6

The Basic Input/Output System

This chapter takes a closer look at the Basic Input/Output System (BIOS). The BIOS provides the software link between the Console Command Processor (CCP), the Basic Disk Operating System (BDOS), and the physical hardware of your computer system. The CCP and BDOS interact with the parts of your computer system only as logical devices. They can therefore remain unchanged from one computer system to the next. The BIOS, however, is customized for your particular type of computer and disk drives. The only predictable part of the BIOS is the way in which it interfaces to the CCP and BDOS. This must remain the same no matter what special features are built into the BIOS.

The BIOS Components

A standard BIOS consists of low-level subroutines that drive four types of physical devices:

- Console: CP/M communicates with the outside world via the console. Normally this will be a video terminal or a hard-copy terminal.
- "Reader" and "punch": These devices are normally used to communicate between computer systems—the names "reader" and "punch" are just historical relics from the early days of CP/M.
- List: This is a hard-copy printer, either letter-quality or dot-matrix.
- Disk drives: These can be anything from the industry standard single-sided, single-density, 8-inch floppy diskette drives to hard disk drives with capacities of several hundred megabytes.

The BIOS Entry Points

The first few instructions of the BIOS are all jump (JMP) instructions. They transfer control to the 17 different subroutines in the BIOS. The CCP and the BDOS, when making a specific request of the BIOS, do so by transferring control to the appropriate JMP instruction in this BIOS *jump table* or *jump vector.* The BIOS jump vector always starts at the beginning of a 256-byte page, so the address of the first jump instruction is always of the form xx00H, where "xx" is the page address. Location 0000H to 0002H has a jump instruction to the second entry of the BIOS jump vector—so you can always find the page address of the jump vector by looking in location 0002H.

Figure 6-1 shows the contents of the BIOS jump vector along with the page-relative address of each jump. The labels used in the jump instructions have been adopted by convention.

The following sections describe the functions of each of the BIOS's main subroutines. You should also refer to Digital Research's manual *CP/M 2.0 Alteration Guide* for their description of the BIOS routines.

Bootstrap Functions

There are two bootstrap functions. The cold bootstrap loads the entire CP/M operating system when the system is either first turned on or reset. The warm bootstrap reloads the CCP whenever a program branches to location 0000H.

```
xx00H   JMP  BOOT     ;"Cold" (first time) bootstrap
xx03H   JMP  WBOOT    ;"Warm" bootstrap
xx06H   JMP  CONST    ;Console input status
xx09H   JMP  CONIN    ;Console input
xx0CH   JMP  CONOUT   ;Console output
xx0FH   JMP  LIST     ;List output
xx12H   JMP  PUNCH    ;"Punch" output
xx15H   JMP  READER   ;"Reader" input
xx18H   JMP  HOME     ;Home disk heads (to track 0)
xx1BH   JMP  SELDSK   ;Select logical disk
xx1EH   JMP  SETTRK   ;Set track number
xx21H   JMP  SETSEC   ;Set sector number
xx24H   JMP  SETDMA   ;Set DMA address
xx27H   JMP  READ     ;Read (128-byte) sector
xx2AH   JMP  WRITE    ;Write (128-byte) sector
xx2DH   JMP  LISTST   ;List device output status
xx30H   JMP  SECTRAN  ;Sector translate
```

Figure 6-1. Layout of the standard BIOS jump vector

BOOT: "Cold" Bootstrap

The BOOT jump instruction is the first instruction executed in CP/M. The bootstrap sequence must transfer control to the BOOT entry point in order to bring up CP/M. In general, a PROM receives control either when power is first applied or after you press the RESET button on the computer. This reads in the CP/M loader on the first sector of the physical disk drive chosen to be logical disk A. This CP/M loader program reads the binary image of the CCP, BDOS, and BIOS into memory at some predetermined address. Then it transfers control to the BOOT entry point in the BIOS jump vector.

This BOOT routine must initialize all of the required computer hardware. It sets up the baud rates for the physical console (if this has not already been done during the bootstrap sequence), the "reader," "punch," and list devices, and the disk controller. It must also set up the base page of memory so that there is a jump at location 0000H to the warm boot entry point in the BIOS jump vector (at xx03H) and a jump at location 0005H to the BDOS entry point.

Most BOOT routines sign on by displaying a short message on the console, indicating the current version of CP/M and the computer hardware that this BIOS can support.

The BOOT routine terminates by transferring control to the start of the CCP +6 bytes (the CCP has its own small jump vector at the beginning). Just before the BOOT routine jumps into the CCP, it sets the C register to 0 to indicate that logical disk A is to be the default disk drive. This is what causes "A>" to be the CCP's initial prompt.

The actual CCP entry point is derived from the base address of the BIOS. The CCP and BDOS together require 1E00H bytes of code, so the first instruction of the CCP starts at BIOS −1E00H.

WBOOT: "Warm" Bootstrap

Unlike the "cold" bootstrap entry point, which executes only once, the WBOOT or warm boot routine will be executed every time a program terminates by jumping to location 0000H, or whenever you type a CONTROL-C on the console as the first character of an input line.

The WBOOT routine is responsible for reloading the CCP into memory. Programs often use all of memory up to the starting point of the BDOS, overwriting the CCP in the process. The underlying philosophy is that while a program is executing, the CCP is not needed, so the program can use the memory previously occupied by the CCP. The CCP occupies 800H (2048) bytes of memory—and this is frequently just enough to make the difference between a program that cannot run and one that can.

A few programs that are self-contained and do not require the BDOS's facilities will also overwrite the BDOS to get another 1600H (5632) bytes of memory. Therefore, to be really safe, the WBOOT routine should read in both the CCP and the BDOS. It also needs to set up the two JMPs at location 0000H (to WBOOT itself) and at location 0005H (to the BDOS). Location 0003H should be set to the initial value of the IOBYTE if this is implemented in the BIOS.

As its last act, the WBOOT routine sets register C to indicate which logical disk is to be selected (C = 0 for A, 1 for B, and so on). It then transfers control into the CCP at the first instruction in order to restart the CCP. Again, the actual address is computed based on the knowledge that the CCP starts 1E00H bytes lower in memory than the base address of the BIOS.

Character Input/Output Functions

Character input/output functions deal with logical devices: the console, "reader," "punch," and list devices. Because these logical devices can in practice be connected by software to one of several physical character I/O devices, many BIOS's use CP/M's IOBYTE features to assign logical devices to physical ones.

In this case, each of the BIOS functions must check the appropriate bit fields of the IOBYTE (see Figure 4-2 and Table 4-1) to transfer control to the correct physical device *driver* (program that controls a physical device).

CONST: Console Input Status

CONST simply returns an indicator showing whether there is an incoming character from the console device. The convention is that A = 0FFH if a character is waiting to be processed, A = 0 if one is not. Note that the zero flag need not be set to reflect the contents of the A register—it is the contents that are important.

CONST is called by the CCP whenever the CCP is in the middle of an operation that can be interrupted by pressing a keyboard character.

The BDOS will call CONST if a program makes a Read Console Status function call (B$CONST, code 11, 0BH). It is also called by the console input BIOS routine, CONIN (described next).

CONIN: Console Input

CONIN reads the next character from the console to the A register and sets the most significant (parity) bit to 0.

Normally, CONIN will call the CONST routine until it detects A = 0FFH. Only then will it input the data character and mask off the parity bit.

CONIN is called by the CCP and by the BDOS when a program executes a Read Console Byte function (B$CONIN, code 1).

CONOUT: Console Output

CONOUT outputs the character (in ASCII) in register C to the console. The most significant (parity) bit of the character will always be 0.

CONOUT must first check that the console device is ready to receive more data, delaying if necessary until it is, and only then sending the character to the device.

CONOUT is called by the CCP and by the BDOS when a program executes a Write Console Byte function (B$CONOUT, code 2).

LIST: List Output

LIST is similar to CONOUT except that it sends the character in register C to the list device. It too checks first that the list device is ready to receive the character.

LIST is called by the CCP in response to the CONTROL-P toggle for printer echo of console output, and by the BDOS when a program makes a Write Printer Byte or Display String call (B$LISTOUT and B$PRINTS, codes 5 and 9).

PUNCH: "Punch" Output

PUNCH sends the character in register C to the "punch" device. As mentioned earlier, the "punch" is rarely a real paper tape punch. In most BIOS's, the PUNCH entry point either returns immediately and is effectively a null routine, or it outputs the character to a communications device, such as a modem, on your computer.

PUNCH must check that the "punch" device is indeed ready to accept another character for output, and must wait if it is not.

Digital Research's documentation states that the character to be output will always have its most significant bit set to 0. This is not true. The BDOS simply transfers control over to the PUNCH entry point in the BIOS; the setting of the most significant bit will be determined by the program making the BDOS function request (B$PUNOUT, code 4). This is important because the requirement of a zero would preclude being able to send pure binary data via the BIOS PUNCH function.

READER: "Reader" Input

As with the PUNCH entry point, the READER entry point rarely connects to a real paper tape reader.

The READER function must return the next character from the reader device in the A register, waiting, if need be, until there is a character.

Digital Research's documentation again says that the most significant bit of the A register must be 0, but this is not the case if you wish to receive pure binary information via this function.

READER is called whenever a program makes a Read "Reader" Byte function request (B$READIN, code 3).

Disk Functions

All of the disk functions that follow were originally designed to operate on the 128-byte sectors used on single-sided, single-density, 8-inch floppy diskettes that were standard in the industry at the time. Now that CP/M runs on many different types of disks, some of the BIOS disk functions seem strange because most of the new disk drives use sector sizes other than 128 bytes.

To handle larger sector sizes, the BIOS has some additional code that makes the BDOS respond as if it were still handling 128-byte sectors. This code is referred to as the *blocking/deblocking* code. As its name implies, it blocks together several 128-byte "sectors" and only writes to the disk when a complete *physical* sector has been assembled. When reading, it reads in a physical sector and then deblocks it, handing back several 128-byte "sectors" to the BDOS.

To do all of this, the blocking/deblocking code uses a special buffer area of the same size as the physical sectors on the disk. This is known as the host disk buffer or HSTBUF. Physical sectors are read into this buffer and written to the disk from it.

In order to optimize this blocking/deblocking routine, the BIOS has code in it to reduce the number of times that an actual disk read or write occurs. A side effect is that at any given moment, several 128-byte "sectors" may be stored in the HSTBUF, waiting to be written out to the disk when HSTBUF becomes full. This sometimes complicates the logic of the BIOS disk functions. You cannot simply select a new disk drive, for example, when the HSTBUF contains data destined for another disk drive. You will see this complication in the BIOS only in the form of added logical operations; the BIOS disk functions rarely trigger immediate physical operations. It is easier to understand these BIOS functions if you consider that they make *requests*—and that these requests are satisfied only when it makes sense to do so, taking into account the blocking/deblocking logic.

HOME: Home Disk

HOME sets the requested track and sector to 0.

SELDSK: Select Disk

SELDSK does not do what its name implies. It does not (and must not) physically select a logical disk. Instead, it returns a pointer in the HL register pair to the disk parameter header for the logical disk specified in register C on entry. C = 0 for drive A, 1 for drive B, and so on. SELDSK also stores this code for the requested disk to be used later in the READ and WRITE functions.

If the logical disk code in register C refers to a nonexistent disk or to one for which no disk parameter header exists, then SELDSK must return with HL set to 0000H. Then the BDOS will output a message of the form

```
"BDOS Err on X: Select"
```

Note that SELDSK not only does not select the disk, but also does not indicate whether or not the requested disk is physically present—merely whether or not there are disk tables present for the disk.

SELDSK is called by the BDOS either during disk file operations or by a program issuing a Select Disk request (B$SELDSK, code 14).

SETTRK: Set Track

SETTRK saves the requested disk track that is in the BC register pair when SETTRK gets control. Note that this is an absolute track number; that is, the number of reserved tracks before the file directory will have been added to the track number relative to the start of the logical disk.

The number of the requested track will be used in the next BIOS READ or WRITE function (described later in this chapter).

SETTRK is called by the BDOS when it needs to read or write a 128-byte sector. Legitimate track numbers are from 0 to 0FFFFH (65,535).

SETSEC: Set Sector

SETSEC is similar to SETTRK in that it stores the requested sector number for later use in BIOS READ or WRITE functions. The requested sector number is handed to SETSEC in the A register; legitimate values are from 0 to 0FFH (255).

The sector number is a logical sector number. It does not take into account any sector skewing that might be used to improve disk performance.

SETSEC is called by the BDOS when it needs to read or write a 128-byte sector.

SETDMA: Set DMA Address

SETDMA saves the address in the BC register pair in the requested DMA address. The next BIOS READ or WRITE function will use the DMA address as a pointer to the 128-byte sector buffer into which data will be read or from which data will be written.

The default DMA address is 0080H. SETDMA is called by the BDOS when it needs to READ or WRITE a 128-byte sector.

READ: Read Sector

READ reads in a 128-byte sector provided that there have been previous BIOS function calls to

SELDSK—"select" the disk

SETDMA—set the DMA address

SETTRK—set the track number

SETSEC—set the sector number.

Because of the blocking/deblocking code in the BIOS, there are frequent occasions when the requested sector will already be in the host buffer (HSTBUF), so that a physical disk read is not required. All that is then required is for the BIOS to move the appropriate 128 bytes from the HSTBUF into the buffer pointed at by the DMA address.

Only during the READ function will the BIOS normally communicate with the physical disk drive, selecting it and seeking to read the requested track and sector. During this process, the READ function must also handle any hardware errors that occur, trying an operation again if a "soft," or recoverable, error occurs.

The READ function must return with the A register set to 00H if the read operation is completed successfully. If the READ function returns with the A register set to 01H, the BDOS will display an error message of the form

```
BDOS Err on X: Bad Sector
```

Under these circumstances, you have only two choices. You can enter a CARRIAGE RETURN, ignore the fact that there was an error, and attempt to make sense of the data in the DMA buffer. Or you can type a CONTROL-C to abort the operation, perform a warm boot, and return control to the CCP.

As you can see, CP/M's error handling is not particularly helpful, so most BIOS writers add more sophisticated error recovery right in the disk driver. This can include some interaction with the console so that a more determined effort can be made to correct errors or, if nothing else, give you more information as to what has gone wrong. Such error handling is discussed in Chapter 9.

If you are working with a hard disk system, the BIOS driver must also handle the management of bad sectors. You cannot simply replace a hard disk drive if one or two sectors become unreadable. This bad sector management normally requires that a directory of "spare" sectors be put on the hard disk before it is used to store data. Then, when a sector is found to be bad, one of the spare sectors is substituted in its place. This is also discussed in Chapter 9.

WRITE: Write Sector

WRITE is similar to READ but with the obvious difference that data is transferred from the DMA buffer to the specified 128-byte sector. Like READ, this function requires that the following function calls have already been made:

SELDSK—"select" the disk

SETDMA—set the DMA address

SETTRK—set the track number

SETSEC—set the sector number.

Again, it is only in the WRITE routine that the driver will start to talk directly to the physical hardware, selecting the disk unit, track, and sector, and transferring the data to the disk.

With the blocking/deblocking code, the BDOS optimizes the number of disk writes that are needed by indicating in register C the type of disk write that is to be performed:

0 = normal sector write

1 = write to file directory sector

2 = write to sector of previously unused allocation block.

Type 0 occurs whenever the BDOS is writing to a data sector in an already used allocation block. Under these circumstances, the disk driver must preread the appropriate host sector because there may be previously stored information on it.

Type 1 occurs whenever the BDOS is writing to a file directory sector—in this case, the BIOS must not defer writing the sector to the disk, as the information is too valuable to hold in memory until the HSTBUF is full. The longer the information resides in the HSTBUF, the greater the chance of a power failure or glitch, making file data already physically written to the disk inaccessible because the file directory is out of date.

Type 2 occurs whenever the BDOS needs to write to the first sector of a previously unused allocation block. Unused, in this context, includes an allocation block that has become available as a result of a file being erased. In this case, there is no need for the disk driver to preread an entire host-sized sector into the HSTBUF, as there is no data of value in the physical sector.

As with the READ routine, the WRITE function returns with A set to 00H if the operation has been completed successfully. If the WRITE function returns with A set to 01H, then the BDOS will display the *same* message as for READ:

```
BDOS Err on X: Bad Sector
```

You can see now why most BIOS writers add extensive error-recovery and user-interaction routines to their disk drivers.

For hard disk systems, some disk drivers are written so that they automatically "spare out" a failing sector, writing the data to one of the spare sectors on the disk.

LISTST: List Status

As you can tell from its position in the list of BIOS functions, the LISTST function was a latecomer. It was added when CP/M was upgraded from version 1.4 to version 2.0.

This function returns the current status of the list device, using the IOBYTE if necessary to select the correct physical device. It sets the A register to 0FFH if the list device can accept another character for output or to 00H if it is not ready.

Digital Research's documentation states that this function is used by the DESPOOL utility program (which allows you to print a file "simultaneously" with other operations) to improve console response during its operation, and that it is acceptable for the routine always to return 00H if you choose not to implement it fully.

Unfortunately, this statement is wrong. Many other programs use the LISTST function to "poll" the list device to make sure it is ready, and if it fails to come ready after a predetermined time, to output a message to the console indicating that the printer is not ready. If you ever make a call to the BDOS list output functions, Write Printer Byte and Print String (codes 5 and 9), and the printer is not ready, then CP/M will wait forever—and your program will have lost control so it cannot even detect that the problem has occurred. If LISTST always returns a 00H, then the printer will always appear not to be ready. Not only does this make nonsense out of the LISTST function, but it also causes a stream of false "Printer not Ready" error messages to appear on the console.

SECTRAN: Sector Translate

SECTRAN, given a logical sector number, locates the correct physical sector number in the sector translate table for the previously selected (via SELDSK) logical disk drive.

Note that both logical and physical sector numbers are 128-byte sectors, so if you are working with a hard disk system, it is not too efficient to impose a sector interlace at the 128-byte sector level. It is better to impose the sector interlace right inside the hard disk driver, if at all; in general, hard disks spin so rapidly that CP/M simply cannot take advantage of sector interlace.

The BDOS hands over the logical sector number in the BC register pair, with the address of the sector translate table in the DE register pair. SECTRAN must return the physical sector number in HL.

If SECTRAN is to be a null routine, it must move the contents of BC to HL and return.

Calling the BIOS Functions Directly

As a general rule, you should not make direct calls to the BIOS. To do so makes your programs less transportable from one CP/M system to the next. It precludes being able to run these programs under MP/M, which has a different form of BIOS called an extended I/O system, or XIOS.

There are one or two problems, however, that can only be solved by making direct BIOS calls. These occur in utility programs that, for example, need to make direct access to the CP/M file directory, or need to access some "private" jump instructions which have been added to the standard BIOS jump vector.

If you really do need direct access to the BIOS, Figure 6-2 shows an example subroutine that does this. It requires that the A register contain a BIOS function code indicating the offset in the jump vector of the jump instruction to which control is to be passed.

```
                ;       Equates for use with BIOS subroutine
                ;
0003 =          WBOOT   EQU     03H     ;Warm boot
0006 =          CONST   EQU     06H     ;Console status
0009 =          CONIN   EQU     09H     ;Console input
000C =          CONOUT  EQU     0CH     ;Console output
000F =          LIST    EQU     0FH     ;Output to list device
0012 =          PUNCH   EQU     12H     ;Output to punch device
0015 =          READER  EQU     15H     ;Input from reader
0018 =          HOME    EQU     18H     ;Home selected disk to track 0
001B =          SELDSK  EQU     1BH     ;Select disk
001E =          SETTRK  EQU     1EH     ;Set track
0021 =          SETSEC  EQU     21H     ;Set sector
0024 =          SETDMA  EQU     24H     ;Set DMA address
0027 =          READ    EQU     27H     ;Read 128-byte sector
002A =          WRITE   EQU     2AH     ;Write 128-byte sector
002D =          LISTST  EQU     2DH     ;Return list status
0030 =          SECTRAN EQU     30H     ;Sector translate
                ;
                                        ;Add further "private" BIOS codes here
                ;
                ;       BIOS
                ;       This subroutine transfers control to the appropriate
                ;       entry in the BIOS Jump Vector, based on a code number
                ;       handed to it in the L register.
                ;
                ;       Entry parameters
                ;
                ;       L = Code number (which is in fact the page-relative
                ;               address of the correct JMP instruction within
                ;               the jump vector)
                ;       All other registers are preserved and handed over to
                ;               the BIOS routine intact.
                ;
                ;       Exit parameters
                ;
```

Figure 6-2. BIOS equates

```
                        ;       This routine does not CALL the BIOS routine, therefore
                        ;       when the BIOS routine RETurns, it will do so directly
                        ;       to this routine's caller.
                        ;
                        ;       Calling sequence
                        ;
                        ;               MVI     L,Code$Number
                        ;               CALL    BIOS
                        ;
                        BIOS:
0000 F5                         PUSH    PSW     ;Save user's A register
0001 3A0200                     LDA     0002H   ;Get BIOS JMP vector page from
                                                ;  warm boot JMP
0004 67                         MOV     H,A     ;HL -> BIOS JMP vector entry
0005 F1                         POP     PSW     ;Recover user's A register
0006 E9                         PCHL            ;Transfer control into the BIOS routine
```

Figure 6-2. BIOS equates (continued)

Line Numbers	Functional Component or Routine
0072-0116	BIOS Jump Vector
0120-0270	Initialization Code
0275-0286	Display Message
0289-0310	Enter CP/M
0333-0364	CONST - Console Status
0369-0393	CONIN - Console Input
0397-0410	CONOUT - Console Output
0414-0451	LISTST - List Status
0456-0471	LIST - List Output
0476-0492	PUNCH - Punch Output
0496-0511	READER - Reader Input
0516-0536	IOBYTE Driver Select
0540-0584	Device Control Tables
0589-0744	Low-level Drivers for Console, List,etc.
0769-0824	Disk Parameter Header Tables
0831-0878	Disk Parameter Blocks
0881-0907	Other Disk data areas
0910-0955	SELDSK - Select Disk
0958-0964	SETTRK - Set Track
0967-0973	SETSEC - Set Sector
0978-0984	SETDMA - Set DMA Address
0987-1025	Sector Skew Tables
1028-1037	SECTRAN - Logical to Physical Sector translation
1041-1056	HOME - Home to Track 0
1059-1154	Deblocking Algorithm data areas
1157-1183	READ - Read 128-byte sector
1185-1204	WRITE - Write 128-byte sector
1206-1378	Deblocking Algorithm
1381-1432	Buffer Move
1435-1478	Deblocking subroutines
1481-1590	8" Floppy Physical Read/Write
1595-1681	5 1/4" Floppy Physical Read/Write
1685-1764	WBOOT - Warm Boot

Figure 6-3. Functional Index to Figure 6-4

Example BIOS

The remainder of this chapter is devoted to an example BIOS listing. This actual working BIOS shows the overall structure and interface to the individual BIOS subroutines.

Unlike most BIOS's, this one has been written specifically to be understood easily. The variable names are uncharacteristically long and descriptive, and each block of code has commentary to put it into context.

Each source line has been sequentially numbered (an infrequently used option that Digital Research's Assembler, ASM, permits). Figure 6-3 contains a functional index to the BIOS as a whole so that you can find particular functions in the listing in Figure 6-4 by line number.

```
0001 <-- Line Number ;  Figure 6-4.
0002                 ;
0003                 ;******************************************************
0004                 ;*                                                    *
0005                 ;*            Simple BIOS Listing                     *
0006                 ;*                                                    *
0007                 ;******************************************************
0008                 ;
0009                 ;
0010  3030 =         VERSION        EQU     '00'    ;Equates used in the sign on message
0011  3730 =         MONTH          EQU     '07'
0012  3531 =         DAY            EQU     '15'
0013  3238 =         YEAR           EQU     '82'
0014                 ;
0015                 ;*****************************************************************************
0016                 ;*                                                                           *
0017                 ;* This BIOS is for a computer system with the following                     *
0018                 ;* hardware configuration :                                                  *
0019                 ;*                                                                           *
0020                 ;*           - 8080 CPU                                                      *
0021                 ;*           - 64KBytes of RAM                                               *
0022                 ;*           - CRT/keyboard controller that transfers data                   *
0023                 ;*             as though it were a serial port (but requires                 *
0024                 ;*             no baud rate generator or USART programming)                  *
0025                 ;*           - A serial port, used for both list and "reader"/               *
0026                 ;*             "punch" devices. The serial port chip is an                   *
0027                 ;*             Intel 8251A with an 8253 baud rate generator.                 *
0028                 ;*           - Two 5 1/4" mini-floppy, double-sided, double-                 *
0029                 ;*             density drives. These drives use 512-byte sectors.            *
0030                 ;*             These are used as logical disks A: and B:.                    *
0031                 ;*           - Two 8" standard diskette drives (128-byte sectors).           *
0032                 ;*             These are used as logical disks C: and D:.                    *
0033                 ;*                                                                           *
0034                 ;*             Two intelligent disk controllers are used, one for            *
0035                 ;*             each diskette type. These controllers access memory           *
0036                 ;*             directly, both to read the details of the                     *
0037                 ;*             operations they are to perform and also to read               *
0038                 ;*             and write data from and to the diskettes.                     *
0039                 ;*                                                                           *
0040                 ;*                                                                           *
0041                 ;*****************************************************************************
0042
0043                 ;
0044                 ;  Equates for defining memory size and the base address and
0045                 ;  length of the system components.
```

Figure 6-4. Simple BIOS listing

```
                    ;
0040 =              Memory$Size         EQU     64      ;Number of Kbytes of RAM
                    ;
                    ;  The BIOS Length must be determined by inspection.
                    ;  Comment out the ORG BIOS$Entry line below by changing the first
                    ;  character to a semicolon. (This will make the Assembler start
                    ;  the BIOS at location 0.) Then assemble the BIOS and round up to
                    ;  the nearest 100H the address displayed on the console at the end
                    ;  of the assembly.
                    ;
0900 =              BIOS$Length         EQU     0900H
                    ;
0800 =              CCP$Length EQU      0800H   ;Constant
0E00 =              BDOS$Length         EQU     0E00H   ;Constant
                    ;
0008 =              Overall$Length      EQU     ((CCP$Length + BDOS$Length + BIOS$Length) / 1024) + 1
                    ;
E000 =              CCP$Entry  EQU      (Memory$Size - Overall$Length) * 1024
E806 =              BDOS$Entry EQU      CCP$Entry + CCP$Length + 6
F600 =              BIOS$Entry EQU      CCP$Entry + CCP$Length + BDOS$Length
                    ;
                    ;
                    ;

F600                   ORG     BIOS$Entry      ;Assemble code at BIOS address
                    ;
                    ;  BIOS jump vector
                    ;  Control will be transferred to the appropriate entry point
                    ;  from the CCP or the BDOS, both of which compute the relative
                    ;  address of the BIOS jump vector in order to locate it.
                    ;  Transient programs can also make direct BIOS calls transferring
                    ;  control to location xx00H, where xx is the value in location
                    ;  0002H.
                    ;
F600 C3F9F6            JMP     BOOT    ;Cold boot -- entered from CP/M bootstrap loader
                    Warm$Boot$Entry:   ;  Labelled so that the initialization code can
                                       ;  put the warm boot entry address down in location
                                       ;  0001H and 0002H of the base page
F603 C329FE            JMP     WBOOT   ;Warm boot -- entered by jumping to location 0000H.
                                       ;  Reloads the CCP which could have been
                                       ;  overwritten by previous program in transient
                                       ;  program area
F606 C362F8            JMP     CONST   ;Console status -- returns A = 0FFH if there is a
                                       ;  console keyboard character waiting
F609 C378F8            JMP     CONIN   ;Console input -- returns the next console keyboard
                                       ;  character in A
F60C C386F8            JMP     CONOUT  ;Console output -- outputs the character in C to
                                       ;  the console device
F60F C3ACF8            JMP     LIST    ;List output -- outputs the character in C to the
                                       ;  list device
F612 C3BCF8            JMP     PUNCH   ;Punch output -- outputs the character in C to the
                                       ;  logical punch device
F615 C3CDF8            JMP     READER  ;Reader input -- returns the next input character from
                                       ;  the logical reader device in A
F618 C3D3FB            JMP     HOME    ;Homes the currently selected disk to track 0
F61B C32BFB            JMP     SELDSK  ;Selects the disk drive specified in register C and
                                       ;  returns the address of the disk parameter header
F61E C358FB            JMP     SETTRK  ;Sets the track for the next read or write operation
                                       ;  from the BC register pair
F621 C35EFB            JMP     SETSEC  ;Sets the sector for the next read or write operation
                                       ;  from the A register
F624 C365FB            JMP     SETDMA  ;Sets the direct memory address (disk read/write)
                                       ;  address for the next read or write operation
                                       ;  from the DE register pair
F627 C3FBFB            JMP     READ    ;Reads the previously specified track and sector from
                                       ;  the selected disk into the DMA address
F62A C315FC            JMP     WRITE   ;Writes the previously specified track and sector onto
                                       ;  the selected disk from the DMA address
F62D C394F8            JMP     LISTST  ;Returns A = 0FFH if the list device can accept
                                       ;  another output character
F630 C3CDFB            JMP     SECTRAN ;Translates a logical sector into a physical one
                    ;
                    ;
                    ;
                    ;  The cold boot initialization code is only needed once.
```

Figure 6-4. (Continued)

```
                ;  It can be overwritten once it has been executed.
                ;  Therefore, it is "hidden" inside the main disk buffer.
                ;  When control is transferred to the BOOT entry point, this
                ;  code will be executed, only being overwritten by data from
                ;  the disk once the initialization procedure is complete.
                ;
                ;  To hide code in the buffer, the buffer is first declared
                ;  normally. Then the value of the location counter following
                ;  the buffer is noted. Then, using an ORG (ORiGin) statement, the
                ;  location counter is "wound back" to the start of the buffer
                ;  again and the initialization code written normally.
                ;  At the end of this code, another ORG statement is used to
                ;  set the location counter back as it was after the buffer had
                ;  been declared.
                ;
                ;
0200 =          Physical$Sector$Size      EQU     512     ;This is the actual sector size
                                                          ;for the 5 1/4" mini-floppy diskettes.
                                                          ;The 8" diskettes use 128-byte sectors.
                                                          ;Declare the physical disk buffer for the
                                                          ;5 1/4" diskettes
F633            Disk$buffer:       DS     Physical$Sector$Size
                ;
                                                          ;Save the location counter
F833 =          After$Disk$Buffer  EQU    $               ;$ = Current value of location counter
                ;
F633                               ORG    Disk$Buffer     ;Wind the location counter back
                ;
                Initialize$Stream: ;This stream of data is used by the
                                   ;initialize subroutine. It has the following
                                   ;format:
                                   ;
                                   ;      DB      Port number to be initialized
                                   ;      DB      Number of bytes to be output
                                   ;      DB      xx,xx,xx,xx data to be output
                                   ;      :
                                   ;      :
                                   ;      DB      Port number of 00H terminator
                                   ;
                                   ;Note : On this machine, the console port does
                                   ;       not need to be initialized. This has
                                   ;       already been done by the PROM bootstrap code.
                                   ;
                                          ;Initialize the 8251A USART used for
                                          ;  the list and communications devices.
F633 ED            DB     Communication$Status$Port    ;Port number
F634 06            DB     6                            ;Number of bytes
F635 00            DB     0               ;Get chip ready to be programmed by
F636 00            DB     0               ;  sending dummy data out to it
F637 00            DB     0
F638 42            DB     0100$0010B      ;Reset and raise data terminal ready
F639 6E            DB     01$10$11$10B    ;1 stop bit, no parity, 8 bits per character
                                          ;  baud rate divide factor of 16.
F63A 25            DB     0010$0101B      ;Raise request to send, and enable
                                          ;  transmit and receive.
                ;
                                          ;Initialize the 8253 programmable interval
                                          ;  timer used to generate the baud rate for
                                          ;  the 8251A USART
F63B DF            DB     Communication$Baud$Mode      ;Port number
F63C 01            DB     1                            ;Number of bytes
F63D B6            DB     10$11$011$0B    ;Select counter 2, load LS byte first,
                                          ;  Mode 3 (for baud rates), binary count.
                ;
F63E DE            DB     Communication$Baud$Rate      ;Port number
F63F 02            DB     2                            ;Number of bytes
F640 3800          DW     0038H           ;1200 baud (based on 16X divide-down selected
                                          ;  in the 8251A USART)
                ;
F642 00            DB     0               ;Port number of 0 terminates
                ;
                ;
                ;  Equates for the sign-on message
                ;
000D =          CR EQU     0DH            ;Carriage return
```

Figure 6-4. (Continued)

```
000A =          LF EQU      0AH             ;Line feed
                ;
                Signon$Message:             ;Main sign-on message
F643 43502F4D20    DB       'CP/M 2.2.'
F64C 3030          DW       VERSION         ;Current version number
F64E 20            DB       ' '
F64F 3037          DW       MONTH           ;Current date
F651 2F            DB       '/'
F652 3135          DW       DAY
F654 2F            DB       '/'
F655 3832          DW       YEAR
F657 0D0A0A        DB       CR,LF,LF
F65A 53696D706C    DB       'Simple BIOS',CR,LF,LF
F668 4469736B20    DB       'Disk configuration :',CR,LF,LF
F67F 2020202020    DB       '     A: 0.35 Mbyte 5" Floppy',CR,LF
F69D 2020202020    DB       '     B: 0.35 Mbyte 5" Floppy',CR,LF,LF
F6BC 2020202020    DB       '     C: 0.24 Mbyte 8" Floppy',CR,LF
F6DA 2020202020    DB       '     D: 0.24 Mbyte 8" Floppy',CR,LF
                ;
F6F8 00            DB       0
                ;
0004 =          Default$Disk        EQU     0004H   ;Default disk in base page
                ;
                BOOT:       ;Entered directly from the BIOS JMP vector.
                            ;Control will be transferred here by the CP/M
                            ;  bootstrap loader.
                            ;The initialization state of the computer system
                            ;  will be determined by the
                            ;  PROM bootstrap and the CP/M loader setup.
                            ;
                                            ;Initialize system.
                                            ;This routine uses the Initialize$Stream
                                            ;  declared above.
F6F9 F3            DI                       ;Disable interrupts to prevent any
                                            ;  side effects during initialization.
F6FA 2133F6        LXI      H,Initialize$Stream     ;HL -> Data stream
                ;
                Initialize$Loop:
F6FD 7E            MOV      A,M             ;Get port number
F6FE B7            ORA      A               ;If 00H, then initialization complete
F6FF CA13F7        JZ       Initialize$Complete
F702 320AF7        STA      Initialize$Port ;Set up OUT instruction
F705 23            INX      H               ;HL -> Count of number of bytes to output
F706 4E            MOV      C,M             ;Get byte count
                ;
                Initialize$Next$Byte:
F707 23            INX      H               ;HL -> Next data byte
F708 7E            MOV      A,M             ;Get next data byte
F709 D3            DB       OUT             ;Output to correct port
                Initialize$Port:
F70A 00            DB       0               ;<- Set above
F70B 0D            DCR      C               ;Count down
F70C C207F7        JNZ      Initialize$Next$Byte    ;Go back if more bytes
F70F 23            INX      H               ;HL -> Next port number
F710 C3FDF6        JMP      Initialize$Loop ;Go back for next port initialization
                ;
                Initialize$Complete:
                ;

F713 3E01          MVI      A,00$00$00$01B          ;Set IOBYTE to indicate terminal
F715 320300        STA      IOBYTE                  ;  is to act as console

F718 2143F6        LXI      H,Signon$Message        ;Display sign-on message on console
F71B CD33F8        CALL     Display$Message
                ;

F71E AF            XRA      A               ;Set default disk drive to A:
F71F 320400        STA      Default$Disk
F722 FB            EI                       ;Interrupts can now be enabled
                ;
F723 C340F8        JMP      Enter$CPM       ;Complete initialization and enter
                                            ;  CP/M by going to the Console Command
                                            ;  Processor.
                ;
                ;  End of cold boot initialization code
                ;
```

Figure 6-4. (Continued)

```
F833                ORG     After$Disk$Buffer       ;Reset location counter
                ;
                ;
                Display$Message:   ;Displays the specified message on the console.
                                   ;On entry, HL points to a stream of bytes to be
                                   ;  output. A 00H-byte terminates the message.
F833 7E             MOV     A,M             ;Get next message byte
F834 B7             ORA     A               ;Check if terminator
F835 C8             RZ                      ;Yes, return to caller
F836 4F             MOV     C,A             ;Prepare for output
F837 E5             PUSH    H               ;Save message pointer
F838 CD86F8         CALL    CONOUT          ;Go to main console output routine
F83B E1             POP     H               ;Recover message pointer
F83C 23             INX     H               ;Move to next byte of message
F83D C333F8         JMP     Display$Message ;Loop until complete message output
                ;
                ;
                Enter$CPM: ;This routine is entered either from the cold or warm
                           ;  boot code. It sets up the JMP instructions in the
                           ;  base page, and also sets the high-level disk driver's
                           ;  input/output address (also known as the DMA address).
                ;
F840 3EC3           MVI     A,JMP           ;Get machine code for JMP
F842 320000         STA     0000H           ;Set up JMP at location 0000H
F845 320500         STA     0005H           ;  and at location 0005H
                ;
F848 2103F6         LXI     H,Warm$Boot$Entry       ;Get BIOS vector address
F84B 220100         SHLD    0001H           ;Put address at location 0001H

F84E 2106E8         LXI     H,BDOS$Entry    ;Get BDOS entry point address
F851 220600         SHLD    6               ;Put address at location 0005H
                ;
F854 018000         LXI     B,80H           ;Set disk I/O address to default
F857 CD65FB         CALL    SETDMA          ;Use normal BIOS routine
                ;
F85A FB             EI                      ;Ensure interrupts are enabled
F85B 3A0400         LDA     Default$Disk    ;Transfer current default disk to
F85E 4F             MOV     C,A             ;  Console Command Processor
F85F C300E0         JMP     CCP$Entry       ;Transfer to CCP
                ;
                ;
                ;   Serial input/output drivers
                ;
                ;   These drivers all look at the IOBYTE at location
                ;   0003H, which will have been set by the cold boot routine.
                ;   The IOBYTE can be modified by the STAT utility, by
                ;   BDOS calls, or by a program that puts a value directly
                ;   into location 0003H.
                ;
                ;   All of the routines make use of a subroutine, Select$Routine,
                ;   that takes the least significant two bits of the A register
                ;   and uses them to transfer control to one of the routines whose
                ;   address immediately follows the call to Select$Routine.
                ;   A second entry point, Select$Routine$21, uses bits
                ;   2 and 1 to do the same job -- this saves some space
                ;   by avoiding an unnecessary instruction.
                ;
0003 =          IOBYTE      EQU     0003H   ;I/O redirection byte
                ;
                ;
                ;
                CONST:                 ;Get console status
                                       ;Entered directly from the BIOS JMP vector
                                       ;  and returns a parameter that reflects whether
                                       ;  there is incoming data from the console.
                                       ;
                                       ;A = 00H (zero flag set) if no data
                                       ;A = 0FFH (zero flag clear) if data
                                       ;
                                       ;CONST will be called by programs that
                                       ;  make periodic checks to see if the computer
                                       ;  operator has pressed any keys -- for example,
                                       ;  to interrupt an executing program.
                                       ;
F862 CD6AF8         CALL    Get$Console$Status      ;Return A = zero or nonzero
                                       ;According to status, then convert
```

Figure 6-4. (Continued)

```
                                                ;  to return parameter convention.
F865 B7          ORA     A                      ;Set flags to reflect status
F866 C8          RZ                             ;If 0, no incoming data
F867 3EFF        MVI     A,0FFH                 ;Otherwise return A = 0FFH to
F869 C9          RET                            ;  indicate incoming data
                 ;
             Get$Console$Status:
F86A 3A0300      LDA     IOBYTE                 ;Get I/O redirection byte
                                                ;Console is selected according to
                                                ;  bits 1,0 of IOBYTE
F86D CDDCF8      CALL    Select$Routine         ;Select appropriate routine
                                                ;These routines return to the caller
                                                ;  of Get$Console$Status.
F870 F6F8        DW      Teletype$In$Status        ;00 <- IOBYTE bits 1,0
F872 FCF8        DW      Terminal$In$Status        ;01
F874 02F9        DW      Communication$In$Status   ;10
F876 08F9        DW      Dummy$In$Status           ;11
             ;
             ;
             ;
             ;
             CONIN:                 ;Get console input character
                                    ;Entered directly from the BIOS JMP vector;
                                    ;  returns the next data character from the
                                    ;  Console in the A register. The most significant
                                    ;  bit of the data character will be 0, except
                                    ;  when "reader" (communication port) input has
                                    ;  been selected. In this case, the full eight bits
                                    ;  of data are returned to permit binary data to be
                                    ;  received.
                                    ;
                                    ;Normally, this routine will be called after
                                    ;  a call to CONST has indicated that a data character
                                    ;  is ready, but whenever the CCP or the BDOS can
                                    ;  proceed no further until console input occurs,
                                    ;  then CONIN will be called without a preceding
                                    ;  CONST call.
                                    ;
F878 3A0300      LDA     IOBYTE                 ;Get I/O redirection byte
F87B CDDCF8      CALL    Select$Routine         ;Select correct CONIN routine
                                                   ;These routines return directly
                                                   ; to CONIN's caller.
F87E 20F9        DW      Teletype$Input            ;00 <- IOBYTE bits 1,0
F880 26F9        DW      Terminal$Input            ;01
F882 2FF9        DW      Communication$Input       ;10
F884 35F9        DW      Dummy$Input               ;11
             ;
             ;
             ;
             CONOUT:                ;Console output
                                    ;Entered directly from BIOS JMP vector;
                                    ; outputs the data character in the C register
                                    ; to the appropriate device according to bits
                                    ; 1,0 of IOBYTE
                                    ;
F886 3A0300      LDA     IOBYTE                 ;Get I/O redirection byte
F889 CDDCF8      CALL    Select$Routine         ;Select correct CONOUT routine
                                                   ;These routines return directly
                                                   ; to CONOUT's caller.
F88C 38F9        DW      Teletype$Output           ;00 <- IOBYTE bits 1,0
F88E 3EF9        DW      Terminal$Output           ;01
F890 44F9        DW      Communication$Output      ;10
F892 4AF9        DW      Dummy$Output              ;11
             ;
             ;
             ;
             LISTST:                ;List device (output) status
                                    ;Entered directly from the BIOS JMP vector;
                                    ; returns in A list device status that
                                    ; indicates whether the list device can accept
                                    ; another output character. The IOBYTE's bits
                                    ; 7,6 determine the physical device used.
                                    ;
                                    ;A = 00H (zero flag set): cannot accept data
                                    ;A = 0FFH (zero flag clear): can accept data
                                    ;
```

Figure 6-4. (Continued)

```
                                          ;Digital Research's documentation indicates
                                          ; that you can always return with A = 00H
                                          ; ("Cannot accept data") if you do not wish to
                                          ; implement the LISTST routine. This is NOT TRUE.
                                          ;If you do not wish to implement the LISTST routine
                                          ; always return with A = 0FFH ("Can accept data").
                                          ;The LIST driver will then take care of things rather
                                          ; than potentially hanging the system.
                                          ;
F894 CD9CF8         CALL    Get$List$Status ;Return A = zero or nonzero
                                          ; according to status, then convert
                                          ; to return parameter convention
F897 B7             ORA     A             ;Set flags to reflect status
F898 C8             RZ                    ;If 0, cannot accept data for output
F899 3EFF           MVI     A,0FFH        ;Otherwise return A = 0FFH to
F89B C9             RET                   ; indicate can accept data for output
                ;
                Get$List$Status:
F89C 3A0300         LDA     IOBYTE        ;Get I/O redirection byte
F89F 07             RLC                   ;Move bits 7,6 to 1,0
F8A0 07             RLC
F8A1 CDDCF8         CALL    Select$Routine ;Select appropriate routine
                                          ;These routines return directly
                                          ; to Get$List$Status's caller.
F8A4 0BF9           DW      Teletype$Out$Status        ;00 <- IOBYTE bits 1,0
F8A6 11F9           DW      Terminal$Out$Status        ;01
F8A8 17F9           DW      Communication$Out$Status   ;10
F8AA 1DF9           DW      Dummy$Out$Status           ;11

                ;
                ;
                ;
                LIST:                     ;List output
                                          ;Entered directly from BIOS JMP vector;
                                          ; outputs the data character in the C register
                                          ; to the appropriate device according to bits
                                          ; 7,6 of IOBYTE
                                          ;
F8AC 3A0300         LDA     IOBYTE        ;Get I/O redirection byte
F8AF 07             RLC                   ;Move bits 7,6 to 1,0
F8B0 07             RLC
F8B1 CDDCF8         CALL    Select$Routine        ;Select correct LIST routine
                                                  ;These routines return directly
                                                  ; to LIST's caller.
F8B4 38F9           DW      Teletype$Output       ;00 <- IOBYTE bits 1,0
F8B6 3EF9           DW      Terminal$Output       ;01
F8B8 44F9           DW      Communication$Output  ;10
F8BA 4AF9           DW      Dummy$Output          ;11

                ;
                ;
                ;
                PUNCH:                    ;Punch output
                                          ;Entered directly from BIOS JMP vector;
                                          ; outputs the data character in the C register
                                          ; to the appropriate device according to bits
                                          ; 5,4 of IOBYTE
                                          ;
F8BC 3A0300         LDA     IOBYTE        ;Get I/O redirection byte
F8BF 0F             RRC                   ;Move bits 5,4 to 2,1
F8C0 0F             RRC
F8C1 0F             RRC
F8C2 CDDDF8         CALL    Select$Routine$21     ;Select correct PUNCH routine
                                                  ;These routines return directly
                                                  ; to PUNCH's caller.
F8C5 38F9           DW      Teletype$Output       ;00 <- IOBYTE bits 1,0
F8C7 4AF9           DW      Dummy$Output          ;01
F8C9 44F9           DW      Communication$Output  ;10
F8CB 3EF9           DW      Terminal$Output       ;11
                ;
                ;
                ;
                READER:                   ;Reader input
                                          ;Entered directly from BIOS JMP vector;
                                          ; inputs the next data character from the
                                          ; reader device into the A register
```

Figure 6-4. (Continued)

```
                                        ;The appropriate device is selected according
                                        ; to bits 3,2 of IOBYTE.
                                        ;
F8CD 3A0300         LDA     IOBYTE                  ;Get I/O redirection byte
F8D0 0F             RRC                             ;Move bits 3,2 to 2,1
F8D1 CDDDF8         CALL    Select$Routine$21       ;Select correct READER routine
                                                    ;These routines return directly
                                                    ; to READER's caller.
F8D4 38F9           DW      Teletype$Output         ;00 <- IOBYTE bits 1,0
F8D6 4AF9           DW      Dummy$Output            ;01
F8D8 44F9           DW      Communication$Output    ;10
F8DA 3EF9           DW      Terminal$Output         ;11

                ;
                ;
                ;
                Select$Routine:             ;Transfers control to a specified address
                                            ; following its calling address according to
                                            ; the value of bits 1,0 in A.
F8DC 07             RLC                     ;Shift select values into bits 2,1
                                            ; in order to do word arithmetic
                ;
                Select$Routine$21:          ;Entry point to select routine selection bits
                                            ; are already in bits 2,1
F8DD E606           ANI     0000$0110B      ;Isolate just bits 2,1
F8DF E3             XTHL                    ;HL -> first word of addresses after
                                            ; CALL instruction
F8E0 5F             MOV     E,A             ;Add on selection value to address table
F8E1 1600           MVI     D,0             ; base
F8E3 19             DAD     D               ;HL -> selected routine address
                                            ;Get routine address into HL
F8E4 7E             MOV     A,M             ;LS byte
F8E5 23             INX     H               ;HL -> MS byte
F8E6 66             MOV     H,M             ;MS byte
F8E7 6F             MOV     L,A             ;HL -> routine
F8E8 E3             XTHL                    ;Top of stack -> routine
F8E9 C9             RET                     ;Transfer to selected routine
                ;
                ;
                ;
                ;   Input/Output Equates
                ;
00ED =          Teletype$Status$Port                EQU     0EDH
00EC =          Teletype$Data$Port          EQU     0ECH
0001 =          Teletype$Output$Ready               EQU     0000$0001B      ;Status mask
0002 =          Teletype$Input$Ready                EQU     0000$0010B      ;Status mask
                ;
0001 =          Terminal$Status$Port                EQU     01H
0002 =          Terminal$Data$Port          EQU     02H
0001 =          Terminal$Output$Ready               EQU     0000$0001B      ;Status mask
0002 =          Terminal$Input$Ready                EQU     0000$0010B      ;Status mask
                ;
00ED =          Communication$Status$Port   EQU     0EDH
00EC =          Communication$Data$Port             EQU     0ECH
0001 =          Communication$Output$Ready  EQU     0000$0001B      ;Status mask
0002 =          Communication$Input$Ready   EQU     0000$0010B      ;Status mask
                ;
00DF =          Communication$Baud$Mode             EQU     0DFH            ;Mode Select
00DE =          Communication$Baud$Rate             EQU     0DEH            ;Rate Select
                ;
                ;
                ;   Serial device control tables
                ;
                ;   In order to reduce the amount of executable code,
                ;   the same low-level driver code is used for all serial ports.
                ;   On entry to the low-level driver, HL points to the
                ;   appropriate control table.
                ;
                Teletype$Table:
F8EA ED             DB      Teletype$Status$Port
F8EB EC             DB      Teletype$Data$Port
F8EC 01             DB      Teletype$Output$Ready
F8ED 02             DB      Teletype$Input$Ready
                ;
                Terminal$Table:
F8EE 01             DB      Terminal$Status$Port
```

Figure 6-4. (Continued)

```
F8EF 02            DB      Terminal$Data$Port
F8F0 01            DB      Terminal$Output$Ready
F8F1 02            DB      Terminal$Input$Ready
                ;
                Communication$Table:
F8F2 ED            DB      Communication$Status$Port
F8F3 EC            DB      Communication$Data$Port
F8F4 01            DB      Communication$Output$Ready
F8F5 02            DB      Communication$Input$Ready
                ;
                ;
                ;
                ;
                ;  The following routines are "called" by Select$Routine
                ;  to perform the low-level input/output
                ;
                Teletype$In$Status:
F8F6 21EAF8        LXI     H,Teletype$Table          ;HL -> control table
F8F9 C34BF9        JMP     Input$Status              ;Note use of JMP. Input$Status
                                                     ; will execute the RETurn.
                ;
                Terminal$In$Status:
F8FC 21EEF8        LXI     H,Terminal$Table          ;HL -> control table
F8FF C34BF9        JMP     Input$Status              ;Note use of JMP. Input$Status
                                                     ; will execute the RETurn.
                ;
                Communication$In$Status:
F902 21F2F8        LXI     H,Communication$Table     ;HL -> control table
F905 C34BF9        JMP     Input$Status              ;Note use of JMP. Input$Status
                                                     ; will execute the RETurn.
                ;
                Dummy$In$Status:                     ;Dummy status, always returns
F908 3EFF          MVI     A,0FFH                    ; indicating incoming data is ready
F90A C9            RET
                ;
                ;
                Teletype$Out$Status:
F90B 21EAF8        LXI     H,Teletype$Table          ;HL -> control table
F90E C356F9        JMP     Output$Status             ;Note use of JMP. Output$Status
                                                     ; will execute the RETurn.
                ;
                Terminal$Out$Status:
F911 21EEF8        LXI     H,Terminal$Table          ;HL -> control table
F914 C356F9        JMP     Output$Status             ;Note use of JMP. Output$Status
                                                     ; will execute the RETurn.
                ;
                Communication$Out$Status:
F917 21F2F8        LXI     H,Communication$Table     ;HL -> control table
F91A C356F9        JMP     Output$Status             ;Note use of JMP. Output$Status
                                                     ; will execute the RETurn.
                ;
                Dummy$Out$Status:                    ;Dummy status, always returns
F91D 3EFF          MVI     A,0FFH                    ; indicating ready for output
F91F C9            RET
                ;
                ;
                Teletype$Input:
F920 21EAF8        LXI     H,Teletype$Table          ;HL -> control table
F923 C360F9        JMP     Input$Data                ;Note use of JMP. Input$Data
                                                     ; will execute the RETurn.
                ;
                Terminal$Input:
F926 21EEF8        LXI     H,Terminal$Table          ;HL -> control table
                                                     ; will execute the RETurn.
F929 CD60F9        CALL    Input$Data                ;** Special case **
                                                     ;Input$Data will return here
F92C E67F          ANI     7FH                       ; so that parity bit can be set 0
F92E C9            RET
                ;
                Communication$Input:
F92F 21F2F8        LXI     H,Communication$Table     ;HL -> control table
F932 C360F9        JMP     Input$Data                ;Note use of JMP. Input$Data
                                                     ; will execute the RETurn.
                ;
                Dummy$Input:                         ;Dummy input, always returns
F935 3E1A          MVI     A,1AH                     ; indicating CP/M end of file
```

Figure 6-4. (Continued)

```
F937 C9           RET
                ;
                ;
                ;
                ;
                Teletype$Output:
F938 21EAF8       LXI     H,Teletype$Table          ;HL -> control table
F93B C370F9       JMP     Output$Data               ;Note use of JMP. Output$Data
                                                    ; will execute the RETurn.
                ;
                Terminal$Output:
F93E 21EEF8       LXI     H,Terminal$Table          ;HL -> control table
                                                    ; will execute the RETurn.
F941 C370F9       JMP     Output$Data               ;Note use of JMP. Output$Data
                                                    ; will execute the RETurn.
                ;
                Communication$Output:
F944 21F2F8       LXI     H,Communication$Table     ;HL -> control table
F947 C370F9       JMP     Output$Data               ;Note use of JMP. Output$Data
                                                    ; will execute the RETurn.
                ;
                Dummy$Output:                       ;Dummy output, always discards
F94A C9           RET                               ; the output character
                ;
                ;
                ;
                ;
                ;  These are the general purpose low-level drivers.
                ;  On entry, HL points to the appropriate control table.
                ;  For output, the C register contains the data to be output.
                ;
                Input$Status:                ;Return with A = 00H if no incoming data,
                                             ; otherwise A = nonzero.
F94B 7E           MOV     A,M                ;Get status port
F94C 3250F9       STA     Input$Status$Port         ;*** Self-modifying code ***
F94F DB           DB      IN                 ;Input to A from correct status port
                ;
                Input$Status$Port:
F950 00           DB      00                 ;<- Set above
F951 23           INX     H                  ;Move HL to point to input data mask
F952 23           INX     H
F953 23           INX     H
F954 A6           ANA     M                  ;Mask with input status
F955 C9           RET
                ;
                ;
                Output$Status:               ;Return with A = 00H if not ready for output
                                             ; otherwise A = nonzero.
F956 7E           MOV     A,M                ;Get status port
F957 325BF9       STA     Output$Status$Port        ;*** Self-modifying code ***
F95A DB           DB      IN                 ;Input to A from correct status port
                ;
                Output$Status$Port:
F95B 00           DB      00                 ;<- Set above
F95C 23           INX     H                  ;Move HL to point to output data mask
F95D 23           INX     H
F95E A6           ANA     M                  ;Mask with output status
F95F C9           RET
                ;
                ;
                Input$Data:                  ;Return with next data character in A.
                                             ;Wait for status routine to indicate
                                             ; incoming data.
F960 E5           PUSH    H                  ;Save control table pointer
F961 CD4BF9       CALL    Input$Status       ;Get input status in zero flag
F964 E1           POP     H                  ;Recover control table pointer
F965 CA60F9       JZ      Input$Data         ;Wait until incoming data
F968 23           INX     H                  ;HL -> data port
F969 7E           MOV     A,M                ;Get data port
F96A 326EF9       STA     Input$Data$Port    ;*** Self-modifying code ***
F96D DB           DB      IN                 ;Input to A from correct data port
                ;
                Input$Data$Port:
F96E 00           DB      0                  ;<- Set above
F96F C9           RET
                ;
```

Figure 6-4. (Continued)

```
                          ;
                          Output$Data:                ;Output the data character in the C register.
                                                      ;Wait for status routine to indicate device
                                                      ; ready to accept another character
F970 E5                       PUSH    H               ;Save control table pointer
F971 CD56F9                   CALL    Output$Status   ;Get output status in zero flag
F974 E1                       POP     H               ;Recover control table pointer
F975 CA70F9                   JZ      Output$Data     ;Wait until ready for output
F978 23                       INX     H               ;HL -> output port
F979 7E                       MOV     A,M             ;Get output port
F97A 327FF9                   STA     Output$Data$Port        ;*** Self-modifying code ***
F97D 79                       MOV     A,C             ;Get data character to be output
F97E D3                       DB      OUT             ;Output data to correct port
                          ;
                          Output$Data$Port:
F97F 00                       DB      0               ;<- Set above
F980 C9                       RET
                          ;
                          ;
                          ;   High level diskette drivers
                          ;
                          ;   These drivers perform the following functions:
                          ;
                          ;   SELDSK  Select a specified disk and return the address of
                          ;           the appropriate disk parameter header
                          ;   SETTRK  Set the track number for the next read or write
                          ;   SETSEC  Set the sector number for the next read or write
                          ;   SETDMA  Set the DMA (read/write) address for the next read or write.
                          ;   SECTRAN Translate a logical sector number into a physical
                          ;   HOME    Set the track to 0 so that the next read or write will
                          ;           be on Track 0
                          ;
                          ;   In addition, the high-level drivers are responsible for making
                          ;   the 5 1/4" floppy diskettes that use a 512-byte sector appear
                          ;   to CP/M as though they used a 128-byte sector. They do this
                          ;   by using what is called blocking/deblocking code,
                          ;   described in more detail later in this listing,
                          ;   just prior to the code itself.
                          ;
                          ;
                          ;
                          ;   Disk parameter tables
                          ;
                          ;   As discussed in Chapter 3, these describe the physical
                          ;   characteristics of the disk drives. In this example BIOS,
                          ;   there are two types of disk drives; standard single-sided,
                          ;   single-density 8", and double-sided, double-density 5 1/4"
                          ;   diskettes.
                          ;
                          ;   The standard 8" diskettes do not need to use the blocking/
                          ;   deblocking code, but the 5 1/4" drives do. Therefore an additional
                          ;   byte has been prefixed to the disk parameter block to
                          ;   tell the disk drivers each logical disk's physical
                          ;   diskette type, and whether or not it needs deblocking.
                          ;
                          ;
                          ;   Disk definition tables
                          ;
                          ;   These consist of disk parameter headers, with one entry
                          ;   per logical disk driver, and disk parameter blocks, with
                          ;   either one parameter block per logical disk or the same
                          ;   parameter block for several logical disks.
                          ;
                          ;
                          Disk$Parameter$Headers:                     ;Described in Chapter 3
                          ;
                                              ;Logical Disk A: (5 1/4" Diskette)
F981 6BFB                     DW      Floppy$5$Skewtable              ;5 1/4" skew table
F983 0000000000               DW      0,0,0                           ;Reserved for CP/M
F989 C1F9                     DW      Directory$Buffer
F98B 42FA                     DW      Floppy$5$Parameter$Block
F98D 61FA                     DW      Disk$A$Workarea
F98F C1FA                     DW      Disk$A$Allocation$Vector
                          ;
                                              ;Logical Disk B: (5 1/4" Diskette)
F991 6BFB                     DW      Floppy$5$Skewtable              ;Shares same skew table as A:
```

Figure 6-4. (Continued)

```
F993 0000000000   DW      0,0,0                          ;Reserved for CP/M
F999 C1F9         DW      Directory$Buffer               ;Share same buffer as A:
F99B 42FA         DW      Floppy$5$Parameter$Block       ;Same DPB as A:
F99D 81FA         DW      Disk$B$Workarea                ;Private work area
F99F D7FA         DW      Disk$B$Allocation$Vector       ;Private allocation vector
                ;
                          ;Logical Disk C: (8" Floppy)
F9A1 B3FB         DW      Floppy$8$Skewtable             ;8" skew table
F9A3 0000000000   DW      0,0,0                          ;Reserved for CP/M
F9A9 C1F9         DW      Directory$Buffer               ;Share same buffer as A:
F9AB 52FA         DW      Floppy$8$Parameter$Block
F9AD A1FA         DW      Disk$C$Workarea                ;Private work area
F9AF EDFA         DW      Disk$C$Allocation$Vector       ;Private allocation vector
                ;
                          ;Logical Disk D: (8" Floppy)
F9B1 6BFB         DW      Floppy$5$Skewtable             ;Shares same skew table as A:
F9B3 0000000000   DW      0,0,0                          ;Reserved for CP/M
F9B9 C1F9         DW      Directory$Buffer               ;Share same buffer as A:
F9BB 52FA         DW      Floppy$8$Parameter$Block       ;Same DPB as C:
F9BD B1FA         DW      Disk$D$Workarea                ;Private work area
F9BF 0CFB         DW      Disk$D$Allocation$Vector       ;Private allocation vector

                ;
                ;
F9C1            Directory$Buffer:  DS     128
                ;
                ;
                ;
                ;  Disk Types
                ;
0001 =          Floppy$5   EQU    1         ;5 1/4" mini floppy
0002 =          Floppy$8   EQU    2         ;8" floppy (SS SD)
                ;
                ;  Blocking/deblocking indicator
                ;
0080 =          Need$Deblocking    EQU    1000$0000B     ;Sector size > 128 bytes
                ;
                ;
                ;  Disk parameter blocks
                ;
                ;  5 1/4" mini floppy
                ;
                                            ;Extra byte prefixed to indicate
                                            ; disk type and blocking required
FA41 81            DB     Floppy$5 + Need$Deblocking
                Floppy$5$Parameter$Block:
FA42 4800          DW     72                ;128-byte sectors per track
FA44 04            DB     4                 ;Block shift
FA45 0F            DB     15                ;Block mask
FA46 01            DB     1                 ;Extent mask
FA47 AE00          DW     174               ;Maximum allocation block number
FA49 7F00          DW     127               ;Number of directory entries - 1
FA4B C0            DB     1100$0000B        ;Bit map for reserving 1 alloc. block
FA4C 00            DB     0000$0000B        ;  for file directory
FA4D 2000          DW     32                ;Disk changed work area size
FA4F 0100          DW     1                 ;Number of tracks before directory
                ;
                ;
                ;  Standard 8" Floppy
                                            ;Extra byte prefixed to DPB for
                                            ; this version of the BIOS
FA51 02            DB     Floppy$8          ;Indicates disk type and the fact
                                            ; that no deblocking is required
                Floppy$8$Parameter$Block:
FA52 1A00          DW     26                ;Sectors per track
FA54 03            DB     3                 ;Block shift
FA55 07            DB     7                 ;Block mask
FA56 00            DB     0                 ;Extent mask
FA57 F200          DW     242               ;Maximum allocation block number
FA59 3F00          DW     63                ;Number of directory entries - 1
FA5B C0            DB     1100$0000B        ;Bit map for reserving 2 alloc. blocks
FA5C 00            DB     0000$0000B        ;  for file directory
FA5D 1000          DW     16                ;Disk changed work area size
FA5F 0200          DW     2                 ;Number of tracks before directory
                ;
                ;
```

Figure 6-4. (Continued)

```
                        ;   Disk work areas
                        ;
                        ;   These are used by the BDOS to detect any unexpected
                        ;   change of diskettes. The BDOS will automatically set
                        ;   such a changed diskette to read-only status.
                        ;
FA61                    Disk$A$Workarea:    DS      32      ; A:
FA81                    Disk$B$Workarea:    DS      32      ; B:
FAA1                    Disk$C$Workarea:    DS      16      ; C:
FAB1                    Disk$D$Workarea:    DS      16      ; D:
                        ;
                        ;
                        ;   Disk allocation vectors
                        ;
                        ;   These are used by the BDOS to maintain a bit map of
                        ;   which allocation blocks are used and which are free.
                        ;   One byte is used for eight allocation blocks, hence the
                        ;   expression of the form (allocation blocks/8)+1.
                        ;
FAC1                    Disk$A$Allocation$Vector    DS      (174/8)+1       ; A:
FAD7                    Disk$B$Allocation$Vector    DS      (174/8)+1       ; B:
                        ;
FAED                    Disk$C$Allocation$Vector    DS      (242/8)+1       ; C:
FB0C                    Disk$D$Allocation$Vector    DS      (242/8)+1       ; D:
                        ;
                        ;
0004 =                  Number$of$Logical$Disks             EQU     4
                        ;
                        ;
                        SELDSK:                         ;Select disk in C
                                                        ;C = 0 for drive A, 1 for B, etc.
                                                        ;Return the address of the appropriate
                                                        ; disk parameter header in HL, or 0000H
                                                        ; if the selected disk does not exist.
                                                        ;
FB2B 210000                LXI     H,0                  ;Assume an error
FB2E 79                    MOV     A,C                  ;Check if requested disk valid
FB2F FE04                  CPI     Number$of$Logical$Disks
FB31 D0                    RNC                          ;Return if > maximum number of disks
                        ;
FB32 32EAFB                STA     Selected$Disk        ;Save selected disk number
                                                        ;Set up to return DPH address
FB35 6F                    MOV     L,A                  ;Make disk into word value
FB36 2600                  MVI     H,0
                                                        ;Compute offset down disk parameter
                                                        ; header table by multiplying by
                                                        ; parameter header length (16 bytes)
FB38 29                    DAD     H                    ; *2
FB39 29                    DAD     H                    ; *4
FB3A 29                    DAD     H                    ; *8
FB3B 29                    DAD     H                    ; *16
FB3C 1181F9                LXI     D,Disk$Parameter$Headers        ;Get base address
FB3F 19                    DAD     D                    ;DE -> Appropriate DPH
FB40 E5                    PUSH    H                    ;Save DPH address
                        ;
                                                        ;Access disk parameter block
                                                        ; to extract special prefix byte that
                                                        ; identifies disk type and whether
                                                        ; deblocking is required
                                                        ;
FB41 110A00                LXI     D,10                 ;Get DPB pointer offset in DPH
FB44 19                    DAD     D                    ;DE -> DPB address in DPH
FB45 5E                    MOV     E,M                  ;Get DPB address in DE
FB46 23                    INX     H
FB47 56                    MOV     D,M
FB48 EB                    XCHG                         ;DE -> DPB
FB49 2B                    DCX     H                    ;DE -> prefix byte
FB4A 7E                    MOV     A,M                  ;Get prefix byte
FB4B E60F                  ANI     0FH                  ;Isolate disk type
FB4D 32FAFB                STA     Disk$Type            ;Save for use in low-level driver
FB50 7E                    MOV     A,M                  ;Get another copy of prefix byte
FB51 E680                  ANI     Need$Deblocking              ;Isolate deblocking flag
FB53 32F9FB                STA     Deblocking$Required          ;Save for use in low-level driver
FB56 E1                    POP     H                    ;Recover DPH pointer
FB57 C9                    RET
                        ;
```

Figure 6-4. (Continued)

```
                        ;
                        ;  Set logical track for next read or write
                        ;
                        SETTRK:
FB58 60                    MOV     H,B                ;Selected track in BC on entry
FB59 69                    MOV     L,C
FB5A 22EBFB                SHLD    Selected$Track     ;Save for low-level driver
FB5D C9                    RET
                        ;
                        ;
                        ;  Set logical sector for next read or write
                        ;
                        ;
                        SETSEC:                       ;Logical sector in C on entry
FB5E 79                    MOV     A,C
FB5F 32EDFB                STA     Selected$Sector    ;Save for low-level driver
FB62 C9                    RET
                        ;
                        ;
                        ;  Set disk DMA (input/output) address for next read or write
                        ;
FB63 0000               DMA$Address:       DW      0        ;DMA address
                        ;
                        SETDMA:                       ;Address in BC on entry
FB65 69                    MOV     L,C                ;Move to HL to save
FB66 60                    MOV     H,B
FB67 2263FB                SHLD    DMA$Address        ;Save for low-level driver
FB6A C9                    RET
                        ;
                        ;
                        ;  Translate logical sector number to physical
                        ;
                        ;  Sector translation tables
                        ;  These tables are indexed using the logical sector number,
                        ;  and contain the corresponding physical sector number.
                        ;
                        Floppy$5$Skewtable:           ;Each physical sector contains four
                                                      ; 128-byte sectors.
                           ;       Physical 128b      Logical 128b      Physical 512-byte
FB6B 00010203              DB      00,01,02,03        ;00,01,02,03            0  )
FB6F 10111213              DB      16,17,18,19        ;04,05,06,07            4  )
FB73 20212223              DB      32,33,34,35        ;08,09,10,11            8  )
FB77 0C0D0E0F              DB      12,13,14,15        ;12,13,14,15            3  ) Head
FB7B 1C1D1E1F              DB      28,29,30,31        ;16,17,18,19            7  )  0
FB7F 08090A0B              DB      08,09,10,11        ;20,21,22,23            2  )
FB83 18191A1B              DB      24,25,26,27        ;24,25,26,27            6  )
FB87 04050607              DB      04,05,06,07        ;28,29,30,31            1  )
FB8B 14151617              DB      20,21,22,23        ;32,33,34,35            5  )
                        ;
FB8F 24252627              DB      36,37,38,39        ;36,37,38,39            0  ]
FB93 34353637              DB      52,53,54,55        ;40,41,42,43            4  ]
FB97 44454647              DB      68,69,70,71        ;44,45,46,47            8  ]
FB9B 30313233              DB      48,49,50,51        ;48,49,50,51            3  ] Head
FB9F 40414243              DB      64,65,66,67        ;52,53,54,55            7  ]  1
FBA3 2C2D2E2F              DB      44,45,46,47        ;56,57,58,59            2  ]
FBA7 3C3D3E3F              DB      60,61,62,63        ;60,61,62,63            6  ]
FBAB 28292A2B              DB      40,41,42,43        ;64,65,66,67            1  ]
FBAF 38393A3B              DB      56,57,58,59        ;68,69,70,71            5  ]
                        ;
                        ;
                        Floppy$8$Skewtable:           ;Standard 8" Driver
                           ;       01,02,03,04,05,06,07,08,09,10    Logical sectors
FBB3 01070D1319            DB      01,07,13,19,25,05,11,17,23,03   ;Physical sectors
                           ;
                           ;       11,12,13,14,15,16,17,18,19,20    Logical sectors
FBBD 090F150208            DB      09,15,21,02,08,14,20,26,06,12   ;Physical sectors
                           ;
                           ;       21,22,23,24,25,26        Logical sectors
FBC7 1218040A10            DB      18,24,04,10,16,22       ;Physical sectors

                        ;
                        SECTRAN:                      ;Translate logical sector into physical
                                                      ;On entry, BC = logical sector number
                                                      ;         DE -> appropriate skew table
                                                      ;
                                                      ;on exit, HL = physical sector number
```

Figure 6-4. (Continued)

```
FBCD EB            XCHG                        ;HL -> skew table base
FBCE 09            DAD     B                   ;Add on logical sector number
FBCF 6E            MOV     L,M                 ;Get physical sector number
FBD0 2600          MVI     H,0                 ;Make into a 16-bit value
FBD2 C9            RET
                 ;
                 ;
                 ;
                 HOME:                         ;Home the selected logical disk to track 0.
                                               ;Before doing this, a check must be made to see
                                               ; if the physical disk buffer has information
                                               ; that must be written out. This is indicated by
                                               ; a flag, Must$Write$Buffer, set in the
                                               ; deblocking code.
                                               ;
FBD3 3AE9FB        LDA     Must$Write$Buffer          ;Check if physical buffer must
FBD6 B7            ORA     A                          ;  be written out to disk
FBD7 C2DDFB        JNZ     HOME$No$Write
FBDA 32E8FB        STA     Data$In$Disk$Buffer        ;No, so indicate that buffer
                                                      ;  is now unoccupied.
                 HOME$No$Write:
FBDD 0E00          MVI     C,0                        ;Set to track 0 (logically --
FBDF CD58FB        CALL    SETTRK                     ;  no actual disk operation occurs)
FBE2 C9            RET

                 ;
                 ;  Data written to or read from the mini-floppy drive is transferred
                 ;  via a physical buffer that is actually 512 bytes long (it was
                 ;  declared at the front of the BIOS and holds the "one-time"
                 ;  initialization code used for the cold boot procedure).
                 ;
                 ;  The blocking/deblocking code attempts to minimize the amount
                 ;  of actual disk I/O by storing the disk, track, and physical sector
                 ;  currently residing in the Physical Buffer. If a read request is for
                 ;  a 128-byte CP/M "sector" that already is in the physical buffer,
                 ;  then no disk access occurs.
                 ;
                 ;
0800 =           Allocation$Block$Size       EQU     2048
0012 =           Physical$Sec$Per$Track      EQU     18
0004 =           CPM$Sec$Per$Physical        EQU     Physical$Sector$Size/128
0048 =           CPM$Sec$Per$Track           EQU     CPM$Sec$Per$Physical*Physical$Sec$Per$Track
0003 =           Sector$Mask                 EQU     CPM$Sec$Per$Physical-1
0002 =           Sector$Bit$Shift            EQU     2         ;LOG2(CPM$Sec$Per$Physical)
                 ;
                                    ;These are the values handed over by the BDOS
                                    ;  when it calls the WRITE operation.
                                    ;The allocated/unallocated indicates whether the
                                    ;  BDOS is set to write to an unallocated allocation
                                    ;  block (it only indicates this for the first
                                    ;  128-byte sector write) or to an allocation block
                                    ;  that has already been allocated to a file.
                                    ;The BDOS also indicates if it is set to write to
                                    ;  the file directory.
                                    ;
0000 =           Write$Allocated             EQU     0
0001 =           Write$Directory             EQU     1
0002 =           Write$Unallocated           EQU     2
                 ;
FBE3 00          Write$Type:                 DB      0         ;Contains the type of write
                                                               ;  indicated by the BDOS.
                 ;
                 ;
                 In$Buffer$Dk$Trk$Sec:                         ;Variables for physical sector
                                                               ; currently in Disk$Buffer in memory
FBE4 00          In$Buffer$Disk:             DB      0         ; These are moved and compared
FBE5 0000        In$Buffer$Track:            DW      0         ; as a group, so do not alter
FBE7 00          In$Buffer$Sector:           DB      0         ; these lines.
                 ;
FBE8 00          Data$In$Disk$Buffer:        DB      0         ;When nonzero, the disk buffer has
                                                               ;  data from the disk in it.
FBE9 00          Must$Write$Buffer:          DB      0         ;Nonzero when data has been
                                                               ;  written into Disk$Buffer but
                                                               ;  not yet written out to disk
                 ;
                 Selected$Dk$Trk$Sec:        ;Variables for selected disk, track, and sector
```

Figure 6-4. (Continued)

```
                                                ; (Selected by SELDSK, SETTRK,and SETSEC)
FBEA 00         Selected$Disk:                  DB      0       ; These are moved and
FBEB 0000       Selected$Track:                 DW      0       ; compared as a group so
FBED 00         Selected$Sector:                DB      0       ; do not alter order.

FBEE 00         Selected$Physical$Sector:       DB      0       ;Selected physical sector derived
                                                                ;  from selected (CP/M) sector by
                                                                ;  shifting it right the number of
                                                                ;  of bits specified by
                                                                ;  Sector$Bit$Shift
                ;
FBEF 00         Selected$Disk$Type:             DB      0       ;Set by SELDSK to indicate either
                                                                ;  8" or 5 1/4" floppy
FBF0 00         Selected$Disk$Deblock:          DB      0       ;Set by SELDSK to indicate whether
                                                                ;  deblocking is required.


                Unallocated$Dk$Trk$Sec:                 ;Parameters for writing to a previously
                                                                ;  unallocated allocation block.
FBF1 00         Unallocated$Disk:               DB      0       ; These are moved and compared
FBF2 0000       Unallocated$Track:              DW      0       ; as a group so do not alter
FBF4 00         Unallocated$Sector:             DB      0       ; these lines.

FBF5 00         Unallocated$Record$Count:       DB      0       ;Number of unallocated "records"
                                                                ; in current previously unallocated
                                                                ; allocation block.

FBF6 00         Disk$Error$Flag:                DB      0       ;Nonzero to indicate an error
                                                                ;  that could not be recovered
                                                                ;  by the disk drivers. BDOS will
                                                                ;  output a "bad sector" message.
                ;
                ;Flags used inside the deblocking code

FBF7 00         Must$Preread$Sector:            DB      0       ;Nonzero if a physical sector must
                                                                ;  be read into the disk buffer
                                                                ;  either before a write to an
                                                                ;  allocated block can occur, or
                                                                ;  for a normal CP/M 128-byte
                                                                ;  sector read
FBF8 00         Read$Operation:                 DB      0       ;Nonzero when a CP/M 128-byte
                                                                ;  sector is to be read
FBF9 00         Deblocking$Required:            DB      0       ;Nonzero when the selected disk
                                                                ;  needs deblocking (set in SELDSK)
FBFA 00         Disk$Type:                      DB      0       ;Indicates 8" or 5 1/4" floppy
                                                                ;  selected (set in SELDSK).
                ;
                ;
                ;   Read in the 128-byte CP/M sector specified by previous calls
                ;   to select disk and to set track and sector. The sector will be read
                ;   into the address specified in the previous call to set DMA address.
                ;
                ;   If reading from a disk drive using sectors larger than 128 bytes,
                ;   deblocking code will be used to "unpack" a 128-byte sector from
                ;   the physical sector.
                READ:
FBFB 3AF9FB        LDA      Deblocking$Required     ;Check if deblocking needed
FBFE B7            ORA      A                       ;(flag was set in SELDSK call)
FBFF CA52FD        JZ       Read$No$Deblock         ;No, use normal nondeblocked

                                ;The deblocking algorithm used is such
                                ;  that a read operation can be viewed
                                ;  up until the actual data transfer as
                                ;  though it was the first write to an
                                ;  unallocated allocation block.
FC02 AF            XRA      A                       ;Set the record count to 0
FC03 32F5FB        STA      Unallocated$Record$Count ;  for first "write"
FC06 3C            INR      A                       ;Indicate that it is really a read
FC07 32F8FB        STA      Read$Operation          ;  that is to be performed
FC0A 32F7FB        STA      Must$Preread$Sector     ;  and force a preread of the sector
                                                    ;  to get it into the disk buffer
FC0D 3E02          MVI      A,Write$Unallocated     ;Fake deblocking code into responding
FC0F 32E3FB        STA      Write$Type              ;  as if this is the first write to an
                                                    ;  unallocated allocation block.
FC12 C36EFC        JMP      Perform$Read$Write      ;Use common code to execute read
```

Figure 6-4. (Continued)

```
                  ;
                  ;  Write a 128-byte sector from the current DMA address to
                  ;  the previously selected disk, track, and sector.
                  ;
                  ;  On arrival here, the BDOS will have set register C to indicate
                  ;  whether this write operation is to an already allocated allocation
                  ;  block (which means a preread of the sector may be needed),
                  ;  to the directory (in which case the data will be written to the
                  ;  disk immediately), or to the first 128-byte sector of a previously
                  ;  unallocated allocation block (in which case no preread is required).
                  ;
                  ;  Only writes to the directory take place immediately. In all other
                  ;  cases, the data will be moved from the DMA address into the disk
                  ;  buffer, and only written out when circumstances force the
                  ;  transfer. The number of physical disk operations can therefore
                  ;  be reduced considerably.
                  ;
                  WRITE:
FC15 3AF9FB          LDA     Deblocking$Required     ;Check if deblocking is required
FC18 B7              ORA     A                       ;(flag set in SELDSK call)
FC19 CA4DFD          JZ      Write$No$Deblock

FC1C AF              XRA     A                       ;Indicate that a write operation
FC1D 32F8FB          STA     Read$Operation          ;  is required (i.e. NOT a read)
FC20 79              MOV     A,C                     ;Save the BDOS write type
FC21 32E3FB          STA     Write$Type
FC24 FE02            CPI     Write$Unallocated       ;Check if the first write to an
                                                     ;  unallocated allocation block
FC26 C237FC          JNZ     Check$Unallocated$Block ;No, check if in the middle of
                                                     ;  writing to an unallocated block
                                                     ;Yes, first write to unallocated
                                                     ;  allocation block -- initialize
                                                     ;  variables associated with
                                                     ;  unallocated writes.
FC29 3E10            MVI     A,Allocation$Block$Size/128     ;Get number of 128-byte
                                                             ;  sectors and
FC2B 32F5FB          STA     Unallocated$Record$Count        ;  set up a count.
                                                             ;
FC2E 21EAFB          LXI     H,Selected$Dk$Trk$Sec           ;Copy disk, track, and sector
FC31 11F1FB          LXI     D,Unallocated$Dk$Trk$Sec        ;  into unallocated variables
FC34 CD35FD          CALL    Move$Dk$Trk$Sec
                  ;
                  ;  Check if this is not the first write to an unallocated
                  ;  allocation block -- if it is, the unallocated record count
                  ;  has just been set to the number of 128-byte sectors in the
                  ;  allocation block.
                  ;
                  Check$Unallocated$Block:
FC37 3AF5FB          LDA     Unallocated$Record$Count
FC3A B7              ORA     A
FC3B CA66FC          JZ      Request$Preread         ;No, this is a write to an
                                                     ;  allocated block
                                                     ;Yes, this is a write to an
                                                     ;  unallocated block
FC3E 3D              DCR     A                       ;Count down on number of 128-byte sectors
                                                     ;  left unwritten to in allocation block
FC3F 32F5FB          STA     Unallocated$Record$Count        ;  and store back new value.

FC42 21EAFB          LXI     H,Selected$Dk$Trk$Sec   ;Check if the selected disk, track,
FC45 11F1FB          LXI     D,Unallocated$Dk$Trk$Sec;  and sector are the same as for
FC48 CD29FD          CALL    Compare$Dk$Trk$Sec      ;  those in the unallocated block.
FC4B C266FC          JNZ     Request$Preread         ;No, a preread is required
                                                     ;Yes, no preread is needed.
                                                     ;Now is a convenient time to
                                                     ;  update the current sector and see
                                                     ;  if the track also needs updating.
                  ;
                                                     ;By design, Compare$Dk$Trk$Sec
                                                     ; returns with
                                                     ; DE -> Unallocated$Sector
FC4E EB              XCHG                            ; HL -> Unallocated$Sector
FC4F 34              INR     M                       ;Update Unallocated$Sector
FC50 7E              MOV     A,M                     ;Check if sector now > maximum
FC51 FE48            CPI     CPM$Sec$Per$Track       ; on a track
FC53 DA5FFC          JC      No$Track$Change         ;No (A < M)
                                                     ;Yes,
```

Figure 6-4. (Continued)

```
FC56 3600          MVI     M,0                     ;Reset sector to 0
FC58 2AF2FB        LHLD    Unallocated$Track       ;Increase track by 1
FC5B 23            INX     H
FC5C 22F2FB        SHLD    Unallocated$Track
               ;
               No$Track$Change:
                                                   ;Indicate to later code that
                                                   ;  no preread is needed.
FC5F AF            XRA     A
FC60 32F7FB        STA     Must$Preread$Sector     ;Must$Preread$Sector=0
FC63 C36EFC        JMP     Perform$Read$Write
               ;
               Request$Preread:
                                                   ;Indicate that this is not a write
FC66 AF            XRA     A
FC67 32F5FB        STA     Unallocated$Record$Count      ;  into an unallocated block.
FC6A 3C            INR     A
FC6B 32F7FB        STA     Must$Preread$Sector     ;Indicate that a preread of the
                                                   ;  physical sector is required.
               ;
               ;
               Perform$Read$Write:                 ;Common code to execute both reads and
                                                   ;  writes of 128-byte sectors.
FC6E AF            XRA     A                       ;Assume that no disk errors will
FC6F 32F6FB        STA     Disk$Error$Flag         ;  occur

FC72 3AEDFB        LDA     Selected$Sector         ;Convert selected 128-byte sector
FC75 1F            RAR                             ;  into physical sector by dividing by 4
FC76 1F            RAR
FC77 E63F          ANI     3FH                     ;Remove any unwanted bits
FC79 32EEFB        STA     Selected$Physical$Sector
                                                   ;
FC7C 21E8FB        LXI     H,Data$In$Disk$Buffer   ;Check if disk buffer already has
FC7F 7E            MOV     A,M                     ;  data in it.
FC80 3601          MVI     M,1                     ;(Unconditionally indicate that
                                                   ; the buffer now has data in it)
FC82 B7            ORA     A                       ;Did it indeed have data in it?
FC83 CAA3FC        JZ      Read$Sector$into$Buffer ;No, proceed to read a physical
                                                   ;  sector into the buffer.
                                                   ;
                                           ;The buffer does have a physical sector
                                           ;  in it.
                                           ;  Note: The disk, track, and PHYSICAL
                                           ;  sector in the buffer need to be
                                           ;  checked, hence the use of the
                                           ;  Compare$Dk$Trk subroutine.
                                           ;
FC86 11E4FB        LXI     D,In$Buffer$Dk$Trk$Sec  ;Check if sector in buffer is the
FC89 21EAFB        LXI     H,Selected$Dk$Trk$Sec   ;  same as that selected earlier
FC8C CD24FD        CALL    Compare$Dk$Trk          ;Compare ONLY disk and track
FC8F C29CFC        JNZ     Sector$Not$In$Buffer    ;No, it must be read in

FC92 3AE7FB        LDA     In$Buffer$Sector        ;Get physical sector in buffer
FC95 21EEFB        LXI     H,Selected$Physical$Sector
FC98 BE            CMP     M                       ;Check if correct physical sector
FC99 CAB1FC        JZ      Sector$In$Buffer        ;Yes, it is already in memory
               ;
               Sector$Not$In$Buffer:
                                                   ;No, it will have to be read in
                                                   ;  over current contents of buffer
FC9C 3AE9FB        LDA     Must$Write$Buffer       ;Check if buffer has data in that
FC9F B7            ORA     A                       ;  must be written out first
FCA0 C495FD        CNZ     Write$Physical          ;Yes, write it out
               ;
               Read$Sector$into$Buffer:
FCA3 CD11FD        CALL    Set$In$Buffer$Dk$Trk$Sec       ;Set in buffer variables from
                                                   ; selected disk, track, and sector
                                                   ;  to reflect which sector is in the
                                                   ;  buffer now
FCA6 3AF7FB        LDA     Must$Preread$Sector     ;In practice, the sector need only
FCA9 B7            ORA     A                       ;  be physically read in if a preread
                                                   ;  is required
FCAA C49AFD        CNZ     Read$Physical           ;Yes, preread the sector
FCAD AF            XRA     A                       ;Reset the flag to reflect buffer
FCAE 32E9FB        STA     Must$Write$Buffer       ;  contents.
               ;
               Sector$In$Buffer:           ;Selected sector on correct track and
```

Figure 6-4. (Continued)

```
                                                ;  disk is already in the buffer.
                                                ;Convert the selected CP/M (128-byte)
                                                ;  sector into a relative address down
                                                ;  the buffer.
FCB1 3AEDFB         LDA     Selected$Sector     ;Get selected sector number
FCB4 E603           ANI     Sector$Mask         ;Mask off only the least significant bits
FCB6 6F             MOV     L,A                 ;Multiply by 128 by shifting 16-bit value
FCB7 2600           MVI     H,0                 ;  left 7 bits
FCB9 29             DAD     H                   ;* 2
FCBA 29             DAD     H                   ;* 4
FCBB 29             DAD     H                   ;* 8
FCBC 29             DAD     H                   ;* 16
FCBD 29             DAD     H                   ;* 32
FCBE 29             DAD     H                   ;* 64
FCBF 29             DAD     H                   ;* 128
               ;
FCC0 1133F6         LXI     D,Disk$Buffer       ;Get base address of disk buffer
FCC3 19             DAD     D                   ;Add on sector number * 128
                                                ;HL -> 128-byte sector number start
                                                ;  address in disk buffer
FCC4 EB             XCHG                        ;DE -> sector in disk buffer
FCC5 2A63FB         LHLD    DMA$Address         ;Get DMA address set in SETDMA call
FCC8 EB             XCHG                        ;Assume a read operation, so
                                                ;  DE -> DMA address
                                                ;  HL -> sector in disk buffer
FCC9 0E10           MVI     C,128/8             ;Because of the faster method used
                                                ;  to move data in and out of the
                                                ;  disk buffer, (eight bytes moved per
                                                ;  loop iteration) the count need only
                                                ;  be 1/8th of normal.
                                                ;At this point -
                                                ;       C = loop count
                                                ;       DE -> DMA address
                                                ;       HL -> sector in disk buffer
FCCB 3AF8FB         LDA     Read$Operation      ;Determine whether data is to be moved
FCCE B7             ORA     A                   ;  out of the buffer (read) or into the
FCCF C2D7FC         JNZ     Buffer$Move         ;  buffer (write)
                                                ;Writing into buffer
                                                        ;(A must be 0 get here)
FCD2 3C             INR     A                           ;Set flag to force a write
FCD3 32E9FB         STA     Must$Write$Buffer           ;  of the disk buffer later on.
FCD6 EB             XCHG                                ;Make DE -> sector in disk buffer
                                                        ;       HL -> DMA address
               ;
               ;
               Buffer$Move:                     ;The folowing move loop moves eight bytes
                                                ;  at a time from (HL) to (DE), C contains
                                                ;  the loop count.
FCD7 7E             MOV     A,M                 ;Get byte from source
FCD8 12             STAX    D                   ;Put into destination
FCD9 13             INX     D                   ;Update pointers
FCDA 23             INX     H
FCDB 7E             MOV     A,M                 ;Get byte from source
FCDC 12             STAX    D                   ;Put into destination
FCDD 13             INX     D                   ;Update pointers
FCDE 23             INX     H
FCDF 7E             MOV     A,M                 ;Get byte from source
FCE0 12             STAX    D                   ;Put into destination
FCE1 13             INX     D                   ;Update pointers
FCE2 23             INX     H
FCE3 7E             MOV     A,M                 ;Get byte from source
FCE4 12             STAX    D                   ;Put into destination
FCE5 13             INX     D                   ;Update pointers
FCE6 23             INX     H
FCE7 7E             MOV     A,M                 ;Get byte from source
FCE8 12             STAX    D                   ;Put into destination
FCE9 13             INX     D                   ;Update pointers
FCEA 23             INX     H
FCEB 7E             MOV     A,M                 ;Get byte from source
FCEC 12             STAX    D                   ;Put into destination
FCED 13             INX     D                   ;Update pointers
FCEE 23             INX     H
FCEF 7E             MOV     A,M                 ;Get byte from source
FCF0 12             STAX    D                   ;Put into destination
FCF1 13             INX     D                   ;Update pointers
```

Figure 6-4. (Continued)

```
FCF2 23             INX     H
FCF3 7E             MOV     A,M                 ;Get byte from source
FCF4 12             STAX    D                   ;Put into destination
FCF5 13             INX     D                   ;Update pointers
FCF6 23             INX     H

FCF7 0D             DCR     C                   ;Count down on loop counter
FCF8 C2D7FC         JNZ     Buffer$Move         ;Repeat until CP/M sector moved
                                                ;
FCFB 3AE3FB         LDA     Write$Type          ;If write to directory, write out
FCFE FE01           CPI     Write$Directory     ;  buffer immediately
FD00 3AF6FB         LDA     Disk$Error$Flag     ;Get error flag in case delayed write or read
FD03 C0             RNZ                         ;Return if delayed write or read
                                                ;
FD04 B7             ORA     A                   ;Check if any disk errors have occurred
FD05 C0             RNZ                         ;Yes, abandon attempt to write to directory
                                                ;
FD06 AF             XRA     A                   ;Clear flag that indicates buffer must be
FD07 32E9FB         STA     Must$Write$Buffer   ;  written out
FD0A CD95FD         CALL    Write$Physical      ;Write buffer out to physical sector
FD0D 3AF6FB         LDA     Disk$Error$Flag     ;Return error flag to caller
FD10 C9             RET
                ;
                ;
                Set$In$Buffer$Dk$Trk$Sec:           ;Indicate selected disk, track, and
                                                    ;  sector now residing in buffer
FD11 3AEAFB         LDA     Selected$Disk
FD14 32E4FB         STA     In$Buffer$Disk

FD17 2AEBFB         LHLD    Selected$Track
FD1A 22E5FB         SHLD    In$Buffer$Track

FD1D 3AEEFB         LDA     Selected$Physical$Sector
FD20 32E7FB         STA     In$Buffer$Sector

FD23 C9             RET
                ;
                Compare$Dk$Trk:                 ;Compares just the disk and track
                                                ;  pointed to by DE and HL
FD24 0E03           MVI     C,3                 ;Disk (1), track (2)
FD26 C32BFD         JMP     Compare$Dk$Trk$Sec$Loop ;Use common code

                Compare$Dk$Trk$Sec:             ;Compares the disk, track, and sector
                                                ;  variables pointed to by DE and HL
FD29 0E04           MVI     C,4                 ;Disk (1), track (2), and sector (1)
                Compare$Dk$Trk$Sec$Loop:
FD2B 1A             LDAX    D                   ;Get comparitor
FD2C BE             CMP     M                   ;Compare with comparand
FD2D C0             RNZ                         ;Abandon comparison if inequality found
FD2E 13             INX     D                   ;Update comparitor pointer
FD2F 23             INX     H                   ;Update comparand pointer
FD30 0D             DCR     C                   ;Count down on loop count
FD31 C8             RZ                          ;Return (with zero flag set)
FD32 C32BFD         JMP     Compare$Dk$Trk$Sec$Loop
                ;
                ;
                Move$Dk$Trk$Sec:                ;Moves the disk, track,and sector
                                                ;  variables pointed at by HL to
                                                ;  those pointed at by DE
FD35 0E04           MVI     C,4                 ;Disk (1), track (2), and sector (1)
                Move$Dk$Trk$Sec$Loop:
FD37 7E             MOV     A,M                 ;Get source byte
FD38 12             STAX    D                   ;Store in destination
FD39 13             INX     D                   ;Update pointers
FD3A 23             INX     H
FD3B 0D             DCR     C                   ;Count down on byte count
FD3C C8             RZ                          ;Return if all bytes moved
FD3D C337FD         JMP     Move$Dk$Trk$Sec$Loop
                ;
                ;
                ;
                ;   There are two "smart" disk controllers on this system, one
                ;   for the 8" floppy diskette drives, and one for the 5 1/4"
                ;   mini-diskette drives.
                ;
                ;   The controllers are "hard-wired" to monitor certain locations
```

Figure 6-4. (Continued)

```
                ;  in memory to detect when they are to perform some disk
                ;  operation. The 8" controller monitors location 0040H, and
                ;  the 5 1/4" controller monitors location 0045H. These are
                ;  called their disk control bytes. If the most significant
                ;  bit of a disk control byte is set, the controller will
                ;  look at the word following the respective control bytes.
                ;  This word must contain the address of a valid disk control
                ;  table that specifies the exact disk operation to be performed.
                ;
                ;  Once the operation has been completed, the controller resets
                ;  its disk control byte to 00H. This indicates completion
                ;  to the disk driver code.
                ;
                ;  The controller also sets a return code in a disk status block --
                ;  both controllers use the SAME location for this: 0043H.
                ;  If the first byte of this status block is less than 80H, then
                ;  a disk error has occurred. For this simple BIOS, no further details
                ;  of the status settings are relevant. Note that the disk controller
                ;  has built-in retry logic -- reads and writes are attempted ten
                ;  times before the controller returns an error.
                ;
                ;  The disk control table layout is shown below. Note that the
                ;  controllers have the capability for control tables to be
                ;  chained together so that a sequence of disk operations can
                ;  be initiated. In this BIOS this feature is not used. However,
                ;  the controller requires that the chain pointers in the
                ;  disk control tables be pointed back to the main control bytes
                ;  in order to indicate the end of the chain.
                ;
0040 =          Disk$Control$8          EQU     40H     ;8" control byte
0041 =          Command$Block$8         EQU     41H     ;Control table pointer
                ;
0043 =          Disk$Status$Block       EQU     43H     ;8" AND 5 1/4" status block
                ;
0045 =          Disk$Control$5          EQU     45H     ;5 1/4" control byte
0046 =          Command$Block$5         EQU     46H     ;Control table pointer
                ;
                ;
                ;  Floppy Disk Control Tables
                ;
FD40 00         Floppy$Command:         DB      0       ;Command
0001 =          Floppy$Read$Code        EQU     01H
0002 =          Floppy$Write$Code       EQU     02H
FD41 00         Floppy$Unit:            DB      0       ;Unit (drive) number = 0 or 1
FD42 00         Floppy$Head:            DB      0       ;Head number = 0 or 1
FD43 00         Floppy$Track:           DB      0       ;Track number
FD44 00         Floppy$Sector:          DB      0       ;Sector number
FD45 0000       Floppy$Byte$Count:      DW      0       ;Number of bytes to read/write
FD47 0000       Floppy$DMA$Address:     DW      0       ;Transfer address
FD49 0000       Floppy$Next$Status$Block: DW    0       ;Pointer to next status block
                                                        ;  if commands are chained.
FD4B 0000       Floppy$Next$Control$Location: DW 0      ;Pointer to next control byte
                                                        ;  if commands are chained.
                ;
                ;
                ;
                Write$No$Deblock:                       ;Write contents of disk buffer to
                                                        ;  correct sector.
FD4D 3E02          MVI     A,Floppy$Write$Code          ;Get write function code
FD4F C354FD        JMP     Common$No$Deblock            ;Go to common code
                Read$No$Deblock:                        ;Read previously selected sector
                                                        ;  into disk buffer.
FD52 3E01          MVI     A,Floppy$Read$Code           ;Get read function code
                Common$No$Deblock:
FD54 3240FD        STA     Floppy$Command  ;Set command function code
                                           ;Set up nondeblocked command table
FD57 218000        LXI     H,128           ;Bytes per sector
FD5A 2245FD        SHLD    Floppy$Byte$Count
FD5D AF            XRA     A               ;8" floppy only has head 0
FD5E 3242FD        STA     Floppy$Head
                                           ;
FD61 3AEAFB        LDA     Selected$Disk   ;8" Floppy controller only has information
                                           ;  on units 0 and 1 so Selected$Disk must
                                           ;  be converted
FD64 E601          ANI     01H             ;Turn into 0 or 1
FD66 3241FD        STA     Floppy$Unit     ;Set unit number
```

Figure 6-4. (Continued)

```
                                                  ;
FD69 3AEBFB         LDA     Selected$Track
FD6C 3243FD         STA     Floppy$Track          ;Set track number
                                                  ;
FD6F 3AEDFB         LDA     Selected$Sector
FD72 3244FD         STA     Floppy$Sector         ;Set sector number
                                                  ;
FD75 2A63FB         LHLD    DMA$Address           ;Transfer directly between DMA address
FD78 2247FD         SHLD    Floppy$DMA$Address          ;and 8" controller.
                                                  ;
                                                  ;The disk controller can accept chained
                                                  ;  disk control tables, but in this case,
                                                  ;  they are not used, so the "Next" pointers
                                                  ;  must be pointed back at the initial
                                                  ;  control bytes in the base page.
FD7B 214300         LXI     H,Disk$Status$Block             ;Point next status back at
FD7E 2249FD         SHLD    Floppy$Next$Status$Block        ;  main status block
                                                            ;
FD81 214000         LXI     H,Disk$Control$8                ;Point next control byte
FD84 224BFD         SHLD    Floppy$Next$Control$Location    ;  back at main control byte
                                                            ;
FD87 2140FD         LXI     H,Floppy$Command                ;Point controller at control table
FD8A 224100         SHLD    Command$Block$8
                                                            ;
FD8D 214000         LXI     H,Disk$Control$8                ;Activate controller to perform
FD90 3680           MVI     M,80H                           ;  operation.
FD92 C3F7FD         JMP     Wait$For$Disk$Complete

                ;
                ;
                Write$Physical:                     ;Write contents of disk buffer to
                                                    ;  correct sector.
FD95 3E02           MVI     A,Floppy$Write$Code     ;Get write function code
FD97 C39CFD         JMP     Common$Physical         ;Go to common code
                Read$Physical:                      ;Read previously selected sector
                                                    ;  into disk buffer.
FD9A 3E01           MVI     A,Floppy$Read$Code      ;Get read function code
                ;
                Common$Physical:
FD9C 3240FD         STA     Floppy$Command          ;Set command table

                ;
FD9F 3AFAFB         LDA     Disk$Type               ;Get disk type (set in SELDSK)
FDA2 FE01           CPI     Floppy$5                ;Confirm it is a 5 1/4" Floppy
FDA4 CAADFD         JZ      Correct$Disk$Type       ;Yes
FDA7 3E01           MVI     A,1                     ;No, indicate disk error
FDA9 32F6FB         STA     Disk$Error$Flag
FDAC C9             RET
                Correct$Disk$Type:                  ;Set up disk control table
                                                    ;
FDAD 3AE4FB         LDA     In$Buffer$Disk          ;Convert disk number to 0 or 1
FDB0 E601           ANI     1                       ;  for disk controller
FDB2 3241FD         STA     Floppy$Unit

FDB5 2AE5FB         LHLD    In$Buffer$Track         ;Set up track number
FDB8 7D             MOV     A,L                     ;Note: This is single byte value
FDB9 3243FD         STA     Floppy$Track            ; for the controller.
                                                    ;
                                                    ;The sector must be converted into a
                                                    ;  head number and sector number.
                                                    ;  Sectors 0 - 8 are head 0, 9 - 17
                                                    ;  are head 1
FDBC 0600           MVI     B,0                     ;Assume head 0
FDBE 3AE7FB         LDA     In$Buffer$Sector        ;Get physical sector number
FDC1 4F             MOV     C,A                     ;Save copy in case it is head 0
FDC2 FE09           CPI     9                       ;Check if < 9
FDC4 DACBFD         JC      Head$0                  ;Yes it is < 9
FDC7 D609           SUI     9                       ;No, modify sector number back
                                                    ;  in the 0 - 8 range.
FDC9 4F             MOV     C,A                     ;Put sector in B
FDCA 04             INR     B                       ;Set to head 1
                Head$0:
FDCB 78             MOV     A,B                     ;Set head number
FDCC 3242FD         STA     Floppy$Head
FDCF 79             MOV     A,C                     ;Set sector number
```

Figure 6-4. (Continued)

```
FDD0 3C          INR    A                           ;  (physical sectors start at 1)
FDD1 3244FD      STA    Floppy$Sector
                                                    ;
FDD4 210002      LXI    H,Physical$Sector$Size      ;Set byte count
FDD7 2245FD      SHLD   Floppy$Byte$Count
                                                    ;
FDDA 2133F6      LXI    H,Disk$Buffer               ;Set transfer address to be
FDDD 2247FD      SHLD   Floppy$DMA$Address          ;  disk buffer
                                                    ;
                                                    ;As only one control table is in
                                                    ;  use, close the status and busy
                                                    ;  chain pointers back to the
                                                    ;  main control bytes.
FDE0 214300      LXI    H,Disk$Status$Block
FDE3 2249FD      SHLD   Floppy$Next$Status$Block
FDE6 214500      LXI    H,Disk$Control$5
FDE9 224BFD      SHLD   Floppy$Next$Control$Location

FDEC 2140FD      LXI    H,Floppy$Command            ;Set up command block pointer
FDEF 224600      SHLD   Command$Block$5

FDF2 214500      LXI    H,Disk$Control$5            ;Activate 5 1/4" disk controller
FDF5 3680        MVI    M,80H
                 ;
                 Wait$For$Disk$Complete:            ;Wait until Disk Status Block indicates
                                                    ;  operation complete, then check
                                                    ;  if any errors occurred.
                                                    ;On entry HL -> disk control byte
FDF7 7E          MOV    A,M                         ;Get control byte
FDF8 B7          ORA    A
FDF9 C2F7FD      JNZ    Wait$For$Disk$Complete      ;Operation still not yet done
                                                    ;
FDFC 3A4300      LDA    Disk$Status$Block           ;Complete -- now check status
FDFF FE80        CPI    80H                         ;Check if any errors occurred
FE01 DA09FE      JC     Disk$Error                  ;Yes
FE04 AF          XRA    A                           ;No
FE05 32F6FB      STA    Disk$Error$Flag             ;Clear error flag
FE08 C9          RET
                 Disk$Error:
FE09 3E01        MVI    A,1                         ;Set disk-error flag nonzero
FE0B 32F6FB      STA    Disk$Error$Flag
FE0E C9          RET
                 ;
                 ;
                 ;
                 ;  Disk control table images for warm boot
                 ;
                 Boot$Control$Part$1:
FE0F 01          DB     1                           ;Read function
FE10 00          DB     0                           ;Unit (drive) number
FE11 00          DB     0                           ;Head number
FE12 00          DB     0                           ;Track number
FE13 02          DB     2                           ;Starting sector number
FE14 0010        DW     8*512                       ;Number of bytes to read
FE16 00E0        DW     CCP$Entry                   ;Read into this address
FE18 4300        DW     Disk$Status$Block           ;Pointer to next status block
FE1A 4500        DW     Disk$Control$5              ;Pointer to next control table
                 Boot$Control$Part2:
FE1C 01          DB     1                           ;Read function
FE1D 00          DB     0                           ;Unit (drive) number
FE1E 01          DB     1                           ;Head number
FE1F 00          DB     0                           ;Track number
FE20 01          DB     1                           ;Starting sector number
FE21 0006        DW     3*512                       ;Number of bytes to read
FE23 00F0        DW     CCP$Entry + (8*512)         ;Read into this address
FE25 4300        DW     Disk$Status$Block           ;Pointer to next status block
FE27 4500        DW     Disk$Control$5              ;Pointer to next control table

                 ;
                 ;
                 ;
                 WBOOT:               ;Warm boot entry
                                      ;On warm boot, the CCP and BDOS must be reloaded
                                      ;  into memory. In this BIOS, only the 5 1/4"
                                      ;  diskettes will be used. Therefore this code
```

Figure 6-4. (Continued)

```
                                   ;  is hardware specific to the controller. Two
                                   ;  prefabricated control tables are used.
FE29 318000        LXI     SP,80H
FE2C 110FFE        LXI     D,Boot$Control$Part1      ;Execute first read of warm boot
FE2F CD3BFE        CALL    Warm$Boot$Read            ;Load drive 0, track 0,
                                                     ;  head 0, sectors 2 to 8
FE32 111CFE        LXI     D,Boot$Control$Part2      ;Execute second read
FE35 CD3BFE        CALL    Warm$Boot$Read            ;Load drive 0, track 0,
                                                     ;  head 1, sectors 1 - 3
FE38 C340F8        JMP     Enter$CPM                 ;Set up base page and enter CCP
                ;
                Warm$Boot$Read:                      ;On entry, DE -> control table image
                                                     ;This control table is moved into
                                                     ;  the main disk control table and
                                                     ;  then the controller activated.
FE3B 2140FD        LXI     H,Floppy$Command          ;HL -> actual control table
FE3E 224600        SHLD    Command$Block$5           ;Tell the controller its address
                                                     ;Move the control table image
                                                     ;  into the control table itself
FE41 0E0D          MVI     C,13                      ;Set byte count
                Warm$Boot$Move:
FE43 1A            LDAX    D                         ;Get image byte
FE44 77            MOV     M,A                       ;Store into actual control table
FE45 23            INX     H                         ;Update pointers
FE46 13            INX     D
FE47 0D            DCR     C                         ;Count down on byte count
FE48 C243FE        JNZ     Warm$Boot$Move            ;Continue until all bytes moved

FE4B 214500        LXI     H,Disk$Control$5          ;Activate controller
FE4E 3680          MVI     M,80H
                Wait$For$Boot$Complete:
FE50 7E            MOV     A,M                       ;Get status byte
FE51 B7            ORA     A                         ;Check if complete
FE52 C250FE        JNZ     Wait$For$Boot$Complete    ;No
                                                     ;Yes, check for errors
FE55 3A4300        LDA     Disk$Status$Block
FE58 FE80          CPI     80H
FE5A DA5EFE        JC      Warm$Boot$Error           ;Yes, an error occurred
FE5D C9            RET
                ;
                Warm$Boot$Error:
FE5E 2167FE        LXI     H,Warm$Boot$Error$Message
FE61 CD33F8        CALL    Display$Message
FE64 C329FE        JMP     WBOOT                     ;Restart warm boot
                ;
                Warm$Boot$Error$Message:
FE67 0D0A576172    DB      CR,LF,'Warm Boot Error - retrying...',CR,LF,0
                ;
                ;
FE89               END     ;Of simple BIOS listing
```

Figure 6-4. (Continued)

7

Building a New CP/M System

This chapter describes how to build a version of CP/M with your own BIOS built into it. It also shows you how to put CP/M onto a floppy disk and how to write a bootstrap loader to bring CP/M into memory.

The manufacturer of your computer system plays a significant role in building a new CP/M system. Several of CP/M's utility programs may be modified by manufacturers to adapt them to individual computer systems. Unfortunately, not all manufacturers customize these programs. You should therefore invest some time in studying the documentation provided with your system to see what and how much customizing may have already been done. You should also assemble and print out listings of all assembly language source files from your CP/M release diskette.

It is impossible to predict the details of customization and special procedures that the manufacturer may have installed on your particular system. Therefore, this chapter describes first the overall mechanism of building a CP/M system, and second the details of building a CP/M system around the example BIOS shown in the previous chapter as Figure 6-4.

The Major Steps

Building a new CP/M system consists of the following major steps:

- Create a new or modified BIOS with the appropriate device drivers in it. Assemble this so that it will execute at the top end of memory (by using an *origin* statement (ORG) to set the location counter).
- Create new versions of the CCP and BDOS with all addresses in the instructions changed so that they will be correctly located in memory just below the new BIOS. Digital Research provides a special utility called MOVCPM to do this.
- Create or modify a CP/M bootstrap loader that will be loaded by the firmware that executes when you first switch on your computer (or press the RESET button). Normally, the CP/M bootstrap loader executes in the low-address end of memory. The exact address and the details of any hardware initialization that it must perform will depend entirely on your particular computer system.
- Using Digital Research standard utility programs, bring the bootstrap loader, the CCP and BDOS, and the BIOS together in the low part of memory. Then write this new version of CP/M onto a disk in the appropriate places. Again, depending on the design of your computer system, you may be able to use the standard utility program, SYSGEN, to write the entire CP/M *image* onto disk. Otherwise you may have to write a special program to do this.

When CP/M is already running on your computer system and you want to add new features to the BIOS, all you need to do is change the BIOS and rebuild the system. The CCP and BDOS will need to be moved down in memory if the changes expand the BIOS significantly. If this happens, you will have to make minor changes in the bootstrap loader so that it reads the new CP/M image into memory at a lower address and transfers control to the correct location (the first instruction of the BIOS jump vector).

Building Your First System

The first time that you build CP/M, it is a good idea to make no changes to the BIOS at all. Simply reassemble the BIOS source code and proceed with the system build. Then, if the new system does not run, you know that it must be something in the procedure you used rather than any new features or modification to the BIOS source code. Changes in the BIOS could easily obscure any problems you have with the build procedure itself.

The Ingredients

To build CP/M, you will need the following files and utility programs:

- The assembly language source code for your BIOS. Check your CP/M release diskette for a file with a name like CBIOS.ASM (Customized Basic Input/Output System). Some manufacturers do not supply you with the source code for their BIOS; it may be sold separately or not released at all. If you cannot get hold of the source code, the only way that you can add new features to the BIOS is by writing the entire BIOS from scratch.
- The source code for the CP/M bootstrap loader. This too may be on the release diskette or available separately from your computer's manufacturer.
- The Digital Research assembler, which converts source code into machine language in hexadecimal form. This program, called ASM.COM, will be on your CP/M release diskette. Equivalent assemblers, such as Digital Research's macro-assemblers MAC and RMAC or Microsoft's M80, can also be used.
- The Digital Research utility called MOVCPM, which prepares a memory image of the CCP and BDOS with all addresses adjusted to the right values.
- The Digital Research debugging utility, called DDT (Dynamic Debugging Tool), or the more enhanced version for the Z80 CPU chip, ZSID (Z80 Symbolic Interactive Debugger). DDT is used to read in the various program files and piece together a memory image of the CP/M system.
- The Digital Research utility program SYSGEN. This writes the composite memory image of the bootstrap, CCP, BDOS, and BIOS onto the disk. SYSGEN was designed to work on floppy disk systems. If your computer uses a hard disk, you may have a program with a name like PUTCPM or WRITECPM that performs the same function.

The Ultimate Goal

In Figure 6-4, lines 0044 to 0065, you can see the equates that define the base addresses for the CCP, the BDOS, and the BIOS. Figure 7-1 shows how the top of memory will look when this version of CP/M has been loaded into memory.

Life would be simple if you could build this image in memory at the addresses shown and write the image out to disk. Building this image, however, would probably overwrite the version of CP/M that you were operating since it too lives at the top of memory. Therefore, the goal is to create a replica of this image lower down in memory, but with all the instruction addresses set to *execute* at the addresses shown in Figure 7-1.

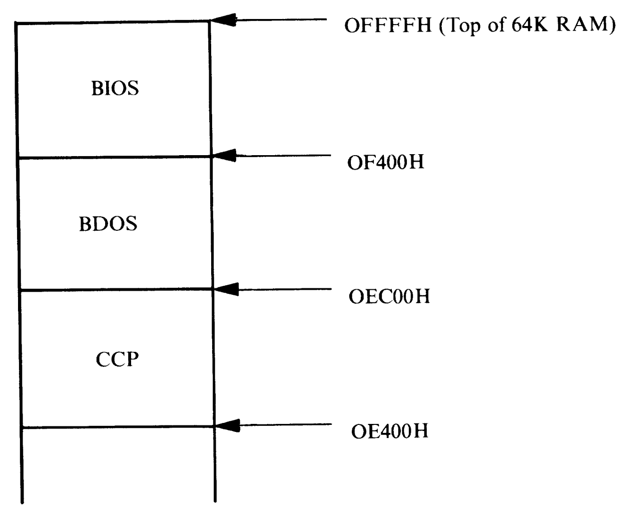


Figure 7-1. Memory layout of CP/M

Using SYSGEN to Write CP/M to Disk

The SYSGEN utility writes a memory image onto a specified logical disk. It can use a memory image that you arrange to be in memory before you invoke SYSGEN, or you can direct SYSGEN to read in a disk file that contains the image. You can also use SYSGEN to transport an existing CP/M system from one diskette to another by directing it to load the CP/M image from one diskette into memory and then to write that image out to another diskette.

Check the documentation supplied by your computer's manufacturer to make sure that you can use SYSGEN on your system. SYSGEN, as released by Digital Research, is constructed to run on 8-inch, single-sided, single-density diskettes. If your system does not use these standard diskettes, SYSGEN must be customized to your disk system.

When SYSGEN loads a CP/M image into memory, it will place the bootstrap, CCP, BDOS, and BIOS at the predetermined addresses shown in Figure 7-2, regardless of where this CP/M originated.

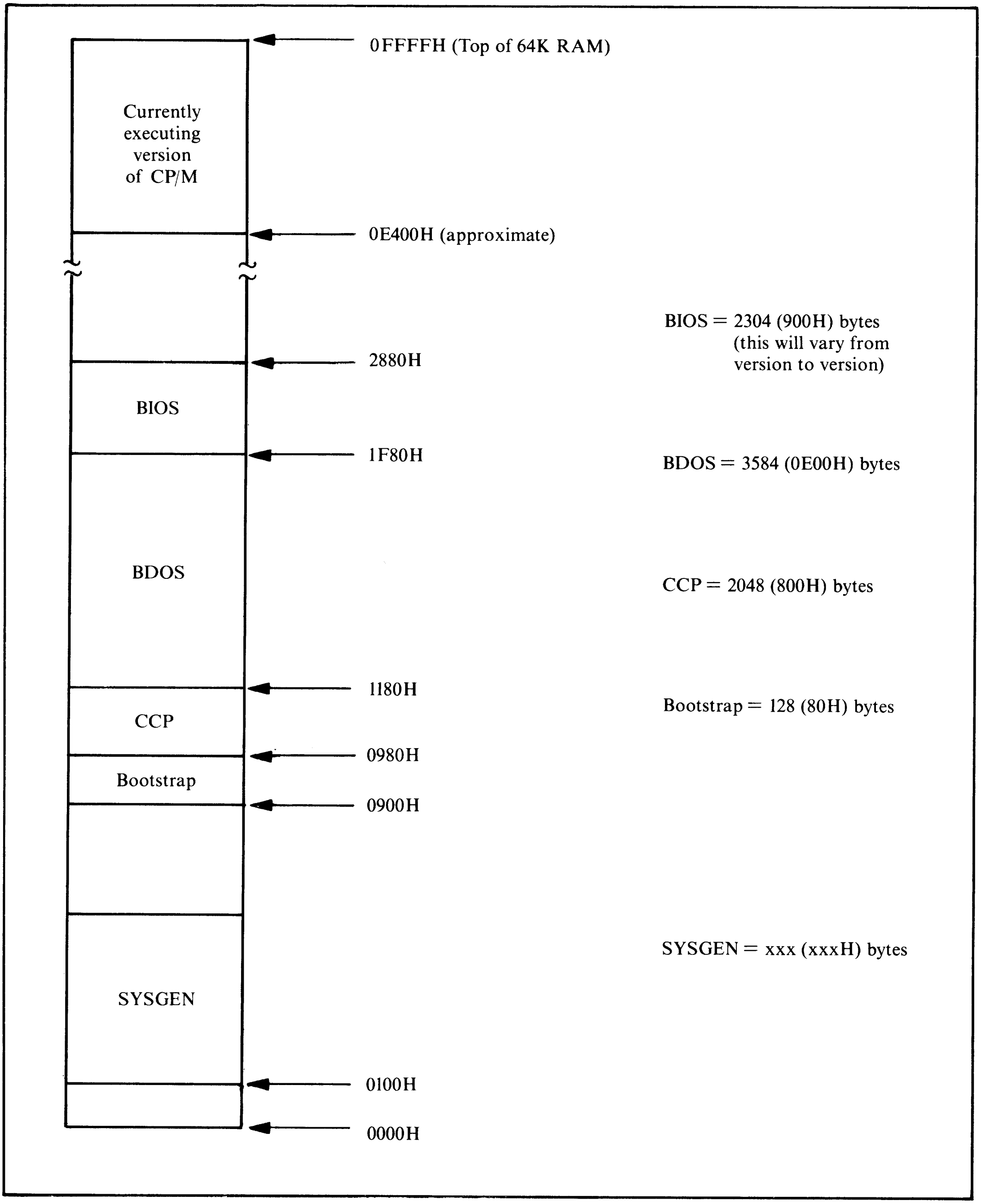


Figure 7-2. SYSGEN's memory layout

You can see that the *relative* arrangement between the components has not changed; the whole image has simply been moved down in memory well below the currently executing version of CP/M. The bootstrap has been added to the picture just beneath the CCP.

The SYSGEN utility writes this image onto a floppy diskette starting at sector 1 of track 0 and continuing to sector 26 on track 1. Refer back to Figure 2-2 to see the layout of CP/M on a standard 8-inch, single-sided, single-density diskette.

If you request SYSGEN to read the memory image from a file (which you do by calling SYSGEN with the file name on the same line as the SYSGEN call), then SYSGEN presumes that you have previously created the correct memory image and saved it (with the SAVE command). SYSGEN then skips over the first 16 sectors of the file so as to avoid overwriting itself.

Here is an example of how to use SYSGEN to move the CP/M image from one diskette to another:

```
A>SYSGEN<CR>
SYSGEN VER 2.0
SOURCE DRIVE NAME (OR RETURN TO SKIP) A
SOURCE ON A:, THEN TYPE RETURN <cr>
FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT) B
DESTINATION ON B: THEN TYPE RETURN <cr>
FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT) <cr>
A>_
```

As you can see, SYSGEN gives you the choice of specifying the source drive name or typing CARRIAGE RETURN. If you enter a CARRIAGE RETURN, SYSGEN assumes that the CP/M image is already in memory. Note that you need to call up SYSGEN only once to write out the same CP/M image to more than one disk.

A larger than standard BIOS can cause difficulties in using SYSGEN. The standard SYSGEN format only allows for six 128-byte sectors to contain the BIOS, so if your BIOS is larger than 768 (300H) bytes, it will be a problem. The CP/M image will not fit on the first two tracks of a standard 8-inch diskette.

Nowadays it is rare to find an 8-inch floppy diskette system where you must load CP/M from a single-sided, single-density diskette. Most systems now use double-sided or double-density diskettes as the normal format, but can switch to single-sided, single-density diskettes to interchange information with other computer systems.

Because there is no "standard" format for 8-inch, double-sided and double-density diskettes, you probably won't be able to read diskettes written on systems of a different make or model. Therefore, you need only be concerned about using a disk layout that will keep your disks compatible with other machines that are exactly the same as yours.

This is also true if you have 5 1/4-inch diskettes. There is no industry standard for these either, so your main consideration is to place the file directory in the same place as it will be on diskettes written by other users of your model of computer. You must also be sure to use the same sector skewing. Otherwise, you will get a garbled version whenever you try to read files originating on other systems.

With the higher capacity diskettes, you can reserve more space to hold the CP/M image on the diskette. For example, in the case of the BIOS shown in Figure 6-4, the CP/M image is written to a 5 1/4-inch, double-sided, double-density diskette using 512-byte sectors. Figure 7-3 shows the layout of this diskette. Note that the bootstrap loader is placed in a 512-byte sector all by itself. Doing so makes the bootstrap code and warm boot code in the BIOS much simpler.

The memory image must be altered to reflect the fact that the bootstrap now occupies an entire 512-byte sector. Rather than change all of the addresses, the bootstrap is loaded into memory 384 (180H) bytes lower, so that it ends at the same address as before. Figure 7-4 shows the revised memory image.

Writing a PUTCPM Utility

Because the example system uses 5 1/4-inch floppy diskettes with 512-byte sectors, the standard version of SYSGEN cannot be used to write the CP/M image onto a diskette. You will have to use a functional replacement provided by your computer's manufacturer or develop a small utility program to do the job.

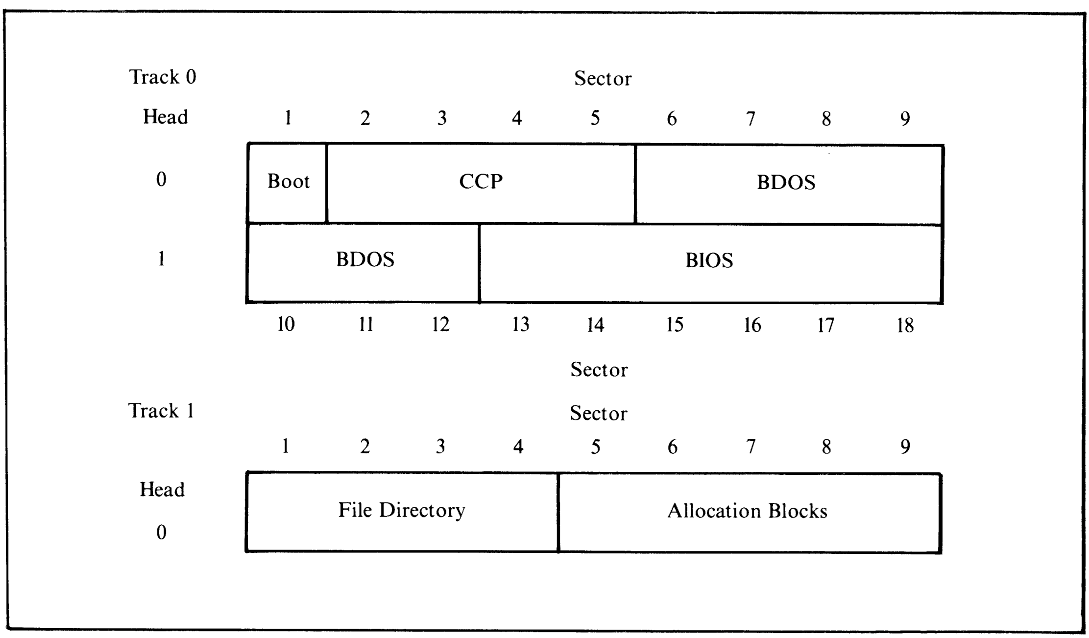


Figure 7-3. Disk layout for example BIOS on 5 1/4-inch diskettes

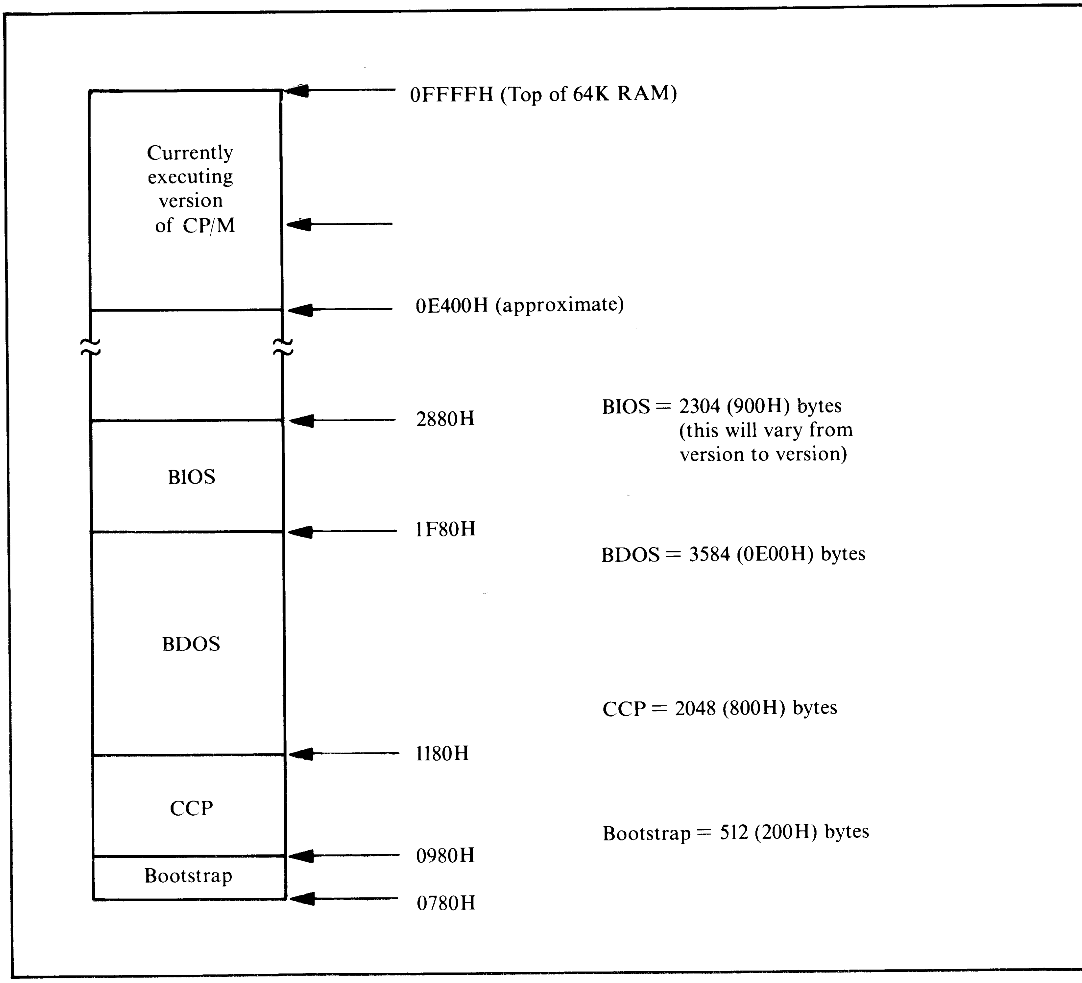


Figure 7-4. Addresses for example BIOS image

Figure 7-5 shows an example of such a program. It is written in a general-purpose way, so that you may be able to use it for your system by changing the equates at the front of the program to reflect the specifics of your disk drives.

Note that there are two problems to be solved. First, the area of the disk on which the CP/M image resides cannot be accessed by the BDOS, as it is outside the file system area on the disk. Second, it is rare to write the CP/M image onto the disk with any kind of sector skewing; to do so would slow down the loading process. In any case, skewing would be redundant, since the loader is doing no processing other than reading the disk and can therefore read the disk without skewing.

```
                ;       This program writes out the CP/M cold boot loader,
                ;       CCP, BDOS, and BIOS to a floppy diskette. It runs
                ;       under CP/M as a normal transient program.
                ;
3130 =          Version         EQU     '01'    ;Equates used in the sign-on
                                                ; message
3730 =          Month           EQU     '07'
3432 =          Day             EQU     '24'
3238 =          Year            EQU     '82'
                ;
                ;
                ;       The actual PUTCPMF5.COM program consists of this code,
                ;       plus the BOOTF5.HEX, CCP, BDOS, and BIOS.
                ;
                ;       When this program executes, the memory image should
                ;       look like this:
                ;
                ;               Component       Base Address
                ;                 BIOS             1F80H
                ;                 BDOS             1180H
                ;                 CCP              0980H
                ;                 BOOTF5           0780H
                ;
                ;       The components are produced as follows:
                ;
                ;               BIOS.HEX        By assembling source code
                ;               BDOS )          From a CPMnn.COM file output
                ;               CCP  )            by MOVCPM and SAVEd on disk
                ;               BOOTF5.HEX      By assembling source code
                ;
                ;       The components are pieced together using DDT with the
                ;       following commands:
                ;
                ;               DDT CPMnn.COM
                ;               IPUTCPMF5.HEX
                ;               R                       (Reads in this program)
                ;               IBOOTF5.HEX
                ;               R680                    (Reads in BOOT at 0780H)
                ;               IBIOS.HEX
                ;               R2980                   (Reads in BIOS at 1F80H)
                ;               G0                      (Exit from DDT)
                ;               SAVE 40 PUTCPMF5.COM    (Create final .COM file)
                ;
                ;       The actual layout of the diskette is as follows:
                ;
                ; Track 0                         Sector
                ;          1     2     3     4     5     6     7     8     9
                ; Head  +-----+-----+-----+-----+-----+-----+-----+-----+-----+
                ;  0    |Boot |<======== CCP ========>|<=======  BDOS ========|
                ;       +-----+-----+-----+-----+-----+-----+-----+-----+-----+
                ;  1    |====== BDOS ====>|<=============  BIOS  ============>|
                ;       +-----+-----+-----+-----+-----+-----+-----+-----+-----+
                ;         10    11    12    13    14    15    16    17    18
                ;                               Sector
                ;
                ;       Equates for defining memory size and the base address and
                ;       length of the system components
                ;
0040 =          Memory$Size     EQU     64      ;Number of Kbytes of RAM
                ;
                ;       The BIOS Length must match that declared in the BIOS.
                ;
0900 =          BIOS$Length     EQU     0900H
                ;
0200 =          Boot$Length     EQU     512
0800 =          CCP$Length      EQU     0800H   ;Constant
0E00 =          BDOS$Length     EQU     0E00H   ;Constant
                ;
1F00 =          Length$In$Bytes EQU     CCP$Length + BDOS$Length + BIOS$Length
                ;
0780 =          Start$Image     EQU     980H - Boot$Length      ;Address of CP/M image
2100 =          Length$Image    EQU     Length$In$Bytes + Boot$Length
                ;
                ;
```

Figure 7-5. Example PUTCPM

```
                        ;       Disk characteristics
                        ;
                        ;       These equates describe the physical characteristics of
                        ;       the floppy diskette so that the program can move from
                        ;       one sector to the next, updating the track and resetting
                        ;       the sector when necessary.
                        ;
0001 =                  First$Sector$on$Track   EQU     1
0012 =                  Last$Sector$on$Track    EQU     18
0009 =                  Last$Sector$on$Head$0   EQU     9
0200 =                  Sector$Size             EQU     512
                        ;
                        ;
                        ;       Controller characteristics
                        ;
                        ;       On this computer system, the floppy disk controller can write
                        ;       multiple sectors in a single command. However, in order
                        ;       to produce a more general example it is shown only reading one
                        ;       sector at a time.
                        ;
0001 =                  Sectors$Per$Write       EQU     1
                        ;
                        ;
                        ;       Cold boot characteristics
                        ;
0000 =                  Start$Track             EQU     0       ;Initial values for CP/M image
0001 =                  Start$Sector            EQU     1       ;= " =
0011 =                  Sectors$To$Write        EQU     (Length$Image + Sector$Size - 1) / Sector$Size
                        ;

                        ;
0009 =                  B$PRINTS        EQU     9       ;Print string terminated by $
0005 =                  BDOS            EQU     5       ;BDOS entry point
                        ;
                        ;

0100                            ORG     100H
                        Put$CPM:
0100 C33F01                     JMP     Main$Code       ;Enter main code body
                                                        ;For reasons of clarity, the main
                                                        ;  data structures are shown before the
                                                        ;  executable code.
000D =                  CR      EQU     0DH             ;Carriage return
000A =                  LF      EQU     0AH             ;Line feed
                        ;
                        Signon$Message:
0103 0D0A507574                 DB      CR,LF,'Put CP/M on Diskette'
0119 0D0A                       DB      CR,LF
011B 5665727369                 DB      'Version '
0123 3031                       DW      Version
0125 20                         DB      ' '
0126 3037                       DW      Month
0128 2F                         DB      '/'
0129 3234                       DW      Day
012B 2F                         DB      '/'
012C 3832                       DW      Year
012E 0D0A24                     DB      CR,LF,'$'

                        ;
                        ;       Disk control tables
                        ;
0045 =                  Disk$Control$5  EQU     45H     ;5 1/4" control byte
0046 =                  Command$Block$5 EQU     46H     ;Control table pointer
0043 =                  Disk$Status     EQU     43H     ;Completion status
                        ;
                        ;
                        ;       The command table track and DMA$Address can also be used
                        ;       as working storage and updated as the load process
                        ;       continues. The sector in the command table cannot be
                        ;       used directly as the disk controller requires it to be
                        ;       the sector number on the specified head (1 -- 9) rather
                        ;       than the sector number on track. Hence a separate variable
                        ;       must be used.
                        ;
```

Figure 7-5. (Continued)

```
0131 01         Sector:           DB      Start$Sector
                ;
0132 02         Command$Table:    DB      02H              ;Command -- Write
0133 00         Unit:             DB      0                ;Unit (drive) number = 0 or 1
0134 00         Head:             DB      0                ;Head number = 0 or 1
0135 00         Track:            DB      Start$Track      ;Used as working variable
0136 00         Sector$on$head:   DB      0                ;Converted by low-level driver
0137 0002       Byte$Count:       DW      Sector$Size * Sectors$Per$Write
0139 8007       DMA$Address:      DW      Start$Image
013B 4300       Next$Status:      DW      Disk$Status      ;Pointer to next status block
                                                           ;  if commands are chained
013D 4500       Next$Control:     DW      Disk$Control$5   ;Pointer to next control byte
                                                           ;  if commands are chained

                Main$Code:
013F 310001             LXI     SP,Put$CPM      ;Stack grows down below code

0142 110301             LXI     D,Signon$Message        ;Sign on
0145 0E09               MVI     C,B$PRINTS              ;Print string until $
0147 CD0500             CALL    BDOS

014A 213201             LXI     H,Command$Table         ;Point the disk controller at

014D 224600             SHLD    Command$Block$5         ;  the command block

0150 0E11               MVI     C,Sectors$To$Write      ;Set sector count
                Write$Loop:
0152 CD7C01             CALL    Put$CPM$Write           ;Write data onto diskette
0155 0D                 DCR     C                       ;Downdate sector count
0156 CA0000             JZ      0                       ;Warm boot

0159 213101             LXI     H,Sector                ;Update sector number
015C 3E01               MVI     A,Sectors$Per$Write     ;  by adding on number of sectors
015E 86                 ADD     M                       ;  by controller
015F 77                 MOV     M,A                     ;Save result
0160 3E13               MVI     A,Last$Sector$On$Track + 1      ;Check if at end of track
0162 BE                 CMP     M
0163 C26F01             JNZ     Not$End$Track

0166 3601               MVI     M,First$Sector$On$Track ;Yes, reset to beginning
0168 2A3501             LHLD    Track                   ;Update track number
016B 23                 INX     H
016C 223501             SHLD    Track

                Not$End$Track:
016F 2A3901             LHLD    DMA$Address             ;Update DMA address
0172 110002             LXI     D,Sector$Size * Sectors$Per$Write
0175 19                 DAD     D
0176 223901             SHLD    DMA$Address
0179 C35201             JMP     Write$Loop              ;Write next block
                ;
                Put$CPM$Write:                  ;At this point, the description of the
                                                ;  operation required is in the variables
                                                ;  contained in the command table, along
                                                ;  with the sector variable.

017C C5                 PUSH    B                       ;Save sector count in C

                ;------ Change this routine to match the disk controller in use ------

017D 0600               MVI     B,0                     ;Assume head 0
017F 3A3101             LDA     Sector                  ;Get requested sector
0182 4F                 MOV     C,A                     ;Take a copy of it
0183 FE0A               CPI     Last$Sector$on$Head$0+1 ;Check if on head 1
0185 DA8C01             JC      Head$0                  ;No
0188 D609               SUI     Last$Sector$on$Head$0   ;Bias down for head 1
018A 4F                 MOV     C,A                     ;Save copy
018B 04                 INR     B                       ;Set head 1
                Head$0:
018C 78                 MOV     A,B                     ;Get head
018D 323401             STA     Head
0190 79                 MOV     A,C                     ;Get sector
0191 323601             STA     Sector$On$Head
```

Figure 7-5. (Continued)

```
0194 214500          LXI     H,Disk$Control$5        ;Activate controller
0197 3680            MVI     M,80H

                Wait$For$Boot$Complete:
0199 7E              MOV     A,M                     ;Get status byte
019A B7              ORA     A                       ;Check if complete
019B C29901          JNZ     Wait$For$Boot$Complete  ;No
                                                     ;Yes, check for errors
019E 3A4300          LDA     Disk$Status
01A1 FE80            CPI     80H
01A3 DAA801          JC      Put$CPM$Error           ;Yes, an error occurred

                ;------ End of physical write routine ------

01A6 C1              POP     B                       ;Recover sector count in C
01A7 C9              RET
                ;
                Put$CPM$Error:
01A8 11B301          LXI     D,Put$CPM$Error$Message
01AB 0E09            MVI     C,B$PRINTS              ;Print string until $
01AD CD0500          CALL    BDOS                    ;Output error message
01B0 C33F01          JMP     Main$Code               ;Restart the loader
                ;

                Put$CPM$Error$Message:
01B3 0D0A457272      DB      CR,LF,'Error in writing CP/M - retrying...',CR,LF,'$'
01DB                 END     Put$CPM
```

Figure 7-5. (Continued)

Using DDT to Build the CP/M Memory Image

DDT, the Digital Research debug program, is used to read files of type ".COM" and ".HEX" into memory. Understanding the internal structure of these file types is important, both to understand what DDT can do and to understand how the MOVCPM utility can effectively change a machine code file so that it can be executed at a new address in memory.

".COM" File Structure

A COM file is a memory image. It is a replica of the bit patterns that are to be created when the file is loaded into memory. COM files are normally designed to load at location 100H upwards. No internal structure to the file requires this, however, so if you know what the contents of a COM file are, there is nothing to preclude you from loading it into memory starting at some address other than 100H.

As you may recall from the description of the CCP in Chapter 4, the SAVE command built into the CCP allows you to create a COM file by specifying the number of 256-byte "pages" of memory and the name of the file. The CCP will write out an exact image of memory from location 100H up.

".HEX" File Structure

HEX files are output by the assembler. They contain an ASCII character representation of hexadecimal values. For example, the contents of a single byte of memory with the binary value 10101111 would be represented by two ASCII characters, A F, in a HEX file.

The HEX file has a higher level structure than just a series of ASCII characters however. Each line of ASCII characters is terminated by CARRIAGE RETURN/LINE FEED. The overall structure is shown in Figure 7-6.

The most important aspect of a HEX file is that each line contains the address at which the data bytes are loaded. Each line is processed independently, so the load addresses of succeeding lines need not be in order.

DDT can read in a HEX file at an address different from the address where the code must be in order to execute. For example, you can read in the HEX file of the BIOS at the correct place for the memory image (shown in Figure 7-4). There are two ways of using DDT to read in a COM or HEX file. You can specify the name of the file on the same command line with DDT. For example:

```
A>DDT B:XYZ.HEX<cr>          <- Call up DDT with file name
DDT VERS 2.0                 <- DDT signs on
NEXT  PC
0180 0100                    <- ... and displays next free byte
                                and entry point address
                             <- ... and prompts for a commmand
```

The advantage of this method of loading a file is that you can specify which logical disk is to be searched for the file. The second way of using DDT is to load DDT first, and then, when it has given its prompt, specify the file name and request that DDT load it like this:

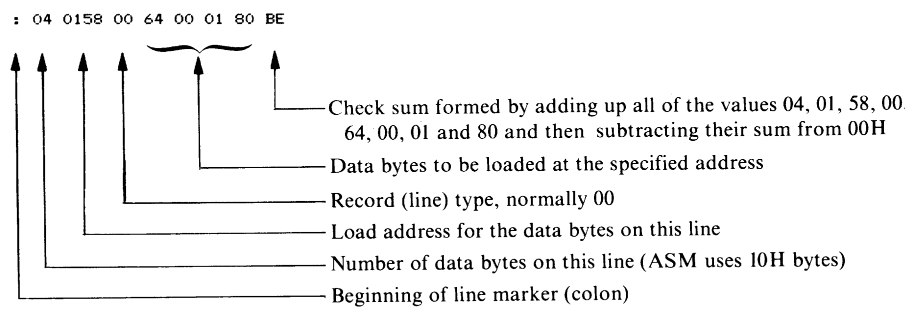
```
-Ifilename.typ<cr>           <- Enter the file name and type
-R<cr>                       <- Read in the file
```

The "I" command initializes the default file control block in the base page (at location 005CH) with the file name and type; it does *not* set up the logical disk. If you need to do this, you must set the first byte of the default FCB manually like this:

```
-Ifilename.typ<cr>           <- Specify file name
-S5C<cr>                     <- "S"et location 5C
005C 00 02<cr>               <- Was 00, you enter 02<cr>
005D 41 .<cr>                <- Enter "." to terminate
-R<cr>                       <- Read in the file
```

Location 005CH should be set to 01H for Drive A, 02H for B, and so on.

The "R" command will read in HEX files to the *execution* addresses specified in each line of the HEX file, so be careful—if you forget to put an ORG (origin)



NOTE: HEX files do not have embedded blank characters; the example above is shown with gaps between individual fields only for clarity.

Figure 7-6. Example line from HEX file

statement at the front of the assembly language source code, reading in the resultant HEX file will overwrite location 0000H on up, destroying the contents of the base page. Similarly, if you were trying to read in the HEX file for a BIOS, there is an excellent chance that you will overwrite the currently executing CP/M system.

DDT reacts to the file type you enter as part of the file name. For file types other than .HEX, DDT loads the file starting at location 0100H on up.

The "R" command can also be used to read files into memory at different addresses. You do this by typing a hexadecimal number immediately after the R, with no intervening punctuation. For HEX files, the number that you enter is added to the address in each line of the HEX file and the sum is used as the address into which the data bytes are loaded. The data bytes themselves are not changed, just the load address.

For COM files, the number that you enter is added to 0100H and the sum is used as the starting address for loading the file.

The sum is performed as 16-bit, unsigned arithmetic with any carry ignored, so you can load a BIOS HEX file into low memory by using the "R" command with what is called an "offset value."

If a HEX file has been assembled to execute at address "exec," and you need to use DDT to read in this file to address "load," you need to solve the following equation:

offset = load − exec.

DDT's "H" command performs hexadecimal arithmetic. It calculates and displays the sum of and difference between two hexadecimal values. For example, the BIOS in Figure 6-4 has been assembled to *execute* at location 0F600H, but needs to be *loaded* into memory at location 1F80H. Here is how to compute the correct offset for the "R" command:

```
-H1F80,F600<cr>          <- Use the H command
1580,2980                <- Sum, difference
```

Thus, to read in the BIOS HEX file called FIG6-4.HEX at location 1F80H, you would enter the following commands to DDT:

```
-IFIG6-4.HEX<cr>         <- Specify file name and type
-R2980<cr>               <- Load at 0F600H + 2980H (= 1F80H)
```

In this way, using DDT, you can read in the HEX files for both the BIOS and the bootstrap loader.

The CP/M Bootstrap Loader

The bootstrap loader is brought into memory by PROM-based firmware in the computer system. It loads in the CCP, BDOS, and BIOS and then transfers control to the cold boot entry point in the BIOS—the first jump instruction in the BIOS jump vector.

The bootstrap loader is a stand-alone program; it cannot make use of any CP/M functions because no part of CP/M is in memory when the bootstrap loader is needed. The firmware in the PROM that loaded the bootstrap may contain some subroutines that can be used by the bootstrap, but this will vary from system to system.

Figure 7-7 shows the bootstrap code for the example BIOS (from Figure 6-4). This code has been written in a general way, so that you can adapt it to your system. The disk controller on the example system can in fact read in multiple sectors from the disk, but for generality the code shown reads in only one sector at a time. This considerably increases the time it takes to load CP/M, but does make the bootstrap loader more general.

Note that almost the first thing that the bootstrap does is to output to the console a sign-on message. Not only does this confirm the version number, but it shows that the bootstrap has been successfully loaded.

The PROM-based code has been designed to load the CP/M bootstrap into location 100H, allowing the code to be debugged as though it were a normal transient program, albeit with minor changes to the address at which it loads the CP/M image from disk. Clearly, this feature is not very helpful if CP/M is being brought up for the first time on a computer system. It helps a great deal, however, if you need to modify the bootstrap or add the capability to boot your system from a new type of disk drive.

```
                ;       Example CP/M cold bootstrap loader
                ;
                ;       This program is written out to track 0, head 0, sector 1
                ;       by the PUTCPMF5 program.
                ;       It is loaded into memory at location 100H on up by the
                ;       PROM-based bootstrap mechanism that gets control of the
                ;       CPU on power up or system reset.
                ;
3130 =          Version         EQU     '01'    ;Equates used in the sign-on message
3730 =          Month           EQU     '07'
3432 =          Day             EQU     '24'
3238 =          Year            EQU     '82'
                ;
0000 =          Debug           EQU     0       ;Set nonzero to debug as normal
                                                ;  transient program
                ;
                ;       The actual layout of the diskette is as follows :
                ;
                ; Track 0                       Sector
                ;         1     2     3     4     5     6     7     8     9
                ; Head  +-----+-----+-----+-----+-----+-----+-----+-----+-----+
                ;  0    |Boot |<======== CCP ========>|<======= BDOS ========|
                ;       +-----+-----+-----+-----+-----+-----+-----+-----+-----+
                ;  1    |====== BDOS ====>|<============= BIOS ============>|
                ;       +-----+-----+-----+-----+-----+-----+-----+-----+-----+
                ;         10    11    12    13    14    15    16    17    18
                ;                               Sector
                ;
                ;       Equates for defining memory size and the base address and
                ;       length of the system components.
                ;
0040 =          Memory$Size     EQU     64      ;Number of Kbytes of RAM
                ;
                ;       The BIOS Length must match that declared in the BIOS.
                ;
0900 =          BIOS$Length     EQU     0900H
                ;
0800 =          CCP$Length      EQU     0800H   ;Constant
0E00 =          BDOS$Length     EQU     0E00H   ;Constant
                ;
0008 =          Length$In$K     EQU     ((CCP$Length + BDOS$Length + BIOS$Length) / 1024) + 1
1F00 =          Length$In$Bytes EQU     CCP$Length + BDOS$Length + BIOS$Length
                ;
                        IF      NOT Debug
E000 =          CCP$Entry       EQU     (Memory$Size - Length$In$K) * 1024
                        ENDIF
                        IF      Debug
                CCP$Entry       EQU     3980H   ;Read into a lower address.
                                                ;This address is chosen to be above
                                                ;  the area into which DDT initially loads
                                                ;  and the 980H makes the addresses similar
                                                ;  to the SYSGEN values so that the memory
                                                ;  image can be checked with DDT.
                        ENDIF

E806 =          BDOS$Entry      EQU     CCP$Entry + CCP$Length + 6
F600 =          BIOS$Entry      EQU     CCP$Entry + CCP$Length + BDOS$Length
                ;

                ;
                ;       Disk characteristics
                ;
                ;       These equates describe the physical characteristics of
                ;       the floppy diskette so that the program can move from
                ;       one sector to the next, updating the track and resetting
                ;       the sector when necessary.
                ;
0001 =          First$Sector$on$Track   EQU     1
0012 =          Last$Sector$on$Track    EQU     18
0009 =          Last$Sector$on$Head$0   EQU     9
0200 =          Sector$Size             EQU     512
                ;
                ;
                ;       Controller characteristics
                ;
```

Figure 7-7. Example CP/M cold bootstrap loader

```
                ;       On this computer system, the floppy disk controller can read
                ;       multiple sectors in a single command. However, in order to
                ;       produce a more general example it is shown only reading one
                ;       sector at a time.
                ;
0001 =          Sectors$Per$Read        EQU     1
                ;
                ;
                ;       Cold boot characteristics
                ;
0000 =          Start$Track             EQU     0       ;Initial values for CP/M image
0002 =          Start$Sector            EQU     2       ;= " =
0010 =          Sectors$To$Read         EQU     (Length$In$Bytes + Sector$Size - 1) / Sector$Size
                ;
                ;
                ;

0100                    ORG     100H
                Cold$Boot$Loader:
0100 C34001             JMP     Main$Code       ;Enter main code body
                                                ;For reasons of clarity, the main
                                                ;  data structures are shown before the
                                                ;  executable code.
000D =          CR      EQU     0DH             ;Carriage return
000A =          LF      EQU     0AH             ;Line feed
                ;
                Signon$Message:
0103 0D0A43502F         DB      CR,LF,'CP/M Bootstrap Loader'
                        IF      Debug
                        DB      ' (Debug)'
                        ENDIF
011A 0D0A               DB      CR,LF
011C 5665727369         DB      'Version '
0124 3031               DW      Version
0126 20                 DB      ' '
0127 3037               DW      Month
0129 2F                 DB      '/'
012A 3234               DW      Day
012C 2F                 DB      '/'
012D 3832               DW      Year
012F 0D0A00             DB      CR,LF,0

                ;
                ;        Disk Control Tables
                ;
0045 =          Disk$Control$5  EQU     45H     ;5 1/4" control byte
0046 =          Command$Block$5 EQU     46H     ;Control table pointer
0043 =          Disk$Status     EQU     43H     ;Completion status
                ;
                ;
                ;       The command table track and DMA$Address can also be used
                ;       as working storage and updated as the load process
                ;       continues. The sector in the command table cannot be
                ;       used directly as the disk controller requires it to be
                ;       the sector number on the specified head (1 -- 9) rather
                ;       than the sector number on track. Hence a separate variable
                ;       must be used.
                ;
0132 02         Sector:         DB      Start$Sector
                ;
0133 01         Command$Table:  DB      01H     ;Command -- read
0134 00         Unit:           DB      0       ;Unit (drive) number = 0 or 1
0135 00         Head:           DB      0       ;Head number = 0 or 1
0136 00         Track:          DB      Start$Track     ;Used as working variable
0137 00         Sector$on$head: DB      0       ;Converted by low-level driver
0138 0002       Byte$Count:     DW      Sector$Size * Sectors$Per$Read
013A 00E0       DMA$Address:    DW      CCP$Entry
013C 4300       Next$Status:    DW      Disk$Status     ;Pointer to next status block
                                                        ;  if commands are chained.
013E 4500       Next$Control:   DW      Disk$Control$5  ;Pointer to next control byte
                                                        ;  if commands are chained.

                Main$Code:
0140 310001             LXI     SP,Cold$Boot$Loader     ;Stack grows down below code
```

Figure 7-7. (Continued)

```
0143 210301         LXI     H,Signon$Message        ;Sign on
0146 CDD901         CALL    Display$Message

0149 213301         LXI     H,Command$Table         ;Point the disk controller at
014C 224600         SHLD    Command$Block$5         ;  the command block

014F 0E10           MVI     C,Sectors$To$Read       ;Set sector count
               Load$Loop:
0151 CD7B01         CALL    Cold$Boot$Read          ;Read data into memory
0154 0D             DCR     C                       ;Downdate sector count

                    IF      NOT Debug
0155 CA00F6         JZ      BIOS$Entry              ;Enter BIOS when load done
                    ENDIF
                    IF      Debug
                    JZ      0                       ;Warm boot
                    ENDIF

0158 213201         LXI     H,Sector                ;Update sector number
015B 3E01           MVI     A,Sectors$Per$Read      ;  by adding on number of sectors
015D 86             ADD     M                       ;  by controller
015E 77             MOV     M,A                     ;Save result
015F 3E13           MVI     A,Last$Sector$On$Track + 1      ;Check if at end of track
0161 BE             CMP     M
0162 C26E01         JNZ     Not$End$Track

0165 3601           MVI     M,First$Sector$On$Track ;Yes, reset to beginning
0167 2A3601         LHLD    Track                   ;Update track number
016A 23             INX     H
016B 223601         SHLD    Track

               Not$End$Track:
016E 2A3A01         LHLD    DMA$Address             ;Update DMA Address
0171 110002         LXI     D,Sector$Size * Sectors$Per$Read
0174 19             DAD     D
0175 223A01         SHLD    DMA$Address
0178 C35101         JMP     Load$Loop               ;Read next block
               ;
               Cold$Boot$Read:                  ;At this point, the description of the
                                                ;  operation required is in the variables
                                                ;  contained in the command table, along
                                                ;  with the sector variable.

017B C5             PUSH    B                       ;Save sector count in C

               ;------ Change this routine to match the disk controller in use ------

017C 0600           MVI     B,0                     ;Assume head 0
017E 3A3201         LDA     Sector                  ;Get requested sector
0181 4F             MOV     C,A                     ;Take a copy of it
0182 FE0A           CPI     Last$Sector$on$Head$0+1 ;Check if on head 1
0184 DA8B01         JC      Head$0                  ;No
0187 D609           SUI     Last$Sector$on$Head$0   ;Bias down for head 1
0189 4F             MOV     C,A                     ;Save copy
018A 04             INR     B                       ;Set head 1
               Head$0:
018B 78             MOV     A,B                     ;Get head
018C 323501         STA     Head
018F 79             MOV     A,C                     ;Get sector
0190 323701         STA     Sector$On$Head

0193 214500         LXI     H,Disk$Control$5        ;Activate controller
0196 3680           MVI     M,80H

               Wait$For$Boot$Complete:
0198 7E             MOV     A,M                     ;Get status byte
0199 B7             ORA     A                       ;Check if complete
019A C29801         JNZ     Wait$For$Boot$Complete  ;No
                                                    ;Yes, check for errors
019D 3A4300         LDA     Disk$Status
01A0 FE80           CPI     80H
01A2 DAA701         JC      Cold$Boot$Error         ;Yes, an error occurred

               ;------ End of physical read routine ------
```

Figure 7-7. (Continued)

```
01A5 C1                 POP     B                       ;Recover sector count in C
01A6 C9                 RET
                ;
                Cold$Boot$Error:
01A7 21B001             LXI     H,Cold$Boot$Error$Message
01AA CDD901             CALL    Display$Message         ;Output error message
01AD C34001             JMP     Main$Code               ;Restart the loader
                ;
                Cold$Boot$Error$Message:
01B0 0D0A426F6F         DB      CR,LF,'Bootstrap Loader Error - retrying...',CR,LF,0
                ;
                ;       Equates for Terminal Output
                ;
0001 =          Terminal$Status$Port    EQU     01H
0002 =          Terminal$Data$Port      EQU     02H
                ;
0001 =          Terminal$Output$Ready   EQU     0000$0001B
                ;
                ;
                Display$Message:        ;Displays the specified message on the console.
                                        ;On entry, HL points to a stream of bytes to be
                                        ;output. A 00H-byte terminates the message.
01D9 7E                 MOV     A,M             ;Get next message byte
01DA B7                 ORA     A               ;Check if terminator
01DB C8                 RZ                      ;Yes, return to caller
01DC 4F                 MOV     C,A             ;Prepare for output

                Output$Not$Ready:
01DD DB01               IN      Terminal$Status$Port    ;Check if ready for output
01DF E601               ANI     Terminal$Output$Ready
01E1 CADD01             JZ      Output$Not$Ready        ;No, wait
01E4 79                 MOV     A,C                     ;Get data character
01E5 D302               OUT     Terminal$Data$Port      ;Output to screen

01E7 23                 INX     H               ;Move to next byte of message
01E8 C3D901             JMP     Display$Message ;Loop until complete message output

                                                ;The PROM-based bootstrap loader checks
                                                ;  to see that the characters "CP/M"
                                                ;  are on the diskette bootstrap sector
                                                ;  before it transfers control to it.
02E0                    ORG     2E0H
02E0 43502F4D           DB      'CP/M'
02E4                    END     Cold$Boot$Loader
```

Figure 7-7. (Continued)

In this case, the bootstrap code must be loaded at location 0780H, not the normal 0980H, because the bootstrap takes a complete 512-byte sector (200H). The same principle applies in determining the offset value to be used with DDT's "R" command to read the bootstrap HEX file, namely:

offset = load address − execution address.

In this case, the values are the following:

0680H = 0780H − 0100H

Using MOVCPM to Relocate the CCP and BDOS

MOVCPM builds a CP/M memory image at the correct locations for SYSGEN, but with the instructions modified to execute at a specific address. Inside MOVCPM is not only a complete replica of CP/M, but also enough information to tell MOVCPM which bytes of which instructions need be changed whenever the execution address of the image needs to be moved.

MOVCPM, as released from Digital Research, contains the bootstrap and BIOS for an Intel MDS-800 computer along with the generic CCP and BDOS. Unless you have an MDS-800, all you use is the CCP and BDOS. Some manufacturers have customized MOVCPM to include the correct bootstrap and BIOS for their own computers; consult their documentation to see if this applies to your computer system.

When you invoke MOVCPM, you have the following options:

- MOVCPM<cr>
  MOVCPM will relocate its built-in copy of CP/M to the top of available memory and will then transfer control to this new image of CP/M. Unless your manufacturer has included the correct BIOS into MOVCPM, using this option will cause an immediate system crash.
- MOVCPM nn<cr>
  This is similar to the option above, except that MOVCPM assumes that *nn*K bytes of memory are available and will relocate the CP/M image to the top of that before transferring control. Again, this will crash the system unless the correct BIOS has been installed into MOVCPM.
- MOVCPM * *<cr>
  MOVCPM will adjust all of the internal addresses inside the CP/M image so that the image could execute at the top of available memory, but instead of actually putting this image at the top of memory, MOVCPM will leave it in low memory at the correct place for SYSGEN to write it onto a disk. The SAVE command could also preserve the image on a disk.
- MOVCPM *nn* *<cr>
  MOVCPM proceeds as above for the "* *" option except that the CP/M image is modified to execute at the top of *nn*K.

MOVCPM has a fundamental problem. The *nn* value indicates that the top of available memory is computed, assuming that your BIOS is small—less that 890 (380H) bytes. If your BIOS is larger (as is the case with the example in Figure 6-4), then you will have to reduce the value of "*nn*" artificially.

Figure 7-8 shows the relationship between the size of the BIOS and the "*nn*" value to use with MOVCPM. It also shows, for different lengths of BIOS, the BIOS base address, the offset value to be used in DDT to read in the BIOS to location 1F80H (preparatory to using SYSGEN or PUTCPM to write it out), and also the base addresses for the CCP and the BDOS. The base address of the BDOS indicates how much memory is available for loading transient programs, as the CCP can be overwritten if necessary.

The numbers in Figure 7-8 are based on the assumption that you have 64K of memory in your computer system. If this is not the case, then proceed as follows:

1. Convert the amount of memory in your system to hex. Remember that 1K is 1024 bytes.
2. Determine the length of your BIOS in hex.
3. Locate the line in Figure 7-8 that shows a BIOS length equal to or greater than the length of your BIOS.
4. Using the "H" command in DDT, compute the BIOS Base Address using the formula:

   Memory in system − BIOS length from Figure 7-8
5. Find the line in Figure 7-8 that shows the same BIOS Base Address as the result of the computation above. Use this line to derive the other relevant numbers.

It is helpful to use DDT to examine a CP/M image in memory to check that all of the components are correctly placed, and, in the case of the CCP and BDOS, correctly relocated.

Figure 7-9 shows an example console dialog in which DDT is used first to examine the memory image produced by MOVCPM and second to examine the image built into the PUTCPMF utility shown in Figure 7-5.

BIOS Length	BIOS Base	DDT Offset	MOVCPM 'nn'	CCP Base	BDOS Base
600	FA00	2580	64	E400	EC00
A00	F600	2980	63	E000	E800
E00	F200	2D80	62	DC00	E400
1200	EE00	3180	61	D800	E000
1600	EA00	3580	60	D400	DC00
1A00	E600	3980	59	D000	D800
1E00	E200	3D80	58	CC00	D400
2200	DE00	4180	57	C800	D000
2600	DA00	4580	56	C400	CC00
2A00	D600	4980	55	C000	C800
2E00	D200	4D80	54	BC00	C400
3200	CE00	5180	53	B800	C000
3600	CA00	5580	52	B400	BC00
3A00	C600	5980	51	B000	B800
3E00	C200	5D80	50	AC00	B400
4200	BE00	6180	49	A800	B000
4600	BA00	6580	48	A400	AC00
4A00	B600	6980	47	A000	A800
4E00	B200	6D80	46	9C00	A400
5200	AE00	7180	45	9800	A000
5600	AA00	7580	44	9400	9C00
5A00	A600	7980	43	9000	9800
5E00	A200	7D80	42	8C00	9400
6200	9E00	8180	41	8800	9000
6600	9A00	8580	40	8400	8C00
6A00	9600	8980	39	8000	8800

Apart from the MOVCPM 'nn' value all other values are in hexadecimal

Figure 7-8. CP/M addresses for different BIOS lengths

```
                      Call up MOVCPM requesting a '63K' system
                      and the image to be left in memory.
A>Movcpm 63 *<cr>
CONSTRUCTING 63k CP/M vers 2.2
READY FOR "SYSGEN" OR
"SAVE 34 CPM63.COM"

                      Save the image from location 100H up. By
                      convention, the file name is CPMnn.COM, so
                      in this case it will be CPM63.COM
A>Save 34 cpm63.com<cr>

                      Call up DDT and request that it read in
                      CPM63.COM
A>ddt cpm63.com<cr>
DDT VERS 2.2
NEXT  PC
2300 0100

                      Display memory to show the first few bytes of
                      the CCP. Note the two JMP (C3H) instructions,
                      followed by 7FH, 00H, 20H's, and the Digital
                      Research Copyright notice. These identify the
                      code as being the CCP. Note that the first
                      JMP instruction is to 35CH into the CCP -- you
                      can therefore infer the base address of the
                      CCP. In this case the JMP is to locat;on E35C,
                      therefore this version of the CCP has been
                      configured to execute based at E000H.
-d980,9cf<cr>
0980 C3 5C E3 C3 58 E3 7F 00 20 20 20 20 20 20 20 20 .\..X...
0990 20 20 20 20 20 20 20 20 43 4F 50 59 52 49 47 48         COPYRIGH
09A0 54 20 28 43 29 20 31 39 37 39 2C 20 44 49 47 49 T (C) 1979, DIGI
09B0 54 41 4C 20 52 45 53 45 41 52 43 48 20 20 00 00 TAL RESEARCH  ..
09C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................


                      Display the first few bytes of the BDOS. Note
                      the JMP instruction at 1186. This is the
                      instruction to which control is transferred
                      by the JMP in location 5.
-d1180,118F<cr>
1180 00 16 00 00 09 85 C3 11 E8 99 E8 A5 E8 AB E8 B1 ................

                      Displaying further up in the BDOS identifies
                      it unambiguously -- there are some ASCII error
                      messages.
-d1230,126f<cr>
1230 E8 21 DC E8 CD E5 E8 C3 00 00 42 64 6F 73 20 45 .!........Bdos E
1240 72 72 20 4F 6E 20 20 3A 20 24 42 61 64 20 53 65 rr On  : $Bad Se
1250 63 74 6F 72 24 53 65 6C 65 63 74 24 46 69 6C 65 ctor$Select$File
1260 20 52 2F 4F 24 E5 CD C9 E9 3A 42 EB C6 41 32 C6  R/O$....:B..A2.

                      Display the first few bytes of the BIOS.
                      Notice the BIOS JMP vector -- the series of C3H
                      instructions. Normally the first instruction
                      in the vector can be used to infer the base
                      address of the BIOS; in this case it is
                      F600H. But there is no rule that says that
                      the cold boot code must be close to the BIOS
                      JMP vector -- so this is only a rough guide.
-d1f80<cr>
1F80 C3 B3 F6 C3 C3 F6 C3 61 F7 C3 64 F7 C3 6A F7 C3 .......a..d..j..
1F90 6D F7 C3 72 F7 C3 75 F7 C3 78 F7 C3 7D F7 C3 A7 m..r..u..x..}...
1FA0 F7 C3 AC F7 C3 BB F7 C3 C1 F7 C3 CA F7 C3 70 F7 ..............p.
1FB0 C3 B1 F7 82 F6 00 00 00 00 00 00 6E F8 73 F6 0D ...........n.s..
1FC0 F9 EE F8 82 F6 00 00 00 00 00 00 6E F8 73 F6 3C ...........n.s.<
1FD0 F9 1D F9 82 F6 00 00 00 00 00 00 6E F8 73 F6 6B ...........n.s.k
1FE0 F9 4C F9 82 F6 00 00 00 00 00 00 6E F8 73 F6 9A .L.........n.s..
1FF0 F9 7B F9 1A 00 03 07 00 F2 00 3F 00 C0 00 10 00 .{.........?.....
2000 02 00 01 07 0D 13 19 05 0B 11 17 03 09 0F 15 02 ................
2010 08 0E 14 1A 06 0C 12 18 04 0A 10 16 0D 0A 0A 36 ...............6
2020 33 6B 20 43 50 2F 4D 20 76 65 72 73 20 32 2E 32 3k CP/M vers 2.2
2030 0D 0A 00 31 00 01 21 9C F6 CD D3 F7 AF 32 04 00 ...1..!......2..
-^C
```

Figure 7-9. Using DDT to check CP/M images

```
                               In contrast, load DDT and request that it
                               load the PUTCPMF5.COM program.
A>ddt putcpmf5.com<cr>
DDT VERS 2.2
NEXT  PC
2900 0100

                               Display the special bootstrap loader that
                               starts at location 0780H (compared to the
                               MDS-800 bootstrap which is at 0980H). Note
                               the sign-on message.
-d780,7af<cr>
0780 C3 40 01 0D 0A 43 50 2F 4D 20 42 6F 6F 74 73 74 .@...CP/M Bootst
0790 72 61 70 20 4C 6F 61 64 65 72 0D 0A 56 65 72 73 rap Loader..Vers
07A0 69 6F 6E 20 30 31 20 30 37 2F 32 34 2F 38 32 0D ion 01 07/24/82.

                               Confirm that the CCP is loaded in the correct
                               place. Check the address of the first JMP
                               instruction (0E35CH).
-d980,9bf<cr>
0980 C3 5C E3 C3 58 E3 7F 00 20 20 20 20 20 20 20 20 .\..X...
0990 20 20 20 20 20 20 20 20 43 4F 50 59 52 49 47 48          COPYRIGH
09A0 54 20 28 43 29 20 31 39 37 39 2C 20 44 49 47 49 T (C) 1979, DIGI
09B0 54 41 4C 20 52 45 53 45 41 52 43 48 20 20 00 00 TAL RESEARCH  ..

                               Confirm that the BDOS is also in place.
-d1180,118f<cr>
1180 00 16 00 00 09 85 C3 11 E8 99 E8 A5 E8 AB E8 B1 ................

                               Confirm that the BIOS has been loaded in the
                               correct place. Check the first JMP to get
                               some idea of the BIOS base address. Note the
                               sign-on message.
-d1f80<cr>
1F80 C3 F9 F6 C3 0C FE C3 62 F8 C3 78 F8 C3 86 F8 C3 .......b..x.....
1F90 A4 F8 C3 B4 F8 C3 C5 F8 C3 B6 FB C3 0E FB C3 3B ...............;
1FA0 FB C3 41 FB C3 48 FB C3 DE FB C3 F8 FB C3 94 F8 ..A..H..........
1FB0 C3 B0 FB ED 06 00 00 00 42 6E 25 DF 01 B6 DE 02 ........Bn%.....
1FC0 38 00 00 43 50 2F 4D 20 32 2E 32 2E 30 30 20 30 8..CP/M 2.2.00 0
1FD0 37 2F 31 35 2F 38 32 0D 0A 0A 53 69 6D 70 6C 65 7/15/82...Simple
1FE0 20 42 49 4F 53 0D 0A 0A 44 69 73 6B 20 43 6F 6E  BIOS...Disk Con
1FF0 66 69 67 75 72 61 74 69 6F 6E 20 3A 0D 0A 0A 20 figuration :...
2000 20 20 20 20 41 3A 20 30 2E 33 35 20 4D 62 79 74     A: 0.35 Mbyt
2010 65 20 35 22 20 46 6C 6F 70 70 79 0D 0A 20 20 20 e 5" Floppy..
2020 20 20 42 3A 20 30 2E 33 35 20 4D 62 79 74 65 20   B: 0.35 Mbyte
2030 35 22 20 46 6C 6F 70 70 79 0D 0A 0A 20 20 20 20 5" Floppy...
-^C
A>_
```

Figure 7-9. Using DDT to check CP/M images (continued)

Putting it all Together

Figure 7-10 shows an annotated console dialog for the complete generation of a new CP/M system. Note that the following file names appear in the dialog:

```
BIOS1.ASM      Figure 6-4.
PUTCPMF5.ASM   Figure 7-5.
BOOTF5.ASM     Figure 7-7.
```

Assemble the CP/M Bootstrap Loader, with the source code and HEX file on drive C:, no listing output.

```
C>asm bootf5.ccz<cr>
CP/M ASSEMBLER - VER 2.0
02E4
004H USE FACTOR
END OF ASSEMBLY
```

Assemble the PUTCPMF5 program (that writes CP/M onto the disk), with the source code and HEX file on drive C:, no listing output.

```
C>asm putcpmf5.ccz<cr>
CP/M ASSEMBLER - VER 2.0
01DB
003H USE FACTOR
END OF ASSEMBLY
```

Assemble the BIOS with the source code and HEX file on drive C:, no listing output.

```
C>asm bios1.ccz<cr>
CP/M ASSEMBLER - VER 2.0
FE6C
011H USE FACTOR
END OF ASSEMBLY
```

Start piecing the CP/M image together. Load DDT and ask it to read in the file previously SAVEd after a MOVCPM 63 *.

```
C>ddt cpm63.com<cr>
DDT VERS 2.2
NEXT  PC
2300 0100
```

Indicate the file name of PUTCPMF5.HEX, and read in without any offset (i.e. it will load at 100H because of the ORG 100H it contains). -iputcpmf5.hex<cr>

```
-r<cr>
NEXT  PC
2300 0100
```

Indicate the file name of BOOTF5.HEX and read in with an offset of 680H to make it load at 780H on up (it contains ORG 100H too).

```
-ibootf5.hex<cr>
-r680<cr>
NEXT  PC
2300 0100
```

Indicate the file name of the BIOS HEX file, and read it in with an offset of 2980 such that it will load at 1F80H (it contains an ORG 0F600H).

```
-ibios1.hex<cr>
-r2980<cr>
NEXT  PC
27EC 0000
```

Exit from DDT by going to location 0000H and executing a warm boot.

```
-g0<cr>
```

Save the complete CP/M image on disk. Saving 40 256-byte pages from location 100H to 2900H.

```
C>save 40 putcpmf5.com<cr>
```

Figure 7-10. Console dialog for system build

```
                              Load and execute the PUTCPMF5
                              program.
C>putcpmf5<cr>

                              PUTCPMF5 signs on
Put CP/M on Diskette
Version 01 07/24/82
                              and writes the CP/M image to
                              disk.
C>
```

Figure 7-10. Console dialog for system build (continued)