DIGITAL RESEARCH

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

CP/M 2.0 USER'S GUIDE

FOR CP/M 1.4 OWNERS

COPYRIGHT (c) 1979

DIGITAL RESEARCH

Copyright

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.

Trademarks

CP/M is a registered trademark of Digital Research. MP/M, MAC, and SID are trademarks of Digital Research.

DIGITAL RESEARCH

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

CP/M 2.0 USER'S GUIDE

FOR CP/M 1.4 OWNERS

COPYRIGHT (c) 1979

DIGITAL RESEARCH

Copyright

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.

Trademarks

CP/M is a registered trademark of Digital Research. MP/M, MAC, and SID are trademarks of Digital Research.

Technical consultation is available for any problems you may encounter in verifying proper operation of this product. We are sorry, but we are not able to evaluate or assist in the debugging of any programs you may develop with this product. For technical assistance, call:

(616) 982-8334 Application Software
(616) 982-3860 Operating Systems/Language Software

Consultation is available between 8:00 am and 4:30 pm (Eastern), on normal business days.

Zenith Data Systems
Software Consultation
Hilltop Road
St. Joseph, MI 49085

CP/M 2.0 USER'S GUIDE FOR CP/M 1.4 OWNERS

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

1. AN OVERVIEW OF CP/M 2.0 FACILITIES.

CP/M 2.0 is a high-performance single-console operating system which uses table driven techniques to allow field reconfiguration to match a wide variety of disk capacities. All of the fundamental file restrictions are removed, while maintaining upward compatibility from previous versions of release 1. Features of CP/M 2.0 include field specification of one to sixteen logical drives, each containing up to eight megabytes. Any particular file can reach the full drive size with the capability to expand to thirty-two megabytes in future releases. The directory size can be field configured to contain any reasonable number of entries, and each file is optionally tagged with read/only and system attributes. Users of CP/M 2.0 are physically separated by user numbers, with facilities for file copy operations from one user area to another. Powerful relative-record random access functions are present in CP/M 2.0 which provide direct access to any of the 65536 records of an eight megabyte file.

All disk-dependent portions of CP/M 2.0 are placed into a BIOS-resident "disk parameter block" which is either hand coded or produced automatically using the disk definition macro library provided with CP/M 2.0. The end user need only specify the maximum number of active disks, the starting and ending sector numbers, the data allocation size, the maximum extent of the logical disk, directory size information, and reserved track values. The macros use this information to generate the appropriate tables and table references for use during CP/M 2.0 operation. Deblocking information is also provided which aids in assembly or disassembly of sector sizes which are multiples of the fundamental 128 byte data unit, and the system alteration manual includes general-purpose subroutines which use the this deblocking information to take advantage of larger sector sizes. Use of these subroutines, together with the table driven data access algorithms, make CP/M 2.0 truly a universal data management system.

File expansion is achieved by providing up to 512 logical file extents, where each logical extent contains 16K bytes of data. CP/M 2.0 is structured, however, so that as much as 128K bytes of data is addressed by a single physical extent (corresponding to a single directory entry), thus maintaining compatibility with previous versions while taking full advantage of directory space.

Random access facilities are present in CP/M 2.0 which allow immediate reference to any record of an eight megabyte file. Using CP/M's unique data organization, data blocks are only allocated when actually required and movement to a record position requires little search time. Sequential file access is upward compatible from earlier versions to the full eight megabytes, while random access compatibility stops at 512K byte files. Due to CP/M 2.0's simpler and faster random access, application programmers are encouraged to alter their programs to take full advantage of the 2.0 facilities.

Several CP/M 2.0 modules and utilities have improvements which correspond to the enhanced file system. STAT and PIP both account for file attributes and user areas, while the CCP provides a "login"

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

function to change from one user area to another. The CCP also formats directory displays in a more convenient manner and accounts for both CRT and hard-copy devices in its enhanced line editing functions.

The sections below point out the individual differences between CP/M 1.4 and CP/M 2.0, with the understanding that the reader is either familiar with CP/M 1.4, or has access to the 1.4 manuals. Additional information dealing with CP/M 2.0 I/O system alteration is presented in the Digital Research manual "CP/M 2.0 Alteration Guide."

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

2. USER INTERFACE.

Console line processing takes CRT-type devices into account with three new control characters, shown with an asterisk in the list below (the symbol "ctl" below indicates that the control key is simultaneously depressed):

rub/del	removes and echoes last character
ctl-C	reboot when at beginning of line
ctl-E	physical end of line
ctl-H	backspace one character position*
ctl-J	(line feed) terminates current input*
ctl-M	(carriage return) terminates input
ctl-R	retype current line after new line
ctl-U	remove current line after new line
ctl-X	backspace to beginning of current line*

In particular, note that ctl-H produces the proper backspace overwrite function (ctl-H can be changed internally to another character, such as delete, through a simple single byte change). Further, the line editor keeps track of the current prompt column position so that the operator can properly align data input following a ctl-U, ctl-R, or ctl-X command.

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

3. CONSOLE COMMAND PROCESSOR (CCP) INTERFACE.

There are four functional differences between CP/M 1.4 and CP/M 2.0 at the console command processor (CCP) level. The CCP now displays directory information across the screen (four elements per line), the USER command is present to allow maintenance of separate files in the same directory, and the actions of the "ERA *.*" and "SAVE" commands have changed. The altered DIR format is self-explanatory, while the USER command takes the form:

USER n

where n is an integer value in the range 0 to 15. Upon cold start, the operator is automatically "logged" into user area number 0, which is compatible with standard CP/M 1.4 directories. The operator may issue the USER command at any time to move to another logical area within the same directory. Drives which are logged-in while addressing one user number are automatically active when the operator moves to another user number since a user number is simply a prefix which accesses particular directory entries on the active disks.

The active user number is maintained until changed by a subsequent USER command, or until a cold start operation when user 0 is again assumed.

Due to the fact that user numbers now tag individual directory entries, the ERA *.* command has a different effect. In version 1.4, this command can be used to erase a directory which has "garbage" information, perhaps resulting from use of a diskette under another operating system (heaven forbid!). In 2.0, however, the ERA *.* command affects only the current user number. Thus, it is necessary to write a simple utility to erase a nonsense disk (the program simply writes the hexadecimal pattern E5 throughout the disk).

The SAVE command in version 1.4 allows only a single memory save operation, with the potential of destroying the memory image due to directory operations following extent boundary changes. Version 2.0, however, does not perform directory operations in user data areas after disk writes, and thus the SAVE operation can be used any number of times without altering the memory image.

4. STAT ENHANCEMENTS.

The STAT program has a number of additional functions which allow disk parameter display, user number display, and file indicator manipulation. The command:

STAT VAL:

produces a summary of the available status commands, resulting in the output:

```
Temp R/O Disk: d:=R/O
Set Indicator: d:filename.typ $R/O $R/W $SYS $DIR
Disk Status  : DSK: d:DSK:
User Status  : USR:
Iobyte Assign:
(list of possible assignments)
```

which gives an instant summary of the possible STAT commands. The command form:

STAT d:filename.typ $S

where "d:" is an optional drive name, and "filename.typ" is an unambiguous or ambiguous file name, produces the output display format:

```
 Size  Recs  Bytes  Ext Acc
   48    48     6k    1 R/O A:ED.COM
   55    55    12k    1 R/O (A:PIP.COM)
65536   128     2k    2 R/W A:X.DAT
```

where the $S parameter causes the "Size" field to be displayed (without the $S, the Size field is skipped, but the remaining fields are displayed). The Size field lists the virtual file size in records, while the "Recs" field sums the number of virtual records in each extent. For files constructed sequentially, the Size and Recs fields are identical. The "Bytes" field lists the actual number of bytes allocated to the corresponding file. The minimum allocation unit is determined at configuration time, and thus the number of bytes corresponds to the record count plus the remaining unused space in the last allocated block for sequential files. Random access files are given data areas only when written, so the Bytes field contains the only accurate allocation figure. In the case of random access, the Size field gives the logical end-of-file record position and the Recs field counts the logical records of each extent (each of these extents, however, may contain unallocated "holes" even though they are added into the record count). The "Ext" field counts the number of logical 16K extents allocated to the file. Unlike version 1.4, the Ext count does not necessarily correspond to the number of directory entries given to the file, since there can be up to 128K bytes (8 logical extents) directly addressed by a single directory entry, depending upon allocation size (in a special case, there are actually 256K bytes which can be directly addressed by a physical extent).

The "Acc" field gives the R/O or R/W access mode, which is changed using the commands shown below. Similarly, the parentheses

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

shown around the PIP.COM file name indicate that it has the "system" indicator set, so that it will not be listed in DIR commands. The four command forms

```
STAT d:filename.typ $R/O
STAT d:filename.typ $R/W
STAT d:filename.typ $SYS
STAT d:filename.typ $DIR
```

set or reset various permanent file indicators. The R/O indicator places the file (or set of files) in a read-only status until changed by a subsequent STAT command. The R/O status is recorded in the directory with the file so that it remains R/O through intervening cold start operations. The R/W indicator places the file in a permanent read/write status. The SYS indicator attaches the system indicator to the file, while the DIR command removes the system indicator. The "filename.typ" may be ambiguous or unambiguous, but in either case, the files whose attributes are changed are listed at the console when the change occurs. The drive name denoted by "d:" is optional.

When a file is marked R/O, subsequent attempts to erase or write into the file result in a terminal BDOS message

```
Bdos Err on d: File R/O
```

The BDOS then waits for a console input before performing a subsequent warm start (a "return" is sufficient to continue). The command form

```
STAT d:DSK:
```

lists the drive characteristics of the disk named by "d:" which is in the range A:, B:, ..., P:. The drive characteristics are listed in the format:

```
    d: Drive Characteristics
65536: 128 Byte record Capacity
 8192: Kilobyte Drive Capacity
  128: 32  Byte Directory Entries
    Ø: Checked  Directory Entries
 1Ø24: Records/ Extent
  128: Records/ Block
   58: Sectors/ Track
    2: Reserved Tracks
```

where "d:" is the selected drive, followed by the total record capacity (65536 is an 8 megabyte drive), followed by the total capacity listed in Kilobytes. The directory size is listed next, followed by the "checked" entries. The number of checked entries is usually identical to the directory size for removable media, since this mechanism is used to detect changed media during CP/M operation without an intervening warm start. For fixed media, the number is usually zero, since the media is not changed without at least a cold or warm start. The number of records per extent determines the addressing capacity of each directory entry (1Ø24 times 128 bytes, or

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

128K in the example above). The number of records per block shows the basic allocation size (in the example, 128 records/block times 128 bytes per record, or 16K bytes per block). The listing is then followed by the number of physical sectors per track and the number of reserved tracks. For logical drives which share the same physical disk, the number of reserved tracks may be quite large, since this mechanism is used to skip lower-numbered disk areas allocated to other logical disks. The command form

```
STAT DSK:
```

produces a drive characteristics table for all currently active drives. The final STAT command form is

```
STAT USR:
```

which produces a list of the user numbers which have files on the currently addressed disk. The display format is:

```
Active User : 0
Active Files: 0 1 3
```

where the first line lists the currently addressed user number, as set by the last CCP USER command, followed by a list of user numbers scanned from the current directory. In the above case, the active user number is 0 (default at cold start), with three user numbers which have active files on the current disk. The operator can subsequently examine the directories of the other user numbers by logging-in with USER 1, USER 2, or USER 3 commands, followed by a DIR command at the CCP level.

5. PIP ENHANCEMENTS.

PIP provides three new functions which account for the features of CP/M 2.Ø. All three functions take the form of file parameters which are enclosed in square brackets following the appropriate file names. The commands are:

Gn	Get File from User number n (n in the range Ø - 15)
W	Write over R/O files without console interrogation
R	Read system files

The G command allows one user area to receive data files from another. Assuming the operator has issued the USER 4 command at the CCP level, the PIP statement

```
PIP X.Y = X.Y[G2]
```

reads file X.Y from user number 2 into user area number 4. The command

```
PIP A:=A:*.*[G2]
```

copies all of the files from the A drive directory for user number 2 into the A drive directory of the currently logged user number. Note that to ensure file security, one cannot copy files into a different area than the one which is currently addressed by the USER command.

Note also that the PIP program itself is initially copied to a user area (so that subsequent files can be copied) using the SAVE command. The sequence of operations shown below effectively moves PIP from one user area to the next.

```
USER Ø              login user Ø
DDT PIP.COM         load PIP to memory
(note PIP size s)
GØ                  return to CCP
USER 3              login user 3
SAVE s PIP.COM
```

where s is the integral number of memory "pages" (256 byte segments) occupied by PIP. The number s can be determined when PIP.COM is loaded under DDT, by referring to the value under the "NEXT" display. If for example, the next available address is 1DØØ, then PIP.COM requires 1C hexadecimal pages (or 1 times 16 + 12 = 28 pages), and thus the value of s is 28 in the subsequent save. Once PIP is copied in this manner, it can then be copied to another disk belonging to the same user number through normal pip transfers.

Under normal operation, PIP will not overwrite a file which is set to a permanent R/O status. If attempt is made to overwrite a R/O file, the prompt

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

DESTINATION FILE IS R/O, DELETE (Y/N)?

is issued. If the operator responds with the character "y" then the file is overwritten. Otherwise, the response

** NOT DELETED **

is issued, the file transfer is skippped, and PIP continues with the next operation in sequence. In order to avoid the prompt and response in the case of R/O file overwrite, the command line can include the W parameter, as shown below

PIP A:=B:*.COM[W]

which copies all non-system files to the A drive from the B drive, and overwrites any R/O files in the process. If the operation involves several concatenated files, the W parameter need only be included with the last file in the list, as shown in the following example

PIP A.DAT = B.DAT,F:NEW.DAT,G:OLD.DAT[W]

Files with the system attribute can be included in PIP transfers if the R parameter is included, otherwise system files are not recognized. The command line

PIP ED.COM = B:ED.COM[R]

for example, reads the ED.COM file from the B drive, even if it has been marked as a R/O and system file. The system file attributes are copied, if present.

It should be noted that downward compatibility with previous versions of CP/M is only maintained if the file does not exceed one megabyte, no file attributes are set, and the file is created by user 0. If compatibility is required with non-standard (e.g., "double density") versions of 1.4, it may be necessary to select 1.4 compatibility mode when constructing the internal disk parameter block (see the "CP/M 2.0 Alteration Guide," and refer to Section 10 which describes BIOS differences).

6. ED ENHANCEMENTS.

The CP/M standard program editor provides several new facilities in the 2.0 release. Experience has shown that most operators use the relative line numbering feature of ED, and thus the editor has the "v" (Verify Line) option set as an initial value. The operator can, of course, disable line numbering by typing the "-v" command. If you are not familiar with the ED line number mode, you may wish to refer to the Appendix in the ED user's guide, where the "v" command is described.

ED also takes file attributes into account. If the operator attempts to edit a read/only file, the message

** FILE IS READ/ONLY **

appears at the console. The file can be loaded and examined, but cannot be altered in any way. Normally, the operator simply ends the edit session, and uses STAT to change the file attribute to R/W. If the edited file has the "system" attribute set, the message

"SYSTEM" FILE NOT ACCESSIBLE

is displayed at the console, and the edit session is aborted. Again, the STAT program can be used to change the system attribute, if desired.

Finally, the insert mode ("i") command allows CRT line editing functions, as described in Section 2, above.

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

7. THE XSUB FUNCTION.

An additional utility program is supplied with version 2.0 of CP/M, called XSUB, which extends the power of the SUBMIT facility to include line input to programs as well as the console command processor. The XSUB command is included as the first line of your submit file and, when executed, self-relocates directly below the CCP. All subsequent submit command lines are processed by XSUB, so that programs which read buffered console input (BDOS function 10) receive their input directly from the submit file. For example, the file SAVER.SUB could contain the submit lines:

```
XSUB
DDT
I$1.HEX
R
G0
SAVE 1 $2.COM
```

with a subsequent SUBMIT command:

```
SUBMIT SAVER X Y
```

which substitutes X for $1 and Y for $2 in the command stream. The XSUB program loads, followed by DDT which is sent the command lines "IX.HEX" "R" and "G0" thus returning to the CCP. The final command "SAVE 1 Y.COM" is processed by the CCP.

The XSUB program remains in memory, and prints the message

```
(xsub active)
```

on each warm start operation to indicate its presence. Subsequent submit command streams do not require the XSUB, unless an intervening cold start has occurred. Note that XSUB must be loaded after DESPOOL, if both are to run simultaneously.

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

8. BDOS INTERFACE CONVENTIONS.

CP/M 2.0 system calls take place in exactly the same manner as earlier versions, with a call to location 0005H, function number in register C, and information address in register pair DE. Single byte values are returned in register A, with double byte values returned in HL (for reasons of compatibility, register A = L and register B = H upon return in all cases). A list of CP/M 2.0 calls is given below, with an asterisk following functions which are either new or revised from version 1.4 to 2.0. Note that a zero value is returned for out-of range function numbers.

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 Addr(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, 29, and 32 should be avoided in application programs to maintain upward compatibility with MP/M.) The new or revised functions are described below.

Function 6: Direct Console I/O.

Direct Console I/O is supported under CP/M 2.0 for those applications where it is necessary to avoid the BDOS console I/O operations. Programs which currently perform direct I/O through the BIOS 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.

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

Function 10: Read Console Buffer.

The console buffer read operation remains unchanged except that console line editing is supported, as described in Section 2. 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 (previously, the carriage returned to the extreme left margin). This new convention makes operator data input and line correction more legible.

Function 12: Return Version Number.

Function 12 has been redefined to provide information which allows version-independent programming (this was previously the "lift head" function which returned HL=0000 in version 1.4, but performed no operation). The value returned by function 12 is a two-byte value, with H = 00 for 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.

In the file operations described below, DE addresses a file control block (FCB). Further, all directory operations take place in a reserved area which does not affect write buffers as was the case in version 1.4, 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 bytes 007DH, 007EH, and 007FH are available for this purpose. For notational purposes, the FCB format is shown with the following fields:

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

```
--------------------------------------------------------------
|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
```

Function 15: Open File.

The Open File operation is identical to previous definitions, with the exception that byte s2 is automatically zeroed. Note that previous versions of CP/M defined this byte as zero, but made no

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

checks to assure compliance. Thus, the byte is cleared to ensure upward compatibility with the latest version, where it is required.

Function 17: Search for First.

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 a value of A equal to Ø, 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 f1 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.

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.

Function 19: Delete File.

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 reference file or files could not be found, otherwise a value in the range Ø to 3 is returned.

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

Function 22: Make File.

The Make File operation is identical to previous versions of CP/M, except that byte s2 is zeroed upon entry to the BDOS.

Function 23: Rename File.

The Actions of the file rename functions are the same as previous releases except that the value 255 is returned if the rename function is unsuccessful (the file to rename could not be found), otherwise a value in the range Ø to 3 is returned.

Function 24: Return Login Vector.

The login vector value returned by CP/M 2.Ø 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. Note that compatibility is maintained with earlier releases, since registers A and L contain the same values upon return.

Function 28: Write Protect Current Disk.

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

Function 29: Get R/O Vector.

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 3Ø: Set File Attributes.

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' above) can be set or reset. The DE pair addresses an unambiguous file name with the appropriate attributes set or reset. Function 3Ø 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 Disk Parameter Block 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 or Get User Code.

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

Function 33: Read Random.

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 rØ at 33, r1 at 34, and r2 at 35). Note that the sequence of 24 bits is stored with least significant byte first (rØ), middle byte next (r1), and high byte last (r2). CP/M release 2.Ø 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, in version 2.Ø, the rØ,r1 byte pair is treated as a double-byte, or "word" value, which contains the record to read. This value ranges from Ø 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 Ø) 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 (rØ,r1), and the BDOS is called to read the record. Upon return from the call, register A either contains an

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

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.

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 under either CP/M 1.4 or CP/M 2.0.

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.

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 in version 2.0. 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.

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

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

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.

This section 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 (ok, so the program's not so brief), 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,

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

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 =      writer  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 cdda0          call    print
0113 c3000          jmp     reboot
            ;
            versok:
            ;       correct version for random access
```

```
0116 0e0f           mvi     c,openf ;open default fcb
0118 115c0          lxi     d,fcb
011b cd050          call    bdos
011e 3c             inr     a       ;err 255 becomes zero
011f c2370          jnz     ready
               ;
               ;        cannot open file, so create it
0122 0e16           mvi     c,makef
0124 115c0          lxi     d,fcb
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 cdda0          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 3600           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,fcb
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 cdda0          call    print   ;data prompt
```

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

```
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
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,fcb
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,fcb
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,buff  ;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
```

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

```
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:
```

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

```
                        ;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
                ;       not zero, numeric?
01fd d630               sui     '0'
01ff fe0a               cpi     10      ;carry if numeric
0201 d2130              jnc     endrd
                ;       add-in next digit
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:
                ;       end of read, restore value in a
0213 c630               adi     '0'     ;command
0215 fe61               cpi     'a'     ;translate case?
0217 d8                 rc
                ;       lower case, mask lower case bits
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? $'
                ;
```

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

```
                ;*************************************************
                ;*                                               *
                ;* 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
```

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

9. CP/M 2.0 MEMORY ORGANIZATION.

Similar to earlier versions, CP/M 2.0 is field-altered to fit various memory sizes, depending upon the host computer memory configuration. Typical base addresses for popular memory sizes are shown in the table below.

Module	20k	24k	32k	48k	64k
CCP	3400H	4400H	6400H	A400H	E400H
BDOS	3C00H	4C00H	6C00H	AC00H	EC00H
BIOS	4A00H	5A00H	7A00H	BA00H	FA00H
Top of Ram	4FFFH	5FFFH	7FFFH	BFFFH	FFFFH

The distribution disk contains a CP/M 2.0 system configured for a 20k Intel MDS-800 with standard IBM 8" floppy disk drives. The disk layout is shown below:

Sector	Track 00	Module	Track 01	Module
1	(Bootstrap Loader)		4080H	BDOS + 480H
2	3400H	CCP + 000H	4100H	BDOS + 500H
3	3480H	CCP + 080H	4180H	BDOS + 580H
4	3500H	CCP + 100H	4200H	BDOS + 600H
5	3580H	CCP + 180H	4280H	BDOS + 680H
6	3600H	CCP + 200H	4300H	BDOS + 700H
7	3680H	CCP + 280H	4380H	BDOS + 780H
8	3700H	CCP + 300H	4400H	BDOS + 800H
9	3780H	CCP + 380H	4480H	BDOS + 880H
10	3800H	CCP + 400H	4500H	BDOS + 900H
11	3880H	CCP + 480H	4580H	BDOS + 980H
12	3900H	CCP + 500H	4600H	BDOS + A00H
13	3980H	CCP + 580H	4680H	BDOS + A80H
14	3A00H	CCP + 600H	4700H	BDOS + B00H
15	3A80H	CCP + 680H	4780H	BDOS + B80H
16	3B00H	CCP + 700H	4800H	BDOS + C00H
17	3B80H	CCP + 780H	4880H	BDOS + C80H
18	3C00H	BDOS + 000H	4900H	BDOS + D00H
19	3C80H	BDOS + 080H	4980H	BDOS + D80H
20	3D00H	BDOS + 100H	4A00H	BIOS + 000H
21	3D80H	BDOS + 180H	4A80H	BIOS + 080H
22	3E00H	BDOS + 200H	4B00H	BIOS + 100H
23	3E80H	BDOS + 280H	4B80H	BIOS + 180H
24	3F00H	BDOS + 300H	4C00H	BIOS + 200H
25	3F80H	BDOS + 380H	4C80H	BIOS + 280H
26	4000H	BDOS + 400H	4D00H	BIOS + 300H

In particular, note that the CCP is at the same position on the disk, and occupies the same space as version 1.4. The BDOS portion, however, occupies one more 256-byte page and the BIOS portion extends through the remainder of track 01. Thus, the CCP is 800H (2048 decimal) bytes in length, the BDOS is E00H (3584 decimal) bytes in length, and the BIOS is up to 380H (898 decimal) bytes in length. In version 2.0, the BIOS portion contains the standard subroutines of 1.4, along with some initialized table space, as described in the following section.

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

10. BIOS DIFFERENCES.

The CP/M 2.0 Basic I/O System differs only slightly in concept from its predecesssors. Two new jump vector entry points are defined, a new sector translation subroutine is included, and a disk characteristics table must be defined. The skeletal form of these changes are found in the program shown below.

```
                org     4000h
                maclib  diskdef
                jmp     boot
        ;       ...
                jmp     listst  ;list status
                jmp     sectran ;sector translate
                disks   4
        ;       large capacity drive
        bpb     equ     16*1024 ;bytes per block
        rpb     equ     bpb/128 ;records per block
        maxb    equ     65535/rpb ;max block number
                diskdef 0,1,58,3,bpb,maxb+1,128,0,2
                diskdef 1,1,58,,bpb,maxb+1,128,0,2
                diskdef 2,0
                diskdef 3,1
        ;
        boot:   ret     ;nop
        ;
        listst: xra     a       ;nop
                ret
        ;
        seldsk:
                ;drive number in c
                lxi     h,0     ;0000 in hl produces select error
                mov     a,c     ;a is disk number 0 ... ndisks-1
                cpi     ndisks  ;less than ndisks?
                rnc             ;return with HL = 0000 if not
        ;       proper disk number, return dpb element address
                mov     l,c
                dad     h       ;*2
                dad     h       ;*4
                dad     h       ;*8
                dad     h       ;*16
                lxi     d,dpbase
                dad     d       ;HL=.dpb
                ret
        ;
        selsec:
                ;sector number in c
                lxi     h,sector
                mov     m,c
                ret
        ;
        sectran:
                ;translate sector BC using table at DE
                xchg            ;HL = .tran
                dad     b       ;single precision tran
```

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

```
48: ;       dad b again if double precision tran
49:         mov     l,m     ;only low byte necessary here
50: ;       fill both H and L if double precision tran
51:         ret             ;HL = ??ss
52: ;
53: sector: ds      1
54:         endef
55:         end
```

Referring to the program shown above, lines 3-6 represent the BIOS entry vector of 17 elements (version 1.4 defines only 15 jump vector elements). The last two elements provide access to the "LISTST" (List Status) entry point for DESPOOL. The use of this particular entry point is defined in the DESPOOL documentation, and is no different than the previous 1.4 release. It should be noted that the 1.4 DESPOOL program will not operate under version 2.0, but an update version will be available from Digital Research in the near future.

The "SECTRAN" (Sector Number Translate) entry shown in the jump vector at line 6 provides access to a BIOS-resident sector translation subroutine. This mechanism allows the user to specify the sector skew factor and translation for a particular disk system, and is described below.

A macro library is shown in the listing, called DISKDEF, included on line 2, and referenced in 12-15. Although it is not necessary to use the macro library, it greatly simplifies the disk definition process. You must have access to the MAC macro assembler, of course, to use the DISKDEF facility, while the macro library is included with all CP/M 2.0 distribution disks. (See the CP/M 2.0 Alteration Guide for formulas which you can use to hand-code the tables produced by the DISKDEF library).

A BIOS disk definition consists of the following sequence of macro statements:

```
MACLIB   DISKDEF
......
DISKS    n
DISKDEF  0,...
DISKDEF  1,...
......
DISKDEF  n-1
......
ENDEF
```

where the MACLIB statement loads the DISKDEF.LIB file (on the same disk as your BIOS) into MAC's internal tables. The DISKS macro call follows, which specifies the number of drives to be configured with your system, where n is an integer in the range 1 to 16. A series of DISKDEF macro calls then follow which define the characteristics of each logical disk, 0 through n-1 (corresponding to logical drives A through P). Note that the DISKS and DISKDEF macros generate in-line

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

fixed data tables, and thus must be placed in a non-executable portion of your BIOS, typically directly following the BIOS jump vector.

The remaining portion of your BIOS is defined following the DISKDEF macros, with the ENDEF macro call immediately preceding the END statement. The ENDEF (End of Diskdef) macro generates the necessary uninitialized RAM areas which are located above your BIOS.

The form of the DISKDEF macro call is

```
DISKDEF dn,fsc,lsc,[skf],bls,dks,dir,cks,ofs,[0]
```

where

dn	is the logical disk number, 0 to n-1
fsc	is the first physical sector number (0 or 1)
lsc	is the last sector number
skf	is the optional sector skew factor
bls	is the data allocation block size
dir	is the number of directory entries
cks	is the number of "checked" directory entries
ofs	is the track offset to logical track 00
[0]	is an optional 1.4 compatibility flag

The value "dn" is the drive number being defined with this DISKDEF macro invocation. The "fsc" parameter accounts for differing sector numbering systems, and is usually 0 or 1. The "lsc" is the last numbered sector on a track. When present, the "skf" parameter defines the sector skew factor which is used to create a sector translation table according to the skew. If the number of sectors is less than 256, a single-byte table is created, otherwise each translation table element occupies two bytes. No translation table is created if the skf parameter is omitted (or equal to 0). The "bls" parameter specifies the number of bytes allocated to each data block, and takes on the values 1024, 2048, 4096, 8192, or 16384. Generally, performance increases with larger data block sizes since there are fewer directory references and logically connected data records are physically close on the disk. Further, each directory entry addresses more data and the BIOS-resident ram space is reduced. The "dks" specifies the total disk size in "bls" units. That is, if the bls = 2048 and dks = 1000, then the total disk capacity is 2,048,000 bytes. If dks is greater than 255, then the block size parameter bls must be greater than 1024. The value of "dir" is the total number of directory entries which may exceed 255, if desired. The "cks" parameter determines the number of directory items to check on each directory scan, and is used internally to detect changed disks during system operation, where an intervening cold or warm start has not occurred (when this situation is detected, CP/M automatically marks the disk read/only so that data is not subsequently destroyed). Normally the value of cks = dir when the media is easily changed, as is the case with a floppy disk subsystem. If the disk is permanently mounted, then the value of cks is typically 0, since the probability of changing disks without a restart is quite low. The "ofs" value determines the number of tracks to skip when this particular drive is addressed, which can be used to reserve additional operating system

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

space or to simulate several logical drives on a single large capacity physical drive. Finally, the [Ø] parameter is included when file compatibility is required with versions of 1.4 which have been modified for higher density disks. This parameter ensures that only 16K is allocated for each directory record, as was the case for previous versions. Normally, this parameter is not included.

For convenience and economy of table space, the special form

```
DISKDEF    i,j
```

gives disk i the same characteristics as a previously defined drive j. A standard four-drive single density system, which is compatible with version 1.4, is defined using the following macro invocations:

```
DISKS      4
 DISKDEF   Ø,1,26,6,1Ø24,243,64,64,2
 DISKDEF   1,Ø
 DISKDEF   2,Ø
 DISKDEF   3,Ø

 ....
 ENDEF
```

with all disks having the same parameter values of 26 sectors per track (numbered 1 through 26), with 6 sectors skipped between each access, 1Ø24 bytes per data block, 243 data blocks for a total of 243k byte disk capacity, 64 checked directory entries, and two operating system tracks.

The definitions given in the program shown above (lines 12 through 15) provide access to the largest disks addressable by CP/M 2.Ø. All disks have identical parameters, except that drives Ø and 2 skip three sectors on every data access, while disks 1 and 3 access each sector in sequence as the disk revolves (there may, however, be a transparent hardware skew factor on these drives).

The DISKS macro generates n "disk header blocks," starting at address DPBASE which is a label generated by the macro. Each disk header block contains sixteen bytes, and correspond, in sequence, to each of the defined drives. In the four drive standard system, for example, the DISKS macro generates a table of the form:

```
DPBASE  EQU  $
DPEØ:   DW   XLTØ,ØØØØH,ØØØØH,ØØØØH,DIRBUF,DPBØ,CSVØ,ALVØ
DPE1:   DW   XLTØ,ØØØØH,ØØØØH,ØØØØH,DIRBUF,DPBØ,CSV1,ALV1
DPE2:   DW   XLTØ,ØØØØH,ØØØØH,ØØØØH,DIRBUF,DPBØ,CSV2,ALV2
DPE3:   DW   XLTØ,ØØØØH,ØØØØH,ØØØØH,DIRBUF,DPBØ,CSV3,ALV3
```

where the DPE (disk parameter entry) labels are included for reference purposes to show the beginning table addresses for each drive Ø through 3. The values contained within the disk parameter header are described in detail in the CP/M 2.Ø Alteration Guide, but basically address the translation vector for the drive (all reference XLTØ, which is the translation vector for drive Ø in the above example),

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

followed by three 16-bit "scratch" addresses, followed by the directory buffer address, disk parameter block address, check vector address, and allocation vector address. The check and allocation vector addresses are generated by the ENDEF macro in the ram area following the BIOS code and tables.

The SELDSK function is extended somewhat in version 2.0. In particular, the selected disk number is passed to the BIOS in register C, as before, and the SELDSK subroutine performs the appropriate software or hardware actions to select the disk. Version 2.0, however, also requires the SELDSK subroutine to return the address of the selected disk parameter header (DPE0, DPE1, DPE2, or DPE3, in the above example) in register HL. If SELDSK returns the value HL = 0000H, then the BDOS assumes the disk does not exist, and prints a select error mesage at the terminal. Program lines 22 through 36 give a sample CP/M 2.0 SELDSK subroutine, showing only the disk parameter header address calculation.

The subroutine SECTRAN is also included in version 2.0 which performs the actual logical to physical sector translation. In earlier versions of CP/M, the sector translation process was a part of the BDOS, and set to skip six sectors between each read. Due differing rotational speeds of various disks, the translation function has become a part of the BIOS in version 2.0. Thus, the BDOS sends sequential sector numbers to SECTRAN, starting at sector number 0. The SECTRAN subroutine uses the sequential sector number to produce a translated sector number which is returned to the BDOS. The BDOS subsequently sends the translated sector number to SELSEC before the actual read or write is performed. Note that many controllers have the capability to record the sector skew on the disk itself, and thus there is no translation necessary. In this case, the "skf" parameter is omitted in the macro call, and SECTRAN simply returns the same value which it receives. The table shown below, for example, is constructed when the standard skew factor skf = 6 is specified in the DISKDEF macro call:

```
XLT0:  DB    1,7,13,19,25,5,11,17,23,3,9,15,21
       DB    2,8,14,20,26,6,12,18,24,4,10,16,22
```

If SECTRAN is required to translate a sector, then the following process takes place. The sector to translate is received in register pair BC. Only the C register is significant if the sector value does not exceed 255 (B = 00 in this case). Register pair DE addresses the sector translate table for this drive, determined by a previous call on SELDSK, corresponding to the first element of a disk parameter header (XLT0 in the case shown above). The SECTRAN subroutine then fetches the translated sector number by adding the input sector number to the base of the translate table, to get the indexed translate table address (see lines 46, 47, and 48 in the above program). The value at this location is then returned in register L. Note that if the number of sectors exceeds 255, the translate table contains 16-bit elements whose value must be returned in HL.

Following the ENDEF macro call, a number of uninitialized data areas are defined. These data areas need not be a part of the BIOS

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

which is loaded upon cold start, but must be available between the BIOS and the end of memory. The size of the uninitialized RAM area is determined by EQU statements generated by the ENDEF macro. For a standard four-drive system, the ENDEF macro might produce

```
4C72 =          BEGDAT EQU $
                (data areas)
4DBØ =          ENDDAT EQU $
Ø13C =          DATSIZ EQU $-BEGDAT
```

which indicates that uninitialized RAM begins at location 4C72H, ends at 4DBØH-1, and occupies Ø13CH bytes. You must ensure that these addresses are free for use after the system is loaded.

CP/M 2.Ø is also easily adapated to disk subsystems whose sector size is a multiple of 128 bytes. Information is provided by the BDOS on sector write operations which eliminates the need for pre-read operations, thus allowing blocking and deblocking to take place at the BIOS level.

See the "CP/M 2.Ø Alteration Guide" for additional details concerning tailoring your CP/M system to your particular hardware.