5

The Basic Disk Operating System

The Basic Disk Operating System is the real heart of CP/M. Unlike the Console Command Processor, it must be in memory all the time. It provides all of the input/output services to CP/M programs, including the CCP.

As a general rule, unless you are writing a system-dependent utility program, you should use the BDOS for *all* of your program's input/output. If you circumvent the BDOS you will probably create problems for yourself later.

What the BDOS Does

The BDOS does all of the system input/output for you. These services can be grouped into two types of functions:

Simple Byte-by-Byte I/O
This is sending and receiving data between the computer system and its logical devices—the console, the "reader" and "punch" (or their substitutes), and the printer.

Disk File I/O
This covers such tasks as creating new files, deleting old files, opening existing files, and reading and writing 128-byte long "records" to and from these files.

The remainder of this chapter explains each of the BDOS functions, shows how to make each operating system request, and gives additional information for each function. You should also refer to Digital Research's manual, *CP/M 2 Interface Guide,* for their standard description of these functions.

BDOS Function Calls

The BDOS function calls are described in the order of their function code numbers. Figure 5-1 summarizes these calls.

Naming Conventions

In practice, whenever you write programs that make BDOS calls, you should include a series of equates for the BDOS function code numbers. We shall be making reference to these values in subsequent examples, so they are shown in Figure 5-2 as they will appear in the programs.

The function names used to define the equates in Figure 5-2 are shorter than those in Figure 5-1 to strike a balance between the abbreviated function names used in Digital Research's documentation and the need for clearer function descriptions.

Making a BDOS Function Request

All BDOS functions are requested by issuing a CALL instruction to location 0005H. You can also request a function by transferring control to location 0005H with the return address on the stack.

In order to tell the BDOS what you need it to do, you must arrange for the internal registers of the CPU to contain the required information before the CALL instruction is executed.

Function Code	Description	
	Simple Byte-by-Byte I/O	
0	Overall system and BDOS reset	
1	Read a byte from the console keyboard	
2	Write a byte to the console screen	
3	Read a byte from the logical reader device	
4	Write a byte to the logical punch device	
5	Write a byte to the logical list device	
6	Direct console I/O (no CCP-style editing)	
7*	Read the current setting of the IOBYTE	
8*	Set a new value of the IOBYTE	
9	Send a "$"-terminated string to the console	
10	Read a string from the console into a buffer	
11	Check if a console key is waiting to be read	
12	Return the CP/M version number	
	Disk File I/O	
13	Reset disk system	
14	Select specified logical disk drive	
15	Open specified file for reading/writing	
16	Close specified file after reading/writing	
17	Search file directory for first match with filename	
18	Search file directory for next match with filename	
19	Delete (erase) file	
20	Read the next "record" sequentially	
21	Write the next "record" sequentially	
22	Create a new file with the specified name	
23	Rename a file to a new name	
24	Indicate which logical disks are active	
25	Return the current default disk drive number	
26	Set the DMA address (read/write address)	
27	Return the address of an allocation vector	
28*	Set specified logical disk drive to Read-Only status	
29	Indicate which disks are currently Read-Only status	
30	Set specified file to System or Read-Only status	
31	Return address of disk parameter block (DPB)	
32*	Set/Get the current user number	
33	Read a "record" randomly	
34	Write a "record" randomly	
35	Return logical file size (even for random files)	
36	Set record number for the next random read/write	
37	Reset specified drive	
40	Write a "record" randomly with zero fill	*These do not work under MP/M.

Figure 5-1. BDOS function calls

```
0000 =          B$SYSRESET      EQU     0       ;System Reset
0001 =          B$CONIN         EQU     1       ;Read Console Byte
0002 =          B$CONOUT        EQU     2       ;Write Console Byte
0003 =          B$READIN        EQU     3       ;Read "Reader" Byte
0004 =          B$PUNOUT        EQU     4       ;Write "Punch" Byte
0005 =          B$LISTOUT       EQU     5       ;Write Printer Byte
0006 =          B$DIRCONIO      EQU     6       ;Direct Console I/O
0007 =          B$GETIO         EQU     7       ;Get IOBYTE
0008 =          B$SETIO         EQU     8       ;Set IOBYTE
0009 =          B$PRINTS        EQU     9       ;Print Console String
000A =          B$READCONS      EQU     10      ;Read Console String
000B =          B$CONST         EQU     11      ;Read Console Status
000C =          B$GETVER        EQU     12      ;Get CP/M Version Number
000D =          B$DSKRESET      EQU     13      ;Disk System Reset
000E =          B$SELDSK        EQU     14      ;Select Disk
000F =          B$OPEN          EQU     15      ;Open File
0010 =          B$CLOSE         EQU     16      ;Close File
0011 =          B$SEARCHF       EQU     17      ;Search for First Name Match
0012 =          B$SEARCHN       EQU     18      ;Search for Next Name Match
0013 =          B$ERASE         EQU     19      ;Erase (delete) File
0014 =          B$READSEQ       EQU     20      ;Read Sequential
0015 =          B$WRITESEQ      EQU     21      ;Write Sequential
0016 =          B$CREATE        EQU     22      ;Create File
0017 =          B$RENAME        EQU     23      ;Rename File
0018 =          B$GETACTDSK     EQU     24      ;Get Active (Logged-in) Disks
0019 =          B$GETCURDSK     EQU     25      ;Get Current Default Disk
001A =          B$SETDMA        EQU     26      ;Set DMA (Read/Write) Address
001B =          B$GETALVEC      EQU     27      ;Get Allocation Vector Address
001C =          B$SETDSKRO      EQU     28      ;Set Disk to Read Only
001D =          B$GETRODSKS     EQU     29      ;Get Read Only Disks
001E =          B$SETFAT        EQU     30      ;Set File Attributes
001F =          B$GETDPB        EQU     31      ;Get Disk Parameter Block Address
0020 =          B$SETGETUN      EQU     32      ;Set/Get User Number
0021 =          B$READRAN       EQU     33      ;Read Random
0022 =          B$WRITERAN      EQU     34      ;Write Random
0023 =          B$GETFSIZ       EQU     35      ;Get File Size
0024 =          B$SETRANREC     EQU     36      ;Set Random Record Number
0025 =          B$RESETD        EQU     37      ;Reset Drive
0028 =          B$WRITERANZ     EQU     40      ;Write Random with Zero-Fill
```

Figure 5-2. Equates for BDOS function code numbers

The function code number of the specific function call you want performed must be in register C.

If you need to hand a single-byte value to the BDOS, such as a character to be sent to the console, then you must arrange for this value to be in register E. If the value you wish to pass to the BDOS is a 16-bit value, such as the address of a buffer or a file control block (FCB), this value must be in register pair DE.

When the BDOS hands back a single-byte value, such as a keyboard character or a return code indicating the success or failure of the function you requested, it will be returned in register A. When the BDOS returns a 16-bit value, it will be in register pair HL.

On return from the BDOS, registers A and L will contain the same value, as will registers B and H. This odd convention stems from CP/M's origins in PL/M (Programming Language/Microprocessor), a language used by Intel on their MDS system. Thus, PL/M laid the foundations for what are known as "register calling conventions."

The BDOS makes no guarantee about the contents of the other registers. If you need to preserve a value that is in a register, either store the value in memory or push it onto the stack. The BDOS uses its own stack space, so there is no need to worry about it consuming your stack.

To sum up, when you make a function request to the BDOS that requires a byte value, the code and the required entry and exit parameters will be as follows:

```
MVI     C,FUNCTION$CODE         ;C = function code
MVI     E,SINGLE$BYTE           ;E = single byte value
CALL    BDOS                    ;Location 5
                                ;A = return code or value
                                ;or HL = return value
```

For those function requests that need to have an address passed to the BDOS, the calling sequence is

```
MVI     C,FUNCTION$CODE         ;C = function code
LXI     D,ADDRESS               ;DE = address
CALL    BDOS                    ;Location 5
                                ;A = return code or value
                                ;or HL = return value
```

If a function request involves disk files, you will have to tell the BDOS the address of the FCB that you have created for the file. (Refer back to Chapter 3 for descriptions of the FCB.)

Many file processing functions return a value in register A that is either 0FFH, indicating that the file named in the FCB could not be found, or equal to a value of 0, 1, 2, or 3. In the latter case, the BDOS is returning what is called a "directory code." The number is the directory entry number that the BDOS matched to the file name in your FCB. At any given moment, the BDOS has a 128-byte sector from the directory in memory. Each file directory entry is 32 bytes, so four of them (numbered 0, 1, 2, and 3) can be processed at a time. The directory code indicates which one has been matched to your FCB.

References to CP/M "records" in the following descriptions mean 128-byte sectors. Do not confuse them with the logical records used by applications programs. Think of CP/M records as 128-byte sectors throughout.

Function 0: System Reset

Function Code: C = 00H
Entry Parameters: None
Exit Parameters: Does not return

Example

```
0000 =          B$SYSRESET      EQU     0       ;System Reset
0005 =          BDOS            EQU     5       ;BDOS entry point

0000 0E00               MVI     C,B$SYSRESET    ;Set function code
0002 C30500             JMP     BDOS            ;Note: you can use a JMP since
                                                ; you don't get control back
```

Purpose The system reset function makes CP/M do a complete reset, exactly the same as the warm boot function invoked when you transfer control to the WARMBOOT point (refer to Figure 4-1).

In addition to resetting the BDOS, this function reloads the CCP, rebuilds the allocation vectors for the currently logged disks, sets the DMA address (used by CP/M to address the disk read/write buffer) to 80H, marks all disks as being Read/Write status, and transfers control to the CCP. The CCP then outputs its prompt to the console.

Notes This function is most useful when you are working in a high-level language that does not permit a jump instruction to an absolute address in memory. Use it when your program has finished and you need to return control back to CP/M.

Function 1: Read Console Byte

Function Code: C = 01H
Entry Parameters: None
Exit Parameters: A = Data byte from console

Example

```
0001 =          B$CONIN         EQU     1       ;Console input
0005 =          BDOS            EQU     5       ;BDOS entry

0000 0E01               MVI     C,B$CONIN       ;Get function code
0002 CD0500             CALL    BDOS
```

Purpose This function reads the next byte of data from the console keyboard and puts it into register A. If the character input is a graphic character, it will be echoed back to the console. The only control characters that are echoed are CARRIAGE RETURN, LINE FEED, BACKSPACE, and TAB. In the case of a TAB character, the BDOS outputs as many spaces as are required to move the cursor to the next multiple of eight columns. All of the other control characters, including CONTROL-C, are input but are not echoed.

This function also checks for CONTROL-S (XOFF) to see if console output should be suspended, and for CONTROL-P (printer echo toggle) to see if console output should also be sent to the list device. If CONTROL-S is found, further output will be suspended until you type another character. CONTROL-P will enable the echoing of console output the first time it is pressed and disable it the second time.

If there is no incoming data character, this function will wait until there is one.

Notes This function often hinders rather than helps, because it echoes the input. Whenever you need console input at the byte-by-byte level, you will usually want to suppress this echo back to the console. For instance, you may know that the "console" is actually a communications line such as a modem. You may be trying to accept a password that should not be echoed back. Or you may need to read a

cursor control character that would cause an undesirable side effect on the terminal if echoed there.

In addition, if you need more than a single character from the console, your program will be easier to use if the person at the console can take full advantage of the CCP-style line editing. This can best be done by using the Read Console String function (code 10, 0AH).

Read Console String also is more useful for single character input, especially when you are expecting a "Y" or "N" (yes or no) response. If you use the Read Console Byte function, the operator will have only one chance to enter the data. When you use Read Console String, however, users have the chance to type one character, change their minds, backspace, and type another character.

Function 2: Write Console Byte

Function Code: C = 02H
Entry Parameters: E = Data byte to be output
Exit Parameters: None

Example

```
0002 =          B$CONOUT        EQU     2       ;Write Console Byte
0005 =          BDOS            EQU     5       ;BDOS entry

0000 0E02               MVI     C,B$CONOUT      ;Function code
0002 1E2A               MVI     E,'*'           ;E = data byte to be output
0004 CD0500             CALL    BDOS
```

Purpose

This function outputs the data byte in register E to the console. As with function 1, if the data byte is a TAB character, it will be expanded by the BDOS to the next column that is a multiple of eight. The BDOS also checks to see if there is an incoming character, and if there is, checks to see if it is a CONTROL-S (in which case console output is suspended) or CONTROL-P (in which case echoing of console output to the printer is toggled on or off).

Notes

You may have problems using this function to output cursor-addressing control sequences to the console. If you try to output a true binary cursor address to position 9, the BDOS will interpret this as a TAB character (ASCII code 9) and dutifully replace it with zero to eight blanks. If you need to output binary values, you must set the most significant bit of the character (use an ORI 80H, for example) so that it will not be taken as the ASCII TAB.

Here are two general-purpose subroutines that you will need for outputting messages. The first one, shown in Figure 5-3, outputs a null-byte-terminated message from a specified address. The second, in Figure 5-4, does essentially the same thing *except* that the message string follows immediately after the call to the subroutine.

```
                        ;MSGOUT (message out)
                        ;Output null-byte-terminated message.

                        ;Calling sequence
                        ;       MESSAGE:        DB      'Message',0
                        ;               :
                        ;       LXI     H,MESSAGE
                        ;       CALL    MSGOUT

                        ;Exit Parameters
                        ;       HL -> Null byte terminator

0002 =          B$CONOUT        EQU     2       ;Write Console Byte
0005 =          BDOS            EQU     5       ;BDOS entry point

                MSGOUT:
0000 7E                 MOV     A,M             ;Get next byte for output
0001 B7                 ORA     A
0002 C8                 RZ                      ;Return when null-byte
0003 23                 INX     H               ;Update message pointer
0004 E5                 PUSH    H               ;Save updated pointer
0005 5F                 MOV     E,A             ;Ready for BDOS
0006 0E02               MVI     C,B$CONOUT
0008 CD0500             CALL    BDOS
000B E1                 POP     H               ;Recover message pointer
000C C30000             JMP     MSGOUT          ;Go back for next character
```

Figure 5-3. Write console byte example, output null-byte terminated message from specified address

```
                        ;MSGOUTI (message out in-line)
                        ;Output null-byte-terminated message that
                        ;follows the CALL to MSGOUTI.

                        ;Calling sequence
                        ;       CALL    MSGOUTI
                        ;       DB      'Message',0
                        ;       ... next instruction

                        ;Exit Parameters
                        ;       HL -> instruction following message

0002 =          B$CONOUT        EQU     2       ;Write Console Byte
0005 =          BDOS            EQU     5       ;BDOS entry point

                MSGOUTI:
0000 E1                 POP     H               ;HL -> message
0001 7E                 MOV     A,M             ;Get next data byte
0002 23                 INX     H               ;Update message pointer
0003 B7                 ORA     A               ;Check if null byte
0004 C20800             JNZ     MSGOUTIC        ;No, continue
0007 E9                 PCHL                    ;Yes, return to next instruction
                                                ; after in-line message

                MSGOUTIC:
0008 E5                 PUSH    H               ;Save message pointer
0009 5F                 MOV     E,A             ;Ready for BDOS
000A 0E02               MVI     C,B$CONOUT      ;Function code
000C CD0500             CALL    BDOS
000F C30000             JMP     MSGOUTI         ;Go back for next char.
```

Figure 5-4. Write console byte example, output null-byte terminated message following call to subroutine

Function 3: Read "Reader" Byte

Function Code: C = 03H
Entry Parameters: None
Exit Parameters: A = Character input

Example

```
0003 =          B$READIN        EQU     3       ;Read "Reader" Byte
0005 =          BDOS            EQU     5       ;BDOS entry

0000 0E03               MVI     C,B$READIN      ;Function code
0002 CD0500             CALL    BDOS            ;A = reader byte
```

Purpose

This function reads the next character from the logical "reader" device into register A. In practice, the physical device that is accessed depends entirely on how your BIOS is configured. In some systems, there is no reader at all; this function will return some arbitrary value such as 1AH (the ASCII CONTROL-Z character, used by CP/M to denote "End of File").

Control is not returned to the calling program until a character has been read.

Notes

Since the physical device (if any) used when you issue this request depends entirely on your particular BIOS, there can be no default standard for all CP/M implementations. This is one of the weaker parts of the BDOS.

You should "connect" the reader device by means of BIOS software to a serial port that can be used for communication with another system. This is only a partial solution to the problem, however, because this function call does not return control to your program until an incoming character has been received. There is no direct way that you can "poll" the reader device to see if an incoming character has been received. Once you make this function call, you lose control until the next character arrives; there is no function corresponding to the Read Console Status (function code 11, 0BH) that will simply read status and return to your program.

One possible solution is to build a timer into the BIOS reader driver that returns control to your program with a dummy value in A if a specified period of time goes by with no incoming character. But this brings up the problem of what dummy value to use. If you ever intend to send and receive files containing pure binary information, there is no character in ASCII that you might not encounter in a legitimate context. Therefore, any dummy character you might choose could also be true data.

The most cunning solution is to arrange for one setting of the IOBYTE (which controls logical-device-to-physical-device mapping) to connect the console to the serial communication line. This done, you can make use of the Read Console Status function, which will return not the physical console status but the serial line status. Your program can then act appropriately if no characters are received within a specified time. Figure 5-11 shows a subroutine that uses this technique in the Set IOBYTE function (code 8, 08H).

Figure 5-5 shows an example subroutine to read lines of data from the reader device. It reads characters from the reader, stacking them in memory until either a LINE FEED or a specified number of characters has been received. Note that CARRIAGE RETURNs are ignored, and the input line is terminated by a byte of 00H. The convention of 00H-byte terminated strings and no CARRIAGE RETURNs is used because it makes for much easier program logic. It also conforms to the conventions of the C language.

```
                        ;RL$RDR
                        ;Read line from reader device.
                        ;Carriage returns are ignored, and input terminates
                        ;when specified number of characters have been read
                        ;or a line feed is input.

                        ;Note: Potential weakness is that there is no
                        ;timeout in this subroutine. It will wait forever
                        ;if no more characters arrive at the reader device.

                        ;Calling sequence
                        ;       LXI     H,BUFFER
                        ;       LXI     B,MAXCOUNT
                        ;       CALL    RL$RDR

                        ;Exit Parameters
                        ;       HL -> 00H byte terminating string
                        ;       BC = residual count (0 if max. chars.read)
                        ;       E = last character read

0003 =          B$READIN        EQU     3       ;Reader input
0005 =          BDOS            EQU     5       ;BDOS entry point

000D =          CR              EQU     0DH     ;Carriage return
000A =          LF              EQU     0AH     ;Line feed (terminator)

                RL$RDR:
0000 79                 MOV     A,C             ;Check if count 0
0001 B0                 ORA     B               ;If count 0 on entry, fake
0002 5F                 MOV     E,A             ; last char. read (00H)
0003 CA2000             JZ      RL$RDRX         ;Yes, exit
0006 C5                 PUSH    B               ;Save max. chars. count
0007 E5                 PUSH    H               ;Save buffer pointer

                RL$RDRI:                        ;Loop back here to ignore
0008 0E03               MVI     C,B$READIN
000A CD0500             CALL    BDOS            ;A = character input
000D 5F                 MOV     E,A             ;Preserve copy of chars.
000E FE0D               CPI     CR              ;Check if carriage return
0010 CA0800             JZ      RL$RDRI         ;Yes, ignore it
0013 E1                 POP     H               ;Recover buffer pointer
0014 C1                 POP     B               ;Recover max. Count
0015 FE0A               CPI     LF              ;Check if line feed
0017 CA2000             JZ      RL$RDRX         ;Yes, exit
001A 77                 MOV     M,A             ;No, store char. in buffer
001B 23                 INX     H               ;Update buffer pointer
001C 0B                 DCX     B               ;Downdate count
001D C30000             JMP     RL$RDR          ;Loop back for next char.

                RL$RDRX:
0020 3600               MVI     M,0             ;Null-byte-terminate buffer
0022 C9                 RET
```

Figure 5-5. Read line from reader device

Function 4: Write "Punch" Byte

Function Code: C = 04H
Entry Parameters: E = Byte to be output
Exit Parameters: None

Example

```
0004 =          B$PUNOUT        EQU     4       ;Write "Punch" Byte
0005 =          BDOS            EQU     5

0000 0E04               MVI     C,B$PUNOUT      ;Function code
0002 1E2A               MVI     E,'*'           ;Data byte to output
0004 CD0500             CALL    BDOS
```

Purpose

This function is a counterpart to the Read "Reader" Byte described above. It outputs the specified character from register E to the logical punch device. Again, the actual physical device used, if any, is determined by the BIOS. There is no set standard for this device; in some systems the punch device is a "bit bucket," so called because it absorbs all data that you output to it.

Notes

The problems and possible solutions discussed under the Read "Reader" Byte function call also apply here. One difference, of course, is that this function outputs data, so the problem of an indefinite loop waiting for the next character is less likely to occur. However, if your punch device is connected to a communications line, and if the output hardware is not ready, the BIOS line driver will wait forever. Unfortunately, there is no legitimate way to deal with this problem since the BDOS does not have a function call that checks whether a logical device is ready for output.

Figure 5-6 shows a useful subroutine that outputs a 00H-byte terminated string to the punch. Wherever it encounters a LINE FEED, it inserts a CARRIAGE RETURN into the output data.

Function 5: Write List Byte

Function Code: C = 05H
Entry Parameters: E = Byte to be output
Exit Parameters: None

Example

```
0005 =          B$LSTOUT        EQU     5       ;Write List Byte
0005 =          BDOS            EQU     5

0000 0E05               MVI     C,B$LSTOUT      ;Function code
0002 1E2A               MVI     E,'*'           ;Data byte to output
0004 CD0500             CALL    BDOS
```

Purpose

This function outputs the specified byte in register E to the logical list device. As with the reader and the punch, the physical device used depends entirely on the BIOS.

```
                        ;WL$PUN
                        ;Write line to punch device. Output terminates
                        ;when a 00H byte is encountered.
                        ;A carriage return is output when a line feed is
                        ;encountered.

                        ;Calling sequence
                        ;       LXI     H,BUFFER
                        ;       CALL    WL$PUN

                        ;Exit parameters
                        ;       HL -> 00H byte terminator

0004 =          B$PUNOUT        EQU     4
0005 =          BDOS            EQU     5

000D =          CR              EQU     0DH     ;Carriage return
000A =          LF              EQU     0AH     ;Line feed

                WL$PUN:
0000 E5                 PUSH    H               ;Save buffer pointer
0001 7E                 MOV     A,M             ;Get next character
0002 B7                 ORA     A               ;Check if 00H
0003 CA2000             JZ      WL$PUNX         ;Yes, exit
0006 FE0A               CPI     LF              ;Check if line feed
0008 CC1600             CZ      WL$PUNLF        ;Yes, O/P CR
000B 5F                 MOV     E,A             ;Character to be output
000C 0E04               MVI     C,B$PUNOUT      ;Function code
000E CD0500             CALL    BDOS            ;Output character
0011 E1                 POP     H               ;Recover buffer pointer
0012 23                 INX     H               ;Increment to next char.
0013 C30000             JMP     WL$PUN          ;Output next char

                WL$PUNLF:                       ;Line feed encountered
0016 0E04               MVI     C,B$PUNOUT      ;Function code
0018 1E0D               MVI     E,CR            ;Output a CR
001A CD0500             CALL    BDOS
001D 3E0A               MVI     A,LF            ;Recreate line feed
001F C9                 RET                     ;Output LF

                WL$PUNX:                        ;Exit
0020 E1                 POP     H               ;Balance the stack
0021 C9                 RET
```

Figure 5-6. Write line to punch device

Notes

One of the major problems associated with this function is that it does not deal with error conditions very intelligently. You cannot be sure which physical device will be used as the logical list device, and most standard BIOS implementations will cause your program to wait forever if the printer is not ready or has run out of paper. The BDOS has no provision to return any kind of error status to indicate that there is a problem with the list device. Therefore, the BIOS will have to be changed in order to handle this situation.

Figure 5-7 is a subroutine which outputs data to the list device. As you can see, this is essentially a repeat of Figure 5-6, which performs the same function for the logical punch device.

```
                        ;WL$LST
                        ;Write line to list device. Output terminates
                        ;when a 00H byte is encountered.
                        ;A carriage return is output when a line feed is
                        ;encountered.

                        ;Calling sequence
                        ;       LXI     H,BUFFER
                        ;       CALL    WL$LST

                        ;Exit parameters
                        ;       HL -> 00H byte terminator

0005 =          B$LSTOUT        EQU     5
0005 =          BDOS            EQU     5

000D =          CR              EQU     0DH     ;Carriage return
000A =          LF              EQU     0AH     ;Line feed

                WL$LST:
0000 E5                 PUSH    H               ;Save buffer pointer
0001 7E                 MOV     A,M             ;Get next character
0002 B7                 ORA     A               ;Check if 00H
0003 CA2000             JZ      WL$LSTX         ;Yes, exit
0006 FE0A               CPI     LF              ;Check if line feed
0008 CC1600             CZ      WL$LSTLF        ;Yes, O/P CR
000B 5F                 MOV     E,A             ;Character to be output
000C 0E05               MVI     C,B$LSTOUT      ;Function code
000E CD0500             CALL    BDOS            ;Output character
0011 E1                 POP     H               ;Recover buffer pointer
0012 23                 INX     H               ;Update to next char.
0013 C30000             JMP     WL$LST          ;Output next char.

                WL$LSTLF:                       ;Line feed encountered
0016 0E05               MVI     C,B$LSTOUT      ;Function code
0018 1E0D               MVI     E,CR            ;Output a CR
001A CD0500             CALL    BDOS
001D 3E0A               MVI     A,LF            ;Recreate line feed
001F C9                 RET                     ;Output LF

                WL$LSTX:                        ;Exit
0020 E1                 POP     H               ;Balance the stack
0021 C9                 RET
```

Figure 5-7. Write line to list device

Function 6: Direct Console I/O

Function Code: C = 06H
Entry Parameters: E = 0FFH for Input
E = Other than 0FFH for output
Exit Parameters: A = Input byte or status

Example

```
0006 =          B$DIRCONIO      EQU     6       ;Direct (raw) Console I/O
0005 =          BDOS            EQU     5       ;BDOS entry point

                                                ;Example of console input

0000 0E06               MVI     C,B$DIRCONIO    ;Function code
0002 1EFF               MVI     E,0FFH          ;0FFH means input
0004' CD0500            CALL    BDOS            ;A = 00 if no char. waiting
                                                ;A = NZ if character input
```

```
                                        ;Example of console output

0007 0E06               MVI     C,B$DIRCONIO    ;Function code
0009 1E2A               MVI     E,'*'           ;Not OFFH means output char.
000B CD0500             CALL    BDOS
```

Purpose

This function serves double duty: it both inputs and outputs characters from the console. However, it bypasses the normal control characters and line editing features (such as CONTROL-P and CONTROL-S) normally associated with console I/O. Hence the name "direct" (or "unadorned" as Digital Research describes it). If the value in register E is *not* 0FFH, then E contains a valid ASCII character that is output to the console. The logic used is most easily understood when written in pseudo-code:

```
if this is an input request (E = OFFH)
     {
     if console status indicates a character is waiting
          {
          read the char from the console and
          return to caller with char in A
          }
     else (no input character waiting) and
          return to caller with A = 00
     }
else (output request)
     {
     output the char in E to the console and
     return to caller
     }
```

Notes

This function works well provided you never have to send a value of 0FFH or expect to receive a value of 00H. If you do need to send or receive pure binary data, you cannot use this function, since these values are likely to be part of the data stream.

To understand why you might want to send and receive binary data, remember that the logical "reader" does not have any method for you to check its status to see if an incoming character has arrived. All you can do is attempt to read a character (Read Reader Byte, function code 3). However, the BDOS will not give control back to you until a character arrives (which could be a very long time). One possibility is to logically assign the console to a communications line by the use of the IOBYTE (or some similar means) and then use this Direct I/O call to send and receive data to and from the line. Then you could indeed "poll" the communications line and avoid having your program go into an indefinite wait for an incoming character. An example subroutine using this technique is shown in Figure 5-11 under Set IOBYTE (function code 8).

Figure 5-8 shows a subroutine that uses the Direct Console Input and Output. Because this example is more complex than any shown so far, the code used to check the subroutine has also been included.

Function 7: Get IOBYTE Setting

Function Code: C = 07H
Entry Parameters: None
Exit Parameters: A = IOBYTE current value

```
                ;-------------------------------------------------------
                ;TESTBED CODE
                ;Because of the complexity of this subroutine, the
                ; actual testbed code has been left in this example.
                ; It assumes that DDT or ZSID
                ; will be used for checkout.
                ;-------------------------------------------------------

                        IF      1                       ;Change to IF 0 to disable testbed
0100                    ORG     100H
0100 C31101             JMP     START                   ;Bypass "variables" setup by DDT

0103 00         OPTIONS:        DB      0               ;Option flags
0104 41454900   TERMS:          DB      'A','E','I',0   ;Terminators
0108 05         BUFFER          DB      5               ;Max. characters in buffer
0109 00                         DB      0               ;Actual count
010A 6363636363                 DB      99,99,99,99,99  ;Data bytes
010F 6363                       DB      99,99

                START:
0111 210801             LXI     H,BUFFER        ;Get address of buffer
0114 110401             LXI     D,TERMS         ;Address of terminator table
0117 3A0301             LDA     OPTIONS         ;Get options set by DDT
011A 47                 MOV     B,A             ;Put in correct register
011B CD2B01             CALL    RCS             ;Enter subroutine
011E CD3800             CALL    38H             ;Force DDT breakpoint
0121 C31101             JMP     START           ;Test again
                        ENDIF                   ;End of testbed

                ;RCS: Read console string (using raw input)
                ;Reads a string of characters into a memory
                ; buffer using raw input.

                ;Supports options:
                ;       o to echo characters or not (when echoing,
                ;         a carriage return will be echoed followed
                ;         by line feed)
                ;       o warm boot on input of control-C or not
                ;       o terminating input either on:
                ;               o max. no of chars input
                ;               o matching terminator character

                ; Calling Sequence
                ;       LXI     H,BUFFER
                ;               Buffer has structure:
                ;                       BUFFER: DB      10      Max. size
                ;                               DB      0       Actual Read
                ;                               DS      10+1    Buffer area
                ;       MVI     B,OPTIONS       Options required
                ;                               (see equates)
                ;       LXI     D,TERMS         Pointer to 00H-byte
                ;                               terminated Chars,
                ;                               any one of which is a
                ;                               terminator.
                ;       CALL    RCS

                ; Exit Parameters
                ;       BUFFER: Updated with data bytes and actual
                ;               character count input.
                ;               (Does not include the terminator).
                ;       A = Terminating Code
                ;               0 =     Maximum number of characters input.
                ;               NZ =    Terminator character found.

0001 =          RCS$ECHO        EQU     0000$0001B      ;Input characters to be echoed
0002 =          RCS$ABORT       EQU     0000$0010B      ;Abort on Control-C
0004 =          RCS$FOLD        EQU     0000$0100B      ;Fold lowercase to uppercase
0008 =          RCS$TERM        EQU     0000$1000B      ;DE -> term. char. set

0006 =          B$DIRCONIO      EQU     6       ;Direct console I/O
0005 =          BDOS            EQU     5       ;BDOS entry point

0003 =          CTL$C           EQU     03H     ;Control-C
000D =          CR              EQU     0DH     ;Carriage return
```

Figure 5-8. Read/write string from/to console using raw I/O

```
000A =          LF              EQU     0AH     ;Line feed
0008 =          BS              EQU     08H     ;Backspace

                RCS$ST:                         ;Internal standard terminator table
0124 0D                 DB      0DH             ;Carriage return
0125 0A                 DB      0AH             ;Line feed
0126 00                 DB      0               ;End of table

                RCS$BSS:                        ;Destructive backspace sequence
0127 08200800           DB      BS,' ',BS,0

                RCS:                            ;<<<<< Main entry
012B 23                 INX     H               ;HL -> actual count
012C 3600               MVI     M,0             ;Reset to initial state
012E 2B                 DCX     H               ;HL -> max. count

                RCS$L:
012F E5                 PUSH    H               ;Save buffer pointer
0130 CD9201             CALL    RCS$GC          ;Get character and execute:
                                                ; ECHO, ABORT, and FOLD options
                                                ;C = character input
0133 E1                 POP     H               ;Recover buffer pointer
0134 3E08               MVI     A,RCS$TERM      ;Check if user-specified terminator
0136 A0                 ANA     B               ;B = options
0137 C23D01             JNZ     RCS$UST         ;User specified terminators
013A 112401             LXI     D,RCS$ST        ;Standard terminators

                RCS$UST:
013D CDD401             CALL    RCS$CT          ;Check for terminator
0140 CA4C01             JZ      RCS$NOTT        ;Not terminator
0143 47                 MOV     B,A             ;Preserve terminating char.

                RCS$MCI:                        ;(Max. char. input shares this code)
0144 0E00               MVI     C,0             ;Terminate buffer
0146 CD7F01             CALL    RCS$SC          ;Save character
0149 78                 MOV     A,B             ;Recover terminating char.
014A B7                 ORA     A               ;Set flags
014B C9                 RET

                RCS$NOTT:                       ;Not a terminator
014C 3E08               MVI     A,BS            ;Check for backspace
014E B9                 CMP     C
014F CA6001             JZ      RCS$BS          ;Backspace entered
0152 CD7F01             CALL    RCS$SC          ;Save character in buffer
0155 CD8B01             CALL    RCS$UC          ;Update count
0158 C22F01             JNZ     RCS$L           ;Not max. so get another char.
015B 0600               MVI     B,0             ;Fake terminating char.
015D C34401             JMP     RCS$MCI         ;A = 0 for max. chars. input

                RCS$BS:                         ;Backspace entered
0160 E5                 PUSH    H               ;Save buffer pointer
0161 23                 INX     H               ;HL -> actual count
0162 35                 DCR     M               ;Back up one
0163 FA7A01             JM      RCS$NBS         ;Check if count negative
0166 212701             LXI     H,RCS$BSS       ;HL -> backspacing sequence
0169 3E01               MVI     A,RCS$ECHO      ;No, check if echoing
016B A0                 ANA     B               ;BS will have been echoed if so
016C CA7001             JZ      RCS$BSNE        ;No, input BS not echoed
016F 23                 INX     H               ;Bypass initial backspace

                RCS$BSNE:
0170 C5                 PUSH    B               ;Save options and character
0171 D5                 PUSH    D               ;Save terminator table pointer
0172 CDF601             CALL    WCS             ;Write console string
0175 D1                 POP     D               ;Recover terminator table pointer
0176 C1                 POP     B               ;Recover options and character
0177 C37B01             JMP     RCS$BSX         ;Exit from backspace logic

                RCS$NBS:
017A 34                 INR     M               ;Reset count to 0

                RCS$BSX:
017B E1                 POP     H               ;Recover buffer pointer
017C C32F01             JMP     RCS$L           ;Get next character
```

Figure 5-8. (Continued)

```
                RCS$SC:                                 ;Save character in C in buffer
                                                        ;HL -> buffer pointer
017F D5                 PUSH    D                       ;Save terminator table pointer
0180 E5                 PUSH    H                       ;Save buffer pointer
0181 23                 INX     H                       ;HL -> actual count in buffer
0182 5E                 MOV     E,M                     ;Get actual count
0183 1C                 INR     E                       ;Count of 0 points to first data byte
0184 1600               MVI     D,0                     ;Make word value of actual count
0186 19                 DAD     D                       ;HL -> next free data byte
0187 71                 MOV     M,C                     ;Save data byte away
0188 E1                 POP     H                       ;Recover buffer pointer
0189 D1                 POP     D                       ;Recover terminator table
                                                        ; pointer
018A C9                 RET

                RCS$UC:                                 ;Update buffer count and check for max.
                                                        ;Return Z set if = to max., NZ
                                                        ; if not HL -> buffer on entry
018B E5                 PUSH    H                       ;Save buffer pointer
018C 7E                 MOV     A,M                     ;Get max. count
018D 23                 INX     H                       ;HL -> actual count
018E 34                 INR     M                       ;Increase actual count
018F BE                 CMP     M                       ;Compare max. to actual
0190 E1                 POP     H                       ;Recover buffer pointer
0191 C9                 RET                             ;Z-flag set

                RCS$GC:                                 ;Get character and execute
                                                        ; ECHO, ABORT and FOLD options
0192 D5                 PUSH    D                       ;Save terminator table pointer
0193 E5                 PUSH    H                       ;Save buffer pointer
0194 C5                 PUSH    B                       ;Save option flags
                RCS$WT:
0195 0E06               MVI     C,B$DIRCONIO            ;Function code
0197 1EFF               MVI     E,0FFH                  ;Specify input
0199 CD0500             CALL    BDOS
019C B7                 ORA     A                       ;Check if data waiting
019D CA9501             JZ      RCS$WT                  ;Go back and wait
01A0 C1                 POP     B                       ;Recover option flags
01A1 4F                 MOV     C,A                     ;Save data byte
01A2 3E02               MVI     A,RCS$ABORT             ;Check if abort option enabled
01A4 A0                 ANA     B
01A5 CAAE01             JZ      RCS$NA                  ;No abort
01A8 3E03               MVI     A,CTL$C                 ;Check for control-C
01AA B9                 CMP     C
01AB CA0000             JZ      0                       ;Warm boot
                RCS$NA:
01AE 3E04               MVI     A,RCS$FOLD              ;Check if folding enabled
01B0 A0                 ANA     B
01B1 C4E501             CNZ     TOUPPER                 ;Convert to uppercase
01B4 3E01               MVI     A,RCS$ECHO              ;Check if echo required
01B6 A0                 ANA     B
01B7 CAD101             JZ      RCS$NE                  ;No echo required
01BA C5                 PUSH    B                       ;Save options and character
01BB 59                 MOV     E,C                     ;Move character for output
01BC 0E06               MVI     C,B$DIRCONIO            ;Function code
01BE CD0500             CALL    BDOS                    ;Echo character
01C1 C1                 POP     B                       ;Recover options and character
01C2 3E0D               MVI     A,CR                    ;Check if carriage return
01C4 B9                 CMP     C
01C5 C2D101             JNZ     RCS$NE                  ;No
01C8 C5                 PUSH    B                       ;Save options and character
01C9 0E06               MVI     C,B$DIRCONIO            ;Function code
01CB 1E0A               MVI     E,LF                    ;Output line feed
01CD CD0500             CALL    BDOS
01D0 C1                 POP     B                       ;Recover options and character
                RCS$NE:
01D1 E1                 POP     H                       ;Recover buffer pointer
01D2 D1                 POP     D                       ;Recover terminator table
01D3 C9                 RET                             ;Character in C
```

Figure 5-8. (Continued)

```
                RCS$CT:                         ;Check for terminator
                                                ;C = character just input
                                                ;DE -> 00-byte character
                                                ; string of term. chars.
                                                ;Returns Z status if no
                                                ; match found, NZ if found
                                                ; (with A = C = terminating
                                                ; character)
01D4 D5                 PUSH    D               ;Save table pointer

                RCS$CTL:
01D5 1A                 LDAX    D               ;Get next terminator character
01D6 B7                 ORA     A               ;Check for end of table
01D7 CAE201             JZ      RCS$CTX         ;No terminator matched
01DA B9                 CMP     C               ;Compare to input character
01DB CAE201             JZ      RCS$CTX         ;Terminator matched
01DE 13                 INX     D               ;Move to next terminator
01DF C3D501             JMP     RCS$CTL         ; loop to try next character in table

                RCS$CTX:                        ;Check terminator exit
01E2 B7                 ORA     A               ;At this point, A will either
                                                ; be 0 if the end of the
                                                ; table has been reached, or
                                                ; NZ if a match has been
                                                ; found. The Z-flag will be
                                                ; set.
01E3 D1                 POP     D               ;Recover table pointer
01E4 C9                 RET

                ;TOUPPER - Fold lowercase letters to upper
                ;       C = Character on entry and exit

                TOUPPER:
01E5 3E60               MVI     A,'a'-1         ;Check if folding needed
01E7 B9                 CMP     C               ;Compare to input char.
01E8 D2F501             JNC     TOUPX           ;No, char. is < or =  "a"-1
01EB 3E7A               MVI     A,'z'           ;Maybe, char. is = or > "a"
01ED B9                 CMP     C
01EE DAF501             JC      TOUPX           ;No, char. is > "z"
01F1 3EDF               MVI     A,0DFH          ;Fold character
01F3 A1                 ANA     C
01F4 4F                 MOV     C,A             ;Return folded character

                TOUPX:
01F5 C9                 RET

                ;WCS - Write console string (using raw I/O)
                ;Output terminates when a 00H byte is encountered.
                ;A carriage return is output when a line feed is
                ;encountered.

                ;Calling sequence
                ;       LXI     H,BUFFER
                ;       CALL    WCS

                ;Exit parameters
                ;       HL -> 00H byte terminator

                WCS:
01F6 E5                 PUSH    H               ;Save buffer pointer
01F7 7E                 MOV     A,M             ;Get next character
01F8 B7                 ORA     A               ;Check if 00H
01F9 CA1602             JZ      WCSX            ;Yes, exit
01FC FE0A               CPI     LF              ;Check if line feed
01FE CC0C02             CZ      WCSLF           ;Yes, output a carriage return
0201 5F                 MOV     E,A             ;Character to be output
0202 0E06               MVI     C,B$DIRCONIO    ;Function code
0204 CD0500             CALL    BDOS            ;Output character
0207 E1                 POP     H               ;Recover buffer pointer
0208 23                 INX     H               ;Update to next char.
0209 C3F601             JMP     WCS             ;Output next char.

                WCSLF:                          ;Line feed encountered
020C 0E06               MVI     C,B$DIRCONIO    ;Function code
```

Figure 5-8. (Continued)

```
020E 1E0D               MVI     E,CR            ;Output a CR
0210 CD0500             CALL    BDOS
0213 3E0A               MVI     A,LF            ;Recreate line feed
0215 C9                 RET                     ;Output LF

                WCSX:                           ;Exit
0216 E1                 POP     H               ;Balance the stack
0217 C9                 RET
```

Figure 5-8. (Continued)

Example

```
0007 =          B$GETIO         EQU     7       ;Get IOBYTE
0005 =          BDOS            EQU     5       ;BDOS entry point

0000 0E07               MVI     C,B$GETIO       ;Function code
0002 CD0500             CALL    BDOS            ;A = IOBYTE
```

Purpose This function places the current value of the IOBYTE in register A.

Notes As we saw in Chapter 4, the IOBYTE is a means of associating CP/M's logical devices (console, reader, punch, and list) with the physical devices supported by a particular BIOS. Use of the IOBYTE is completely optional. CP/M, to quote from the Digital Research *CP/M 2.0 Alteration Guide,* "...tolerate[s] the existence of the IOBYTE at location 0003H."

In practice, the STAT utility provided by Digital Research does have some features that set the IOBYTE to different values from the system console.

Figure 5-9 summarizes the IOBYTE structure. A more detailed description was given in Chapter 4.

Each two-bit field can take on one of four values: 00, 01, 10, and 11. The value can be interpreted by the BIOS to mean a specific physical device, as shown in Table 4-1.

Figure 5-10 has equates that are used to refer to the IOBYTE. You can see that the values shown are declared using the SHL (shift left) operator in the Digital Research Assembler. This is just a reminder that the values are structured this way in the IOBYTE itself.

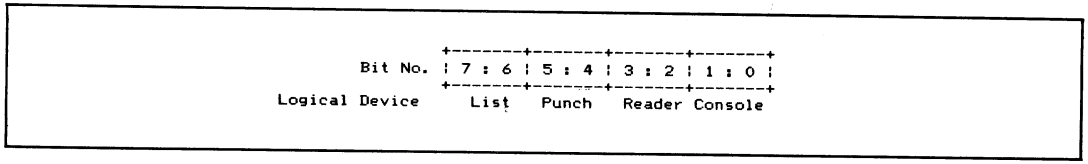

Figure 5-9. The IOBYTE structure

```
                ;IOBYTE equates
                ;These are for accessing the IOBYTE.

                ;Mask values to isolate specific devices.
                ;(These can also be inverted to preserve all BUT the
                ; specific device)

0003 =          IO$CONM EQU     0000$0011B      ;Console mask
000C =          IO$RDRM EQU     0000$1100B      ;Reader mask
0030 =          IO$PUNM EQU     0011$0000B      ;Punch mask
00C0 =          IO$LSTM EQU     1100$0000B      ;List mask

                                                ;Console values
0000 =          IO$CTTY EQU     0               ;Console -> TTY:
0001 =          IO$CCRT EQU     1               ;Console -> CRT:
0002 =          IO$CBAT EQU     2               ;Console input <- RDR:
                                                ;Console output -> LST:
0003 =          IO$CUC1 EQU     3               ;Console -> UC1: (user console 1)

                                                ;Reader values
0000 =          IO$RTTY EQU     0 SHL 2         ;Reader <- TTY:
0004 =          IO$RRDR EQU     1 SHL 2         ;Reader <- RDR:
0008 =          IO$RUR1 EQU     2 SHL 2         ;Reader <- UR1: (user reader 1)
000C =          IO$RUR2 EQU     3 SHL 2         ;Reader <- UR2: (user reader 2)

                                                ;Punch values
0000 =          IO$PTTY EQU     0 SHL 4         ;Punch -> TTY:
0010 =          IO$PPUN EQU     1 SHL 4         ;Punch -> PUN:
0020 =          IO$PUP1 EQU     2 SHL 4         ;Punch -> UP1: (user punch 1)
0030 =          IO$PUP2 EQU     3 SHL 4         ;Punch -> UP2: (user punch 2)

                                                ;List values
0000 =          IO$LTTY EQU     0 SHL 6         ;List -> TTY:
0040 =          IO$LCRT EQU     1 SHL 6         ;List -> CRT:
0080 =          IO$LLPT EQU     2 SHL 6         ;List -> LPT: (physical line printer)
00C0 =          IO$LUL1 EQU     3 SHL 6         ;List -> UL1: (user list 1)
```

Figure 5-10. IOBYTE equates

Function 8: Set IOBYTE

Function Code: C = 08H
Entry Parameters: E = New IOBYTE value
Exit Parameters: None

Example This listing shows you how to assign the logical reader device to the BIOS's console driver. It makes use of some equates from Figure 5-10.

```
0007 =          B$GETIO         EQU     7           ;Get IOBYTE
0008 =          B$SETIO         EQU     8           ;Set IOBYTE
0005 =          BDOS            EQU     5           ;BDOS entry point

000C =          IO$RDRM         EQU     0000$1100B      ;Reader bit mask
0008 =          IO$RUR1         EQU     2 SHL 2         ;User reader select

                ;This example shows how to assign the logical
                ;reader to the user-defined reader #1 (UR1:)

0100                    ORG     100H
0100 0E07               MVI     C,B$GETIO       ;First, get current IOBYTE
```

```
0102 CD0500          CALL    BDOS
0105 E6F3            ANI     (NOT IO$RDRM) AND OFFH ;Preserve all but
                                                    ; reader bits
0107 F608            ORI     IO$RUR1          ;OR in new setting
0109 5F              MOV     E,A              ;Ready for set IOBYTE
010A 0E08            MVI     C,B$SETIO        ;Set new value
010C CD0500          CALL    BDOS
```

Purpose This function sets the IOBYTE to a new value which is given in register E. Because of the individual bit fields in the IOBYTE, you will normally use the Get IOBYTE function, change some bits in the current value, and then call the Set IOBYTE function.

Notes You can use the Set IOBYTE, Get IOBYTE, and Direct Console I/O functions together to create a small program that transforms your computer system into a "smart" terminal. Any data that you type on your keyboard can be sent out of a serial communications line to another computer, and any data received on the line can be sent to the screen.

Figure 5-11 shows this program and illustrates the use of all of these functions.

For this program to function correctly, your BIOS must check the IOBYTE and detect whether the logical console is connected to the physical console (with the IOBYTE set to TTY:) or to the input side of the serial communications line (with the IOBYTE set to RDR:).

Figure 5-11 shows how to use the Get and Set IOBYTE functions to make a simple terminal emulator. For this example to work, the BIOS must detect the Console Value as 3 (IO$CUC1) and connect Console Status, Input, and Output functions to the communications line.

```
0006 =          B$DIRCONIO      EQU     6       ;Direct console input/output
0007 =          B$GETIO         EQU     7       ;Get IOBYTE
0008 =          B$SETIO         EQU     8       ;Set IOBYTE
000B =          B$CONST         EQU     11      ;Get console status (sneak preview)
0005 =          BDOS            EQU     5       ;BDOS entry point

0003 =          IO$CONM EQU     0000$0011B      ;Console mask for IOBYTE
0001 =          IO$CCRT EQU     1               ;Console -> CRT:
0003 =          IO$CUC1 EQU     3               ;Console -> user console #1

                TERM:
0000 CD2A00             CALL    SETCRT          ;Connect console -> CRT:

                TERM$CKS:
0003 CD5200             CALL    CONST           ;Get CRT status
0006 CA2400             JZ      TERM$NOKI       ;No console input
0009 CD4B00             CALL    CONIN           ;Get keyboard character
000C CD3000             CALL    SETCOMM         ;Connect console -> comm. line
000F CD4500             CALL    CONOUT          ;Output to comm. line

                TERM$CCS:                       ;Check comm. status
0012 CD5200             CALL    CONST           ;Get "console" status
0015 CA0000             JZ      TERM            ;No incoming comm. character
0018 CD4B00             CALL    CONIN           ;Get incoming comm. character
```

Figure 5-11. Simple terminal emulator

```
001B CD2A00             CALL    SETCRT          ;Connect console -> CRT:
001E CD4500             CALL    CONOUT          ;Output to CRT
0021 C30300             JMP     TERM$CKS        ;Loop back to check keyboard status

                TERM$NOKI:
0024 CD3000             CALL    SETCOMM         ;Connect console -> comm. line
0027 C31200             JMP     TERM$CCS        ;Loop back to check comm. status

                SETCRT:                         ;Connect console -> CRT:
002A F5                 PUSH    PSW             ;Save possible data character
002B 0601               MVI     B,IO$CCRT       ;Connect console -> CRT:
002D C33300             JMP     SETCON          ;Common code

                SETCOMM:                        ;Connect console -> comm. line
0030 F5                 PUSH    PSW             ;Save possible data character
0031 0603               MVI     B,IO$CUC1       ;Connect console -> comm. line
                                                ;Drop into SETCON

                SETCON:                         ;Set console device
                                                ;New code in B (in bits 1,0)
0033 C5                 PUSH    B               ;Save code
0034 0E07               MVI     C,B$GETIO       ;Get current IOBYTE
0036 CD0500             CALL    BDOS
0039 E6FC               ANI     (NOT IO$CONM) AND OFFH  ;Preserve all but console
003B C1                 POP     B               ;Recover required code
003C B0                 ORA     B               ;OR in new bits
003D 5F                 MOV     E,A             ;Ready for setting
003E 0E08               MVI     C,B$SETIO       ;Function code
0040 CD0500             CALL    BDOS
0043 F1                 POP     PSW             ;Recover possible data character
0044 C9                 RET

                CONOUT:
0045 5F                 MOV     E,A             ;Get data byte for output
0046 0E06               MVI     C,B$DIRCONIO    ;Function code
0048 C30500             JMP     BDOS            ;BDOS returns to CONOUT's caller

                CONIN:
004B 0E06               MVI     C,B$DIRCONIO    ;Function code
004D 1EFF               MVI     E,0FFH          ;Indicate console input
004F C30500             JMP     BDOS            ;BDOS returns to CONIN's caller

                CONST:
0052 0E0B               MVI     C,B$CONST       ;Function code
0054 CD0500             CALL    BDOS
0057 B7                 ORA     A               ;Set Z-flag to result
0058 C9                 RET
```

Figure 5-11. (Continued)

Function 9: Display "$"-Terminated String

Function Code: C = 09H
Entry Parameters: DE = Address of first byte of string
Exit Parameters: None

Example

```
0009 =          B$PRINTS        EQU     9       ;Print $-Terminated String
0005 =          BDOS            EQU     5       ;BDOS entry point

000D =          CR              EQU     0DH     ;Carriage return
000A =          LF              EQU     0AH     ;Line feed
0009 =          TAB             EQU     09H     ;Horizontal tab
```

```
0000 0D0A095468MESSAGE:         DB      CR,LF,TAB,'This is a message',CR,LF,'$'

0017 0E09               MVI     C,B$PRINTS      ;Function code
0019 110000             LXI     D,MESSAGE       ;Pointer to message
001C CD0500             CALL    BDOS
```

Purpose

This function outputs a string of characters to the console device. The address of this string is in registers DE. You must make sure that the last character of the string is "$"; the BDOS uses this character as a marker for the end of the string. The "$" itself does not get output to the console.

While the BDOS is outputting the string, it expands tabs as previously described, checks to see if there is an incoming character, and checks for CONTROL-S (XOFF, which stops the output until another character is entered) or CONTROL-P (which turns on or off echoing of console characters to the printer).

Notes

One of the biggest drawbacks of this function is its use of "$" as a terminating character. As a result, you cannot output a string with a "$" in it. To be truly general-purpose, it would be better to use a subroutine that used an ASCII NUL (00H) character as a terminator, and simply make repetitive calls to the BDOS CONOUT function (code 2). Figure 5-3 is an example of such a subroutine.

Figure 5-12 shows an example of a subroutine that outputs one of several messages. It selects the message based on a message code that you give it as a parameter. Therefore, it is useful for handling error messages; the calling code can pass it an 8-bit error code. You may find it more flexible to convert this subroutine to using 00H-byte-terminated messages using the techniques shown in Figure 5-3.

```
;OM (Output message)
;This subroutine selects one of several messages based on
; the contents of the A register on entry. It then displays
; this message on the console.

;Each message is declared with a "$" as its last character.
; If the A register contains a value larger than the number
; of messages declared, OM will output "Unknown Message".

;As an option, OM can output carriage return / line feed
; prior to outputting the message text.

;Entry parameters
;     HL -> message table
;             This has the form :
;               DB      3       ;Number of messages in table
;               DW      MSG0    ;Address of text (A = 0)
;               DW      MSG1    ;(A = 1)
;               DW      MSG2    ;(A = 2)
;
;       MSG0:   DB      'Message text$'
;                       ...etc.
;             A = Message code (from 0 on up)
;             B = Output CR/LF if non-zero
```

Figure 5-12. Display $-terminated message on console

```
                        ;       Calling sequence
                        ;               LXI     H,MSG$TABLE
                        ;               LDA     MSGCODE
                        ;               MVI     B,0     ;Suppress CR/LF
                        ;               CALL    OM

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

000D =          CR              EQU     0DH     ;Carriage return
000A =          LF              EQU     0AH     ;Line feed

0000 0D0A24     OM$CRLF:        DB      CR,LF,'$'
0003 556E6B6E6FOM$UM:           DB      'Unknown Message$'

                OM:
0013 F5                 PUSH    PSW             ;Save message code
0014 E5                 PUSH    H               ;Save message table pointer
0015 78                 MOV     A,B             ;Check if CR/LF required
0016 B7                 ORA     A
0017 CA2200             JZ      OM$NOCR         ;No
001A 110000             LXI     D,OM$CRLF       ;Output CR/LF
001D 0E09               MVI     C,B$PRINTS
001F CD0500             CALL    BDOS

                OM$NOCR:
0022 E1                 POP     H               ;Recover message table pointer
0023 F1                 POP     PSW             ;Recover message code
0024 BE                 CMP     M               ;Compare message to max. value
0025 D23700             JNC     OM$ERR          ;Error-code not <= max.
0028 23                 INX     H               ;Bypass max. value in table
0029 87                 ADD     A               ;Message code * 2
002A 5F                 MOV     E,A             ;Make (code * 2) a word value
002B 1600               MVI     D,0
002D 19                 DAD     D               ;HL -> address of message text
002E 5E                 MOV     E,M             ;Get LS byte
002F 23                 INX     H               ;HL -> MS byte
0030 56                 MOV     D,M             ;Get MS byte
                                                ;DE -> message text itself

                OM$PS:                          ;Print string entry point
0031 0E09               MVI     C,B$PRINTS      ;Function code
0033 CD0500             CALL    BDOS
0036 C9                 RET                     ;Return to caller

                OM$ERR:                         ;Error
0037 110300             LXI     D,OM$UM         ;Point to "Unknown Message"
003A C33100             JMP     OM$PS           ;Print string
```

Figure 5-12. (Continued)

Function 10: Read Console String

Function Code: C = 0AH
Entry Parameters: DE = Address of string buffer
Exit Parameters: String buffer with console bytes in it

Example

```
000A =          B$READCONS      EQU     10      ;Read Console String
0005 =          BDOS            EQU     5       ;BDOS entry point
```

```
0050 =          BUFLEN          EQU     80      ;Buffer length

                BUFFER:                         ;Console input buffer
0000 50         BUFMAXCH:       DB      BUFLEN  ;Max. no. of characters in
                                                ; buffer
0001 00         BUFACTCH:       DB      0       ;Actual no. of characters input
0002            BUFCH:          DS      BUFLEN  ;Buffer characters

0052 0E0A               MVI     C,B$READCONS    ;Function code
0054 110000             LXI     D,BUFFER        ;Pointer to buffer
0057 CD0500             CALL    BDOS
```

Purpose

This function reads a string of characters from the console device and stores them in a buffer (address in DE) that you define. Full line editing is possible: the operator can backspace, cancel the line and start over, and use all the normal control functions. What you will ultimately see in the buffer is the final version of the character string entered, without any of the errors or control characters used to do the line editing.

The buffer that you define has a special format. The first byte in the buffer tells the BDOS the maximum number of characters to be accepted. The second byte is reserved for the BDOS to tell you how many characters were actually placed in the buffer. The following bytes contain the characters of the string.

Character input will cease either when a CARRIAGE RETURN is entered or when the maximum number of characters, as specified in the buffer, has been received. The CARRIAGE RETURN is not stored in the buffer as a character—it just serves as a terminator.

If the first character entered is a CARRIAGE RETURN, then the BDOS sets the "characters input" byte to 0. If you attempt to input more than the maximum number of characters, the "characters input" count will be the same as the maximum value allowed.

Notes

This function is useful for accepting console input, especially because of the line editing that it allows. It should be used even for single-character responses, such as "Y/N" (yes or no), because the operator can type "Y", backspace, and overtype with "N". This makes for more "forgiving" programs, tolerant of humans who change their minds.

Figure 5-13 shows an example subroutine that uses this function. It accepts console input, matches the input against a table, and transfers control to the appropriate subroutine. Many interactive programs need to do this; they accept an operator command and then transfer control to the appropriate command processor to deal with that command.

This example also includes two other subroutines that are useful in their own right. One compares null-byte-terminated strings (FSCMP), and the other converts, or "folds," lowercase letters to uppercase (FOLD).

```
                ;RSA
                ;Return subprocessor address
                ;This subroutine returns one of several addresses selected
                ; from a table by matching keyboard input against specified
                ; strings. It is normally used to switch control to a
                ; particular subprocessor according to an option entered
                ; by the operator from the keyboard.
                ;
                ;Character string comparisons are performed with case-folding;
                ; that is, lowercase letters are converted to uppercase.
                ;
                ;If the operator input fails to match any of the specified
                ; strings, then the carry flag is set. Otherwise, it is
                ; cleared.

                ;Entry parameters
                ;       HL -> Subprocessor select table
                ;             This has the form :
                ;             DW      TEXT0,SUBPROC0
                ;             DW      TEXT1,SUBPROC1
                ;             DW      0       ;Terminator
                ;       TEXT0:  DB      'add',0 ;00H-byte terminated
                ;       TEXT1:  DB      'subtract',0
                ;       SUBPROC0:
                ;             Code for processing ADD function.
                ;       SUBPROC1:
                ;             Code for processing SUBTRACT function.

                ;Exit parameters
                ;       DE -> operator input string (00H-terminated
                ;             input string).
                ;       Carry Clear, HL -> subprocessor.
                ;       Carry Set, HL = 0000H.

                ;Calling sequence
                ;       LXI     H,SUBPROCTAB    ;Subprocessor table
                ;       CALL    RSA
                ;       JC      ERROR           ;Carry set only on error
                ;       LXI     D,RETURN        ;Fake CALL instruction
                ;       PUSH    D               ;Push return address on stack
                ;       PCHL                    ;"CALL" to subprocessor
                ;       RETURN:

000A =          B$READCONS      EQU     10      ;Read console string into buffer
0005 =          BDOS            EQU     5       ;BDOS entry point

0050 =          RSA$BL          EQU     80      ;Buffer length
0000 50         RSA$BUF:        DB      RSA$BL  ;Max. no. of characters
0001 00         RSA$ACTC:       DB      0       ;Actual no. of characters
0002            RSA$BUFC:       DS      RSA$BL  ;Buffer characters
0052 00                         DB      0       ;Safety terminator

                RSA:
0053 2B                 DCX     H               ;Adjust Subprocessor pointer
0054 2B                 DCX     H               ; for code below
0055 E5                 PUSH    H               ;Top of stack (TOS) -> subproc. table - 2
0056 0E0A               MVI     C,B$READCONS    ;Function code
0058 110000             LXI     D,RSA$BUF       ;DE -> buffer
005B CD0500             CALL    BDOS            ;Read operator input and
                                                ; Convert to 00H-terminated
005E 210100             LXI     H,RSA$ACTC      ;HL -> actual no. of chars. input
0061 5E                 MOV     E,M             ;Get actual no. of chars. input
0062 1600               MVI     D,0             ;Make into word value
0064 23                 INX     H               ;HL -> first data character
0065 19                 DAD     D               ;HL -> first UNUSED character in buffer
0066 3600               MVI     M,0             ;Make input buffer 00H terminated

                RSA$ML:                         ;Compare input to specified values
                                                ; Main loop
0068 E1                 POP     H               ;Recover subprocessor table pointer
0069 23                 INX     H               ;Move to top of next entry
006A 23                 INX     H               ;HL -> text address
006B 5E                 MOV     E,M             ;Get text address
```

Figure 5-13. Read console string for keyboard options

```
006C 23                 INX     H
006D 56                 MOV     D,M             ;DE -> text

006E 7A                 MOV     A,D             ;Check if at end of subprocessor table
006F B3                 ORA     E
0070 CA8500             JZ      RSA$NFND        ;Match not found

0073 23                 INX     H               ;HL -> subprocessor address
0074 E5                 PUSH    H               ;Save ptr. to subprocessor table
0075 210200             LXI     H,RSA$BUFC      ;HL -> input characters
0078 CD8A00             CALL    FSCMP           ;Folded string compare
007B C26800             JNZ     RSA$ML          ;No match, move to next entry
007E E1                 POP     H               ;Match found, recover subprocessor ptr.
007F 5E                 MOV     E,M             ;Get actual subprocessor address
0080 23                 INX     H
0081 56                 MOV     D,M             ;DE -> Subprocessor code
0082 EB                 XCHG                    ;HL -> Subprocessor code
0083 B7                 ORA     A               ;Clear carry (match found)
0084 C9                 RET

                RSA$NFND:
0085 210000             LXI     H,0             ;Indicate no match found
0088 37                 STC                     ;Set carry
0089 C9                 RET

                ;FSCMP
                ;Compare folded (lowercase to upper) string.
                ;This subroutine compares two 00H-byte terminated
                ;strings and returns with the condition flags set
                ;to indicate their relationship.

                ;Entry parameters
                ;       DE -> string 1
                ;       HL -> string 2

                ;Exit parameters
                ;       Flags set (based on string 1 - string 2, on a
                ;       character-by-character basis)

                FSCMP:
008A 1A                 LDAX    D               ;Get string 1 character
008B CD9E00             CALL    FOLD            ;Fold to uppercase
008E F5                 PUSH    PSW             ;Save string 1 character
008F 7E                 MOV     A,M             ;Get string 2 character
0090 CD9E00             CALL    FOLD            ;Fold to uppercase
0093 47                 MOV     B,A             ;Save string 2 character
0094 F1                 POP     PSW             ;Recover string 1 character
0095 B8                 CMP     B               ;String 1 - string 2
0096 C0                 RNZ                     ;Return if not equal
0097 B7                 ORA     A               ;Equal, so check if end of strings
0098 C8                 RZ                      ;Yes
0099 13                 INX     D               ;No, update string 1 pointer
009A 23                 INX     H               ; and string 2 pointer
009B C38A00             JMP     FSCMP           ;Check next character

                ;FOLD
                ;Folds a lowercase letter (a-z) to uppercase (A-Z)
                ;The character to be folded is in A on entry and on exit.

                FOLD:
009E 4F                 MOV     C,A             ;Preserve input character
009F 3E60               MVI     A,'a'-1         ;Check if folding needed
00A1 B9                 CMP     C               ;Compare to input character
00A2 D2AF00             JNC     FOLDX           ;No, char. is <= "a"
00A5 3E7A               MVI     A,'z'           ;Check if < "z"
00A7 B9                 CMP     C
00A8 DAAF00             JC      FOLDX           ;No, char. is > "z"
00AB 3EDF               MVI     A,0DFH          ;Fold character
00AD A1                 ANA     C
00AE C9                 RET

                FOLDX:
00AF 79                 MOV     A,C             ;Recover original input char.
00B0 C9                 RET
```

Figure 5-13. (Continued)

Function 11: Read Console Status

Function Code: C = 0BH
Entry Parameters: None
Exit Parameters: A = 00H if no incoming data byte
A = 0FFH if incoming data byte

Example

```
000B =          B$CONST         EQU     11      ;Get Console Status
0005 =          BDOS            EQU     5       ;BDOS entry point

0000 0E0B               MVI     C,B$CONST       ;Function code
0002 CD0500             CALL    BDOS            ;A = 00 if no character waiting
                                                ;A = 0FFH if character waiting
```

Purpose This function tells you whether a console input character is waiting to be processed. Unlike the Console Input functions, which will wait until there is input, this function simply checks and returns immediately.

Notes Use this function wherever you want to interrupt an executing program if a console keyboard character is entered. Just put a Console Status call in the main loop of the program. Then, if the program detects that keyboard data is waiting, it can take the appropriate action. Normally this would be to jump to location 0000H, thereby aborting the current program and initiating a warm boot.

Figure 5-11 is an example subroutine that shows how to use this function.

Function 12: Get CP/M Number

Function Code: C = 0CH
Entry Parameters: None
Exit Parameters: HL = Version number code

Example

```
000C =          B$GETVER        EQU     12      ;Get CP/M Version Number
0005 =          BDOS            EQU     5       ;BDOS entry point

0000 0E0C               MVI     C,B$GETVER      ;Function code
0002 CD0500             CALL    BDOS            ;H = 00 for CP/M
                                                ;L = version (e.g. 22H for 2.2)
```

Purpose This function tells you which version of CP/M you are currently running. A two-byte value is returned:

H = 00H for CP/M, H = 01H for MP/M
L = 00H for all releases before CP/M 2.0
L = 20H for CP/M 2.0, 21H for 2.1, 22H for 2.2, and so on for any subsequent releases.

This information is of interest only if your program has some version-specific logic built into it. For example, CP/M version 1.4 does not support the same Random File Input/Output operations that CP/M 2.2 does. Therefore, if your program uses Random I/O, put this check at the beginning to ensure that it is indeed running under the appropriate version of CP/M.

Notes Figure 5-14 is a subroutine that checks the current CP/M version number, and, if it is not CP/M 2.2, displays an explanatory message on the console and does a warm boot by jumping to location 0000H.

Function 13: Reset Disk System

Function Code: C = 0DH
Entry Parameters: None
Exit Parameters: None

```
                        ;CCPM
                        ;Check if CP/M
                        ;This subroutine determines the version number of the
                        ;operating system and, if not CP/M version 2, displays
                        ;an error message and executes a warm boot.

                        ;Entry and exit parameters
                        ;       None

                        ;Calling sequence
                        ;       CALL    CCPM    ;Warm boots if not CP/M 2

0009 =          B$PRINTS        EQU     9       ;Display $-terminated string
000C =          B$GETVER        EQU     12      ;Get version number
0005 =          BDOS            EQU     5       ;BDOS entry point

000D =          CR              EQU     0DH     ;Carriage return
000A =          LF              EQU     0AH     ;Line feed

0000 0D0A       CCPMM:  DB      CR,LF
0002 5468697320         DB      'This program can only run under CP/M version 2.'
0031 0D0A24             DB      CR,LF,'$'

                CCPM:
0034 0E0C               MVI     C,B$GETVER      ;Get version number
0036 CD0500             CALL    BDOS
0039 7C                 MOV     A,H             ;H must be 0 for CP/M
003A B7                 ORA     A
003B C24700             JNZ     CCPME           ;Must be MP/M
003E 7D                 MOV     A,L             ;L = version number of CP/M
003F E6F0               ANI     0F0H            ;Version number in MS nibble
0041 FE20               CPI     20H             ;Check if version 2
0043 C24700             JNZ     CCPME           ;Must be an earlier version
0046 C9                 RET                     ;Yes, CP/M version 2

                CCPME:                          ;Error
0047 0E09               MVI     C,B$PRINTS      ;Display error message
0049 110000             LXI     D,CCPMM
004C CD0500             CALL    BDOS
004F C30000             JMP     0               ;Warm boot
```

Figure 5-14. Determine the CP/M version number

Example

```
000D =          B$DSKRESET      EQU     13      ;Reset Disk System
0005 =          BDOS            EQU     5       ;BDOS entry point

0000 0E0D               MVI     C,B$DSKRESET    ;Function code
0002 CD0500             CALL    BDOS
```

Purpose

This function requests CP/M to completely reset the disk file system. CP/M then resets its internal tables, selects logical disk A as the default disk, resets the DMA address back to 0080H (the address of the buffer used by the BDOS to read and write to the disk), and marks all logical disks as having Read/Write status.

The BDOS will then have to log in each logical disk as each disk is accessed. This involves reading the entire file directory for the disk and rebuilding the allocation vectors (which keep track of which allocation blocks are free and which are used for file storage).

Notes

This function lets you change the diskettes under program control. If the operator were to simply change diskettes, without CP/M knowing about it, the next access to the (now different) diskette would force CP/M to declare the disk Read-Only, thwarting any further attempts to write on the diskette. If you need to reset one or two disks, rather than the entire disk system, look ahead to the Reset Disk function (code 37) described at the end of this chapter.

Figure 5-15 shows a simple subroutine that outputs a message on the console, requesting that the diskette in a specified drive be changed. It then issues a Reset Disk function call to make sure that CP/M will log in the diskette on the next access to the drive.

```
                        ;CDISK
                        ;Change disk
                        ;This subroutine displays a message requesting the
                        ;user to change the specified logical disk, then waits
                        ;for a carriage return to be pressed. It then issues
                        ;a Disk Reset and returns to the caller.

                        ;Entry parameters
                        ;       A = Logical disk to be changed (A = 0, B = 1)

                        ;Exit parameters
                        ;       None

                        ;Calling sequence
                        ;       MVI     A,0             ;Change drive A:
                        ;       CALL    CDISK


000D =          B$DSKRESET      EQU     13      ;Disk Reset function code
0009 =          B$PRINTS        EQU     9       ;Print $-terminated string
0001 =          B$CONIN         EQU     1       ;Get console input
0005 =          BDOS            EQU     5       ;BDOS entry point
```

Figure 5-15. Reset requested disk drive

```
000D =          CR              EQU     0DH
000A =          LF              EQU     0AH

0000 0D0A436861CDISKM:          DB      CR,LF,'Change logical disk '
0016 00         CDISKD:         DB      0
0017 3A20616E64                 DB      ': and press Carriage Return to continue$'

                CDISK:
003F C640               ADI     'A'-1           ;Convert to letter
0041 321600             STA     CDISKD          ;Store in message
0044 0E09               MVI     C,B$PRINTS      ;Display message
0046 110000             LXI     D,CDISKM
0049 CD0500             CALL    BDOS

                CDISKW:
004C 0E01               MVI     C,B$CONIN       ;Get keyboard character
004E CD0500             CALL    BDOS
0051 FE0D               CPI     CR
0053 C24C00             JNZ     CDISKW
0056 0E0D               MVI     C,B$DSKRESET    ;Now reset disk system
0058 CD0500             CALL    BDOS
005B C9                 RET
```

Figure 5-15. Reset requested disk drive (continued)

Function 14: Select Logical Disk

Function Code: C = 0EH
Entry Parameters: E = Logical Disk Code
00H = Drive A
01H = Drive B and so on
Exit Parameters: None

Example

```
000E =          B$SELDSK        EQU     14      ;Select Logical Disk
0005 =          BDOS            EQU     5       ;BDOS entry point

0000 0E0E               MVI     C,B$SELDSK      ;Function code
0002 1E00               MVI     E,0             ;E = 0 for A:, 1 for B: etc.
0004 CD0500             CALL    BDOS
```

Purpose

This function makes the logical disk named in register E the default disk. All subsequent references to disk files that do not specify the disk will use this default.

When you reference a disk file that *does* have an explicit logical disk in its name you do not have to issue another Select Disk function; the BDOS will take care of that for you.

Notes

Notice the way in which the logical disk is specified in register E. It is not the same as the disk drive specification in the first byte of the file control block. In the FCB, a value of 00H is used to mean "use the current default disk" (as specified in the last Select Disk call or by the operator on the console). With this function, a

value of 00H in register A means that A is the selected drive, a value of 01H means drive B, and so on to 0FH for drive P, allowing 16 drives in the system.

If you select a logical disk that does not exist in your computer system, the BDOS will display the following message:

```
BDOS Err on J: Select
```

If you type a CARRIAGE RETURN in order to proceed, the BDOS will do a warm boot and transfer control back to the CCP. To avoid this, you must rely on the computer operator not to specify nonexistent disks or build into your program the knowledge of how many logical disk drives are on the system.

Another problem with this function is that you cannot distinguish a logical disk for which the appropriate tables have been built into the BIOS, but for which there is no physical disk drive. The BDOS does not check to see if the drive is physically present when you make the Select Disk call. It merely sets up some internal values ready to access the logical disk. If you then attempt to access this nonexistent drive, the BIOS will detect the error. What happens next is completely up to the BIOS. The standard BIOS will return control to the BDOS, indicating an error condition. The BDOS will output the message

```
BDOS Err on C: Bad Sector
```

You then have a choice. You can press CARRIAGE RETURN, in which case the BDOS will ignore the error and attempt to continue with whatever appears to have been read in. Or you can enter a CONTROL-C, causing the program to abort and CP/M to perform a warm boot.

Note that the Select Disk function does not return any values. If your program gets control back, you can assume that the logical disk you asked for at least has tables declared for it.

Function 15: Open File

Function Code: C = 0FH
Entry Parameters: DE = Address of file control block
Exit Parameters: A = Directory code

Example

```
000F =          B$OPEN          EQU     15      ;Open File
0005 =          BDOS            EQU     5       ;BDOS entry point

                FCB:                            ;File control block
0000 00         FCB$DISK:       DB      0       ;Search on default disk drive
0001 46494C454EFCB$NAME:        DB      'FILENAME'      ;File name
0009 545950     FCB$TYP:        DB      'TYP'   ;File type
000C 00         FCB$EXTENT:     DB      0       ;Extent
000D 0000       FCB$RESV:       DB      0,0     ;Reserved for CP/M
000F 00         FCB$RECUSED:    DB      0       ;Records used in this extent
0010 0000000000FCB$ABUSED:      DB      0,0,0,0,0,0,0,0 ;Allocation blocks used
0018 0000000000                 DB      0,0,0,0,0,0,0,0
0020 00         FCB$SEQREC:     DB      0       ;Sequential rec. to read/write
```

```
0021 0000       FCB$RANREC:     DW      0       ;Random rec. to read/write
0023 00         FCB$RANRECO:    DB      0       ;Random rec. overflow byte (MS)

0024 0E0F               MVI     C,B$OPEN        ;Function code
0026 110000             LXI     D,FCB           ;DE -> File control block
0029 CD0500             CALL    BDOS            ;A = OFFH if file not found
```

Purpose This function opens a specified file for reading or writing. The FCB, whose address must be in register DE, tells CP/M the user number, the logical disk, the file name, and the file type. All other bytes of the FCB will normally be set to 0.

The code returned by the BDOS in register A indicates whether the file has been opened successfully. If A contains 0FFH, then the BDOS was unable to find the correct entry in the directory. If A = 0, 1, 2, or 3, then the file has been opened.

Notes The Open File function searches the entire file directory on the specified logical disk looking for the file name, type, and extent specified in the FCB; that is, it is looking for an exact match for bytes 1 through 14 of the FCB. The file name and type may be ambiguous; that is, they may contain "?" characters. In this case, the BDOS will open the first file in the directory that matches the ambiguous name in the FCB. If the file name or type is shorter than eight or three characters respectively, then the remaining characters must be filled with blanks.

When the BDOS searches the file directory, it expects to find an *exact* match with each character of the file name and type, including lowercase letters or nongraphic characters. However, the BDOS uses only the least significant seven bits of each character—the most significant bit is used to indicate special file status characteristics, or *attributes.*

By matching the file extent as well as the name and type, you can, if you wish, open the file at some point other than its beginning. For normal sequential access, you would not usually want to do this, but if your program can predict which file extent is required, this is a method of moving directly to it.

It is also possible to open the same file more than once. Each instance requires a separate FCB. The BDOS is not aware that this is happening. It is really only safe to do this when you are reading the file. Each FCB can be used to read the file independently.

Once the file has been found in the directory, the number of records and the allocation blocks used are copied from the directory entry into the FCB (bytes 16 through 31). If the file is to be accessed sequentially from the beginning of the file, the current record (byte 32) must be set to zero by your program.

The value returned in register A is the relative directory entry number of the entry that matched the FCB. As previously explained, the buffer that CP/M uses holds a 128-byte record from the directory with four directory entries numbered 0, 1, 2, and 3. This *directory code* is returned by almost all of the file-related BDOS functions, but under normal circumstances you will be concerned only with whether the value returned in A is 0FFH or not.

Figure 5-16 shows a subroutine that takes a 00H-byte terminated character

string, creates a valid FCB, and then opens the specified file. Shown as part of this example is the subroutine BF (Build FCB). It performs the brunt of the work of converting a string of ASCII characters into an FCB-style disk, file name, and type.

```
                        ;OPENF
                        ;Open File

                        ;Given a pointer to a 00H-byte-terminated file name,
                        ;and an area that can be used for a file control
                        ;block, this subroutine builds a valid file control
                        ;block and attempts to open the file.

                        ;If the file is opened, it returns with the carry flag clear.
                        ;If the file cannot be opened, this subroutine returns
                        ;with the carry flag set.

                        ;Entry parameters
                        ;       DE -> 36-byte area for file control block
                        ;       HL -> 00H-byte terminated file name of the
                        ;               form {disk:} Name {.typ}
                        ;               (disk and typ are optional)

                        ;Exit parameters
                        ;       Carry clear : File opened correctly.
                        ;       Carry set   : File not opened.

                        ;Calling Sequence
                        ;       LXI     D,FCB
                        ;       LXI     H,FNAME
                        ;       CALL    OPENF
                        ;       JC      ERROR
                        ;where
                        ;FCB:   DS      36              ;Space for file control block
                        ;FNAME: DB      'A:TESTFILE.DAT',0

000F =                  B$OPEN          EQU     15      ;File Open function code
0005 =                  BDOS            EQU     5       ;BDOS entry point

                        OPENF:
0000 D5                         PUSH    D               ;Preserve pointer to FCB
0001 CD0C00                     CALL    BF              ;Build file control block
0004 0E0F                       MVI     C,B$OPEN
0006 D1                         POP     D               ;Recover pointer to FCB
0007 CD0500                     CALL    BDOS
000A 17                         RAL                     ;If A=0FFH, carry set
                                                        ;otherwise carry clear
000B C9                         RET

                        ;BF
                        ;Build file control block
                        ;This subroutine formats a 00H-byte-terminated string
                        ;(presumed to be a file name) into an FCB, setting
                        ;the disk and file name and type and clearing the
                        ;remainder of the FCB to 0's.

                        ;Entry parameters
                        ;       DE -> file control block (36 Bytes)
                        ;       HL -> file name string (00H-byte-terminated)

                        ;Exit parameters
                        ;       The built file control block
                        ;Calling sequence
                        ;       LXI     D,FCB
                        ;       LXI     H,FILENAME
                        ;       CALL    BF

                        BF:
```

Figure 5-16. Open file request

```
000C 23              INX     H               ;Check if 2nd char. is ":"
000D 7E              MOV     A,M             ;Get character from file name
000E 2B              DCX     H               ;HL -> now back at 1st char.
000F FE3A            CPI     ':'             ;If ":", then disk specified
0011 C21C00          JNZ     BF$ND           ;No disk
0014 7E              MOV     A,M             ;Get disk letter
0015 E61F            ANI     0001$1111B      ;A (41H) -> 1, B (42H) -> 2 ...
0017 23              INX     H               ;Bypass disk letter
0018 23              INX     H               ;Bypass ":"
0019 C31D00          JMP     BF$SD           ;Store disk in FCB

                BF$ND:                       ;No disk present
001C AF              XRA     A               ;Indicate default disk

                BF$SD:
001D 12              STAX    D               ;Store disk in FCB
001E 13              INX     D               ;DE -> 1st char. of name in FCB
001F 0E08            MVI     C,8             ;File name length
0021 CD3700          CALL    BF$GT           ;Get token
                                             ;Note -- at this point, BF$GT
                                             ;will have advanced the string
                                             ;pointer to either a "." or
                                             ;00H byte
0024 FE2E            CPI     '.'             ;Check terminating character
0026 C22A00          JNZ     BF$NT           ;No file type specified
0029 23              INX     H               ;Bypass "." in file name

                BF$NT:
002A 0E03            MVI     C,3             ;File type length
002C CD3700          CALL    BF$GT           ;Get token
                                             ;Note -- if no file type is
                                             ;present BF$GT will merely
                                             ;spacefill the FCB
002F 0600            MVI     B,0             ;0-fill the remainder of the FCB
0031 0E18            MVI     C,24            ;36 - 12 (disk, name, type = 12 chars.)
0033 CD6400          CALL    BF$FT           ;Re-use fill token S/R
0036 C9              RET

                ;BF$GT
                ;Build FCB -- get token

                ;This subroutine scans a file name string,
                ;placing characters into a file control block.
                ;On encountering a terminator character ("." or 00H),
                ;the remainder of the token is space filled.
                ;If an "*" is encountered, the remainder of the token
                ;is filled with "?".

                ;Entry parameters
                ;       DE -> Into file control block
                ;       HL -> Into file name string
                ;       C = Maximum no. of characters in token

                ;Exit parameters
                ;       File control block contains next token
                ;       A = Terminating character

                BF$GT:
0037 7E              MOV     A,M             ;Get next string character
0038 B7              ORA     A               ;Check if end of string
0039 CA5700          JZ      BF$SFT          ;Yes, space fill token
003C FE2A            CPI     '*'             ;Check if ?-fill required
003E CA5C00          JZ      BF$QFT          ;Yes, fill with ?
0041 FE2E            CPI     '.'             ;Assume current token is file
                                             ;name
                                             ;Check if file type coming up
                                             ;(If current token is file
                                             ;type this check is
                                             ;benignly redundant)
0043 CA5700          JZ      BF$SFT          ;Yes, space fill token
0046 12              STAX    D               ;None of the above, so store
                                             ;in FCB
0047 13              INX     D               ;Update FCB pointer
0048 23              INX     H               ;Update string pointer
```

Figure 5-16. (Continued)

```
0049 0D              DCR     C              ;Countdown on token length
004A C23700          JNZ     BF$GT          ;Still more characters to go

                BF$SKIP:                    ;Skip chars. until "." or 00H
004D 7E              MOV     A,M            ;Get next string character
004E B7              ORA     A              ;Check if 00H
004F C8              RZ                     ;Yes
0050 FE2E            CPI     '.'            ;Check if "."
0052 C8              RZ                     ;Yes
0053 23              INX     H              ;Update string pointer (only)
0054 C34D00          JMP     BF$SKIP        ;Try next character

                BF$SFT:                     ;Space fill token
0057 0620            MVI     B,' '
0059 C36400          JMP     BF$FT          ;Common fill token code
                                            ;BF$FT returns to caller

                BF$QFT:                     ;Question mark fill token
005C 063F            MVI     B,'?'
005E CD6400          CALL    BF$FT          ;Common fill token code
0061 C34D00          JMP     BF$SKIP        ;Bypass multiple "*" etc.

                BF$FT:                      ;Fill token
0064 F5              PUSH    PSW            ;Save terminating character
0065 78              MOV     A,B            ;Get fill characer

                BF$FTL:                     ;Inner loop
0066 12              STAX    D              ;Store in FCB
0067 13              INX     D              ;Update FCB Pointer
0068 0D              DCR     C              ;Downdate residual count
0069 C26600          JNZ     BF$FTL         ;Keep going
006C F1              POP     PSW            ;Recover terminating character
006D C9              RET
```

Figure 5-16. (Continued)

Function 16: Close File

Function Code: C = 10H
Entry Parameters: DE = Address of file control block
Exit Parameters: A = Directory code

Example

```
0010 =          B$CLOSE         EQU     16      ;Close File
0005 =          BDOS            EQU     5       ;BDOS entry point

0000            FCB:            DS      36      ;File control block

0024 0E10               MVI     C,B$CLOSE       ;Function code
0026 110000             LXI     D,FCB           ;DE -> File control block
0029 CD0500             CALL    BDOS            ;A = 0,1,2,3 if successful
                                                ;A = OFFH if file name not
                                                ; in directory
```

Purpose This function terminates the processing of a file to which you have written information. Under CP/M you do not need to close a file that you have been reading. However, if you ever intend for your program to function correctly under MP/M (the multi-user version of CP/M) you should close all files regardless of their use.

The Close File function, like Open File, returns a directory code in the A register. Register A will contain 0FFH if the BDOS could not close the file successfully. If A is 0, 1, 2, or 3, then the file has been closed.

Notes

When the BDOS closes a file to which data has been written, it writes the current contents of the FCB out to the disk directory, updating an existing directory entry by matching the disk, name, type, and extent number in the same manner that the Open File function does.

Note that the BDOS does not transfer the last record of the file to the disk during the close operation. It merely updates the file directory. You must arrange to flush any partly filled record to the disk. If the file that you have created is a standard CP/M ASCII text file, you must arrange to fill the unused portion of the record with the standard 1AH end-of-file characters as CP/M expects, as explained in the section on the Write Sequential function (code 21).

Function 17: Search for First Name Match

Function Code: C = 11H
Entry Parameters: DE = Address of file control block
Exit Parameters: A = Directory code

Example

```
0011 =          B$SEARCHF       EQU     17      ;Search First
0005 =          BDOS            EQU     5       ;BDOS entry point

                FCB:                            ;File control block
0000 00         FCB$DISK:       DB      0       ;Search on default disk drive
0001 46494C453FFCB$NAME:        DB      'FILE????'      ;Ambiguous file name
0009 543F50     FCB$TYP:        DB      'T?P'   ;Ambiguous file type
000C 00         FCB$EXTENT:     DB      0       ;Extent
000D 0000       FCB$RESV:       DB      0,0     ;Reserved for CP/M
000F 00         FCB$RECUSED:    DB      0       ;Records used in this extent
0010 0000000000FCB$ABUSED:      DB      0,0,0,0,0,0,0,0 ;Allocation blocks used
0018 0000000000                 DB      0,0,0,0,0,0,0,0
0020 00         FCB$SEQREC:     DB      0       ;Sequential rec. to read/write
0021 0000       FCB$RANREC:     DW      0       ;Random rec. to read/write
0023 00         FCB$RANRECO:    DB      0       ;Random rec. overflow byte (MS)

0024 0E11               MVI     C,B$SEARCHF     ;Function code
0026 110000             LXI     D,FCB           ;DE -> File control block
0029 CD0500             CALL    BDOS            ;A = 0,1,2,3.
                                                ;(A * 32) + DMA -> directory
                                                ; entry
                                                ;A = 0FFH if file name not
                                                ; found
```

Purpose

This function scans down the file directory for the first entry that matches the file name, type, and extent in the FCB addressed by DE. The file name, type, and extent may contain a "?" (ASCII 3FH) in one or more character positions. Where a "?" occurs, the BDOS will match *any* character in the corresponding position in the file directory. This is known as ambiguous file name matching.

The first byte of an FCB normally contains the logical disk number code. A value of 0 indicates the default disk, while 1 means disk A, 2 is B, and so on up to a

possible maximum of 16 for disk P. However, if this byte contains a "?", the BDOS will search the default logical disk and will match the file name and type regardless of the user number. This function is normally used in conjunction with the Search Next function (which is described immediately after this function). Search First, in the process of matching a file, leaves certain variables in the BDOS set, ready for a subsequent Search Next.

Both Search First and Search Next return a directory code in the A register. With Search First, A = 0FFH when no files match the FCB; if a file match is found, A will have a value of 0, 1, 2, or 3.

Notes

To locate the particular directory entry that either the Search First or Search Next function matched, multiply the directory code returned in A by the length of a directory entry (32 bytes). This is easily done by adding the A register to itself five times (see the code in Figure 5-17 near the label GNFC). Then add the DMA address to get the actual address where the matched directory entry is stored.

There are many occasions when you may need to write a program that will accept an ambiguous file name and operate on all of the file names that match it. (The DIR and ERA commands built into the CCP are examples that use ambiguous file names.) To do this, you must use several BDOS functions: the Set DMA Address function (code 26, described later in this chapter), this function (Search First), and Search Next (code 18). All of this is shown in the subroutine given in Figure 5-17.

```
;GNF
;This subroutine returns an FCB setup with either the
;first file matched by an ambiguous file name, or (if
;specified by entry parameter) the next file name.

;Note : this subroutine is context sensitive. You must
;       not have more than one ambiguous file name
;       sequence in process at any given time.

;>>>    Warning : This subroutine changes the DMA address
;>>>              inside the BDOS.

;Entry parameters
;       DE -> Possibly ambiguous file name
;             (00-byte terminated)
;             (Only needed for FIRST request)
;       HL -> File control block
;       A = 0  : Return FIRST file name that matches
;         = NZ : Return NEXT file name that matches

;Exit parameters
;Carry set : A = FF, no file name matches
;          A not = 0FFH, error in input file name
;Carry clear : FCB setup with next name
;            HL -> Directory entry returned
;                  by Search First/Next

;Calling sequence
;       LXI     D,FILENAME
;       LXI     H,FCB
```

Figure 5-17. Search first/next calls for ambiguous file name

```
                ;       MVI     A,0     ;or MVI A,1 for NEXT
                ;       CALL    GNF

0011 =          B$SEARCHF       EQU     17      ;Search for first file name
0012 =          B$SEARCHN       EQU     18      ;Search for next file name
001A =          B$SETDMA        EQU     26      ;Set up DMA address
0005 =          BDOS            EQU     5       ;BDOS entry point

0080 =          GNFDMA  EQU     80H             ;Default DMA address
000D =          GNFSVL  EQU     13              ;Save length (no. of chars to move)
0024 =          GNFFCL  EQU     36              ;File control block length
0000            GNFSV:  DS      GNFSVL          ;Save area for file name/type

                GNF:
000D E5                 PUSH    H               ;Save FCB pointer
000E D5                 PUSH    D               ;Save file name pointer
000F F5                 PUSH    PSW             ;Save first/next flag

0010 118000             LXI     D,GNFDMA        ;Set DMA to known address
0013 0E1A               MVI     C,B$SETDMA      ;Function code
0015 CD0500             CALL    BDOS
0018 F1                 POP     PSW             ;Recover first/next flag
0019 E1                 POP     H               ;Recover file name pointer
001A D1                 POP     D               ;Recover FCB pointer
001B D5                 PUSH    D               ;Resave FCB pointer

001C B7                 ORA     A               ;Check if FIRST or NEXT
001D C23E00             JNZ     GNFN            ;NEXT
0020 CD9300             CALL    BF              ;Build file control block
0023 E1                 POP     H               ;Recover FCB pointer (to balance stack)
0024 D8                 RC                      ;Return if error in file name
0025 E5                 PUSH    H               ;Resave FCB pointer

                                                ;Move ambiguous file name to
                                                ;save area
                                                ;HL -> FCB
0026 110000             LXI     D,GNFSV         ;DE -> save area
0029 0E0D               MVI     C,GNFSVL        ;Get save length
002B CD8A00             CALL    MOVE
002E D1                 POP     D               ;Recover FCB pointer
002F D5                 PUSH    D               ;and resave

0030 0E11               MVI     C,B$SEARCHF     ;Search FIRST
0032 CD0500             CALL    BDOS
0035 E1                 POP     H               ;Recover FCB pointer
0036 FEFF               CPI     0FFH            ;Check for error
0038 CA7D00             JZ      GNFEX           ;Error exit
003B C35D00             JMP     GNFC            ;Common code

                GNFN:                           ;Execute search FIRST to re-
                                                ;establish contact with
                                                ;previous file
                                                ;User's FCB still has
                                                ;name/type in it
003E CD7F00             CALL    GNFZF           ;Zero-fill all but file name/type
0041 D1                 POP     D               ;Recover FCB address
0042 D5                 PUSH    D               ;and resave
0043 0E11               MVI     C,B$SEARCHF     ;Re-find the file
0045 CD0500             CALL    BDOS
0048 D1                 POP     D               ;Recover FCB pointer
0049 D5                 PUSH    D               ;and resave
004A 210000             LXI     H,GNFSV         ;Move file name from save area
                                                ;into FCB
004D 0E0D               MVI     C,GNFSVL        ;Save area length
004F CD8A00             CALL    MOVE

0052 0E12               MVI     C,B$SEARCHN     ;Search NEXT
0054 CD0500             CALL    BDOS
0057 E1                 POP     H               ;Recover FCB address
0058 FEFF               CPI     0FFH            ;Check for error
005A CA7D00             JZ      GNFEX           ;Error exit

                GNFC:
005D E5                 PUSH    H               ;Save FCB address
005E 87                 ADD     A               ;Multiply BDOS return code * 32
```

Figure 5-17. (Continued)

```
005F 87              ADD     A               ;* 4
0060 87              ADD     A               ;* 8
0061 87              ADD     A               ;* 16
0062 87              ADD     A               ;* 32
0063 218000          LXI     H,GNFDMA        ;HL -> DMA address
0066 5F              MOV     E,A             ;Make (code * 32) a word value
                                             ;in DE
0067 1600            MVI     D,0
0069 19              DAD     D               ;HL -> file's directory entry

                                             ;Move file name into FCB
006A D1              POP     D               ;Recover FCB address
006B E5              PUSH    H               ;Save directory entry pointer
006C D5              PUSH    D               ;and resave
006D 0E0D            MVI     C,GNFSVL        ;Length of save area
006F CD8A00          CALL    MOVE
0072 3A0000          LDA     GNFSV           ;Get disk from save area
0075 D1              POP     D               ;Recover FCB address
0076 12              STAX    D               ;Overwrite user number in FCB

                                             ;Set up to zero-fill tail end
                                             ;of FCB
0077 CD7F00          CALL    GNFZF           ;Zero-fill
007A E1              POP     H               ;Recover directory entry
                                             ;pointer
007B AF              XRA     A               ;Clear carry
007C C9              RET

                GNFEX:
007D 37              STC                     ;Set carry to indicate error
007E C9              RET

                ;GNFZF
                ;Get next file -- zero fill
                ;This subroutine zero-fills the bytes that follow the
                ;file name and type in an FCB.

                ;Entry parameters
                ;       DE -> file control block

                GNFZF:
007F 210D00          LXI     H,GNFSVL        ;Bypass area that holds file name
0082 19              DAD     D               ;HL -> FCB + GNFSVL
0083 54              MOV     D,H             ;DE -> FCB + GNFSVL
0084 5D              MOV     E,L
0085 13              INX     D               ;DE -> FCB + GNFSVL + 1
0086 3600            MVI     M,0             ;FCB + GNFSVL = 0
0088 0E17            MVI     C,GNFFCL-GNFSVL ;Remainder of file control block

                ;Drop into MOVE
                ;Spread 0's through remainder
                ;of FCB

                ;MOVE
                ;This subroutine moves C bytes from HL to DE.

                MOVE:
008A 7E              MOV     A,M             ;Get source byte
008B 12              STAX    D               ;Save destination byte
008C 13              INX     D               ;Increment destination pointer
008D 23              INX     H               ;Increment source pointer
008E 0D              DCR     C               ;Decrement count
008F C28A00          JNZ     MOVE            ;Go back for more
0092 C9              RET

                ;BF
                ;Build file control block

                ;This subroutine formats a 00H-byte terminated string
                ;(presumed to be a file name) into an FCB, setting the
                ;disk and file name and type, and clearing the
                ;remainder of the FCB to 0's.
```

Figure 5-17. (Continued)

```
                        ;Entry parameters
                        ;       DE -> File control block (36 bytes)
                        ;       HL -> File name string (00H-byte-terminated)

                        ;Exit parameters
                        ;       The built file control block

                        ;This subroutine is shown in full in Figure 5-16

0093 C9                 BF:     RET                     ;Dummy subroutine for this example
```

Figure 5-17. (Continued)

Function 18: Search for Next Name Match

Function Code: C = 12H
Entry Parameters: None (assumes previous Search First call)
Exit Parameters: A = Directory code

Example

```
0012 =          B$SEARCHN       EQU     18      ;Search Next
0005 =          BDOS            EQU     5       ;BDOS entry point

0000 0E12               MVI     C,B$SEARCHN     ;Function code
                                                ;Note: No FCB pointer
                                                ;You must precede this call
                                                ; with a call to Search First
0002 CD0500             CALL    BDOS            ;A = 0,1,2,3
                                                ;(A * 32) + DMA -> directory
                                                ; entry
                                                ;A = 0FFH if file name not
                                                ; found
```

Purpose This function searches down the file directory for the *next* file name, type, and extent that match the FCB specified in a previous Search First function call.

Search First and Search Next are the only BDOS functions that must be used together. As you can see, the Search Next function does not require an FCB address as an input parameter—all the necessary information will have been left in the BDOS on the Search First call.

Like Search First, Search Next returns a directory code in the A register; in this case, if A = 0FFH, it means that there are no *more* files that match the file control block. If A is not 0FFH, it will be a value of 0, 1, 2, or 3, indicating the relative directory entry number.

Notes There are two ways of using the Search First/Next calls. Consider a simple file copying program that takes as input an ambiguous file name. You could scan the file directory, matching all of the possible file names, possibly displaying them on the console, and storing the names of the files to be copied in a table inside your program. This would have the advantage of enabling you to present the file names

to the operator before any copying occurred. You could even arrange for the operator to select which files to copy on a file-by-file basis. One disadvantage would be that you could not accurately predict how many files might be selected. On some hard disk systems you might have to accommodate several thousand file names.

The alternative way of handling the problem would be to match one file name, copy it, then match the next file name, copy it, and so on. If you gave the operator the choice of selecting which files to copy, this person would have to wait at the terminal as each file was being copied, but the program would not need to have large table areas set aside to hold file names. This solution to the problem is slightly more complicated, as you can see from the logic in Figure 5-17.

The subroutine in Figure 5-17, Get Next File (GNF), contains all of the necessary logic to search down a directory for both alternatives described. It does require that you indicate *on entry* whether it should search for the first or next file match, by setting A to zero or some nonzero value respectively.

You can see from Figure 5-17 that whenever the subroutine is called to get the *next* file, you must execute a Search First function to re-find the previous file. Only then can a Search Next be issued.

As with all functions that return a directory code in A, if this value is not 0FFH, it will be the relative directory entry number in the directory record currently in memory. This directory record will have been read into memory at whatever address was specified at the last Set DMA Address function call (code 26, 1AH). Notwithstanding its odd name, the DMA Address is simply the address into which any record input from disk will be placed. If the Set DMA Address function has not been used to change the value, then the CP/M default DMA address, location 0080H, will be used to hold the directory record.

The actual code for locating the address of the particular directory entry matched by the Search First/Next functions is shown in Figure 5-17 near the label GNFC. The method involves multiplying the directory code by 32 and then adding this product to the current DMA address.

Function 19: Erase (Delete) File

Function Code: C = 13H
Entry Parameters: DE = Address of file control block
Exit Parameters: A = Directory code

Example

```
0013 =          B$ERASE         EQU     19      ;Erase File
0005 =          BDOS            EQU     5       ;BDOS entry point

                FCB:                            ;File control block
0000 00         FCB$DISK:       DB      0       ;Search on default disk drive
0001 3F3F4C454EFCB$NAME:        DB      '??LENAME'      ;Ambiguous file name
0009 3F5950     FCB$TYP:        DB      '?YP'   ;Ambiguous file type
000C 00         FCB$EXTENT:     DB      0       ;Extent
```

```
000D 0000        FCB$RESV:       DB      0,0     ;Reserved for CP/M
000F 00          FCB$RECUSED:    DB      0       ;Records used in this extent
0010 0000000000FCB$ABUSED:       DB      0,0,0,0,0,0,0,0 ;Allocation blocks used
0018 0000000000                  DB      0,0,0,0,0,0,0,0
0020 00          FCB$SEQREC:     DB      0       ;Sequential rec. to read/write
0021 0000        FCB$RANREC:     DW      0       ;Random rec. to read/write
0023 00          FCB$RANRECO:    DB      0       ;Random rec. overflow byte (MS)

0024 0E13                MVI     C,B$ERASE       ;Function code
0026 110000              LXI     D,FCB           ;DE -> file control block
0029 CD0500              CALL    BDOS            ;A = OFFH if file not found
```

Purpose This function logically deletes from the file directory files that match the FCB addressed by DE. It does so by replacing the first byte of each relevant directory entry (remember, a single file can have several entries, one for each extent) by the value 0E5H. This flags the directory entry as being available for use.

Notes Like the previous two functions, Search First and Search Next, this function can take an ambiguous file name and type as part of the file control block, but unlike those functions, the logical disk select code cannot be a "?".

This function returns a directory code in A in the same way as the previous file operations.

Function 20: Read Sequential

Function Code: C = 14H
Entry Parameters: DE = Address of file control block
Exit Parameters: A = Directory code

Example

```
0014 =           B$READSEQ       EQU     20      ;Read Sequential
0005 =           BDOS            EQU     5       ;BDOS entry point

                 FCB:                            ;File control block
0000 00          FCB$DISK:       DB      0       ;Search on default disk drive.
0001 46494C454EFCB$NAME:         DB      'FILENAME'      ;file name
0009 545950      FCB$TYP:        DB      'TYP'   ;File type
000C                             DS      24      ;Set by file open


                                                 ;Record will be read into
                                                 ; address set by prior SETDMA
                                                 ; call
0024 0E14                MVI     C,B$READSEQ     ;Function code
0026 110000              LXI     D,FCB           ;DE -> File control block
0029 CD0500              CALL    BDOS            ;A = 00 if operation successful
                                                 ;A  = nonzero if no data in
                                                 ; file
```

Purpose This function reads the next record (128-byte sector) from the designated file into memory at the address set by the last Set DMA function call (code 26, 1AH). The record read is specified by the FCB's sequential record field (FCB$SEQREC in the example listing for the Open File function, code 15). This field is incremented by 1 so that a subsequent call to Read Sequential will get the next record from the file. If the end of the current extent is reached, then the BDOS will

```
                        ;GETC
                        ;This subroutine gets the next character from a
                        ;sequential disk file. It assumes that the file has
                        ;already been opened.

                        ;>>>    Note : this subroutine changes CP/M's DMA address.

                        ;Entry parameters
                        ;       DE -> file control block

                        ;Exit parameters
                        ;       A = next character from file
                        ;           (= 0FFH on physical end of file)
                        ;           Note : 1AH is normal EOF character for
                        ;                  ASCII Files.

                        ;Calling sequence
                        ;       LXI     DE,FCB
                        ;       CALL    GETC
                        ;       CPI     1AH
                        ;       JZ      EOFCHAR
                        ;       CPI     0FFH
                        ;       JZ      ACTUALEOF

0014 =          B$READSEQ       EQU     20      ;Read sequential
001A =          B$SETDMA        EQU     26      ;Set DMA address
0005 =          BDOS            EQU     5       ;BDOS entry point

0080 =          GETCBS  EQU     128             ;Buffer size
0000            GETCBF: DS      GETCBS          ;Declare buffer
0080 00         GETCCC: DB      0               ;Char. count (initially
                                                ;"empty")

                GETC:
0081 3A8000             LDA     GETCCC          ;Check if buffer is empty
0084 B7                 ORA     A
0085 CA9900             JZ      GETCFB          ;Yes, fill buffer

                GETCRE:                         ;Re-entry point after buffer filled
0088 3D                 DCR     A               ;No, downdate count
0089 328000             STA     GETCCC          ;Save downdated count

008C 47                 MOV     B,A             ;Compute offset of next
                                                ;character
008D 3E7F               MVI     A,GETCBS-1      ;By subtracting
008F 90                 SUB     B               ;(buffer size -- downdated count)
0090 5F                 MOV     E,A             ;Make result into word value
0091 1600               MVI     D,0
0093 210000             LXI     H,GETCBF        ;HL -> base of buffer
0096 19                 DAD     D               ;HL -> next character in buffer
0097 7E                 MOV     A,M             ;Get next character
0098 C9                 RET

                GETCFB:                         ;Fill buffer
0099 D5                 PUSH    D               ;Save FCB pointer
009A 110000             LXI     D,GETCBF        ;Set DMA address to buffer
009D 0E1A               MVI     C,B$SETDMA      ;function code
009F CD0500             CALL    BDOS
00A2 D1                 POP     D               ;Recover FCB pointer
00A3 0E14               MVI     C,B$READSEQ     ;Read sequential "record" (sector)
00A5 CD0500             CALL    BDOS
00A8 B7                 ORA     A               ;Check if read unsuccessful (A = NZ)
00A9 C2B400             JNZ     GETCX           ;Yes
00AC 3E80               MVI     A,GETCBS        ;Reset count
00AE 328000             STA     GETCCC
00B1 C38800             JMP     GETCRE          ;Re-enter subroutine

                GETCX:                          ;Physical end of file
00B4 3EFF               MVI     A,0FFH          ;Indicate such
00B6 C9                 RET
```

Figure 5-18. Read next character from sequential disk file

automatically open the next extent and reset the sequential record field to 0, ready for the next Read function call.

The file specified in the FCB must have been readied for input by issuing an Open File (code 15, 0FH) or a Create File (code 22, 16H) BDOS call.

The value 00H is returned in A to indicate a successful Read Sequential operation, while a nonzero value shows that the Read could not be completed because there was no data in the next record, as at the end of file.

Notes

Although it is not immediately obvious, you can change the sequential record number, FCB$SEQREC, and within a given extent, read a record at random. If you want to access any given record within a file, you must compute which extent that record would be in and set the extent field in the file control block (FCB$EXTENT) before you open the file. Thus, although the function name implies sequential access, in practice you can use it to perform a simple type of random access. If you need to do true random access, look ahead to the Random Read function (code 33), which takes care of opening the correct extent automatically.

Figure 5-18 shows an example of a subroutine that returns the data from a sequential file byte-by-byte, reading in records from the file as necessary. This subroutine, GETC, is useful as a low-level "primitive" on which you can build more sophisticated functions, such as those that read a fixed number of characters or read characters up to a CARRIAGE RETURN/LINE FEED combination.

When you read data from a CP/M text file, the normal convention is to fill the last record of the file with 1AH characters (CONTROL-Z). Therefore, two possible conditions can indicate end-of-file: either encountering a 1AH, or receiving a return code from the BDOS function (in the A register) of 0FFH. However, if the file that you are reading is not an ASCII text file, then a 1AH character has no special meaning—it is just a normal data byte in the body of the file.

Function 21: Write Sequential

Function Code: C = 15H
Entry Parameters: DE = Address of file control block
Exit Parameters: A = Directory code

Example

```
0015 =          B$WRITESEQ      EQU     21      ;Write Sequential
0005 =          BDOS            EQU     5       ;BDOS entry point

                FCB:                            ;File control block
0000 00         FCB$DISK:       DB      0       ;Search on default disk drive
0001 46494C454EFCB$NAME:        DB      'FILENAME'      ;file name
0009 545950     FCB$TYP:        DB      'TYP'   ;File type
000C                            DS      24      ;Set by Open or Create File

                                                ;Record must be in address
                                                ; set by prior SETDMA call
0024 0E15               MVI     C,B$WRITESEQ    ;Function code
0026 110000             LXI     D,FCB           ;DE -> File control block
0029 CD0500             CALL    BDOS            ;A = 00H if operation
                                                ; successful
                                                ;A = nonzero if disk full
```

Purpose This function writes a record from the address specified in the last Set DMA (code 26, 1AH) function call to the file defined in the FCB. The sequential record number in the FCB (FCB$SEQREC) is updated by 1 so that the next call to Write Sequential will write to the next record position in the file. If necessary, a new extent will be opened to receive the new record.

This function is directly analogous to the Read Sequential function, writing instead of reading. The file specified in the FCB must first be activated by an Open File (code 15, 0FH) or create File call (code 22, 16H).

A directory code of 00H is returned in A to indicate that the Write was successful; a nonzero value is returned if the Write could not be completed because the disk was full.

Notes As with the Read Sequential function (code 20, 14H), you can achieve a simple form of random writing to the file by manipulating the sequential record number (FCB$SEQREC). However, you can only overwrite *existing* records in the file, and if you want to move to another extent, you must close the file and reopen it with the FCB$EXTENT field set to the correct value. For true random writing to the file, look ahead to the Write Random function (code 34, 22H). This takes care of opening or creating the correct extent of the file automatically.

The only logical error condition that can occur when writing to a file is insufficient room on the disk to accommodate the next extent of the file. Any hardware errors detected will be handled by the disk driver built into the BIOS or BDOS.

Figure 5-19 shows a subroutine, PUTC, to which you can pass data a byte at a time. It assembles this data into a buffer, making a call to Write Sequential whenever the buffer becomes full. You can see that provision is made in the entry parameters (by setting register B to a nonzero value) for the subroutine to fill the remaining unused characters of the buffer with 1AH characters. You must do this to denote the end of an ASCII text file.

Function 22: Create (Make) File

Function Code: C = 16H
Entry Parameters: DE = Address of file control block
Exit Parameters: A = Directory code

Example

```
0016 =          B$CREATE        EQU     22      ;File Create
0005 =          BDOS            EQU     5       ;BDOS entry point

                FCB:                            ;File control block
0000 00         FCB$DISK:       DB      0       ;Search on default disk drive
0001 46494C454EFCB$NAME:        DB      'FILENAME'      ;file name
0009 545950     FCB$TYP:        DB      'TYP'   ;File type
000C 00         FCB$EXTENT:     DB      0       ;Extent
```

```
000D 0000        FCB$RESV:       DB      0,0       ;Reserved for CP/M
000F 00          FCB$RECUSED:    DB      0         ;Records used in this extent
0010 0000000000FCB$ABUSED:       DB      0,0,0,0,0,0,0,0 ;Allocation blocks used
0018 0000000000                  DB      0,0,0,0,0,0,0,0
0020 00          FCB$SEQREC:     DB      0         ;Sequential rec. to read/write
0021 0000        FCB$RANREC:     DW      0         ;Random rec. to read/write
0023 00          FCB$RANRECO:    DB      0         ;Random rec. overflow byte (MS)

                                                   ;Note : file to be created
                                                   ;must not already exist....
0024 0E16                MVI     C,B$CREATE        ;Function code
0026 110000              LXI     D,FCB             ;DE -> file control block
0029 CD0500              CALL    BDOS              ;A = 0,1,2,3 if operation
                                                   ; successful
                                                   ;A = OFFH if directory full
```

```
                ;PUTC
                ;This subroutine either puts the next chararacter out
                ;to a sequential file, writing out completed "records"
                ;(128-byte sectors) or, if requested to, will fill the
                ;remainder of the current "record" with 1AH's to
                ;indicate end of file to CP/M.

                ;Entry parameters
                ;       DE -> File control block
                ;       B = 0, A = next data character to be output
                ;       B /= 0, fill the current "record" with 1AH's

                ;Exit parameters
                ;       none.

                ;Calling sequence
                ;       LXI     D,FCB
                ;       MVI     B,0     ;Not end of file
                ;       LDA     CHAR
                ;       CALL    PUTC
                ;     or
                ;       LXI     D,FCB
                ;       MVI     B,1     ;Indicate end of file
                ;       CALL    PUTC

0015 =          B$WRITESEQ      EQU     21      ;Write sequential
001A =          B$SETDMA        EQU     26      ;Set DMA address
0005 =          BDOS            EQU     5       ;BDOS entry point

0080 =          PUTCBS  EQU     128             ;Buffer size
0000            PUTCBF: DS      PUTCBS          ;Declare buffer
0080 00         PUTCCC: DB      0               ;Char. count (initially "empty")

                PUTC:
0081 D5                 PUSH    D               ;Save FCB address
0082 F5                 PUSH    PSW             ;Save data character
0083 78                 MOV     A,B             ;Check if end of file requested
0084 B7                 ORA     A
0085 C29900             JNZ     PUTCEF          ;Yes
0088 CDC300             CALL    PUTCGA          ;No, get address of next free byte
                                                ;HL -> next free byte
                                                ;E = Current char. count (as
                                                ;well as A)
008B F1                 POP     PSW             ;Recover data character
008C 77                 MOV     M,A             ;Save in buffer
008D 7B                 MOV     A,E             ;Get current character count
008E 3C                 INR     A               ;Update character count
008F FE80               CPI     PUTCBS          ;Check if buffer full
0091 CAA900             JZ      PUTCWB          ;Yes, write buffer
0094 328000             STA     PUTCCC          ;No, save updated count
0097 D1                 POP     D               ;Dump FCB address for return
0098 C9                 RET
```

Figure 5-19. Write next character to sequential disk file

```
                PUTCEF:                         ;End of file
0099 F1                 POP     PSW             ;Dump data character
009A CDC300             CALL    PUTCGA          ;HL -> next free byte
                                                ;A = current character count

                PUTCCE:                         ;Copy EOF character
009D FE80               CPI     PUTCBS          ;Check for end of buffer
009F CAA900             JZ      PUTCWB          ;Yes, write out the buffer
00A2 361A               MVI     M,1AH           ;No, store EOF in buffer
00A4 3C                 INR     A               ;Update count
00A5 23                 INX     H               ;Update buffer pointer
00A6 C39D00             JMP     PUTCCE          ;Continue until end of buffer

                PUTCWB:                         ;Write buffer
00A9 AF                 XRA     A               ;Reset character count to 0
00AA 328000             STA     PUTCCC
00AD 110000             LXI     D,PUTCBF        ;DE -> buffer
00B0 0E1A               MVI     C,B$SETDMA      ;Set DMA address -> buffer
00B2 CD0500             CALL    BDOS
00B5 D1                 POP     D               ;Recover FCB address
00B6 0E15               MVI     C,B$WRITESEQ    ;Write sequential record
00B8 CD0500             CALL    BDOS
00BB B7                 ORA     A               ;Check if error
00BC C2C000             JNZ     PUTCX           ;Yes if A = NZ
00BF C9                 RET                     ;No, return to caller

                PUTCX:                          ;Error exit
00C0 3EFF               MVI     A,0FFH          ;Indicate such
00C2 C9                 RET

                PUTCGA:                         ;Return with HL -> next free char.
                                                ;and A = current char. count
00C3 3A8000             LDA     PUTCCC          ;Get current character count
00C6 5F                 MOV     E,A             ;Make word value in DE
00C7 1600               MVI     D,0
00C9 210000             LXI     H,PUTCBF        ;HL -> Base of buffer
00CC 19                 DAD     D               ;HL -> next free character
00CD C9                 RET
```

Figure 5-19. Write next character to sequential disk file (continued)

Purpose This function creates a new file of the specified name and type. You must first ensure that no file of the same name and type already exists on the same logical disk, either by trying to open the file (if this succeeds, the file already exists) or by unconditionally erasing the file.

In addition to creating the file and its associated file directory entry, this function also effectively opens the file so that it is ready for records to be written to it.

This function returns a normal directory code if the file creation has completed successfully or a value of 0FFH if there is insufficient disk or directory space.

Notes Under some circumstances, you may want to create a file that is slightly more "secure" than normal CP/M files. You can do this by using either lowercase letters or nongraphic ASCII characters such as ASCII NUL (00H) in the file name or type. Neither of these classes of characters can be generated from the keyboard; in the first case, the CCP changes all lowercase characters to uppercase, and in the second, it rejects names with odd characters in them. Thus, computer operators

cannot erase such a file because there is no way that they can create the same file name from the CCP.

The converse is also true; the only way that you can erase these files is by using a program that *can* set the exact file name into an FCB and then issue an Erase File function call.

Note that this function cannot accept an ambiguous file name in the FCB.

Figure 5-20 shows a subroutine that creates a file only after it has erased any existing files of the same name.

Function 23: Rename File

Function Code: C = 17H
Entry Parameters: DE = Address of file control block
Exit Parameters: A = Directory code

Example

```
0017 =          B$RENAME        EQU     23      ;Rename file
0005 =          BDOS            EQU     5       ;BDOS entry point

                FCB:                            ;File control block
0000 00                         DB      0       ;Search on default disk drive
0001 4F4C444E41                 DB      'OLDNAME'       ;File name
0009 545950                     DB      'TYP'   ;File type
000C 00000000                   DB      0,0,0,0
```

```
                ;CF
                ;Create file
                ;This subroutine creates a file. It erases any
                ;previous file before creating the new one.

                ;Entry parameters
                ;       DE -> File control block for new file

                ;Exit parameters
                ;       Carry clear if operation successful
                ;               (A = 0,1,2,3)
                ;       Carry set if error (A = 0FFH)

                ;Calling sequence
                ;       LXI     D,FCB
                ;       CALL    CF
                ;       JC      ERROR

0013 =          B$ERASE         EQU     19      ;Erase file
0016 =          B$CREATE        EQU     22      ;Create file
0005 =          BDOS            EQU     5       ;BDOS entry point

                CF:
0000 D5                 PUSH    D               ;Preserve FCB pointer
0001 0E13               MVI     C,B$ERASE       ;Erase any existing file
0003 CD0500             CALL    BDOS
0006 D1                 POP     D               ;Recover FCB pointer
0007 0E16               MVI     C,B$CREATE      ;Create (and open new file)
0009 CD0500             CALL    BDOS
000C FEFF               CPI     0FFH            ;Carry set if OK, clear if error
000E 3F                 CMC                     ;Complete to use Carry set if Error
000F C9                 RET
```

Figure 5-20. Create file request

```
0010 00                         DB      0       ;FCB + 16
0011 4E45574E41                 DB      'NEWNAME'       ;File name
0019 545950                     DB      'TYP'   ;File type
001C 00000000                   DB      0,0,0,0

0020 0E17               MVI     C,B$RENAME      ;Function code
0022 110000             LXI     D,FCB           ;DE -> file control block
0025 CD0500             CALL    BDOS            ;A = 00H if operation succesful
                                                ;A = 0FFH if file not found
```

Purpose This function renames an existing file name and type to a new name and type. It is unusual in that it uses a single FCB to store both the old file name and type (in the first 16 bytes) and the new file name and type (in the second 16 bytes).

This function returns a normal directory code if the file rename was completed successfully or a value of 0FFH if the old file name could not be found.

Notes The Rename File function only checks that the old file name and type exist; it makes no check to ensure that the new name and type combination does not already exist. Therefore, you should try to open the new file name and type. If you succeed, do not attempt the rename operation. CP/M will create more than one file of the same name and type, and you stand to lose the information in both files as you attempt to sort out the problem.

For security, you can also use lowercase letters and nongraphic characters in the file name and type, as described under the File Create function (code 22, 16H) above.

Never use ambiguous file names in a rename operation; it produces strange effects and may result in files being irreparably damaged. This function will change *all* occurrences of the old file name to the new name.

Figure 5-21 shows a subroutine that will accept an existing file name and type and a new name and type and rename the old to the new. It checks to make sure that the new file name does not already exist, returning an error code if it does.

Function 24: Get Active Disks (Login Vector)

Function Code: C = 18H
Entry Parameters: None
Exit Parameters: HL = Active disk map (login vector)

Example

```
0018 =          B$GETACTDSK     EQU     24      ;Get Active Disks
0005 =          BDOS            EQU     5       ;BDOS entry point

                                                ;Example of getting active
0000 0E18               MVI     C,B$GETACTDSK   ; disk function code
0002 CD0500             CALL    BDOS            ;HL = active disk bit map
                                                ;Bits are = 1 if disk active
                                                ;Bits 15 14 13 ... 2 1 0
                                                ;Disk  P  O  N ... C B A
```

Purpose This function returns a bit map, called the *login vector,* in register pair HL, indicating which logical disk drives have been selected since the last warm boot or

```
                        ;RF
                        ;Rename file
                        ;This subroutine renames a file.
                        ;It uses the BF (build FCB) subroutine shown in Figure 5.16

                        ;Entry parameters
                        ;       *** No case-folding of file names occurs ***
                        ;       HL -> old file name (00-byte terminated)
                        ;       DE -> new file name (00-byte terminated)

                        ;Exit parameters
                        ;       Carry clear if operation successful
                        ;                (A = 0,1,2,3)
                        ;       Carry set if error
                        ;               A = 0FEH if new file name already exists
                        ;               A = 0FFH if old file name does not exist

                        ;Calling sequence
                        ;       LXI     H,OLDNAME       ;HL -> old name
                        ;       LXI     D,NEWNAME       ;DE -> new name
                        ;       CALL    RF
                        ;       JC      ERROR

000F =          B$OPEN          EQU     15      ;Open file
0017 =          B$RENAME        EQU     23      ;Rename file
0005 =          BDOS            EQU     5       ;BDOS entry point

0000 000000000RFFCB:    DW      0,0,0,0,0,0,0,0 ;1 1/2 FCB's long
0010 0000000000         DW      0,0,0,0,0,0,0,0
0020 0000000000         DW      0,0,0,0,0,0,0,0
0030 000000             DW      0,0,0

                        RF:
0036 D5                 PUSH    D               ;Save new name pointer
0037 110000             LXI     D,RFFCB         ;Build old name FCB
                                                ;HL already -> old name
003A CD5D00             CALL    BF

003D E1                 POP     H               ;Recover new name pointer
003E 111000             LXI     D,RFFCB+16      ;Build new name in second part of file
0041 CD5D00             CALL    BF              ;control block

0044 111000             LXI     D,RFFCB+16      ;Experimentally try
0047 0E0F               MVI     C,B$OPEN        ;to open the new file
0049 CD0500             CALL    BDOS            ;to ensure it does
004C FEFF               CPI     0FFH            ;not already exist
004E 3EFE               MVI     A,0FEH          ;Assume error (flags unchanged)
0050 D8                 RC                      ;Carry set if A was 0,1,2,3

0051 110000             LXI     D,RFFCB         ;Rename the file
0054 0E17               MVI     C,B$RENAME
0056 CD0500             CALL    BDOS
0059 FEFF               CPI     0FFH            ;Carry set if OK, clear if error
005B 3F                 CMC                     ;Invert to use carry, set if error
005C C9                 RET

                        ;BF
                        ;Build file control block
                        ;This subroutine formats a 00H-byte terminated string
                        ;(presumed to be a file name) into an FCB, setting the
                        ;disk and the file name and type, and clearing the
                        ;remainder of the FCB to 0's.

                        ;Entry parameters
                        ;       DE -> file control block (36 bytes)
                        ;       HL -> file name string (00H-byte terminated)

                        ;Exit parameters
                        ;       The built file control block.

                        ;Calling sequence
                        ;       LXI     D,FCB
                        ;       LXI     H,FILENAME
                        ;       CALL    BF

                        BF:
005D C9                 RET                     ;Dummy subroutine : see Figure 5.16.
```

Figure 5-21. Rename file request

Reset Disk function (code 13, 0DH). The least significant bit of L corresponds to disk A, while the highest order bit in H maps disk P. The bit corresponding to the specific logical disk is set to 1 if the disk has been selected or to 0 if the disk is not currently on-line.

Logical disks can be selected programmatically through any file operation that sets the drive field to a nonzero value, through the Select Disk function (code 14, 0EH), or by the operator entering an "X:" command where "X" is equal to A, B, ..., P.

Notes This function is intended for programs that need to know which logical disks are currently active in the system—that is, those logical disks which have been selected.

Function 25: Get Current Default Disk

Function Code: C = 19H
Entry Parameters: None
Exit Parameters: A = Current disk
(0 = A, 1 = B, ..., F=P)

Example

```
0019 =          B$GETCURDSK     EQU     25      ;Get Current Disk
0005 =          BDOS            EQU     5       ;BDOS entry point

0000 0E19               MVI     C,B$GETCURDSK   ;Function code
0002 CD0500             CALL    BDOS            ;A = 0 if A:, 1 if B: ...
```

Purpose This function returns the current default disk set by the last Select Disk function call (code 14, 0EH) or by the operator entering the "X:" command (where "X" is A, B, ..., P) to the CCP.

Notes This function returns the current default disk in coded form. Register A = 0 if drive A is the current drive, 1 if drive B, and so on. If you need to convert this to the corresponding ASCII character, simply add 41H to register A.

Use this function when you convert a file name and type in an FCB to an ASCII string in order to display it. If the first byte of the FCB is 00H, the current default drive is to be used. You must therefore use this function to determine the logical disk letter for the default drive.

Function 26: Set DMA (Read/Write) Address

Function Code: C = 1AH
Entry Parameters: DE = DMA (read/write) address
Exit Parameters: None

Example

```
001A =          B$SETDMA        EQU     26      ;Set DMA Address
0005 =          BDOS            EQU     5       ;BDOS entry point
```

```
0000            SECBUFF:        DS      128     ;Sector buffer

0080 0E1A               MVI     C,B$SETDMA      ;Function code
0082 110000             LXI     D,SECBUFF       ;Pointer to buffer
0085 CD0500             CALL    BDOS
```

Purpose This function sets the BDOS's direct memory access (DMA) address to a new value. The name is an historic relic dating back to the Intel Development System on which CP/M was originally developed. This machine, by virtue of its hardware, could read data from a diskette directly into memory or write data to a diskette directly from memory. The name *DMA address* now applies to the address of the buffer to and from which data is transferred whenever a diskette Read, Write, or directory operation is performed.

Whenever CP/M first starts up (cold boot) or a warm boot or Reset Disk operation occurs, the DMA address is reset to its default value of 0080H.

Notes No function call can tell you the current value of the DMA address. All you can do is make a Set DMA function call to ensure that it is where you want it.

Once you have set the DMA address to the correct place for your program, it will remain set there until another Set DMA call, Reset Disk, or warm boot occurs.

The Read and Write Sequential and Random operations use the current setting of the DMA address, as do the directory operations Search First and Search Next.

Function 27: Get Allocation Vector

Function Code: C = 1BH
Entry Parameters: None
Exit Parameters: HL = Address of allocation vector

Example

```
001B =          B$GETALVEC      EQU     27      ;Get Allocation Vector Address
0005 =          BDOS            EQU     5       ;BDOS entry point

0000 0E1B               MVI     C,B$GETALVEC    ;Function code
0002 CD0500             CALL    BDOS            ;HL -> Base address of
                                                ;      allocation vector
```

Purpose This function returns the base, or starting, address of the allocation vector for the currently selected logical disk. This information, indicating which parts of the disk are assigned, is used by utility programs and the BDOS itself to determine how much unused space is on the logical disk, to locate an unused allocation block in order to extend a file, or to relinquish an allocation block when a file is deleted.

Notes Digital Research considers the actual layout of the allocation vector to be proprietary information.

Function 28: Set Logical Disk to Read-Only Status

Function Code: C = 1CH
Entry Parameters: None
Exit Parameters: None

Example

```
001C =          B$SETDSKRO      EQU     28      ;Set disk to Read Only
                                                ; function code
0005 =          BDOS            EQU     5       ;BDOS entry point

                                                ;Sets disk selected by prior
                                                ;Select disk function call
0000 0E1C               MVI     C,B$SETDSKRO    ;Function code
0002 CD0500             CALL    BDOS
```

Purpose This function logically sets the currently selected disk to a Read-Only state. Any attempts to execute a Write Sequential or Write Random function to the selected disk will be intercepted by the BDOS, and the following message will appear on the console:

```
BDOS Err on X: R/O
```

where X: is the selected disk.

Notes Once you have requested Read-Only status for the currently selected logical disk, this status will persist even if you proceed to select other logical disks. In fact, it will remain in force until the next warm boot or Reset Disk System function call.

Digital Research documentation refers to this function code as Disk Write Protect. The Read-Only description is used here because it corresponds to the error message produced if your program attempts to write on the disk.

Function 29: Get Read-Only Disks

Function Code: C = 1DH
Entry Parameters: None
Exit Parameters: HL = Read-Only disk map

Example

```
001D =          B$GETRODSKS     EQU     29      ;Get Read Only disks
0005 =          BDOS            EQU     5       ;BDOS entry point

0000 0E19               MVI     C,B$GETRODSKS   ;Function code
0002 CD0500             CALL    BDOS            ;HL = Read Only disk bit map
                                                ;Bits are = 1 if disk Read Only
                                                ;Bits 15 14 13 ... 2 1 0
                                                ;Disk  P  O  N ... C B A
```

Purpose This function returns a bit map in registers H and L showing which logical disks in the system have been set to Read-Only status, either by the Set Logical

Disk to Read-Only function call (code 28, 1CH), or by the BDOS itself, because it detected that a diskette had been changed.

The least significant bit of L corresponds to logical disk A, while the most significant bit of H corresponds to disk P. The bit corresponding to the specific logical disk is set to 1 if the disk has been set to Read-Only status.

Function 30: Set File Attributes

Function Code: C = 1EH
Entry Parameters: DE = Address of FCB
Exit Parameters: A = Directory code

Example

```
001E =          B$SETFAT        EQU     30      ;Set File Attribute
0005 =          BDOS            EQU     5       ;BDOS entry point

                FCB:                            ;File control block
0000 00         FCB$DISK:       DB      0       ;Search on default disk drive
0001 46494C454EFCB$NAME:        DB      'FILENAME'      ;File name
0009 D4         FCB$TYP:        DB      'T'+80H         ;Type with R/O
                                                ; attribute
000A 5950                       DB      'YP'
000C 0000000000                 DW      0,0,0,0,0,0,0,0,0,0,0

0022 0E1E               MVI     C,B$SETFAT      ;Function code
0024 110000             LXI     D,FCB           ;DE -> file control block
                                                ;MS bits set in file name/type
0027 CD0500             CALL    BDOS            ;A = 0FFH if file not found
```

Purpose

This function sets the bits that describe attributes of a file in the relevant directory entries for the specified file. Each file can be assigned up to 11 file attributes. Of these 11, two have predefined meanings, four others are available for you to use, and the remaining five are reserved for future use by CP/M.

Each attribute consists of a single bit. The most significant bit of each byte of the file name and type is used to store the attributes. The file attributes are known by a code consisting of the letter "f" (for file name) or "t" (for file type), followed by the number of the character position and a single quotation mark. For example, the Read-Only attribute is t1'.

The significance of the attributes is as follows:

- f1' to f4' Available for you to use
- f5' to f8' Reserved for future CP/M use
- t1' Read-Only File attribute
- t2' System File attribute
- t3' Reserved for future CP/M use

Attributes are set by presenting this function with an FCB in which the unambiguous file name has been preset with the most significant bits set appropriately. This function then searches the directory for a match and changes the matched entries to contain the attributes which have been set in the FCB.

The BDOS will intercept any attempt to write on a file that has the Read-Only attribute set. The DIR command in the CCP does not display any file with System status.

Notes

You can use the four attributes available to you to set up a file security system, or perhaps to flag certain files that must be backed up to other disks. The Search First and Search Next functions allow you to view the complete file directory entry, so your programs can test the attributes easily.

The example subroutines in Figures 5-22 and 5-23 show how to set file attributes (SFA) and get file attributes (GFA), respectively. They both use a bit map in which the most significant 11 bits of the HL register pair are used to indicate the corresponding high bits of the 11 characters of the file name/type combination. You will also see some equates that have been declared to make it easier to manipulate the attributes in this bit map.

```
                        ;SFA
                        ;Set file attributes
                        ;This subroutine takes a compressed bit map of all the
                        ;file attribute bits, expands them into an existing
                        ;file control block and then requests CP/M to set
                        ;the attributes in the file directory.

                        ;Entry parameters
                        ;       DE -> file control block
                        ;       HL = bit map. Only the most significant 11
                        ;            bits are used. These correspond directly
                        ;            with the possible attribute bytes.

                        ;Exit parameters
                        ;       Carry clear if operation successful (A = 0,1,2,3)
                        ;       Carry set if error (A = 0FFH)

                        ;Calling sequence
                        ;       LXI     D,FCB
                        ;       LXI     H,0000$0000$1100$0000B  ;Bit Map
                        ;       CALL    SFA
                        ;       JC      ERROR

                                                        ;File Attribute Equates

8000 =          FA$F1   EQU     1000$0000$0000$0000B    ;F1' - F4'
4000 =          FA$F2   EQU     0100$0000$0000$0000B    ;Available for use by
2000 =          FA$F3   EQU     0010$0000$0000$0000B    ; application programs
1000 =          FA$F4   EQU     0001$0000$0000$0000B

0800 =          FA$F5   EQU     0000$1000$0000$0000B    ;F5' - F8'
0400 =          FA$F6   EQU     0000$0100$0000$0000B    ;Reserved for CP/M
0200 =          FA$F7   EQU     0000$0010$0000$0000B
0100 =          FA$F8   EQU     0000$0001$0000$0000B

0080 =          FA$T1   EQU     0000$0000$1000$0000B    ;T1' -- read/only file
0080 =          FA$RO   EQU     FA$T1
0040 =          FA$T2   EQU     0000$0000$0100$0000B    ;T2' -- system files
0040 =          FA$SYS  EQU     FA$T2
0020 =          FA$T3   EQU     0000$0000$0010$0000B    ;T3' -- reserved for CP/M

001E =          B$SETFAT        EQU     30      ;Set file attributes
0005 =          BDOS            EQU     5       ;BDOS entry point
```

Figure 5-22. Set file attributes

```
                SFA:
0000 D5                 PUSH    D               ;Save FCB pointer
0001 13                 INX     D               ;HL -> 1st character of file name
0002 0E0B               MVI     C,8+3           ;Loop count for file name and type

                SFAL:                           ;Main processing loop
0004 AF                 XRA     A               ;Clear carry and A
0005 29                 DAD     H               ;Shift next MS bit into carry
0006 CE00               ACI     0               ;A = 0 or 1 depending on carry
0008 0F                 RRC                     ;Rotate LS bit of A into MS bit
0009 47                 MOV     B,A             ;Save result (00H or 80H)
000A EB                 XCHG                    ;HL -> FCB character
000B 7E                 MOV     A,M             ;Get FCB character
000C E67F               ANI     7FH             ;Isolate all but attribute bit
000E B0                 ORA     B               ;Set attribute with result
000F 77                 MOV     M,A             ;and store back into FCB
0010 EB                 XCHG                    ;DE -> FCB, HL = remaining bit map
0011 13                 INX     D               ;DE -> next character in FCB
0012 0D                 DCR     C               ;Downdate character count
0013 C20400             JNZ     SFAL            ;Loop back for next character
0016 0E1E               MVI     C,B$SETFAT      ;Set file attribute function code
0018 D1                 POP     D               ;Recover FCB pointer
0019 CD0500             CALL    BDOS
001C FEFF               CPI     0FFH            ;Carry set if OK, clear if error
001E 3F                 CMC                     ;Invert to use carry set if error
001F C9                 RET
```

Figure 5-22. Set file attributes (continued)

```
                ;GFA
                ;Get file attributes
                ;This subroutine finds the appropriate file using a
                ;search for First Name Match function rather than opening
                ;the file. It then builds a bit map of the file attribute
                ;bits in the file name and type. This bit map is then ANDed
                ;with the input bit map, and the result is returned in the
                ;zero flag. The actual bit map built is also returned in case
                ;more complex check is required.

                ;>>>     Note: This subroutine changes the CP/M DMA address.

                ;Entry parameters
                ;       DE -> File control block
                ;       HL = Bit map mask to be ANDed with attribute
                ;            results

                ;Exit parameters
                ;       Carry clear, operation successful
                ;               Nonzero status set to result of AND between
                ;               input mask and attribute bits set.
                ;               HL = Unmasked attribute bytes set.
                ;       Carry set, file could not be found

001A =          B$SETDMA        EQU     26      ;Set DMA address
0011 =          B$SEARCHF       EQU     17      ;Search for first entry to match
0005 =          BDOS            EQU     5       ;BDOS entry point
0080 =          GFADMA          EQU     80H     ;Default DMA address

                ;Calling sequence
                ;       LXI     D,FCB
                ;       LXI     H,0000$0000$1100$0000B  ;Bit map
                ;       CALL    GFA
                ;       JC      ERROR

                                                ;File attribute equates

8000 =          FA$F1   EQU     1000$0000$0000$0000B    ;F1' - F5'
4000 =          FA$F2   EQU     0100$0000$0000$0000B    ;Available for use by
```

Figure 5-23. Get file attributes

```
2000 =          FA$F3   EQU     0010$0000$0000$0000B    ;Application programs
1000 =          FA$F4   EQU     0001$0000$0000$0000B

0800 =          FA$F5   EQU     0000$1000$0000$0000B    ;F6' - F8'
0400 =          FA$F6   EQU     0000$0100$0000$0000B    ;Reserved for CP/M
0200 =          FA$F7   EQU     0000$0010$0000$0000B
0100 =          FA$F8   EQU     0000$0001$0000$0000B

0080 =          FA$T1   EQU     0000$0000$1000$0000B    ;T1' -- read/only file
0080 =          FA$RO   EQU     FA$T1
0040 =          FA$T2   EQU     0000$0000$0100$0000B    ;T2' -- system files
0040 =          FA$SYS  EQU     FA$T2
0020 =          FA$T3   EQU     0000$0000$0010$0000B    ;T3' -- reserved for CP/M

                GFA:
0000 E5                 PUSH    H               ;Save AND-mask
0001 D5                 PUSH    D               ;Save FCB pointer
0002 0E1A               MVI     C,B$SETDMA      ;Set DMA to default address
0004 118000             LXI     D,GFADMA        ;DE -> DMA address
0007 CD0500             CALL    BDOS

000A D1                 POP     D               ;Recover FCB pointer
000B 0E11               MVI     C,B$SEARCHF     ;Search for match with name
000D CD0500             CALL    BDOS
0010 FEFF               CPI     OFFH            ;Carry set if OK, clear if error
0012 3F                 CMC                     ;Invert to use set carry if error
0013 DA4100             JC      GFAX            ;Return if error
                                                ;Multiply by 32 to get offset into DMA buffer
0016 87                 ADD     A               ;* 2
0017 87                 ADD     A               ;* 4
0018 87                 ADD     A               ;* 8
0019 87                 ADD     A               ;* 16
001A 87                 ADD     A               ;* 32
001B 5F                 MOV     E,A             ;Make into a word value
001C 1600               MVI     D,0
001E 218000             LXI     H,GFADMA        ;HL -> DMA address
0021 19                 DAD     D               ;HL -> Directory entry in DMA buffer
0022 23                 INX     H               ;HL -> 1st character of file name
0023 EB                 XCHG                    ;DE -> 1st character of file name

0024 0E0B               MVI     C,8+3           ;Count of characters in file name and type
0026 210000             LXI     H,0             ;Clear bit map

                GFAL:                           ;Main loop
0029 1A                 LDAX    D               ;Get next character of file name
002A E680               ANI     80H             ;Isolate attribute bit
002C 07                 RLC                     ;Move MS bit into LS bit
002D B5                 ORA     L               ;OR in any previously set bits
002E 6F                 MOV     L,A             ;Save result
002F 29                 DAD     H               ;Shift HL left one bit for next time
0030 13                 INX     D               ;DE -> next character in file name, type
0031 0D                 DCR     C               ;Downdate count
0032 C22900             JNZ     GFAL            ;Go back for next character

0035 29                 DAD     H               ;Left justify attribute bits in HL
0036 29                 DAD     H               ;MS attribute bit will already be in
0037 29                 DAD     H               ;bit 11 of HL, so only 4 shifts are
0038 29                 DAD     H               ;necessary

0039 D1                 POP     D               ;Recover AND-mask
003A 7A                 MOV     A,D             ;Get MS byte of mask
003B A4                 ANA     H               ;AND with MS byte of result
003C 47                 MOV     B,A             ;Save interim result
003D 7B                 MOV     A,E             ;Get LS byte of mask
003E A5                 ANA     L               ;AND with LS byte of result
003F B0                 ORA     B               ;Combine two results to set Z flag

0040 C9                 RET

                GFAX:                           ;Error exit
0041 E1                 POP     H               ;Balance stack
0042 C9                 RET
```

Figure 5-23. Get file attributes (continued)

Function 31: Get Disk Parameter Block Address

Function Code: C = 1FH
Entry Parameters: None
Exit Parameters: HL = Address of DPB

Example

```
001F =          B$GETDPB        EQU     31      ;Get Disk Parameter Block
                                                ; Address
0005 =          BDOS            EQU     5       ;BDOS entry point

                                                ;Returns DPB address of
                                                ; logical disk previously
                                                ; selected with a Select
                                                ; Disk function.
0000 0E1F               MVI     C,B$GETDPB      ;Function code
0002 CD0500             CALL    BDOS            ;HL -> Base address of current
                                                ; disk's parameter block
```

Purpose

This function returns the address of the disk parameter block (DPB) for the last selected logical disk. The DPB, explained in Chapter 3, describes the physical characteristics of a specific logical disk—information mainly of interest for system utility programs.

Notes

The subroutines shown in Figure 5-24 deal with two major problems. First, given a track and sector number, what allocation block will they fall into? Conversely, given an allocation block, what is its starting track and sector?

These subroutines are normally used by system utilities. They first get the DPB address using this BDOS function. Then they switch to using direct BIOS calls to perform their other functions, such as selecting disks, tracks, and sectors and reading and writing the disk.

The first subroutine, GTAS (Get Track and Sector), in Figure 5-24, takes an allocation block number and converts it to give you the starting track and sector number. GMTAS (Get Maximum Track and Sector) returns the maximum track and sector number for the specified disk. GDTAS (Get Directory Track and Sector) tells you not only the starting track and sector for the file directory, but also the number of 128-byte sectors in the directory.

Note that whenever a track number is used as an entry or an exit parameter, it is an absolute track number. That is, the number of reserved tracks on the disk before the directory has already been added to it.

GNTAS (Get Next Track and Sector) helps you read sectors sequentially. It adds 1 to the sector number, and when you reach the end of a track, updates the track number by 1 and resets the sector number to 1.

GAB (Get Allocation Block) is the converse of GTAS (Get Track and Sector). It returns the allocation block number, given a track and sector.

Finally, Figure 5-24 includes several useful 16-bit subroutines to divide the HL register pair by DE (DIVHL), to multiply HL by DE (MULHL), to subtract DE from HL (SUBHL —this can also be used as a 16-bit compare), and to shift HL right one bit (SHLR). The divide and multiply subroutines are somewhat primitive, using iterative subtraction and addition, respectively. Nevertheless, they do perform their role as supporting subroutines.

```
                ;Useful subroutines for accessing the data in the
                ;disk parameter block

000E =          B$SELDSK        EQU     14      ;Select Disk function code
001F =          B$GETDPB        EQU     31      ;Get DPB address
0005 =          BDOS            EQU     5       ;BDOS entry point

                ;It makes for easier, more compact code to copy the
                ;specific disk parameter block into local variables
                ;while manipulating the information.
                ;Here are those variables --

                DPB:                            ;Disk parameter block
0000 0000       DPBSPT: DW      0               ;128-byte sectors per track
0002 00         DPBBS:  DB      0               ;Block shift
0003 00         DPBBM:  DB      0               ;Block mask
0004 00         DPBEM:  DB      0               ;Extent mask
0005 0000       DPBMAB: DW      0               ;Maximum allocation block number
0007 0000       DPBNOD: DW      0               ;Number of directory entries - 1
0009 0000       DPBDAB: DW      0               ;Directory allocation blocks
000B 0000       DPBCBS: DW      0               ;Check buffer size
000D 0000       DPBTBD: DW      0               ;Tracks before directory (reserved tracks)

000F =          DPBSZ   EQU     $-DPB           ;Disk parameter block size

                ;GETDPB
                ;Gets disk parameter block
                ;This subroutine copies the DPB for the specified
                ;logical disk into the local DPB variables above.

                ;Entry parameters
                ;       A = Logical disk number (A: = 0, B: = 1...)

                ;Exit parameters
                ;       Local variables contain DPB

                GETDPB:
000F 5F                 MOV     E,A             ;Get disk code for select disk
0010 0E0E               MVI     C,B$SELDSK      ;Select the disk
0012 CD0500             CALL    BDOS
0015 0E1F               MVI     C,B$GETDPB      ;Get the disk parameter base address
0017 CD0500             CALL    BDOS            ;HL -> DPB
001A 0E0F               MVI     C,DPBSZ         ;Set count
001C 110000             LXI     D,DPB           ;Get base address of local variables

                GDPBL:                          ;Copy DPB into local variables
001F 7E                 MOV     A,M             ;Get byte from DPB
0020 12                 STAX    D               ;Store into local variable
0021 13                 INX     D               ;Update local variable pointer
0022 23                 INX     H               ;Update DPB pointer
0023 0D                 DCR     C               ;Downdate count
0024 C21F00             JNZ     GDPBL           ;Loop back for next byte
0027 C9                 RET

                ;GTAS
                ;Get track and sector (given allocation block number)

                ;This subroutine converts an allocation block into a
                ;track and sector number -- note that this is based on
                ;128-byte sectors.

                ;>>>>>  Note: You must call GETDPB before
                ;>>>>>        you call this subroutine

                ;Entry parameters
                ;       HL = allocation block number

                ;Exit parameters
                ;       HL = track number
                ;       DE = sector number

                ;Method :
                ;In mathematical terms, the track can be derived from:
                ;Trk = ((allocation block * sec. per all. block) / sec. per trk)
                ;       + tracks before directory
```

Figure 5-24. Accessing disk parameter block data

```
                ;The sector is derived from:
                ;Sec = ((allocation block * sec. per all. block) modulo/
                ;       sec. per trk) + 1

                GTAS:
0028 3A0200             LDA     DPBBS           ;Get block shift -- this will be 3 to
                                                ;7 depending on allocation block size
                                                ;It will be used as a count for shifting

                GTASS:
002B 29                 DAD     H               ;Shift allocation block left one place
002C 3D                 DCR     A               ;Decrement block shift count
002D C22B00             JNZ     GTASS           ;More shifts required
0030 EB                 XCHG                    ;DE = all. block * sec. per block
                                                ;i.e. DE = total number of sectors
0031 2A0000             LHLD    DPBSPT          ;Get sectors per track
0034 EB                 XCHG                    ;HL = sec. per trk, DE = tot. no. of sec.
0035 CD8F00             CALL    DIVHL           ;BC = HL/DE, HL = remainder
                                                ;BC = track, HL = sector
0038 23                 INX     H               ;Sector numbering starts from 1
0039 EB                 XCHG                    ;DE = sector, HL = track
003A 2A0D00             LHLD    DPBTBD          ;Tracks before directory
003D 09                 DAD     B               ;DE = sector, HL = absolute track
003E C9                 RET

                ;GMTAS
                ;Get maximum track and sector

                ;This is just a call to GTAS with the maximum
                ;allocation block as the input parameter

                ;>>>>>  Note: You must call GETDPB before
                ;>>>>>        you call this subroutine

                ;Entry parameters: none

                ;Exit parameters:
                ;       HL = maximum track number
                ;       DE = maximum sector
                GMTAS:
003F 2A0500             LHLD    DPBMAB          ;Get maximum allocation block
0042 C32800             JMP     GTAS            ;Return from GTAS with parameters in HL and DE

                ;GDTAS
                ;Get directory track and sector

                ;This returns the START track and sector for the
                ;file directory, along with the number of sectors
                ;in the directory.

                ;>>>>>  Note: You must call GETDPB before
                ;>>>>>        you call this subroutine

                ;Entry parameters: none

                ;Exit parameters:
                ;       BC = number of sectors in directory
                ;       DE = directory start sector
                ;       HL = directory start track

                GDTAS:
0045 2A0700             LHLD    DPBNOD          ;Get number of directory entries - 1
0048 23                 INX     H               ;Make true number of entries
                                                ;Each entry is 32 bytes long, so to
                                                ;convert to 128 byte sectors, divide by 4
0049 CDD000             CALL    SHLR            ;/ 2 (by shifting HL right one bit)
004C CDD000             CALL    SHLR            ;/ 4
004F E5                 PUSH    H               ;Save number of sectors
0050 210000             LXI     H,0             ;Directory starts in allocation block 0
0053 CD2800             CALL    GTAS            ;HL = track, DE = sector
0056 C1                 POP     B               ;Recover number of sectors
0057 C9                 RET
```

Figure 5-24. (Continued)

```
                        ;GNTAS
                        ;Get NEXT track and sector

                        ;This subroutine updates the input track and sector
                        ;by one, incrementing the track and resetting the
                        ;sector number as required.

                        ;>>>>>  Note: You must call GETDPB before
                        ;>>>>>        you call this subroutine

                        ; Note: you must check for end of disk by comparing
                        ;       the track number returned by this subroutine
                        ;       to that returned by by GMTAS + 1. When
                        ;       equality occurs, the end of disk has been reached.

                        ;Entry parameters
                        ;       HL = current track number
                        ;       DE = current sector number

                        ;Exit parameters
                        ;       HL = updated track number
                        ;       DE = updated sector number

                        GNTAS:
0058 E5                         PUSH    H               ;Save track
0059 13                         INX     D               ;Update sector
005A 2A0000                     LHLD    DPBSPT          ;Get sectors per track
005D CDC900                     CALL    SUBHL           ;HL = HL - DE
0060 E1                         POP     H               ;Recover current track
0061 D0                         RNC                     ;Return if updated sector <= sec. per trk.
0062 23                         INX     H               ;Update track if upd. sec > sec. per trk.
0063 110100                     LXI     D,1             ;Reset sector to 1
0066 C9                         RET

                        ;GAB
                        ;Get allocation block

                        ;This subroutine returns an allocation block number
                        ;given a specific track and sector. It also returns
                        ;the offset down the allocation block at which the
                        ;sector will be found. This offset is in units of
                        ;128-byte sectors.

                        ;>>>>>   Note: You must call GETDPB before
                        ;>>>>>         you call this subroutine

                        ;Entry parameters
                        ;       HL = track number
                        ;       DE = sector number

                        ;Exit parameters
                        ;       HL = allocation block number

                        ;Method
                        ;The allocation block is formed from:
                        ;AB = (sector + ((track - tracks before directory)
                        ;       * sectors per track)) / log2 (sectors per all. block)

                        ;The sector offset within allocation block is formed from:
                        ;Offset = (sector + ((track - tracks before directory)
                        ;       * sectors per track)) / AND (sectors per all. block - 1)

                        GAB:
0067 D5                         PUSH    D               ;Save sector
0068 EB                         XCHG                    ;DE = track
0069 2A0D00                     LHLD    DPBTBD          ;Get no. of tracks before directory
006C EB                         XCHG                    ;DE = no. of tracks before dir. HL = track
006D CDC900                     CALL    SUBHL           ;HL = HL - DE
                                                        ;HL = relative track within logical disk
0070 EB                         XCHG                    ;DE = relative track
0071 2A0000                     LHLD    DPBSPT          ;Get sectors per track
0074 CDA400                     CALL    MULHL           ;HL = HL * DE
                                                        ;HL = number of sectors
0077 EB                         XCHG                    ;DE = number of sectors
```

Figure 5-24. (Continued)

```
0078 E1                 POP     H               ;Recover sector
0079 2B                 DCX     H               ;Make relative to 0
007A 19                 DAD     D               ;HL = relative sector
007B 3A0300             LDA     DPBBM           ;Get block mask
007E 47                 MOV     B,A             ;Ready for AND operation
007F 7D                 MOV     A,L             ;Get LS byte of relative sector
0080 A0                 ANA     B               ;AND with block mask
0081 F5                 PUSH    PSW             ;A = sector displacement
0082 3A0200             LDA     DPBBS           ;Get block shift
0085 4F                 MOV     C,A             ;Make into counter

                GABS:                           ;Shift loop
0086 CDD000             CALL    SHLR            ;HL shifted right (divided by 2)
0089 0D                 DCR     C               ;Count down
008A C28600             JNZ     GABS            ;Shift again if necessary
008D F1                 POP     PSW             ;Recover offset
008E C9                 RET

                ;Utility subroutines
                ;These perform 16-bit arithmetic on the HL register pair.

                ;DIVHL
                ;Divides HL by DE using an iterative subtract.
                ;In practice, it uses an iterative ADD of the complemented divisor.

                ;Entry parameters
                ;       HL = dividend
                ;       DE = divisor

                ;Exit parameters
                ;       BC = quotient
                ;       HL = remainder

                DIVHL:
008F D5                 PUSH    D               ;Save divisor
                                                ;Note : 2's complement is formed by
                                                ;inverting all bits and adding 1.
0090 7B                 MOV     A,E             ;Complement divisor (for iterative
0091 2F                 CMA                     ;ADD later on)
0092 5F                 MOV     E,A
0093 7A                 MOV     A,D             ;Get MS byte
0094 2F                 CMA                     ;Complement it
0095 57                 MOV     D,A
0096 13                 INX     D               ;Make 2's complement
                                                ;Now, subtract negative divisor until
                                                ;dividend goes negative, counting the number
                                                ;of times the subtract occurs
0097 010000             LXI     B,0             ;Initialize quotient
                DIVHLS:                         ;Subtract loop
009A 03                 INX     B               ;Add 1 to quotient
009B 19                 DAD     D               ;"Subtract" divisor
009C DA9A00             JC      DIVHLS          ;Dividend not yet negative
                                                ;Dividend now negative, quotient 1 too large
009F 0B                 DCX     B               ;Correct quotient
                                                ;Compute correct remainder
00A0 EB                 XCHG                    ;DE = remainder - divisor
00A1 E1                 POP     H               ;Recover positive divisor
00A2 19                 DAD     D               ;HL = remainder
00A3 C9                 RET                     ;BC = quotient, HL = remainder

                ;MULHL
                ;Multiply HL * DE using iterative ADD.

                ;Entry parameters
                ;       HL = multiplicand
                ;       DE = multiplier

                ;Exit parameters
                ;       HL = product
                ;       DE = multiplier

                MULHL:
00A4 C5                 PUSH    B               ;Save user register
                                                ;Check if either multiplicand
                                                ; or multiplier is 0
```

Figure 5-24. (Continued)

```
00A5 7C              MOV     A,H
00A6 B5              ORA     L
00A7 CAC400          JZ      MULHLZ          ;Yes, fake product
00AA 7A              MOV     A,D
00AB B3              ORA     E
00AC CAC400          JZ      MULHLZ          ;Yes, fake product

                                             ;This routine will be faster if
                                             ; the smaller value is in DE
00AF 7A              MOV     A,D             ;Get MS byte of current DE value
00B0 BC              CMP     H               ;Check which is smaller
00B1 DAB500          JC      MULHLN          ;C set if D < H, so no exchange
00B4 EB              XCHG

                MULHLN:
00B5 42              MOV     B,D             ;BC = multiplier
00B6 4B              MOV     C,E
00B7 54              MOV     D,H             ;DE = HL = multiplicand
00B8 5D              MOV     E,L
00B9 0B              DCX     B               ;Adjust count as
                                             ;1 * multiplicand = multiplicand

                MULHLA:                      ;ADD loop
00BA 78              MOV     A,B             ;Check if all iterations completed
00BB B1              ORA     C
00BC CAC700          JZ      MULHLX          ;Yes, exit
00BF 19              DAD     D               ;HL = multiplicand + multiplicand
00C0 0B              DCX     B               ;Countdown on multiplier - 1
00C1 C3BA00          JMP     MULHLA          ;Loop back until all ADDs done

                MULHLZ:
00C4 210000          LXI     H,0             ;Fake product as either multiplicand
                                             ; or multiplier is 0

                MULHLX:
00C7 C1              POP     B               ;Recover user register
00C8 C9              RET

                ;SUBHL
                ;Subtract HL - DE

                ;Entry parameters
                ;       HL = subtrahend
                ;       DE = subtractor

                ;Exit parameters
                ;       HL = difference

                SUBHL:
00C9 7D              MOV     A,L             ;Get LS byte
00CA 93              SUB     E               ;Subtract without regard to carry
00CB 6F              MOV     L,A             ;Put back into difference
00CC 7C              MOV     A,H             ;Get MS byte
00CD 9A              SBB     D               ;Subtract including carry
00CE 67              MOV     H,A             ;Move back into difference
00CF C9              RET

                ;SHLR
                ;Shift HL right one place (dividing HL by 2)

                ;Entry parameters
                ;       HL = value to be shifted

                ;Exit parameters
                ;       HL = value/2

                SHLR:
00D0 B7              ORA     A               ;Clear carry
00D1 7C              MOV     A,H             ;Get MS byte
00D2 1F              RAR                     ;Bit 7 set from previous carry,
                                             ; bit 0 goes into carry
00D3 67              MOV     H,A             ;Put shift MS byte back
00D4 7D              MOV     A,L             ;Get LS byte
00D5 1F              RAR                     ;Bit 7 = bit 0 of MS byte
00D6 6F              MOV     L,A             ;Put back into result
00D7 C9              RET
```

Figure 5-24. (Continued)

Function 32: Set/Get User Number

Function Code: C = 20H
Entry Parameters: E = 0FFH to get user number, or
E = 0 to 15 to set user number
Exit Parameters: A = Current user number if E was 0FFH

Example

```
0020 =          B$SETGETUN      EQU     32      ;Set/Get User Number
0005 =          BDOS            EQU     5       ;BDOS entry point

                                                ;To set user number
0000 0E20               MVI     C,B$SETGETUN    ;Function code
0002 1E0F               MVI     E,15            ;Required user number
0004 CD0500             CALL    BDOS            ;To get user number
0007 0E20               MVI     C,B$SETGETUN    ;Function code
0009 1EFF               MVI     E,0FFH          ;Indicate request to GET
000B CD0500             CALL    BDOS            ;A = Current user no. (0 -- 15)
```

Purpose

This subroutine either sets or gets the current user number. The current user number determines which file directory entries are matched during all disk file operations.

When you call this function, the contents of the E register specify what action is to be taken. If E = 0FFH, then the function will return the current user number in the A register. If you set E to a number in the range 0 to 15 (that is, a valid user number), the function will set the current user number to this value.

Notes

You can use this function to share files with other users. You can locate a file by attempting to open a file and switching through all of the user numbers. Or you can share a file in another user number by setting to that number, operating on the file, and then reverting back to the original user number.

If you do change the current user number, make provisions in your program to return to the original number before your program terminates. It is disconcerting for computer operators to find that they are in a different user number after a program. Files can easily be damaged or accidentally erased this way.

Function 33: Read Random

Function Code: C = 21H
Entry Parameters: DE = Address of FCB
Exit Parameters: A = Return code

Example

```
0021 =          B$READRAN       EQU     33      ;Read Random
0005 =          BDOS            EQU     5       ;BDOS entry point

                FCB:                            ;File control block
0000 00         FCB$DISK:       DB      0       ;Search on default disk drive
0001 46494C454EFCB$NAME:        DB      'FILENAME'      ;File name
0009 545950     FCB$TYP:        DB      'TYP'   ;File type
```

```
000C 00          FCB$EXTENT:     DB      0       ;Extent
000D 0000        FCB$RESV:       DB      0,0     ;Reserved for CP/M
000F 00          FCB$RECUSED:    DB      0       ;Records used in this extent
0010 0000000000FCB$ABUSED:       DB      0,0,0,0,0,0,0,0 ;Allocation blocks used
0018 0000000000                  DB      0,0,0,0,0,0,0,0
0020 00          FCB$SEQREC:     DB      0       ;Sequential rec. to read/write
0021 0000        FCB$RANREC:     DW      0       ;Random rec. to read/write
0023 00          FCB$RANRECO:    DB      0       ;Random rec. overflow byte (MS)

0024 D204        RANRECNO:       DW      1234    ;Example random record number

                                                 ;Record will be read into
                                                 ; address set by prior
                                                 ; SETDMA call
0026 2A2400              LHLD    RANRECNO        ;Get random record number
0029 222100              SHLD    FCB$RANREC      ;Set up file control block
002C 0E21                MVI     C,B$READRAN     ;Function code
002E 110000              LXI     D,FCB           ;DE -> file control block
0031 CD0500              CALL    BDOS            ;A = 00 if operation successful
                                                 ;A  = nonzero if no data in
                                                 ; file specifically:
                                                 ;A = 01 -- attempt to read
                                                 ;        unwritten record
                                                 ;    03 -- CP/M could not
                                                 ;        close current extent
                                                 ;    04 -- attempt to read
                                                 ;        unwritten extent
                                                 ;    06 -- attempt to read
                                                 ;        beyond end of disk
```

Purpose This function reads a specific CP/M record (128 bytes) from a random file—that is, a file in which records can be accessed directly. It assumes that you have already opened the file, set the DMA address using the BDOS Set DMA function, and set the specific record to be read into the random record number in the FCB. This function computes the extent of the specified record number and attempts to open it and read the correct CP/M record into the DMA address.

The random record number in the FCB is three bytes long (at relative bytes 33, 34, and 35). Byte 33 is the least significant byte, 34 is the middle byte, and 35 the most significant. CP/M uses only the most significant byte (35) for computing the overall file size (function 35). You must set this byte to 0 when setting up the FCB. Bytes 33 and 34 are used together for the Read Random, so you can access from record 0 to 65535 (a maximum file size of 8,388,480 bytes).

This function returns with A set to 0 to indicate that the operation has been completed successfully, or A set to a nonzero value if an error has occurred. The error codes are as follows:

A = 01 (attempt to read unwritten record)

A = 03 (CP/M could not close current extent)

A = 04 (attempt to read unwritten extent)

A = 06 (attempt to read beyond end of disk)

Unlike the Read Sequential BDOS function (code 20, 14H), which updates the current (sequential) record number in the FCB, the Read Random function leaves the record number unchanged, so that a subsequent Write Random will replace the record just read.

You can follow a Read Random with a Write Sequential (code 21, 15H). This

will rewrite the record just read, but will then update the sequential record number. Or you may choose to use a Read Sequential after the Read Random. In this case, the same record will be reread and the sequential record number will be incremented. In short, the file can be sequentially read or written once the Read Random has been used to position to the required place in the file.

Notes

To use the Read Random function, you must first open the *base extent* of the file, that is, extent 0. Even though there may be no actual data records in this extent, opening permits the file to be processed correctly.

One problem that is not immediately obvious with random files is that they can easily be created with gaps in the file. If you were to create the file with record number 0 and record number 5000, there would be no intervening file extents. Should you attempt to read or copy the file sequentially, even using CP/M's file copy utility, only the first extent (and in this case, record 0) would get copied. A Read Sequential function would return an "end of file" error after reading record 0. You must therefore be conscious of the type of the file that you try and read.

See Figure 5-26 for an example subroutine that performs Random File Reads and Writes. It reads or writes records of sizes other than 128 bytes, where necessary reading or writing several CP/M records, prereading them into its own buffer when the record being written occupies only part of a CP/M record. It also contains subroutines to produce a 32-bit product from multiplying HL by DE (MLDL—Multiply double length) and a right bit shift for DE, HL (SDLR—Shift double length right).

Function 34: Write Random

Function Code: C = 22H
Entry Parameters: DE = Address of file control block
Exit Parameters: A = Return code

Example

```
0022 =          B$WRITERAN      EQU     34      ;Write Random
0005 =          BDOS            EQU     5       ;BDOS entry point

                FCB:                            ;File control block
0000 00         FCB$DISK:       DB      0       ;Search on default disk drive
0001 46494C454EFCB$NAME:        DB      'FILENAME'      ;File name
0009 545950     FCB$TYP:        DB      'TYP'   ;File type
000C 00         FCB$EXTENT:     DB      0       ;Extent
000D 0000       FCB$RESV:       DB      0,0     ;Reserved for CP/M
000F 00         FCB$RECUSED:    DB      0       ;Records used in this extent
0010 0000000000FCB$ABUSED:      DB      0,0,0,0,0,0,0,0 ;Allocation blocks used
0018 0000000000                 DB      0,0,0,0,0,0,0,0
0020 00         FCB$SEQREC:     DB      0       ;Sequential rec. to read/write
0021 0000       FCB$RANREC:     DW      0       ;Random rec. to read/write
0023 00         FCB$RANRECO:    DB      0       ;Random rec. overflow byte (MS)

0024 D204       RANRECNO:       DW      1234    ;Example random record number

                                                ;Record will be written from
                                                ; address set by prior
                                                ; SETDMA call
```

```
0026 2A2400          LHLD    RANRECNO        ;Get random record number
0029 222100          SHLD    FCB$RANREC      ;Set up file control block
002C 0E22            MVI     C,B$WRITERAN    ;Function code
002E 110000          LXI     D,FCB           ;DE -> file control block
0031 CD0500          CALL    BDOS            ;A = 00 if operation successful
                                             ;A = nonzero if no data in file
                                             ; specifically:
                                             ;A = 03 -- CP/M could not
                                             ;        close current extent
                                             ;    05 -- directory full
                                             ;    06 -- attempt to write
                                             ;        beyond end of disk
```

Purpose

This function writes a specific CP/M record (128 bytes) into a random file. It is initiated in much the same way as the companion function, Read Random (code 33, 21H). It assumes that you have already opened the file, set the DMA address to the address in memory containing the record to be written to disk, and set the random record number in the FCB to the specified record being written. This function also computes the extent in which the specified record number lies and opens the extent (creating it if it does not already exist). The error codes returned in A by this call are the same as those for Read Random, with the addition of error code 05, which indicates a full directory.

Like the Read Random (but unlike the Write Sequential), this function does not update the logical extent and sequential (current) record number in the FCB. Therefore, any subsequent sequential operation will access the record just written by the Read Random call, but these functions will update the sequential record number. The Write Random can therefore be used to position to the required place in the file, which can then be accessed sequentially.

Notes

In order to use the Write Random, you must first open the base extent (extent 0) of the file. Even though there may be no data records in this extent, opening permits the file to be processed correctly.

As explained in the notes for the Read Random function, you can easily create a random file with gaps in it. If you were to create a file with record number 0 and record number 5000, there would be no intervening file extents.

Figure 5-25 shows an example subroutine that creates a random file (CRF) but avoids this problem. You specify the number of 128-byte CP/M records in the file. The subroutine creates the file and then writes zero-filled records throughout. This makes it easier to process the file and permits standard CP/M utility programs to copy the file because there is a data record in every logical record position in the file. It is no longer a "sparse" file.

Figure 5-26 shows a subroutine that ties the Read and Write Random functions together. It performs Random Operations (RO). Unlike the standard BDOS functions that operate on 128-byte CP/M records, RO can handle arbitrary record size from one to several thousand bytes. You specify the relative record number of your record, not the CP/M record number (RO computes this). RO also prereads a CP/M record when your logical record occupies part of a 128-byte record, either because your record is less than 128 bytes or because it spans more than one

```
                        ;CRF
                        ;Create random file
                        ;This subroutine creates a random file. It erases any previous
                        ;file before creating the new one, and then writes 0-filled
                        ;records throughout the entire file.

                        ;Entry parameters
                        ;       DE -> file control block for new file
                        ;       HL = Number of 128-byte CP/M records to be
                        ;               zero-filled.

                        ;Exit parameters
                        ;       Carry clear if operation successful (A = 0,1,2,3)
                        ;       Carry set if error (A = 0FFH)

                        ;Calling sequence
                        ;       LXI     D,FCB
                        ;       CALL    CRF
                        ;       JC      ERROR

0013 =          B$ERASE         EQU     19      ;Erase file
0016 =          B$CREATE        EQU     22      ;Create file
001A =          B$SETDMA        EQU     26      ;Set DMA address
0015 =          B$WRITESEQ      EQU     21      ;Write sequential record
0005 =          BDOS            EQU     5       ;BDOS entry point

                CRFBUF:                         ;Zero-filled buffer
0000 0000000000         DW      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                  0,0,0
0032 0000000000         DW      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                  0,0,0
0064 0000000000         DW      0,0,0,0,0,0,0,0,0,0,0,0,0,0

0080 0000       CRFRC:  DW      0               ;Record count

                CRF:
0082 228000             SHLD    CRFRC           ;Save record count
0085 D5                 PUSH    D               ;Preserve FCB pointer
0086 0E13               MVI     C,B$ERASE       ;Erase any existing file
0088 CD0500             CALL    BDOS
008B D1                 POP     D               ;Recover FCB pointer
008C D5                 PUSH    D               ; and resave
008D 0E16               MVI     C,B$CREATE      ;Create (and open new file)
008F CD0500             CALL    BDOS
0092 FEFF               CPI     0FFH            ;Carry set if OK, clear if error
0094 3F                 CMC                     ;Complete to use carry set if error
0095 D1                 POP     D               ;Recover FCB address
0096 D8                 RC                      ;Return if error
0097 D5                 PUSH    D               ;Resave FCB pointer

0098 0E1A               MVI     C,B$SETDMA      ;Set DMA address to 0-buffer
009A 110000             LXI     D,CRFBUF
009D CD0500             CALL    BDOS
00A0 D1                 POP     D               ;Recover FCB pointer

                CRFL:
00A1 2A8000             LHLD    CRFRC           ;Get record count
00A4 7D                 MOV     A,L
00A5 B4                 ORA     H               ;Check if count now zero
00A6 C8                 RZ                      ;Yes, exit
00A7 2B                 DCX     H               ;Downdate count
00A8 228000             SHLD    CRFRC           ;Save count
00AB D5                 PUSH    D               ;Resave FCB address
00AC 0E15               MVI     C,B$WRITESEQ    ;Write sequentially
00AE CD0500             CALL    BDOS

00B1 D1                 POP     D               ;Recover FCB
00B2 C3A100             JMP     CRFL            ;Write next record
```

Figure 5-25. Create random file

128-byte sector. The subroutine suppresses this preread if you happen to use a record size that is some multiple of 128 bytes. In this case, your records will fit exactly onto a 128-byte record, so there will never be some partially occupied 128-byte sector.

This example also contains subroutines to produce a 32-bit product from multiplying HL by DE (MLDL—Multiply double length) and a right bit shift for DE, HL (SDLR—Shift double length right).

```
                        ;RO
                        ;Random operation (read or write)

                        ;This subroutine reads or writes a random record from a file.
                        ;The record length can be other than 128-bytes. This
                        ;subroutine computes the start CP/M record (which
                        ;is 128 bytes), and, if reading, performs a random read
                        ;and moves the user-specified record into a user buffer.
                        ;If necessary, more CP/M records will be read until the complete
                        ;user-specified record has been input.
                        ;For writing, if the size of the user-specified record is not an exact
                        ;multiple of CP/M records, the appropriate sectors will be preread.
                        ;It is not necessary to preread when the user-specified record
                        ;is an exact CP/M record, nor when subroutine is processing
                        ;CP/M records entirely spanned by a user-specified record.

                        ;Entry parameters
                        ;       HL -> parameter block of the form:
                        ;               DB      0       ;0FFH when reading, 00H for write
                        ;               DW      FCB     ;Pointer to FCB
                        ;               DW      RECNO   ;User record number
                        ;               DW      RECSZ   ;User record size
                        ;               DW      BUFFER  ;Pointer to buffer of
                        ;                               ; RECSZ bytes in length

                        ;Exit parameters
                        ;       A = 0 if operation completed (and user record
                        ;             copied into user buffer)
                        ;           1 if attempt to read unwritten CP/M record
                        ;           3 if CP/M could not close an extent
                        ;           4 if attempt to read unwritten extent
                        ;           5 if CP/M could not create a new extent
                        ;           6 if attempt to read beyond end of disk

                        ;Calling sequence
                        ;       LXI     H,PARAMS        ;HL -> parameter block
                        ;       CALL    RO
                        ;       ORA     A               ;Check if error
                        ;       JNZ     ERROR

0021 =          FCBE$RANREC     EQU     33      ;Offset of random record no. in FCB
001A =          B$SETDMA        EQU     26      ;Set the DMA address
0021 =          B$READRAN       EQU     33      ;Read random record
0028 =          B$WRITERANZ     EQU     40      ;Write random record with zero-fill
                                                ; previously unallocated allocation
                                                ; blocks
0005 =          BDOS            EQU     5       ;BDOS entry point

                ROPB:                           ;Parameter block image
0000 00         ROREAD: DB      0               ;NZ when reading, Z when writing
0001 0000       ROFCB:  DW      0               ;Pointer to FCB
0003 0000       ROURN:  DW      0               ;User record number
0005 0000       ROURL:  DW      0               ;User record length
0007 0000       ROUB:   DW      0               ;Pointer to user buffer
0009 =          ROPBL   EQU     $-ROPB          ;Parameter block length

0009 0000       ROFRP:  DW      0               ;Pointer to start of user record fragment
                                                ; in first CP/M-record read in
```

Figure 5-26. Read/Write variable length records randomly

```
000B 00         ROFRL:  DB      0               ;Fragment length
000C 0000       RORNP:  DW      0               ;Record number pointer (in user FCB)
000E 00         ROWECR: DB      0               ;NZ when writing user records that are an
                                                ; exact super-multiple of CP/M-record (and
                                                ; therefore no preread is required)

000F            ROBUF:  DS      128             ;Buffer for CP/M record

                RO:
008F 110000             LXI     D,ROPB          ;DE -> local parameter block
0092 0E09               MVI     C,ROPBL         ;Parameter block length
0094 CDFE01             CALL    MOVE            ;Move C bytes from HL to DE

                        ;To compute offset of user record in CP/M record,
                        ; compute the relative BYTE offset of the start
                        ; of the user record within the file (i.e.
                        ; user record number * record size). The least
                        ; significant 7 bits of this product give the
                        ; byte offset of the start of the user record.
                        ;The product / 128 (shifted left 7 bits) gives the
                        ;CP/M record number of the start of the user record.

0097 2A0500             LHLD    ROURL           ;Get user record length
009A 7D                 MOV     A,L             ;Get LS bytes of user rec. length
009B E67F               ANI     7FH             ;Check if exact multiple of 128
009D B7                 ORA     A               ;(i.e. exact CP/M records)
009E 3E00               MVI     A,0             ;A = 0, flags unchanged
00A0 C2A400             JNZ     RONE            ;Not exact CP/M records
00A3 3D                 DCR     A               ;A =FF

                RONE:
00A4 320E00             STA     ROWECR          ;Set write-exact-CP/M-records flag
00A7 EB                 XCHG                    ;DE = user record length
00A8 2A0300             LHLD    ROURN           ;Get user record number
00AB CDB801             CALL    MLDL            ;DE,HL = HL * DE
                                                ;DE,HL = user-record byte offset in file
00AE D5                 PUSH    D               ;Save user-record byte offset
00AF E5                 PUSH    H
00B0 7D                 MOV     A,L             ;Get LS byte of product
00B1 E67F               ANI     7FH             ;Isolate byte offset within

00B3 4F                 MOV     C,A             ;CP/M record
00B4 0600               MVI     B,0             ;Make into word value
00B6 210F00             LXI     H,ROBUF         ;Get base address of local buffer
00B9 09                 DAD     B               ;HL -> Start of fragment in buffer
00BA 220900             SHLD    ROFRP           ;Save fragment pointer

                        ;Compute maximum fragment length that could reside in
                        ;remainder of CP/M record, based on the offset in the
                        ;CP/M record where the fragment starts.

00BD 47                 MOV     B,A             ;Take copy of offset in CP/M record
00BE 3E80               MVI     A,128           ;CP/M record size
00C0 90                 SUB     B               ;Compute 128 - offset
00C1 320B00             STA     ROFRL           ;Assume this is the fragment length

                        ;If the user record length is less than the assumed
                        ; fragment length, use it in place of the result above

00C4 47                 MOV     B,A             ;Get copy of assume frag. length
00C5 3A0600             LDA     ROURL+1         ;Get MS byte of user record length
00C8 B7                 ORA     A               ;If NZ, rec. len. must be > 128
00C9 C2D600             JNZ     ROFLOK          ;So fragment length is OK
00CC 3A0500             LDA     ROURL           ;Still a chance that rec. len.
00CF B8                 CMP     B               ; less than fragment len.
00D0 D2D600             JNC     ROFLOK          ;NC if user rec. len. => frag. len.
00D3 320B00             STA     ROFRL           ;User rec. len. < frag. len. so
                                                ; reset fragment length to smaller

                ROFLOK:
00D6 3A0E00             LDA     ROWECR          ;Get exact CP/M record flag
00D9 47                 MOV     B,A             ;for ANDing with READ flag
00DA 3A0000             LDA     ROREAD          ;Get read operation flag
00DD 2F                 CMA                     ;Invert so NZ when writing
```

Figure 5-26. (Continued)

```
00DE A0             ANA     B               ;Form logical AND
00DF 320E00         STA     ROWECR          ;Save back in flag

                    ;Recover the double length byte offset within the file
                    ;of the start of the user record. Shift 7 places right
                    ;to divide by 128 and get the CP/M record number for
                    ;the start of the user record.

00E2 E1             POP     H               ;Recover user rec. byte offset
00E3 D1             POP     D
00E4 0E07           MVI     C,7             ;Count for shift right

                ROS:
00E6 CDF101         CALL    SDLR            ;DE,HL = DE,HL / 2
00E9 0D             DCR     C
00EA C2E600         JNZ     ROS

00ED 7A             MOV     A,D             ;Error if DE still NZ after
00EE B3             ORA     E               ; division by 128.
00EF C2AC01         JNZ     ROERO

                                            ;Set CP/M record number in FCB
00F2 EB             XCHG                    ;DE = CP/M record number
00F3 2A0100         LHLD    ROFCB           ;Get pointer to FCB
00F6 012100         LXI     B,FCBE$RANREC   ;Offset of random record no. in FCB
00F9 09             DAD     B               ;HL -> ran. rec. no. in FCB
00FA 220C00         SHLD    RORNP           ;Save record number pointer
00FD 73             MOV     M,E             ;Store LS byte
00FE 23             INX     H
00FF 72             MOV     M,D             ;Store MS byte

0100 0E1A           MVI     C,B$SETDMA      ;Set DMA address to local buffer
0102 110F00         LXI     D,ROBUF
0105 CD0500         CALL    BDOS

0108 3A0E00         LDA     'ROWECR         ;Bypass preread if exact sector write
010B B7             ORA     A
010C C21F01         JNZ     ROMNF

010F 2A0100         LHLD    ROFCB           ;Get pointer to FCB
0112 EB             XCHG                    ;DE -> FCB
0113 0E21           MVI     C,B$READRAN     ;Read random function
0115 CD0500         CALL    BDOS

0118 FE05           CPI     5               ;Check if error code < 5
011A DCAF01         CC      ROCIE           ;Yes, check if ignorable error
                                            ; (i.e. error reading unwritten part
                                            ; of file for write operation preread)
011D B7             ORA     A               ;Check if error
011E C0             RNZ                     ;Yes

              ROMNF:                        ;Move next fragment
011F 2A0700         LHLD    ROUB            ;Get pointer to user buffer
0122 EB             XCHG                    ;DE -> user buffer
0123 2A0900         LHLD    ROFRP           ;HL -> start of user rec. in local buffer
0126 3A0B00         LDA     ROFRL           ;Get fragment length
0129 4F             MOV     C,A             ;Ready for MOVE

012A 3A0000         LDA     ROREAD          ;Check if reading
012D B7             ORA     A
012E C23201         JNZ     RORD1           ;Yes, so leave DE, HL unchanged
0131 EB             XCHG                    ;Writing, so swap source and destination
                                            ;DE -> start of user rec. in local buffer
                                            ;HL -> user buffer

              RORD1:
0132 CDFE01         CALL    MOVE            ;Reading - fragment local -> user buffer
                                            ;Writing - fragment user -> local buffer
0135 3A0000         LDA     ROREAD          ;Check if writing
0138 B7             ORA     A
0139 CA3D01         JZ      ROWR1           ;Writing, so leave HL -> user buffer
013C EB             XCHG                    ;HL -> next byte in user buffer

              ROWR1:
013D 220700         SHLD    ROUB            ;Save updated user buffer pointer
0140 3A0000         LDA     ROREAD          ;Check if reading
```

Figure 5-26. (Continued)

```
0143 B7              ORA     A
0144 C25001          JNZ     RORD3           ;Yes, bypass write code

0147 0E28            MVI     C,B$WRITERANZ   ;Write random
0149 2A0100          LHLD    ROFCB           ;Get address of FCB
014C EB              XCHG                    ;DE -> FCB
014D CD0500          CALL    BDOS

                RORD3:  ;Compute residual length of user record as yet unmoved.
                        ;If necessary (because more data needs to be transferred)
                        ;more CP/M records will be read. In this case
                        ;the start of the fragment will be offset 0. The fragment
                        ;length depends on whether the user record finishes within
                        ;the next sector or spans it. If the residual length  of the
                        ;user record is > 128, the fragment length will be set to
                        ;128.

0150 2A0500          LHLD    ROURL           ;Get residual user rec. length
0153 3A0B00          LDA     ROFRL           ;Get fragment length just moved
0156 5F              MOV     E,A             ;Make into a word value
0157 1600            MVI     D,0
0159 CDEA01          CALL    SUBHL           ;Compute ROURL - ROFRL
015C 7C              MOV     A,H             ;Check if result 0
015D B5              ORA     L
015E C8              RZ                      ;Return when complete USER
                                             ; record has been transferred
015F 220500          SHLD    ROURL           ;Save downdated residual rec. length
0162 4D              MOV     C,L             ;Assume residual length < 128
0163 118000          LXI     D,128           ;Check if residual length is < 128
0166 CDEA01          CALL    SUBHL           ;HL = HL - DE
0169 FA6E01          JM      ROLT128         ;negative if < 128
016C 0E80            MVI     C,128           ;=> 128, so set frag.length to 128

                ROLT128:
016E 79              MOV     A,C
016F 320B00          STA     ROFRL           ;Fragment length now is either 128
                                             ; if more than 128 bytes left to input
                                             ; in user record, or just the right
                                             ; number of bytes (< 128) to complete
                                             ; the user record.
0172 210F00          LXI     H,ROBUF         ;All subsequent CP/M records will start
0175 220900          SHLD    ROFRP           ; at beginning of buffer

                                             ;Update random record number in FCB
0178 2A0C00          LHLD    RORNP           ;HL -> random record number in  user FCB
017B 5E              MOV     E,M             ;Increment the random record number
017C 23              INX     H               ;HL -> MS byte of record number
017D 56              MOV     D,M             ;Get MS byte
017E 13              INX     D               ;Update record number itself
017F 7A              MOV     A,D             ;Check if record now 0
0180 B3              ORA     E
0181 C28701          JNZ     ROSRN           ;No, so save record number
0184 3E06            MVI     A,6             ;Indicate "seek past end of disk"
0186 C9              RET                     ;Return to user
                ROSRN:
0187 72              MOV     M,D             ;Save record number
0188 2B              DCX     H               ;HL -> LS byte
0189 73              MOV     M,E

                                             ;If writing, check if preread required
018A 3A0E00          LDA     ROWECR          ;Check if exact CP/M record write
018D B7              ORA     A
018E C21F01          JNZ     ROMNF           ;Yes, go move next fragment

0191 3A0000          LDA     ROREAD          ;If reading, perform read unconditionally
0194 B7              ORA     A
0195 C2A001          JNZ     RORD2

0198 3A0B00          LDA     ROFRL           ;For writes, bypass preread if
019B FE80            CPI     128             ; whole CP/M-record is to be overwritten
019D CA1F01          JZ      ROMNF           ; (fragment length = 128)

                RORD2:
01A0 0E21            MVI     C,B$READRAN     ;Read the next CP/M record
01A2 2A0100          LHLD    ROFCB           ; in sequence
```

Figure 5-26. (Continued)

```
01A5 EB              XCHG                ;DE -> FCB
01A6 CD0500          CALL    BDOS
01A9 C31F01          JMP     ROMNF       ;Go back to move next fragment

                     ROERO:              ;Error because user record number
                                         ; * User record length / 128 gives
                                         ; a CP/M record number > 65535.
01AC 3E04            MVI     A,4         ;Indicate "attempt to read unwritten
01AE C9              RET                 ; extent"

                     ROCIE:              ;Check ignorable error (preread
                                         ; for write operation)
01AF 47              MOV     B,A         ;Save original error code
01B0 3A0000          LDA     ROREAD      ;Check if read operation
01B3 B7              ORA     A
01B4 78              MOV     A,B         ;Restore original error code but
                                         ; leave flags unchanged
01B5 C0              RNZ                 ;Return if reading
01B6 AF              XRA     A           ;Fake "no error" indicator
01B7 C9              RET

                     ;MLDL
                     ;Multiply HL * DE using iterative ADD with product
                     ;returned in DE,HL.

                     ;Entry parameters
                     ;       HL = multiplicand
                     ;       DE = multiplier

                     ;Exit parameters
                     ;       DE,HL = product
                     ;       DE = multiplier

                     MLDL:
01B8 010000          LXI     B,0         ;Put 0 on top of stack
01BB C5              PUSH    B           ; to act as MS byte of product
                                         ;Check if either multiplicand
                                         ; or multiplier is 0
01BC 7C              MOV     A,H
01BD B5              ORA     L
01BE CAE501          JZ      MLDLZ       ;Yes, fake product
01C1 7A              MOV     A,D
01C2 B3              ORA     E
01C3 CAE501          JZ      MLDLZ       ;Yes, fake product

                                         ;This routine will be faster if
                                         ; the smaller value is in DE
01C6 7A              MOV     A,D         ;Get MS byte of current DE value
01C7 BC              CMP     H           ;Check which is smaller
01C8 DACC01          JC      MLDLNX      ;C set if D < H, so no exchange
01CB EB              XCHG
                     MLDLNX:
01CC 42              MOV     B,D         ;BC = multiplier
01CD 4B              MOV     C,E

01CE 54              MOV     D,H         ;DE = HL = multiplicand
01CF 5D              MOV     E,L

01D0 0B              DCX     B           ;Adjust count as
                                         ; 1 * multiplicand = multiplicand
                     MLDLA:              ;ADD loop
01D1 78              MOV     A,B         ;Check if all iterations completed
01D2 B1              ORA     C
01D3 CAE801          JZ      MLDLX       ;Yes, exit
01D6 19              DAD     D           ;HL = multiplicand + multiplicand
01D7 E3              XTHL                ;HL = MS bytes of result, TOS = part prod.
01D8 7D              MOV     A,L         ;Get LS byte of top half of product
01D9 CE00            ACI     0           ;Add one if carry set
01DB 6F              MOV     L,A         ;Replace
01DC 7C              MOV     A,H         ;Repeat for MS byte
01DD CE00            ACI     0
01DF 67              MOV     H,A
01E0 E3              XTHL
01E1 0B              DCX     B           ;Countdown on multiplier - 1
01E2 C3D101          JMP     MLDLA       ;Loop back until all ADDs done
```

Figure 5-26. (Continued)

```
                MLDLZ:
01E5 210000             LXI     H,0             ;Fake product as either multiplicand
                                                ; or multiplier is 0

                MLDLX:
01E8 D1                 POP     D               ;Recover MS part of product
01E9 C9                 RET


                ;SUBHL
                ;Subtract HL - DE.

                ;Entry parameters
                ;       HL = subtrahend
                ;       DE = subtractor

                ;Exit parameters
                ;       HL = difference

                SUBHL:
01EA 7D                 MOV     A,L             ;Get LS byte
01EB 93                 SUB     E               ;Subtract without regard to carry
01EC 6F                 MOV     L,A             ;Put back into difference
01ED 7C                 MOV     A,H             ;Get MS byte
01EE 9A                 SBB     D               ;Subtract including carry
01EF 67                 MOV     H,A             ;Move back into difference
01F0 C9                 RET

                ;SDLR
                ;Shift DE,HL right one place (dividing DE,HL by 2)

                ;Entry parameters
                ;       DE,HL = value to be shifted

                ;Exit parameters
                ;       DE,HL = value / 2

                SDLR:
01F1 B7                 ORA     A               ;Clear carry
01F2 EB                 XCHG                    ;Shift DE first
01F3 CDF701             CALL    SDLR2
01F6 EB                 XCHG                    ;Now shift HL

                                                ;Drop into SDLR2 with carry
                                                ; set correctly from LS bit
                                                ; of DE
                SDLR2:                          ;Shift HL right one place
01F7 7C                 MOV     A,H             ;Get MS byte
01F8 1F                 RAR                     ;Bit 7 set from previous carry,
                                                ;Bit 0 goes into carry
01F9 67                 MOV     H,A             ;Put shift MS byte back
01FA 7D                 MOV     A,L             ;Get LS byte
01FB 1F                 RAR                     ;Bit 7 = bit 0 of MS byte
01FC 6F                 MOV     L,A             ;Put back into result
01FD C9                 RET

                ;MOVE
                ;Moves C bytes from HL to DE

                MOVE:
01FE 7E                 MOV     A,M             ;Get source byte
01FF 12                 STAX    D               ;Store in destination
0200 13                 INX     D               ;Update destination pointer
0201 23                 INX     H               ;Update source pointer
0202 0D                 DCR     C               ;Downdate count
0203 C2FE01             JNZ     MOVE            ;Get next byte
0206 C9                 RET
```

Figure 5-26. (Continued)

Function 35: Get File Size

Function Code: C = 23H
Entry Parameters: DE = Address of FCB
Exit Parameters: Random record field set in FCB

Example

```
0023 =          B$GETFSIZ       EQU     35      ;Get Random File LOGICAL size
0005 =          BDOS            EQU     5       ;BDOS entry point

                FCB:                            ;File control block
0000 00         FCB$DISK:       DB      0       ;Search on default disk drive
0001 46494C454EFCB$NAME:        DB      'FILENAME'      ;File name
0009 545950     FCB$TYP:        DB      'TYP'   ;File type
000C 00         FCB$EXTENT:     DB      0       ;Extent
000D 0000       FCB$RESV:       DB      0,0     ;Reserved for CP/M
000F 00         FCB$RECUSED:    DB      0       ;Records used in this extent
0010 0000000000FCB$ABUSED:      DB      0,0,0,0,0,0,0,0 ;Allocation blocks used
0018 0000000000                 DB      0,0,0,0,0,0,0,0
0020 00         FCB$SEQREC:     DB      0       ;Sequential rec. to read/write
0021 0000       FCB$RANREC:     DW      0       ;Random rec. to read/write
0023 00         FCB$RANRECO:    DB      0       ;Random rec. overflow byte (MS)

0024 0E23               MVI     C,B$GETFSIZ     ;Function code
0026 110000             LXI     D,FCB           ;DE -> file control block
0029 CD0500             CALL    BDOS
002C 2A2100             LHLD    FCB$RANREC      ;Get random record number
                                                ;HL = LOGICAL file size
                                                ; i.e. the record number of the
                                                ; last record
```

Purpose This function returns the virtual size of the specified file. It does so by setting the random record number (bytes 33-35) in the specified FCB to the maximum 128-byte record number in the file. The virtual file size is calculated from the record address of the record following the end of the file. Bytes 33 and 34 form a 16-bit value that contains the record number, with overflow indicated in byte 35. If byte 35 is 01, this means that the file has the maximum record count of 65,536.

If the function cannot find the file specified by the FCB, it returns with the random record field set to 0.

You can use this function when you want to add data to the end of an existing file. By calling this function first, the random record bytes will be set to the end of file. Subsequent Write Random calls will write out records to this preset address.

Notes Do not confuse the virtual file size with the actual file size. In a random file, if you write just a single CP/M record to record number 1000 and then call this function, it will return with the random record number field set in the FCB to 1000—even though only a single record exists in the file.

For sequential files, this function returns the number of records in the file. In this case, the virtual and actual file sizes coincide.

Function 36: Set Random Record Number

Function Code: C = 24H
Entry Parameters: DE = Address of FCB
Exit Parameters: Random record field set in FCB

Example

```
0024 =          B$SETRANREC     EQU     36      ;Set Random Record Number
0005 =          BDOS            EQU     5       ;BDOS entry point

                FCB:                            ;File control block
0000 00         FCB$DISK:       DB      0       ;Search on default disk drive
0001 46494C454EFCB$NAME:        DB      'FILENAME'      ;File name
0009 545950     FCB$TYP:        DB      'TYP'   ;File type
000C 00         FCB$EXTENT:     DB      0       ;Extent
000D 0000       FCB$RESV:       DB      0,0     ;Reserved for CP/M
000F 00         FCB$RECUSED:    DB      0       ;Records used in this extent
0010 0000000000FCB$ABUSED:      DB      0,0,0,0,0,0,0,0 ;Allocation blocks used
0018 0000000000                 DB      0,0,0,0,0,0,0,0
0020 00         FCB$SEQREC:     DB      0       ;Sequential rec. to read/write
0021 0000       FCB$RANREC:     DW      0       ;Random rec. to read/write
0023 00         FCB$RANRECO:    DB      0       ;Random rec. overflow byte (MS)

                                                ;... file opened and read
                                                ; or written sequentially...

0024 0E24               MVI     C,B$SETRANREC   ;Function code
0026 110000             LXI     D,FCB           ;DE -> file control block
0029 CD0500             CALL    BDOS
002C 2A2100             LHLD    FCB$RANREC      ;Get random record number
                                                ;HL = random record number
                                                ; that corresponds to the
                                                ; sequential progress down
                                                ; the file.
```

Purpose

This function sets the random record number in the FCB to the correct value for the last record read or written sequentially to the file.

Notes

This function provides you with a convenient way to build an index file so that you can randomly access a sequential file. Open the sequential file, and as you read each record, extract the appropriate key field from the data record. Make the BDOS Set Random Record request and create a new data record with just the key field and the random record number. Write the new data record out to the index file.

Once you have done this for each record in the file, your index file provides a convenient method, given a search key value, of finding the appropriate CP/M record in which the data lies.

You can also use this function as a means of finding out where you are currently positioned in a sequential file—either to relate a CP/M record number to the position, or simply as a place-marker to allow a repositioning to the same place later.

Function 37: Reset Logical Disk Drive

Function Code: C = 25H
Entry Parameters: DE = Logical drive bit map
Exit Parameters: A = 00H

Example

```
0025 =          B$RESETD        EQU     37      ;Reset Logical Disks
0005 =          BDOS            EQU     5       ;BDOS entry point
```

```
                                        ;DE = Bit map of disks to be
                                        ; reset
                                        ;Bits are = 1 if disk to be
                                        ; reset
                                        ;Bits 15 14 13 ... 2 1 0
                                        ;Disk  P  O  N ... C B A

0000 110200             LXI     D,0000$0000$0000$0010B  ;Reset drive B:
0003 0E25               MVI     C,B$RESETD      ;Function code
0005 CD0500             CALL    BDOS
```

Purpose

This function resets individual disk drives. It is a more precise version of the Reset Disk System function (code 13,ODH), in that you can set specific logical disks rather than all of them.

The bit map in DE shows which disks are to be reset. The least significant bit of E represents disk A, and the most significant bit of D, disk P. The bits set to 1 indicate the disks to be reset.

Note that this function returns a zero value in A in order to maintain compatibility with MP/M.

Notes

Use this function when only specific diskettes need to be changed. Changing a diskette without requesting CP/M to log it in will cause the BDOS to assume that an error has occurred and to set the new diskette to Read-Only status as a protective measure.

Function 40: Write Random with Zero-fill

Function Code: C = 28H
Entry Parameters: DE = Address of FCB
Exit Parameters: A = Return Code

Example

```
0028 =          B$WRITERANZ     EQU     40      ;Write Random with Zero-Fill
0005 =          BDOS            EQU     5       ;BDOS entry point

                FCB:                            ;File control block
0000 00         FCB$DISK:       DB      0       ;Search on default disk drive
0001 46494C454EFCB$NAME:        DB      'FILENAME'      ;File name
0009 545950     FCB$TYP:        DB      'TYP'   ;File type
000C 00         FCB$EXTENT:     DB      0       ;Extent
000D 0000       FCB$RESV:       DB      0,0     ;Reserved for CP/M
000F 00         FCB$RECUSED:    DB      0       ;Records used in this extent
0010 0000000000FCB$ABUSED:      DB      0,0,0,0,0,0,0,0 ;Allocation blocks used
0018 0000000000                 DB      0,0,0,0,0,0,0,0
0020 00         FCB$SEQREC:     DB      0       ;Sequential rec. to read/write
0021 0000       FCB$RANREC:     DW      0       ;Random rec. to read/write
0023 00         FCB$RANRECO:    DB      0       ;Random rec. overflow byte (MS)

0024 D204       RANRECNO:       DW      1234    ;Example random record number

                                                ;Record will be written from
                                                ; address set by prior
                                                ; SETDMA call
0026 2A2400             LHLD    RANRECNO        ;Get random record number
0029 222100             SHLD    FCB$RANREC      ;Set up file control block
002C 0E28               MVI     C,B$WRITERANZ   ;Function code
002E 110000             LXI     D,FCB           ;DE -> file control block
0031 CD0500             CALL    BDOS            ;A = 00 if operation successful
```

```
;A = nonzero if no data in file
; specifically :
;A = 03 -- CP/M could not
;        close current extent
;    05 -- directory full
;    06 -- attempt to write
;        beyond end of disk
```

Purpose This function is an extension to the Write Random function described previously. In addition to performing the Write Random, it will also fill each new allocation block with 00H's. Digital Research added this function to assist Microsoft with the production of its COBOL compiler—it makes the logic of the file handling code easier. It also is an economical way to completely fill a random file with 00H's. You need only write one record per allocation block; the BDOS will clear the rest of the block for you.

Notes Refer to the description of the Write Random function (code 34).