



Post Office Box 579, Pacific Grove, California 93950, (408) 649-3896

CP/M 2 INTERFACE GUIDE

Copyright (c) 1979

DIGITAL RESEARCH

Copyright (c) 1979 by Digital Research. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without the prior written permission of Digital Research, Post Office Box 579, Pacific Grove, California 93950.

Disclaimer

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

CP/M 2 INTERFACE GUIDE

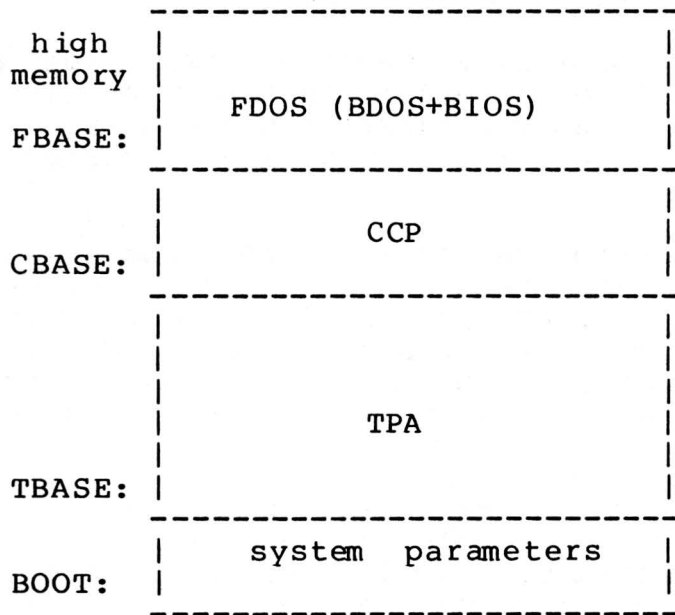
Copyright (c) 1979
Digital Research, Box 579
Pacific Grove, California

1. Introduction	1
2. Operating System Call Conventions	3
3. A Sample File-to-File Copy Program	29
4. A Sample File Dump Utility	34
5. A Sample Random Access Program	37
6. System Function Summary	46

1. INTRODUCTION.

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

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



The exact memory addresses corresponding to BOOT, TBASE, CBASE, and FBASE vary from version to version, and are described fully in the "CP/M Alteration Guide." All standard CP/M versions, however, assume BOOT = 0000H, which is the base of random access memory. The machine code found at location BOOT performs a system "warm start" which loads and initializes the programs and variables necessary to return control to the CCP. Thus, transient programs need only jump to location BOOT

(All Information Contained Herein is Proprietary to Digital Research.)

to return control to CP/M at the command level. Further, the standard versions assume TBASE = BOOT+0100H which is normally location 0100H. The principal entry point to the FDOS is at location BOOT+0005H (normally 0005H) where a jump to FBASE is found. The address field at BOOT+0006H (normally 0006H) contains the value of FBASE and can be used to determine the size of available memory, assuming the CCP is being overlaid by a transient program.

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

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

where "command" is either a built-in function such as DIR or TYPE, or the name of a transient command or program. If the command is a built-in function of CP/M, it is executed immediately. Otherwise, the CCP searches the currently addressed disk for a file by the name

command.COM

If the file is found, it is assumed to be a memory image of a program which executes in the TPA, and thus implicitly originates at TBASE in memory. The CCP loads the COM file from the disk into memory starting at TBASE and possibly extending up to CBASE.

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

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

The transient program may use the CP/M I/O facilities to communicate with the operator's console and peripheral devices, including the disk subsystem. The I/O system is accessed by passing a "function number" and an "information address" to CP/M through the FDOS entry point at BOOT+0005H. In the case of a disk read, for example, the transient program sends the number corresponding to a disk read, along with the address of an FCB to the CP/M FDOS. The FDOS, in turn, performs the operation and returns with either a disk read completion indication or an error number indicating that the disk read was unsuccessful. The function numbers and error indicators are given in below.

(All Information Contained Herein is Proprietary to Digital Research.)

2. OPERATING SYSTEM CALL CONVENTIONS.

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

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

- Read a Console Character
- Write a Console Character
- Read a Sequential Tape Character
- Write a Sequential Tape Character
- Write a List Device Character
- Get or Set I/O Status
- Print Console Buffer
- Read Console Buffer
- Interrogate Console Ready

The FDOS operations which perform disk Input/Output are

- Disk System Reset
- Drive Selection
- File Creation
- File Open
- File Close
- Directory Search
- File Delete
- File Rename
- Random or Sequential Read
- Random or Sequential Write
- Interrogate Available Disks
- Interrogate Selected Disk
- Set DMA Address
- Set/Reset File Indicators

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

(All Information Contained Herein is Proprietary to Digital Research.)

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

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

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

```

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

```

CP/M implements a named file structure on each disk, providing a logical organization which allows any particular file to contain any number of records from completely empty, to the full capacity of the drive. Each drive is logically distinct with a disk directory and file data area. The disk file names are in three parts: the drive select code, the file name consisting of one to eight non-blank characters, and the file type consisting of zero to three non-blank characters. The file type names the generic category of a particular file, while the file name distinguishes individual files in each category. The file types listed below name a few generic categories

(All Information Contained Herein is Proprietary to Digital Research.)

which have been established, although they are generally arbitrary:

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

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

Files in CP/M can be thought of as a sequence of up to 65536 records of 128 bytes each, numbered from 0 through 65535, thus allowing a maximum of 8 megabytes per file. Note, however, that although the records may be considered logically contiguous, they may not be physically contiguous in the disk data area. Internally, all files are broken into 16K byte segments called logical extents, so that counters are easily maintained as 8-bit values. Although the decomposition into extents is discussed in the paragraphs which follow, they are of no particular consequence to the programmer since each extent is automatically accessed in both sequential and random access modes.

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

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

|dr|f1|f2|/ /|f8|t1|t2|t3|ex|s1|s2|rc|d0|/ /|dn|cr|r0|r1|r2|

00 01 02 ... 08 09 10 11 12 13 14 15 16 ... 31 32 33 34 35

where

dr drive code (0 - 16)
 0 => use default drive for file
 1 => auto disk select drive A,
 2 => auto disk select drive B,
 ...
 16=> auto disk select drive P.

f1...f8 contain the file name in ASCII
 upper case, with high bit = 0

t1,t2,t3 contain the file type in ASCII
 upper case, with high bit = 0
 t1', t2', and t3' denote the
 bit of these positions,
 t1' = 1 => Read/Only file,
 t2' = 1 => SYS file, no DIR list

ex contains the current extent number,
 normally set to 00 by the user, but
 in range 0 - 31 during file I/O

s1 reserved for internal system use

s2 reserved for internal system use, set
 to zero on call to OPEN, MAKE, SEARCH

rc record count for extent "ex,"
 takes on values from 0 - 128

d0...dn filled-in by CP/M, reserved for
 system use

cr current record to read or write in
 a sequential file operation, normally
 set to zero by user

r0,r1,r2 optional random record number in the
 range 0-65535, with overflow to r2,
 r0,r1 constitute a 16-bit value with
 low byte r0, and high byte r1

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

(All Information Contained Herein is Proprietary to Digital Research.)

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

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

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

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

If no file names are specified in the original command, then the fields beginning at BOOT+005DH and BOOT+006DH contain blanks. In all cases, the CCP translates lower case alphabetic to upper case to be consistent with the CP/M file naming conventions.

As an added convenience, the default buffer area at location BOOT+0080H is initialized to the command line tail typed by the operator following the program name. The first position contains the number of characters, with the characters themselves following the character count. Given the above command line, the area beginning at BOOT+0080H is initialized as follows:

```
BOOT+0080H:
+00 +01 +02 +03 +04 +05 +06 +07 +08 +09 +10 +11 +12 +13 +14
14 " " "B" ":" "X" "." "Z" "O" "T" " " "Y" "." "Z" "A" "P"
```

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

The individual functions are described in detail in the pages which follow.

(All Information Contained Herein is Proprietary to Digital Research.)

```

*****
*
* FUNCTION 0: System Reset
*
*****
* Entry Parameters:
* Register C: 00H
*
*****

```

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

```

*****
*
* FUNCTION 1: CONSOLE INPUT
*
*****
* Entry Parameters:
* Register C: 01H
*
* Returned Value:
* Register A: ASCII Character
*
*****

```

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

```

*****
*
* FUNCTION 2: CONSOLE OUTPUT
*
*****
* Entry Parameters:
* Register C: 02H
* Register E: ASCII Character
*
*****

```

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

```

*****
*
* FUNCTION 3: READER INPUT
*
*****
* Entry Parameters:
*   Register C: 03H
*
* Returned Value:
*   Register A: ASCII Character
*****

```

The Reader Input function reads the next character from the logical reader into register A (see the IOBYTE definition in the "CP/M Alteration Guide"). Control does not return until the character has been read.

```

*****
*
* FUNCTION 4: PUNCH OUTPUT
*
*****
* Entry Parameters:
*   Register C: 04H
*   Register E: ASCII Character
*
*****

```

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

```

*****
*
* FUNCTION 5: LIST OUTPUT
*
*****
* Entry Parameters:
*   Register C: 05H
*   Register E: ASCII Character
*
*****

```

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

```

*****
*
* FUNCTION 6: DIRECT CONSOLE I/O *
*
*****
* Entry Parameters: *
* Register C: 06H *
* Register E: 0FFH (input) or *
* char (output) *
*
* Returned Value: *
* Register A: char or status *
* (no value) *
*****

```

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

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

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

```

*****
*
* FUNCTION 7: GET I/O BYTE
*
*****
* Entry Parameters:
*   Register C: 07H
*
* Returned Value:
*   Register A: I/O Byte Value
*****

```

The Get I/O Byte function returns the current value of IOBYTE in register A. See the "CP/M Alteration Guide" for IOBYTE definition.

```

*****
*
* FUNCTION 8: SET I/O BYTE
*
*****
* Entry Parameters:
*   Register C: 08H
*   Register E: I/O Byte Value
*
*****

```

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

```

*****
*
* FUNCTION 9: PRINT STRING
*
*****
* Entry Parameters:
*   Register C: 09H
*   Registers DE: String Address
*
*****

```

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

```

*****
*
* FUNCTION 10: READ CONSOLE BUFFER *
*
*****
* Entry Parameters: *
* Register C: 0AH *
* Registers DE: Buffer Address *
*
* Returned Value: *
* Console Characters in Buffer *
*****

```

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

```

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

```

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

```

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

```

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

```

*****
*
* FUNCTION 11: GET CONSOLE STATUS *
*
*****
* Entry Parameters: *
* Register C: 0BH *
*
* Returned Value: *
* Register A: Console Status *
*****

```

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

```

*****
*
* FUNCTION 12: RETURN VERSION NUMBER *
*
*****
* Entry Parameters: *
* Register C: 0CH *
*
* Returned Value: *
* Registers HL: Version Number *
*****

```

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

```

*****
*
* FUNCTION 13: RESET DISK SYSTEM
*
*****
* Entry Parameters:
* Register C: 0DH
*
*****

```

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

```

*****
*
* FUNCTION 14: SELECT DISK
*
*****
* Entry Parameters:
* Register C: 0EH
* Register E: Selected Disk
*
*****

```

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

```

*****
*
* FUNCTION 15: OPEN FILE
*
*****
* Entry Parameters:
*   Register C: 0FH
*   Registers DE: FCB Address
*
* Returned Value:
*   Register A: Directory Code
*****

```

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

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

```

*****
*
* FUNCTION 16: CLOSE FILE
*
*****
* Entry Parameters:
*   Register C: 10H
*   Registers DE: FCB Address
*
* Returned Value:
*   Register A: Directory Code
*****

```

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

```

*****
*
*  FUNCTION 17: SEARCH FOR FIRST      *
*
*****
*  Entry Parameters:                 *
*      Register   C:  11H             *
*      Registers DE: FCB Address      *
*
*  Returned   Value:                 *
*      Register   A:  Directory Code  *
*****

```

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

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

```

*****
*
*  FUNCTION 18: SEARCH FOR NEXT      *
*
*****
*  Entry Parameters:                 *
*      Register   C:  12H             *
*
*  Returned   Value:                 *
*      Register   A:  Directory Code  *
*****

```

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

(All Information Contained Herein is Proprietary to Digital Research.)

```

*****
*
* FUNCTION 19: DELETE FILE
*
*****
* Entry Parameters:
*   Register C: 13H
*   Registers DE: FCB Address
*
* Returned Value:
*   Register A: Directory Code
*****

```

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

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

```

*****
*
* FUNCTION 20: READ SEQUENTIAL
*
*****
* Entry Parameters:
*   Register C: 14H
*   Registers DE: FCB Address
*
* Returned Value:
*   Register A: Directory Code
*****

```

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

```

*****
*
* FUNCTION 21: WRITE SEQUENTIAL
*
*****
* Entry Parameters:
*   Register C: 15H
*   Registers DE: FCB Address
*
* Returned Value:
*   Register A: Directory Code
*****

```

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

```

*****
*
* FUNCTION 22: MAKE FILE
*
*****
* Entry Parameters:
*   Register C: 16H
*   Registers DE: FCB Address
*
* Returned Value:
*   Register A: Directory Code
*****

```

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

(All Information Contained Herein is Proprietary to Digital Research.)

```

*****
*
* FUNCTION 23: RENAME FILE
*
*****
* Entry Parameters:
*   Register C: 17H
*   Registers DE: FCB Address
*
* Returned Value:
*   Register A: Directory Code
*****

```

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

```

*****
*
* FUNCTION 24: RETURN LOGIN VECTOR
*
*****
* Entry Parameters:
*   Register C: 18H
*
* Returned Value:
*   Registers HL: Login Vector
*****

```

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

```

*****
*
* FUNCTION 25: RETURN CURRENT DISK *
*
*****
* Entry Parameters: *
* Register C: 19H *
*
* Returned Value: *
* Register A: Current Disk *
*****

```

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

```

*****
*
* FUNCTION 26: SET DMA ADDRESS *
*
*****
* Entry Parameters: *
* Register C: 1AH *
* Registers DE: DMA Address *
*
*****

```

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

(All Information Contained Herein is Proprietary to Digital Research.)

```

*****
*
* FUNCTION 27: GET ADDR(ALLOC)
*
*****
* Entry Parameters:
*   Register C: 1BH
*
* Returned Value:
*   Registers HL: ALLOC Address
*****

```

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

```

*****
*
* FUNCTION 28: WRITE PROTECT DISK
*
*****
* Entry Parameters:
*   Register C: 1CH
*
*****

```

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

Bdos Err on d: R/O

(All Information Contained Herein is Proprietary to Digital Research.)

```

*****
*
* FUNCTION 29: GET READ/ONLY VECTOR *
*
*****
* Entry Parameters: *
* Register C: 1DH *
*
* Returned Value: *
* Registers HL: R/O Vector Value*
*****

```

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

```

*****
*
* FUNCTION 30: SET FILE ATTRIBUTES *
*
*****
* Entry Parameters: *
* Register C: 1EH *
* Registers DE: FCB Address *
*
* Returned Value: *
* Register A: Directory Code *
*****

```

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

```

*****
*
* FUNCTION 31: GET ADDR(DISK PARMS) *
*
*****
* Entry Parameters: *
* Register C: 1FH *
*
* Returned Value: *
* Registers HL: DPB Address *
*****

```

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

```

*****
*
* FUNCTION 32: SET/GET USER CODE *
*
*****
* Entry Parameters: *
* Register C: 20H *
* Register E: 0FFH (get) or *
* User Code (set) *
*
* Returned Value: *
* Register A: Current Code or *
* (no value) *
*****

```

An application program can change or interrogate the currently active user number by calling function 32. If register E = 0FFH, then the value of the current user number is returned in register A, where the value is in the range 0 to 31. If register E is not 0FFH, then the current user number is changed to the value of E (modulo 32).

```

*****
*
* FUNCTION 33: READ RANDOM
*
*****
* Entry Parameters:
*   Register C: 21H
*   Registers DE: FCB Address
*
* Returned Value:
*   Register A: Return Code
*****

```

The Read Random function is similar to the sequential file read operation of previous releases, except that the read operation takes place at a particular record number, selected by the 24-bit value constructed from the three byte field following the FCB (byte positions r0 at 33, r1 at 34, and r2 at 35). Note that the sequence of 24 bits is stored with least significant byte first (r0), middle byte next (r1), and high byte last (r2). CP/M does not reference byte r2, except in computing the size of a file (function 35). Byte r2 must be zero, however, since a 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, which contains the record to read. This value ranges from 0 to 65535, providing access to any particular record of the 8 megabyte file. In order to process a file using random access, the base extent (extent 0) must first be opened. Although the base extent may or may not contain any allocated data, this ensures that the file is properly recorded in the directory, and is visible in DIR requests. The selected record number is then stored into the random record field (r0,r1), and the BDOS is called to read the record. Upon return from the call, register A either contains an error code, as listed below, or the value 00 indicating the operation was successful. In the latter case, the current DMA address contains the randomly accessed record. Note that contrary to the sequential read operation, the record number is not advanced. Thus, subsequent random read operations continue to read the same record.

Upon each random read operation, the logical extent and current record values are automatically set. Thus, the file can be sequentially read or written, starting from the current randomly accessed position. Note, however, that in this case, the last randomly read record will be re-read as you switch from random mode to sequential read, and the last record will be re-written as you switch to a sequential write operation. You can, of course, simply advance the random record position following each random read or write to obtain the effect of a sequential I/O operation.

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

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

Error code 01 and 04 occur when a random read operation accesses a data block which has not been previously written, or an extent which has not been created, which are equivalent conditions. Error 3 does not normally occur under proper system operation, but can be cleared by simply re-reading, or re-opening extent zero as long as the disk is not physically write protected. Error code 06 occurs whenever byte r2 is non-zero under the current 2.0 release. Normally, non-zero return codes can be treated as missing data, with zero return codes indicating operation complete.

```

*****
*
* FUNCTION 34: WRITE RANDOM
*
*****
* Entry Parameters:
* Register C: 22H
* Registers DE: FCB Address
*
* Returned Value:
* Register A: Return Code
*****

```

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

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

```

*****
*
* FUNCTION 35: COMPUTE FILE SIZE
*
*****
* Entry Parameters:
*   Register C: 23H
*   Registers DE: FCB Address
*
* Returned Value:
*   Random Record Field Set
*****

```

When computing the size of a file, the DE register pair addresses an FCB in random mode format (bytes r0, r1, and r2 are present). The FCB contains an unambiguous file name which 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. If, following a call to function 35, the high record byte r2 is 01, then the file contains the maximum record count 65536. Otherwise, bytes r0 and r1 constitute a 16-bit value (r0 is the least significant byte, as before) which is the file size.

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

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

```

*****
*
* FUNCTION 36: SET RANDOM RECORD *
*
*****
* Entry Parameters: *
* Register C: 24H *
* Registers DE: FCB Address *
*
* Returned Value: *
* Random Record Field Set *
*****

```

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

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

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

3. A SAMPLE FILE-TO-FILE COPY PROGRAM.

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

```

;      sample file-to-file copy program
;
;      at the ccp level, the command
;
;      copy a:x.y b:u.v
;
;      copies the file named x.y from drive
;      a to a file named u.v on drive b.
;
0000 = boot      equ      0000h   ; system reboot
0005 = bdos      equ      0005h   ; bdos entry point
005c = fcbl      equ      005ch   ; first file name
005c = sfcbl     equ      fcbl    ; source fcb
006c = fcb2      equ      006ch   ; second file name
0080 = dbuff     equ      0080h   ; default buffer
0100 = tpa       equ      0100h   ; beginning of tpa
;
0009 = printf    equ      9       ; print buffer func#
000f = openf     equ      15      ; open file func#
0010 = closef    equ      16      ; close file func#
0013 = deletef   equ      19      ; delete file func#
0014 = readf     equ      20      ; sequential read
0015 = writef    equ      21      ; sequential write
0016 = makef     equ      22      ; make file func#
;
0100          org      tpa       ; beginning of tpa
0100 311b02    lxi      sp,stack; local stack
;
;      move second file name to dfcb
0103 0e10     mvi      c,16     ; half an fcb
```

(All Information Contained Herein is Proprietary to Digital Research.)

```

0105 116c00      lxi      d,fc b2  ; source of move
0108 21da01      lxi      h,dfcb  ; destination fcb
010b 1a          mfc b:  ldax     d        ; source fcb
010c 13          inx      d        ; ready next
010d 77          mov      m,a     ; dest fcb
010e 23          inx      h        ; ready next
010f 0d          dcr      c        ; count 16...0
0110 c20b01      jnz      mfc b    ; loop 16 times
;
;
0113 af          xra      a        ; a = 00h
0114 32fa01      sta      dfcbcr  ; current rec = 0
;
;
; source and destination fcb's ready
;
0117 115c00      lxi      d,sfcb  ; source file
011a cd6901      call     open    ; error if 255
011d 118701      lxi      d,nofile; ready message
0120 3c          inr      a        ; 255 becomes 0
0121 cc6101      cz       finis   ; done if no file
;
;
; source file open, prep destination
0124 11da01      lxi      d,dfcb  ; destination
0127 cd7301      call     delete  ; remove if present
;
;
012a 11da01      lxi      d,dfcb  ; destination
012d cd8201      call     make    ; create the file
0130 119601      lxi      d,nodir ; ready message
0133 3c          inr      a        ; 255 becomes 0
0134 cc6101      cz       finis   ; done if no dir space
;
;
; source file open, dest file open
; copy until end of file on source
;
0137 115c00      copy:  lxi      d,sfcb  ; source
013a cd7801      call     read    ; read next record
013d b7          ora      a        ; end of file?
013e c25101      jnz      eofile  ; skip write if so
;
;
; not end of file, write the record
0141 11da01      lxi      d,dfcb  ; destination
0144 cd7d01      call     write   ; write record
0147 11a901      lxi      d,space ; ready message
014a b7          ora      a        ; 00 if write ok
014b c46101      cnz     finis   ; end if so
014e c33701      jmp     copy    ; loop until eof
;
; eofile: ; end of file, close destination
0151 11da01      lxi      d,dfcb  ; destination
0154 cd6e01      call     close   ; 255 if error
0157 21bb01      lxi      h,wrprot; ready message
015a 3c          inr      a        ; 255 becomes 00
015b cc6101      cz       finis   ; shouldn't happen
;
;
; copy operation complete, end

```

```

015e 1lcc01      lxi      d,normal; ready message
;
; finis: ; write message given by de, reboot
0161 0e09      mvi      c,printf
0163 cd0500     call     bdos      ; write message
0166 c30000     jmp      boot      ; reboot system
;
; system interface subroutines
; (all return directly from bdos)
;
0169 0e0f      open:   mvi      c,openf
016b c30500     jmp      bdos
;
016e 0e10      close:  mvi      c,closef
0170 c30500     jmp      bdos
;
0173 0e13      delete: mvi      c,deletf
0175 c30500     jmp      bdos
;
0178 0e14      read:   mvi      c,readf
017a c30500     jmp      bdos
;
017d 0e15      write:  mvi      c,writf
017f c30500     jmp      bdos
;
0182 0e16      make:   mvi      c,makef
0184 c30500     jmp      bdos
;
; console messages
0187 6e6f20fnofile: db      'no source file$'
0196 6e6f209nodir:  db      'no directory space$'
01a9 6f7574fspace:  db      'out of data space$'
01bb 7772695wrprot: db      'write protected?$'
01cc 636f700normal: db      'copy complete$'
;
; data areas
01da          dfcb:   ds      33      ; destination fcb
01fa =        dfcbcr equ    dfcb+32 ; current record
;
01fb          ds      32      ; 16 level stack
stack:
021b          end

```

Note that there are several simplifications in this particular program. First, there are no checks for invalid file names which could, for example, contain ambiguous references. This situation could be detected by scanning the 32 byte default area starting at location 005CH for ASCII question marks. A check should also be made to ensure that the file names have, in fact, been included (check locations 005DH and 006DH for non-blank ASCII characters). Finally, a check should be made to ensure that the source and destination file names are different. A speed improvement could be made by buffering more data on each read operation. One could, for example, determine

(All Information Contained Herein is Proprietary to Digital Research.)

the size of memory by fetching FBASE from location 0006H and use the entire remaining portion of memory for a data buffer. In this case, the programmer simply resets the DMA address to the next successive 128 byte area before each read. Upon writing to the destination file, the DMA address is reset to the beginning of the buffer and incremented by 128 bytes to the end as each record is transferred to the destination file.

4. A SAMPLE FILE DUMP UTILITY.

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

```

; DUMP program reads input file and displays hex data
;
0100          org      100h
0005 =       bdos     equ      0005h    ;dos entry point
0001 =       cons     equ      1        ;read console
0002 =       typef    equ      2        ;type function
0009 =       printf   equ      9        ;buffer print entry
000b =       brkf     equ      11       ;break key function (true if char
000f =       openf    equ      15       ;file open
0014 =       readf    equ      20       ;read function

;
005c =       fcb      equ      5ch      ;file control block address
0080 =       buff     equ      80h      ;input disk buffer address
;
; non graphic characters
000d =       cr       equ      0dh      ;carriage return
000a =       lf       equ      0ah      ;line feed
;
; file control block definitions
005c =       fcdbn    equ      fcb+0    ;disk name
005d =       fcdbfn   equ      fcb+1    ;file name
0065 =       fcdbft   equ      fcb+9    ;disk file type (3 characters)
0068 =       fcdbrl   equ      fcb+12   ;file's current reel number
006b =       fcdbrc   equ      fcb+15   ;file's record count (0 to 128)
007c =       fcdbcr   equ      fcb+32   ;current (next) record number (0
007d =       fcdbln   equ      fcb+33   ;fcb length
;
; set up stack
0100 210000   lxi      h,0
0103 39       dad      sp
;
; entry stack pointer in hl from the ccp
0104 221502   shld     oldsp
;
; set sp to local stack area (restored at finis)
0107 315702   lxi      sp,stktop
;
; read and print successive buffers
010a cdcl01   call     setup    ;set up input file
010d feff     cpi      255      ;255 if file not present
010f c21b01   jnz      openok   ;skip if open is ok
;
; file not there, give error message and return
0112 11f301   lxi      d,opnmsg
0115 cd9c01   call     err
0118 c35101   jmp      finis   ;to return
;

```

(All Information Contained Herein is Proprietary to Digital Research.)

```

openok: ;open operation ok, set buffer index to end
011b 3e80      mvi      a,80h
011d 321302    sta      ibp      ;set buffer pointer to 80h
;            hl contains next address to print
0120 210000    lxi      h,0      ;start with 0000
;
gloop:
0123 e5        push     h        ;save line position
0124 cda201    call    gnb
0127 e1        pop     h        ;recall line position
0128 da5101    jc     finis    ;carry set by gnb if end file
012b 47        mov     b,a
;            print hex values
;            check for line fold
012c 7d        mov     a,l
012d e60f      ani     0fh      ;check low 4 bits
012f c24401    jnz     nonum
;            print line number
0132 cd7201    call    crlf
;
;            check for break key
0135 cd5901    call    break
;            accum lsb = 1 if character ready
0138 0f        rrc     ;into carry
0139 da5101    jc     finis    ;don't print any more
;
013c 7c        mov     a,h
013d cd8f01    call    phex
0140 7d        mov     a,l
0141 cd8f01    call    phex
nonum:
0144 23        inx     h        ;to next line number
0145 3e20      mvi     a,' '
0147 cd6501    call    pchar
014a 78        mov     a,b
014b cd8f01    call    phex
014e c32301    jmp     gloop
;
finis:
;            end of dump, return to ccp
;            (note that a jmp to 0000h reboots)
0151 cd7201    call    crlf
0154 2a1502    lhld   oldsp
0157 f9        sphl
;            stack pointer contains ccp's stack location
0158 c9        ret     ;to the ccp
;
;
;            subroutines
;
break: ;check break key (actually any key will do)
0159 e5d5c5    push  h! push d! push b; environment saved
015c 0e0b      mvi     c,brkf
015e cd0500    call    bdos
0161 c1d1e1    pop  b! pop d! pop h; environment restored

```

(All Information Contained Herein is Proprietary to Digital Research.)

```

0164 c9                ret
;
pchar: ;print a character
0165 e5d5c5          push h! push d! push b; saved
0168 0e02            mvi    c,typef
016a 5f              mov    e,a
016b cd0500          call   bdos
016e c1d1e1          pop    b! pop d! pop h; restored
0171 c9                ret
;
;
crlf:
0172 3e0d            mvi    a,cr
0174 cd6501          call   pchar
0177 3e0a            mvi    a,lf
0179 cd6501          call   pchar
017c c9                ret
;
;
;
pnib: ;print nibble in reg a
017d e60f            ani    0fh ;low 4 bits
017f fe0a            cpi    10
0181 d28901          jnc    pl0
; less than or equal to 9
0184 c630            adi    '0'
0186 c38b01          jmp    prn
;
; greater or equal to 10
0189 c637 pl0:      adi    'a' - 10
018b cd6501 prn:    call   pchar
018e c9                ret
;
;
phex: ;print hex char in reg a
018f f5              push   psw
0190 0f              rrc
0191 0f              rrc
0192 0f              rrc
0193 0f              rrc
0194 cd7d01          call   pnib ;print nibble
0197 f1              pop    psw
0198 cd7d01          call   pnib
019b c9                ret
;
;
err: ;print error message
; d,e addresses message ending with "$"
019c 0e09            mvi    c,printf ;print buffer function
019e cd0500          call   bdos
01a1 c9                ret
;
;
;
gnb: ;get next byte
01a2 3a1302          lda    ibp
01a5 fe80            cpi    80h
01a7 c2b301          jnz    g0
; read another buffer
;

```

```

;
01aa cdce01      call    diskr
01ad b7          ora     a      ;zero value if read ok
01ae cab301      jz      g0      ;for another byte
;
01b1 37          ;      end of data, return with carry set for eof
01b2 c9          stc
ret

;
g0:              ;read the byte at buff+reg a
01b3 5f          mov     e,a      ;ls byte of buffer index
01b4 1600        mvi    d,0      ;double precision index to de
01b6 3c          inr    a      ;index=index+1
01b7 321302      sta    ibp      ;back to memory
;
;      pointer is incremented
;      save the current file address
01ba 218000      lxi    h,buff
01bd 19          dad    d
;
;      absolute character address is in hl
01be 7e          mov    a,m
;
;      byte is in the accumulator
01bf b7          ora    a      ;reset carry bit
01c0 c9          ret

;
setup:           ;set up file
;
01c1 af          xra    a      ;zero to accum
01c2 327c00      sta    fcbr    ;clear current record
;
01c5 115c00      lxi    d,fcbr
01c8 0e0f        mvi    c,openf
01ca cd0500      call   bdos
;
;      255 in accum if open error
01cd c9          ret

;
diskr:           ;read disk file record
01ce e5d5c5      push h! push d! push b
01d1 115c00      lxi    d,fcbr
01d4 0e14        mvi    c,readf
01d6 cd0500      call   bdos
01d9 c1d1e1      pop b! pop d! pop h
01dc c9          ret

;
;      fixed message area
01dd 46494c0     signon: db    'file dump version 2.0$'
01f3 0d0a4e0     opnmsg: db    cr,lf,'no input file present on disk$'

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

```

(All Information Contained Herein is Proprietary to Digital Research.)

5. A SAMPLE RANDOM ACCESS PROGRAM.

This manual is concluded with a rather extensive, but complete example of random access operation. The program listed below performs the simple function of reading or writing random records upon command from the terminal. Given that the program has been created, assembled, and placed into a file labelled RANDOM.COM, the CCP level command:

RANDOM X.DAT

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

next command?

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

nW nR Q

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

type data:

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

error, try again

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

```

;*****
;*
;* sample random access program for cp/m 2.0
;*
;*****
0100          org      100h      ;base of tpa
;
0000 =       reboot  equ      0000h  ;system reboot
0005 =       bdos    equ      0005h  ;bdos entry point
;
0001 =       coninp  equ      1       ;console input function
0002 =       conout  equ      2       ;console output function
0009 =       pstring equ      9       ;print string until '$'
000a =       rstring equ     10       ;read console buffer
000c =       version equ     12       ;return version number
000f =       openf   equ     15       ;file open function
0010 =       closef  equ     16       ;close function
0016 =       makef   equ     22       ;make file function
0021 =       readr   equ     33       ;read random
0022 =       writr   equ     34       ;write random
;
005c =       fcb     equ     005ch   ;default file control block
007d =       ranrec  equ     fcb+33   ;random record position
007f =       ranovf  equ     fcb+35   ;high order (overflow) byte
0080 =       buff    equ     0080h   ;buffer address
;
000d =       cr      equ     0dh     ;carriage return
000a =       lf      equ     0ah     ;line feed
;
;*****
;*
;* load SP, set-up file for random access
;*
;*****
0100 31bc0    lxi      sp,stack
;
;          version 2.0?
0103 0e0c    mvi      c,version
0105 cd050   call     bdos
0108 fe20    cpi      20h      ;version 2.0 or better?
010a d2160   jnc      versok
;          bad version, message and go back
010d 111b0   lxi      d,badver
0110 cd0a0   call     print
0113 c3000   jmp      reboot
;
versok:
;          correct version for random access
0116 0e0f    mvi      c,openf  ;open default fcb
0118 115c0   lxi      d,fcbl
011b cd050   call     bdos
011e 3c      inr      a        ;err 255 becomes zero
011f c2370   jnz      ready
;
;          cannot open file, so create it

```

(All Information Contained Herein is Proprietary to Digital Research.)

```

0122 0e16      mvi      c,makef
0124 115c0     lxi      d,fcf
0127 cd050     call     bdos
012a 3c        inr      a          ;err 255 becomes zero
012b c2370     jnz      ready

;
;          cannot create file, directory full
012e 113a0     lxi      d,nospace
0131 cd0a0     call     print
0134 c3000     jmp      reboot ;back to ccp

;
;*****
;*
;* loop back to "ready" after each command
;*
;*****
;
ready:
;          file is ready for processing
;
0137 cde50     call     readcom ;read next command
013a 227d0     shld    ranrec ;store input record#
013d 217f0     lxi      h,ranovf
0140 36000     mvi      m,0      ;clear high byte if set
0142 fe51     cpi      'Q'      ;quit?
0144 c2560     jnz      notq

;
;          quit processing, close file
0147 0e10     mvi      c,closef
0149 115c0     lxi      d,fcf
014c cd050     call     bdos
014f 3c        inr      a          ;err 255 becomes 0
0150 cab90     jz       error    ;error message, retry
0153 c3000     jmp      reboot ;back to ccp

;
;*****
;*
;* end of quit command, process write
;*
;*****
notq:
;          not the quit command, random write?
0156 fe57     cpi      'W'
0158 c2890     jnz      notw

;
;          this is a random write, fill buffer until cr
015b 114d0     lxi      d,datmsg
015e cd0a0     call     print ;data prompt
0161 0e7f     mvi      c,127    ;up to 127 characters
0163 21800     lxi      h,buff ;destination
rloop: ;read next character to buff
0166 c5        push     b          ;save counter
0167 e5        push     h          ;next destination
0168 cdc20     call     getchr ;character to a
016b e1        pop      h          ;restore counter

```

(All Information Contained Herein is Proprietary to Digital Research.)

```

016c c1      pop      b      ;restore next to fill
016d fe0d   cpi      cr      ;end of line?
016f ca780  jz       erloop
;          not end, store character
0172 77     mov      m,a
0173 23     inx      h      ;next to fill
0174 0d     dcr      c      ;counter goes down
0175 c2660  jnz      rloop   ;end of buffer?
erloop:
;          end of read loop, store 00
0178 3600   mvi      m,0
;
;          write the record to selected record number
017a 0e22   mvi      c,writer
017c 115c0  lxi      d,fcbl
017f cd050  call     bdos
0182 b7     ora      a      ;error code zero?
0183 c2b90  jnz      error   ;message if not
0186 c3370  jmp      ready   ;for another record
;
;*****
;*
;* end of write command, process read
;*
;*****
notw:
;          not a write command, read record?
0189 fe52   cpi      'R'
018b c2b90  jnz      error   ;skip if not
;
;          read random record
018e 0e21   mvi      c,readr
0190 115c0  lxi      d,fcbl
0193 cd050  call     bdos
0196 b7     ora      a      ;return code 00?
0197 c2b90  jnz      error
;
;          read was successful, write to console
019a cdcf0   call     crlf    ;new line
019d 0e80   mvi      c,128   ;max 128 characters
019f 21800  lxi      h,buffer ;next to get
wloop:
01a2 7e     mov      a,m     ;next character
01a3 23     inx      h     ;next to get
01a4 e67f   ani      7fh    ;mask parity
01a6 ca370  jz       ready  ;for another command if 00
01a9 c5     push     b     ;save counter
01aa e5     push     h     ;save next to get
01ab fe20   cpi      ' '    ;graphic?
01ad d4c80  cnc      putchr ;skip output if not
01b0 e1     pop      h
01b1 c1     pop      b
01b2 0d     dcr      c     ;count=count-1
01b3 c2a20  jnz      wloop
01b6 c3370  jmp      ready

```

```

;
;*****
;*
;* end of read command, all errors end-up here
;*
;*****
;
error:
01b9 11590      lxi      d,errmsg
01bc cdda0      call     print
01bf c3370      jmp      ready
;
;*****
;*
;* utility subroutines for console i/o
;*
;*****
getchr:
      ;read next console character to a
01c2 0e01      mvi     c,coninp
01c4 cd050      call    bdos
01c7 c9        ret
;
putchr:
      ;write character from a to console
01c8 0e02      mvi     c,conout
01ca 5f        mov     e,a      ;character to send
01cb cd050      call    bdos     ;send character
01ce c9        ret
;
crlf:
      ;send carriage return line feed
01cf 3e0d      mvi     a,cr     ;carriage return
01d1 cdc80      call    putchr
01d4 3e0a      mvi     a,lf     ;line feed
01d6 cdc80      call    putchr
01d9 c9        ret
;
print:
      ;print the buffer addressed by de until $
01da d5        push    d
01db cdcf0      call    crlf
01de d1        pop     d        ;new line
01df 0e09      mvi     c,pstring
01e1 cd050      call    bdos     ;print the string
01e4 c9        ret
;
readcom:
      ;read the next command line to the conbuf
01e5 116b0      lxi     d,prompt
01e8 cdda0      call    print    ;command?
01eb 0e0a      mvi     c,rstring
01ed 117a0      lxi     d,conbuf
01f0 cd050      call    bdos     ;read command line
;          command line is present, scan it

```

```

01f3 21000      lxi      h,0      ;start with 0000
01f6 117c0      lxi      d,conlin;command line
01f9 1a        readc:  ldax     d      ;next command character
01fa 13         inx      d      ;to next command position
01fb b7         ora      a      ;cannot be end of command
01fc c8         rz
;
01fd d630       sui      '0'
01ff fe0a       cpi      l0      ;carry if numeric
0201 d2130     jnc      endrd
;
0204 29         dad      h      ;*2
0205 4d         mov      c,l
0206 44         mov      b,h      ;bc = value * 2
0207 29         dad      h      ;*4
0208 29         dad      h      ;*8
0209 09         dad      b      ;*2 + *8 = *10
020a 85         add      l      ;+digit
020b 6f         mov      l,a
020c d2f90     jnc      readc   ;for another char
020f 24         inr      h      ;overflow
0210 c3f90     jmp      readc   ;for another char
;
endrd:
;
0213 c630       adi      '0'      ;command
0215 fe61       cpi      'a'      ;translate case?
0217 d8         rc
;
0218 e65f       ani      101$1111b
021a c9         ret
;
;*****
;*
;* string data area for console messages
;*
;*****
badver:
021b 536f79     db      'sorry, you need cp/m version 2$'
nospace:
023a 4e6f29     db      'no directory space$'
datmsg:
024d 547970     db      'type data: $'
errmsg:
0259 457272     db      'error, try again.$'
prompt:
026b 4e6570     db      'next command? $'
;

```

```

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

```

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

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

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

Rename the program shown above as QUERY, and massage it a bit so that it reads a sorted key file into memory. The command line might appear as:

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

Instead of reading a number, the QUERY program reads an alphanumeric string which is a particular key to find in the NAMES.DAT data base. Since the LASTNAME.KEY list is sorted, you can find a particular entry quite rapidly by performing a "binary search," similar to looking up a name in the telephone book. That is, starting at both ends of the list, you examine the entry halfway in between and, if not matched, split either the upper half or the lower half for the next search. You'll quickly reach the item you're looking for (in $\log_2(n)$ steps) where you'll find the corresponding record number. Fetch and display this record at the console, just as we have done in the program shown above.

(All Information Contained Herein is Proprietary to Digital Research.)

At this point you're just getting started. With a little more work, you can allow a fixed grouping size which differs from the 128 byte record shown above. This is accomplished by keeping track of the record number as well as the byte offset within the record. Knowing the group size, 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 which satisfy several relationships, such as a LASTNAME between HARDY and LAUREL, and an AGE less than 45. Display all the records which fit this description. Finally, if your lists are getting too big to fit into memory, randomly access your key files from the disk as well. One note of consolation after all this work: if you make it through the project, you'll have no more need for this manual!

6. SYSTEM FUNCTION SUMMARY.

FUNC	FUNCTION NAME	INPUT PARAMETERS	OUTPUT RESULTS
0	System Reset	none	none
1	Console Input	none	A = char
2	Console Output	E = char	none
3	Reader Input	none	A = char
4	Punch Output	E = char	none
5	List Output	E = char	none
6	Direct Console I/O	see def	see def
7	Get I/O Byte	none	A = IOBYTE
8	Set I/O Byte	E = IOBYTE	none
9	Print String	DE = .Buffer	none
10	Read Console Buffer	DE = .Buffer	see def
11	Get Console Status	none	A = 00/FF
12	Return Version Number	none	HL= Version*
13	Reset Disk System	none	see def
14	Select Disk	E = Disk Number	see def
15	Open File	DE = .FCB	A = Dir Code
16	Close File	DE = .FCB	A = Dir Code
17	Search for First	DE = .FCB	A = Dir Code
18	Search for Next	none	A = Dir Code
19	Delete File	DE = .FCB	A = Dir Code
20	Read Sequential	DE = .FCB	A = Err Code
21	Write Sequential	DE = .FCB	A = Err Code
22	Make File	DE = .FCB	A = Dir Code
23	Rename File	DE = .FCB	A = Dir Code
24	Return Login Vector	none	HL= Login Vect*
25	Return Current Disk	none	A = Cur Disk#
26	Set DMA Address	DE = .DMA	none
27	Get Addr(Alloc)	none	HL= .Alloc
28	Write Protect Disk	none	see def
29	Get R/O Vector	none	HL= R/O Vect*
30	Set File Attributes	DE = .FCB	see def
31	Get Addr(disk parms)	none	HL= .DPB
32	Set/Get User Code	see def	see def
33	Read Random	DE = .FCB	A = Err Code
34	Write Random	DE = .FCB	A = Err Code
35	Compute File Size	DE = .FCB	r0, r1, r2
36	Set Random Record	DE = .FCB	r0, r1, r2

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

