8

Writing An Enhanced BIOS

This chapter describes ways in which you can enhance your BIOS to make CP/M easier to use, faster, and more versatile.

Get a standard BIOS working on your computer system, and then install the additional features. Although you can write an enhanced BIOS from the outset, it will take considerably longer to get it functioning correctly.

A complete listing of an enhanced BIOS is included at the end of this chapter. It is quite large: approximately 4500 lines of source code, with extensive comments and long variable names to make it more understandable.

The sections that follow describe the main concepts embodied in the enhanced BIOS listing.

BIOS Enhancements

BIOS enhancements fall into two classes: those that add new capabilities and those that extend existing features.

Some enhancements are normally accompanied by utility programs that allow you to select the enhancement option from the console. For example, when the BIOS is enhanced to include a *real time clock,* you need a utility program to set the clock to the correct time. Other enhancements will not require supporting utilities. For example, if the disk drivers are improved to read and write data faster, the enhancement is "transparent." As a user, you are aware of the results of the enhancement but not of the enhancement itself.

Viewed at its simplest, the BIOS deals with two broad classes of input/output:

Character input/output
This includes the console, auxiliary, and list devices.

Disk input/output
This can accommodate several types of floppy and hard disks.

Enhancements in these areas do not fundamentally change the way that the BDOS and CCP interact with these devices. Instead, enhancements improve the way in which the *device drivers* deal with the devices. They can improve the speed of manipulating data, the way of handling external devices, or the user's control over the behavior of the system.

The example enhanced BIOS has capabilities not found in standard CP/M systems. These can be grouped in several main categories:

Character input/output
This area probably benefits most from enhancement. This is partly because such a wide range of peripheral devices needs to be supported and partly because this is the most visible area of interaction between you and your computer. Any improvements here will therefore be immediate and obvious to you as a user.

Error handling
CP/M's error handling is, at best, startling in its simplicity. Enhanced error handling gives you more information about the nature of the failure, and then gives you the options of retrying the operation, ignoring the error, or aborting the program. This topic is covered in detail in Chapter 9.

System date and time
This is the ability to maintain a time-of-day clock and the current date. It allows your programs to set and access the date and time. In addition, your system can react to the passing of time, and you can move certain operations into the time domain. For example, you can set upper limits on the

number of seconds, or milliseconds, that each operation should take, and arrange for emergency action if the operation takes too long.

Logical-to-physical device assignment

CP/M's logical-to-physical device assignment is primitive. With enhancements, you can use any character input/output device as the system console, and output data to several devices at the same time.

Disk input/output

CP/M only knows about the 128-byte sector. Even with the deblocking routines shown in Figure 6-4, overall disk performance can be slow. Performance can be improved dramatically by "track buffering" (in which entire tracks are read and written at one time) or by using a *memory disk* (that is, using large areas of RAM as though they were a disk). These have a cost, though, in increased memory requirements.

Public files

CP/M's user number system needs improvements to function well in conjunction with large hard disks.

Preserving User-Settable Options

A by-product of adding features to the BIOS is that many of these features have options that you can alter, either from the console using a utility program or from within one of your programs.

Each of these options, once set according to your preferences, or to the requirements of your hardware, do not normally change from day to day. Therefore, the BIOS should be designed so that options set by the user can be "frozen" or preserved on the disk by using a utility program, FREEZE. All of the variables recording these options are gathered into a single area and then this area is written out to the disk.

This area is called the *configuration block*. In practice, there are two configuration blocks: one short term and the other long term. The short term block is not preservable — you can set options within it, but they cannot be preserved after you switch your computer off. The system date, for example, is normally set each time you turn your computer on, and therefore is kept in the short term block. The baud rate for your printer, on the other hand, is kept in the long term block so that it can be saved permanently.

An extra BIOS entry point, CBGetAddress, has been built into the enhanced BIOS so that utility programs can locate variables in both configuration blocks. For example, when a utility needs to know where the date is kept in memory, it calls CBGetAddress using a code number (specific for date) in a register. CBGetAddress returns the address of the date in memory. If a new version of the BIOS is produced with the date in a different location, CBGetAddress will still hand the correct, although different, address back to the utility program.

Two other variables that CBGetAddress can access pertain to the configuration block itself. One is the relative address of the start of the long term configuration block. The other is the length of the long term block. These are used by the FREEZE utility when it needs to preserve the long term block on a disk. FREEZE must (1) read in the sectors containing the long term block from the CP/M BIOS image on the reserved area of the disk, (2) copy the current RAM-resident version of the long term block over the disk image version, and then (3) write the sectors back onto the disk.

Figure 8-1 shows how the long term block appears on disk and in memory. The

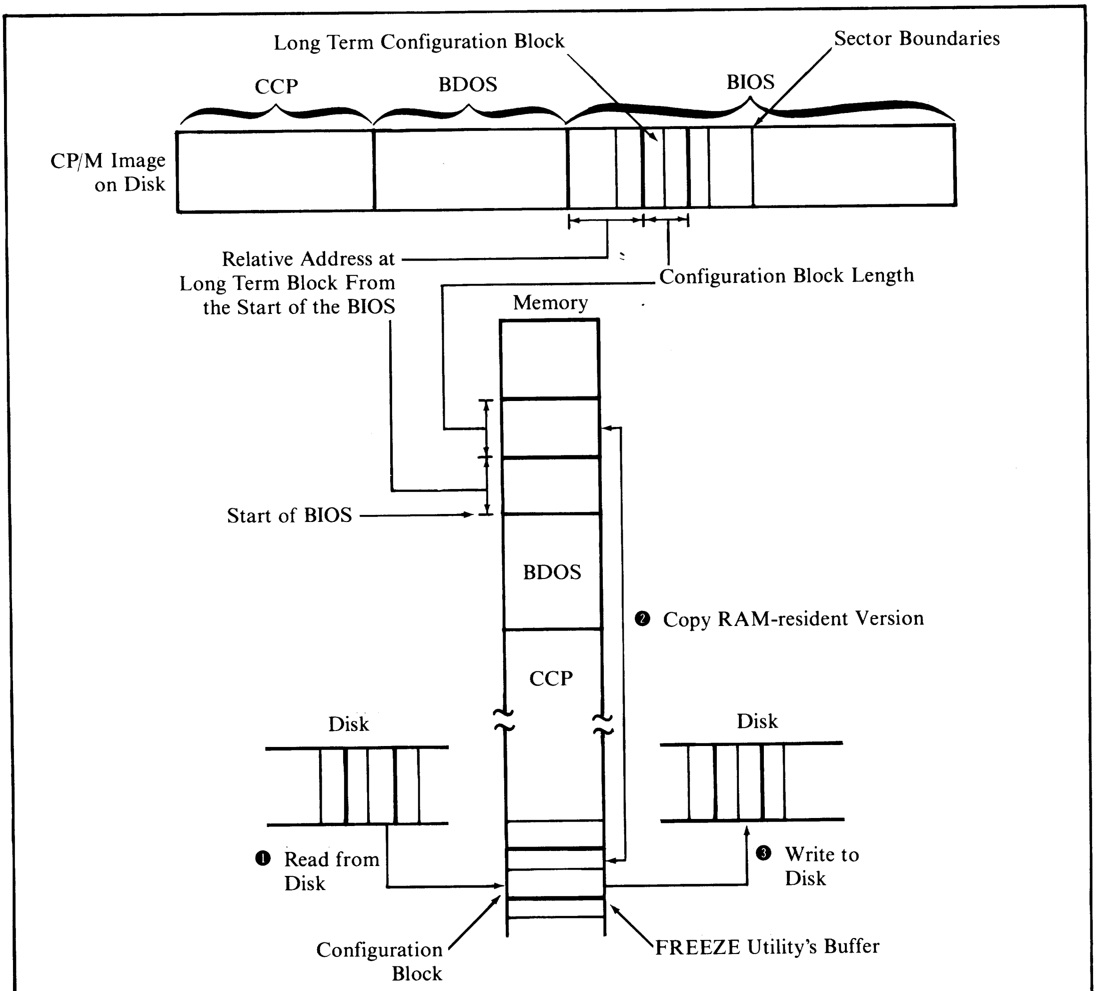

Figure 8-1. Saving the long term configuration block

size of the CCP and BDOS do not change, even if the BIOS does. Therefore, the sector containing the start of the BIOS will not change. The formula (using decimal numbers)

BIOS Start Sector + INT(Relative LTB Address / 128)

then gives the start sector number to be read in. The number of sectors to read is calculated as follows:

(Long Term Block Length + 127) / 128

The relative address and length can be used to locate the long term block in the BIOS executing in RAM.

Character Input/Output

The character I/O drivers shown in the example BIOS, Figure 8-10, have been enhanced to have the following features:

- A single set of driver subroutines controlling all character devices
- Preservation of option settings
- Flexible redirection of input/output between logical and physical devices
- Interrupt-driven input drivers, to get user "type-ahead" capability
- Support of several different protocols to avoid loss of data during high-speed output to printers or other operations
- Forced input of characters into the console input stream, allowing automatic commands at system start-up
- Conversion of terminal function keys into useful character strings
- Ability to recognize "escape sequences" output to the console and to take special action as a result
- Ability to read the current time and date as though they were typed on the console
- "Timeout" signaling when the printer is busy for too long.

Each of these features is discussed in the following sections, as an introduction to the actual code example.

Single Set of Driver Subroutines

In the following examples, only a single set of subroutines is used to process the input and output for all of the physical devices in the system.

This is made possible by grouping all of the individual device's characteristics into a table called the *device table*. For example, in order to get a character from the current console device, the address of its device table will be handed over to the subroutines. These in turn will use the appropriate values from the device table when they need to access a port number or any unique attribute of that device.

In our example, the drivers assume that all of the physical devices use serial input/output. To support a device with parallel input/output, you would need to extend the device table to include a field that would enable the drivers to detect whether they were operating on a serial or parallel device. You would probably also have to add different device initialization and input/output routines more suited to the problems of dealing with a parallel port.

The device table structure consists of a series of equate (EQU) instructions. These define the relative offset of each field in the table. Each definition is expressed by referencing the *preceding* field so that you can insert additional fields without revising the definitions for all the other fields.

Individual instances of device tables are then defined as a series of define byte (DB) and define word (DW) lines. The drivers are given the base address of the device table whenever they need to do something with a device. By adding the base address to the relative address (defined by the equate), the drivers can determine the actual address in memory that contains the required value. The detailed contents of the device table are described later in this chapter.

Permanent Setting of Options

About the only options that need preserving in the long term configuration block are the values used to initialize the hardware chips. Other options can be set during automatic execution of the command file when CP/M is first loaded.

Redirection of Input/Output Between Devices

As you recall, the BDOS only "knows about" the *logical* devices console, reader, punch, and list. Using the IOBYTE at location 0003H in conjunction with the STAT utility, you can redirect the BDOS to assign the logical devices to specific physical devices. However, the redirection provided by CP/M is rather primitive. It permits only four physical devices per logical device. Input and output of a logical device must always come from the same physical device. Output data can only be sent to a single destination, or (using the CONTROL-P toggle) to the console and the list device.

The system in Figure 8-10 supports up to 16 physical devices. Any one of these devices can act as the console, reader, punch, or list device. Input can come from any single device. Output can be sent to any or all of the devices. Each logical device's input and output are separate—that is, console input can come from physical device X while the output can be sent to physical devices Y and Z.

Device redirection can be done dynamically, either from within a program or by using a system utility program. For example, if you have some special input device, your program can momentarily switch over to reading input from this device as though it were the console, and then revert back to reading data from the "real" console.

This redirection scheme is achieved by defining a 16-bit word, called the *redirection word,* in the long term configuration block for each of the following logical devices:

- Console input
- Console output
- Auxiliary (reader/punch) input
- Auxiliary (reader/punch) output
- List input (printers need to send data, too)
- List output.

Each bit in a given redirection word is assigned to a physical device. For input, the drivers use the device corresponding to the first 1 bit that they find in the redirection word. For output, the drivers send the character to be output to all of the devices for which the corresponding bit is set.

The example code does not select a different driver for each bit set — it selects a specific device table and then hands over the base address of this table to the common driver used for all character operations.

Interrupt-Driven Input Drivers

With a standard CP/M BIOS, character data is read from the hardware chips only when control is transferred to the CONIN or READER subroutines. If this character data arrives faster than the BIOS can handle, data overrun occurs and incoming characters are lost.

By using interrupts, the hardware can transfer control to the appropriate interrupt service routine whenever an incoming character arrives. This routine reads the data character and places it into a buffer area to wait for the next CONIN or READER call, which will get the character from the buffer and feed it into the incoming data stream.

User programs and the CCP are "unaware" of this process, perceiving only that data characters are available. However, users will become aware of the process; they will be able to enter data characters from the keyboard before the program is ready for them. This gives the technique its other name — "type-ahead." Although this technique does not alter the speed of execution of any programs running under CP/M, it does create the illusion of greater speed, since pauses while a program accepts data vanish completely. The user can enter data at a rate convenient to the tasks or thoughts at hand, without regard to the rate at which the program can accept that data.

The example contains the code necessary to handle arriving characters under interrupt control. In order to be of general applicability, the code assumes a "flat" interrupt structure: that is, all character input interrupts cause control to be transferred to the same address in memory. The address is determined by the actual hardware interrupt architecture.

The simplest interrupt schemes use the restart (RST) instructions built into the 8080 CPU chip. In the RST scheme, the external hardware interrupts what the CPU chip is doing and forces one of the eight RST instructions into the processor. Each RST instruction causes the processor to execute what is, in effect, a CALL instruction to a predetermined address in memory.

In more complicated systems, a specific interrupt controller chip (such as the Intel 8259A) will be used. In addition to providing very sophisticated (and complicated) prioritization of interrupts, the interrupt controller can transfer control to a *different* address depending on which physical device causes the interrupt. It does this by forcing the CPU to execute a CALL instruction to a different address for each device.

In both architectures, it is the responsibility of the BIOS writer to initialize all the hardware chips so that an interrupt occurs under the correct circumstances. The BIOS writer also must plant instructions at the correct places in memory to receive control from an RST instruction or from the fake CALL instruction emitted by the interrupt controller.

Some hardware requires that the interrupt service subroutine inform it as soon as the interrupt has been serviced and the character has been input. The example drivers provide for this.

This section deals with using interrupts for the *input* drivers, not the output drivers. All of today's microcomputers can output data much faster than external peripherals can handle. After the first few minutes of output, the computer will fill any reasonably sized buffer—and from this point there is no advantage in having a buffered output system. The computer still must slow down to the peripheral's data rate for each character, although now it is waiting to put the character in the output buffer rather than out to the peripheral.

One exception to this is where you have a large amount of "spare" memory and a "slow" printer (which most of them are). Increasing numbers of systems have more than 64K of RAM. The 8080 or Z80 can't address more than this, but a "bank switched" memory system can switch blocks of memory in and out of that 64K address space.

Using this trick, you can access memory "unknown" to CP/M, store some characters in it, switch back to the normal 64K memory, and return control to the caller of the BIOS output routine. When the physical device is ready to accept another output data character from the CPU, it will generate an interrupt. The interrupt service routine then will access the "secret" buffer, output the characters to the device, and switch back to the normal memory.

For example, if you have a printer that prints at 80 characters per second and

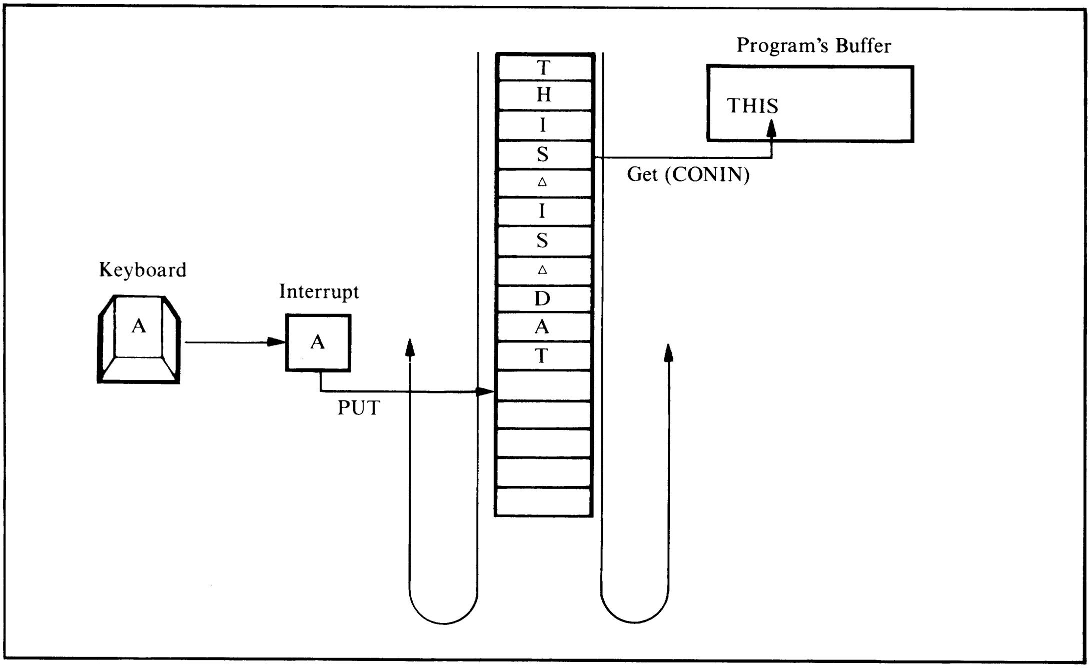

Figure 8-2. Circular buffer type-ahead

you can afford to use 64K of bank switched memory, you can squirrel away 13 minutes of printing—or even more if you design a scheme to compress blanks, storing them in the hidden buffer as a special control sequence.

From the point of view of software, interrupt-driven input drivers are divided into two major groups: the interrupt service routine that reads the characters and stacks them in a buffer, and the non-interrupt routines that get the characters from the buffer and handle the other BIOS functions such as returning console status.

The input character buffer serves as a transfer mechanism between the two groups of subroutines, although the device table also plays an important role.

The example code uses a circular buffer, as shown in Figure 8-2.

The drivers start putting data into the beginning of the buffer. When the last character in the buffer has been reached, the drivers reset to the beginning of the buffer and start over. This, of course, assumes that the non-interrupt drivers have been getting data from the front of the buffer, thus creating space for additional incoming data.

Each device table contains the address of the input buffer, a "put" pointer (for the interrupt service routine), and a "get" pointer (for the non-interrupt service routine). It also contains two character counts: the total number of characters and the number of control characters in the input buffer. You can see how the put and get pointers operate asynchronously. The put pointer is used every time an incoming character generates an interrupt. The get pointer is used for each CONIN call.

The get and put pointers are only single-byte values and are more accurately described as "relative offsets." That is, they contain a value which, when converted to a word and added to the base address of the buffer, will point directly to the appropriate position inside the buffer.

By making the buffer a binary number of characters long — 32 characters, for example — a programming trick can be used to make the buffer appear circular. The device tables contain a mask value formed from the buffer's length minus one (length − 1). Whenever the get or put pointers are incremented by one (to "point" to the next character position), the updated value is ANDed with this (length − 1) mask. In this example, if the get value goes from 31 (the relative address of the last character in the buffer) to 32 (which would be "off the end"), the masking operation will reset it to zero (the relative address of the first character of the buffer). This avoids having to compare pointers to know when to reset them.

It is also simpler to use a count of the number of characters in the buffer, rather than comparing the get and put pointers, to distinguish between an empty and a full buffer. To support different serial protocols, the driver must be able to react when the buffer is within five characters of being full and when it drops below half empty. Both of these conditions are much easier to detect using a simple count that is incremented as a character is put into the buffer and decremented as a character is retrieved from the buffer.

The count of control characters is used to deal with a class of programs that incessantly "gobble" characters, thereby rendering any type-ahead useless. An example is Microsoft's BASIC interpreter. When it is interpreting a program, you can enter a CONTROL-C from the keyboard and the interpreter will come to an orderly stop. It does this by constantly making calls to CONST (console status). If it ever detects an incoming character, it makes a call to CONIN to input the character. A character that is not CONTROL-C is discarded without further ado. Thus, any characters that are input are consumed, destroying the effect of type-ahead.

To deal with this problem, the CONST routine shown in the example can be told to "lie" about the console's status. In this mode, CONST will only indicate that characters are waiting in the input buffer if a control character is received. It uses the control character count to determine whether there are control characters in the buffer; this count is incremented by the interrupt service routine when it detects one, and decremented by the CONIN routine when it gets a control character from the buffer.

Protocol Support

In this context, a protocol is a scheme to avoid loss of data that would otherwise occur if a device sent data faster than the receiving device could handle it. For example, protocols are used to prevent the CPU sending data out to a printer faster than the printer can print the characters and move the paper. The drivers also support input protocols, indicating to a transmitting device when the input buffer gets close to being full.

Two basic methods are used to implement protocols. The first uses the control lines found in the normal RS-232C serial interface cables. For data being output by the computer, the data terminal ready (DTR) signal is used, and for incoming data, the request to send (RTS) signal. These signals conform to the electrical standards for the RS-232C interface; they are considered true when they are at some positive voltage between +3 and +12 volts, and false when they are between −3 and −12 volts.

The second method uses ASCII control characters instead of control signals. Two separate protocols are supported by this method. One uses the ASCII characters XON and XOFF. Before the sending device (the computer or some peripheral device) sends a data character, it checks to see if an XOFF character has been received. If so, the sender will wait for an XON character. The receiving device will only send an XON when it is ready to receive more data.

The second protocol uses the characters ETX (end of transmission) and ACK (acknowledge). This method is normally used only when transmitting data from the computer to a buffered printer. A message length (usually half the printer's buffer size) is defined. When this number of characters has been output, the computer will send an ETX character. No further output will occur until the computer receives an ACK character from the printer.

The example drivers support the DTR high-to-send, the XON/XOFF, and the ETX/ACK protocols for output data. For input, they support RTS high-to-receive and XON/XOFF.

The input protocols are invoked when the input buffer gets within five characters of being full. Then the drivers output an XOFF character or lower the RTS signal voltage, or do both. Only when the input buffer has been emptied to 50% capacity will the drivers send XON or raise the RTS line, or both.

As an emergency measure, if the input buffer becomes completely full, notwithstanding protocols, the drivers will output a predetermined character (defined in the device table) each time they discard an incoming character. This is normally the ASCII BEL (bell) character. When you type too far ahead, the terminal will start beeping to tell you that data is being dropped.

Forced Input into the Console Stream

All application languages provide a means of reading data from the console keyboard. This makes the console input stream a useful gateway to the system. A simple enhancement to the CONIN/CONST routines makes it easy to "fool" the system into acting as if data had been input from the keyboard when in fact the data is coming in from a character string in memory.

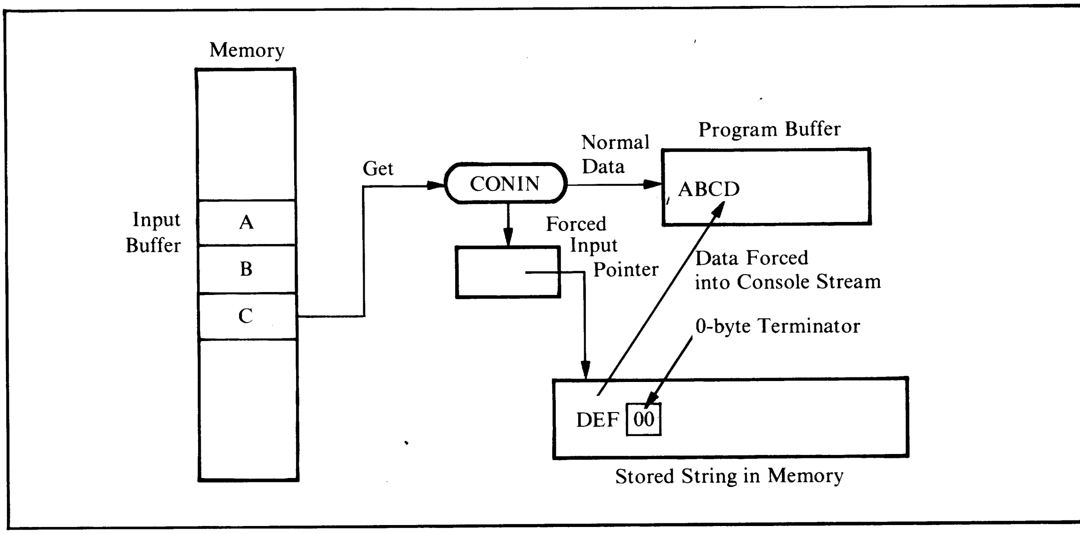

Figure 8-3. CONIN uses forced input data if pointer points to nonzero byte

In the enhanced BIOS, both CONIN and CONST are extended to check a pointer in the long term configuration block, as shown in Figure 8-3.

If this pointer is pointing at a nonzero byte, then that byte is returned as though it had come from the console keyboard. The forced input pointer is then moved up one byte in memory. The process of forcing input continues until a zero byte is encountered.

Forced input serves several purposes. It can be used to force a command or commands into the system when the system first starts up. In conjunction with a utility program, it can allow the user to enter several CP/M commands on a single command line, injecting the characters as each of the commands is executed. It also makes possible the features described in the next two sections.

Support of Terminal Function Keys

Many terminals on the market today have special function keys on their keyboards. When you press one of these keys, the terminal will emit several characters, the first of which is normally the ASCII ESC (escape) character. The remaining one or two characters identify the specific function key that was pressed.

For these function keys to be of any practical use, an applications program must detect the incoming escape sequence and take appropriate action. The problem is that not all terminal manufacturers support the ANSI standard escape sequences.

The example drivers avoid this problem by providing a general-purpose method, shown in Figure 8-4, of detecting escape sequences and of substituting a user-defined character string that is injected into the console input stream as though it had been entered from the keyboard.

This scheme permits function keys to be used very flexibly, even for off-the-shelf programs that have not been designed specifically to accept function key input.

There is, however, one stumbling block. When an ESCAPE character is received, the progam must detect whether this is the start of a function key sequence or the user pressing the ESCAPE key on the terminal's keyboard. In the former case, the

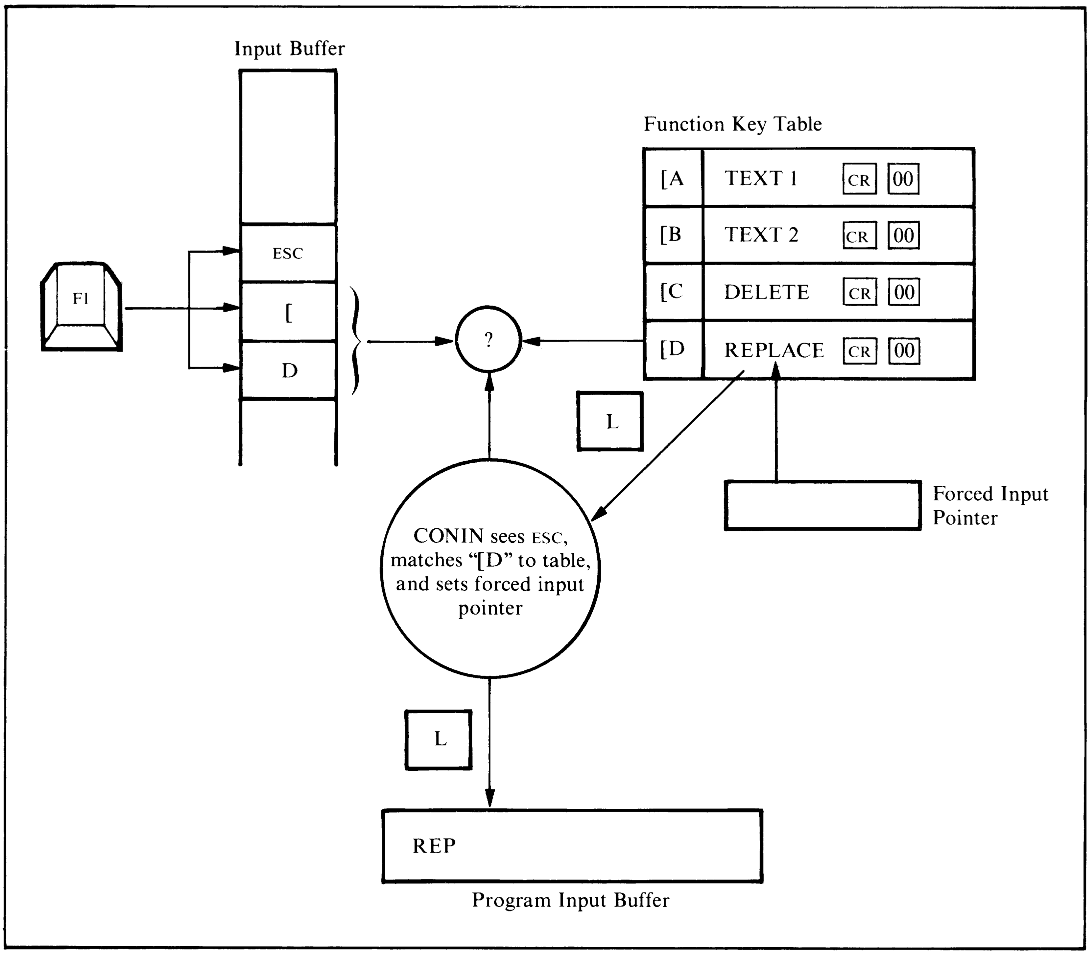

Figure 8-4. CONIN decodes terminal function keys

driver must wait to determine whether a function key string must be substituted for the escape sequence. In the latter case, the driver must input the ESCAPE character as it would other incoming data characters.

This recognition can only be done by moving into the time domain. When the CONIN routine (the non-interrupt routine) gets an ESCAPE character from the input buffer, it delays for approximately 90 milliseconds, enough time for a terminal-generated character sequence to arrive. CONIN then checks the input buffer to see if it contains at least two characters. If it does, the driver checks for a match in a function key table in the long term configuration block. If the characters match a defined function key, then the string associated with the function key will be injected into the console stream by pointing the forced input pointer at it. If the characters do not match anything in the function key table, then the ESCAPE and subsequent characters are handed over as normal data characters.

If after the 90-millisecond delay no further characters have arrived, the ESCAPE character is handed over as a normal character, on the basis that it must have been a manually entered ESCAPE character rather than part of a terminal-generated sequence.

The example drivers show the necessary code and tables for function keys that emit three characters. You could modify them easily for two-character sequences, or, if you are fortunate enough to have a keyboard that uses all eight bits of a byte, to recognize single incoming characters.

Processing Output Escape Sequences

The output side of the console driver, the CONOUT routine, can also be enhanced to recognize escape sequences. It uses a vectored JMP instruction to keep track of the current state of affairs. The CONOUT driver gets an address from the vector and transfers control to it. Normally this vector is set to direct control to the output byte routine. However, if an ESCAPE character is detected in the output stream, the vector is changed to transfer control to a routine that will recognize the character following the ESCAPE. If recognition does not occur, the driver will output an ESCAPE followed by the character that arrived after it.

If the second character is recognized, then the driver can transfer control to the correct escape-sequence processor. This processor can then take whatever action is appropriate. It must also make sure that when all processing is finished, the console output vector is set to process normal output characters again.

This technique is described in more practical detail in the next section, where it is used to preset and read the date and time. You can easily extend the recognition tables in the long term configuration block to perform any special processing that you need, ranging from altering the I/O redirection words to changing any other variable in the system or programming special hardware in your computer.

Be careful not to embed any pure binary values in the sequence of characters going out to the CONOUT routine. If you attempt to send a value of 09H (the TAB character) out via the BDOS, it will gratuitously expand the tab out to some number of blanks. If you need to send out a bit pattern, such as the I/O redirection word, split it up into a series of 7-bit long values. Then send it out with each byte having the most significant bit set to 1. A value of 09H will then become 89H, preventing the BDOS from expanding it to blanks.

Reading Date and Time From Console

For the moment, set aside the question of how the date and time get into the system. Since the date and time are stored in the short term configuration block (there being no need to save them from one work session to the next), all that the BIOS needs to be able to do is recognize a request from an applications program to read either the date or the time and then set the forced input pointer to the appropriate string in memory. Both the date and time strings are terminated by a LINE FEED followed by a 00 byte.

This sequence of events is shown in Figure 8-5.

You can see that the characters "ESC d" output to CONOUT cause it to point the forced input pointer at the date in memory. Subsequent calls to CONIN bring the characters in the date into the program as though they were being entered on the keyboard.

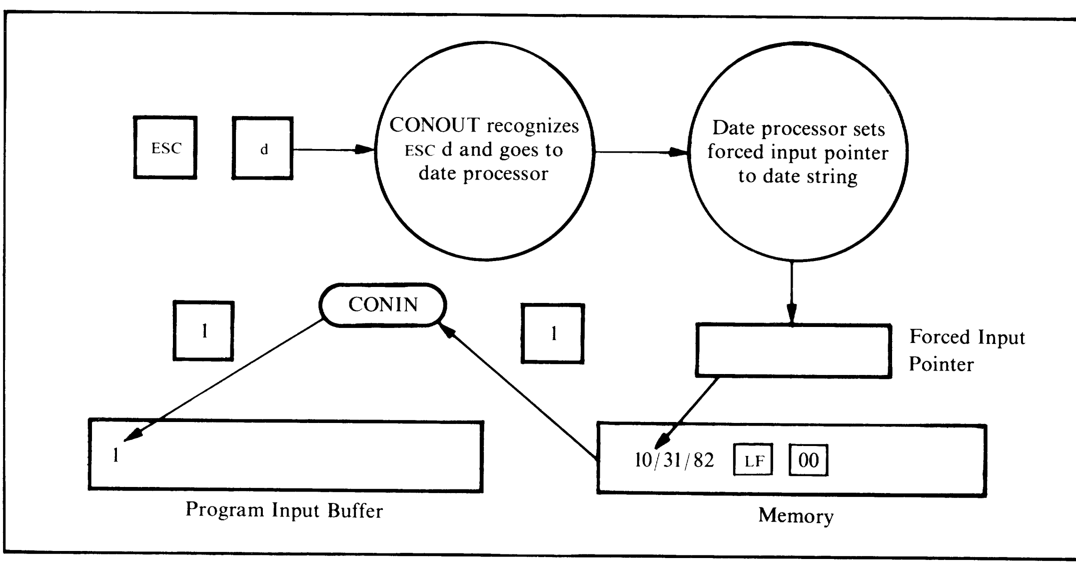

Figure 8-5. Escape sequences sent to CONOUT allow the date to be read by CONIN

"Watchdog" Timeout on Printer

There is no provision in CP/M to deal with a hardware device that for one reason or another is permanently unavailable. Unless special steps are taken in the drivers, the system will screech to a halt in a loop, reading status and testing for the peripheral to be ready.

The example enhancement code shows a scheme, using a real time clock, that can detect when a device such as a printer fails to come ready for more than 30 seconds. On detecting this situation, the code outputs a message to all of the console devices that are not also being used as printers. This type of output is needed to avoid "deadly embraces" where a printer not being ready generates a message that cannot be output because the printer is not ready.

The code that performs the timing function is known as a *watchdog timer*. Each time the real time clock "ticks," the interrupt service routine checks the watchdog count. If the count is nonzero, it is decremented. If the watchdog timer reaches zero, exceeding the time allowed, the drivers will display a message on the console indicating that the printer has been busy for too long. The user then has the option of making the printer ready and trying again to output data, ignoring the error and carrying on, or aborting the program by doing a BDOS System Reset (function 0).

Although sending an error message to the console sounds simple, it is complicated if console output is directed to the offending printer itself. The drivers attempt to solve this problem by sending the message only to those devices being used as consoles and *not* as printers. If all consoles are being used as printer devices as well, the driver will send the message to device 0—normally the main console.

Keeping Time and Date

CP/M does not have provision for keeping the current time and date in the system. The example enhancement shows how to keep the time of day and the current date in the short term configuration block by using escape sequences output to the console (1) to set them to the correct values and (2) to "read" them from the console input stream.

The example presupposes that the system has a hardware chip that can be programmed to generate an interrupt every 1/60th of a second (16.666 milliseconds). This provides a divide-down counter to measure seconds elapsed. Of course, if your computer has a *true* real time clock that you can read and get the current time in hours, minutes, and seconds, your code will be very simple. You still will need to have the clock generate a periodic interrupt, however, in order to use the watchdog feature for timing printer and disk operations.

Actual time is kept as ASCII characters, using another ASCII control table to determine when "carry and reset to zero" should occur. By changing two bytes in this table, the time can be kept in 12- or 24-hour format.

The date is simply stored as a string. The example code does not attempt to make sure that the date is valid, nor to update when midnight rolls around. This could be done easily by the BIOS — but it would take a fairly large amount of code.

Watchdog Timer

Having a periodic source of interrupts also opens the door to building in an emergency or watchdog timer. This is nothing more than a 16-bit counter. Each time the real time clock interrupts, or ticks, the interrupt service routine checks the watchdog count. If it is already at zero, nothing more happens — the watchdog is not in use. If it is nonzero, the routine decrements the count by one. If this results in a zero value, the interrupt service routine CALLs a predetermined address. This will be the address of some emergency interrupt service routine that can then take special action, such as investigating the cause of the timeout.

The watchdog routine has a non-interrupt-level subroutine associated with it. Calling this set watchdog subroutine provides a means of setting the count to a predetermined number of real time clock "ticks" and setting the address to which control should be transferred if the count reaches zero.

Having called the set watchdog subroutine, the driver can then sit in a status loop, with interrupts enabled, waiting for some event to occur. If the event happens before the watchdog count hits zero, the driver must call the set watchdog routine again to set the count back to zero, thereby disabling the watchdog mechanism.

The watchdog timer can be used to detect printers that are busy for too long or disk drives that take too long to complete an action either because of a hardware failure or because the user has not loaded the disk into the drive.

Data Structures

As already stated, each character I/O device has its own device table that describes all of its unique characteristics.

The other major data structure is the configuration blocks — both short and long term.

This section describes each field in these data structures.

Device Table

Figure 8-6 shows the contents of a device table. More correctly, it shows a series of equates that define the offsets of each field in the device table. The drivers are given the base address of a specific device table. They then access each field by adding the required offset to this base address.

The first part of the device table is devoted to the physical aspect of the device, defining which port numbers are to be used to communicate with it. The drivers need to know several different port numbers since each one is used for a particular

```
                ;       The drivers use a device table for each
                ;       physical device they service. The equates that follow
                ;       are used to access the various fields within the
                ;       device table.
                ;
                ;                           Port numbers and status bits
0000 =          DT$Status$Port              EQU     0         ;Device status port number
0001 =          DT$Data$Port                EQU     DT$Status$Port+1
                                                              ;Device data port number
0002 =          DT$Output$Ready             EQU     DT$DataPort+1
                                                              ;Output ready status mask
0003 =          DT$Input$Ready              EQU     DT$Output$Ready+1
                                                              ;Input ready status mask
0004 =          DT$DTR$Ready                EQU     DT$Input$Ready+1
                                                              ;DTR ready to send mask
0005 =          DT$Reset$Int$Port           EQU     DT$DTR$Ready+1
                                                              ;Port number used to reset an
                                                              ;  interrupt
0006 =          DT$Reset$Int$Value          EQU     DT$Reset$Int$Port+1
                                                              ;Value output to reset interrupt
0007 =          DT$Detect$Error$Port        EQU     DT$Reset$Int$Value+1
                                                              ;Port number for error detect
0008 =          DT$Detect$Error$Value       EQU     DT$Detect$Error$Port+1
                                                              ;Mask for detecting error (parity etc.)
0009 =          DT$Reset$Error$Port         EQU     DT$Detect$Error$Value+1
                                                              ;Output to port to reset error
000A =          DT$Reset$Error$Value        EQU     DT$Reset$Error$Port+1
                                                              ;Value to output to reset error
000B =          DT$RTS$Control$Port         EQU     DT$Reset$Error$Value+1
                                                              ;Control port for lowering RTS
000C =          DT$Drop$RTS$Value           EQU     DT$RTS$Control$Port+1
                                                              ;Value, when output, to drop RTS
000D =          DT$Raise$RTS$Value          EQU     DT$Drop$RTS$Value+1
                                                              ;Value, when output, to raise RTS
                ;
                ;               Device logical status (incl. protocols)
000E =          DT$Status                   EQU     DT$Raise$RTS$Value+1
                                                              ;Status bits
0001 =          DT$Output$Suspend           EQU     0000$0001B        ;Output suspended pending
                                                                      ;  protocol action
0002 =          DT$Input$Suspend            EQU     0000$0010B        ;Input suspended until
                                                                      ;  buffer empties
0004 =          DT$Output$DTR               EQU     0000$0100B        ;Output uses DTR-high-to-send
0008 =          DT$Output$Xon               EQU     0000$1000B        ;Output uses Xon/Xoff
0010 =          DT$Output$Etx               EQU     0001$0000B        ;Output uses Etx/Ack
0020 =          DT$Output$Timeout           EQU     0010$0000B        ;Output uses Timeout
0040 =          DT$Input$RTS                EQU     0100$0000B        ;Input uses RTS-high-to-receive
0080 =          DT$Input$Xon                EQU     1000$0000B        ;Input uses Xon/Xoff
                ;
000F =          DT$Status$2                 EQU     DT$Status+1       ;Secondary status byte
0001 =          DT$Fake$Typeahead           EQU     0000$0001B        ;Requests Input$Status to
                                                                      ;  return "Data Ready" when
                                                                      ;  control characters are in
                                                                      ;  input buffer
                ;
0010 =          DT$Etx$Count                EQU     DT$Status$2+1
                                                              ;No. of chars. sent in Etx protocol
0012 =          DT$Etx$Message$Length       EQU     DT$Etx$Count+2
                                                              ;Specified message length
                ;
                ;                           Input buffer values
0014 =          DT$Buffer$Base              EQU     DT$Etx$Message$Length+2
                                                              ;Address of input buffer
0016 =          DT$Put$Offset               EQU     DT$Buffer$Base+2
                                                              ;Offset for putting chars. into buffer
0017 =          DT$Get$Offset               EQU     DT$Put$Offset+1
                                                              ;Offset for getting chars. from buffer
0018 =          DT$Buffer$Length$Mask       EQU     DT$Get$Offset+1
                                                              ;Length of buffer - 1
                                                              ;Note: Buffer length must always be
                                                              ;  a binary number; e.g. 32, 64, or 128,
                                                              ;This mask then becomes:
                                                              ;  32 ->  31 (0001$1111B)
                                                              ;  64 ->  63 (0011$1111B)
                                                              ; 128 -> 127 (0111$1111B)
```

Figure 8-6. Device table equates

```
                                                         ;After the get/put offset has been
                                                         ; incremented it is ANDed with the mask
                                                         ; to reset it to zero when the end of
                                                         ; the buffer has been reached.
0019 =          DT$Character$Count         EQU    DT$Buffer$Length$Mask+1
                                                         ;Count of the number of characters
                                                         ;  currently in the buffer
001A =          DT$Stop$Input$Count        EQU    DT$Character$Count+1
                                                         ;Stop input when the count reaches
                                                         ;  this value
001B =          DT$Resume$Input$Count      EQU    DT$Stop$Input$Count+1
                                                         ;Resume input when the count reaches
                                                         ;  this value
001C =          DT$Control$Count           EQU    DT$Resume$Input$Count+1
                                                         ;Count of the number of control
                                                         ;  characters in the buffer
001D =          DT$Function$Delay          EQU    DT$Control$Count+1
                                                         ;Number of clock ticks to delay to
                                                         ;  allow all characters after function
                                                         ;  key lead-in to arrive
001E =          DT$Initialize$Stream       EQU    DT$Function$Delay+1
                                                         ;Address of byte stream necessary to
                                                         ;  initialize this device
```

Figure 8-6. Device table equates (continued)

function. Depending upon your hardware, each port number could be different; however, with standard Intel or Zilog chips, you will often find that the same port number is used for several functions. The drivers also need to know what bit patterns to expect when they read some ports and what values to output to ports in order to obtain particular results.

The layout of the device table and the manner in which the equates are declared are designed to make it easy for you to change the contents of the table to meet your own special requirements. The fields in this first section of the device table are discussed in the sections that follow.

DT$Status$Port The driver reads this port to determine whether the hardware chip has incoming data ready to be input to the computer or whether the chip is capable of accepting another data character for output to the physical device.

DT$Data$Port The driver reads from this port to access the next data character from the physical device. The driver also writes to this port to output the next data character to the device.

If your computer hardware requires that the input data port be a different number from the output data port, you will have to alter the coding in the device table equates as well as make the necessary changes in the input and output subroutines in the body of the code.

DT$Output$Ready This is the bit mask that the driver will AND with the current device status (obtained by reading the DT$Status$Port) to see whether the device is ready to accept another output character. It assumes that the device is ready if the result of the AND instruction is nonzero. You may have to change some JNZ (jump nonzero) instructions to JZ (jump zero) instructions if your hardware device uses inverted logic, with bits in the status byte set to 0 to indicate that the device can accept another character for output.

Note that this status check relates only to the output chip — it is completely separate from the question of whether the peripheral itself is ready to accept data.

DT$Input$Ready This is the bit mask that the driver will AND with the current device status to see if there is an incoming data character. The drivers again presume that if the result of the AND is nonzero, then an incoming data character is waiting to be read from the data port. You will need to make changes similar to those for the output subroutines described in the previous section if your hardware uses inverted logic (0 bit means incoming data).

DTDTRReady DTR stands for *data terminal ready*. It refers to one of the control lines connected from the actual peripheral device to the I/O chip (via several other integrated circuits). The drivers, as an option, will only output data to the device when the DTR signal is at a positive voltage. If the peripheral, in order to stop the flow of data characters being output to it, lowers the DTR signal to a negative voltage, the drivers will wait. Once DTR goes positive again, the drivers will resume sending data. Many hard-copy devices use this scheme to give themselves a chance to print out data received from the computer. They may have to lower DTR for several seconds, while they perform paper movement, for example.

The value in this field is a bit mask that the drivers use on the device status to determine the state of the data-terminal-ready control signal.

DT$Reset$Int$Port Since the input side of the drivers uses interrupts, when an incoming character is ready to be input by the CPU, the hardware generates an interrupt signal, and control is transferred to the interrupt service routine. This routine "services" the interrupt by reading the incoming data character, saving it in memory, and then transferring control back to whatever was being executed when the interrupt occurred.

The more complicated interrupt controller chips (such as the Intel 8259A) must be told as soon as a given interrupt has been serviced so that they can permit servicing of any lower priority interrupts that may be waiting.

This field contains the port number that will be used to "reset" the interrupt, or more correctly, to indicate the end of the previous interrupt's servicing.

DT$Reset$Int$Value This is the value that will be output to the DT$Reset$Int$Port to tell the hardware that the previous interrupt service has been completed.

DT$Detect$Error$Port Before the driver attempts to read any incoming data from the DT$Data$Port, it checks to see if any hardware errors have occurred. It does so by reading status from this port.

DT$Detect$Error$Value The status byte that is input from the DT$Detect$Error$Port is ANDed with this value. If the result is nonzero, the driver assumes that an error has occurred.

DT$Reset$Error$Port If an error has occurred, the driver outputs an error reset value to this port number.

DT$Reset$Error$Value This is the value that will be output to the DT$Reset$Error$Port to reset an error.

DTRTSControl$Port The drivers use this port number to control the request-to-send line if the RTS protocol option is selected.

DT$Drop$RTS$Value This value is output to the RTS control port to lower the RTS line so that some external device will stop sending data to the computer.

DT$Raise$RTS$Value This value is output to raise the RTS line so that the external device will resume sending data to the computer.

DT$Status This is the first of two status bytes. It contains bit flags that are set to a 1 bit to indicate the following conditions:

DT$Output$Suspend
Because of protocol, the device is currently suspended from receiving any further output characters.

DT$Input$Suspend
Because of protocol, the device has been requested not to send any more input characters.

DT$Output$DTR
The driver will maintain DTR-high-to-send protocol for output data.

DT$Output$Xon
The driver will maintain XON/XOFF protocol for output data.

DT$Output$Etx
The driver will maintain ETX/ACK protocol for output data.

DT$Input$RTS
The driver will maintain RTS-high-to-receive protocol for input data.

DT$Input$Xon
The driver will maintain XON/XOFF protocol for input data.

DT$Status$2 This is another status byte, also with the following bit flag:

DT$Fake$Typeahead
CONST will “lie” about the availability of incoming console characters. It

will only indicate that data is waiting if there are control characters other than CARRIAGE RETURN, LINE FEED, or TAB in the input buffer.

DTEtxCount This value is only used for ETX/ACK protocol. It is a count of the number of characters sent in the current message. When this count reaches the defined message length, then the driver will send an ETX character and suspend any further output.

DTEtxMessage$Length This value is the defined message length for the ETX/ACK protocol. It is used to reset the DTEtxCount.

DT$Buffer$Base This is the address of the first byte of the device's input buffer.

DTPutOffset This *byte* contains the relative offset indicating where the next incoming character is to be "put" in the input buffer. This byte must then be converted into a word value and added to the DT$Buffer$Base address to get the absolute memory location.

DTGetOffset This byte contains the relative offset indicating where the next character is to be "got" in the input buffer.

DT$Buffer$Length$Mask This byte contains the length of the buffer minus one. The length of the buffer must always be a binary number (8, 16, 32, 64...). Therefore, one less than the length forms a mask value. Both the get and put offsets, after being incremented, are masked with this value. When the offset reaches the end of the buffer, this masking operation will "automatically" reset the offset to zero.

DT$Character$Count This is a count of the total number of characters in the buffer. It is incremented by the interrupt service routine each time a character is placed in the buffer, and decremented by the CONIN routine each time it gets a character from the buffer.

CONST uses this value to determine whether any characters are available for input.

DT$Stop$Input$Count When the interrupt service routines detect that the DT$Character$Count is equal to this value (normally buffer length minus five), the drivers will invoke the selected input protocol, lowering RTS or sending XOFF, to shut off the incoming data stream.

DT$Resume$Input$Count When the CONIN routine detects that the DT$Character$Count has become equal to this value, the drivers will again invoke the selected input protocol, either raising RTS or sending XON to resume receiving input data.

DT$Control$Count This is a count of the number of control characters in the input buffer. CARRIAGE RETURN, LINE FEED, and TAB characters are not included in this count.

It is incremented by the interrupt service routine and decremented by CONIN. CONST uses the count when the DT$Fake$Typeahead mode is active; it will only indicate that characters are waiting in the input buffer if the control count is nonzero.

DT$Function$Delay This is the number of clock ticks that should be allowed to elapse after the first character of an incoming escape sequence has been detected. It allows time for the remaining characters in the escape sequence to arrive, assuming that these are being emitted by a terminal at maximum baud rate. Normally, this will correspond to a delay of approximately 90 milliseconds.

DT$Initialize$Stream This is the address of the first byte of a string. This string has the following format:

```
DB  ppH         Port number
DB  nnH         Number of bytes to be output
DB  vvH,vvH...  Initialization bytes to be output to the specified port number
```

This sequence can be repeated as many times as is necessary, with a "port" number of 00H acting as a terminator.

Disk Input/Output

The example drivers show three main disk I/O enhancements:

- Full track buffering
- Using memory as an ultra-fast disk
- Improved error handling.

Full Track Buffering

The 5 1/4″ diskettes used in the example system are double-sided. Each side has a separate read/write head in the disk drive. The disk controller is fast enough that, if so commanded, it can read in a complete track's worth of data from one side of the diskette in a single revolution of the diskette.

The drivers have been modified to do just this. The main disk buffer has been dramatically enlarged to accommodate nine 512-byte sectors.

In the earlier standard BIOS, CP/M was configured for tracks of 18 512-byte sectors. The data from each head on a given track was laid "end-to-end" to create the illusion of a single surface with twice as much data on it. For track buffering, performance would be reduced if each read required two revolutions of the diskette, and so in this BIOS the tables and the low-level driver logic have been changed. Each surface is separated, with even numbered tracks on head 0, odd on head 1.

The track number given to the low-level drivers serves two purposes. The least significant bit identifies the head number. When the track number is shifted one bit right, the result is the *physical* track number to which the head assembly must be positioned.

The deblocking algorithm has also been modified by deleting references to sectors. The code is now concerned only with whether the correct disk and track are in the buffer. If this is true, the correct sector must, by definition, be in the buffer.

The deblocking code no longer takes any note when the BDOS indicates that it is writing to an unallocated allocation block—knowledge it used to bypass a sector preread in the standard BIOS. The track size in this enhanced BIOS is much larger than an allocation block, and so the question is meaningless; the whole track must be preread to write just a single sector.

This enhancement really excels when the BDOS is doing directory operations, which always involve a series of sequential reads. The entire directory can be brought into memory, updated, and written back in just two disk revolutions.

One point to watch out for is what is known as "deferred writes." Imagine a program instructed to write on a sector on track 20. The drivers will read in track 20, copy the contents of the designated sector into the track buffer, and return to the program *without* actually writing the data to the disk. The program could "write" to all of the sectors on this track without any actual disk writes. During all this time, this data would exist only in memory and not on the disk drive, so if a power failure occurred, several thousand bytes of data would be lost. Writing to the directory is an exception. The drivers always physically write to the disk when the BDOS indicates that it is writing to a directory sector.

In reality, the increased risk is small. Most programs are constantly reading and writing files, so that the track buffer will be written out frequently in order to read in another track. When programs end, they close output files. This in turn triggers directory writes that force data tracks onto the disk.

If high security is a requirement for your computer, you could extend the watchdog routine to include another separate timer. You could preset this timer for, say, a ten-second delay each time you write into the track buffer but do not write the buffer to the disk. When the count expires, it would set a flag that could be tested by all of the BIOS entry points. If set, they would initiate a write of the track buffer to the disk.

Using Memory as an Ultra-Fast Disk

As you can see from the preceding section, increased performance tends to go hand in hand with increased memory requirements. This is certainly true with a "memory disk," commonly called a RAM-disk or M-disk. In fact, to have an M-disk with reasonable storage capacity, your computer must have at least 128K bytes of additional memory.

Since the 8080 or Z80 can only address 64K of memory at one time, to get access to any of this additional memory, some part of your computer's "normal" memory must be removed from the 64K address space and the additional memory must be switched in. This is known as bank-switched memory.

Figure 8-7 shows the memory organization that is supported by the example M-disk drivers.

You can see that the system has a total of 256K bytes of RAM, organized with the top 16K, from 64K down to 48K, being "common"—that is, switched into the address space all the time. The lower 48K can be selected from five banks, numbered 0 to 4. Bank 0 is switched in for normal CP/M operations.

The M-disk parameter blocks describe a disk with eight "tracks," numbered 0 to 7. The least significant bit of the track number determines whether the base address of the track will be 0000H or 6000H. Shifting the track number right one bit gives the bank number. Each track consists of 192 sectors. To get the relative address of a sector within its "track," shift the sector number eight bits left, thus multiplying it by 128.

The M-disk is referenced by logical disk M:. A few special-case instructions are required to return the special M-disk parameter header in SELDSK.

One problem, fortunately easily solved, is that the user's DMA address coexists in the address space with the M-disk image itself. There is no direct way to move data between bank 0 and any other bank. The M-disk uses an intermediary buffer in common memory (above 48K), moving data into this, switching banks, and then moving the data down again. Figure 8-8 shows an example of this sequence, as used when reading from the M-disk.

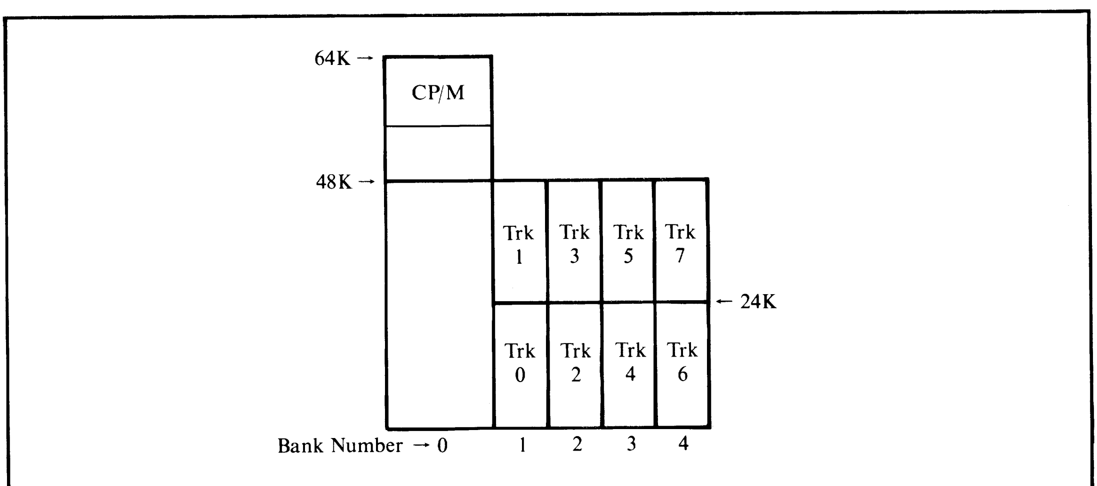

Figure 8-7. Memory organization for M-disk

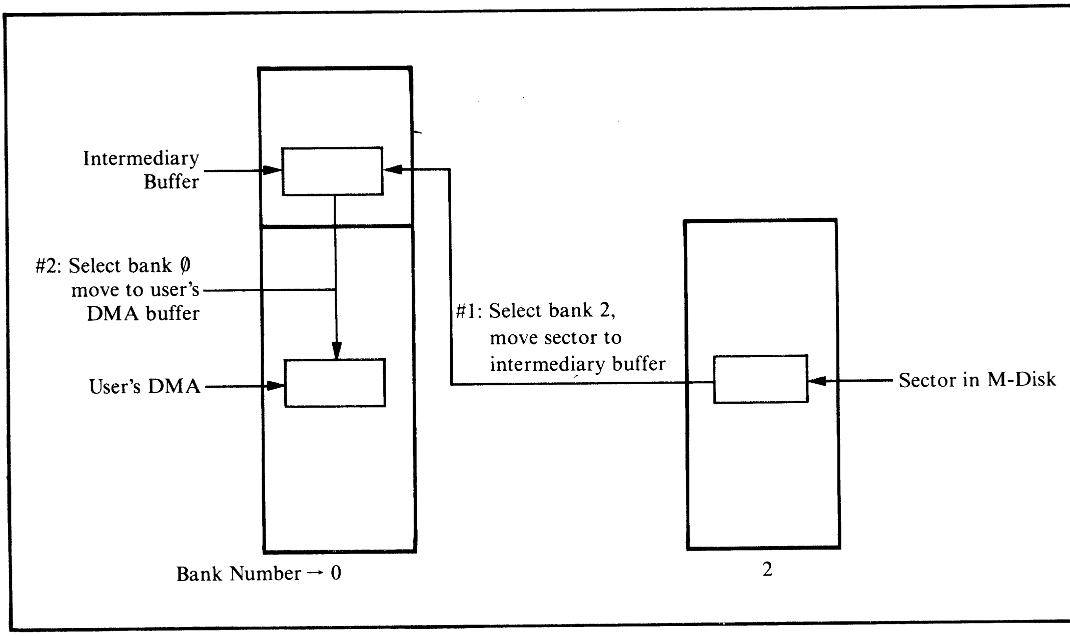

Figure 8-8. Reading a sector from the M-disk image

During cold boot initialization, the M-disk driver checks the very first directory entry (in bank 1) to see if it matches a dummy entry for a file called "M$Disk." If this entry is present, the M-disk is assumed to contain valid information. If the entry is absent, the initialization code makes this special directory entry and fills the remainder of the directory with 0E5H, making it appear empty. The dummy entry makes it appear that the "M$Disk" file is in user 15, marked System status and Read-Only—all of which are designed to prevent its accidental erasure.

Custom Patches to CP/M

Two features shown in the enhanced BIOS, one in the CCP and one in the BDOS, require changes to CP/M itself. These features are implemented by modifying the CCP and BDOS to transfer control to the BIOS at specific points, execute a few instructions in the BIOS, and then return to CP/M. The patches could be made by modifying the MOVCPM program to install the changes permanently. The changed version of MOVCPM, however, *must* be used with a specific version of the BIOS. Therefore, patching CP/M "on the fly" ensures that there will be no mismatch between the BIOS and the rest of CP/M.

Both of these patches were produced with the assistance of Digital Research.

User 0 Files Made Public

The first change permits files created in user area 0 to be accessible from all other user numbers. This feature comes into its own only with hard disk systems. On a hard disk, user numbers can partition the disk, but the frequently used utilities must then be duplicated in each user area. Allowing files in user area 0 to be public means that these files will be accessible from all the other user numbers. Hence the files need not be copied into each user area.

The public files feature alters the way that the BDOS performs the Search Next function, allowing access to files declared in user area 0 even when the current user number is not 0. However, the feature is a double-edged sword—user 0 files can be accidentally erased or damaged as well as accessed. Therefore, user 0 files should be declared as System status and Read-Only to protect them. As an additional precaution, public files can be turned off by a control flag in the long term configuration block. This flag is set to an initial state that disables public files.

Modified User Prompt

This modification makes the CCP display the current user number as well as the default disk. For example,

```
3B>
```

indicates that you are currently in user number 3, with disk B: as the default. In addition, if you have enabled public files, the prompt is preceded by the letter "P" to serve as a reminder:

```
P3B>
```

An Enhanced BIOS

The remainder of this chapter consists of the assembly language source code for the enhanced BIOS described here. It is rather a daunting listing, but will be well worth your study. The copious commentary has been written to make this study easier, and emphasis has been placed on explaining *why* as well as *what* things are done.

As with the standard BIOS, each line is numbered so that you can use the functional index in Figure 8-9 to find areas of interest in the listing. Note that the line numbers are not contiguous. They jump several hundred at the start of each major section or subroutine. This facilitates minor changes in the listing without revision of the functional index. The full listing is given in Figure 8-10.

Start Line	Functional Component or Routine
00001	Introductory Comments and Equates
00200	BIOS Jump Table with Additional Private Entries
00400	Long Term Configuration Block
00800	Interrupt Vector
00900	Device Port Numbers and Other Equates
01100	Display$Message Subroutine
01200	Enter$CPM Setup
01300	Device Table Equates
01500	Device Table Declarations
01700	General Device Initialization
01800	Specific Device Initialization
02000	Output Byte Stream
02100	CONST Routine
02200	CONIN Routine with Function Key Processing
02500	Console Output
02700	CONOUT Routine with Escape Sequence Processing
02900	AUXIST—Auxiliary Input Status Routine
03000	AUXOST—Auxiliary Output Status Routine
03100	AUXIN—Auxiliary Input Routine
03200	AUXOUT—Auxiliary Output Routine
03300	LISTST—List Status Routine
03400	LIST—List Output Routine
03500	Request User Choice—Request Action After Error
03600	Output Error Message
03656	Get Composite Status from Selected Output Devices
03800	Multiple Output of Byte to All Output Devices
04000	Check Output Device Logically (Protocol) Ready
04200	Process ETX/ACK Protocol
04400	Select Device Table from I/O Redirection Bit Map
04600	Get Input Character from Input Buffer
04800	Introductory Comments for Interrupt-Driven Drivers
04900	Character Interrupt Service Routine
05000	Service Device—Puts Character into Input Buffer
05300	Get Address of Character in Input Buffer
05400	Check if Control Character (not CR, LF, TAB)
05500	Output Data Byte
05700	Input Status Routine
05900	Set Watchdog Timer Routine
06000	Real Time Clock Interrupt Service Routine
06200	Shift HL Right One Bit Routine
06300	Introductory Comments for High-Level Disk Drivers
06400	Disk Parameter Headers
06600	Disk Parameter Blocks
06800	SELDSK—Select Disk Routine
07000	SETTRK—Set Track Routine
07100	SETSEC—Set Sector Routine

Figure 8-9. Functional index for listing in Figure 8-10

FIGURE 8-9. Functional index for listing in Figure 8-10 (continued)

```
                00001   ;       This is a skeletal example of an enhanced BIOS.
                00010   ;       It includes fragments of the standard BIOS
                00011   ;       shown as Figure 6-4 in outline, so as to
                00012   ;       avoid cluttering up the enhancements with the
                00013   ;       supporting substructure. Many of the original
                00014   ;       comment blocks have been abbreviated or deleted
                00015   ;       entirely.
                00016   ;
                00017   ;< --   NOTE:   The line numbers at the left are included
                00018   ;               to allow reference to the code from the text.
                00019   ;               There are deliberate discontinuities in the
                00020   ;               numbers to allow space for expansion.
                00021   ;
3030 =          00022   VERSION         EQU     '00'    ;Equates used in the sign-on message
3230 =          00023   MONTH           EQU     '02'
3632 =          00024   DAY             EQU     '26'
3338 =          00025   YEAR            EQU     '83'
                00026   ;
                00027   ;*******************************************************************
                00028   ;*                                                                 *
                00029   ;*      This BIOS is for a computer system with the following      *
                00030   ;*      hardware configuration :                                   *
                00031   ;*                                                                 *
                00032   ;*              -- 8080 CPU                                        *
                00033   ;*              -- 64K bytes of RAM                                *
                00034   ;*              -- 3 serial I/O ports (using signetics 2651) for:  *
                00035   ;*                 console, communications and list                *
                00036   ;*              -- Two 5 1/4" mini floppy, double-sided, double-   *
                00037   ;*                 density drives. These drives use 512-byte sectors. *
                00038   ;*                 These are used as logical disks A: and B:.      *
                00039   ;*                 Full track buffering is supported.              *
```

Figure 8-10. Enhanced BIOS listing

```
                ;*          -- Two 8" standard diskette drives (128-byte sectors)   *
                ;*             These are used as logical disks C: and D:.           *
                ;*          -- A memory-based disk (M-disk) is supported.           *
                ;*                                                                  *
                ;*             Two intelligent disk controllers are used, one for   *
                ;*             each diskette type. These controllers access memory *
                ;*             directly, both to read the details of the            *
                ;*             operations they are to perform and also to read      *
                ;*             and write data from and to the diskettes.            *
                ;*                                                                  *
                ;*                                                                  *
                ;*******************************************************************
                ;       Equates for characters in the ASCII character set
                ;
0011 =          XON     EQU     11H     ;Reenables transmission of data
0013 =          XOFF    EQU     13H     ;Disables transmission of data
0003 =          ETX     EQU     03H     ;End of transmission
0006 =          ACK     EQU     06H     ;Acknowledge
000D =          CR      EQU     0DH     ;Carriage return
000A =          LF      EQU     0AH     ;Line feed
0009 =          TAB     EQU     09H     ;Horizontal tab
0007 =          BELL    EQU     07H     ;Sound terminal's bell
                ;
                ;
                ;       Equates for defining memory size and the base address and
                ;       length of the system components
                ;
0040 =          Memory$Size     EQU     64      ;Number of Kbytes of RAM
                ;
                ;       The BIOS length must be determined by inspection.
                ;       Comment out the ORG BIOS$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.
                ;
2500 =          BIOS$Length     EQU     2500H   ;<-- Revised to an approximate value
                                                ;    to reflect enhancements
                ;
0800 =          CCP$Length      EQU     0800H   ;Constant
0E00 =          BDOS$Length     EQU     0E00H   ;Constant
                ;
000F =          Overall$Length  EQU     (CCP$Length + BDOS$Length + BIOS$Length + 1023) / 1024
                ;
C400 =          CCP$Entry       EQU     (Memory$Size - Overall$Length) * 1024
CC06 =          BDOS$Entry      EQU     CCP$Entry + CCP$Length + 6
DA00 =          BIOS$Entry      EQU     CCP$Entry + CCP$Length + BDOS$Length
                ;
0005 =          BDOS            EQU     0005H   ;BDOS entry point (used for making
                                                ;  system reset requests)
                ;
                ;#
                ;       ORG     BIOS$Entry      ;Assemble code at BIOS address
                ;
                ;       BIOS jump vector
                ;
0000 C31311             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 in location
                                        ;  0001H and 0002H of the base page
0003 C3750E             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
0006 C32D03             JMP     CONST   ;Console status -- returns A = 0FFH if there is a
                                        ;  console keyboard character waiting
0009 C33A03             JMP     CONIN   ;Console input -- returns the next console keyboard
                                        ;  character in A
000C C3D703             JMP     CONOUT  ;Console output -- outputs the character in C to
                                        ;  the console device
000F C3F504             JMP     LIST    ;List output -- outputs the character in C to the
                                        ;  list device
0012 C3CE04             JMP     AUXOUT  ;Auxiliary output -- outputs the character in C to the
                                        ;  logical auxiliary device
```

Figure 8-10. (Continued)

```
0015 C3A104           JMP     AUXIN   ;Auxiliary input -- returns the next input character from
                                      ;  the logical auxiliary device in A
0018 C3160A           JMP     HOME    ;Homes the currently selected disk to track 0
001B C36309           JMP     SELDSK  ;Selects the disk drive specified in register C and
                                      ;  returns the address of the disk parameter header
001E C39B09           JMP     SETTRK  ;Sets the track for the next read or write operation
                                      ;  from the BC register pair
0021 C3A109           JMP     SETSEC  ;Sets the sector for the next read or write operation
                                      ;  from the A register
0024 C3A809           JMP     SETDMA  ;Sets the direct memory address (disk read/write)
                                      ;  address for the next read or write operation
                                      ;  from the DE register pair
0027 C3370A           JMP     READ    ;Reads the previously specified track and sector from
                                      ;  the selected disk into the DMA address
002A C34B0A           JMP     WRITE   ;Writes the previously specified track and sector onto
                                      ;  the selected disk from the DMA address
002D C3D704           JMP     LISTST  ;Returns A = 0FFH if the list device(s) are
                                      ;  logically ready to accept another output byte
0030 C3100A           JMP     SECTRAN ;Translates a logical sector into a physical one
                ;
                ;       Additional "private" BIOS entry points
                ;
0033 C38F04           JMP     AUXIST  ;Returns A = 0FFH if there is input data for
                                      ;  the logical auxiliary device
0036 C39B04           JMP     AUXOST  ;Returns A = 0FFH if the auxiliary device(s) are
                                      ;  logically ready to accept another output byte
0039 C3FA02           JMP     Specific$CIO$Initialization
                                      ;Initializes character device whose device
                                      ;  number is in register A on entry
003C C36D08           JMP     Set$Watchdog
                                      ;Sets up watchdog timer to CALL address specified
                                      ;  in HL, after BC clock ticks have elapsed
003F C33C0F           JMP     CB$Get$Address
                                      ;Configuration block get address
                                      ;  Returns address in HL of data element whose
                                      ;  code number is specified in C
                ;
                ;#
                ;       Long term configuration block
                ;
                Long$Term$CB:
                ;
                ;
                ;       Public files (files in user 0 accessible from all
                ;       other user numbers) enabled when this flag is set
                ;       nonzero.
                ;
0042 00         CB$Public$Files:        DB      0       ;Default is OFF
                ;
                ;
                ;       The forced input pointer is initialized to point to the
                ;       following string of characters. These are injected into
                ;       the console input stream on system start-up.
                ;
0043 5355424D49 CB$Startup:             DB      'SUBMIT STARTUP',LF,0,0,0,0,0,0
                ;
                ;       Logical to physical device redirection
                ;
                ;               Each logical device has a 16-bit word associated
                ;               with it. Each bit in the word is assigned to a
                ;               specific physical device. For input, only one bit
                ;               can be set -- input will be read from the
                ;               corresponding physical device. Output can be
                ;               directed to several devices, so more than one
                ;               bit can be set.
                ;
                ;               The following equates are used to indicate
                ;               specific physical devices.
                ;
                ;                       1111 11            )
                ;                       5432 1098 7654 3210 )<- Device number
0001 =          Device$0        EQU     0000$0000$0000$0001B
0002 =          Device$1        EQU     0000$0000$0000$0010B
0004 =          Device$2        EQU     0000$0000$0000$0100B
                ;
                ;               The following words are tested by the logical
                ;               device drivers to transfer control to
```

Figure 8-10. (Continued)

```
                ;               the appropriate physical device drivers
                ;
0058 0100       CB$Console$Input:       DW      Device$0
005A 0100       CB$Console$Output:      DW      Device$0
                ;
005C 0200       CB$Auxiliary$Input:     DW      Device$1
005E 0200       CB$Auxiliary$Output:    DW      Device$1
                ;
0060 0400       CB$List$Input:          DW      Device$2
0062 0400       CB$List$Output:         DW      Device$2
                ;
                ;               The table below relates specific bits in the
                ;               redirection words above to specific device
                ;               tables used by the physical drivers
                ;
                CB$Device$Table$Addresses:
0064 8E02               DW      DT$0
0066 AE02               DW      DT$1
0068 CE02               DW      DT$2
006A 0000000000         DW      0,0,0,0,0,0,0,0,0,0,0,0,0       ;Unassigned
                ;
                ;
                ;       Device initialization byte streams
                ;
                ;       These initialization streams are output during the device
                ;       initialization phase, or on request whenever the baud rate
                ;       needs to be changed. They are defined in the long term
                ;       configuration block so as to "freeze" their contents from one
                ;       system startup until the next.
                ;
                ;       The address of each stream is contained in each device table.
                ;
                ;       The stream format is:
                ;
                ;               DB      xx              ;Port number (00H terminates)
                ;               DB      nn              ;Number of bytes to output to port
                ;               DB      vv,vv,vv..      ;Values to be output
                ;
                D0$Initialize$Stream:           ;Example data for an 8251A chip
0084 ED                 DB      0EDH            ;Port number for 8251A
0085 06                 DB      6               ;Number of bytes
0086 000000             DB      0,0,0           ;Dummy bytes to get chip ready
0089 42                 DB      0100$0010B      ;Reset and raise DTR
008A 6E                 DB      01$10$11$10B    ;1 stop, no parity, 8 bits/char,
                                                ;  divide down of 16
008B 25                 DB      0010$0101B      ;RTS high, enable Tx/Rx
                                                ;Example data for an 8253 chip
008C DF                 DB      0DFH            ;Port number for 8253 mode
008D 01                 DB      1               ;Number of bytes to output
008E 76                 DB      01$11$011$0B    ;Select:
                                                ;       Counter 1
                                                ;       Load LS byte first
                                                ;       Mode 3, binary count
008F DE                 DB      0DEH            ;Port number for counter
0090 02                 DB      2               ;Number of bytes to output
                D0$Baud$Rate$Constant:          ;Label used by utilities
0091 0700               DW      0007H           ;9600 Baud (based on 16x divider)
0093 00                 DB      0               ;Port number of 00 terminates stream

                D1$Initialize$Stream:           ;Example data for an 8251A chip
0094 DD                 DB      0DDH            ;Port number for 8251A
0095 06                 DB      6               ;Number of bytes
0096 000000             DB      0,0,0           ;Dummy bytes to get chip ready
0099 42                 DB      0100$0010B      ;Reset and raise DTR
009A 6E                 DB      01$10$11$10B    ;1 stop, no parity, 8 bits/char,
                                                ;  divide down of 16
009B 25                 DB      0010$0101B      ;RTS high, enable Tx/Rx

                                                ;Example data for an 8253 chip
009C DF                 DB      0DFH            ;Port number for 8253 mode
009D 01                 DB      1               ;Number of bytes to output
009E B6                 DB      10$11$011$0B    ;Select:
                                                ;       Counter 2
                                                ;       Load LS byte first
                                                ;       Mode 3, binary count
009F DE                 DB      0DEH            ;Port number for counter
00A0 02                 DB      2               ;Number of bytes to output
```

Figure 8-10. (Continued)

```
                D1$Baud$Rate$Constant:
00A1 3800               DW      0038H                   ;1200 baud (based on 16x divider)
00A3 00                 DB      0                       ;Port number of 00 terminates stream

                D2$Initialize$Stream:           ;Example data for an 8251A chip
00A4 DD                 DB      0DDH                    ;Port number for 8251A
00A5 06                 DB      6                       ;Number of bytes
00A6 000000             DB      0,0,0                   ;Dummy bytes to get chip ready
00A9 42                 DB      0100$0010B              ;Reset and raise DTR
00AA 6E                 DB      01$10$11$10B            ;1 stop, no parity, 8 bits/char,
                                                        ;  divide down of 16
00AB 25                 DB      0010$0101B              ;RTS high, enable Tx/Rx

                                                ;Example data for an 8253 chip
00AC DF                 DB      0DFH                    ;Port number for 8253 mode
00AD 01                 DB      1                       ;Number of bytes to output
00AE F6                 DB      11$11$011$0B            ;Select:
                                                        ;       Counter 3
                                                        ;       Load LS byte first
                                                        ;       Mode 3, binary count
00AF DE                 DB      0DEH                    ;Port number for counter
00B0 02                 DB      2                       ;Number of bytes to output
                D2$Baud$Rate$Constant:
00B1 3800               DW      0038H                   ;1200 baud (based on 16x divider)
00B3 00                 DB      0                       ;Port number of 00 terminates stream

                ;
                ;       This following table is used to determine the maximum
                ;       value for each character position in the ASCII time
                ;       value above (except the ":"). Note -- this table is
                ;       in the long term configuration block so that the clock
                ;       can be set "permanently" to either 12 or 24 hour format.
                ;
                ;       NOTE: The table is processed backwards -- to correspond
                ;       with the ASCII time.
                ;       Each character represents the value for the corresponding
                ;       character in the ASCII time at which a carry-and-reset-to-zero
                ;       should occur.
                ;
00B4 00                 DB      0               ;"Terminator"
                CB$12$24$Clock:
00B5 3334               DB      '34'            ;Change to '23' for a 12-hour clock
00B7 FF                 DB      0FFH            ;"Skip" character
00B8 363A               DB      '6:'            ;Maximum minutes are 59
00BA FF                 DB      0FFH            ;"Skip" character
00BB 363A               DB      '6:'            ;Maximum seconds are 59
                Update$Time$End:                ;Used when updating the time
                ;
                ;
                ;       Variables for the real time clock and watchdog
                ;       timer
                ;
00BD 3C         RTC$Ticks$per$Second    DB      60      ;Number of real time clock
                                                        ;  ticks per elapsed second
00BE 3C         RTC$Tick$Count          DB      60      ;Residual count before next
                                                        ;  second will elapse
00BF 0000       RTC$Watchdog$Count      DW      0       ;Watchdog timer tick count
                                                        ;(0 = no watchdog timer set)
00C1 0000       RTC$Watchdog$Address    DW      0       ;Address to which control
                                                        ;  will be transferred if the
                                                        ;  watchdog count hits 0

                ;
                ;       Function key table
                ;
                ;       This table consists of a series of entries, each one having the
                ;       following structure:
                ;
                ;               DB      Second character of sequence emitted by
                ;                       terminal's function key
                ;       (       DB      Third character of sequence -- NOTE: this        )
                ;       (               field will not be present if the source code     )
                ;       (               has been configured to accept only two characters )
                ;       (               in function key sequences.                       )
                ;       (               NOTE: Adjust the equates for:                    )
                ;       (                       Function$Key$Length                      )
                ;       (                       Three$Character$Function                 )
```

Figure 8-10. (Continued)

```
                ;
                ;                DB      A character string to be forced into the console
                ;                        input stream when the corresponding function key
                ;                        is pressed. The last byte of this string must be
                ;                        00H to terminate the forced input.
                ;
001B =          Function$Key$Lead        EQU     1BH     ;Signals function key sequence
0003 =          Function$Key$Length      EQU     3       ;Number of characters in function
                                                         ;  key input sequence (NOTE: this
                                                         ;  can only be 3 or 2 characters).

                ;
                                                         ;The logic associated with function
                                                         ;  key recognition is made easier with
                                                         ;  the following equate
0001 =          Three$Character$Function         EQU     Function$Key$Length - 2
                                                 ;Three$Character$Function will be TRUE if the
                                                 ;  function keys emit a three character
                                                 ;  sequence, FALSE if they emit a two character
                                                 ;  sequence.

                ;       Each entry in the table must be the same length, as defined by:
                ;
0013 =          CB$Function$Key$Entry$Size       EQU     16 + 1 + Function$Key$Length - 1
                ;                                         ^    ^                         ^
                ;                                         :    :                         :
                ;               Maximum length of substitute :          Lead character is not
                ;               string                        :          in table entry
                ;                                         For the terminating 00H
                ;
                ;       The last entry in the table is marked by a 00-byte.
                ;
                ;       The example values shown below are for a VT-100 terminal.
                ;
                CB$Function$Key$Table:
                ;                         123456789.1234  5  6 7 <- Use to check length
00C3 4F5046756E         DB      'O','P','Function Key 1',LF,0,0
00D6 4F5146756E         DB      'O','Q','Function Key 2',LF,0,0
00E9 4F5246756E         DB      'O','R','Function Key 3',LF,0,0
00FC 4F5346756E         DB      'O','S','Function Key 4',LF,0,0
                ;
                ;                         123456789.1
010F 5B41557020         DB      '[','A','Up Arrow',LF,0,0,0,0,0,0,0,0
0122 5B42446F77         DB      '[','B','Down Arrow',LF,0,0,0,0,0,0
0135 5B43526967         DB      '[','C','Right Arrow',LF,0,0,0,0,0
0148 5B444C6566         DB      '[','D','Left Arrow',LF,0,0,0,0,0,0

015B 0000000000         DB      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0    ;Spare entries
016E 0000000000         DB      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0181 0000000000         DB      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0194 0000000000         DB      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
01A7 0000000000         DB      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
01BA 0000000000         DB      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
01CD 0000000000         DB      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
01E0 0000000000         DB      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
01F3 0000000000         DB      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0206 0000000000         DB      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

0219 FFFF               DB      0FFH,0FFH       ;Terminator for utility that preprograms
                                                ;  function key sequence

                ;
                ;
                ;       Console output escape sequence control table
                ;
                ;       This table is referenced after a Function$Key$Lead character
                ;       has been detected in the CONOUT routine. The next character
                ;       to be output to the console is compared to the first byte
                ;       in each 3-byte table entry. If a match is found, then control
                ;       is transferred to the address following the byte that matched.
                ;
                CONOUT$Escape$Table:
021B 74                 DB      't'             ;Read current time
021C 4804               DW      CONOUT$Time
021E 64                 DB      'd'             ;Read current date
021F 4104               DW      CONOUT$Date
0221 75                 DB      'u'             ;Set current time
0222 5D04               DW      CONOUT$Set$Time
```

Figure 8-10. (Continued)

```
0224 65                 DB      'e'             ;Set current date
0225 4E04               DW      CONOUT$Set$Date

0227 00                 DB      0               ;Terminator
               ;
               Long$Term$CB$End:
               ;
               ;#
               ;
               ;        Interrupt vector
               ;
               ;        Control is transferred here by the programmable interrupt
               ;        controller -- an Intel 8259A.
               ;
               ;        NOTE: The interrupt controller chip requires that the
               ;              interrupt vector table start on a paragraph
               ;              boundary. This is achieved by the following ORG line
0240                    ORG     ($ AND OFFEOH) + 20H
               Interrupt$Vector:
                                        ;Interrupt number
0240 C37808             JMP     RTC$Interrupt           ;0 -- clock
0243 00                 DB      0                       ;Skip a byte
0244 C3E806             JMP     Character$Interrupt     ;1 -- character I/O
0247 00                 DB      0
0248 C3D80E             JMP     Ghost$Interrupt         ;2 -- not used
024B 00                 DB      0
024C C3D80E             JMP     Ghost$Interrupt         ;3 -- not used
024F 00                 DB      0
0250 C3D80E             JMP     Ghost$Interrupt         ;4 -- not used
0253 00                 DB      0
0254 C3D80E             JMP     Ghost$Interrupt         ;5 -- not used
0257 00                 DB      0
0258 C3D80E             JMP     Ghost$Interrupt         ;6 -- not used
025B 00                 DB      0
025C C3D80E             JMP     Ghost$Interrupt         ;7 -- not used
               ;
               ;#

               ;        Device port numbers and other equates
               ;
0080 =         CIO$Base$Port     EQU    80H             ;Base port number

0080 =         D0$Base$Port      EQU    CIO$Base$Port           ;Device 0
0080 =         D0$Data$Port      EQU    D0$Base$Port
0081 =         D0$Status$Port    EQU    D0$Base$Port + 1
0082 =         D0$Mode$Port      EQU    D0$Base$Port + 2
0083 =         D0$Command$Port   EQU    D0$Base$Port + 3
               ;

0084 =         D1$Base$Port      EQU    CIO$Base$Port + 4       ;Device 1
0084 =         D1$Data$Port      EQU    D1$Base$Port
0085 =         D1$Status$Port    EQU    D1$Base$Port + 1
0086 =         D1$Mode$Port      EQU    D1$Base$Port + 2
0087 =         D1$Command$Port   EQU    D1$Base$Port + 3

0088 =         D2$Base$Port      EQU    CIO$Base$Port + 8       ;Device 2
0088 =         D2$Data$Port      EQU    D2$Base$Port
0089 =         D2$Status$Port    EQU    D2$Base$Port + 1
008A =         D2$Mode$Port      EQU    D2$Base$Port + 2
008B =         D2$Command$Port   EQU    D2$Base$Port + 3

004E =         D$Mode$Value$1    EQU    01$00$11$10B
                                                ;1 stop bit, no parity
                                                ;8 bits, Async. 16x rate
003C =         D$Mode$Value$2    EQU    00$11$1100B
                                                ;Tx/Rx on internal clock
                                                ;9600 baud
0027 =         D$Command$Value   EQU    00$100111B
                                                ;Normal mode
                                                ;Enable Tx/Rx
                                                ;RTS and DTR active
0038 =         D$Error           EQU    0011$1000B
0037 =         D$Error$Reset     EQU    00$110111B
                                                ;Same as command value plus error reset
0001 =         D$Output$Ready    EQU    0000$0001B
0002 =         D$Input$Ready     EQU    0000$0010B
0080 =         D$DTR$High        EQU    1000$0000B      ;Note: this is actually the
```

Figure 8-10. (Continued)

```
                                                          ;   data-set-ready pin
                                                          ;   on the chip. It is connected
                                                          ;   to the DTR pin on the cable
0027 =          D$Raise$RTS      EQU     00$1$00111B      ;Raise RTS, Tx/Rx enable
0007 =          D$Drop$RTS       EQU     00$0$00111B      ;Drop RTS, Tx/Rx enable
                ;
                ;
                ;       Interrupt controller ports (Intel 8259A)
                ;
                ;       Note : these equates are placed here so that they
                ;              follow the  definition of the interrupt vector
                ;              and thus avoid 'P' (phase) errors in ASM.
                ;
00D9 =          IC$OCW1$Port     EQU     0D9H     ;Operational control word 1
00D8 =          IC$OCW2$Port     EQU     0D8H     ;Operational control word 2
00D8 =          IC$OCW3$Port     EQU     0D8H     ;Operational control word 3
00D8 =          IC$ICW1$Port     EQU     0D8H     ;Initialization control word 1
00D9 =          IC$ICW2$Port     EQU     0D9H     ;Initialization control word 2
                ;
0020 =          IC$EOI           EQU     20H      ;Nonspecific end of interrupt
                ;
0056 =          IC$ICW1          EQU     (Interrupt$Vector AND 1110$0000B) + 000$10110B
                                                  ;Sets the A7 - A5 bits of the interrupt
                                                  ;  vector address plus:
                                                  ;       Edge triggered
                                                  ;       4-byte interval
                                                  ;       Single 8259 in system
                                                  ;       No ICW4 needed
0002 =          IC$ICW2          EQU     Interrupt$Vector SHR 8
                                                  ;Address bits A15 - A8 of the interrupt
                                                  ;  vector address. Note the interrupt
                                                  ;  vector is the first structure in
                                                  ;  the long term configuration block

00FC =          IC$OCW1          EQU     1111$1100B        ;Interrupt mask
                                                  ;Interrupt 0 (clock) enabled
                                                  ;Interrupt 1 (character input) enabled
                ;
                ;#
                ;
                ;
                Display$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.
025F 7E                 MOV     A,M               ;Get next message byte
0260 B7                 ORA     A                 ;Check if terminator
0261 C8                 RZ                        ;Yes, return to caller
0262 4F                 MOV     C,A               ;Prepare for output
0263 E5                 PUSH    H                 ;Save message pointer
0264 CDD703             CALL    CONOUT            ;Go to main console output routine
0267 E1                 POP     H                 ;Recover message pointer
0268 23                 INX     H                 ;Move to next byte of message
0269 C35F02             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 (the DMA address).
                ;
026C 3EC3               MVI     A,JMP             ;Get machine code for JMP
026E 320000             STA     0000H             ;Set up JMP at location 0000H
0271 320500             STA     0005H             ; and at location 0005H
                ;
0274 210300             LXI     H,Warm$Boot$Entry        ;Get BIOS vector address
0277 220100             SHLD    0001H             ;Put address at location 0001H

027A 2106CC             LXI     H,BDOS$Entry      ;Get BDOS entry point address
027D 220600             SHLD    6                 ;Put address at location 0005H
                ;
0280 018000             LXI     B,80H             ;Set disk I/O address to default
0283 CDA809             CALL    SETDMA            ;Use normal BIOS routine
                ;
0286 FB                 EI                        ;Ensure interrupts are enabled
0287 3A0400             LDA     Default$Disk      ;Handover current default disk to
028A 4F                 MOV     C,A               ; console command processor
```

Figure 8-10. (Continued)

```
028B C300C4         JMP     CCP$Entry         ;Transfer to CCP
              ;
              ;#
              ;
              ;       Device table equates
              ;       The drivers use a device table for each
              ;       physical device they service. The equates that follow
              ;       are used to access the various fields within the
              ;       device table.
              ;
              ;                          Port numbers and status bits
0000 =        DT$Status$Port             EQU     0       ;Device status port number
0001 =        DT$Data$Port               EQU     DT$Status$Port+1
                                                         ;Device data port number
0002 =        DT$Output$Ready            EQU     DT$DataPort+1
                                                         ;Output ready status mask
0003 =        DT$Input$Ready             EQU     DT$Output$Ready+1
                                                         ;Input ready status mask
0004 =        DT$DTR$Ready               EQU     DT$Input$Ready+1
                                                         ;DTR ready to send mask
0005 =        DT$Reset$Int$Port          EQU     DT$DTR$Ready+1
                                                         ;Port number used to reset an
                                                         ;  interrupt
0006 =        DT$Reset$Int$Value         EQU     DT$Reset$Int$Port+1
                                                         ;Value output to reset interrupt
0007 =        DT$Detect$Error$Port       EQU     DT$Reset$Int$Value+1
                                                         ;Port number for detecting error
0008 =        DT$Detect$Error$Value      EQU     DT$Detect$Error$Port+1
                                                         ;Mask for detecting error (parity etc.)
0009 =        DT$Reset$Error$Port        EQU     DT$Detect$Error$Value+1
                                                         ;Output to port to reset error
000A =        DT$Reset$Error$Value       EQU     DT$Reset$Error$Port+1
                                                         ;Value to output to reset error
000B =        DT$RTS$Control$Port        EQU     DT$Reset$Error$Value+1
                                                         ;Control port for lowering RTS
000C =        DT$Drop$RTS$Value          EQU     DT$RTS$Control$Port+1
                                                         ;Value, when output, to drop RTS
000D =        DT$Raise$RTS$Value         EQU     DT$Drop$RTS$Value+1
                                                         ;Value, when output, to raise RTS
              ;
              ;               Device logical status (incl. protocols)
000E =        DT$Status                  EQU     DT$Raise$RTS$Value+1
                                                                 ;Status bits
0001 =        DT$Output$Suspend          EQU     0000$0001B      ;Output suspended pending
                                                                 ;  protocol action
0002 =        DT$Input$Suspend           EQU     0000$0010B      ;Input suspended until
                                                                 ;  buffer empties
0004 =        DT$Output$DTR              EQU     0000$0100B      ;Output uses DTR-high-to-send
0008 =        DT$Output$Xon              EQU     0000$1000B      ;Output uses XON/XOFF
0010 =        DT$Output$Etx              EQU     0001$0000B      ;Output uses ETX/ACK
0020 =        DT$Output$Timeout          EQU     0010$0000B      ;Output uses timeout
0040 =        DT$Input$RTS               EQU     0100$0000B      ;Input uses RTS-high-to-receive
0080 =        DT$Input$Xon               EQU     1000$0000B      ;Input uses XON/XOFF
              ;
000F =        DT$Status$2                EQU     DT$Status+1     ;Secondary status byte
0001 =        DT$Fake$Typeahead          EQU     0000$0001B      ;Requests Input$Status to
                                                                 ; return "Data Ready" when
                                                                 ; control dharacters are in
                                                                 ; input buffer
              ;
0010 =        DT$Etx$Count               EQU     DT$Status$2+1
                                                         ;No. of chars. sent in Etx protocol
0012 =        DT$Etx$Message$Length      EQU     DT$Etx$Count+2
                                                         ;Specified message length
              ;
              ;                          Input buffer values
0014 =        DT$Buffer$Base             EQU     DT$Etx$Message$Length+2
                                                         ;Address of Input buffer
0016 =        DT$Put$Offset              EQU     DT$Buffer$Base+2
                                                         ;Offset for putting chars. into buffer
0017 =        DT$Get$Offset              EQU     DT$Put$Offset+1
                                                         ;Offset for getting chars. from buffer
0018 =        DT$Buffer$Length$Mask      EQU     DT$Get$Offset+1
                                                         ;Length of buffer - 1
                                                         ;Note: Buffer length must always be
                                                         ;  a binary number; e.g. 32, 64 or 128
```

Figure 8-10. (Continued)

```
                                                      ;This mask then becomes:
                                                      ;   32 ->  31 (0001$1111B)
                                                      ;   64 ->  63 (0011$1111B)
                                                      ;  128 -> 127 (0111$1111B)
                                                      ;After the get/put offset has been
                                                      ;  incremented, it is ANDed with the mask
                                                      ;  to reset it to zero when the end of
                                                      ;  the buffer has been reached
0019 =        DT$Character$Count        EQU     DT$Buffer$Length$Mask+1
                                                      ;Count of the number of characters
                                                      ;  currently in the buffer
001A =        DT$Stop$Input$Count       EQU     DT$Character$Count+1
                                                      ;Stop input when the count reaches
                                                      ;  this value
001B =        DT$Resume$Input$Count     EQU     DT$Stop$Input$Count+1
                                                      ;Resume input when the count reaches
                                                      ;  this value
001C =        DT$Control$Count          EQU     DT$Resume$Input$Count+1
                                                      ;Count of the number of control
                                                      ;  characters in the buffer
001D =        DT$Function$Delay         EQU     DT$Control$Count+1
                                                      ;Number of clock ticks to delay to
                                                      ;  allow all characters after function
                                                      ;  key lead-in to arrive
001E =        DT$Initialize$Stream      EQU     DT$Function$Delay+1
                                                      ;Address of byte stream necessary to
                                                      ;  initialize this device


              ;#
              ;
              ;         Device tables
              ;
              DT$0:
028E 81                 DB      D0$Status$Port   ;Status port (8251A chip)
028F 80                 DB      D0$Data$Port     ;Data port
0290 01                 DB      D$Output$Ready   ;Output data ready
0291 02                 DB      D$Input$Ready    ;Input data ready
0292 80                 DB      D$DTR$High       ;DTR ready to send
0293 D8                 DB      IC$OCW2$Port     ;Reset interrupt port (00H is an unused port)
0294 20                 DB      IC$EOI           ;Reset interrupt value (nonspecific EOI)
0295 81                 DB      D0$Status$Port   ;Detect error port
0296 38                 DB      D$Error          ;Mask: framing, overrun, parity errors
0297 83                 DB      D0$Command$Port  ;Reset error port
0298 37                 DB      D$Error$Reset    ;Reset error: RTS high, reset, Tx/Rx enable
0299 83                 DB      D0$Command$Port  ;Drop/raise RTS port
029A 07                 DB      D$Drop$RTS       ;Drop RTS Value (keep Tx & Rx enabled)
029B 27                 DB      D$Raise$RTS      ;Raise RTS value (keep Tx & Rx enabled)
029C C0                 DB      DT$Input$Xon + DT$Input$RTS     ;Protocol and status
029D 00                 DB      0                ;Status #2
029E 0004               DW      1024             ;Etx/Ack message count
02A0 0004               DW      1024             ;Etx/Ack message length
02A2 2422               DW      D0$Buffer        ;Input buffer
02A4 00                 DB      0                ;Put offset into buffer
02A5 00                 DB      0                ;Get offset into buffer
02A6 1F                 DB      D0$Buffer$Length -1 ;Buffer length mask
02A7 00                 DB      0                ;Count of characters in buffer
02A8 1B                 DB      D0$Buffer$Length - 5 ;Stop input when count hits this value
02A9 10                 DB      D0$Buffer$Length / 2 ;Resume input when count hits this value
02AA 00                 DB      0                ;Count of control characters in buffer
02AB 06                 DB      6                ;Number of 16.66ms ticks to allow function
                                                 ;  key sequence to arrive (approx. 90ms)
02AC 8400               DW      D0$Initialize$Stream    ;Address of initialization stream
              ;
              DT$1:
02AE 85                 DB      D1$Status$Port   ;Status port (8251A chip)
02AF 84                 DB      D1$Data$Port     ;Data port
02B0 01                 DB      D$Output$Ready   ;Output data ready
02B1 02                 DB      D$Input$Ready    ;Input data ready
02B2 80                 DB      D$DTR$High       ;DTR ready to send
02B3 D8                 DB      IC$OCW2$Port          ;Reset interrupt port (00H is an unused port)
02B4 20                 DB      IC$EOI           ;Reset interrupt value (nonspecific EOI)
02B5 85                 DB      D1$Status$Port   ;Detect error port
02B6 38                 DB      D$Error          ;Mask: framing, overrun, parity errors
02B7 87                 DB      D1$Command$Port  ;Reset error port
02B8 37                 DB      D$Error$Reset    ;Reset error: RTS high, reset, Tx/Rx enable
02B9 87                 DB      D1$Command$Port  ;Drop/raise RTS port
02BA 07                 DB      D$Drop$RTS       ;Drop RTS value (keep Tx & Rx enabled)
```

Figure 8-10. (Continued)

```
02BB 27                DB      D$Raise$RTS     ;Raise RTS value (keep Tx & Rx enabled)
02BC C0                DB      DT$Input$Xon + DT$Input$RTS     ;Protocol and status
02BD 00                DB      0               ;Status #2
02BE 0004              DW      1024            ;Etx/Ack message count
02C0 0004              DW      1024            ;Etx/Ack message length
02C2 4422              DW      D1$Buffer       ;Input buffer
02C4 00                DB      0               ;Put offset into buffer
02C5 00                DB      0               ;Get offset into buffer
02C6 1F                DB      D1$Buffer$Length -1 ;Buffer length mask
02C7 00                DB      0               ;Count of characters in buffer
02C8 1B                DB      D1$Buffer$Length - 5 ;Stop input when count hits this value
02C9 10                DB      D1$Buffer$Length / 2 ;Resume input when count hits this value
02CA 00                DB      0               ;Count of control characters in buffer
02CB 06                DB      6               ;Number of 16.66ms ticks to allow function
                                               ;  key sequence to arrive (approx. 90ms)
02CC 9400              DW      D1$Initialize$Stream    ;Address of initialization stream
                ;
                ;
                DT$2:
02CE 89                DB      D2$Status$Port  ;Status port (8251A chip)
02CF 88                DB      D2$Data$Port    ;Data port
02D0 01                DB      D$Output$Ready  ;Output data ready
02D1 02                DB      D$Input$Ready   ;Input data ready
02D2 80                DB      D$DTR$High      ;DTR ready to send
02D3 D8                DB      IC$OCW2$Port    ;Reset interrupt port (00H is an unused port)
02D4 20                DB      IC$EOI          ;Reset interrupt value (nonspecific EOI)
02D5 89                DB      D2$Status$Port  ;Detect error port
02D6 38                DB      D$Error         ;Mask: framing, overrun, parity errors
02D7 8B                DB      D2$Command$Port ;Reset error port
02D8 37                DB      D$Error$Reset   ;Reset error: RTS high, reset, Tx/Rx enable
02D9 8B                DB      D2$Command$Port ;Drop/raise RTS port
02DA 07                DB      D$Drop$RTS      ;Drop RTS value (keep Tx & Rx enabled)
02DB 27                DB      D$Raise$RTS     ;Raise RTS value (keep Tx & Rx enabled)
02DC C0                DB      DT$Input$Xon + DT$Input$RTS     ;Protocol and status
02DD 00                DB      0               ;Status #2
02DE 0004              DW      1024            ;Etx/Ack message count
02E0 0004              DW      1024            ;Etx/Ack message length
02E2 6422              DW      D2$Buffer       ;Input buffer
02E4 00                DB      0               ;Put offset into buffer
02E5 00                DB      0               ;Get offset into buffer
02E6 1F                DB      D2$Buffer$Length -1 ;Buffer length mask
02E7 00                DB      0               ;Count of characters in buffer
02E8 1B                DB      D2$Buffer$Length - 5 ;Stop input when count hits this value
02E9 10                DB      D2$Buffer$Length / 2 ;Resume input when count hits this value
02EA 00                DB      0               ;Count of control characters in buffer
02EB 06                DB      6               ;Number of 16.66ms ticks to allow function
                                               ;  Key sequence to arrive (approx. 90ms)
02EC A400              DW      D2$Initialize$Stream    ;Address of initialization stream
                ;
                ;#
                ;       General character I/O device initialization
                ;
                ;       This routine will be called from the main CP/M
                ;       initialization code.
                ;
                ;       It makes repeated calls to the specific character I/O
                ;       device initialization routine.
                ;
                General$CIO$Initialization:
02EE AF                XRA     A               ;Set device number (used to access the
                                               ;  table of device table addresses in the
                                               ;  configuration block)
02EF 4F                MOV     C,A             ;Match to externally CALLable interface
                GCI$Next$Device:
02F0 CDFA02            CALL    Specific$CIO$Initialization     ;Initialize the device
02F3 3C                INR     A               ;Move to next device
02F4 FE10              CPI     16              ;Check if all possible devices (0 - 15)
02F6 C8                RZ                      ;  have been initialized
02F7 C3F002            JMP     GCI$Next$Device
                ;
                ;#
                ;
                ;       Specific character I/O initialization
                ;
                ;       This routine outputs the specified byte values to the specified
                ;       ports as controlled by the initialization streams in the
                ;       configuration block. Each device table contains a pointer to
```

Figure 8-10. (Continued)

```
            ;       these streams. The device table itself is selected according
            ;       to the device NUMBER -- this is an entry parameter for this
            ;       routine.
            ;       This routine will be called either from the general device
            ;       initialization routine above, or directly by a BIOS call from
            ;       a system utility executing in the TPA.
            ;
            ;       Entry parameters
            ;
            ;               C = device number
            ;
            ;       Exit parameters
            ;
            ;               A = Device number (preserved)
            ;
            ;===========================
            Specific$CIO$Initialization:            ;<=== BIOS entry point (private)
            ;===========================
02FA 79             MOV     A,C             ;Get device number
02FB F5             PUSH    PSW             ;Preserve device number
02FC 87             ADD     A               ;Make device number into word pointer
02FD 4F             MOV     C,A
02FE 0600           MVI     B,0             ;Make into a word
0300 216400         LXI     H,CB$Device$Table$Addresses     ;Get table base
0303 09             DAD     B               ;HL -> device table address
0304 5E             MOV     E,M             ;Get LS byte
0305 23             INX     H
0306 56             MOV     D,M             ;Get MS byte: DE -> device table

0307 7A             MOV     A,D             ;Check if device table address = 0
0308 B3             ORA     E
0309 CA1703         JZ      SCI$Exit        ;Yes, device table nonexistent

030C 211E00         LXI     H,DT$Initialize$Stream
030F 19             DAD     D               ;HL -> initialization stream address
0310 5E             MOV     E,M             ;Get LS byte
0311 23             INX     H
0312 56             MOV     D,M             ;Get MS byte
0313 EB             XCHG                    ;HL -> initialization stream itself
0314 CD1903         CALL    Output$Byte$Stream      ;Output byte stream to various
                                                    ;  ports
            ;
            SCI$Exit:
0317 F1             POP     PSW             ;Recover user's device number in C
0318 C9             RET
            ;
            ;#
            ;       Output byte stream
            ;
            ;       This routine outputs initialization bytes to port
            ;       numbers. The byte stream has the following format:
            ;
            ;               DB      ppH     Port number
            ;               DB      nn      Number of bytes to output
            ;               DB      vvH,vvH...   Bytes to be output
            ;               :
            ;               :       Repeated
            ;               :
            ;               DB      00H     Port number of 0 terminates
            ;
            ;       Entry parameters
            ;
            ;               HL -> Byte stream
            ;
            Output$Byte$Stream:
            OBS$Loop:
0319 7E             MOV     A,M             ;Get port number
031A B7             ORA     A               ;Check if 00H (terminator)
031B C8             RZ                      ;Exit if at end of stream
031C 322503         STA     OBS$Port        ;Store in port number below
031F 23             INX     H               ;HL -> count of bytes
0320 4E             MOV     C,M             ;Get count
0321 23             INX     H               ;HL -> first initialization byte
            ;
            OBS$Next$Byte:
0322 7E             MOV     A,M             ;Get next byte
0323 23             INX     H               ;HL -> next data byte (or port number)
```

Figure 8-10. (Continued)

```
0324 D3                      DB      OUT
                     OBS$Port:
0325 00                      DB      0                 ;<- Set up in instruction above
0326 0D                      DCR     C                 ;Count down on byte counter
0327 C22203                  JNZ     OBS$Next$Byte     ;Output next data byte
032A C31903                  JMP     OBS$Loop          ;Go back for next port number
                     ;
                     ;#
                     ;       CONST - Console status
                     ;
                     ;       This routine checks both the forced input pointer and
                     ;       the character count for the appropriate input buffer.
                     ;       The A register is set to indicate whether or not there
                     ;       is data waiting.
                     ;
                     ;       Entry parameters: none.
                     ;
                     ;       Exit parameters
                     ;
                     ;               A = 000H if there is no data waiting
                     ;               A = 0FFH if there is data waiting
                     ;
                     ;=========================
                     CONST:                                    ;<=== BIOS entry point (standard)
                     ;=========================
032D 2A5800                  LHLD    CB$Console$Input          ;Get redirection word
0330 116400                  LXI     D,CB$Device$Table$Addresses
0333 CD6F06                  CALL    Select$Device$Table       ;Get device table address
0336 C34708                  JMP     Get$Input$Status          ;Get status from input device
                                                               ;  and return to caller
                     ;#
                     ;
                     ;       CONIN -- console input
                     ;
                     ;       This routine returns the next character for the console input
                     ;       stream. Depending on the circumstances, this can be a character
                     ;       from the console input buffer, or from a previously stored
                     ;       string of characters to be "forced" into the input stream,for
                     ;       the automatic execution of system initialization routines.
                     ;       The "forced input" can come from any previously stored character
                     ;       string in memory. It is used to inject the current time and date
                     ;       or a string associated with a function key into the console
                     ;       stream. On system startup, a string of "SUBMIT STARTUP" is
                     ;       forced into the console input stream to provide a mechanism.
                     ;
                     ;       Normal ("unforced") input comes from whichever physical device
                     ;       is specified in the console input redirection word (see the
                     ;       configuration block).
                     ;
0339 00              CONIN$Delay$Elapsed:    DB      0         ;Flag used during function key
                                                               ;  processing to indicate that
                                                               ;  a predetermined delay has
                                                               ;  elapsed
                     ;
                     ;=========================
                     CONIN:                                    ;<=== BIOS entry point (standard)
                     ;=========================
033A 2A8D0F                  LHLD    CB$Forced$Input           ;Get the forced input pointer
033D 7E                      MOV     A,M                       ;Get the next character of input
033E B7                      ORA     A                         ;Check if a null
033F CA4703                  JZ      CONIN$No$FI               ;Yes, no forced input
0342 23                      INX     H                         ;Yes, update the pointer
0343 228D0F                  SHLD    CB$Forced$Input           ;  and store it back
0346 C9                      RET
                     ;
                     CONIN$No$FI                               ;No forced input
0347 2A5800                  LHLD    CB$Console$Input          ;Get redirection word
034A 116400                  LXI     D,CB$Device$Table$Addresses
034D CD6F06                  CALL    Select$Device$Table       ;Get device table address
0350 CD9106                  CALL    Get$Input$Character       ;Get next character from input device

                                                       ;Function key processing
0353 FE1B                    CPI     Function$Key$Lead         ;Check if first character of function
                                                               ;  key sequence (normally escape)
0355 C0                      RNZ                               ;Return to BIOS caller if not
0356 F5                      PUSH    PSW                       ;Save lead in character
```

Figure 8-10. (Continued)

```
0357 211D00          LXI     H,DT$Function$Delay      ;Get delay time constant for
                                                      ;  delay while waiting for subsequent
                                                      ;  characters of function key sequence
                                                      ;  to arrive
035A 19              DAD     D
035B 4E              MOV     C,M                      ;Get delay value
035C 0600            MVI     B,0                      ;Make into word value
035E AF              XRA     A                        ;Indicate timer not yet out of time
035F 323903          STA     CONIN$Delay$Elapsed
0362 217B03          LXI     H,CONIN$Set$Delay$Elapsed ;Address to resume at after delay
0365 CD6D08          CALL    Set$Watchdog             ;Sets up delay based on real time
                                                      ;  clock such that control will be
                                                      ;  transferred to specified address
                                                      ;  after time interval has elapsed
             CONIN$Wait$for$Delay:                    ;Wait here until delay has elapsed
0368 3A3903          LDA     CONIN$Delay$Elapsed      ;Check flag set by watchdog routine
036B B7              ORA     A
036C CA6803          JZ      CONIN$Wait$for$Delay

             CONIN$Check$for$Function:
036F 211900          LXI     H,DT$Character$Count     ;Now check if the remaining characters
                                                      ;  of the sequence have been input
0372 19              DAD     D
0373 7E              MOV     A,M                      ;Get count of characters in buffer
0374 FE02            CPI     Function$Key$Length - 1
0376 D28103          JNC     CONIN$Check$Function     ;Enough characters in buffer for
                                                      ;  possible function key sequence
0379 F1              POP     PSW                      ;Insufficient characters in buffer
                                                      ;  to be a function key, so return
                                                      ;  to caller with lead character
037A C9              RET

             ;
             ;       The following routine is called by the watchdog routine
             ;       when the specified delay has elapsed.
             ;
             CONIN$Set$Delay$Elapsed:
037B 3EFF            MVI     A,0FFH                   ;Indicate watchdog timer out of time
037D 323903          STA     CONIN$Delay$Elapsed
0380 C9              RET                              ;Return to watchdog routine
             ;
             ;
             CONIN$Check$Function:
0381 211700          LXI     H,DT$Get$Offset          ;Save the current "get pointer"
0384 19              DAD     D                        ;  in the buffer
0385 7E              MOV     A,M                      ;Get the pointer
0386 F5              PUSH    PSW                      ;Save pointer on the stack

0387 211700          LXI     H,DT$Get$Offset          ;Check the second (and possibly third)
038A CDF007          CALL    Get$Address$in$Buffer    ;  character in the sequence
038D 46              MOV     B,M                      ;Get the second character

                     IF      Three$Character$Function
038E C5              PUSH    B                        ;Save for later use
038F 211700          LXI     H,DT$Get$Offset          ;Retrieve the third character
0392 CDF007          CALL    Get$Address$in$Buffer
0395 C1              POP     B                        ;Recover second character
0396 4E              MOV     C,M                      ;Now BC = Char 2, Char 3
                     ENDIF

0397 D5              PUSH    D                        ;Save device table pointer
0398 21B000          LXI     H,CB$Function$Key$Table - CB$Function$Key$Entry$Size
                                                      ;Get pointer to function key table
                                                      ;  in configuration block
039B 111300          LXI     D,CB$Function$Key$Entry$Size    ;Get entry size ready for loop
             CONIN$Next$Function:
039E 19              DAD     D                        ;Move to next (or first) entry
039F 7E              MOV     A,M                      ;Get second character of sequence
03A0 B7              ORA     A                        ;Check if end of function key table
03A1 CAC203          JZ      CONIN$Not$Function       ;Yes -- it is not a function key
03A4 B8              CMP     B                        ;Compare second characters
03A5 C29E03          JNZ     CONIN$Next$Function      ;No match, so try next entry in table

                     IF      Three$Character$Function
03A8 23              INX     H                        ;HL -> third character
03A9 7E              MOV     A,M                      ;Get third character of sequence
03AA 2B              DCX     H                        ;Simplify logic for 2 & 3 char. seq.
```

Figure 8-10. (Continued)

```
03AB B9              CMP     C                       ;Compare third characters
03AC C29E03          JNZ     CONIN$Next$Function     ;No match, so try next entry in table
03AF 23              INX     H                       ;When match found, compensate for
                                                     ;  extra decrement
                     ENDIF

03B0 23              INX     H                       ;HL -> first character of substitute
                                                     ;  string of characters (00-byte term.)
03B1 228D0F          SHLD    CB$Forced$Input         ;Make the CONIN routine inject the
                                                     ;  substitute string into the input
                                                     ;  stream

                                                     ;Now that a function sequence has been
                                                     ;  identified, the stack must be
                                                     ;  balanced prior to return
03B4 D1              POP     D                       ;Get the device table pointer
03B5 F1              POP     PSW                     ;Dump the "get" offset value
03B6 F1              POP     PSW                     ;Dump the function sequence lead char.

03B7 211900          LXI     H,DT$Character$Count    ;Downdate the character count
03BA 19              DAD     D                       ;  to reflect the characters removed
                                                     ;  from the buffer
03BB 7E              MOV     A,M                     ;Get the count
03BC D602            SUI     Function$Key$Length -1  ; (the lead character has already
03BE 77              MOV     M,A                     ;  been deducted)
03BF C33A03          JMP     CONIN                   ;Return to CONIN processing to get
                                                     ;  the forced input characters
             CONIN$Not$Function:
                                             ;Attempts to recognize a function key sequence
                                             ;  have failed. The "get" offset pointer must be
                                             ;  restored to its previous value so that
                                             ;  the character(s) presumed to be part of
                                             ;  the function sequence are not lost.

03C2 D1              POP     D                       ;Recover device table pointer
03C3 F1              POP     PSW                     ;Recover previous "get" offset
03C4 211700          LXI     H,DT$Get$Offset
03C7 19              DAD     D                       ;HL -> "get" offset in table
03C8 77              MOV     M,A                     ;Reset "get" offset as it was after
                                                     ;  the lead character was detected
03C9 F1              POP     PSW                     ;Recover lead character
03CA C9              RET                             ;Return the lead character to the user
             ;
             ;#
             ;       Console output
             ;
             ;       This routine outputs data characters to the console device(s).
             ;       It also "traps" escape sequences being output to the console,
             ;       triggering specific actions according to the sequences.
             ;       A primitive "state-machine" is used to step through escape
             ;       sequence recognition.
             ;       In addition to outputting the next character to all of the
             ;       devices currently selected in the console output redirection word,
             ;       it checks to see that output to the selected device has not been
             ;       suspended by XON/XOFF protocol, and that DTR is high if
             ;       it should be.
             ;       Once the character has been output, if ETX/ACK protocol is in use,
             ;       and the specified length of message has been output, an Etx
             ;       character is output and the device is flagged as being suspended.
             ;
             ;   '   Entry parameters
             ;
             ;               C = character to be output
             ;
             ;       CONOUT storage variables
             ;
03CB 00      CONOUT$Character:       DB      0       ;Save area for character to be output

03CC DB03    CONOUT$Processor:       DW      CONOUT$Normal
                                                     ;This is the address of the piece of
                                                     ;  code that will process the next
                                                     ;  character. The default case is
                                                     ;  CONOUT$Normal
03CE 0000    CONOUT$String$Pointer:  DW      0       ;This points to a string (normally
                                                     ;  in the configuration block) that
                                                     ;  is being preset by characters from
                                                     ;  the console output stream
```

Figure 8-10. (Continued)

```
03D0 00         CONOUT$String$Length:   DB      0       ;This contains the maximum number of
                                                        ;  characters to be preset into a
                                                        ;  from the console output stream

                ;
                ;       *** WARNING ***
                ;       The output error message routine shares the code in this
                ;       subroutine. On entry here, the data byte to be output
      -         ;       will be on the stack, and the DE registers set up correctly.
                ;
                ;
                CONOUT$OEM$Entry:
03D1 32CB03             STA     CONOUT$Character        ;Save data byte
03D4 C3E803             JMP     CONOUT$Entry2           ;HL already has special bit map
                ;
                ;=====================
                CONOUT:                         ;<=== BIOS entry point (standard)
                ;=====================
03D7 2ACC03             LHLD    CONOUT$Processor        ;Get address of processor to handle
                                                        ;  the next character to be output
                                                        ;(Default is CONOUT$Normal)
03DA E9                 PCHL                            ;Transfer control to the processor
                ;
                ;
                CONOUT$Normal:                          ;Normal processor for console output
03DB 79                 MOV     A,C                     ;Check if possible start of escape
03DC FE1B               CPI     Function$Key$Lead       ;  sequence
03DE CA1204             JZ      CONOUT$Escape$Found     ;Perhaps
                CONOUT$Forced:
03E1 79                 MOV     A,C                     ;Forced output entry point
03E2 32CB03             STA     CONOUT$Character        ;Not escape sequence -- Save data byte

03E5 2A5A00             LHLD    CB$Console$Output       ;Get console redirection word
                ;
                CONOUT$Entry2:                  ;<=== output error message entry point
                ;
03E8 116400             LXI     D,CB$Device$Table$Addresses     ;Addresses of dev. tables
03EB D5                 PUSH    D                               ;Put onto stack ready for loop
03EC E5                 PUSH    H

                CONOUT$Next$Device:
03ED E1                 POP     H                       ;Recover redirection bit map
03EE D1                 POP     D                       ;Recover device table addresses pointer
03EF CD6F06             CALL    Select$Device$Table     ;Get device table in DE
03F2 B7                 ORA     A                       ;Check if a device has been
                                                        ;  selected (i.e. bit map not all zero)
03F3 CA0D04             JZ      CONOUT$Exit             ;No, exit
03F6 C5                 PUSH    B       ;Yes - B..      ;Save redirection bit map
03F7 E5                 PUSH    H                       ;Save device table addresses pointer
                CONOUT$Wait:
03F8 CD0F06             CALL    Check$Output$Ready      ;Check if device not suspended and
                                                        ;  (if appropriate) DTR is high
03FB CAF803             JZ      CONOUT$Wait             ;No, wait

03FE F3                 DI                              ;Interrupts off to avoid
                                                        ;  involuntary re-entrance
03FF 3ACB03             LDA     CONOUT$Character        ;Recover the data byte
0402 4F                 MOV     C,A                     ;Ready for output
0403 CD2608             CALL    Output$Data$Byte        ;Output the data byte
0406 FB                 EI

0407 CD3A06             CALL    Process$Etx$Protocol    ;Deal with Etx/Ack protocol
040A C3ED03             JMP     CONOUT$Next$Device      ;Loop back for next device

                CONOUT$Exit:
040D 3ACB03             LDA     CONOUT$Character        ;Recover data character
0410 79                 MOV     A,C                     ;CP/M "convention"
0411 C9                 RET
                ;
                CONOUT$Escape$Found:                    ;Possible escape sequence
0412 211904             LXI     H,CONOUT$Process$Escape ;Vector processing of next character
                CONOUT$Set$Processor:
0415 22CC03             SHLD    CONOUT$Processor        ;Set vector address
0418 C9                 RET                             ;Return to BIOS caller
                ;#
                ;
                ;       Console output: escape sequence processing
```

Figure 8-10. (Continued)

```
                ;
                CONOUT$Process$Escape:                  ;Control arrives here with character
                                                        ;  after escape in C
0419 211B02             LXI     H,CONOUT$Escape$Table   ;Get base of recognition table
                CONOUT$Next$Entry:
041C 7E                 MOV     A,M                     ;Check if at end of table
041D B7                 ORA     A
041E CA2B04             JZ      CONOUT$No$Match         ;Yes, no match found
0421 B9                 CMP     C                       ;Compare to data character
0422 CA3B04             JZ      CONOUT$Match            ;They match
0425 23                 INX     H                       ;Move to next entry in table
0426 23                 INX     H
0427 23                 INX     H
0428 C31C04             JMP     CONOUT$Next$Entry       ;Go back and check again
                ;
                CONOUT$No$Match:                        ;No match found, so original
                                                        ;  escape and following character
                                                        ;  must be output
042B C5                 PUSH    B                       ;Save character after escape
042C 0E1B               MVI     C,Function$Key$Lead     ;Get escape character
042E CDE103             CALL    CONOUT$Forced           ;Output to console devices
0431 C1                 POP     B                       ;Get character after escape
0432 CDE103             CALL    CONOUT$Forced           ;Output it, too
                ;
                CONOUT$Set$Normal:
0435 21DB03             LXI     H,CONOUT$Normal         ;Set vector back to normal
0438 C31504             JMP     CONOUT$Set$Processor    ;  for subsequent characters
                ;

                CONOUT$Match:
043B 23                 INX     H                       ;HL -> LS byte of subprocessor
043C 5E                 MOV     E,M                     ;Get LS byte
043D 23                 INX     H
043E 56                 MOV     D,M                     ;Get MS byte
043F EB                 XCHG                            ;HL -> subprocessor
0440 E9                 PCHL                            ;Goto subprocessor
                ;
                CONOUT$Date:                    ;Subprocessor to inject current date
                                                ;  into console input stream (using
                                                ;  forced input)
0441 218F0F             LXI     H,Date
                CONOUT$Set$Forced$Input:
0444 228D0F             SHLD    CB$Forced$Input
0447 C9                 RET                     ;Return to BIOS' caller
                ;
                CONOUT$Time:                    ;Subprocessor to inject time into
                                                ;  console input stream
0448 21990F             LXI     H,Time$In$ASCII
044B C34404             JMP     CONOUT$Set$Forced$Input
                ;
                CONOUT$Set$Date:                ;Subprocessor to set the date by taking
                                                ;  the next 8 characters of console output
                                                ;  and storing them in the date string
044E 21A30F             LXI     H,Time$Date$Flags       ;Set flag to indicate that the
0451 3E02               MVI     A,Date$Set              ;  date has been set by program
0453 B6                 ORA     M
0454 77                 MOV     M,A
0455 3E08               MVI     A,8                     ;Set character count
0457 218F0F             LXI     H,Date                  ;Set address
045A C36C04             JMP     CONOUT$Set$String$Pointer
                ;
                ;
                CONOUT$Set$Time:                ;Subprocessor to set the time by taking
                                                ;  the next 8 characters of console output
                                                ;  and storing them in the time string
045D 21A30F             LXI     H,Time$Date$Flags       ;Set flag to indicate that the
0460 3E01               MVI     A,Time$Set              ;  time has been set by program
0462 B6                 ORA     M
0463 77                 MOV     M,A
0464 3E08               MVI     A,8                     ;Set character count
0466 21990F             LXI     H,Time$in$ASCII         ;Set address
0469 C36C04             JMP     CONOUT$Set$String$Pointer
                ;
                CONOUT$Set$String$Pointer:              ;HL -> string, A = count
046C 32D003             STA     CONOUT$String$Length    ;Save count
046F 22CE03             SHLD    CONOUT$String$Pointer   ;Save address
0472 217804             LXI     H,CONOUT$Process$String ;Vector further output
```

Figure 8-10. (Continued)

```
0475 C31504                 JMP     CONOUT$Set$Processor
                    ;
                    CONOUT$Process$String:              ;Control arrives here for each character
                                                        ;  in the string in register C. The
                                                        ;  characters are stacked into the
                                                        ;  receiving string until either a 00-byte
                                                        ;  is encountered or the specified number
                                                        ;  of characters is stacked.
0478 2ACE03                 LHLD    CONOUT$String$Pointer   ;Get current address for stacking chars
047B 79                     MOV     A,C                     ;Check if current character is 00H
047C B7                     ORA     A
047D CA3504                 JZ      CONOUT$Set$Normal       ;Revert to normal processing
0480 77                     MOV     M,A                     ;Otherwise, stack character
0481 23                     INX     H                       ;Update pointer
0482 3600                   MVI     M,00H                   ;Stack fail-safe terminator
0484 22CE03                 SHLD    CONOUT$String$Pointer   ;Save updated pointer
0487 21D003                 LXI     H,CONOUT$String$Length  ;Downdate count
048A 35                     DCR     M
048B CA3504                 JZ      CONOUT$Set$Normal       ;Revert to normal processing
                                                            ;  if count hits 0
048E C9                     RET                             ;Return with output vectored back
                                                            ; to CONOUT$Process$String
                    ;
                    ;#
                    ;
                    ;       Auxiliary input status
                    ;
                    ;       This routine checks the character count in the
                    ;       appropriate input buffer.
                    ;       The A register is set to indicate whether or not
                    ;       data is waiting.
                    ;
                    ;       Entry parameters: none.
                    ;
                    ;       Exit parameters
                    ;
                    ;               A = 000H if there is no data waiting
                    ;               A = 0FFH if there is data waiting
                    ;
                    ;=========================
                    AUXIST:                         ;<=== BIOS entry point (Private)
                    ;=========================
048F 2A5C00                 LHLD    CB$Auxiliary$Input          ;Get redirection word
0492 116400                 LXI     D,CB$Device$Table$Addresses ;  and table pointer
0495 CD6F06                 CALL    Select$Device$Table     ;Get device table address
0498 C34708                 JMP     Get$Input$Status        ;Get status from input device
                                                            ;  and return to caller
                    ;
                    ;#
                    ;
                    ;       Auxiliary output status
                    ;
                    ;       This routine sets the A register to indicate whether the
                    ;       Auxiliary device(s) is/are ready to accept output data.
                    ;       As more than one device can be used for auxiliary output, this
                    ;       routine returns a Boolean AND of all of their statuses.
                    ;
                    ;       Entry parameters: none
                    ;
                    ;       Exit parameters
                    ;
                    ;               A = 000H if one or more list devices are not ready
                    ;               A = 0FFH if all list devices are ready
                    ;
                    ;=====================
                    AUXOST:                         ;<=== BIOS entry point (Private)
                    ;=====================
049B 2A5E00                 LHLD    CB$Auxiliary$Output         ;Get list redirection word
049E C37905                 JMP     Get$Composite$Status
                    ;
                    ;#
                    ;
                    ;       Auxiliary input (replacement for READER)
                    ;
                    ;       This routine returns the next input character from the
```

Figure 8-10. (Continued)

```
                ;       appropriate logical auxiliary device.
                ;
                ;       Entry parameters: none.
                ;
                ;       Exit parameters
                ;
                ;               A = data character
                ;
                ;=========================
                AUXIN:                          ;<=== BIOS entry point (standard)
                ;=========================
04A1 2A5C00             LHLD    CB$Auxiliary$Input              ;Get redirection word
04A4 116400             LXI     D,CB$Device$Table$Addresses     ; and table pointer
04A7 CD6F06             CALL    Select$Device$Table     ;Get device table address
04AA C39106             JMP     Get$Input$Character     ;Get next input character
                                                        ; and return to caller
                ;
                ;#
                ;       Auxiliary output (replaces PUNCH)
                ;
                ;       This routine outputs a data byte to the auxiliary device(s).
                ;       It is similar to CONOUT except that it uses the watchdog
                ;       timer to detect if a device stays busy for more than
                ;       30 seconds at a time. It outputs a message to the console
                ;       if this happens.
                ;
                ;       Entry parameters
                ;
                ;               C = data byte
                ;
04AD 0D0A074175 AUXOUT$Busy$Message:    DB      CR,LF,7,'Auxiliary device not Ready?',CR,LF,0
                ;
                ;=====================
                AUXOUT:                         ;<=== BIOS entry point (standard)
                ;=====================
04CE 2A5E00             LHLD    CB$Auxiliary$Output     ;Get aux. redirection word
04D1 11AD04             LXI     D,AUXOUT$Busy$Message   ;Message to be output if time
                                                        ;  runs out
04D4 C3A205             JMP     Multiple$Output$Byte
                ;
                ;#
                ;
                ;       List status
                ;
                ;       This routine sets the A register to indicate whether the
                ;       List Device(s) is/are ready to accept output data.
                ;       As more than one device can be used for list output, this
                ;       routine returns a Boolean AND of all of their statuses.
                ;
                ;       Entry parameters: none
                ;
                ;       Exit parameters
                ;
                ;               A = 00OH if one or more list devices are not ready
                ;               A = 0FFH if all list devices are ready
                ;
                ;
                ;=====================
                LISTST:                                 ;<=== BIOS entry point (standard)
                ;=====================
04D7 2A6200             LHLD    CB$List$Output          ;Get list redirection word
04DA C37905             JMP     Get$Composite$Status
                ;
                ;#
                ;       List output
                ;
                ;       This routine outputs a data byte to the list device.
                ;       It is similar to CONOUT except that it uses the watchdog
                ;       timer to detect if the printer stays busy for more
                ;       than 30 seconds at a time. It outputs a message to the console
                ;       if this happens.
                ;
                ;       Entry parameters
                ;
                ;               C = data byte
                ;
```

Figure 8-10. (Continued)

```
04DD 0D0A075072     LIST$Busy$Message:      DB      CR,LF,7,'Printer not Ready?',CR,LF,0
                    ;
                    ;=====================
                    LIST:                           ;<=== BIOS entry point (standard)
                    ;=====================
04F5 2A6200                 LHLD    CB$List$Output          ;Get list redirection word
04F8 11DD04                 LXI     D,LIST$Busy$Message     ;Message to be output if time
                                                            ;  runs out
04FB C3A205                 JMP     Multiple$Output$Byte
                    ;
                    ;#
                    ;       Request user choice
                    ;
                    ;       This routine displays an error message, requesting
                    ;       a choice of:
                    ;
                    ;               R -- Retry the operation that caused the error
                    ;               I -- Ignore the error and attempt to continue
                    ;               A -- Abort the program and return to CP/M
                    ;
                    ;       This routine accepts a character from the console,
                    ;       converts it to uppercase and returns to the caller
                    ;       with the response in the A register.
                    ;
                    RUC$Message:
04FE 0D0A                   DB      CR,LF
0500 2020202020             DB      '     Enter R - Retry, I - Ignore, A - Abort : ',0
                    ;
                    ;
                    Request$User$Choice:
052F CD2D03                 CALL    CONST                   ;Gobble up any type-ahead
0532 CA3B05                 JZ      RUC$Buffer$Empty
0535 CD3A03                 CALL    CONIN
0538 C32F05                 JMP     Request$User$Choice

                    RUC$Buffer$Empty:
053B 21FE04                 LXI     H,RUC$Message           ;Display prompt
053E CD5305                 CALL    Output$Error$Message

0541 CD3A03                 CALL    CONIN                   ;Get console character
0544 CD3B0E                 CALL    A$To$Upper              ;Make uppercase for comparisons
0547 32B00D                 STA     Disk$Action$Confirm     ;Save in confirmatory message
054A F5                     PUSH    PSW                     ;Save for later

054B 21B00D                 LXI     H,Disk$Action$Confirm
054E CD5305                 CALL    Output$Error$Message

0551 F1                     POP     PSW                     ;Recover action code
0552 C9                     RET
                    ;
                    ;#
                    ;
                    ;       Output error message
                    ;
                    ;       This routine outputs an error message to all the currently
                    ;       selected console devices except those being used to receive
                    ;       LIST output as well. This is to avoid "deadly embrace" situations
                    ;       where the printer's being busy for too long causes an error message
                    ;       to be output -- and console output is being directed to the
                    ;       printer as well.
                    ;
                    ;       This subroutine makes use of most of the CONOUT subroutine.
                    ;       For memory economy it enters CONOUT using a private
                    ;       entry point.
                    ;
                    ;       Entry parameters
                    ;
                    ;               HL -> 00-byte terminated error message
                    ;
                    Output$Error$Message:
0553 E5                     PUSH    H                       ;Save message address
0554 2A5A00                 LHLD    CB$Console$Output       ;Get console redirection bit map
0557 EB                     XCHG
0558 2A6200                 LHLD    CB$List$Output          ;Get list redirection bit map
                                                            ;HL = list, DE = console
                                                            ;Now set to 0 all bits in the console
```

Figure 8-10. (Continued)

```
                                                ;  bit map that are set to 1 in the
                                                ;  list bit map
055B 7C                 MOV     A,H             ;Get MS byte of list
055C 2F                 CMA                     ;Invert
055D A2                 ANA     D               ;Preserve only bits with 0's
055E 67                 MOV     H,A             ;Save result
055F 7D                 MOV     A,L             ;Repeat for LS byte of list
0560 2F                 CMA
0561 A3                 ANA     E
0562 6F                 MOV     L,A             ;HL now has only pure console
                                                ;  devices
0563 B4                 ORA     H               ;Ensure that at least one device
0564 CA6A05             JZ      OEM$Device$Present  ;  is selected
0567 210100             LXI     H,0001H         ;Otherwise use default of device 0
                OEM$Device$Present:
                OEM$Next$Character:
056A D1                 POP     D               ;Recover message address into DE
056B 1A                 LDAX    D               ;Get next byte of message
056C 13                 INX     D               ;Update message pointer
056D B7                 ORA     A               ;Check if end of message
056E C8                 RZ                      ;Yes, exit
056F D5                 PUSH    D               ;Save message address for later
0570 E5                 PUSH    H               ;Save special bit map
                                                ;Data character is in A
0571 CDD103             CALL    CONOUT$OEM$Entry    ;Enter shared code
0574 E1                 POP     H               ;Recover special bit map
0575 C36A05             JMP     OEM$Next$Character
                ;
                ;
                ;
                ;       Get composite status
                ;
                ;       This routine sets the A register to indicate whether the
                ;       output device(s) is/are ready to accept output data.
                ;       As more than one device can be used for output, this
                ;       routine returns a Boolean AND of all of their statuses.
                ;
                ;       Entry parameters
                ;
                ;               HL = I/O redirection bit map for output device(s)
                ;
                ;       Exit parameters
                ;
                ;               A = 000H if one or more list devices are not ready
                ;               A = 0FFH if all list devices are ready
                ;
0578 00         GCS$Status:     DB      0       ;Composite status of all devices
                ;
                ;
                Get$Composite$Status:
0579 3EFF               MVI     A,0FFH          ;Assume all devices are ready
057B 327805             STA     GCS$Status      ;Preset composite status byte

057E 116400             LXI     D,CB$Device$Table$Addresses     ;Addresses of dev. tables
0581 D5                 PUSH    D               ;Put onto stack ready for loop
0582 E5                 PUSH    H               ;Save bit map
                GCS$Next$Device:
0583 E1                 POP     H               ;Recover redirection bit map
0584 D1                 POP     D               ;Recover device table addresses pointer
0585 CD6F06             CALL    Select$Device$Table ;Get device table in DE
0588 B7                 ORA     A               ;Check if a device has been
                                                ;  selected (i.e. bit map not all zero)
0589 CA9905             JZ      GCS$Exit        ;No, exit
058C C5                 PUSH    B       ;Yes - B..  ;Save redirection bit map
058D E5                 PUSH    H               ;Save device table addresses pointer
058E CD0F06             CALL    Check$Output$Ready  ;Check if device ready
0591 217805             LXI     H,GCS$Status    ;AND together with previous devices
0594 A6                 ANA     M               ;  status
0595 77                 MOV     M,A             ;Save composite status

0596 C38305             JMP     GCS$Next$Device ;Loop back for next device
                ;
                GCS$Exit:
0599 3A7805             LDA     GCS$Status      ;Return with composite status
059C B7                 ORA     A
059D C9                 RET
```

Figure 8-10. (Continued)

```
                 ;
                 ;#
                 ;
                 ;       Multiple output byte
                 ;
                 ;       This routine outputs a data byte to the all of the
                 ;       devices specified in the I/O redirection word.
                 ;       It is similar to CONOUT except that it uses the watchdog
                 ;       timer to detect if any of the devices stays busy for more
                 ;       than 30 seconds at a time. It outputs a message to the console
                 ;       if this happens.
                 ;
                 ;       Entry parameters
                 ;
                 ;               HL = I/O redirection bit map
                 ;               DE -> Message to be output if time runs out
                 ;               C = data byte
                 ;
0708 =           MOB$Maximum$Busy        EQU     1800    ;Number of clock ticks (each at
                                                         ;  16.666 milliseconds) for which the
                                                         ;  device might be busy
059E 00          MOB$Character:          DB      0       ;Character to be output
059F 0000        MOB$Busy$Message:       DW      0       ;Address of message to be
                                                         ;  output if time runs out
05A1 00          MOB$Need$Message:       DB      0       ;Flag used to detect that the
                                                         ;  watchdog timer timed out
                 ;
                 Multiple$Output$Byte:
05A2 79                  MOV     A,C                     ;Get data byte
05A3 320807              STA     MOB$Maximum$Busy        ;Save copy
05A6 EB                  XCHG                            ;HL -> timeout message
05A7 229F05              SHLD    MOB$Busy$Message        ;Save for later use
05AA EB                  XCHG                            ;HL = bit map again

05AB 116400              LXI     D,CB$Device$Table$Addresses     ;Addresses of dev. tables
05AE D5                  PUSH    D                       ;Save on stack ready for loop
05AF E5                  PUSH    H                       ;Save I/O redirection bit map
                 MOB$Next$Device:
05B0 E1                  POP     H                       ;Recover redirection bit map
05B1 D1                  POP     D                       ;Recover device table addresses pointer
05B2 CD6F06              CALL    Select$Device$Table     ;Get device table in DE
05B5 B7                  ORA     A                       ;Check if any device selected
05B6 CAEC05              JZ      MOB$Exit

05B9 C5                  PUSH    B       ;<- Yes : B     ;Save device table addresses pointer
05BA E5                  PUSH    H                       ;Save redirection bit map
                 ;
                 MOB$Start$Watchdog:
05BB AF                  XRA     A                               ;Reset message needed flag
05BC 32A105              STA     MOB$Need$Message
05BF 010807              LXI     B,MOB$Maximum$Busy              ;Time delay
05C2 210906              LXI     H,MOB$Not$Ready                 ;Address to go to
05C5 CD6D08              CALL    Set$Watchdog                    ;Start timer

                 MOB$Wait:
05C8 3AA105              LDA     MOB$Need$Message                ;Check if watchdog timed out
05CB B7                  ORA     A
05CC C2EE05              JNZ     MOB$Output$Message              ;Yes, output warning message
05CF CD0F06              CALL    Check$Output$Ready              ;Check if device ready
05D2 CAC805              JZ      MOB$Wait                        ;No, wait
                 ;
05D5 F3                  DI                                      ;Interrupts off to avoid
                                                                 ;  involuntary reentrance
05D6 010000              LXI     B,0                             ;Turn off watchdog
05D9 CD6D08              CALL    Set$Watchdog                    ; (HL setting is irrelevant)

05DC 3A9E05              LDA     MOB$Character                   ;Get data byte
05DF 4F                  MOV     C,A
05E0 CD2608              CALL    Output$Data$Byte                ;Output the data byte
05E3 FB                  EI
05E4 CD3A06              CALL    Process$Etx$Protocol            ;Deal with ETX/ACK protocol
05E7 C3B005              JMP     MOB$Next$Device
                 ;
                 MOB$Ignore$Exit:                                ;Ignore timeout error
05EA E1                  POP     H                               ;Balance the stack
05EB D1                  POP     D
```

Figure 8-10. (Continued)

```
                ;
                MOB$Exit:
05EC 79                 MOV     A,C                             ;CP/M "convention"
05ED C9                 RET
                ;
                MOB$Output$Message:
05EE 2A9F05             LHLD    MOB$Busy$Message                ;Display warning message
05F1 CD5305             CALL    Output$Error$Message            ;  on selected console devices
                MOB$Request$Choice:
05F4 CD2F05             CALL    Request$User$Choice             ;Display message and get
                                                                ;  action character
05F7 FE52               CPI     'R'                             ;Retry
05F9 CABB05             JZ      MOB$Start$Watchdog              ;Restart watchdog and try again
05FC FE49               CPI     'I'                             ;Ignore
05FE CAEA05             JZ      MOB$Ignore$Exit
0601 FE41               CPI     'A'                             ;Abort
0603 CA360E             JZ      System$Reset                    ;  Give BDOS function 0
0606 C3F405             JMP     MOB$Request$Choice
                ;
                MOB$Not$Ready:                          ;Watchdog timer routine will call this
                                                        ;  routine if the device is busy
                                                        ;  for more than approximately 30 seconds
                                                        ;Note: This is an interrupt service routine
0609 3EFF               MVI     A,0FFH                  ;Set request to output message
060B 32A105             STA     MOB$Need$Message
060E C9                 RET                             ;Return to the watchdog routine
                ;
                ;#
                ;       Check output ready
                ;
                ;       This routine checks to see if the specified device is ready
                ;       to receive output data.
                ;       It does so by checking to see if the device has been suspended
                ;       for protocol reasons and if DTR is low.
                ;
                ;       NOTE: This routine does NOT check if the USART itself is ready.
                ;             This test is done in the output data byte routine itself.
                ;
                ;       Entry parameters
                ;
                ;               DE -> device table
                ;
                ;       Exit parameters
                ;
                ;               A = 000H (Zero-flag set)   : Device not ready
                ;               A = 0FFH (Zero-flag clear) : Device ready
                ;
                Check$Output$Ready:
060F 210E00             LXI     H,DT$Status             ;Get device status
0612 19                 DAD     D                       ;HL -> status byte
0613 7E                 MOV     A,M                     ;Get status byte
0614 47                 MOV     B,A                     ;Take a copy of the status byte
0615 E601               ANI     DT$Output$Suspend       ;Check if output is suspended
0617 C23806             JNZ     COR$Not$Ready           ;Yes, indicate not ready

061A 3E04               MVI     A,DT$Output$DTR         ;Check if DTR must be high to send
061C A0                 ANA     B                       ;Mask with device status from table
061D CA3406             JZ      COR$Ready               ;No, device is logically ready

0620 210000             LXI     H,DT$Status$Port        ;Set up to read device status
0623 19                 DAD     D
0624 7E                 MOV     A,M                     ;Get status port number
0625 322906             STA     COR$Status$Port         ;Set up instruction below

0628 DB                 DB      IN
                COR$Status$Port:
0629 00                 DB      0               ;<-- Set up by instruction above
062A 4F                 MOV     C,A                     ;Save hardware status

062B 210400             LXI     H,DT$DTR$Ready          ;Yes, set up to check chip status
062E 19                 DAD     D                       ;  to see if DTR is high
062F 7E                 MOV     A,M                     ;Get DTR high status mask
0630 A1                 ANA     C                       ;Test chip status
0631 CA3806             JZ      COR$Not$Ready           ;DTR low, indicate not ready
                ;
                COR$Ready:
```

Figure 8-10. (Continued)

```
0634 3EFF               MVI     A,0FFH          ;Indicate device ready for output
0636 B7                 ORA     A
0637 C9                 RET
                ;
                COR$Not$Ready:                  ;Indicate device not ready for output
0638 AF                 XRA     A
0639 C9                 RET
                ;
                ;#
                ;
                ;       Process ETX/ACK protocol
                ;
                ;       This routine maintains ETX/ACK protocol.
                ;       After a specified number of data characters have been output
                ;       to the device, an ETX character is output and the device
                ;       put into output suspended state. Only when an incoming
                ;       ACK character is received (under interrupt control) will
                ;       output be resumed to the device.
                ;
                ;       Entry parameters
                ;
                ;               DE -> device table
                ;
                ;       Exit parameters
                ;
                ;               Message count downdated (and reset if necessary)
                ;
                Process$Etx$Protocol:
063A 210E00             LXI     H,DT$Status             ;Check if ETX/ACK protocol enabled
063D 19                 DAD     D
063E 7E                 MOV     A,M
063F E610               ANI     DT$Output$Etx
0641 C8                 RZ                              ;No, so return immediately
0642 211000             LXI     H,DT$Etx$Count          ;Yes, so downdate count
0645 19                 DAD     D
0646 E5                 PUSH    H                       ;Save address of count for later
0647 4E                 MOV     C,M                     ;Get LS byte
0648 23                 INX     H
0649 46                 MOV     B,M                     ;Get MS byte
064A 0B                 DCX     B
064B 78                 MOV     A,B
064C B1                 ORA     C                       ;Check if count now zero
064D C25706             JNZ     PEP$Save$Count          ;No
0650 211200             LXI     H,DT$Etx$Message$Length ;Yes, reset to message length
0653 19                 DAD     D
0654 4E                 MOV     C,M                     ;Get LS byte
0655 23                 INX     H
0656 46                 MOV     B,M                     ;Get MS byte
                PEP$Save$Count:
0657 E1                 POP     H                       ;Recover address of count
0658 71                 MOV     M,C                     ;Save count back in table
0659 23                 INX     H
065A 70                 MOV     M,B
                ;
065B B7                 ORA     A                       ;Reestablish whether count hit 0
065C C0                 RNZ                             ;No, no further processing required
065D 0E03               MVI     C,ETX                   ;Yes, send ETX to device
065F F3                 DI                              ;Avoids involuntary reentrance
0660 CD2608             CALL    Output$Data$Byte
0663 FB                 EI
0664 210E00             LXI     H,DT$Status             ;Flag device as output suspended
0667 19                 DAD     D
0668 F3                 DI                              ;Avoid interaction with interrupts
0669 7E                 MOV     A,M                     ;Get status byte
066A F601               ORI     DT$Output$Suspend       ;Set bit
066C 77                 MOV     M,A                     ;Save back in table
066D FB                 EI
066E C9                 RET
                ;
                ;#
                ;
                ;       Select device table
                ;
                ;       This routine scans a 16-bit word, and depending on which is the
                ;       first 1-bit set, selects the corresponding device table address.
                ;
```

Figure 8-10. (Continued)

```
                ;       Entry parameters
                ;
                ;               HL = Bit map
                ;               DE -> Table of device table addresses
                ;                       The first address in the list is called
                ;                       if the least significant bit of the bit map is
                ;                       nonzero, and so on.
                ;
                ;       Exit parameters
                ;
                ;               BC -> Current entry in device table addresses
                ;               DE = Selected device table address
                ;               HL = Shifted bit map
                ;                    Nonzero if a 1-bit was found
                ;                    Zero if bit map now entirely 0000
                ;
                ;       Note: If HL is 0000H on input, then the first entry in the
                ;       device table addresses will be returned in DE.
                ;
                Select$Device$Table:
066F 7C                 MOV     A,H             ;Get most significant byte of bit map
0670 B5                 ORA     L               ;Check if HL completely 0
0671 C8                 RZ                      ;Return indicating no more bits set
0672 7D                 MOV     A,L             ;Check if the LS bit is nonzero
0673 E601               ANI     1
0675 C28006             JNZ     SDT$Bit$Set     ;Yes, return corresponding address
0678 13                 INX     D               ;No, update table pointer
0679 13                 INX     D
067A CDDB08             CALL    SHLR            ;Shift HL right one bit
067D C36F06             JMP     Select$Device$Table     ;Check next bit
                SDT$Bit$Set:
0680 E5                 PUSH    H               ;Save shifted bit map
0681 42                 MOV     B,D             ;Take copy of table pointer
0682 4B                 MOV     C,E
0683 EB                 XCHG                    ;HL -> address in table
0684 5E                 MOV     E,M
0685 23                 INX     H
0686 56                 MOV     D,M             ;DE -> selected device table
                                                ;Set up registers for another
                                                ;  entry
0687 E1                 POP     H               ;Recover shifted bit map
0688 CDDB08             CALL    SHLR            ;Shift bit map right one bit
068B 03                 INX     B               ;Update DT address table pointer to
068C 03                 INX     B               ;  entry
068D 3E01               MVI     A,1             ;Indicate that a one bit was found
068F B7                 ORA     A               ;  and registers are set up correctly
0690 C9                 RET
                ;
                ;#
                ;
                ;       Get input character
                ;
                ;       This routine gets the next input character from the device
                ;       specified in the device table handed over as an input
                ;       parameter.
                ;
                Get$Input$Character:
0691 211900             LXI     H,DT$Character$Count    ;Check if any characters have
0694 19                 DAD     D                       ;  been stored in the buffer
                GIC$Wait:
0695 FB                 EI                              ;Ensure that incoming chars. will
                                                        ;  be detected
0696 7E                 MOV     A,M                     ;Get character count
0697 B7                 ORA     A
0698 CA9506             JZ      GIC$Wait                ;No characters, so wait
069B 35                 DCR     M                       ;Down date character count for
                                                        ;  the character about to be
                                                        ;  removed from the buffer
069C 211700             LXI     H,DT$Get$Offset         ;Use the get offset to access
069F CDF007             CALL    Get$Address$in$Buffer   ;Returns HL -> character
                                                        ;  and with get offset updated
06A2 7E                 MOV     A,M                     ;Get the actual data character
06A3 F5                 PUSH    PSW                     ;Save until later

06A4 211900             LXI     H,DT$Character$Count    ;Check downdated count of chars. in
06A7 19                 DAD     D                       ;  buffer, checking if input should be
```

Figure 8-10. (Continued)

```
0702 11CE02          LXI     D,DT$2          ;Device 2
0705 CD1607          CALL    Service$Device
0708 3E20            MVI     A,IC$EOI        ;Tell the interrupt controller chip
070A D3D8            OUT     IC$OCW2$Port    ;  that the interrupt has been serviced
070C D1              POP     D               ;Restore registers
070D C1              POP     B
070E F1              POP     PSW
070F 2A8422          LHLD    PI$User$Stack   ;Switch back to user's stack
0712 F9              SPHL
0713 E1              POP     H
0714 FB              EI                      ;Reenable interrupts in the CPU
0715 C9              RET                     ;Resume pre-interrupt processing
             ;
             ;#
             ;
             ;       Service device
             ;
             ;       This routine performs the device interrupt servicing,
             ;       checking to see if the device described in the specified
             ;       device table (address in DE) is actually interrupting,
             ;       and if so, inputs the character. Depending on which data character
             ;       is input, this routine will either stack it in the input buffer
             ;       (shutting off the input stream if the buffer is nearly full),
             ;       or will suspend or resume the output to the device.
             ;
             ;       Entry parameters
             ;
             ;               DE -> device table
             ;
             Service$Device:
0716 210000          LXI     H,DT$Status$Port          ;Check if this device is really
0719 19              DAD     D                         ;  interrupting
071A 7E              MOV     A,M                       ;Get status port number
071B 321F07          STA     SD$Status$Port            ;Store in instruction below

071E DB              DB      IN                        ;Input status
             SD$Status$Port:
071F 00              DB      0       ;<-- Set up by instruction above
             ;
0720 210300          LXI     H,DT$Input$Ready          ;Check if status indicates data ready
0723 19              DAD     D
0724 A6              ANA     M                         ;Mask with input ready value
0725 C8              RZ                                ;No, return to interrupt service
                                                       ;Check if any errors have occurred
0726 210700          LXI     H,DT$Detect$Error$Port    ;Set up to read error status
0729 19              DAD     D                         ;  interrupting
072A 7E              MOV     A,M                       ;Get status port number
072B 322F07          STA     SD$Error$Port             ;Store in instruction below

072E DB              DB      IN                        ;Input error status
             SD$Error$Port:
072F 00              DB      0       ;<-- Set up by instruction above
             ;
0730 210800          LXI     H,DT$Detect$Error$Value ;Mask with error bit(s)
0733 19              DAD     D
0734 A6              ANA     M
0735 CA4707          JZ      SD$No$Error               ;No bit(s) set
0738 210900          LXI     H,DT$Reset$Error$Port     ;Set up to reset error
073B 19              DAD     D
073C 7E              MOV     A,M                       ;Get reset port number
073D 324607          STA     SD$Reset$Error$Port       ;Store in instruction below
0740 210A00          LXI     H,DT$Reset$Error$Value
0743 19              DAD     D
0744 7E              MOV     A,M                       ;Get reset interrupt value

0745 D3              DB      OUT
             SD$Reset$Error$Port:
0746 00              DB      0       ;<-- Set up in instruction above

             SD$No$Error:
0747 210100          LXI     H,DT$Data$Port            ;Input the data character (this may
074A 19              DAD     D                         ;  be garbled if an error occurred)
074B 7E              MOV     A,M                       ;Get data port number
074C 325007          STA     SD$Data$Port              ;Store in instruction below
```

Figure 8-10. (Continued)

```
                                                             
074F DB                     DB      IN                      ;Input data character
               SD$Data$Port:
0750 00                     DB      0       ;<-- Set up by instruction above
                                                             
0751 47                     MOV     B,A                     ;Take copy of data character above
0752 210E00                 LXI     H,DT$Status             ;Check if either XON or ETX protocols
0755 19                     DAD     D                       ;  is currently active
0756 7E                     MOV     A,M                     ;Get protocol byte
0757 E618                   ANI     DT$Output$Xon + DT$Output$Etx
0759 CA8107                 JZ      SD$No$Protocol          ;Neither is active
075C E608                   ANI     DT$Output$Xon           ;Check if XON/XOFF is active
075E C26E07                 JNZ     SD$Check$if$Xon         ;Yes, check if XON char. input
                                                            ;No, assume ETX/ACK active
0761 3E06                   MVI     A,ACK                   ;Check if input character is ACK
0763 B8                     CMP     B
0764 C28107                 JNZ     SD$No$Protocol          ;No, process character as data
               SD$Output$Desuspend:                         ;Yes, device now ready
                                                            ;  to accept more data, so indicate
                                                            ;  output to device can resume
                                                            ;The noninterrupt driven output
                                                            ;  routine checks the suspend bit
0767 7E                     MOV     A,M                     ;Get status/protocol byte again
0768 E6FE                   ANI     OFFH AND NOT DT$Output$Suspend ;Preserve all bits BUT suspend
076A 77                     MOV     M,A                     ;Save back with suspend = 0
076B C3D907                 JMP     SD$Exit                 ;Exit to interrupt service without
                                                            ;  saving data character
               ;
               SD$Check$if$Xon:                             ;XON/XOFF protocol active, so
                                                            ;  if XOFF received, suspend output
                                                            ;  if XON received, resume output
                                                            ;The noninterrupt driven output
                                                            ;  routine checks the suspend bit
076E 3E11                   MVI     A,XON                   ;Check if XON character input
0770 B8                     CMP     B
0771 CA6707                 JZ      SD$Output$Desuspend     ;Yes, enable output to device
0774 3E13                   MVI     A,XOFF                  ;Check if XOFF character input
0776 B8                     CMP     B
0777 C28107                 JNZ     SD$No$Protocol          ;No, process character as data
               SD$Output$Suspend:                           ;Device needs pause in output of
                                                            ;  data, so indicate output suspended
077A 7E                     MOV     A,M                     ;Get status/protocol byte again
077B F601                   ORI     DT$Output$Suspend       ;Set suspend bit to 1
077D 77                     MOV     M,A                     ;Save back in device table
077E C3D907                 JMP     SD$Exit                 ;Exit to interrupt service without
                                                            ;  saving the input character
               ;
               SD$No$Protocol:
0781 211800                 LXI     H,DT$Buffer$Length$Mask ;Check if there is still space
0784 19                     DAD     D                       ;  in the input buffer
0785 7E                     MOV     A,M                     ;Get length - 1
0786 3C                     INR     A                       ;Update to actual length
0787 211900                 LXI     H,DT$Character$Count    ;Get current count of characters
078A 19                     DAD     D                       ;  in buffer
078B BE                     CMP     M                       ;Check if count = length
078C CAEB07                 JZ      SD$Buffer$Full          ;Yes, output bell character
078F C5                     PUSH    B                       ;Save data character
0790 211600                 LXI     H,DT$Put$Offset         ;Compute address of character in
                                                            ;  input buffer
0793 CDF007                 CALL    Get$Address$In$Buffer   ;HL -> character position
0796 C1                     POP     B                       ;Recover input character
0797 70                     MOV     M,B                     ;Save character in input buffer
                                                    ;Update number of characters in input
                                                    ;  buffer, checking if input should
                                                    ;  be temporarily halted
0798 211900                 LXI     H,DT$Character$Count
079B 19                     DAD     D
079C 34                     INR     M                       ;Update character count
079D 7E                     MOV     A,M                     ;Get updated count
079E 211A00                 LXI     H,DT$Stop$Input$Count   ;Check if current count matches
07A1 19                     DAD     D                       ;  buffer-full threshold
07A2 BE                     CMP     M
07A3 C2CE07                 JNZ     SD$Check$Control        ;Not at threshold, check if control
                                                            ;  character input
07A6 210E00                 LXI     H,DT$Status             ;At threshold, check which means
07A9 19                     DAD     D                       ;  for pausing input are to be used
```

Figure 8-10. (Continued)

```
07AA 7E                      MOV     A,M                     ;Get status/protocol byte
07AB F602                    ORI     DT$Input$Suspend        ;Indicate input is suspended
07AD 77                      MOV     M,A                     ;Save updated status in table
07AE F5                      PUSH    PSW                     ;Save for later use
07AF E640                    ANI     DT$Input$RTS            ;Check if clear to send to be dropped
07B1 CAC307                  JZ      SD$Check$Input$Xon      ;No
07B4 210B00                  LXI     H,DT$RTS$Control$Port   ;Yes, get control port number
07B7 19                      DAD     D
07B8 7E                      MOV     A,M
07B9 32C207                  STA     SD$Drop$RTS$Port                ;Store in instruction below
07BC 210C00                  LXI     H,DT$Drop$RTS$Value
07BF 19                      DAD     D
07C0 7E                      MOV     A,M                     ;Get value needed to drop RTS

07C1 D3                      DB      OUT
                     SD$Drop$RTS$Port:
07C2 00                      DB      0               ;<- Set up in instruction above
                                                             ;Drop into input XON test
                     SD$Check$Input$Xon:                     ;Check if XON/XOFF protocol being used
                                                     ;  to temporarily suspend input
07C3 F1                      POP     PSW                     ;Recover status/protocol byte
07C4 E680                    ANI     DT$Input$Xon            ;Check if XON bit set
07C6 CACE07                  JZ      SD$Check$Control        ;No, see if control char. input
07C9 0E13                    MVI     C,XOFF                  ;Yes, output XOFF character
07CB CD2608                  CALL    Output$Data$Byte        ;Output data byte
                     ;
                     SD$Check$Control:               ;Check if control character (other than
                                                     ;  CR, LF, or TAB) input, and update
                                                     ;  count of control characters in buffer
07CE CD0808                  CALL    Check$Control$Char      ;Check if control character
07D1 CAD907                  JZ      SD$Exit                 ;No,it is not a control character
07D4 211C00                  LXI     H,DT$Control$Count
07D7 19                      DAD     D
07D8 34                      INR     M                       ;Update count of control chars.
                     ;
                     SD$Exit:                        ;Reset hardware interrupt system
07D9 210500                  LXI     H,DT$Reset$Int$Port
07DC 19                      DAD     D
07DD 7E                      MOV     A,M                     ;Get reset port number
07DE B7                      ORA     A                       ;Check if port specified
                                                             ;  (assumes it will always be NZ)
07DF C8                      RZ                              ;Bypass reset if no port specified
07E0 32E907                  STA     SD$Reset$Int$Port       ;Store in instruction below
07E3 210600                  LXI     H,DT$Reset$Int$Value
07E6 19                      DAD     D
07E7 7E                      MOV     A,M                     ;Get reset interrupt value

07E8 D3                      DB      OUT
                     SD$Reset$Int$Port:
07E9 00                      DB      0       ;<-- Set up in instruction above
07EA C9                      RET                             ;Return to interrupt service routine
                     ;
                     SD$Buffer$Full:                         ;Input buffer completely full
07EB 0E07                    MVI     C,BELL                  ;Send bell character as desperate
07ED C32608                  JMP     Output$Data$Byte        ;  measure. Note JMP return to
                                                             ;  caller will be done by subroutine
                     ;
                     ;#
                     ;
                     ;       Get address in buffer
                     ;
                     ;       This routine computes the address of the next character to
                     ;       access in a device buffer.
                     ;
                     ;       Entry parameters
                     ;
                     ;               DE -> appropriate device table
                     ;               HL =  offset in the device table of either the
                     ;                       Get$Offset or the Put$Offset
                     ;
                     ;       Exit parameters
                     ;
                     ;               DE unchanged
                     ;               HL -> address in character buffer
                     ;
                     Get$Address$In$Buffer:
```

Figure 8-10. (Continued)

```
07F0 19                 DAD     D                       ;HL -> get/put offset in dev. table
07F1 E5                 PUSH    H                       ;Preserve pointer to table
07F2 4E                 MOV     C,M                     ;Get offset value
07F3 0600               MVI     B,0                     ;Make into word value
                                                ;Update offset value, resetting to
                                                ;  0 at end of buffer
07F5 79                 MOV     A,C                     ;Get copy of offset
07F6 3C                 INR     A                       ;Update to next position
07F7 211800             LXI     H,DT$Buffer$Length$Mask
07FA 19                 DAD     D
07FB A6                 ANA     M                       ;Mask LS bits with length - 1
07FC E1                 POP     H                       ;Recover pointer to offset in table
07FD 77                 MOV     M,A                     ;Save new value (set to 0 if nec.)
07FE 211400             LXI     H,DT$Buffer$Base        ;Get base address of input buffer
0801 19                 DAD     D                       ;HL -> address of buffer in table
0802 7E                 MOV     A,M                     ;Get LS byte of address
0803 23                 INX     H                       ;HL -> MS byte of address
0804 66                 MOV     H,M                     ;H = MS byte
0805 6F                 MOV     L,A                     ;L = LS byte
0806 09                 DAD     B                       ;Add on offset to base
0807 C9                 RET

                ;
                ;#
                ;
                ;       Check control character
                ;
                ;       This routine checks the character in A to see if it is a
                ;       control character other than CR, LF, or TAB. The result is
                ;       returned in the Z-flag.
                ;
                ;       Entry parameters
                ;
                ;               A = character to be checked
                ;
                ;       Exit parameters
                ;
                ;               Zero status if A does not contain a control character
                ;                       or if it is CR, LF, or TAB
                ;
                ;               Nonzero if A contains a control character other than
                ;                       CR, LF, or TAB.
                Check$Control$Char:
0808 3E1F               MVI     A,' ' - 1               ;Space is first noncontrol char.
080A B8                 CMP     B
080B DA2408             JC      CCC$No                  ;Not a control character
080E 3E0D               MVI     A,CR                    ;Check if carriage return
0810 B8                 CMP     B
0811 CA2408             JZ      CCC$No                  ;Not really a control character
0814 3E0A               MVI     A,LF                    ;Check if LF
0816 B8                 CMP     B
0817 CA2408             JZ      CCC$No                  ;Not really a control character
081A 3E09               MVI     A,TAB                   ;Check if horizontal tab
081C B8                 CMP     B
081D CA2408             JZ      CCC$No                  ;Not really a control character
0820 3E01               MVI     A,1                     ;Indicate a control character
0822 B7                 ORA     A
0823 C9                 RET
                CCC$No:                                 ;Indicate A does not contain
0824 AF                 XRA     A                       ;  a control character
0825 C9                 RET
                ;
                ;#
                ;
                ;       Output data byte
                ;
                ;       This is a simple polled output routine that outputs a single
                ;       character (in register C on entry) to the device specified in
                ;       the device table.
                ;       Preferably, this routine would have been re-entrant; however
                ;       it does have to store the port numbers. Therefore, to use it
                ;       from code executed with interrupts enabled, the instruction
                ;       sequence must be:
                ;
                ;               DI              ;Interrupts off
                ;               CALL    Output$Data$Byte
```

Figure 8-10. (Continued)

```
                ;               EI              ;Interrupts on
                ;
                ;       Failure to do this may cause involuntary re-entrance.
                ;
                ;       Entry parameters
                ;
                ;               C = character to be output
                ;               DE -> device table
                ;
                Output$Data$Byte:
0826 C5                 PUSH    B                               ;Save registers
0827 210200             LXI     H,DT$Output$Ready               ;Get output ready status mask
082A 19                 DAD     D
082B 46                 MOV     B,M
082C 210000             LXI     H,DT$Status$Port                ;Get status port number
082F 19                 DAD     D
0830 7E                 MOV     A,M
0831 323508             STA     ODB$Status$Port                 ;Store in instruction below
                ODB$Wait$until$Ready:

0834 DB                 DB      IN                              ;Read status
                ODB$Status$Port:
0835 00                 DB      0       ;<--  Set up in instruction above

0836 A0                 ANA     B                               ;Check if ready for output
0837 CA3408             JZ      ODB$Wait$until$Ready            ;No
083A 210100             LXI     H,DT$Data$Port                  ;Get data port
083D 19                 DAD     D
083E 7E                 MOV     A,M
083F 324408             STA     ODB$Data$Port                   ;Store in instruction below
0842 79                 MOV     A,C                             ;Get character to output

0843 D3                 DB      OUT
                ODB$Data$Port:
0844 00                 DB      0       ;<--  Set up in instruction above

0845 C1                 POP     B                               ;Restore registers
0846 C9                 RET
                ;
                ;#
                ;
                ;
                ;       Input status routine
                ;
                ;       This routine returns a value in the A register indicating whether
                ;       one or more data characters is/are waiting in the input buffer.
                ;       Some products, such as Microsoft BASIC, defeat normal type-ahead
                ;       by constantly "gobbling" characters in order to see if an incoming
                ;       Control-S, -Q or -C has been received. In order to preserve
                ;       type-ahead under these circumstances, the input status return
                ;       can, as an option selected by the user, return "data waiting" only
                ;       if the input buffer contains a Control-S, -Q or -C. This fools
                ;       Microsoft BASIC into allowing type-ahead.
                ;
                ;       Entry parameters
                ;
                ;               DE -> device table
                ;
                ;       Exit parameters
                ;
                ;               A = 000H if no characters are waiting in the input
                ;                       buffer
                ;
                ;
                Get$Input$Status:
0847 210F00             LXI     H,DT$Status$2                   ;Check if fake mode enabled
084A 19                 DAD     D                               ;HL -> status byte in table
084B 7E                 MOV     A,M                             ;Get status byte
084C E601               ANI     DT$Fake$Typeahead               ;Isolate status bit
084E CA5B08             JZ      GIS$True$Status                 ;Fake mode disabled
                                                        ;
                                                        ;Fake mode -- only indicates data
                                                        ;ready if control chars. in buffer
0851 211C00             LXI     H,DT$Control$Count              ;Check if any control characters
0854 19                 DAD     D                               ;  in the input buffer
0855 AF                 XRA     A                               ;Cheap 0
```

Figure 8-10. (Continued)

```
0856 B6                  ORA     M                      ;Set flags according to count
0857 C8                  RZ                             ;Return indicating zero
                 GIS$Data$Ready:
0858 AF                  XRA     A                      ;Cheap 0
0859 3D                  DCR     A                      ;Set A = OFFH and flags NZ
085A C9                  RET                            ;Return to caller
                 ;
                 GIS$True$Status:                ;
                                                 ;True status, based on any characters
                                                 ;ready in input buffer
085B 2A8D0F              LHLD    CB$Forced$Input        ;Check if any forced input waiting
085E 7E                  MOV     A,M                    ;Get next character of forced input
085F B7                  ORA     A                      ;Check if nonzero
0860 C25808              JNZ     GIS$Data$Ready         ;Yes, indicate data waiting

0863 211900              LXI     H,DT$Character$Count   ;Check if any characters
0866 19                  DAD     D                      ;  in buffer
0867 7E                  MOV     A,M                    ;Get character count
0868 B7                  ORA     A
0869 C8                  RZ                             ;Empty buffer, A = 0, Z-set
086A C35808              JMP     GIS$Data$Ready
                 ;
                 ;
                 ;#
                 ;
                 ;       Real time clock processing
                 ;
                 ;       Control is transferred to the RTC$Interrupt routine each time
                 ;       the real time clock ticks. The tick count is downdated to see
                 ;       if a complete second has elapsed. If so, the ASCII time in
                 ;       the configuration block is updated.
                 ;
                 ;       With each tick, the watchdog count is downdated to see if control
                 ;       must be "forced" to a previously specified address on return
                 ;       from the RTC interrupt. The watchdog timer can be used to pull
                 ;       control out of what would otherwise be an infinite loop, such
                 ;       as waiting for the printer to come ready.
                 ;
                 ;
                 ;       Set watchdog
                 ;
                 ;       This is a noninterrupt level subroutine that simply sets the
                 ;       watchdog count and address
                 ;
                 ;       Entry parameters
                 ;
                 ;               BC = number of clock ticks before watchdog should
                 ;                       "time out"
                 ;               HL = address to which control will be transferred when
                 ;                       watchdog times out
                 ;
                 Set$Watchdog:
086D F3                  DI                             ;Avoid interference from interrupts
086E 22C100              SHLD    RTC$Watchdog$Address   ;Set address
0871 60                  MOV     H,B
0872 69                  MOV     L,C
0873 22BF00              SHLD    RTC$Watchdog$Count     ;Set count
0876 FB                  EI
0877 C9                  RET
                 ;
                 ;
                 ;#      /
                 ;
                                         ;Control is received here each time the
                                         ;  real time clock ticks
                 RTC$Interrupt:
0878 F5                  PUSH    PSW                    ;Save other registers
0879 228622              SHLD    PI$User$HL             ;Switch to local stack
087C 210000              LXI     H,0
087F 39                  DAD     SP                     ;Get user's stack
0880 228422              SHLD    PI$User$Stack          ;Save it
0883 31B022              LXI     SP,PI$Stack            ;Switch to local stack
0886 C5                  PUSH    B
0887 D5                  PUSH    D

0888 21BE00              LXI     H,RTC$Tick$Count       ;Downdate tick count
```

Figure 8-10. (Continued)

```
088B 35                 DCR     M
088C C2B008             JNZ     RTC$Check$Watchdog      ;Is not at 0 yet
                                                        ;One second has elapsed so
088F 3ABD00             LDA     RTC$Ticks$per$Second    ;  reset to original value
0892 77                 MOV     M,A
                                                        ;Update ASCII real time clock
0893 11A10F             LXI     D,Time$in$ASCII$End     ;DE -> 1 character after ASCII time
0896 21BD00             LXI     H,Update$Time$End       ;HL -> 1 character after control table
                RTC$Update$Digit:
0899 1B                 DCX     D                       ;Downdate pointer to time in ASCII
089A 2B                 DCX     H                       ;Downdate pointer to control table
089B 7E                 MOV     A,M                     ;Get next control character
089C B7                 ORA     A                       ;Check if end of table and therefore
089D CAB008             JZ      RTC$Clock$Updated       ;  all digits of clock updated
08A0 FA9908             JM      RTC$Update$Digit        ;Skip over ":" in ASCII time
08A3 1A                 LDAX    D                       ;Get next ASCII time digit
08A4 3C                 INR     A                       ;Update it
08A5 12                 STAX    D                       ;  and store it back
08A6 BE                 CMP     M                       ;Compare to maximum value
08A7 C2B008             JNZ     RTC$Clock$Updated       ;No carry needed so update complete
08AA 3E30               MVI     A,'0'                   ;Reset digit to ASCII 0
08AC 12                 STAX    D                       ;  and store back in ASCII time
08AD C39908             JMP     RTC$Update$Digit        ;Go back for next digit
                ;
                RTC$Clock$Updated:
                RTC$Check$Watchdog:
08B0 2ABF00             LHLD    RTC$Watchdog$Count      ;Get current watchdog count
08B3 2B                 DCX     H                       ;Downdate it
08B4 7C                 MOV     A,H                     ;Check if it is now 0FFFFH
08B5 B7                 ORA     A
08B6 FACB08             JM      RTC$Dog$Not$Set         ;It must have been 0 beforehand
08B9 B5                 ORA     L                       ;Check if it is now 0
08BA C2C808             JNZ     RTC$Dog$NZ              ;No, it is not out of time

                                                        ;Watchdog time elapsed, so "call"
                                                        ;  appropriate routine
08BD 21C508             LXI     H,RTC$Watchdog$Return   ;Set up return address
08C0 E5                 PUSH    H                       ;  ready for return
08C1 2AC100             LHLD    RTC$Watchdog$Address    ;Transfer control as though by CALL
08C4 E9                 PCHL
                RTC$Watchdog$Return:                    ;Control will come back here from
                                                        ;  the user's watchdog routine
08C5 C3CB08             JMP     RTC$Dog$Not$Set         ;Behave as though watchdog not active

                RTC$Dog$NZ:
08C8 22BF00             SHLD    RTC$Watchdog$Count      ;Save downdated count
                RTC$Dog$Not$Set:                        ;  (Leaves count unchanged)
08CB 3E20               MVI     A,IC$EOI                ;Reset the interrupt controller chip
08CD D3D8               OUT     IC$OCW2$Port

08CF D1                 POP     D                       ;Restore registers from local stack
08D0 C1                 POP     B
08D1 2A8422             LHLD    PI$User$Stack           ;Switch back to user's stack
08D4 F9                 SPHL
08D5 2A8622             LHLD    PI$User$HL              ;Recover user's registers
08D8 F1                 POP     PSW
08D9 FB                 EI                              ;Re-enable interrupts
08DA C9                 RET
                ;
                ;#
                ;
                ;       Shift HL Right one bit
                ;
                SHLR:
08DB B7                 ORA     A               ;Clear carry
08DC 7C                 MOV     A,H             ;Get MS byte
08DD 1F                 RAR                     ;Bit 7 set from previous carry
                                                ;Bit 0 goes into carry
08DE 67                 MOV     H,A             ;Put shifted MS byte back
08DF 7D                 MOV     A,L             ;Get LS byte
08E0 1F                 RAR                     ;Bit 7 = bit 0 of MS byte
08E1 6F                 MOV     L,A             ;Put back into result
08E2 C9                 RET

                ;
                ;#
```

Figure 8-10. (Continued)

```
          ;       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 blocking/deblocking code. This blocking/deblocking
          ;       code is 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"
          ;       mini-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 onto the disk parameter block to
          ;       tell the disk drivers what each logical disk's physical
          ;       diskette type is, 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)
08E3 AE09         DW      Floppy$5$Skewtable      ;5 1/4" skew table
08E5 000000000000 DW      0,0,0                   ;Reserved for CP/M
08EB B022         DW      Directory$Buffer
08ED 3409         DW      Floppy$5$Parameter$Block
08EF B023         DW      Disk$A$Workarea
08F1 1024         DW      Disk$A$Allocation$Vector
          ;
                                  ;Logical disk B: (5 1/4" diskette)
08F3 AE09         DW      Floppy$5$Skewtable      ;Shares same skew table as A:
08F5 000000000000 DW      0,0,0                   ;Reserved for CP/M
08FB B022         DW      Directory$Buffer        ;Shares same buffer as A:
08FD 3409         DW      Floppy$5$Parameter$Block        ;Same DPB as A:
08FF D023         DW      Disk$B$Workarea                 ;Private work area
0901 2624         DW      Disk$B$Allocation$Vector        ;Private allocation vector
          ;
                                  ;Logical disk C: (8" floppy)
0903 F609         DW      Floppy$8$Skewtable      ;8" skew table
0905 000000000000 DW      0,0,0                   ;Reserved for CP/M
090B B022         DW      Directory$Buffer        ;Shares same buffer as A:
090D 4409         DW      Floppy$8$Parameter$Block
090F F023         DW      Disk$C$Workarea                 ;Private work area
0911 3C24         DW      Disk$C$Allocation$Vector        ;Private allocation vector
          ;
                                  ;Logical disk D: (8" floppy)
0913 AE09         DW      Floppy$5$Skewtable      ;Shares same skew table as A:
0915 000000000000 DW      0,0,0                   ;Reserved for CP/M
091B B022         DW      Directory$Buffer        ;Shares same buffer as A:
```

Figure 8-10. (Continued)

```
091D 4409               DW      Floppy$8$Parameter$Block        ;Same DPB as C:
091F 0024               DW      Disk$D$Workarea                 ;Private work area
0921 5B24               DW      Disk$D$Allocation$Vector        ;Private allocation vector

                                        ;Logical disk M: (memory disk)
                M$Disk$DPH:
0923 0000               DW      0                          ;No skew required
0925 000000000000       DW      0,0,0                      ;Reserved for CP/M
092B B022               DW      Directory$Buffer
092D 5409               DW      M$Disk$Parameter$Block
092F 0000               DW      0                          ;Disk cannot be changed, therefore
                                                           ;  no work area is required
0931 7A24               DW      M$Disk$Allocation$Vector
                ;
                ;
                ;       Equates for disk parameter block
                ;
                ;       Disk Types
                ;
0001 =          Floppy$5        EQU     1          ;5 1/4" mini floppy
0002 =          Floppy$8        EQU     2          ;8" floppy (SS SD)
0003 =          M$Disk          EQU     3          ;Memory disk
                ;
                ;       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
0933 81                 DB      Floppy$5 + Need$Deblocking
                                                   ;The parameter block has been amended
                                                   ;  to reflect the new layout of one
                                                   ;  track per diskette side, rather
                                                   ;  than viewing one track as both
                                                   ;  sides on a given head position.
                                                   ;It has also been adjusted to reflect
                                                   ;  one "new" track more being used for
                                                   ;  the CP/M image, with the resulting
                                                   ;  change in the number of allocation
                                                   ;  blocks and the number of reserved
                                                   ;  tracks.
                Floppy$5$Parameter$Block:
0934 2400               DW      36                 ;128-byte sectors per track
0936 04                 DB      4                  ;Block shift
0937 0F                 DB      15                 ;Block mask
0938 01                 DB      1                  ;Extent mask
0939 AB00               DW      171                ;Maximum allocation block number
093B 7F00               DW      127                ;Number of directory entries - 1
093D C0                 DB      1100$0000B         ;Bit map for reserving 1 alloc. block
093E 00                 DB      0000$0000B         ;  for file directory
093F 2000               DW      32                 ;Disk-changed work area size
0941 0300               DW      3                  ;Number of tracks before directory
                ;
                ;
                ;       Standard 8" Floppy
                                                   ;Extra byte prefixed to DPB for
                                                   ; this version of the BIOS
0943 02                 DB      Floppy$8           ;Indicates disk type and the fact
                                                   ; that no deblocking is required
                Floppy$8$Parameter$Block:
0944 1A00               DW      26                 ;Sectors per track
0946 03                 DB      3                  ;Block shift
0947 07                 DB      7                  ;Block mask
0948 00                 DB      0                  ;Extent mask
0949 F200               DW      242                ;Maximum allocation block number
094B 3F00               DW      63                 ;Number of directory entries - 1
094D C0                 DB      1100$0000B         ;Bit map for reserving 2 alloc. blocks
094E 00                 DB      0000$0000B         ;  for file directory
094F 1000               DW      16                 ;Disk-changed work area size
0951 0200               DW      2                  ;Number of tracks before directory
```

Figure 8-10. (Continued)

```
                ;
                ;       M$Disk
                ;
                                                ;The M$Disk presumes that 4 x 48K memory
                                                ;  banks are available. The following
                                                ;  table describes the disk as having
                                                ;  8 tracks: two tracks per memory bank
                                                ;  with each track having 192 128-byte
                                                ;  sectors.
                                                ;  The track number divided by 2 will be
                                                ;  used to select the bank
0953 03                 DB      M$Disk          ;Type is M$Disk, no deblocking
                M$Disk$Parameter$Block:
0954 C000               DW      192             ;Sectors per "track". Each track is
                                                ;  24K of memory
0956 03                 DB      3               ;Block shift (1024 byte allocation)
0957 07                 DB      7               ;Block mask
0958 00                 DB      0               ;Extent mask
0959 C000               DW      192             ;Maximum allocation block number
095B 3F00               DW      63              ;Number of directory entries -1
095D C0                 DB      1100$0000B      ;Bit map for reserving 2 allocation blocks
095E 00                 DB      0000$0000B      ;  for file directory
095F 0000               DW      0               ;Disk cannot be changed, therefore no
                                                ;  work area
0961 0000               DW      0               ;No reserved tracks
                ;
0004 =          Number$of$Logical$Disks         EQU     4
                ;
                ;#
                ;
                SELDSK:                 ;Select disk in register 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.
                                        ;
0963 210000             LXI     H,0             ;Assume an error
0966 79                 MOV     A,C             ;Check if requested disk valid

0967 FE0C               CPI     'M' - 'A'       ;Check if memory disk
0969 CA9509             JZ      SELDSK$M$Disk   ;Yes

096C FE04               CPI     Number$of$Logical$Disks
096E D0                 RNC                     ;Return if > maximum number of disks
                ;
096F 322D0A             STA     Selected$Disk   ;Save selected disk number
                                                ;Set up to return DPH address
0972 6F                 MOV     L,A             ;Make disk into word value
0973 2600               MVI     H,0
                                                ;Compute offset down disk parameter
                                                ; header table by multiplying by
                                                ; parameter header length (16 bytes)
0975 29                 DAD     H               ;*2
0976 29                 DAD     H               ;*4
0977 29                 DAD     H               ;*8
0978 29                 DAD     H               ;*16
0979 11E308             LXI     D,Disk$Parameter$Headers        ;Get base address
097C 19                 DAD     D               ;DE -> appropriate DPH
097D E5                 PUSH    H               ;Save DPH address
                ;
                                                ;Access disk parameter block to
                                                ;  extract special prefix byte that
                                                ;  identifies disk type and whether
                                                ;  deblocking is required
                                                ;
097E 110A00             LXI     D,10            ;Get DPB pointer offset in DPH
0981 19                 DAD     D               ;DE -> DPB address in DPH
0982 5E                 MOV     E,M             ;Get DPB address in DE
0983 23                 INX     H
0984 56                 MOV     D,M
0985 EB                 XCHG                    ;DE -> DPB

                SELDSK$Set$Disk$Type:
0986 2B                 DCX     H               ;DE -> prefix byte
0987 7E                 MOV     A,M             ;Get prefix byte
0988 E60F               ANI     0FH             ;Isolate disk type
```

Figure 8-10. (Continued)

```
098A 32360A           STA     Selected$Disk$Type      ;Save for use in low level driver
098D 7E               MOV     A,M             ;Get another copy of prefix byte
098E E680             ANI     Need$Deblocking         ;Isolate deblocking flag
0990 32350A           STA     Selected$Disk$Deblock   ;Save for use in low level driver
0993 E1               POP     H               ;Recover DPH pointer
0994 C9               RET
                ;
                SELDSK$M$Disk:                        ;M$Disk selected
0995 212309           LXI     H,M$Disk$DPH            ;Return correct parameter header
0998 C38609           JMP     SELDSK$Set$Disk$Type    ;Resume normal processing
                ;
                ;#
                ;
                ;       Set logical track for next read or write
                ;
                SETTRK:
099B 60               MOV     H,B             ;Selected track in BC on entry
099C 69               MOV     L,C
099D 222E0A           SHLD    Selected$Track  ;Save for low level driver
09A0 C9               RET
                ;
                ;#
                ;
                ;       Set logical sector for next read or write
                ;
                ;
                SETSEC:                               ;Logical sector in C on entry
09A1 79               MOV     A,C
09A2 32300A           STA     Selected$Sector ;Save for low level driver
09A5 C9               RET
                ;
                ;#
                ;
                ;       Set disk DMA (Input/Output) address for next read or write
                ;
09A6 0000       DMA$Address:  DW      0               ;DMA address
                ;
                SETDMA:                               ;Address in BC on entry
09A8 69               MOV     L,C             ;Move to HL to save
09A9 60               MOV     H,B
09AA 22A609           SHLD    DMA$Address     ;Save for low level driver
09AD 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
09AE 00010203         DB      00,01,02,03     ;00,01,02,03          0  )
09B2 10111213         DB      16,17,18,19     ;04,05,06,07          4  )
09B6 20212223         DB      32,33,34,35     ;08,09,10,11          8  )
09BA 0C0D0E0F         DB      12,13,14,15     ;12,13,14,15          3  ) Head
09BE 1C1D1E1F         DB      28,29,30,31     ;16,17,18,19          7  )  0
09C2 08090A0B         DB      08,09,10,11     ;20,21,22,23          2  )
09C6 18191A1B         DB      24,25,26,27     ;24,25,26,27          6  )
09CA 04050607         DB      04,05,06,07     ;28,29,30,31          1  )
09CE 14151617         DB      20,21,22,23     ;32,33,34,35          5  )
                ;
09D2 24252627         DB      36,37,38,39     ;36,37,38,39          0  ]
09D6 34353637         DB      52,53,54,55     ;40,41,42,43          4  ]
09DA 44454647         DB      68,69,70,71     ;44,45,46,47          8  ]
09DE 30313233         DB      48,49,50,51     ;48,49,50,51          3  ] Head
09E2 40414243         DB      64,65,66,67     ;52,53,54,55          7  ]  1
09E6 2C2D2E2F         DB      44,45,46,47     ;56,57,58,59          2  ]
09EA 3C3D3E3F         DB      60,61,62,63     ;60,61,62,63          6  ]
09EE 28292A2B         DB      40,41,42,43     ;64,65,66,67          1  ]
09F2 38393A3B         DB      56,57,58,59     ;68,69,70,71          5  ]
                ;
                ;
                Floppy$8$Skewtable:     ;Standard 8" Driver
```

Figure 8-10. (Continued)

```
                      ;       01,02,03,04,05,06,07,08,09,10    Logical sectors
09F6 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
0A00 090F150208       DB      09,15,21,02,08,14,20,26,06,12   ;Physical sectors
                      ;
                      ;       21,22,23,24,25,26        Logical sectors
0A0A 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
0A10 EB                XCHG                       ;HL -> skew table base
0A11 09                DAD     B                  ;Add on logical sector number
0A12 6E                MOV     L,M                ;Get physical sector number
0A13 2600              MVI     H,0                ;Make into a 16-bit value
0A15 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 in
                                          ;  it that must be written out. This is indicated by
                                          ;  a flag, Must$Write$Buffer, that is set in the
                                          ;  deblocking code.
                                          ;
0A16 3A2C0A            LDA     Must$Write$Buffer       ;Check if physical buffer must
0A19 B7                ORA     A                       ;  be written to a disk
0A1A C2200A            JNZ     HOME$No$Write
0A1D 322B0A            STA     Data$In$Disk$Buffer     ;No, so indicate that buffer
                                                       ;  is now unoccupied
              HOME$No$Write:
0A20 0E00              MVI     C,0                     ;Set to track 0 (logically,
0A22 CD9B09            CALL    SETTRK                  ;  no actual disk operation occurs)
0A25 C9                RET

              ;
              ;#
              ;       Data written to or read from the mini-floppy drive is transferred
              ;       via a physical buffer that is one complete track in length,
              ;       9 * 512 bytes. It is declared at the end of the BIOS, and has
              ;       some small amount of initialization code "hidden" in it.
              ;
              ;       The blocking/deblocking code attempts to minimize the amount
              ;       of actual disk I/O by storing the disk and track
              ;       currently residing in the physical buffer.
              ;       If a read request occurs of a 128-byte CP/M "sector"
              ;       that already is in the physical buffer, no disk access occurs
              ;       If a write request occurs if and the 128-byte CP/M 'sector'
              ;       is already in the physical buffer, no disk access will occur,
              ;       UNLESS the BDOS indicates that it is writing to the directory.
              ;       Directory writes cause an immediate write to disk of the entire
              ;       track in the physical buffer.
              ;
              ;
0800 =        Allocation$Block$Size    EQU     2048
0009 =        Physical$Sec$Per$Track   EQU     9       ;Adjusted to reflect a "new"
                                                       ;  track is only one side of the
                                                       ;  disk
0200 =        Physical$Sector$Size     EQU     512     ;This is the actual sector size
                                                       ;  for the 5 1/4" mini-floppy diskettes
                                                       ;The 8" diskettes and memory disk
                                                       ;  use 128-byte sectors
                                                       ;Declare the physical disk buffer for the
                                                       ;  5 1/4" diskettes
0004 =        CPM$Sec$Per$Physical     EQU     Physical$Sector$Size/128
0024 =        CPM$Sec$Per$Track        EQU     CPM$Sec$Per$Physical*Physical$Sec$Per$Track
1200 =        Bytes$Per$Track          EQU     Physical$Sec$Per$Track*Physical$Sector$Size
0003 =        Sector$Mask              EQU     CPM$Sec$Per$Physical-1
0002 =        Sector$Bit$Shift         EQU     2               ;LOG2(CPM$Sec$Per$Physical)
```

Figure 8-10. (Continued)

```
                 ;
                                          ;These are the values handed over by the BDOS
                                          ;  when it calls the write operation.
                                          ;The allocated/unallocated indicates whether the
                                          ;  BDOS wishes 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 wishes to write to
                                          ;  the file directory.
                                          ;
0000 =           Write$Allocated          EQU     0
0001 =           Write$Directory          EQU     1
0002 =           Write$Unallocated        EQU     2       ;<== ignored for track buffering
                 ;
0A26 00          Write$Type:              DB      0       ;Contains the type of write
                                                          ;  indicated by the BDOS

                 ;
                 ;
                 In$Buffer$Dk$Trk:                ;Variables for physical sector currently
                                                  ;  in Disk$Buffer in memory
0A27 00          In$Buffer$Disk:          DB      0       ;) These are moved and compared
0A28 0000        In$Buffer$Track:         DW      0       ;) as a group, so do not alter
                                                          ;  these lines
0A2A 00          In$Buffer$Disk$Type:     DB      0       ;Disk type for sector in buffer
                 ;
0A2B 00          Data$In$Disk$Buffer:     DB      0       ;When nonzero, the disk buffer has
                                                          ;  data from the disk in it
0A2C 00          Must$Write$Buffer:       DB      0       ;Nonzero when data has been written
                                                          ;  into Disk$Buffer but not yet
                                                          ;  written out to disk
                 ;
                 Selected$Dk$Trk:                 ;Variables for selected disk, track and sector
                                                  ; (Selected by SELDSK, SETTRK and SETSEC)
0A2D 00          Selected$Disk:           DB      0       ;) These are moved and compared
0A2E 0000        Selected$Track:          DW      0       ;) as a group so do not alter order

0A30 00          Selected$Sector:         DB      0       ;Not part of group but needed here

0A31 00          Selected$Physical$Sector: DB     0       ;Selected physical sector derived
                                                          ;  from selected (CP/M) sector by
                                                          ;  shifting it right the number of
                                                          ;  bits specified by Sector$Bit$Shift

                 ;

0A32 00          Disk$Error$Flag:         DB      0       ;Nonzero to indicate an error
                                                          ;  that could not be recovered
                                                          ;  by the disk drivers. The BDOS
                                                          ;  will output a "Bad Sector" message
0A33 00          Disk$Hung$Flag:          DB      0       ;Nonzero if a watchdog timeout
                                                          ;  occurs
0258 =           Disk$Timer               EQU     600     ;Number of 16.66 ms clock ticks
                                                          ;  for a 10 second timeout
                 ;
                                                  ;Flags used inside the deblocking code

0A34 00          Read$Operation:          DB      0       ;Nonzero when a CP/M 128-byte
                                                          ;  sector is to be read
0A35 00          Selected$Disk$Deblock:   DB      0       ;Nonzero when the selected disk
                                                          ;  needs deblocking (set in SELDSK)
0A36 00          Selected$Disk$Type:      DB      0       ;Indicates 8" or 5 1/4" floppy or
                                                          ;  M$Disk selected. (set in SELDSK)
                 ;
                 ;#
                 ;
                 ;       Read in the 128-byte CP/M sector specified by previous calls
                 ;       to Select Disk, Set Track and Sector. The sector will be read
                 ;       into the address specified in the previous Set DMA Address call.
                 ;
                 ;       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:
0A37 3A350A              LDA     Selected$Disk$Deblock    ;Check if deblocking needed
0A3A B7                  ORA     A                        ;  (flag was set in SELDSK call)
```

Figure 8-10. (Continued)

```
0A3B CA2F0B                 JZ      Read$No$Deblock         ;No, use normal nondeblocked

                                                    ;The deblocking algorithm used is such
                                                    ;  that a read operation can be viewed
                                                    ;  until the actual data transfer as though
                                                    ;  it was the first write to an unallocated
                                                    ;  allocation block
0A3E 3E01                   MVI     A,1                     ;Indicate that a read actually
0A40 32340A                 STA     Read$Operation          ;  is to be performed

0A43 3E00                   MVI     A,Write$Allocated       ;Fake deblocking code into believing
0A45 32260A                 STA     Write$Type              ;  that this is a write to an
                                                            ;  allocated allocation block
0A48 C35C0A                 JMP     Perform$Read$Write      ;Use common code to execute read
                    ;
                    ;#
                    ;       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), or
                    ;       to the directory (in which case the data will be written to the
                    ;       disk immediately).
                    ;
                    ;       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 be written out when circumstances force the
                    ;       transfer. The number of physical disk operations can therefore
                    ;       be reduced considerably.
                    ;
                    WRITE:
0A4B 3A350A                 LDA     Selected$Disk$Deblock   ;Check if deblocking is required
0A4E B7                     ORA     A                       ; (flag set in SELDSK call)
0A4F CA2A0B                 JZ      Write$No$Deblock

0A52 AF                     XRA     A                       ;Indicate that a write operation
0A53 32340A                 STA     Read$Operation          ;  is required (i.e NOT a read)
0A56 79                     MOV     A,C                     ;Save the BDOS write type
0A57 E601                   ANI     1                       ;  but only distinguish between
                                                            ;  write to allocated block or
0A59 32260A                 STA     Write$Type              ;  directory write
                    ;
                    ;
                    ;#
                    ;
                    Perform$Read$Write:     ;Common code to execute both reads and
                                            ;  writes of 128-byte sectors.
0A5C AF                     XRA     A               ;Assume that no disk errors will
0A5D 32320A                 STA     Disk$Error$Flag ;  occur

0A60 3A300A                 LDA     Selected$Sector ;Convert selected 128-byte sector
0A63 1F                     RAR                     ;  into physical sector by dividing by 4
0A64 1F                     RAR
0A65 E63F                   ANI     3FH             ;Remove any unwanted bits
0A67 32310A                 STA     Selected$Physical$Sector
                                                            ;
0A6A 212B0A                 LXI     H,Data$In$Disk$Buffer   ;Check if disk buffer already has
0A6D 7E                     MOV     A,M                     ;  data in it
0A6E 3601                   MVI     M,1                     ;(Unconditionally indicate that
                                                            ;  the buffer now has data in it)
0A70 B7                     ORA     A                       ;Did it indeed have data in it?
0A71 CA870A                 JZ      Read$Track$into$Buffer  ;No, proceed to read a physical
                                                            ;  track into the buffer
                                                    ;
                                                    ;The buffer does have a physical track
                                                    ;  in it. Check if it is the right one
                                                    ;
0A74 11270A                 LXI     D,In$Buffer$Dk$Trk      ;Check if track in buffer is the
0A77 212D0A                 LXI     H,Selected$Dk$Trk       ;  same as that selected earlier
0A7A CDE10A                 CALL    Compare$Dk$Trk          ;Compare ONLY disk and track
0A7D CA910A                 JZ      Track$In$Buffer         ;Yes, it is already in buffer

                                                            ;No, it will have to be read in
                                                            ;  over current contents of buffer
0A80 3A2C0A                 LDA     Must$Write$Buffer       ;Check if buffer has data in that
```

Figure 8-10. (Continued)

```
0A83 B7                  ORA   A                        ;  must be written out first
0A84 C4E50B              CNZ   Write$Physical           ;Yes, write it out
                ;
                Read$Track$into$Buffer:
0A87 CDCE0A              CALL  Set$In$Buffer$Dk$Trk     ;Set in buffer variables from
                                                        ;  selected disk, track
                                                        ;  to reflect which track is in the
                                                        ;  buffer now
0A8A CDEA0B              CALL  Read$Physical            ;Read the track into the buffer
0A8D AF                  XRA   A                        ;Reset the flag to reflect buffer
0A8E 322C0A              STA   Must$Write$Buffer        ;  contents
                ;
                Track$In$Buffer:                 ;Selected track and
                                                 ;  disk is already in the buffer
                                                 ;Convert the selected CP/M (128-byte)
                                                 ;  sector into a relative address down
                                                 ;  the buffer
0A91 3A300A              LDA   Selected$Sector   ;Get selected sector number
0A94 6F                  MOV   L,A               ;Multiply by 128 by shifting 16-bit value
0A95 2600                MVI   H,0               ;left 7 bits
0A97 29                  DAD   H                 ;* 2
0A98 29                  DAD   H                 ;* 4
0A99 29                  DAD   H                 ;* 8
0A9A 29                  DAD   H                 ;* 16
0A9B 29                  DAD   H                 ;* 32
0A9C 29                  DAD   H                 ;* 64
0A9D 29                  DAD   H                 ;* 128
                ;
0A9E 11A40F              LXI   D,Disk$Buffer     ;Get base address of disk buffer
0AA1 19                  DAD   D                 ;Add on sector number * 128
                                                 ;HL -> 128-byte sector number start
                                                 ;  address in disk buffer
0AA2 EB                  XCHG                    ;DE -> sector in disk buffer
0AA3 2AA609              LHLD  DMA$Address       ;Get DMA address set in SETDMA call
0AA6 EB                  XCHG                    ;Assume a read operation, so
                                                 ;  DE -> DMA address
                                                 ;  HL -> sector in disk buffer
0AA7 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 1/8 of normal
                                                 ;At this point,
                                                 ;  C = loop count
                                                 ;  DE -> DMA address
                                                 ;  HL -> sector in disk buffer
0AA9 3A340A              LDA   Read$Operation    ;Determine whether data is to be moved
0AAC B7                  ORA   A                 ;  out of the buffer (read) or into the
0AAD C2B50A              JNZ   Buffer$Move       ;  buffer (write)
                                                 ;Writing into buffer
                                                         ;(A must be 0 get here)
0AB0 3C                  INR   A                         ;Set flag to force a write
0AB1 322C0A              STA   Must$Write$Buffer         ;  of the disk buffer later on.
0AB4 EB                  XCHG                            ;Make DE -> sector in disk buffer
                                                         ;     HL -> DMA address
                ;
                ;
                Buffer$Move:
0AB5 CDF80A              CALL  Move$8            ;Moves 8 bytes * C times from (HL)
                                                 ;  to (DE)

                                                 ;
0AB8 3A260A              LDA   Write$Type        ;If write to directory, write out
0ABB FE01                CPI   Write$Directory   ;  buffer immediately
0ABD 3A320A              LDA   Disk$Error$Flag   ;Get error flag in case delayed write or read
0AC0 C0                  RNZ                     ;Return if delayed write or read
                                                 ;
0AC1 B7                  ORA   A                 ;Check if any disk errors have occured
0AC2 C0                  RNZ                     ;Yes, abandon attempt to write to directory
                                                 ;
0AC3 AF                  XRA   A                 ;Clear flag that indicates buffer must be
0AC4 322C0A              STA   Must$Write$Buffer        ;  written out
0AC7 CDE50B              CALL  Write$Physical    ;Write buffer out to physical track
0ACA 3A320A              LDA   Disk$Error$Flag   ;Return error flag to caller
0ACD C9                  RET
                ;
```

Figure 8-10. (Continued)

```
                ;
                ;
                Set$In$Buffer$Dk$Trk:                     ;Indicate selected disk, track
                                                          ;  now residing in buffer
0ACE 3A2D0A             LDA     Selected$Disk
0AD1 32270A             STA     In$Buffer$Disk

0AD4 2A2E0A             LHLD    Selected$Track
0AD7 22280A             SHLD    In$Buffer$Track

0ADA 3A360A             LDA     Selected$Disk$Type        ;Also reflect disk type
0ADD 322A0A             STA     In$Buffer$Disk$Type

0AE0 C9                 RET
                ;
                ;
                Compare$Dk$Trk:                   ;Compares just the disk and track
                                                  ;  pointed to by DE and HL
0AE1 0E03               MVI     C,3               ;Disk (1), track (2)
                Compare$Dk$Trk$Loop:
0AE3 1A                 LDAX    D                 ;Get comparitor
0AE4 BE                 CMP     M                 ;Compare with comparand
0AE5 C0                 RNZ                       ;Abandon comparison if inequality found
0AE6 13                 INX     D                 ;Update comparitor pointer
0AE7 23                 INX     H                 ;Update comparand pointer
0AE8 0D                 DCR     C                 ;Count down on loop count
0AE9 C8                 RZ                        ;Return (with zero flag set)
0AEA C3E30A             JMP     Compare$Dk$Trk$Loop
                ;
                ;
                Move$Dk$Trk:                      ;Moves the disk, track
                                                  ;  variables pointed at by HL to
                                                  ;  those pointed at by DE
0AED 0E03               MVI     C,3               ;Disk (1), Track (2)
                Move$Dk$Trk$Loop:
0AEF 7E                 MOV     A,M               ;Get source byte
0AF0 12                 STAX    D                 ;Store in destination
0AF1 13                 INX     D                 ;Update pointers
0AF2 23                 INX     H
0AF3 0D                 DCR     C                 ;Count down on byte count
0AF4 C8                 RZ                        ;Return if all bytes moved
0AF5 C3EF0A             JMP     Move$Dk$Trk$Loop
                ;
                ;#
                ;
                ;       Move eight bytes
                ;
                ;       This routine moves eight bytes in a block, C times, from
                ;       (HL) to (DE). It uses "drop through" coding to speed
                ;       up execution.
                ;
                ;       Entry Parameters
                ;
                ;               C = number of 8-byte blocks to move
                ;               DE -> destination address
                ;               HL -> source address
                ;
                Move$8:
0AF8 7E                 MOV     A,M               ;Get byte from source
0AF9 12                 STAX    D                 ;Put into destination
0AFA 13                 INX     D                 ;Update pointers
0AFB 23                 INX     H
0AFC 7E                 MOV     A,M               ;Get byte from source
0AFD 12                 STAX    D                 ;Put into destination
0AFE 13                 INX     D                 ;Update pointers
0AFF 23                 INX     H
0B00 7E                 MOV     A,M               ;Get byte from source
0B01 12                 STAX    D                 ;Put into destination
0B02 13                 INX     D                 ;Update pointers
0B03 23                 INX     H
0B04 7E                 MOV     A,M               ;Get byte from source
0B05 12                 STAX    D                 ;Put into destination
0B06 13                 INX     D                 ;Update pointers
0B07 23                 INX     H
0B08 7E                 MOV     A,M               ;Get byte from source
0B09 12                 STAX    D                 ;Put into destination
```

Figure 8-10. (Continued)

```
0B0A 13                 INX     D               ;Update pointers
0B0B 23                 INX     H
0B0C 7E                 MOV     A,M             ;Get byte from source
0B0D 12                 STAX    D               ;Put into destination
0B0E 13                 INX     D               ;Update pointers
0B0F 23                 INX     H
0B10 7E                 MOV     A,M             ;Get byte from source
0B11 12                 STAX    D               ;Put into destination
0B12 13                 INX     D               ;Update pointers
0B13 23                 INX     H
0B14 7E                 MOV     A,M             ;Get byte from source
0B15 12                 STAX    D               ;Put into destination
0B16 13                 INX     D               ;Update pointers
0B17 23                 INX     H

0B18 0D                 DCR     C               ;Count down on loop counter
0B19 C2F80A             JNZ     Move$8          ;Repeat until done
0B1C C9                 RET

                ;
                ;#
                ;
                ;       Introduction to the disk controllers on this computer system
                ;
                ;       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
                ;       in memory to detect when they are to perform some disk
                ;       operation. The 8" controller looks at location 0040H, and
                ;       the 5 1/4" controller looks at location 0045H. These are
                ;       called their disk control bytes. If the most significant
                ;       bit of a disk control byte is set, the controller will then
                ;       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, and 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 (0043H) for this.
                ;       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$5         EQU     46H     ;Control table pointer
                ;
                ;
                ;       Floppy Disk Control Tables
                ;
0B1D 00         Floppy$Command:                 DB      0       ;Command
0001 =          Floppy$Read$Code                EQU     01H
0002 =          Floppy$Write$Code               EQU     02H
0B1E 00         Floppy$Unit:                    DB      0       ;Unit (drive) number = 0 or 1
0B1F 00         Floppy$Head:                    DB      0       ;Head number = 0 or 1
0B20 00         Floppy$Track:                   DB      0       ;Track number
0B21 00         Floppy$Sector:                  DB      0       ;Sector number
```

Figure 8-10. (Continued)

```
0B22 0000    Floppy$Byte$Count:             DW      0       ;Number of bytes to read/write
0B24 0000    Floppy$DMA$Address:            DW      0       ;Transfer address
0B26 0000    Floppy$Next$Status$Block:      DW      0       ;Pointer to next status block
                                                            ;  if commands are chained.
0B28 0000    Floppy$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
0B2A 3E02            MVI     A,Floppy$Write$Code    ;Get write function code
0B2C C3310B          JMP     Common$No$Deblock      ;Go to common code
             Read$No$Deblock:               ;Read previously selected sector
                                            ;  into disk buffer.
0B2F 3E01            MVI     A,Floppy$Read$Code     ;Get read function code
             Common$No$Deblock:
0B31 321D0B          STA     Floppy$Command ;Set command function code
                                            ;Set up nondeblocked command table

0B34 3A360A          LDA     Selected$Disk$Type     ;Check if memory disk operation
0B37 FE03            CPI     M$Disk
0B39 CA7A0B          JZ      M$Disk$Transfer ;Yes, it is M$Disk

             No$Deblock$Retry:              ;Re-entry point to retry after error
0B3C 218000          LXI     H,128          ;Bytes per sector
0B3F 22220B          SHLD    Floppy$Byte$Count
0B42 AF              XRA     A              ;8" floppy only has head 0
0B43 321F0B          STA     Floppy$Head
                                            ;
0B46 3A2D0A          LDA     Selected$Disk  ;8" floppy controller only knows about
                                            ;  units 0 and 1 so Selected$Disk must
                                            ;  be converted
0B49 E601            ANI     01H            ;Turn into 0 or 1
0B4B 321E0B          STA     Floppy$Unit    ;Set unit number
                                            ;
0B4E 3A2E0A          LDA     Selected$Track
0B51 32200B          STA     Floppy$Track   ;Set track number
                                            ;
0B54 3A300A          LDA     Selected$Sector
0B57 32210B          STA     Floppy$Sector  ;Set sector number
                                            ;
0B5A 2AA609          LHLD    DMA$Address    ;Transfer directly between DMA Address
0B5D 22240B          SHLD    Floppy$DMA$Address     ; and 8" controller.
                                            ;
                                            ;The disk controller can accept chained
                                            ;  disk control tables, but in this case,
                                            ;  they are not used, so the "Next" pointers
                                            ;  must be pointed back at the initial
                                            ;  control bytes in the base page.
0B60 214300          LXI     H,Disk$Status$Block            ;Point next status back at
0B63 22260B          SHLD    Floppy$Next$Status$Block       ;  main status block
                                                            ;
0B66 214000          LXI     H,Disk$Control$8               ;Point next control byte
0B69 22280B          SHLD    Floppy$Next$Control$Location   ;  back at main control byte
                                                    ;
0B6C 211D0B          LXI     H,Floppy$Command       ;Point controller at control table
0B6F 224100          SHLD    Command$Block$8
                                                    ;
0B72 214000          LXI     H,Disk$Control$8       ;Activate controller to perform
0B75 3680            MVI     M,80H                  ;  operation
0B77 C33B0C          JMP     Wait$For$Disk$Complete

             ;
             ;#
             ;       Memory disk driver
             ;
             ;       This routine must use an intermediary buffer, since the
             ;       DMA address in bank ("track") 0 occupies the same
             ;       place in the overall address space as the M$Disk itself.
             ;       The M$Disk$Buffer is above the 48K mark, and therefore
             ;       remains in the address space regardless of which bank/track
             ;       is selected.
             ;
             ;
```

Figure 8-10. (Continued)

```
                 ;       For writing, the 128-byte sector must be processed:
                 ;
                 ;               1. Move sector DMA$Address -> M$Disk$Buffer
                 ;               2. Select correct track (+1 to get bank number)
                 ;               3. Move sector M$Disk$Buffer -> M$Disk image
                 ;               4. Select bank 0
                 ;
                 ;       For reading, the processing is:
                 ;
                 ;               1. Select correct track/bank
                 ;               2. Move sector M$Disk image -> M$Disk$Buffer
                 ;               3. Select Bank 0
                 ;               4. Move sector M$Disk$Buffer -> DMA$Address
                 ;
                 ;       If there is any risk of any interrupt causing control
                 ;       to be transferred to an address below 48K, interrupts must
                 ;       be disabled when any bank other than 0 is selected.
                 ;
                 M$Disk$Transfer:
0B7A 3A300A              LDA     Selected$Sector ;Compute address in memory
0B7D 6F                  MOV     L,A             ; by muliplying sector * 128
0B7E 2600                MVI     H,0
0B80 29                  DAD     H               ;* 2
0B81 29                  DAD     H               ;* 4
0B82 29                  DAD     H               ;* 8
0B83 29                  DAD     H               ;* 16
0B84 29                  DAD     H               ;* 32
0B85 29                  DAD     H               ;* 64
0B86 29                  DAD     H               ;* 128

0B87 3A2E0A              LDA     Selected$Track  ;Compute which half of bank sector
                                                 ; is in by using LS bit of track
0B8A 47                  MOV     B,A             ;Save copy for later
0B8B E601                ANI     1               ;Isolate lower/upper indicator
0B8D CA940B              JZ      M$Disk$Lower$Half

0B90 110060              LXI     D,(48 * 1024) / 2       ;Upper half, so bias address
0B93 19                  DAD     D

                 M$Disk$Lower$Half:                      ;HL -> sector in memory
0B94 78                  MOV     A,B             ;Recover selected track
0B95 1F                  RAR                     ;Divide by 2 to get bank number
0B96 3C                  INR     A               ;Bank 1 is first track
0B97 47                  MOV     B,A             ;Preserve for later use

0B98 3A1D0B              LDA     Floppy$Command  ;Check if reading or writing
0B9B FE02                CPI     Floppy$Write$Code
0B9D CABE0B              JZ      M$Disk$Write    ;Writing
                                                 ;Reading

0BA0 CDDD0B              CALL    Select$Bank     ;Select correct memory bank
0BA3 113023              LXI     D,M$Disk$Buffer ;DE -> M$Disk$Buffer, HL -> M$Disk image
0BA6 0E10                MVI     C,128/8         ;Number of 8-byte blocks to move
0BA8 CDF80A              CALL    Move$8

0BAB 0600                MVI     B,0             ;Revert to normal memory bank
0BAD CDDD0B              CALL    Select$Bank

0BB0 2AA609              LHLD    DMA$Address     ;Get user's DMA address
0BB3 113023              LXI     D,M$Disk$Buffer
0BB6 EB                  XCHG                    ;DE -> User's DMA, HL -> M$Disk buffer
0BB7 0E10                MVI     C,128/8         ;Number of 8-byte blocks to move
0BB9 CDF80A              CALL    Move$8

0BBC AF                  XRA     A               ;Indicate no error
0BBD C9                  RET

                 M$Disk$Write:                           ;Writing
0BBE E5                  PUSH    H               ;Save sector's address in M$Disk image
0BBF 2AA609              LHLD    DMA$Address     ;Move sector into M$Disk$Buffer
0BC2 113023              LXI     D,M$Disk$Buffer
0BC5 0E10                MVI     C,128/8         ;Number of 8-byte blocks to move
0BC7 CDF80A              CALL    Move$8          ;(Does not use B register)
                                                 ;B = memory bank to select
0BCA CDDD0B              CALL    Select$Bank
```

Figure 8-10. (Continued)

```
0BCD D1                     POP     D               ;Recover sector's M$Disk image address
0BCE 213023                 LXI     H,M$Disk$Buffer
0BD1 0E10                   MVI     C,128/8
0BD3 CDF80A                 CALL    Move$8          ;Move into M$Disk image

0BD6 0600                   MVI     B,0             ;Select bank 0
0BD8 CDDD0B                 CALL    Select$Bank

0BDB AF                     XRA     A               ;Indicate no error
0BDC C9                     RET
                    ;
                    ;#
                    ;       Select bank
                    ;
                    ;       This routine switches in the required memory bank.
                    ;       Note that the hardware port that controls bank selection
                    ;       also has other bits in it. These are preserved across
                    ;       bank selections.
                    ;
                    ;       Entry parameter
                    ;
                    ;               B = bank number
                    ;
0040 =              Bank$Control$Port       EQU     40H
00F8 =              Bank$Mask               EQU     1111$1000B      ;To preserve other bits
                    ;
                    Select$Bank:
0BDD DB40                   IN      Bank$Control$Port       ;Get current setting in port
0BDF E6F8                   ANI     Bank$Mask               ;Preserve all other bits
0BE1 B0                     ORA     B                       ;Set bank code
0BE2 D340                   OUT     Bank$Control$Port       ;Select the bank
0BE4 C9                     RET
                    ;
                    ;#
                    ;

                    Write$Physical:                 ;Write contents of disk buffer to
                                                    ;  correct sector
0BE5 3E02                   MVI     A,Floppy$Write$Code     ;Get write function code
0BE7 C3EC0B                 JMP     Common$Physical ;Go to common code
                    Read$Physical:                  ;Read previously selected sector
                                                    ;  into disk buffer
0BEA 3E01                   MVI     A,Floppy$Read$Code      ;Get read function code
                    ;
                    Common$Physical:
0BEC 321D0B                 STA     Floppy$Command  ;Set command table

                    ;
                    Deblock$Retry:                  ;Re-entry point to retry after error
0BEF 3A2A0A                 LDA     In$Buffer$Disk$Type     ;Get disk type currently in buffer
0BF2 FE01                   CPI     Floppy$5                ;Confirm it is a 5 1/4" floppy
0BF4 CAFD0B                 JZ      Correct$Disk$Type       ;Yes
0BF7 3E01                   MVI     A,1                     ;No, indicate disk error
0BF9 32320A                 STA     Disk$Error$Flag
0BFC C9                     RET
                    Correct$Disk$Type:              ;Set up disk control table
                                                    ;
0BFD 3A270A                 LDA     In$Buffer$Disk  ;Convert disk number to 0 or 1
0C00 E601                   ANI     1               ;  for disk controller
0C02 321E0B                 STA     Floppy$Unit

0C05 2A280A                 LHLD    In$Buffer$Track ;Set up head and track number
0C08 7D                     MOV     A,L             ;Even numbered tracks will be on
0C09 E601                   ANI     1               ;  head 0, odd numbered on head 1
0C0B 321F0B                 STA     Floppy$Head     ;Set head number

0C0E 7D                     MOV     A,L             ;Note: this is single byte value
0C0F 1F                     RAR                     ; /2 for track (carry off from ANI above)
0C10 32200B                 STA     Floppy$Track

0C13 3E01                   MVI     A,1                     ;Start with sector 1 as a whole
0C15 32210B                 STA     Floppy$Sector           ;  track will be transferred

0C18 210012                 LXI     H,Bytes$Per$Track       ;Set byte count for complete
0C1B 22220B                 SHLD    Floppy$Byte$Count       ;  track to be transferred
                                                            ;
```

Figure 8-10. (Continued)

```
0C1E 21A40F             LXI     H,Disk$Buffer           ;Set transfer address to be
0C21 22240B             SHLD    Floppy$DMA$Address      ;  disk buffer
                                                        ;
                                                        ;As only one control table is in
                                                        ;  use, close the status and busy
                                                        ;  chain pointers back to the
                                                        ;  main control bytes
0C24 214300             LXI     H,Disk$Status$Block
0C27 22260B             SHLD    Floppy$Next$Status$Block
0C2A 214500             LXI     H,Disk$Control$5
0C2D 22280B             SHLD    Floppy$Next$Control$Location

0C30 211D0B             LXI     H,Floppy$Command        ;Set up command block pointer
0C33 224600             SHLD    Command$Block$5

0C36 214500             LXI     H,Disk$Control$5        ;Activate 5 1/4" disk controller
0C39 3680               MVI     M,80H
                ;
                Wait$For$Disk$Complete:         ;Wait until disk status block indicates
                                                ;  operation has completed, then check
                                                ;  if any errors occurred.
                                                ;On entry HL -> disk control byte
0C3B AF                 XRA     A                       ;Ensure hung flag clear
0C3C 32330A             STA     Disk$Hung$Flag

0C3F 21570C             LXI     H,Disk$Timed$Out        ;Set up watchdog timer
0C42 015802             LXI     B,Disk$Timer            ;Time delay
0C45 CD6D08             CALL    Set$Watchdog
                Disk$Wait$Loop:
0C48 7E                 MOV     A,M                     ;Get control byte
0C49 B7                 ORA     A
0C4A CA5D0C             JZ      Disk$Complete           ;Operation done

0C4D 3A330A             LDA     Disk$Hung$Flag          ;Also check if time expired
0C50 B7                 ORA     A
0C51 C2B40D             JNZ     Disk$Error              ;Will be set to 40H

0C54 C3480C             JMP     Disk$Wait$Loop

                Disk$Timed$Out:                 ;Control arrives here from watchdog
                                                ;  routine itself -- so this is effectively
                                                ;  part of the interrupt service routine.
0C57 3E40               MVI     A,40H                   ;Set disk hung error code
0C59 32330A             STA     Disk$Hung$Flag          ;  into error flag to pull
                                                        ;  control out of loop
0C5C C9                 RET                             ;Return to watchdog routine

                Disk$Complete:
0C5D 010000             LXI     B,0                     ;Reset watchdog timer
                                                        ;HL is irrelevant here
0C60 CD6D08             CALL    Set$Watchdog

0C63 3A4300             LDA     Disk$Status$Block       ;Complete, now check status
0C66 FE80               CPI     80H                     ;Check if any errors occurred
0C68 DAB40D             JC      Disk$Error              ;Yes
                ;
                Disk$Error$Ignore:
0C6B AF                 XRA     A                       ;No
0C6C 32320A             STA     Disk$Error$Flag         ;Clear error flag
0C6F C9                 RET

                ;
                ;#
                ;       Disk error message handling
                ;
                ;
                Disk$Error$Messages:            ;This table is scanned, comparing the
                                                ;  disk error status with those in the
                                                ;  table. Given a match, or even when
                                                ;  then end of the table is reached, the
                                                ;  address following the status value
                                                ;  points to the correct message text.
0C70 40                 DB      40H
0C71 9D0C               DW      Disk$Msg$40
0C73 41                 DB      41H
0C74 A20C               DW      Disk$Msg$41
```

Figure 8-10. (Continued)

```
0C76 42                         DB      42H
0C77 AC0C                       DW      Disk$Msg$42
0C79 21                         DB      21H
0C7A BC0C                       DW      Disk$Msg$21
0C7C 22                         DB      22H
0C7D C10C                       DW      Disk$Msg$22
0C7F 23                         DB      23H
0C80 C80C                       DW      Disk$Msg$23
0C82 24                         DB      24H
0C83 DA0C                       DW      Disk$Msg$24
0C85 25                         DB      25H
0C86 E60C                       DW      Disk$Msg$25
0C88 11                         DB      11H
0C89 F90C                       DW      Disk$Msg$11
0C8B 12                         DB      12H
0C8C 070D                       DW      Disk$Msg$12
0C8E 13                         DB      13H
0C8F 140D                       DW      Disk$Msg$13
0C91 14                         DB      14H
0C92 220D                       DW      Disk$Msg$14
0C94 15                         DB      15H
0C95 310D                       DW      Disk$Msg$15
0C97 16                         DB      16H
0C98 3D0D                       DW      Disk$Msg$16
0C9A 00                         DB      0                       ;<== Terminator
0C9B 4D0D                       DW      Disk$Msg$Unknown        ;Unmatched code
                        ;
0003 =                  DEM$Entry$Size  EQU     3       ;Disk error message table entry size
                        ;
                        ;       Message texts
                        ;
0C9D 48756E6700         Disk$Msg$40:    DB      'Hung',0        ;Timeout message
0CA2 4E6F742052         Disk$Msg$41:    DB      'Not Ready',0
0CAC 5772697465         Disk$Msg$42:    DB      'Write Protected',0
0CBC 4461746100         Disk$Msg$21:    DB      'Data',0
0CC1 466F726D61         Disk$Msg$22:    DB      'Format',0
0CC8 4D69737369         Disk$Msg$23:    DB      'Missing Data Mark',0
0CDA 4275732054         Disk$Msg$24:    DB      'Bus Timeout',0
0CE6 436F6E7472         Disk$Msg$25:    DB      'Controller Timeout',0
0CF9 4472697665         Disk$Msg$11:    DB      'Drive Address',0
0D07 4865616420         Disk$Msg$12:    DB      'Head Address',0
0D14 547261636B         Disk$Msg$13:    DB      'Track Address',0
0D22 536563746F         Disk$Msg$14:    DB      'Sector Address',0
0D31 4275732041         Disk$Msg$15:    DB      'Bus Address',0
0D3D 496C6C6567         Disk$Msg$16:    DB      'Illegal Command',0
0D4D 556E6B6E6F         Disk$Msg$Unknown:       DB      'Unknown',0
                        ;
                        Disk$EM$1:                      ;Main disk error message -- part 1
0D55 070D0A                             DB      BELL,CR,LF
0D58 4469736B20                         DB      'Disk ',0
                        ;
                                                        ;Error text output next
                        ;
                        Disk$EM$2:                      ;Main disk error message -- part 2
0D5E 204572726F                         DB      ' Error ('
0D66 0000               Disk$EM$Status: DB      0,0     ;Status code in Hex.
0D68 290D0A2020                         DB      ')',CR,LF,'     Drive '
0D76 00                 Disk$EM$Drive:  DB      0       ;Disk drive code, A,B...
0D77 2C20486561                         DB      ', Head '
0D7E 00                 Disk$EM$Head:   DB      0       ;Head number
0D7F 2C20547261                         DB      ', Track '
0D87 0000               Disk$EM$Track:  DB      0,0     ;Track number
0D89 2C20536563                         DB      ', Sector '
0D92 0000               Disk$EM$Sector: DB      0,0     ;Sector number
0D94 2C204F7065                         DB      ', Operation - '
0DA2 00                                 DB      0               ;Terminator
                        ;
0DA3 526561642E         Disk$EM$Read:   DB      'Read.',0       ;Operation names
0DA9 5772697465         Disk$EM$Write:  DB      'Write.',0
                        ;
                        ;
                        Disk$Action$Confirm:
0DB0 00                                 DB      0       ;Set to character entered by user
0DB1 0D0A00                             DB      CR,LF,0
                        ;
                        ;       Disk error processor
```

Figure 8-10. (Continued)

```
                ;
                ;       This routine builds and outputs an error message.
                ;       The user is then given the opportunity to:
                ;
                ;               R -- retry the operation that caused the error
                ;               I -- ignore the error and attempt to continue
                ;               A -- abort the program and return to CP/M.
                ;
                Disk$Error:
0DB4 F5                 PUSH    PSW             ;Preserve error code from controller
0DB5 21660D             LXI     H,Disk$EM$Status        ;Convert code for message
0DB8 CD440E             CALL    CAH                     ;Converts A to hex.

0DBB 3A270A             LDA     In$Buffer$Disk          ;Convert disk id. for message
0DBE C641               ADI     'A'                     ;Make into letter
0DC0 32760D             STA     Disk$EM$Drive

0DC3 3A1F0B             LDA     Floppy$Head             ;Convert head number
0DC6 C630               ADI     '0'
0DC8 327E0D             STA     Disk$EM$Head

0DCB 3A200B             LDA     Floppy$Track            ;Convert track number
0DCE 21870D             LXI     H,Disk$EM$Track
0DD1 CD440E             CALL    CAH

0DD4 3A210B             LDA     Floppy$Sector           ;Convert sector number
0DD7 21920D             LXI     H,Disk$EM$Sector
0DDA CD440E             CALL    CAH

0DDD 21550D             LXI     H,Disk$EM$1             ;Output first part of message
0DE0 CD5305             CALL    Output$Error$Message

0DE3 F1                 POP     PSW                     ;Recover error status code
0DE4 47                 MOV     B,A                     ;For comparisons
0DE5 216D0C             LXI     H,Disk$Error$Messages - DEM$Entry$Size
                                                        ;HL -> table -  one entry
0DE8 110300             LXI     D,DEM$Entry$Size        ;Get entry size for loop below
                Disk$Error$Next$Code:
0DEB 19                 DAD     D                       ;Move to next (or first) entry

0DEC 7E                 MOV     A,M                     ;Get code number from table
0DED B7                 ORA     A                       ;Check if end of table
0DEE CAF80D             JZ      Disk$Error$Matched      ;Yes, pretend a match occurred
0DF1 B8                 CMP     B                       ;Compare to actual code
0DF2 CAF80D             JZ      Disk$Error$Matched      ;Yes, exit from loop
0DF5 C3EB0D             JMP     Disk$Error$Next$Code    ;Check next code
                ;
                Disk$Error$Matched:
0DF8 23                 INX     H                       ;HL -> address of text
0DF9 5E                 MOV     E,M                     ;Get address into DE
0DFA 23                 INX     H
0DFB 56                 MOV     D,M
0DFC EB                 XCHG                            ;HL -> text
0DFD CD5305             CALL    Output$Error$Message    ;Display explanatory text

0E00 215E0D             LXI     H,Disk$EM$2             ;Display second part of message
0E03 CD5305             CALL    Output$Error$Message

0E06 21A30D             LXI     H,Disk$EM$Read          ;Choose operation text
                                                        ;  (assume a read)
0E09 3A1D0B             LDA     Floppy$Command          ;Get controller command
0E0C FE01               CPI     Floppy$Read$Code
0E0E CA140E             JZ      Disk$Error$Read         ;Yes
0E11 21A90D             LXI     H,Disk$EM$Write         ;No, change address in HL
                Disk$Error$Read:
0E14 CD5305             CALL    Output$Error$Message    ;Display operation type
                ;
                Disk$Error$Request$Action:              ;Ask the user what to do next
0E17 CD2F05             CALL    Request$User$Choice     ;Display prompt and wait for input
                                                        ;  Returns with A = uppercase char.
0E1A FE52               CPI     'R'                     ;Retry?
0E1C CA2C0E             JZ      Disk$Error$Retry
0E1F FE41               CPI     'A'                     ;Abort
0E21 CA360E             JZ      System$Reset
0E24 FE49               CPI     'I'                     ;Ignore
0E26 CA6B0C             JZ      Disk$Error$Ignore
```

Figure 8-10. (Continued)

```
0E29 C3170E            JMP     Disk$Error$Request$Action
               ;
               Disk$Error$Retry:                        ;The decision on where to return
                                                        ;  depends on whether the operation
                                                        ;  failed on a deblocked or
                                                        ;  nondeblocked drive.
0E2C 3A350A            LDA     Selected$Disk$Deblock
0E2F B7                ORA     A
0E30 C2EF0B            JNZ     Deblock$Retry
0E33 C33C0B            JMP     No$Deblock$Retry
               ;
               System$Reset:                            ;This is a radical approach, but
                                                        ;  it does cause CP/M to restart.
0E36 0E00              MVI     C,0                      ;System reset
0E38 CD0500            CALL    BDOS

               ;
               ;
               ;       A to upper
               ;
               ;       Converts the contents of the A register to an upper-
               ;       case letter if it is currently a lowercase letter.
               ;
               ;       Entry parameters
               ;
               ;               A = character to be converted
               ;
               ;       Exit parameters
               ;
               ;               A = converted character
               ;
               A$To$Upper:
0E3B FE61              CPI     'a'             ;Compare to lower limit
0E3D D8                RC                      ;No need to convert
0E3E FE7B              CPI     'z' + 1         ;Compare to upper limit
0E40 D0                RNC                     ;No need to convert
0E41 E65F              ANI     5FH             ;Convert to uppercase
0E43 C9                RET
               ;
               ;       Convert A register to hexadecimal
               ;
               ;       This subroutine converts the A register to hexadecimal.
               ;
               ;       Entry parameters
               ;
               ;               A = value to be converted and output
               ;               HL -> buffer area to receive two characters of output
               ;
               ;       Exit parameters
               ;
               ;               HL -> byte following last hex byte output
               ;
               CAH:
0E44 F5                PUSH    PSW             ;Take a copy of the value to be converted
0E45 0F                RRC                     ;Shift A right four places
0E46 0F                RRC
0E47 0F                RRC
0E48 0F                RRC
0E49 CD4D0E            CALL    CAH$Convert     ;Convert to ASCII
0E4C F1                POP     PSW             ;Get original value again
                                               ;Drop into subroutine, which converts
                                               ;  and returns to caller
               CAH$Convert:
0E4D E60F              ANI     0000$1111B      ;Isolate LS four bits
0E4F C630              ADI     '0'             ;Convert to ASCII
0E51 FE3A              CPI     '9' + 1         ;Compare to maximum
0E53 DA580E            JC      CAH$Numeric     ;No need to convert to A -> F
0E56 C607              ADI     7               ;Convert to a letter
               CAH$Numeric:
0E58 77                MOV     M,A             ;Save character
0E59 23                INX     H               ;Update character pointer
0E5A C9                RET

               ;
               ;
               ;#
```

Figure 8-10. (Continued)

```
                ;
                ;       Disk control table images for warm boot
                ;
                Boot$Control$Part$1:
0E5B 01                 DB      1                       ;Read function
0E5C 00                 DB      0                       ;Unit (drive) number
0E5D 00                 DB      0                       ;Head number
0E5E 00                 DB      0                       ;Track number
0E5F 02                 DB      2                       ;Starting sector number
0E60 0010               DW      8*512                   ;Number of bytes to read
0E62 00C4               DW      CCP$Entry               ;Read into this address
0E64 4300               DW      Disk$Status$Block       ;Pointer to next status block
0E66 4500               DW      Disk$Control$5          ;Pointer to next control table
                Boot$Control$Part2:
0E68 01                 DB      1                       ;Read function
0E69 00                 DB      0                       ;Unit (drive) number
0E6A 01                 DB      1                       ;Head number
0E6B 00                 DB      0                       ;Track number
0E6C 01                 DB      1                       ;Starting sector number
0E6D 0006               DW      3*512                   ;Number of bytes to read
0E6F 00D4               DW      CCP$Entry + (8*512)     ;Read into this address
0E71 4300               DW      Disk$Status$Block       ;Pointer to next status block
0E73 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
                                        ;  is hardware specific to the controller. Two
                                        ;  prefabricated control tables are used.
0E75 318000             LXI     SP,80H
0E78 115B0E             LXI     D,Boot$Control$Part1    ;Execute first read of warm boot
0E7B CD8A0E             CALL    Warm$Boot$Read          ;Load drive 0, track 0,
                                                        ;  head 0, sectors 2 - 8
0E7E 11680E             LXI     D,Boot$Control$Part2    ;Execute second read
0E81 CD8A0E             CALL    Warm$Boot$Read          ;Load drive 0, track 0,
                                                        ;  head 1, sectors 1 - 3
0E84 CDDF0E             CALL    Patch$CPM               ;Make custom enhancements patches
0E87 C36C02             JMP     Enter$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.
0E8A 211D0B             LXI     H,Floppy$Command        ;HL -> actual control table
0E8D 224600             SHLD    Command$Block$5         ;Tell the controller its address
                                                        ;Move the control table image
                                                        ;  into the control table itself.
0E90 0E0D               MVI     C,13            ;Set byte count
                Warm$Boot$Move:
0E92 1A                 LDAX    D               ;Get image byte
0E93 77                 MOV     M,A             ;Store into actual control table
0E94 23                 INX     H               ;Update pointers
0E95 13                 INX     D
0E96 0D                 DCR     C               ;Count down on byte count
0E97 C2920E             JNZ     Warm$Boot$Move  ;Continue until all bytes moved

0E9A 214500             LXI     H,Disk$Control$5        ;Activate controller
0E9D 3680               MVI     M,80H
                Wait$For$Boot$Complete:
0E9F 7E                 MOV     A,M                     ;Get status byte
0EA0 B7                 ORA     A                       ;Check if complete
0EA1 C29F0E             JNZ     Wait$For$Boot$Complete  ;No
                                                        ;Yes, check for errors
0EA4 3A4300             LDA     Disk$Status$Block
0EA7 FE80               CPI     80H
0EA9 DAAD0E             JC      Warm$Boot$Error         ;Yes, an error occurred
0EAC C9                 RET
                ;
                Warm$Boot$Error:
0EAD 21B60E             LXI     H,Warm$Boot$Error$Message
0EB0 CD5F02             CALL    Display$Message
```

Figure 8-10. (Continued)

```
0EB3 C3750E             JMP     WBOOT                   ;Restart warm boot
                ;
                Warm$Boot$Error$Message:
0EB6 0D0A576172         DB      CR,LF,'Warm Boot Error - retrying...',CR,LF,0
                ;
                ;
                ;#
                ;
                Ghost$Interrupt:        ;Control will only arrive here under the most
                                        ;  unusual circumstances, as the interrupt
                                        ;  controller will have been programmed to
                                        ;  suppress unused interrupts.
                                        ;
0ED8 F5                 PUSH    PSW                     ;Save pre-interrupt registers
0ED9 3E20               MVI     A,IC$EOI                ;Indicate end of interrupt
0EDB D3D8               OUT     IC$OCW2$Port
0EDD F1                 POP     PSW
0EDE C9                 RET
                ;
                ;
                ;#
                ;
                ;       Patch CP/M
                ;
                ;       This routine makes some very special patches to the
                ;       CCP and BDOS in order to make some custom enhancements
                ;
                ;       Public files:
                ;               On large hard disk systems it is extremely useful
                ;               to partition the disk using the user number features.
                ;               However, it becomes wasteful of disk space because
                ;               multiple copies of common programs must be stored in
                ;               each user area. This patch makes User 0 public --
                ;               accessible from any other user area.
                ;               *** WARNING ***
                ;               Files in User 0 MUST be set to system and read/only
                ;               status to avoid their being accidentally damaged.
                ;               Because of the side effects associated with public
                ;               files, the patch can be turned on or off using
                ;               a flag in the long term configuration block.
                ;
                ;       User prompt:
                ;               When using CP/M's USER command and user numbers
                ;               in general, it is all too easy to become confused
                ;               and forget which user number you are "in." This
                ;               patch modifies the CCP to display a prompt which
                ;               shows not only the default disk id., but also the
                ;               current user number, and an indication of whether
                ;               public files are enabled:
                ;
                ;                               P3B>  or  3B>
                ;                               ^
                ;                               When public files are enabled.
                ;
                ;       Equates for public files
                ;
D35E =          PF$BDOS$Exit$Point      EQU     BDOS$Entry + 758H
D37C =          PF$BDOS$Char$Matches    EQU     BDOS$Entry + 776H
D361 =          PF$BDOS$Resume$Point    EQU     BDOS$Entry + 75BH
000D =          PF$BDOS$Unused$Bytes    EQU     13
                ;
                ;
                ;       Equates for user prompt
                ;
C788 =          UP$CCP$Exit$Point       EQU     CCP$Entry + 388H
C78B =          UP$CCP$Resume$Point     EQU     CCP$Entry + 38BH
C513 =          UP$CCP$Get$User         EQU     CCP$Entry + 113H
C5D0 =          UP$CCP$Get$Disk$Id      EQU     CCP$Entry + 1D0H
C48C =          UP$CCP$CONOUT           EQU     CCP$Entry + 8CH
                ;
                ;
                ;       Set up the intervention points
                ;
                Patch$CPM:
0EDF 3EC3               MVI     A,JMP           ;Set up opcode
0EE1 325ED3             STA     PF$BDOS$Exit$Point
```

Figure 8-10. (Continued)

```
0EE4 3288C7                 STA     UP$CCP$Exit$Point
0EE7 21F40E                 LXI     H,Public$Patch
0EEA 225FD3                 SHLD    PF$BDOS$Exit$Point + 1
0EED 21110F                 LXI     H,Prompt$Patch  ;Get address of intervening code
0EF0 2289C7                 SHLD    UP$CCP$Exit$Point + 1
0EF3 C9                     RET                     ;Return to enter CP/M
                    ;
                    ;
                    ;
                    Public$Patch:                   ;Control arrives here from the BDOS
                                                    ;The BDOS is in the process of scanning
                                                    ;  down the target file name in the
                                                    ;  search next function
                                                    ;  HL -> the name of the file searched for
                                                    ;  DE -> directory entry
                                                    ;  B = character count
0EF4 3A4200                 LDA     CB$Public$Files ;Check if public files are to be enabled
0EF7 B7                     ORA     A
0EF8 CA0B0F                 JZ      No$Public$Files ;No
0EFB 78                     MOV     A,B             ;Get character count
0EFC B7                     ORA     A               ;Check if looking at first byte
                                                    ;  (that contains the user number)
0EFD C20B0F                 JNZ     No$Public$Files ;No, ignore this patch
0F00 1A                     LDAX    D               ;Get user number from directory entry
0F01 FEE5                   CPI     0E5H            ;Check if active directory entry
0F03 CA0B0F                 JZ      No$Public$Files ;Yes, ignore this patch
0F06 7E                     MOV     A,M             ;Get user number
0F07 B7                     ORA     A               ;Check if User 0
0F08 CA7CD3                 JZ      PF$BDOS$Char$Matches    ;Force character match
                    No$Public$Files:                ;Replaced patched out code
0F0B 78                     MOV     A,B                     ;Check if count indicates that
0F0C FE0D                   CPI     PF$BDOS$Unused$Bytes    ;  registers are pointing at
                                                            ;  unused bytes field of FCB
0F0E C361D3                 JMP     PF$BDOS$Resume$Point    ;Return to BDOS
                    ;
                    Prompt$Patch:                   ;Control arrives here from the CCP
                                                    ;The CCP is just about to get the
                                                    ;  drive id. when control gets here.
                                                    ;The CCP's version of CONOUT is used
                                                    ;  so that the CCP can keep track of
                                                    ;  the cursor position.
0F11 3A4200                 LDA     CB$Public$Files ;Check if public files are enabled
0F14 B7                     ORA     A
0F15 CA1D0F                 JZ      UP$Private$Files        ;No
0F18 3E50                   MVI     A,'P'
0F1A CD8CC4                 CALL    UP$CCP$CONOUT   ;Use CCP's CONOUT routine
                    UP$Private$Files:
0F1D CD13C5                 CALL    UP$CCP$Get$User ;Get current user number
0F20 FE0A                   CPI     9 + 1           ;Check if one or two digits
0F22 D2300F                 JNC     UP$2$Digits
0F25 C630                   ADI     '0'             ;Convert to ASCII
                    UP$1$Digit:
0F27 CD8CC4                 CALL    UP$CCP$CONOUT   ;Output the character
0F2A CD0DC5                 CALL    UP$CCP$Get$Disk$Id      ;Get disk identifier
0F2D C38BC7                 JMP     UP$CCP$Resume$Point     ;Return to CCP
                    ;
                    UP$2$Digits:
0F30 C626                   ADI     '0' - 10        ;Subtract 10 and convert to ASCII
0F32 F5                     PUSH    PSW             ;Save converted second digit
0F33 3E31                   MVI     A,'1'           ;Output leading '1'
0F35 CD8CC4                 CALL    UP$CCP$CONOUT
0F38 F1                     POP     PSW             ;Recover second digit
0F39 C3270F                 JMP     UP$1$Digit      ;Output remainder of prompt and return to
                                                    ;  the CCP
                    ;
                    ;#
```

Figure 8-10. (Continued)

```
                ;
                ;       Configuration block get address
                ;
                ;       This routine is called by utility programs running in the TPA.
                ;       Given a specific code number, it returns the address of a specific
                ;       object in the configuration block.
                ;
                ;       By using this routine, utility programs need not know the exact
                ;       layout of the configuration block.
                ;
                ;       Entry parameters
                ;
                ;               C = Object identity code (in effect, this is the
                ;                       subscript of the object's address in the
                ;                       table below)
                ;
                ;=========================
                CB$Get$Address:                         ;<=== BIOS entry point (private)
                ;=========================
0F3C F5                 PUSH    PSW                     ;Save user's registers
0F3D C5                 PUSH    B
0F3E D5                 PUSH    D

0F3F 69                 MOV     L,C                     ;Make code into a word
0F40 2600               MVI     H,0
0F42 29                 DAD     H                       ;Convert code into word offset
0F43 114F0F             LXI     D,CB$Object$Table       ;Get base address of table
0F46 19                 DAD     D                       ;HL -> object's address in table
0F47 5E                 MOV     E,M                     ;Get LS byte
0F48 23                 INX     H
0F49 56                 MOV     D,M                     ;Get MS byte
0F4A EB                 XCHG                            ;HL = address of object

0F4B D1                 POP     D                       ;Recover user's registers
0F4C C1                 POP     B
0F4D F1                 POP     PSW

0F4E C9                 RET
                ;
                ;#
                ;
                CB$Object$Table:
                                                ;       Code
                                                ;        vv
0F4F 8F0F               DW      Date                    ;01 date in ASCII
0F51 990F               DW      Time$In$ASCII           ;02 time in ASCII
0F53 A30F               DW      Time$Date$Flags         ;03 flags indicated if time/date set
0F55 8D0F               DW      CB$Forced$Input         ;04 forced input pointer
0F57 4300               DW      CB$Startup              ;05 system startup message
                                                        ;   Redirection words
0F59 5800               DW      CB$Console$Input        ;06
0F5B 5A00               DW      CB$Console$Output       ;07
0F5D 5C00               DW      CB$Auxiliary$Input      ;08
0F5F 5E00               DW      CB$Auxiliary$Output     ;09
0F61 6000               DW      CB$List$Input           ;10
0F63 6200               DW      CB$List$Output          ;11

0F65 6400               DW      CB$Device$Table$Addresses ;12
0F67 B500               DW      CB$12$24$Clock          ;13 Selects 12/24 hr. format clock
0F69 BD00               DW      RTC$Ticks$per$Second    ;14
0F6B BF00               DW      RTC$Watchdog$Count      ;15
0F6D C100               DW      RTC$Watchdog$Address    ;16
0F6F C300               DW      CB$Function$Key$Table   ;17
0F71 1B02               DW      CONOUT$Escape$Table     ;18

0F73 8400               DW      D0$Initialize$Stream    ;19
0F75 9100               DW      D0$Baud$Rate$Constant   ;20
0F77 9400               DW      D1$Initialize$Stream    ;21
0F79 A100               DW      D1$Baud$Rate$Constant   ;22
0F7B A400               DW      D2$Initialize$Stream    ;23
0F7D B100               DW      D2$Baud$Rate$Constant   ;24
0F7F 4002               DW      Interrupt$Vector        ;25
0F81 890F               DW      LTCB$Offset             ;26
0F83 8B0F               DW      LTCB$Length             ;27
0F85 4200               DW      CB$Public$Files         ;30
```

Figure 8-10. (Continued)

```
0F87 A421                  DW      Multi$Command$Buffer    ;31
                   ;
                   ;#
                   ;       The short term configuration block.
                   ;
                   ;       This contains variables that can be set once CP/M
                   ;       has been initiated, but that are never preserved
                   ;       from one loading of CP/M to the next. This part of
                   ;       the configuration block form the last initialized bytes
                   ;       in the BIOS.
                   ;
                   ;       The two values below are used by utility programs that
                   ;       need to read in the long term configuration block from disk.
                   ;       The BIOS starts on a 256-byte page boundary, and therefore
                   ;       will always be on a 128-byte sector boundary in the reserved
                   ;       area on the disk. A utility program can then, using the
                   ;       CB$Get$Address Private BIOS call, determine how many 128-byte
                   ;       sectors need to be read in by the formula:
                   ;
                   ;               (LCTB$Offset + LTCB$Length) / 128
                   ;
                   ;       The LTCB$Offset is the offset from the start of the BIOS to
                   ;       where the first byte of the long term configuration block
                   ;       starts. Using the offset and the length, the utility can
                   ;       copy the RAM version of the LTCB over the disk image
                   ;       that it has read from the disk, and then write the
                   ;       updated LTCB back onto the disk.
                   ;
0F89 BED9          LTCB$Offset:    DW      BIOS$Entry - Long$Term$CB
0F8B E601          LTCB$Length:    DW      Long$Term$CB$End - Long$Term$CB
                   ;
                   ;       Forced input pointer
                   ;
                   ;       If CONIN ever finds that this pointer is pointing to a nonzero
                   ;       byte, then this byte will be injected into the console input
                   ;       stream as though it had been typed on the console. The
                   ;       pointer is then updated to the next byte in memory.

0F8D 4300          CB$Forced$Input:        DW      CB$Startup
                   ;
                   ;
                   Date:                   ;Current system date
0F8F 31302F3137            DB      '10/17/82',LF   ;Unless otherwise set to the contrary
                                           ;  this is the release date of the system
                                           ;Normally, it will be set by the DATE utility
0F98 00                    DB      0               ;00-byte terminator

                   Time$in$ASCII:          ;Current system time
0F99 3030          HH:     DB      '00'            ;Hours
0F9B 3A                    DB      ':'
0F9C 3030          MM:     DB      '00'            ;Minutes
0F9E 3A                    DB      ':'
0F9F 3030          SS:     DB      '00'            ;Seconds
                   Time$in$ASCII$End:              ;Used when updating the time
0FA1 0A                    DB      LF
0FA2 00                    DB      0               ;00-byte terminator
                   ;
                   ;
                   Time$Date$Flags:        ;This byte contains two flags that are  used
                                           ;  to indicate whether the time and/or date
                                           ;  have been set either programmatically or
                                           ;  by using the TIME and DATE utilities. These
                                           ;  flags can be tested by utility programs that
                                           ;  need to have the correct time and date set.
0FA3 00                    DB      0
0001 =             Time$Set        EQU     0000$0001B
0002 =             Date$Set        EQU     0000$0010B

                   ;
                   ;#
                   ;       Uninitialized buffer areas
                   ;
                   ;       With the exception of the main Disk$Buffer, which contains a few
                   ;       bytes of code, all of the other uninitialized variables
                   ;       occur here. This has the effect of reducing the number of
                   ;       bytes that need be stored in the CP/M image on the disk,
```

Figure 8-10. (Continued)

```
                ;       since uninitialized areas do not need to be kept on the disk.
                ;
                ;
                ;#
                ;
                ;       The cold boot initialization code is only needed once.
                ;       It can be overwritten once it has been executed.
                ;       Therefore, it is "hidden" inside the main disk buffer.
                ;
                ;
0FA4            Disk$buffer:    DS      Physical$Sector$Size * Physical$Sec$Per$Track
                ;
                                                        ;Save the location counter
21A4 =          After$Disk$Buffer       EQU     $       ;$ = current value of location counter
                ;
0FA4                                    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 byte to be output
                                        ;       DB      xx,xx,xx,xx data to be output
                                        ;       :
                                        ;       :
                                        ;       DB      Port number of 00H terminates
                                        ;
                                        ;
                ;
                ;       Initialization stream declared here
0FA4 D8                 DB      IC$ICW1$Port    ;Program the 8259 interrupt controller
0FA5 01                 DB      1
0FA6 56                 DB      IC$ICW1

0FA7 D9                 DB      IC$ICW2$Port
0FA8 01                 DB      1
0FA9 02                 DB      IC$ICW2

0FAA D9                 DB      IC$OCW1$Port
0FAB 01                 DB      1
0FAC FC                 DB      IC$OCW1

0FAD 83                 DB      83H                     ;Program the 8253 clock generator
0FAE 01                 DB      1
0FAF 34                 DB      00$11$010$0B            ;Counter 0, periodic interrupt, mode 2

0FB0 80                 DB      80H                     ;RTC uses channel 0
0FB1 02                 DB      2
0FB2 0146               DW      17921                   ;19721 * 930 nanoseconds =
                                                        ;  16.666 milliseconds). 60 ticks/sec.
0FB4 00                 DB      0               ;Port number of 0 terminates
                ;
                ;
                Signon$Message:
0FB5 43502F4D20         DB      'CP/M 2.2.'
0FBE 3030               DW      VERSION         ;Current version number
0FC0 20                 DB      ' '
0FC1 3032               DW      MONTH           ;Current date
0FC3 2F                 DB      '/'
0FC4 3236               DW      DAY
0FC6 2F                 DB      '/'
0FC7 3833               DW      YEAR
0FC9 0D0A0A             DB      CR,LF,LF
0FCC 456E68616E         DB      'Enhanced BIOS',CR,LF,LF
0FDC 4469736B20         DB      'Disk Configuration :',CR,LF,LF
0FF3 2020202020         DB      '     A: 0.35 Mbyte 5" Floppy',CR,LF
1011 2020202020         DB      '     B: 0.35 Mbyte 5" Floppy',CR,LF,LF
1030 2020202020         DB      '     C: 0.24 Mbyte 8" Floppy',CR,LF
104E 2020202020         DB      '     D: 0.24 Mbyte 8" Floppy',CR,LF
106C 2020202020         DB      '     M: 0.19 Mbyte Memory Disk',CR,LF,LF
                ;
108D 00                 DB      0
                ;
                ;       Messages for M$Disk
                ;
```

Figure 8-10. (Continued)

```
                  M$Disk$Setup$Message:
108E 2020202020           DB      '      M$Disk already contains valid information.',CR,LF,0
                  M$Disk$Not$Setup$Message:
10C0 2020202020           DB      '      M$Disk has been initialized to empty state.',CR,LF,0
                  ;
                  M$Disk$Dir$Entry:                       ;Dummy directory entry used to determine
                                                          ;  if the M$Disk contains valid information
10F3 0F                   DB      15                      ;User 15
10F4 4D24446973           DB      'M$Disk   '
10FC A0A020               DB      ' '+80H,' '+80H,' '     ;System and read/only
10FF 00000000             DB      0,0,0,0
1103 0000000000           DB      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                  ;
0004 =            Default$Disk    EQU     0004H   ;Default disk in base page
                  ;
                  BOOT:           ;Entered directly from the BIOS JMP Vector
                                  ;Control will be transferred here by the CP/M
                                  ;  bootstrap loader
                                  ;
                                                          ;Initialize system
                                                          ;This routine uses the Initialize$Stream
                                                          ;  declared above

1113 F3                   DI                              ;Disable interrupts to prevent any
                                                          ; side effects during initialization
1114 21A40F               LXI     H,Initialize$Stream     ;HL -> data stream
1117 CD1903               CALL    Output$Byte$Stream      ;Output it to the specified
                                                          ;  ports

111A CDEE02               CALL    General$CIO$Initialization ;Initialize character devices

111D 21B50F               LXI     H,Signon$Message        ;Display sign-on message on console
1120 CD5F02               CALL    Display$Message
                  ;
1123 CDDF0E               CALL    Patch$CPM       ;Make necessary patches to CCP and BDOS
                                                  ;  for custom enhancements

                                                  ;Initialize M$Disk
                                                  ;If the M$Disk directory has the
                                                  ;  special reserved file name "M$disk"
                                                  ;  (with lowercase letters and marked
                                                  ;  SYS and R/O), then the M$Disk is
                                                  ;  assumed to contain valid data.
                                                  ;If the "M$Disk" file is absent, the
                                                  ;  M$Disk Directory entry is moved into
                                                  ;  the M$Disk image, and the remainder of
                                                  ;  the directory set to 0E5H.
1126 0601                 MVI     B,1             ;Select bank 1
1128 CDDD0B               CALL    Select$Bank     ;  which contains the M$Disk directory

                                                  ;Check if M$Disk directory entry present
112B 210000               LXI     H,0             ;Start address for first directory
112E 11F310               LXI     D,M$Disk$Dir$Entry
1131 0E20                 MVI     C,32            ;Length to compare
                  M$Disk$Test:
1133 1A                   LDAX    D               ;Get byte from initialized variable
1134 BE                   CMP     M               ;Compare with M$Disk image
1135 C24F11               JNZ     M$Disk$Not$Setup        ;Match fails
1138 13                   INX     D
1139 23                   INX     H
113A 0D                   DCR     C
113B CA4111               JZ      M$Disk$Setup    ;All bytes match
113E C33311               JMP     M$Disk$Test
                  ;
                  M$Disk$Setup:
1141 218E10               LXI     H,M$Disk$Setup$Message  ;Inform user
                  ;
                  M$Disk$Setup$Done:
1144 CD5F02               CALL    Display$Message

1147 AF                   XRA     A               ;Set default disk drive to A:
1148 320400               STA     Default$Disk
114B FB                   EI                      ;Interrupts can now be enabled

114C C36C02               JMP     Enter$CPM       ;Go into CP/M
                  ;
```

Figure 8-10. (Continued)

```
                M$Disk$Not$Setup:
114F 110000             LXI     D,0             ;Move M$Disk directory entry into
1152 21F310             LXI     H,M$Disk$Dir$Entry      ;  M$Disk image
1155 0E04               MVI     C,32/8                  ;Number of 8-byte blocks to move
1157 CDF80A             CALL    Move$8
                ;
                                                ;DE -> next byte after M$Disk directory
                                                ;  entry in image
115A 3EE5               MVI     A,0E5H                  ;Set up to do memory fill
115C 12                 STAX    D                       ;Store first byte in "source" area
115D 62                 MOV     H,D                     ;Set HL to DE +1
115E 6B                 MOV     L,E
115F 23                 INX     H
1160 0EFC               MVI     C,((2 * 1024) - 32) / 8 ;Two allocation blocks
                                                        ;  less 32 bytes for M$Disk entry
1162 CDF80A             CALL    Move$8                  ;Use Move$8 to do fill operation

1165 21C010             LXI     H,M$Disk$Not$Setup$Message
1168 C34411             JMP     M$Disk$Setup$Done       ;Output message and enter CP/M
                ;
                ;
116B 00                 DB      0               ;Dummy
                Last$Initialized$Byte:          ;<== address of last initialized byte
                ;
                ;       End of cold boot initialization code
                ;
21A4                    ORG     After$Disk$Buffer       ;Reset location counter
                ;
21A4            Multi$Command$Buffer:   DS      128     ;This can be used to insert long
                                                        ;  command sequences into the
                                                        ;  console input stream by setting
                                                        ;  the forced input pointer here
                ;
0020 =          D0$Buffer$Length        EQU     32      ;Must be binary number
2224            D0$Buffer:      DS      D0$Buffer$Length
                ;
0020 =          D1$Buffer$Length        EQU     32      ;Must be binary number
2244            D1$Buffer:      DS      D1$Buffer$Length
                ;
0020 =          D2$Buffer$Length        EQU     32      ;Must be binary number
2264            D2$Buffer:      DS      D2$Buffer$Length
                ;
                ;       Data areas for the character drivers
                ;
2284            PI$User$Stack:  DS      2       ;Storage area for user's stack pointer
                                                ;  when an interrupt occurs
2286            PI$User$HL:     DS      2       ;Save area for user's HL
2288                            DS      40      ;Stack area for use by interrupt service
                PI$Stack:                       ;  routines to avoid overflowing the
                                                ;  user's stack area
                ;
22B0            Directory$Buffer:       DS      128     ;Disk directory buffer
                ;
2330            M$Disk$Buffer:          DS      128     ;Intermediary buffer for
                                                        ;  M$Disk
                ;
                ;       Disk work areas
                ;
                ;       These are used by the BDOS to detect any unexpected
                ;       change of diskettes. The BDOS will automatically set
                ;       such a changed diskette to read-only status.
                ;
23B0            Disk$A$Workarea:        DS      32      ; A:
23D0            Disk$B$Workarea:        DS      32      ; B:
23F0            Disk$C$Workarea:        DS      16      ; C:
2400            Disk$D$Workarea:        DS      16      ; D:
                ;
                ;
                ;       Disk allocation vectors
                ;
                ;       These are used by the BDOS to maintain a bit map of
                ;       which allocation blocks are used and which are free.
                ;       One byte is used for eight allocation blocks, hence the
                ;       expression of the form (allocation blocks/8)+1.
                ;
2410            Disk$A$Allocation$Vector        DS      (174/8)+1       ; A:
```

Figure 8-10. (Continued)

```
2426    Disk$B$Allocation$Vector      DS    (174/8)+1     ; B:
        ;
243C    Disk$C$Allocation$Vector      DS    (242/8)+1     ; C:
245B    Disk$D$Allocation$Vector      DS    (242/8)+1     ; D:
        ;
247A    M$Disk$Allocation$Vector      DS    (192/8)+1     ; M$Disk

2493            END     ;of enhanced BIOS listing
```

Figure 8-10. (Continued)