TECHNICAL INFORMATION EXPLANATION

This technical information portion of your CP/M-85 documentation package is intended for users who wish to modify the CP/M Operating System or examine the internal components of the system.

Knowledge of this information is not necessary for most CP/M-85 users who only wish to use CP/M resident commands and to run utilities and application programs with the CP/M system.

This technical information consists of the following documentation items bound within three booklets:

- CP/M 2 System Interface: Chapter 5
- CP/M 2 Alteration: Chapter 6
- Appendix A: The MDS Basic I/O System (BIOS)
- Appendix B: A Skeletal CBIOS
- Appendix C: A Skeletal GETSYS/PUTSYS Program
- Appendix D: The MDS-800 Cold Start Loader for CP/M 2
- Appendix E: A Skeletal Cold Start Loader
- Appendix F: CP/M Disk Definition Library
- Appendix G: Blocking and Deblocking Algorithms.

This technical information was written by Digital Research Corporation, the original producers of CP/M Version 2. The source listings in the appendices are for illustration purposes only and do not correspond to the modules actually supplied with your system.

The CP/M-85 software products you have purchased are modifications of CP/M Version 2. In these modifications, the Basic Input/Output System (BIOS) has been customized for your hardware by Zenith Data Systems and Heath. Source listings for the actual modules used by your system are contained on the disks supplied with the system.

597-3187

CP/M®

OPERATING SYSTEM

MANUAL

CP/M 2 System Interface
Chapter 5

DIGITAL RESEARCH™
P.O. Box 579
Pacific Grove, California 93950

595-2838

COPYRIGHT

DISCLAIMER

TRADEMARKS

First Printing: July 1982

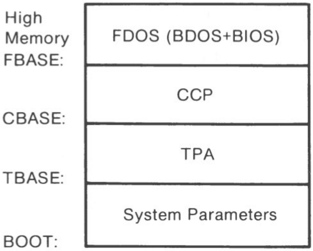
5

CP/M 2 System Interface

5.1 Introduction

This chapter describes CP/M, release 2, system organization including the structure of memory and system entry points. The intention is to provide necessary information required to write programs that operate under CP/M and that use the peripheral and disk I/O facilities of the system.

CP/M is logically divided into four parts, called the Basic I/O System (BIOS), the Basic Disk Operating System (BDOS), the Console Command Processor (CCP), and the Transient Program Area (TPA). The BIOS is a hardware-dependent module that defines the exact low level interface with a particular computer system that is necessary for peripheral device I/O. Although a standard BIOS is supplied by Digital Research, explicit instructions are provided for field reconfiguration of the BIOS to match nearly any hardware environment (see Chapter 6). The BIOS and BDOS are logically combined into a single module with a common entry point and referred to as the FDOS. The CCP is a distinct program that uses the FDOS to provide a human-oriented interface with the information that is cataloged on the backup storage device. The TPA is an area of memory (i.e., the portion that is not used by the FDOS and CCP) where various nonresident operating system commands and user programs are executed. The lower portion of memory is reserved for system information and is detailed in later sections. Memory organization of the CP/M system is shown below.



The exact memory addresses corresponding to BOOT, TBASE, CBASE, and FBASE vary from version to version and are described fully in Chapter 6. All standard CP/M versions, however, assume BOOT = 0000H, which is the base of random access memory. The machine code found at location BOOT performs a system "warm start," which loads and initializes the programs and variables necessary to return control to the CCP. Thus, transient programs need only jump to location BOOT to return control to CP/M at the command level. Further, the standard versions assume TBASE = BOOT+0100H, which is normally location 0100H. The principal entry point to the FDOS is at location BOOT+0005H (normally 0005H) where a jump to FBASE is found. The address field at BOOT+0006H (normally 0006H) contains the value of FBASE and can be used to determine the size of available memory, assuming that the CCP is being overlayed by a transient program.

Transient programs are loaded into the TPA and executed as follows. The operator communicates with the CCP by typing command lines following each prompt. Each command line takes one of the forms:

```
command

command file1

command file1 file2
```

where "command" is either a built-in function such as 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 that executes in the TPA and thus implicitly originates at TBASE in memory. The CCP loads the COM file from the disk into memory starting at TBASE and can extend up to CBASE.

If the command is followed by one or two file specifications, the CCP prepares one or two file control block (FCB) names in the system parameter area. These optional FCBs are in the form necessary to access files through the FDOS and are described in the next section.

The transient program receives control from the CCP and begins execution, using the I/O facilities of the FDOS. The transient program is "called" from the CCP. Thus, it can simply return to the CCP upon completion of its processing or can jump to BOOT to pass control back to CP/M. In the first case, the transient program must not use memory above CBASE, while in the latter case, memory up through FBASE-1 can be used.

The transient program can use the CP/M I/O facilities to communicate with the operator's console and peripheral devices, including the disk subsystem. The I/O system is accessed by passing a function number and an information address to CP/M through the FDOS entry point at BOOT+0005H. 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 to the CP/M FDOS. The FDOS, in turn, performs the operation and returns with either a disk read completion indication or an error number indicating that the disk read was unsuccessful.

5.2 Operating System Call Conventions

This section provides detailed information for performing direct operating system calls from user programs. Many of the functions listed below, however, are accessed more simply through the I/O macro library provided with the MAC macro assembler and listed in the Digital Research manual entitled, *MAC Macro Assembler: Language Manual and Applications Guide.*

CP/M facilities that are available for access by transient programs fall into two general categories: simple device I/O and disk file I/O. The simple device operations include:

Read a Console Character

Write a Console Character

Read a Sequential Tape Character

Write a Sequential Tape Character

Write a List Device Character

Get or Set I/O Status

Print Console Buffer

Read Console Buffer

Interrogate Console Ready

The FDOS operations that perform disk I/O are

Disk System Reset

Drive Selection

File Creation

File Open

File Close

Directory Search

File Delete

File Rename

Random or Sequential Read

Random or Sequential Write

Interrogate Available Disks

Interrogate Selected Disk

Set DMA Address

Set/Reset File Indicators.

As mentioned above, access to the FDOS functions is accomplished by passing a function number and information address through the primary point at location BOOT+0005H. In general, the function number is passed in register C with the information address in the double byte pair DE. Single byte values are returned in register A, with double byte values returned in HL (a zero value is returned when the function number is out of range). For reasons of compatibility, register A = L and register B = H upon return in all cases. The user should note that the register passing conventions of CP/M agree with those of Intel's PL/M systems programming language. CP/M functions and their numbers are listed below.

0	System Reset	19	Delete File
1	Console Input	20	Read Sequential
2	Console Output	21	Write Sequential
3	Reader Input	22	Make File
4	Punch Output	23	Rename File
5	List Output	24	Return Login Vector
6	Direct Console I/O	25	Return Current Disk
7	Get I/O Byte	26	Set DMA Address
8	Set I/O Byte	27	Get Addr(Alloc)
9	Print String	28	Write Protect Disk
10	Read Console Buffer	29	Get R/O Vector
11	Get Console Status	30	Set File Attributes
12	Return Version Number	31	Get Addr(Disk Parms)
13	Reset Disk System	32	Set/Get User Code
14	Select Disk	33	Read Random
15	Open File	34	Write Random
16	Close File	35	Compute File Size
17	Search for First	36	Set Random Record
18	Search for Next	37	Reset Drive
		40	Write Random with Zero Fill

(Functions 28 and 32 should be avoided in application programs to maintain upward compatibility with CP/M.)

Upon entry to a transient program, the CCP leaves the stack pointer set to an eight-level stack area with the CCP return address pushed onto the stack, leaving seven levels before overflow occurs. Although this stack is usually not used by a transient program (i.e., most transients return to the CCP through a jump to location 0000H), it is sufficiently large to make CP/M system calls since the FDOS switches to a local stack at system entry. The assembly language program segment below, for example, reads characters continuously until an asterisk is encountered, at which time control returns to the CCP (assuming a standard CP/M system with BOOT = 0000H).

```
BDOS      EQU    0005H      ;STANDARD CP/M ENTRY
CONIN     EQU    1          ;CONSOLE INPUT FUNCTION
;
          ORG    0100H      ;BASE OF TPA
NEXTC:    MVI    C,CONIN    ;READ NEXT CHARACTER
          CALL   BDOS       ;RETURN CHARACTER IN <A>
          CPI    '*'        ;END OF PROCESSING?
          JNZ    NEXTC      ;LOOP IF NOT
          RET               ;RETURN TO CCP
          END
```

CP/M implements a named file structure on each disk, providing a logical organization that allows any particular file to contain any number of records from completely empty to the full capacity of the drive. Each drive is logically distinct with a disk directory and file data area. The disk file names are in three parts: the drive select code, the filename consisting of one to eight nonblank characters, and the filetype consisting of zero to three nonblank characters. The filetype names the generic category of a particular file, while the filename distinguishes individual files in each category. The filetypes listed below name a few generic categories that have been established, although they are somewhat arbitrary.

ASM	Assembler Source	PLI	PL/I Source File
PRN	Printer Listing	REL	Relocatable Module
HEX	Hex Machine Code	TEX	TEX Formatter Source
BAS	Basic Source File	BAK	ED Source Backup
INT	Intermediate Code	SYM	SID Symbol File
COM	Command File	$$$	Temporary File

Source files are treated as a sequence of ASCII characters, where each "line" of the source file is followed by a carriage-return line-feed sequence (0DH followed by 0AH). Thus one 128-byte CP/M record could contain several lines of source text. The end of an ASCII file is denoted by a control-Z character (1AH) or a real end-of-file returned by the CP/M read operation. Control-Z characters embedded within machine code files (e.g., COM files) are ignored, however, and the end-of-file condition returned by CP/M is used to terminate read operations.

Files in CP/M can be thought of as a sequence of up to 65536 records of 128 bytes each, numbered from 0 through 65535, thus allowing a maximum of 8 megabytes per file. However, the user should note that although the records may be considered logically contiguous, they may not be physically contiguous in the disk data area. Internally, all files are divided into 16K byte segments called logical extents, so that counters are easily maintained as 8-bit values. The division into extents is discussed in the paragraphs that follow; however, they are not particularly significant for the programmer, since each extent is automatically accessed in both sequential and random access modes.

In the file operations starting with function number 15, DE usually addresses a file control block (FCB). Transient programs often use the default file control block area reserved by CP/M at location BOOT+005CH (normally 005CH) for simple file operations. The basic unit of file information is a 128-byte record used for all file operations; thus, a default location for disk I/O is provided by CP/M at location BOOT+0080H (normally 0080H), which is the initial default DMA address (see function 26). All directory operations take place in a reserved area that does not affect write buffers as was the case in release 1, with the exception of Search First and Search Next, where compatibility is required.

The FCB data area consists of a sequence of 33 bytes for sequential access and a series of 36 bytes in the case when the file is accessed randomly. The default FCB normally located at 005CH can be used for random access files, since the three bytes starting at BOOT+007DH are available for this purpose. The FCB format is shown with the following fields:

```
|dr |f1 |f2 |/  /|f8 |t1 |t2 |t3 |ex |s1 |s2 |rc |d0 |/  /|dn |cr |r0 |r1 |r2|
 00  01  02  ...  08  09  10  11  12  13  14  15  16  ...  31  32  33  34  35
```

where

dr	drive code (0-16) 0 => use default drive for file 1 => auto disk select drive A, 2 => auto disk select drive B, ... 16=> auto disk select drive P.
f1...f8	contain the file name in ASCII upper case, with high bit = 0
t1,t2,t3	contain the file type in ASCII upper case, with high bit = 0 t1', t2', and t3' denote the bit of these positions, t1' = 1 => Read/Only file, t2' = 1 => SYS file, no DIR list
ex	contains the current extent number, normally set to 00 by the user, but in range 0-31 during file I/O
s1	reserved for internal system use
s2	reserved for internal system use, set to zero on call to OPEN, MAKE, SEARCH
rc	record count for extent "ex," takes on values from 0-127
d0...dn	filled-in by CP/M, reserved for system use
cr	current record to read or write in a sequential file operation, normally set to zero by user
r0,r1,r2	optional random record number in the range 0-65535, with overflow to r2, r0, r1 constitute a 16-bit value with low byte r0, and high byte r1

Each file being accessed through CP/M must have a corresponding FCB, which provides the name and allocation information for all subsequent file operations. When accessing files, it is the programmer's responsibility to fill the lower 16 bytes of the FCB and initialize the cr field. Normally, bytes 1 through 11 are set to the ASCII character values for the file name and file type, while all other fields are zero.

FCBs are stored in a directory area of the disk, and are brought into central memory before the programmer proceeds with file operations (see the OPEN and MAKE functions). The memory copy of the FCB is updated as file operations take place and later recorded permanently on disk at the termination of the file operation (see the CLOSE command).

The CCP constructs the first 16 bytes of two optional FCBs for a transient by scanning the remainder of the line following the transient name, denoted by file1 and file2 in the prototype command line described above, with unspecified fields set to ASCII blanks. The first FCB is constructed at location BOOT+005CH and can be used as is for subsequent file operations. The second FCB occupies the d0 ... dn portion of the first FCB and must be moved to another area of memory before use. If, for example, the operator types

```
PROGNAME B:X.ZOT Y.ZAP
```

the file PROGNAME.COM is loaded into the TPA and the default FCB at BOOT+005CH is initialized to drive code 2, file name X, and file type ZOT. The second drive code takes the default value 0, which is placed at BOOT+006CH, with the file name Y placed into location BOOT+006DH and file type ZAP located 8 bytes later at BOOT+0075H. All remaining fields through cr are set to zero. The user should note again that it is the programmer's responsibility to move this second file name and type to another area, usually a separate file control block, before opening the file that begins at BOOT+005CH, because the open operation will overwrite the second name and type.

If no file names are specified in the original command, the fields beginning at BOOT+005DH and BOOT+006DH 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 convenience, the default buffer area at location BOOT+0080H is initialized to the command line tail typed by the operator following the program name. The first position contains the number of characters, with the characters themselves following the character count. Given the above command line, the area beginning at BOOT+0080H is initialized as follows:

BOOT+0080H:

+00	+01	+02	+03	+04	+05	+06	+07	+08	+09	+A	+B	+C	+D	+E
E	' '	'B'	':'	'X'	'.'	'Z'	'O'	'T'	' '	'Y'	'.'	'Z'	'A'	'P'

where the characters are translated to upper case ASCII with uninitialized memory following the last valid character. Again, it is the responsibility of the programmer to extract the information from this buffer before any file operations are performed, unless the default DMA address is explicitly changed.

Individual functions are described in detail in the pages that follow.

Function 0: System Reset

Entry Parameters:
  Register C: 00H

The system reset function returns control to the CP/M operating system at the CCP level. The CCP reinitializes the disk subsystem by selecting and logging in disk drive A. This function has exactly the same effect as a jump to location BOOT.

Function 1: Console Input

Entry Parameters:
  Register C: 01H

Returned Value:
  Register A: ASCII Character

The console input function reads the next console character to register A. Graphic characters, along with carriage return, line feed, and back space (ctl-H) are echoed to the console. Tab characters (ctl-I) move the cursor to the next tab stop. A check is made for start/stop scroll (ctl-S) and start/stop printer echo (ctl-P). The FDOS does not return to the calling program until a character has been typed, thus suspending execution if a character is not ready.

Function 2: Console Output

Entry Parameters:
    Register C: 02H
    Register E: ASCII Character

The ASCII character from register E is sent to the console device. As in function 1, tabs are expanded and checks are made for start/stop scroll and printer echo.

Function 3: Reader Input

Entry Parameters:
    Register C: 03H

Returned Value:
    Register A: ASCII Character

The Reader Input function reads the next character from the logical reader into register A (see the IOBYTE definition in Chapter 6). Control does not return until the character has been read.

Function 4: Punch Output

Entry Parameters:
    Register C: 04H
    Register E: ASCII Character

The Punch Output function sends the character from register E to the logical punch device.

Function 5: List Output

Entry Parameters:
    Register C: 05H
    Register E: ASCII Character

The List Output function sends the ASCII character in register E to the logical listing device.

Function 6: Direct Console I/O

Entry Parameters:
Register C: 06H
Register E: 0FFH (input) or char (output)

Returned Value:
Register A: char or status

Direct console I/O is supported under CP/M for those specialized applications where basic console input and output are required. Use of this function should, in general, be avoided since it bypasses all of CP/M's normal control character functions (e.g., control-S and control-P). Programs that perform direct I/O through the BIOS under previous releases of CP/M, however, should be changed to use direct I/O under BDOS so that they can be fully supported under future releases of MP/M and CP/M.

Upon entry to function 6, register E either contains hexadecimal FF, denoting a console input request, or an ASCII character. If the input value is FF, function 6 returns A = 00 if no character is ready, otherwise A contains the next console input character.

If the input value in E is not FF, function 6 assumes that E contains a valid ASCII character that is sent to the console.

Function 6 must not be used in conjunction with other console I/O functions.

Function 7: Get I/O Byte

Entry Parameters:
Register C: 07H

Returned Value:
Register A: I/O Byte Value

The Get I/O Byte function returns the current value of IOBYTE in register A. See Chapter 6 for IOBYTE definition.

Function 8: Set I/O Byte

Entry Parameters:
Register C: 08H
Register E: I/O Byte Value

The Set I/O Byte function changes the IOBYTE value to that given in register E.

Function 9: Print String

Entry Parameters:
Register C: 09H
Registers DE: String Address

The Print String function sends the character string stored in memory at the location given by DE to the console device, until a $ is encountered in the string. Tabs are expanded as in function 2, and checks are made for start/stop scroll and printer echo.

Function 10: Read Console Buffer

Entry Parameters:
Register C: 0AH
Registers DE: Buffer Address

Returned Value:
Console Characters in Buffer

The Read Buffer function reads a line of edited console input into a buffer addressed by registers DE. Console input is terminated when either input buffer overflows or a carriage return or line feed is typed. The Read Buffer takes the form:

```
DE: +0  +1  +2  +3  +4  +5  +6  +7  +8 . . .+n
    |mx|nc |c1 |c2 |c3 |c4 |c5 |c6 |c7 | . . .|??|
```

where mx is the maximum number of characters that the buffer will hold (1 to 255) and nc is the number of characters read (set by FDOS upon return), followed by the characters read from the console. If nc < mx, then uninitialized positions follow the last character, denoted by ?? in the above figure. A number of control functions are recognized during line editing:

rub/del	removes and echoes the last character
ctl-C	reboots when at the beginning of line
ctl-E	causes physical end of line
ctl-H	backspaces one character position
ctl-J	(line feed) terminates input line
ctl-M	(return) terminates input line
ctl-R	retypes the current line after new line
ctl-U	removes current line
ctl-X	same as ctl-U.

The user should also note that certain functions that return the carriage to the leftmost position (e.g., ctl-X) do so only to the column position where the prompt ended (in earlier releases, the carriage returned to the extreme left margin). This convention makes operator data input and line correction more legible.

Function 11: Get Console Status

Entry Parameters:
    Register C: 0BH

Returned Value:
    Register A: Console Status

The Console Status function checks to see if a character has been typed at the console. If a character is ready, the value 0FFH is returned in register A. Otherwise a 00H value is returned.

Function 12: Return Version Number

Entry Parameters:
    Register C: 0CH

Returned Value:
    Registers HL: Version Number

Function 12 provides information that allows version independent programming. A two-byte value is returned, with H = 00 designating the CP/M release (H = 01 for MP/M), and L = 00 for all releases previous to 2.0. CP/M 2.0 returns a hexadecimal 20 in register L, with subsequent version 2 releases in the hexadecimal range 21, 22, through 2F. Using function 12, for example, the user can write application programs that provide both sequential and random access functions.

Function 13: Reset Disk System

Entry Parameters:
    Register C: 0DH

The Reset Disk Function is used to programmatically restore the file system to a reset state where all disks are set to read/write (see functions 28 and 29), only disk drive A is selected, and the default DMA address is reset to BOOT+0080H. This function can be used, for example, by an application program that requires a disk change without a system reboot.

Function 14: Select Disk

Entry Parameters:
 Register C: 0EH
 Register E: Selected Disk

The Select Disk function designates the disk drive named in register E as the default disk for subsequent file operations, with E = 0 for drive A, 1 for drive B, and so on through 15, corresponding to drive P in a full 16 drive system. The drive is placed in an on-line status, which activates its directory until the next cold start, warm start, or disk system reset operation. If the disk medium is changed while it is on-line, the drive automatically goes to a read/only status in a standard CP/M environment (see function 28). FCBs that specify drive code zero (dr = 00H) automatically reference the currently selected default drive. Drive code values between 1 and 16, however, ignore the selected default drive and directly reference drives A through P.

Function 15: Open File

Entry Parameters:
 Register C: 0FH
 Registers DE: FCB Address

Returned Value:
 Register A: Directory Code

The Open File operation is used to activate a file that currently exists in the disk directory for the currently active user number. The FDOS scans the referenced disk directory for a match in positions 1 through 14 of the FCB referenced by DE (byte s1 is automatically zeroed), where an ASCII question mark (3FH) matches any directory character in any of these positions. Normally, no question marks are included, and bytes ex and s2 of the FCB are zero.

If a directory element is matched, the relevant directory information is copied into bytes d0 through dn of the FCB, thus allowing access to the files through subsequent read and write operations. The user should note that an existing file must not be accessed until a successful open operation is completed. Upon return, the open function returns a directory code with the value 0 through 3 if the open was successful or 0FFH (255 decimal) if the file cannot be found. If question marks occur in the FCB, the first matching FCB is activated. Note that the current record (cr) must be zeroed by the program if the file is to be accessed sequentially from the first record.

Function 16: Close File	
Entry Parameters:	
Register C:	10H
Registers DE:	FCB Address
Returned Value:	
Register A:	Directory Code

The Close File function performs the inverse of the open file function. Given that the FCB addressed by DE has been previously activated through an open or make function (see functions 15 and 22), the close function permanently records the new FCB in the referenced disk directory. The FCB matching process for the close is identical to the open function. The directory code returned for a successful close operation is 0, 1, 2, or 3, while a 0FFH (255 decimal) is returned if the file name cannot be found in the directory. A file need not be closed if only read operations have taken place. If write operations have occurred, however, the close operation is necessary to record the new directory information permanently.

Function 17: Search for First	
Entry Parameters	
Register C:	11H
Registers DE:	FCB Address
Returned Value:	
Register A:	Directory Code

Search First scans the directory for a match with the file given by the FCB addressed by DE. The value 255 (hexadecimal FF) is returned if the file is not found; otherwise, 0, 1, 2, or 3 is returned indicating the file is present. When the file is found, the current DMA address is filled with the record containing the directory entry, and the relative starting position is A * 32 (i.e., rotate the A register left 5 bits, or ADD A five times). Although not normally required for application programs, the directory information can be extracted from the buffer at this position.

An ASCII question mark (63 decimal, 3F hexadecimal) in any position from f1 through ex matches the corresponding field of any directory entry on the default or auto-selected disk drive. If the dr field contains an ASCII question mark, the auto disk select function is disabled and the default disk is searched, with the search function returning any matched entry, allocated or free, belonging to any user number. This latter function is not normally used by application programs, but it allows complete flexibility to scan all current directory values. If the dr field is not a question mark, the s2 byte is automatically zeroed.

Function 18: Search for Next

Entry Parameters:
    Register C: 12H

Returned Value:
    Register A: Directory Code

The Search Next function is similar to the Search First function, except that the directory scan continues from the last matched entry. Similar to function 17, function 18 returns the decimal value 255 in A when no more directory items match.

Function 19: Delete File

Entry Parameters:
    Register C: 13H
    Registers DE: FCB Address

Returned Value:
    Register A: Directory Code

The Delete File function removes files that match the FCB addressed by DE. The filename and type may contain ambiguous references (i.e., question marks in various positions), but the drive select code cannot be ambiguous, as in the Search and Search Next functions.

Function 19 returns a decimal 255 if the referenced file or files cannot be found; otherwise, a value in the range 0 to 3 is returned.

Function 20: Read Sequential

Entry Parameters:
    Register C: 14H
    Registers DE: FCB Address

Returned Value:
    Register A: Directory Code

Given that the FCB addressed by DE has been activated through an open or make function (numbers 15 and 22), the Read Sequential function reads the next 128-byte record from the file into memory at the current DMA address. The record is read from position cr of the extent, and the cr field is automatically incremented to the next record position. If the cr field overflows, the next logical extent is automatically opened and the cr field is reset to zero in preparation for the next read operation. The value 00H is returned in the A register if the read operation was successful, while a nonzero value is returned if no data exist at the next record position (e.g., end-of-file occurs).

Function 21: Write Sequential	
Entry Parameters:	
Register C:	15H
Registers DE:	FCB Address
Returned Value:	
Register A:	Directory Code

Given that the FCB addressed by DE has been activated through an open or make function (numbers 15 and 22), the Write Sequential function writes the 128-byte data record at the current DMA address to the file named by the FCB. The record is placed at position cr of the file, and the cr field is automatically incremented to the next record position. If the cr field overflows, the next logical extent is automatically opened and the cr field is reset to zero in preparation for the next write operation. Write operations can take place into an existing file, in which case, newly written records overlay those that already exist in the file. Register A = 00H upon return from a successful write operation, while a nonzero value indicates an unsuccessful write caused by a full disk.

Function 22: Make File	
Entry Parameters:	
Register C:	16H
Registers DE:	FCB Address
Returned Value:	
Register A:	Directory Code

The Make File operation is similar to the open file operation except that the FCB must name a file that does not exist in the currently referenced disk directory (i.e., the one named explicitly by a nonzero dr code or the default disk if dr is zero). The FDOS creates the file and initializes both the directory and main memory value to an empty file. The programmer must ensure that no duplicate file names occur, and a preceding delete operation is sufficient if there is any possibility of duplication. Upon return, register A = 0, 1, 2, or 3 if the operation was successful and 0FFH (255 decimal) if no more directory space is available. The make function has the side effect of activating the FCB and thus a subsequent open is not necessary.

Function 23: Rename File	
Entry Parameters:	
Register C:	17H
Registers DE:	FCB Address
Returned Value:	
Register A:	Directory Code

The Rename function uses the FCB addressed by DE to change all occurrences of the file named in the first 16 bytes to the file named in the second 16 bytes. The drive code dr at position 0 is used to select the drive, while the drive code for the new file name at position 16 of the FCB is assumed to be zero. Upon return, register A is set to a value between 0 and 3 if the rename was successful and 0FFH (255 decimal) if the first file name could not be found in the directory scan.

Function 24: Return Log-in Vector	
Entry Parameters:	
Register C:	18H
Returned Value:	
Registers HL:	Log-in Vector

The log-in vector value returned by CP/M is a 16-bit value in HL, where the least significant bit of L corresponds to the first drive A and the high order bit of H corresponds to the sixteenth drive, labeled P. A 0 bit indicates that the drive is not on-line, while a 1 bit marks a drive that is actively on-line as a result of an explicit disk drive selection or an implicit drive select caused by a file operation that specified a nonzero dr field. The user should note that compatibility is maintained with earlier releases, since registers A and L contain the same values upon return.

Function 25: Return Current Disk	
Entry Parameters:	
Register C:	19H
Returned Value:	
Register A:	Current Disk

Function 25 returns the currently selected default disk number in register A. The disk numbers range from 0 through 15 corresponding to drives A through P.

Function 26: Set DMA Address	
Entry Parameters:	
Register C:	1AH
Registers DE:	DMA Address

DMA is an acronym for Direct Memory Address, which is often used in connection with disk controllers that directly access the memory of the mainframe computer to transfer data to and from the disk subsystem. Although many computer systems use non-DMA access (i.e., the data are transferred through programmed I/O operations), the DMA address has, in CP/M, come to mean the address at which the 128-byte data record resides before a disk write and after a disk read. Upon cold start, warm start, or disk system reset, the DMA address is automatically set to BOOT+0080H. The Set DMA function, however, can be used to change this default value to address another area of memory where the data records reside. Thus, the DMA address becomes the value specified by DE until it is changed by a subsequent Set DMA function, cold start, warm start, or disk system reset.

Function 27: Get ADDR(Alloc)

Entry Parameters:
    Register C: 1BH

Returned Value:
    Registers HL: ALLOC Address

An allocation vector is maintained in main memory for each on-line disk drive. Various system programs use the information provided by the allocation vector to determine the amount of remaining storage (see the STAT program). Function 27 returns the base address of the allocation vector for the currently selected disk drive. However, the allocation information may be invalid if the selected disk has been marked read/only. Although this function is not normally used by application programs, additional details of the allocation vector are found in Chapter 6.

Function 28: Write Protect Disk

Entry Parameters:
    Register C: 1CH

The disk write protect function provides temporary write protection for the currently selected disk. Any attempt to write to the disk before the next cold or warm start operation produces the message:

```
BDOS ERR on d: R/O
```

Function 29: Get Read/Only Vector

Entry Parameters:
    Register C: 1DH

Returned Value:
    Registers HL: R/O Vector Value

Function 29 returns a bit vector in register pair HL, which indicates drives that have the temporary read-only bit set. As in function 24, the least significant bit corresponds to drive A, while the most significant bit corresponds to drive P. The R/O bit is set either by an explicit call to function 28 or by the automatic software mechanisms within CP/M that detect changed disks.

Function 30: Set File Attributes

Entry Parameters:
    Register C: 1EH
    Registers DE: FCB Address

Returned Value:
    Register A: Directory Code

The Set File Attributes function allows programmatic manipulation of permanent indicators attached to files. In particular, the R/O and System attributes (t1' and t2') can be set or reset. The DE pair addresses an unambiguous file name with the appropriate attributes set or reset. Function 30 searches for a match and changes the matched directory entry to contain the selected indicators. Indicators f1' through f4' are not currently used, but may be useful for applications programs, since they are not involved in the matching process during file open and close operations. Indicators f5' through f8' and t3' are reserved for future system expansion.

Function 31: Get ADDR(Disk Parms)

Entry Parameters:
    Register C: 1FH

Returned Value:
    Registers HL: DPB Address

The address of the BIOS resident disk parameter block is returned in HL as a result of this function call. This address can be used for either of two purposes. First, the disk parameter values can be extracted for display and space computation purposes, or transient programs can dynamically change the values of current disk parameters when the disk environment changes, if required. Normally, application programs will not require this facility.

Function 32: Set/Get User Code

Entry Parameters:
    Register C: 20H
    Register E: 0FFH (get) or
        User Code (set)

Returned Value:
    Register A: Current Code or
        (no value)

An application program can change or interrogate the currently active user number by calling function 32. If register E = 0FFH, the value of the current user number is returned in register A, where the value is in the range of 0 to 15. If register E is not 0FFH, the current user number is changed to the value of E (modulo 16).

Function 33: Read Random

Entry Parameters:
    Register C: 21H
    Registers DE: FCB Address

Returned Value:
    Register A: Return Code

The Read Random function is similar to the sequential file read operation of previous releases, except that the read operation takes place at a particular record number, selected by the 24-bit value constructed from the 3-byte field following the FCB (byte positions r0 at 33, r1 at 34, and r2 at 35). The user should note that the sequence of 24 bits is stored with least significant byte first (r0), middle byte next (r1), and high byte last (r2). CP/M does not reference byte r2, except in computing the size of a file (function 35). Byte r2 must be zero, however, since a nonzero value indicates overflow past the end of file.

Thus, the r0, r1 byte pair is treated as a double-byte, or "word" value, which contains the record to read. This value ranges from 0 to 65535, providing access to any particular record of the 8-megabyte file. To process a file using random access, the base extent (extent 0) must first be opened. Although the base extent may or may not contain any allocated data, this ensures that the file is properly recorded in the directory and is visible in DIR requests. The selected record number is then stored in the random record field (r0, r1), and the BDOS is called to read the record. Upon return from the call, register A either contains an error code, as listed below, or the value 00, indicating the operation was successful. In the latter case, the current DMA address contains the randomly accessed record. The user should note that contrary to the sequential read operation, the record number is not advanced. Thus, subsequent random read operations continue to read the same record.

Upon each random read operation, the logical extent and current record values are automatically set. Thus, the file can be sequentially read or written, starting from the current randomly accessed position. However, the user should note that, in this case, the last randomly read record will be reread as one switches from random mode to sequential read and the last record will be rewritten as one switches to a sequential write operation. The user can, of course, simply advance the random record position following each random read or write to obtain the effect of a sequential I/O operation.

Error codes returned in register A following a random read are listed below.

01	reading unwritten data
02	(not returned in random mode)
03	cannot close current extent
04	seek to unwritten extent
05	(not returned in read mode)
06	seek past physical end of disk

Error codes 01 and 04 occur when a random read operation accesses a data block that has not been previously written or an extent that has not been created, which are equivalent conditions. Error code 03 does not normally occur under proper system operation. If it does, it can be cleared by simply rereading or reopening extent zero as long as the disk is not physically write protected. Error code 06 occurs whenever byte r2 is nonzero under the current 2.0 release. Normally, nonzero return codes can be treated as missing data, with zero return codes indicating operation complete.

Function 34: Write Random

Entry Parameters:
  Register C: 22H
  Registers DE: FCB Address

Returned Value:
  Register A: Return Code

The Write Random operation is initiated similarly to the Read Random call, except that data are written to the disk from the current DMA address. Further, if the disk extent or data block that is the target of the write has not yet been allocated, the allocation is performed before the write operation continues. As in the Read Random operation, the random record number is not changed as a result of the write. The logical extent number and current record positions of the file control block are set to correspond to the random record that is being written. Again, sequential read or write operations can begin following a random write, with the notation that the currently addressed record is either read or rewritten again as the sequential operation begins. The user can also simply advance the random record position following each write to get the effect of a sequential write operation. The user should note that, in particular, reading or writing the last record of an extent in random mode does not cause an automatic extent switch as it does in sequential mode.

The error codes returned by a random write are identical to the random read operation with the addition of error code 05, which indicates that a new extent cannot be created as a result of directory overflow.

Function 35: Compute File Size

Entry Parameters:
  Register C: 23H
  Registers DE: FCB Address

Returned Value:
  Random Record Field Set

When computing the size of a file, the DE register pair addresses an FCB in random mode format (bytes r0, r1, and r2 are present). The FCB contains an unambiguous file name that is used in the directory scan. Upon return, the random record bytes contain the "virtual" file size, which is, in effect, the record address of the record following the end of the file. Following a call to function 35, if the high record byte r2 is 01, the file contains the maximum record count 65536. Otherwise, bytes r0 and r1 constitute a 16-bit value (r0 is the least significant byte, as before), which is the file size.

Data can be appended to the end of an existing file by simply calling function 35 to set the random record position to the end of file and then performing a sequence of random writes starting at the preset record address.

The virtual size of a file corresponds to the physical size when the file is written sequentially. If the file was created in random mode and "holes" exist in the allocation, the file may in fact contain fewer records than the size indicates. For example, if only the last record of an 8-megabyte file is written in random mode (i.e., record number 65535), the virtual size is 65536 records, although only one block of data is actually allocated.

Function 36: Set Random Record

Entry Parameters:
    Register C: 24H
    Registers DE: FCB Address

Returned Value:
    Random Record Field Set

The Set Random Record function causes the BDOS automatically to produce the random record position from a file that has been read or written sequentially to a particular point. The function can be useful in two ways.

First, it is often necessary initially to read and scan a sequential file to extract the positions of various "key" fields. As each key is encountered, function 36 is called to compute the random record position for the data corresponding to this key. If the data unit size is 128 bytes, the resulting record position is placed into a table with the key for later retrieval. After scanning the entire file and tabulating the keys and their record numbers, the user can move instantly to a particular keyed record by performing a random read, using the corresponding random record number that was saved earlier. The scheme is easily generalized for variable record lengths, since the program need only store the buffer-relative byte position along with the key and record number to find the exact starting position of the keyed data at a later time.

A second use of function 36 occurs when switching from a sequential read or write over to random read or write. A file is sequentially accessed to a particular point in the file, function 36 is called, which sets the record number, and subsequent random read and write operations continue from the selected point in the file.

Function 37: Reset Drive

Entry Parameters:
    Register C: 25H
    Registers DE: Drive Vector

Returned Value:
    Register A: 00H

The Reset Drive function allows resetting of specified drives. The passed parameter is a 16 bit vector of drives to be reset; the least significant bit is drive A:.

To maintain compatibility with MP/M, CP/M returns a zero value.

Function 40: Write Random With Zero Fill	
Entry Parameters:	
Register C:	28H
Registers DE:	FCB Address
Returned Value:	
Register A:	Return Code

The Write Random With Zero Fill operation is similar to Function 34, with the exception that a previously unallocated block is filled with zeros before the data are written.

5.3 A Sample File-to-File Copy Program

The program shown below provides a relatively simple example of file operations. The program source file is created as COPY.ASM using the CP/M ED program and then assembled using ASM or MAC, resulting in a HEX file. The LOAD program is used to produce a COPY.COM file, which executes directly under the CCP. The program begins by setting the stack pointer to a local area and proceeds to move the second name from the default area at 006CH to a 33-byte file control block called DFCB. The DFCB is then prepared for file operations by clearing the current record field. At this point, the source and destination FCBs are ready for processing, since the SFCB at 005CH is properly set up by the CCP upon entry to the COPY program. That is, the first name is placed into the default FCB, with the proper fields zeroed, including the current record field at 007CH. The program continues by opening the source file, deleting any existing destination file, and creating the destination file. If all this is successful, the program loops at the label COPY until each record has been read from the source file and placed into the destination file. Upon completion of the data transfer, the destination file is closed and the program returns to the CCP command level by jumping to BOOT.

```
                        ;       sample file-to-file copy program
                        ;
                        ;       at the ccp level, the command
                        ;
                        ;               copy a:x.y b:u.v
                        ;
                        ;       copies the file named x.y from drive
                        ;       a to a file named u.v. on drive b.
                        ;
0000 =                  boot    equ  0000h     ; system reboot
0005 =                  bdos    equ  0005h     ; bdos entry point
005c =                  fcbl    equ  005ch     ; first file name
005c =                  sfcb    equ  fcbl      ; source fcb
006c =                  fcb2    equ  006ch     ; second file name
0080 =                  dbuff   equ  0080h     ; default buffer
0100 =                  tpa     equ  0100h     ; beginning of tpa
                        ;
0009 =                  printf  equ  9         ; print buffer func#
000f =                  openf   equ  15        ; open file func#
0010 =                  closef  equ  16        ; close file func#
```

```
0013 =              deletef  equ  19        ; delete file func#
0014 =              readf    equ  20        ; sequential read
0015 =              writef   equ  21        ; sequential write
0016 =              makef    equ  22        ; make file func#
                    ;
0100                         org  tpa       ; beginning of tpa
0100 311b02                  lxi  sp,stack  ; local stack
                    ;
                    ;        move second file name to dfcb
0103 0e10                    mvi  c,16      ; half an fcb
0105 116c00                  lxi  d,fcb2    ; source of move
0108 21da01                  lxi  h,dfcb    ; destination fcb
010b 1a             mfcb:    ldax d         ; source fcb
010c 13                      inx  d         ; ready next
010d 77                      mov  m,a       ; dest fcb
010e 23                      inx  h         ; ready next
010f 0d                      dcr  c         ; count 16...0
0110 c20b01                  jnz  mfcb      ; loop 16 times
                    ;
                    ;        name has been removed, zero cr
0113 af                      xra  a         ; a = 00h
0114 32fa01                  sta  dfcbcr    ; current rec = 0
                    ;
                    ;        source and destination fcb's ready
                    ;
0117 115c00                  lxi  d,sfcb    ; source file
011a cd6901                  call open      ; error if 255
011d 118701                  lxi  d,nofile  ; ready message
0120 3c                      inr  a         ; 255 becomes 0
0121 cc6101                  cz   finis     ; done if no file
                    ;
                    ;        source file open, prep destination
0124 11da01                  lxi  d,dfcb    ; destination
0127 cd7301                  call delete    ; remove if present
                    ;
012a 11da01                  lxi  d,dfcb    ; destination
012d cd8201                  call make      ; create the file
0130 119601                  lxi  d,nodir   ; ready message
0133 3c                      inr  a         ; 255 becomes 0
0134 cc6101                  cz   finis     ; done if no dir space
                    ;
                    ;        source file open, dest file open
                    ;        copy until end of file on source
                    ;
0137 115c00         copy:    lxi  d,sfcb    ; source
013a cd7801                  call read      ; read next record
013d b7                      ora  a         ; end of file?
013e c25101                  jnz  eofile    ; skip write if so
                    ;
                    ;        not end of file, write the record
0141 11da01                  lxi  d,dfcb    ; destination
0144 cd7d01                  call write     ; write record
0147 11a901                  lxi  d,space   ; ready message
014a b7                      ora  a         ; 00 if write ok
014b c46101                  cnz  finis     ; end if so
```

```
014e c33701             jmp  copy       ; loop until eof
                ;
                eofile: ; end of file, close destination
0151 11da01             lxi  d,dfcb     ; destination
0154 cd6e01             call close      ; 255 if error
0157 21bb01             lxi  h,wrprot   ; ready message
015a 3c                 inr  a          ; 255 becomes 00
015b cc6101             cz   finis      ; shouldn't happen
                ;
                ;       copy operation complete, end
015e 11cc01             lxi  d,normal   ; ready message
                ;
                finis:  ; write message given by de, reboot
0161 0e09               mvi  c,printf
0163 cd0500             call bdos       ; write message
0166 c30000             jmp  boot       ; reboot system
                ;
                ;       system interface subroutines
                ;       (all return directly from bdos)
                ;
0169 0e0f       open:   mvi  c,openf
016b c30500             jmp  bdos
                ;
016e 0e10       close:  mvi  c,closef
0170 c30500             jmp  bdos
                ;
0173 0e13       delete  mvi  c,deletef
0175 c30500             jmp  bdos
                ;
0178 0e14       read:   mvi  c,readf
017a c30500             jmp  bdos
                ;
017d 0e15       write:  mvi  c,writef
017f c30500             jmp  bdos
                ;
0182 0e16       make:   mvi  c,makef
0184 c30500             jmp  bdos
                ;
                ;       console messages
0187 6e6f20f    nofile: db   'no source file$'
0196 6e6f209    nodir:  db   'no directory space$'
01a9 6f7574f    space:  db   'out of data space$'
01bb 7772695    wrprot: db   'write protected?$'
01cc 636f700    normal: db   'copy complete$'
                ;
                ;       data areas
01da            dfcb:   ds   33         ; destination fcb
01fa =          dfcbcr  equ  dfcb+32    ; current record
                ;
01fb                    ds   32         ; 16 level stack
                stack:
021b                    end
```


The user should note that there are several simplifications in this particular program. First, there are no checks for invalid file names that could, for example, contain ambiguous references. This situation could be detected by scanning the 32-byte default area starting at location 005CH for ASCII question marks. A check should also be made to ensure that the file names have, in fact, been included (check locations 005DH and 006DH for nonblank ASCII characters). Finally, a check should be made to ensure that the source and destination file names are different. An improvement in speed could be obtained by buffering more data on each read operation. One could, for example, determine the size of memory by fetching FBASE from location 0006H and using the entire remaining portion of memory for a data buffer. In this case, the programmer simply resets the DMA address to the next successive 128-byte area before each read. Upon writing to the destination file, the DMA address is reset to the beginning of the buffer and incremented by 128 bytes to the end as each record is transferred to the destination file.

5.4 A Sample File Dump Utility

The file dump program shown below is slightly more complex than the simple copy program given in the previous section. The dump program reads an input file, specified in the CCP command line, and displays the content of each record in hexadecimal format at the console. Note that the dump program saves the CCP's stack upon entry, resets the stack to a local area, and restores the CCP's stack before returning directly to the CCP. Thus, the dump program does not perform and warm start at the end of processing.

```
                                  ; DUMP program reads input file and displays hex
                                  data
                                  ;
0100                                        org  100h
0005 =                            bdos      equ  0005h =   ;bdos entry point
0001 =                            cons      equ  1         ;read console
0002 =                            typef     equ  2         ;type function
0009 =                            printf    equ  9         ;buffer print entry
000b =                            brkf      equ  11        ;break key function
                                                           ;(true if char
000f =                            openf     equ  15        ;file open
0014 =                            readf     equ  20        ;read function
                                  ;
005c =                            fcb       equ  5ch       ;file control block
                                                           ;address
0080 =                            buff      equ  80h       ;input disk buffer
                                                           ;address
                                  ;
                                  ;         non graphic characters
000d =                            cr        equ  0dh       ;carriage return
000a =                            lf        equ  0ah       ;line feed
                                  ;
                                  ;         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)128)
007c =                            fcbcr     equ  fcb+32    ;current (next) record
                                                           ;number (0
```

```
007d =               fcbln    equ  fcb+33    ;fcb length
                     ;
                     ;        set up stack
0100 210000                   lxi   h,0
0103 39                       dad  sp
                     ;        entry stack pointer in hl from the ccp
0104 221502                   shld oldsp
                     ;        set sp to local stack area (restored at
                     ;        finis)
0107 315702                   lxi   sp,stktop
                     ;        read and print successive buffers
010a cdc101                   call  setup      ;set up input file
010d feff                     cpi   255        ;255 if file not present
010f c21b01                   jnz   openok     ;skip if open is ok
                     ;
                     ;        file not there, give error message and
                     ;        return
0112 11f301                   lxi   d,opnmsg
0115 cd9c01                   call  err
0118 c35101                   jmp  finis       ;to return
                     ;
                     openok:  ;open operation ok, set buffer index to
                              ;end
011b 3e80                     mvi  a,80h
011d 321302                   sta   ibp        ;set buffer pointer to 80h
                     ;        hl contains next address to print
0120 210000                   lxi   h,0        ;start with 0000
                     ;
                     gloop:
0123 e5                       pushh            ;save line position
0124 cda201                   call  gnb
0127 e1                       pop  h           ;recall line position
0138 da5101                   jc    finis      ;carry set by gnb if end
                                               ;file
012b 47                       mov  b,a
                     ;        print hex values
                     ;        check for line fold
012c 7d                       mov  a,l
012d e60f                     ani   0fh        ;check low 4 bits
012f c24401                   jnz   nonum
                     ;        print line number
0132 cd7201                   call  crlf
                     ;
                     ;        check for break key
0135 cd5901                   call  break
                     ;        accum lsb = 1 if character ready
0138 0f                       rrc              ;into carry
0139 da5101                   jc    finis      ;don't print any more
                     ;
013c 7c                       mov  a,h
013d cd8f01                   call  phex
0140 7d                       mov  a,l
0141 cd8f01                   call  phex
                     nonum:
0144 23                       inx   h          ;to next line number
```

```
0145 3e20                      mvi  a,' '
0147 cd6501                    call  pchar
014a 78                        mov a,b
014b cd8f01                    call  phex
014e c32301                    jmp  gloop
                     ;
                     finis:
                     ;         end of dump, return to cco
                     ;         (note that a jmp to 0000h reboots)
0151 cd7201                    call  crlf
0154 2a1502                    lhld  oldsp
0157 f9                        sphl
                     ;         stack pointer contains ccp's stack
                     ;         location
0158 c9                        ret              ;to the ccp
                     ;
                     ;
                     ;         subroutines
                     ;
                     break:    ;check break key (actually any key will
                               ;do)
0159 e5d5c5                    push h! push d! push b; environment
                               ; saved
015c 0e0b                      mvi  c,brkf
015e cd0500                    call  bdos
0161 c1d1e1                    pop b! pop d! pop h; environment
                               restored
0164 c9                        ret
                     ;
                     pchar:    ;print a character
0165 e5d5c5                    push h! push d! push b; saved
0168 0e02                      mvi  c,typef
016a 5f                        mov e,a
016b cd0500                    call  bdos
016e c1d1e1                    pop b! pop d! pop h; restored
0171 c9                        ret
                     ;
                     crlf:
0172 3e0d                      mvi  a,cr
0174 cd6501                    call  pchar
0177 3e0a                      mvi  a,lf
0179 cd6501                    call  pchar
017c c9                        ret
                     ;
                     ;
                     pnib:     ;print nibble in reg a
017d e60f                      ani  0fh         ;low 4 bits
017f fe0a                      cpi  10
0181 d28901                    jnc  p10
                     ;         less than or equal to 9
0184 c630                      adi  '0'
0186 c38b01                    jmp  prn
                     ;
                     ;         greater or equal to 10
0189 c637            p10:      adi  'a' - 10
```

```
018b cd6501     prn:    call  pchar
018e c9                 ret
                ;
                phex:   ;print hex char in reg a
018f f5                 pushpsw
0190 0f                 rrc
0191 0f                 rrc
0192 0f                 rrc
0193 0f                 rrc
0194 cd7d01             call  pnib      ;print nibble
0197 f1                 pop  psw
0198 cd7d01             call  pnib
019b c9                 ret
                ;
                err:    ;print error message
                ;       d,e addresses message ending with "$"
019c 0e09               mvi  c,printf         ;print buffer
                                              ;function
019e cd0500             call  bdos
01a1 c9                 ret
                ;
                ;
                gnb:    ;get next byte
01a2 3a1302             lda  ibp
01a5 fe80               cpi  80h
01a7 c2b301             jnz  g0
                ;       read another buffer
                ;
                ;
01aa cdce01             call  diskr
01ad b7                 ora  a          ;zero value if read ok
01ae cab301             jz   g0         ;for another byte
                ;       end of data, return with carry set for eof
01b1 37                 stc
01b2 c9                 ret
                ;
                g0:     ;read the byte at buff+reg a
01b3 5f                 mov e,a         ;ls byte of buffer index
01b4 1600               mvi  d,0        ;double precision
                                        ;index to de
01b6 3c                 inr  a          ;index=index+1
01b7 321302             sta  ibp        ;back to memory
                ;       pointer is incremented
                ;       save the current file address
01ba 218000             lxi  h,buff
01bd 19                 dad  d
                ;       absolute character address is in hl
01be 7e                 mov a,m
                ;       byte is in the accumulator
01bf b7                 ora  a          ;reset carry bit
01c0 c9                 ret
                ;
                setup:  ;set up file
                ;       open the file for input
01c1 af                 xra  a          ;zero to accum
```

```
01c2 327c00              sta   fcbcr        ;clear current record
                 ;
01c5 115c00              lxi   d,fcb
01c8 0e0f                mvi   c,openf
01ca cd0500              call  bdos
                 ;       255 in accum if open error
01cd c9                  ret
                 ;
                 diskr:  ;read disk file record
01ce e5d5c5              push h! push d! push b
01d1 115c00              lxi   d,fcb
01d4 0e14                mvi   c,readf
01d6 cd0500              call  bdos
01d9 c1d1e1              pop b! pop d! pop h
01dc c9                  ret
                 ;
                 ;       fixed message area
01dd 46494c0     signon: db    'file dump version 2.0$'
01f3 0d0a4e0     opnmsg: db    cr,lf,'no input file present on
                               disk$'

                 ;       variable area
0213             ibp:    ds    2            ;input buffer pointer
0215             oldsp:  ds    2            ;entry sp value from ccp
                 ;
                 ;       stack area
0217                     ds    64           ;reserve 32 level stack
                 stktop:
                 ;
0257                     end
```

5.5 A Sample Random Access Program

This chapter concludes with an extensive example of random access operation. The program listed below performs the simple function of reading or writing random records upon command from the terminal. Given that the program has been created, assembled, and placed into a file labeled RANDOM.COM, the CCP level command

```
RANDOM X.DAT
```

starts the test program. The program looks for a file by the name X.DAT (in this particular case) and, if found, proceeds to prompt the console for input. If not found, the file is created before the prompt is given. Each prompt takes the form

```
next command?
```

and is followed by operator input, terminated by a carriage return. The input commands take the form

```
nW   nR   Q
```

where n is an integer value in the range 0 to 65535, and W, R, and Q are simple command characters corresponding to random write, random read, and quit processing, respectively. If the W command is issued, the RANDOM program issues the prompt type data:.

The operator then responds by typing up to 127 characters, followed by a carriage return. RANDOM then writes the character string into the X.DAT file at record n. If the R command is issued, RANDOM reads record number n and displays the string value at the console. If the Q command is issued, the X.DAT file is closed, and the program returns to the CCP. In the interest of brevity, the only error message is

error, try again.

The program begins with an initialization section where the input file is opened or created, followed by a continuous loop at the label "ready" where the individual commands are interpreted. The default file control block at 005CH and the default buffer at 0080H are used in all disk operations. The utility subroutines then follow, which contain the principal input line processor, called "readc." This particular program shows the elements of random access processing, and can be used as the basis for further program development.

Sample Random Access Program for CP/M 2.0

```
0100                    org      100h      ;base of tpa
                ;
0000 =          reboot  equ      0000h     ;system reboot
0005 =          bdos    equ      0005h     ;bdos entry point
                ;
0001 =          coninp  equ      1         ;console input function
0002 =          conout  equ      2         ;console output function
0009 =          pstring equ      9         ;print string until '$'
000a =          rstring equ      10        ;read console buffer
000c =          version equ      12        ;return version number
000f =          openf   equ      15        ;file open function
0010 =          closef  equ      16        ;close function
0016 =          makef   equ      22        ;make file function
0021 =          readr   equ      33        ;read random
0022 =          writer  equ      34        ;write random
                ;
005c =          fcb     equ      005ch     ;default file control
                                           ;block
007d =          ranrec  equ      fcb+33    ;random record position
007f =          ranovf  equ      fcb+35    ;high order (overflow)
                                           ;byte
0080 =          buff    equ      0080h     ;buffer address
                ;
000d =          cr      equ      0dh       ;carriage return
000a =          lf      equ      0ah       ;line feed
                ;
```

Load SP, Set-Up File for Random Access

```
0100 31bc00             lxi      sp,stack
                ;
                ;       version 2.0
0103 0e0c               mvi      c,version
```

```
0105 cd0500             call    bdos
0108 fe20               cpi     20h         ;version 2.0 or better?
010a d21600             jnc     versok
                ;       bad version, message and go back
010d 111b00             lxi     d,badver
0110 cdda00             call    print
0113 c30000             jmp     reboot
                ;
                versok:
                ;       correct version for random access
0116 0e0f               mvi     c,openf     ;open default fcb
0118 115c00             lxi     d,fcb
011b cd0500             call    bdos
011e 3c                 inr     a           ;err 255 becomes zero
011f c23700             jnz     ready
                ;
                ;       cannot open file, so create it
0122 0e16               mvi     c,makef
0124 115c00             lxi     d,fcb
0127 cd0500             call    bdos
012a 3c                 inr     a           ;err 255 becomes zero
012b c23700             jnz     ready
                ;
                ;       cannot create file, directory full
012e 113a00             lxi     d,nospace
0131 cdda00             call    print
0134 c30000             jmp     reboot      ;back to ccp
```

Loop Back to Ready After Each Command

```
                ;
                ready:
                ;       file is ready for processing
                ;
0137 cde500             call    readcom     ;read next command
013a 227d00             shld    ranrec      ;store input record#
013d 217f00             lxi     h,ranovf
0140 3600               mvi     m,0         ;clear high byte if set
0142 fe51               cpi     'Q'         ;quit?
0144 c25600             jnz     notq
                ;
                ;       quit processing, close file
0147 0e10               mvi     c,closef
0149 115c00             lxi     d,fcb
014c cd0500             call    bdos
014f 3c                 inr     a           ;err 255 becomes 0
0150 cab900             jz      error       ;error message, retry
0153 c30000             jmp     reboot      ;back to ccp
                ;
```

End of Quit Command, Process Write

```
                          notq:
                          ;         not the quit command, random write?
0156 fe57                           cpi       'W'
0158 c28900                         jnz       notw
                          ;
                          ;         this is a random write, fill buffer until cr
015b 114d00                         lxi       d,datmsg
015e cdda00                         call      print     ;data prompt
0161 0e7f                           mvi       c,127     ;up to 127 characters
0163 218000                         lxi       h,buff    ;destination
                          rloop:    ;read next character to buff
0166 c5                             push      b         ;save counter
0167 e5                             push      h         ;next destination
0168 cdc200                         call      getchr    ;character to a
016b e1                             pop       h         ;restore counter
016c c1                             pop       b         ;restore next to fill
016d fe0d                           cpi       cr        ;end of line?
016f ca7800                         jz        erloop
                          ;         not end, store character
0172 77                             mov       m,a
0173 23                             inx       h         ;next to fill
0174 0d                             dcr       c         ;counter goes down
0175 c26600                         jnz       rloop     ;end of buffer?
                          erloop:
                          ;         end of read loop, store 00
0178 3600                           mvi       m,0
                          ;
                          ;         write the record to selected record number
017a 0e22                           mvi       c,writer
017c 115c00                         lxi       d,fcb
017f cd0500                         call      bdos
0182 b7                             ora       a         ;error code zero?
0183 c2b900                         jnz       error     ;message if not
0186 c33700                         jmp       ready     ;for another record
                          ;
```

End of Write Command, Process Read

```
                          notw:
                          ;         not a write command, read record?
0189 fe52                           cpi       'R'
018b c2b900                         jnz       error     ;skip if not
                          ;
                          ;         read random record
018e 0e21                           mvi       c,readr
0190 115c00                         lxi       d,fcb
0193 cd0500                         call      bdos
0196 b7                             ora       a         ;return code 00?
0197 c2b900                         jnz       error
                          ;
                          ;         read was successful, write to console
```

```
019a cdcf00                 call    crlf      ;new line
019d 0e80                   mvi     c,128     ;max 128 characters
019f 218000                 lxi     h,buff    ;next to get
                    wloop:
01a2 7e                     mov     a,m       ;next character
01a3 23                     inx     h         ;next to get
01a4 e67f                   ani     7fh       ;mask parity
01a6 ca3700                 jz      ready     ;for another command
                                              ;if 00
01a9 c5                     push    b         ;save counter
01aa e5                     push    h         ;save next to get
01ab fe20                   cpi     ' '       ;graphic?
01ad d4c800                 cnc     putchr    ;skip output if not
01b0 e1                     pop     h
01b1 c1                     pop     b
01b2 0d                     dcr     c         ;count=count-1
01b3 c2a200                 jnz     wloop
01b6 c33700                 jmp     ready
```

End of Read Command, All Errors End Up Here

```
                    ;
                    error:
01b9 115900                 lxi     d,errmsg
01bc cdda00                 call    print
01bf c33700                 jmp     ready
                    ;
```

Utility Subroutines for Console I/O

```
                    getchr:
                            ;read next console character to a
01c2 0e01                   mvi     c,coninp
01c4 cd0500                 call    bdos
01c7 c9                     ret
                    ;
                    putchr:
                            ;write character from a to console
01c8 0e02                   mvi     c,conout
01ca 5f                     mov     e,a       ;character to send
01cb cd0500                 call    bdos      ;send character
01ce c9                     ret
                    ;
                    crlf:
                            ;send carriage return line feed
01cf 3e0d                   mvi     a,cr      ;carriage return
01d1 cdc800                 call    putchr
01d4 3e0a                   mvi     a,lf      ;line feed
01d6 cdc800                 call    putchr
01d9 c9                     ret
```

```
                              ;
                              print:
                                        ;print the buffer addressed by de until $
01da d5                                 push    d
01db cdcf00                             call    crlf
01de d1                                 pop     d         ;new line
01df 0e09                               mvi     c,pstring
01e1 cd0500                             call    bdos      ;print the string
01e4 c9                                 ret
                              ;
                              readcom:
                                        ;read the next command line to the conbuf
01e5 116b00                             lxi     d,prompt
01e8 cdda00                             call    print     ;command?
01eb 0e0a                               mvi     c,rstring
01ed 117a00                             lxi     d,conbuf
01f0 cd0500                             call    bdos      ;read command line
                              ;         command line is present, scan it
01f3 210000                             lxi     h,0       ;start with 0000
01f6 117c00                             lxi     d,conlin  ;command line
01f9 1a                       readc:    ldax    d         ;next command
                                                          ;character
01fa 13                                 inx     d         ;to next command
                                                          ;position
01fb b7                                 ora     a         ;cannot be end of
                                                          ;command
01fc c8                                 rz
                              ;         not zero, numeric?
01fd d630                               sui     '0'
01ff fe0a                               cpi     10        ;carry if numeric
0201 d21300                             jnc     endrd
                              ;         add-in next digit
0204 29                                 dad     h         ;*2
0205 4d                                 mov     c,l
0206 44                                 mov     b,h       ;bc = value * 2
0207 29                                 dad     h         ;*4
0208 29                                 dad     h         ;*8
0209 09                                 dad     b         ;*2 + *8 = *10
020a 85                                 add     l         ;+digit
020b 6f                                 mov     l,a
020c d2f900                             jnc     readc     ;for another char
020f 24                                 inr     h         ;overflow
0210 c3f900                             jmp     readc     ;for another char
                              endrd:
                              ;         end of read, restore value in a
0213 c630                               adi     '0'       ;command
0215 fe61                               cpi     'a'       ;translate case?
0217 d8                                 rc
                              ;         lower case, mask lower case bits
0218 e65f                               ani     101$1111b
021a c9                                 ret
                              ;
```

String Data Area for Console Messages

```
                badver:
021b 536f79             db      'sorry, you need cp/m version 2$'
                nospace:
023a 4e6f29             db      'no directory space$'
                datmsg:
024d 547970             db      'type data: $'
                errmsg:
0259 457272             db      'error, try again.$'
                prompt:
026b 4e6570             db      'next command? $'
                ;
```

Fixed and Variable Data Area

```
027a 21         conbuf: db      conlen  ;length of console buffer
027b            consiz: ds      1       ;resulting size after read
027c            conlin: ds      32      ;length 32 buffer
0021 =          conlen  equ     $-consiz
                ;
029c                    ds      32     ´;16 level stack
                stack:
02bc                    end
```

Again, major improvements could be made to this particular program to enhance its operation. In fact, with some work, this program could evolve into a simple data base management system. One could, for example, assume a standard record size of 128 bytes, consisting of arbitrary fields within the record. A program, called GETKEY, could be developed that first reads a sequential file and extracts a specific field defined by the operator. For example, the command

```
GETKEY NAMES.DAT LASTNAME 10 20
```

would cause GETKEY to read the data base file NAMES.DAT and extract the "LASTNAME" field from each record, starting in position 10 and ending at character 20. GETKEY builds a table in memory consisting of each particular LASTNAME field, along with its 16-bit record number location within the file. The GETKEY program then sorts this list and writes a new file, called LASTNAME.KEY, which is an alphabetical list of LASTNAME fields with their corresponding record numbers. (This list is called an *inverted index* in information retrieval parlance.)

If the programmer were to rename the program shown above as QUERY and massage it so that it reads a sorted key file into memory, the command line might appear as

```
QUERY NAMES.DAT LASTNAME.KEY.
```

Instead of reading a number, the QUERY program reads an alphanumeric string that is a particular key to find in the NAMES.DAT data base. Since the LASTNAME.KEY list is sorted, one can find a particular entry rapidly by performing a "binary search," similar to looking up a name in the telephone book. That is, starting at both ends of the list, one examines the entry halfway in between and, if not matched, splits either the upper half or the lower half for the next search. The user will quickly reach the item he or she is looking for and find the corresponding record number. The user should fetch and display this record at the console, just as was done in the program shown above.

With some more work, the user can allow a fixed grouping size that differs from the 128-byte record shown above. This is accomplished by keeping track of the record number as well as the byte offset within the record. Knowing the group size, one randomly accesses the record containing the proper group, offset to the beginning of the group within the record read sequentially until the group size has been exhausted.

Finally, one can improve QUERY considerably by allowing boolean expressions, which compute the set of records that satisfy several relationships, such as a LASTNAME between HARDY and LAUREL and an AGE lower than 45. Display all the records that fit this description. Finally, if the user's lists are getting too big to fit into memory, he or she should randomly access key files from the disk as well.

5.6 System Function Summary

FUNCTION NUMBER Decimal	Hex	FUNCTION NAME	INPUT	OUTPUT
0	0	System Reset	C = 00H	none
1	1	Console Input	C = 01H	A = ASCII char
2	2	Console Output	E = char	none
3	3	Reader Input		A = ASCII char
4	4	Punch Output	E = char	none
5	5	List Output	E = char	none
6	6	Direct Console I/O	C = 06H E = OFFH (input) or OFEH (status) or char (output)	A = char or status (no value)
7	7	Get I/O Byte	none	A = I/O Byte Value
8	8	Set I/O Byte	E = I/O Byte	none
9	9	Print String	DE = Buffer Address	none
10	A	Read Console Buffer	DE = Buffer	Console Characters in Buffer
11	B	Get Console Status	none	A = 00/non zero
12	C	Return Version Number	none	HL: Version Number
13	D	Reset Disk System	none	none
14	E	Select Disk	E = Disk Number	none
15	F	Open File	DE = FCB Address	FF if not found
16	10	Close File	DE = FCB Address	FF if not found

CP/M®
OPERATING SYSTEM
MANUAL

CP/M 2 Alteration
Chapter 6

DIGITAL RESEARCH™
P.O. Box 579
Pacific Grove, California 93950

595-2839

DISCLAIMER

Digital Research makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, Digital Research reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Digital Research to notify any person of such revision or changes.

TRADEMARKS

CP/M is a registered trademark of Digital Research. MP/M, MAC, and SID are trademarks of Digital Research. Z-80 is a trademark of Zilog, Inc.

First Printing: July 1982

6
CP/M 2 Alteration

6.1 Introduction

The standard CP/M system assumes operation on an Intel MDS-800 microcomputer development system, but is designed so the user can alter a specific set of subroutines that define the hardware operating environment.

Although standard CP/M 2 is configured for single density floppy disks, field-alteration features allow adaptation to a wide variety of disk subsystems from single drive minidisks through high-capacity, "hard disk" systems. To simplify the following adaptation process, it is assumed that CP/M 2 will first be configured for single density floppy disks where minimal editing and debugging tools are available. If an earlier version of CP/M is available, the customizing process is eased considerably. In this latter case, the user may wish to review the system generation process and skip to later sections that discuss system alteration for nonstandard disk systems.

To achieve device independence, CP/M is separated into three distinct modules:

BIOS	basic I/O system, which is environment dependent
BDOS	basic disk operating system, which is not dependent upon the hardware configuration
CCP	the console command processor, which uses the BDOS

Of these modules, only the BIOS is dependent upon the particular hardware. That is, the user can "patch" the distribution version of CP/M to provide a new BIOS that provides a customized interface between the remaining CP/M modules and the user's own hardware system. This document provides a step-by-step procedure for patching a new BIOS into CP/M.

All disk-dependent portions of CP/M 2 are placed into a BIOS, a resident "disk parameter block," which is either hand coded or produced automatically using the disk definition macro library provided with CP/M 2. The end user need only specify the maximum number of active disks, the starting and ending sector numbers, the data allocation size, the maximum extent of the logical disk, directory size information, and reserved track values. The macros use this information to generate the appropriate tables and table references for use during CP/M 2 operation. Deblocking information is provided, which aids in assembly or disassembly of sector sizes that are multiples of the fundamental 128 byte data unit, and the system alteration manual includes general purpose subroutines that use the deblocking information to take advantage of larger sector sizes. Use of these subroutines, together with the table-drive data access algorithms, makes CP/M 2 a universal data management system.

File expansion is achieved by providing up to 512 logical file extents, where each logical extent contains 16K bytes of data. CP/M 2 is structured, however, so that as much as 128K bytes of data are addressed by a single physical extent (corresponding to a single directory entry) maintaining compatibility with previous versions while taking advantage of directory space.

If CP/M is being tailored to a computer system for the first time, the new BIOS requires some simple software development and testing. The standard BIOS is listed in Appendix A and can be used as a model for the customized package. A skeletal version of the BIOS given in Appendix B can serve as the basis for a modified BIOS. In addition to the BIOS, the user must write a simple memory loader, called GETSYS, that brings the operating system into memory. To patch the new BIOS into CP/M, the user must write the reverse of GETSYS, called PUTSYS, which places an altered version of CP/M back onto the diskette. PUTSYS can be derived from GETSYS by changing the disk read commands into disk write commands. Sample skeletal GETSYS and PUTSYS programs are described in Section 6.4 and listed in Appendix C. To make the CP/M system load automatically, the user must also supply a cold start loader, similar to the one provided with CP/M (listed in Appendices A and D). A skeletal form of a cold start loader is given in Appendix E, which serves as a model for the loader.

6.2 First Level System Regeneration

The procedure to patch the CP/M system is given below. Address references in each step are shown with "H" denoting the hexadecimal radix, and are given for a 20K CP/M system. For larger CP/M systems, a "bias" is added to each address that is shown with a "+b" following it, where b is equal to the memory size—20K. Values for b in various standard memory sizes are

```
24K:    b = 24K - 20K =  4K = 1000H
32K:    b = 32K - 20K = 12K = 3000H
40K:    b = 40K - 20K = 20K = 5000H
48K:    b = 48K - 20K = 28K = 7000H
56K:    b = 56K - 20K = 36K = 9000H
62K:    b = 62K - 20K = 42K = A800H
64K:    b = 64K - 20K = 44K = B000H
```

It should be noted that the standard distribution version of CP/M is set for operation within a 20K memory system. Therefore, the user must first bring up the 20K CP/M system, then configure it for actual memory size (the user should see Section 6.3).

The user should:

1. Read Section 6.4 and write a GETSYS program that reads the first two tracks of a diskette into memory. The program from the diskette must be loaded starting at location 3380H. GETSYS is coded to start at location 100H (base of the TPA), as shown in Appendix C.

2. Test the GETSYS program by reading a blank diskette into memory and check to see that the data have been read properly and that the diskette has not been altered in any way by the GETSYS program.

3. Run the GETSYS program using an initialized CP/M diskette to see if GETSYS loads CP/M starting at 3380H (the operating system actually starts 128 bytes later at 3400H).

4. Read Section 6.4 and write the PUTSYS program. This writes memory starting at 3380H back onto the first two tracks of the diskette. The PUTSYS program should be located at 200H, as shown in Appendix C.

5. Test the PUTSYS program using a blank, uninitialized diskette by writing a portion of memory to the first two tracks; clear memory and read it back using GETSYS. Test PUTSYS completely, since this program will be used to alter CP/M on disk.

6. Study Sections 6.5, 6.6, and 6.7 along with the distribution version of the BIOS given in Appendix A and write a simple version that performs a similar function for the customized environment. Use the program given in Appendix B as a model. Call this new BIOS by the name CBIOS (customized BIOS). Implement only the primitive disk operations on a single drive and simple console input/output functions in this phase.

7. Test CBIOS completely to ensure that it properly performs console character I/O and disk reads and writes. Be careful to ensure that no disk write operations occur during read operations and check that the proper track and sectors are addressed on all reads and writes. Failure to make these checks may cause destruction of the initialized CP/M system after it is patched.

8. Referring to the table in Section 6.5, note that the BIOS is placed between locations 4AOOH and 4FFFH. Read the CP/M system using GETSYS and replace the BIOS segment by the CBIOS developed in step 6 and tested in step 7. This replacement is done in memory.

9. Use PUTSYS to place the patched memory image of CP/M onto the first two tracks of a blank diskette for testing.

10. Use GETSYS to bring the copied memory image from the test diskette back into memory at 3380H and check to ensure that it has loaded back properly (clear memory, if possible, before the load). Upon successful load, branch to the cold start code at location 4A00H. The cold start routine will initialize page zero, then jump to the CCP at location 3400H, which will call the BDOS, which will call the CBIOS. The CBIOS will be asked by the CCP to read sixteen sectors on track 2, and CP/M will type "A>", the system prompt.

If difficulties are encountered, use whatever debug facilities are available to trace and breakpoint the CBIOS.

11. Upon completion of step 10, CP/M has prompted the console for a command input. Test the disk write operation by typing

```
SAVE 1 X.COM
```

(All commands must be followed by a carriage return.) CP/M responds with another prompt (after several disk accesses)

```
A>
```

If it does not, debug the disk write functions and retry.

12. Test the directory command by typing

```
DIR
```

CP/M responds with

```
A: X      COM
```

13. Test the erase command by typing

```
ERA X.COM
```

CP/M responds with the A prompt. This is now an operational system that only requires a bootstrap loader to function completely.

14. Write a bootstrap loader that is similar to GETSYS and place it on track 0, sector 1 using PUTSYS (again using the test diskette, not the distribution diskette). See Sections 6.5 and 6.8 for more information on the bootstrap operation.

15. Retest the new test diskette with the bootstrap loader installed by executing steps 11, 12, and 13. Upon completion of these tests, type a control-C (control and C keys simultaneously). The system executes a "warm start" that reboots the system, and types the A prompt.

16. At this point, there is probably a good version of the customized CP/M system on the test diskette. Use GETSYS to load CP/M from the test diskette. Remove the test diskette, place the distribution diskette (or a legal copy) into the drive, and use PUTSYS to replace the distribution version with the customized version. The user should not make this replacement if unsure of the patch because this step destroys the system that was obtained from Digital Research.

17. Load the modified CP/M system and test it by typing

```
DIR
```

CP/M responds with a list of files that are provided on the initialized diskette. One file is the memory image for the debugger

```
DDT.COM
```

Note that from now on, it is important always to reboot the CP/M system (ctl-C is sufficient) when the diskette is removed and replaced by another diskette, unless the new diskette is to be read only.

18. Load and test the debugger by typing

```
DDT
```

(See Chapter 4 for operating procedures.)

19. Before making further CBIOS modifications, practice using the editor (see Chapter 2), and assembler (see Chapter 3). Recode and test the GETSYS, PUTSYS, and CBIOS programs using ED, ASM, and DDT. Code and test a COPY program that does a sector-to-sector copy from one diskette to another to obtain back-up copies of the original diskette. (Read the CP/M Licensing Agreement specifying legal responsibilities when copying the CP/M system.) Place the copyright notice

Copyright ©, 1979
Digital Research

on each copy that is made with the COPY program.

20. Modify the CBIOS to include the extra functions for punches, readers, and sign-on messages, and add the facilities for additional disk drives, if desired. These changes can be made with the GETSYS and PUTSYS programs or by referring to the regeneration process in Section 6.3.

The user should now have a good copy of the customized CP/M system. Although the CBIOS portion of CP/M belongs to the user, the modified version cannot be legally copied for anyone else's use.

It should be noted that the system remains file-compatible with all other CP/M systems (assuming media compatibility), which allows transfer of nonproprietary software between CP/M users.

6.3 Second Level System Generation

Once the system is running, the user will want to configure CP/M for the desired memory size. Usually a memory image is first produced with the "MOVCPM" program (system relocator) and then placed into a named disk file. The disk file can then be loaded, examined, patched, and replaced using the debugger and the system generation program. (The user should refer to Chapter 1.)

The CBIOS and BOOT are modified using ED and assembled using ASM, producing files called CBIOS.HEX and BOOT. HEX, which contain the code for CBIOS and BOOT in Intel hex format.

To get the memory image of CP/M into the TPA configured for the desired memory size, the user should type the command

```
MOVCPM xx *
```

where xx is the memory size in decimal K bytes (e.g., 32 for 32K). The response will be

```
CONSTRUCTING xxK CP/M VERS 2.0
READY FOR "SYSGEN" OR
"SAVE 34 CPMxx.COM"
```

An image of CP/M in the TPA is configured for the requested memory size. The memory image is at location 0900H through 227FH (i.e., the BOOT is at 0900H, the CCP is at 980H, the BDOS starts at 1180H, and the BIOS is at 1F80H). The user should note that the memory image has the standard MDS-800 BIOS and BOOT on it. It is now necessary to save the memory image in a file so that the user can patch the CBIOS and CBOOT into it:

```
SAVE 34 CPMxx.COM
```

Upon completion of the read, the user should reexamine the area where the CBIOS has been loaded (use an "L1F80" command) to ensure that it was loaded properly. When satisfied that the change has been made, the user should return from DDT using a control-C or, "G0" command.

SYSGEN is used to replace the patched memory image back onto a diskette (the user should utilize a test diskette until sure of the patch), as shown in the following interaction:

SYSGEN	Start the SYSGEN program
SYSGEN VERSION 2.0	Sign-on message from SYSGEN
SOURCE DRIVE NAME (OR RETURN TO SKIP)	Respond with a carriage return to skip the CP/M read operation since the system is already in memory
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)	Respond with "B" to write the new system to the diskette in drive B
DESTINATION ON B, THEN TYPE RETURN	Place a scratch diskette in drive B, then type return.
FUNCTION COMPLETE DESTINATION DRIVE NAME (OR RETURN TO REBOOT)	

The user should place the scratch diskette in drive A and then perform a cold start to bring up the newly configured CP/M system.

The new CP/M system is then tested and the Digital Research copyright notice is placed on the diskette, as specified in the Licensing Agreement:

Copyright ©, 1979
Digital Research

6.4 Sample GETSYS and PUTSYS Programs

The following program provides a framework for the GETSYS and PUTSYS programs referenced in Sections 6.1 and 6.2. The READSEC and WRITESEC subroutines must be inserted by the user to read and write the specific sectors.

```
;   GETSYS PROGRAM — READ TRACKS 0 AND 1 TO MEMORY AT 3380H
;   REGISTER                         USE
;               A                (SCRATCH REGISTER)
;               B                TRACK COUNT (0, 1)
;               C                SECTOR COUNT (1,2,. . .,26)
;               DE               (SCRATCH REGISTER PAIR)
;               HL               LOAD ADDRESS
;               SP               SET TO STACK ADDRESS
;
START:          LXI  SP,3380H    ;SET STACK POINTER TO SCRATCH
                                 ;AREA
                LXI  H, 3380H    ;SET BASE LOAD ADDRESS
                MVI  B, 0        ;START WITH TRACK 0
```

```
RDTRK:                      ;READ NEXT TRACK (INITIALLY 0)
            MVI C,1         ;READ STARTING WITH SECTOR 1
RDSEC:                      ;READ NEXT SECTOR
            CALL READSEC    ;USER-SUPPLIED SUBROUTINE
            LXI  D,128      ;MOVE LOAD ADDRESS TO NEXT 1/2
                            ;PAGE
            DAD D           ;HL = HL + 128
            INR C           ;SECTOR = SECTOR + 1
            MOV A,C         ;CHECK FOR END OF TRACK
            CPI 27
            JC   RDSEC      ;CARRY GENERATED IF SECTOR < 27
;
;   ARRIVE HERE AT END OF TRACK, MOVE TO NEXT TRACK
            INR B
            MOV A,B         ;TEST FOR LAST TRACK
            CPI 2
            JC   RDTRK      ;CARRY GENERATED IF TRACK < 2
;
;   ARRIVE HERE AT END OF LOAD, HALT FOR NOW
            HLT
;
;   USER-SUPPLIED SUBROUTINE TO READ THE DISK
READSEC:
;   ENTER WITH TRACK NUMBER IN REGISTER B,
;           SECTOR NUMBER IN REGISTER C, AND
;           ADDRESS TO FILL IN HL
;
            PUSH B          ;SAVE B AND C REGISTERS
            PUSH H          ;SAVE HL REGISTERS
            ..........................................................
            perform disk read at this point, branch to
            label START if an error occurs
            ..........................................................
            POP H           ;RECOVER HL
            POP B           ;RECOVER B AND C REGISTERS
            RET             ;BACK TO MAIN PROGRAM

            END START
```

This program is assembled and listed in Appendix B for reference purposes, with an assumed origin of 100H. The hexadecimal operation codes that are listed on the left may be useful if the program has to be entered through the panel switches.

The PUTSYS program can be constructed from GETSYS by changing only a few operations in the GETSYS program given above, as shown in Appendix C. The register pair HL becomes the dump address (next address to write), and operations upon these registers do not change within the program. The READSEC subroutine is replaced by a WRITESEC subroutine, which performs the opposite function: data from address HL are written to the track given by register B and sector given by register C. It is often useful to combine GETSYS and PUTSYS into a single program during the test and development phase, as shown in Appendix C.

4A24H+b	JMP SETDMA	; SET DMA ADDRESS
4A27H+b	JMP READ	; READ SELECTED SECTOR
4A2AH+b	JMP WRITE	; WRITE SELECTED SECTOR
4A2DH+b	JMP LISTST	; RETURN LIST STATUS
4A30H+b	JMP SECTRAN	; SECTOR TRANSLATE SUBROUTINE

Each jump address corresponds to a particular subroutine that performs the specific function, as outlined below. There are three major divisions in the jump table: the system (re)initialization, which results from calls on BOOT and WBOOT; simple character I/O performed by calls on CONST, CONIN, CONOUT, LIST, PUNCH, READER, and LISTST; and diskette I/O performed by calls on HOME, SELDSK, SETTRK, SETSEC, SETDMA, READ, WRITE, and SECTRAN.

All simple character I/O operations are assumed to be performed in ASCII, upper and lower case, with high order (parity bit) set to zero. An end-of-file condition for an input device is given by an ASCII control-z (1AH). Peripheral devices are seen by CP/M as "logical" devices and are assigned to physical devices within the BIOS.

To operate, the BDOS needs only the CONST, CONIN, and CONOUT subroutines (LIST, PUNCH, and READER may be used by PIP, but not the BDOS). Further, the LISTST entry is currently used only by DESPOOL, the print spooling utility. Thus, the initial version of CBIOS may have empty subroutines for the remaining ASCII devices.

The characteristics of each device are

CONSOLE	The principal interactive console that communicates with the operator, accessed through CONST, CONIN, and CONOUT; typically, the CONSOLE is a device such as a CRT or teletype.
LIST	The principal listing device, if it exists on the user's system, is usually a hard-copy device, such as a printer or teletype.
PUNCH	The principal tape punching device, if it exists, is normally a high-speed paper tape punch or teletype.
READER	The principal tape reading device, such as a simple optical reader or teletype.

A single peripheral can be assigned as the LIST, PUNCH, and READER device simultaneously. If no peripheral device is assigned as the LIST, PUNCH, or READER device, the CBIOS created by the user may give an appropriate error message so that the system does not "hang" if the device is accessed by PIP or some other user program. Alternately, the PUNCH and LIST routines can just simply return, and the READER routine can return with a 1AH (ctl-Z) in register A to indicate immediate end-of-file.

For added flexibility, the user can optionally implement the "IOBYTE" function, which allows reassignment of physical and logical devices. The IOBYTE function creates a mapping of logical to physical devices that can be altered during CP/M processing (the user should see the STAT command). The definition of the IOBYTE function corresponds to the Intel standard as follows: a single location in memory (currently location 0003H) is maintained, called IOBYTE, which defines the logical to physical device mapping that is in effect at a particular time. The mapping is performed by splitting the

IOBYTE into four distinct fields of two bits each, called the CONSOLE, READER, PUNCH, and LIST fields, as shown below.

	most significant			least significant
IOBYTE AT 003H	LIST	PUNCH	READER	CONSOLE
	bits 6, 7	bits 4, 5	bits 2, 3	bits 0, 1

The value in each field can be in the range 0-3, defining the assigned source or destination of each logical device. The values that can be assigned to each field are given below

CONSOLE field (bits 0,1)

- 0 console is assigned to the console printer device (TTY:)
- 1 console is assigned to the CRT device (CRT:)
- 2 batch mode: use the READER as the CONSOLE input, and the LIST device as the CONSOLE output (BAT:)
- 3 user defined console device (UC1:)

READER field (bits 2,3)

- 0 READER is the teletype device (TTY:)
- 1 READER is the high speed reader device (PTR:)
- 2 user defined reader # 1 (UR1:)
- 3 user defined reader # 2 (UR2:)

PUNCH field (bits 4,5)

- 0 PUNCH is the teletype device (TTY:)
- 1 PUNCH is the high speed punch device (PTP:)
- 2 user defined punch # 1 (UP1:)
- 3 user defined punch # 2 (UP2:)

LIST field (bits 6,7)

- 0 LIST is the teletype device (TTY:)
- 1 LIST is the CRT device (CRT:)
- 2 LIST is the line printer device (LPT:)
- 3 user defined list device (UL1:)

The implementation of the IOBYTE is optional and affects only the organization of the CBIOS. No CP/M systems use the IOBYTE (although they tolerate the existence of the IOBYTE at location 0003H), except for PIP, which allows access to the physical devices, and STAT, which allows logical-physical assignments to be made or displayed (for more information, the user should see Chapter 1). In any case the IOBYTE implementation should be omitted until the basic CBIOS is fully implemented and tested; then the user should add the IOBYTE to increase the facilities.

Disk I/O is always performed through a sequence of calls on the various disk access subroutines that set up the disk number to access, the track and sector on a particular disk, and the direct memory access (DMA) address involved in the I/O operation. After all these parameters have been set up, a call is made to the READ or WRITE function to perform the actual I/O operation. There is often a single call to SELDSK to select a disk drive, followed by a number of read or write operations to the selected disk before selecting another drive for subsequent operations. Similarly, there may be a single call to set the DMA address, followed by several calls that read or write from the selected DMA address before the DMA address is changed. The track and sector subroutines are always called before the READ or WRITE operations are performed.

The READ and WRITE routines should perform several retries (10 is standard) before reporting the error condition to the BDOS. If the error condition is returned to the BDOS, it will report the error to the user. The HOME subroutine may or may not actually perform the track 00 seek, depending upon controller characteristics; the important point is that track 00 has been selected for the next operation and is often treated in exactly the same manner as SETTRK with a parameter of 00.

The exact responsibilities of each entry point subroutine are given below.

BOOT — The BOOT entry point gets control from the cold start loader and is responsible for basic system initialization, including sending a sign-on message (which can be omitted in the first version). If the IOBYTE function is implemented, it must be set at this point. The various system parameters that are set by the WBOOT entry point must be initialized, and control is transferred to the CCP at 3400+b for further processing. Note that register C must be set to zero to select drive A.

WBOOT — The WBOOT entry point gets control when a warm start occurs. A warm start is performed whenever a user program branches to location 0000H, or when the CPU is reset from the front panel. The CP/M system must be loaded from the first two tracks of drive A up to, but not including, the BIOS (or CBIOS, if the user has completed the patch). System parameters must be initialized as shown below:

location 0,1,2	Set to JMP WBOOT for warm starts (000H: JMP 4A03H+b)
location 3	Set initial value of IOBYTE, if implemented in the CBIOS
location 4	High nibble = current user no; low nibble = current drive
location 5,6,7	Set to JMP BDOS, which is the primary entry point to CP/M for transient programs. (0005H: JMP 3C06H+b)

(The user should refer to Section 6.9 for complete details of page zero use.) Upon completion of the initialization, the WBOOT program must branch to the CCP at 3400H+b to (re)start the system. Upon entry to the CCP, register C is set to the drive to select after system initialization. The WBOOT routine should read location 4 in memory, verify that it is a legal drive, and pass it to the CCP in register C.

CONST — The user should sample the status of the currently assigned console device and return 0FFH in register A if a character is ready to read and 00H in register A if no console characters are ready.

CONIN — The next console character is read into register A, and the parity bit is set (high order bit) to zero. If no console character is ready, the user waits until a character is typed before returning.

CONOUT — The user sends the character from register C to the console output device. The character is in ASCII, with high order parity bit set to zero. The user may want to include a time-out on a line feed or carriage return, if the console device requires some time interval at the end of the line (such as a TI Silent 700 terminal). The user can filter out control characters that cause the console device to react in a strange way (a control-z causes the Lear Seigler terminal to clear the screen, for example).

LIST — The user sends the character from register C to the currently assigned listing device. The character is in ASCII with zero parity bit.

PUNCH — The user sends the character from register C to the currently assigned punch device. The character is in ASCII with zero parity.

READER — The user reads the next character from the currently assigned reader device into register A with zero parity (high order bit must be zero); an end-of-file condition is reported by returning an ASCII control-z(1AH).

HOME — The user moves the disk head of the currently selected disk (initially disk A) to the track 00 position. If the controller allows access to the track 0 flag from the drive, the head is stepped until the track 0 flag is detected. If the controller does not support this feature, the HOME call is translated into a call to SETTRK with a parameter of 0.

SELDSK — The user selects the disk drive given by register C for further operations, where register C contains 0 for drive A, 1 for drive B, and so on up to 15 for drive P (the standard CP/M distribution version supports four drives). On each disk select, SELDSK must return in HL the base address of a 16-byte area, called the Disk Parameter Header, described in Section 6.10. For standard floppy disk drives, the contents of the header and associated tables do not change; thus, the program segment included in the sample CBIOS performs this operation automatically. If there is an attempt to select a nonexistent drive, SELDSK returns HL=0000H as an error indicator. Although SELDSK must return the header address on each call, it is advisable to postpone the physical disk select operation until an I/O function (seek, read, or write) is actually performed, since disk selects often occur without utimately performing any disk I/O, and many controllers will unload the head of the current disk before selecting the new drive. This would cause an excessive amount of noise and disk wear. The least significant bit of register E is zero if this is the first occurrence of the drive select since the last cold or warm start.

SETTRK — Register BC contains the track number for subsequent disk accesses on the currently selected drive. The sector number in BC is the same as the number returned from the SECTRAN entry point. The user can choose to seek the selected track at this time or delay the seek until the next read or write actually occurs. Register BC can take on values in the range 0-76 corresponding to valid track numbers for standard floppy disk drives and 0-65535 for nonstandard disk subsystems.

SETSEC

Register BC contains the sector number (1 through 26) for subsequent disk accesses on the currently selected drive. The sector number in BC is the same as the number returned from the SECTRAN entry point. The user can choose to send this information to the controller at this point or delay sector selection until a read or write operation occurs.

SETDMA

Register BC contains the DMA (disk memory access) address for subsequent read or write operations. For example, if B = 00H and C = 80H when SETDMA is called, all subsequent read operations read their data into 80H through 0FFH and all subsequent write operations get their data from 80H through 0FFH, until the next call to SETDMA occurs. The initial DMA address is assumed to be 80H. The controller need not actually support direct memory access. If, for example, all data transfers are through I/O ports, the CBIOS that is constructed will use the 128-byte area starting at the selected DMA address for the memory buffer during the subsequent read or write operations.

READ

Assuming the drive has been selected, the track has been set, the sector has been set, and the DMA address has been specified, the READ subroutine attempts to read one sector based upon these parameters and returns the following error codes in register A:

0 no errors occurred

1 nonrecoverable error condition occurred

Currently, CP/M responds only to a zero or nonzero value as the return code. That is, if the value in register A is 0, CP/M assumes that the disk operation was completed properly. If an error occurs, however, the CBIOS should attempt at least 10 retries to see if the error is recoverable. When an error is reported the BDOS will print the message "BDOS ERR ON x: BAD SECTOR". The operator then has the option of typing carriage-return to ignore the error, or ctl-C to abort.

WRITE

The user writes the data from the currently selected DMA address to the currently selected drive, track, and sector. For floppy disks, the data should be marked as "nondeleted data" to maintain compatibility with other CP/M systems. The error codes given in the READ command are returned in register A, with error recovery attempts as described above.

LISTST

The user returns the ready status of the list device used by the DESPOOL program to improve console response during its operation. The value 00 is returned in A if the list device is not ready to accept a character and 0FFH if a character can be sent to the printer. A 00 value should be returned if LIST status is not implemented.

SECTRAN — The user performs logical to physical sector translation to improve the overall response of CP/M. Standard CP/M systems are shipped with a "skew factor" of 6, where six physical sectors are skipped between each logical read operation. This skew factor allows enough time between sectors for most programs to load their buffers without missing the next sector. In particular computer systems that use fast processors, memory, and disk subsystems, the skew factor may be changed to improve overall response. However, the user should mtaintain a single density IBM-compatible version of CP/M for information transfer into and out of the computer system, using a skew factor of 6. In general, SECTRAN receives a logical sector number relative to zero in BC and a translate table address in DE. The sector number is used as an index into the translate table, with the resulting physical sector number in HL. For standard systems, the table and indexing code is provided in the CBIOS and need not be changed.

6.7 A Sample BIOS

The program shown in Appendix B can serve as a basis for a user's first BIOS. The simplest functions are assumed in this BIOS, so that the user can enter it through a front panel, if absolutely necessary. The user must alter and insert code into the subroutines for CONST, CONIN, CONOUT, READ, WRITE, and WAITIO subroutines. Storage is reserved for user-supplied code in these regions. The scratch area reserved in page zero (see section 6.9) for the BIOS is used in this program, so that it could be implemented in ROM, if desired.

Once operational, this skeletal version can be enhanced to print the initial sign-on message and perform better error recovery. The subroutines for LIST, PUNCH, and READER can be filled out and the IOBYTE function can be implemented.

6.8 A Sample Cold Start Loader

The program shown in Appendix E can serve as a basis for a cold start loader. The disk read function must be supplied by the user, and the program must be loaded somehow starting at location 0000. Space is reserved for the patch code so that the total amount of storage required for the cold start loader is 128 bytes. Eventually, the user will probably want to get this loader onto the first disk sector (track 0, sector 1) and cause the controller to load it into memory automatically upon system start up. Alternatively, the cold start loader can be placed into ROM, and above the CP/M system. In this case, it will be necessary to originate the program at a higher address and key in a jump instruction at system start up that branches to the loader. Subsequent warm starts will not require this key-in operation, since the entry point WBOOT gets control thus bringing the system in from disk automatically. The skeletal cold start loader has minimal error recover, which may be enhanced in later versions.

6.9 Reserved Locations in Page Zero

Main memory page zero, between locations 00H and 0FFH, contains several segments of code and data that are used during CP/M processing. The code and data areas are given below for reference

Locations from to	Contents
0000H-0002H	Contains a jump instruction to the warm start entry point at location 4A03H+b. This allows a simple programmed restart (JMP 0000H) or manual restart from the front panel.
0003H-0003H	Contains the Intel standard IOBYTE, which is optionally included in the user's CBIOS, as described in Section 6.6.
0004H-0004H	Current default drive number (0=A,...,15=P).
0005H-0007H	Contains a jump instruction to the BDOS and serves two purposes: JMP 0005H provides the primary entry point to the BDOS, as described in Chapter 5, and LHLD 0006H brings the address field of the instruction to the HL register pair. This value is the lowest address in memory used by CP/M (assuming the CCP is being overlaid). The DDT program will change the address field to reflect the reduced memory size in debug mode.
0008H-0027H	(Interrupt locations 1 through 5 not used.)
0030H-0037H	(Interrupt location 6, not currently used; reserved.)
0038H-003AH	Restart 7; contains a jump instruction into the DDT or SID program when running in debug mode for programmed breakpoints, but is not otherwise used by CP/M.
003BH-003FH	(Not currently used; reserved.)
0040H-004FH	A 16-byte area reserved for scratch by CBIOS, but is not used for any purpose in the distribution version of CP/M.
0050H-005BH	(Not currently used; reserved.)
005CH-007CH	Default file control block produced for a transient program by the Console Command Processor.
007DH-007FH	Optional default random record position.
0080H-00FFH	Default 128-byte disk buffer (also filled with the command line when a transient is loaded under the CCP).

This information is set up for normal operation under the CP/M system, but can be overwritten by a transient program if the BDOS facilities are not required by the transient.

If, for example, a particular program performs only simple I/O and must begin execution at location 0, it can first be loaded into the TPA, using normal CP/M facilities, with a small memory move program that gets control when loaded (the memory move program must get control from location 0100H, which is the assumed beginning of all transient programs). The move program can then proceed to move the entire memory image down to location 0 and pass control to the starting address of the memory load. If the BIOS is overwritten or if location 0 (containing the warm start entry point) is overwritten, the operator must bring the CP/M system back into memory with a cold start sequence.

6.10 Disk Parameter Tables

Tables are included in the BIOS that describe the particular characteristics of the disk subsystem used with CP/M. These tables can be either hand-coded, as shown in the sample CBIOS in Appendix B, or automatically generated using the DISKDEF macro library, as shown in Appendix F. The purpose here is to describe the elements of these tables.

In general, each disk drive has an associated (16-byte) disk parameter header that contains information about the disk drive and provides a scratchpad area for certain BDOS operations. The format of the disk parameter header for each drive is shown below.

Disk Parameter Header

XLT	0000	0000	0000	DIRBUF	DPB	CSV	ALV
16b	16b	16b	16b	16b	16b	16b	16b

where each element is a word (16-bit) value. The meaning of each Disk Parameter Header (DPH) element is

XLT	Address of the logical to physical translation vector, if used for this particular drive, or the value 0000H if no sector translation takes place (i.e., the physical and logical sector numbers are the same). Disk drives with identical sector skew factors share the same translate tables.
0000	Scratchpad values for use within the BDOS (initial value is unimportant).
DIRBUF	Address of a 128-byte scratchpad area for directory operations within BDOS. All DPHs address the same scratchpad area.
DPB	Address of a disk parameter block for this drive. Drives with identical disk characteristics address the same disk parameter block.
CSV	Address of a scratchpad area used for software check for changed disks. This address is different for each DPH.
ALV	Address of a scratchpad area used by the BDOS to keep disk storage allocation information. This address is different for each DPH.

Given n disk drives, the DPHs are arranged in a table whose first row of 16 bytes corresponds to drive 0, with the last row corresponding to drive n-1. The table thus appears as

```
DPBASE:
  00 | XLT 00  | 0000  | 0000  | 0000  | DIRBUF| DBP 00| CSV 00| ALV 00|
  01 | XLT 01  | 0000  | 0000  | 0000  | DIRBUF| DBP 01| CSV 01| ALV 01|
                          (and so on through)
 n-1 | XLTn-1  | 0000  | 0000  | 0000  | DIRBUF| DBPn-1| CSVn-1| ALVn-1|
```

where the label DPBASE defines the base address of the DPH table.

A responsibility of the SELDSK subroutine is to return the base address of the DPH for the selected drive. The following sequence of operations returns the table address, with a 0000H returned if the selected drive does not exist.

```
NDISKS      EQU     4           ;NUMBER OF DISK DRIVES
......
SELDSK:     ;SELECT DISK GIVEN BY BC
            LXI     H,0000H     ;ERROR CODE
            MOV     A,C         ;DRIVE OK?
            CPI     NDISKS      ;CY IF SO
            RNC                 ;RET IF ERROR
            ;NO ERROR, CONTINUE
            MOV     L,C         ;LOW(DISK)
            MOV     H,B         ;HIGH(DISK)
            DAD     H           ;*2
            DAD     H           ;*4
            DAD     H           ;*8
            DAD     H           ;*16
            LXI     D,DPBASE;FIRST DPH
            DAD     D           ;DPH(DISK)
            RET
```

The translation vectors (XLT 00 through XLTn-1) are located elsewhere in the BIOS, and simply correspond one-for-one with the logical sector numbers zero through the sector count 1. The Disk Parameter Block (DPB) for each drive is more complex. A particular DPB, which is addressed by one or more DPHs, takes the general form

SPT	BSH	BLM	EXM	DSM	DRM	AL0	AL1	CKS	OFF
16b	8b	8b	8b	16b	16b	8b	8b	16b	16b

where each is a byte or word value, as shown by the 8b or 16b indicator below the field.

SPT is the total number of sectors per track.

BSH is the data allocation block shift factor, determined by the data block allocation size.

BLM is the data allocation block mask (2[BSH-1]).

EXM is the extent mask, determined by the data block allocation size and the number of disk blocks.

DSM determines the total storage capacity of the disk drive.

DRM determines the total number of directory entries that can be stored on this drive. (AL0,AL1 determine reserved directory blocks.)

CKS is the size of the directory check vector.

OFF is the number of reserved tracks at the beginning of the (logical) disk.

The values of BSH and BLM determine (implicitly) the data allocation size BLS, which is not an entry in the DPB. Given that the designer has selected a value for BLS, the values of BSH and BLM are shown in the tabulation below.

BLS	BSH	BLM
1024	3	7
2048	4	15
4096	5	31
8192	6	63
16384	7	127

where all values are in decimal. The value of EXM depends upon both the BLS and whether the DSM value is less than 256 or greater than 255. For DSM < 256 the value of EXM is given by:

BLS	EXM
1024	0
2048	1
4096	3
8192	7
16384	15

For DSM > 255 the value of EXM is given by:

BLS	EXM
1024	N/A
2048	0
4096	1
8192	3
16384	7

The value of DSM is the maximum data block number supported by this particular drive, measured in BLS units. The product BLS times (DSM+1) is the total number of bytes held by the drive and, of course, must be within the capacity of the physical disk, not counting the reserved operating system tracks.

The DRM entry is the one less than the total number of directory entries that can take on a 16-bit value. The values of AL0 and AL1, however, are determined by DRM. The values AL0 and AL1 can together be considered a string of 16-bits, as shown below.

AL0								AL1							
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15

where position 00 corresponds to the high order bit of the byte labeled AL0 and 15 corresponds to the low order bit of the byte labeled AL1. Each bit position reserves a data block for number of directory entries, thus allowing a total of 16 data blocks to be assigned for directory entries (bits are assigned starting at 00 and filled to the right until position 15). Each directory entry occupies 32 bytes, resulting in the tabulation below.

BLS	Directory Entries
1024	32 times # bits
2048	64 times # bits
4096	128 times # bits
8192	256 times # bits
16384	512 times # bits

Thus, if DRM = 127 (128 directory entries) and BLS = 1024, there are 32 directory entries per block, requiring 4 reserved blocks. In this case, the 4 high order bits of AL0 are set, resulting in the values AL0 = 0F0H and AL1 = 00H.

The CKS value is determined as follows: if the disk drive media is removable, then CKS = (DRM+1)/4, where DRM is the last directory entry number. If the media are fixed, then set CKS = 0 (no directory records are checked in this case).

Finally, the OFF field determines the number of tracks that are skipped at the beginning of the physical disk. This value is automatically added whenever SETTRK is called and can be used as a mechanism for skipping reserved operating system tracks or for partitioning a large disk into smaller segmented sections.

To complete the discussion of the DPB, several DPHs can address the same DPB if their drive characteristics are identical. Further, the DPB can be dynamically changed when a new drive is addressed by simply changing the pointer in the DPH since the BDOS copies the DPB values to a local area whenever the SELDSK function is invoked.

Returning back to the DPH for a particular drive, the two address values CSV and ALV remain. Both addresses reference an area of uninitialized memory following the BIOS. The areas must be unique for each drive, and the size of each area is determined by the values in the DPB.

The size of the area addressed by CSV is CKS bytes, which is sufficient to hold the directory check information for this particular drive. If CKS = (DRM+1)/4, one must reserve (DRM+1)/4 bytes for directory check use. If CKS = 0, no storage is reserved.

The size of the area addressed by ALV is determined by the maximum number of data blocks allowed for this particular disk and is computed as (DSM/8)+1.

The CBIOS shown in Appendix B demonstrates an instance of these tables for standard 8-inch single density drives. It may be useful to examine this program and compare the tabular values with the definitions given above.

6.11 The DISKDEF Macro Library

A macro library is shown in Appendix F, called DISKDEF, which greatly simplifies the table construction process. One must have access to the MAC macro assembler, of course, to use the DISKDEF facility, while the macro library is included with all CP/M 2 distribution disks.

A BIOS disk definition consists of the following sequence of macro statements:

```
MACLIB          DISKDEF
......
DISKS           n
DISKDEF         0,...
DISKDEF         1,...
```

```
......
DISKDEF        n-1
......
ENDEF
```

where the MACLIB statement loads the DISKDEF.LIB file (on the same disk as the BIOS) into MAC's internal tables. The DISKS macro call follows, which specifies the number of drives to be configured with the user's system, where n is an integer in the range 1 to 16. A series of DISKDEF macro calls then follow that define the characteristics of each logical disk, 0 through n-1 (corresponding to logical drives A through P). The DISKS and DISKDEF macros generate the in-line fixed data tables described in the previous section and thus must be placed in a nonexecutable portion of the BIOS, typically directly following the BIOS jump vector.

The remaining portion of the BIOS is defined following the DISKDEF macros, with the ENDEF macro call immediately preceding the END statement. The ENDEF (End of Diskdef) macro generates the necessary uninitialized RAM areas, which are located in memory above the BIOS.

The form of the DISKDEF macro call is

```
DISKDEF  dn,fsc,lsc,[skf],bls,dks,dir,cks,ofs,[0]
```

where

dn	is the logical disk number, 0 to n-1.
fsc	is the first physical sector number (0 or 1).
lsc	is the last sector number.
skf	is the optional sector skew factor.
bls	is the data allocation block size.
dks	is the number of blocks on the disk.
dir	is the number of directory entries.
cks	is the number of "checked" directory entries.
ofs	is the track offset to logical track 00.
[0]	is an optional 1.4 compatibility flag.

The value dn is the drive number being defined with this DISKDEF macro invocation. The fsc parameter accounts for differing sector numbering systems and is usually 0 or 1. The lsc is the last numbered sector on a track. When present, the skf parameter defines the sector skew factor, which is used to create a sector translation table accodrding to the skew.

If the number of sectors is less than 256, a single-byte table is created, otherwise each translation table element occupies two bytes. No translation table is created if the skf parameter is omitted (or equal to 0). The bls parameter specifies the number of bytes allocated to each data block, and takes on the values 1024, 2048, 4096, 8192, or 16384. Generally, performance increases with larger data block sizes since there are fewer directory references and logically connected data records are physically close on the disk. Further, each directory entry addresses more data and the BIOS-resident ram space is reduced.

The dks parameter specifies the total disk size in bls units. That is, if the bls = 2048 and dks = 1000, the total disk capacity is 2,048,000 bytes. If dks is greater than 255, the block size parameter bls must be greater than 1024. The value of dir is the total number of directory entries, which may exceed 255, if desired. The cks parameter determines the number of directory items to check on each directory scan and is used internally to detect changed disks during system operation, where an intervening cold or warm start has not occurred (when this situation is detected, CP/M automatically marks the disk read/only so that data are not subsequently destroyed).

As stated in the previous section, the value of cks = dir when the medium is easily changed, as is the case with a floppy disk subsystem. If the disk is permanently mounted, the value of cks is typically 0, since the probability of changing disks without a restart is low. The ofs value determines the number of tracks to skip when this particular drive is addressed, which can be used to reserve additional operating system space or to simulate several logical drives on a single large capacity physical drive. Finally, the [0] parameter is included when file compatibility is required with versions of 1.4 that have been modified for higher density disks. This parameter ensures that only 16K is allocated for each directory record, as was the case for previous versions. Normally, this parameter is not included.

For convenience and economy of table space, the special form

```
DISKDEF        i,j
```

gives disk i the same characteristics as a previously defined drive j. A standard four-drive single density system, which is compatible with version 1.4, is defined using the following macro invocations:

```
DISKS          4
DISKDEF        0,1,26,6,1024,243,64,64,2
DISKDEF        1,0
DISKDEF        2,0
DISKDEF        3,0
....
ENDEF
```

with all disks having the same parameter values of 26 sectors per track (numbered 1 through 26), with 6 sectors skipped between each access, 1024 bytes per data block, 243 data blocks for a total of 243K-byte disk capacity, 64 checked directory entries, and two operating system tracks.

The DISKS macro generates n DPHs, starting at the DPH table address DPBASE generated by the macro. Each disk header block contains sixteen bytes, as described above, and correspond one-for-one to each of the defined drives. In the four-drive standard system, for example, the DISKS macro generates a table of the form:

```
DPBASE         EQU $
DPE0:          DW  XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV0,ALV0
DPE1:          DW  XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV1,ALV1
DPE2:          DW  XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV2,ALV2
DPE3:          DW  XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV3,ALV3
```

where the DPH labels are included for reference purposes to show the beginning table addresses for each drive 0 through 3. The values contained within the DPH are described in detail in the previous section. The check and allocation vector addresses are generated by the ENDEF macro in the ram area following the BIOS code and tables.

The user should note that if the skf (skew factor) parameter is omitted (or equal to 0), the translation table is omitted and a 0000H value is inserted in the XLT position of the DPH for the disk. In a subsequent call to perform the logical to physical translation, SECTRAN receives a translation table address of DE = 0000H and simply returns the original logical sector from BC in the HL register pair. A translate table is constructed when the skf parameter is present, and the (nonzero) table address is placed into the corresponding DPHs. The tabulation shown below, for example, is constructed when the standard skew factor skf = 6 is specified in the DISKDEF macro call:

```
XLT0:   DB   1,7,13,19,25,5,11,17,23,3,9,15,21
        DB   2,8,14,20,26,6,12,18,24,4,10,16,22
```

Following the ENDEF macro call, a number of uninitialized data areas are defined. These data areas need not be a part of the BIOS that is loaded upon cold start, but must be available between the BIOS and the end of memory. The size of the uninitialized RAM area is determined by EQU statements generated by the ENDEF macro. For a standard four-drive system, the ENDEF macro might produce

```
4C72 =    BEGDAT EQU $
          (data areas)
4DB0 =    ENDDAT EQU $
013C =    DATSIZ EQU $-BEGDAT
```

which indicates that uninitialized RAM begins at location 4C72H, ends at 4DB0H-1, and occupies 013CH bytes. The user must ensure that these addresses are free for use after the system is loaded.

After modification, the user can utilize the STAT program to check drive characteristics, since STAT uses the disk parameter block to decode the drive information. The STAT command form

```
STAT d:DSK:
```

decodes the disk parameter block for drive d (d=A,...,P) and displays the values shown below.

r: 128-byte record capacity
k: kilobyte drive capacity
d: 32-byte directory entries
c: checked directory entries
e: records/extent
b: records/block
s: sectors/track
t: reserved tracks

Three examples of DISKDEF macro invocations are shown below with corresponding STAT parameter values (the last produces a full 8-megabyte system).

```
           DISKDEF 0,1,58,,2048,256,128,128,2
r=4096,    k=512, d=128, c=128, e=256, b=16, s=58, t=2

           DISKDEF 0,1,58,,2048,1024,300,0,2
r=16384,   k=2048, d=300, c=0, e=128, b=16, s=58, t=2

           DISKDEF 0,1,58,,16384,512,128,128,2
r=65536,   k=8192, d=128, c=128, e=1024, b=128, s=58, t=2
```

6.12 Sector Blocking and Deblocking

Upon each call to the BIOS WRITE entry point, the CP/M BDOS includes information that allows effective sector blocking and deblocking where the host disk subsystem has a sector size that is a multiple of the basic 128-byte unit. The purpose here is to present a general-purpose algorithm that can be included within the BIOS and that uses the BDOS information to perform the operations automatically.

On each call to WRITE, the BDOS provides the following information in register C:

0	=	normal sector write
1	=	write to directory sector
2	=	write to the first sector of a new data block

Condition 0 occurs whenever the next write operation is into a previously written area, such as a random mode record update, when the write is to other than the first sector of an unallocated block, or when the write is not into the directory area. Condition 1 occurs when a write into the directory area is performed. Condition 2 occurs when the first record (only) of a newly allocated data block is written. In most cases, application programs read or write multiple 128-byte sectors in sequence; thus, there is little overhead involved in either operation when blocking and deblocking records, since preread operations can be avoided when writing records.

Appendix G lists the blocking and deblocking algorithms in skeletal form (this file is included on your CP/M disk). enerally, the algorithms map all CP/M sector read operations onto the host disk through an intermediate buffer that is the size of the host disk sector. Throughout the program, values and variables that relate to the CP/M sector involved in a seek operation are prefixed by sek, while those related to the host disk system are prefixed by hst. The equate statements beginning on line 29 of Appendix G define the mapping between CP/M and the host system, and must be changed if other than the sample host system is involved.

The entry points BOOT and WBOOT must contain the initialization code starting on line 57, while the SELDSK entry point must be augmented by the code starting on line 65. The user should note that although the SELDSK entry point computes and returns the Disk Parameter Header address, it does not physically select the host disk at this point (it is selected later at READHST or WRITEHST). Further, SETTRK, SETTRK, and SETDMA simply store the values, but do not take any other action at this point. SECTRAN performs a trivial function of returning the physical sector number.

The principal entry points are READ and WRITE, starting on lines 110 and 125, respectively. These subroutines take the place of your previous READ and WRITE operations.

The actual physical read or write takes place at either WRITEHST or READHST, where all values have been prepared: hstdsk is the host disk number, hsttrk is the host track number, and hstsec is the host sector number (which may require translation to a physical sector number). The user must insert code at this point that performs the full host sector read or write into or out of the buffer at hstbuf of length hstsiz. All other mapping functions are performed by the algorithms.

This particular algorithm was tested using an 80-megabyte hard disk unit that was originally configured for 128-byte sectors, producing approximately 35 megabytes of formatted storage. When configured for 512-byte host sectors, usable storage increased to 57 megabytes, with a corresponding 400% improvement in overall response. In this situation, there is no apparent overhead involved in deblocking sectors, with the advantage that user programs still maintain 128-byte sectors. This is primarily because of the information provided by the BDOS, which eliminates the necessity for preread operations.

CP/M®
OPERATING SYSTEM
MANUAL

Appendices A-G

DIGITAL RESEARCH™
P.O. Box 579
Pacific Grove, California 93950

595-2843

First Printing: July 1982

Appendix A: The MDS Basic I/O System (BIOS)

```
                ;       mds-800 i/o drivers for cp/m 2.2
                ;       (four drive single density version)
                ;
                ;       version 2.2 february, 1980
                ;
0016 =          vers    equ     22              ;version 2.2
                ;
                ;       copyright (c) 1980
                ;       digital research
                ;       box 579, pacific grove
                ;       california, 93950
                ;
                ;
ffff =          true    equ     offffh          ;value of "true"
0000 =          false   equ     not true        ;"false"
0000 =          test    equ     false           ; true if test bios
                ;
                        if      test
                bias    equ     03400h          ;base of ccp in test system
                        endif
                        if      not test
0000 =          bias    equ     0000h           ;generate relocatable cp/m system
                        endif
                ;
1600 =          patch   equ     1600h
                ;
1600                    org     patch
0000 =          cpmb    equ     $-patch         ;base of cpm console processor
0806 =          bdos    equ     806h+cpmb       ;basic dos (resident portion)
```

```
1600 =          cpml    equ     $-cpmb      ;length (in bytes) of cpm system
002c =          nsects  equ     cpml/128    ;number of sectors to load
0002 =          offset  equ     2           ;number of disk tracks used by cp/m
0004 =          cdisk   equ     0004h       ;address of last logged disk on warm start
0080 =          buff    equ     0080h       ;default buffer address
000a =          retry   equ     10          ;max retries on disk i/o before error
                ;
                ;       perform following functions
                ;       boot    cold start
                ;       wboot   warm start (save i/o byte)
                ;       (boot and wboot are the same for mds)
                ;       const   console status
                ;               reg-a = 00 if no character ready
                ;               reg-a = ff if character ready
                ;       conin   console character in (result in reg-a)
                ;       conout  console character out (char in reg-c)
                ;       list    list out (char in reg-c)
                ;       punch   punch out (char in reg-c)
                ;       reader  paper tape reader in (result to reg-a)
                ;       home    move to track 00
                ;
                ;       (the following calls set-up the io parameter block for the
                ;       mds, which is used to perform subsequent reads and writes)
                ;       seldsk  select disk given by reg-c (0, 1, 2...)
                ;       settrk  set track address (0, . . . 76) for subsequent read/write
                ;       setsec  set sector address (1, . . . , 26) for subsequent read/write
                ;       setdma  set subsequent dma address (initially 80h)
                ;
                ;       (read and write assume previous calls to set up the io parameters)
                ;       read    read track/sector to preset dma address
                ;       write   track/sector from preset dma address
                ;
                ;       jump vector for individual routines
1600 c3b316             jmp     boot
```

```
1603 c3c316     wboote: jmp     wboot
1606 c36117             jmp     const
1609 c36417             jmp     conin
160c c36a17             jmp     conout
160f c36d17             jmp     list
1612 c37217             jmp     punch
1615 c37517             jmp     reader
1618 c37817             jmp     home
161b c37d17             jmp     seldsk
161e c3a717             jmp     settrk
1621 c3ac17             jmp     setsec
1624 c3bb17             jmp     setdma
1627 c3c117             jmp     read
162a c3ca17             jmp     write
162d c37017             jmp     listst          ;list status
1630 c3b117             jmp     sectran
                ;
                        maclib  diskdef         ;load the disk definition library
                        disks   4               ;four disks
1633+=          dpbase  equ     $               ;base of disk parameter blocks
1633+82160000   dpe0:   dw      xlt0, 0000h          ;translate table
1637+00000000           dw      0000h, 0000h         ;scratch area
163b+6e187316           dw      dirbuf, dpb0         ;dir buff, parm block
163f+0d19ee18           dw      csv0, alv0           ;check, alloc vectors
1643+82160000   dpe1:   dw      xlt1, 0000h          ;translate table
1647+00000000           dw      0000h, 0000h         ;scratch area
164b+6e187316           dw      dirbuf, dpb1         ;dir buff, parm block
164f+3c191d19           dw      csv1, alv1           ;check, alloc vectors
1653+82160000   dpe2:   dw      xlt2, 0000h          ;translate table
1657+00000000           dw      0000h, 0000h         ;scratch area
165b+6e187316           dw      dirbuf, dpb2         ;dir buff, parm block
165f+6b194c19           dw      csv2, alv2           ;check, alloc vectors
1663+82160000   dpe3:   dw      xlt3, 0000h          ;translate table
1667+00000000           dw      0000h, 0000h         ;scratch area
```

```
166b+6e187316          dw       dirbuf, dpb3          ;check, alloc block
166f+9a197b19          dw       csv3, alv3            ;dir buff, parm vectors
                       diskdef  0, 1, 26, 6, 1024, 243, 64, 64, offset
1673+=         dpb0    equ      $                     ;disk parm block
1673+1a00              dw       26                    ;sec per track
1675+03                db       3                     ;block shift
1676+07                db       7                     ;block mask
1677+00                db       0                     ;extnt mask
1678+f200              dw       242                   ;disk size-1
167a+3f00              dw       63                    ;directory max
167c+c0                db       192                   ;alloc0
167d+00                db       0                     ;alloc1
167e+1000              dw       16                    ;check size
1680+0200              dw       2                     ;offset
1682+=         xlt0    equ      $                     ;translate table
1682+01                db       1
1683+07                db       7
1684+0d                db       13
1685+13                db       19
1686+19                db       25
1687+05                db       5
1688+0b                db       11
1689+11                db       17
168a+17                db       23
168b+03                db       3
168c+09                db       9
168d+0f                db       15
168e+15                db       21
168f+02                db       2
1690+08                db       8
1691+0e                db       14
1692+14                db       20
1693+1a                db       26
1694+06                db       6
```

```
1695+0c                  db       12
1696+12                  db       18
1697+18                  db       24
1698+04                  db       4
1699+0a                  db       10
169a+10                  db       16
169b+16                  db       22
                         diskdef  1, 0
1673+=          dpb1     equ      dpb0     ;equivalent parameters
001f+=          als1     equ      als0     ;same allocation vector size
0010+=          css1     equ      css0     ;same checksum vector size
1682+=          xlt1     equ      xlt0     ;same translate table
                         diskdef  2, 0
1673+=          dpb2     equ      dpb0     ;equivalent parameters
001f+=          als2     equ      als0     ;same allocation vector size
0010+=          css2     equ      css0     ;same checksum vector size
1682+=          xlt2     equ      xlt0     ;same translate table
                         diskdef  3, 0
1673+=          dpb3     equ      dpb0     ;equivalent parameters
001f+=          als3     equ      als0     ;same allocation vector size
0010+=          css3     equ      css0     ;same checksum vector size
1682+=          xlt3     equ      xlt0     ;same translate table
                ;        endef occurs at end of assembly
                ;
                ;        end of controller—independent code, the remaining subroutines
                ;        are tailored to the particular operating environment, and must
                ;        be altered for any system which differs from the intel mds.
                ;
                ;        the following code assumes the mds monitor exists at 0f800h
                ;        and uses the i/o subroutines within the monitor
                ;
                ;        we also assume the mds system has four disk drives
00fd =          revrt    equ      0fdh     ;interrupt revert port
00fc =          intc     equ      0fch     ;interrupt mask port
```

```
00f3 =          icon    equ     0f3h            ;interrupt control port
007E =          inte    equ     0111$1110b              ;enable rst 0 (warm boot), rst 7 (monitor)
                ;
                ;       mds monitor equates
f800 =          mon80   equ     0f800h          ;mds monitor
ff0f =          rmon80  equ     0ff0fh          ;restart mon80 (boot error)
f803 =          ci      equ     0f803h          ;console character to reg-a
f806 =          ri      equ     0f806h          ;reader in to reg-a
f809 =          co      equ     0f809h          ;console char from c to console out
f80c =          po      equ     0f80ch          ;punch char from c to punch device
f80f =          lo      equ     0f80fh          ;list from c to list device
f812 =          csts    equ     0f812h          ;console status 00/ff to register a
                ;
                ;       disk ports and commands
0078 =          base    equ     78h             ;base of disk command io ports
0078 =          dstat   equ     base            ;disk status (input)
0079 =          rtype   equ     base+1          ;result type (input)
007b =          rbyte   equ     base+3          ;result byte (input)
                ;
0079 =          ilow    equ     base+1          ;iopb low address (output)
007a =          ihigh   equ     base+2          ;iopb high address (output)
                ;
0004 =          readf   equ     4h              ;read function
0006 =          writf   equ     6h              ;write function
0003 =          recal   equ     3h              ;recalibrate drive
0004 =          iordy   equ     4h              ;i/o finished mask
000d =          cr      equ     0dh             ;carriage return
000a =          lf      equ     0ah             ;line feed
                ;
                signon: ;signon message: xxk cp/m vers y.y
169c 0d0a0a             db      cr, lf, lf
                        if      test
                        db      '32'            ;32k example bios
                        endif
```

```
                          if      not test
169f 3030                 db      '00'          ;memory size filled by relocator
                          endif
16a1 6b2043502f           db      'k cp/m vers  '
16ad 322e32               db      vers/10+'0',  ',' vers mod 10+'0'
16b0 0d0a00               db      cr, lf, 0
                 ;
                 boot:    ;print signon message and go to ccp
                 ;        (note: mds boot initialized iobyte at 0003h)
16b3 310001               lxi     sp, buff+80h
16b6 219c16               lxi     h, signon
16b9 cdd317               call    prmsg         ;print message
16bc af                   xra     a             ;clear accumulator
16bd 320400               sta     cdisk         ;set initially to disk a
16c0 c30f17               jmp     gocpm         ;go to cp/m
                 ;
                 ;
                 wboot:;  loader on track 0, sector 1, which will be skipped for warm
                 ;        read cp/m from disk—assuming there is a 128 byte cold start
                 ;        start
                 ;
16c3 318000               lxi     sp, buff      ;using dma—thus 80 thru ff available for stack
                 ;
16c6 0e0a                 mvi     c, retry      ;max retries
16c8 c5                   push    b
                 wboot0:  ;enter here on error retries
16c9 010000               lxi     b, cpmb       ;set dma address to start of disk system
16cc cdbb17               call    setdma
16cf 0e00                 mvi     c, 0          ;boot from drive 0
16d1 cd7d17               call    seldsk
16d4 0e00                 mvi     c, 0
16d6 cda717               call    settrk        ;start with track 0
16d9 0e02                 mvi     c, 2          ;start reading sector 2
16db cdac17               call    setsec
```

```
                         ;
                         ;       read sectors, count nsects to zero
16de c1                          pop     b               ;10-error count
16df 062c                        mvi     b, nsects
                         rdsec:  ;read next sector
16e1 c5                          push    b               ;save sector count
16e2 cdc117                      call    read
16e5 c24917                      jnz     booterr         ;retry if errors occur
16e8 2a6c18                      lhld    iod             ;increment dma address
16eb 118000                      lxi     d, 128          ;sector size
16ee 19                          dad     d               ;incremented dma address in hl
16ef 44                          mov     b, h
16f0 4d                          mov     c, l            ;ready for call to set dma
16f1 cdbb17                      call    setdma
16f4 3a6b18                      lda     ios             ;sector number just read
16f7 fe1a                        cpi     26              ;read last sector?
16f9 da0517                      jc      rd1
                         ;       must be sector 26, zero and go to next track
16fc 3a6a18                      lda     iot             ;get track to register a
16ff 3c                          inr     a
1700 4f                          mov     c, a            ;ready for call
1701 cda717                      call    settrk
1704 af                          xra     a               ;clear sector number
1705 3c                  rd1:    inr     a               ;to next sector
1706 4f                          mov     c, a            ;ready for call
1707 cdac17                      call    setsec
170a c1                          pop     b               ;recall sector count
170b 05                          dcr     b               ;done?
170c c2e116                      jnz     rdsec
                         ;
                         ;       done with the load, reset default buffer address
                         gocpm:  ;(enter here from cold start boot)
                         ;       enable rst0 and rst7
```

```
170f f3                         di
1710 3e12                       mvi     a, 12h          ;initialize command
1712 d3fd                       out     revrt
1714 af                         xra     a
1715 d3fc                       out     intc            ;cleared
1717 3e7e                       mvi     a, inte         ;rst0 and rst7 bits on
1719 d3fc                       out     intc
171b af                         xra     a
171c d3f3                       out     icon            ;interrupt control
                        ;
                        ;       set default buffer address to 80h
171e 018000                     lxi     b, buff
1721 cdbb17                     call    setdma
                        ;
                        ;       reset monitor entry points
1724 3ec3                       mvi     a, jmp
1726 320000                     sta     0
1729 210316                     lxi     h, wboote
172c 220100                     shld    1               ;jump wboot at location 00
172f 320500                     sta     5
1732 210608                     lxi     h, bdos
1735 220600                     shld    6               ;jmp bdos at location 5
                                if      not test
1738 323800                     sta     7*8             ;jmp to mon80 (may have changed by ddt)
173b 2100f8                     lxi     h, mon80
173e 223900                     shld    7*8+1
                                endif
                        ;       leave iobyte set
                        ;       previously selected disk was b, send parameter to cpm
1741 3a0400                     lda     cdisk           ;last logged disk number
1744 4f                         mov     c, a            ;send to ccp to log it in
1745 fb                         ei
1746 c30000                     jmp     cpmb
                        ;
```

```
                  ;        error condition occurred, print message and retry
                  booterr:
1749 c1                    pop     b           ;recall counts
174a 0d                    dcr     c
174b ca5217                jz      booter0
                  ;        try again
174e c5                    push    b
174f c3c916                jmp     wboot0
                  ;
                  booter0:
                  ;        otherwise too many retries
1752 215b17                lxi     h, bootmsg
1755 cdd317                call    prmsg
1758 c30fff                jmp     rmon80      ;mds hardware monitor
                  ;
                  bootmsg:
175b 3f626f6f74            db      '?boot', 0
                  ;
                  ;
                  const:   console status to reg-a
                  ;        (exactly the same as mds call)
1761 c312f8                jmp     csts
                  ;
                  conin:   ;console character to reg-a
1764 cd03f8                call    ci
1767 e67f                  ani     7fh         ;remove parity bit
1769 c9                    ret
                  ;
                  conout:  ;console character from c to console out
176a c309f8                jmp     co
                  ;
                  list:    ;list device out
                  ;        (exactly the same as mds call)
176d c30ff8                jmp     lo
```

```
                        ;
                        listst:
                                ;return list status
1770 af                         xra     a
1771 c9                         ret                     ;always not ready
                        ;
                        punch:  ;punch device out
                        ;       (exactly the same as mds call)
1772 c30cf8                     jmp     po
                        ;
                        reader: ;reader character in to reg-a
                        ;       (exactly the same as mds call)
1775 c306f8                     jmp     ri
                        ;
                        home:   ;move to home position
                        ;       treat as track 00 seek
1778 0e00                       mvi     c, 0
177a c3a717                     jmp     settrk
                        ;
                        seldsk: ;select disk given by register c
177d 210000                     lxi     h, 0000h        ;return 0000 if error
1780 79                         mov     a, c
1781 fe04                       cpi     ndisks          ;too large?
1783 d0                         rnc                     ;leave hl = 0000
                        ;
1784 e602                       ani     10b             ;00 00 for drive 0, 1 and 10 10 for drive 2, 3
1786 326618                     sta     dbank           ;to select drive bank
1789 79                         mov     a, c            ;00, 01, 10, 11
178a e601                       ani     1b              ;mds has 0, 1 at 78, 2, 3 at 88
178c b7                         ora     a               ;result 00?
178d ca9217                     jz      setdrive
1790 3e30                       mvi     a, 00110000b        ;selects drive 1 in bank
                        setdrive:
1792 47                         mov     b, a            ;save the function
```

```
1793 216818            lxi     h, iof      ;io function
1796 7e                mov     a, m
1797 e6cf              ani     11001111b   ;mask out disk number
1799 b0                ora     b           ;mask in new disk number
179a 77                mov     m, a        ;save it in iopb
179b 69                mov     l, c
179c 2600              mvi     h, 0        ;hl=disk number
179e 29                dad     h           ;*2
179f 29                dad     h           ;*4
17a0 29                dad     h           ;*8
17a1 29                dad     h           ;*16
17a2 113316            lxi     d, dpbase
17a5 19                dad     d           ;hl=disk header table address
17a6 c9                ret
               ;
               ;
               settrk: ;set track address given by c
17a7 216a18            lxi     h, iot
17aa 71                mov     m, c
17ab c9                ret
               ;
               setsec: ;set sector number given by c
17ac 216b18            lxi     h, ios
17af 71                mov     m, c
17b0 c9                ret
               sectran:
                               ;translate sector bc using table at de
17b1 0600              mvi     b, 0        ;double precision sector number in bc
17b3 eb                xchg                ;translate table address to hl
17b4 09                dad     b           ;translate (sector) address
17b5 7e                mov     a, m        ;translated sector number to a
17b6 326b18            sta     ios
17b9 6f                mov     l, a        ;return sector number in l
17ba c9                ret
```

```
                         ;
                         setdma: ;set dma address given by regs b, c
        17bb 69                  mov     l, c
        17bc 60                  mov     h, b
        17bd 226c18              shld    iod
        17c0 c9                  ret
                         ;
                         read:   ;read next disk record (assuming disk/trk/ sec/dma set)
        17c1 0e04                mvi     c, readf        ;set to read function
        17c3 cde017              call    setfunc
        17c6 cdf017              call    waitio          ;perform read function
        17c9 c9                  ret                     ;may have error set in reg-a
                         ;
                         ;
                         write:  ;disk write function
        17ca 0e06                mvi     c, writf
        17cc cde017              call    setfunc         ;set to write function
        17cf cdf017              call    waitio
        17d2 c9                  ret                     ;may have error set
                         ;
                         ;
                         ;       utility subroutines
                         prmsg:  ;print message at h, l to 0
        17d3 7e                  mov     a, m
        17d4 b7                  ora     a               zero?
        17d5 c8                  rz
                         ;       more to print
        17d6 e5                  push    h
        17d7 4f                  mov     c, a
        17d8 cd6a17              call    conout
        17db e1                  pop     h
        17dc 23                  inx     h
        17dd c3d317              jmp     prmsg
                         ;
```

```
                        setfunc:
                        ;       set function for next i/o (command in reg-c)
17e0 216818                     lxi     h, iof          ;io function address
17e3 7e                         mov     a, m            ;get it to accumulator for masking
17e4 e6f8                       ani     11111000b       ;remove previous command
17e6 b1                         ora     c               ;set to new command
17e7 77                         mov     m, a            ;replaced in iopb
                        ;       the mds-800 controller requires disk bank bit in sector byte
                        ;       mask the bit from the current i/o function
17e8 e620                       ani     00100000b               ;mask the disk select bit
17ea 216b18                     lxi     h, ios                  ;address the sector select byte
17ed b6                         ora     m                       ;select proper disk bank
17ee 77                         mov     m, a                    ;set disk select bit on/off
17ef c9                         ret
                        ;
                        waitio:
17f0 0e0a                       mvi     c, retry        ;max retries before perm error
                        rewait:
                        ;       start the i/o function and wait for completion
17f2 cd3f18                     call    intype          ;in rtype
17f5 cd4c18                     call    inbyte          ;clears the controller
                        ;
17f8 3a6618                     lda     dbank                   ;set bank flags
17fb b7                         ora     a                       ;zero if drive 0, 1 and nz if 2, 3
17fc 3e67                       mvi     a, iopb and offh        ;low address for iopb
17fe 0618                       mvi     b, iopb shr 8           ;high address for iopb
1800 c20b18                     jnz     iodr1           ;drive bank 1?
1803 d379                       out     ilow                    ;low address to controller
1805 78                         mov     a, b
1806 d37a                       out     ihigh           ;high address
1808 c31018                     jmp     waito                   ;to wait for complete
                        ;
                        iodr1:  ;drive bank 1
180b d389                       out     ilow+10h                ;88 for drive bank 10
```

```
180d 78                    mov    a, b
180e d38a                  out    ihigh+10h
                  ;
1810 cd5918       waito:   call   instat          ;wait for completion
1813 e604                  ani    iordy           ;ready?
1815 ca1018                jz     waito
                  ;
                  ;        check io completion ok
1818 cd3f18                call   intype          ;must be io complete (00) unlinked
                  ;        00 unlinked i/o complete, 01 linked i/o complete (not used)
                  ;        io disk status changed   11 (not used)
181b fe02                  cpi    10b             ;ready status change?
181d ca3218                jz     wready
                  ;
                  ;        must be 00 in the accumulator
1820 b7                    ora    a
1821 c23818                jnz    werror          ;some other condition, retry
                  ;
                  ;        check i/o error bits
1824 cd4c18                call   inbyte
1827 17                    ral
1828 da3218                jc     wready          ;unit not ready
182b 1f                    rar
182c e6fe                  ani    11111110b       ;any other errors? (deleted data ok)
182e c23818                jnz    werror
                  ;
                  ;        read or write is ok, accumulator contains zero
1831 c9                    ret
                  ;
                  wready:  ;not ready, treat as error for now
1832 cd4c18                call   inbyte          ;clear result byte
1835 c33818                jmp    trycount
                  ;
                  werror:  ;return hardware malfunction (crc, track, seek, etc.)
```

```
                        ;       the mds controller has returned a bit in each position
                        ;       of the accumulator, corresponding to the conditions:
                        ;       0       —deleted data (accepted as ok above)
                        ;       1       —crc error
                        ;       2       —seek error
                        ;       3       —address error (hardware malfunction)
                        ;       4       —data over/under flow (hardware malfunction)
                        ;       5       —write protect (treated as not ready)
                        ;       6       —write error (hardware malfunction)
                        ;       j       —not ready
                        ;       (accumulator bits are numbered 7 6 5 4 3 2 1 0)
                        ;
                        ;       it may be useful to filter out the various conditions,
                        ;       but we will get a permanent error message if it is not
                        ;       recoverable. in any case, the not ready condition is
                        ;       treated as a separated condition for later improvement
                        trycount:
                        ;       register c contains retry count, decrement 'til zero
1838 0d                         dcr     c
1839 c2f217                     jnz     rewait          ;for another try
                        ;
                        ;       cannot recover from error
183c 3e01                       mvi     a, 1            ;error code
183e c9                         ret
                        ;
                        ;       intype, inbyte, instat read drive bank 00 or 10
183f 3a6618             intype: lda     dbank
1842 b7                         ora     a
1843 c24918                     jnz     intyp1          ;skip to bank 10
1846 db79                       in      rtype
1848 c9                         ret
1849 db89               intyp1: in      rtype+10h               ;78 for 0, 1  88 for 2, 3
184b c9                         ret
```

```
                    ;
184c 3a6618         inbyte:   lda     dbank
184f b7                       ora     a
1850 c25618                   jnz     inbyt1
1853 db7b                     in      rbyte
1855 c9                       ret
1856 db8b           inbyt1:   in      rbyte+10h
1858 c9                       ret
                    ;
1859 3a6618         instat:   lda     dbank
185c b7                       ora     a
185d c26318                   jnz     insta1
1860 db78                     in      dstat
1862 c9                       ret
1863 db88           insta1:   in      dstat+10h
1865 c9                       ret
                    ;
                    ;
                    ;
                    ;         data areas (must be in ram)
1866 00             dbank:    db      0           ;disk bank 00 if drive 0, 1
                                                  ;           10 if drive 2, 3
                    iopb:     ;io parameter block
1867 80                       db      80h         ;normal i/o operation
1868 04             iof:      db      readf       ;io function, initial read
1869 01             ion:      db      1           ;number of sectors to read
186a 02             iot:      db      offset      ;track number
186b 01             ios:      db      1           ;sector number
186c 8000           iod:      dw      buff        ;io address
                    ;
                    ;
                    ;         define ram areas for bdos operation
                              endef
```

```
571     186e+=          begdat  equ     $
572     186e+           dirbuf: ds      128             ;directory access buffer
573     18ee+           alv0:   ds      31
574     190d+           csv0:   ds      16
575     191d+           alv1:   ds      31
576     193c+           csv1:   ds      16
577     194c+           alv2:   ds      31
578     196b+           csv2:   ds      16
579     197b+           alv3:   ds      31
580     199a+           csv3:   ds      16
581     19aa+=          enddat  equ     $
582     013c+=          datsiz  equ     $-begdat
583     19aa                    end

als1            001f    141#
als2            001f    146#
als3            001f    151#
alv0            18ee     87     573#
alv1            191d     91     575#
alv2            194c     95     577#
alv3            197b     99     579#
base            0078    180#    181     182     183     185     186
bdos            0806     29#    287
begdat          186e    571#    582
bias            0000     19#     22#
boot            16b3     63     207#
booter0         1752    305     310#
booterr         1749    241     302#
bootmsg         175b    312     316#
buff            0080     34#    209     221     278     566
cdisk           0004     33#    213     296
ci              f803    172#    325
co              f809    174#    330
```

conin	1764	66	324#				
conout	176a	67	329#	432			
const	1761	65	320#				
cpmb	0000	28#	29	30	226	299	
cpml	1600	30#	31				
cr	000d	192#	196	205			
css1	0010	142#					
css2	0010	147#					
css3	0010	152#					
csts	f812	177#	322				
csv0	190d	87	574#				
csv1	193c	91	576#				
csv2	196b	95	578#				
csv3	199a	99	580#				
datsiz	013c	582#					
dbank	1866	361	459	531	539	547	558#
dirbuf	186e	86	90	94	98	572#	
dpb0	1673	86	101#	140	145	150	
dpb1	1673	90	140#				
dpb2	1673	94	145#				
dpb3	1673	98	150#				
dpbase	1633	83#	380				
dpe0	1633	84#					
dpe1	1643	88#					
dpe2	1653	92#					
dpe3	1663	96#					
dstat	0078	181#	550	552			
enddat	19aa	581#					
false	0000	15#	16				
gocpm	170f	214	265#				
home	1778	71	349#				
icon	00f3	166#	275				
ihigh	007a	186#	466	472			

ilow	0079	185#	464	470		
inbyt1	1856	541	544#			
inbyte	184c	457	490	501	539#	
insta1	1863	549	552#			
instat	1859	474	547#			
intc	00fc	165#	271	273		
inte	007e	167#	272			
intyp1	1849	533	536#			
intype	183f	456	479	531#		
iod	186c	242	407	566#		
iodr1	180b	463	469#			
iof	1868	369	439	562#		
ion	1869	563#				
iopb	1867	461	462	560#		
iordy	0004	191#	475			
ios	186b	248	391	400	447	565#
iot	186a	252	386	564#		
lf	000a	193#	196	196	205	
list	176d	68	332#			
listst	1770	78	336#			
lo	f80f	176#	334			
mon80	f800	170#	291			
nsects	002c	31#	237			
offset	0002	32#	100	564		
patch	1600	25#	27	28		
po	f80c	175#	343			
prmsg	17d3	211	313	425#	435	
punch	1772	69	341#			
rbyte	007b	183#	542	544		
rd1	1705	250	257#			
rdsec	16e1	238#	262			
read	17c1	76	240	410#		
reader	1775	70	345#			
readf	0004	188#	411	562		

recal	0003	190#					
retry	000a	35#	223	453			
revrt	00fd	164#	269				
rewait	17f2	454#	524				
ri	f806	173#	347				
rmon80	ff0f	171#	314				
rtype	0079	182#	534	536			
sectran	17b1	79	394#				
seldsk	177d	72	229	354#			
setdma	17bb	75	227	247	279	404#	
setdrive	1792	365	367#				
setfunc	17e0	412	419	437#			
setsec	17ac	74	233	259	390#		
settrk	17a7	73	231	255	352	385#	
signon	169c	195#	210				
test	0000	16#	18	21	197	200	289
true	ffff	14#	15				
trycount	1838	502	521#				
vers	0016	6#	204	204			
waito	1810	467	474#	476			
waitio	17f0	413	420	452#			
wboot	16c3	64	217#				
wboot0	16c9	225#	308				
wboote	1603	64#	284				
werror	1838	487	495	504#			
wready	1832	483	492	500#			
write	17ca	77	417#				
writf	0006	189#	418				
xlt0	1682	84	112#	143	148	153	
xlt1	1682	88	143#				
xlt2	1682	92	148#				
xlt3	1682	96	153#				

Appendix B: A Skeletal CBIOS

```
                      ;          skeletal cbios for first level of cp/m 2.0 alteration
                      ;
0014 =                msize      equu  20             ;cp/m version memory size in kilobytes
                      ;
                      ;          "bias" is address offset from 3400h for memory systems
                      ;          than 16k (referred to as "b" throughout the text)
                      ;
0000 =                bias       equ   (msize-20)*1024
3400 =                ccp        equ   3400h+bias     ;base of ccp
3c06 =                bdos       equ   ccp+806h       ;base of bdos
4a00 =                bios       equ   ccp+1600h      ;base of bios
0004 =                cdisk      equ   0004h          ;current disk number 0=a, . . . , 15=p
0003 =                iobyte     equ   0003h          ;intel i/o byte
                      ;
4a00                             org   bios           ;origin of this program
002c =                nsects     equ   ($-ccp)/128    ;warm start sector count
                      ;
                      ;          jump vector for individual subroutines
4a00 c39c4a                      jmp   boot           ;cold start
4a03 c3a64a           wboote:    jmp   wboot          ;warm start
4a06 c3114b                      jmp   const          ;console status
4a09 c3244b                      jmp   conin          ;console character in
4a0c c3374b                      jmp   conout         ;console character out
4a0f c3494b                      jmp   list           ;list character out
4a12 c34d4b                      jmp   punch          ;punch character out
4a15 c34f4b                      jmp   reader         ;reader character out
4a18 c3544b                      jmp   home           ;move head to home position
```

```
4a1b c35a4b                 jmp     seldsk          ;select disk
4a1e c37d4b                 jmp     settrk          ;set track number
4a21 c3924b                 jmp     setsec          ;set sector number
4a24 c3ad4b                 jmp     setdma          ;set dma address
4a27 c3c34b                 jmp     read            ;read disk
4a2a c3d64b                 jmp     write           ;write disk
4a2d c34b4b                 jmp     listst          ;return list status
4a30 c3a74b                 jmp     sectran         ;sector translate
                    ;
                    ;       fixed data tables for four-drive standard
                    ;       ibm-compatible 8" disks
                    ;       disk parameter header for disk 00
4a33 734a0000       dpbase: dw      trans, 0000h
4a37 00000000               dw      0000h, 0000h
4a3b f04c8d4a               dw      dirbf, dpblk
4a3f ec4d704d               dw      chk00, all00
                    ;       disk parameter header for disk 01
4a43 734a0000               dw      trans, 0000h
4a47 00000000               dw      0000h, 0000h
4a4b f04c8d4a               dw      dirbf, dpblk
4a4f fc4d8f4d               dw      chk01, all01
                    ;       disk parameter header for disk 02
4a53 734a0000               dw      trans, 0000h
4a57 00000000               dw      0000h, 0000h
4a5b f04c8d4a               dw      dirbf, dpblk
4a5f 0c4eae4d               dw      chk02, all02
                    ;       disk parameter header for disk 03
4a63 734a0000               dw      trans, 0000h
4a67 00000000               dw      0000h, 0000h
4a6b f04c8d4a               dw      dirbf, dpblk
4a6f 1c4ecd4d               dw      chk03, all03
                    ;
                    ;       sector translate vector
```

```
4a73 01070d13   trans:  db      1, 7, 13, 19    ;sectors 1, 2, 3, 4
4a77 19050b11           db      25, 5, 11, 17   ;sectors 5, 6, 7, 8
4a7b 1703090f           db      23, 3, 9, 15    ;sectors 9, 10, 11, 12
4a7f 1502080e           db      21, 2, 8, 14    ;sectors 13, 14, 15, 16
4a83 141a060c           db      20, 26, 6, 12   ;sectors 17, 18, 19, 20
4a87 1218040a           db      18, 24, 4, 10   ;sectors 21, 22, 23, 24
4a8b 1016               db      16, 22          ;sectors 25, 26
                ;
                dpblk:  ;disk parameter block, common to all disks
4a8d 1a00               dw      26              ;sectors per track
4a8f 03                 db      3               ;block shift factor
4a90 07                 db      7               ;block mask
4a91 00                 db      0               ;null mask
4a92 f200               dw      242             ;disk size-1
4a94 3f00               dw      63              ;directory max
4a96 c0                 db      192             ;alloc 0
4a97 00                 db      0               ;alloc 1
4a98 1000               dw      16              ;check size
4a9a 0200               dw      2               ;track offset
                ;
                ;       end of fixed tables
                ;
                ;       individual subroutines to perform each function
                boot:   ;simplest case is to just perform parameter initialization
4a9c af                 xra     a               ;zero in the accum
4a9d 320300             sta     iobyte          ;clear the iobyte
4aa0 320400             sta     cdisk           ;select disk zero
4aa3 c3ef4a             jmp     gocpm           ;initialize and go to cp/m
                ;
                wboot:  ;simplest case is to read the disk until all sectors loaded
4aa6 318000             lxi     sp, 80h         ;use space below buffer for stack
4aa9 0e00               mvi     c, 0            ;select disk 0
4aab cd5a4b             call    seldsk
4aae cd544b             call    home            ;go to track 00
```

```
                        ;
4ab1 062c                       mvi     b, nsects       ;b counts # of sectors to load
4ab3 0e00                       mvi     c, 0            ;c has the current track number
4ab5 1602                       mvi     d, 2            ;d has the next sector to read
                        ;       note that we begin by reading track 0, sector 2 since sector 1
                        ;       contains the cold start loader, which is skipped in a warm start
4ab7 210034                     lxi     h, ccp          ;base of cp/m (initial load point)
                        load1:  ;load one more sector
4aba c5                         push    b               ;save sector count, current track
4abb d5                         push    d               ;save next sector to read
4abc e5                         push    h               ;save dma address
4abd 4a                         mov     c, d            ;get sector address to register c
4abe cd924b                     call    setsec          ;set sector address from register c
4ac1 c1                         pop     b               ;recall dma address to b, c
4ac2 c5                         push    b               ;replace on stack for later recall
4ac3 cdad4b                     call    setdma          ;set dma address from b, c
                        ;
                        ;       drive set to 0, track set, sector set, dma address set
4ac6 cdc34b                     call    read
4ac9 fe00                       cpi     00h             ;any errors?
4acb c2a64a                     jnz     wboot           ;retry the entire boot if an error occurs
                        ;
                        ;       no error, move to next sector
4ace e1                         pop     h               ;recall dma address
4acf 118000                     lxi     d, 128          ;dma=dma+128
4ad2 19                         dad     d               ;new dma address is in h, l
4ad3 d1                         pop     d               ;recall sector address
4ad4 c1                         pop     b               ;recall number of sectors remaining, and current trk
4ad5 05                         dcr     b               ;sectors=sectors-1
4ad6 caef4a                     jz      gocpm           ;transfer to cp/m if all have been loaded
                        ;
                        ;       more sectors remain to load, check for track change
4ad9 14                         inr     d
```

```
4ada 7a                  mov   a, d       ;sector=27?, if so, change tracks
4adb fe1b                cpi   27
4add daba4a              jc    load1      ;carry generated if sector<27
                   ;
                   ;     end of current track, go to next track
4ae0 1601                mvi   d, 1       ;begin with first sector of next track
4ae2 0c                  inr   c          ;track=track+1
                   ;
                   ;     save register state, and change tracks
4ae3 c5                  push  b
4ae4 d5                  push  d
4ae5 e5                  push  h
4ae6 cd7d4b              call  settrk     ;track address set from register c
4ae9 e1                  pop   h
4aea d1                  pop   d
4aeb c1                  pop   b
4aec c3ba4a              jmp   load1      ;for another sector
                   ;
                   ;     end of load operation, set parameters and go to cp/m
                   gocpm:
4aef 3ec3                mvi   a, 0c3h    ;c3 is a jmp instruction
4af1 320000              sta   0          ;for jmp to wboot
4af4 21034a              lxi   h, wboote  ;wboot entry point
4af7 220100              shld  1          ;set address field for jmp at 0
                   ;
4afa 320500              sta   5          ;for jmp to bdos
4afd 21063c              lxi   h, bdos    ;bdos entry point
4b00 220600              shld  6          ;address field of jump at 5 to bdos
                   ;
4b03 018000              lxi   b, 80h     ;default dma address is 80h
4b06 cdad4b              call  setdma
                   ;
4b09 fb                  ei               ;enable the interrupt system
```

```
4b0a 3a0400             lda     cdisk       ;get current disk number
4b0d 4f                 mov     c, a        ;send to the ccp
4b0e c30034             jmp     ccp         ;go to cp/m for further processing
                ;
                ;
                ;       simple i/o handlers (must be filled in by user)
                ;       in each case, the entry point is provided, with space reserved
                ;       to insert your own code
                ;
                const:  ;console status, return 0ffh if character ready, 00h if not
4b11                    ds      10h         ;space for status subroutine
4b21 3e00               mvi     a, 00h
4b23 c9                 ret
                ;
                conin:  ;console character into register a
4b24                    ds      10h         ;space for input routine
4b34 e67f               ani     7fh         ;strip parity bit
4b36 c9                 ret
                ;
                conout: ;console character output from register c
4b37 79                 mov     a, c        ;get to accumulator
4b38                    ds      10h         ;space for output routine
4b48 c9                 ret
                ;
                list:   ;list character from register c
4b49 79                 mov     a, c        ;character to register a
4b4a c9                 ret                 ;null subroutine
                ;
                listst: ;return list status (0 if not ready, 1 if ready)
4b4b af                 xra     a           ;0 is always ok to return
4b4c c9                 ret
                ;
                punch:  ;punch character from register c
```

```
4b4d 79                 mov   a, c         ;character to register a
4b4e c9                 ret                ;null subroutine
                ;
                ;
                reader: ;read character into register a from reader device
4b4f 3e1a               mvi   a, 1ah       ;enter end of file for now (replace later)
4b51 e67f               ani   7fh          ;remember to strip parity bit
4b53 c9                 ret
                ;
                ;
                ;       i/o drivers for the disk follow
                ;       for now, we will simply store the parameters away for use
                ;       in the read and write subroutines
                ;
                home:   ;move to the track 00 position of current drive
                ;       translate this call into a settrk call with parameter 00
4b54 0e00               mvi   c, 0         ;select track 0
4b56 cd7d4b             call  settrk
4b59 c9                 ret                ;we will move to 00 on first read/write
                ;
                seldsk: ;select disk given by register c
4b5a 210000             lxi   h, 0000h     ;error return code
4b5d 79                 mov   a,c.
4b5e 32ef4c             sta   diskno
4b61 fe04               cpi   4            ;must be between 0 and 3
4b63 d0                 rnc                ;no carry if 4, 5, . . .
                ;       disk number is in the proper range
4b64                    ds    10           ;space for disk select
                ;       compute proper disk parameter header address
4b6e 3aef4c             lda   diskno
4b71 6f                 mov   l, a         ;l=disk number 0, 1, 2, 3
4b72 2600               mvi   h, 0         ;high order zero
4b74 29                 dad   h            ;*2
```

```
4b75 29                  dad    h           ;*4
4b76 29                  dad    h           ;*8
4b77 29                  dad    h           ;*16 (size of each header)
4b78 11334a              lxi    d, dpbase
4b7b 19                  dad    0           ;hl=.dpbase(diskno*16)
4b7c c9                  ret
                ;
                settrk:  ;set track given by register c
4b7d 79                  mov    a, c
4b7e 32e94c              sta    track
4b81                     ds     10h         ;space for track select
4b91 c9                  ret
                ;
                setsec:  ;set sector given by register c
4b92 79                  mov    a, c
4b93 32eb4c              sta    sector
4b96                     ds     10h         ;space for sector select
4ba6 c9                  ret
                ;
                sectran:
                         ;translate the sector given by bc using the
                         ;translate table given by de
4ba7 eb                  xchg               ;hl=.trans
4ba8 09                  dad    b           ;hl=.trans(sector)
4ba9 6e                  mov    l, m        ;l = trans(sector)
4baa 2600                mvi    h, 0        ;hl = trans(sector)
4bac c9                  ret                ;with value in hl
                ;
                setdma:  ;set dma address given by registers b and c
4bad 69                  mov    l, c        ;low order address
4bae 60                  mov    h, b        ;high order address
4baf 22ed4c              shld   dmaad       ;save the address
4bb2                     ds     10h         ;space for setting the dma address
```

```
4bc2 c9                ret
                ;
                read:   ;perform read operation (usually this is similar to write
                ;       so we will allow space to set up read command, then use
                ;       common code in write)
4bc3                    ds      10h             ;set up read command
4bd3 c3e64b             jmp     waitio          ;to perform the actual i/o
                ;
                write:  ;perform a write operation
4bd6                    ds      10h             ;set up write command
                ;
                waitio: ;enter here from read and write to perform the actual i/o
                ;       operation. return a 00h in register a if the operation completes
                ;       properly, and 01h if an error occurs during the read or write
                ;
                ;       in this case, we have saved the disk number in 'diskno' (0, 1)
                ;                               the track number in 'track' (0-76)
                ;                               the sector number in 'sector' (1-26)
                ;                               the dma address in 'dmaad' (0-65535)
4be6                    ds      256             ;space reserved for i/o drivers
4ce6 3e01               mvi     a, 1            ;error condition
4ce8 c9                 ret                     ;replaced when filled- in
                ;
                ;       the remainder of the cbios is reserved uninitialized
                ;       data area, and does not need to be a part of the
                ;       system memory image (the space must be available,
                ;       however, between "begdat" and "enddat").
                ;
4ce9            track:  ds      2               ;two bytes for expansion
4ceb            sector: ds      2               ;two bytes for expansion
4ced            dmaad:  ds      2               ;direct memory address
```

```
291     4cef            diskno:  ds    1          ;disk number 0-15
292                     ;
293                     ;        scratch ram area for bdos use
294     4cf0 =          begdat   equ   $          ;beginning of data area
295     4cf0            dirbf:   ds    128        ;scratch directory area
296     4d70            all00:   ds    31         ;allocation vector 0
297     4d8f            all01:   ds    31         ;allocation vector 1
298     4dae            all02:   ds    31         ;allocation vector 2
299     4dcd            all03:   ds    31         ;allocation vector 3
300     4dec            chk00:   ds    16         ;check vector 0
301     4dfc            chk01:   ds    16         ;check vector 1
302     4e0c            chk02:   ds    16         ;check vector 2
303     4e1c            chk03:   ds    16         ;check vector 3
304                     ;
305     4e2c            enddat   equ   $          ;end of data area
306     013c =          datsiz   equ   $-begdat;  ;size of data area
307     4e2c                     end

all00           4d70    43      296#
all01           4d8f    48      297#
all02           4dae    53      298#
all03           4dcd    58      299#
bdos            3c06    10#     154
begdat          4cf0    294#    306
bias            0000    8#      9
bios            4a00    11#     15
boot            4a9c    19      84#
ccp             3400    9#      10      11      16      101     163
cdisk           0004    12#     87      161
chk00           4dec    43      300#
chk01           4dfc    48      301#
chk02           4e0c    53      302#
chk03           4e1c    58      303#
```

conin	4b24	22	175#			
conout	4b37	23	180#			
const	4b11	21	170#			
datsiz	013c	306#				
dirbf	4cf0	42	47	52	57	295#
diskno	4cef	217	223	291#		
dmaad	4ced	258	290#			
dpbase	4a33	40#	230			
dpblk	4a8d	42	47	52	57	69#
enddat	4e2c	305#				
gocpm	4aef	88	124	147#		
home	4b54	27	94	208#		
iobyte	0003	13#	86			
list	4b49	24	185#			
listst	4b4b	34	189#			
load1	4aba	102#	130	144		
msize	0014	3#	8			
nsects	002c	16#	96			
punch	4b4d	25	193#			
read	4bc3	32	113	262#		
reader	4b4f	26	198#			
sector	4ceb	242	289#			
sectran	4ba7	35	246#			
seldsk	4b5a	28	93	214#		
setdma	4bad	31	110	158	255#	
setsec	4b92	30	107	240#		
settrk	4b7d	29	140	211	234#	
track	4ce9	236	288#			
trans	4a73	40	45	50	55	61#
waitio	4be6	266	271#			
wboot	4aa6	20	90#	115		
wboote	4a03	20#	150			
write	4bd6	33	268#			

Appendix C: A Skeletal GETSYS/PUTSYS Program

```
                         ;          combined getsys and putsys programs from
                         ;          Sec 6.4
                         ;          Start the programs at the base of the TPA

0100                                org  0100h

0014 =                   msize      equu 20                ; size of cp/m in Kbytes

                         ; "bias" is the amount to add to addresses for > 20k
                         ;          (referred to as "b" throughout the text)

0000 =                   bias       equ  (msize-20)*1024
3400 =                   ccp        equ  3400h+bias
3c00 =                   bdos       equ  ccp+0800h
4a00 =                   bios       equ  ccp+1600h

                         ;          getsys programs tracks 0 and 1 to memory at
                         ;          3880h + bias

                         ;          register               usage
                         ;          a                      (scratch register)
                         ;          b                      track count (0...76)
                         ;          c                      sector count (1...26)
                         ;          d,e                    (scratch register pair)
                         ;          h,l                    load address
                         ;          sp                     set to track address

                         gstart:                           ; start of getsys
0100 318033                         lxi  sp,ccp-0080h      ; convenient place
0103 218033                         lxi  h,ccp-0080h       ; set initial load
0106 0600                           mvi  b,0               ; start with track
                         rd$trk:                           ; read next track
```

```
0108 0e01                mvi  c,1           ; each track start
                 rd$sec:
010a cd0003              call read$sec      ; get the next sector
010d 118000              lxi  d,128         ; offset by one sector
0110 19                  dad  d             ;  (hl=hl+128)
0111 0c                  inr  c             ; next sector
0112 79                  mov  a,c           ; fetch sector number
0113 felb                cpi  27            ;  and see if last
0115 da0a01              jc   rdsec         ; <, do one more

                 ; arrive here at end of track, move to next track

0118 04                  inr  b             ; track = track+1
0119 78                  mov  a,b           ; check for last
011a fe02                cpi  2             ; track = 2 ?
011c da0801              jc   rd$trk        ; <, do another

                 ; arrive here at end of load, halt for lack of anything
                 ; better

011f fb                  ei
0120 76                  hlt
                 ;       putsys program, places memory image
                 ;       starting at
                 ;       3880h + bias back to tracks 0 and 1
                 ;       start this program at the next page boundary
0200                     org  ($+0100h) and 0ff00h

                 put$sys:
0200 318033              lxi  sp,ccp-0080h  ; convenient place
0203 218033              lxi  h,ccp-0080h   ; start of dump
0206 0600                mvi  b,0           ; start with track
                 wr$trk:
0208 0e01                mvi  c,1           ; start with sector
                 wr$sec:
020a cd0004              call write$sec     ; write one sector
020d 118000              lxi  d,128         ; length of each
0210 19                  dad  d             ; <hl>=<hl> + 128
0211 0c                  inr  c             ; <c> =<c> + 1
0212 79                  mov  a,c           ; see if
0213 felb                cpi  27            ; past end of track
0215 da0a02              jc   wr$sec        ; no, do another

                 ; arrive here at end of track, move to next track

0218 04                  inr  b             ; track = track+1
0219 78                  mov  a,b           ; see if
021a fe02                cpi  2             ;  last track
021c da0802              jc   wr$trk        ; no, do another

                 ;       done with putsys, halt for lack of anything
                 ;       better

02lf fb                  ei
0220 76                  hit
```

```
                  ; user supplied subroutines for sector read and write

                  ;         move to next page boundary

0300                        org  ($+0100h) and 0ff00h

                  read$sec:
                            ; read the next sector
                            ; track in <b>,
                            ; sector in <c>
                            ; dmaaddr in <hl>

0300 c5                     pushb
0301 e5                     pushh

                  ; user defined read operation goes here
0302                        ds   64

0342 el                     pop  h
0343 cl                     pop  b
0344 c9                     ret

0400                        org  ($+0100h) and 0ff00h ;another page
                                                      ;boundary

                  write$sec:

                            ; same parameters as read$sec

0400 c5                     pushb
0401 e5                     pushh

                  ; user defined write operation goes here
0402                        ds   64

0442 el                     pop  h
0443 cl                     pop  b
0444 c9                     ret

                  ; end of getsys/putsys program

0445                        end
```

Appendix D: The MDS-800 Cold Start Loader for CP/M 2

```
                              title   mds cold start loader at 3000h'
                     ;
                     ;        mds-800 cold start loader for cp/m 2.0
                     ;
                     ;        version 2.0 august, 1979
                     ;
0000 =               false    equ     0
ffff                 true     equ     not false
0000 =               testing  equ     false        if true, then go to mon80 on errors
                     ;
                              if      testing
                     bias     equ     03400h
                              endif
                              if      not testing
0000 =               bias     equ     0000h
                              endif
0000 =               cpmb     equ     bias         ;base of dos load
0806 =               bdos     equ     806h+bias    ;entry to dos for calls
1880 =               bdose    equ     1880h+bias   ;end of dos load
1600 =               boot     equ     1600h+bias   ;cold start entry point
1603 =               rboot    equ     boot+3       ;warm start entry point
                     ;
3000                          org     03000h       ;loaded down from hardware boot at 3000H
                     ;
1880 =               bdosl    equ     bdose-cpmb
0002 =               ntrks    equ     2            ;number of tracks to read
0031 =               bdoss    equ     bdosl/128    ;number of sectors in dos
0019 =               bdoso    equ     25           ;number of bdos sectors on track 0
0018 =               bdos1    equ     bdoss-bdoso  ;number of sectors on track 1
```

```
                        ;
f800 =                  mon80     equ   of800h    ;intel monitor base
ffof =                  rmon80    equ   offofh    ;restart location for mon80
0078 =                  base      equ   078h      ;'base' used by controller
0079 =                  rtype     equ   base+1    ;result type
007b =                  rbyte     equ   base+3    ;result byte
007f =                  reset     equ   base+7    ;reset controller
                        ;
0078 =                  dstat     equ   base      ;disk status port
0079 =                  ilow      equ   base+1    ;low iopb address
007a =                  ihigh     equ   base+2    ;high iopb address
00ff =                  bsw       equ   offh      ;boot switch
0003 =                  recal     equ   3h        ;recalibrate selected drive
0004 =                  readf     equ   4h        ;disk read function
0100 =                  stack     equ   100h      ;use end of boot for stack
                        ;
                        rstart:
3000 310001                       lxi   sp,stack; ;in case of call to mon80
                        ;         clear disk status
3003 db79                         in    rtype
3005 db7b                         in    rbyte
                        ;         check if boot switch is off
                        coldstart:
3007 dbff                         in    bsw
3009 e602                         ani   02h       ;switch on?
300b c20730                       jnz   coldstart
                        ;         clear the controller
300e d37f                         out   reset     ;logic cleared
                        ;
                        ;
3010 0602                         mvi   b,ntrks   ;number of tracks to read
3012 214230                       lxi   h,iopbo
                        ;
```

```
                        start:
                        ;
                        ;       read first/next track into cpmb
3015 7d                         mov     a,l
3016 d379                       out     ilow
3018 7c                         mov     a,h
3019 d37a                       out     ihigh
301b db78               waito:  in      dstat
301d e604                       ani     4
301f ca1b30                     jz      waito
                        ;
                        ;       check disk status
3022 db79                       in      rtype
3024 e603                       ani     11b
3026 fe02                       cpi     2
                        ;
                                if      testing
                                cnc     rmon80  ;go to monitor if 11 or 10
                                endif
                                if      not testing
3028 d20030                     jnc     rstart  ;retry the load
                                endif
                        ;
302b db7b                       in      rbyte   ;i/o complete, check status
                        ;       if not ready, then go to mon80
302d 17                         ral
302e dc0fff                     cc      rmon80  ;not ready bit set
3031 1f                         rar             ;restore
3032 e61e                       ani     11110b  ;overrun/addr err/seek/crc/xxxx
                        ;
                                if      testing
                                cnz     rmon80  ;go to monitor
                                endif
                                if      not testing
```

```
3034 c20030              jnz    rstart    ;retry the load
                         endif
                ;
                ;
3037 110700              lxi    d,iopbl   ;length of iopb
303a 19                  dad    d         ;addressing next iopb
303b 05                  dcr    b         ;count down tracks
303c c21530              jnz    start
                ;
                ;
                ;        jmp to boot to print initial message, and set up jmps
303f c30016              jmp    boot
                ;
                ;        parameter blocks
3042 80         iopbo:   db     80h       ;iocw, no update
3043 04                  db     readf     ;read function
3044 19                  db     bdoso     ;# sectors to read on track 0
3045 00                  db     0         ;track 0
3046 02                  db     2         ;start with sector 2 on track 0
3047 0000                dw     cpmb      ;start at base of bdos
0007 =          iopbl    equ    $-iopbo
                ;
3049 80         iopb1:   db     80h
304a 04                  db     readf
304b 18                  db     bdos1     ;sectors to read on track 1
304c 01                  db     1         ;track 1
304d 01                  db     1         ;sector 1
304e 800c                dw     cmpb+bdos0*128 ;base of second read
                ;
3050                     end
```

base	0078	33#	34	35	36	38	39	40
bdos	0806	18#						
bdoso	0019	28#	29	113	124			
bdos1	0018	29#	121					
bdose	1880	19#	25					
bdosl	1880	25#	27					
bdoss	0031	27#	29					
bias	0000	12#	15#	17	18	19	20	
boot	1600	20#	21	108				
bsw	00ff	41#	53					
coldstart	3007	52#	55					
cpmb	0000	17#	25	116	124			
dstat	0078	38#	70					
false	0000	7#	8	9				
ihigh	007a	40#	69					
ilow	0079	39#	67					
iopbo	3042	61	111#	117				
iopb1	3049	119#						
iopbl	0007	101	117#					
mon80	f800	31#						
ntrks	0002	26#	60					
rboot	1603	21#						
rbyte	007b	35#	50	86				
readf	0004	43#	112	120				
recal	0003	42#						
reset	007f	36#	57					
rmon80	ff0f	32#	80	89	94			
rstart	3000	46#	83	97				
rtype	0079	34#	49	75				
stack	0100	44#	47					
start	3015	63#	104					
testing	0000	9#	11	14	79	82	93	96
true	ffff	8#						
waito	301b	70#	72					

Appendix E: A Skeletal Cold Start Loader

```
                    ; this is a sample cold start loader, which, when
                    ; modified
                    ; resides on track 00, sector 01 (the first sector on the
                    ; diskette). we assume that the controller has loaded
                    ; this sector into memory upon system start-up (this
                    ; program can be keyed-in, or can exist in read/only
                    ; memory
                    ; beyond the address space of the cp/m version you are
                    ; running). the cold start loader brings the cp/m system
                    ; into memory at "loadp" (3400h + "bias"). in a 20k
                    ; memory system, the value of "bias" is 0000h, with
                    ; large
                    ; values for increased memory sizes (see section 2).
                    ; after
                    ; loading the cp/m system, the cold start loader
                    ; branches
                    ; to the "boot" entry point of the bios, which begins at
                    ; "bios" + "bias." the cold start loader is not used un-
                    ; til the system is powered up again, as long as the bios
                    ; is not overwritten. the origin is assumed at 0000h, an
                    ; must be changed if the controller brings the cold start
                    ; loader into another area, or if a read/only memory
                    ; area
                    ; is used.

0000                        org  0          ; base of ram in
                                            ; cp/m

0014 =              msize   equ  20         ; min mem size in
                                            ; kbytes
```

```
0000 =              bias    equ  (msize-20)*1024   ; offset from 20k
                                                   ; system
3400 =              ccp     equ  3400h+bias        ; base of the ccp
4a00 =              bios    equ  ccp+1600h         ; base of the bios
0300 =              biosl   equ  0300h             ; length of the bios
4a00 =              boot    equ  bios
1900 =              size    equ  bios+biosl-ccp    ; size of cp/m
                                                   ; system
0032 =              sects   equ  size/128          ; # of sectors to load

                    ;       begin the load operation

                    cold:
0000 010200                 lxi  b,2               ; b=0, c=sector 2
0003 1632                   mvi  d,sects           ; d=# sectors to
                                                   ; load
0005 210034                 lxi  h,ccp             ; base transfer
                                                   ; address
                    lsect:  ; load the next sector

                    ;       insert inline code at this point to
                    ;       read one 128 byte sector from the
                    ;       track given in register b, sector
                    ;       given in register c,
                    ;       into the address given by <hl>
                    ; branch to location "cold" if a read error occurs
                    ;
                    ;
                    ;            user supplied read operation goes
                    ;            here...
                    ;
                    ;

0008 c36b00                 jmp  past$patch        ; remove this
                                                   ; when patched
000b                        ds   60h

                    past$patch:
                    ; go to next sector if load is incomplete
006b 15                     dcr  d                 ; sects=sects-1
006c ca004a                 jz   boot              ; head for the bios

                    ;       more sectors to load
                    ;
                    ; we aren't using a stack, so use <sp> as scratch
                    ; register
                    ;       to hold the load address increment

006f 318000                 lxi  sp,128            ; 128 bytes per
                                                   ; sector
0072 39                     dad  sp                ; <hl> = <hl> +
                                                   128
```

```
0073 0c          inr  c        ; sector = sector + 1
0074 79          mov  a,c
0075 felb        cpi  27       ; last sector of
                               ; track?
0077 da0800      jc   lsect    ; no, go read
                               ; another

                 ; end of track, increment to next track

007a 0e01        mvi  c,l      ; sector = 1
007c 04          inr  b        ; track = track + 1
007d c30800      jmp  lsect    ; for another group
0080             end           ; of boot loader
```

Appendix F: CP/M Disk Definition Library

```
;               CP/M 2.0 disk re-definition library
;
;               Copyright © 1979
;               Digital Research
;               Box 579
;               Pacific Grove, CA
;               93950
;
;               CP/M logical disk drives are defined using the
;               macros given below, where the sequence of calls
;               is:
;
;               disks   n
;               diskdef parameter-list-0
;               diskdef parameter-list-1
;               ...
;               diskdef parameter-list-n
;               endef
;
;               where n is the number of logical disk drives attached
;               to the CP/M system, and parameter-list-i defines the
;               characteristics of the ith drive (i=0,1,...,n-1)
;
;               each parameter-list-i takes the form
;                       dn,fsc,lsc,[skf],bls,dks,dir,cks,ofs,[0]
;               where
;               dn      is the disk number 0,1,...,n-1
;               fsc     is the first sector number (usually 0 or 1)
;               lsc     is the last sector number on a track
;               skf     is optional "skew factor" for sector translate
;               bls     is the data block size (1024,2048,...,16384)
```

```
;          dks      is the disk size in bls increments (word)
;          dir      is the number of directory elements (word)
;          cks      is the number of dir elements to checksum
;          ofs      is the number of tracks to skip (word)
;          [0]      is an optional 0 which forces 16K/directory end
;
;          for convenience, the form
;                   dn,dm
;          defines disk dn as having the same characteristics as
;          a previously defined disk dm.
;
;          a standard four drive CP/M system is defined by
;                   disks            4
;                   diskdef          0,1,26,6,1024,243,64,64,2
;          dsk      set              0
;                   rept             3
;          dsk      set              dsk+1
;                   diskdef          %dsk,0
;                   endm
;                   endef
;
;          the value of "begdat" at the end of assembly defines the
;          beginning of the uninitialize ram area above the bios,
;          while the value of "enddat" defines the next location
;          following the end of the data area. the size of this
;          area is given by the value of "datsiz" at the end of the
;          assembly. note that the allocation vector will be quite
;          large if a large disk size is defined with a small block
;          size.
;
dskhdr     macro    dn
;;         define a single disk header list
dpe&dn:    dw       xlt&dn,0000h       ;translate table
```

```
                dw      0000h,0000h     ;scratch area
                dw      dirbuf,dpb&dn   ;dir buff,parm block
                dw      csv&dn,alv&dn   ;check, alloc vectors
                endm
;
disks           macro   nd
;;              define nd disks
ndisks          set     nd              ;; for later reference
dpbase          equ     $               ;base of disk parameter blocks
;;              generate the nd elements
dsknxt          set     0
                rept    nd
                dskhdr  %dsknxt
dsknxt          set     dsknxc+1
                endm
                endm
;
dpbhdr          macro   dn
dpb&dn          equ     $               ;disk parm block
                endm
;
ddb             macro   data,comment
;;              define a db statement
                db      data            comment
                endm
;
ddw             macro   data,comment
;;              define a dw statement
                dw      data            comment
                endm
;
gcd             macro   m,n
;;              greatest common divisor of m,n
;;              produces value gcdn as result
```

```
;;                 (used in sector translate table generation)
gcdm               set      m                     ;;variable for m
gcdn               set      n                     ;;variable for n
gcdr               set      0                     ;;variable for r
                   rept     65535
gcdx               set      gcdm/gcdn
gcdr               set      gcdm-gcdx*gcdn
                   if       gcdr = 0
                   exitm
                   endif
gcdm               set      gcdn
gcdn               set      gcdr
                   endm
                   endm
;
diskdef            macro    dn,fsc,lsc,skf,bls,dks,dir,cks,ofs,k16
;;                 generate the set statements for later tables
                   if       nul lsc
;;                 current  disk dn               same as previous fsc
dpb&dn             equ      dpb&fsc               ;equivalent parameters
als&dn             equ      als&fsc               ;same allocation vector size
css&dn             equ      css&fsc               ;same checksum vector size
xlt&dn             equ      xlt&fsc               ;same translate table
                   else
secmax             set      lsc-(fsc)             ;;sectors 0...secmax
sectors            set      secmax+1              ;;number of sectors
als&dn             set      (dks)/8               ;;size of allocation vector
                   if       ((dks) mod 8) ne 0
als&dn             set      als&dn+1
                   endif
css&dn             set      (cks)/4               ;;number of checksum elements
;;                 generate the block shift value
blkval             set      bls/128               ;;number of sectors/ block
```

```
blkshf  set     0                           ;;counts right 0's in blkval
blkmsk  set     0                           ;;fills with l's from right
        rept    16                          ;;once for each bit position
        if      blkval=1
        exitm
        endif
;;      otherwise, high order 1 not found yet
blkshf  set     blkshf+1
blkmsk  set     (blkmsk shl l) or l
blkval  set     blkval/2
        endm
;;      generate the extent mask byte
blkval  set     bls/1024                    ;;number of kilobytes/ block
extmsk  set     0                           ;;fill from right with l's
        rept    16
        if      blkval=1
        exitm
        endif
;;      otherwise more to shift
extmsk  set     (extmsk shl l) or l
blkval  set     blkval/2
        endm
;;      may be double byte allocation
        if      (dks) > 256
extmsk  set     (extmsk shr l)
        endif
;;      may be optional [0] in last position
        if      not nul k16
extmsk  set     k16
        endif
;;      now generate directory reservation bit vector
dirrem  set     dir                         ;;# remaining to process
dirbks  set     bls/32                      ;;number of entries per block
dirblk  set     0                           ;;fill with l's on each loop
        rept    16
        if      dirrem=0
        exitm
        endif
```

```
;;              not complete, iterate once again
;;              shift right and add 1 high order bit
dirblk          set     (dirblk shr l) or 8000h
                if      dirrem > dirbks
dirrem          set     dirrem-dirbks
                else
direem          set     0
                endif
                endm
                dpbhdr  dn                      ;;generate equ $
                ddw     %sectors,<;sec per track>
                ddb     %blkshf,<;block shift>
                ddb     %blkmsk,<;block mask>
                ddb     %extmsk,<;extnt mask>
                ddw     %(dks)-1,<;disk size-1>
                ddw     %(dir)-1,<;directory max>
                ddb     %dirblk shr 8,<;alloc0>
                ddb     %dirblk and 0ffh,<;allocl>
                ddw     %(cks)/4,<;check size>
                ddw     %ofs,<;offset>
;;              generate the translate table, if requested
                if      nul skf
xlt&dn          equ     0                       ;no xlate table
                else
                if      skf = 0
xlt&dn          equ     0                       ;no xlate table
                else
;;              generate the translate table
nxtsec          set     0                       ;;next sector to fill
nxtbas          set     0                       ;;moves by one on overflow
                gcd     %sectors,skf
;;              gcdn = gcd(sectors,skew)
neltst          set     sectors/gcdn
;;              neltst is number of elements to generate
```

```
;;              before we overlap previous elements
nelts           set     neltst          ;;counter
xlt&dn          equ     $               ;translate table
                rept    sectors         ;;once for each sector
                if      sectors < 256
                ddb     %nxtsec+(fsc)
                else
                ddw     %nxtsec+(fsc)
                endif
nxtsec          set     nxtsec+(skf)
                if      nxtsec >= sectors
nxtsec          set     nxtsec-sectors
                endif
nelts           set     nelts-1
                if      nelts = 0
nxtbas          set     nxtbas+1
nxtsec          set     nxtbas
nelts           set     neltst
                endif
                endm
                endif   ;;end of nul fac test
                endif   ;;end of nul bls test
                endm
;
defds           macro   lab,space
lab:            ds      space
                endm
;
lds             macro   lb,dn,val
                defds   lb&dn,%val&dn
                endm
;
endef           macro
;;              generate the necessary ram data areas
```

```
begdat      equ     $
dirbuf:     ds      128             ;directory access buffer
dsknxt      set     0
            rept    ndisks          ;;once for each disk
            lds     alv,%dsknxt,als
            lds     csv,%dsknxt,ccs
dsknxt      set     dsknxt+1
            endm
enddat      equ     $
datsiz      equ     $-begdat
;;          db 0 at this point forces hex record
            endm
```

Appendix G: Blocking and Deblocking Algorithms

```
                ;
                ;
                ;       sector deblocking algorithms for cp/m 2.0
                ;
                ;
                ;
                ;       utility macro to compute sector mask
                smask   macro   hblk
                ;;      compute log2(hblk), return @x as result
                ;;      (2 ** @x = hblk on return)
                @y      set     hblk
                @x      set     0
                ;;      count right shifts of @y until = 1
                        rept    8
                        if      @y = 1
                        exitm
                        endif
                ;;      @y is not 1, shift right one position
                @y      set     @y shr 1
                @x      set     @x + 1
                        endm
                        endm
                ;
                ;
                ;
                ;       cp/m to host disk constants
                ;
                ;
     0800 =     blksiz  equ     2048            ;cp/m allocation size
     0200 =     hstsiz  equ     512             ;host disk sector size
     0014 =     hstspt  equ     20              ;host disk sectors/trk
     0004 =     hstblk  equ     hstsiz/128      ;cp/m sects/host buff
     0050 =     cpmspt  equ     hstblk * hstspt ;cp/m sectors/track
```

```
0003 =    secmsk   equ     hstblk-1            ;sector mask
                   smask   hstblk              ;compute sector mask
0002 =    secshf   equ     @x                  ;log2(hstblk)
          ;
          ;
          ;
          ;        bdos constants on entry to write
          ;
          ;
0000 =    wrall    equ     0                   ;write to allocated
0001 =    wrdir    equ     1                   ;write to directory
0002 =    wrual    equ     2                   ;write to unallocated
          ;
          ;
          ;
          ;        the bdos entry points given below show the
          ;        code which is relevant to deblocking only.
          ;
          ;
          ;
          ;        diskdef macro, or hand coded tables go here
0000 =    dpbase   equ     $                   ;disk param block base
          ;
          boot:
          wboot:
                   ;enter here on system boot to initialize
0000 af            xra     a                   ;0 to accumulator
0001 326a01        sta     hstact              ;host buffer inactive
0004 326c01        sta     unacnt              ;clear unalloc count
0007 c9            ret
          ;
          home:
                   ;home the selected disk
          home:
0008 3a6b01        lda     hstwrt            ;check for pending write
000b b7            ora     a
000c c21200        jnz     homed
000f 326a01        sta     hstact            ;clear host active flag
          homed:
0012 c9            ret
          ;
          seldsk:
                   ;select disk
0013 79            mov     a,c                 ;selected disk number
0014 326101        sta     sekdsk              ;seek disk number
0017 6f            mov     l,a                 ;disk number to hl
0018 2600          mvi     h,0
                   rept    4                   ;multiply by 16
                   dad     h
                   endm
001a+29            dad     h
001b+29            dad     h
001c+29            dad     h
001d+29            dad     h
001e 110000        lxi     d,dpbase            ;base of parm block
```

```
0021 19             dad     d                   ;hl=.dpb(curdsk)
0022 c9             ret
          ;
          settrk:
                    ;set track given by registers bc
0023 60             mov     h,b
0024 69             mov     l,c
0025 226201         shld    sektrk              ;track to seek
0028 c9             ret
          ;
          setsec:
                    ;set sector given by register c
0029 79             mov     a,c
002a 326401         sta     seksec              ;sector to seek
002d c9             ret
          ;
          setdma:
                    ;set dma address given by bc
002e 60             mov     h,b
002f 69             mov     l,c
0030 227501         shld    dmaadr
0033 c9             ret
          ;
          sectran:
                    ;translate sector number bc
0034 60             mov     h,b
0035 69             mov     l,c
0036 c9             ret
          ;
          ;
          ;
          ;         the read entry point takes the place of
          ;         the previous bios definition for read.
          ;
          ;
          read:
                    ;read the selected cp/m sector
0037 af             xra     a
0038 326c01         sta     unacnt
003b 3e01           mvi     a,1
003d 327301         sta     readop              ;read operation
0040 327201         sta     rsflag              ;must read data
0043 3e02           mvi     a,wrual
0045 327401         sta     wrtype              ;treat as unalloc
0048 c3b600         jmp     rwoper              ;to perform the read
          ;
          ;
          ;
          ;         the write entry point takes the place of
          ;         the previous bios definition for write.
          ;
          ;
          write:
                    ;write the selected cp/m sector
004b af             xra     a                   ;0 to accumulator
```

```
0004c 327301        sta     readop          ;not a read operation
004f 79             mov     a,c             ;write type in c
0050 327401         sta     wrtype
0053 fe02           cpi     wrual           ;write unallocated?
0055 c26f00         jnz     chkuna          ;check for unalloc
               ;
               ;    write to unallocated, set parameters
0058 3e10           mvi     a,blksiz/128    ;next unalloc recs
005a 326c01         sta     unacnt
005d 3a6101         lda     sekdsk          ;disk to seek
0060 326d01         sta     unadsk          ;unadsk = sekdsk
0063 2a6201         lhld    sektrk
0066 226e01         shld    unatrk          ;unatrk = sectrk
0069 3a6401         lda     seksec
006c 327001         sta     unasec          ;unasec = seksec
               ;
               chkuna:
                    ;check for write to unallocated sector
006f 3a6c01         lda     unacnt          ;any unalloc remain?
0072 b7             ora     a
0073 caae00         jz      alloc           ;skip if not
               ;
               ;    more unallocated records remain
0076 3d             dcr     a               ;unacnt = unacnt-1
0077 326c01         sta     unacnt
007a 3a6101         lda     sekdsk          ;same disk?
007d 216d01         lxi     h,unadsk
0080 be             cmp     m               ;sekdsk = unadsk?
0081 c2ae00         jnz     alloc           ;skip if not
               ;
               ;    disks are the same
0084 216e01         lxi     h,unatrk
0087 cd5301         call    sektrkcmp       ;sektrk = unatrk?
008a c2ae00         jnz     alloc           ;skip if not
               ;
               ;    tracks are the same
008d 3a6401         lda     seksec          ;same sector?
0090 217001         lxi     h,unasec
0093 be             cmp     m               ;seksec = unasec?
0094 c2ae00         jnz     alloc           ;skip if not
               ;
               ;    match, move to next sector for future ref
0097 34             inr     m               ;unasec = unasec+1
0098 7e             mov     a,m             ;end of track?
0099 fe50           cpi     cpmspt          ;count cp/m sectors
009b daa700         jc      noovf           ;skip if no overflow
               ;
               ;    overflow to next track
009e 3600           mvi     m,o             ;unasec = o
00a0 2a6e01         lhld    unatrk
00a3 23             inx     h
00a4 226e01         shld    unatrk          ;unatrk = unatrk+1
               ;
               noovf:
                    ;match found, mark as unnecessary read
00a7 af             xra     a               ;0 to accumulator
```

```
00ab 327201     sta     rsflag          ;rsflag = 0
00ab c3b600     jmp     rwoper          ;to perform the write
            ;
            alloc:
                ;not an unallocated record, requires pre-read
00ae af         xra     a               ;0 to accum
00af 326c01     sta     unacnt          ;unacnt = 0
00b2 3c         inr     a               ;1 to accum
00b3 327201     sta     rsflag  = 1     ;rsflag = 1
            ;
            ;
            ;
            ;   common code for read and write follows
            ;
            ;
            rwoper:
                ;enter here to perform the read/write
00b6 af         xra     a               ;zero to accum
00b7 327101     sta     erflag          ;no errors (yet)
00ba 3a6401     lda     seksec          ;compute host sector
                rept    secshf
                ora     a               ;carry = 0
                rar                     ;shift right
                endm
00bd+b7         ora     a               ;carry = 0
00be+1f         rar                     ;shift right
00bf+b7         ora     a               ;carry = 0
00c0+1f         rar                     ;shift right
00c1 326901     sta     sekhst          ;host sector to seek
            ;
            ;   active host sector?
00c4 216a01     lxi     h,hstact        ;host active flag
00c7 7e         mov     a,m
00c8 3601       mvi     m,1             ;always becomes 1
00ca b7         ora     a               ;was it already?
00cb caf200     jz      filhst          ;fill host if not
            ;
            ;   host buffer active, same as seek buffer?
00ce 3a6101     lda     sekdsk
00d1 216501     lxi     h,hstdsk        ;same disk?
00d4 be         cmp     m               ;sekdsk = hstdsk?
00d5 c2eb00     jnz     nomatch
            ;
            ;   same disk, same track?
00d8 216601     lxi     h,hsttrk
00db cd5301     call    sektrkcmp       ;sektrk = hsttrk?
00de c2eb00     jnz     nomatch
            ;
            ;   same disk, same track, same buffer?
00e1 3a6901     lda     sekhst
00e4 216801     lxi     h,hstsec        ;sekhst = hstsec?
00e7 be         cmp     m
00e8 ca0f01     jz      match           ;skip if match
            ;
            nomatch:
```

```
                        ;proper disk, but not correct sector
00eb 3a6b01             lda     hstwrt          ;host written?
00ee b7                 ora     a
00ef c45f01             cnz     writehst        ;clear host buff
            ;
            filhst:
                        ;may have to fill the host buffer
00f2 3a6101             lda     sekdsk
00f5 326501             sta     hstdsk
00f8 2a6201             lhld    sektrk
00fb 226601             shld    hsttrk
00fe 3a6901             lda     sekhst
0101 326801             sta     hstsec
0104 3a7201             lda     rsflag          ;need to read?
0107 b7                 ora     a
0108 c46001             cnz     readhst         ;yes, if 1
010b af                 xra     a               ;0 to accum
010c 326b01             sta     hstwrt          ;no pending write
                        ;
            match:
                        ;copy data to or from buffer
010f 3a6401             lda     seksec          ;mask buffer number
0112 e603               ani     secmsk          ;least signif bits
0114 6f                 mov     l,a             ;ready to shift
0115 2600               mvi     h,0             ;double count
                        rept    7               ;shift left 7
                        dad     h
                        endm
0117+29                 dad     h
0118+29                 dad     h
0119+29                 dad     h
011a+29                 dad     h
011b+29                 dad     h
011c+29                 dad     h
011d+29                 dad     h
            ;           hl has relative host buffer address
011e 117701             lxi     d,hstbuf
0121 19                 dad     d               ;hl = host address
0122 eb                 xchg                    ;now in de
0123 2a7501             lhld    dmaadr          ;get/put cp/m data
0126 0e80               mvi     c,128           ;length of move
0128 3a7301             lda     readop          ;which way?
012b b7                 ora     a
012c c23501             jnz     rwmove          ;skip if read
            ;
            ;           write operation, mark and switch direction
012f 3e01               mvi     a,1
0131 326b01             sta     hstwrt          ;hstwrt = 1
0134 eb                 xchg                    ;source/dest swap
            ;
            rwmove:
                        ;c initially 128, de is source, hl is dest
0135 1a                 ldax    d               ;source character
0136 13                 inx     d
0137 77                 mov     m,a             ;to dest
```

```
0138 23         inx     h
0139 od         dcr     c               ;loop 128 times
013a c23501     jnz     rwmove
          ;
          ;     data has been moved to/from host buffer
013d 3a7401     lda     wrtype          ;write type
0140 fe01       cpi     wrdir           ;to directory?
0142 3a7101     lda     erflag          ;in case of errors
0145 c0         rnz                     ;no further processing
          ;
          ;     clear host buffer for directory write
0146 b7         ora     a               ;errors?
0147 c0         rnz                     ;skip if so
0148 af         xra     a               ;0 to accum
0149 326b01     sta     hstwrt          ;buffer written
014c cd5f01     call    writehst
014f 3a7101     lda     erflag
0152 c9         ret
          ;
          ;
          ;
          ;     utility subroutine for 16-bit compare
          ;
          ;
          sektrkcmp:
                ;hl = .unatrk or .hsttrk, compare with sektrk
0153 eb         xchg
0154 216201     lxi     h,sektrk
0157 1a         ldax    d               ;low byte compare
0158 be         cmp     m               ;same?
0159 c0         rnz                     ;return if not
          ;     low bytes equal, test high 1s
015a 13         inx     d
015b 23         inx     h
015c 1a         ldax    d
015d be         cmp     m               ;sets flags
015e c9         ret
          ;
          ;
          ;
          ;     writehst performs the physical write to
          ;     the host disk, readhst reads the physical
          ;     disk.
          ;
          ;
          writehst:
                ;hstdsk = host disk #, hsttrk = host track #,
                ;hstsec = host sect #. write "hstsiz" bytes
                ;from hstbuf and return error flag in erflag.
                ;return erflag non-zero if error
015f c9         ret
          ;
          readhst:
                ;hstdsk = host disk #, hsttrk = host track #,
                ;hstsec = host sect #. read "hstsiz" bytes
```

```
                        ;into hstbuf and return error flag in erflag.
0160 c9                 ret
            ;
            ;
            ;
            ;           uninitialized ram data areas
            ;
            ;
            ;
0161        sekdsk:     ds      1               ;seek disk number
0162        sektrk:     ds      2               ;seek track number
0164        seksec:     ds      1               ;seek sector number
            ;
0165        hstdsk:     ds      1               ;host disk number
0166        hsttrk:     ds      2               ;host track number
0168        hstsec:     ds      1               ;host sector number
            ;
0169        sekhst:     ds      1               ;seek shr secshf
016a        hstact:     ds      1               ;host active flag
016b        hstwrt:     ds      1               ;host written flag
            ;
016c        unacnt:     ds      1               ;unalloc rec cnt
016d        unadsk:     ds      1               ;last unalloc disk
016e        unatrk:     ds      2               ;last unalloc track
0170        unasec:     ds      1               ;last unalloc sector
            ;
0171        erflag:     ds      1               ;error reporting
0172        rsflag:     ds      1               ;read sector flag
0173        readop:     ds      1               ;1 if read operation
0174        wrtype:     ds      1               ;write operation type
0175        dmaadr:     ds      2               ;last dma address
0177        hstbuf:     ds      hstsiz          ;host buffer
            ;
            ;
            ;
            ;           the endef macro invocation goes here
            ;
            ;
0377                    end
```


alloc	00ae	164	172	177	183	203#	
blksiz	0800	29#	151				
boot	0000	57#					
chkuna	006f	148	160#				
cpmspt	0050	33#	188				
dmaadr	0175	109	294	395#			
dpbase	0000	55#	88				
erflag	0171	218	317	326	391#		
filhst	00f2	235	260#				
home	0008	65#	67#				
homed	0012	70	72#				
hstact	016a	61	71	231	383#		
hstblk	0004	32#	33	34	35		
hstbuf	0177	291	396#				
hstdsk	0165	239	263	378#			
hstsec	0168	250	267	380#			
hstsiz	0200	30#	32	396			
hstspt	0014	31#	33				
hsttrk	0166	244	265	379#			
hstwrt	016b	68	256	272	302	324	384#
match	010f	252	274#				
nomatch	00eb	241	246	254#			
noovf	00a7	189	197#				
read	0037	124#					
readhst	0160	270	362#				
readop	0173	129	144	296	393#		
rsflag	0172	130	200	208	268	392#	
rwmove	0135	298	305#	312			
rwoper	00b6	133	201	215#			
secmsk	0003	34#	277				
secshf	0002	36#	220				
sectran	0034	112#					
sekdsk	0161	78	153	169	238	262	374#
sekhst	0169	228	249	266	382#		
seksec	0164	102	157	180	219	276	376#
sektrk	0162	96	155	264	337	375#	
sektrkcmp	0153	176	245	334#			
seldsk	0013	75#					
setdma	002e	105#					
setsec	0029	99#					
settrk	0023	92#					
unacnt	016c	62	127	152	162	168	206	386#
unadsk	016d	154	170	387#			
unasec	0170	158	181	389#			
unatrk	016e	156	175	193	195	388#	
wboot	0000	58#					
wrall	0000	43#					
wrdir	0001	44#	316				
write	004b	141#					
writehst	015f	258	325	355#			
wrtype	0174	132	146	315	394#		
wrual	0002	45#	131	147			

17	11	Search For First	DE = FCB Address	A = Directory Code
18	12	Search For Next	none	A = Directory Code
19	13	Delete File	DE = FCB Address	A = none
20	14	Read Sequential	DE = FCB Address	A = Error Code
21	15	Write Sequential	DE = FCB Address	A = Error Code
22	16	Make File	DE = FCB Address	A = FF if no DIR Space
23	17	Rename File	DE = FCB Address	A = FF if not found
24	18	Return Login Vector	none	HL = Login Vector*
25	19	Return Current Disk	none	A = Current Disk Number
26	1A	Set DMA Address	DE = DMA Address	none
27	1B	Get ADDR (ALLOC)	none	HL = ALLOC Address*
28	1C	Write Protect Disk	none	none
29	1D	Get Read/only Vector	none	HL = R/O Vector Value*
30	1E	Set File Attributes	DE = FCB Address	A = none
31	1F	Get ADDR (Disk Parms)	none	HL = DPB Address
32	20	Set/Get User Code	E = 0FFH for Get E = 00 to 0FH for Set	User Number
33	21	Read Random	DE = FCB Address	A = Error Code
34	22	Write Random	DE = FCB Address	A = Error Code
35	23	Compute File Size	DE = FCB Address	r0, r1, r2
36	24	Set Random Record	DE = FCB Address	r0, r1, r2
37	25	Reset Drive	DE = Drive Vector	A = 0
38	26	Access Drive	not supported	
39	27	Free Drive	not supported	
40	28	Write Random with Fill	DE = FCB	A = Error Code

*Note that A = L, and B = H upon return.