PCPM11PG.WS4   (= Personal CP/M version 1.1 Programmer's Guide)

- "Personal CP/M Version 1.1 -- Programmer's Guide"

(Retyped by Emmanuel ROCHE.)

Foreword

Personal CP/M is a microcomputer operating system designed for the Zilog Z-80 or any compatible microprocessor. To run Personal CP/M, your computer must have an ASCII console (including at least a keyboard and a video display screen), 1 to 16 disk drives, and a minimum of 32 kilobytes of Random Access Memory (RAM).

This manual describes the Basic Disk Operating System (BDOS) functions of Personal CP/M, and how to call the functions using Zilog Z-80 assembler language (ROCHE>??? All the sample assembly language programs found are in Intel 8080 mnemonics, and ZSID is not provided, nor any Z-80 assembler...). It is written for experienced programmers who are writing application software in the Personal CP/M environment. It assumes that you are familiar with the system features and facilities described in the "Personal CP/M User's Guide", the "Personal CP/M System Guide", and the "Programmer's Utilities Guide for the CP/M Family of Operating Systems".

Section 1 of this manual describes the components of the operating system, where they reside in memory, and how they work together to provide a standard operating environment for application programs.

Section 2 describes how an application program can call on Personal CP/M to perform serial input and output, and manage disk files. It also provides a detailed description of each operating system function.

Section 3 presents four example programs.

The appendixes contain a summary of system functions, BDOS error handling information, and user number conventions.

Table of Contents

(To be done by WS4...)

Appendixes

(idem)

Tables, Figures, and Listing

Tables

(idem)

Figures

(idem)

Listing

(idem)

Section 1: Introduction to Personal CP/M

This manual describes Personal CP/M system organization, including the structure of memory and system entry points. This manual provides information necessary to write programs that operate under Personal CP/M, and use the peripheral and disk I/O facilities of the system.

1.1 Components of Personal CP/M

Personal CP/M is divided into the Basic Input/Output System (BIOS), the Basic Disk Operating System (BDOS), and the Console Command Processor (CCP) which executes in the upper portion of the Transient Program Area (TPA). The BIOS, a hardware-dependent module, is the exact low-level interface to a particular computer system for peripheral device I/O. Although a standard BIOS is supplied by Digital Research, explicit instructions are provided in the "Personal CP/M System Guide" for field reconfiguration of the BIOS to match most hardware environments. The BDOS is a hardware-independent module that provides a standard operating environment for transient programs, by making services available through numbered system function calls.

1.2 Personal CP/M memory organization

The BIOS and BDOS are combined into a single module with a common entry point, and referred to as the FDOS ("Full Disk Operating System"). The CCP module is a distinct program that uses the FDOS to provide you with the user interface to the operating system. The TPA is an area of memory where non-resident operating system utilities and user (transient) programs are executed. If necessary, programs in the TPA can overwrite the CCP to use all available memory to do its job. The presence of the CCP is not required for any application program. The lower portion of memory (ROCHE> Called "Page Zero".) is reserved for system information, and is detailed in Section 2.2.2, "File Control Block", and in the "Personal CP/M System Guide". The memory organization of the Personal CP/M system is shown in Figure 1-1.

```
        +-----------+
        |   FDOS    |
FBASE:  +--+-----+--+
        |  | CCP |  |
        |  +-----+  |
        |           |
        |    TPA    |
TBASE:  +-----------+
        | Page Zero |
BOOT:   +-----------+
```

Figure 1-1. Personal CP/M memory organization

The memory address corresponding to FBASE varies from version to version, and is described in the "Personal CP/M System Guide". As seen from the preceding diagram, TBASE=0100H and BOOT=0000H, which is the base of Random Access Memory (RAM). At location BOOT, there is a jump to the machine code in the BIOS, which performs a system warm start. The BIOS warm start routine loads and initializes the program variables necessary to return control to the CCP. Thus, transient programs need only jump to location BOOT to return control to Personal CP/M at the command level. The principal entry point to the FDOS is at location 0005H, where there is a jump to FBASE. The address field at 0006H contains the value of FBASE, and can be used to determine the size of available memory, assuming that a transient program is overlaying the CCP.

1.3 Program execution

Transient programs are loaded into the TPA, and executed through the CCP by typing command lines following each Personal CP/M system prompt ("A>"). The CCP is capable of parsing the following general form of the command line:

```
command
command file1
command file1 file2
```

Programs with different command tail formats must do their own parsing of the command tail stored in the buffer at 0080H.

If the command is a built-in function of Personal CP/M, it is executed immediately. Otherwise, the CCP searches the currently-logged drive for a command file in the following form:

```
command.COM
```

If the COMmand file is found, it is assumed to be a memory image of a program that executes in the TPA, and thus implicitly originates at TBASE in memory.

The CCP loads the COM file from the disk into memory, starting at TBASE. The COM file can extend up to the beginning of FBASE, using all the TPA area. Personal CP/M loads a command file from user 0 if it has the system attribute set, when the current user number is greater than zero. (See Appendix C for more information on user number conventions.)

If the command is followed by one or two file specifications, the CCP prepares one or two File Control Block (FCB) names in the system parameter area, the Page Zero of memory. These optional FCBs are in the form necessary to access files through the FDOS, and are described in Section 2, "Operating system call conventions".

The transient program receives control from the CCP, and begins execution using the I/O facilities of the FDOS. The CCP uses a "call" instruction to transfer control to the transient program. Thus, the program can execute a "return" to the CCP upon completion of its processing, provided that it has not written over any portion of the CCP, or it can execute a jump to location BOOT to pass control back to the warm boot routine in Personal CP/M. In no case should the program use any memory above the TPA (FBASE-1).

1.4 Specifying options to a command

From time to time, a command needs to know more than the name of one or two files in its command tail. Those "run-time parameters" are known as options. The problem is how to pass them to the command file.

Personal CP/M is a member of the CP/M family of Operating Systems, wich started to be sold with CP/M Version 1.3, followed by Version 1.4. Then came CP/M Version 2.0, which used a table-driven BIOS and BDOS (the file system) Version 2. A quick update, Version 2.1, followed, before CP/M became the de facto industry standard for 8-bit microcomputers with CP/M Version 2.2.

This was the single-user branch of the family. There is also a multi-user branch, still 8-bit, called MP/M, which saw 2 versions, using another BDOS called BDOS 3. In 1982, it was decided to make a single-user version of MP/M that could use the then awaited Z-800 microprocessor and hard disks that were providing much more capacity than floppies. This more rational version of CP/M is CP/M Version 3.0, better known as "CP/M Plus".

Unfortunately, Zilog was unable to provide the Z-800 in time, while Intel was selling the 8086, and hard disks remained prohibitively expensive, especially now that the market shifted from home/hobby computing to business with the badly-named "Personal Computer"... And, of course, companies have more money than individuals, so why lower the price?

Following its success with MS-DOS, Microsoft then decided to attack the 8-bit market with an 8-bit version called MSX-DOS. One of their particularities was that those microcomputers were booting from ROM, rather than from floppies. In reaction, Digital Research decided to introduce Personal CP/M, a version of Good Old CP/M Version 2.2 which was rewritten to use Z-80 mnemonics, and designed to be able to boot from ROM. Unfortunately, Personal CP/M was only implemented on one microcomputer, the MZ-800 (note the "Z-800" in its name...) made by Sharp, and it was booting from floppy! At the time, microcomputer magazines were crazy about the IBM PC, so it was a total flop...

Also, since it was the last son of CP/M 2.2, it was using BDOS Version 2, that can only manage disks up to 8 Megabytes (versus the 512 Megabytes of CP/M Plus), just when hard disks became common! And floppy disks died after reaching 1.44 Megabyte capacity... The problem of Personal CP/M is to find a drive with circa 8 Megabyte capacity.

(The same year, Digital Research introduced 2 products that could both have revolutionized microcomputers if the IBM PC had not focused all the attention: GSX and CP/NET. GSX provides a PORTABLE graphics system (it also works under Personal CP/M) and CP/NET provides access to another computer's files via a network (it also works under Personal CP/M). Both products were 20 years in advance upon their time (and time is running quickly for IBM PCs). Unfortunately, the market had then shifted to the IBM PC, where hardware compatibility became essential (contrary to CP/M, which is portable across many hardware configurations), and all those great programs were soon forgotten.)

Back to options.

One of the problem with CP/M is that there is no standard for specifying options to a program. At the beginning, this was not foreseen.

Under CP/M Version 1.4, ASM (the 8080 absolute assembler) was using the filetype to hold its options.

Under CP/M Version 2.2, MAC (the 8080 macro assembler) was using a "$" character, at the end of the command tail, to specify the options.

Under CP/M Version 3.0, alias CP/M Plus, a standard was finally settled, which uses the "[" character to indicate options. Thus, one uses:

```
        A>command file1 file2 [option1,option2,...,optionN]
```

The closing square bracket ("]") is optional. Options are separated by a comma (",").

This is this more rational option specification that is used by Personal CP/M. If you write application programs for Personal CP/M, or modify a CP/M 2.2 program to run under Personal CP/M, please follow this standard.

1.5 Calling a function

The transient program can use the Personal CP/M I/O facilities to communicate with your console and peripheral devices, including the disk subsystem. To access the I/O system, the transient program passes a function number and a parameter to Personal CP/M through the FDOS entry point at location 0005H. In the case of a disk read, for example, the transient program sends the function number corresponding to the disk read, with the address of an FCB, to the Personal CP/M BDOS. In turn, the FDOS performs the operation, and returns with either a disk read completion indicator, or an error number indicating that the disk read was unsuccessful.

Some functions have been added or changed from previous CP/M versions to increase the programming capabilities of Personal CP/M. Functions 7 (AUXILIARY INPUT STATUS) and 8 (AUXILIARY OUTPUT STATUS) have been changed from GET and SET I/O BYTE, to enable you to write a program that performs auxiliary I/O (such as a communications program, file transfer program, or a terminal emulator), and which is portable across different hardware environments. Function 45 (SET BDOS ERROR MODE) gives you more control over error handling, by allowing a BDOS error to be reported back to the program that called the function. This feature allows a program to control how it responds to BDOS errors. Normally, programs that encounter a BDOS error would be automatically terminated. Function 48 (FLUSH BUFFERS) takes all sector buffers, and immediately writes them to the disk. This feature reduces the risk of losing file information when the BIOS uses blocking and deblocking, and a system failure occurs. Functions 109 (GET/SET CONSOLE MODE), 110 (GET/SET OUTPUT DELIMITER), 111 (PRINT BLOCK), 113 (DIRECT SCREEN FUNCTIONS) are present in Personal CP/M to improve console I/O performance. Function 112 (LIST BLOCK) is included to improve printer I/O performance.

(ROCHE> Functions 124 (Byte BLT Copy) and 125 (Byte BLT Alter) are present in the BDOS, but are not implemented...)

Section 2: Operating system call conventions

This section provides detailed information for making direct operating system calls from user programs. Many of the functions listed below, however, can be accessed more easily through the sequential I/O macro library provided with the MAC macro assembler, and listed in the "Programmer's Utilities Guide".

Personal CP/M functions available for access by transient programs fall into 3 categories: simple device I/O, disk file I/O, and high-performance video.

The simple device I/O operations include the following:

- read a console character
- write a console character
- read character from auxiliary device
- write character to auxiliary device
- write a character to list device
- get auxiliary I/O status
- print console buffer
- read console buffer
- interrogate console ready
- get/set output delimiter
- print block
- list block

The following FDOS operations perform disk I/O:

- disk system reset
- drive selection
- file creation
- file open
- file close
- directory search
- file delete
- file rename
- random or sequential read
- random or sequential write
- interrogate selected disk
- set DMA address
- set/reset file attributes
- return current disk
- compute file size
- set BDOS error mode
- flush buffers

The high-performance video operations are as follows:

- direct screen functions
- (ROCHE> Missing line(s) listing BDOS Function 124 and 125.)

2.1 FDOS operation

As mentioned in Section 1.4, to access the FDOS functions, the transient program passes a function number and information address to location 0005H. In general, the program passes a function number in register C. Single-byte entry parameters are passed in register E, and double-byte entry parameters are passed in register pair DE. Single-byte values are returned in register A, and double-byte values are returned in register pair HL. A zero value is returned when the function number is out of range. Register A = L and register B = H upon return in all cases. Personal CP/M functions and their numbers are listed in Appendix A, "System function summary".

Note: Functions 28 (WRITE PROTECT DISK) and 32 (GET/SET USER CODE) should be avoided in application programs, to maintain upward compatibility with multi-user CP/M versions.

Upon entry to a transient program, the CCP leaves the stack pointer set to a 32-level stack area, with the CCP return address pushed onto the stack, leaving 31 levels before overflow occurs. Although this stack is usually not used by a transient program (most transients return to the CCP through a jump to location BOOT), it is large enough to make Personal CP/M system calls because the FDOS switches to a local stack at system entry. For example, the Intel 8080 assembly-language program segment below reads characters continuously until an asterisk is encountered, at which time control returns to the CCP, assuming a standard CP/M system with BOOT = 0000H.

Listing 2-1. Assembly language program segment

```
bdos    EQU     5               ; Standard CP/M entry
conin   EQU     1               ; Console input function
;
        ORG     0100H           ; Base of TPA
;
nextc:  MVI     C,conin         ; Read next character
        CALL    bdos            ; Return character in A-reg
        CPI     '*'             ; End of processing?
        JNZ     nextc           ; Loop if not
        RET                     ; Return to CCP
;
        END
```

2.2 File structure

Personal CP/M implements a named file structure on each disk, providing an organization that allows a particular file to contain any number of logical sectors -- from none to full drive capacity. Each logical drive has a separate disk directory and file data area. The disk filenames are in 3 parts: the drive specifier (one character: A through P), the filename (consisting of one-to-eight non-blank characters), and the filetype (consisting of zero-to-three non-blank characters). Valid characters used in creating filenames and filetypes are alphabetic characters, numeric characters, and the following punctuation characters: " # $ % ^ & @ + ' - ` /. You can also create a filename or filetype with lowercase characters, but you can only access it from a program. It will not be accessible from the CCP. The filename distinguishes individual files in each category. The filetype names the generic category of a particular file. The filetypes listed in Table 2-1 name a few generic categories that have been established, although some filetypes are arbitrary.

Table 2-1. Personal CP/M filetypes

Filetype	Meaning
ASC	ASCII text file
ASM	Assembler source
BAK	Backup file
BAS	BASIC source file
COM	Command file
HEX	Hex machine code
PLI	PL/I source file
PRN	Printer listing
REL	Relocatable module
SYM	SID symbol file
WS4	WordStar Version 4.0 document file
$$$	Temporary file

2.2.1 File records

Files in Personal CP/M can be thought of as a sequence of up to 65,535 logical sectors of 128 bytes each, numbered from 0 through 65,535, thus allowing a maximum of 8 megabytes for each file. Note, however, that, although the logical sectors may be considered logically contiguous, they may not be physically contiguous in the disk data area. Internally, all files are divided into 16-kilobyte segments called logical extents, so that counters are easily maintained as 8-bit values. The division into extents is discussed in Section 2.2.2, "File Control Block"; however, they are not particularly significant for the programmer, because each extent is automatically accessed in both sequential and random access modes.

Personal CP/M BDOS calls recognize only 128-byte logical sectors. To create a text (ASCII) file, insert 0DH followed by 0AH (Carriage Return and Line Feed) at the end of each line of the source file. Add 1AH (Ctrl-Z) at the end of the ASCII file. To find the end of a binary file, call Function 35 (COMPUTE FILE SIZE) with the FCB address in register pair DE. Function 35 returns the number of logical sectors that have been written.

2.2.2 File Control Block

In the file operations starting with Function 15, DE usually addresses an FCB. Transient programs often use the default FCB area reserved by Personal CP/M at location 005CH (normally 005CH) for simple file operations. Personal CP/M provides a default buffer location for disk I/O at location 0080H (normally 0080H), which is the initial default DMA address (see Function 26).

The FCB data area consists of a sequence of 33 bytes when the file is accessed sequentially, and a series of 36 bytes when the file is accessed randomly. The default FCB, located at 005CH, can be used for random access files, because the 3 bytes starting at 007DH are available for this purpose. Figure 2-1 shows the FCB format with the following fields:

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

Figure 2-1. File Control Block format

Table 2-2 describes each of the fields in the file control block format.

Table 2-2. File Control Block fields

Format: Field
Definition

DR
Drive code (0-16)
0 = use default drive
1 = auto disk specifier drive A
2 = auto disk specifier drive B
...
16 = auto disk specifier drive P

F1...F8
Contain the filename in ASCII uppercase, with high bit = 0.

T1,T2,T3
Contain the filetype in ASCII uppercase, with high bit = 0. T1', T2', and T3' denote the bit of these positions.
T1' = 1 --> Read-Only file
T2' = 1 --> SYS file, no DIR list

EX
Contains the current extent number, normally set to 00H by the user, but in range 0-31 during file I/O.

S1
Reserved for internal system use.

S2
Reserved for internal system use, set to zero on call to OPEN, MAKE, SEARCH.

RC
Record count for extent EX; takes on values from 0-127.

D0...Dn
Filled in by Personal CP/M; reserved for system use.

CR
Current record to read or write in a sequential file operation; normally set to zero by user.

R0,R1,R2
Optional random record number in the range 0-65535 (0-FFFF); R0, R1, R2 constitute an 16-bit value with low byte R0, high byte R1, and byte R2 = 0.

Each file being accessed through Personal CP/M must have a corresponding FCB, which provides the name and allocation information for all subsequent file operations. Bytes 1 through 11 are set by the CCP to the ASCII character values for the filename and filetype. Byte 0 is set to the drive specifier. All other fields are set to zero. When constructing your own FCB, it is your responsibility to fill the lower 12 bytes of the FCB, and initialize the CR field to zero.

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

The CCP constructs the first 12 bytes of two optional FCBs for a transient command by scanning the remainder of the command line following the transient filename, denoted by FILE1 and FILE2 in the prototype command line described in Section 1.3, "Program execution", with unspecified fields set to ASCII blanks. If no filenames are specified in the original command, the fields beginning at 005DH and 006DH contain blanks. In all cases, the CCP translates lowercase letters to uppercase, to be consistent with the Personal CP/M file-naming conventions. The first FCB is constructed at location 005CH, and can be used as is for subsequent file operations. The second FCB occupies the D0...Dn portion of the first FCB, and must be moved to another area of memory before use. For example, assume that the command line shown below is typed; then, the CCP loads the file PROGNAME.COM into the TPA, and initializes the default FCB at 005CH to drive specifier 2, filename X, and filetype ZOT:

```
        A>progname b:x.zot y.zap
```

The drive specifier for the second file specification takes the default value 0, which the CCP places at 006CH, with the filename Y placed into location 006DH, and filetype ZAP located 8 byte later at 0075H. The CCP sets all remaining fields through CR to zero. Note again that it is your responsibility to move this second file specification to another area, usually a separate file control block that you create, before opening the file that begins at 005CH, because the OPEN operation overwrites the second file specification.

As an added convenience, the default buffer area at location 0080H is initialized to the command tail typed by the operator following the program name. The first position contains the number of characters, followed by the actual characters. Given the above command line, the area beginning at 0080H is initialized as follows. The characters are translated to uppercase ASCII. Uninitialized memory follows the last valid character:

```
        0080H:
        +00 +01 +02 +03 +04 +05 +06 +07 +08 +09 +0A +0B +0C +0D +0E +0F
         0E ' ' 'B' ':' 'X' '.' 'Z' 'O' 'T' ' ' 'Y' '.' 'Z' 'A' 'P'  00
```

Again, it is your responsibility as the programmer to extract the information from this buffer before any file operations are performed, unless you explicitly change the default DMA address.

2.3 BDOS function calls

Individual functions are described in detail in the following pages.

```
        BDOS Function  0: System Reset
        Entry parameters:
              Register C: 00H
        Returned   value: None
```

The SYSTEM RESET function returns control to the Personal CP/M operating system at the CCP level. The CCP reinitializes the disk subsystem. It also selects drive A. This function has exactly the same effect as a jump to location BOOT (0000H).

```
        BDOS Function  1: Console Input
        Entry parameters:
              Register C: 01H
        Returned   value:
              Register A: ASCII character
```

The CONSOLE INPUT function reads the next console character to register A. Graphic characters, along with Carriage Return, Line Feed, and Backspace (Ctrl-H), are echoed to the console. Tab characters (Ctrl-I) move the cursor to the next tab stop. A check is made for start/stop scroll (Ctrl-S). The FDOS does not return to the calling program until a character has been typed, thus suspending execution if a character is not ready.

```
        BDOS Function  2: Console Output
        Entry parameters:
              Register C: 02H
              Register E: ASCII character
        Returned   value: None
```

The CONSOLE OUTPUT function sends the ASCII character from register E to the console device. As in function 1, tabs are expanded, and checks are made for start/stop scroll and printer echo (see Function 109).

```
        BDOS Function  3: Auxiliary Input
        Entry parameters:
              Register C: 03H
        Returned   value:
              Register A: ASCII character
```

The AUXILIARY INPUT function reads the next character from the auxiliary input device into register A. Control does not return until the character has been read.

```
        BDOS Function  4: Auxiliary Output
        Entry parameters:
              Register C: 04H
              Register E: ASCII character
        Returned   value: None
```

The AUXILIARY OUTPUT function sends the character from register E to the auxiliary output device. Control does not return until the character can be sent.

```
        BDOS Function  5: List Output
        Entry parameters:
              Register C: 05H
              Register E: ASCII character
        Returned   value: None
```

The LIST OUTPUT function sends the ASCII character in register E to the logical listing device. Control does not return until the character can be sent.

```
        BDOS Function  6: Direct Console I/O
        Entry parameters:
              Register C: 06H
              Register E: 0FFH (input/status), or
                          0FEH (status), or
                          char (output)
        Returned   value:
              Register A: char or status: no value
```

DIRECT CONSOLE I/O is supported under Personal CP/M for those specialized applications where basic console input and output are required. Use of this function bypasses all Personal CP/M normal control character functions (for example, Ctrl-S and Ctrl-P). Programs that performed direct console I/O through the BIOS under previous CP/M products should be changed to use this function, so that they can be fully supported under future products. A program calls Function 6 by passing one of the 3 different values in register E. The values and their meanings are summarized in Table 2-3.

Table 2-3. Function 6 entry parameters

Format: Register E value
        Meaning

0FFH -- Console input/status command
Returns an input character; if no character is ready, a value of zero is returned.

0FEH -- Console status command
On return, register A contains a zero if no character is ready; otherwise, it contains 0FFH.

ASCII character -- Console output command
Function 6 assumes that register E contains a valid ASCII character, and sends it to the console.

```
        BDOS Function  7: Auxiliary Input Status
        Entry parameters:
              Register C: 07H
        Returned   value:
              Register A: Auxiliary input status
                          00H = no character for input
                          0FFH = character ready for input
```

The AUXILIARY INPUT STATUS function returns the value 0FFH in register A if a character is ready for input from the logical auxiliary input device, AUXIN:. If no character is ready for input, the value 00H is returned.

```
        BDOS Function  8: Auxiliary Output Status
        Entry parameters:
              Register C: 08H
        Returned   value:
              Register A: Auxiliary Output Status
                          00H = device not ready for output
                          0FFH = device ready for output
```

The AUXILIARY OUTPUT STATUS function returns the value 0FFH in register A if the logical auxiliary output device, AUXOUT:, is ready to accept a character for output. If the device is not ready for output, the value 00H is returned.

```
        BDOS Function  9: Print String
        Entry parameters:
              Register C: 09H
             Register DE: String address
        Returned   value: None
```

The PRINT STRING function sends the character string stored in memory at the location given by DE to the console device, until a dollar sign ("$") is encountered in the string. Function 110 can change the delimiter for Function 9. However, the delimiter is initialized to $ when a program begins execution. Tabs are expanded as in Function 2, and checks are made for start/stop scroll and printer echo (see Function 109).

```
        BDOS Function 10: Read Console Buffer
        Entry parameters:
              Register C: 0AH
             Register DE: Buffer address
        Returned   value: Console characters in buffer
```

The READ CONSOLE BUFFER functions reads a line of edited console input into a buffer addressed by register pair DE. Console input is terminated when either input buffer overflows, or a Carriage Return or Line Feed is typed. Function 10 takes the following form, where MX is the maximum number of characters that the buffer will hold (1 to 255) and NC is the number of characters read (set by FDOS upon return), followed by the characters read from the console.

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

If NC < MX, then uninitialized positions follow the last character, denoted by 2 question marks ("??") in the above figure. A number of control functions, summarized in Table 2-4, are recognized during line editing. Note that, if a Ctrl-P is encountered by Function 10, it toggles printer echo.

Table 2-4. Edit control character

Character	Edit control function
DEL	Removes and echoes the last character
Ctrl-C	Reboots when at the beginning of line
Ctrl-E	Causes physical end of line
Ctrl-H	Backspaces one character position
Ctrl-J	(Line Feed) Terminates input line
Ctrl-M	(Carriage Return) Terminates input line
Ctrl-R	Retypes the current line after new line
Ctrl-U	Removes current line
Ctrl-X	(Same as Ctrl-U)

```
        BDOS Function 11: Get Console Status
        Entry parameters:
              Register C: 0BH
        Returned   value:
              Register A: Console status
                          00H = no character ready
                          0FFH = character ready
```

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

```
        BDOS Function 12: Return Version Number
        Entry parameters:
              Register C: 0CH
        Returned   value:
             Register HL: Version number (0028H)
```

The RETURN VERSION NUMBER function provides information that allows version-independent programming. A 2-byte value is returned, with H = 00H designating the 8-bit CP/M operating system. BDOS version numbers 20H through 27H are designated for all previous CP/M-80 versions. Personal CP/M returns a hexadecimal 28 in register L. Function 12 is useful for writing application programs that must run on multiple CP/M versions.

```
        BDOS Function 13: Reset Disk System
        Entry parameters:
              Register C: 0DH
        Returned   value: None
```

The RESET DISK SYSTEM function is used to restore the file system to a reset state where all disks are set to Read/Write (see Function 28 and 29). Disk drive A is selected, and the default DMA address is reset to 0080H. This function can be used, for example, by an application program that requires a disk change without a system reboot.

```
        BDOS Function 14: Specify Disk Drive
        Entry parameters:
              Register C: 0EH
              Register E: Specified disk drive number
        Returned   value:
              Register A: Error flag
                          00H if successful, or
                          0FFH if failed
```

```
               Register H: Physical error
```

The SPECIFY DISK DRIVE function designates the disk drive numbered in register E as the default disk for subsequent file operations, with E = 0 for drive A, 1 for drive B, and so on through 15 corresponding to drive P in a full 16-drive system. The drive is placed in an on-line status, which activates its directory until the next cold start, warm start, or RESET DISK SYSTEM operation. FCBs that specify drive code zero (DR = 00H) automatically reference the currently-specified default drive. Drive code values between 1 and 16 ignore the specified default drive, and directly reference drives A through P.

Upon return, register A contains a zero if the SPECIFY DISK DRIVE operation was successful. If a physical error was encountered, the SPECIFY DISK DRIVE function performs different actions, depending on the BDOS error mode (see Function 45). If the BDOS error mode is in the default mode, a message identifying the error is displayed at the console (see Appendix A, "BDOS error handling"), and the calling program is terminated. Otherwise, the SPECIFY DISK DRIVE function returns to the calling program, with register A set to 0FFH, and register H set to one of the following physical error codes:

```
        01: Disk I/O error
        04: Invalid drive
```

```
        BDOS Function 15: Open File
        Entry parameters:
              Register C: 0FH
             Register DE: FCB address
        Returned   value:
              Register A: 00H if successful, or
                          0FFH if failed
              Register H: 00H if successful, or
                          Physical error (see below)
```

The OPEN FILE function is used to activate a file that currently exists in the disk directory for the currently active user number. The FDOS scans the referenced disk directory for a match in positions 1 through 14 of the FCB referenced by DE (byte S2 is automatically zeroed). Normally, bytes EX and S1 of the FCB are zero.

If a directory element is matched, the relevant directory information is copied into bytes D0 through Dn of the FCB, thus allowing access to the files through subsequent read and write operations. An existing file must not be accessed until a successful OPEN FILE operation is completed. Upon return, the OPEN FILE function returns a directory code with the value 00H if the operation was successful, or 0FFH (logical FALSE) if the file cannot be found. If question marks occur in the FCB, the first matching FCB is activated. The current field (CR) must be zeroed by the program if the file is to be accessed sequentially from the first record.

Function 15 opens a file under user 0 when the current user number is non-zero, if 2 conditions exist:

```
        1) the file is not present under the current user number, and
        2) the file under user 0 has the system attribute T2' set.
```

However, files opened in this way cannot be written to. (See Function 32 and Appendix C, "User number conventions", for more discussion of user numbers.)

Upon return, the OPEN FILE function returns a 00H in register A if the open was successful, or 0FFH (logical FALSE) if the file was not found. Register H is set to zero in both cases. If a physical error was encountered, the OPEN FILE function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is in the default mode, a message identifying the error is displayed at the console (see Appendix B, "BDOS error handling"), and the program is terminated. Otherwise, the OPEN FILE function returns to the calling program with register A set to 0FFH, and register H set to one of the following physical error codes:

```
        01: Disk I/O error
        04: Invalid drive error
```

```
        BDOS Function 16: Close File
        Entry parameters:
              Register C: 10H
             Register DE: FCB address
        Returned   value:
              Register A: 00H if successful, or
                          0FFH if failed
             Register H: 00H if successful, or
                         Physical error (see below)
```

The CLOSE FILE function is the reverse of the OPEN FILE function. Given that the FCB addressed by register pair DE has been previously activated through an OPEN FILE or MAKE FILE function, the CLOSE FILE function permanently records the new FCB in the reference disk directory (see Functions 15 and 22). The FCB matching process for the CLOSE FILE function is identical to the OPEN FILE function. The directory code returned for a successful CLOSE FILE operation is 00H, while a 0FFH (logical FALSE) is returned if the filename cannot be found in the directory. If write operations have occurred, the CLOSE FILE operation is necessary, to record the new directory information permanently.

Upon return, the CLOSE FILE function returns a 00H in register A if the close was successful, or 0FFH (logical FALSE) if the file was not found. Register H is set to zero in both cases. If a physical error was encountered, the CLOSE FILE function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is in the default mode, a message identifying the error is displayed at the console (see Appendix B, "BDOS error handling"), and the program is terminated. Otherwise, the CLOSE FILE function returns to the calling program with register A set to 0FFH, and register H set to one of the following physical error codes:

```
        01: Disk I/O error
        02: Read/Only disk
        04: Invalid drive error
```

```
        BDOS Function 17: Search For First
        Entry parameters:
              Register C: 11H
             Register DE: FCB address
        Returned   value:
              Register A: Directory code
                          00-03H if successful, or
                          0FFH if failed
              Register H: 00H if successful, or
                          Physical error (see below)
```

SEARCH FOR FIRST scans the directory for a match with the file given by the FCB addressed by DE. The value 0FFH (logical FALSE) is returned if the file is not found; otherwise, 0, 1, 2, or 3 is returned, indicating that the file is present. When the file is found, the current DMA address is filled with the record containing the directory entry, and the relative starting position is A*32 (that is to say: rotate the A register left 5 bits, or ADD A five times). Although not normally required for application programs, the directory information can be extracted from the buffer at this position. Byte 0 of a returned directory entry contains the file's user number.

An ASCII question mark (63 decimal, 3H hexadecimal) in any position from F1 through EX matches the corresponding field of any directory entry on the default or auto-specified disk drive. If the DR field contains an ASCII question mark, the auto disk specify function is disabled and the default disk is searched, with the SEARCH FOR FIRST function returning any matched entry, allocated or free, belonging to any user number. This latter function is not normally used by application programs, but it allows complete flexibility to scan all current directory values. If the DR field is not an ASCII question mark, the S2 byte is automatically zeroed.

If a physical error is encountered, the SEARCH FOR FIRST function performs actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is in the default mode, a message identifying the error is displayed at the console, and the calling program is terminated. Otherwise, the SEARCH FOR FIRST function returns to the calling program with register A set to 0FFH (logical FALSE) and register H set to one of the following physical error codes:

```
        01: Disk I/O error
        04: Invalid drive error
```

```
        BDOS Function 18: Search For Next
        Entry parameters:
              Register C: 12H
        Returned   value:
              Register A: Directory code
                          00-03H if successful, or
                          0FFH if failed
              Register H: 00H if successful, or
                          Physical error (see Function 17)
```

The SEARCH FOR NEXT function is similar to the SEARCH FOR FIRST function, except that the directory scan continues from the last matched entry. Similar to Function 17, Function 18 returns the value 0FFH (logical FALSE) in register A when no more directory items match.

```
        BDOS Function 19: Delete File
        Entry parameters:
              Register C: 13H
             Register DE: FCB address
        Returned   value:
              Register A: 00H if successful, or
                          0FFH if failed
              Register H: 00H if successful, or
                          Physical error (see below)
```

The DELETE FILE functions deletes files that match the FCB addressed by register pair DE from the directory. The filename and type may contain ambiguous references (that is to say: question marks in various positions), but the drive specifier cannot be ambiguous, as in the SEARCH FOR FIRST and SEARCH FOR NEXT functions. For files that have question marks (ambiguous deletes) in the filename and/or filetype, no files are deleted if any of the files are marked Read-Only.

Upon return, the DELETE FILE function returns a 00H in register A if successful, or 0FFH (logical FALSE) if no file that matches the referenced FCB is found. Register H is set to zero in both cases. If a physical error is encountered, the DELETE FILE function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is the default mode, a message identifying the error is displayed at the console, and the calling program is terminated. Otherwise, the DELETE FILE function returns to the calling program with register A set to 0FFH (logical FALSE) and register H set to one of the following physical error codes:

```
        01: Disk I/O error
        02: Read-Only disk
        03: Read-Only file
        04: Invalid drive error
```

```
        BDOS Function 20: Read Sequential
        Entry parameters:
              Register C: 14H
             Register DE: FCB address
        Returned   value:
              Register A: Error code
                          00H if successful, or
                          01H, 0AH, or 0FFH if failed
              Register H: 00H if successful, or
                          Physical error if failed
```

Given that the FCB addressed by register pair DE has been activated through an OPEN FILE or MAKE FILE function, the READ SEQUENTIAL function reads the next 128-byte record from the file into memory at the current DMA address. The record is read from position CR of the extent, and the CR field is automatically incremented to the next record position. If the CR field overflows, the next logical extent is automatically opened, and the CR field is reset to zero in preparation for the next read operation.

Upon return, the READ SEQUENTIAL function sets register A to 00H if the read operation is successful. Otherwise, register A contains an error code identifying the error as shown:

```
        01H: Reading unwritten data (end-of-file)
        0AH: Media change occurred
        0FFH: Physical error, refer to register H
```

Error code 01H is returned if no data exists at the next record position of the file. Usually, the no-data situation is encountered at the end of a file. However, it can also occur if an attempt is made to read a data block that has not been created. These situations are usually restricted to files created or appended with the BDOS random write functions (see Functions 34 and 40).

Error code 0AH is returned if a media change occurs on the drive after the referenced FCB is activated by a OPEN FILE or MAKE FILE function.

Error code 0FFH is returned if a physical error is encountered and the BDOS error mode is "return error" mode, or "return and display error" mode (see Function 45). If the BDOS error mode is the default mode, a message identifying the physical error is displayed at the console, and the calling program is terminated. When a physical error is returned to the calling program, register H contains one of the following error codes:

```
        01: Disk I/O error
        04: Invalid drive error
```

```
        BDOS Function 21: Write Sequential
        Entry parameters:
              Register C: 15H
             Register DE: FCB address
        Returned   value:
              Register A: Error code
                          00H if successful, or
                          01H, 02H, 0AH, or 0FFH if failed
              Register H: 00H if successful, or
                          Physical error if failed
```

Given that the FCB addressed by register pair DE has been activated through an OPEN FILE or MAKE FILE function, the WRITE SEQUENTIAL function writes the 128-byte data record at the current DMA address to the file named by the FCB. The record is placed at position CR of the file, and the CR field is automatically incremented to the next record position. If the CR field overflows, the next logical extent is automatically opened, and the CR field is reset to zero in preparation for the next WRITE SEQUENTIAL operation. WRITE SEQUENTIAL operations take place into an existing file, in which case newly written records overlay those already existing in the file.

Upon return, the WRITE SEQUENTIAL function sets register A to 00H if the operation is successful. Otherwise, register A contains an error code identifying the error as shown below:

```
        01H: No available directory space
        02H: No available data block
        0AH: Media change occurred
        0FFH: Physical error, refer to register H
```

Error code 01H is returned when the WRITE SEQUENTIAL function attempts to create a new extent that requires a new directory entry, and no available directory entries exist on the selected disk drive.

Error code 02H is returned when the WRITE SEQUENTIAL function attempts to allocate a new data block to the file, and no unallocated data blocks exist on the selected disk drive.

Error code 0AH is returned if a media change occurs on the drive after the referenced FCB is activated by a OPEN FILE or MAKE FILE function.

Error code 0FFH is returned if a physical error is encountered and the BDOS error mode is "return error" mode, or "return and display error" mode (see Function 45). If the error mode is the default mode, a message identifying the physical error is displayed at the console, and the calling program is terminated. When a physical error is returned to the calling program, register H contains one of the following error codes:

```
        01: Disk I/O error
        02: Read-Only disk
        03: Read-Only file, or
            File open from user zero when the current user number is non-zero
        04: Invalid drive error
```

```
        BDOS Function 22: Make File
        Entry parameters:
              Register C: 16H
             Register DE: FCB address
        Returned   value:
              Register A: 00H if successful, or
                          0FFH if failed
              Register H: 00H if successful, or
                          Physical error (see below)
```

The MAKE FILE function is similar to the OPEN FILE function, except that the FCB must name a file that does not exist in the currently-referenced disk directory (that is to say: the one specifed explicitly by a non-zero DR code, or the default disk if DR is zero). The FDOS creates the file and initializes both the directory and main memory value to an empty file. As the programmer, you must ensure that no duplicate filenames occur, and a preceding DELETE FILE operation is sufficient if there are any possibility of duplication. The MAKE FILE function has the side effect of activating the FCB, and thus a subsequent OPEN FILE is not necessary.

Upon return, the MAKE FILE function returns a 00H in register A if the operation is successful, or 0FFH (logical FALSE) if no directory space is available. Register H is set to 00H in both of these cases. If a physical error is encountered, the MAKE FILE function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is the default error mode, a message identifying the error is displayed at the console, and the calling program is terminated. Otherwise, the MAKE FILE function returns to the calling program with register A set to 0FFH, and register H set to one of the following physical error codes:

```
        01: Disk I/O error
        02: Read-Only disk
        04: Invalid drive error
```

```
        BDOS Function 23: Rename File
        Entry parameters:
              Register C: 17H
             Register DE: FCB address
        Returned   value:
              Register A: 00H if successful, or
                          0FFH if failed
              Register H: 00H if successful, or
                          Physical error (see below)
```

The RENAME FILE function uses the FCB addressed by register pair DE to change the file named in the first 16 bytes, to the file named in the second 16 bytes. The drive code DR at position 0 is used to specify the drive, while the drive code for the new filename at position 16 of the FCB is assumed to be zero.

Upon return, the RENAME FILE function returns a 00H in register A if the operation is successful, or 0FFH (logical FALSE) if the file named by the first filename in the FCB is not found. Register H is set to 00H in both cases. If a physical error is encountered, the RENAME FILE function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is the default mode, a message identifying the error is displayed at the console, and the program is terminated. Otherwise, the RENAME FILE function returns to the calling program with register A set to 0FFH and register H set to one of the following physical error codes:

```
        01: Disk I/O error
        02: Read-Only disk
        03: Read-Only file
        04: Invalid drive error
```

```
        BDOS Function 24: Return Login Vector
        Entry parameters:
              Register C: 18H
        Returned   value:
             Register HL: Login vector
```

The login vector value returned by Personal CP/M is a 16-bit value in register pair HL, where the least significant bit of L corresponds to the first drive A, and the high-order bit of H corresponds to the 16th drive, P. A 0 bit indicates that the drive is not on-line, while a 1 bit marks a drive that is actively on-line as a result of an explicit disk drive specification, or an implicit drive spec caused by a file operation that specified a non-zero DR field. The user should note that compatibility is maintained with previous CP/M versions, because register A and L contain the same value upon return.

```
        BDOS Function 25: Return Current Drive
        Entry parameters:
              Register C: 19H
        Returned   value:
              Register A: Current drive number
```

The RETURN CURRENT DRIVE function returns the currently-specified default drive number in register A. The drive numbers range from 0 through 15, corresponding to drives A through P.

```
        BDOS Function 26: Set DMA Address
        Entry parameters:
              Register C: 1AH
             Register DE: DMA address
        Returned   value: None
```

DMA is an acronym for "Direct Memory Access", which is often used in connection with disk controllers that directly access the memory of the mainframe computer to transfer data to and from the disk subsystem. Many computer systems use non-DMA access (that is to say: the data is transferred through programmed I/O operations). In Personal CP/M, the DMA address means the address at which the 128-byte data record resides before a disk write, and after a disk read. Upon cold start, warm start, or RESET DISK SYSTEM, the DMA address is automatically set to 0080H. The SET DMA ADDRESS function can be used to change this default value to address another area of memory where the data records reside. Thus, the DMA address becomes the value specified by register pair DE until it is changed by a subsequent SET DMA ADDRESS function, cold start, warm start, or RESET DISK SYSTEM.

```
        BDOS Function 27: Get Addr (Alloc)
        Entry parameters:
              Register C: 1BH
        Returned   value:
             Register HL: Alloc address if successful, or
                          0FFFFH if failed
```

An allocation vector is maintained in main memory for each on-line disk drive. Various system programs use the information provided by the allocation vector to determine the amount of remaining storage (see the STAT program). The GET ADDR (ALLOC) function returns the base address of the allocation vector for the currently-specified drive. However, the allocation information might be invalid if the specified drive has been marked Read-Only. Although this function is not normally used by application programs, additional details of the allocation vector are found in the "Personal CP/M System Guide".

If a physical error is encountered when the BDOS error mode is one of the return modes (see Function 45), the GET ADDR (ALLOC) function returns the value 0FFFFH in register pair HL.

```
        BDOS Function 28: Write Protect Disk
        Entry parameters:
              Register C: 1CH
        Returned   value: None
```

The WRITE PROTECT DISK function provides temporary write protection for the currently-selected disk. A Read-Only disk stays Read-Only until reset by Function 13 or 37. Any attempt to write to a Read-Only disk produces the message:

```
        CP/M Error on x: Read-Only Disk
```

or returns a physical error 2 if in BDOS "return error" mode (see Function 45).

```
        BDOS Function 29: Get Read-Only Vector
        Entry parameters:
              Register C: 1DH
        Returned   value:
             Register HL: Read-Only vector
```

The GET READ-ONLY VECTOR function returns a 16-bit value in register pair HL which indicates drives that have the temporary Read-Only bit set. As in Function 24, the least significant bit corresponds to drive A, while the most significant bit corresponds to drive P. The Read-Only bit can only be set by an explicit call to Function 28.

```
        BDOS Function 30: Set File Attributes
        Entry parameters:
              Register C: 1EH
             Register DE: FCB address
        Returned   value:
              Register A: 00H if successful, or
                          0FFH if failed
              Register H: 00H if successful, or
                          Physical error (see below)
```

The SET FILE ATTRIBUTES function allows the permanent attributes attached to files to be modified by a program. In particular, the Read-Only and system attributes (T1' and T2') can be set or reset. The DE register pair addresses an unambiguous filename with the appropriate attributes set or reset. The SET FILE ATTRIBUTES function searches for a match, and changes the matched directory entry to contain the selected attributes. Attributes F1' through F8' are not currently used in Personal CP/M. Attributes F1' through F4' may be used by application programs, since they are not involved in the matching process during OPEN FILE and CLOSE FILE operations. Attributes F5' through F8' and T3' are reserved.

Upon return, the SET FILE ATTRIBUTES function returns a 00H in register A if the function is successful, or 0FFH (logical FALSE) if the file specified by the referenced FCB is not found. Register H is set to 00H in both cases. If a physical error is encountered, the SET FILE ATTRIBUTES function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is the default mode, a message identifying the error is displayed at the console, and the program is terminated. Otherwise, the SET FILE ATTRIBUTES function returns to the calling program with register A set to 0FFH, and register H set to one of the following physical error codes:

```
        01: Disk I/O error
        02: Read-Only disk
        04: Invalid drive error
```

```
        BDOS Function 31: Get Addr (Disk Parms)
        Entry parameters:
              Register C: 1FH
        Returned   value:
             Register HL: DPB address if successful, or
                          0FFFFH if failed
```

The GET ADDR (DISK PARMS) function returns the address of the BIOS-resident Disk Parameter Block in register pair HL. This address can be used for either of 2 purposes. First, the disk parameter values can be extracted for display and space computation purposes, or transient programs can dynamically change the values of current disk parameters when the disk environment changes, if required. Normally, application programs do not require this facility.

If a physical error is encountered when the BDOS error mode is one of the return modes (see Function 45), the GET ADDR (DISK PARMS) function returns the value 0FFFFH in register pair HL.

```
        BDOS Function 32: Get/Set User Number
        Entry parameters:
              Register C: 20H
              Register E: 0FFH (Get), or
                          User number (Set)
        Returned   value:
              Register A: Current user number (Get)
```

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

```
        BDOS Function 33: Read Random
        Entry parameters:
              Register C: 21H
             Register DE: FCB address
        Returned   value:
              Register A: Error code
                          00H if successful, or
                          non-zero if failed (see below)
              Register H: 00H if successful, or
                          Physical error (see below)
```

The READ RANDOM function is similar to the READ SEQUENTIAL operation of previous CP/M versions, except that the read operation takes place at a particular record number, selected by the 24-bit value constructed from the 3-byte field following the FCB (byte positions R0 at 33, R1 at 34, and R2 at 35). You should note that the sequence of 24 bits is stored with least significant byte first (R0), middle byte next (R1), and high byte last (R2). Personal CP/M does not reference byte R2, except in computing the size of a file (Function 35). Byte R2 must be zero, however, since a non-zero value indicates overflow past the end of file.

Thus, the R0, R1 byte pair is treated as a double byte, or word, value that contains the record to read. This value ranges from 0 to 65,535, providing access to any particular record of the 8-megabyte file. To process a file using random access, the base extent (extent 0) must first be opened. Although the base extent may or may not contain any allocated data, this step ensures that the file is properly recorded in the directory, and is visible in DIR requests. The selected record number is then stored in the random record field (R0, R1), and the BDOS is called to read the record.

Upon return from the call, register A either contains an error code, as listed below, or the value 00H indicating that the operation was successful. In the latter case, the current DMA address contains the randomly-accessed record. Note that, contrary to the READ SEQUENTIAL operation, the record number is not advanced. Thus, subsequent READ RANDOM operations continue to read the same record.

Upon each READ RANDOM operation, the logical extent and current record values are automatically set. Thus, the file can be sequentially read or written, starting from the current randomly-accessed position. In this case, the last randomly read record will be re-read as one switches from random mode to SEQUENTIAL READ, and the last record will be re-written as one switches to a SEQUENTIAL WRITE operation. The user can advance the random record position following each READ RANDOM or WRITE RANDOM, to obtain the effect of sequential I/O operation.

Upon return, the READ RANDOM function sets register A to 00H if the read operation was successful. Otherwise, register A contains one of the following error codes:

```
        01: Reading unwritten data (end of file)
        03: Cannot close current extent
        04: Seek to unwritten extent
        06: Random record number out of range
        0AH: Media change occurred
        0FFH: Physical error, refer to register H
```

Error code 01 is returned if no data exists at the next record position of the file. Usually, the no-data situation is encountered at the end of a file. However, it can also occur if an attempt is made to read a data block that has not been previously written.

Error code 03 is returned when the READ RANDOM function cannot close the current extent prior to moving to a new extent.

Error code 04 is returned when a READ RANDOM operation accesses an extent that has not been created.

Error code 06 is returned when byte 35, R2, of the referenced FCB is non-zero.

Error code 0AH is returned if a media change occurs on the drive after the referenced FCB is activated by a OPEN FILE or MAKE FILE function call.

Error code 0FFH is returned if a physical error is encountered and the BDOS error mode is one of the return modes (see Function 45). If the BDOS error mode is in the default mode, a message identifying the physical error is displayed at the console, and the calling program is terminated. When a physical error is returned to the calling program, register H contains one of the following physical error codes:

```
        01: Disk I/O error
        04: Invalid drive error
```

```
        BDOS Function 34: Write Random
        Entry parameters:
              Register C: 22H
             Register DE: FCB address
        Returned   value:
              Register A: Error code
                          00H if successful, or
                          non-zero if failed (see below)
              Register H: 00H if successful, or
                          Physical error (see below)
```

The WRITE RANDOM function is initiated similarly to the READ RANDOM operation, except that data is written to the disk from the current DMA address. If the disk extent or data block that is the target of the WRITE RANDOM operation has not yet been allocated, the allocation is performed before the operation continues. As in the READ RANDOM operation, the random record number is not changed as a result of the WRITE RANDOM function. The logical extent number and current record positions of the FCB are set to correspond to the random record being written. Again, READ SEQUENTIAL or WRITE SEQUENTIAL operations can begin following a WRITE RANDOM operation, remembering that the currently-addressed record is either read or rewritten again, as the sequential operation begins. You can also advance the random record position following each WRITE RANDOM operation, to get the effect of a WRITE SEQUENTIAL operation. Note that reading or writing the last record of an extent in random mode does not cause an automatic extent switch, as it does in sequential mode.

Upon return, the WRITE RANDOM function sets register A to 00H if the operation is successful. Otherwise, register A contains one of the following error codes:

```
        02: No available data block
        03: Cannot close current extent
        05: No available directory space
        06: Random record number out of range
        0AH: Media change occurred
        0FFH: Physical error, refer to register H
```

Error code 02 is returned when the WRITE RANDOM function attempts to allocate a new data block to the file, and no unallocated data blocks exist on the selected disk drive.

Error code 03 is returned when the WRITE RANDOM function cannot close the current extent prior to moving to a new extent.

Error code 05 is returned when the WRITE RANDOM function attempts to create a new extent that requires a new directory entry, and no available directory entries exist on the selected disk drive.

Error code 06 is returned when byte 35, R2, of the referenced FCB is non-zero.

Error code 0AH is returned if a media change occurs on the drive after the referenced FCB is activated by an OPEN FILE or MAKE FILE operation.

Error code 0FFH is returned if a physical error is encountered and the BDOS error mode is one of the return modes (see Function 45). If the BDOS error mode is the default mode, a message identifying the physical error is displayed at the console, and the calling program is terminated. When a physical error is returned to the calling program, register H contains one of the following error codes:

```
        01: Disk I/O error
        02: Read-Only disk
        03: Read-Only file, or
            File opened from user zero when current user number is non-zero
        04: Invalid drive error
```

```
        BDOS Function 35: Compute File Size
        Entry parameters:
              Register C: 23H
             Register DE: FCB address
        Returned   value: Random record number set
```

When computing the size of a file, the register pair DE addresses an FCB in random mode format (bytes R0, R1, and R2 are present). The FCB contains an unambiguous filename that is used in the directory scan. Upon return, the random record bytes contain the virtual file size, which is, in effect, the record address of the record following the end of the file. Following a call to the COMPUTE FILE SIZE function, if the high record byte R2 is 01H, the file contains the maximum record count 65,536. Otherwise, bytes R0 and R1 constitutes a 16-bit value as before (R0 is the least significant byte), which is the file size.

Data can be appended to the end of an existing file by calling the COMPUTE FILE SIZE function to set the random record position to the end of the file, and then performing a sequence of WRITE RANDOM operations, starting at the preset record address.

The virtual size of a file corresponds to the physical size when the file is written sequentially. If the file was created in random mode, and holes exist in the allocation, the file might contain fewer records than the size indicated. For example, if only the last record of an 8-megabyte file is written in random mode (that is to say: record number 65,535), the virtual size is 65,536 record, although only one block of data is actually allocated.

Note: The BDOS does not require that the file be open to use Function 35. However, if the file has been written to, it must be closed before calling Function 35. Otherwise, an incorrect file size might be returned.

Upon return, the COMPUTE FILE SIZE function returns a 00H in register A if the file specified by the referenced FCB is found, or an 0FFH (logical FALSE) in register A if the file is not found. Register H is set to 00H in both cases. If a physical error is encountered, the COMPUTE FILE SIZE function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is the default mode, a message identifying the error is displayed at the console, and the program is terminated. Otherwise, the COMPUTE FILE SIZE function returns to the calling program with register A set to 0FFH, and register H set to one of the following physical error codes:

```
        01: Disk I/O error
        04: Invalid drive error
```

```
        BDOS Function 36: Set Random Record
        Entry parameters:
              Register C: 24H
             Register DE: FCB address
        Returned   value: Random record number set
```

The SET RANDOM RECORD function returns the random record number of the next record to be accessed from a file that has been read or written sequentially to a particular point. This value is returned in the random record number, bytes R0, R1, and R2 of the FCB addressed by register pair DE. The SET RANDOM RECORD function can be useful in 2 ways.

First, it is often necessary to read and scan a sequential file, to extract the positions of various key fields. As each key is encountered, the SET RANDOM RECORD function is called to compute the random record position for the data corresponding to this key. If the data unit size is 128 bytes, the resulting record number minus one is placed into a table, with the key, for later retrieval. After scanning the entire file, and tabularizing the keys and their record numbers, you can move directly to a particular record by performing a RANDOM RECORD operation using the corresponding random record number that you saved earlier. The scheme is easily generalized when variable record lengths are involved, because the program need only store the buffer-relative byte position, along with the key and record number, to find the exact starting position of the keyed data at a later time.

A second use of the SET RANDOM RECORD function occurs when switching from a SEQUENTIAL READ or SEQUENTIAL WRITE operation over to a RANDOM READ or RANDOM WRITE operation. A file is sequentially accessed to a particular point in the file; then the SET RANDOM RECORD function is called to set the record number, and subsequent RANDOM READ and RANDOM WRITE operations continue from the next record in the file.

```
        BDOS Function 37: Reset Drive
        Entry parameters:
              Register C: 25H
             Register DE: Drive vector
        Returned   value: None
```

The RESET DRIVE function programmatically restores specified drives to the reset state. A reset drive is not logged in, and is in Read/Write status. The passed parameter in register pair DE is a 16-bit vector of drives to be reset, where the least significant bit corresponds to the first drive A, and the most significant bit corresponds to drive P. Bit values of 1 indicate that the specified drive is to be reset.

```
        BDOS Function 40: Write Random with Zero Fill
        Entry parameters:
              Register C: 28H
             Register DE: FCB address
        Returned   value:
              Register A: Error code
                          00H if successful, or
                          non-zero if failed (see Function 34)
              Register H: 00H if successful, or
                          Physical error (see Function 34)
```

The WRITE RANDOM WITH ZERO FILL function is similar to the WRITE RANDOM function, except that a previously-unallocated block is filled with 00H before the data is written.

```
        BDOS Function 45: Set BDOS Error Mode
        Entry parameters:
              Register C: 2DH
              Register E: BDOS error mode
                          0FFH: Return error mode
                          0FEH: Return and display mode
                          Any other: Default mode
        Returned   value: None
```

The SET BDOS ERROR MODE function sets the BDOS error mode for the calling program to the mode specified in register E. If register E is set to 0FFH, the BDOS error mode is set to "return error" mode. If register E is set to 0FEH, the BDOS error mode is set to "return and display" mode. If register E is set to any other value, the BDOS error mode is set to the "default" mode.

The SET BDOS ERROR MODE function determines how physical errors are handled for a program. The operation can exist in 3 modes: the default mode, the return error mode, and the return and display mode. In the "default" mode, the BDOS displays a system message at the console that identifies the error, and terminates the calling program. In the return modes, the BDOS sets register A to 0FFH (logical FALSE), places an error code that identifies the physical error in register H, and returns to the calling program. In "return and display" mode, the BDOS displays the system message before returning to the calling program. No system messages are displayed, however, when the BDOS is in "return error" mode.

Table 2-5 lists the physical error codes and their corresponding CP/M error messages. See Appendix B for more information on BDOS error handling.

Table 2-5. Messages for physical errors returned

Physical error codes returned	Corresponding Personal CP/M messages
1	Disk I/O error
2	Read-Only disk
3	Read-Only file
4	Invalid drive error

```
        BDOS Function 48: Flush Buffers
        Entry parameters:
              Register C: 30H
        Returned   value:
              Register A: 00H if successful, or
                          Error flag if failed
              Register H: Physical error
```

The FLUSH BUFFERS function forces the write of any write-pending records contained in internal blocking/deblocking buffers.

Upon return, register A is set to 00H if the operation is successful. If a physical error is encountered, the FLUSH BUFFERS function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is in the default mode, a message identifying the error is displayed at the console, and the calling program is terminated. Otherwise, the FLUSH BUFFERS function returns to the calling program with register A set to 0FFH, and register H set to one of the following physical error codes:

```
        01: Disk I/O error
        02: Read-Only disk
        04: Invalid drive error
```

```
        BDOS Function 109: Get/Set Console Mode
        Entry parameters:
              Register C: 6DH
             Register DE: 0FFFFH (Get), or
                          Console mode (Set)
        Returned   value:
             Register HL: Console mode (Get)
```

A program can set or interrogate the console mode by calling the GET/SET CONSOLE MODE function. If register pair DE = 0FFFFH, the current console mode is returned in register pair HL. Otherwise, the GET/SET CONSOLE MODE function sets the console mode to the value contained in register pair DE.

The console mode is a 16-bit system parameter that determines the action of certain BDOS console I/O functions. The definition of the bits of the console mode is described in Table 2-6.

Table 2-6. Get/Set Console Mode description

Bits	Definition
0-3	Reserved
4=0	Enable tab expansion, printer echo, and Ctrl-S checking for Functions 2, 9, and 111.
4=1	Raw console output. Disable tab expansion, printer echo, and Ctrl-S checking.
5-15	Reserved

Note that the console mode bits are numbered from right to left. The CCP initializes the console mode to zero when it loads a program.

```
        BDOS Function 110: Get/Set Output Delimiter
        Entry parameters:
              Register C: 6EH
             Register DE: 0FFFFH (Get), or
                          Output delimiter (Set)
        Returned   value:
              Register A: Output delimiter (Get)
```

A program can set or interrogate the current output delimiter by calling the GET/SET OUTPUT DELIMITER function. If register pair DE = 0FFFFH, the current output delimiter is returned in register A. Otherwise, the GET/SET OUTPUT DELIMITER function sets the output delimiter to the value contained in register E. The GET/SET OUTPUT DELIMITER function sets the string delimiter for Function 9, PRINT STRING. The default delimiter value is a dollar sign ("$"). The CCP sets the output delimiter to the default value when a transient program is loaded.

```
        BDOS Function 111: Print Block
        Entry parameters:
              Register C: 6FH
             Register DE: CCB address
        Returned   value: None
```

The PRINT BLOCK functions sends the character string located by the Character Control Block, CCB, addressed in register pair DE to the logical console ("CONOUT:"). If the console mode is in the default mode (zero), the PRINT BLOCK function expands tab characters ("Ctrl-I") in columns of 8 characters. It also checks for Ctrl-S (start/stop scroll), and echoes to the logical list device ("LST:") if printer echo ("Ctrl-P") has been invoked. The CCB format is as follows:

```
        Byte 0-1: Address of character string (word value)
        Byte 2-3: Length  of character string (word value)
```

```
        BDOS Function 112: List Block
        Entry parameters:
              Register C: 70H
             Register DE: CCB address
        Returned   value: None
```

The LIST BLOCK function sends the character string located by the Character Control Block, CCB, addressed in register pair DE to the logical list device ("LST:"). The CCB format is as follows:

```
        Byte 0-1: Address of character string (word value)
        Byte 2-3: Length  of character string (word value)
```

```
        BDOS Function 113: Direct Screen Functions
        Entry parameters:
              Register C: 71H
             Register DE: SFB address
                          Byte 0  : Subfunction number
                          Byte 1-2: Address of extended information
                       or
                          Byte 0  : Column value
                          Byte 1  : Row value
        Returned   value: None
```

The DIRECT SCREEN FUNCTIONS function provides direct access to cursor movement and screen editing functions typically used by video-intensive applications, such as word-processing and spreadsheets. The direct access is important primarily on computers systems with memory-mapped displays. While most BIOS display drivers provide some terminal emulation for these functions, the overhead involved in interpreting an ESCape sequence into the corresponding screen function can slow a video-intensive application to an unacceptable degree. The DIRECT SCREEN FUNCTIONS function not only allows direct access to these screen functions, but also can return information to the calling program about whether a specific function executes fast or slowly on a particular system (see Subfunction 1). Table 2-7 lists the subfunctions for the DIRECT SCREEN FUNCTIONS function.

The Screen Functions Block (SFB) format is as follows:

```
        Byte 0  : Subfunction number
        Byte 1-2: Address of extended information
```

or
  Byte 0 : Column value
  Byte 1 : Row value

Table 2-7. Subfunctions for Function 113

Format: Subfunction
        Description

0 -- Subfunctions supported
Returned value:
   Register HL: Pointer to bit vector stored as

Byte 0:	07	06	05	04	03	02	01	00
Byte 1:	15	14	13	12	11	10	09	08
Byte 2:	23	22	21	20	19	18	17	16
Byte 3:					27	26	25	24

The corresponding bit is ON if the subfunction is supported.

1 -- Subfunctions emulated
Returned value:
   Register HL: Pointer to bit vector stored as above. Bit is ON if subfunction is emulated (and therefore slower).

2 -- Display size
Returned value:
    Register H: Number of columns (n-1)
    Register L: Number of rows (n-1)

3 -- Identify terminal
Returned value:
   Register HL: Pointer to null-terminated identifier string. For a VT-52 type terminal, it would return a pointer to the byte string ESC, '/', 'K', NULL.

4 -- Cursor up
Move cursor up, but does not scroll screen down if cursor was at top of page.

5 -- Cursor down
Move cursor down, but does not scroll screen up if cursor was at bottom of page.

6 -- Cursor left
Wrap depends on mode set up by Subfunction 26 or 27.

7 -- Cursor right
Wrap depends on mode set by Subfunction 26 or 27.

8 -- Cursor home
Move cursor to top left-hand corner of screen.

9 -- Cursor on
Make cursor visible.

10 -- Cursor off
Make cursor invisible.

11 -- Direct cursor addressing
Move cursor to absolute column and row in SFB.

12 -- Clear display
Move cursor to top left-hand corner of screen, and erase screen.

13 -- Erase to end of line

14 -- Erase to end of screen

15 -- Enter ANSI mode

16 -- Enter VT-52 mode

17 -- Enter graphics mode (*)

18 -- Exit graphics mode (*)

19 -- Enter alternate keypad mode (*)

20 -- Exit alternate keypad mode (*)

(* = Not supported by Personal CP/M for the Sharp MZ-800.)

21 -- Enter hold screen mode

22 -- Exit hold screen mode

23 -- Enter reverse video mode

24 -- Exit reverse video mode

25 -- Reverse line-feed

26 -- Enable wrap-around at end of line

27 -- Truncate characters at end of line

28-255 -- Reserved

(ROCHE>??? The above bit vector supports only 32 Subfunctions, not 255...)

```
        BDOS Function 124: Byte BLT Copy
        Entry parameters:
              Register C: 7CH
             Register DE: BCB address
        Returned   value:
              Register A: 00H if implemented, or
                          0FFH if not implemented
```

ROCHE> BCB = Byte Copy Block. That's all I know...

```
        BDOS Function 125: Byte BLT Alter
        Entry parameters:
              Register C: 7DH
             Register DE: BCB address
        Returned   value:
              Register A: 00H if implemented, or
                          0FFH if not implemented
```

ROCHE> BCB = Byte Copy Block. That's all I know...

Section 3: Sample programs

3.1 Sample file-to-file copy program

This program provides a relatively simple example of file operations. (Refer to the assembler source for the program on the Personal CP/M distribution disk, in file COPY.ASM.) The program source file is created using the CP/M ED program, and then assembled using ASM or MAC, resulting in a HEX file. The LOAD program is used to produce a COPY.COM file, that executes directly under Personal CP/M. The program begins by setting the Stack Pointer of the microprocessor of your microcomputer to a local area, and proceeds to move the second name from the default area at 006CH to a 33-byte file control block called DFCB (Destination File Control Block). The DFCB is then prepared for file operations by clearing the current record field.

At this point, the source and destination FCBs are ready for processing, because the SFCB (Source File Control Block) at 005CH is properly set up by the CCP upon entry to the COPY program. That is to say: the first name is placed into the default FCB, with the proper fields zeroed, including the current record field at 007CH. The program continues by opening the source file, deleting any existing destination file, and creating the destination file. If all this is successful, the program loops at the label COPY until each 128-byte record is read from the source file, and placed into the destination file. When the data transfer is complete, the destination file is closed, and the program returns to the CCP command level by jumping to BOOT.

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

3.2 Sample file dump utility

The file dump program is more complex than the simple copy program. (Refer to the assembler source for the program on the Personal CP/M distribution disk, in file DUMP.ASM.) The dump program reads an input file specified on the command line, and displays the content of each record in hexadecimal format at the console. Note that the dump program saves the CCP's stack upon entry, resets the stack to a local area, and restores the CCP's stack before returning directly to the CCP. Thus, the dump program does not perform a warm start at the end of processing.

3.3 Sample random access program

The random access program presents an extensive example of random access operation. (Refer to the assembler source for the program on the Personal CP/M distribution disk, in file RANDOM.ASM.) The program performs the simple function of reading or writing random records upon command from the terminal. When a program has been created, assembled, and placed into a file labeled RANDOM.COM, the following CCP-level command line starts the sample program:

```
A>random x.dat
```

The RANDOM program looks for a file named X.DAT and, if found, proceed to prompt the console for input. If not found, the file is created before the prompt is given. Each prompt takes the following form, and is followed by operator input, followed by a Carriage Return ("RETURN" key).

```
Next command?
```

The input commands take the following form, where n is an integer value in the range 0 to 65535, and W, R, and Q are simple command characters corresponding to WRITE RANDOM, READ RANDOM, and Quit processing, respectively.

```
nW  nR  Q
```

If the W command is issued, the RANDOM program issues the following prompt:

```
Type data:
```

The operator then responds by typing up to 127 characters, followed by a Carriage Return ("RETURN" key). RANDOM then writes the character string into the X.DAT file at record n. If the R command is issued, RANDOM reads record number n, and displays the character string at the console. If the Q command is issued, the X.DAT file is closed, and the program returns to the CCP. For brevity, the only error message is

```
Error, try again.
```

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

This particular program could be improved to enhance its operation. In fact, the sample random access program could even evolve into a simple data base management system. For example, you could assume a standard record size of 128 bytes, consisting of arbitrary fields within the record. A program, called GETKEY, could be developed that first reads a sequential file, and extracts a specific field defined by the operator. For example, the following command would cause GETKEY to read the data base file NAMES.DAT and extract the LASTNAME filed from each record, starting in position 10, and ending at character 20.

```
A>getkey names.dat lastname 10 20
```

GETKEY builds a table in memory consisting of each particular LASTNAME field, along with its 16-bit record number location within the file. The GETKEY program then sorts this list, and writes a new file, called LASTNAME.KEY, which is an alphabetical list of LASTNAME fields with their corresponding record numbers (also called an inverted index).

If you renamed the program shown as QUERY, and modified it so that it reads a sorted key file into memory, the command line might appear as:

```
        A>query names.dat lastname.key
```

Instead of reading a number, the QUERY program reads an alphanumeric string that is a particular key to find in the NAMES.DAT data base. Because the LASTNAME.KEY list is sorted, one can find a particular entry rapidly by performing a binary search, similar to looking up a name in the telephone book. Starting at both ends of the list, you examine the entry half-way in between and, if not matched, splits either the upper half or the lower half for the next search. You will quickly reach the item you are looking for, and find the corresponding record number. You should fetch and display this record at the console, just as was done in this program.

With more work, you can allow a fixed grouping size that differs from the 128-byte record shown above. Do this by keeping track of the record number and the byte offset within the record. Knowing the group size, you randomly access the record containing the proper group, offset to the beginning of the group within the record, read sequentially until the group size has been exhausted.

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

3.4 Full-duplex terminal emulator

The purpose of this sample program is to show how you can use Function 7 (AUXILIARY INPUT STATUS) and Function 8 (AUXILIARY OUTPUT STATUS). The sample program demonstrates a simple case of a terminal emulator as used in a portable communications program. "Portable", in this case, means a hardware-independent program that can be used with many different kinds of computer systems. (Refer to the assembler source for the program on the Personal CP/M distribution disk, in file TERMINAL.ASM.)

In Figure 3-1, the flowchart shows how this example works. The numbers in parentheses on the flowchart refer to Personal CP/M function calls.

```
        +------------------+
        |                  |
        |                  V
        |       +----------------------+
        |       | Get console char (6) |
        |       +----------------------+
        |                  |
        |                  V
        |   No       -------------
        | +--------< Char present? >
        | |          -------------
        | |                | Yes
        | |                V
        | |          --------------   Yes
        | |        < Char = Ctrl-C? >------> Return to Personal CP/M
        | |          --------------
        | | +-------------->| No
        | | |              V
        | | | No   ------------------
        | | +----< AUXOUT: ready? (8) >
        | |        ------------------
        | |                | Yes
        | |                V
        | |        +---------------+
        | |        | Send char (4) |
        | |        +---------------+
        | +--------------->|
        |                  V
        |  No       -----------------
        +<--------< AUXIN: ready? (7) >
        |           -----------------
        |                  | Yes
        |                  V
        |         +--------------------+
        |         | Get AUX: char (3) |
        |         +--------------------+
        |                  |
        |                  V
        |       +--------------------------+
        |       | Send char to console (6) |
        |       +--------------------------+
        |                  |
        +<-----------------+
```

Figure 3-1. Full-duplex terminal emulator flowchart

AUXILIARY INPUT STATUS (Function 7) and AUXILIARY OUTPUT STATUS (Function 8) allows you to write portable communications programs. You can use Function 7 to check if a character is available on an auxiliary input device. Previously, if AUXILIARY INPUT (Function 3) was invoked and no character was available, the program would stop completely until a character was ready. The other alternative was for the program to go directly to the hardware port to check if a character was ready, thus creating machine-dependent code.

Under Personal CP/M, the program can check the status and, even if no character is available for input from the auxiliary input device, the program can continue to execute. The program in this example alternates between checking the keyboard and the auxiliary input device for available characters. The program can process a character from either the remote computer or the local terminal -- whichever has a character available.

In the loop highlighted on the flowchart, the program asks if a character is available at the first Function 6 (DIRECT CONSOLE I/O) decision box. If the answer is no, the program queries AUXILIARY INPUT STATUS (Function 7) to see if a character is available. If the answer is still no, the program returns to Function 6. The decision loop in this example shows how Function 7 allows the program to continue execution, even though no characters are ready.

3.5 BLTMEMO

ROCHE> Only the name of this sample program is known. Any information welcomed. (What does mean "BLT"? Why MEMO, instead of DEMO?)

Appendix A: System function summary

Table A-1. System function summary

```
Format: Dec, Hex, Name
        Input parameters
        Returned value

  0   0 System reset
        none
        none

  1   1 Console input
        none
        A = ASCII char

  2   2 Console output
        A = ASCII char
        none

  3   3 Auxiliary input
        none
        A = ASCII char

  4   4 Auxiliary output
        E = ASCII char
        none

  5   5 List output
        E = ASCII char
        none

  6   6 Direct console I/O
        E = 0FFH/0FEH/char
        A = char/status/none
```

```
     7   7 Auxiliary input status
           none
           A = AUXIN: status

     8   8 Auxiliary output status
           none
           A = AUXOUT: status

     9   9 Print string
           DE = string addr
           none

    10   A Read console buffer
           DE = buffer addr
           Chars in buffer

    11   B Get console status
           none
           A = 00H/0FFH

    12   C Return version number
           none
           HL = BDOS version number (0028H)

    13   D Reset disk system
           none
           none

    14   E Specify drive
           E = specified drive
           A = err flag/00H/0FFH

    15   F Open file
           DE = FCB addr
           A = 00H/0FFH, H = 00H/phys err

    16  10 Close file
           DE = FCB addr
           A = 00H/0FFH, H = 00H/phys err

    17  11 Search for first
           DE = FCB addr
           A = dir code/00-03H/0FFH, H = 00H/phys err

    18  12 Search for next
           none
           A = dir code/00-03H/0FFH, H = 00H/phys err

    19  13 Delete file
           DE = FCB addr
           A = 00H/0FFH, H = 00H/phys err

    20  14 Read sequential
           DE = FCB addr
           A = err code/00H/01H, 0AH, 0FFH

    21  15 Write sequential
           DE = FCB addr
           A = err code/00H/01H, 02H, 0AH, 0FFH

    22  16 Make file
           DE = FCB addr
           A = 00H/0FFH, H = 00H/phys err

    23  17 Rename file
           DE = FCB addr
           A = 00H/0FFH, H = 00H/phys err

    24  18 Return login vector
           none
           HL = login vector (*)

    25  19 Return current drive
           none
           A = current drive number

    26  1A Set DMA address
           DE = DMA addr
           none

    27  1B Get addr (Alloc)
```

```
          none
          HL = alloc addr/0FFFFH (*)

   28  1C Write protect disk
          none
          none

   29  1D Get Read-Only vector
          none
          HL = R-O vector (*)

   30  1E Set file
          DE = FCB addr
          A = 00H/0FFH, H = 00H/phys err

   31  1F Get addr (disk parms)
          none
          HL = DPB addr/0FFFFH

   32  20 Get/set user number
          E = user number/0FFH (Get)
          A = curr number/none

   33  21 Read random
          DE = FCB addr
          A = err code/00H/non-zero, H = 00H/phys err

   34  22 Write random
          DE = FCB addr
          A = err code/00H/non-zero, H = 00H/phys err

   35  23 Compute file size
          DE = FCB addr
          A = 00H/0FFH, H = 00H/phys err

   36  24 Set random record
          DE = FCB addr
          Random Record Number set (in FCB)

   37  25 Reset drive
          DE = drive vector
          none

   40  28 Write random with zero fill
          DE = FCB addr
          A = err code/00H/non-zero

   45  2D Set BDOS error mode
          E = BDOS err mode
          none

   48  30 Flush buffers
          none
          A = err flag/00H, H = phys err

  109  6D Get/set console mode
          DE = 0FFFFH/con mode
          HL = con mode/none

  110  6E Get/set output delimiter
          DE = 0FFFFH/E = output delimiter
          A = output delimiter/none

  111  6F Print block
          DE = CCB addr
          none

  112  70 List block
          DE = CCB addr
          none

  113  71 Direct screen functions
          DE = SFB addr
          none

  124  7C Byte BLT copy
          DE = BCB addr
          A = 00H/0FFH

  125  7D Byte BLT alter
          DE = BCB addr
```

```
        A = 00H/0FFH
```

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

Appendix B: BDOS error handling

The BDOS file system responds to error situations in one of three ways:

Method 1
It returns to the calling program with return codes in register A, H, and L identifying the error.

Method 2
It displays an error message on the console, and branches to the BIOS warm start entry point, thereby terminating execution of the calling program.

Method 3
It displays an error message on the console, and returns to the calling program as in Method 1.

The BDOS file system handles the majority of errors it detects by Method 1. Two examples of this kind of error are the "File not found" error for the OPEN FILE function, and the "Reading unwritten data" error for a READ function. More serious errors, such as disk I/O errors, are usually handled by Method 2. Errors in this category, called physical errors, can also be reported by Methods 1 and 3 under program control.

The BDOS error mode, which can exist in 3 states, determines how the file system handles physical errors. In the default state, the BDOS displays the error message, and terminates the calling program, Method 2. In return error mode, the BDOS returns control to the calling program with the error identified in registers A, H, and L, Method 1. In return and display mode, the BDOS returns control to the calling program with the error identified in registers A, H, and L, and also displays the error message at the console, Method 3. While both return modes protect a program from termination because of a physical error, the return and display mode also allows the calling program to take advantage of the built-in error reporting of the BDOS file system. Physical errors are displayed on the console in the following format, where d: identifies the drive selected when the error condition is detected; "error message" identifies the error.

```
        CP/M Error on x: error message
```

The BDOS physical errors are identified by the following error messages:

- Disk I/O
- Invalid drive
- Read-Only file
- Read-Only disk

The disk I/O error results from an error condition returned to the BDOS from the BIOS module. The file system make BIOS read and write calls to execute file-related BDOS calls. If the BIOS read or write routine detects an error, it returns an error code to the BDOS, resulting in this error.

The invalid drive error also results from an error condition returned to the BDOS from the BIOS module. The BDOS makes a BIOS SELECT DISK call prior to accessing a drive to perform a requested BDOS function. If the BIOS does not support the selected disk, the BDOS returns an error code resulting in this error message.

The Read-Only file error is returned when a program attempts to write to a file that is marked with the Read-Only attribute. It is also returned to a program that attempts to write to a system file opened under user zero from a non-zero user number.

The Read-Only disk error is returned when a program writes to a disk that is in Read-Only status. A drive can be placed in Read-Only status explicitly with the BDOS WRITE PROTECT DISK function.

The following paragraphs describe the error return code conventions of the BDOS file system functions. Most BDOS file system functions fall into 3 categories in regard to return codes: they return an error code, a directory code, or an error flag.

Error code

The following BDOS functions return an error code in register A:

```
        20  Read sequential
        21  Write sequential
        33  Read random
        34  Write random
        40  Write random with zero fill
```

The error code definitions for register A are shown in Table B-1.

Table B-1. Register A BDOS error codes

Code	Meaning
0	Successful function
1	Reading unwritten data, or no available directory space (Write seq)
2	No available data block
3	Cannot close current extent
4	Seek to unwritten extent
5	No available directory space
6	Random Record Number out of range
10	Media changed (a media change was detected on the FCB's drive after the FCB was opened)
255	Physical error, refer to register H

The following BDOS functions return a directory code in register A:

```
        17  Search for first
        18  Search for next
```

Directory code

The directory code definitions for register A are shown in Table B-2.

Table B-2. Register A BDOS directory codes

Code	Meaning
0-3	Successful function
255	Unsuccessful function

A successful directory code identifies the relative starting position of the directory entry in the calling program's current DMA buffer.

Error flag

If the SET BDOS ERROR MODE function is used to place the BDOS in return error mode, the following functions return an error flag on physical errors:

```
        14  Select disk
        15  Open file
        16  Close file
        19  Delete file
        22  Make file
        23  Rename file
        30  Set file attributes
        35  Compute file size
        48  Flush buffers
```

The error flag definitions for register A are shown in Table B-3.

Table B-3. Register A BDOS error flags

Code	Meaning
0	Successful function
255	Physical error, refer to register H

The BDOS returns non-zero values in register H to identify a physical error if the BDOS error mode is in one of the return modes. Except for functions that return a directory code, register A equal to 255 indicates that register H identifies the physical error. For functions that return a directory code, if register A equals 255, and register H is not equal to 0, register H identifies the physical error. Table B-4 shows the physical error codes returned in register H.

Table B-4. Register H BDOS physical errors

Code	Meaning
0	No error, or not a physical error
1	Disk I/O error
2	Read-Only disk
3	Read-Only file, or file opened under user zero from another user number
4	Invalid drive error: drive specify error

The following 2 functions represent a special case, because they return an address in register pair HL.

```
        27  Get addr (Alloc)
        31  Get addr (Disk Parms)
```

When the BDOS is in return error mode and it detects a physical error for these functions, it returns to the calling program with registers A, H, and L all set to zero. Otherwise, they return no error code.

Appendix C: User number conventions

The Personal CP/M user facility divides each drive directory into 16 logically independent directories, designated as user 0 through user 15. Physically, all user directories share the directory area of a drive. In most other aspects, however, they are independent. For example, files with the same name can exist on different user numbers of the same drive with no conflict. However, a single file cannot reside under more than one user number.

Only one user number is active for a program at one time, and the current user number applies to all drives on the system. Furthermore, the FCB format does not contain any field that can be used to override the current user number. As a result, all file and directory operations reference directories associated with the current user number. However, it is possible for a program to access files on different user numbers; this can be accomplished by setting the user number to the file's user number with the BDOS Get/Set User Code function before making the desired BDOS function call for the file. Note that this technique must be used carefully. An error occurs if a program attempts to read or write to a file under a user number different from the user number that was active when the file was opened.

When the CCP loads and executes a transient program, it initializes the user number to the value displayed in the system prompt. If the system prompt does not display a user number, user zero is implied. A transient program can change its user number by making a BDOS Get/Set User Code function call. Changing the user number in this way does not affect the CCP's user number displayed in the system prompt. When the transient program terminates, the CCP's user number is restored.

User zero has special properties under Personal CP/M. When the current user number is not equal to zero, and if a requested file is not present under the current user number, the file system automatically attempts to open the file under user zero. If the file exists under user zero, and if it has the system attribute, T2', set, the file is opened from user zero. Note, however, that files opened in this way cannot be written to; they are available only for read access. This procedure allows utilities, that may include overlays and any other commonly-accessed files, to be placed on user zero, but also be available for access from other user numbers. As a result, commonly-needed utilities need not be copied to all user numbers on a directory, and you can control which user zero files are directly accessible from other user numbers.

Index

(To be done by WS4...)

EOF