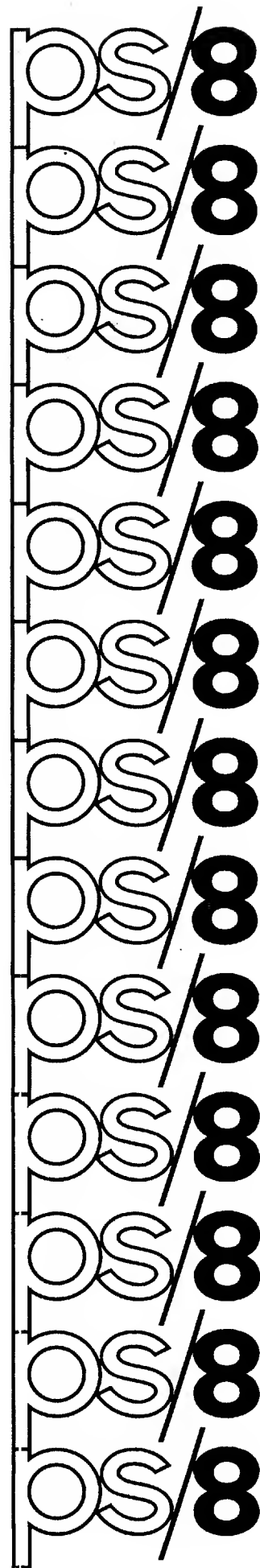



digital

software support manual

digital equipment corporation



PS/8 SOFTWARE SUPPORT MANUAL

For additional copies, order No. DEC-08-MEXB-D from the
Program Library, Digital Equipment Corporation, Maynard,
Massachusetts, 01754. Price: 

First edition, July 1970
Second edition, April 1971

Copyright © 1970, 1971, 1972 by
Digital Equipment Corporation

The following are trademarks of Digital Equipment Corporation, Maynard, Massachusetts:

DEC	PDP	COMPUTERLAB
FLIP CHIP	FOCAL	DIGITAL
UNIBUS		OMNIBUS

CONTENTS

CHAPTER 1	PS/8 CONCEPTS AND TERMINOLOGY	1-1
	1.1 Software Components of PS/8	1-1
	1.2 Files	1-2
	1.2.1 File Names and Extensions	1-2
	1.2.2 File Structured Devices	1-3
	1.2.3 File Types	1-3
	1.2.4 File Directories and Additional Information Words	1-4
	1.3 Core Control Block	1-5
	1.3.1 Program Starting Address	1-6
	1.3.2 Job Status Word	1-6
	1.4 Device Names and Device Numbers	1-7
	1.5 The DEVICE and FILENAME Pseudo-ops	1-8
CHAPTER 2	USER SERVICE ROUTINE	2-1
	2.1 Calling the USR	2-1
	2.1.1 Standard USR Call	2-1
	2.1.2 Direct and Indirect Calling Sequence	2-2
	2.2 Summary of USR Functions	2-3
	2.2.1 FETCH Device Handler	2-4
	2.2.2 LOOKUP Permanent File	2-7
	2.2.3 ENTER Output (Tentative) File	2-8
	2.2.4 The CLOSE Function	2-10
	2.2.5 Call Command Decoder (DECODE)	2-12
	2.2.6 CHAIN Function	2-13
	2.2.7 Signal User ERROR	2-14
	2.2.8 Lock USR in Core (USRIN)	2-15
	2.2.9 Dismiss USR from Core (USROUT)	2-16
	2.2.10 Ascertain Device Information (INQUIRE)	2-17
	2.2.11 RESET System Tables	2-18

CHAPTER 3	THE COMMAND DECODER	3-1
3.1	Command Decoder Convention	3-1
3.2	Command Decoder Error Messages	3-3
3.3	Calling the Command Decoder	3-4
3.4	Command Decoder Tables	3-5
3.4.1	Output Files	3-5
3.4.2	Input Files	3-6
3.4.3	Command Decoder Option Table	3-7
3.4.4	Example	3-8
3.5	Special Mode of the Command Decoder	3-9
3.5.1	Calling the Command Decoder Special Mode	3-10
3.5.2	Operation of the Command Decoder in Special Mode	3-10
CHAPTER 4	USING DEVICE HANDLERS	4-1
4.1	Calling Device Handlers	4-1
4.2	Device Dependent Operations	4-4
4.2.1	Teletype (TTY)	4-4
4.2.2	High-Speed Paper Tape Reader (PTR)	4-5
4.2.3	High-Speed Paper Tape Punch (PTP)	4-5
4.2.4	Line Printer (LPT)	4-6
4.2.5	Card Reader (CDR)	4-7
4.2.6	File Structured Devices	4-7
CHAPTER 5	RECONFIGURING THE PS/8 SYSTEM	5-1
5.1	Conditional Assembly of CONFIG	5-1
5.1.1	System Device Selection	5-2
5.1.2	Optional Device Parameters	5-3
5.1.3	Other Options	5-5
5.1.4	Example	5-6
5.2	Building a System on DECtape or LINCTape	5-7
5.3	Adding New Device Handlers	5-7
5.3.1	Writing Device Handlers	5-8
5.3.2	Editing Device Handlers Into CONFIG	5-12

APPENDIX A	PS/8 FILE STRUCTURES	A-1
	A.1 File Directories	A-1
	A.1.1 Directory Entries	A-2
	A.1.2 Number and Size of PS/8 Files	A-3
	A.1.3 Sample Directory	A-4
	A.2 File Formats	A-5
	A.2.1 ASCII and Binary Files	A-5
	A.2.2 Core Image (.SV format) Files	A-6
	A.2.3 Relocatable FORTRAN Library File	A-8
APPENDIX B	DETAILED LAYOUT OF THE SYSTEM	B-1
	B.1 Layout of the System Device	B-1
	B.2 Layout of the PS/8 Resident Program	B-2
	B.3 System Device Tables	B-3
	B.3.1 Permanent Device Name Table	B-3
	B.3.2 User Device Name Table	B-4
	B.3.3 Device Handler Residency Table	B-5
	B.3.4 Device Handler Information Table	B-5
	B.3.5 Device Control Word Table	B-6
	B.3.6 Device Length Table	B-7
APPENDIX C	SYSTEM ERROR CONDITIONS AND MESSAGES	C-1
	C.1 System Halts	C-1
	C.2 USR Errors	C-2
	C.3 Keyboard Monitor Errors	C-4
	C.4 Command Decoder Errors	C-5
APPENDIX D	PROGRAMMING NOTES	D-1
	D.1 The Default File Storage Device, DSK	D-2
	D.2 Modification to Card Reader Handler	D-2
	D.3 Suppression of Carriage Return/Line Feed in FORTRAN	D-5
	D.4 Accessing the System Date in a FORTRAN Program	D-6

D.5	Determining Core Size on PDP-8 Family Computers	D-7
D.6	Relocating Code	D-9
D.7	Using PRTCl2-F to Convert PS/8 DECTapes to PS/12 LINCTapes	D-10
D.8	Notes on Loading Device Handlers	D-11
D.9	Available Locations in the USR Area	D-13
D.10	Accessing Additional Information Words in PS/8	D-14
D.11	SABR Programming Notes	D-16
APPENDIX E	CHARACTER CODES AND CONVENTIONS	E-1
APPENDIX F	<u>DOCUMENTATION UPDATE FOR THE 8K PROGRAMMING SYSTEM USER'S GUIDE</u>	F-1
F.1	CREF, Cross-Reference Program (DEC-P8-YRXA-PB)	F-1
F.1.1	Loading, Calling, and Using CREF	F-1
F.1.2	Interpreting CREF Output	F-3
F.1.3	CREF Pseudo-ops	F-5
F.1.4	Restrictions	F-6
F.2	LIBSET (DEC-P8-SYXB-PB)	F-9
F.2.1	Loading, Calling, and Using LIBSET	F-9
F.2.2	Examples of LIBSET Usage	F-10
F.2.3	Subroutine Names	F-11
F.2.4	Sequence for Loading Subroutines	F-11
F.2.5	LIBSET Error Messages	F-12

INTRODUCTION

The 8K Programming System PS/8, is an extremely powerful program development system. PS/8 greatly expands the capabilities of any 8K PDP-8, 8/I, 8/L, 8/E, or PDP-12 computer having the necessary disk or DECTape storage. Use of PS/8 is described in detail in the 8K Programming System User's Guide, with which the reader should be familiar.

This manual covers a wide range of advanced topics pertinent to the experienced user. In Chapter 1 the various basic system concepts are described and terms defined. Chapter 2 explains the process by which user programs call upon the system for the performance of important operations; including loading device handlers, opening and closing files, and chaining to other programs. Chapter 3 covers the functions of the Command Decoder and the means by which the user program can employ its services. Chapter 4 explains the use and operation of the device handlers in detail. Chapter 5 covers the details of "custom tailoring" a system, including how to write a device handler for a non-standard device.

Technical information, intended to enhance the information in the 8K Programming System User's Guide as well as this manual, can be found in the Appendices. Appendix A details the PS/8 directory structure and gives standard file formats. Appendix B describes the system data base and gives the layouts of the system areas. Appendix C gives a complete list of system error messages. Appendix D illustrates some useful advanced techniques and programming "tricks" for use with the PS/8 system. Appendix E is a complete list of the standard ASCII character codes meaningful to PS/8. Finally, Appendix F contains additions to the 8K Programming System User's Guide, sections covering the CREF and Library Setup programs that will be included in later editions of the User's Guide.

PS/8 does not attempt to solve all problems. Since it runs with the interrupt disabled, any calls to the system must be made with the interrupt turned off. Nonetheless, PS/8 provides a powerful means to solve the most frequently encountered problems in developing software for PDP-8 family computers.

CHAPTER 1

PS/8 CONCEPTS AND TERMINOLOGY

Before examining the details of the PS/8 system the reader should first be familiar with the simpler techniques and terms used within the framework of the PS/8 system. The material in this chapter, along with that contained in the 8K Programming System User's Guide, provides the tools needed to pursue the later chapters.

1.1 Software Components of PS/8

There are four main components of the PS/8 system:

- a. The Keyboard Monitor performs commands specified by the user at the console. The nine Keyboard Monitor commands (ASSIGN, DEASSIGN, GET, SAVE, ODT, RUN, R, START, and DATE) are explained in Chapter 2 of the 8K Programming System User's Guide.

User programs can exit to the Keyboard Monitor by executing a JMP to location 7600 in field 0.

NOTE

All JMPs to 7600 must be made
with the DATA FIELD set to zero.

This saves the contents of location 0 to 1777 in field 0 and loads the Keyboard Monitor which could be called by a JMP to location 7605 in field 0. In this latter case the contents of core are not saved, which conserves some time.

Existing system programs, device handlers, and the Command Decoder test for the CTRL/C character in the Teletype* input buffer and on finding this character abort the current operation and perform a JMP to 7600 in field 0. Thus, typing CTRL/C is the conventional method of calling the Keyboard Monitor from the console.

*Teletype is a registered trademark of the Teletype Corporation.

- b. Device handlers, which are subroutines for performing all device-oriented input/output operations, can be utilized by any program. These subroutines have standard calling sequences and "mask" from the user program the special characteristics of the I/O device. In this way, device independent I/O is achieved. A detailed description of device handlers is found in Chapter 4.
- c. The User Service Routine (USR) is to a program what the Keyboard Monitor is to the user. For example, programs can request the USR to fetch device handlers, perform file operations on any device, chain to another program, or call the Command Decoder. A full description of the USR functions is found in Chapter 2.
- d. The Command Decoder interprets a command line typed by the user to indicate input and output files and various options. The command line format is described in detail in Chapter 3 of the 8K Programming System User's Guide. The Command Decoder removes the burden of this repetitive operation from the user's program. A full description of the Command Decoder's function is found in Chapter 3.

1.2 Files

Files are basic units of the PS/8 system, and a thorough understanding of file structure is required for its use. A file is any collection of data to be treated as a unit. The format of this data is unimportant; for example, PS/8 can manipulate several standard formats, including ASCII files, binary files, and core image files. The important consideration is that the data forms a single unit within the system.

1.2.1 File Names and Extensions

An individual file is identified by its file name and extension. The file name consists of up to six alphanumeric characters, optionally followed by a two character extension. The extension is often used to clarify the format of the data within the file. For example, an ASCII file used as input to PAL8 might be given a .PA extension, while a core image file has a .SV extension.

1.2.2 File Structured Devices

Devices that can be logically divided into a number of 256 word blocks and have the ability to read and write from any desired block are called file structured devices. Disks and DECtapes are file structured devices while a paper tape reader or Teletype is not.

The system device (SYS:) in any PS/8 system is always file structured.

All PS/8 file structured devices must be logically divided into these 256 word blocks. Hence, 256 words is considered the standard PS/8 block size. Some devices, like RK8, DECtape, and LINCtape, are physically divided into blocks. These physical blocks should not be confused with the logical 256-word blocks. For example, DECtapes must be formatted with standard 129 word physical blocks. A logical PS/8 block consists of the first 128 words of two consecutive physical DECtape blocks. The 129th word of every DECtape block is not used by PS/8. Similarly, LINCtapes are formatted with 129 (or 128) words per block but never 256, as this format is unacceptable to PS/8.

A given PS/8 file consists of one or more sequential blocks of 256 words each (consecutively numbered). A minimum of one block per file is required, although a single file could occupy all of the blocks on a device.

1.2.3 File Types

Three different types of files exist in the PS/8 system:

- a. An empty file is a contiguous area of unused blocks. Empty files are created when permanent files are deleted.

- b. A tentative file is a file that is open to accept output and has not yet been closed. Only one tentative file can be open on any single device at one time.
- c. A permanent file is a file that has been given a fixed size and is no longer expandable. A tentative file becomes permanent when it is closed.

To further understand file types, consider what occurs when a file is created. Normally, the User Service Routine in creating a tentative file first locates the largest empty file available and creates a tentative file in that space. This insures the maximum space into which the file can expand. The user program then writes data into the tentative file. At the end of the data, the program calls the USR to close the tentative file, making it a permanent file. The USR does so and allocates whatever space remains on the end of the tentative file to a new, smaller, empty file.

1.2.4 File Directories and Additional Information Words

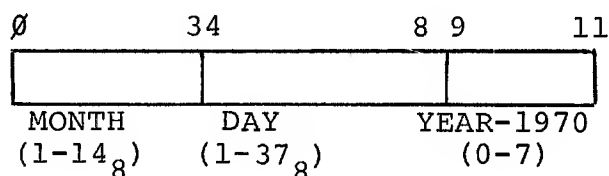
To maintain records of the files on a device, PS/8 allocates blocks 1 through 6 of each file structured device as the file directory. Entries in this directory inform the system of the name, size, and location of each file, including all empty files and the tentative file, if one exists. For a detailed description of the entries in the file directory, see Appendix A.

Each entry in a directory can, optionally, have extra storage words called Additional Information Words. The number of Additional Information Words is determined at the time the directory is initially created (normally by using the /S or /Z features of PIP; see Chapter 6 in the 8K Programming System User's Guide).

Whenever Additional Information Words are used, the first one for each file entry is used to store the value of the System Date Word at the time the file was created. This value is set by executing a DATE command (see Chapter 2 in the 8K Programming System User's Guide).

NOTE

The value of the system DATE word is contained in location 7666 in field 1; this word has the following format:



A date word of Ø implies that no DATE command has been executed since the system initialization.

The values of Additional Information Words beyond the first are user-defined. See Appendix D for further information on Additional Information Words.

1.3 Core Control Block

Associated with each core image file (.SV file) is a block of data called the Core Control Block. The Core Control Block is a table of information containing the program's starting address, areas of core into which the program is loaded, and the program's Job Status Word. The Core Control Block is created at the time the program is loaded by the ABSLDR or LOADER program and is written onto the .SV file by the SAVE operation. More information on the Core Control Block can be found in the description of core image files in section A.2.2.

NOTE

Specifying arguments to the SAVE command as described in the 8K Programming System User's Guide can alter the contents of that program's Core Control Block.

When a program is loaded the starting address and Job Status Word are loaded from the Core Control Block and saved in core. The Core Control Block itself is saved in one of the system scratch blocks unless the program was loaded with the R (rather than GET or RUN) command.

1.3.1 Program Starting Address

The current starting address (used by the START command) is stored in two words at locations 7744 and 7745 in field 0. The format of these words is:

<u>LOCATION</u>	<u>CONTENTS</u>	<u>NOTES</u>
7744	62N3	N is the field in which to start
7745	addr	Starting address of the program

1.3.2 Job Status Word

The Job Status Word contains certain flags that affect PS/8 operations, such as whether to save core when loading the USR or Command Decoder. The Job Status Word for the program currently in core is saved at location 7746 in field 0 and contains the following information:

<u>Bit Condition</u>	<u>Meaning</u>
Bit 0 = 1	File does not load into locations 0 to 1777 in field 0.
Bit 1 = 1	File does not load into locations 0 to 1777 in field 1.
Bit 2 = 1	Program must be reloaded before it can be restarted.

<u>Bit Condition</u>	<u>Meaning</u>
Bits 3-9	Reserved for future use.
Bit 10 = 1	Locations 0 to 1777 in field 0 need not be saved when calling the Command Decoder.
Bit 11 = 1	Locations 0 to 1777 in field 1 need not be saved when calling the USR.

More information on the Core Control Block can be found in the description of Core Image (.SV) files found in Appendix A.

NOTE

When bit 2 is 1, any attempt to perform a START (without an explicit address) results in a

NO!!

error message being printed. As this bit is always zero in the Core Control Block, the user program is expected to internally set this bit (in location 7746) if a program is not restartable. This could be done as follows:

```

CDF 0
TAD 7746      /LOAD JOB STATUS WORD
AND (6777
TAD (10000
DCA 7746      /JOB IS NOT RESTARTABLE

```

1.4 Device Names and Device Numbers

The PS/8 system can accommodate up to 15 separate devices. In the 8K Programming System User's Guide the reader is introduced to the concept of device names. Briefly, each device on the system is recognized by either a permanent device name (such as PTR or DTA1) which is created when the system is built, or a user-defined device name determined by an ASSIGN command. The system insures that the user-defined device name takes precedence. For example,

```
.ASSIGN DSK DTA4
```

causes all future references to DTA4 to address the device DSK.

In calling the User Service Routine, a device can be alternatively recognized by a device number. Each device on the system has a unique predefined number in the range 1 to 15 (17_8) assigned at the time the system is generated.

Thus, user programs have the choice of referring to a device by either name or number. Referencing a device by name is preferable as it maintains device independence for the program.

1.5 The DEVICE and FILENAME Pseudo-ops

Several of the User Service Routine functions take device names or file names as arguments. To simplify the task of generating these arguments, the DEVICE and FILENAME pseudo-ops have been added to the PAL8 Assembler.

A device name consists of a two word block, containing four alphanumeric characters in six-bit ASCII format. A block in this format can be created by the DEVICE pseudo-op as follows:

```
DEVICE DTA1
```

generates the following two words:

```
ø424  
ø161
```

Similarly, the FILENAME pseudo-op creates a four word block, the first three words of which contain the file name and the fourth word of which contains the file extension. For example:

```
FILENAME PIP.SV
```

generates the following four words:

2011
2000
0000
2326

Note that positions for characters 4 through 6 are filled with zeros.

The DEVICE and FILENAME pseudo-ops are used in examples in the following chapters.

CHAPTER 2

User Service Routine

The User Service Routine, or USR, is a collection of sub-routines which perform the operations of opening and closing files, loading device handlers, chaining programs together, and calling the Command Decoder. It provides these functions not only for the system itself, but for all programs running under the PS/8 system.

2.1 Calling the USR

Performing any USR function is as simple as giving a JMS followed by the proper arguments. Calls to the USR take a standardized calling sequence. This standard call should be studied before progressing to the operation of the various USR functions.

2.1.1 Standard USR Call

In the remainder of this chapter, the following calling sequence is referenced:

TAD VAL	The contents of the AC is applicable in some cases only.
CDF N	Where N is the value of the current program field multiplied by 10 octal.
CIF 1Ø	
JMS I (USR	Where USR is either 7700 or 0200, see section 2.1.2).
FUNCTION	This word contains an integer from 1 to 13 octal indicating which USR operation is to be performed.
ARG(1) ⋮ ⋮	The number and meaning of these argument words varies with the particular USR function to be performed.
error return	When applicable, this is the return address for all errors.
normal return	

This calling sequence can change from function to function. For example, some functions take no value in the AC and others have fewer or greater numbers of arguments. Nonetheless, this format is generally followed.

NOTE

The CDF and CIF instructions preceding the JMS to the USR in the calling sequence are very important. The DATA FIELD must be set to the current field and the INSTRUCTION FIELD must be set to 1 when calling the USR. In the examples given in this chapter, it is arbitrarily assumed that the call to the USR is made from field 0. On return from any USR function the DATA FIELD remains set to the current field and the AC is zero.

There are three other restrictions which apply to all USR calls, as follows:

- a. The USR can never be called from any address between 10000 and 11777*. Attempting to do so results in the printing of the:

MONITOR ERROR 4 AT xxxxx

message and termination of program execution. The value of xxxxx is the address of the calling sequence (in all such MONITOR ERROR messages).

- b. Several USR calls take address pointers as arguments. These pointers always refer to data in the same field as the call.
- c. When calling the USR from field 1, these address pointers must never refer to data that lies in the area 10000 to 11777.

2.1.2 Direct and Indirect Calling Sequence

A user program can call the USR in two ways. First, by performing a JMS to location 17700 (location 7700 in field 1, addresses in field 1 are written in this format). In this case,

*Where five digit addresses are given the leading digit refers to the field. Thus, 11777 is location 1777 in field 1.

locations 10000 to 11777 are saved on a special area on the system device, and the USR is then loaded into 10000 to 11777. When the USR operation is completed, locations 10000 to 11777 are restored to their previous values.

NOTE

By setting bit 11 of the Job Status Word to a 1, the user can avoid this saving and restoring of core when preserving core is unnecessary.

Alternatively, a program can opt to keep the USR permanently resident in core at locations 10000 to 11777 by using the USRIN function (see section 2.2.8). Once the USR has been brought into core, a USR call can be made by performing a JMS to location 10200. This is a more efficient way of calling the USR. When USR operations have been completed, the program restores locations 10000 to 11777 to their initial state by executing the USROUT function (see section 2.2.9).

2.2 Summary of USR Functions

<u>Function Code</u>	<u>Name</u>	<u>Operation</u>
1	FETCH	Load a device handler into core. Returns entry address of the handler.
2	LOOKUP*	Searches the file directory on any device to locate a specified permanent file.
3	ENTER*	Creates and opens for output a tentative file on a specified device.
4	CLOSE*	The currently open tentative file on the specified device is closed and becomes a permanent file. Also, any previous permanent file with the same name is deleted.

*If the specified device is not file structured, the LOOKUP, ENTER, and CLOSE functions verify only that the device is acceptable for input in the case of LOOKUP, or output in the case of ENTER or CLOSE. For example, ENTERing a file on the paper tape punch is a legal function.

<u>Function Code</u>	<u>Name</u>	<u>Operation</u>
5	DECODE	The Command Decoder is called. The function of the Command Decoder is described in Chapter 3.
6	CHAIN	Loads a specified core image file from the system device and starts it.
7	ERROR	Prints an error message of the form USER ERROR n AT LOCATION xxxxx
10	USRIN	The USR is loaded into core. Subsequent calls to the USR are by a JMS to 10200.
11	USROUT	Dismisses the USR from core and restores the previous contents of locations 10000 to 11777.
12	INQUIRE	Ascertains whether a given device exists and, if so, whether its handler is in core.
13	RESET	Resets system tables to their initial cleared state.
14-17		Not currently used, these request numbers are reserved for future use.

2.2.1 FETCH Device Handler

Function Code = 1

Device handlers must be loaded into core so as to be available to the USR and user program for I/O operations on that device. Before performing a LOOKUP, ENTER, or CLOSE function on any device, the handler for that device must be loaded by FETCH.

The FETCH function takes two distinct forms:

- a. Load a device handler corresponding to a given device name, and
- b. Load a device handler corresponding to a given device number.

First, the following is an example of loading a handler by name:

```
CLA                      /AC MUST BE CLEAR
CDF 0
CIF 10
JMS I (USR
1                        /FUNCTION CODE = 1
DEVICE DTA3             /GENERATES TWO WORDS: ARG(1)
                        /AND ARG(2)
60001                   /ARG(3)
JMP ERR                 /ERROR RETURN
:                       /NORMAL RETURN
:
```

ARG(1) and ARG(2) contain the device name in standard format. If the normal return is taken, ARG(2) is changed to the device number corresponding to the device loaded. ARG(3) contains the following information:

Bits 0 to 4 contain the page number into which the handler is loaded.

Bit 11 is 0 if the user program can only accept a one page handler.

Bit 11 is 1 if there is room for a two page handler.

Notice that in the example above, the handler for DTA3 is to be loaded into locations 6000 to 6177. If necessary, a two page handler could be loaded; the second page would be placed in locations 6200 to 6377. After a normal return, ARG(3) is changed to contain the entry point of the handler.

A different set of arguments is used to fetch a device handler by number. The following is an example of this form:

```
TAD VAL                 /AC IS NOT ZERO
CDF 0
CIF 10
JMS I (USR
1                        /FUNCTION CODE = 1
60001                   /ARG(1)
JMP ERR                 /ERROR RETURN
:                       /NORMAL RETURN
:
```

On entry to the USR the AC contains the device number in bits 8 to 11 (bits 0 to 7 are ignored).

The format for ARG(1) is the same as that for ARG(3) in the previous example. Following a normal return ARG(1) is replaced with the entry point of the handler.

The conditions that can cause an error return to occur in both cases are as follows:

- a. There is no device corresponding to the given device name or device number, or
- b. An attempt was made to load a two page handler into one page. If this is an attempt to load the handler by name, the contents of ARG(2) have been changed already to the internal device number.

In addition, one of the following Monitor errors can be printed, followed by a return to the Keyboard Monitor:

<u>Error Message</u>	<u>Meaning</u>
MONITOR ERROR 4 AT xxxxx	Results if bits 8 to 11 of the AC are zero (and bits 0 to 7 are non-zero).
MONITOR ERROR 5 AT xxxxx	Results if a read error occurs while loading the device handler.

The FETCH function checks to see if the handler is in core, and if it is not, then the handler and all co-resident* handlers are loaded. While the FETCH operation is essentially a simple one, the user should be aware of the following points:

- a. Device handlers are always loaded into field 0.

*Two or more device handlers are "co-resident" when they are both included in the same one or two core pages. For example, the paper tape reader and punch routines are co-resident, as are the eight DECTape handler routines.

- b. The entry point that is returned may not be on the page desired. This would happen if the handler were already resident.
- c. Never attempt to load a handler into 7600 or into page 0. Never load a two page handler into 7400.

For more information on using device handlers, see Chapter 4.

2.2.2 LOOKUP Permanent File

Function Code = 2

This request locates a permanent file entry on a given device, if one exists. An example of a typical LOOKUP would be:

```

TAD VAL          /LOAD DEVICE NUMBER
CDF Ø
CIF 1Ø
JMS I (USR
2                /FUNCTION CODE = 2
NAME            /ARG(1), POINTS TO FILE NAME
Ø              /ARG(2)
JMP ERR         /ERROR RETURN
:              /NORMAL RETURN
:
NAME,FILENAME PROG.PA

```

This request looks up a permanent file entry with the name PROG.PA. The device number on which the lookup is to be performed is in AC bits 8 to 11. ARG(1) contains a pointer to the file name; note that the file name block must be in the same field as the call, and that it cannot be in locations 10000 to 11777. The device handler must have been previously loaded into core. If the normal return is taken, ARG(1) is changed to the starting block of the file and ARG(2) contains the file length in blocks as a negative number. If the device specified is a readable, non-file structured device (for example, the paper tape reader), then ARG(1) and ARG(2) are both set to zero.

If the error return is taken, ARG(1) and ARG(2) are unchanged. The following conditions cause an error return:

- a. The device specified is a write-only device.
- b. The file specified was not found.

In addition, specifying illegal arguments can cause one of the following monitor errors, followed by a return to the Keyboard Monitor:

<u>Error Message</u>	<u>Meaning</u>
MONITOR ERROR 2 AT xxxxx	Results if an I/O error occurred while reading the device directory.
MONITOR ERROR 3 AT xxxxx	Results if the device handler for the specified device is not in core.
MONITOR ERROR 4 AT xxxxx	Results if bits 8 to 11 of the AC are zero.

The LOOKUP function is the standard method of opening a permanent file for input.

2.2.3 ENTER Output (Tentative) File Function Code = 3

The ENTER function is used to create a tentative file entry to be used for output. An example of a typical ENTER function is as follows:

TAD VAL	/AC IS NOT ZERO
CDF Ø	
CIF 1Ø	
JMS I (USR	
3	/FUNCTION CODE = 3
NAME	/ARG(1) POINTS TO FILE NAME
Ø	/ARG(2)
JMP ERROR	/ERROR RETURN
:	/NORMAL RETURN
:	
NAME,FILENAME PROG.LS	

Bits 8 to 11 of the AC contain the device number of the selected device; the device handler for this device must be loaded into core before performing an ENTER function. If bits 0 to 7 of the AC are non-zero, this value is considered to be a declaration of

the maximum length of the file. The ENTER function searches the file directory for the smallest empty file that contains at least the declared number of blocks. If bits 0 to 7 of the AC are zero, the ENTER function locates the largest available empty file.

On a normal return, the contents of ARG(1) are replaced with the starting block of the file. The 2's complement of the actual length of the created tentative file in blocks (which can be equal to or greater than the requested length) replaces ARG(2). If the file directory contains any Additional Information Words, the system DATE (location 17666) is written as the first Additional Information Word of the newly created tentative file at this time.

NOTE

If the selected device is not file structured but permits output operations (e.g., the high speed punch), the ENTER operation always succeeds. In this case, ARG(1) and ARG(2) are both zeroed on return.

If the error return is taken, ARG(1) and ARG(2) are unchanged. The following conditions cause an error return:

- a. The device specified by bits 8 to 11 of the AC is a read only device.
- b. No empty file exists which satisfies the requested length requirement.
- c. Another tentative file is already active on this device (only one output file can be active at any given time).
- d. The first word of the file name was 0 (an illegal file name).

In addition, one of the following monitor errors can occur, followed by a return to the Keyboard Monitor:

<u>Error Message</u>	<u>Meaning</u>
MONITOR ERROR 2 AT xxxxx	Results if an I/O error occurred while reading or writing the device directory.
MONITOR ERROR 3 AT xxxxx	Results if the device handler for the specified device is not in core.
MONITOR ERROR 4 AT xxxxx	Results if AC bits 8 to 11 are zero.
MONITOR ERROR 5 AT xxxxx	Read error on the system device while bringing in the overlay code for the ENTER function.
MONITOR ERROR 6 AT xxxxx	Results if a directory overflow occurred (no room for tentative file entry in directory).

2.2.4 The CLOSE Function

Function Code = 4

The CLOSE function has a dual purpose: first, it is used to close the current active tentative file, making it a permanent file. Second, when a tentative file becomes permanent it is necessary to remove any permanent file having the same name; this operation is also performed by the CLOSE function. An example of CLOSE usage follows:

```

TAD VAL          /GET DEVICE NUMBER
CDF Ø
CIF 1Ø
JMS I (USR
4                /FUNCTION CODE = 4
NAME             /ARG(1)
15              /ARG(2)
JMP ERR          /ERROR RETURN
:               /NORMAL RETURN
:
NAME,FILENAME PROG.LS

```

The device number is contained in AC bits 8 to 11 when calling the USR. ARG(1) is a pointer to the name of the file to be deleted and ARG(2) contains the number of blocks to be used for the new permanent file.

The normal sequence of operations on an output file is:

- a. FETCH the device handler for the output device.
- b. ENTER the tentative file on the output device, getting the starting block and the maximum number of blocks available for the file.
- c. Perform the actual output, using the device handler, keeping track of how many blocks are written, and checking to insure that the file does not exceed the available space.
- d. CLOSE the tentative file, making it permanent. The CLOSE operation would always use the same file name as the ENTER performed in step b. The closing file length would have been computed in step

After a normal return from CLOSE, the active tentative file is permanent and any permanent file having the specified file name already stored on the device is deleted. If the specified device is a non-file structured device that permits output (the paper tape punch, for example), the CLOSE function will always succeed.

NOTE

The user must be careful to specify the same file names to the ENTER and the CLOSE functions. Failure to do so can cause several permanent files with identical names to appear in the directory. If CLOSE is intended only to be used to delete some existing file, then the number of blocks, ARG(2) should be zero.

The following conditions cause the error return to be taken:

- a. The device specified by bits 8 to 11 of the AC is a read only device.
- b. There is neither an active tentative file to be made into a permanent file, nor a permanent file with the specified name to be deleted.

In addition, one of the following Monitor errors can occur:

<u>Error Message</u>	<u>Meaning</u>
MONITOR ERROR 1 AT xxxxx	Results if the length specified by ARG(2) exceeded the allotted space.
MONITOR ERROR 2 AT xxxxx	Results if an I/O error occurred while reading or writing the device directory.
MONITOR ERROR 3 AT xxxxx	Results if the device handler for the specified device is not in core.
MONITOR ERROR 4 AT xxxxx	Results if AC bits 8 to 11 are zero.

2.2.5 Call Command Decoder (DECODE) Function Code = 5

The DECODE function causes the USR to load and execute the Command Decoder. The Command Decoder accepts (from the Teletype) a list of input and output devices and files, along with various options. The Command Decoder performs a LOOKUP on all input files, sets up necessary tables in the top page of field 1, and returns to the user program. These operations are described in detail in Chapter 3, which should be read before attempting to use the DECODE function.

A typical call to the Command Decoder looks as follows:

CDF Ø	
CIF 1Ø	
JMS I (USR	
5	/FUNCTION CODE = 5
2ØØ1	/ARG(1), ASSUMED INPUT EXTENSION
Ø	/ARG(2), ZERO TO PRESERVE ALL
	/TENTATIVE FILES
	/NORMAL RETURN
:	
:	

ARG(1) is the assumed input extension, in the above example it is ".PA". On return from the Command Decoder, information is stored in tables located in the top page of field 1. The

DECODE function also resets all system tables as in the RESET function (see RESET function, section 2.2.11) if ARG(2) is \emptyset all currently active tentative files remain open; if ARG(2) is non-zero all tentative files are deleted and the normal return is to ARG(2) instead of ARG(2)+1.

The DECODE function has no error return (Command Decoder error messages are given in Chapter 3). However, the following Monitor error can occur:

<u>Error Message</u>	<u>Meaning</u>
MONITOR ERROR 5 AT xxxxx	I/O error occurred while reading or writing on the system device.

2.2.6 CHAIN Function

Function Code = 6

The CHAIN function permits a program to load and start another program with the restriction that the program chained to must be a core image (.SV) file located on the system device. A typical implementation of the CHAIN function looks as follows:

```

CDF  $\emptyset$ 
CIF 1 $\emptyset$ 
JMS I (USR
6           /FUNCTION CODE = 6
BLOCK      /ARG(1), STARTING BLOCK NUMBER

```

There is no normal or error return from CHAIN. However, the following monitor error can occur:

<u>Error Message</u>	<u>Meaning</u>
MONITOR ERROR 5 AT xxxxx	I/O error occurred while reading or writing on the system device.

The CHAIN function loads a core image file located on the system device beginning at the block number specified as ARG(1) (which

is normally determined by performing a LOOKUP on the desired file name). Once loaded, the program is started at an address one greater than the starting address specified by the program's Core Control Block.

CHAIN automatically performs a USROUT function (see section 2.2.9) to dismiss the USR from core, and a RESET to clear all system tables (see section 2.2.11), but CHAIN does not delete tentative files.

The areas of core altered by the CHAIN function are determined by the contents of the Core Control Block of the core image file loaded by CHAIN. The Core Control Block for the file is set up by the ABSLDR or LOADER programs. It can be modified by performing a SAVE command with specific arguments. Every page of core in which at least one location was saved is loaded. If the page is one of the "odd numbered" pages (pages 1, 3, etc.; locations 0200 to 0377, 0600 to 0777, etc.), the previous page is always loaded. In addition, CHAIN always alters the contents of locations 07200 to 07577.

NOTE

CHAIN destroys a necessary part of the ODT resident breakpoint routine. Thus an ODT breakpoint should never be maintained across a CHAIN.

With the above exceptions, programs can pass data back and forth in core while chaining. For example, FORTRAN programs normally leave the COMMON area in field 1 unchanged. This COMMON area can then be accessed by the program activated by the CHAIN.

2.2.7 Signal User ERROR

Function Code = 7

The USR can be called to print a user error message for a program. The following is a possible ERROR call:

```
CDF Ø
CIF 1Ø
JMS I (USR
7          /FUNCTION CODE = 7
2          /ARG(1), ERROR NUMBER
```

The ERROR function causes a message of the form:

USER ERROR n AT xxxxx

to be printed. Here n is the error number given as ARG(1); n must be between 0 and 11₈, and xxxxx is the address of ARG(1). If ARG(1) in the sample call above was at location 500 in field 0, the message:

USER ERROR 2 AT 005000

would be printed. Following the message, the USR returns control to the Keyboard Monitor, preserving the user program intact.

The error number is arbitrary. Two numbers have currently assigned meanings:

<u>Error Message</u>	<u>Meaning</u>
USER ERROR 0 AT xxxxx	During a RUN, GET, or R command, this error message indicates that an error occurred while loading the core image.
USER ERROR 1 AT xxxxx	While executing a FORTRAN or SABR program, this error indicates that a call was made to a subroutine that was not loaded.

2.2.8 Lock USR in Core (USRIN)

Function Code = 10

When making a number of calls to the USR it is advantageous for a program to avoid reloading the USR each time a USR call is made. The USR can be brought into core and kept there for subsequent use by the USRIN function. The calling sequence for the USRIN function looks as follows:

```
CDF 0
CIF 10
JMS I (7700
10      /FUNCTION CODE = 10
:      /NORMAL RETURN
:
```

The USRIN function saves the contents of locations 10000 to 11777 on the system scratch blocks, loads the USR into this area in core, and returns control to the user program.

NOTE

If bit 11 of the current Job Status Word is a one, the USRIN function will not save the contents of locations 10000 to 11777.

Subsequent calls to the USR can be made directly by performing a JMS to location 0200 in field 1, saving the time necessary to reload the USR each time it is called.

2.2.9 Dismiss USR from Core (USROUT) Function Code = 11

When a program has loaded the USR into core with the USRIN function and no longer wants or needs the USR in core, the USROUT function is used to restore the original contents of locations 10000 to 11777. The calling sequence for the USROUT function is as follows:

```
CDF 0
CIF 10
JMS I (2000                      /DO NOT JMS TO 17700 !!
11                                /FUNCTION CODE = 11
:                                /NORMAL RETURN
:
```

The USROUT function and the USRIN function are complementary operations. Subsequent calls to the USR must be made by performing a JMS to location 7700 in field 1.

NOTE

If bit 11 of the current Job Status Word is a 1, the contents of core are not changed by the USROUT function. In this case USROUT is a redundant operation since core was not preserved by the USRIN function.

2.2.10 Ascertain Device Information (INQUIRE) Function Code = 12

On some occasions a user may wish to determine what internal device number corresponds to a given device name or whether the device handler for a specified device is in core, without actually performing a FETCH operation. INQUIRE performs these operations for the user. The function call for INQUIRE closely resembles the FETCH handler call.

INQUIRE, like FETCH, has two distinct forms:

- a. Obtain the device number corresponding to a given device name and determine if the handler for that device is in core (example shown below), and
- b. Determine if the handler corresponding to a given device number is in core.

An example of the INQUIRE call is shown below:

CLA	/AC MUST BE CLEAR
CDF Ø	
CIF 1Ø	
JMS I (USR	
12	/FUNCTION CODE = 12
DEVICE DTA3	/GENERATES TWO WORDS:
	/ARG(1) AND ARG(2)
Ø	/ARG(3)
JMP ERR	/ERROR RETURN
⋮	/NORMAL RETURN
⋮	

ARG(1) and ARG(2) contain the device name in standard format. When the normal return is taken ARG(2) is changed to the device number corresponding to the given name, and ARG(3) contains either the entry point of the device handler if it is already in core, or zero if the corresponding device handler has not yet been loaded.

A slightly different set of arguments is used to inquire about a device by its device number:

```
TAD VAL          /AC IS NON-ZERO
CDF 0
CIF 10
JMS I (USR
12              /FUNCTION CODE = 12
0              /ARG(1)
JMP ERR         /ERROR RETURN
:              /NORMAL RETURN
:
```

On entry to INQUIRE, AC bits 8 to 11 contain the device number (bits 0 to 7 are ignored).

NOTE

If AC bits 0 to 7 are non-zero, and bits 8 to 11 are zero (an illegal device number) a:

MONITOR ERROR 4 AT xxxxx

message is printed and program execution is terminated.

On normal return ARG(1) is set to the entry point of the device handler if it is already in core, or zero if the corresponding device handler has not yet been loaded.

The error return in both cases is taken only if there is no device corresponding to the device name or number specified.

2.2.11 RESET System Tables

Function Code = 13

There are certain occasions when it is desired to reset the system tables, effectively removing from core all device handlers except the system handler. An example of the RESET function is shown below:

```

CDF Ø
CIF 1Ø
JMS I (USR
13          /FUNCTION CODE = 13
Ø          /Ø PRESERVES TENTATIVE FILES
:          /NORMAL RETURN
:

```

RESET zeros all entries except the one for the system device in the Device Handler Residency Table (see section B.3.3), removing all device handlers, other than that for the system device, from core. This should be done anytime a user program modifies any page in which a device handler was loaded.

RESET has the additional function of deleting all currently active tentative files (files that have been entered but not closed). This is accomplished by zeroing bits 9 through 11 of every entry in the Device Control Word Table (see section B.3.5).

If RESET is to be used in this last fashion, to delete all active tentative files, then ARG(1) must be non-zero and the normal return is to ARG(1) rather than to ARG(1)+1. For example, the following call would serve this purpose:

```

CDF Ø
CIF 1Ø
JMS I (USR
13          /FUNCTION CODE = 13
CLA CMA     /NON-ZERO!

```

The normal return would execute the CLA CMA and all active tentative files on all devices would be deleted.

CHAPTER 3

THE COMMAND DECODER

PS/8 provides a powerful subroutine called the Command Decoder for use by all system programs. The Command Decoder is normally called when a program starts running. When called, the Command Decoder prints a * and then accepts a command line from the console Teletype that includes a list of I/O devices, file names, and various option specifications. The Command Decoder validates the command line for accuracy, performs a LOOKUP on all input files, and sets up various tables for the calling program.

The operations performed by the Command Decoder greatly simplify the initialization routines of all PS/8 programs. Also, since command lines all have a standard basic structure, the Command Decoder makes learning to use PS/8 much simpler.

3.1 Command Decoder Conventions

Chapter 3 in the 8K Programming System User's Guide describes the syntax for the command line in detail. A brief synopsis is given here only to clarify the later discussion in this chapter.

The command line has the following general form:

* output files < input files (options)

There can be 0 to 3 output files and 0 to 9 input files specified.

<u>Output File Format</u>	<u>Meaning</u>
EXPLE.EX	Output to a file named EXPLE.EX on device DSK (the default file storage device).

Output File FormatMeaning

LPT:	Output to the LPT. This format generally specifies a non-file structured device.
DTA2:EXPLE.EX	Output to a file named EXPLE.EX on device DTA2.
DTA2:EXPLE.EX[99]	Output to a file named EXPLE.EX on device DTA2. A maximum output file size of 99 blocks is specified.
null	No output specified.

An input file specification has one of the following forms:

Input File FormatMeaning

DTA2:INPUT	Input from a file named INPUT.df on device DTA2. "df" is the assumed input file extension specified in the Command Decoder.*
DTA2:INPUT.EX	Input from a file named INPUT.EX on device DTA2. In this case .EX overrides the assumed input file extension.
INPUT.EX	Input from a file named INPUT.EX. If there is no previously specified input device, input is from device DSK, the default file storage device; otherwise, the input device is the same as the last specified input device.
PTR:	Input from device PTR; no file name is needed for non-file structured devices.
DTA2:	Input from device DTA2 treated as a non-file structured device, as, for example, in the PIP command line:

*TTY:/L< DTA2:

*Whenever a file extension is left off an input file specification, the Command Decoder first performs a LOOKUP for the given name appending a specified assumed extension. If that LOOKUP fails, a second LOOKUP is made for the file appending a null (zero) extension.

Input File Format

Meaning

In both of the last two formats, no LOOKUP operation is performed since the device is assumed to be non-file structured.

null Repeats input from the previous device specified (must not be first in input list, and must refer to a non-file structured device). For example:

* < PTR:,,

(two null files) indicates that three paper tapes are to be loaded.

The Command Decoder verifies that the specified device names, file names, and extensions consist only of the characters A through Z and 0 through 9. If not, a syntax error is generated and the command line is considered to be invalid.

There are two kinds of options that can be specified: first, alphanumeric option switches are denoted by a single alphanumeric character preceded by a slash (/) or a string of characters enclosed in parentheses; secondly, a numeric option can be specified as an octal number from 1 to 37777777 preceded by an equal sign (=). These options are passed to the user program and are interpreted differently by each program.

Finally, the Command Decoder permits the command line to be terminated by either the RETURN or ALT MODE key. This information is also passed to the user program.

3.2 Command Decoder Error Messages

If an error in the command line is detected by the Command Decoder, one of the following error messages is printed. After the error message, the Command Decoder starts a new line, prints a *, and waits for another command line. The erroneous command is ignored.

<u>Error Message</u>	<u>Meaning</u>
ILLEGAL SYNTAX	The command line is formatted incorrectly.
TOO MANY FILES	More than three output files or nine input files were specified.
device DOES NOT EXIST	The specified device name does not correspond to any permanent device name or any user assigned device name.
name NOT FOUND	The specified input file name was not found on the selected device.

3.3 Calling the Command Decoder

The Command Decoder is initiated by the DECODE function of the USR. DECODE causes the contents of locations 0 to 1777 of field 0 to be saved on the system scratch blocks, and the Command Decoder to be brought into that area of core and started. When the command line has been entered and properly interpreted, the Command Decoder exits to the USR, which restores the original contents of 0 to 1777 and returns to the calling program.

NOTE

By setting bit 10 of the Job Status Word to a 1 the user can avoid this saving and restoring of core for programs that do not occupy locations 0 to 1777.

The DECODE call can reside in the area between 0 and 1777 in field 0 and still function correctly. A typical call would appear as follows:

CDF 0	/SET DATA FIELD TO CURRENT FIELD
CIF 10	/INSTRUCTION FIELD MUST BE 1
JMS I (USR	/USR=7700 IF USR IS NOT IN CORE
	/OR USR=0200 IF USRIN WAS PERFORMED
5	/DECODE FUNCTION = 5
2001	/ARG(1), ASSUMED INPUT EXTENSION
0	/ARG(2), ZERO TO PRESERVE
	/ALL TENTATIVE FILES
:	/NORMAL RETURN
:	

ARG(1) is the assumed input extension. If an input file name is given with no specified extension, the Command Decoder first performs a LOOKUP for a file having the given name with the assumed extension. If the LOOKUP fails, the Command Decoder performs a second LOOKUP for a file having the given name and a null (zero) extension. In this example, the assumed input extension is ".PA".

DECODE performs an automatic RESET operation (see section 2.2.11) to remove from core all device handlers except those equivalent to the system device. As in the RESET function, if ARG(2) is zero all currently active tentative files are preserved. If ARG(2) is non-zero, all tentative files are deleted and DECODE returns to ARG(2) instead of ARG(2)+1.

As the Command Decoder normally handles all of its own errors, there is no error return from the DECODE operation.

3.4 Command Decoder Tables

The Command decoder sets up various tables in the top page of field 1 that describe the command line typed to the user program.

3.4.1 Output Files

There is room for three entries in the output file table that begins at location 176000. Each entry is five words long and has the following format:

	0	1	2	3	4	5	6	7	8	9	10	11
WORD 1	USER SPECIFIED FILE LENGTH							4-BIT DEVICE NUMBER				
WORD 2	FILE NAME CHARACTER 1							FILE NAME CHARACTER 2				
WORD 3	FILE NAME CHARACTER 3							FILE NAME CHARACTER 4				
WORD 4	FILE NAME CHARACTER 5							FILE NAME CHARACTER 6				
WORD 5	FILE EXTENSION CHARACTER 1							FILE EXTENSION CHARACTER 2				

OUTPUT FILE NAME
6 CHARACTERS

FILE EXTENSION
2 CHARACTERS

Bits 0 to 7 of word 1 in each entry contain the file length, if the file length was specified with the square bracket construction in the command line. Otherwise, those bits are zero.

The entry for the first output file is in locations 17600 to 17604, the second is in locations 17605 to 17611, and the third is in locations 17612 to 17616. If word 1 of any entry is zero, the corresponding output file was not specified. A zero in word 2 means that no file name was specified.

Also, if word 5 of any entry is zero no file extension was specified for the corresponding file. It is left to the user program to take the proper action in these cases.

These entries are in a format that is acceptable to the ENTER function.

3.4.2 Input Files

There is room for nine entries in the input file table that begins at location 17617. Each entry is two words long and has the following format:

	0	1	2	3	4	5	6	7	8	9	10	11
WORD 1	MINUS FILE LENGTH								4-BIT DEVICE NUMBER			
WORD 2	STARTING BLOCK OF FILE											

Bits 0 to 7 of word 1 contain the file length as a negative number. Thus, 377_8 in these bits is a length of one block, 376_8 is a length of two blocks, etc. If bits 0 to 7 are zero, the specified file has a length greater than or equal to 256 blocks or a non-file structured device was specified.

NOTE

This restriction to 255 blocks of actual specified size can cause some problems if the program has no way of detecting end-of-file conditions. For example, PIP cannot copy in image mode any file on a file structured device that is greater than 255 blocks long, although it can handle in /A or /B modes (ASCII or Binary) files of unlimited size. In /A or /B modes PIP will detect the CTRL/Z marking the end-of-file.

If this is liable to be a problem, it is suggested that the user program employ the special mode of the Command Decoder described in section 3.5 and perform its own LOOKUP on the input files to obtain the exact file length.

The two word input tables begin in locations 17617, 17621, 17623, 17625, 17627, 17631, 17633, 17635, and 17637. If location 17617 is zero no input files were indicated in the command line. If less than nine input files were specified, the unused entries in the input file list are zeroed (location 17641 is always set to provide a zero terminator even when nine files are specified).

3.4.3 Command Decoder Option Table

Five words are reserved beginning at location 17642 to store the various options specified in the command line. The format of these five words is as follows:

	0	1	2	3	4	5	6	7	8	9	10	11
17642	*	HIGH ORDER 11 BITS OF = N OPTIONS										
17643	A	B	C	D	E	F	G	H	I	J	K	L
17644	M	N	O	P	Q	R	S	T	U	V	W	X
17645	Y	Z	Ø	1	2	3	4	5	6	7	8	9
17646	LOW ORDER 12 BITS OF = N OPTIONS											

Each of these bits corresponds to one of the possible alphanumeric option switches. The corresponding bit is 1 if the switch was specified, Ø otherwise.

*Bit Ø of location 17642 is Ø if the command line was terminated by a carriage return, 1 if it was terminated by an ALT MODE.

NOTE

If no =n option is specified, the Command Decoder zeroes 17646 and bits 1 to 11 of 17642. Thus, typing =0 is meaningless since the user program cannot tell that any option was specified.

3.4.4 Example

To clarify some of the preceding, consider the interpretation of the following command line:

```
*BIN[10],<PTR:,,DTA2:PARA,MAIN /L=14200$
```

If this command line is typed to PAL8, it would cause assembly of a program consisting of four separate parts: two paper tapes, one file named PARA.PA on DTA2, and one file named MAIN.PA also on DTA2. The binary output is placed on a file named BIN.BN on device DSK, for which only 10 blocks need be allocated. No listing is generated. In addition, automatic loading of the binary output is specified by the /L option, with the starting address given as 4200 in field 1. Finally, the line is terminated by the ALT MODE key (which echoes as \$) causing a return to the Keyboard Monitor after the program is loaded.

In the case of this example, the Command Decoder returns to PAL8 with the following values in the system tables:

NOTE

The entries for PTR (where no input file name is specified) have a starting block number and file size of zero. This is always true of the input table for a non-file structured device, or a file structured device on which no file name is given.

17600	0242	DSK1 IS DEVICE NUMBER 2
	0211	FILE NAME IS BIN
	1600	
	0000	
17604	0000	NULL EXTENSION
17605		REMAINING ENTRIES IN OUTPUT TABLES ARE ZERO
17616		
17617	0016	FIRST PTR: INPUT
	0000	
17620		SECOND PTR: INPUT
17621	0016	
	0000	
17622		DTA2: PARA. PA IS 5 BLOCKS LONG, BEGINNING AT 100 _B
17623	0127	
	0100	
17624		DTA2: MAIN. PA IS 256 _B OR MORE BLOCKS LONG, BEGINNING AT BLOCK 105 _B
17625	0007	
	0105	
17626		REMAINING ENTRIES IN INPUT TABLES ARE ZERO.
17627		
17641		
17642	4001	LINE WAS TERMINATED BY ALT MODE
17643	0001	/L WAS ONLY OPTION SWITCH SPECIFIED
17644	0000	
17645	0000	
17646	4200	*14200 WAS SPECIFIED

3.5 Special Mode of the Command Decoder

Occasionally the user program does not want the Command Decoder to perform the LOOKUP on input files, leaving this option to the user program itself. For example CONVRT, which translates Disk Monitor format DECTapes to PS/8 format DECTapes, cannot permit an erroneous LOOKUP to occur on DECTapes that are not in PS/8 format. The capability to handle this case is provided in the PS/8 Command Decoder. This capability is generally referred to as the "special mode" of the Command Decoder.

3.5.1 Calling the Command Decoder Special Mode

The special mode call to the Command Decoder is identical to the standard DECODE call except that the assumed input file extension, specified by ARG(1), is equal to 5200. The value 5200 corresponds to an assumed extension of ".*", which is illegal. Therefore, the special mode of the Command Decoder in no way conflicts with the normal mode.

3.5.2 Operation of the Command Decoder in Special Mode

In special mode the Command Decoder is loaded and inputs a command line as usual. The appearance of the command line is altered by the special mode in these respects:

- a. Only one output file can be specified.
- b. No more than five input files can be specified, rather than the nine acceptable in normal mode.
- c. The character asterisk (*) is legal in file names and extensions, both in input files and on output files. It is strongly suggested that this character be tested by the user program and treated either as a special option or as an illegal file name. The user program must be careful not to ENTER an output file with an asterisk in its name as such a file cannot be manipulated or deleted by the standard system programs.

The output and option table set up by the Command Decoder is not altered in special mode. Entries in the input table are changed to the following format:

	0	1	2	3	4	5	6	7	8	9	10	11		
WORD 1									4-BIT DEVICE NUMBER				BITS 0-7 ARE ALWAYS 0	
WORD 2	FILE NAME CHARACTER 1				FILE NAME CHARACTER 2									
WORD 3	FILE NAME CHARACTER 3				FILE NAME CHARACTER 4									INPUT FILE NAME 6 CHARACTER
WORD 4	FILE NAME CHARACTER 5				FILE NAME CHARACTER 6									
WORD 5	FILE EXTENSION CHARACTER 1				FILE EXTENSION CHARACTER 1									FILE EXTENSION 2 CHARACTERS

The table entry for the first input file is in locations 17605 to 17611; the second in locations 17612 to 17616; the third in locations 17617 to 17623; the fourth in locations 17624 to 17630; and the fifth in locations 17631 to 17635. A zero in word 1 terminates the list of input files. If word 2 of an entry is zero, no input file name was specified.

CHAPTER 4

USING DEVICE HANDLERS

A device handler is a system subroutine that is used by all parts of the PS/8 system and by all standard system programs to perform I/O transfers. All device handlers are called in the same way and they all perform the same basic operation: reading or writing a specified number of 128 word records* beginning at a selected core address.

These subroutines effectively mask the unique characteristics of different I/O devices from the calling program; thus, programs that use device handlers properly are effective "device independent". Changing devices involves merely changing the device handlers used for I/O.

PS/8 device handlers have another important feature. They are able to transfer a number of records as a single operation. On a device like DECTape this permits many blocks of data to be transferred without stopping the tape motion. On a disk, a single operation could transfer an entire track or more. This capability significantly increases the speed of operation of PS/8 programs, such as PIP, that have large buffer areas.

4.1 Calling Device Handlers

Device handlers are loaded into a user selected area in field 0 by the FETCH function. FETCH returns in ARG(1) the entry point of the handler loaded. The handler is called by perform-

*The word "record" is defined to mean 128 words of data; thus, a PS/8 block consists of two 128 word records.

ing a JMS to the specified entry point address. It has the following format:

CDF N	/WHERE N IS THE VALUE OF THE CURRENT
	/PROGRAM INSTRUCTION FIELD TIMES 10 (OCTAL)
CIF 0	/DEVICE HANDLER ALWAYS IN FIELD 0
JMS I ENTRY	
ARG(1)	/FUNCTION CONTROL WORD
ARG(2)	/BUFFER ADDRESS
ARG(3)	/STARTING BLOCK NUMBER
JMP ERR	/ERROR RETURN
.	/NORMAL RETURN (I/O TRANSFER COMPLETE)
.	
ENTRY,0	/ENTRY CONTAINS THE ENTRY POINT OF THE
	/HANDLER, DETERMINED WHEN LOADED BY FETCH

As with calls to the USR, it is important that the Data Field is set to the current program field before the device handler is called. On exit from the device handler, the Data Field will remain set to the current program field.

ARG(1) is the function control word, and contains the following information:

<u>Bit(s)</u>	<u>Contents</u>
Bit 0	0 for an input operation, 1 for an output operation.
Bits 1 to 5	The number of 128 word records to be transferred must not be 0.
Bits 6 to 8	The memory field in which the transfer is to be performed.
Bits 9 to 11	Device dependent bits, can be left zero. Currently only bit 11 is used; on DEC- tape bit 11 determines the direction in which the tape is started. If bit 11 is 0 the tape starts in reverse. If bit 11 is 1 the tape starts forward.* All other handlers ignore these bits at present.

*Starting forward saves time as long as the block number, ARG(3), is seven or more blocks greater than the number of the block at which the tape is currently positioned.

ARG(2) is the starting location of the transfer buffer.

ARG(3) is the number of the block on which the transfer is to begin. The user program initially determines this value by performing a LOOKUP or ENTER operation. After each transfer the user program should itself add to the current block number the actual number of blocks transferred, equal to one-half the number of 128 word records specified, rounded up if the number of records was odd.

There are two kinds of error returns: fatal and non-fatal. When the error return occurs and the contents of the AC are negative the error is fatal. A fatal error can be caused by a parity error on input, a write lock error on output, or an attempt to write on a read-only device (or vice versa). The meaning can vary from device to device, but in all cases it is serious enough to indicate that the data transferred, if any, is invalid.

When the error return occurs and the contents of the AC are greater than or equal to zero, a non-fatal error has occurred. This error always indicates detection of the logical end-of-file. For example, when the paper tape reader handler detects the end of a paper tape it inserts a CTRL/Z code in the buffer and takes the error exit with the AC equal to zero. While all non-file structured input devices can detect the end-of-file condition, no file structured device can; and no device handler takes the non-fatal error return when doing output.

The following restrictions apply to the use of device handlers:

- a. Bits 1 to 5 of the function control word, ARG(1), must not be zero as this value is currently undefined.
- b. The user program must never specify an input into locations 7600 to 7777 or 17600 to 17777 or the page(s) in which the device handler itself resides.

- c. Note that the amount of data transferred is given as a number of 128 word records, exactly one half of a PS/8 block. Attempting to output an odd number of records can change the contents of the last 128 words of the last block written.
- d. The specified buffer address does not have to begin at the start of a page. The specified buffer cannot overlap fields, rather the address will "wrap around" in a single field. For example, writing two records from location 07600 would output 07600 to 07777 and page 0 of field 0, not field 1.

4.2 Device Dependent Operations

This section describes briefly the operation of each of the standard PS/8 device handlers, including normal operation, any special initialization operations for block 0, terminating conditions, and response to control characters typed at the keyboard. Further information on device handlers can be found in Chapter 5.

4.2.1 Teletype (TTY)

a. Normal Operation

This handler inputs characters from the Teletype keyboard and packs them into the buffer or unpacks characters from the buffer and outputs them to the teleprinter. It functions properly only on ASCII data.

On input, characters are echoed as they are typed. Following a carriage return, a line feed character is inserted into the input buffer and printed on the Teletype.

b. Initialization for Block 0

None.

c. Terminating Conditions

On input, detection of a CTRL/Z causes a CTRL/Z to be placed in the input buffer, the remaining words of the buffer filled with zeros, and a non-fatal error to be returned. On output, detection of a CTRL/Z character in the output buffer causes output to be terminated and the normal return to be taken. There are no fatal errors associated with the Teletype handler.

d. Teletype Interaction

CTRL/C forces a return to the Keyboard Monitor. CTRL/Z forces an end-of-file on input (see c). CTRL/O terminates printing of the contents of the current buffer on output.

4.2.2 High-Speed Paper Tape Reader (PTR)

a. Normal Operation

This handler inputs characters from the high-speed paper tape reader and packs them into the buffer.

b. Initialization for Block Ø

The handler prints an up-arrow (↑)* on the teleprinter and waits for the user to load the paper tape reader. By typing any single character (except CTRL/C) the user initiates reading of the paper tape.

c. Terminating Conditions

Detection of an end-of-tape condition, indicated by the failure to get a character in a specified period of time, causes a CTRL/Z to be entered in the buffer, the remaining words of the buffer to be filled with zeros, and a non-fatal error to be returned. Attempting to output to the paper tape reader causes a fatal error to be returned.

d. Teletype Interaction

Typing CTRL/C forces a return to the Keyboard Monitor.

4.2.3 High-Speed Paper Tape Punch (PTP)

a. Normal Operation

This handler unpacks characters from the output buffer and punches them on the paper tape punch.

b. Initialization for Block Ø

None.

*On some Teletypes, up-arrow is replaced by the circumflex (^) character.

c. Terminating Conditions

Attempting to input from the paper tape punch causes a fatal error to be returned. There are no non-fatal errors associated with this handler.

d. Teletype Interaction

Typing CTRL/C forces a return to the Keyboard Monitor.

4.2.4 Line Printer (LPT)

a. Normal Operation

This handler unpacks characters from the buffer and prints them on the line printer. The characters horizontal tab (ASCII 211) causes sufficient spaces to be inserted to position the next character at a "tab stop" (every eighth column, by definition). The character vertical tab (ASCII 213) causes nine line feeds to be output. The character Form Feed (ASCII 214) causes a skip to the top of the next page. Finally, the handler maintains a record of the current print column and starts a new line after 80 columns have been printed. This handler functions properly only on ASCII data.

b. Initialization for Block Ø

Before printing, the line printer handler issues a form feed to space to the top of the next page.

c. Terminating Conditions

On detection of a CTRL/Z character in the buffer, the line printer handler issues a form feed and immediately takes the normal return. Attempting to input from the line printer forces a fatal error to be returned. Also if the line printer error flag is set a fatal error is returned. There are no non-fatal errors associated with the line printer handler.

d. Teletype Interaction

Typing CTRL/C forces a return to the Keyboard Monitor.

4.2.5 Card Reader (CDR)

a. Normal Operation

This handler reads characters from the card reader and packs them into the input buffer. Trailing spaces (blank columns) on a card are deleted from input. The handler can accept only alphanumeric format data on cards (the DECØ29 standard card codes are used).

b. Initialization for Block Ø

None.

c. Terminating Conditions

A card which contains an underline character in column 1 (an 0-8-5 punch) with the remaining columns blank is an end-of-file card. In addition, after reading each card the handler checks to see if a CTRL/Z was typed at the keyboard. After either an end-of-file card or a CTRL/Z being typed, a CTRL/Z is inserted in the buffer, the remaining words of the input buffer are filled with zeros, and a non-fatal error is returned. Attempting to output to the card reader causes a fatal error to be returned.

d. Teletype Interaction

Typing CTRL/C forces a return to the Keyboard Monitor. Typing CTRL/Z forces an end-of-file to occur (see c).

4.2.6 File Structured Devices

a. Normal Operation (DECtape, LINCTape, DF32, RFØ8, and RK8)

These handlers transfer data directly between the device and the buffer.

b. Initialization for Block Ø

None.

c. Terminating Conditions

A fatal error is returned whenever the transfer caused one of the error flags in the device status register to be set. For example, a fatal error would result if a parity error occurred on input, or a write lock error occurred on output. The device handlers generally try three times to perform the operation before giving up and returning a fatal error. There are no non-fatal errors associated with file structured devices.

d. Teletype Interaction

Typing CTRL/C forces a return to the Keyboard Monitor.

NOTE

The system device handler does NOT respond to a typed CTRL/C.

CHAPTER 5

RECONFIGURING THE PS/8 SYSTEM

In the instructions on building PS/8 in the 8K Programming System User's Guide,* mention is made of the several paper tapes marked CONFIG distributed with each system. Each PS/8 system contains the source file CONFIG.PA, either on DECTape, paper tape (in two parts), or LINCTape. CONFIG contains all configuration dependent parts of the PS/8 system, including all device handlers, device dependent system tables, bootstrap routines, and system device handlers.

The problem of changing the system configuration is reduced to editing the CONFIG file and assembling the result to produce a new binary tape. This new binary tape is then used in building a modified system. The following chapter describes in detail the ways in which CONFIG can be modified.

5.1 Conditional Assembly of CONFIG

The source file CONFIG.PA contains a number of conditional symbols. These are symbols that, in conjunction with some IFDEF, IFNDEF, IFZERO, or IFNZRO pseudo-ops, cause various sections of the program to be assembled with PAL8. Creating standard system variations requires the proper definition of one or more of these parameters.

*Although it is not explicitly stated, the instructions given could also be used to create a PS/8 System DECTape. Mount the tape to be built on Unit 0, WRITE ENABLED, and follow the instructions given. The binary tape marked "DECTape CONFIG" would be used in place of one of the disk CONFIG's.

5.1.1 System Device Selection

One and only one of the system device parameters can be defined and assigned a non-zero value.

a. RFØ8 Disk System

Defining RFØ8 = n, where n is 1,2,3, or 4 causes an n disk RFØ8 to be the system device. For example:

RFØ8=2

would generate a 512K RF/RSØ8 system.

b. DF32 Disk System

Defining DF32 = n, where n is 1,2,3, or 4 causes an n disk DF32 to be the system device. For example:

DF32=2

would generate a 64K DF/DS32 system.

NOTE

Because PS/8 alone takes 14K of the system device, it is felt that a single disk DF32 has insufficient storage for PS/8. The system is not supported by DEC on a 32K disk.

c. RK8 Disk System

Defining RK8=1 causes an RK8 disk to be the system device, generating an RK8 system.

d. PDP-12 LINCTape System

Defining LINC SYS=n, where n is 1 or 2 causes LINCTape unit Ø to be the system device. If LINC SYS=1, the default file storage device (device DSK) is also LINCTape unit Ø. If LINC SYS=2, the default file storage device is LINCTape unit 1, which is to be preferred as it minimizes tape motion.

NOTE

When LINC SYS is the specified system parameter, the user must be certain to explicitly specify LINCTAPE=1 (see section 5.1.2), otherwise DECTape rather than LINCtape handlers would be included in the system.

e. DECTape System

If no other system device parameters are specified, DECTape unit 0 is the default system device. One can also define DECTAPE=n, where n is 1 or 2 to cause DECTape unit 0 to be the system device. If DECTAPE=1, the default file storage device (device DSK) is DECTape unit 0. If DECTAPE=2, the default file storage device is DECTape unit 1, which is to be preferred as it minimizes tape motion.

5.1.2 Optional Device Parameters

The standard system, generated by defining one of the system device parameters, contains the following device handlers:

<u>Device Code</u>	<u>Meaning</u>
SYS	Selected system device
TTY	Teletype
PTR	High-speed paper tape reader
PTP	High-speed paper tape punch
CDR	Card reader (CR8 or CM8)
LPT	LP08 Line printer
DTA0-DTA7	Handlers for eight DECTape drives
DSK	Default file storage device, always the same as the system device unless LINC SYS=2 or DECTAPE=2 are used.

These device handlers can be replaced by others by defining one of the following parameters:

a. LINCtape handlers

If the computer is a PDP-12, it is necessary to replace the standard DECTape handlers with LINCtape handlers. This must be done explicitly by defining LINCTAPE=1.

b. LP12 Line Printer, Type 645

A small number of systems have the old style line printer rather than the new LP08. Defining LP08=0 causes a device handler for the LP12 (Type 645) line printer to replace the standard LP08 handler.

NOTE

The FORTRAN run time I/O routines, with the exception of the device-independent I/O routines, do not use PS/8 handlers. Therefore, in FORTRAN use the statement WRITE (3,n) only works on the LP08 line printer. To use a different printer the FORTRAN subroutine UTILITY.SB (available on source DECTape #3) must be modified.

c. Low-Speed Paper Tape

The standard Teletype handler (TTY) checks for ↑C and ↑Z control characters (CTRL/C and CTRL/Z). To enable the handler to work on terminals that utilize parity ASCII, the handler ignores the leading bit when checking for these characters. For this reason non-ASCII input is not acceptable to the Teletype handler.

While this is no problem to systems having a high-speed paper tape reader, those who lack one would be unable to load any binary paper tapes under PS/8. For this reason there is a special low-speed paper tape handler available in CONFIG. To assemble this option, define the following:

NOHSPT=1

When NOHSPT=1 is used the PTR and PTP handlers are replaced with new Teletype routines. (The names of the new handlers remain PTR for input and PTP for output.)

WARNING!

The source of CONFIG released in November 1970 contains an error in this low-speed paper tape routine. To correct this, insert the following code immediately before the terminating \$ in CONFIG.PA:

```
IFNZRO NOHSPT <*6522
RTL
RTL
DCA PTR
TAD PTR >
```

This omission will be corrected in later releases of CONFIG.

Like the high-speed paper tape reader handler, the low-speed handler prints a ↑ character before reading a tape. The user then loads the tape to be read in the low-speed reader and sets the Teletype reader switch to START. While reading from the low-speed reader, do NOT type anything on the keyboard. At the end of the reading process, turn the Teletype reader control switch to STOP and remove the tape.

The PTR and PTP handlers differ from the standard TTY handler in that they ignore control characters. The PTR handler recognizes an end-of-tape condition by "timing out" the low-speed reader - it expects the keyboard/reader flag to be reset within 150 ms and if it is not, an end-of-file occurs. At the end-of-file a CTRL/Z is automatically inserted in the buffer following the last character read.

5.1.3 Other Options

There is one more option that can be used in building a non-standard PS/8 system. The parameter DIRECT determines whether or not the system directory is to be zeroed when the system is rebuilt. Defining DIRECT=1 causes a new system to be built without zeroing the old directory. This feature is useful when reconfiguring a system to avoid having to reload all of the files currently on the system device.

5.1.4 Example

As an example of reconfiguring a PS/8 system, suppose a machine has the following configuration:

- a. PDP-8/I computer
- b. DECTape
- c. RF08 disk with two RS08's (768K of storage)
- d. Type 645 line printer (LP12)
- e. High-speed reader/punch
- f. Card reader

The following steps would be taken to build a system tailored to the above configuration.

- a. Build the system in the usual manner using the RF08 CONFIG provided and the instructions in the 8K Programming System User's Guide.
- b. Use PIP to put the file CONFIG.PA on the disk.
- c. Execute the following commands:

```
.R EDIT
*PARA.PA <

#A
/PARAMETER FILE
RF08=3           /3 PLATTER RF08 SYSTEM
LP08=0           /OLD STYLE LINE PRINTER
DIRECT=1         /PRESERVE SYSTEM DIRECTORY

#E

.R PAL8
*PTP:<PARA,CONFIG
```

- d. Use the paper tape punched by PAL8 as the CONFIG binary in building a new system.
- e. Finally, since the DIRECT parameter prevented the new size of the system device (changed from 256K to 768K for this example) from being automatically written in the directory, it must be updated by the following operation:

```
.R PIP
*SYS: < SYS:/S=1
ARE YOU SURE?
YES
```

5.2 Building a System on DECTape or LINCtape

To avoid the time involved in rebuilding the system off paper tape each time it is changed, the binary tapes of PS/8, the Command Decoder (CD), and CONFIG can be placed on DECTape (or LINCtape) and the system built by the following procedure:

- a. Put the system tape on unit 0 and the tape containing the binary files PS8.BN, CD.BN, and CONFIG.BN on unit 1 (where CONFIG.BN corresponds to the system configuration).
- b. Bootstrap the DECTape (or LINCtape) system and execute the following command:

```
.R ABSLDR
*DTA1:PS8,CONFIG/G
```

- c. The system should halt with 7777 in the AC (if not, an error has occurred, try again from step b); press CONTINUE to proceed. If the system device is not being changed, the build is complete at this point, otherwise execute the following command:

```
.R ABSLDR
*DTA1:PS8,CONFIG,CD/G
```

- d. The new system is built and responds to the RETURN key by printing a dot when ready to accept input. Now, if necessary, transfer files to the new system device with PIP.

Of course the above procedure is not limited to DECTape or LINCtape. The files PS8.BN, CD.BN, and CONFIG.BN could just as easily be placed on a disk and loaded from there. The example is intended to illustrate a useful alternative technique for building a system.

5.3 Adding New Device Handlers

PS/8 system programs have been organized so that all non-Teletype I/O is done via calls to standard system device handlers. Any new device for which a handler is written can be added to the system by changing CONFIG and rebuilding the

system. Once this is done, all existing system programs are able to use the new device. This flexibility is one of the most important features of the PS/8 system.

NOTE

In the November 1970 version of PS/8 the following cannot use devices that require two page device handlers:

- a. GET, RUN, and SAVE operations,
- b. CONVERT output device must have a one-page handler,
- c. PIP cannot perform /Z or /D operations on devices requiring two page handlers,
- d. SABR cannot use two page handlers, and
- e. General I/O in FORTRAN, READ(4,n) and WRITE(4,n), cannot use two page device handlers.

Future versions of PS/8 will not contain these restrictions.

The following sections describe in detail how to add a new device handler. For further information on device handlers, the reader should consult Chapter 4. Examples of standard handlers can be found in the listing of CONFIG which can be ordered from the DEC Program Library (DEC-P8-MW3A-LA).

5.3.1 Writing Device Handlers

A device handler is a page-independent one or two page long subroutine. The device handler must run properly in any single page or two contiguous pages in field 0 (except 0000 to 0177 or 7600 to 7777). All device handlers have the same calling sequence:

CDF N	/N IS CURRENT FIELD TIMES 10 (OCTAL)
CIF 0	/DEVICE HANDLER LOCATED IN FIELD 0
JMS I ENTRY	/ENTRY IS DETERMINED BY USR "FETCH"
FUNCTION	/FUNCTION IS BROKEN DOWN AS FOLLOWS:
	/ BIT 0 = 0 FOR READ
	/ BIT 0 = 1 FOR WRITE
	/ BITS 1 TO 5 = NUMBER OF PGS TO TRANSFER
	/ BITS 6 TO 8 = FIELD FOR TRANSFER
	/ BITS 9 TO 11 = DEVICE DEPENDENT BITS
BUFFER	/CORE ADDRESS OF TRANSFER BUFFER
BLOCK	/STARTING BLOCK NUMBER FOR TRANSFER
ERROR	/ERROR RETURN, AC>=0 MEANS END-OF-FILE
	/ AC<0 MEANS FATAL ERROR
NORMAL	/NORMAL RETURN

The device handler reads or writes a number of 128 word records beginning at the selected block. In general, device handlers should conform to the following standards:

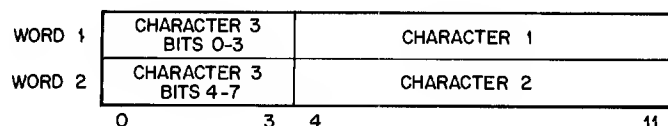
- a. On normal return from a device handler the AC is zero and the DATA FIELD is always restored to its original entry value.
- b. Although the starting block number has true significance only for file structured devices, handlers for non-file structured devices can check the block number and perform initialization if the block number is zero. For example, the line printer handler outputs a form feed before printing when the specified block number is zero.
- c. Handlers should be written to be as foolproof as possible. Examples of typical user errors are: calling handler with non-zero AC (always perform a CLA in the handler); trying to read on a write-only device, or trying to write on a read-only device (give a fatal error return); specifying 0 pages to be transferred (accept as meaning no actual transfer is to take place); or attempting to access a nonexistent block (give a fatal error return).
- d. Device handlers normally check to see if a CTRL/C (ASCII 203) has been typed by the user. If one has, the handler aborts I/O and JMP's to location 7600 in field 0.

- e. Device handlers should be able to detect standard error conditions like checksum or parity errors. Whenever possible, several attempts to perform the transfer should be made before aborting I/O and taking the error exit. In addition, when operator intervention is required, the handler would normally wait for the action rather than take a fatal error exit. For example, if the paper tape punch is not turned on, the PTP handler waits for the punch to be turned on.
- f. By convention, in any handler for a device (like DECTape) that can search either forward or backward for a block, Bit 11 of the function word (one of the device-dependent bits) controls the starting direction of the search. Bit 11 is a 1 if the starting direction is forward and a 0 if it is reverse. The other two device dependent bits are not assigned any significance at the present time.
- g. Remember that the user specifies a multiple of 128 words to transfer, whereas the transfer starts at the beginning of a 256 word block. This means that the handler must provide the capability of reading or writing the first half of a block. When writing the first half of the block, the contents of the second half of the block can be altered.
- h. The entry point to a two page device handler must be in the first page.
- i. A number of handlers can be included in the one or two pages of code. Where more than one handler is included in a single handler subroutine, the handlers are called co-resident. Co-resident handlers are always brought into core together. For example, all eight DECTape handlers fit into one page; hence, the DECTape handlers are co-resident. One restriction on co-resident handlers is that if they are two pages long all entry points must be in the first page.
- j. The USR, while doing file operations, maintains in core the last directory block read in order to reduce the number of directory reads necessary. The proper functioning of this feature depends on the fact that every handler for a file-structured device on a single system has a unique entry point relative to the beginning of the handler. The relative entry points currently assigned for file structured handlers are:

Device HandlersRelative Entry Points

MAGtapes	0 to 6
System Device Handler	7
DECTape or LINCTape	10 to 17
RKA0	20
RKA1	21
RKA2	22
RKA3	23

- k. If the device is block oriented (such as DECTape, LINCTape, or disk), then the handler transfers data directly with no alteration. However, if the device is character oriented (such as a paper tape reader, Teletype, or line printer), the handler is required to pack characters into the buffer on input and unpack them on output. The standard PS/8 character packing format puts three 8-bit characters into two words as follows:



When packing characters on input, the character CTRL/Z (octal 232) is inserted at the logical end-of-file (for example, at the end of the tape in the paper tape reader handler). Following CTRL/Z, the remaining words of the input buffer should be zeroed.

1. The device handler, whether one or two pages long, must be completely page independent: it can be loaded and executed in any page in field 0 except page 0 and 7600 to 7777. Page independent code can have no address constants. For example, the following routine illustrates how a table lookup would be performed in page independent and non-page independent code:

Non-Page Independent Code

```

TAD INDEX
TAD (TABLE)
DCA TEMP
TAD I TEMP
:
:
INDEX, 0
TEMP, 0
TABLE, ...

```

Page Independent Code

```

TAD INDEX
TAD (TAD TABLE)
DCA .+1
0
:
:
INDEX, 0
TABLE, ...

```

The page independent method works only because the table must be in the same page. Writing page independent code for one page handlers is quite easy. Two page handlers are considerably more difficult, since communication between the two pages requires the handler to determine where in core it was loaded. Specifically, two page handlers often include one-time initialization code that performs a JMS to determine where it is. The card reader handler in CONFIG should be studied by anyone who writes a two page handler.

5.3.2 Editing Device Handlers Into CONFIG

Once a new handler has been written and thoroughly debugged as a stand-alone subroutine, it can be added to the system by editing the handler into CONFIG.PA, changing certain sections in CONFIG, reassembling the result, and building a new system from the binary tape produced. Follow the following steps to build a new PS/8 system:

- a. Edit the handler into CONFIG. The handler should be originated into the areas of core between 4400 to 5577 or 14000 to 17577.
- b. Select a system block on which to write the handler. The first device handler storage block is "DVHORG"*, and the last available block for device handlers is "DVHORG"+7 (blocks 16 [octal] to 25 [octal] in the current system). Existing device handlers require blocks "DVHORG" through "DVHORG"+4.
- c. In the generation section of CONFIG (the subroutine "WRDEVH") edit in a call to the system handler to write the desired device handler onto the selected device handler storage block. The calls already included in this subroutine should provide sufficient examples.
- d. Select a device number for the new device. The number must be in the range 3 to 15 because device number 1 is reserved for SYS and device number 2 is reserved for DSK. The number selected will remove some standard device from the system; the device removed should be a "useless" one (for example, remove DTA7 if the system has less than 8 DECTape

*Names in quotes refer to symbols used in CONFIG.PA.

drives). If the handler contains co-resident device handlers, several device numbers must be selected (not necessarily consecutive numbers).

- e. Edit into CONFIG the new entries for the Permanent Device Name Table ("SDNAME") and the Device Handler Information Table ("SDVHND"). In both cases the device number is an index to the table entry to be changed. See Appendix B for information about entries in these tables. Remember that entries must be made for all co-resident handlers.

NOTE

When considering a name for the device, be certain to select one that does not conflict with an existing device name. See section B.3.1.

- f. Check the device Type Table explained in section B.3.5. If the new device is similar to an existing device then assign the new device the same device type code as the existing device. (If the device is file structured, it can only be similar to devices of the same size.) Otherwise select a new device type in the range 21 to 77 (octal). Now edit into CONFIG a new entry in the Device Control Word Table ("DCB"), see section B.3.5 to get the format for an entry. Again, entries must be made for all co-resident handlers.
- g. If there is any device other than the system device that should be the new default file storage device (DSK), change the entries for device number 2 in the Device Handler Information Table ("SDVHND") and the Device Control Word Table ("DCB") to be identical to the entries for the selected device.

NOTE

The new device DSK will no longer be equivalent to the system device, hence its device handler will no longer be permanently resident. The entry in the Device Handler Residency Table (see section B.3.3) must be zeroed. This is done by editing the following code into the end of CONFIG:

*7450;0 /DSK IS NON-RESIDENT

*Names in quotes refer to symbols used in CONFIG.PA.

- h. All the edits to CONFIG are now complete. Assemble the new source of CONFIG and rebuild the system by following the instructions given in section 5.1.
- i. Finally, if the new device is a file structured device for which it was necessary to select a new device type code in step f, the Device Length Table in the file PIP.SV must be patched so that PIP can perform /Z and /S operations on the device correctly. This can be done easily by using the system ODT as follows:

<u>Teleprinter Output</u>	<u>Notes</u>
.GET SYS:PIP	
.ODT	
136nn/0000 xxxx	where nn is the new device type code in octal and xxxx is minus the highest PS/8 block number for the device, also in octal.
↑C	
.SAVE SYS:PIP	

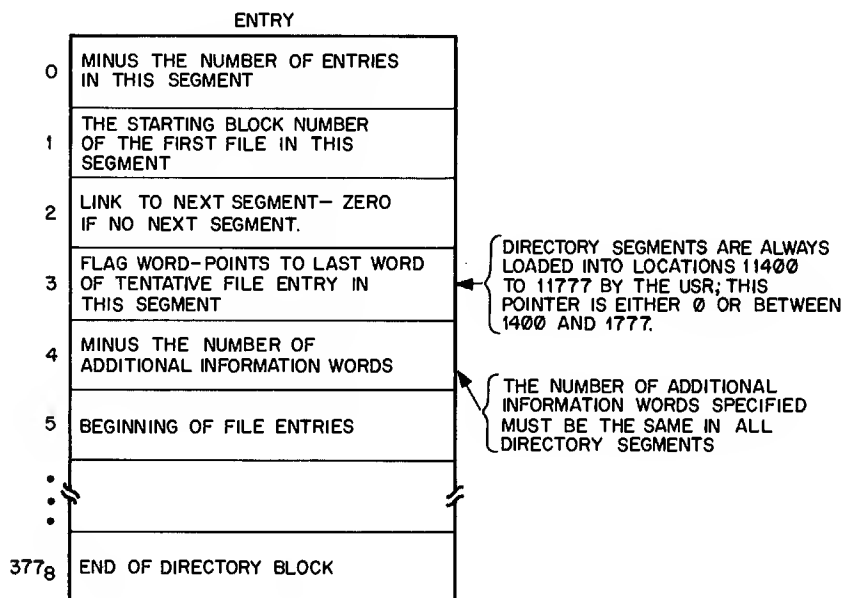
APPENDIX A

PS/8 File Structures

A.1 File Directories

Blocks 1 through 6 on all file structured devices are reserved for the file directory of that device. Six blocks are always allocated but all are not necessarily active at any given time. To minimize the number of directory reads and writes necessary, PS/8 fills one directory block completely before overflowing onto a second block. Thus the user with only a few files can perform directory LOOKUPS and ENTERs faster than one with many files.

The directory blocks are each structured according to the following format:



Locations 0 through 4 of each directory block are called the segment header.

A.1.1 Directory Entries

There are three types of file directory entries: A permanent file entry appears as follows:

<u>Location</u>	<u>Contents</u>	<u>Notes</u>	
0	FILE NAME CHARACTER 1	THE FILE NAME AND EXTENSION IS PACKED IN SIXBIT ASCII (i.e., "A" WOULD BE 01).	
	FILE NAME CHARACTER 2		
1	FILE NAME CHARACTER 3		
	FILE NAME CHARACTER 4		
2	FILE NAME CHARACTER 5	N, THE NUMBER OF ADDITIONAL INFORMATION WORDS, IS GIVEN BY WORD 4 OF THE DIRECTORY HEADER. IF $N \neq \emptyset$, THEN WORD 4 OF THE ENTRY IS THE CREATION DATE OF THE FILE.	
	FILE NAME CHARACTER 6		
3	FILE EXTENSION CHARACTER 1		
	FILE EXTENSION CHARACTER 2		
4	ADDITIONAL INFORMATION WORDS		
N+3			
N+4	MINUS FILE LENGTH IN BLOCKS		

Note - if word 3 is zero, the given file has a null extension.

An empty file entry appears as follows:

<u>Location</u>	<u>Contents</u>
0	ENTRY IS ALWAYS 0000
1	MINUS THE NUMBER OF BLOCKS IN THIS EMPTY FILE

A tentative file entry appears as a permanent file entry with a length of zero. It is always immediately followed by an empty file entry. When the tentative file is entered in a directory, location 3 in the segment header becomes a pointer to this entry. The CLOSE function inserts the length word of the tentative file entry, making it a permanent file, and adjusts the length of the following empty file entry (deleting that entry if the length becomes zero).

Whether or not there is a tentative file open on any device is determined by examination of bits 9 to 11 of the system Device Control Word Table (see section B.3.5) not the contents of location 3 in the segment header. Zeroing these bits in the Device Control Word Table makes the active tentative file on the device inactive. The next time that the system has to write the directory segment, the inactive tentative file entry is removed. The distinction between active and inactive tentative files is made so that PS/8 can avoid spending the time required to perform an extra read and write of the device directory.

A.1.2 Number and Size of PS/8 Files

All files on a PS/8 device must occupy a contiguous group of blocks on the device. The length of any file is indicated in its directory entry, and the starting block of the file is deduced by adding together word 1 of the segment header and the lengths of all files whose entries precede it in the directory segment.

Each directory segment must have enough unused words at the end to accommodate a permanent file entry (N+5 words, where N is the number of Additional Information Words). Thus, if N is the number of Additional Information Words the maximum number of permanent file entries in any one segment is:

$$\text{MAX} = \left[\frac{256-5 - (N+5)}{N+5} \right] = \left[\frac{246-N}{N+5} \right]$$

Directory fragmentation (alternation of permanent file entries with empty file entries) reduces this maximum, and in the worst case the number of permanent file entries in any one segment is limited to:

$$\text{MIN} = \left\lfloor \frac{256-7 - (N+5)}{N+7} \right\rfloor = \left\lfloor \frac{244-N}{N+7} \right\rfloor$$

with $N=1$, $\text{MAX}=40$, and $\text{MIN}=30$. Since there are six segments in the directory, the maximum number of files possible (with $N=1$) would be 240.

Finally, PS/8 devices are limited to 4095 blocks, each 256 words long. Thus, the maximum size of any single PS/8 file structured device is 1,048,320 words. Blocks 0 through 6 of the device are unavailable for file storage; therefore, the largest possible file is 4088 blocks long, or 1,046,528 words.

A.1.3 Sample Directory

The initial directory written when the PS/8 system is built looks as follows:

<u>Location</u>	<u>Contents</u>	<u>Notes</u>
SEGMENT HEADER	0 7776	TWO ENTRIES
	1 0070	FILE STORAGE STARTS AT BLOCK 70_8 *
	2 0000	NO ADDITIONAL DIRECTORY SEGMENTS
	3 0000	NO TENTATIVE FILES
	4 7777	ONE ADDITIONAL INFORMATION WORD
PERMANENT FILE ENTRY	5 0102	} FILE NAME IS "ABSLDR"
	6 2314	
	7 0422	
	10 2326	FILE EXTENSION IS .SV
	11 5370	DATE IS 10/31/70
EMPTY FILE ENTRY	12 7773	LENGTH IS FIVE BLOCKS
	13 0000	EMPTY FILE
	14 6534	LENGTH IS 1244_8 (676_{10}) BLOCKS -
		THIS IS DEPENDENT ON THE SYSTEM DEVICE USED, 676 IS THE VALUE FOR A DEC TAPE SYSTEM.
3778		

* THIS LEAVES ROOM FOR THE
PS/8 SYSTEM AREAS

A.2 File Formats

There are three different standard file formats used by PS/8 and associated system programs:

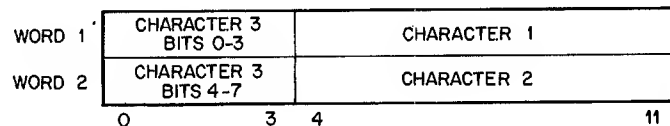
ASCII and Binary files*

Core Image files (.SV format)

Relocatable FORTRAN library files (LIB8.RL is the only current example of this format)

A.2.1 ASCII and Binary Files

ASCII and Binary files are packed three characters into two words, as follows:



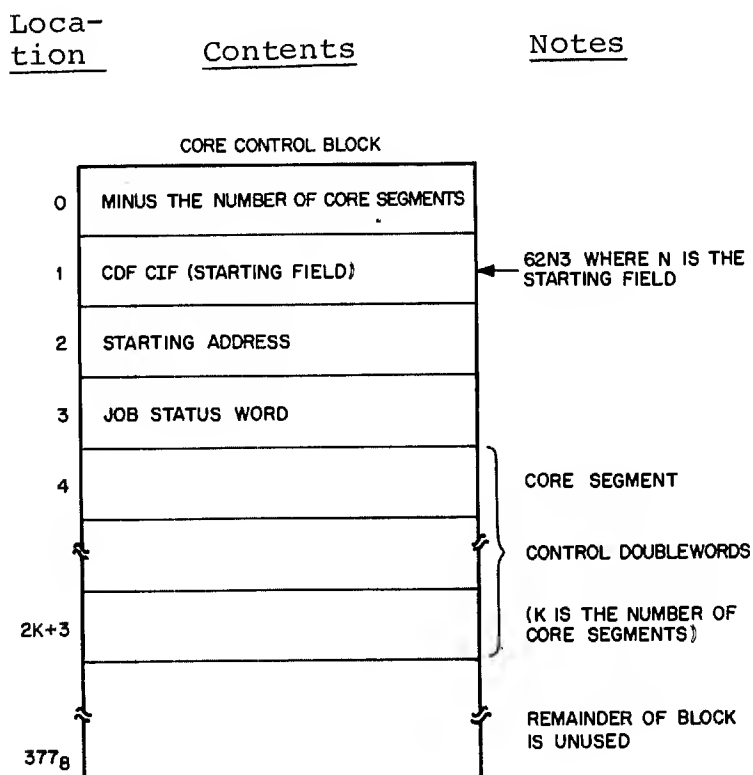
The following conventions are used by PS/8 system programs:

- a. In ASCII files the character NULL (ASCII 000) is always ignored.
- b. In Binary files the binary data must be preceded by one or more frames of leader/trailer code (ASCII 200 code). The first character of binary data must be either 100 to 177 octal (an origin setting for absolute binary files) or 240 to 257 octal (a COMMON declaration frame for relocatable binary files). The end of binary data is indicated by one or more frames of leader/trailer code.
- c. ASCII and Binary files are terminated by a CTRL/Z code (ASCII 232). In binary files, a CTRL/Z code occurring before the trailer code is treated as data rather than end-of-file.

*Binary files can contain either absolute binary data (i.e., output from PAL8) or relocatable binary data (i.e., output from SABR).

A.2.2 Core Image (.SV format) Files

A core image file consists of a header block followed by the actual core image. The header block is called the Core Control Block. The Core Control Block consists of the first 128 words of the 256 word block reserved for that purpose. The second 128 words are unused. The Core Control Block is formatted as follows:



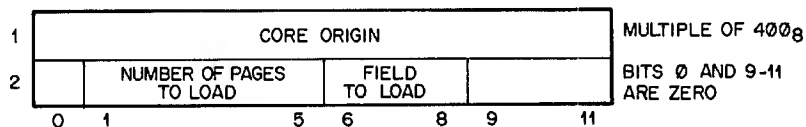
The format of the Job Status Word is as follows:

<u>Bit Condition</u>	<u>Meaning</u>
Bit 0 = 1	File does not load into locations 0 to 1777 in field 0.
Bit 1 = 1	File does not load into locations 0 to 1777 in field 1.
Bit 2 = 1	Program must be reloaded before it can be restarted.

Bit ConditionMeaning

Bit 10 = 1	Locations 0 to 1777 in field 0 need not be preserved when the Command Decoder is called.
Bit 11 = 1	Locations 0 to 1777 in field 1 need not be preserved with the USR is called.

The Core Segment Doublewords control the reading and writing of the associated areas of core. The format of each entry is as follows:

LocationContentsNotes

The core origin must be a multiple of 400₈. The Core Segment Control Doublewords are sorted within the header block in order of decreasing field and increasing origin within the same field. There can be no more than 32₁₀ Core Segment Control Doublewords in any Core Control Block.

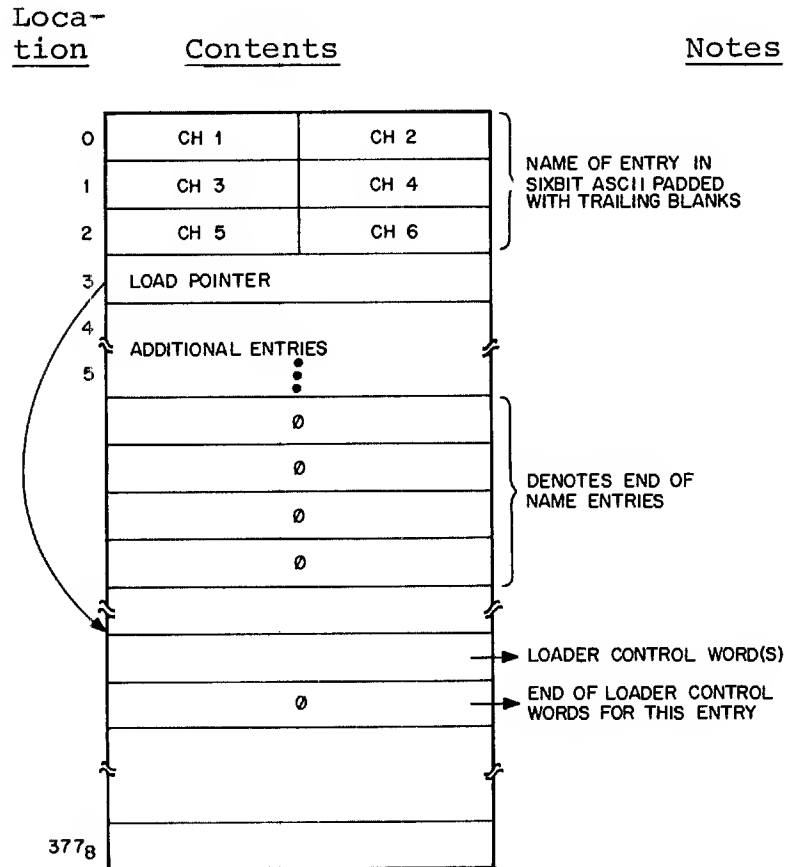
The Core Control Block for the program at the time it is loaded into core is always saved in words 200₈ through 377₈ of block 37₈ (one of the system scratch blocks) on the system device. It is placed there by the GET and RUN operations or by the ABSLDR or LOADER programs. This Core Control Block is used when performing a SAVE without arguments.

NOTE

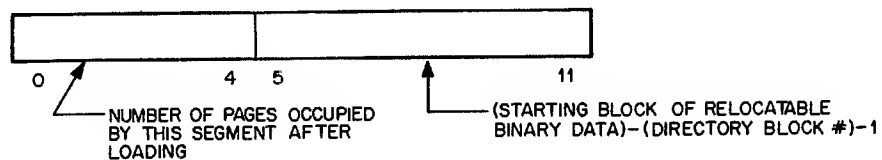
The R command differs from the RUN command in that the program's Core Control Block is not written on- to the scratch area when using the R command. In order to SAVE a program that has been loaded by the R command all of the argument of the SAVE command must be explicitly stated.

A.2.3 Relocatable FORTRAN Library File

A relocatable FORTRAN library consists of a library directory block followed by relocatable binary segments. The directory block has the following format:



The Load Pointer is a number between 0 and 377₈ which points (relative to the beginning of the block) to an array of Loader Control Words. The Loader Control Words have the following information:



There can be one or more Loader Control Words for each entry. The Loader Control Words for an entry are terminated by a word of zero. The following is a simple directory block.

<u>Location</u>	<u>Contents</u>	<u>Notes</u>
0	1117	NAME OF ENTRY IS "IOH_--"
1	1040	
2	4040	
3	0376	LOAD POINTER FOR "IOH"
4	0530	NAME OF ENTRY IS "EXIT_--"
5	1124	
6	4040	
7	0373	LOAD POINTER FOR "EXIT"
10	0000	MARKS END OF ENTRIES
11	0000	
12	0000	
13	0000	
LOADER CONTROL WORDS FOR "EXIT"	373	0207 (RELATIVE BLOCK 10 ₈) (ONE PAGE LONG)
	374	0411 (RELATIVE BLOCK 12 ₈) (TWO PAGES LONG)
	375	0000
LOADER CONTROL WORDS FOR "IOH"	376	2400 (RELATIVE BLOCK 1)(12 ₈ PAGES LONG)
	377 ₈	0000

APPENDIX B

DETAILED LAYOUT OF THE SYSTEM

This appendix covers three topics: the reserved areas on the system device, the resident portion of PS/8, and the various system tables.

B.1 Layout of the System Device

The first 70₈ blocks (14K words) on the system device are reserved by the PS/8 system. These blocks are used as follows:

<u>Block(s) in Octal</u>	<u>Contents</u>
0	System Bootstrap Routine
1-6	Device Directory
7-12	Keyboard Monitor
13-15	User Service Routine
16-25	Device Handlers
26	ENTER Processor for USR
27-50	System Scratch Blocks
51-53	Command Decoder
54-55	SAVE and DATE Overlays
56	Monitor Error Routine
57	CHAIN Processor for USR
60-63	System ODT
64-67	Reserved for System Expansion

File storage begins with block 70₈.

The system scratch blocks are used for preserving the contents of core when the Keyboard Monitor, USR, Command Decoder, or ODT are loaded. In addition, various system programs use the scratch area. Most importantly, the SAVE command expects the Core Control Block to be loaded in words 200₈ to 377₈ of block 37₈. The Core Control Block is stored at those locations by the GET or RUN command or by the ABSLDR or LOADER program.

A detailed breakdown of system scratch block usage follows:

Block(s) in OctalContents

27-32	The contents of locations 10000 to 11777 are saved in this area when the USR is loaded.
33-36	The contents of locations 0 to 1777 are saved in this area when the Command Decoder, Keyboard Monitor, or ODT is loaded.
37	Words 200 ₈ to 377 ₈ of this block contain the Core Control Block for the last program loaded by the GET or RUN command, or the ABSLDR or LOADER program.
40-47	Used as scratch storage by the ABSLDR and LOADER programs.
50	Reserved for future expansion.

B.2 Layout of the PS/8 Resident Program

The top core pages in fields 0 and 1 are used by the resident portion of PS/8 and are not accessible by the user. Future expansion of PS/8 may later require space to be allocated in the top page of field 2. As a general rule, system and user programs should never destroy the contents of locations 7600 to 7777 of any field.

The resident portion of PS/8 is structured as follows:

<u>Location</u>	<u>Contents</u>	<u>Notes</u>
7600	WRITE OPERATION	NON-DESTRUCTIVE ENTRY TO PS/8
7605	JMP TO FIELD 1 FOR READ	DESTRUCTIVE ENTRY TO PS/8
7607	SYSTEM DEVICE HANDLER	ENTRY TO SYSTEM DEVICE HANDLER
7743		
7744	CURRENT STARTING ADDRESS	
7745		
7746	JOB STATUS WORD	
7747	0	MUST ALWAYS BE ZERO
7750	RESERVED FOR DATA BREAK LOCATIONS	
7755		
7756	PROGRAM SETUP AREA	THE KEY BOARD MONITOR AND ODT MODIFY THIS AREA
7777		

TOP PAGE OF FIELD 1

<u>Location</u>	<u>Contents</u>	<u>Notes</u>
7600	OUTPUT FILE LIST (3 ENTRIES)	
7616		
7617	INPUT FILE LIST (MAXIMUM 9 ENTRIES)	
7641	0	0 MARKS END OF LIST
7642	* HIGH 11 BITS OF =N	* BIT 0 = 1 IF COMMAND LINE TERMINATED BY ALTMODE
7643	SPECIFIED OPTIONS	
7645		
7646	LOW 12 BITS OF =N	
7647	DEVICE HANDLER RESIDENCY TABLE	
7665		
7666	SYSTEM DATE WORD	
7667	READ OPERATION (LOAD KEYBOARD MONITOR)	
7677		
7700	USR CALL AND RETURN AREA	ENTRY TO USR
7740		
7741	USER DEVICE NAME TABLE	NOTE SYSTEM ODT DESTROYS CONTENTS OF THIS TABLE WHEN SETTING BREAKPOINTS
7757		
7760	DEVICE CONTROL WORD TABLE	
7776		
7777	UNUSED	RESERVED FOR FUTURE USE

B.3 System Device Tables

Each device is described to the system by entries in five system tables. Each of these tables is fifteen words long, where the device number is the index into the table. The five tables are described below.

B.3.1 Permanent Device Name Table

Entries in this table specify the permanent name of each device. The entries are computed by encoding the actual four-character device name in a single word as follows:

- a. The device name is expressed as two words in the standard DEVICE format. For example, if the device name were "PTR", the two words would be:

WORD 1: 2024
WORD 2: 2200

Note that when the device name is left justified; 0's are inserted to fill four characters.

- b. A single word is created by adding together these two words.
- c. If word 2 is non-zero, bit 0 of the resulting word is forced to be a one. For example, the table entry for "PTR" would be 4224.

An entry of zero means that there is no device for the corresponding device number.

NOTE

Conventionally, device names consist only of the characters A to Z and 0 to 9. The first character of the device name should be alphabetic. The coding used makes all one and two character device names unique; however, names of more than two characters are not unique. For example, "PTR" and "RTP" have the same encoding.

The Permanent Device Name Table is fifteen locations long; it resides in the USR. When the USR is in core the beginning of the table is in field 1 at a location the address of which is contained in location 10036.

B.3.2 User Device Name Table

Entries are made in this table whenever the user performs an ASSIGN and are restored to zero by a DEASSIGN. These entries have the same format as those in the Permanent Device Name Table.

The User Device Name Table resides in locations 17741 through 17757.

NOTE

The User Device Name Table is used by ODT for setting breakpoints. For this reason the user should never ASSIGN any user device names when debugging with ODT.

B.3.3 Device Handler Residency Table

When a device handler is loaded by the USR, the entry in this table for the device loaded (and entries for all devices whose handlers are co-resident, if any) is set to contain the entry point for the device handler. Entries other than those that contain an address above 7600 (thus referring to the system handler) are restored to 0 when a RESET, DECODE or CHAIN function is executed. When a program exits to the Keyboard Monitor this table is not cleared. The Keyboard Monitor Commands GET, RUN, R, SAVE, and START (with no explicit address) clear this table.

NOTE

Since the system device handler is always resident, the first entry (since SYS is always device number 1) in the Device Handler Residency Table is always 7607 (the entry point of the system device handler).

The Device Handler Residency Table resides in locations 17647 through 17665.

B.3.4 Device Handler Information Table

Each entry in this table contains all the information needed by the USR to load the corresponding handler. The format of these entries is as follows:

<u>Bit Condition</u>	<u>Meaning</u>
Bit 0 = 1	If this is a two page device handler.
Bits 1 to 4	Contain the relative block location of the device handler record on the system device. This is computed by subtracting 15 octal (one less than the first device handler block) from the actual block number.
Bits 5 to 11	Contain the offset of the handler entry point from the beginning of the page. Note that two page device handlers must have their entry points in the first page.

If an entry is 0 the corresponding device handler is not saved in any of the device handler storage blocks. This is always true of device number 1 (the system device) and for all device numbers that are not used in a given configuration. The Device Handler Information Table is 15 locations long and resides in the USR. When the USR is in core the beginning of the table is in field 1 at a location the address of which is contained in location 10037.

B.3.5 Device Control Word Table

Entries in this table specify special device characteristics, including the physical device type. The entry format is as follows:

<u>Bit Condition</u>	<u>Meaning</u>
Bit 0 = 1	If the device is file-structured.
Bit 1 = 1	If the device is read-only.
Bit 2 = 1	If the device is write-only.
Bits 3 to 8	Contain the physical device type code (described below),

Bit ConditionMeaning

Bits 9 to 11

For file structured devices, these bits contain the directory block number of the currently active tentative file. If bits 9 to 11 are zero, there is no active tentative file on the device. For non-file structured devices, bits 9 to 11 are always zero. Bits 9 to 11 are reset to zero by the commands GET, RUN, R, SAVE, START (with no explicit address) and optionally by the USR functions RESET and DECODE.

The device type is a number between 0 and 77₈, of which 0 through 20₈ are currently assigned to existing devices, as follows:

Device Type
CodeDevice

0	Teletype
1	High-speed paper tape reader
2	High-speed paper tape punch
3	Card Reader
4	Line Printer
5	RK8 Disk
6	256K Disk (RF08)
7	512K Disk (RF08 + RS08)
10	768K Disk (RF08 + 2 RS08's)
11	1024K Disk (RF08 + 3 RS08's)
12	32K Disk (DF32)
13	64K Disk (DF32 + DS32)
14	96K Disk (DF32 + 2 DS32's)
15	128K Disk (DF32 + 3 DS32's)
16	DEctape
17	LINctape (PDP-12 only)
20	Magnetic Tape
21-77	To be assigned

The Device Control Word Table resides in locations 17760 through 17776.

B.3.6 Device Length Table

There is a sixth table that is not normally considered part of the system tables. This is the Device Length Table and is used only by PIP to perform the /Z (zero directory) and /S

(compress device) options. This table is 64 locations long, one entry for each possible physical device type. In this table an entry of 0 means that the corresponding device is non-file structured; otherwise the entry contains the negative of the number of available 256-word blocks on the device.

For example, the entry for a 256K disk would be 6000_8 (minus 2000_8 , or 1024_{10} , 256-word blocks).

The Device Length Table resides in PIP. When PIP is brought into core the Device Length Table is in locations 13600 to 13677. When new device types are added to the system this table should be patched with ODT to reflect the device length of the new device.

APPENDIX C

SYSTEM ERROR CONDITIONS AND MESSAGES

This is a summary of all error messages that are a result of system errors. These errors are also described in the relevant sections of this manual and in the 8K Programming System User's Guide.

C.1 System Halts

Errors that occur as a result of a major I/O failure on the system device can cause a system halt to occur. These are as follows:

<u>Value of PC</u>	<u>Meaning</u>
00601	A read error occurred while attempting to load ODT. Return to the Keyboard Monitor by restarting at 07605.
07461	An error occurred while reading a program into core during a CHAIN. Return to the Keyboard Monitor by restarting at 07605.
07605	An error occurred while attempting to write the Keyboard Monitor area onto the system scratch blocks. Verify that the system device is <u>not</u> WRITE LOCKED and restart at location 07600 to try again.
07702	A user program has performed a JMS to 7700 in field 0. This is a result of trying to call the USR without first performing a CIF 10. As location 07700 has been destroyed, the user must <u>re-bootstrap</u> the system!
07764	A read error occurred while loading a program. Return to the Keyboard Monitor by restarting at 07605.
07772	A read error occurred on the system scratch area while loading a program. Return to the Keyboard Monitor by restarting at 07605.

<u>Value of PC</u>	<u>Meaning</u>
10066	An input error occurred while attempting to restore the USR. Return to the Keyboard Monitor by restarting at 07605.
10256	A read error occurred while attempting to load the Monitor Error routine. Return to the Keyboard Monitor by restarting at 07605.
17676	An error occurred while attempting to read the Keyboard Monitor from the system device. Try again by restarting at location 07605. DO NOT PRESS CONTINUE.
17721	An error occurred while saving the USR area. Verify that the system device is <u>not</u> WRITE LOCKED, and press CONTINUE to try again.
17727	An error occurred while attempting to read the USR from the system device. Return to the Keyboard Monitor by restarting at 07605.
17736	An error occurred while reading the scratch blocks to restore the USR area. Return to the Keyboard Monitor by restarting at 07605.

Also, there is one halt in the LOADER program:

00005	A parity error occurred when attempting to overlay the LOADER from the system scratch blocks. Return to the Keyboard Monitor by restarting at 07605, and try again.
-------	---

After retrying the operation which caused the failure, if the error persists it is the result of a hardware malfunction or a parity error in the system area. Run the appropriate diagnostic program to check the device and rebuild the system.

C.2 USR Errors

Fatal errors that occur during operation of the USR cause the message:

MONITOR ERROR n AT xxxxx

to be printed. In these cases, the value "n" describes the

error and "xxxxx" is the address of the call to the USR that caused the error. The six Monitor errors are:

<u>Message</u>	<u>Meaning</u>
MONITOR ERROR 1 AT xxxxx	File length in CLOSE function is too large.
MONITOR ERROR 2 AT xxxxx	An I/O error occurred while attempting to read or write a directory block. This is generally caused by the device being WRITE LOCKED.
MONITOR ERROR 3 AT xxxxx	The device handler required for a file operation (LOOKUP, ENTER, CLOSE) is not in core.
MONITOR ERROR 4 AT xxxxx	Illegal call to the USR; either an attempt has been made to call the USR from locations 10000 to 11777 or a device number of zero was specified.
MONITOR ERROR 5 AT xxxxx	I/O error occurred while reading or writing on the system device. Verify that the system device is <u>not</u> WRITE LOCKED.
MONITOR ERROR 6 AT xxxxx	Directory overflow occurred (see section A.1.2 for limitations on number of directory entries).

In addition to the MONITOR ERROR messages, system and user programs can use the USR to print:

USER ERROR n AT xxxxx

by using the ERROR function. In this case the value of "n" is user-defined and "xxxxx" is the address of the call to the USR.

Currently, two USER ERROR numbers have been assigned:

<u>Message</u>	<u>Meaning</u>
USER ERROR 0 AT xxxxx	An I/O error occurred while attempting to load a program with the GET, RUN, or R command.

<u>Message</u>	<u>Meaning</u>
USER ERROR 1 AT xxxxx	While running a FORTRAN or SABR program, an attempt was made to call a subroutine that had not been loaded.

Following either a MONITOR ERROR message or a USER ERROR message the USR exits to the Keyboard Monitor; the current contents of core are preserved and bit 2 of the Job Status Word is set to a 1 to prevent continuing from the error.

C.3 Keyboard Monitor Errors

In addition to the USR errors described previously, the following errors can occur after a command is given to the Keyboard Monitor:

<u>Message</u>	<u>Meaning</u>
aaaa?	The Keyboard Monitor cannot interpret the command "aaaa". For example, if the user types HELLO, the system will respond HELLO?
TOO FEW ARGS	An argument has been omitted from a command.
device NOT AVAILABLE	The permanent device name specified in an ASSIGN, SAVE, RUN, or GET command does not exist.
name NOT FOUND	The file name specified was not located on the device indicated. This error can also be caused by trying to RUN or GET from an output only device.
BAD ARGS	Arguments to a SAVE command are inconsistent.
ILLEGAL ARG.	Illegal syntax in a SAVE command.
SAVE ERROR	An I/O error occurred while saving the program. The contents of core remain intact.

<u>Message</u>	<u>Meaning</u>
BAD CORE IMAGE	The file requested with an R, RUN, or GET command is not a core image file.
NO!!	A START command (with no address specified) is prohibited when bit 2 of the Job Status Word (location 07746) is a 1.
BAD DATE	Improper syntax in a DATE command.
SYSTEM ERROR	An error occurred while doing I/O to the system device.

C.4 Command Decoder Errors

The following errors are printed by the Command Decoder. After the error message, the Command Decoder starts a new line, prints a *, and waits for another command line. The erroneous command is ignored.

<u>Message</u>	<u>Meaning</u>
ILLEGAL SYNTAX	The command line is formatted incorrectly.
TOO MANY FILES	More than three output files or nine input files were specified.*
device DOES NOT EXIST	The specified device name does not correspond to any permanent device name or any user assigned device name.
name NOT FOUND	The specified input file name was not found on the device indicated.

*In the special mode of the Command Decoder this message would be printed if more than one output file or five input files were specified. See section 3.5.

APPENDIX D

PROGRAMMING NOTES

This appendix is a potpourri of ideas and techniques that have proven useful in programming the PDP-8. PS/8 users may find some use in their own programs for the techniques mentioned here.

- D.1 The Default File Storage Device, DSK
- D.2 Modification to Card Reader Handler
- D.3 Suppression of Carriage Return/Line Feed in FORTRAN I/O
- D.4 Accessing the System Date in a FORTRAN Program
- D.5 Determining Core Size on PDP-8 Family Computers
- D.6 Relocating Code
- D.7 Using PRTCl2-F to Convert PS/8 DECTapes to PS/12 LINCTapes
- D.8 Notes on Loading Device Handlers
- D.9 Available Locations in the USR Area
- D.10 Accessing Additional Information Words in PS/8
- D.11 SABR Programming Notes

D.1 The Default File Storage Device, DSK

The Command Decoder, as noted earlier, makes certain assumptions about the I/O device where none is explicitly stated. Namely, on all output files where no device name is given, the device DSK is assumed. On the first input file where no device name is given, DSK is assumed. Subsequent input files assume the same device as the previous input file. This convention was adopted to simplify typing command lines.

The permanent device name DSK is assigned when the system is built. On all standard systems, DSK is equivalent to SYS. Changing the default file storage device is described in Chapter 5. A useful technique is to use the ASSIGN command to redefine the meaning of DSK temporarily. For example, where device DTAØ is equivalent to DSK and it becomes desirable to change DSK to DTAl, the following command can be given:

```
.ASSIGN DTAl DSK
```

DTAl remains the default file storage device until it is assigned a new name or a DEASSIGN command is executed. This technique is considerably easier to use than rebuilding the entire system.

D.2 Modification to Card Reader Handler

The standard card reader handler for PS/8 uses the DECØ29 standard card codes. Some installations may prefer to use the DECØ26 codes instead. This can be done by changing the card conversion table in CONFIG, reassembling CONFIG, and rebuilding the system, or by rebuilding the system using the following procedure:

- a. Follow steps 1, 2, and 3 given in section 8.2 of the 8K Programming System User's Guide (if building a DECTape system, mount the tape on which the system is to be built on unit #Ø, WRITE ENABLED, and load the DECTape CONFIG binary tape in step 3).

- b. Make the following patch:

<u>CHANGE LOCATION</u>	<u>FROM</u>	<u>TO</u>
5704	3203	7735
5705	4007	4076
5706	3502	0774
5714	7514	3314
5715	0577	1002
5716	3637	0305
5724	0104	3204
5725	1211	1273
5726	3374	3606
5727	0641	1341
5734	7316	3716
5735	3410	1175
5736	1376	3401

- c. Now continue building the system with step 4 of section 8.2 in the 8K Programming System User's Guide. The new system will have modified card codes.

NOTE

This procedure does not affect FORTRAN run time card input with READ (3,n). The conversion table for FORTRAN is UTILTY.SB on source DECTape #3.

Ø26 PUNCH CARD CODES

Octal 8-bit Code	DECØ26 Code	Character	Octal 8-bit Code	DECØ26 Code	Character
240	blank	SPACE	300	8-4	@
241	12-8-7	!	301	12-1	A
242	0-8-5	"	302	12-2	B
243	0-8-6	#	303	12-3	C
244	11-8-3	\$	304	12-4	D
245	0-8-7	%	305	12-5	E
246	11-8-7	&	306	12-6	F
247	8-6	'	307	12-7	G
250	0-8-4	(310	12-8	H
251	12-8-4 ¹)	311	12-9	I
252	11-8-4	*	312	11-1	J
253	12	+	313	11-2	K
254	0-8-3	,	314	11-3	L
255	11	-	315	11-4	M
256	12-8-3	.	316	11-5	N
257	0-1	/	317	11-6	O
260	0	0	320	11-7	P
261	1	1	321	11-8	Q
262	2	2	322	11-9	R
263	3	3	323	0-2	S
264	4	4	324	0-3	T
265	5	5	325	0-4	U
266	6	6	326	0-5	V
267	7	7	327	0-6	W
270	8	8	330	0-7	X
271	9	9	331	0-8	Y
272	11-8-2	:	332	0-9	Z
273	0-8-2	;	333	11-8-5	[
274	12-8-6	<	334	8-7	\
275	8-3	=	335	12-8-5]
276	11-8-6	>	336	8-5	^
277	12-8-2	?	337	8-2 ²	_

¹ On some IBM 026 Keyboards this character is graphically represented as □ .

² A card containing an 8-2 in column 1 with all remaining columns blank is an end-of-file card.

D.3 Suppression of Carriage Return/Line Feed in FORTRAN

It is often desirable to suppress the automatic carriage return/line feed (CR/LF) following FORTRAN WRITE statements to achieve an easily readable text. The following three methods in PS/8 FORTRAN can be used to achieve this result:

- a. Follow the I/O list of a WRITE statement with a comma. Thus, the following statements:

```
      WRITE (1,100) N,  
100  FORMAT (1X,15HTHE VALUE OF A(,I2,5H) IS )  
      READ (1,101) A(N)  
101  FORMAT (F8.4)
```

result in the following single line (assume N has a value of 12 and a value of 147.83 is being input):

THE VALUE OF A(12) IS 147.83

- b. Use of an empty field print statement enables a text to be printed without a following CR/LF when there is no variable to be printed. For example:

```
      WRITE (1,102) IDUMMY,  
102  FORMAT ('DESIRED TEXT',I0)
```

- c. READ statement using break character, as follows:

```
      READ (1,101) IA,IB,IC  
101  FORMAT ('A=',I1,'B=',I1,'C=',I1)
```

results in no CR/LF after each phrase is printed. That is, the output is all printed on a single line.

D.4 Accessing the System Date in a FORTRAN Program

The availability of the system Date word in location 17666 is useful to many PS/8 programs. The following FORTRAN program illustrates how the Date can be accessed in SABR code:

```
C          PROGRAM PRINTS THE CURRENT DATE
C
S          DUMMY DATE
S          TAD I DATE
S          DCA TEMP
S          TAD TEMP
S          AND (7
S          DCA \IYR
S          TAD TEMP
S          RAR;RTR
S          AND (37
S          DCA \IDAY
S          TAD TEMP
S          CLL RAL;RTL;RTL
S          AND (17
S          DCA \IMO
1000        WRITE (1,1000) IMO,IDAY,IYR
          FORMAT (/'DATE: 'I2'-'I2'-197'I1/)
          CALL EXIT
S          CPAGE 2
SDATE,     6211
S          7666
STEMP,     0
```

D.5 Determining Core Size on PDP-8 Family Computers

Many times system programs need to determine the amount of core available to them at run time. For example, the PS/8 system programs LOADER, PAL8, and CREF perform this calculation. Because of differences in the extended memory control of PDP-8 family computers, subroutines that work on one machine might not work on another.

The following three conditions cause the most difficulty:

- a. On a PDP-8 with an extended memory control, addressing nonexistent memory from field 0 causes the following instruction to be skipped and the contents of the corresponding field 0 location to be executed. For example:

CDF 70	/NONEXISTENT FIELD
TAD I (X)	/EXECUTED LOCATION X
HLT	/THIS INSTRUCTION SKIPPED

.
.
.

X, CLA CLL CML RAR /LOAD 4000

the preceding code causes 4000 to be loaded into the AC and the HLT instruction to be skipped when executed on a PDP-8.

- b. On a PDP-12 with an odd number of 4K banks (12K, 20K, 28K), all reads in the first nonexistent field load zeros. Reads to higher fields, as well as all reads to nonexistent memory on a machine with an even number of 4K banks load all one bits.
- c. The PDP-8/L normally treats all CDF's to fields 2 through 7 as NOP's. (It tests bits 6 and 7 of all CDF and CIF instructions for 0's before executing the IOT.) However, there is a special 12K option for the PDP-8/L called a BM08. With this option a CDF to field 2 is valid, but a CDF to field 3 resets the Data Field to 0. CDF's to fields 4 through 7 remain NOP's.

For those who are interested, the following subroutine has been tested on the PDP-8, 8/S, 8/L, 8/I, 8/E, PDP-12, and LINC-8 computers. For the purpose of this example, it is assembled at 00200. This is not essential, it can be in any 40 (octal) locations of any page in field 0.

/SUBROUTINE TO DETERMINE CORE SIZE.

/THIS SUBROUTINE WORKS ON ANY PDP-8 FAMILY
/COMPUTER. THE VALUE, FROM 1 TO 10 (OCTAL),
/OF THE FIRST NON-EXISTENT MEMORY FIELD IS
/RETURNED IN THE AC.

/NOTE -- THIS ROUTINE MUST BE PLACED IN FIELD 0

0200	0000	CORE,	0		
0201	7300		CLA CLL		
0202	6201	COR0,	CDF 0		/(NEEDED FOR PDP-8L)
0203	1237		TAD	CORSIZ	/GET FIELD TO TEST
0204	7006		RTL		
0205	7004		RAL		
0206	0217		AND	COR70	/MASK USEFUL BITS
0207	1232		TAD	COREX	
0210	3211		DCA	.+1	/SET UP CDF TO FIELD
0211	6201	COR1,	CDF	/N	/N IS FIELD TO TEST
0212	1635		TAD I	CORLOC	/SAVE CURRENT CONTENTS
0213	7000	COR2,	NOP		/(HACK FOR PDP-8!)
0214	3211		DCA	COR1	
0215	1213		TAD	COR2	/7000 IS A "GOOD" PATTERN
0216	3635		DCA I	CORLOC	
0217	0070	COR70,	70		/(HACK FOR PDP-8.,NO-OP)
0220	1635		TAD I	CORLOC	/TRY TO READ BACK 7000
0221	7400	CORX,	7400		/(HACK FOR PDP-8.,NO-OP)
0222	1221		TAD	CORX	/GUARD AGAINST "WRAP AROUND"
0223	1236		TAD	CORV	/TAD (1400)
0224	7640		SZA CLA		
0225	5232		JMP	COREX	/NON-EXISTENT FIELD EXIT
0226	1211		TAD	COR1	/RESTORE CONTENTS DESTROYED
0227	3635		DCA I	CORLOC	
0230	2237		ISZ	CORSIZ	/TRY NEXT HIGHER FIELD
0231	5202		JMP	COR0	
0232	6201	COREX,	CDF 0		/LEAVE WITH DATA FIELD 0
0233	1237		TAD	CORSIZ	/1ST NON-EXISTENT FIELD
0234	5600		JMP I	CORE	
0235	0221	CORLOC,	CORX		/ADDRESS TO TEST IN EACH FIELD
0236	1400	CORV,	1400		/7000+7400+1400 = 0
0237	0001	CORSIZ,	1		/CURRENT FIELD TO TEST

D.6 Relocating Code

One useful programming trick is generating relocated code by means of the ENPUNCH and NOPUNCH features of PAL8.

In this case, relocated code is code that, for some reason, is to be loaded into an area of core different from the area in which it is to be executed. For example, the system device handler for PS/8 is loaded into 6600 through 6777, so as not to affect the Binary Loader, and during the build process it is moved to the top page of field 0 where it resides. Of course, it cannot be simply assembled directly into 6600, since various address constants would be generated incorrectly. The way around this situation is to do two origins: the first to the location in which the code is loaded and the second to the location in which it is eventually executed. The second origin is preceded by a NOPUNCH so that no origin punch is put onto the binary output of PAL8.

For example, if some code were to be loaded into 1277 through 1476 but executed at 2000 through 2377, the following should appear in the source file preceding the code:

```
*1277                /ADDRESS TO LOAD
NOPUNCH
*2000                /ADDRESS OF EXECUTION
ENPUNCH
.                    /CODE BEGINS HERE
.
.
*1477                /RESET ACTUAL ASSEMBLY ORIGIN
```

This technique is used in several places in the source of PS/8.

NOTE

Code that is relocated in this fashion must not use current page literals as they will be loaded into the wrong area. In addition, current page literals should not be used in any code that immediately precedes the relocated code, and that is to be loaded onto the same page.

D.7 Using PRTC12-F to Convert PS/8 DECTapes to PS/12 LINCTapes

Many users of PS/8 on the PDP-12 will be interested in the fact that, since PS/8 uses an identical file structure on all devices, PDP-8 DECTape in PS/8 format may be directly copied to PS/8 LINCTapes by the PRTC12-F program.

The PRTC12-F program uses the PDP-12 TC12-F hardware option to read DECTapes and convert these tapes to LINCTape. This hardware option is required to read DECTapes on the PDP-12.

The PRTC12-F program is described in the document DEC-12-YIYA-D. This document describes the program operation in detail, and must be read before attempting to use PRTC12-F. The operations that convert PS/8 format DECTapes are as follows:

- a. Mount the PS/8 DECTape on unit 1 and a PDP-12 LINCTape formatted with 129 words per block on unit 2.
- b. When the READ questionnaire is displayed, respond as follows (responses are underlined; the character) stands for carriage return and ↓ stands for line feed):

```
READ 1777) BLOCKS
TAPE FORMAT A) UNIT 1)
STARTING WITH BLOCK 0)↓
etc.
```

- c. When the WRITE questionnaire is displayed, respond as follows:

```
WRITE THE RESULT
IN TAPE FORMAT B) ON UNIT 2)
STARTING AT BLOCK 0)↓
etc.
```

D.8 Notes on Loading Device Handlers

A. Problem with multiple input files

There is a problem associated with reusing Device Handler areas in PS/8. This problem is best illustrated by an example:

Assume a program has reserved locations 1000-1377 for its input handler and locations 7400-7577 for its output handler. If the program gives a `USR FETCH` command to load the `DTA1` handler as an input device handler, all 8 DECTape handlers will load into 1000-1377, since they are all co-resident. If another `FETCH` is issued to load the `DTA2` handler as an output device handler, that handler will not be loaded, because it shares space with the `DTA1` handler currently in core. This is fine -- however, if the user now switches input devices and `FETCHes` the paper tape reader handler as an input device handler it will destroy the `DTA2` handler and the next attempt to output using the `DTA2` handler will produce errors. There are two ways to get around this problem.

1. Always assign the handler which you expect to stay in core the longest first. Most programs can process more than one input file per program step (e.g., an assembly pass is one program step) but only one output file; therefore, they assign the output handler before any of the input handlers. In the above example, the problem would be eliminated if the `DTA2` handler were assigned first.
2. Always give a `USR RESET` call before each `FETCH`. Obviously, this call should not delete any open output files. This means that the `USR` will always load the new handler, even if another copy is in core. The user must `FETCH` the output handler again before issuing the `USR CLOSE` call, otherwise the

USR will determine that the output handler is not
in core and give a MONITOR ERROR 3 message.

8K FORTRAN uses this second method for device-independent I/O
at run time.

B. Dynamically Loading Device Handlers

Some programs which use dynamic core allocations will want to
use PS/8 Device handlers but cannot afford to always allocate
the maximum of two pages per handler. The following is a sub-
routine which loads a device handler dynamically, returning its
entry in the AC. It assumes that the name of the handler is in
locations NAME1 and NAME2, and a subroutine GETPAG exists which
gets a page from the bottom of available field 0 of storage and
returns its address in the AC. This example subroutine runs in
field 1 and can only be called from field 1, but can be re-
written for any other possibility.

```

ASSIGN, 0
        TAD NAME1
        DCA N1
        TAD NAME2
        DCA N2          /MOVE DEVICE NAME INTO "INQUIRE" COMMAND
        CDF CIF 10
        JMS I (7700
10      /USRIN - FORCE USR INTO CORE
        JMS I (200
12      /INQUIRE
N1,     0
N2,     0
LOC1,   0
        JMP ASSERR    /NO SUCH DEVICE - QUIT
        TAD LOC1
        SZA           /IS THE HANDLER ALREADY IN CORE?
        JMP I ASSIGN  /YES - RETURN ITS ENTRY POINT
        JMS GETPAG    /GET A PAGE DYNAMICALLY
        DCA LOC2
ASSTRY, TAD N2        /LOAD DEVICE NUMBER
        JMS I (200
1      /FETCH
LOC2,   0             /PAGE TO FETCH INTO
        JMP TWOPAG    /FAILED - MUST BE A TWO-PAGE HANDLER
        TAD LOC2
        JMP I ASSIGN  /RETURN ENTRY POINT
TWOPAG, JMS GETPAG    /GET ANOTHER PAGE
        ISZ LOC2      /SET "TWO PAGE HANDLER ALLOWED" BIT
        CLA
        JMP ASSTRY    /FETCH WILL SUCCEED THIS TIME
ASSERR, .....       /ERROR ROUTINE

```

D.9 Available Locations in the USR Area

A few programs may need additional storage space in field 1 when the USR is in core. A number of locations in the USR area (10000 to 11777) are available and may be used whenever the USR is in core. The locations are as follows:

- A. Locations 10000 to 10006 are available for scratch storage and/or ODT breakpoint usage, without restriction.
- B. All auto-index registers (locations 10010 to 10017) may be used, but these locations are destroyed by USR operations.
- C. Location 10020 to 10037 may be used as scratch storage with no restrictions.
- D. Locations 11400 to 11777 are used by the USR to preserve the last directory segment read while performing a LOOKUP, ENTER, or CLOSE operation. Location 10007 contains a key specifying which segment of which device is currently in core.

Any user program may use locations 11400 to 11777 as scratch storage as long as location 10007 is set to 0 before the first use. Of course, the LOOKUP, ENTER, and CLOSE operations will read a directory segment into 11400 to 11777 and set 10007 to a non-zero value again.

D.10 Accessing Additional Information Words in PS/8

In all of these cases, the USR must have been previously brought into core with the USRIN function.

A. After a LOOKUP or ENTER

After a LOOKUP or ENTER, location 10017 points to the length word of the file entry. To get a pointer to the first Additional Information Word, a program would execute the following code:

```
CDF 10
TAD I (1404           /GET # OF ADDITIONAL INFORMATION WORDS
                        /FROM DIRECTORY

SNA
JMP NONE             /NO ADDITIONAL INFORMATION WORDS
TAD I (0017
DCA POINTER
.
.
.
```

"POINTER" now points to the first Additional Information Word.

B. After a CLOSE

Because CLOSE is a legal operation even if no output file is present, it is not suggested that Additional Information Words be modified following a CLOSE. To alter the Additional Information Words of a permanent file, do a LOOKUP to get the directory segment into core, then alter the words and rewrite the directory segment.

C. Rewriting the Current Directory Segment

Whenever a user program changes the Additional Information Words, of a file, it must rewrite the directory segment containing that file entry in order to make sure the changes are permanently recorded.

The following code, which must be in field 1, will rewrite the current directory segment:

CDF 1Ø	/CODE IS IN FIELD 1
TAD 7	/GET DIRECTORY KEY WORD
AND (7	/EXTRACT SEGMENT NUMBER
DCA SEGNO	
CIF Ø	
JMS I 51	/LOC 51 POINTS TO THE DEVICE HANDLER
421Ø	/WRITE OPERATION
1400	/DIRECTORY SEGMENT CORE ADDRESS
SEGNO, Ø	
JMP ERROR	/ERROR REWRITING DIRECTORY

Location 10051 will always point to the device handler entry point used to read in the last directory segment, following a LOOKUP or ENTER operation.

D.11 SABR Programming Notes

A. Optimizing SABR Code

There are two types of users who will be using the SABR assembler - those who like the convenience of page-boundary-independent code and are willing to pay the price for it, and those who need a relocatable assembler but are still very location conscious. These optimizing hints are directed to the latter user.

One way to beat the high cost of non-paged code is to Page It Yourself. This is done by using the LAP (Leave Automatic Paging) pseudo-op and the PAGE pseudo-op to force paging where needed. This saves 2 to 4 instructions per page from elimination of the page escape. In addition, the fact that the program must be properly segmented may save a considerable amount.

Wasted core may be reduced by eliminating the ever-present CDF instructions which SABR inserts into a program. This is done by using "fake indirects". Define the following op codes:

```
OPDEF   ANDI  0400
OPDEF   TADI  1400
OPDEF   ISZI  2400
OPDEF   DCAI  3400
      .
      .
      .
```

These codes correspond to the PDP-8 memory reference instructions but they include an indirect bit. The difference can best be appreciated by an example:

If X is off-page, the sequence

```
    LABEL, SZA
        DCA  X
```

is assembled by SABR into

```

    LABEL, SZA
        JMS 45
        SKP
        DCA I (X)

```

or four instructions and one literal.

The sequence

```

    PX, X
      .
      .
      .
    LABEL, SZA
        DCAI PX

```

assembles into three instructions for a saving of 40 percent. Note, however, that the user must be sure that the data field will be correct when the code at LABEL is encountered. Also note that the SABR assumes that the Data Field is equal to the Instruction Field after a JMS instruction, so subroutine returns should not use the JMPI op code.

The standard method to fetch a scalar integer argument of a subroutine in SABR is:

	<u>Code</u>
IARG,	DUMMY X
	Ø
SUBR,	BLOCK 2
	TAD I SUBR
	DCA X
	INC SUBR#
	TAD I SUBR
	DCA X#
	INC SUBR#
	TAD I X
	DCA IARG
	.
	.
	.
X,	BLOCK 2

This code requires 19 words of core and takes several hundred

microseconds to execute. The following sequence:

	<u>Code</u>	
IARG,	Ø	
SUBR,	BLOCK 2	
	TAD I SUBR	
	DCA X	
	INC SUBR#	
	TADI SUBR#	
	DCA IARG	
	INC SUBR#	
X,	HLT	/THIS IS A CDF
	TAD I IARG	
	DCA IARG	
	.	
	.	
	.	

takes only 14 words and executes in approximately 1/3 the time.

B. Calling the USR and Device Handlers from SABR code

One important thing to remember is that any code which calls the USR must not reside in locations 10000 to 11777. Therefore, any SABR routine which calls the USR must be loaded into a field other than field 1 or above location 2000 in field 1. To call the USR from SABR use the sequence:

CPAGE n	/N=7+(# OF ARGUMENTS)
6212	/CIF 1Ø
JMS 77ØØ	/OR 2ØØ IF USR IN CORE
REQUEST	
ARGUMENTS	/OPTIONAL DEPENDING ON REQUEST
ERROR RETURN	/OPTIONAL DEPENDING ON REQUEST

To call a device handler from SABR use the sequence:

CPAGE 12	/1Ø IF "HAND" IN PAGE Ø
62Ø2	/CIF Ø
JMS I HAND	/DO NOT USE JMSI
FUNCT	
ADDR	
BLOCK	
ERROR RETURN	
SKP	
HAND, Ø	/"HAND" MUST BE ON SAME PAGE
	/AS CALL, OR IN PAGE Ø!!

APPENDIX E

CHARACTER CODES AND CONVENTIONS

Table E-1 contains a list of the control characters used by PS/8 and associated system programs. Table E-2 contains the PS/8 character set, which is a subset of the complete ASCII code, the unlisted codes are generally not used by PS/8 or the system programs. Note the following:

- a. On some terminals, the character back-arrow (+) is replaced by an underline (_) character, and the up-arrow (↑) is replaced by circumflex (^).
- b. Some terminals use parity codes rather than forcing the leading bit of the 8-bit character code to be a 1. To avoid problems, PS/8 system programs always ignore the parity bit during ASCII input.
- c. PS/8 does not handle lower case characters (octal codes 341 through 372).

Table E-1

PS/8 Control Characters

<u>Octal 8-bit Code</u>	<u>Character Name</u>	<u>Remarks</u>
000	null	Ignored in ASCII input.
200	leader/trailer	Leader/trailer code precedes and follows the data portion of binary files.
203	CTRL/C	PS/8 break character, forces re- turn to Keyboard Monitor, echoed as ↑C.
207	BELL	CTRL/G.
211	TAB	CTRL/I, horizontal tabulation.
212	LINE FEED	Used as a control character by the Command Decoder and ODT.
213	VT	CTRL/K, vertical tabulation.
214	FORM	CTRL/L, form feed.
215	RETURN	Carriage return, generally echoed as carriage return followed by a line feed.
217	CTRL/O	Break Character, used conventionally to suppress Teletype output, echoed as ↑O.
225	CTRL/U	Delete current input line, echoed as ↑U.
232	CTRL/Z	End-of-File character for all ASCII and binary files (in relocatable binary files CTRL/Z is not a termi- nator if it occurs before the trailer code).

Table E-1

PS/8 Control Characters

<u>Octal 8-bit Code</u>	<u>Character Name</u>	<u>Remarks</u>
233	ESC	Escape replaces ALTMODE on some terminals. Considered equivalent to ALTMODE.
375	ALTMODE	Special break character for Teletype input.
376	PREFIX	PREFIX replaces ALTMODE on some terminals. Considered equivalent to ALTMODE.
377	RUBOUT	Key is labeled DELETE on some terminals. Deletes the previous character typed.

Table E-2

ASCII Character Codes

<u>Octal 8-bit Code</u>	<u>6-bit Code</u>	<u>Punched Card Code (1)</u>	<u>Character Representa- tion</u>	<u>Remarks</u>
240	40	blank		space (non-printing)
241	41	11-8-2	!	exclamation point
242	42	8-7	"	quotation marks
243	43	8-3	#	number sign (10)
244	44	11-8-3	\$	dollar sign
245	45	0-8-4	%	percent
246	46	12	&	ampersand
247	47	8-5	'	apostrophe or acute accent
250	50	12-8-5	(opening parenthesis
251	51	11-8-5)	closing parenthesis
252	52	11-8-4	*	asterisk
253	53	12-8-6	+	plus
254	54	0-8-3	,	comma
255	55	11	-	minus sign or hyphen
256	56	12-8-3	.	period or decimal point
257	57	0-1	/	slash
260	60	0	0	
261	61	1	1	
262	62	2	2	
263	63	3	3	
264	64	4	4	
265	65	5	5	
266	66	6	6	
267	67	7	7	
270	70	8	8	
271	71	9	9	
272	72	8-2	:	colon
273	73	11-8-6	;	semicolon
274	74	12-8-4	<	less than
275	75	8-6	=	equals
276	76	0-8-6	>	greater than
277	77	0-8-7	?	question mark
300	00	8-4	@	at sign
301	01	12-1	A	
302	02	12-2	B	
303	03	12-3	C	
304	04	12-4	D	
305	05	12-5	E	
306	06	12-6	F	
307	07	12-7	G	

Table E-2 (Cont'd)
ASCII Character Codes

<u>Octal 8-bit Code</u>	<u>6-bit Code</u>	<u>Punched Card Code (1)</u>	<u>Character Representa- tion</u>	<u>Remarks</u>
310	10	12-8	H	
311	11	12-9	I	
312	12	11-1	J	
313	13	11-2	K	
314	14	11-3	L	
315	15	11-4	M	
316	16	11-5	N	
317	17	11-6	O	
320	20	11-7	P	
321	21	11-8	Q	
322	22	11-9	R	
323	23	0-2	S	
324	24	0-3	T	
325	25	0-4	U	
326	26	0-5	V	
327	27	0-6	W	
330	30	0-7	X	
331	31	0-8	Y	
332	32	0-9	Z	
333	33	12-8-2 (5)	[opening bracket, SHIFT/K
334	34	11-8-7 (6)	\	backslash, SHIFT/L (8)
335	35	0-8-2]	closing bracket, SHIFT/M
336	36	12-8-7 (7)	^	circumflex (2)
337	37	0-8-5 (3)	_	underline (4,9)

Footnotes:

- (1) These are the DEC029 standard card codes.
- (2) On most DEC Teletypes circumflex is replaced by up-arrow (↑).
- (3) A card containing 0-8-5 in column 1 with all remaining columns blank is an end-of-file card.
- (4) On most DEC Teletypes underline is replaced by backarrow (←).
- (5) On some IBM 029 keyboards this character is graphically represented as a cent sign (¢).
- (6) On some IBM 029 keyboards this character is graphically represented as logical NOT (¬).
- (7) On some IBM 029 keyboards this character is graphically represented as vertical bar (|).
- (8) On a very few LP08 line printers, the character diamond (◊) is printed instead of backslash.
- (9) On a very few LP08 line printers, the character heart (♥) is printed instead of underline.
- (10) The character number sign on some terminals is replaced by pound sign (£).

APPENDIX F

DOCUMENTATION UPDATE FOR THE 8K PROGRAMMING SYSTEM USER'S GUIDE

This appendix contains two sections on new PS/8 system programs: PS/8 CREF, a cross reference listing program, and LIBSET, a new program for building a library of FORTRAN subroutines.

F.1 CREF, Cross-Reference Program (DEC-P8-YRXA-PB)

CREF is a PS/8 program which aids the programmer in writing, debugging and maintaining assembly language programs. CREF provides the ability to pinpoint all references to a particular symbol in assembly language programs. CREF operates on output from either the PAL8 or SABR assemblers.

F.1.1 Loading, Calling, and Using CREF

In order to load CREF, place the CREF binary tape (DEC-P8-YRXA-PB) into the reader and load as follows:

```
.R ABSLDR
*PTR:/9$
```

where the \$ character indicates typing of the ALT MODE key. When the ↑ character is printed, type any keyboard character to initiate reading the paper tape. When reading is completed, Keyboard Monitor responds with another dot. Type:

```
.SAVE SYS:CREF
```

and CREF is saved on the system device.

To call CREF from the system device, type:

```
.R CREF
```

where the . is the Keyboard Monitor active signal. The Command Decoder is then loaded and replies with a star at the left margin. The user then enters one output file specification and one input file specification.

NOTE

The input to CREF must be the listing pass output from either assembler. If this is not the case, CREF will not operate properly.

If no output file is specified, CREF assumes the output is to be sent to the line printer. If no input file extension is specified, the extension .LS is assumed. If no input specification is given, control returns to the Command Decoder until an input file is specified.

An example of calling CREF is shown below:

```
.R CREF
*PTMP
```

The Command Decoder prints *, CREF assigns LPT: as the output device. The input file is DSK:PTMP.LS. If the file PTMP.LS is not found, a search for DSK:PTMP is attempted.

No output file extensions are appended. Options which can be specified to CREF are described in Table F-1.

TABLE F-1

CREF OPTIONS

<u>Option Code</u>	<u>Meaning</u>
/X	Do not process literals. For programs with too many symbols and literals for CREF, this option may create enough space for CREF to operate.

<u>Option Code</u>	<u>Meaning</u>
/R	Interpret input as SABR code. Signal to CREF to accept special SABR characters. Also, if R is used, /X is forced on.
/P	Disables pass one listing output. The output is re-enabled when \$ (or END if SABR code) is encountered. Thus the \$ (END) and symbol table are printed if /P is used.

Examples of calling CREF are shown below.

```
.R CREF
*SBRLS/R
```

The line to the Command Decoder causes output to be sent to the line printer. The input is expected to be a SABR listing file named SBRLS.LS or SBRLS from device DSK:.

```
.R CREF
*DTA1:LIST< DTA3:PALIST/X
```

The above line to the Command Decoder causes output to be sent to DEctape unit 1, to a file named LIST. Input is expected to be a PAL8 listing file called PALIST.LS or PALIST. No literals appear in the CREF output table.

F.1.2 Interpreting CREF Output

The output from CREF consists of two parts. On the first pass through the input file, CREF generates a sequence numbered listing of the file. The sequence numbers are decimal. /P disables this part of the output.

Following the listing, the cross reference table appears. This table contains every user defined symbol and literal sorted alphabetically. For each symbol, there appears a list of numbers specifying the lines in which that symbol is referenced.

If CREF finds too many references to fit in core at one time, multiple passes are required to process all symbols. The minimum number of passes is two. The maximum depends on two things:

- a. Size of the input file, and
- b. Amount of core available

CREF calculates the number of core fields available and uses all available space for reference tables. If enough core is not available, three or more passes are required.

For example, the current PS/8 SABR assembler (5518 source lines, 849 symbols) requires a total of four passes through CREF on an 8K machine.

/EXAMPLE CREF OUTPUT

PAL8-V6

PAGE 1

```

1          /EXAMPLE CREF OUTPUT
2          *200
3          /THIS IS WHAT TYPICAL CREF OUTPUT LOOKS LIKE
4
5      0200 1211          TAD A1
6      0201 3212          DCA COUNT          /SET COUNTER
7      0202 7001          IAC
8      0203 7104          CLL RAL
9      0204 2212          ISZ COUNT
10     0205 5203          JMP .-2
11     0206 3213          DCA F2
12     0207 1377          TAD (7777
13     0210 7402          HLT
14     0211 7400  A1,      -400
15     0212 0000  COUNT,   0
16     0213 0000  F2,      0
17     0377 7777
18
          $

```

(FORM FEED)

/EXAMPLE CREF OUTPUT

PAL8-V6

PAGE 1-1

```

A1      0211
COUNT  0212
F2      0213
19

```

(FORM FEED)

```

A1          5      14#
COUNT      6      9      15#
F2          11     16#
_L0377      12

```

Form feeds on the Teletype are converted to a series of carriage return/line feeds and a dotted tear line. Notice that in the CREF table the line where the symbol is defined is followed by a #. Symbols defined by OPDEF or SKPDF in SABR and all literals do not have a # following them.

F.1.3 CREF Pseudo-Ops

CREF recognizes the pseudo-ops of the assembler whose output it is processing. Certain pseudo-ops cause CREF to perform actions similar to those taken by the assembler.

<u>PAL8 Pseudo-Op</u>	<u>Action Taken by CREF</u>
EXPUNGE	CREF purges its current symbol table of all permanent and user defined symbols. If any literals were in the table they are not deleted.
FIXTAB	Causes all symbols (except literals) to be marked as permanent symbols. After a FIXTAB, no references to previously defined symbols will be reported by CREF.
TEXT	Ignores characters between delimiters.
\$	End of input signal.
<u>SABR Pseudo-Op</u>	<u>Action Taken by CREF</u>
END	End of input signal.
OPDEF	Creates a new permanent symbol, a non-skip type instruction.
SKPDF	Creates a new permanent symbol, a skip-type instruction.

NOTE

Symbols entered by OPDEF and SKPDF are processed by CREF. All references to these defined symbols are listed. However, no reference is flagged as a definition (i.e., no reference is followed by a # in the CREF listing).

TEXT	Ignore characters between delimiters.
------	---------------------------------------

F.1.4 Restrictions

CREF has the following restrictions:

- a. Input format -- CREF can only detect errors of a simple form (described below). If the input is neither a PAL8 or SABR listing file, the results of CREF are unpredictable in the cross reference table.
- b. CREF can handle a maximum of 896 (decimal) symbols. In 8K, PAL8 is limited to 897 symbols while SABR is limited to fewer than 800 symbols. If more than 896 symbols are found, an error message is generated.
- c. If any symbol in the input file has more than 2044 (decimal) references, an error message is generated.
- d. If more than 8192 (decimal) source lines are input, sequence numbers return to 4096, not 0.
- e. If the /X option is used in PAL8 (to generate a DDT compatible symbol table) and the output listing is put through CREF, no symbol table listing will appear.
- f. Use of semi-colons -- This is a restriction which, when not observed, could cause errors in the CREF table. It is recommended that the user follow these suggestions when preparing source files to insure a proper CREF listing.
 1. Semi-colons should not be used on lines with pseudo-ops. In particular, a combination such as the following must not be used:

```
*30000
TEXT %ERROR% ; TAD [42
```

In this case, CREF does not process the page zero literal properly. A literal is generated which is derived from the expanded TEXT message. No error message is generated, but the literal table entry is meaningless.

2. When using conditional code a good rule to observe is to not use semi-colons inside conditional code. For example:

```
EXOR = Ø
IFNZRO EXOR <CLA;TAD B; HLT /ERROR>
/THIS IS THE NEXT LINE PAST IFNZRO
```

The conditional code is not assembled but CREF does not realize that and does try to process the bracketed instructions. As a result of these semi-colons, extra symbols may be processed and some valid references missed. If the code had been assembled, however, CREF would operate properly.

There are two solutions to this restriction:

Write straight line code:

```
EXOR = Ø
IFNZRO EXOR <
CLA
TAD B
HLT ERROR
>
```

or use XLIST around conditional code, in the above example:

```
IFZERO EXOR <XLIST>
IFNZRO EXOR <CLA;TAD B; HLT/ERROR>
IFZERO EXOR <XLIST>
```

XLIST turns off the listing if the code does not assemble and turns it back on after the conditional code.

- h. Formats -- There are several output formats that can be used in generating a PAL8 listing file:

```
/T  Form feeds converted to carriage return/
    line feeds

/H  No headings or form feeds generated

/D  DDT compatible symbol table is generated.
```

For best results with CREF, none of these switches should be used. This generates a heading and form feeds in the output. CREF automatically converts form feeds to carriage return/line feeds if output is to the Teletype.

F.1.5 CREF Error Messages

CREF errors are non-recoverable errors, and control returns to the Keyboard Monitor through 07605 (no core saved). CREF is not restartable and typing START results in

NO!!

being printed as a reply.

<u>Error Message</u>	<u>Meaning</u>
SYM OVERFLOW	More than 896 (decimal) symbols and literals were encountered.
ENTER FAILED	Entering an output file was unsuccessful, possibly output was specified to a read only device.
OUT DEV FULL	The output device is full (directory devices only).
CLOSE FAILED	CLOSE on output file failed.
INPUT ERROR	A read from input device failed.
DEV LPT BAD	The default output device, LPT, cannot be used as it is not available on this system.
2045 REFS	More than 2044 (decimal) references to one symbol were made.
HANDLER FAIL	Fatal error on output. Can occur if either the system device or the selected input device is write locked.

F.2 LIBSET (DEC-P8-SYXB-PB)

LIBSET, the FORTRAN Library Setup program, creates a library of subroutines from the relocatable binary output of SABR. These library files can be quickly and efficiently scanned by the Linking Loader saving a great deal of time in loading frequently used subroutines. How the LOADER uses relocatable library files, including automatic loading the LIB8.RL file and the /L option, is described in the 8K Programming System User's Guide.

F.2.1 Loading, Calling, and Using LIBSET

The LIBSET program is available from the Program Library on binary paper tape (DEC-P8-SYXB-PB). In order to load LIBSET, place the binary tape into the reader and load as follows:

```
.R ABSLDR
*PTR:=126000$
```

where the character \$ indicates typing of the ALT MODE key. When the ↑ character is printed, type any keyboard character to initiate reading of the paper tape. When reading is completed, the Keyboard Monitor responds with another dot. Type:

```
.SAVE SYS:LIBSET
```

and LIBSET is saved on the system device.

To call LIBSET from the system device, type:

```
.R LIBSET
```

in response to the dot printed by the Keyboard Monitor. The Command Decoder then prints a star at the left margin of the teleprinter paper and waits to receive a line of input. The general form of input required to build a library file is:

```
*output<(input file list)
*(additional input files)$
```

No more than nine input files are allowed on any one line, but several input lines can be entered. The last input line must end with the user typing the ALT MODE key (which echoes as \$). Only the first line can contain an output file. If no output file is specified a file named LIB8.RL is created on the system device. The assumed extension for both input and output files is .RL.

NOTE

Files output from LIBSET are in a special relocatable library format and must not be copied with the /B option in PIP. Instead they should be copied by PIP in image (/I) mode.

TABLE F-2

LIBSET OPTIONS

<u>Option Code</u>	<u>Meaning</u>
/S	The /S option means that all input files on a line are to be regarded as containing more than one relocatable binary file. (This is analogous to the /S option in ABSLDR.)

NOTE

If /S is used on a line that contains no input files, input from PTR: is assumed.

F.2.2 Examples of LIBSET Usage

Example 1:

```
*DTA2:SUBS<DTA1:SUB1,SUB2,SUB3,PTR:
↑*SYS:FUNC1,FUNC2.V5$
```

This example creates a relocatable library file on DTA2 named SUBS.RL. This library will contain six FORTRAN (or SABR) subroutines built by combining the relocatable binary file SUB1.RL, SUB2.RL, and SUB3.RL from DTA1 together with one relocatable binary paper tape (note the ↑ printed by PS/8 before loading from PTR:) and the files FUNC1.RL and FUNC2.V5 from the system device.

Example 2:

```
*ASIN,ACOS |  
*/S$↑
```

Since no output file was specified this example creates a relocatable library file LIB8.RL on the system device. This produces a new FORTRAN Library including the subroutines contained in the files ASIN and ACOS on device DSK, and several subroutines combined on a single paper tape loaded from the high-speed reader.

F.2.3 Subroutine Names

It is important to distinguish between the PS/8 file name of a relocatable binary program and its assigned Entry Point Name. The file name has meaning only to the Command Decoder, the Entry Point Name (or Names) are the true subroutine names that are meaningful to the LOADER.

Further details on the format of relocatable binary files and relocatable library files can be found in Appendix A.

F.2.4 Sequence for Loading Subroutines

LIBSET can combine files in any sequence to form a relocatable library file. However, the subroutines in any single library are loaded by the LOADER in the order in which they were origin-

ally specified to LIBSET. Therefore, it is important to make sure that subroutines are specified in order of size, with the largest subroutine being loaded first. If this is not done, cases can occur in which insufficient core is available in any single field to load a large subroutine, whereas space would have been available if the subroutine had been loaded earlier.

F.2.5 LIBSET Error Messages

All errors are fatal. LIBSET recalls the Keyboard Monitor upon encountering any of the following error conditions. LIBSET must be rerun in order to try again.

<u>Error Message</u>	<u>Meaning</u>
BAD FORMAT OR CHECKSUM - TRY AGAIN	Error in reading relocatable binary file.
ERROR WHILE WRITING OUTPUT FILE	Fatal output error occurred.
INPUT ERROR	Parity error on input.
LIBRARY DIRECTORY OVERFLOW	Too many subroutines were specified. Every subroutine name in the input file requires four words, and every relocatable binary file read requires two words. If the total number of words exceeds 250, the library must be split into two separate files.

Additional Information
 Words, 1-4, 1-5, 2-9, D-14
 Alphanumeric options, command
 decoder, 3-3
 ASCII
 character codes, E-4
 files, A-5
 ASSIGN entry, B-4
 asterisk symbol (*), 3-1, 3-10
 Auto-index registers, D-13

 Binary files, A-5
 Blocks
 core control, 1-5
 DEctape, 1-3
 LINctape, 1-3
 logical, 1-3
 physical, 1-3
 standard size, 1-3

 Calling
 command decoder, 3-4
 command decoder special mode, 3-1
 CREF, F-1
 device handlers, 4-1
 USR and device handlers
 from SABR code, D-18
 User Service Routine, 2-1, 2-3
 Card Reader handler modifica-
 tion, D-2
 Card Reader (CDR) operations, 4-7
 Carriage return/line feed sup-
 pression in FORTRAN, D-5
 CDF instructions, 2-2
 CDR, see Card Reader
 CHAIN function, 2-4, 2-13, 2-14
 core area alteration, 2-14
 data passing, 2-14
 monitor error, 2-13
 Character codes and conventions, E-1
 ASCII, E-4
 Characters, lower case, E-1
 Checksum errors, 5-10
 CIF instructions, 2-2
 Circumflex, E-1
 CLOSE function, 2-3, 2-10, 2-11,
 2-12, A-3, D-14
 Code relocation, D-9
 Command decoder subroutine, 1-2,
 3-1, D-2
 calling, 3-4
 command line format, 3-1
 conventions, 3-1
 error messages, 3-3, 3-4
 errors, C-5
 example command line, 3-8
 input file format, 3-2, 3-3
 input files, 3-6
 legal device names, 3-3
 option table, 3-7
 output file format, 3-1, 3-2
 output files, 3-5
 special mode, 3-9, 3-10
 tables, 3-5
 termination, 3-3
 Command line format, 3-1
 Components, PS/8 system, 1-1
 Conditional assembly of CONFIG, 5-1
 CONFIG, D-2
 CONFIG.PA source file, 5-1
 conditional assembly, 5-1
 device handler code, 5-3
 optional device parameters, 5-3
 other options, 5-5
 system device selection, 5-2
 Control characters, E-2, E-3
 Conversion PS/8 DEctapes to PS/12
 LINctapes, D-10
 Co-resident handlers, 5-10
 Core
 area, CHAIN, 2-14
 image (.SV format) files, A-6
 segment doublewords format, A-7
 size PDP-8 computers, D-7
 size, subroutine to determine, D-8
 Core control block, 1-5, -6, -7, A-6
 format, A-6
 Job Status Word, 1-5, 1-6
 starting address, 1-5, 1-6
 CREF, Cross Reference Program, F-1
 calling and loading, F-1
 error detection, F-6
 error messages, F-8
 examples, F-2, F-3
 formats, F-7
 options, F-2, F-3
 pseudo-ops, F-5
 restarting, F-8
 restrictions, F-6
 using, F-1
 CTRL/C, 1-1
 Current directory segment, re-
 writing, D-14

- Data field, 2-2
- Data passing, CHAIN, 2-14
- DATE
 - command, 1-5
 - system word, 1-5, 2-9
 - system word (FORTRAN), D-6
- DEASSIGN command, B-4
- DECØ29 standard card code, D-2
- DECØ26 standard card code, D-2, D-4
- DECODE function, 2-4, 2-12, 2-13, 3-4
 - monitor error, 2-13
 - normal return, 2-13
- Decoder, Command see Command
 - decoder subroutine
- DECTape to LINCTape conversion, D-10
- DECTape
 - operations, 4-7
 - system, 5-3
 - system building, 5-7
- DEVICE pseudo-op, 1-8
- Device control word table, B-6
- Device dependent operations, 4-4
- Device handler
 - adding new, 5-7
 - call format, 4-2
 - call sequence, 5-8, 5-9
 - code, 5-3
 - editing into CONFIG, 5-12
 - errors, 5-9
 - information table format, B-5,-6
 - loading, 2-5, D-11, D-12
 - residency table, B-5
 - standards, 5-9, 5-10
 - writing, 5-8
 - utilization, 1-2
- Device handler usage, 4-1
 - calling device handlers, 4-1, 4-2
 - card reader, 4-7
 - device dependent operations, 4-4
 - file structured devices, 4-7
 - high-speed paper tape punch, 4-5
 - high-speed paper tape
 - reader (PTR), 4-5
 - line printer, 4-6
 - TTY, 4-4
- Device length table, B-7
- Device parameters, optional, 5-3
- Devices, file structured, 1-3
- Device names, B-4
 - assumed, D-2
 - command decoder, 3-3
 - and numbers, 1-7, 1-8
 - selection, 5-13
- DF32 Disk system, 5-2
 - operations, 4-7
- DIRECT option, 5-5
- Direct calling sequence, USR, 2-2
- Directory block
 - example, A-9
 - structure, A-1
- Directory
 - entries, A-2
 - file, A-1
 - fragmentation, A-4
 - sample, A-4
 - segment, rewriting current, D-14
- DSK, Default file storage device, D-2
- Editing device handlers into
 - CONFIG, 5-12
- Empty file, 1-4, A-2
- End-of-file condition, command
 - decoder, 3-7
- ENPUNCH, D-8
- ENTER function, 2-3, 2-8, 2-9, 3-6
 - D-14
 - error return, 2-9, 2-10
 - normal return, 2-9
- ERROR function, 2-4, 2-14
- Error messages
 - command decoder, 3-3, 3-4
 - CREF, F-8
 - LIBSET, F-12
 - Monitor, 2-15
 - summary, C-1
- Error return
 - CLOSE, 2-11, 2-12
 - device handler, 4-3
 - ENTER, 2-9
 - FETCH, 2-6
 - INQUIRE, 2-18
 - LOOKUP, 2-7, 2-8
- Errors
 - checksum, 5-10
 - command decoder, C-5
 - CREF, F-6
 - keyboard monitor, C-4
 - parity, 5-10
 - USR, C-2
- Example
 - command line, 3-8
 - reconfiguration, 5-6
- Exit, system, 1-1
- Extensions, file names and, 1-2

- FETCH command, D-11
- FETCH function, 2-3,-4,-5,-6, 4-1
- File
 - formats, A-5
 - number, A-3, A-4
 - size, A-3, A-4
 - starting block, A-3
- Files, 1-2
 - ASCII, A-5
 - core image (.SV format), A-6
 - data format, 1-2
 - directories, 1-4
 - empty, 1-3, A-2
 - multiple input, D-11
 - names and extensions, 1-2
 - permanent, 1-4, A-2
 - relocatable FORTRAN library, A-8
 - system device (SYS:), 1-3
 - tentative, 1-4, A-3
 - types, 1-3
- FILENAME pseudo-op, 1-8
- File structured devices, 1-3
- File structured devices operation
 - 4-7, 4-8
- File structures, A-1
 - ASCII, A-5
 - binary, A-5
 - directories, A-1
 - formats, A-5
 - relocatable FORTRAN library, A-8
- Formats
 - command decoder option table, 3-7
 - CREF, F-7
 - file data, 1-2
- FORTRAN
 - carriage return/line feed suppression, D-5
 - 8K, D-12
 - library file, relocatable, A-8
 - Library Setup program, F-9
 - program system date, D-6
 - run time I/O routines, 5-4
- Halt
 - LOADER program, C-2
 - system, C-1, C-2
- High speed paper tape punch (PTP)
 - operation, 4-5, 4-6
- High speed paper tape reader
 - (PTR) operation, 4-5
- Indirect calling sequence USR, 2-2
- Information words, additional, 1-4, 1-5
- Input file format, command
 - decoder, 3-2, 3-3
 - format, 3-6
- Input tables, command decoder, 3-7
 - special mode, 3-10
- INQUIRE function, 2-4, 2-17, 2-18
- Instruction field, 2-2
- Job Status Word, 1-6
 - format, A-6
- Keyboard monitor, 1-1
 - calling, 1-1
 - errors, C-4
- LAP (Leave Automatic Paging pseudo-op), D-16
- Layout of resident program, B-2
- Layout of system, B-1
- LIBSET, FORTRAN Library Setup Program, F-9
 - calling, F-9
 - copying LIBSET files, F-10
 - error messages, F-12
 - examples, F-10, F-11
 - loading, F-9
 - loading sequence, F-11
 - options, F-10
 - subroutine names, F-11
 - using, F-9
- LINCSYS, 5-2, 5-3
- LINCTape
 - handlers, 5-4
 - operation, 4-7
 - system building, 5-7
- Line printer (LPT) operation, 4-6
- LOADER program halt, C-2
- Loader control words, A-8, A-9
- Loading
 - CREF, F-1
 - device handlers, D-11
 - device handlers dynamically, D-12
- Logical end-of-file, device
 - handler, 4-3
- LOOKUP function, 2-3, 2-7, D-14
- Lower case characters, E-1
- Low-speed paper tape, 5-4
- LP12 line printer, type 645, 5-4

Masking, 1-2, 4-1
Maximum number of files, A-4
Maximum size of file, A-4
Monitor error,
 CHAIN, 2-13
 CLOSE, 2-13
 ERROR, 2-15
Multiple input files, D-11

Names
 device, 1-7, 1-8, B-4
 file, 1-8
Normal return,
 CLOSE, 2-11
 ENTER, 2-9
 FETCH, 2-6
 INQUIRE, 2-17, 2-18
 LOOKUP, 2-7
 RESET, 2-19
Null files, 3-3
Number and size of file, A-3
Numbers, device, 1-7, 1-8
Number symbol (#), F-5
Numeric options, command
 decoder, 3-3

ODT breakpoint, CHAIN, 2-14
ODT breakpoint usage, D-13
Operating command decoder in
 special mode, 3-10
Optional device parameters
 CONFIG, 5-3
Options, command line
 alphanumeric, 3-3
 numeric, 3-3
Option tables, command decoder, 3-7
 special mode, 3-10
Output files, command decoder, 3-5
 format, 3-1, 3-2
Output table, command decoder
 in special mode, 3-10

PAGE pseudo-op, D-16
PAL8, CREF pseudo-ops, F-5
Parity
 codes, E-1
 error, device handler, 4-3
 errors, 5-10
PDP-12 LINCtape system, 5-2
Permanent device name table, B-3
Permanent file, 1-4, 2-10, A-2
Permanent file entry, 2-7

Program starting address, 1-6
 format, 1-6
 START command, 1-6
PRTCl2-F program, D-10
Pseudo-ops, CREF
 PAL8, F-5
 SABR, F-5
PTP see High-speed paper tape
 punch
PTR see High-speed paper tape
 reader
PUNCH feature, D-9

R command, A-7
Record, definition of, 4-1
Record transfer, 4-1
Reconfiguration, 5-1
 adding new device handlers, 5-7
 building system on DECTape or
 LINCtape, 5-7
 editing device handlers into
 CONFIG, 5-12
 example, 5-6
 writing device handlers, 5-8
Relocating code, D-9
RESET function, 2-4, 2-18, 2-19
 deleting tentative files, 2-19
 normal return, 2-19
Resident program layout, B-2
Restarting CREF, F-8
Restrictions
 command decoder in special
 mode, 3-10
 device handlers, 4-3
 USR calls, 2-2
Rewriting current directory seg-
 ment, D-14
RF08 disk system, 5-2
 operations, 4-7
RK8 disk system, 5-2
 operations, 4-7
RUN command, A-7

/S (compress device) option, B-8
SABR
 code, D-16
 code, calling USR and device
 handlers, D-18
 CREF pseudo-ops, F-5
 programming notes, D-16
Sample directory, A-4
SAVE command, 1-5, 1-6
Segment header, A-1

Semicolons, CREF, F-6, F-7
 Special mode of the command de-
 coder, 3-9, 3-10
 START command, 1-6, 1-7
 Storage space USR area, D-13
 .SV (core image) file, 1-5, A-6
 Subroutine to determine core
 size, D-8
 Summary USR functions, 2-3
 CHAIN, 2-13
 CLOSE, 2-10
 DECODE, 2-12
 ENTER, 2-8
 ERROR, 2-14
 FETCH, 2-4
 INQUIRE, 2-17
 LOOKUP, 2-7
 RESET, 2-18
 USRIN, 2-15
 USROUT, 2-16
 Syntax error, command line, 3-3
 SYS, D-2
 System
 building on DECTape or LINC-
 tape, 5-7
 DATE, 1-5, 2-9
 Date word, 1-5
 Date word, FORTRAN, D-6
 device selection, CONFIG, 5-2
 device (SYS:), 1-3
 device tables, B-3
 exit, 1-1
 layout, B-1
 scratch blocks, B-1, B-2
 software components, 1-1

 Tables
 permanent device name, B-3, B-4
 system device, B-3
 user device name, B-4, B-5
 Table lookup example, 5-11
 Teletype (TTY) operation, 4-4, 4-5
 Tentative file, 1-4, 2-10, A-3
 deletion, 2-19
 entry, 2-8
 Termination, command decoder, 3-3
 Terminology, 1-1
 TTY see Teletype

 Up arrow character (↑), 5-5
 User device name tables, B-4, B-5
 USR (User Service Routine) 1-2
 area storage space, D-13
 calling, 2-1
 direct calling, 2-2
 ERROR, C-3
 function summary, 2-3
 indirect calling, 2-2
 restrictions, 2-2
 USRIN function, 2-3, 2-4, 2-15, 2-16
 USROUT function, 2-4, 2-14, 2-16

 Write lock error, device handler,
 4-3

 /X option, CREF (PAL8), F-6

Digital Equipment Corporation
Maynard, Massachusetts

digital