DIGITAL RESEARCH

Post Office Box 579, Pacific Grove, California 93950, (408) 373-3403

CP/M INTERFACE GUIDE

TABLE OF CONTENTS

CP/M INTERFACE GUIDE

1. INTRODUCTION

This manual describes the CP/M system organization including the structure of memory, as well as system entry points. The intention here is to provide the necessary information required to write programs which operate under CP/M, and which use the peripheral and disk I/O facilities of the system.

1.1 CP/M Organization

CP/M is logically divided into four parts:

BIOS - the basic I/O system for serial peripheral control

BDOS - the basic disk operating system primitives

CCP - the console command processer

TPA - the transient program area

The BIOS and BDOS are combined into a single program with a common entry point and referred to as the FDOS. The CCP is a distinct program which uses the FDOS to provide a human-oriented interface to the information which is cataloged on the diskette. The TPA is an area of memory (i.e, the portion which is not used by the FDOS and CCP) where various non-resident operating system commands are executed. User programs also execute in the TPA. The organization of memory in a standard CP/M system is shown in Figure 1.

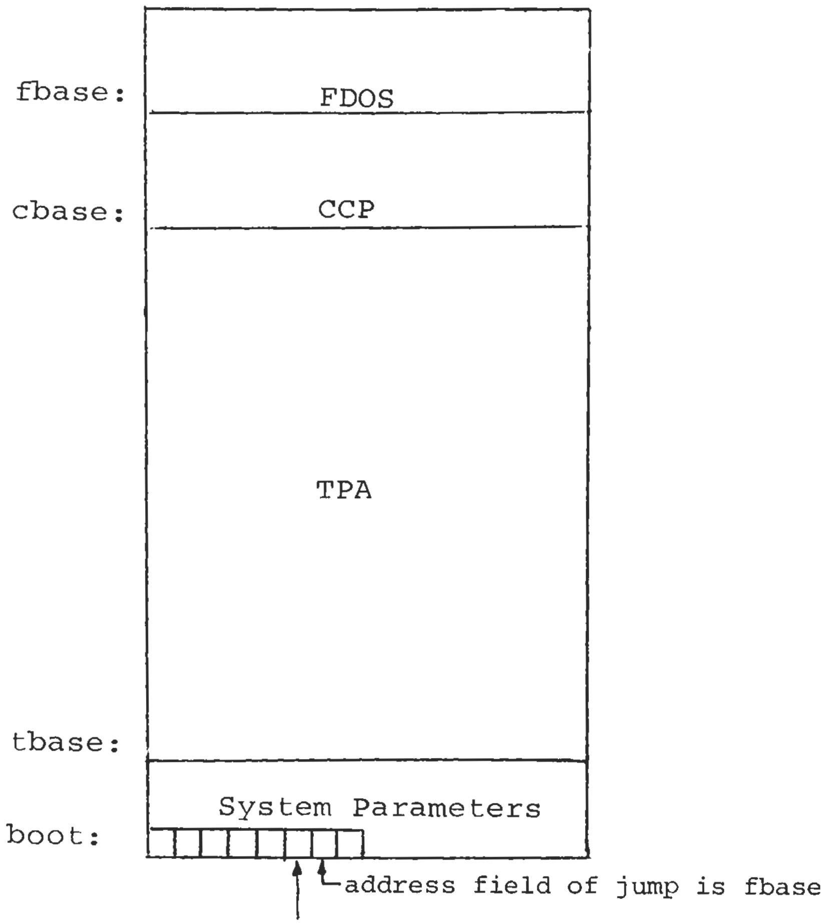
The lower portion of memory is reserved for system information (which is detailed in later sections), including user defined interrupt locations. The portion between tbase and cbase is reserved for the transient operating system commands, while the portion above cbase contains the resident CCP and FDOS. The last three locations of memory contain a jump instruction to the FDOS entry point which provides access to system functions.

1.2 Operation of Transient Programs

Transient programs (system functions and user-defined programs) are loaded into the TPA and executed as follows. The operator communicates with the CCP by typing command lines following each prompt character. Each command line takes one of the forms:

<command>
<command> <filename>
<command> <filename>.<filetype>

Figure 1. CP/M Memory Organization



entry: the principal entry point to FDOS is at location 0005 which contains a JMP to fbase. The address field at location 0006 can be used to determine the size of available memory, assuming the CCP is being overlayed.

Note: The exact addresses for boot, tbase, cbase, fbase, and entry vary with the CP/M version (see Section 6. for version correspondence).

Where <command> is either a built-in command (e.g., DIR or TYPE), or the name of a transient command or program. If the <command> is a built-in function of CP/M, it is executed immediately; otherwise the CCP searches the currently addressed disk for a file by the name

<command>.COM

If the file is found, it is assumed to be a memory image of a program which executes in the TPA, and thus implicitly originates at tbase in memory (see the CP/M LOAD command). The CCP loads the COM file from the diskette into memory starting at tbase, and extending up to address cbase.

If the <command> is followed by either a <filename> or <filename>.<filetype>, then the CCP prepares a file control-block (FCB) in the system information area of memory. This FCB is in the form required to access the file through the FDOS, and is given in detail in Section 3.2.

The program then executes, perhaps using the I/O facilities of the FDOS. If the program uses no FDOS facilities, then the entire remaining memory area is available for data used by the program. If the FDOS is to remain in memory, then the transient program can use only up to location fbase as data.* In any case, if the CCP area is used by the transient, the entire CP/M system must be reloaded upon the transient's completion. This system reload is accomplished by a direct branch to location "boot" in memory.

The transient uses the CP/M I/O facilities to communicate with the operator's console and peripheral devices, including the floppy disk subsystem. The I/O system is accessed by passing a "function number" and an "information address" to CP/M through the address marked "entry" in Figure 1. In the case of a disk read, for example, the transient program sends the number corresponding to a disk read, along with the address of an FCB, and CP/M performs the operation, returning with either a disk read complete indication or an error number indicating that the disk operation was unsuccessful. The function numbers and error indicators are given in detail in Section 3.3.

1.3 Operating System Facilities

CP/M facilities which are available to transients are divided into two categories: BIOS operations, and BDOS primitives. The BIOS operations are listed first:**

* Address "entry" contains a jump to the lowest address in the FDOS, and thus "entry+1" contains the first FDOS address which cannot be overlayed.

**The device support (exclusive of the disk subsystem) corresponds exactly to Intel's peripheral definition, including I/O port assignment and status byte format (see the Intel manual which discusses the Intellec MDS hardware environment).

Read Console Character
Write Console Character
Read Reader Character
Write Punch Character
Write List Device Character
Set I/O Status
Interrogate Device Status
Print Console Buffer
Read Console Buffer
Interrogate Console Status

The exact details of BIOS access are given in Section 2. The BDOS primitives include the following operations:

Disk System Reset
Drive Select
File Creation
File Open
File Close
Directory Search
File Delete
File Rename
Read Record
Write Record
Interrogate Available Disks
Interrogate Selected Disk
Set DMA Address

The details of BDOS access are given in Section 3.

2. BASIC I/O FACILITIES

Access to common peripherals is accomplished by passing a function number and information address to the BIOS. In general, the function number is passed in Register C, while the information address is passed in Register pair D,E. Note that this conforms to the PL/M Conventions for parameter passing, and thus the following PL/M procedure is sufficient to link to the BIOS when a value is returned:

```
DECLARE ENTRY LITERALLY    '0005H'; /* MONITOR ENTRY */

MON2:  PROCEDURE (FUNC, INFO) BYTE;
       DECLARE FUNC BYTE, INFO ADDRESS;
       GO TO ENTRY;

       END MON2;
```

or

```
MON1: PROCEDURE (FUNC,INFO);
      DECLARE FUNC BYTE, INFO ADDRESS;
      GO TO ENTRY;
      END MON1
```

if no returned value is expected.

2.1 Direct and Buffered I/O.

The BIOS entry points are given in Table I. In the case of simple character I/O to the console, the BIOS reads the console device, and removes the parity bit. The character is echoed back to the console, and tab characters (control-I) are expanded to tab positions starting at column one and separated by eight character positions. The I/O status byte takes the form shown in Table I, and can be programmatically interrogated or changed. The buffered read operation takes advantage of the CP/M line editing facilities. That is, the program sends the address of a read buffer whose first byte is the length of the buffer. The second byte is initially empty, but is filled-in by CP/M to the number of characters read from the console after the operation (not including the terminating carriage-return). The remaining positions are used to hold the characters read from the console. The BIOS line editing functions which are performed during this operation are given below:

break	- line delete and transmit
rubout	- delete last character typed, and echo
control-C	- system rebout
control-U	- delete entire line
control-E	- return carriage, but do not transmit buffer (physical carriage return)
<cr>	- transmit buffer

The read routine also detects control character sequences other than those shown above, and echos them with a preceding "↑" symbol. The print entry point allows an entire string of symbols to be printed before returning from the BIOS. The string is terminated by a "$" symbol.

2.2 A Simple Example

As an example, consider the following PL/M procedures and procedure calls which print a heading, and successively read the console buffer. Each console buffer is then echoed back in reverse order:

```
PRINTCHAR:  PROCEDURE (B);
      /* SEND THE ASCII CHARACTER B TO THE CONSOLE */
      DECLARE B BYTE;
      CALL MON1(2,B);
      END PRINTCHAR;


CRLF:  PROCEDURE;
      /* SEND CARRIAGE-RETURN-LINE-FEED CHARACTERS */
      CALL PRINTCHAR (0DH); CALL PRINTCHAR (0AH);
      END CRLF;


PRINT:  PROCEDURE (A);
      /* PRINT THE BUFFER STARTING AT ADDRESS A */
      DECLARE A ADDRESS;
      CALL MON1(9,A);
      END PRINT;

DECLARE  RDBUFF (130) BYTE;

READ:  PROCEDURE;
      /* READ CONSOLE CHARACTERS INTO 'RDBUFF' */
      RDBUFF=128; /* FIRST BYTE SET TO BUFFER LENGTH */
      CALL MON1(10,.RDBUFF);
      END READ;

DECLARE I BYTE;
CALL CRLF;                  CALL PRINT (.'TYPE INPUT LINES $');
      DO WHILE 1; /* INFINITE LOOP-UNTIL CONTROL-C */
      CALL CRLF; CALL PRINTCHAR ('*'); /* PROMPT WITH '*' */
      CALL READ; I = RDBUFF(1);
             DO WHILE (I := I -1) <> 255;
             CALL PRINTCHAR (RDBUFF(I+2));
             END;
      END;
```

The execution of this program might proceed as follows:

```
TYPE INPUT LINES
*HELLO
OLLEH
*WALL WALLA WASH
HSAW ALLAW ALLAW
*MOM WOW
WOW MOM
*↑C              (system reboot)
```

TABLE I

BASIC I/O OPERATIONS

FUNCTION/ NUMBER	ENTRY PARAMETERS	RETURNED VALUE	TYPICAL CALL
Read Console 1	None	ASCII Character	I = MON2(1,0)
Write Console 2	ASCII Character	None	CALL MON1(2,'A')
Read Reader 3	None	ASCII Character	I = MON2(3,0)
Write Punch 4	ASCII Character	None	CALL MON1(4,'B')
Write List 5	ASCII Character	None	CALL MON1(5,'C')
Get I/O Status 7	None	I/O Status Byte	IOSTAT=MON2(7,0)
Set I/O Status 8	I/O Status Byte	None	CALL MON1(8,IOSTAT)
Print Buffer 9	Address of string terminated by '$'	None	CALL MON1(9, .'PRINT THIS $')

TABLE I (continued)

FUNCTION/ NUMBER	ENTRY PARAMETERS	RETURNED VALUE	TYPICAL CALL
Read Buffer 10	Address of Read Buffer* (See $Note_1$)	Read buffer is filled to maximum length with console characters	CALL MON1(10, .RDBUFF);
Interrogate Console Ready 11	None	Byte value with least significant bit = 1 (true) if console character is ready	I = MON2(11,0)

$Note_1$: Read buffer is a sequence of memory locations of the form:

m	k	c_1	c_2	c_3					c_k			

↑ current buffer length (k)
↑ Maximum buffer length (m)

$Note_2$: The I/O status byte is defined as three fields A,B,C, and D

2b	2b	2b	2b
A	B	C	D
MSB			LSB

requiring two bits each, listed from most significant to least significant bit, which define the current device assignment as follows:

$$\underset{\text{Console}}{D} = \begin{Bmatrix} 0 & \text{TTY} \\ 1 & \text{CRT} \\ 2 & \text{BATCH} \\ 3 & - \end{Bmatrix} \quad \underset{\text{Reader}}{C} = \begin{Bmatrix} 0 & \text{TTY} \\ 1 & \text{FAST READER} \\ 2 & - \\ 3 & - \end{Bmatrix} \quad \underset{\text{Punch}}{B} = \begin{Bmatrix} 0 & \text{TTY} \\ 1 & \text{FAST PUNCH} \\ 2 & - \\ 3 & - \end{Bmatrix} \quad \underset{\text{List}}{A} = \begin{Bmatrix} 0 & \text{TTY} \\ 1 & \text{CRT} \\ 2 & - \\ 3 & - \end{Bmatrix}$$

3. DISK I/O FACILITIES

The BDOS section of CP/M provides access to files stored on diskettes. The discussion which follows gives the overall file organization, along with file access mechanisms.

3.1 File Organization

CP/M implements a named file structure on each diskette, providing a logical organization which allows any particular file to contain any number of records, from completely empty, to the full capacity of a diskette. Each diskette is logically distinct, with a complete operating system, disk directory, and file data area. The disk file names are in two parts: the <filename> which can be from one to eight alphanumeric characters, and the <filetype> which consists of zero through three alphanumeric characters. The <filetype> names the generic category of a particular file, while the <filename> distinguishes a particular file within the category. The <filetype>s listed below give some generic categories which have been established, although they are generally arbitrary:

ASM	assembler source file
PRN	assembler listing file
HEX	assembler or PL/M machine code in "hex" format
BAS	BASIC Source file
INT	BASIC Intermediate file
COM	Memory image file (i.e., "Command" file for transients, produced by LOAD)
BAK	Backup file produced by editor (see ED manual)
$$$	Temporary files created and normally erased by editor and utilities

Thus, the name

X.ASM

is interpreted as an assembly language source file by the CCP with <filename> X.

The files in CP/M are organized as a logically contiguous sequence of 128 byte records (although the records may not be physically contiguous on the diskette), which are normally read or written in sequential order. Random access is allowed under CP/M however, as described in Section 3.4. No particular format within records is assumed by CP/M, although some transients expect particular formats:

(1) Source files are considered a sequence of ASCII characters, where each "line" of the source file is followed by carriage-return-line-feed characters. Thus, one 128 byte CP/M record could contain several logical lines of source text. Machine code "hex" tapes are also assumed to be in this format, although the loader does not require the carriage-return-line-feed characters. End of text is given by the character control-z, or real end-of-file returned by CP/M.

and

(2) COM files are assumed to be absolute machine code in memory image form, starting at tbase in memory. In this case, control-z is not considered an end of file, but instead is determined by the actual space allocated to the file being accessed.

3.2 File Control Block Format

Each file being accessed through CP/M has a corresponding file control block (FCB) which provides name and allocation information for all file operations. The FCB is a 33-byte area in the transient program's memory space which is set up for each file. The FCB format is given in Figure 2. When accessing CP/M files, it is the programmer's responsibility to fill the lower 16 bytes of the FCB, along with the CR field. Normally, the FN and FT fields are set to the ASCII <filename> and <filetype>, while all other fields are set to zero. Each FCB describes up to 16K bytes of a particular file (0 to 128 records of 128 bytes each), and, using automatic mechanisms of CP/M, up to 15 additional extensions of the file can be addressed. Thus, each FCB can potentially describe files up to 256K bytes (which is slightly larger than the diskette capacity).

FCB's are stored in a directory area of the diskette, and are brought into central memory before file operations (see the OPEN and MAKE commands) then updated in memory as file operations proceed, and finally recorded on the diskette at the termination of the file operation (see the CLOSE command). This organization makes CP/M file organization highly reliable, since diskette file integrity can only be disrupted in the unlikely case of hardware failure during update of a single directory entry.

It should be noted that the CCP constructs an FCB for all transients by scanning the remainder of the line following the transient name for a <filename> or <filename>.<filetype> combination. Any field not specified is assumed to be all blanks. A properly formed FCB is set up at location tfcb (see Section 6), with an assumed I/O buffer at tbuff. The transient can use tfcb as an address in subsequent input or output operations on this file.

In addition to the default fcb which is set-up at address tfcb, the CCP also constructs a second default fcb at address tfcb + 16 (i.e., the disk map field of the fcb at tbase). Thus, if the user types

PROGNAME X.ZOT Y.ZAP

the file PROGNAME.COM is loaded to the TPA, and the default fcb at tfcb is initialized to the filename X with filetype ZOT. Since the user typed a second file name, the 16 byte area beginning at tfcb + 16_{10} is also initialized with the filename Y and filetype ZAP. It is the responsibility of the program to move this second filename and filetype to another area (usually a separate file control block) before opening the file which begins at tbase, since the open operation will fill the disk map portion, thus overwriting the second name and type.

If no file names were specified in the original command, then the fields beginning at tfcb and tfcb + 16 both contain blanks (20H). If one file name was specified, then the field at tfcb + 16 contains blanks. If the filetype is omitted, then the field is assumed to contain blanks. In all cases, the CCP translates lower case alphabetics to upper case to be consistent with the CP/M file naming conventions.

As an added programming convenience, the default buffer at tbuff is initialized to hold the entire command line past the program name. Address tbuff contains the number of characters, and tbuff+1, tbuff+2, ..., contain the remaining characters up to, but not including, the carriage return. Given that the above command has been typed at the console, the area beginning at tbuff is set up as follows:

tbuff:

+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+12	+13	+14	+15
12	b̸	X	.	Z	O	T	b̸	Y	.	Z	A	P	?	?	?

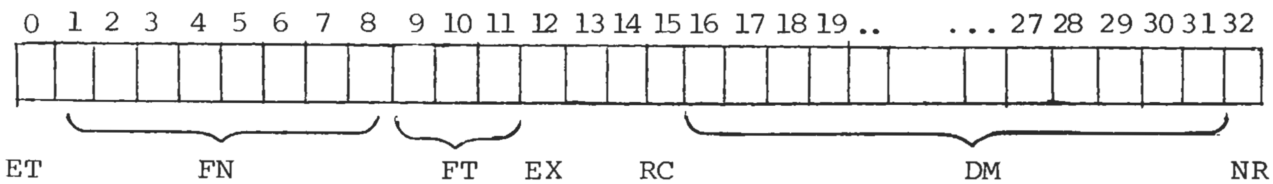
where 12 is the number of valid characters (in binary), and b̸ represents an ASCII blank. Characters are given in ASCII upper case, with uninitialized memory following the last valid character.

Again, it is the responsibility of the program to extract the information from this buffer before any file operations are performed since the FDOS uses the tbuff area to perform directory functions.

In a standard CP/M system, the following values are assumed:

boot:	0000H	bootstrap load (warm start)
entry:	0005H	entry point to FDOS
tfcb:	005CH	first default file control block
tfcb+16	006CH	second file name
tbuff	0080H	default buffer address
tbase:	0100H	base of transient area

Figure 2. File Control Block Format



FIELD	FCB POSITIONS	PURPOSE
ET	0	Entry type (currently not used, but assumed zero)
FN	1-8	File name, padded with ASCII blanks
FT	9-11	File type, padded with ASCII blanks
EX	12	File extent, normally set to zero
	13-14	Not used, but assumed zero
RC	15	Record count is current extent Size (0 to 128 records)
DM	16-31	Disk allocation map, filled-in and used by CP/M
NR	32	Next record number to read or write

3.3 Disk Access Primitives

Given that a program has properly initialized the FCB's for each of its files, there are several operations which can be performed, as shown in Table II. In each case, the operation is applied to the currently selected disk (see the disk select operation in Table II), using the file information in a specific FCB. The following PL/M program segment, for example, copies the contents of the file X.Y to the (new) file NEW.FIL:

```
DECLARE RET BYTE;

OPEN:      PROCEDURE (A)
           DECLARE A ADDRESS;
           RET=MON2(15,A);
           END OPEN;

CLOSE:     PROCEDURE (A);
           DECLARE A ADDRESS;
           RET=MON2(16,A);
           END;

MAKE:      PROCEDURE (A);
           DECLARE A ADDRESS;
           RET=MON2(22,A);
           END MAKE;

DELETE:    PROCEDURE (A);
           DECLARE A ADDRESS;
           /* IGNORE RETURNED VALUE */
           CALL MON1(19,A);
           END DELETE;

READBF:    PROCEDURE (A);
           DECLARE A ADDRESS;
           RET=MON2(20,A);
           END READBF;

WRITEBF:   PROCEDURE (A);
           DECLARE A ADDRESS;
           RET=MON2(21,A);
           END WRITEBF;

INIT:      PROCEDURE;
           CALL MON1(13,0);
           END INIT;

/* SET UP FILE CONTROL BLOCKS */
DECLARE FCB1 (33) BYTE
   INITIAL (0,'X       ','Y  ',0,0,0,0),
   FCB2 (33) BYTE
   INITIAL (0,'NEW     ','FIL',0,0,0,0);
```

```
CALL INIT;
/* ERASE 'NEW.FIL' IF IT EXISTS */
CALL DELETE (.FCB2);
/* CREATE''NEW.FIL' AND CHECK SUCCESS */
CALL MAKE (.FCB2);
IF RET = 255 THEN CALL PRINT (.'NO DIRECTORY SPACE $');
   ELSE
   DO; /* FILE SUCCESSFULLY CREATED, NOW OPEN 'X.Y' */
   CALL OPEN (.FCB1);
   IF RET = 255 THEN CALL PRINT (.'FILE NOT PRESENT $');
      ELSE
      DO; /* FILE X.Y FOUND AND OPENED, SET
      NEXT RECORD TO ZERO FOR BOTH FILES */
      FCB1(32), FCB2(32) = 0;
      /* READ FILE X.Y UNTIL EOF OR ERROR */
      CALL READBF (.FCB1); /*READ TO 80H*/
         DO WHILE RET = 0;
         CALL WRITEBF (.FCB2) /*WRITE FROM 80H*/
         IF RET = 0 THEN /*GET ANOTHER RECORD*/
            CALL READBF (.FCB1); ELSE
            CALL PRINT (.'DISK WRITE ERROR $');
         END;
      IF RET < >1 THEN CALL PRINT (.' TRANSFER ERROR $');
      ELSE
         DO; CALL CLOSE (.FCB2);
         IF RET = 255 THEN CALL PRINT (.'CLOSE ERROR$');
         END;
      END;
   END;
EOF
```

This program consists of a number of utility procedures for opening, closing, creating, and deleting files, as well as two procedures for reading and writing data. These utility procedures are followed by two FCB's for the input and output files. In both cases, the first 16 bytes are initialized to the <filename> and <filetype> of the input and output files. The main program first initializes the disk system, then deletes any existing copy of "NEW.FIL" before starting. The next step is to create a new directory entry (and empty file) for "NEW.FIL". If file creation is successful, the input file "X.Y" is opened. If this second operation is also successful, then the disk to disk copy can proceed. The NR fields are set to zero so that the first record of each file is accessed on subsequent disk I/O operations. The first call to READBF fills the (implied) DMA buffer at 80H with the first record from X.Y. The loop which follows copies the record at 80H to "NEW.FIL" and then reports any errors, or reads another 128 bytes from X.Y. This transfer operation continues until either all data has been transferred, or an error condition arises. If an error occurs, it is reported; otherwise the new file is closed and the program halts.

TABLE II

DISK ACCESS PRIMITIVES

FUNCTION/NUMBER	ENTRY PARAMETERS	RETURNED VALUE	TYPICAL CALL
Lift Head 12	None	None Head is lifted from current drive	CALL MON2(12,0)
Initialize BDOS and select disk "A" Set DMA address to 80H 13	None	None Side effect is that disk A is "logged-in" while all others are considered "off-line"	CALL MON1(13,0)
Log-in and select disk X 14	An integer value corresponding to the disk to log-in: A=0, B=1, C=2, etc.	None Disk X is considered "on-line" and selected for subsequent file operations	CALL MON1(14,1) (log-in disk "B")
Open file 15	Address of the FCB for the file to be accessed	Byte address of the FCB in the directory, if found, or 255 if file not present. The DM bytes are set by the BDOS.	I = MON2(15,.FCB)
Close file 16	Address of an FCB which has been previously created or opened	Byte address of the directory entry corresponding to the FCB, or 255 if not present	I = MON2(16,.FCB)

TABLE II (continued)

FUNCTION/NUMBER	ENTRY PARAMETERS	RETURNED VALUE	TYPICAL CALL
Search for file 17	Address of FCB containing <filename> and <filetype> to match. ASCII "?" in FCB matches any character.	Byte address of first FCB in directory that matches input FCB, if any; otherwise 255 indicates no match.	I = MON2(17,.FCB)
Search for next occurrence 18	Same as above, but called after function 17 (no other intermediate BDOS calls allowed)	Byte address of next	I = MON2(18,.FCB)
Delete File 19	Address of FCB containing <filename> and <filetype> of file to delete from diskette	None	I = MON2(19,.FCB)
Read Next Record 20	Address of FCB of a successfully OPENed file, with NR set to the next record to read (see $note_1$)	0 = successful read 1 = read past end of file 2 = reading unwritten data in random access	I = MON2(20,.FCB)

$Note_1$: The I/O operations transfer data to/from address 80H for the next 128 bytes unless the DMA address has been altered (see function 26). Further, the NR field of the FCB is automatically incremented after the operation. If the NR field exceeds 128, the next extent is opened automatically, and the NR field is reset to zero.

TABLE II (continued)

FUNCTION/NUMBER	ENTRY PARAMETERS	RETURNED VALUE	TYPICAL CALL
Write Next Record 21	Same as above, except NR is set to the next record to write	0 = successful write 1 = error in extending file 2 = end of disk data 255 = no more directory space (see $note_2$)	I = MON2(21,.FCB)
Make File 22	Address of FCB with <filename> and <filetype> set. Directory entry is created, the file is initialized to empty.	Byte address of directory entry allocated to the FCB, or 255 if no directory space is available	I = MON2(22,.FCB)
Rename FCB 23	Address of FCB with old FN and FT in first 16 bytes, and new FN and FT in second 16 bytes	Address of the directory entry which matches the first 16 bytes. The <filename>and <filetype> is altered 255 if no match.	I = MON2(2 3,.FCB)

$Note_2$: There are normally 64 directory entries available on each diskette (can be expanded to 255 entries), where one entry is required for the primary file, and one for each additional extent.

TABLE II (continued)

FUNCTION/NUMBER	ENTRY PARAMETERS	RETURNED VALUE	TYPICAL CALL
Interrogate log-in vector 24	None	Byte value with "1" in bit positions of "on line" disks, with least significant bit corresponding to disk "A"	I = MON2(24,0)
Set DMA address 26	Address of 128 byte DMA buffer	None Subsequent disk I/O takes place at specified address in memory	CALL MON1(26,2000H)
Interrogate Allocation 27	None	Address of the allocation vector for the current disk (used by STATUS command)	MON3: PROCEDURE(...) ADDRESS; A = MON3(27,0);
Interrogate Drive number 25	None	Disk number of currently logged disk (i.e., the drive which will be used for the next disk operation	I = MON2(25,0);

3.4 Random Access

Recall that a single FCB describes up to a 16K segment of a (possibly) larger file. Random access within the first 16K segment is accomplished by setting the NR field to the record number of the record to be accessed before the disk I/O takes place. Note, however, that if the 128th record is written, then the BDOS automatically increments the extent field (EX), and opens the next extent, if possible. In this case, the program must explicitly decrement the EX field and re-open the previous extent. If random access outside the first 16K segment is necessary, then the extent number e be explicitly computed, given an absolute record number r as

$$e = \left\lfloor \frac{r}{128} \right\rfloor$$

or equivalently,

$$e = SHR(r,7)$$

this extent number is then placed in the EX field before the segment is opened. The NR value n is then computed as

$$n = r \bmod 128$$

or

$$n = r \text{ AND } 7FH.$$

When the programmer expects considerable cross-segment accesses, it may save time to create an FCB for each of the 16K segments, open all segments for access, and compute the relevant FCB from the absolute record number r.

4. SYSTEM GENERATION

As mentioned previously, every diskette used under CP/M is assumed to contain the entire system (excluding transient commands) on the first two tracks. The operating system need not be present, however, if the diskette is only used as secondary disk storage on drives B, C, ..., since the CP/M system is loaded only from drive A.

The CP/M file system is organized so that an IBM-compatible diskette from the factory (or from a vendor which claims IBM compatibility) looks like a diskette with an empty directory. Thus, the user must first copy a version of the CP/M system from an existing diskette to the first two tracks of the new diskette, followed by a sequence of copy operations, using PIP, which transfer the transient command files from the original diskette to the new diskette.

NOTE: before you begin the CP/M copy operation, read your Licensing Agreement. It gives your exact legal obligations when making reproductions of CP/M in whole or in part, and specifically requires that you place the copyright notice

Copyright (c), 1976
Digital Research

on each diskette which results from the copy operation.

4.1. Initializing CP/M from an Existing Diskette

The first two tracks are placed on a new diskette by running the transient command SYSGEN, as described in the document "An Introduction to CP/M Features and Facilities." The SYSGEN operation brings the CP/M system from an initialized diskette into memory, and then takes the memory image and places it on the new diskette.

Upon completion of the SYSGEN operation, place the original diskette on drive A, and the initialized diskette on drive B. Reboot the system; the response should be

```
A>
```

indicating that drive A is active. Log into drive B by typing

```
B:
```

and CP/M should respond with

```
B>
```

indicating that drive B is active. If the diskette in drive B is factory fresh, it will contain an empty directory. Non-standard diskettes may, however, appear as full directories to CP/M, which can be emptied by typing

```
ERA *.*
```

when the diskette to be initialized is active. Do not give the ERA command if you wish to preserve files on the new diskette since all files will be erased with this command.

After examining disk B, reboot the CP/M system and return to drive A for further operations.

The transient commands are then copied from drive A to drive B using the PIP program. The sequence of commands shown below, for example, copy the principal programs from a standard CP/M diskette to the new diskette:

```
A>PIP
*B:STAT.COM=STAT.COM
*B:PIP.COM=PIP.COM
*B:LOAD.COM=LOAD.COM
*B:ED.COM=ED.COM
```

```
*B:ASM.COM=ASM.COM↵
*B:SYSGEN.COM=SYSGEN.COM↵
*B:DDT.COM=DDT.COM↵
*↵
A>
```

The user should then log in disk B, and type the command

```
DIR *.*↵
```

to ensure that the files were transferred to drive B from drive A. The various programs can then be tested on drive B to check that they were transferred properly.

Note that the copy operation can be simplified somewhat by creating a "submit" file which contains the copy commands. The file could be named GEN.SUB, for example, and might contain

```
SYSGEN↵
PIP B:STAT.COM=STAT.COM↵
PIP B:PIP.COM=PIP.COM↵
PIP B:LOAD.COM=LOAD.COM↵
PIP B:ED.COM=ED.COM↵
PIP B:ASM.COM=ASM.COM↵
PIP B:SYSGEN.COM=SYSGEN.COM↵
PIP B:DDT.COM=DDT.COM↵
```

The generation of a new diskette from the standard diskette is then done by typing simply

```
SUBMIT GEN↵
```

5. CP/M ENTRY POINT SUMMARY

The functions shown below summarize the functions of the FDOS. The function number is passed in Register C (first parameter in PL/M), and the information is passed in Registers D,E (second PL/M parameter). Single byte results are returned in Register A. If a double byte result is returned, then the high-order byte comes back in Register B (normal PL/M return). The transient program enters the FDOS through location "entry" (see Section 7.) as shown in Section 2. for PL/M, or

```
CALL entry
```

in assembly language. All registers are altered in the FDOS.

Function	Number	Information	Result
0	System Reset		
1	Read Console		ASCII character
2	Write Console	ASCII character	
3	Read Reader		ASCII character
4	Write Punch	ASCII character	
5	Write List	ASCII character	
6	(not used)		
7	Interrogate I/O Status		I/O Status Byte
8	Alter I/O Status	I/O Status Byte	
9	Print Console Buffer	Buffer Address	
10	Read Console Buffer	Buffer Address	
11	Check Console Status		True if character Ready
12	Lift Disk Head		
13	Reset Disk System		
14	Select Disk	Disk number	
15	Open File	FCB Address	Completion Code
16	Close File	" "	" "
17	Search First	" "	" "
18	Search Next	" "	" "
19	Delete File	" "	" "
20	Read Record	" "	" "
21	Write Record	" "	" "
22	Create File	" "	" "
23	Rename File	" "	" "
24	Interrogate Login		Login Vector
25	Interrogate Disk		Selected Disk Number
26	Set DMA Address	DMA Address	
27	Interrogate Allocation		Address of Allocation Vector

6. ADDRESS ASSIGNMENTS

The standard distribution version of CP/M is organized for an Intel MDS microcomputer developmental system with 16K of main memory, and two diskette drives. Larger systems are available in 16K increments, providing management of 32K, 48K, and 64K systems (the largest MDS system is 62K since the ROM monitor provided with the MDS resides in the top 2K of the memory space). For each additional 16K increment, add 4000H to the values of cbase and fbase.

The address assignments are

boot = 0000H	warm start operation
tfcb = 005CH	default file control block location
tbuff= 0080H	default buffer location
tbase= 0100H	base of transient program area
cbase= 2900H	base of console command processor
fbase= 3200H	base of disk operating system
entry= 0005H	entry point to disk system from user programs

7. SAMPLE PROGRAMS

This section contains two sample programs which interface with the CP/M operating system. The first program is written in assembly language, and is the source program for the DUMP utility. The second program is the CP/M LOAD utility, written in PL/M.

The assembly language program begins with a number of "equates" for system entry points and program constants. The equate

```
BDOS    EQU    0005H
```

for example, gives the CP/M entry point for peripheral I/O functions. The defualt file control block address is also defined (FCB), along with the default buffer address (BUFF). Note that the program is set up to run at location 100H, which is the base of the transient program area. The stack is first set-up by saving the entry stack pointer into OLDSP, and resetting SP to the local stack. The stack pointer upon entry belongs to the console command processor, and need not be saved unless control is to return to the CCP upon exit. That is, if the program terminates with a reboot (branch to location 0000H) then the entry stack pointer need not be saved.

The program then jumps to MAIN, past a number of subroutines which are listed below:

BREAK - when called, checks to see if there is a console character ready. BREAK is used to stop the listing at the console

PCHAR - print the character which is in register A at the console.

CRLF - send carriage return and line feed to the console

PNIB - print the hexadecimal value in register A in ASCII at the console

PHEX - print the byte value (two ASCII characters) in register A at the console

ERR - print error flag #n at the console, where n is

1 if file cannot be opened
2 if disk read error occurred

GNB - get next byte of data from the input file. If the IBP (input buffer pointer) exceeds the size of the input buffer, then another disk record of 128 bytes is read. Otherwise, the next character in the buffer is returned. IBP is updated to point to the next character

The MAIN program then appears, which begins by calling SETUP. The SETUP subroutine, discussed below, opens the input file and checks for errors. If the file is opened properly, the GLOOP (get loop) label gets control.

On each successive pass through the GLOOP label, the next data byte is fetched using GNB and save in register B. The line addresses are listed every sixteen bytes, so there must be a check to see if the least significant 4 bits is zero on each output. If so, the line address is taken from registers h and l, and typed at the left of the line. In all cases, the byte which was previously saved in register B is brought back to register A, following label NONUM, and printed in the output line. The cycle through GLOOP continues until an end of file condition is detected in DISKR, as described below. Thus, the output lines appear as

```
0000  bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
0010  bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
          ...
```

until the end of file.

The label FINIS gets control upon end of file. CRLF is called first to return the carriage from the last line output. The CCP stack pointer is then reclaimed from OLDSP, followed by a RET to return to the console command processor. Note that a JMP 0000H could be used following the FINIS label, which would cause the CP/M system to be brought in again from the diskette (this operation is necessary only if the CCP has been overlayed by data areas).

The file control block format is then listed (FCBDN ... FCBLN) which overlays the fcb at location 05CH which is setup by the CCP when the DUMP program is initiated. That is, if the user types

```
DUMP  X.Y
```

then the CCP sets up a properly formed fcb at location 05CH for the DUMP (or any other) program when it goes into execution. Thus, the SETUP subroutine simply addresses this default fcb, and calls the disk system to open it. The DISKR (disk read) routine is called whenever GNB needs another buffer full of data. The default buffer at location 80H is used, along with a pointer (IBP) which counts bytes as they are processed. Normally, an end of file condition is taken as either an ASCII 1AH (control-z), or an end of file detection by the DOS. The file dump program, however, stops only on a DOS end of file.

```
                ;       FILE DUMP PROGRAM, READS AN INPUT FILE AND PRINTS IN HEX
                ;
                ;       COPYRIGHT (C), DIGITAL RESEARCH, 1975, 1976
                ;
Ø1ØØ                    ORG     1ØØH
ØØØ5 =          BDOS    EQU     ØØØ5H   ;DOS ENTRY POINT
ØØØF =          OPENF   EQU     15      ;FILE OPEN
ØØ14 =          READF   EQU     2Ø      ;READ FUNCTION
ØØØ2 =          TYPEF   EQU     2       ;TYPE FUNCTION
ØØØ1 =          CONS    EQU     1       ;READ CONSOLE
ØØØB =          BRKF    EQU     11      ;BREAK KEY FUNCTION (TRUE IF CHAR READY)
                ;
ØØ5C =          FCB     EQU     5CH     ;FILE CONTROL BLOCK ADDRESS
ØØ8Ø =          BUFF    EQU     8ØH     ;INPUT DISK BUFFER ADDRESS
                ;
                ;       SET UP STACK
Ø1ØØ 21ØØØØ             LXI     H,Ø
Ø1Ø3 39                 DAD     SP
Ø1Ø4 22ØFØ1             SHLD    OLDSP
Ø1Ø7 3151Ø1             LXI     SP,STKTOP
Ø1ØA C3C4Ø1             JMP     MAIN
                ;       VARIABLES
Ø1ØD            IBP:    DS      2       ;INPUT BUFFER POINTER
                ;
                ;       STACK AREA
Ø1ØF            OLDSP:  DS      2
Ø111            STACK:  DS      64
Ø151 =          STKTOP  EQU     $
                ;
                ;       SUBROUTINES
                ;
                BREAK:  ;CHECK BREAK KEY (ACTUALLY ANY KEY WILL DO)
Ø151 E5D5C5             PUSH H! PUSH D! PUSH B; ENVIRONMENT SAVED
Ø154 ØEØB               MVI     C,BRKF
Ø156 CDØ5ØØ             CALL    BDOS
Ø159 C1D1E1             POP B! POP D! POP H; ENVIRONMENT RESTORED
Ø15C C9                 RET
                ;
                PCHAR:  ;PRINT A CHARACTER
Ø15D E5D5C5             PUSH H! PUSH D! PUSH B; SAVED
Ø16Ø ØEØ2               MVI     C,TYPEF
Ø162 5F                 MOV     E,A
Ø163 CDØ5ØØ             CALL    BDOS
Ø166 C1D1E1             POP B! POP D! POP H; RESTORED
Ø169 C9                 RET
                ;
                CRLF:
Ø16A 3EØD               MVI     A,ØDH
Ø16C CD5DØ1             CALL    PCHAR
Ø16F 3EØA               MVI     A,ØAH
Ø171 CD5DØ1             CALL    PCHAR
Ø174 C9                 RET
                ;
                ;
                PNIB:   ;PRINT NIBBLE IN REG A
Ø175 E6ØF               ANI     ØFH     ;LOW 4 BITS
Ø177 FEØA               CPI     1Ø
Ø179 D281Ø1             JNC     P1Ø
```

```
                    ;       LESS THAN OR EQUAL TO 9
017C C630                   ADI     '0'
017E C38301                 JMP     PRN
                    ;
                    ;       GREATER OR EQUAL TO 10
0181 C637           P10:    ADI     'A' - 10
0183 CD5D01         PRN:    CALL    PCHAR
0186 C9                     RET
                    ;
                    PHEX:   ;PRINT HEX CHAR IN REG A
0187 F5                     PUSH    PSW
0188 0F                     RRC
0189 0F                     RRC
018A 0F                     RRC
018B 0F                     RRC
018C CD7501                 CALL    PNIB    ;PRINT NIBBLE
018F F1                     POP     PSW
0190 CD7501                 CALL    PNIB
0193 C9                     RET
                    ;
                    ERR:    ;PRINT ERROR MESSAGE
0194 CD6A01                 CALL    CRLF
0197 3E23                   MVI     A,'#'
0199 CD5D01                 CALL    PCHAR
019C 78                     MOV     A,B
019D C630                   ADI     '0'
019F CD5D01                 CALL    PCHAR
01A2 CD6A01                 CALL    CRLF
01A5 C3F701                 JMP     FINIS
                    ;
                    GNB:    ;GET NEXT BYTE
01A8 3A0D01                 LDA     IBP
01AB FE80                   CPI     80H
01AD C2B401                 JNZ     G0
                    ;       READ ANOTHER BUFFER
                    ;
                    ;
01B0 CD1602                 CALL    DISKR
01B3 AF                     XRA     A
                    ;
                    G0:     ;READ THE BYTE AT BUFF+REG A
01B4 5F                     MOV     E,A
01B5 1600                   MVI     D,0
01B7 3C                     INR     A
01B8 320D01                 STA     IBP
                    ;       POINTER IS INCREMENTED
                    ;       SAVE THE CURRENT FILE ADDRESS
01BB E5                     PUSH    H
01BC 218000                 LXI     H,BUFF
01BF 19                     DAD     D
01C0 7E                     MOV     A,M
                    ;       BYTE IS IN THE ACCUMULATOR
                    ;
                    ;       RESTORE FILE ADDRESS AND INCREMENT
01C1 E1                     POP     H
01C2 23                     INX     H
01C3 C9                     RET
                    ;
                    MAIN:   ; READ AND PRINT SUCCESSIVE BUFFERS
01C4 CDFF01                 CALL    SETUP   ;SET UP INPUT FILE
```

```
01C7 3E80               MVI     A,80H
01C9 320D01             STA     IBP     ;SET BUFFER POINTER TO 80H
01CC 21FFFF             LXI     H,0FFFFH        ;SET TO -1 TO START
                ;
                GLOOP:
01CF CDA801             CALL    GNB
01D2 47                 MOV     B,A
                ;       PRINT HEX VALUES

                ;       CHECK FOR LINE FOLD
01D3 7D                 MOV     A,L
01D4 E60F               ANI     0FH     ;CHECK LOW 4 BITS
01D6 C2EB01             JNZ     NONUM
                ;       PRINT LINE NUMBER
01D9 CD6A01             CALL    CRLF
                ;
                ;       CHECK FOR BREAK KEY
01DC CD5101             CALL    BREAK
01DF 0F                 RRC
01E0 DAF701             JC      FINIS   ;DON´T PRINT ANY MORE
                ;
01E3 7C                 MOV     A,H
01E4 CD8701             CALL    PHEX
01E7 7D                 MOV     A,L
01E8 CD8701             CALL    PHEX
                NONUM:
01EB 3E20               MVI     A,´ ´
01ED CD5D01             CALL    PCHAR
01F0 78                 MOV     A,B
01F1 CD8701             CALL    PHEX

01F4 C3CF01             JMP     GLOOP
                ;
                EPSA:   ;END PSA
                        ;       END OF INPUT
                FINIS:
01F7 CD6A01             CALL    CRLF
01FA 2A0F01             LHLD    OLDSP
01FD F9                 SPHL
01FE C9                 RET
                ;
                ;
                ;       FILE CONTROL BLOCK DEFINITIONS
005C =          FCBDN   EQU     FCB+0   ;DISK NAME
005D =          FCBFN   EQU     FCB+1   ;FILE NAME
0065 =          FCBFT   EQU     FCB+9   ;DISK FILE TYPE (3 CHARACTERS)
0068 =          FCBRL   EQU     FCB+12  ;FILE´S CURRENT REEL NUMBER
006B =          FCBRC   EQU     FCB+15  ;FILE´S RECORD COUNT (0 TO 128)
007C =          FCBCR   EQU     FCB+32  ;CURRENT (NEXT) RECORD NUMBER (0 TO 127)
007D =          FCBLN   EQU     FCB+33  ;FCB LENGTH
                ;
                ;
                SETUP:  ;SET UP FILE
                ;       OPEN THE FILE FOR INPUT
01FF 115C00             LXI     D,FCB
0202 0E0F               MVI     C,OPENF
0204 CD0500             CALL    BDOS
                ;       CHECK FOR ERRORS
0207 FEFF               CPI     255
0209 C21102             JNZ     OPNOK
```

```
                    ;       BAD OPEN
020C 0601                   MVI     B,1         ;OPEN ERROR
020E CD9401                 CALL    ERR
                    ;
                    OPNOK: ;OPEN IS OK.
0211 AF                     XRA     A
0212 327C00                 STA     FCBCR
0215 C9                     RET
                    ;
                    DISKR: ;READ DISK FILE RECORD
0216 E5D5C5                 PUSH H! PUSH D! PUSH B
0219 115C00                 LXI     D,FCB
021C 0E14                   MVI     C,READF
021E CD0500                 CALL    BDOS
0221 C1D1E1                 POP B! POP D! POP H
0224 FE00                   CPI     0           ;CHECK FOR ERRS
0226 C8                     RZ
                    ;       MAY BE EOF
0227 FE01                   CPI     1
0229 CAF701                 JZ      FINIS
                    ;
022C 0602                   MVI     B,2         ;DISK READ ERROR
022E CD9401                 CALL    ERR
                    ;
0231                        END
```

The PL/M program which follows implements the CP/M LOAD utility. The function is as follows. The user types

LOAD filename

If filename.HEX exists on the diskette, then the LOAD utility reads the "hex" formatted machine code file and produces the file

filename.COM

where the COM file contains an absolute memory image of the machine code, ready for load and execution in the TPA. If the file does not appear on the diskette, the LOAD program types

SOURCE IS READER

and reads an Addmaster paper tape reader which contains the hex file.

The LOAD program is set up to load and run in the TPA, and, upon completion, return to the CCP without rebooting the system. Thus, the program is constructed as a single procedure called LOADCOM which takes the form

```
0FAH:
   LOADCOM: PROCEDURE;
        /* LIBRARY PROCEDURES */
        MON1: ...
        /* END LIBRARY PROCEDURES */
        MOVE: ...
        GETCHAR: ...
        PRINTNIB: ...
        PRINTHEX: ...
        PRINTADDR: ...
        RELOC: ...
           SETMEM:
           READHEX:
           READBYTE:
           READCS:
           MAKEDOUBLE:
           DIAGNOSE:
        END RELOC;

        DECLARE STACK(16) ADDRESS, SP ADDRESS;
        SP = STACKPTR; STACKPTR = .STACK(LENGTH(STACK));

        ...
        CALL RELOC;
        ...
        STACKPTR = SP;
        RETURN 0;
   END LOADCOM;
   ;
   EOF
```

The label OFAH at the beginning sets the origin of the compilation to OFAH, which causes the first 6 bytes of the compilation to be ignored when loaded (i.e., the TPA starts at location 100H and thus OFAH,...,OFFH are deleted from the COM file). In a PL/M compilation, these 6 bytes are used to set up the stack pointer and branch around the subroutines in the program. In this case, there is only one subroutine, called LOADCOM, which results in the following machine memory image for LOAD

```
OFAH: LXI SP,plmstack     ;SET SP TO DEFAULT STACK
OFDH: JMP pastsubr        ;JUMP AROUND LOADCOM
100H: beginning of LOADCOM procedure
      ....
      end of LOADCOM procedure
      RET

pastsubr:
      EI
      HLT
```

Since the machine code between OFAH and OFFH is deleted in the load, execution actually begins at the top of LOADCOM. Note, however, that the initialization of the SP to the default stack has also been deleted; thus, there is a declaration and initialization of an explicit stack and stack pointer before the call to RELOC at the end of LOADCOM. This is necessary only if we wish to return to the CCP without a reboot operation: otherwise the origin of the program is set to 100H, the declaration of LOADCOM as a procedure is not necessary, and termination is accomplished by simply executing a

```
GO TO 0000H;
```

at the end of the program. Note also that the overhead for a system reboot is not great (approximately 2 seconds), but can be bothersome for system utilities which are used quite often, and do not need the extra space.

The procedures listed in LOADCOM as "library procedures" are a standard set of PL/M subroutines which are useful for CP/M interface. The RELOC procedure contains several nested subroutines for local functions, and actually performs the load operation when called from LOADCOM. Control initially starts on line 327 where the stackpointer is saved and re-initialized to the local stack. The default file control block name is copied to another file control block (SFCB) since two files may be open at the same time. The program then calls SEARCH to see if the HEX file exists; if not, then the high speed reader is used. If the file does exist, it is opened for input (if possible). The filetype COM is moved to the default file control block area, and any existing copies of filename.COM files are removed from the diskette before creating a new file. The MAKE operation creates a new file, and, if successful, RELOC is called to read the HEX file and produce the COM file. At the end of processing by RELOC, the COM file is closed (line 350). Note that the HEX file does not need to be closed since it was opened for input only. The data written to a file is not permanently recorded until the file is successfully closed.

Disk input characters are read through the procedure GETCHAR on line 137. Although the DMA facilities of CP/M could be used here, the GETCHAR procedure instead uses the default buffer at location 80H and moves each buffer into a vector called SBUFF (source buffer) as it is read. On exit, the GETCHAR procedure returns the next input character and updates the source buffer pointer (SBP).

The SETMEM procedure on line 191 performs the opposite function from GETCHAR. The SETMEM procedure maintains a buffer of loaded machine code in pure binary form which acts as a "window" on the loaded code. If there is an attempt by RELOC to write below this window, then the data is ignored. If the data is within the window, then it is placed into MBUFF (memory buffer). If the data is to be placed above this window, then the window is moved up to the point where it would include the data address by writing the memory image successively (by 128 byte buffers), and moving the base address of the window. Using this technique, the programmer can recover from checksum errors on the high-speed reader by stopping the reader, rewinding the tape for some distance, then restarting LOAD (in this case, LOADing is resumed by interrupting with a NOP instruction). Again, the SETMEM procedure uses the default buffer at location 80H to perform the disk output by moving 128 byte segments to 80H through 0FFH before each write.

```
1
1   0FAH: DECLARE BDOS LITERALLY '0005H';
1     /* TRANSIENT COMMAND LOADER PROGRAM
1
1                COPYRIGHT (C) DIGITAL RESEARCH
1                        JUNE, 1975
1    */
1
1   LOADCOM: PROCEDURE BYTE;
2       DECLARE FCBA ADDRESS INITIAL(5CH);
2       DECLARE FCB BASED FCBA (33) BYTE;
2
2       DECLARE BUFFA ADDRESS INITIAL(80H), /* I/O BUFFER ADDR
ESS */
2           BUFFER BASED BUFFA (128) BYTE;
2
2       DECLARE SFCB(33) BYTE,  /* SOURCE FILE CONTROL BLOCK *
/
2           BSIZE LITERALLY '1024',
2           EOFILE LITERALLY '1AH',
2           SBUFF(BSIZE) BYTE  /* SOURCE FILE BUFFER */
2               INITIAL(EOFILE),
2           RFLAG BYTE,          /* READER FLAG */
2           SBP ADDRESS;         /* SOURCE FILE BUFFER POINTER
*/
2
2       /* LOADCOM LOADS TRANSIENT COMMAND FILES TO THE DISK F
ROM THE
2       CURRENTLY DEFINED READER PERIPHERAL.  THE LOADER PLACE
S THE MACH
2       CODE INTO A FILE WHICH APPEARS IN THE LOADCOM COMMAND
*/
2    /* **************** LIBRARY PROCEDURES FOR DISKIO *******
****** */
2
2   MON1: PROCEDURE(F,A);
3       DECLARE F BYTE,
3       A ADDRESS;
3       GO TO BDOS;
3       END MON1;
2
2   MON2: PROCEDURE(F,A) BYTE;
3       DECLARE F BYTE,
3       A ADDRESS;
3       GO TO BDOS;
3       END MON2;
2
2   READRDR: PROCEDURE BYTE;
3       /* READ CURRENT READER DEVICE */
3       RETURN MON2(3,0);
3       END READRDR;
2
2   DECLARE
2       TRUE LITERALLY '1',
2       FALSE LITERALLY '0',
2       FOREVER LITERALLY 'WHILE TRUE',
2       CR LITERALLY '13',
```

```
2        LF LITERALLY '10',
2        WHAT LITERALLY '63';
2
2    PRINTCHAR: PROCEDURE(CHAR);
3        DECLARE CHAR BYTE;
3        CALL MON1(2,CHAR);
3        END PRINTCHAR;
2
2    CRLF: PROCEDURE;
3        CALL PRINTCHAR(CR);
3        CALL PRINTCHAR(LF);
3        END CRLF;
2
2    PRINT: PROCEDURE(A);
3        DECLARE A ADDRESS;
3        /* PRINT THE STRING STARTING AT ADDRESS A UNTIL THE
3        NEXT DOLLAR SIGN IS ENCOUNTERED */
3        CALL CRLF;
3        CALL MON1(9,A);
3        END PRINT;
2
2    DECLARE DCNT BYTE;
2
2    INITIALIZE: PROCEDURE;
3        CALL MON1(13,0);
3        END INITIALIZE;
2
2    SELECT: PROCEDURE(D);
3        DECLARE D BYTE;
3        CALL MON1(14,D);
3        END SELECT;
2
2    OPEN: PROCEDURE(FCB);
3        DECLARE FCB ADDRESS;
3        DCNT = MON2(15,FCB);
3        END OPEN;
2
2    CLOSE: PROCEDURE(FCB);
3        DECLARE FCB ADDRESS;
3        DCNT = MON2(16,FCB);
3        END CLOSE;
2
2    SEARCH: PROCEDURE(FCB);
3        DECLARE FCB ADDRESS;
3        DCNT = MON2(17,FCB);
3        END SEARCH;
2
2    SEARCHN: PROCEDURE;
3        DCNT = MON2(18,0);
3        END SEARCHN;
2
2    DELETE: PROCEDURE(FCB);
3        DECLARE FCB ADDRESS;
3        CALL MON1(19,FCB);
3        END DELETE;
2
2    DISKREAD: PROCEDURE(FCB) BYTE;
3        DECLARE FCB ADDRESS;
3        RETURN MON2(20,FCB);
3        END DISKREAD;
```

```
2
2   DISKWRITE: PROCEDURE(FCB) BYTE;
3       DECLARE FCB ADDRESS;
3       RETURN MON2(21,FCB);
3       END DISKWRITE;
2
2   MAKE: PROCEDURE(FCB);
3       DECLARE FCB ADDRESS;
3       DCNT = MON2(22,FCB);
3       END MAKE;
2
2   RENAME: PROCEDURE(FCB);
3       DECLARE FCB ADDRESS;
3       CALL MON1(23,FCB);
3       END RENAME;
2
2    /* ****************** END OF LIBRARY PROCEDURES ********
***** */
2
2       MOVE: PROCEDURE(S,D,N);
3           DECLARE (S,D) ADDRESS, N BYTE,
3           A BASED S BYTE, B BASED D BYTE;
3               DO WHILE (N:=N-1) <> 255;
3               B = A; S=S+1; D=D+1;
4               END;
3           END MOVE;
2
2       GETCHAR: PROCEDURE BYTE;
3           /* GET NEXT CHARACTER */
3           DECLARE I BYTE;
3           IF RFLAG THEN RETURN READRDR;
3           IF (SBP := SBP+1) <= LAST(SBUFF) THEN
3               RETURN SBUFF(SBP);
3           /* OTHERWISE READ ANOTHER BUFFER FULL */
3               DO SBP = 0 TO LAST(SBUFF) BY 128;
3               IF (I:=DISKREAD(.SFCB)) = 0 THEN
4                   CALL MOVE(80H,.SBUFF(SBP),80H); ELSE
4                   DO; IF I<>1 THEN CALL PRINT(.'DISK READ ER
ROR$');
5                   SBUFF(SBP) = EOFILE;
5                   SBP = LAST(SBUFF);
5                   END;
4               END;
3           SBP = 0; RETURN SBUFF;
3           END GETCHAR;
2   DECLARE
2       STACKPOINTER LITERALLY 'STACKPTR';
2
2
2   PRINTNIB: PROCEDURE(N);
3       DECLARE N BYTE;
3           IF N > 9 THEN CALL PRINTCHAR(N+'A'-10); ELSE
3           CALL PRINTCHAR(N+'0');
3       END PRINTNIB;
2
2   PRINTHEX: PROCEDURE(B);
3       DECLARE B BYTE;
3           CALL PRINTNIB(SHR(B,4)); CALL PRINTNIB(B AND 0FH);
3       END PRINTHEX;
2
```

```
2   PRINTADDR: PROCEDURE(A);
3       DECLARE A ADDRESS;
3       CALL PRINTHEX(HIGH(A)); CALL PRINTHEX(LOW(A));
3       END PRINTADDR;
2
2
2    /* INTEL HEX FORMAT LOADER */
2
2   RELOC: PROCEDURE;
3       DECLARE (RL, CS, RT) BYTE;
3       DECLARE
3           LA ADDRESS,       /* LOAD ADDRESS */
3           TA ADDRESS,       /* TEMP ADDRESS */
3           SA ADDRESS,       /* START ADDRESS */
3           FA ADDRESS,       /* FINAL ADDRESS */
3           NB ADDRESS,       /* NUMBER OF BYTES LOADED */
3           SP ADDRESS,       /* STACK POINTER UPON ENTRY TO REL
OC */
3
3       MBUFF(256) BYTE,
3       P BYTE,
3       L ADDRESS;
3
3       SETMEM: PROCEDURE(B);
4           /* SET MBUFF TO B AT LOCATION LA MOD LENGTH(MBUFF)
*/
4           DECLARE (B,I) BYTE;
4           IF LA < L THEN /* MAY BE A RETRY */ RETURN;
4               DO WHILE LA > L + LAST(MBUFF); /* WRITE A PARA
GRAPH */
4                   DO I = 0 TO 127; /* COPY INTO BUFFER */
5                   BUFFER(I) = MBUFF(LOW(L)); L = L + 1;
6                   END;
5               /* WRITE BUFFER ONTO DISK */
5               P = P + 1;
5               IF DISKWRITE(FCBA) <> 0 THEN
5                   DO; CALL PRINT(.'DISK WRITE ERROR$');
6                   HALT;
6                   /* RETRY AFTER INTERRUPT NOP */
6                   L = L - 128;
6                   END;
5               END;
4           MBUFF(LOW(LA)) = B;
4           END SETMEM;
3
3       READHEX: PROCEDURE BYTE;
4           /* READ ONE HEX CHARACTER FROM THE INPUT */
4           DECLARE H BYTE;
4           IF (H := GETCHAR) - '0' <= 9 THEN RETURN H - '0';
4           IF H - 'A' > 5 THEN GO TO CHARERR;
4           RETURN H - 'A' + 10;
4           END READHEX;
3
3       READBYTE: PROCEDURE BYTE;
4           /* READ TWO HEX DIGITS */
4           RETURN SHL(READHEX,4) OR READHEX;
4           END READBYTE;
3
3       READCS: PROCEDURE BYTE;
4           /* READ BYTE WHILE COMPUTING CHECKSUM */
```

36

```
4          DECLARE B BYTE;
4          CS = CS + (B := READBYTE);
4          RETURN B;
4          END READCS;
3
3      MAKE$DOUBLE: PROCEDURE(H,L) ADDRESS;
4          /* CREATE A BOUBLE BYTE VALUE FROM TWO SINGLE BYTE
S */
4          DECLARE (H,L) BYTE;
4          RETURN SHL(DOUBLE(H),8) OR L;
4          END MAKE$DOUBLE;
3
3      DIAGNOSE: PROCEDURE;
4
4      DECLARE M BASED TA BYTE;
4
4      NEWLINE: PROCEDURE;
5          CALL CRLF; CALL PRINTADDR(TA); CALL PRINTCHAR(':')
;
5          CALL PRINTCHAR(' ');
5          END NEWLINE;
4
4      /* PRINT DIAGNOSTIC INFORMATION AT THE CONSOLE */
4          CALL PRINT(.'LOAD  ADDRESS $'); CALL PRINTADDR(TA);
4          CALL PRINT(.'ERROR ADDRESS $'); CALL PRINTADDR(LA);
4
4          CALL PRINT(.'BYTES READ:$'); CALL NEWLINE;
4          DO WHILE TA < LA;
4          IF (LOW(TA) AND 0FH) = 0  THEN CALL NEWLINE;
5          CALL PRINTHEX(MBUFF(TA-L)); TA=TA+1;
5          CALL PRINTCHAR(' ');
5          END;
4      CALL CRLF;
4      HALT;
4      END DIAGNOSE;
3
3
3      /* INITIALIZE */
3      SA, FA, NB = 0;
3      SP = STACKPOINTER;
3      P = 0; /* PARAGRAPH COUNT */
3      TA,LA,L = 100H;  /* BASE ADDRESS OF TRANSIENT ROUTINES
 */
3      IF FALSE THEN
3          CHARERR:  /* ARRIVE HERE IF NON-HEX DIGIT IS ENCOU
NTERED */
3          DO; /* RESTORE STACKPOINTER */ STACKPOINTER = SP;
4          CALL PRINT(.'NON-HEXADECIMAL DIGIT ENCOUNTERED $')
;
4          CALL DIAGNOSE;
4          END;
3
3
3      /* READ RECORDS UNTIL :00XXXX IS ENCOUNTERED */
3
3          DO FOREVER;
3          /* SCAN THE : */
3              DO WHILE GETCHAR <> ':';
4              END;
```

```
4
4           /* SET CHECK SUM TO ZERO, AND SAVE THE RECORD LENG
TH */
4           CS = 0;
4           /* MAY BE THE END OF TAPE */
4           IF (RL := READCS) = 0 THEN
4                   GO TO FIN;
4           NB = NB + RL;
4
4           TA, LA = MAKE$DOUBLE(READCS,READCS);
4           IF SA = 0 THEN SA = LA;
4
4
4           /* READ THE RECORD TYPE (NOT CURRENTLY USED) */
4           RT = READCS;
4
4           /* PROCESS EACH BYTE */
4               DO WHILE (RL := RL - 1) <> 255;
4               CALL SETMEM(READCS); LA = LA+1;
5               END;
4           IF LA > FA THEN FA = LA - 1;
4
4           /* NOW READ CHECKSUM AND COMPARE */
4           IF CS + READBYTE <> 0 THEN
4               DO; CALL PRINT(.'CHECK SUM ERROR $');
5               CALL DIAGNOSE;
5               END;
4           END;
3
3       FIN:
3       /* EMPTY THE BUFFERS */
3       TA = LA;
3           DO WHILE L < TA;
3           CALL SETMEM(0); LA = LA+1;
4           END;
3       /* PRINT FINAL STATISTICS */
3       CALL PRINT(.'FIRST ADDRESS $'); CALL PRINTADDR(SA);
3       CALL PRINT(.'LAST  ADDRESS $'); CALL PRINTADDR(FA);
3       CALL PRINT(.'BYTES READ    $'); CALL PRINTADDR(NB);
3       CALL PRINT(.'RECORDS WRITTEN $'); CALL PRINTHEX(P);
3       CALL CRLF;
3
3       END RELOC;
2
2    /* ARRIVE HERE FROM THE SYSTEM MONITOR, READY TO READ THE
HEX TAPE
2
2    /* SET UP STACKPOINTER IN THE LOCAL AREA */
2   DECLARE STACK(16) ADDRESS, SP ADDRESS;
2   SP = STACKPOINTER; STACKPOINTER = .STACK(LENGTH(STACK));
2
2   SBP = LENGTH(SBUFF);
2      /* SET UP THE SOURCE FILE */
2       CALL MOVE(FCBA,.SFCB,33);
2       CALL MOVE(.('HEX',0),.SFCB(9),4);
2       CALL SEARCH(.SFCB);
2       IF (RFLAG := DCNT = 255) THEN
2           CALL PRINT(.'SOURCE IS READER$'); ELSE
2           DO; CALL PRINT(.'SOURCE IS DISK$');
```

```
3          CALL OPEN(.SFCB);
3          IF DCNT = 255 THEN CALL PRINT(.'-CANNOT OPEN SOURC
E$');
3          END;
2      CALL CRLF;
2
2      CALL MOVE(.'COM',FCBA+9,3);
2
2   /* REMOVE ANY EXISTING FILE BY THIS NAME */
2   CALL DELETE(FCBA);
2   /* THEN OPEN A NEW FILE */
2   CALL MAKE(FCBA);   FCB(32) = 0; /* CREATE AND SET NEXT RECORD */
2   IF DCNT = 255 THEN CALL PRINT(.'NO MORE DIRECTORY SPACE$'
); ELSE
2          DO; CALL RELOC;
3          CALL CLOSE(FCBA);
3          IF DCNT = 255 THEN CALL PRINT(.'CANNOT CLOSE FILE$
');
3          END;
2   CALL CRLF;
2
2   /* RESTORE STACKPOINTER FOR RETURN */
2  STACKPOINTER = SP;
2  RETURN 0;
2  END LOADCOM;
1  ;
1  EOF
```