor is loaded and ready.

If the response

**WRITE ERROR**

occurs:

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

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

A.4 LOADING AND SAVING SYSTEM PROGRAMS

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

```
*. LOAD
*IN-R:
*OPT-1
ST=7600

.SAVE PIP! 0-3177; 1000
```

Calls Binary Loader from the system device.

Input device is paper tape reader (high-speed reader if specified as present at System Builder time; otherwise Teletype reader).

Device is valid.

One-pass loading mode selected.

Return to Monitor after loading.

After each up-arrow typeout, user types 1 P to continue (also must press CONT in teletype reader if Teletype reader is being used).

Saves program (in this case, PIP) on system device.

Note that a ! must follow name of system program.

The SAVE command is explained in Chapter 2. The SAVE command program is given in Appendix E.

Repeat the procedure above for each system program to be saved.
This appendix contains the following information.

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

b. Data Structure
   - Source File (ASCII)
   - Binary File (BINARY, FTC BIN)
   - Saved Files (SYS, USER)

c. PIP Listing of System Device Map (for Disk)
d. Monitor Core Usage Diagrams

B.1 SYSTEM DEVICE LAYOUTS

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

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

B.1.1 Directory Name (DN) Blocks

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

\[ \text{DN}_1 \] Contains entries for internal file numbers \(01 \text{ through } 318 (2510)\) and a link to \(\text{DN}_2\).
\[ \text{DN}_2 \] Contains entries for internal file numbers \(32 \text{ through } 628 (5010)\) and a link to \(\text{DN}_3\).
\[ \text{DN}_3 \] Contains entries for internal file numbers \(63 \text{ through } 778 (6310)\) 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 \((1 \text{ through } 778)\). 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.
<table>
<thead>
<tr>
<th>BLOCK</th>
<th>CONTENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>MONITOR HEAD</td>
</tr>
<tr>
<td>1</td>
<td>MONITOR (1ST PAGE OF SAVE)</td>
</tr>
<tr>
<td>2</td>
<td>MONITOR (START)</td>
</tr>
<tr>
<td>3</td>
<td>DN (BACKUP — SEE PIP)</td>
</tr>
<tr>
<td>4</td>
<td>SAM (BACKUP — SEE PIP)</td>
</tr>
<tr>
<td>5</td>
<td>MONITOR (2ND PAGE OF CALL)</td>
</tr>
<tr>
<td>6</td>
<td>MONITOR (3RD PAGE OF SAVE)</td>
</tr>
<tr>
<td>7</td>
<td>MONITOR (2ND PAGE OF SAVE)</td>
</tr>
<tr>
<td>10</td>
<td>MONITOR (1ST PAGE OF CALL)</td>
</tr>
<tr>
<td>11</td>
<td>MONITOR (4TH PAGE OF SAVE)</td>
</tr>
<tr>
<td>12</td>
<td>LOADER</td>
</tr>
<tr>
<td>13</td>
<td>LOADER</td>
</tr>
<tr>
<td>14</td>
<td>LOADER</td>
</tr>
<tr>
<td>15</td>
<td>COMMAND DECODER</td>
</tr>
<tr>
<td>16</td>
<td>COMMAND DECODER</td>
</tr>
<tr>
<td>17</td>
<td>COMMAND DECODER</td>
</tr>
<tr>
<td>20</td>
<td>COMMAND DECODER</td>
</tr>
<tr>
<td>21</td>
<td>COMMAND DECODER</td>
</tr>
<tr>
<td>22</td>
<td>COMMAND DECODER</td>
</tr>
<tr>
<td>176</td>
<td></td>
</tr>
<tr>
<td>177</td>
<td>DN1 (USER)</td>
</tr>
<tr>
<td>200</td>
<td>SAM1 (USER)</td>
</tr>
<tr>
<td>201</td>
<td>DN2 (USER)</td>
</tr>
<tr>
<td>202</td>
<td>DN3 (USER)</td>
</tr>
<tr>
<td>373</td>
<td>SCRATCH BLOCK</td>
</tr>
<tr>
<td>374</td>
<td>SCRATCH BLOCK</td>
</tr>
<tr>
<td>375</td>
<td>SCRATCH BLOCK</td>
</tr>
<tr>
<td>376</td>
<td></td>
</tr>
<tr>
<td>377</td>
<td>FIRST DISK</td>
</tr>
</tbody>
</table>

**Figure B-1** Disk Storage Layout

DN = Directory Name Block
SAM = Storage Allocation Map Block

AREA AVAILABLE FOR SAVING CORE IMAGES
<table>
<thead>
<tr>
<th>BLOCK</th>
<th>MONITOR HEAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>MONITOR (1ST PAGE OF SAVE)</td>
</tr>
<tr>
<td>1</td>
<td>MONITOR (START)</td>
</tr>
<tr>
<td>2</td>
<td>DN</td>
</tr>
<tr>
<td>3</td>
<td>SAM</td>
</tr>
<tr>
<td>4</td>
<td>SCRATCH BLOCK</td>
</tr>
<tr>
<td>5</td>
<td>SCRATCH BLOCK</td>
</tr>
<tr>
<td>6</td>
<td>SCRATCH BLOCK</td>
</tr>
<tr>
<td>7</td>
<td>SCRATCH BLOCK</td>
</tr>
<tr>
<td>10</td>
<td>MONITOR (2ND PAGE OF CALL)</td>
</tr>
<tr>
<td>11</td>
<td>MONITOR (3RD PAGE OF SAVE)</td>
</tr>
<tr>
<td>12</td>
<td>MONITOR (2ND PAGE OF SAVE)</td>
</tr>
<tr>
<td>13</td>
<td>MONITOR (1ST PAGE OF CALL)</td>
</tr>
<tr>
<td>14</td>
<td>MONITOR (4TH PAGE OF SAVE)</td>
</tr>
<tr>
<td>15</td>
<td>LOADER</td>
</tr>
<tr>
<td>16</td>
<td>LOADER</td>
</tr>
<tr>
<td>17</td>
<td>LOADER</td>
</tr>
<tr>
<td>20</td>
<td>COMMAND DECODER</td>
</tr>
<tr>
<td>21</td>
<td>COMMAND DECODER</td>
</tr>
<tr>
<td>22</td>
<td>COMMAND DECODER</td>
</tr>
<tr>
<td>23</td>
<td>COMMAND DECODER</td>
</tr>
<tr>
<td>24</td>
<td>COMMAND DECODER</td>
</tr>
<tr>
<td>25</td>
<td>COMMAND DECODER</td>
</tr>
<tr>
<td>177</td>
<td>DN₁ (USER)</td>
</tr>
<tr>
<td>200</td>
<td>SAM₁ (USER)</td>
</tr>
<tr>
<td>201</td>
<td>DN₂ (USER)</td>
</tr>
<tr>
<td>202</td>
<td>SAM₂ (USER)</td>
</tr>
<tr>
<td>203</td>
<td>SAM₃ (USER)</td>
</tr>
<tr>
<td>204</td>
<td>SAM₄ (USER)</td>
</tr>
<tr>
<td>205</td>
<td>SAM₅ (USER)</td>
</tr>
<tr>
<td>206</td>
<td>SAM₆ (USER)</td>
</tr>
<tr>
<td>207</td>
<td>DN₃ (USER)</td>
</tr>
</tbody>
</table>

**DN** = Directory Name Block  
**SAM** = Storage Allocation Map Block

Figure B-2  DECtape Storage Layout
| BLOCK NUMBER OF FIRST SCRATCH BLOCK (DISK = 0373; DECTAPE = 0005) |
| --- | --- |
| 2-DIGIT VERSION NUMBER |
| BLOCK NUMBER OF FIRST SAM BLOCK (0200) |

**FIRST DIRECTORY NAME (DN) BLOCK**

- 25 Entries per DN
- DN Entry for First File (Internal File Number = 01)
- DN Entry for Second File (Internal File Number = 02)
- DN Entry for Third File (Internal File Number = 03)
- DN Entry for Twenty-Fifth File (Internal File Number = 318)

**DN Entry Format**

- N = 4-CHARACTER FILENAME
- Start Address
- Entry Point
- Internal File Number

**Program Type**

- 00 = ASCII
- 10 = FTC BIN
- 01 = BINARY
- 11 = SYS OR USER SAVE FILE

**Figure B-3 Directory Name (DN) Block Format**
B.1.2 Storage Allocation Map (SAM) Blocks

SAM blocks contain a record of which files are occupying which blocks on the system device. Each SAM block contains a record of a 377₈-block area. (See Figure B-4.)

- SAM₁ contains the map for blocks 0 through 377₈ and a link to SAM₂.
- SAM₂ contains the map for blocks 400 through 777₈ and a link to SAM₃.
- SAM₃ contains the map for blocks 1000 through 1377₈ and a link to SAM₄.
- SAM₄ contains the map for blocks 1400 through 1777₈ and either an end-of-chain link of 0 (if disk) or a link to SAM₅ (if DECtape).

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

- SAM₅ contains the map for blocks 2000 through 2377₈ and a link to SAM₆.
- SAM₆ contains the map for blocks 2400 through 2701₈ 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 (SAM₁ on the first disk, SAM₂ on the second disk, etc.). When a file is to be added, a search is made through the SAM blocks for an entry containing 0 (block is unoccupied), the internal file number of the file is placed in that entry (and in as many other unoccupied entries as are needed for the file), and the storage block linking is adjusted. When a file is deleted, all SAM block entries containing the file's internal file number are set to 0. The block number of the beginning block of the SAM chain (200) is stored in the third word of the first DN block.
SPECIAL INTERNAL FILE NUMBERS:  
01 = ALL MONITOR, DN, SAM, AND SCRATCH BLOCKS  
04 = LOADER BLOCKS  
05 = COMMAND DECODER BLOCKS

```
<table>
<thead>
<tr>
<th>WORD 0</th>
<th>f_200</th>
<th>f_000</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD 1</td>
<td>f_201</td>
<td>f_001</td>
</tr>
<tr>
<td>WORD 2</td>
<td>f_202</td>
<td>f_002</td>
</tr>
<tr>
<td>WORD 3</td>
<td>f_203</td>
<td>f_003</td>
</tr>
<tr>
<td>WORD 4</td>
<td>f_204</td>
<td>f_004</td>
</tr>
<tr>
<td>WORD 5</td>
<td>f_205</td>
<td>f_005</td>
</tr>
<tr>
<td>WORD 6</td>
<td>f_206</td>
<td>f_006</td>
</tr>
<tr>
<td>WORD 7</td>
<td>f_207</td>
<td>f_007</td>
</tr>
<tr>
<td>WORD 8</td>
<td>f_210</td>
<td>f_010</td>
</tr>
</tbody>
</table>

\( f_{nnn} \) = internal file number of file occupying block  
\( nnn (0 = \text{unoccupied}) \)

EXAMPLE

<table>
<thead>
<tr>
<th>FILE #1 - BLOCKS 0, 1, 2</th>
<th>15</th>
<th>01</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE #3 - BLOCKS 5, 6, 11</td>
<td>13</td>
<td>01</td>
<td>2</td>
</tr>
<tr>
<td>FILE #4 - BLOCK 10</td>
<td>13</td>
<td>01</td>
<td>2</td>
</tr>
<tr>
<td>FILE #13 - BLOCKS 201, 202, 206, 207</td>
<td>15</td>
<td>00</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>00</td>
<td>00</td>
<td>4</td>
</tr>
<tr>
<td>FILE #15 - BLOCKS 200, 203, 205, 210</td>
<td>15</td>
<td>03</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>03</td>
<td>6</td>
</tr>
<tr>
<td>UNUSED - BLOCKS 3, 4, 7, 204, 211</td>
<td>15</td>
<td>04</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>00</td>
<td>03</td>
<td>12</td>
</tr>
</tbody>
</table>

Figure B-4 Storage Allocation Map (SAM) Block Format
Table B-1
System Device and Core Capacities

<table>
<thead>
<tr>
<th>Unit</th>
<th>Words</th>
<th>Highest Page (Block) Number (1st Page = 0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 DISK</td>
<td>32,768</td>
<td>255,₁₀ (375₈)</td>
</tr>
<tr>
<td>2 DISKS</td>
<td>65,536</td>
<td>511,₁₀ (77₃₈)</td>
</tr>
<tr>
<td>3 DISKS</td>
<td>98,304</td>
<td>767,₁₀ (137₇₈)</td>
</tr>
<tr>
<td>4 DISKS</td>
<td>131,072</td>
<td>1023,₁₀ (177₇₈)</td>
</tr>
<tr>
<td>1 DECTAPE</td>
<td>667,520</td>
<td>5215,₁₀ (2701₈)</td>
</tr>
<tr>
<td>4K CORE</td>
<td>4,096</td>
<td>31,₁₀ (37₈)</td>
</tr>
<tr>
<td>8K CORE</td>
<td>8,192</td>
<td>63,₁₀ (77₈)</td>
</tr>
<tr>
<td>12K CORE</td>
<td>12,288</td>
<td>95,₁₀ (137₇₈)</td>
</tr>
<tr>
<td>16K CORE</td>
<td>16,384</td>
<td>127,₁₀ (177₇₈)</td>
</tr>
<tr>
<td>20K CORE</td>
<td>20,480</td>
<td>159,₁₀ (237₈)</td>
</tr>
<tr>
<td>24K CORE</td>
<td>24,576</td>
<td>191,₁₀ (277₇₈)</td>
</tr>
<tr>
<td>28K CORE</td>
<td>28,672</td>
<td>233,₁₀ (337₇₈)</td>
</tr>
<tr>
<td>32K CORE</td>
<td>32,768</td>
<td>255,₁₀ (377₈)</td>
</tr>
</tbody>
</table>

B.2 DATA STRUCTURE
The data structure of each type of program file is described in the following paragraphs.

B.2.1 Source File (ASCII) Data Structure
All characters are stored in 6-bit ASCII code, two words per three paper tape frames as described below. All nonprinting characters (200 through 237 and 340 through 377) have their two most significant bits dropped and a 77 prefixed to them. (The one exception to this rule is RUBOUT, 377, which is nonexistent.) All printing characters are trimmed to six bits, except for ? (277), which is packed as 7777.

B.2.2 Binary File (BINARY, FTC BIN) Data Structure
All binary (BINARY) and FORTRAN binary (FTC BIN) files are stored as two words per three paper tape frames. Frame 1 contains the rightmost eight bits of word 1, frame 2 contains the rightmost eight bits of word 2, and frame 3 contains the leftmost four bits of words 1 and 2 (the most significant bits of frame 3 are those of word 2).
Example:

<table>
<thead>
<tr>
<th>Paper tape</th>
<th>Meaning</th>
<th>Disk (Octal)</th>
<th>Disk (Binary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Leader</td>
<td>5600</td>
<td>1011 10000000</td>
</tr>
<tr>
<td>102</td>
<td>ORG</td>
<td>0502</td>
<td>0001 01000010</td>
</tr>
<tr>
<td>033</td>
<td>Second half of ORG word</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Filename</th>
<th>START ADDRESS</th>
<th>ENTRY POINT</th>
<th>FILE TYPE/FILE NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 2 0 0</td>
<td>0 4 3 3</td>
<td>6 0 2 5</td>
</tr>
</tbody>
</table>

DIRECTORY NAME ENTRY

SYSTEM DEVICE BLOCKS

Figure B-5 Contiguous-Page Save File Format

B-8
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

<table>
<thead>
<tr>
<th>Block 1</th>
<th>Block 2</th>
<th>Block 3</th>
<th>Block 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 List of page assignments</td>
<td>Contains core image of locations 0 through 177</td>
<td>Contains core image of locations 400 through 577</td>
<td>Contains core image of locations 1000 through 1177</td>
</tr>
<tr>
<td>0400</td>
<td>0 through 177</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>400 through 577</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7777</td>
<td>1000 through 1177</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Filename</th>
<th>Start Address</th>
<th>Entry Point</th>
<th>File Type/File Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILN: 0,400,1000;433</td>
<td>7 7 7 7</td>
<td>0 4 3 3</td>
<td>6 0 2 6</td>
</tr>
</tbody>
</table>

DIRECTORY NAME ENTRY

SYSTEM DEVICE BLOCKS

Figure B-6 Noncontiguous-Page Save File Format
B.3 **PIP DIRECTORY LISTING**

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

```
.PIP
*OPT-L
*IN-S:
FB=0110

110 free blocks remain
```

![Diagram of directory listing format]

**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

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

(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 filenameJ

(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)
IF AN OLD FILE IS BEING OVERWRITTEN CONTINUE SEARCHING SAMS FOR OLD FILE NUMBER AND CHANGE TO $\emptyset$

Figure B-11 Monitor Flow Chart (Part 3)
APPENDIX C
COMMAND DECODER

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

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

a. The internal file number of Command Decoder (filename = .CD.) is obtained.
b. The starting block of the Command Decoder file is obtained.
c. This block is then read into the second of the four pages to be used by Command Decoder. Command Decoder is position-independent and can be read into any four contiguous pages of core between locations 200 and 7577 inclusive.
d. Command Decoder is then entered by jumping to the second location of page 2 (the first location is an error return).

C.1 LOCATIONS USED BY COMMAND DECODER

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

<table>
<thead>
<tr>
<th>Location</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>167</td>
<td>Preloaded with 7777 if input and output filenames and extension names are different.</td>
</tr>
<tr>
<td>170</td>
<td>Scratch location.</td>
</tr>
<tr>
<td>171</td>
<td>Scratch location.</td>
</tr>
<tr>
<td>172</td>
<td>Points to the first block of Command Decoder.</td>
</tr>
<tr>
<td>173</td>
<td>Scratch location.</td>
</tr>
<tr>
<td>174</td>
<td>Points to the output list. Information concerning each device request is placed in this list by Command Decoder.</td>
</tr>
<tr>
<td>175</td>
<td>Contains the option bits. This location is not left in its original state upon exit from Command Decoder.</td>
</tr>
<tr>
<td>176</td>
<td>Scratch location.</td>
</tr>
<tr>
<td>177</td>
<td>Contains the address of the return from Command Decoder.</td>
</tr>
</tbody>
</table>
C.2 INPUT AND OUTPUT REQUIREMENTS FOR COMMAND DECODER

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

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

Bits 0 and 1    Contain output file extension code (or input, if no output is requested).\(^1\)

Bits 2 and 3    Contain the input file extension code.\(^1\)

Bit 4          1 = Output file is expected (Command Decoder will type *OUT- query (in addition to *IN-)).

Bit 5          1 = Saved output file is a system program (bit 5 of word 4 in DN entry is set to 1).

Bit 6          1 = Option is available (Command Decoder will type *OPT-).

Bit 7          1 = Saved input file is a system program (bit 5 of word 4 in DN entry is checked for a 1).

Bits 8-11      (Total number of input files allowed) - 1.

---

\(^1\)Extension codes:

00 = ASCII

01 = BINARY

10 = FTC BIN

11 = Saved file (USER, SYS)

\(^2\)1 = System, 0 = User
Figure C-1  Output List Produced by Command Decoder

<table>
<thead>
<tr>
<th>Device Code</th>
<th>Unit Number</th>
<th>Internal File Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6 = System Device  
4 = Not System Device

FOR TELETYPewriter OR HIGH SPEED READER:

<table>
<thead>
<tr>
<th>Device Code</th>
<th>Device Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Teletype</td>
</tr>
<tr>
<td>2</td>
<td>High Speed Reader</td>
</tr>
<tr>
<td>0</td>
<td>Null Device</td>
</tr>
</tbody>
</table>

1 = Teletype  
2 = High Speed Reader  
0 = Null Device

DISK OR DECTAPE (INPUT):

START BLOCK NUMBER

0

DISK OR DECTAPE (OUTPUT):

0

PAPER TAPE:

0
Figure C-2  Command Decoder Core Usage
Figure C-3  Command Decoder Flow Chart (Part 1)
GETX (READ A CHARACTER FROM THE TELETYPewriter KEYBOARD)

SET UP COUNTERS AND POINTERS

READ ONE CHARACTER

CONTROL CHARACTER?

YES

LINE FEED

DISPATCH

START PAGE 1

(BLOCK 2)

NO

CARRIAGE RETURN

RUB OUT

PAGE 2

(BLOCK 1)

END

CONVERT TO 6-BIT FORMAT

LEGAL?

NO

TYPE "?"

START PAGE 1

(BLOCK 2)

END

(CARRIAGE RETURN FOUND IN INPUT STRING)

RETURN

STORE LAST DEVICE/NAME VALUE

CODPA 2

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

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

PASS 1

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

PASS 2

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

The ST= address has a double significance.

a. It initially sets the field bit indicator for loading.

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

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

ST=10000  Begin loading in field 1 and jump to Monitor start (7600) after loading.

ST=31015  Begin loading in field 3 and jump to location 1015, field 3, after loading.

ST=27600  Begin loading in field 2 and jump to location 7600, field 2, after loading.
Figure D-1 Binary Loader Flow Chart (Part 1)
Figure D-1 Binary Loader Flow Chart (Part 2)
Figure D-1  Binary Loader Flow Chart (Part 3)
## APPENDIX E
### SYSTEM PROGRAMS

### E.1 LOADING STATISTICS

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

### E.2 SAVE STATISTICS

- PIP  SAVE PIP!0,1000-4777;1000
- EDIT SAVE EDIT!0-3177;2600
- PALD SAVE PALD!0-3377,3600-4377,4600,5200,6200-6577,7000-7577;6200
- FORT SAVE FORT!0-1777;200
- .FT. SAVE .FT. !200-7377;0
- STBL SAVE STBL!600;600
- FOSL SAVE FOSL!0-1577;200
- .OS. SAVE .OS. !0-5177;0
- DIAG SAVE DIAG!1200-1177;200
.DDT  SAVE .DDT!200-4577;0
.SYM  SAVE .SYM!200-4577;0
DDT  SAVE DDT!7200-7577;0  (User may assemble anywhere above location 4577)
READER'S COMMENTS

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

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

____________________________________

____________________________________

____________________________________

____________________________________

Did you find this manual easy to use?

____________________________________

____________________________________

____________________________________

What is the most serious fault in this manual?

____________________________________

____________________________________

____________________________________

What single feature did you like best in this manual?

____________________________________

____________________________________

____________________________________

Did you find errors in this manual? Please describe.

____________________________________

____________________________________

____________________________________

Please describe your position.

Name ______________________

Organization ______________________

Street ______________________

State ______________________ Zip ______________________