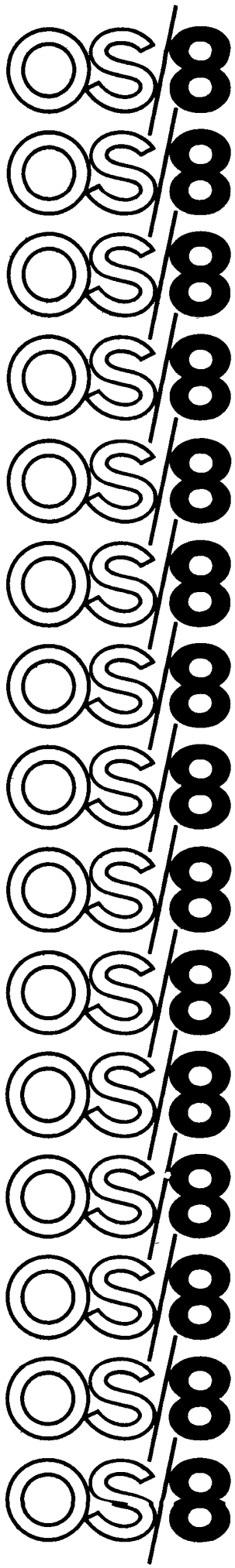
digital



OS/8 software support manual

digital equipment corporation

OS/8 SOFTWARE SUPPORT MANUAL

(Version 3)

DEC-S8-OSSMB-A-D

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

digital equipment corporation • maynard, massachusetts

First Printing, January 1973
Revised, June 1974

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

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

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

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

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

The following are trademarks of Digital Equipment Corporation:

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

PREFACE

The 8K Operating System (OS/8) is an extremely powerful program development system. OS/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 OS/8, is described in detail in the OS/8 HANDBOOK (DEC-S8-OSHBA-A-D).

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 are 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 OS/8 HANDBOOK, as well as this manual, can be found in the Appendices. Appendix A details the OS/8 directory structure and gives standard file format. 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 OS/8 system. Appendix E is a complete list of the standard ASCII character codes meaningful to OS/8. Finally, Appendix F describes a set of generalized I/O routines for use under the OS/8 system.

CONTENTS

CHAPTER 1

OS/8 CONCEPTS AND TERMINOLOGY

Before examining the details of the OS/8 system, the reader should first be familiar with the simpler techniques and terms used within the framework of the OS/8 system. The material in this chapter, along with that contained in the OS/8 HANDBOOK, provides the tools needed to pursue the later chapters.

1.1 SOFTWARE COMPONENTS OF OS/8

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

1. The Keyboard Monitor performs commands specified by the user at the keyboard console. The nine Keyboard Monitor commands (ASSIGN, DEASSIGN, GET, SAVE, ODT, RUN, R, START, and DATE) are explained in Chapter 1 of the OS/8 HANDBOOK.

   User programs can exit to the Keyboard Monitor by executing a JMP to location 7600 in field 0. All JMPs to 7600 must be made with the DATA FIELD set to zero. This saves the contents of locations 0000 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 terminal 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.

2. 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.

3. 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.

4. 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 1 of the OS/8 HANDBOOK. 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.

5. Two other components, ABSLDR and ODT, are not logically part of the OS/8 system. However, in the sources and listings, ABSLDR is combined with the Keyboard Monitor and USR. ODT is combined with the command decoder.

1.2 FILES

Files are basic units of the OS/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, OS/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 terminal is not.

Cassettes and magnetic tapes form an intermediate case. They may be treated directly as non-file structured devices, or the program MCPIP may appear to be file structured.

The system device (SYS) in any OS/8 system is always file structured, as is the default storage device, DSK.

All OS/8 file structured devices must be logically divided into these 256-word blocks. Hence, 256 words is considered the standard OS/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 OS/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 OS/8. Similarly, LINCtapes are formatted with 129 (or 128) words per block but never 256, as this format is unacceptable to OS/8.

A given OS/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 OS/8 system:

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

2. 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.

3. 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 establishes 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.

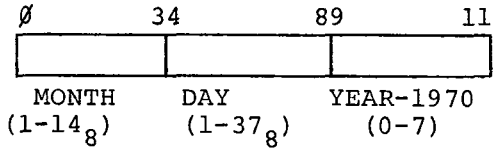
1.2.4 File Directories and Additional Information Words

To maintain records of the files on a device, OS/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 1 of the OS/8 HANDBOOK.

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. OS/8 automatically uses one extra word per entry for the date. This value is set by executing a DATE command

(see Chapter 1 of the OS/8 HANDBOOK) which codes the current date into memory location 07666 in the following format:



A date word of 0 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 ABSLDR or other means 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 that specifying arguments to the SAVE command as described in Chapter 1 of the OS/8 HANDBOOK, can alter the contents of the target program's Core Control Block.

When a program is loaded, the starting address and Job Status Word are read from the Core Control Block and saved in core. The Core Control Block itself is saved in the last 200 (octal) words of block 37 on the system device 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 07744 and 07745. The format of these words is:

LOCATION	CONTENTS	NOTES
07744	62n3	n is the field in which to start.
07745	nnnn	Starting address of the program.

1.3.2 Job Status Word

The Job Status Word contains certain flags that affect OS/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 07746 and contains the following information:

Bit Condition	Meaning
Bit 0 = 1	File does not load into locations 00000 to 01777.
Bit 1 = 1	File does not load into locations 00000 to 01777.
Bit 2 = 1	Program must be reloaded before it can be restarted.
Bit 3 = 1	Program does not destroy the contents of the highest existing memory field, an optimization for the Batch system.
Bits 4 thru 9	Reserved for future use.
Bit 10 = 1	Locations 00000 to 01777 need not be saved when calling the Command Decoder.
Bit 11 = 1	Locations 10000 to 11777 need not be saved when calling the USR.

When bit 2 of the Job Status Word 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 I (7746      /LOAD JOB STATUS WORD
AND (6777
TAD (1000
DCA I (7746      /JOB IS NOT RESTARTABLE
```

The Job Status Word can be updated from the user's program or with the ABSLDR /P option, thus providing optimization of tape (disk) motion. More information on the Core Control Block can be found in the description of Core Image (SV) files found in Appendix A.

Bit 3 of the JSW (Job Status Word) is used as an optimization for the Batch operating system. If a program can never cause the highest existing memory field to be altered, this bit should be set. For example, EDIT, PIP, FORT, and SABR can never use memory above 8K. Thus, they should set bit 3 of the JSW. Programs such as ABSLDR,

LOADER, PAL8 and CREF can alter all of core. They should perhaps not have bit 3 on. Note that the more core that exists, the more unlikely it is that a program will destroy upper core. Thus, on 28K systems, only the largest FORTRAN programs can alter field 6 and, in general, bit 3 should be set.

1.3.3 Software Core Size

Location 07777 contains the software core size in bits 6-8. This represents the highest memory field available for use by OS/8. If bits 6-8 contain 0, all of the available memory is used. Most OS/8 cusps interrogate this word to determine how much memory is available. The other bits of this location are reserved for use by BATCH and should not be touched by user programs.

1.4 DEVICE NAMES AND DEVICE NUMBERS

The OS/8 system can accommodate up to 15 separate devices. In Chapter 1 of the OS/8 HANDBOOK 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 17 (octal) 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.

Accessing devices by number should be done only when the appropriate number has been obtained from a USR INQUIRE CALL. Except for SYS and DSK, the OS/8 peripherals do not have fixed numbers; instead, device numbers vary whenever BUILD is used to modify a system. Thus, it is suggested that reference by name be used whenever possible.

To determine whether a device name is recognized in the system, attempt to ASSIGN that device. For example, to determine whether LINCtape handlers are called LTA or DTA, perform:

```
.DEASSIGN
.AS LTA0
```

If the system responds with a dot (.), LTA0 does indeed exist. If the system responds with:

```
LTA0 NOT AVAILABLE
```

no device named LTA0 is present.

1.5 THE DEVICE AND FILENAME PSEUDO-OPS

Several of the USR 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:

```
0424
0161
```

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 subroutines which perform the operations of opening and closing files, loading device handlers, program chaining, and calling the Command Decoder. The USR provides these functions not only for the system itself, but for all programs running under the OS/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 10 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	The operation was successful. The AC is cleared and the data field is set to current field.

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.

The value of the data field preceding the JMS to the USR is exceedingly important. The data field MUST be set to the current field, and the instruction field MUST be set to 1. Note that a CDF is not explicitly required if the data field is already correct. When a doubt exists as to the data field setting, an explicit CDF should be executed.

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

1. The USR can never be called from any address between 10000 and 11777. Attempting to do so results in the:

   MONITOR ERROR 4 AT xxxxx (ILLEGAL USR CALL)

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

2. Several USR calls take address pointers as arguments. These pointers always refer to data in the same memory field as the call.

3. 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 In this case, 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 the most 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, if necessary (see section 2.2.9).

2.2 Summary of USR Functions

Function Code	Name	Operation
1	FETCH	Loads a device handler into core. Returns the 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	Closes the currently open tentative file on the specified device and becomes a permanent file. Also, any previous permanent file with the same file name and extension is deleted.
5	DECODE	Calls the Command Decoder. 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	Loads the USR into core. Subsequent calls to the USR are by an effective JMS to location 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.

An attempt to call the USR with a code greater than 13 (octal) will currently cause a Monitor Error 4 message to be printed and the program to be aborted.

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:

1. Load a device handler corresponding to a given device name.

2. Load a device handler corresponding to a given device number.

First, the following is an example of loading a handler by name from memory field 0:

```
CLA             /AC MUST BE CLEAR
CDF 0           /DF = CURRENT FIELD
CIF 10          /IF = 1
JMS I (USR
1               /FUNCTION CODE = 1
DEVICE DTA3     /GENERATES TWO WORDS: ARG(1)
                /AND ARG(2)
6001            /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 1-page handler.

Bit 11 is 1 if there is room for a 2-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           /DF = CURRENT FIELD
CIF 10          IF = 1
JMS I (USR
1               /FUNCTION CODE = 1
6001            /ARG(1)
JMP ERR         /ERROR RETURN
.               /NORMAL RETURN
.
.
```

On entry to the USR, the AC contains the device number in bits 8 to 11 (bit 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:

1. There is no device corresponding to the given device name or device number, or

2. 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:

Error Message	Meaning
MONITOR ERROR 4 AT xxxxx (ILLEGAL USR CALL)	Results if bits 8 to 11 of the AC are zero (and bits 0 to 7 are non-zero).
MONITOR ERROR 5 AT xxxxx (I/O ERROR ON SYS)	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:

1. Device handlers are always loaded into memory field 0.

2. The entry point that is returned may not be on the page desired. This would happen if the handler were already resident.

3. Never attempt to load a handler into the 7600 page or into page 0. Never load a two page handler into the 7400 page.

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

NOTE

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.

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 0       /DF=CURRENT FIELD
        CIF 10      /IF = 1
        JMS I (USR
        2           /FUNCTION CODE = 2
        NAME        /ARG(1), POINTS TO FILE NAME
        0           /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 bit 8 to 11. ARG(1) contains a pointer to the file name. Note that the file name block must be in the same memory 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 papertape reader), then ARG(1) and ARG(2) contain 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:

1. The device specified is a write-only device.

2. 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:

Error Message	Meaning
MONITOR ERROR 2 AT xxxxx (DIRECTORY I/O ERROR)	Results if an I/O error occurred while reading the device directory.
MONITOR ERROR 3 AT xxxxx (DEVICE HANDLER NOT IN CORE)	Results if the device handler for the specified device is not in core.
MONITOR ERROR 4 AT xxxxx (ILLEGAL USR CALL)	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 0           /DF = CURRENT FIELD
        CIF 10          /IF = 1
```

```
        JMS I (USR
        3                   /FUNCTION CODE = 3
        NAME                /ARG(1) POINTS TO FILE NAME
        0                   /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:

1. The device specified by bits 8 to 11 of the AC is a read only device.
2. No empty file exists which satisfies the requested length requirement.
3. Another tentative file is already active on this device (only one output file can be active at any given time).
4. 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:

Error Message	Meaning
MONITOR ERROR 2 AT xxxxx (DIRECTORY I/O ERROR)	Result if an I/O error occurred while reading or writing the device directory.
MONITOR ERROR 3 AT xxxxx (DEVICE HANDLER NOT IN CORE)	Results if the device handler for the specified device is not in core.

Error Message	Meaning
MONITOR ERROR 4 AT xxxxx (ILLEGAL USR CALL)	Results if AC bits 8 to 11 are zero.
MONITOR ERROR 5 AT xxxxx (I/O ERROR ON SYS)	Read error on the system device while bringing in the overlay code for the ENTER function.
MONITOR ERROR 6 AT xxxxx (DIRECTORY OVERFLOW)	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 0               /DF=CURRENT FIELD
          CIF 10              /IF=1
          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:

1. FETCH the device handler for the output device.
2. ENTER the tentative file on the output device, getting the starting block and the maximum number of blocks available for the file.
3. 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.
4. CLOSE the tentative file, making it permanent. The CLOSE operation would always use the same file name as the ENTER performed in step 2. The closing file length would have been computed in step 3.

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:

1. The device specified by bits 8 to 11 of the AC is a read only device.

2. 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:

Error Message	Meaning
MONITOR ERROR 1 AT xxxxx (CLOSE,ERROR)	Results if the length specified by ARG(2) exceeded the allotted space.
MONITOR ERROR 2 AT xxxxx (DIRECTORY I/O ERROR)	Results if an I/O error occurred while reading or writing the device directory.
MONITOR ERROR 3 AT xxxxx (DEVICE HANDLER NOT IN CORE)	Results if the device handler for the specified device is not in core.
MONITOR ERROR 4 AT xxxxx (ILLEGAL USR CALL)	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 shoud be read before attempting to use the DECODE function.

A typical call to the Command Decoder looks as follows:

```
        CDF 0               /DF=CURRENT FIELD
        CIF 10              /IF=1
        JMS I (USR
        5                   /FUNCTION CODE = 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, in the preceding example it is ".PA". On return from the Command Decoder, information is stored in tables located in the top page of memory 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 0 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:

Error Message	Meaning
MONITOR ERROR 5 AT xxxxx (I/O ERROR ON SYS)	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 0                   /DF=CURRENT FIELD
CIF 10                  /IF=1
JMS I (USR
6                       /FUNCTION CODE = 6
BLOCK                   /ARG(1), TARTING BLOCK NUMBER
```

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

Error Message	Meaning
MONITOR ERROR 5 AT xxxxx (I/O ERROR ON SYS)	I/O error occurred while reading or writing on the system device.
CHAIN ERR	If an attempt is made to CHAIN to a file which is not a core image (.SV) file. Control returns to the keyboard monitor.

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 other 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 memory 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 0              /DF = CURRENT FIELD
CIF 10             /IF = 1
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 11 (octal), 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 00500

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:

Error Message	Meaning
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                   /DF = CURRENT FIELD
CIF 10                  /IF = 1
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, provided the calling program loads into this area as indicated by the current JSW, and loads the USR, then 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 thru 11777.

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                   /DF = CURRENT FIELD
CIF 10                  /IF = 1
JMS I (200              /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:

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

2. 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 0               /DF = CURRENT FIELD
CIF 10              /IF = 1
JMS I (USR
12                  /FUNCTION CODE = 12
DEVICE DTA3         /GENERATES TWO WORDS:
                    /ARG(1) AND ARG(2)
0                   /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               /DF = CURRENT FIELD
CIF 10              /IF = 1
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.

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 0                   /DF = CURRENT FIELD
CIF 10                  /IF = 1
JMS I (USR
13                      /FUNCTION CODE = 13
0                       /0 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 0                   /DF:CURRENT FIELD
CIF 10                  /IF = 1
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. The Keyboard Monitor currently does not reset the Monitor tables. If user programs which do not call the Command Decoder are used, it is wise to do a RESET operation before loading device handlers. The RESET will ensure that the proper handler will be loaded into core.

CHAPTER 3

THE COMMAND DECODER

OS/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 an * 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 OS/8 programs. Also, since command lines all have a standard basic structure, the Command Decoder makes learning to use OS/8 much easier.

3.1 COMMAND DECODER CONVENTIONS

Chapter 1 of the OS/8 HANDBOOK 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.

Output File Format	Meaning
EXPLE.EX	Output to a file named EXPLE.EX on device DSK (the default file storage device).
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 Format	Meaning
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: 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.

NOTE

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 the LOOKUP fails, a second LOOKUP is made for the file appending a null (zero) extension.

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 valid.

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 an *, and waits for another command line. The erroneous command is ignored.

Error Message	Meaning
ILLEGAL SYNTAX	The command line is formatted incorrectly.
TOO MANY FILES	More than three output files or nine input files were specified. (Or in special mode, more than 1 output file or more than 5 input files.)
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 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 0000 to 1777 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.

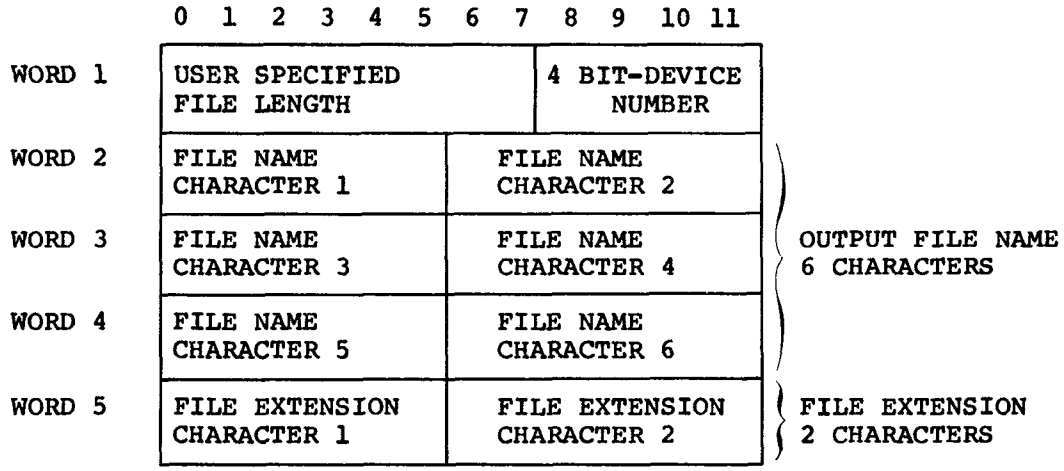
As the Command Decoder normally handles all of its own errors, there is no error return from the DECODE operation.

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 17600. Each entry is five words long and has the following format:



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.

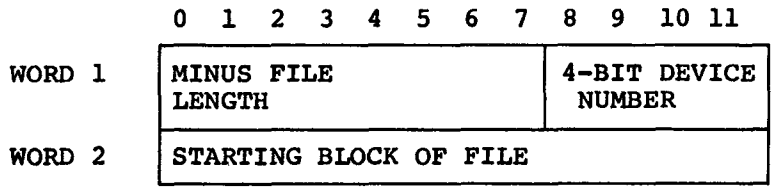
The entry for the first output file is in locations 17600 to 17604, the second is in locations 17605 to 17611, and the third is in locations 17612 to 17616. If word 1 of any entry is zero, the corresponding output file was not specified. A zero in word 2 means that no file name was specified.

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:



Bits 0 to 7 of word 1 contain the file length as a negative number. Thus, 377 (octal) in these bits is a length of one block, 376 (octal) 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 file list entries beginning at odd numbered locations from 17617 to 17637 inclusive. 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 zero to provide a terminator even when no 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 F = 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	0	1	2	3	4	5	6	7	8	9
17646	LOW ORDER 12 BITS OF = > OPTIONS											

Each of these bits corresponds to one of the possible alphanumeric option switches. The corresponding bit is 1 if the switch was specified, 0 otherwise.

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.

Bit 0 of location 17642 is 0 if the command line was terminated by a carriage return, 1 if it was terminated by an ALT MODE.

3.4.4 Example

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

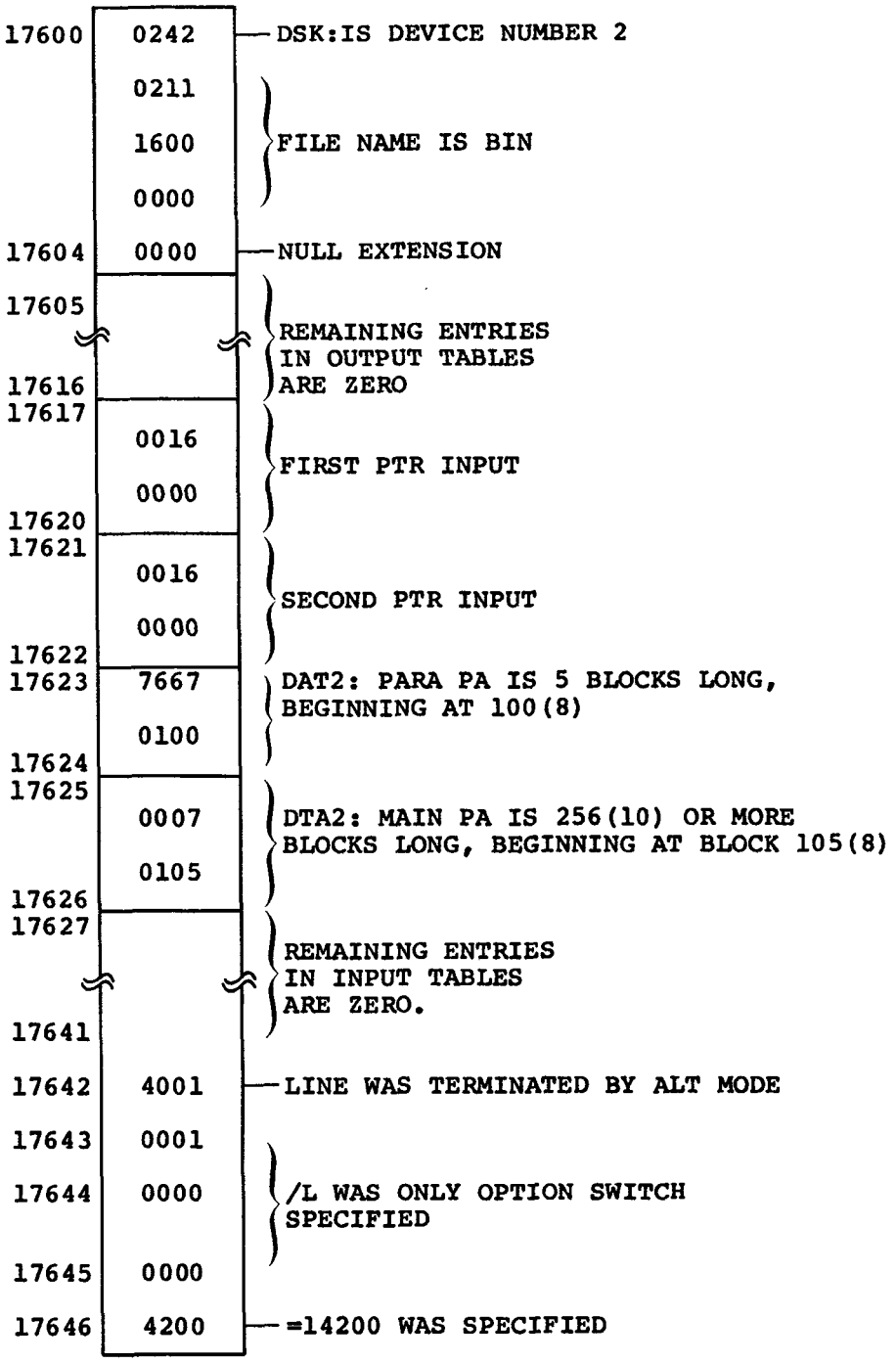
```
*BIN[10]<PTR:,,DTA2:PARA,MAIN /L=14200$
```

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 (or just PARA) on DTA2, and one file named MAIN.PA (or just MAIN) 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 ALTMODE 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.



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. Programs such as format conversion routines which access non-standard file structures could use this special format. If the input files were not OS/8 format, a command decoder LOOKUP operation would fail. The capability to handle this case is provided

in the OS/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

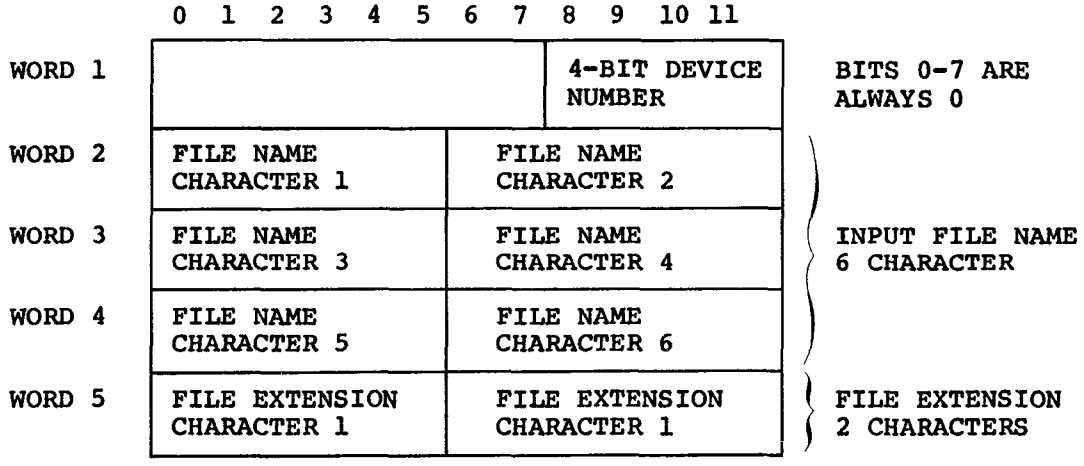
The special mode call to the Command Decoder is identical to the standard DECODE call except that the assumed input file extension, specified by ARG(1), is equal to 5200. The value 5200 corresponds to an assumed extension of ".*", which is illegal. Therefore, the special mode of the Command Decoder in 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:

1. Only one output file can be specified.
2. No more than five input files can be specified, rather than the nine acceptable in normal mode.
3. The characters asterisk (*) and question mark (?) are legal in file names and extensions, both in input files and on output files. It is strongly suggested that these characters be tested by the user program and treated either as special options or as illegal file names. The user program must be careful not to ENTER an output file with an asterisk or question mark in its name as such a file cannot easily 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:



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.

The OS/8 batch generating system will allow calls to the command decoder in special mode.

3.6 CCL AND THE COMMAND DECODER

CCL uses its own copy of the Command Decoder instead of the copy available from the monitor. Thus, the CCL Command Decoder has several options not available via standard USR calls to the OS/8 Command Decoder, e.g., multiple default extensions.

3.7 USEFUL LOCATIONS IN BATCH

BATCH will run whenever bit 0 of location 07777 is a 1. The user may wish to access the following useful locations in BATCH. The locations are in the highest memory field available to OS/8:

BATERR = 7000	JMP here to abort BATCH.
BATOUT = 7400	JMS here to print character in AC in BATCH log.
BATSPL = 7200	JMS here to permit spooling with default extension in AC.

3.8 CCL TABLES

A description of all tables used by CCL is included in the file CCL.PA supplied to all users of OS/8 version 3.

CHAPTER 4

USING DEVICE HANDLERS

A device handler is a system subroutine that is used by all parts of the OS/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 effectively "device independent". Changing devices involves merely changing the device handlers used for I/O.

OS/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 bks 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 OS/8 programs, such as PIP, that have large buffer areas.

NOTE

> The word "record" is defined to mean 128 words of data; thus, an OS/8 block consists of two 128 word records.

4.1 CALLING DEVICE HANDLERS

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

As with calls to the USR, it is important that the data field is set to the current program field before teldevice 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:

Bits	Contents
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. If bits 1-5 are zero and the device is non-file structured (i.e., TTY, LPT, etc.) the operation is device dependent. If the device is file structured (SYS, DECtape, disk, etc.), a read/write of 40 (octal) pages is performed.
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 DECtape 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 (except TM8E and TA8E).

NOTE

Starting forward saves time as long as the block number, ARG(3), is about 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 an 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 an 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; furthermore, no device handler takes the non-fatal error return when doing output.

The following restrictions apply to the use of device handlers:

1. If bits 1 to 5 of the function control word, ARG(1), are zero, a transfer of 40 (octal) pages or an entire memory field is indicated. Care must be used to ensure that the handler is not overlaid in this call. This only applies to file-structured handlers.

2. The user program must never specify an input into locations 07600 to 07777, 17600 to 17777, or 27600-27777, or the page(s) in which the device handler itself resides. In general, 7600-7777 in every memory field are reserved for use by system software. Those areas should be used with caution.

3. Note that the amount of data transferred is given as a number of 128 word records, exactly one half of an OS/8 block. Attempting to output an odd number of records can change the contents of the last 128 words of the last block written. For example, outputting 128 words to a block on the RK8 disk causes the last 128 words of the block to be filled with zeroes.

4. 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" memory. For example, a write of 2 pages starting at location 07600 would cause locations 07600-07777 and 00000-00177 of field 0 to be written.

5. If bits 1-5 of the function control word ARG(1) are zero, a device-dependent operation occurs. Users should not expect a 40-page (full field) transfer of data. The CLOSE operation of the USR calls the handler with bits 1-5 and 9-11 of the function control word 0. This condition means 'perform any special close operations desired'. Non-file structured handlers which need no special handling on the conclusion of data transfers should treat this case as a NOP. Examples of usage of such special codes:

   LPT - perform a form feed
   CSAn, MTAn - write two file marks

4.2 DEVICE DEPENDENT OPERATIONS

This section describes briefly the operation of certain standard OS/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 1-Page Terminal (TTY) (AS33)

1. Normal Operation

   This handler inputs characters from the terminal keyboard and packs them into the buffer or unpacks characters from the buffer and outputs them to the console terminal.

   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 terminal.

2. Initialization for Block 0

   None.

3. Terminating Conditions

   On input, detection of a CTRL/Z causes a CTRL/Z (octal code 232) to be placed in the input buffer, the remaining words of the buffer to be 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 1-page terminal handler.

4. Terminal Interaction

   CTRL/C forces a return to the Keyboard Monitor. CTRL/ forces an end-of-file on input (see 3). CTRL/O terminates printing of the contents of the current buffer on output.

4.2.2 High-Speed Paper Tape Reader (PTR)

1. Normal Operation

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

2. Initialization for Block 0

   The handler prints an up-arrow (↑) on the terminal 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.

NOTE

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

3. 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 output to the paper tape reader causes a fatal error to be returned.

4. Terminal Interaction

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

4.2.3 High-Speed Paper Tape Punch (PTP)

1. Normal Operation

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

2. Initialization for Block 0

None.

3. 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.

4. Terminal Interaction

Typing CTRL/C forces a return to the Keyboard Monitor, but only when actual punching has begun, or if ↑C is typed before punching commences. If the punch is off line, ↑C is only effective immediately before punching would begin.

4.2.4 Line Printer (LPT) (LPSV)

1. 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 a skip to the next paper position for vertical tabulation if the line printer hardware provides that feature. 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 or 128 columns have been printed. This handler

functions properly only on ASCII data. The handler for the LS8E line printer handler utilizes the expanded character capability of the printer. If a 216 (CTRL/N) character appears anywhere in a line of text, the entire line is printed in the expanded character mode. The 216 must be used on a line-by-line basis.

2. Initialization for Block 0

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

3. Terminating Condition

   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. A fatal error is also returned if the line printer error flag is set. There are no non-fatal errors associated with the line printer handler.

4. Terminal Interaction

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

5. Patching the LPSV Handler

   The following patches are available for the LPSV line printer handler.

rel. loc 0: set to -printer width-1	(i.e. set to -121(octal) for an 80 column line printer)
rel. loc 1: set to 4	for the LV8E line printers
set to 14	for the LPOE and LS8E printers
rel. loc 2: set to -40	to convert lower case to upper case
set to 0	if your printer can print lower case

4.2.5 Cassettes

1. Normal Operation

   This handler performs character I/O between the cassettes and the buffer. It treats cassettes as a non-file structured device. Data appears on cassette in 192-byte records.

2. Initialization for Block 0

   On input the cassette is rewound. On output the cassette is rewound and a file gap is written.

3. Terminating Condition

An end-of-file on input is a software error.

4. Terminal Interaction

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

5. Special Codes (device dependent features)

If the handler is called with bits 1-5 of the function word =0, then bits 10-11 are examined. The meaning of these codes are as follows:

0 write a file gap
1 rewind also, then write a file gap if bit 0=1
2 space backwards one record
3 skip one file (direction depends on bit 0)

NOTE

The handler neither reads nor writes standard files. It is merely a paper tape replacement. It writes raw data (organized into 192-byte records) onto the cassettes starting at the beginning; and then later reads it back.

The source is already in OS/8 BUILD format. The handler has only two entry points (for drives A and B of a controller). The decision as to which controller it uses is made at assembly time by changing the symbol code. The result is as follows:

CODE	DEVICE NAME	HANDLER	DEVICE CODE
0	TA8A	A:CSA0 B:CSA1	70
1	TA8B	A:CSA2 B:CSA3	71
2	TA8C	A:CSA4 B:CSA5	72
3	TA8D	A:CSA6 B:CSA7	73

The handler has the internal device code of 27 (see Table 2-12 in Chapter 2 of the OS/8 HANDBOOK. The handler is two pages long.

4.2.6 Card Reader (CDR)

1. 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 DEC029 standard card codes are used).

2. Initialization for Block 0

None.

3. 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.

4. Terminal Interaction

Typing CTRL/C forces a return to the Keyboard Monitor. Typing CTRL/Z forces an end-of-file to occur (see 3.).

4.2.7 TM8-E Handler

1. Normal Operation:

When the handler is used in its normal mode, single-file mode, magtapes may consist of exactly one file. It starts at the beginning of the tape and consists of consecutive records until an end-of-file mark (EOF) is reached. In this sense, a magtape is similar to one big paper tape. This is the same way that OS/8 currently treats cassettes.

Since the capacity of magtapes is so big, provisions have been made for storing multiple files per tape. In such a structure, several files may exist on one magtape. They are unlabeled and are separated from each other by a single file mark. The last one is followed by two file marks. Each 'file' looks like a paper tape. It is referenced in a non-file structured manner. The magtape handler must be altered first to work in file mode. Then the magtape must be positioned to exactly the correct spot where the read or write operation will commence. This may be done with any program using the auxiliary capabilities of the magtape handler (described below), or the positioner program, CAMP. To read a file, the handler must be positioned to just before the first data record of that file. To write file #1, rewind the tape (i.e., be at BOT). To write file #n, (n>1) the handler should be positioned after the (n-1)st file mark on the tape. Previous file n and all files past it then become unreadable (non-existent).

A. Device-Dependent Capabilities:

The TM8-E handler has several auxiliary features which may be invoked by a user program which calls the handler in a device-dependent manner. These features all rely on the contents of bits 9-11 of the function word (argument 1 of the handler call) and some require argument 3 in addition.

These features are brought to life whenever the handler is called with a page count of 0 (bits 1-5 of the function word). Call bits 9-11 of the function word, the Special Function Register (SFR) for short, and also refer to bit 0 of the function word as the direction bit. If the page count is not 0, the contents of the SFR is ignored.

If the page count is 0, then the SFR together with the direction bit (and possibly argument 3) determine what special function to perform, as follows:

SFR	OPERATION
0.	CLOSE. Write two EOF's.
1.	Rewind.
2.	Space forward/reverse records. The direction to space is determined by the direction bit (0 means space forward, 1 means space reverse). The negative (two's complement) of the number of records to space over is given by argument 3 of the handler call. (-1 means space past one record, 0 means 4096 records.) The error return is taken if either a file mark or BOT is encountered. In such cases, you would be left positioned at the beginning of a file.
3.	Space forward/reverse files. The direction to space is determined by the direction bit (0 means space forward, 1 means space reverse). The negative of the number of file marks to space past is given by argument 3 (-1 means space past one file mark; 0 means 4096 file marks). In reverse mode, the tape is left positioned at the end of a file; an error is given if BOT is encountered. In forward mode, the tape is left positioned at the beginning of a file. If EOD is reached, the handler automatically performs a backspace record to leave you between the two file marks; no error is given.
4.	Rewind the unit and put drive off-line.
5.	Write a single EOF.
6.	Special read/write function. The direction bit (as usual) determines read or write (0 means read, 1 means write). The specified I/O operation is performed between the user's buffer (start is specified by argument 2) and the very next magtape record. Only one record is transferred and the user's buffer must be large enough to contain it. The record length is specified by the negative of argument 3 (in words). 0 means a record length of 4096.
7.	Unused. Reserved for future use. If specified, it currently acts as a NO-OP.

In each case, the unit affected is determined by the handler entry point.

B. Other Common Operations:

(a) To backspace n files, use special code 3 to pass over n+1 file marks backwards, then use special code 2 to advance (forward) over one record (EOF) ignoring the EOF error.

(b) To advance to EOD, first perform a backspace of one record (or perform a rewind to play safe) then use special code 3 to advance over 4096 files in the forward direction (argument 3=0).

2. Special Handling for Block 0

If the handler is called to read or write block 0, it will first perform a rewind. This feature can be patched out if desired by altering relative location 1 from a 0 to a 1. This altered handler should be operating in file mode. The original handler should be operating in single-file mode.

A. Special Handling for CLOSE:

A close operation is signaled to the handler by calling it with a function word which has a page count of 0 (bits 1-5) and which has bits 9-11 all zeroes. This is how the USR CLOSE operation calls the handler (OS/8 V3 only. This causes the handler to write two successive file marks on the tape. Two successive EOF's is the software indication of end-of-data (EOD).

B. Restrictions:

In single-file mode, should not have more than 4095 blocks because on trying to write the 4096th block, the handler will think it's writing block 0 and perform a rewind. This restriction does not apply when using the handler in file-mode; but beware, some cusps, such as PIP, are suspected to behave strangely on block 4096 of non-file-structured devices.

3. Terminating Conditions

None.

4. Keyboard Interaction:

Typing ↑C on the keyboard while the handler is in operation causes the handler to abort and return to the OS/8 keyboard monitor via location 7600. Such action is ill-advised since it leaves the magtape without an end-of-file indicator.

5. Error Conditions:

On a hard error, the handler takes the error return (with a negative AC) and the AC contains the contents of the main status register, as follows:

Bit on	Meaning
0	Error flag
1	Tape rewinding
2	BOT
3	Select error
4	Parity error (vertical, longitudinal, or CRC)
5	EOF
6	Record length incorrect
7	Data request late
8	EOT
9	File protect
10	Read compare error
11	Illegal function

4.2.8 File-Structured Devices

1. Normal Operation

   (DECtape, LINCtape, TD8E DECtape, DF32, RF08, and RK8, RK8E)

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

2. Initialization for Block 0

   None.

3. Terminating Conditions

   A fatal error is returned whenever the transfer causes 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.

4. Terminal Interaction

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

   NOTE

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

4.2.9 TD8E DECtape

TD8E DECtape is the new accumulator transfer DECtape. Since OS/8 is a noninterrupt driven system, TD8E DECtape has data transfer rates equivalent to those for TC08 DECtape; however, the interrupt should never be used with the TD8E. Device handlers for TD8 DECtape are supplied as a standard part of OS/8. Each pair of drives (0, and 1, 2 and 3, etc.) requires a 2-page device handler. Thus, to have all

eight TD8E drives in the system at one time will require four separate handlers. Thus for TD8E, it is wise to restrict usage to those units that physically exist. Also, the tape drives are hardwired to select one of two possible unit numbers; thus, the first pair of drives installed must be called units 0 to 1. Any others numbers will cause a SELECT error. In this case, the computer hangs until the correct drive is selected.

4.2.10 KL8E Terminal Handler

Listed are the features of the KL8E handler. Those that are conditional are marked by an asterisk:

1. It reads a line at a time. Whenever the user types CR, it enters CR, LF into the buffer; it echoes CR, LF; and then pads the remainder of the buffer with nulls and returns to the calling program. The characters get put into the buffer, one character per word. Thus every third character is a null as far as OS/8 is concerned.

2. RUBOUT deletes the previous character. It echoes as either a back slash (\) or as the character rubbed out, depending on assembly parameters. RUBOUT at the beginning of a line acts as ↑U.

3. CTRL/U echoes as ↑U and erases the current line, allowing the user to retype it. (It also echoes CR, LF.) The buffer pointer is reset to the beginning of the buffer.

4. CTRL/Z echoes as ↑Z (followed by CR, LF) and signals end-of-input. The ↑Z enters the buffer and the remainder of the buffer is padded with nulls. The error return is taken with a positive AC (non-fatal error).

5. Nulls are ignored.

*6. The altmode characters (octal 175 and 176) are converted to escapes (octal 33).

*7. Lower-case characters typed may be automatically converted to upper case.

8. CTRL/C echoes as ↑C and returns control to the keyboard monitor via location 07600.

On output: (either normal output or when echoing input)

1. CTRL/C on keyboard echoes as ↑C and returns control to the keyboard monitor via location 7600.

2. CTRL/O on keyboard stops further echoing. All echoing ceases (through possibly many buffer loads) until either the handler is reloaded into core or the user types a character other than ↑O on the keyboard. Not operative during input.

3. ↑S causes the handler to stop sending to the terminal. No characters are lost and outputting resumes when a ↑Q is typed. ↑S and ↑Q do not echo. These characters are

operative only upon output. On input, they are treated like any other input characters. This is very useful on high speed CRT displays.

4. Nulls are ignored.

*5. Lower case characters may be optionally printed as upper case and flagged with an apostrophe.

*6. Tabs may be handled in one of three manners:

a. Output as actual tabs,

b. Output as actual tab followed by padding of two rubouts,

c. Output as the correct number of spaces to bring the text to the start of the next tab stop.

7. Whenever the output line reaches the end of the physical line (length set at assembly time), the handler automatically performs a carriage-return line-feed.

*8. The escape character (octal 33) prints as a dollar sign.

*9. The handler may be set to delay about 16 ms after typing any character (specified at assembly time), for example, line feed.

*10. Control characters are printed as their corresponding letter preceded by an up-arrow. Thus CTRL/K prints as ↑K.

CHAPTER 5

RECONFIGURING THE OS/8 SYSTEM

It is sometimes necessary to construct an OS/8 system from scratch, or to make a new peripheral device available to OS/8. Both of these tasks are a part of reconfiguring the OS/8 system. OS/8 BUILD, which is described in detail in Chapter 2 of the OS/8 HANDBOOK, allows the user quickly and easily to build a new system or to alter the device complement of an existing system.

5.1 WRITING DEVICE HANDLERS

A device handler is a page-independent subroutine one or two pages long. 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 SR "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          /BLOCK NUMBER TO START 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:

1. On normal return from a device handler the AC is zero and the DATA FIELD is always restored to its original entry value.

2. 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.

3. Handlers should be written to be as foolproof as possible checking for the most common errors in the calling program.

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 (gives 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 (gives a fatal error return).

4. Device handlers normally check to see if a CTRL/C (ASCII 203) has been typed at the system console by the user. If one has, the handler aborts I/O and JMP's to location 7600 in field 0. The seven low order bits of the keyboard should be checked for a 3 so as to allow parity terminals. KRS should be used over KRB so that any paper tape in the reader will not be advanced if its character is not ↑C.

5. 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.

6. 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.

7. Remember that the user specifies a multiple of 128 words to transfer, whereas the transfer starts at the beginning of a 128 or 256 word block. This means that the handler must provide that capability of reading or writing the first half of a block. Writing the first half of the block causes the contents of the second half of the block to be altered. For example, writing 128 words to the RK8 disk (256 word blocks) causes the second half of the block to be filled with zeroes. This is usually done by the hardware controller.

8. The entry point to a two page device handler must be in the first page.

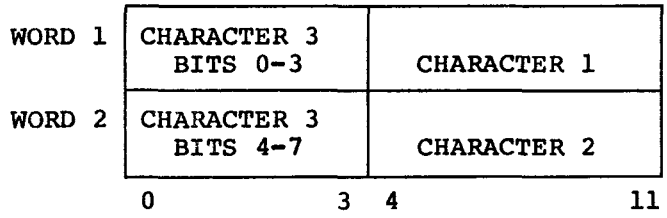
9. A number of handlers (maximum of 15 decimal) can be included in the one or two pages of code. Where more than one handler is included in a single handler subroutine, the handlers are called co-resident. Co-resident handlers are always brought into core together. For example, all eight DECtape handlers 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.

10. 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 Handlers	Relative Entry Points
System Device Handler	7
DECtape, LINCtape or TD8E DECtape	10 to 17
RKA0	20
RKA1	21
RKA2	22
RKA3	23
RK08 or DF32	24
Reserved for user	40 to 67

11. 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 OS/8 character packing format puts three 8-bit characters into two words as follows:



For example, the 3 characters 'ABC' would be packed into 2 words as follows:

Word 1: 6301
Word 2: 1702

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/, the remaining words of the input buffer should be zeroed.

12. A close operation should be performed by non-file structured handlers if bits 1-5 and 9-11 of the function word are 0.

The device handler, whether one or two pages long, must be completely page independent: it must be capable of executing in any page(s) in field 0, except page 0 and 7600 to 7777. Page independent code can have no address constants. Writing one page handlers is relatively easy, since the addressing structure of the PDP-8 is essentially page independent. Writing page relocatable code for two pages, however, is considerably more difficult, as the two pages must communicate. The usual technique utilized in writing two page handlers is to include some initialization code which includes a JMS. This replaces that location by an address on the page the handler was loaded on. Using this, the handler can then determine where the relevant pieces of code are in core.

As an example, the following is the initialization procedure performed by the TD8E DECtape routine. This is by no means the only technique that is possible, but it is a workable solution.

```
                *200
                                /EXECUTED CODE

JINIT,          JMP INIT        /START INITIALIZATION
                    .
                    .
                    .
                    .

INIT,           JMS.            /FOUND OUT WHERE WE ARE.
BASE,           TAD CRDQAD      /INIT GETS ADDRESS OF BASE
                SPA             /NEGATIVE TERMINATES LIST
                JMP NXINIT      /INITIALIZE SECOND PAGE
                TAD INIT        /NOW UPDATE THE LIST OF
                DCA CRDQAD      /ADDRESS DEPENDENT LOCATIONS
                ISZ .-1         /POINT TO NEXT ELEMENT
                ISZ BASE        /NEXT INPUT VALUE
                JMP BASE        /LOOP OVER INPUT TABLE.
                    .
                    .
                    .
                    .

CRDQAD,         R4LINE-BASE     /THESE ARE ALL POSITIVE DIFFERENCES,
CINIT2,         INIT2-BASE      /SINCE THE ROUTINES INDICATED ARE
CSELCT,         SELECT-BASE     /IN THE SECOND AGE. AFTER
CXUNIT,         XUNIT-BASE      /INITIALIZATION, CRDQAD POINTS TO
BUFF,           4000            /THE ACTUAL ADDRESS OF R4LINE, ETC.
                    .           /THE 4000 IN BUFF TERMINATES
                    .           /THE FIRST INITIALIZATION.
                    .           /MORE PAGE INDEPENDENT CODE

NXINIT,         JMS I CINIT2    /INITIALIZE SECOND PAGE
BASE2,          DCA JINIT       /CLEAR OUT JINIT. NO MORE
                JMP JINIT       /RELOCATING IS NEEDED UNTIL THE
                                /HANDLER IS LOADED INTO CORE GAIN.
                *400            /SECOND PAGE OF HANDLER

INIT2,          0               /ADDRESS OF BASE2 GOES HERE
INIT3,          TAD CTRY3
                SNA             /A 0 TERMINATES THIS LIST
                JMP I INIT2
                TAD INIT2       /ADD VALUE OF BASE2 TO LIST
                DCA CTRY3       /PUT BACK INTO LIST
                ISZ .-1         /NEXT LOC. TABLE
                ISZ INIT3
                JMP INIT3
                    .
                    .
                    .
                    .

CTRY3,          TRY3-BASE2      /THIS LIST GETS VALUE OF BASE2
CRWCOM,         TRWCOM-BSSE2    /ADDED IN TO POINT TO THE REAL
XBUFF,          0               /ROUTINE.
```

Writing 2 page independent code can be expensive in terms of core required. The routines should be set up in such a way as to minimize communication between the two pages. Some other points to keep in mind are:

1. Relocation code is once-only code. It is done once when the handler is loaded and need never be done again until the handler is re-loaded from the system device. For this reason, the relocation code can be placed in a buffer area or setup in temporary scratch locations which are later used as temporary storage.

2. A useful hint is that a JMP into the next page of code is not required. The code can just as easily fall through 377 into 400. This may save a few locations of relocation code.

3. Useful techniques for writing 2-page handlers can be found in the source of the KL8E handler.

5.2 INSERTING DEVICE HANDLERS INTO OS/8

After the handler has been written and thoroughly debugged as a stand-alone routine, it can be integrated into the OS/8 Monitor, where it will become a resident device handler. To accomplish the integration, use OS/8 BUILD, described thoroughly in the BUILD section in Chapter 2 of the OS/8 HANDBOOK.

Notes for writing system handlers system handlers may be integrated into BUILD just like non-system handlers with the following additional notes:

1. Body of system handler should be origined to 200 but must start with a 2 BLOCK 7. Entry point must be at relative location 7 (corresponds to location 7607).

2. Name of system handler must be SYS.

3. Each handler entry point has an 8-word handler block associated with it. The following additions apply:

   a. word 5: bits 9-11 should normally be 0.
   If the device can have multiple platters (like RF08) then this field specifies maximum number of platters allowed. Each platter above first bumps internal DCB code by 1.

   word 6: bit 0=1 means system device is two pages long. The second page is origined into 400 but resides in field 2 location 7600. Bit 1=1 if entry point is SYS. Bit 2=1 if entry point is coresident with SYS.

   word 7: must be 0

   word 10: number of blocks in device. Immediately following the header records is the code for the device's bootstrap. This is preceded by minus the number of words in the bootstrap. No origins may appear in this code. It must be less than 47 locations long.

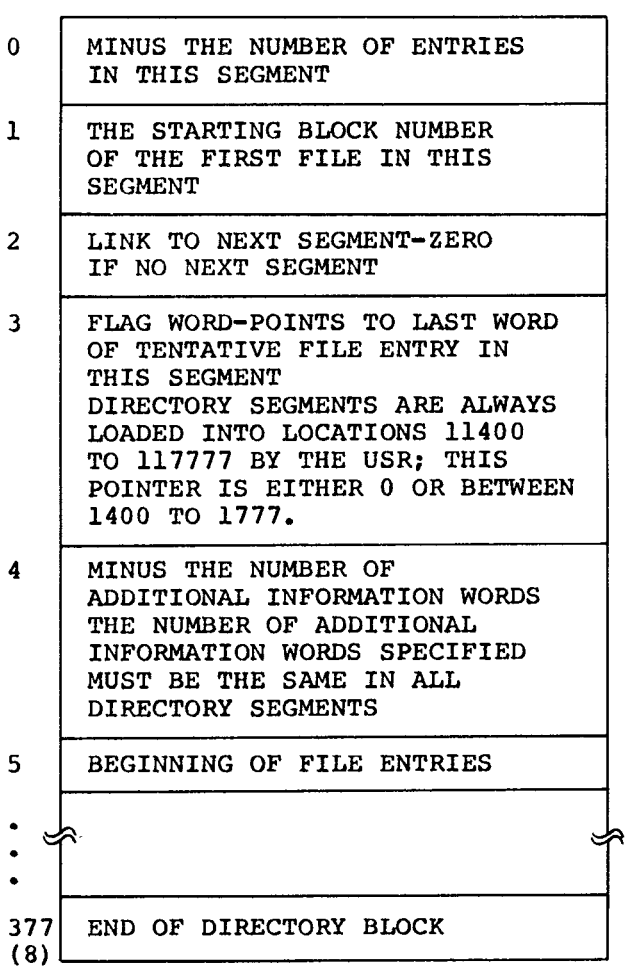
APPENDIX A

OS/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, though all are not necessarily active at any given time. To minimize the number of directory reads and writes necessary, OS/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.

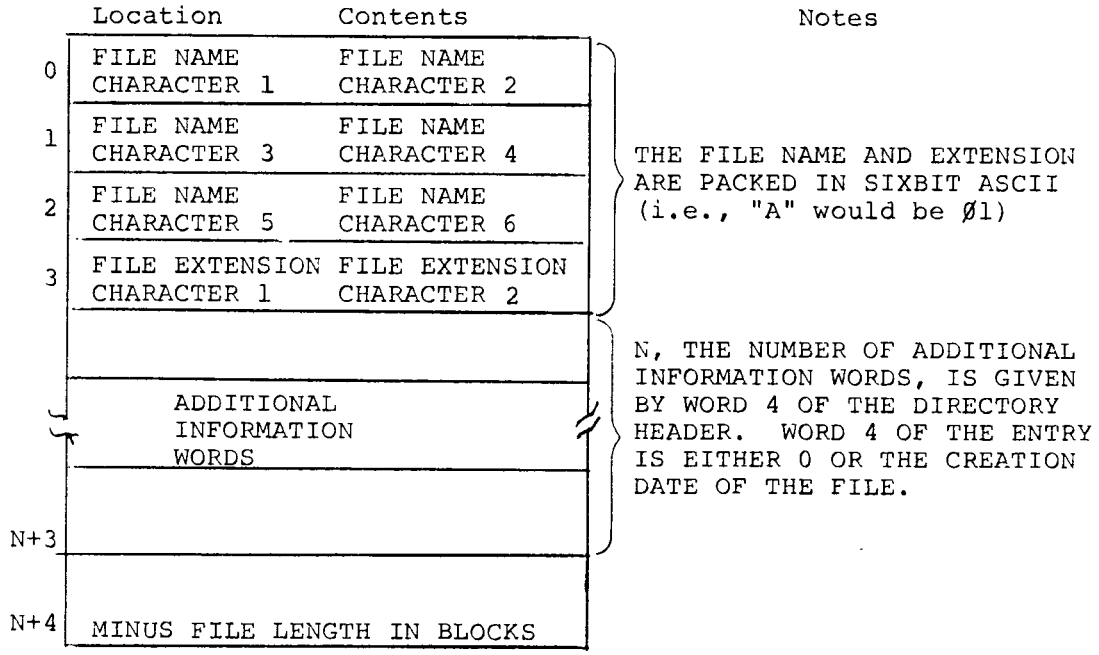
The directory blocks are each structured according to the following format:



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

A.1.1 Directory Entries

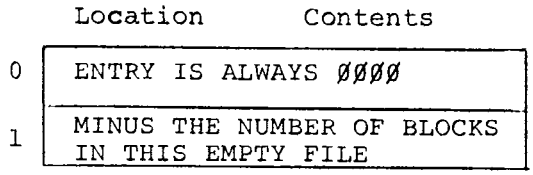
There are three types of file directory entries. They are PERMANENT FILE ENTRY, EMPTY FILE ENTRY, and TENTATIVE FILE ENTRY. A permanent file entry appears as follows:



NOTE

If word 3 is zero, the given file has a null extension.

An empty file entry appears as follows:



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 OS/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 OS/8 Files

All files on an OS/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:

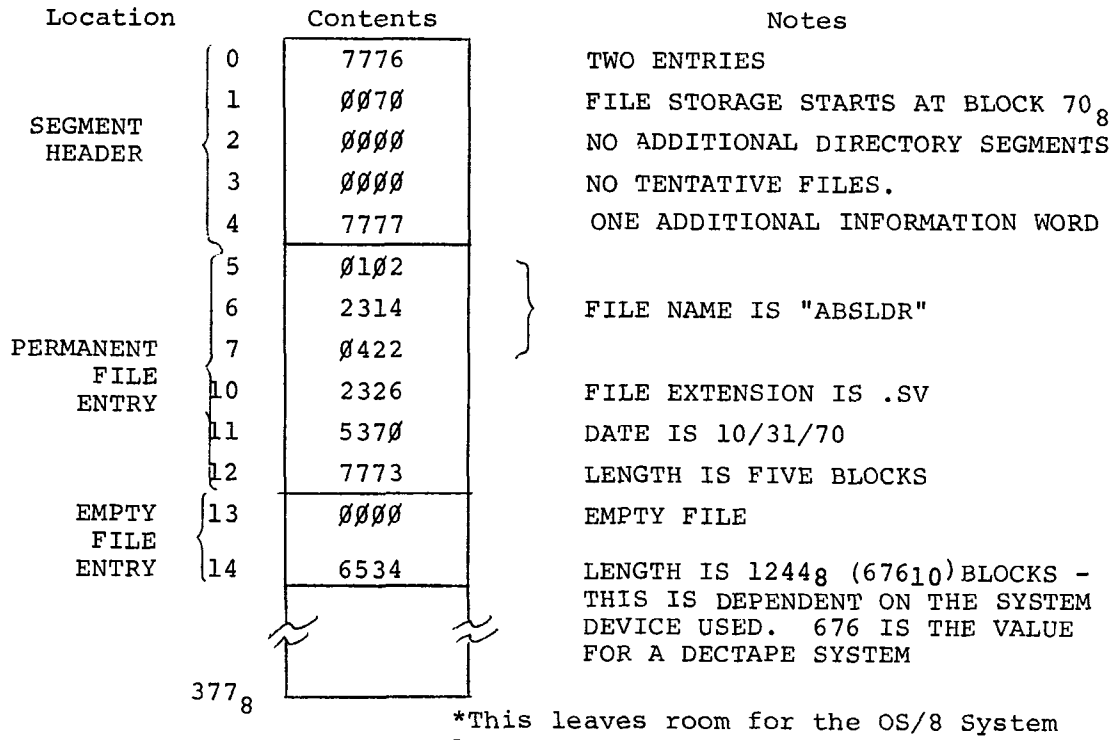
$$MIN = \left[\frac{256-7-(N+5)}{N+7}\right] = \left[\frac{244-N}{N+7}\right]$$

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

Finally, OS/8 devices are limited to 4095 blocks, each block being 256 words long. Thus, the maximum size of any single OS/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 OS/8 system is built looks as follows:



*This leaves room for the OS/8 System Areas.

A.2 FILE FORMATS

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

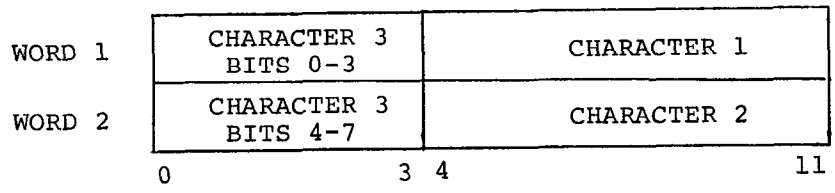
1. ASCII and Binary files.
2. Core Image files (.SV format).
3. Relocatable FORTRAN library files (LIB8.RL is the only current example of this format).

NOTE

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

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 OS/8 system programs:

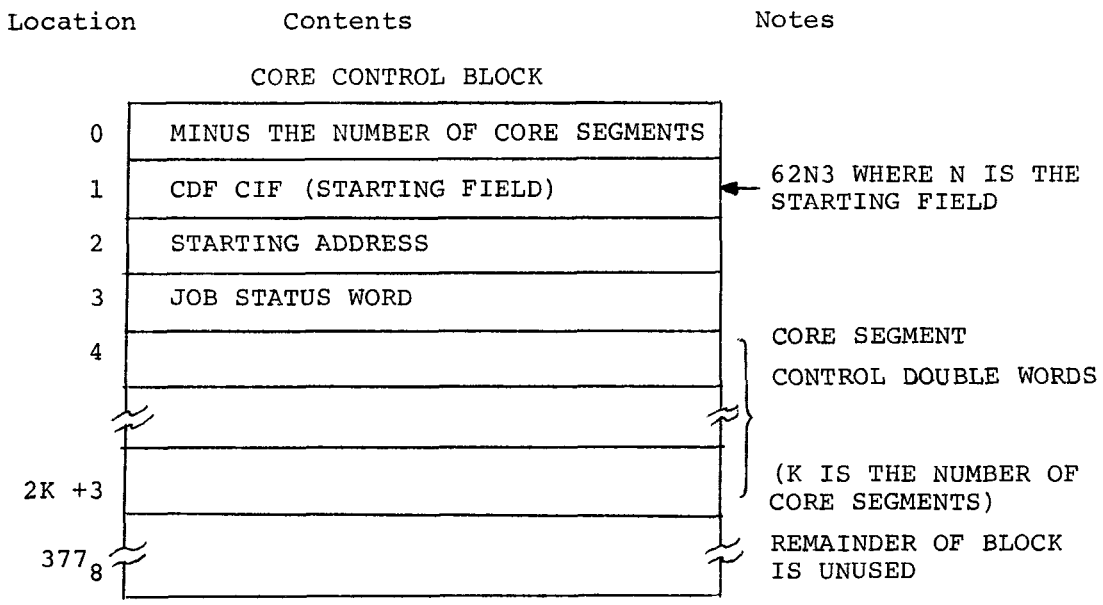
1. In ASCII files the character NULL (ASCII 000) is always ignored. Most programs only examine the low-order 7 bits, in ASCII files. The parity bit is usually ignored; do not assume that this bit is set or that data transfers will preserve it (image mode transfers, always preserve it).

2. 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), 240 to 257 (octal) (a COMMON declaration frame for relocatable binary files), or 300 (octal), which is an origin setting. The end of binary data is indicated by one or more frames of leader/trailer code.

3. ASCII and Binary files are terminated by a CTRL/Z code (ASCII 232). In binary files, a CTRL/Z code data rather than end-of-file.

A.2.2 Core Image (.SV Format) Files

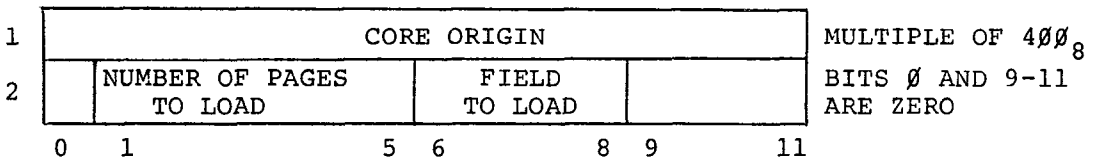
A core image file consists of a header by the actual core image. The header block is called the Core Control Block. The Core Control Block consists of the first 128 words of the 256 word block reserved for that purpose. The second 128 words are unused. The Core Control Block is formatted as follows:



The format of the Job Status Word is as follows:

Bit Condition	Meaning
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 3 = 1	Program never uses above 8K. This is used when Batch processing is active.
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 when 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:



The core origin must be a multiple of 400 (octal). 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 32 (decimal) Core Segment Control Doublewords in any Core Control Block.

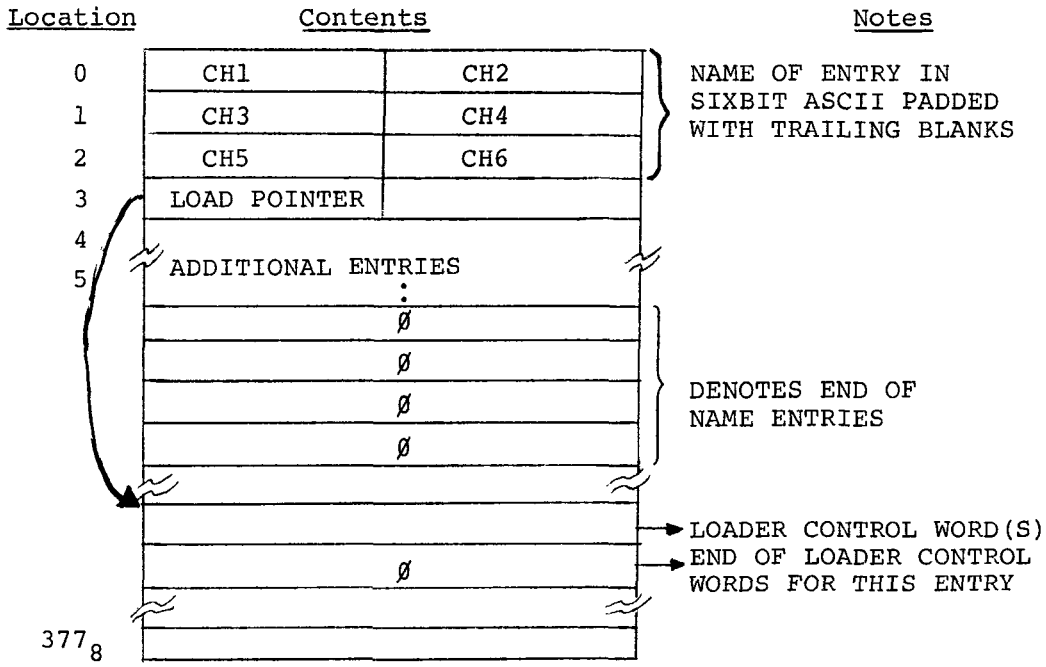
The Core Control Block for the program at the time it is loaded into core is always saved in words 200 (octal) through 377 (octal) of block 37 (octal) (one of the system scratch blocks) on the system device. It is placed there by the GET and RUN operations or by the ABSLDR or LOADER programs. This Core Control Block is used when performing a SAVE without arguments.

NOTE

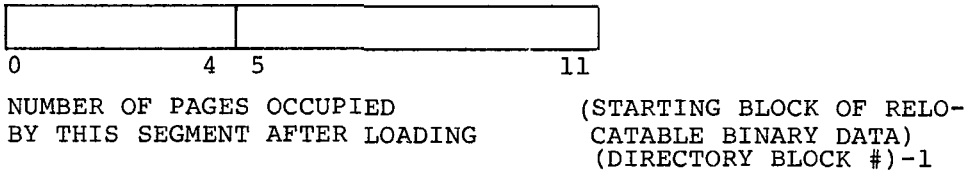
The R command differs from the RUN command in that the program's Core Control Block is not written onto the scratch area when using the R command. In order to SAVE a program that has been loaded by the R command all of the arguments of the SAVE command must be explicitly stated.

A.2.3 Relocatable FORTRAN Library File

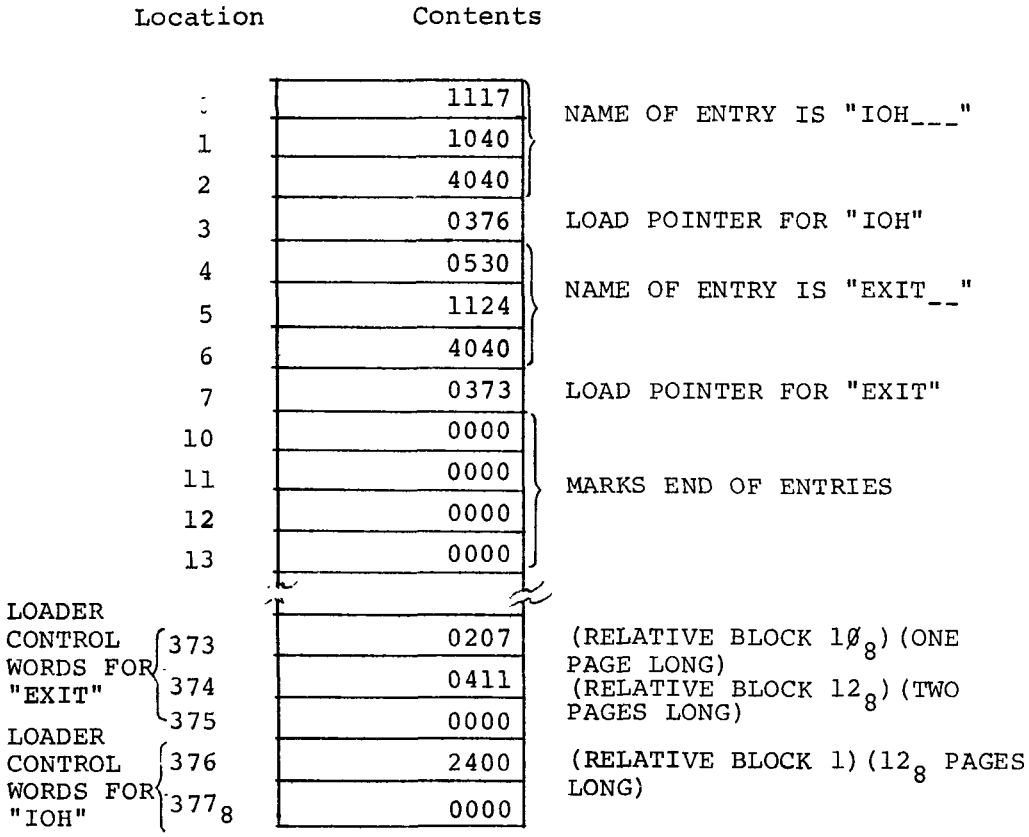
A relocatable FORTRAN library consists of a library directory block followed by relocatable binary segments. The directory block has the following format:



The Load Pointer is a number between 0 and 377 (octal) 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.


Location
Contents
1117
1040
4040
0376
0530
1124
4040
0373
0000
0000
0000
0000
NAME OF ENTRY IS "IOH___"
LOAD POINTER FOR "IOH"
NAME OF ENTRY IS "EXIT__"
LOAD POINTER FOR "EXIT"
MARKS END OF ENTRIES
LOADER CONTROL WORDS FOR "EXIT"
373
374
375
0207
0411
0000
(RELATIVE BLOCK 10$_8$) (ONE PAGE LONG)
(RELATIVE BLOCK 12$_8$) (TWO PAGES LONG)
LOADER CONTROL WORDS FOR "IOH"
376
377$_8$
2400
0000
(RELATIVE BLOCK 1) (12$_8$ PAGES LONG)

APPENDIX B

DETAILED LAYOUT OF THE SYSTEM

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

B.1 LAYOUT OF THE SYSTEM DEVICE

The first 70 octal blocks (14K words) on the system device are reserved by the OS/8 system. These blocks are used as follows:

Block(s) in Octal	Contents
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	Reserved for System Expansion
65	CCL Reminiscences
66	12K TD8E Resident code
67	CCL Overlay

File storage begins with block 70 (octal).

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 200 (octal) to 377 (octal) of block 37 (octal). The Core Control Block is stored at those locations by the GET or RUN command or by the ABSLDR or LOADER program.

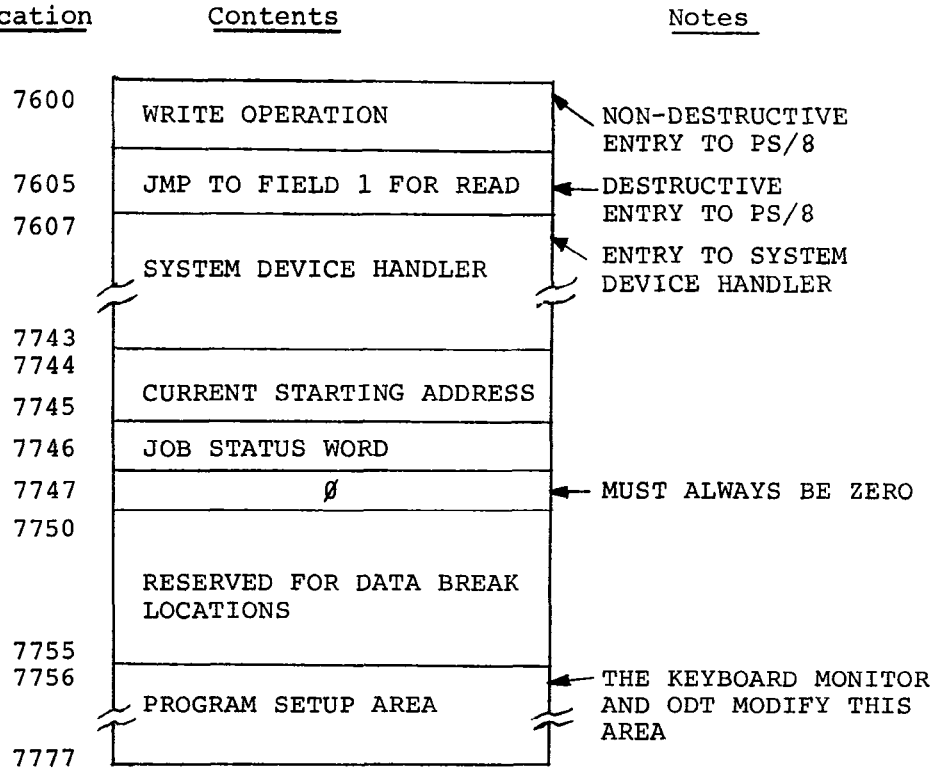
A detailed breakdown of system scratch block usage follows:

Block(s) in Octal	Contents
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 (octal) to 377 (octal) 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.

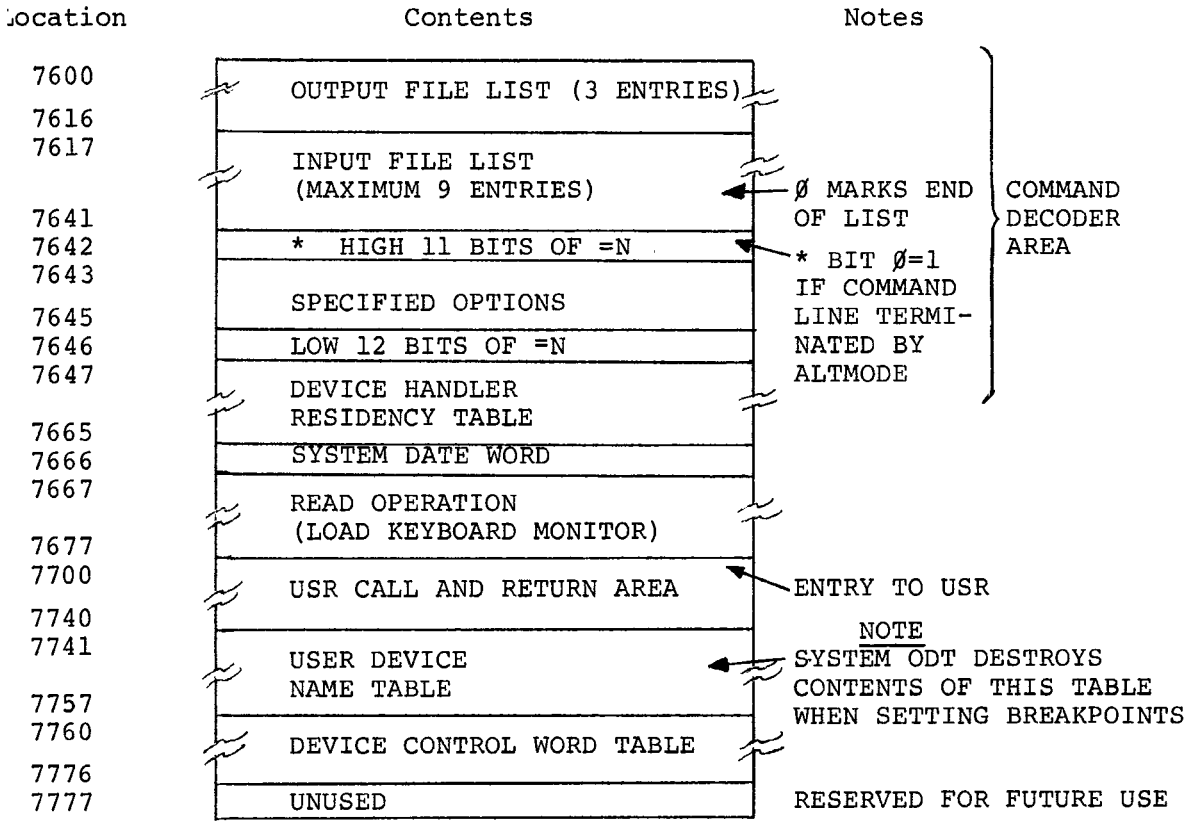
B.2 LAYOUT OF THE OS/8 RESIDENT PROGRAM

The top core pages in fields 0, 1, and 2 are used by the resident portion of OS/8 and are not accessable by the user. 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 OS/8 is structured as follows:

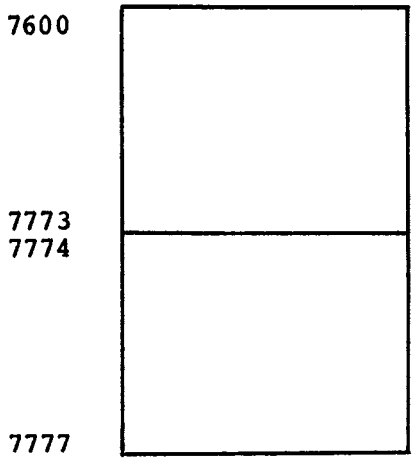


TOP PAGE OF FIELD 1



Systems built around TD8E DECtape without the Read-Only-Memory option use 7600 in field 2 as an extension of the system device handler.

TOP PAGE OF FIELD 2.



Used only for 12K TD8E systems. Part of system handler resides here in that case.

Four words reserved for BATCH use if machine has exactly 12K. Contains pointers into input file.

If the machine has more than 12K, the top 4 location (7774-7777) of the last field are reserved for use by BATCH.

If a ROM (Read-Only-Memory) is being used with an 8K TD8E system, locations 7400-7777 of field 7 are inaccessible to the user. That core is used for system handler functions.

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:

1. 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.

2. A single word is created by adding together these two words.

3. 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 able 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.

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 (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:

Bit Condition	Meaning
Bits 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 the entry points to all handlers must be 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:

Bit Condition	Meaning
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),
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 77 (octal), of which 0 through 20 (octal) are currently assigned to existing devices, as follows:

Device Code	Device
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	TM8E Magnetic Tape
21	TD8E DECtape
22	BATCH handler
23	RK8E Disk
24	NULL
25-26	reserved for future disks
27	TA8E Cassette
30	VR12 Scope
31-37	reserved for future use by DEC
40-57	reserved for use by users

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 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 (octal) (minus 2000 (octal), or 1024 (decimal), 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.

A similar table occurs in RESORC which the user may wish to patch. It is located in field 0 locations 2000-2377 and contains 64 four-word entries; one entry for each device type. Words 1 and 2 of an entry are the names of the device (in sixbit) and word 3 is the negative of the number. Word 4 of the entry should be 0 for non-standard devices.

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 OS/8 HANDBOOK.

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:

Value of PC	Meaning
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 not 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 re-bootstrap the system.
07764	A read error occurred while loading a program. Return to the Keyboard Monitor by restarting at 07605.
07772	A read error ocurred on the system scratch area while loading a program. Return to the Keyboard Monitor by restarting at 07605.

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 by restarting at 07605.
17676	An error occurred while attempting to read the Keyboard Monitor from the system device. Try again be restarting at location 07605. DO NOT PRESS CONTINUE.
17721	An error occurred while saving the USR area. Verify that the system device is not 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:

Message	Meaning
MONITOR ERROR 1 AT xxxxx [CLOSE ERROR]	File length in CLOSE function is too large.
MONITOR ERROR 2 AT xxxxx [DIRECTORY I/O ERROR]	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 [DEVICE HANDLER NOT IN CORE]	The device handler required for a file operation (LOOKUP, ENTER, CLOSE) is not in core.
MONITOR ERROR 4 AT xxxxx [ILLEGAL USR CALL]	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 ON SYS]	I/O error occurred while reading or or writing on the system device. Verify that the system device is not WRITE LOCKed.
MONITOR ERROR 6 AT xxxxx [DIRECTORY OVERFLOW]	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:

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

If an I/O error is made during the monitor CHAIN function the message

CHAIN ERR

is generated, and control returns to the keyboard.

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:

Message	Meaning
aaaa?	The Keyboard Monitor cannot interpret the command "aaaa". For example if the user types HELLO the system will respond HELLO?
BAD ARGS	Arguments to a SAVE command are inconsistent, or illegal.
BAD CORE IMAGE	The file requested with an R, RUN, or GET command is not a core image file.
BAD DATE	Improper syntax in a DATE command.
device NOT AVAILABLE	The permanent device name specified in an ASSIGN, SAVE, RUN, or GET command does not exist.
ILLEGAL ARG	The SAVE command was not expressed correctly.
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.
NO!!	A START command (with no address specified) is prohibited when bit 2 of the Job Status Word (location 07746) is a 1.
NO CCL!	Command was a valid CCL command but CCL.SV is not on the system.
SAVE ERROR	An I/O error occurred while saving the program. The contents of core remain intact.
SYSTEM ERR	An error occurred while doing I/O to the system device.
TOO FEW ARGS	An argument has been omitted from a command.

C.4 CCL ERROR MESSAGES

Message	Meaning
BAD DEVICE	The device specified in a CCL command is not of the correct form, (e.g., DTA0.PA:).
BAD EXTENSION	Either an extension was specified without a file name (e.g., DTA1:.PA) or two extensions were specified (e.g., DTA1:FILE.PA.BN).

BAD MONITOR	The version of the Keyboard Monitor being used is not compatible with CCL. A new version of the monitor must be obtained from Digital before CCL can be used.
BAD NUMBER	A CCL command which uses the # construction does not have the full 16-digit specification that is required.
BAD RECOLLECTION	An attempt was made to use a previously remembered argument when no argument was saved. This error occurs when no argument was previously saved or when the DATE command has been used since the argument was saved.
BAD SWITCH OPTION	The character used with a slash (/) to indicate an option is not a legal option.
CANNOT CHANGE CORE CAPACITY WHILE RUNNING BATCH	A CORE command was issued while the BATCH program was running.
%CANT REMEMBER	The argument specified in a CCL command line is too long to be remembered or an I/O error occurred.
CCL 3X OVERLAY & MONITOR INCOMPATIBLE	The version of CCL being used is not compatible with the Keyboard Monitor present on the system. Type R CCL to retry.
COMMAND LINE OVERFLOW	The command line specified with the @ construction is more than 512 characters in length.
COMMAND TOO LONG	The length of a text argument in a MUNG command is too long.
CONTRADICTOR SWITCHES	Either two CCL processor switches were specified in the same command line (e.g., FILE-PA-FT) or the file extension and the processor switch do not agree (e.g., FILE.FT-BA).
name DOES NOT EXIST	The device with the name given is not present on the OS/8 system.
ERROR IN COMMAND	A command not entered directly from the console terminal is not a legal CCL command. This error occurs when the argument of a UA, UB, or UC command was not a legal command.
ILLEGAL * OR ?	An * or ? was used in a CCL command that does not accept the wild card construction. Only CCL commands that run FOTP or DIRECT allow the wild card construction.

ILLEGAL SYNTAX	The CCL command line was formatted incorrectly.
INPUT ERROR READING INDIRECT FILE	CCL cannot read the file specified with the @ construction.
I/O ERROR ON SYS:	An error occurred while doing I/O to the system device. The system must be restarted at 7600 or 7605. Do not press CONT. as that will surely cause further errors.
I/O ERROR TRYING TO RECALL	An I/O error occurred while CCL was trying to remember an argument.
NO CCL!	CCL.SV is not present on the system device.
NOT ENOUGH CORE	The number specified in a CORE command is larger than the number of 4K core banks on the system.
name NOT FOUND	The file with the name given is not present on the specified device, or the user tried to input from an output-only device.
%SUPERCEDED	The file specified in a MAKE command already exists. This is a warning message indicating that the file is being replaced.
SWITCH NOT ALLOWED HERE	Either a CCL option was specified on the left side of the < or was used when not allowed. For example: COMPARE FILE-NB.
TOO MANY FILES	To many files were included in a CCL command.

C.5 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.

Message	Meaning
ILLEGAL SYNTAX	The command line is formatted incorrectly.
TOO MANY FILES	More than three output files or nine input files were specified (regular mode) or > 1 output or > 5 input (special mode).
name NOT FOUND	The specified input file name was not found on the device indicated.

APPENDIX D

PROGRAMMING NOTES

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

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. A useful technique is to use the ASSIGN command to redefine the meaning of DSK temporarily. For example, where device DTA0 is equivalent to DSK and

it becomes desirable to change DSK to DTA1, the following command can be given:

```
.ASSIGN DTA1 DSK
```

DTA1 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.

If 'DSK' has not been assigned via the ASSIGN command, then 'DSK' always exists and has internal device number 2. User programs wishing to use DSK should do an INQUIRE to find its number in case the operator has re-assigned it.

D.2 MODIFICATION TO CARD READER HANDLER

The standard card reader handler for OS/8 uses the DEC029 standard card codes. Some installations may prefer to use the DEC026 codes instead. This can be done by changing the card conversion codes with the BUILD command ALTER.

1. Call OS/8 BUILD by typing:
   ```
   RUN SYS BUILD
   ```
   in response to the dot printed by the Keyboard Monitor.

2. Load the card reader handler as described on page 2-42 of the OS/8 HANDBOOK.

3. Use the ALTER command (see page 2-49 of the OS/8 HANDBOOK) to make the following changes:

CHANGE RELATIVE LOCATION	FROM	TO
104	3203	7735
105	4007	4076
106	3502	0774
114	7514	3314
115	0577	1002
116	3637	0305
124	0104	3204
125	1211	1273
126	3374	3606
127	0641	1341
134	7316	3716
135	3410	1175
136	1376	3401

The new system will have modified card codes.

Note that 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 #2. (DEC-S8-OSYSB-A-UA2)

026 PUNCH CARD CODES

Octal 8-bit CODE	DEC026 CODE	CHARACTER	Octal 8-bit CODE	DEC026 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	)	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	

NOTE

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

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 OS/8 FORTRAN can be used to achieve this result:

1. 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)4IS )
        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
```

2. 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',IO)
```

3. 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 OS/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
        WRITE (1,100) IMO,IDAY,IYR
100     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 OS/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:

1. 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.

2. 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.

3. The PDP-8/L normally treats all CDF's to fields 2 through 7 as NOP's. (It tests bits 6 to 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'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.

```
        /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
0201 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
0201 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)
0020 1635           TAD I   CORLOC     /TRY TO READ BACK 7000
0221 7400   CORX,   7400               /(HACK FOR PDP-8,.NO-OP)
0022 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,  CDF0               /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 IELD
0236 1400   CORV,   1400               /7000+7400+1400=0
0237 0001   CORSIZ, 1                  /CURRENT FIELD TO TEST
```

D.6 USING PRTC12-F TO CONVERT OS/8 DECTAPES TO OS/12 LINCTAPES

Many users of OS/8 on the PDP-12 will be interested in the fact that, since OS/8 uses an identical file structure on all devices, PDP-8 DECtape in OS/8 format may be directly copied to OS/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 OS/8 format DECtapes are as follows:

1. Mount the OS/8 DECtape on unit 1 and a PDP-12 LINCtape formatted with 129 words per block on unit 2.

2. 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.
```

3. 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.7 NOTES ON LOADING DEVICE HANDLERS

D.7.1 Problem With Multiple Input Files

There is a problem associated with reusing Device Handler areas in OS/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.

D.7.2 Dynamically Loading Device Handlers

Some programs which use dynamic core allocations will want to use OS/8 Device handlers but cannot afford to always allocate the maximum of two pages per handler. The following is a subroutine 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 Ø 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 rewritten for any other possibility.

```
ASSIGN, Ø
        TAD NAME1
        DCA N1
        TAD NAME2
        DCA N2          /MOVE DEVICE NAME INTO "INQUIRE" COMMAND
        CDF CIF 1Ø
        JMS I (77ØØ
        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.8 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:

1. Locations 10000 to 10006 are available for scratch storage and/or ODT breakpoint usage, without restriction.

2. All auto-index registers (locations 10010 to 10017) may be used, but these locations are destroyed by USR operations.

3. Location 10020 to 10037 may be used as scratch storage with no restrictions.

4. 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 100007 to a non-zero value again.

D.9 ACCESSING ADDITIONAL INFORMATION WORDS IN OS/8

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

D.9.1 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.

D.9.2 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.

D.9.3 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 10           /CODE IS IN FIELD 1
TAD 7            /GET DIRECTORY KEY WORD
AND (7           /EXTRACT SEGMENT NUMBER
DCA SEGNO
CIF 0
JMS I 51         /LOC 51 POINTS TO THE DEVICE HANDLER
```

```
        4210            /WRITE OPERATION
        1400            /DIRECTORY SEGMENT CORE ADDRESS
SEGNO,  0
        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.10 SABR PROGRAMMING NOTES

D.10.1 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:

```
                Code

IARG,   DUMMY X
        0
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:

```
                Code

IARG,   0
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.

D.10.2 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 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 10
        JMS 7700         /OR 200 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:

```
        CPAGE12          /10 IF "HAND" IN PAGE 0
        6202             /CIF 0
        JMS I HAND       /DO NOT USE JMSI
        FUNCT
        ADDR
        BLOCK
        ERROR RETURN
        SKP
HAND,   0                /"HAND" MUST BE ON SAME PAGE
                         /AS CALL, OR IN PAGE 0!!
```

APPENDIX E

CHARACTER CODES AND CONVENTIONS

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

1. On some terminals, the character back-arrow (←) is replaced by an underline (_) character, and the up-arrow (↑) is replaced by circumflex (^).

2. Some terminals use parity codes rather than forcing the leading bit of the 8-bit character code to be a 1. To avoid problems, OS/8 system programs always ignore the parity bit during ASCII input.

3. OS/8 does not handle lower case characters (octal codes 341 through 372). The exceptions to this are the editors, EDIT and TECO. The KL8E and LPSV handlers can be modified to handle lower case.

Table E-1
OS/8 Control Characters

Octal 8-bit Code	Character Name	Remarks
000	null	Ignored in ASCII input.
200	leader/trailer	Leader/trailer code precedes and follows the data portion of binary files.
203	CTRL/C	OS/8 break character, forces return 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.

Table E-1 (Cont.)
OS/8 Control Characters

Octal 8-bit Code	Character Name	Remarks
215	RETURN	Carriage return, generally echoed as carriage return followed by a line feed.
216		Used only on LS8E line printer. Puts current line into expanded character mode.
217	CTRL/O	Break Character, used 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 terminator if it occurs before the trailer code).
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

Octal 8-bit Code	6-bit Code	Punched Card Code	Character Representation	Remarks
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
244	44	11-8-3	$	dollar sign
245	45	0-8-3	%	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	
310	10	12-8	H	
311	11	12-9	I	

Table E-2 (Cont.)
ASCII Character Codes

Octal 8-bit Code	6-bit Code	Punched Card Code	Character Representation	Remarks
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	[	opening bracket,SHIFT/K
334	34	11-8-7	\	backslash, SHIFT/L
335	35	0-8-2	]	closing bracket, SHIFT/M
336	36	12-8-7	^	circumflex
337	37	0-8-5	_	underline - EOF signal

NOTES

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 back-arrow (←).
5. On some IBM 029 keyboards [is graphically represented as a cent sign (¢).
6. On some IBM 029 keyboards \ is graphically represented as logical NOT (¬).
7. On some IBM 029 keyboards ^ 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

OS/8 INPUT/OUTPUT ROUTINES

Appendix F describes a set of generalized I/O routines for use under the OS/8 system. The routines presented here are used in all the OS/8 CUSPs (Commonly Used System Programs) in more or less this form. Variations are made depending on the particular application and how errors are to be handled. The routines, as indicated, will work as presented. The routines work most efficiently in field 1, since CIF 10's are not necessary when addressing the Monitor, and the Command Decoder tables are similarly available. Obviously the routines can be modified to run in any memory field or core locations.

F.1 GENERAL DESCRIPTION

These subroutines assume that the Command Decoder tables have been set up to indicate the proper I/O devices. The routines handle device handler assignment without user interference. All I/O is done by simple subroutine calls. The user program never needs to interface with the Monitor or device handlers. All buffering and internal bookkeeping are performed by the routines. In these routines, it is assumed that only one output device is used at a time, i.e., the output routine does not automatically set up for the next output device. This modification can be made if desired. As many as nine inputs are handled automatically. When input from one device is exhausted, the input routine will automatically utilize the next device specified in the Command Decoder list of inputs.

F.2 SUBROUTINES FUNCTIONS

Following is a brief list of the subroutines and their functions.

ICHAR - Character input routine.

Call sequence:

```
JMS ICHAR
ERROR RETURN
NORMAL RETURN
```

Error:

If AC>0, an EOF on input has occurred. No more input is available. If AC<0, a device error has occurred.

Normal:

8 bit character is in the AC.

OCHAR - Character output routine.

Call:

```
TAD CHAR /8 BIT CHARACTER
JMS OCHAR
ERROR RETURN
NORMAL RETURN
```

Error:

AC<0 implies a fatal error.
AC > or = 0 implies that the hole allotted for output was exceeded.

Normal:

AC=0. The character has been put into the device output buffer.

IOPEN - Input initialize routine.

Call:

```
JMS IOPEN
RETURN
```

Return:

Input pointers reset. The next call to ICHAR will read from the first device in the Command Decoder input list.

OOPEN - Output initialize routine.

Call:

```
JMS OOPEN
ERROR RETURN
NORMAL RETURN
```

Error:

If AC> or = 0, no output device was specified.
If AC<0, an error occurred opening the file.

Normal:

An output file has been opened. No action if the output was a non-file structured device.

OCLOSE - Output close routine.

Call:

```
JMS OCLOSE
ERROR RETURN

NORMAL RETURN
```

Error:

Either the closing length is too large for the space allotted or an output error has occurred.

Normal:

The output file is now a permanent file on the output device.

F.3 SUBROUTINE PARAMETERS

These subroutines handle device assignment and internal buffering automatically. To accomplish this, certain parameters must be defined at assembly time. These parameters specify all details of handler location, and buffer size for the routines.

Parameter	Definition
INBUF =	Address of input buffer.
INCTL =	Input buffer control word. See the section on using device handlers for details of the control word format.
OUBUF =	Output buffer address.
OUCTL =	Output buffer control word. This must be a negative number to indicate a write operation.
INRECS =	Number of input records in input buffer. INRECS = INCTL/256 (DECIMAL).
INDEVH =	Address of input device handler.
OUDEVH =	Address of output device handler.

The parameters can either be a part of the actual subroutine source, or they can be contained in a separate parameter file to be assembled with the subroutine file. The latter approach provides greater flexibility in using the routines.

F.3.1 Example

Following is a sample of the use of the subroutines. The program simply calls the Command Decoder, and transfers input from the input devices to the output file, closes the output, and exits.

```
          FIELD 1
          *2000
          JMS I (7700              /LOCK MONITOR INTO CORE.
          10
CALLCD,   JMS I (200               /CALL COMMAND DECODER
          5                        /TO PICK OPTIONS.
          0
          JMS I (IOPEN             /SETUP TO START LOOKING AT
                                   /CD INPUT FILE.
          JMS I (OOPEN             /OPEN UP AN OUTPUT FILE.
          SMA CLA                  /IF AC<0, WE HAD A FATAL
          JMP OK                   /TYPE ERROR. AC>0 IS O.K.
          JMS TERR                 /ERROR.
          TEXT /OPEN FAILED/
OK,       JMS I (ICHAR
          JMP TSTEOF               /EITHER ERROR OR EOF.
                                   /SAVE IT.

          JMS I (OCHAR             /TRANSFER THE CHARACTER
          JMP OUTERR               /OUTPUT ERROR
          JMP OK                   /TRANSFER UNTIL EOF FOUND.
TSTEOF,   SMA CLA                  /IF NEG., FATAL
          JMP CLOSE                /EOF. CLOSE OUTPUT
          JMS TERR
          TEXT /READ ERROR/
CLOSE,    JMS I (OCLOSE            /CLOSE OUTPUT FILE.
          JMP CLERR                /CLOSE ERROR
          JMP CALLCD               /NEXT.
OUTERR,   JMS TERR
          TEXT /OUTPUT ERROR/
CLERR,    JMS TERR
          TEXT /CLOSE ERROR/

TERR,     0
          TAD I TERR
          RTR;RTR;RTR
          JMS TYPIT
          TAD I TERR
          JMS TYPIT
          ISZ TERR
          JMP TERR+1

TYPIT,    0
          AND (77
          SNA
          JMP CRLF
          TAD (300
          JMS TTYOUT
          JMP I TYPIT
CRLF,     TA (215
          JMS TTYOUT
          TAD (212
          JMS TTYOUT
          JMP CALLCD
TTYOUT,   0
          TLS
          TSF
          JMP.-1; CLA; JMP I TTYOUT
```

F.3.2 Subroutine Listing

A listing of the routines follows. The parameters are set up in such a way as to allow them to be put into a separate file. Another parameter, ORIGIN, determines the location of the routines.

INDEX

HOW TO OBTAIN SOFTWARE INFORMATION

SOFTWARE NEWSLETTERS, MAILING LIST

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

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

SOFTWARE PROBLEMS

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

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

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

PROGRAMS AND MANUALS

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

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

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

USERS SOCIETY

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

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

READER'S COMMENTS

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

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

_____

_____

_____

_____

_____

_____

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

_____

_____

_____

_____

_____

_____

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

_____

_____

_____

_____

_____

_____

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

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

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

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

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

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

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

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

Postage will be paid by:

digital

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