CP/M
Software Manual

Exidy Systems
# Table Of Contents

**CP/M 2.2**

## Section:
**CP/M 2 User's Guide**

1. An Overview of CP/M 2.0 Facilities.............1
2. User Interface........................................3
3. Console Command Processor (CCP) Interface..4
4. STAT Enhancements....................................5
5. PIP Enhancements.....................................8
6. ED Enhancements.....................................10
7. The XSUB Function.................................11
8. BDOS Interface Conventions.....................12
9. CP/M 2.0 Memory Organization..................27
10. BIOS Differences..................................28

## Section:
**An Introduction To CP/M Features and Facilities**

1. Introduction..........................................35
2. Functional Description Of CP/M..................37
   2.1 General Command Structure.........................37
   2.2 File References....................................37
3. Switching Disks.......................................40
4. The Form Of Built-In Commands.....................41
   4.1 ERA ufn cr.......................................41
   4.2 DIR ufn cr.......................................42
   4.3 REN ufn1=ufn2 cr................................42
   4.4 SAVE n ufn cr....................................43
   4.5 TYPE ufn cr.....................................43
5. Line Editing And Output Control...................45
6. Transient Commands..................................46
   6.1 STAT cr.........................................47
   6.2 ASM ufn cr.......................................50
   6.3 LOAD ufn cr.....................................51
   6.4 PIP cr...........................................52
   6.5 ED ufn cr........................................59
   6.6 SYSGEN cr........................................61
   6.7 SUBMIT ufn parm#1 parm#n cr.....................62
   6.8 DUMP ufn cr.....................................64
   6.9 MOVCPM cr.......................................64.
7. BDOS Error Messages.................................67
8. Operation Of CP/M On The MDS.....................69
Section:
Exidy Systems' CB IOS User's Guide:
Version 1.0 For CP/M 2.2

1. Introduction..........................249
2. Configuration and System Generation.....251
   A. Hardware for the DDS and the FDS.....251
   B. System Generation....................252
   C. Options..............................253
   D. Incompatibilities....................254
   E. Sector Skew Pattern..................254
   F. Special Video Display Interface......255
3. Features................................256
   A. Error Recovery........................256
   B. CP/M Programming....................257
4. Error Messages..........................259

Section:

1. Introduction..........................262
2. Use...................................263
3. Sample Run............................264
   A. Ex copy With Two Multiple Drives.....264
   B. Ex copy With One Drive................264
   C. Format Only...........................265
4. Error Messages..........................266
   A. Can Not Format, Try Again............266
   B. Destination Is Write Protected......266
   C. Write Error On Track #XX..............267
   D. Read Back Error On Track #XX.........267
   E. Additional Messages...................267
5. Recovery.................................268
CP/M 2 USER'S GUIDE
COPYRIGHT

Copyright (c) 1980 by Exidy Systems, Inc. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of Exidy Systems, Inc., 1234 Elko Drive, Sunnyvale, California 94086.

Since this manual is tutorial in nature, permission is granted to reproduce or abstract the example procedures and sample programs for the purposes of inclusion within the reader's programs.

DISCLAIMER

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

TRADEMARKS

CP/M is a registered trademark of Digital Research.
COPYRIGHT

Copyright (c) 1980 by Exidy Systems, Inc. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of Exidy Systems, Inc., 1234 Elko Drive, Sunnyvale, California 94086.

Since this manual is tutorial in nature, permission is granted to reproduce or abstract the example procedures and sample programs for the purposes of inclusion within the reader's programs.

DISCLAIMER

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

TRADEMARKS

CP/M is a registered trademark of Digital Research.
1. An Overview of CP/M 2.0 Facilities ........................ 1
2. User Interface ........................................ 3
3. Console Command Processor (CCP) Interface ............ 4
4. STAT Enhancements ....................................... 5
5. PIP Enhancements ......................................... 8
6. ED Enhancements ......................................... 10
7. The XSUB Function ...................................... 11
8. 3DOS Interface Conventions .............................. 12
9. CP/M 2.0 Memory Organization ......................... 27
10. 3IOS Differences ...................................... 28
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.

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

\[ \text{STAT VAL:} \]

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

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

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

\[ \text{STAT } d:\text{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:

<table>
<thead>
<tr>
<th>Size</th>
<th>Recs</th>
<th>Bytes</th>
<th>Ext Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>48</td>
<td>6k</td>
<td>1 R/O A:ED.COM</td>
</tr>
<tr>
<td>55</td>
<td>55</td>
<td>12k</td>
<td>1 R/O (A:PIP.COM)</td>
</tr>
<tr>
<td>65536</td>
<td>128</td>
<td>2k</td>
<td>2 R/W A:X.DAT</td>
</tr>
</tbody>
</table>

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

```plaintext
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

```plaintext
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

```plaintext
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:

```plaintext
d: Drive Characteristics
65536: 128 Byte record Capacity
8192: Kilobyte Drive Capacity
128: 32 Byte Directory Entries
0: Checked Directory Entries
1024: 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 (1024 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.

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

PIP provides three new functions which account for the features of CP/M 2.0. 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 0 - 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

```
PPIP X.Y = X.Y[G2]
```
reads file X.Y from user number 2 into user area number 4. The command

```
PPIP 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 0  login user 0
DDT PIP.COM load PIP to memory
(note PIP size s)
G0  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 1D00, 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.)
```
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 skipped, 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.
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
GØ
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 "GØ" 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.

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

(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., ct1-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 005C8 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.)
where

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

\( \text{f1...f8} \)  
contain the file name in ASCII upper case, with high bit = 0

\( \text{t1,t2,t3} \)  
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

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

\( \text{s1} \)  
reserved for internal system use

\( \text{s2} \)  
reserved for internal system use, set to zero on call to OPEN, MAKE, SEARCH

\( \text{rc} \)  
record count for extent "ex," takes on values from 0 - 128

\( \text{d0...dn} \)  
filled-in by CP/M, reserved for system use

\( \text{cr} \)  
current record to read or write in a sequential file operation, normally set to zero by user

\( \text{r0,rl,r2} \)  
optional random record number in the range 0-65535, with overflow to r2, r0,rl constitute a 16-bit value with low byte r0, and high byte rl

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
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 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.

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 0 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 0 to 3 is returned.

Function 24: Return Login Vector.

The login vector value returned by CP/M 2.0 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 30: 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 30 searches for a

(All Information Contained Herein is Proprietary to Digital Research.)
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 hexadecminal, 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 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 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 release 2.0 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 2.0, 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

(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.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>reading unwritten data</td>
</tr>
<tr>
<td>02</td>
<td>(not returned in random mode)</td>
</tr>
<tr>
<td>03</td>
<td>cannot close current extent</td>
</tr>
<tr>
<td>04</td>
<td>seek to unwritten extent</td>
</tr>
<tr>
<td>05</td>
<td>(not returned in read mode)</td>
</tr>
<tr>
<td>06</td>
<td>seek past physical end of disk</td>
</tr>
</tbody>
</table>

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

(All Information Contained Herein is Proprietary to Digital Research.)
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
0d10 = closef equ 16 ;close function
0016 = makef equ 22 ;make file function
0v21 = 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 11b0 lxi d,badver
0110 cdaa0 call print
0113 c3000 jmp reboot

; versok:
; correct version for random access

(All Information Contained Herein is Proprietary to Digital Research.) 21
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  snld  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,closetf
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.)

22
rloop: ;read next character to buff
    push b ;save counter
    push h ;next destination
    call getchr ;character to a
    pop h ;restore counter
    pop b ;restore next to fill
    cpi cr ;end of line?
    jz erloop ;not end, store character

erloop: ;end of read loop, store 00
    mvi m,0 ;write the record to selected record number
    mvi c,writer
    lxi d,fcb
    call bdos
    ora a ;error code zero?
    jnz error ;message if not
    mvi c,readr
    lxi d,fcb
    call bdos
    ora a ;return code 00?
    jnz error
    jmp ready ;for another record

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;; end of write command, process read
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
notw: not a write command, read record?
    cpi 'R'
    jnz error ;skip if not

; read random record
    mvi c,readr
    lxi d,fcb
    call bdos
    ora a
    jnz error

; read was successful, write to console
    call crlf ;new line
    mvi c,128 ;max 128 characters
    lxi h,buff ;next to get

wloop: ;next character
    mov a,m
    inx h ;next to get
    ani 7fh ;mask parity
    jz ready ;for another command if 00
    push b ;save counter
    push h ;save next to get

(All Information Contained Herein is Proprietary to Digital Research.)
; graphic?
cpi
; skip output if not
cnc
putchr
pop h
pop b
dcr c
; count = count - 1
dlb3 c2a20
jnz wloop
jmp ready

;******************************************************
;* end of read command, all errors end up here *
;******************************************************

error:

1xi d,errmsg

1bc cdda0
1bf c3370

jmp ready

;******************************************************
;* utility subroutines for console i/o *
;******************************************************

getchr:
; read next console character to a
1c2 0e01
1c4 cd050
1c7 c9

mvi c,coninp

call bdos

ret

; putchar:
; write character from a to console
1c8 0e02
1ca 5f
1cb cd050
1ce c9

mvi c,conout

mov e,a
; character to send

call bdos
; send character

ret

; crlf:
; send carriage return line feed
1cf 3e0d
1d1 cd080
1d4 3e0a
1d6 cd080
1d9 c9

mvi a,cr
; carriage return

call putchr

mvi a,lf
; line feed

call putchr

ret

; print:
; print the buffer addressed by de until $
1da d5
1db cd0f0
1de d1
1df 0e09
1el cd050
1e4 c9

push d

call crlf

pop d
; new line

mvi c, pstring

call bdos
; print the string

ret

; readcom:

(All Information Contained Herein is Proprietary to Digital Research.)
; read the next command line to the conbuf

;START
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 la 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
; not zero, numeric?
01fd d630 sui '0'
01ff fe0a cpi l0 ;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

;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 l0l$1111
021a c9 ret

;************************************************************
;* * * string data area for console messages * * *
;* * * ************************************************************
badver:
021b 536f79 db 'sorry, you need cp/m version 2$
023a 4e6f29 db 'no directory space$
024d 547970 db 'type data: $
0259 457272 db 'error, try again.$
026b 4e6570 db 'next command? $

(All Information Contained Herein is Proprietary to Digital Research.)
conbuf: db conlen ; length of console buffer
consiz: ds 1 ; resulting size after read
conlin: ds 32 ; length 32 buffer
conlen equ $-consiz

stack: ds 32 ; 16 level stack

end
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.

<table>
<thead>
<tr>
<th>Module</th>
<th>20k</th>
<th>24k</th>
<th>32k</th>
<th>48k</th>
<th>64k</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCP</td>
<td>3400H</td>
<td>4400H</td>
<td>6400H</td>
<td>A400H</td>
<td>E400H</td>
</tr>
<tr>
<td>BIOS</td>
<td>3C00H</td>
<td>4C00H</td>
<td>6C00H</td>
<td>AC00H</td>
<td>EC00H</td>
</tr>
<tr>
<td>Top of Ram</td>
<td>4A00H</td>
<td>5A00H</td>
<td>7A00H</td>
<td>BA00H</td>
<td>FA00H</td>
</tr>
</tbody>
</table>

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:

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 predecessors. 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.

```
1:  org 4000h
2:  maclib diskdef
3:  jmp boot
4:  ...
5:  jmp listst ; list status
6:  jmp sectran ; sector translate
7:  disks 4
8:  ; large capacity drive
9:  bpb equ 16*1024 ; bytes per block
10: rpb equ bpb/128 ; records per block
11: maxb equ 65535/rpb ; max block number
12: diskdef 0,1,58,3,bpb,maxb+1,128,0,2
13: diskdef 1,1,58,,bpb,maxb+1,128,0,2
14: diskdef 2,0
15: diskdef 3,1
16: ;
17: boot: ret ; nop
18: ;
19: listst: xra a ; nop
20: ret
21: ;
22: seldsk:
23: ; drive number in c
24: lxi h,0 ; 0000 in HL produces select error
25: mov a,c ; a is disk number 0 ... ndisks-1
26: cpi ndisks ; less than ndisks?
27: rnc ; return with HL = 0000 if not
28: ; proper disk number, return dpb element address
29: mov l,c
30: dad h ; *2
31: dad h ; *4
32: dad h ; *8
33: dad h ; *16
34: lxi d,dpbase
35: dad d ; HL=.dpb
36: ret
37: ;
38: selsec:
39: ; sector number in c
40: lxi h,sector
41: mov m,c
42: ret
43: ;
44: sectran:
45: ; translate sector BC using table at DE
46: xchg ; HL = .tran
47: dad b ; single precision tran
```

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

\[
\text{DISKDEF } \text{dn}, \text{fsc}, \text{lsc}, [\text{skf}], \text{bls}, \text{dks}, \text{dir}, \text{cks}, \text{ofs}, [0]
\]

where

- \text{dn} is the logical disk number, 0 to n-1
- \text{fsc} is the first physical sector number (0 or 1)
- \text{lsc} is the last sector number
- \text{skf} is the optional sector skew factor
- \text{bls} is the data allocation block size
- \text{dir} is the number of directory entries
- \text{cks} is the number of "checked" directory entries
- \text{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
space or to simulate several logical drives on a single large capacity physical drive. Finally, the [0] 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

\texttt{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:

\begin{verbatim}
DISKS 4
DISKDEF 0,1,26,6,1024,243,64,64,2
DISKDEF 1,0
DISKDEF 2,0
DISKDEF 3,0

....
ENDEF
\end{verbatim}

with all disks having the same parameter values of 26 sectors per track (numbered 1 through 26), with 6 sectors skipped between each access, 1024 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 2.0. All disks have identical parameters, except that drives 0 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 \texttt{DISKS} macro generates n "disk header blocks," starting at address \texttt{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 \texttt{DISKS} macro generates a table of the form:

\begin{verbatim}
DPBASE EQU $
DPE0: DW XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV0,ALV0
DPE1: DW XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV1,ALV1
DPE2: DW XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV2,ALV2
DPE3: DW XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV3,ALV3
\end{verbatim}

where the DPE (disk parameter entry) labels are included for reference purposes to show the beginning table addresses for each drive 0 through 3. The values contained within the disk parameter header are described in detail in the CP/M 2.0 Alteration Guide, but basically address the translation vector for the drive (all reference XLT0, which is the translation vector for drive 0 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 message 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

\[
\begin{align*}
4C72 & \quad \text{BEGDAT EQU } \$ \\
& \quad \text{(data areas)} \\
4DB0 & \quad \text{ENDDAT EQU } \$
\end{align*}
\]

\[
\begin{align*}
013C & \quad \text{DATSIZ EQU } \$-\text{BEGDAT}
\end{align*}
\]

which indicates that uninitialized RAM begins at location 4C72H, ends at 4DB0H-1, and occupies 013CH bytes. You must ensure that these addresses are free for use after the system is loaded.

CP/M 2.0 is also easily adapted 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 2.0 Alteration Guide" for additional details concerning tailoring your CP/M system to your particular hardware.

(All Information Contained Herein is Proprietary to Digital Research.)
AN INTRODUCTION TO CP/M FEATURES AND FACILITIES
COPYRIGHT

Copyright (c) 1980 by Exidy Systems, Inc. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of Exidy Systems, Inc., 1234 Elko Drive, Sunnyvale, California 94086.

Since this manual is tutorial in nature, permission is granted to reproduce or abstract the example procedures and sample programs for the purposes of inclusion within the reader's programs.

DISCLAIMER

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

TRADEMARKS

CP/M is a registered trademark of Digital Research.
AN INTRODUCTION TO CP/M FEATURES AND FACILITIES
COPYRIGHT

Copyright (c) 1980 by Exidy Systems, Inc. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of Exidy Systems, Inc., 1234 Elko Drive, Sunnyvale, California 94086.

Since this manual is tutorial in nature, permission is granted to reproduce or abstract the example procedures and sample programs for the purposes of inclusion within the reader's programs.

DISCLAIMER

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

TRADEMARKS

CP/M is a registered trademark of Digital Research.
# Table Of Contents

An Introduction To CP/M Features and Facilities

1. **Introduction**.................................35
2. **Functional Description Of CP/M**.............37
   2.1 General Command Structure..................37
   2.2 File References............................37
3. **Switching Disks**..............................40
4. **The Form Of Built-in Commands**.............41
   4.1 ERA afn cr..................................41
   4.2 DIR afn cr..................................42
   4.3 REN ufn1=ufn2 cr............................42
   4.4 SAVE n ufn cr................................43
   4.5 TYPE ufn cr..................................43
5. **Line Editing And Output Control**............45
6. **Transient Commands**..........................46
   6.1 STAT cr......................................47
   6.2 ASM ufn cr....................................50
   6.3 LOAD ufn cr...................................51
   6.4 PIP cr........................................52
   6.5 ED ufn cr......................................59
   6.6 SYSGEN cr.....................................61
   6.7 SUBMIT ufn parm#1 parm#n cr...............62
   6.8 DUMP ufn cr...................................64
   6.9 MOVCPM cr......................................64
7. **BDOS Error Messages**........................67
8. **Operation Of CP/M On The MDS**..............69
1. INTRODUCTION.

CP/M is a monitor control program for microcomputer system development which uses IBM-compatible flexible disks for backup storage. Using a computer mainframe based upon Intel's 8080 microcomputer, CP/M provides a general environment for program construction, storage, and editing, along with assembly and program check-out facilities. An important feature of CP/M is that it can be easily altered to execute with any computer configuration which uses an Intel 8080 (or Zilog Z-80) Central Processing Unit, and has at least 16K bytes of main memory with up to four IBM-compatible diskette drives. A detailed discussion of the modifications required for any particular hardware environment is given in the Digital Research document entitled "CP/M System Alteration Guide." Although the standard Digital Research version operates on a single-density Intel 8080, several different hardware manufacturers support their own input-output drivers for CP/M.

The CP/M monitor provides rapid access to programs through a comprehensive file management package. The file subsystem supports a named file structure, allowing dynamic allocation of file space as well as sequential and random file access. Using this file system, a large number of distinct programs can be stored in both source and machine executable form.

CP/M also supports a powerful context editor, Intel-compatible assembler, and debugger subsystems. Optional software includes a powerful Intel-compatible macro assembler, symbolic debugger, along with various high-level languages. When coupled with CP/M's Console Command Processor, the resulting facilities equal or excel similar large computer facilities.

CP/M is logically divided into several distinct parts:

- BIOS            Basic I/O System (hardware dependent)
- BDOS            Basic Disk Operating System
- CCP             Console Command Processor
- TPA             Transient Program Area

The BIOS provides the primitive operations necessary to access the diskette drives and to interface standard peripherals (teletype, CRT, Paper Tape Reader/Punch, and user-defined peripherals), and can be tailored by the user for any particular hardware environment by "patching" this portion of CP/M. The BDOS provides disk management by controlling one or more disk drives containing independent file directories. The BDOS implements disk allocation strategies which provide fully dynamic file construction while minimizing head movement across the disk during access. Any particular file may contain any number of records, not exceeding the size of any single disk. In a standard CP/M system, each disk can contain up to 64 distinct files. The
BDOS has entry points which include the following primitive operations which can be programmatically accessed:

- **SEARCH**: Look for a particular disk file by name.
- **OPEN**: Open a file for further operations.
- **CLOSE**: Close a file after processing.
- **RENAME**: Change the name of a particular file.
- **READ**: Read a record from a particular file.
- **WRITE**: Write a record onto the disk.
- **SELECT**: Select a particular disk drive for further operations.

The CCP provides symbolic interface between the user's console and the remainder of the CP/M system. The CCP reads the console device and processes commands which include listing the file directory, printing the contents of files, and controlling the operation of transient programs, such as assemblers, editors, and debuggers. The standard commands which are available in the CCP are listed in a following section.

The last segment of CP/M is the area called the Transient Program Area (TPA). The TPA holds programs which are loaded from the disk under command of the CCP. During program editing, for example, the TPA holds the CP/M text editor machine code and data areas. Similarly, programs created under CP/M can be checked out by loading and executing these programs in the TPA.

It should be mentioned that any or all of the CP/M component subsystems can be "overlayed" by an executing program. That is, once a user's program is loaded into the TPA, the CCP, BDOS, and BIOS areas can be used as the program's data area. A "bootstrap" loader is programmatically accessible whenever the BIOS portion is not overlayed; thus, the user program need only branch to the bootstrap loader at the end of execution, and the complete CP/M monitor is reloaded from disk.

It should be reiterated that the CP/M operating system is partitioned into distinct modules, including the BIOS portion which defines the hardware environment in which CP/M is executing. Thus, the standard system can be easily modified to any non-standard environment by changing the peripheral drivers to handle the custom system.
2. FUNCTIONAL DESCRIPTION OF CP/M.

The user interacts with CP/M primarily through the CCP, which reads and interprets commands entered through the console. In general, the CCP addresses one of several disks which are online (the standard system addresses up to four different disk drives). These disk drives are labelled A, B, C, and D. A disk is "logged in" if the CCP is currently addressing the disk. In order to clearly indicate which disk is the currently logged disk, the CCP always prompts the operator with the disk name followed by the symbol ">" indicating that the CCP is ready for another command. Upon initial start up, the CP/M system is brought in from disk A, and the CCP displays the message

\[ xxK \text{ CP/M VER } m,m \]

where xx is the memory size (in kilobytes) which this CP/M system manages, and \( m,m \) is the CP/M version number. All CP/M systems are initially set to operate in a 16K-memory space, but can be easily reconfigured to fit any memory size on the host system (see the MOCVM transient command). Following system signon, CP/M automatically logs in disk A, prompts the user with the symbol ">" (indicating that CP/M is currently addressing disk "A"), and waits for a command. The commands are implemented at two levels: built-in commands and transient commands.

2.1. GENERAL COMMAND STRUCTURE.

Built-in commands are a part of the CCP program itself, while transient commands are loaded into the TPA from disk and executed. The built-in commands are

- **ERA**: Erase specified files.
- **DIR**: List file names in the directory.
- **REN**: Rename the specified file.
- **SAVE**: Save memory contents in a file.
- **TYPE**: Type the contents of a file on the logged disk.

Nearly all of the commands reference a particular file or group of files. The form of a file reference is specified below.

2.2. FILE REFERENCES.

A file reference identifies a particular file or group of files on a particular disk attached to CP/M. These file references can be either "unambiguous" (ufn) or "ambiguous" (afn). An unambiguous file reference uniquely identifies a single file, while an ambiguous file reference may be
satisfied by a number of different files.

File references consist of two parts: the primary name and the secondary name. Although the secondary name is optional, it usually is generic; that is, the secondary name "ASM," for example, is used to denote that the file is an assembly language source file, while the primary name distinguishes each particular source file. The two names are separated by a "." as shown below:

```
ppppppppp.sss
```

where ppppppppp represents the primary name of eight characters or less, and sss is the secondary name of no more than three characters. As mentioned above, the name

```
pppppppp
```

is also allowed and is equivalent to a secondary name consisting of three blanks. The characters used in specifying an unambiguous file reference cannot contain any of the special characters

```
< > . , ; : = ? * [ ]
```

while all alphanumerics and remaining special characters are allowed.

An ambiguous file reference is used for directory search and pattern matching. The form of an ambiguous file reference is similar to an unambiguous reference, except the symbol "?" may be interspersed throughout the primary and secondary names. In various commands throughout CP/M, the "?" symbol matches any character of a file name in the "?" position. Thus, the ambiguous reference

```
X?Z.C?M
```

is satisfied by the unambiguous file names

```
XYZ.COM
```

and

```
XZ.CAM
```

Note that the ambiguous reference

```
.*
```

is equivalent to the ambiguous file reference

```
?????????,???
```

while
and 

pppppppp.*

*.*

are abbreviations for

pppppppp.???

and

?????????????.*

respectively. As an example,

DIR *. *

is interpreted by the CCP as a command to list the names of all disk files in the directory, while

DIR X.Y

searches only for a file by the name X.Y Similarly, the command

DIR X?Y.C?M

causes a search for all (unambiguous) file names on the disk which satisfy this ambiguous reference.

The following file names are valid unambiguous file references:

X

XY

XYZ

GAMMA

X.Y

XYZ.COM

GAMMA1

As an added convenience, the programmer can generally specify the disk drive name along with the file name. In this case, the drive name is given as a letter A through Z followed by a colon (:). The specified drive is then "logged in" before the file operation occurs. Thus, the following are valid file names with disk name prefixes:

A:X.Y

B:XY

C:GAMMA

Z:XYZ.COM

B:X.A?M

C:*.*ASM

It should also be noted that all alphabetic lower case letters in file and drive names are always translated to upper case when they are processed by the CCP.
3. SWITCHING DISKS.

The operator can switch the currently logged disk by typing the disk drive name (A, B, C, or D) followed by a colon (:) when the CCP is waiting for console input. Thus, the sequence of prompts and commands shown below might occur after the CP/M system is loaded from disk A:

16K CP/M VER 1.4

A>DIR List all files on disk A.
SAMPLE  ASM
SAMPLE  PRN
A>B: Switch to disk B.
B>DIR *.ASM List all "ASM" files on B.
DUMP  ASM
FILES  ASM
B>A: Switch back to A.
4. THE FORM OF BUILT-IN COMMANDS.

The file and device reference forms described above can now be used to fully specify the structure of the built-in commands. In the description below, assume the following abbreviations:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ufn</td>
<td>unambiguous file reference</td>
</tr>
<tr>
<td>afn</td>
<td>ambiguous file reference</td>
</tr>
<tr>
<td>cr</td>
<td>carriage return</td>
</tr>
</tbody>
</table>

Further, recall that the CCP always translates lower case characters to upper case characters internally. Thus, lower case alphabetics are treated as if they are upper case in command names and file references.

4.1 ERA afn cr

The ERA (erase) command removes files from the currently logged-in disk (i.e., the disk name currently prompted by CP/M preceding the ">"). The files which are erased are those which satisfy the ambiguous file reference afn. The following examples illustrate the use of ERA:

- **ERA X.Y**
  - The file named X.Y on the currently logged disk is removed from the disk directory, and the space is returned.

- **ERA X.***
  - All files with primary name X are removed from the current disk.

- **ERA *.*ASM**
  - All files with secondary name ASM are removed from the current disk.

- **ERA X?Y.C?M**
  - All files on the current disk which satisfy the ambiguous reference X?Y.C?M are deleted.

- **ERA *.***
  - Erase all files on the current disk (in this case the CCP prompts the console with the message "ALL FILES (Y/N)?" which requires a Y response before files are actually removed).

- **ERA B:*.*PRN**
  - All files on drive B which satisfy the ambiguous reference ??????????.PRN are deleted, independently of the currently logged disk.
4.2. DIR ufn cr

The DIR (directory) command causes the names of all files which satisfy the ambiguous file name ufn to be listed at the console device. As a special case, the command

```
DIR
```

lists the files on the currently logged disk (the command "DIR" is equivalent to the command "DIR *.*"). Valid DIR commands are shown below.

```
DIR X.Y
DIR X?Z.C?M
DIR ??Y
```

Similar to other CCP commands, the ufn can be preceded by a drive name. The following DIR commands cause the selected drive to be addressed before the directory search takes place.

```
DIR B:
DIR B:X.Y
DIR B:*.*
```

If no files can be found on the selected diskette which satisfy the directory request, then the message "NOT FOUND" is typed at the console.

4.3. REN ufn1=ufn2 cr

The REN (rename) command allows the user to change the names of files on disk. The file satisfying ufn2 is changed to ufn1. The currently logged disk is assumed to contain the file to rename (ufn1). The CCP also allows the user to type a left-directed arrow instead of the equal sign, if the user's console supports this graphic character. Examples of the REN command are

```
REN X.Y=Q.R The file Q.R is changed to X.Y.
REN XYZ.COM=XYZ.XXX The file XYZ.XXX is changed to XYZ.COM.
```

The operator can precede either ufn1 or ufn2 (or both) by an optional drive address. Given that ufn1 is preceded by a drive name, then ufn2 is assumed to exist on the same drive as ufn1. Similarly, if ufn2 is preceded by a drive name, then ufn1 is assumed to reside on that drive as well. If both ufn1 and ufn2 are preceded by drive names, then the same drive must be
specified in both cases. The following REN commands illustrate this format.

REN A:X.ASM = Y.ASM  The file Y.ASM is changed to X.ASM on drive A.

REN B:ZAP.BAS=ZOT.BAS  The file ZOT.BAS is changed to ZAP.BAS on drive B.

REN B:A.ASM = B:A.BAR  The file A.BAR is renamed to A.ASM on drive B.

If the file ufn1 is already present, the REN command will respond with the error "FILE EXISTS" and not perform the change. If ufn2 does not exist on the specified diskette, then the message "NOT FOUND" is printed at the console.

4.4. SAVE n ufn cr

The SAVE command places n pages (256-byte blocks) onto disk from the TPA and names this file ufn. In the CP/M distribution system, the TPA starts at 100H (hexadecimal), which is the second page of memory. Thus, if the user's program occupies the area from 100H through 2FFH, the SAVE command must specify 2 pages of memory. The machine code file can be subsequently loaded and executed. Examples are:

SAVE 3 X.COM  Copies 100H through 3FFH to X.COM.

SAVE 40 Q  Copies 100H through 2FFH to Q (note that 28 is the page count in 2FFH, and that 28H = 2*16+8 = 40 decimal).

SAVE 4 X.Y  Copies 100H through 4FFH to X.Y.

The SAVE command can also specify a disk drive in the afn portion of the command, as shown below.

SAVE 10 B:ZOT.COM  Copies 10 pages (100H through 0AFFH) to the file ZOT.COM on drive B.

4.5. TYPE ufn cr

The TYPE command displays the contents of the ASCII source file ufn on the currently logged disk at the console device. Valid TYPE commands are

TYPE X.Y

43
The **TYPE** command expands tabs (clt-I characters), assuming tab positions are set at every eighth column. The ufn can also reference a drive name as shown below.

**TYPE** B:X.PRN  

The file *X.PRN* from drive B is displayed.
5. **LINE EDITING AND OUTPUT CONTROL.**

The CCP allows certain line editing functions while typing command lines.

- **rubout** Delete and echo the last character typed at the console.
- **ctl-U** Delete the entire line typed at the console.
- **ctl-X** (Same as ctl-U)
- **ctl-R** Retype current command line: types a "clean line" following character deletion with rubouts.
- **ctl-E** Physical end of line: carriage is returned, but line is not sent until the carriage return key is depressed.
- **ctl-C** CP/M system reboot (warm start)
- **ctl-Z** End input from the console (used in PIP and ED).

The control functions **ctl-P** and **ctl-S** affect console output as shown below.

- **ctl-P** Copy all subsequent console output to the currently assigned list device (see the STAT command). Output is sent to both the list device and the console device until the next ctl-P is typed.
- **ctl-S** Stop the console output temporarily. Program execution and output continue when the next character is typed at the console (e.g., another ctl-S). This feature is used to stop output on high speed consoles, such as CRT's, in order to view a segment of output before continuing.

Note that the ctl-key sequences shown above are obtained by depressing the control and letter keys simultaneously. Further, CCP command lines can generally be up to 255 characters in length; they are not acted upon until the carriage return key is typed.
6. TRANSIENT COMMANDS.

Transient commands are loaded from the currently logged disk and executed in the TPA. The transient commands defined for execution under the CCP are shown below. Additional functions can easily be defined by the user (see the LOAD command definition).

- **STAT**: List the number of bytes of storage remaining on the currently logged disk, provide statistical information about particular files, and display or alter device assignment.

- **ASM**: Load the CP/M assembler and assemble the specified program from disk.

- **LOAD**: Load the file in Intel "hex" machine code format and produce a file in machine executable form which can be loaded into the TPA (this loaded program becomes a new command under the CCP).

- **DDT**: Load the CP/M debugger into TPA and start execution.

- **PIP**: Load the Peripheral Interchange Program for subsequent disk file and peripheral transfer operations.

- **ED**: Load and execute the CP/M text editor program.

- **SYSGEN**: Create a new CP/M system diskette.

- **SUBMIT**: Submit a file of commands for batch processing.

- **DUMP**: Dump the contents of a file in hex.

- **MOVCPM**: Regenerate the CP/M system for a particular memory size.

Transient commands are specified in the same manner as built-in commands, and additional commands can be easily defined by the user. As an added convenience, the transient command can be preceded by a drive name, which causes the transient to be loaded from the specified drive into the TPA for execution. Thus, the command

```
B:STAT
```

causes CP/M to temporarily "log in" drive B for the source of the STAT transient, and then return to the original logged disk for subsequent processing.
The basic transient commands are listed in detail below.

6.1. STAT cr

The STAT command provides general statistical information about file storage and device assignment. It is initiated by typing one of the following forms:

\[\text{STAT cr}\]
\[\text{STAT "command line" cr}\]

Special forms of the "command line" allow the current device assignment to be examined and altered as well. The various command lines which can be specified are shown below, with an explanation of each form shown to the right.

\[\text{STAT cr}\]

If the user types an empty command line, the STAT transient calculates the storage remaining on all active drives, and prints a message

\[x: R/W, \text{SPACE: nnnK}\]

or

\[x: R/O, \text{SPACE: nnnK}\]

for each active drive \(x\), where \(R/W\) indicates the drive may be read or written, and \(R/O\) indicates the drive is read only (a drive becomes \(R/O\) by explicitly setting it to read only, as shown below, or by inadvertently changing diskettes without performing a warm start). The space remaining on the diskette in drive \(x\) is given in kilobytes by \(nnn\).

\[\text{STAT x: cr}\]

If a drive name is given, then the drive is selected before the storage is computed. Thus, the command "STAT B:" could be issued while logged into drive A, resulting in the message

\[\text{BYTES REMAINING ON B: nnnK}\]

\[\text{STAT afn cr}\]

The command line can also specify a set of files to be scanned by STAT. The files which satisfy afn are listed in alphabetical order, with storage requirements for each file under the heading

\[\text{RECS BYTS EX D:FILENAME,TYP}\]
\[rrrr \text{bbbk ee d:ppppppp,sss}\]

where \(rrrr\) is the number of 128-byte records
allocated to the file, bbb is the number of kilobytes allocated to the file (bbb=rrrr*128/1024), ee is the number of 16K extensions (ee=bbb/16), d is the drive name containing the file (A...Z), ppppppp is the (up to) eight-character primary file name, and sss is the (up to) three-character secondary name. After listing the individual files, the storage usage is summarized.

STAT x:afn cr

As a convenience, the drive name can be given ahead of the afn. In this case, the specified drive is first selected, and the form "STAT afn" is executed.

STAT x:=R/O cr

This form sets the drive given by x to read-only, which remains in effect until the next warm or cold start takes place. When a disk is read-only, the message

EDOS ERR ON x: READ ONLY

will appear if there is an attempt to write to the read-only disk x. CP/M waits until a key is depressed before performing an automatic warm start (at which time the disk becomes R/W).

The STAT command also allows control over the physical to logical device assignment (see the IOBYTE function described in the manuals "CP/M Interface Guide" and "CP/M System Alteration Guide"). In general, there are four logical peripheral devices which are, at any particular instant, each assigned to one of several physical peripheral devices. The four logical devices are named:

CON: The system console device (used by CCP for communication with the operator)
RDR: The paper tape reader device
PUN: The paper tape punch device
LST: The output list device

The actual devices attached to any particular computer system are driven by subroutines in the BIOS portion of CP/M. Thus, the logical RDR: device, for example, could actually be a high speed reader, Teletype reader, or cassette tape. In order to allow some flexibility in device naming and assignment, several physical devices are defined, as shown below:

48
TTY: Teletype device (slow speed console)
CRT: Cathode ray tube device (high speed console)
BAT: Batch processing (console is current RDR:, output goes to current LST: device)
UC1: User-defined console
PTR: Paper tape reader (high speed reader)
URL: User-defined reader #1
UR2: User-defined reader #2
PTP: Paper tape punch (high speed punch)
UP1: User-defined punch #1
UP2: User-defined punch #2
LFT: Line printer
ULL: User-defined list device #1

It must be emphasized that the physical device names may or may not actually correspond to devices which the names imply. That is, the PTP: device may be implemented as a cassette write operation, if the user wishes. The exact correspondence and driving subroutine is defined in the BIOS portion of CP/M. In the standard distribution version of CP/M, these devices correspond to their names on the MDS 800 development system.

The possible logical to physical device assignments can be displayed by typing
`STAT VAL: cr`

The `STAT` prints the possible values which can be taken on for each logical device:

```
CON. = TTY: CRT: BAT: UC1:
RDR. = TTY: PTR: URL: UR2:
PUN. = TTY: PTP: UPl: UP2:
LST. = TTY: CRT: LFT: ULL:
```

In each case, the logical device shown to the left can take any of the four physical assignments shown to the right on each line. The current logical to physical mapping is displayed by typing the command

`STAT DEV: cr`
which produces a listing of each logical device to the left, and the current corresponding physical device to the right. For example, the list might appear as follows:

```
CON: = CRT;
RDR: = URI;
PUN: = FTP;
LST: = TTY;
```

The current logical to physical device assignment can be changed by typing a STAT command of the form

```
STAT ldl = pdl, ld2 = pd2, ..., ldn = pdn cr
```

where ldl through ldn are logical device names, and pdl through pdn are compatible physical device names (i.e., ldi and pdi appear on the same line in the "VAL:" command shown above). The following are valid STAT commands which change the current logical to physical device assignments:

```
STAT CON: = CRT: cr
STAT PUN: = TTY: , LST: = LPT: , RDR: = TTY: cr
```

6.2. ASM ufn cr

The ASM command loads and executes the CP/M 8088 assembler. The ufn specifies a source file containing assembly language statements where the secondary name is assumed to be ASM, and thus is not specified. The following ASM commands are valid:

```
ASM X

ASM GAMMA
```

The two-pass assembler is automatically executed. If assembly errors occur during the second pass, the errors are printed at the console.

The assembler produces a file

```
x.PRN
```

where x is the primary name specified in the ASM command. The PRN file contains a listing of the source program (with imbedded tab characters if present in the source program), along with the machine code generated for each statement and diagnostic error messages, if any. The PRN file can be listed
at the console using the TYPE command, or sent to a peripheral device using PIP (see the PIP command structure below). Note also that the PRN file contains the original source program, augmented by miscellaneous assembly information in the leftmost 16 columns (program addresses and hexadecimal machine code, for example). Thus, the PRN file can serve as a backup for the original source file: if the source file is accidently removed or destroyed, the PRN file can be edited (see the ED operator's guide) by removing the leftmost 16 characters of each line (this can be done by issuing a single editor "macro" command). The resulting file is identical to the original source file and can be renamed (REN) from PRN to ASM for subsequent editing and assembly. The file

\[
x.HEX
\]

is also produced which contains 8080 machine language in Intel "hex" format suitable for subsequent loading and execution (see the LOAD command). For complete details of CP/M's assembly language program, see the "CP/M Assembler Language (ASM) User's Guide."

Similar to other transient commands, the source file for assembly can be taken from an alternate disk by prefixing the assembly language file name by a disk drive name. Thus, the command

\[
ASM B:ALPHA cr
\]

loads the assembler from the currently logged drive and operates upon the source program ALPHA.ASM on drive B. The HEX and PRN files are also placed on drive B in this case.

6.3. LOAD ufn cr

The LOAD command reads the file ufn, which is assumed to contain "hex" format machine code, and produces a memory image file which can be subsequently executed. The file name ufn is assumed to be of the form

\[
x.HEX
\]

and thus only the name x need be specified in the command. The LOAD command creates a file named

\[
x.COM
\]

which marks it as containing machine executable code. The file is actually loaded into memory and executed when the user types the file name x immediately after the prompting character ">" printed by the CCP.

In general, the CCP reads the name x following the prompting character and looks for a built-in function name. If no function name is found, the CCP searches the system disk directory for a file by the name
x.COM

If found, the machine code is loaded into the TPA, and the program executes. Thus, the user need only LOAD a hex file once; it can be subsequently executed any number of times by simply typing the primary name. In this way, the user can "invent" new commands in the CCP. (Initialized disks contain the transient commands as COM files, which can be deleted at the user's option.) The operation can take place on an alternate drive if the file name is prefixed by a drive name. Thus,

LOAD B:BETA

brings the LOAD program into the TPA from the currently logged disk and operates upon drive B after execution begins.

It must be noted that the BETA.HEX file must contain valid Intel format hexadecimal machine code records (as produced by the ASM program, for example) which begin at 100H, the beginning of the TPA. Further, the addresses in the hex records must be in ascending order; gaps in unfilled memory regions are filled with zeroes by the LOAD command as the hex records are read. Thus, LOAD must be used only for creating CP/M standard "COM" files which operate in the TPA. Programs which occupy regions of memory other than the TPA can be loaded under DDT.

6.4. PIP cr

PIP is the CP/M Peripheral Interchange Program which implements the basic media conversion operations necessary to load, print, punch, copy, and combine disk files. The PIP program is initiated by typing one of the following forms

(1) PIP cr
(2) PIP "command line" cr

In both cases, PIP is loaded into the TPA and executed. In case (1), PIP reads command lines directly from the console, prompted with the "*" character, until an empty command line is typed (i.e., a single carriage return is issued by the operator). Each successive command line causes some media conversion to take place according to the rules shown below. Form (2) of the PIP command is equivalent to the first, except that the single command line given with the PIP command is automatically executed, and PIP terminates immediately with no further prompting of the console for input command lines. The form of each command line is

destination = source#1, source#2, ..., source#n cr

where "destination" is the file or peripheral device to receive the data, and
"source$1, ..., source$n" represents a series of one or more files or devices which are copied from left to right to the destination.

When multiple files are given in the command line (i.e., n > 1), the individual files are assumed to contain ASCII characters, with an assumed CP/M end-of-file character (ctl-Z) at the end of each file (see the 0 parameter to override this assumption). The equal symbol (=) can be replaced by a left-oriented arrow, if your console supports this ASCII character, to improve readability. Lower case ASCII alphabetics are internally translated to upper case to be consistent with CP/M file and device name conventions. Finally, the total command line length cannot exceed 255 characters (ctl-E can be used to force a physical carriage return for lines which exceed the console width).

The destination and source elements can be unambiguous references to CP/M source files, with or without a preceding disk drive name. That is, any file can be referenced with a preceding drive name (A:, B:, C:, or D:) which defines the particular drive where the file may be obtained or stored. When the drive name is not included, the currently logged disk is assumed. Further, the destination file can also appear as one or more of the source files, in which case the source file is not altered until the entire concatenation is complete. If the destination file already exists, it is removed if the command line is properly formed (it is not removed if an error condition arises). The following command lines (with explanations to the right) are valid as input to PIP:

```
X = Y cr
Copy to file X from file Y, where X and Y are unambiguous file names; Y remains unchanged.
```

```
X = Y,Z cr
Concatenate files Y and Z and copy to file X, with Y and Z unchanged.
```

```
X,ASM=Y,ASM,Z,ASM,FIN,ASM cr
Create the file X,ASM from the concatenation of the Y, Z, and FIN files with type ASM.
```

```
NEW,ZOT = B:OLD,ZAP cr
Move a copy of OLD,ZAP from drive B to the currently logged disk; name the file NEW,ZOT.
```

```
B:A,U = B:B,V,A:C,W,D,X cr
Concatenate file B,V from drive B with C,W from drive A and D,X from the logged disk; create the file A,U on drive B.
```

For more convenient use, PIP allows abbreviated commands for transferring files between disk drives. The abbreviated forms are
The first form copies all files from the currently logged disk which satisfy the afn to the same file names on drive x (x = A...Z). The second form is equivalent to the first, where the source for the copy is drive y (y = A...Z). The third form is equivalent to the command "PIP ufn=y:ufn cr" which copies the file given by ufn from drive y to the file ufn on drive x. The fourth form is equivalent to the third, where the source disk is explicitly given by y.

Note that the source and destination disks must be different in all of these cases. If an afn is specified, PIP lists each ufn which satisfies the afn as it is being copied. If a file exists by the same name as the destination file, it is removed upon successful completion of the copy, and replaced by the copied file.

The following PIP commands give examples of valid disk-to-disk copy operations:

\[ B:=* .COM \text{ cr} \]
Copy all files which have the secondary name "COM" to drive B from the current drive.

\[ A:=B:ZAP.* \text{ cr} \]
Copy all files which have the primary name "ZAP" to drive A from drive B.

\[ ZAP.ASM=B: \text{ cr} \]
Equivalent to ZAP.ASM=B:ZAP.ASM

\[ B:ZOT.COM=A: \text{ cr} \]
Equivalent to B:ZOT.COM=A:ZOT.COM

\[ B:=\text{GAMMA.BAS} \text{ cr} \]
Same as B:GAMMA.BAS=\text{GAMMA.BAS}

\[ B:=A:GAMMA.BAS \text{ cr} \]
Same as B:GAMMA.BAS=A:GAMMA.BAS

PIP also allows reference to physical and logical devices which are attached to the CP/M system. The device names are the same as given under the STAT command, along with a number of specially named devices. The logical devices given in the STAT command are

\[ \text{CON: (console), RDR: (reader), PUN: (punch), and LST: (list)} \]
while the physical devices are
TTY: (console, reader, punch, or list)
CRT: (console, or list), UCI: (console)
PTR: (reader), URL: (reader), UR2: (reader)
PIF: (punch), UPI: (punch), UP2: (punch)
LFT: (list), ULI: (list)

(Note that the "BAT:" physical device is not included, since this assignment is used only to indicate that the RDR: and LST: devices are to be used for console input/output.)

The RDR, LST, RUN, and CON devices are all defined within the BIOS portion of CP/M, and thus are easily altered for any particular I/O system. (The current physical device mapping is defined by I0BYTE; see the "CP/M Interface Guide" for a discussion of this function). The destination device must be capable of receiving data (i.e., data cannot be sent to the punch), and the source devices must be capable of generating data (i.e., the LST: device cannot be read).

The additional device names which can be used in PIP commands are

**NUL:** Send 40 "nulls" (ASCII 0's) to the device (this can be issued at the end of punched output).

**BOF:** Send a CP/M end-of-file (ASCII ctl-Z) to the destination device (sent automatically at the end of all ASCII data transfers through PIP).

**INP:** Special PIP input source which can be "patched" into the PIP program itself: PIP gets the input data character-by-character by CALLing location 103H, with data returned in location 109H (parity bit must be zero).

**OUT:** Special PIP output destination which can be patched into the PIP program: PIP CALLs location 106H with data in register C for each character to transmit. Note that locations 109H through 1FFH of the PIP memory image are not used and can be replaced by special purpose drivers using DDT (see the DDT operator's manual).

**PRN:** Same as LST:, except that tabs are expanded at every eighth character position, lines are numbered, and page ejects are inserted every 60 lines, with an initial eject (same as \[t8np\]).

File and device names can be interspersed in the PIP commands. In each case, the specific device is read until end-of-file (ctl-Z for ASCII files, and a real end of file for non-ASCII disk files). Data from each device or file is concatenated from left to right until the last data source has been
read. The destination device or file is written using the data from the source files, and an end-of-file character (ctl-Z) is appended to the result for ASCII files. Note if the destination is a disk file, then a temporary file is created ($$$ secondary name) which is changed to the actual file name only upon successful completion of the copy. Files with the extension "COM" are always assumed to be non-ASCII.

The copy operation can be aborted at any time by depressing any key on the keyboard (a rubout suffices). PIP will respond with the message "ABORTED" to indicate that the operation was not completed. Note that if any operation is aborted, or if an error occurs during processing, PIP removes any pending commands which were set up while using the SUBMIT command.

It should also be noted that PIP performs a special function if the destination is a disk file with type "HEX" (an Intel hex formatted machine code file), and the source is an external peripheral device, such as a paper tape reader. In this case, the PIP program checks to ensure that the source file contains a properly formed hex file, with legal hexadecimal values and checksum records. When an invalid input record is found, PIP reports an error message at the console and waits for corrective action. It is usually sufficient to open the reader and rerun a section of the tape (pull the tape back about 20 inches). When the tape is ready for the re-read, type a single carriage return at the console, and PIP will attempt another read. If the tape position cannot be properly read, simply continue the read (by typing a return following the error message), and enter the record manually with the ED program after the disk file is constructed. For convenience, PIP allows the end-of-file to be entered from the console if the source file is a PDR device. In this case, the PIP program reads the device and monitors the keyboard. If ctl-Z is typed at the keyboard, then the read operation is terminated normally.

Valid PIP commands are shown below.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIP LST: = X.PRN cr</td>
<td>Copy X.PRN to the LST device and terminate the PIP program.</td>
</tr>
<tr>
<td>PIP cr</td>
<td>Start PIP for a sequence of commands (PIP prompts with **).</td>
</tr>
<tr>
<td>*CON:=X,ASM,Y,ASM,Z,ASM cr</td>
<td>Concatenate three ASM files and copy to the CON device.</td>
</tr>
<tr>
<td>*X,HEX=CON:,Y,HEX,PTR: cr</td>
<td>Create a HEX file by reading the CON (until a ctl-Z is typed), followed by data from Y,HEX, followed by data from PTR until a ctl-Z is encountered.</td>
</tr>
<tr>
<td>*cr</td>
<td>Single carriage return stops PIP.</td>
</tr>
</tbody>
</table>

56
Send 40 nulls to the punch device; then copy the XASM file to the punch, followed by an end-of-file (ctl-Z) and 40 more null characters.

The user can also specify one or more PIP parameters, enclosed in left and right square brackets, separated by zero or more blanks. Each parameter affects the copy operation, and the enclosed list of parameters must immediately follow the affected file or device. Generally, each parameter can be followed by an optional decimal integer value (the S and Q parameters are exceptions). The valid PIP parameters are listed below.

- **B** Block mode transfer: data is buffered by PIP until an ASCII x-off character (ctl-S) is received from the source device. This allows transfer of data to a disk file from a continuous reading device, such as a cassette reader. Upon receipt of the x-off, PIP clears the disk buffers and returns for more input data. The amount of data which can be buffered is dependent upon the memory size of the host system (PIP will issue an error message if the buffers overflow).

- **Dn** Delete characters which extend past column n in the transfer of data to the destination from the character source. This parameter is used most often to truncate long lines which are sent to a (narrow) printer or console device.

- **E** Echo all transfer operations to the console as they are being performed.

- **F** Filter form feeds from the file. All imbedded form feeds are removed. The P parameter can be used simultaneously to insert new form feeds.

- **H** Hex data transfer: all data is checked for proper Intel hex file format. Non-essential characters between hex records are removed during the copy operation. The console will be prompted for corrective action in case errors occur.

- **I** Ignore ":00" records in the transfer of Intel hex format file (the I parameter automatically sets the H parameter).

- **L** Translate upper case alphabetics to lower case.

- **N** Add line numbers to each line transferred to the destination starting at one, and incrementing by 1. Leading zeroes are suppressed, and the number is followed by a colon. If N2 is specified, then leading zeroes are included, and a tab is inserted following the number. The tab is expanded if T is
Object file (non-ASCII) transfer: the normal CP/M end of file is ignored.

Pn Include page ejects at every n lines (with an initial page eject). If n = 1 or is excluded altogether, page ejects occur every 60 lines. If the F parameter is used, form feed suppression takes place before the new page ejects are inserted.

Qsz Quit copying from the source device or file when the string s (terminated by ctl-Z) is encountered.

Ssz Start copying from the source device when the string s is encountered (terminated by ctl-Z). The S and Q parameters can be used to "abstract" a particular section of a file (such as a subroutine). The start and quit strings are always included in the copy operation.

NOTE - the strings following the s and q parameters are translated to upper case by the CCP if form (2) of the PIP command is used. Form (1) of the PIP invocation, however, does not perform the automatic upper case translation.

Tn Expand tabs (ctl-I characters) to every nth column during the transfer of characters to the destination from the source.

U Translate lower case alphabets to upper case during the copy operation.

V Verify that data has been copied correctly by rereading after the write operation (the destination must be a disk file).

Z Zero the parity bit on input for each ASCII character.

The following are valid PIP commands which specify parameters in the file transfer:

PIPX.ASM=B:[v] cr Copy X.ASM from drive B to the current drive and verify that the data was properly copied.

PIPLPT:=X.ASM[nt8u] cr Copy X.ASM to the LPT: device; number each line, expand tabs to every eighth column, and translate lower case alphabets to upper case.
First copy X.HEX to the PUN: device and ignore the trailing ":00" record in X.HEX; then continue the transfer of data by reading Y.ZOT, which contains hex records, including any ":00" records which it contains.

Copy from the file Y.ASM into the file X.LIB. Start the copy when the string "SUBRL:" has been found, and quit copying after the string "JMP L3" is encountered.

Send X.ASM to the LST: device, with line numbers, tabs expanded to every eighth column, and page ejects at every 50th line. Note that nt8p50 is the assumed parameter list for a PRN file; p50 overrides the default value.

6.5. ED ufn cr

The ED program is the CP/M system context editor, which allows creation and alteration of ASCII files in the CP/M environment. Complete details of operation are given in the ED user's manual, "ED: a Context Editor for the CP/M Disk System." In general, ED allows the operator to create and operate upon source files which are organized as a sequence of ASCII characters, separated by end-of-line characters (a carriage-return line-feed sequence). There is no practical restriction on line length (no single line can exceed the size of the working memory), which is instead defined by the number of characters typed between cr's. The ED program has a number of commands for character string searching, replacement, and insertion, which are useful in the creation and correction of programs or text files under CP/M. Although the CP/M has a limited memory work space area (approximately 5000 characters in a 16K CP/M system), the file size which can be edited is not limited, since data is easily "paged" through this work area.

Upon initiation, ED creates the specified source file, if it does not exist, and opens the file for access. The programmer then "appends" data from the source file into the work area, if the source file already exists (see the A command), for editing. The appended data can then be displayed, altered, and written from the work area back to the disk (see the W command). Particular points in the program can be automatically paged and located by context (see the N command), allowing easy access to particular portions of a large file.

Given that the operator has typed

ED X.ASM cr
the ED program creates an intermediate work file with the name

X.$$$. 

to hold the edited data during the ED run. Upon completion of ED, the X.ASM file (original file) is renamed to X.BAK, and the edited work file is renamed to X.ASM. Thus, the X.BAK file contains the original (unedited) file, and the X.ASM file contains the newly edited file. The operator can always return to the previous version of a file by removing the most recent version, and renaming the previous version. Suppose, for example, that the current X.ASM file was improperly edited; the sequence of CCP command shown below would reclaim the backup file.

```
DIR X.* Check to see that BAK file is available.
ERA X.ASM Erase most recent version.
REN X.ASM=X.BAK Rename the BAK file to ASM.
```

Note that the operator can abort the edit at any point (reboot, power failure, ctrl-C, or Q command) without destroying the original file. In this case, the BAK file is not created, and the original file is always intact.

The ED program also allows the user to "ping-pong" the source and create backup files between two disks. The form of the ED command in this case is

```
ED ufn d:
```

where ufn is the name of a file to edit on the currently logged disk, and d is the name of an alternate drive. The ED program reads and processes the source file, and writes the new file to drive d, using the name ufn. Upon completion of processing, the original file becomes the backup file. Thus, if the operator is addressing disk A, the following command is valid:

```
ED X.ASM B:
```

which edits the file X.ASM on drive A, creating the new file X.$$$. on drive B. Upon completion of a successful edit, A:X.ASM is renamed to A:X.BAK, and B:X.$$$. is renamed to B:X.ASM. For user convenience, the currently logged disk becomes drive B at the end of the edit. Note that if a file by the name B:X.ASM exists before the editing begins, the message

```
FILE EXISTS
```

is printed at the console as a precaution against accidently destroying a source file. In this case, the operator must first ERAse the existing file and then restart the edit operation.
Similar to other transient commands, editing can take place on a drive different from the currently logged disk by preceding the source file name by a drive name. Examples of valid edit requests are shown below:

**ED A:X.ASM**
Edit the file X.ASM on drive A, with new file and backup on drive A.

**ED B:X.ASM A:**
Edit the file X.ASM on drive B to the temporary file X.$$ on drive A. On termination of editing, change X.ASM on drive B to X.BAK, and change X.$$ on drive A to X.ASM.

### 6.6. SYSGEN cr

The SYSGEN transient command allows generation of an initialized diskette containing the CP/M operating system. The SYSGEN program prompts the console for commands, with interaction as shown below.

**SYSGEN cr**
Initiate the SYSGEN program.

**SYSGEN VERSION m.m**
SYSGEN sign-on message.

**SOURCE DRIVE NAME (OR RETURN TO SKIP)**
Respond with the drive name (one of the letters A, B, C, or D) of the disk containing a CP/M system; usually A. If a copy of CP/M already exists in memory, due to a MDVCPM command, type a cr only. Typing a drive name x will cause the response:

**SOURCE ON x THEN TYPE RETURN**
Place a diskette containing the CP/M operating system on drive x (x is one of A, B, C, or D). Answer with cr when ready.

**FUNCTION COMPLETE**
System is copied to memory. SYSGEN will then prompt with:

**DESTINATION DRIVE NAME (OR RETURN TO REBOOT)**
If a diskette is being initialized, place the new disk into a drive and answer with the drive name. Otherwise, type a cr and the system will reboot from drive A. Typing drive name x will cause SYSGEN to prompt
with:

DESTINATION ON x THEN TYPE RETURN Place new diskette into drive x; type return when ready.

FUNCTION COMPLETE New diskette is initialized in drive x.

The "DESTINATION" prompt will be repeated until a single carriage return is typed at the console, so that more than one disk can be initialized.

Upon completion of a successful system generation, the new diskette contains the operating system, and only the built-in commands are available. A factory-fresh IBM-compatible diskette appears to CP/M as a diskette with an empty directory; therefore, the operator must copy the appropriate COM files from an existing CP/M diskette to the newly constructed diskette using the PIP transient.

The user can copy all files from an existing diskette by typing the PIP command

```
PIp B: = A: *.*[v] cr
```

which copies all files from disk drive A to disk drive B, and verifies that each file has been copied correctly. The name of each file is displayed at the console as the copy operation proceeds.

It should be noted that a SYSGEN does not destroy the files which already exist on a diskette; it results only in construction of a new operating system. Further, if a diskette is being used only on drives B through D, and will never be the source of a bootstrap operation on drive A, the SYSGEN need not take place. In fact, a new diskette needs absolutely no initialization to be used with CP/M.

6.7. SUBMIT ufn parm#1 ... parm#n cr

The SUBMIT command allows CP/M commands to be batched together for automatic processing. The ufn given in the SUBMIT command must be the filename of a file which exists on the currently logged disk, with an assumed file type of "SUB." The SUB file contains CP/M prototype commands, with possible parameter substitution. The actual parameters parm#1 ... parm#n are substituted into the prototype commands, and, if no errors occur, the file of substituted commands are processed sequentially by CP/M.
The prototype command file is created using the ED program, with interspersed "$" parameters of the form

$1 $2 $3 ... $n

corresponding to the number of actual parameters which will be included when the file is submitted for execution. When the SUBMIT transient is executed, the actual parameters parm#1 ... parm#n are paired with the formal parameters $1 ... $n in the prototype commands. If the number of formal and actual parameters does not correspond, then the submit function is aborted with an error message at the console. The SUBMIT function creates a file of substituted commands with the name

$$$.SUB

on the logged disk. When the system reboots (at the termination of the SUBMIT), this command file is read by the CCP as a source of input, rather than the console. If the SUBMIT function is performed on any disk other than drive A, the commands are not processed until the disk is inserted into drive A and the system reboots. Further, the user can abort command processing at any time by typing a rubout when the command is read and echoed. In this case, the $$$.SUB file is removed, and the subsequent commands come from the console. Command processing is also aborted if the CCP detects an error in any of the commands. Programs which execute under CP/M can abort processing of command files when error conditions occur by simply erasing any existing $$$.SUB file.

In order to introduce dollar signs into a SUBMIT file, the user may type a "$" which reduces to a single "$" within the command file. Further, an up-arrow symbol "↑" may precede an alphabetic character x, which produces a single ctrl-x character within the file.

The last command in a SUB file can initiate another SUB file, thus allowing chained batch commands.

Suppose the file ASMBL.SUB exists on disk and contains the prototype commands

```
ASM $1
DIR $1.*
ERA *.BAK
PIP $2:=*1.PRN
ERA $1.PRN
```

and the command

```
SUBMIT ASMBL X PRN CR
```

is issued by the operator. The SUBMIT program reads the ASMBL.SUB file, substituting "X" for all occurrences of $1 and "PRN" for all occurrences of $2, resulting in a $$$.SUB file containing the commands
which are executed in sequence by the CCP.

The SUBMIT function can access a SUB file which is on an alternate drive by preceding the file name by a drive name. Submitted files are only acted upon, however, when they appear on drive A. Thus, it is possible to create a submitted file on drive B which is executed at a later time when it is inserted in drive A.

6.8. DUMP ufn cr

The DUMP program types the contents of the disk file (ufn) at the console in hexadecimal form. The file contents are listed sixteen bytes at a time, with the absolute byte address listed to the left of each line in hexadecimal. Long typeouts can be aborted by pushing the rubout key during printout. (The source listing of the DUMP program is given in the "CP/M Interface Guide" as an example of a program written for the CP/M environment.)

6.9. MOVCPM cr

The MOVCPM program allows the user to reconfigure the CP/M system for any particular memory size. Two optional parameters may be used to indicate (1) the desired size of the new system and (2) the disposition of the new system at program termination. If the first parameter is omitted or a "*" is given, the MOVCPM program will reconfigure the system to its maximum size, based upon the kilobytes of contiguous RAM in the host system (starting at 0000H). If the second parameter is omitted, the system is executed, but not permanently recorded; if "*" is given, the system is left in memory, ready for a SYSGEN operation. The MOVCPM program relocates a memory image of CP/M and places this image in memory in preparation for a system generation operation. The command forms are:

    MOVCPM cr
    relocate and execute CP/M for management of the current memory configuration (memory is examined for contiguous RAM, starting at 100H). Upon completion of the relocation, the new system is executed but not permanently recorded on the diskette. The system which is constructed contains a BIOS for the Intel MDS 800.
MOVCPM n cr
Create a relocated CP/M system for
management of an n kilobyte system (n
must be in the range 16 to 64), and
execute the system, as described above.

MOVCPM ** cr
Construct a relocated memory image for
the current memory configuration, but
leave the memory image in memory, in
preparation for a SYSGEN operation.

MOVCPM n * cr
Construct a relocated memory image for
an n kilobyte memory system, and leave
the memory image in preparation for a
SYSGEN operation.

The command

MOVCPM **
for example, constructs a new version of the CP/M system and leaves it in
memory, ready for a SYSGEN operation. The message

READY FOR "SYSGEN" OR
"SAVE 32 CPMxx.COM"

is printed at the console upon completion, where xx is the current memory size
in kilobytes. The operator can then type

SYSGEN cr
Start the system generation.

SOURCE DRIVE NAME (OR RETURN TO SKIP) Respond with a cr to skip
the CP/M read operation since the system
is already in memory as a result of the
previous MOVCPM operation.

DESTINATION DRIVE NAME (OR RETURN TO REBOOT)
Respond with B to write new system
to the diskette in drive B. SYSGEN
will prompt with:

DESTINATION ON B, THEN TYPE RETURN
Ready the fresh diskette on drive
B and type a return when ready.

Note that if you respond with "A" rather than "B" above, the system will be
written to drive A rather than B. SYSGEN will continue to type the prompt:

DESTINATION DRIVE NAME (OR RETURN TO REBOOT)

until the operator responds with a single carriage return, which stops the
SYSGEN program with a system reboot.

The user can then go through the reboot process with the old or new diskette. Instead of performing the SYSGEN operation, the user could have typed

SAVE 32 CP/Mxx.COM

at the completion of the MOVCPM function, which would place the CP/M memory image on the currently logged disk in a form which can be "patched." This is necessary when operating in a non-standard environment where the BIOS must be altered for a particular peripheral device configuration, as described in the "CP/M System Alteration Guide."

Valid MOVCPM commands are given below:

MOVCPM 48 cr Construct a 48K version of CP/M and start execution.

MOVCPM 48 * cr Construct a 48K version of CP/M in preparation for permanent recording; response is READY FOR "SYSGEN" OR "SAVE 32CPM48.COM"

MOVCPM ** cr Construct a maximum memory version of CP/M and start execution.

It is important to note that the newly created system is serialized with the number attached to the original diskette and is subject to the conditions of the Digital Research Software Licensing Agreement.
7. BDOS ERROR MESSAGES.

There are three error situations which the Basic Disk Operating System intercepts during file processing. When one of these conditions is detected, the BDOS prints the message:

   BDOS ERR ON x: error

where x is the drive name, and "error" is one of the three error messages:

   BAD SECTOR
   SELECT
   READ ONLY

The "BAD SECTOR" message indicates that the disk controller electronics has detected an error condition in reading or writing the diskette. This condition is generally due to a malfunctioning disk controller, or an extremely worn diskette. If you find that your system reports this error more than once a month, you should check the state of your controller electronics, and the condition of your media. You may also encounter this condition in reading files generated by a controller produced by a different manufacturer. Even though controllers are claimed to be IBM-compatible, one often finds small differences in recording formats. The MDS-800 controller, for example, requires two bytes of one's following the data CRC byte, which is not required in the IBM format. As a result, diskettes generated by the Intel MDS can be read by almost all other IBM-compatible systems, while disk files generated on other manufacturer's equipment will produce the "BAD SECTOR" message when read by the MDS. In any case, recovery from this condition is accomplished by typing a ctl-C to reboot (this is the safest!), or a return, which simply ignores the bad sector in the file operation. Note, however, that typing a return may destroy your diskette integrity if the operation is a directory write, so make sure you have adequate backups in this case.

The "SELECT" error occurs when there is an attempt to address a drive beyond the A through D range. In this case, the value of x in the error message gives the selected drive. The system reboots following any input from the console.

The "READ ONLY" message occurs when there is an attempt to write to a diskette which has been designated as read-only in a STAT command, or has been set to read-only by the BDOS. In general, the operator should reboot CP/M either by using the warm start procedure (ctl-C) or by performing a cold start whenever the diskettes are changed. If a changed diskette is to be read but not written, BDOS allows the diskette to be changed without the warm or cold start, but internally marks the drive as read-only. The status of the drive is subsequently changed to read/write if a warm or cold start occurs. Upon issuing this message, CP/M waits for input from the console. An automatic warm start takes place following any input.
8. **OPERATION OF CP/M ON THE MDS.**

This section gives operating procedures for using CP/M on the Intel MDS microcomputer development system. A basic knowledge of the MDS hardware and software systems is assumed.

CP/M is initiated in essentially the same manner as Intel's ISIS operating system. The disk drives are labelled 0 through 3 on the MDS, corresponding to CP/M drives A through D, respectively. The CP/M system diskette is inserted into drive 0, and the BOOT and RESET switches are depressed in sequence. The interrupt 2 light should go on at this point. The space bar is then depressed on the device which is to be taken as the system console, and the light should go out (if it does not, then check connections and baud rates). The BOOT switch is then turned off, and the CP/M signon message should appear at the selected console device, followed by the "A>" system prompt. The user can then issue the various resident and transient commands.

The CP/M system can be restarted (warm start) at any time by pushing the INT 0 switch on the front panel. The built-in Intel ROM monitor can be initiated by pushing the INT 7 switch (which generates a RST 7), except when operating under DDT, in which case the DDT program gets control instead.

Diskettes can be removed from the drives at any time, and the system can be shut down during operation without affecting data integrity. Note, however, that the user must not remove a diskette and replace it with another without rebooting the system (cold or warm start), unless the inserted diskette is "read only."

Due to hardware hang-ups or malfunctions, CP/M may type the message

**BDOS ERR ON X: BAD SECTOR**

where x is the drive which has a permanent error. This error may occur when drive doors are opened and closed randomly, followed by disk operations, or may be due to a diskette, drive, or controller failure. The user can optionally elect to ignore the error by typing a single return at the console. The error may produce a bad data record, requiring re-initialization of up to 128 bytes of data. The operator can reboot the CP/M system and try the operation again.

Termination of a CP/M session requires no special action, except that it is necessary to remove the diskettes before turning the power off, to avoid random transients which often make their way to the drive electronics.

It should be noted that factory-fresh IBM-compatible diskettes should be used rather than diskettes which have previously been used with any ISIS version. In particular, the ISIS "FORMAT" operation produces non-standard sector numbering throughout the diskette. This non-standard numbering seriously degrades the performance of CP/M, and will operate noticeably slower.
than the distribution version. If it becomes necessary to reformat a diskette (which should not be the case for standard diskettes), a program can be written under CP/M which causes the MDS 800 controller to reformat with sequential sector numbering (1-26) on each track.

Note: "MDS 800" and "ISIS" are registered trademarks of Intel Corporation.
CP/M ASSEMBLER (ASM): USER'S GUIDE
COPYRIGHT

Copyright (c) 1980 by Exidy Systems, Inc. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of Exidy Systems, Inc., 1234 Elko Drive, Sunnyvale, California 94086.

Since this manual is tutorial in nature, permission is granted to reproduce or abstract the example procedures and sample programs for the purposes of inclusion within the reader's programs.

DISCLAIMER

Exidy Systems, Inc. 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, Exidy Systems, Inc. reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Exidy Systems, Inc. to notify any person of such revision or changes.

TRADEMARKS

CP/M is a registered trademark of Digital Research.
CP/M ASSEMBLER (ASM): USER'S GUIDE
COPYRIGHT

Copyright (c) 1980 by Exidy Systems, Inc. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of Exidy Systems, Inc., 1234 Elko Drive, Sunnyvale, California 94086.

Since this manual is tutorial in nature, permission is granted to reproduce or abstract the example procedures and sample programs for the purposes of inclusion within the reader's programs.

DISCLAIMER

Exidy Systems, Inc. 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, Exidy Systems, Inc. reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Exidy Systems, Inc. to notify any person of such revision or changes.

TRADEMARKS

CP/M is a registered trademark of Digital Research.
# Table Of Contents

**CP/M Assembler (ASM): User’s Guide**

1. **Introduction** .................................................. 71
2. **Program Format** ............................................... 72
3. **Forming The Operand** ............................................. 74
   3.1 **Labels** .................................................. 74
   3.2 **Numeric Constants** ......................................... 74
   3.3 **Reserved Words** ............................................ 75
   3.4 **String Constants** ......................................... 76
   3.5 **Arithmetic and Logical Operators** ......................... 76
   3.6 **Precedence Of Operators** ................................ 77
4. **Assembler Directives** ............................................ 78
   4.1 **The ORG Directive** ....................................... 78
   4.2 **The END Directive** ...................................... 79
   4.3 **The EQU Directive** ...................................... 79
   4.4 **The SET Directive** ...................................... 80
   4.5 **The IF And ENDIF Directives** ............................ 80
   4.6 **The DB Directive** ...................................... 81
   4.7 **The DW Directive** ...................................... 82
5. **Operation Codes** .................................................. 82
   5.1 **Jumps, Calls And Returns** .............................. 83
   5.2 **Immediate Operand Instructions** ....................... 84
   5.3 **Increment and Decrement Instructions** .................. 84
   5.4 **Data Movement Instructions** ............................ 84
   5.5 **Arithmetic Logic Unit Operations** ..................... 85
   5.6 **Control Instructions** .................................. 86
6. **Error Messages** .................................................. 86
7. **A Sample Session** ............................................... 87
CP/M Assembler User's Guide

1. INTRODUCTION.

The CP/M assembler reads assembly language source files from the diskette, and produces 8080 machine language in Intel hex format. The CP/M assembler is initiated by typing

```
ASM filename
```
or

```
ASM filename parms
```

In both cases, the assembler assumes there is a file on the diskette with the name

```
filename.ASM
```

which contains an 8080 assembly language source file. The first and second forms shown above differ only in that the second form allows parameters to be passed to the assembler to control source file access and hex and print file destinations.

In either case, the CP/M assembler loads, and prints the message

```
CP/M ASSEMBLER VER n.n
```

where n.n is the current version number. In the case of the first command, the assembler reads the source file with assumed file type "ASM" and creates two output files

```
filename.HEX
```
and

```
filename.PRN
```

the "HEX" file contains the machine code corresponding to the original program in Intel hex format, and the "PRN" file contains an annotated listing showing generated machine code, error flags, and source lines. If errors occur during translation, they will be listed in the PRN file as well as at the console.

The second command form can be used to redirect input and output files from their defaults. In this case, the "parms" portion of the command is a three letter group which specifies the origin of the source file, the destination of the hex file, and the destination of the print file. The form is

```
filename.plp2p3
```
where pl, p2, and p3 are single letters

```
pl: A,B, ..., Y  designates the disk name which contains
```

71
Thus, the command

```
ASM X.AAA
```

indicates that the source file (X.ASM) is to be taken from disk A, and that the hex (X.HEX) and print (X.PRN) files are to be created also on disk A. This form of the command is implied if the assembler is run from disk A. That is, given that the operator is currently addressing disk A, the above command is equivalent to

```
ASM X
```

The command

```
ASM X.ABX
```

indicates that the source file is to be taken from disk A, the hex file is placed on disk B, and the listing file is to be sent to the console. The command

```
ASM X.BZZ
```

takes the source file from disk B, and skips the generation of the hex and print files (this command is useful for fast execution of the assembler to check program syntax).

The source program format is compatible with both the Intel 8080 assembler (macros are not currently implemented in the CP/M assembler, however), as well as the Processor Technology Software Package #1 assembler. That is, the CP/M assembler accepts source programs written in either format. There are certain extensions in the CP/M assembler which make it somewhat easier to use. These extensions are described below.

2. PROGRAM FORMAT.

An assembly language program acceptable as input to the assembler consists of a sequence of statements of the form

```
line# label operation operand ;comment
```

where any or all of the fields may be present in a particular instance. Each
Assemble language statement is terminated with a carriage return and line feed (the line feed is inserted automatically by the ED program), or with the character "!" which is a treated as an end-of-line by the assembler (thus, multiple assemble language statements can be written on the same physical line if separated by exclaim symbols).

The line# is an optional decimal integer value representing the source program line number, which is allowed on any source line to maintain compatibility with the Processor Technology format. In general, these line numbers will be inserted if a line-oriented editor is used to construct the original program, and thus ASM ignores this field if present.

The label field takes the form

```
identifier
```

or

```
identifier:
```

and is optional, except where noted in particular statement types. The identifier is a sequence of alphanumeric characters (alphabetics and numbers), where the first character is alphabetic. Identifiers can be freely used by the programmer to label elements such as program steps and assembler directives, but cannot exceed 16 characters in length. All characters are significant in an identifier, except for the embedded dollar symbol ($) which can be used to improve readability of the name. Further, all lower case alphabetics become are treated as if they were upper case. Note that the ":" following the identifier in a label is optional (to maintain compatibility between Intel and Processor Technology). Thus, the following are all valid instances of labels

```
x      xy      long$name
x:     yx1:    longer$named$data:
X1Y2   X1x2    x234$5678$9012$3456:
```

The operation field contains either an assembler directive, or pseudo operation, or an 8080 machine operation code. The pseudo operations and machine operation codes are described below.

The operand field of the statement, in general, contains an expression formed out of constants and labels, along with arithmetic and logical operations on these elements. Again, the complete details of properly formed expressions are given below.

The comment field contains arbitrary characters following the ";" symbol until the next real or logical end-of-line. These characters are read, listed, and otherwise ignored by the assembler. In order to maintain compatibility with the Processor Technology assembler, the CP/M assembler also treat statements which begin with a "*" in column one as comment statements, which are listed and ignored in the assembly process. Note that the Processor
Technology assembler has the side effect in its operation of ignoring the characters after the operand field has been scanned. This causes an ambiguous situation when attempting to be compatible with Intel's language, since arbitrary expressions are allowed in this case. Hence, programs which use this side effect to introduce comments, must be edited to place a ";" before these fields in order to assemble correctly.

The assembly language program is formulated as a sequence of statements of the above form, terminated optionally by an END statement. All statements following the END are ignored by the assembler.

3. FORMING THE OPERAND.

In order to completely describe the operation codes and pseudo operations, it is necessary to first present the form of the operand field, since it is used in nearly all statements. Expressions in the operand field consist of simple operands (labels, constants, and reserved words), combined in properly formed subexpressions by arithmetic and logical operators. The expression computation is carried out by the assembler as the assembly proceeds. Each expression must produce a 16-bit value during the assembly. Further, the number of significant digits in the result must not exceed the intended use. That is, if an expression is to be used in a byte move immediate instruction, then the most significant 8 bits of the expression must be zero. The restrictions on the expression significance is given with the individual instructions.

3.1. Labels.

As discussed above, a label is an identifier which occurs on a particular statement. In general, the label is given a value determined by the type of statement which it precedes. If the label occurs on a statement which generates machine code or reserves memory space (e.g., a MOV instruction, or a DS pseudo operation), then the label is given the value of the program address which it labels. If the label precedes an EQU or SET, then the label is given the value which results from evaluating the operand field. Except for the SET statement, an identifier can label only one statement.

When a label appears in the operand field, its value is substituted by the assembler. This value can then be combined with other operands and operators to form the operand field for a particular instruction.

3.2. Numeric Constants.

A numeric constant is a 16-bit value in one of several bases. The base, called the radix of the constant, is denoted by a trailing radix indicator. The radix indicators are

B    binary constant (base 2)
O    octal constant (base 8)
Q octal constant (base 8)
D decimal constant (base 10)
H hexadecimal constant (base 16)

Q is an alternate radix indicator for octal numbers since the letter O is easily confused with the digit 0. Any numeric constant which does not terminate with a radix indicator is assumed to be a decimal constant.

A constant is thus composed as a sequence of digits, followed by an optional radix indicator, where the digits are in the appropriate range for the radix. That is binary constants must be composed of 0 and 1 digits, octal constants can contain digits in the range 0 - 7, while decimal constants contain decimal digits. Hexadecimal constants contain decimal digits as well as hexadecimal digits A (10D), B (11D), C (12D), D (13D), E (14D), and F (15D). Note that the leading digit of a hexadecimal constant must be a decimal digit in order to avoid confusing a hexadecimal constant with an identifier (a leading 0 will always suffice). A constant composed in this manner must evaluate to a binary number which can be contained within a 16-bit counter, otherwise it is truncated on the right by the assembler. Similar to identifiers, imbedded "$" are allowed within constants to improve their readability. Finally, the radix indicator is translated to upper case if a lower case letter is encountered. The following are all valid instances of numeric constants

1234 1234D 1100B 1111$0000$1111$0000B
1234H 0FFEh 33770 33$77$22Q
33770 0fe3h 1234d 0ffffh

3.3. Reserved Words.

There are several reserved character sequences which have predefined meanings in the operand field of a statement. The names of 8080 registers are given below, which, when encountered, produce the value shown to the right

A 7
B 0
C 1
D 2
E 3
H 4
L 5
M 6
SP 6
PSW 6

(again, lower case names have the same values as their upper case equivalents). Machine instructions can also be used in the operand field, and evaluate to their internal codes. In the case of instructions which require operands, where the specific operand becomes a part of the binary bit pattern
of the instruction (e.g., MOV A,B), the value of the instruction (in this case MOV) is the bit pattern of the instruction with zeroes in the optional fields (e.g., MOV produces 40H).

When the symbol "$" occurs in the operand field (not imbedded within identifiers and numeric constants) its value becomes the address of the next instruction to generate, not including the instruction contained within the current logical line.

3.4. String Constants.

String constants represent sequences of ASCII characters, and are represented by enclosing the characters within apostrophe symbols ('). All strings must be fully contained within the current physical line (thus allowing "$" symbols within strings), and must not exceed 64 characters in length. The apostrophe character itself can be included within a string by representing it as a double apostrophe (the two keystrokes ''), which becomes a single apostrophe when read by the assembler. In most cases, the string length is restricted to either one or two characters (the DB pseudo operation is an exception), in which case the string becomes an 8 or 16 bit value, respectively. Two character strings become a 16-bit constant, with the second character as the low order byte, and the first character as the high order byte.

The value of a character is its corresponding ASCII code. There is no case translation within strings, and thus both upper and lower case characters can be represented. Note however, that only graphic (printing) ASCII characters are allowed within strings. Valid strings are

'A'  'AB'  'ab'  'C'
'...  'a'  '....  '

"Walla Walla Wash."
"She said "Hello" to me."
"I said "Hello" to her."

3.5. Arithmetic and Logical Operators.

The operands described above can be combined in normal algebraic notation using any combination of properly formed operands, operators, and parenthesized expressions. The operators recognized in the operand field are

\[
\begin{align*}
\text{a + b} & \quad \text{unsigned arithmetic sum of a and b} \\
\text{a - b} & \quad \text{unsigned arithmetic difference between a and b} \\
\text{+ b} & \quad \text{unary plus (produces b)} \\
\text{- b} & \quad \text{unary minus (identical to 0 - b)} \\
\text{a * b} & \quad \text{unsigned magnitude multiplication of a and b} \\
\text{a / b} & \quad \text{unsigned magnitude division of a by b} \\
\text{a MOD b} & \quad \text{remainder after a / b} \\
\text{NOT b} & \quad \text{logical inverse of b (all 0's become 1's, 1's become 0's), where b is considered a 16-bit value}
\end{align*}
\]
a AND b  bit-by-bit logical and of a and b
a OR b   bit-by-bit logical or of a and b
a XOR b  bit-by-bit logical exclusive or of a and b
a SHL b  the value which results from shifting a to the left by an amount b, with zero fill
a SHR b  the value which results from shifting a to the right by an amount b, with zero fill

In each case, a and b represent simple operands (labels, numeric constants, reserved words, and one or two character strings), or fully enclosed parenthesized subexpressions such as

10+20   10h+37Q   Ll /3   (L2+4) SHR 3
('a' and 5fh) + '0'  ('B'B) OR (PSW+M)
(1+(2+c)) SHR (A-(B+1))

Note that all computations are performed at assembly time as 16-bit unsigned operations. Thus, -1 is computed as 0-1 which results in the value 0ffffh (i.e., all 1's). The resulting expression must fit the operation code in which it is used. If, for example, the expression is used in a ADI (add immediate) instruction, then the high order eight bits of the expression must be zero. As a result, the operation "ADI -l" produces an error message (-l becomes 0ffffh which cannot be represented as an 8 bit value), while "ADI (-1) AND 0FFH" is accepted by the assembler since the "AND" operation zeroes the high order bits of the expression.


As a convenience to the programmer, ASM assumes that operators have a relative precedence of application which allows the programmer to write expressions without nested levels of parentheses. The resulting expression has assumed parentheses which are defined by the relative precedence. The order of application of operators in unparenthesize expressions is listed below. Operators listed first have highest precedence (they are applied first in an unparenthesized expression), while operators listed last have lowest precedence. Operators listed on the same line have equal precedence, and are applied from left to right as they are encountered in an expression

* /  MOD  SHL  SHR
- +
 NOT
 AND
 OR  XOR

Thus, the expressions shown to the left below are interpreted by the assembler as the fully parenthesize expressions shown to the right below

a * b + c              (a * b) + c
a + b * c             a + (b * c)
a MOD b * c SHR d     ((a MOD b) * c) SHR d
a OR b AND NOT c + d SHL e  

Balanced parenthesized subexpressions can always be used to override the assumed parentheses, and thus the last expression above could be rewritten to force application of operators in a different order as

(a OR b) AND (NOT c) + d SHL e

resulting in the assumed parentheses

(a OR b) AND ((NOT c) + (d SHL e))

Note that an unparenthesized expression is well-formed only if the expression which results from inserting the assumed parentheses is well-formed.

4. ASSEMBLER DIRECTIVES.

Assembler directives are used to set labels to specific values during the assembly, perform conditional assembly, define storage areas, and specify starting addresses in the program. Each assembler directive is denoted by a "pseudo operation" which appears in the operation field of the line. The acceptable pseudo operations are

<table>
<thead>
<tr>
<th>Pseudo Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORG</td>
<td>set the program or data origin</td>
</tr>
<tr>
<td>END</td>
<td>end program, optional start address</td>
</tr>
<tr>
<td>EQU</td>
<td>numeric &quot;equate&quot;</td>
</tr>
<tr>
<td>SET</td>
<td>numeric &quot;set&quot;</td>
</tr>
<tr>
<td>IF</td>
<td>begin conditional assembly</td>
</tr>
<tr>
<td>ENDIF</td>
<td>end of conditional assembly</td>
</tr>
<tr>
<td>DB</td>
<td>define data bytes</td>
</tr>
<tr>
<td>DW</td>
<td>define data words</td>
</tr>
<tr>
<td>DS</td>
<td>define data storage area</td>
</tr>
</tbody>
</table>

The individual pseudo operations are detailed below

4.1. The ORG directive.

The ORG statement takes the form

label ORG expression

where "label" is an optional program label, and expression is a 16-bit expression, consisting of operands which are defined previous to the ORG statement. The assembler begins machine code generation at the location specified in the expression. There can be any number of ORG statements within a particular program, and there are no checks to ensure that the programmer is not defining overlapping memory areas. Note that most programs written for the CP/M system begin with an ORG statement of the form

ORG 100H
which causes machine code generation to begin at the base of the CP/M transient program area. If a label is specified in the ORG statement, then the label is given the value of the expression (this label can then be used in the operand field of other statements to represent this expression).

4.2. The END directive.

The END statement is optional in an assembly language program, but if it is present it must be the last statement (all subsequent statements are ignored in the assembly). The two forms of the END directive are

```
label END
label END expression
```

where the label is again optional. If the first form is used, the assembly process stops, and the default starting address of the program is taken as \texttt{0000}. Otherwise, the expression is evaluated, and becomes the program starting address (this starting address is included in the last record of the Intel formatted machine code "hex" file which results from the assembly). Thus, most CP/M assembly language programs end with the statement

```
END 100H
```

resulting in the default starting address of \texttt{100H} (beginning of the transient program area).

4.3. The EQU directive.

The EQU (equate) statement is used to set up synonyms for particular numeric values. the form is

```
label EQU expression
```

where the label must be present, and must not label any other statement. The assembler evaluates the expression, and assigns this value to the identifier given in the label field. The identifier is usually a name which describes the value in a more human-oriented manner. Further, this name is used throughout the program to "parameterize" certain functions. Suppose for example, that data received from a Teletype appears on a particular input port, and data is sent to the Teletype through the next output port in sequence. The series of equate statements could be used to define these ports for a particular hardware environment

```
TTYBASE EQU 10H ;BASE PORT NUMBER FOR TTY
TTYIN  EQU TTYBASE ;TTY DATA IN
TTYOUT EQU TTYBASE+1 ;TTY DATA OUT
```

At a later point in the program, the statements which access the Teletype could appear as
### 4.4 The SET Directive.

The SET statement is similar to the EQU, taking the form

```
label   SET   expression
```

except that the label can occur on other SET statements within the program. The expression is evaluated and becomes the current value associated with the label. Thus, the EQU statement defines a label with a single value, while the SET statement defines a value which is valid from the current SET statement to the point where the label occurs on the next SET statement. The use of the SET is similar to the EQU statement, but is used most often in controlling conditional assembly.

### 4.5 The IF and ENDIF directives.

The IF and ENDIF statements define a range of assembly language statements which are to be included or excluded during the assembly process. The form is

```
IF   expression
    statement#1
    statement#2
    ...
    statement#n
ENDIF
```

Upon encountering the IF statement, the assembler evaluates the expression following the IF (all operands in the expression must be defined ahead of the IF statement). If the expression evaluates to a non-zero value, then statement#1 through statement#n are assembled; if the expression evaluates to zero, then the statements are listed but not assembled. Conditional assembly is often used to write a single "generic" program which includes a number of possible run-time environments, with only a few specific portions of the program selected for any particular assembly. The following program segments for example, might be part of a program which communicates with either a Teletype or a CRT console (but not both) by selecting a particular value for TTY before the assembly begins.

```asm
IN   TTYIN   ;READ TTY DATA TO REG-A
...
OUT  TTYOUT  ;WRITE DATA TO TTY FROM REG-A
```

making the program more readable than if the absolute i/o ports had been used. Further, if the hardware environment is redefined to start the Teletype communications ports at 7FH instead of 10H, the first statement need only be changed to

```
TTYBASE   EQU   7FH   ;BASE PORT NUMBER FOR TTY
```

and the program can be reassembled without changing any other statements.
TRUE EQU $FFFFH ;DEFINE VALUE OF TRUE
FALSE EQU NOT TRUE ;DEFINE VALUE OF FALSE
TTY EQU TRUE ;TRUE IF TTY, FALSE IF CRT
TTYBASE EQU 10H ;BASE OF TTY I/O PORTS
CRTBASE EQU 20H ;BASE OF CRT I/O PORTS
IF TTY ;ASSEMBLE RELATIVE TO TTYBASE
CONIN EQU TTYBASE ;CONSOLE INPUT
CONOUT EQU TTYBASE+1 ;CONSOLE OUTPUT
ENDIF

IF NOT TTY ;ASSEMBLE RELATIVE TO CRTBASE
CONIN EQU CRTBASE ;CONSOLE INPUT
CONOUT EQU CRTBASE+1 ;CONSOLE OUTPUT
ENDIF

*** IN CONIN ;READ CONSOLE DATA
*** OUT CONOUT ;WRITE CONSOLE DATA

In this case, the program would assemble for an environment where a Teletype is connected, based at port 10H. The statement defining TTY could be changed to

TTY EQU FALSE

and, in this case, the program would assemble for a CRT based at port 20H.


The DB directive allows the programmer to define initialize storage areas in single precision (byte) format. The statement form is

LABEL DB e$1, e$2, ..., e$n

where e$1 through e$n are either expressions which evaluate to 8-bit values (the high order eight bits must be zero), or are ASCII strings of length no greater than 64 characters. There is no practical restriction on the number of expressions included on a single source line. The expressions are evaluated and placed sequentially into the machine code file following the last program address generated by the assembler. String characters are similarly placed into memory starting with the first character and ending with the last character. Strings of length greater than two characters cannot be used as operands in more complicated expressions (i.e., they must stand alone between the commas). Note that ASCII characters are always placed in memory with the parity bit reset (0). Further, recall that there is no translation from lower to upper case within strings. The optional label can be used to reference the data area throughout the remainder of the program. Examples of
valid DB statements are

```
data:   DB   0,1,2,3,4,5
         DB   data and 0ffh,5,3770,1+2+3+4
signon: DB   'please type your name',cr,lf,0
         DB   'AB' SHR 8, 'C', 'DE' AND 7FH
```

4.7. The DW Directive.

The DW statement is similar to the DB statement except double precision (two byte) words of storage are initialized. The form is

```
label   DW   e#1, e#2, ..., e#n
```

where e#1 through e#n are expressions which evaluate to 16-bit results. Note that ASCII strings of length one or two characters are allowed, but strings longer than two characters disallowed. In all cases, the data storage is consistent with the 8080 processor: the least significant byte of the expression is stored first in memory, followed by the most significant byte. Examples are

```
doub:   DW   0ffefh,doub+4,signon-,255+255
         DW   'a', 5, 'ab', 'CD', 6 shl 8 or 11b
```

4.8. The DS Directive.

The DS statement is used to reserve an area of uninitialized memory, and takes the form

```
label   DS   expression
```

where the label is optional. The assembler begins subsequent code generation after the area reserved by the DS. Thus, the DS statement given above has exactly the same effect as the statement

```
label:   EQU   $   ;LABEL VALUE IS CURRENT CODE LOCATION
         ORG   $+expression   ;MOVE PAST RESERVED AREA
```

5. OPERATION CODES.

Assembly language operation codes form the principal part of assembly language programs, and form the operation field of the instruction. In general, ASM accepts all the standard mnemonics for the Intel 8080 microcomputer, which are given in detail in the Intel manual "8080 Assembly Language Programming Manual." Labels are optional on each input line and, if included, take the value of the instruction address immediately before the instruction is issued. The individual operators are listed briefly in the
following sections for completeness, although it is understood that the Intel manuals should be referenced for exact operator details. In each case,

\[ e_3 \]\nrepresents a 3-bit value in the range 0-7
which can be one of the predefined registers
A, B, C, D, E, H, L, M, SP, or PSW.

\[ e_8 \]\nrepresents an 8-bit value in the range 0-255

\[ e_{16} \]\nrepresents a 16-bit value in the range 0-65535

which can themselves be formed from an arbitrary combination of operands and operators. In some cases, the operands are restricted to particular values within the allowable range, such as the PUSH instruction. These cases will be noted as they are encountered.

In the sections which follow, each operation codes is listed in its most general form, along with a specific example, with a short explanation and special restrictions.

5.1. Jumps, Calls, and Returns.

The Jump, Call, and Return instructions allow several different forms which test the condition flags set in the 8080 microcomputer CPU. The forms are

\[
\begin{align*}
\text{JMP} & \text{ e}_{16} \quad \text{JMP} \text{ L}_1 \quad \text{Jump unconditionally to label} \\
\text{JNZ} & \text{ e}_{16} \quad \text{JMP} \text{ L}_2 \quad \text{Jump on non zero condition to label} \\
\text{JZ} & \text{ e}_{16} \quad \text{JMP} \text{ L}_0H \quad \text{Jump on zero condition to label} \\
\text{JNC} & \text{ e}_{16} \quad \text{JNC} \text{ L}_1+4 \quad \text{Jump no carry to label} \\
\text{JC} & \text{ e}_{16} \quad \text{JC} \text{ L}_3 \quad \text{Jump on carry to label} \\
\text{JPO} & \text{ e}_{16} \quad \text{JPO} \text{ S}+8 \quad \text{Jump on parity odd to label} \\
\text{JPE} & \text{ e}_{16} \quad \text{JPE} \text{ L}_4 \quad \text{Jump on even parity to label} \\
\text{JP} & \text{ e}_{16} \quad \text{JP} \text{ GAMMA} \quad \text{Jump on positive result to label} \\
\text{JM} & \text{ e}_{16} \quad \text{JM} \text{ a}1 \quad \text{Jump on minus to label} \\
\text{CALL} & \text{ e}_{16} \quad \text{CALL} \text{ S}_1 \quad \text{Call subroutine unconditionally} \\
\text{C NZ} & \text{ e}_{16} \quad \text{C NZ} \text{ S}_2 \quad \text{Call subroutine if non zero flag} \\
\text{CZ} & \text{ e}_{16} \quad \text{CZ} \text{ 100H} \quad \text{Call subroutine on zero flag} \\
\text{CNC} & \text{ e}_{16} \quad \text{CNC} \text{ S}_1+4 \quad \text{Call subroutine if no carry set} \\
\text{CC} & \text{ e}_{16} \quad \text{CC} \text{ S}_3 \quad \text{Call subroutine if carry set} \\
\text{CPO} & \text{ e}_{16} \quad \text{CPO} \text{ S}+8 \quad \text{Call subroutine if parity odd} \\
\text{CPE} & \text{ e}_{16} \quad \text{CPE} \text{ S}_4 \quad \text{Call subroutine if parity even} \\
\text{CP} & \text{ e}_{16} \quad \text{CP} \text{ GAMMA} \quad \text{Call subroutine if positive result} \\
\text{CM} & \text{ e}_{16} \quad \text{CM} \text{ b1$c}2 \quad \text{Call subroutine if minus flag} \\
\text{RST} & \text{ e}_{3} \quad \text{RST} \text{ 0} \quad \text{Programmed "restart", equivalent to CALL 8*e}_{3}, except one byte call}
\end{align*}
\]
5.2. Immediate Operand Instructions.

Several instructions are available which load single or double precision registers, or single precision memory cells, with constant values, along with instructions which perform immediate arithmetic or logical operations on the accumulator (register A).

- **MVI e3,e8**: Move immediate data to register A, B, C, D, E, H, L, or M (memory)
- **ADI e8**: Add immediate operand to A without carry
- **ACI e8**: Add immediate operand to A with carry
- **SUI e8**: Subtract from A without borrow (carry)
- **SBI e8**: Subtract from A with borrow (carry)
- **ANI e8**: Logical "and" A with immediate data
- **XRI e8**: "Exclusive or" A with immediate data
- **ORI e8**: Logical "or" A with immediate data
- **CPI e8**: Compare A with immediate data (same as SUI except register A not changed)

- **LXI e3,e16**: Load extended immediate to register pair (e3 must be equivalent to B,D,H, or SP)

5.3. Increment and Decrement Instructions.

Instructions are provided in the 8080 repertoire for incrementing or decrementing single and double precision registers. The instructions are

- **INR e3**: Single precision increment register (e3 produces one of A, B, C, D, E, H, L, M)
- **DCR e3**: Single precision decrement register (e3 produces one of A, B, C, D, E, H, L, M)
- **INX e3**: Double precision increment register pair (e3 must be equivalent to B,D,H, or SP)
- **DCX e3**: Double precision decrement register pair (e3 must be equivalent to B,D,H, or SP)

5.4. Data Movement Instructions.
Instructions which move data from memory to the CPU and from CPU to memory are given below

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV e3,e3</td>
<td>Move data to leftmost element from rightmost element (e3 produces one of A, B, C, D, E, H, L, or M). MOV M,M is disallowed.</td>
</tr>
<tr>
<td>LDAX e3</td>
<td>Load register A from computed address (e3 must produce either B or D).</td>
</tr>
<tr>
<td>STAX e3</td>
<td>Store register A to computed address (e3 must produce either B or D).</td>
</tr>
<tr>
<td>LHL e16</td>
<td>Load HL direct from location e16 (double precision load to H and L).</td>
</tr>
<tr>
<td>SHLD e16</td>
<td>Store HL direct to location e16 (double precision store from H and L to memory).</td>
</tr>
<tr>
<td>LDA e16</td>
<td>Load register A from address e16.</td>
</tr>
<tr>
<td>STA e16</td>
<td>Store register A into memory at e16.</td>
</tr>
<tr>
<td>POP e3</td>
<td>Load register pair from stack, set SP (e3 must produce one of B, D, H, or PSW).</td>
</tr>
<tr>
<td>PUSH e3</td>
<td>Store register pair into stack, set SP (e3 must produce one of B, D, H, or PSW).</td>
</tr>
<tr>
<td>IN e8</td>
<td>Load register A with data from port e8.</td>
</tr>
<tr>
<td>OUT e8</td>
<td>Send data from register A to port e8.</td>
</tr>
<tr>
<td>XTHL</td>
<td>Exchange data from top of stack with HL.</td>
</tr>
<tr>
<td>PCHL</td>
<td>Fill program counter with data from HL.</td>
</tr>
<tr>
<td>SPHL</td>
<td>Fill stack pointer with data from HL.</td>
</tr>
<tr>
<td>XCHG</td>
<td>Exchange DE pair with HL pair.</td>
</tr>
</tbody>
</table>

5.5. Arithmetic Logic Unit Operations.

Instructions which act upon the single precision accumulator to perform arithmetic and logic operations are

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD e3</td>
<td>Add register given by e3 to accumulator without carry (e3 must produce one of A, B, C, D, E, H, or L).</td>
</tr>
<tr>
<td>ADC e3</td>
<td>Add register to A with carry, e3 as above.</td>
</tr>
<tr>
<td>SUB e3</td>
<td>Subtract reg e3 from A without carry, e3 is defined as above.</td>
</tr>
<tr>
<td>SBB e3</td>
<td>Subtract register e3 from A with carry, e3 defined as above.</td>
</tr>
<tr>
<td>ANA e3</td>
<td>Logical &quot;and&quot; reg with A, e3 as above.</td>
</tr>
<tr>
<td>XRA e3</td>
<td>&quot;Exclusive or&quot; with A, e3 as above.</td>
</tr>
<tr>
<td>ORA e3</td>
<td>Logical &quot;or&quot; with A, e3 defined as above.</td>
</tr>
<tr>
<td>CMP e3</td>
<td>Compare register with A, e3 as above.</td>
</tr>
<tr>
<td>DAA</td>
<td>Decimal adjust register A based upon last arithmetic logic unit operation.</td>
</tr>
<tr>
<td>CMA</td>
<td>Complement the bits in register A.</td>
</tr>
<tr>
<td>STC</td>
<td>Set the carry flag to 1.</td>
</tr>
</tbody>
</table>
5.6. Control Instructions.

The four remaining instructions are categorized as control instructions, and are listed below:

- **CMC**
  - Complement the carry flag

- **RLC**
  - Rotate bits left, (re)set carry as a side effect (high order A bit becomes carry)

- **RRC**
  - Rotate bits right, (re)set carry as side effect (low order A bit becomes carry)

- **RAL**
  - Rotate carry/A register to left (carry is involved in the rotate)

- **RAR**
  - Rotate carry/A register to right (carry is involved in the rotate)

- **DAD e3**
- **DAD B**
  - Double precision add register pair e3 to HL (e3 must produce B, D, H, or SP)

5. ERROR MESSAGES.

When errors occur within the assembly language program, they are listed as single character flags in the leftmost position of the source listing. The line in error is also echoed at the console so that the source listing need not be examined to determine if errors are present. The error codes are:

- **D** Data error: element in data statement cannot be placed in the specified data area

- **E** Expression error: expression is ill-formed and cannot be computed at assembly time

- **L** Label error: label cannot appear in this context (may be duplicate label)

- **N** Not implemented: features which will appear in future ASM versions (e.g., macros) are recognized, but flagged in this version

- **O** Overflow: expression is too complicated (i.e., too many pending operators) to computed, simplify it

- **P** Phase error: label does not have the same value on two subsequent passes through the program
R Register error: the value specified as a register is not compatible with the operation code

V Value error: operand encountered in expression is improperly formed

Several error messages are printed which are due to terminal error conditions

NO SOURCE FILE PRESENT The file specified in the ASM command does not exist on disk

NO DIRECTORY SPACE The disk directory is full, erase files which are not needed, and retry

SOURCE FILE NAME ERROR Improperly formed ASM file name (e.g., it is specified with "?" fields)

SOURCE FILE READ ERROR Source file cannot be read properly by the assembler, execute a TYPE to determine the point of error

OUTPUT FILE WRITE ERROR Output files cannot be written properly, most likely cause is a full disk, erase and retry

CANNOT CLOSE FILE Output file cannot be closed, check to see if disk is write protected

7. A SAMPLE SESSION.

The following session shows interaction with the assembler and debugger in the development of a simple assembly language program.
ASSEMBLE SORT.ASM

CP/M ASSEMBLER - VER 1.0

015C NEXT FREE ADDRESS
003H USE FACTOR % of table used 00 TO FF (hexadecimal)

END OF ASSEMBLY

SORT ASM source file
SORT BAK backup from last edit
SORT PRN print file (Contains tab characters)
SORT HEX machine code file
A TYPE SORT.PRIN

Machine code location

0100
0100 214681 SORT: LKI H, SW ; ADDRESS SWITCH TOGGLE
0103 3601 MVI M, I ; SET TO 1 FOR FIRST ITERATION
0105 214701 LKI H, I ; ADDRESS INDEX
0108 3600 MVI M, 0 ; I = 0

010A 7E COMP: MOV A, M ; A REGISTER = I
010B FE09 CPI N-1 ; CY SET IF I < (N-1)
010D D21901 JNC CONT ; CONTINUE IF I < = (N-2)

0110 214681 LKI H, SW ; CHECK FOR ZERO SWITCHES
0113 7EB7C20001 MOV A, M! ORA A! JNZ SORT ; END OF SORT IF SW=0

0118 FF RST 7 ; GO TO THE Debugger INSTEAD OF RE:

CONTINUE THIS PASS

0119 5F16802148CONT: MOV E, A! MOV D, 0! LBI H, AY! DAD D! DAD D
0121 4E792346 MOV C, M! MOV A, C! INX H! MOV B, M

LOW ORDER BYTE IN A AND C, HIGH ORDER BYTE IN B

0125 23 INX H

0126 965778239E SUB M! MOV D, A! MOV A, B! INX H! SBB M ; SUBTRACT

0128 DA3F01 JC INCI ; SKIP IF IN PROPER ORDER

012E B2CA3F01 ORA D! JZ INCI ; SKIP IF AY(I) = AY(I+1)
8132 56702B5E MDV D,M! MOV M,B! DCX H! MDV E,M
8136 712B722873 MDV M,C! DCX H! MOV M,D! DCX H! MOV M,E

INCREMENT SWITCH COUNT
LXI H,SW! INR M
INCREMENT I
LXI H,I! INR M! JMP COMP

DATA DEFINITION SECTION
DB 0 ;RESERVE SPACE FOR SWITCH COUNT
DS 1 ;SPACE FOR INDEX
DW 5,100,30,50,20,7,1000,300,100,-32767
EQU ($-AY)2 ;COMPUTE N INSTEAD OF PRE

;RESERVE SPACE FOR SWITCH COUNT 8147
;SPACE FOR INDEX

0146 00 SW.
0147 I:
0148 05806401EAV.
000A H
015C ＝equate value

A>TYPE SORT.HEX

16K DDT VER 1.0
NEXT PC
015C 0000 default address (no address on END statement)

P=0008 180, change PC to 100

-UFFFF, untrace for 65535 steps

COZOMBE010 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 LXI H,0146+0100

-T18, trace 10, steps

010D, change to a jump on carry
011D JC 119, change to a jump on carry

Stopped at 108H
P=0100 100, reset program counter back to beginning of program

-T10, trace execution for 100 steps

C0Z0M8E010 A=00 B=0000 D=0000 H=0147 S=0100 P=0100 LXI H,0146
C0Z0M8E010 A=00 B=0000 D=0000 H=0146 S=0100 P=0103 MVI M,01
C0Z0M8E010 A=00 B=0000 D=0000 H=0146 S=0100 P=0105 LXI H,0147
C0Z0M8E010 A=00 B=0000 D=0000 H=0147 S=0100 P=0108 MVI M,00
C0Z0M8E010 A=00 B=0000 D=0000 H=0147 S=0100 P=010A MOY A,M
C0Z0M8E010 A=00 B=0000 D=0000 H=0147 S=0100 P=010B CPI 09
C1Z0M1E010 A=00 B=0000 D=0000 H=0147 S=0100 P=0110D JC 0119
C1Z0M1E010 A=00 B=0000 D=0000 H=0147 S=0100 P=0111A MVI D,00
C1Z0M1E010 A=00 B=0000 D=0000 H=0147 S=0100 P=0111C LXI H,0148
C1Z0M1E010 A=00 B=0000 D=0000 H=0148 S=0100 P=0111F DAD D
C0Z0M8E010 A=00 B=0000 D=0000 H=0148 S=0100 P=0120 DAD D
C0Z0M8E010 A=00 B=0005 D=0000 H=0148 S=0100 P=0121 MOY C,M
C0Z0M8E010 A=05 B=0005 D=0000 H=0148 S=0100 P=0122 MOY A,C
C0Z0M8E010 A=05 B=0005 D=0000 H=0148 S=0100 P=0123 INX H
C0Z0M8E010 A=05 B=0005 D=0000 H=0149 S=0100 P=0124 MOY B,M=0125

0100 LXI H,0146
0103 MVI M,01
0105 LXI H,0147
0108 MVI M,00
010A MOY A,M
010B CPI 09
010C JC 0119
0110 LXI H,0146
0113 MOY A,M
0114 ORA A
0115 JNZ 0100

list some code from 100K

0118 RST 07
0119 MOY E,A
011A MVI D,00
011C LXI H,0148

- about list with rebutt

-C,118, start program from current PC (0125H) and run in real time to 115H

*0127 stopped with an external interrupt 7 from front panel (program was looping indefinitely)

C0Z0M8E010 A=38 B=006D D=0906 H=0156 S=0100 P=0127 MOY D,A
C0Z0M8E010 A=38 B=006D D=3006 H=0156 S=0100 P=0128 MOY A,B
C0Z0M8E010 A=00 B=006D D=3006 H=0156 S=0100 P=0129 INX H
C0Z0M8E010 A=00 B=006D D=3806 H=0157 S=0100 P=012A SBB M=012B
-0148

-data is sorted, but program doesn't stop.

0148 05 00 07 00 14 00 1E 00 ...........
0150 32 00 64 00 64 00 2C 01 E8 03 01 00 00 00 00 00 2.D.D ..............
0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..............
-G0, return to CP/M

DDT SORT. HEX, reload the memory image

16K DDT VER 1.0
NEXT PC
015C 0000
-XP

P=000B 100, Set PC to beginning of program
-L10D, list bad opcode
010D JNC 0119
0118 LXI H.0146
- abort list with rubout
-A10D, assemble new opcode
010D JC 119

0110
-L100, list starting section of program
0100 LXI H.0146
0103 MVI M.01
0105 LXI H.0147
0108 MVI M.00
- abort list with rubout
-A103, change "switch" initialization to GO
0103 MVI M.0,

0105,
-
C return to CP/M with ctrl-C (G0 works as well)

SAVE 1 SORT.COM, save 1 page (256 bytes from 100H to AFFH) on disk in case we have to reload later

A) DDT SORT.COM, restart DDT with
Saved memory image

16K DDT VER 1.0
NEXT PC
0200 0100 "Com" file always starts with address 100H
- GO, run the program from PC=100H

*0118 programed stop (RST7) encountered
-D148

0148 05 00 07 00 14 00 1E 00
0150 32 00 64 00 64 00 2C 01 E8 03 01 80 00 00 00 00 2. D.D.,
0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

-G0, return to CP/M

91
ED.SORT.ASM  make changes to original program

ch.2
* N, O DTT: find next "$"
  MVI M, 0; I = 0
  *-2 up one line in text
  LXI H, I; ADDRESS INDEX
  *-2 up another line
  MVI M, I; SET TO 1 FOR FIRST ITERATION
  *KT; kill line and type next line
  LXI H, I; ADDRESS INDEX
  *I; insert new line
  MVI M, 0; ZERO SW
  *T;
  LXI H, I; ADDRESS INDEX
  *NHNC
  JNC  T; CONTINUE IF I <= (N-2)
  *-2DINC
  CONT ; CONTINUE IF I <= (N-2)
  *E
  source from disk A
  USE FACTOR
  END OF ASSEMBLY

ASM SORT.AAZ  skip prin file

CP/M ASSEMBLER - VER 1.0

815C next address to assemble
003H USE FACTOR
END OF ASSEMBLY

DDT SORT.HEX, test program changes

16K DDT VER 1.0
NEXT PC
815C 0000
-G1002

0110
-D1492

0148 05 00 07 00 14 00 1E 00
0150 32 00 64 00 64 00 2C 01 E8 03 01 80 00 00 00 00 00 00 00 00 00 00 00 00 .2. D. D.
0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
-
abort withrubout
-C0, return to CP/M - program checks OK.
ED: A CONTEXT EDITOR FOR THE CP/M DISK SYSTEM: USER'S MANUAL
COPYRIGHT

Copyright (c) 1980 by Exidy Systems, Inc. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of Exidy Systems, Inc., 1234 Elko Drive, Sunnyvale, California 94086.

Since this manual is tutorial in nature, permission is granted to reproduce or abstract the example procedures and sample programs for the purposes of inclusion within the reader's programs.

DISCLAIMER

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

TRADEMARKS

CP/M is a registered trademark of Digital Research.
ED: A CONTEXT EDITOR FOR THE CP/M DISK SYSTEM: USER'S MANUAL
COPYRIGHT

Copyright (c) 1980 by Exidy Systems, Inc. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of Exidy Systems, Inc., 1234 Elko Drive, Sunnyvale, California 94086.

Since this manual is tutorial in nature, permission is granted to reproduce or abstract the example procedures and sample programs for the purposes of inclusion within the reader's programs.

DISCLAIMER

Exidy Systems, Inc. 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, Exidy Systems, Inc. reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Exidy Systems, Inc. to notify any person of such revision or changes.

TRADEMARKS

CP/M is a registered trademark of Digital Research.
Table Of Contents

ED: A Content Editor For The CP/M Disk System:  
User's Manual

1. ED Tutorial ............................................... 93
   1.1 Introduction to ED.................................. 93
   1.2 ED Operation .................................. 93
   1.3 Text Transfer Functions .......................... 93
   1.4 Memory Buffer Organization .................. 97
   1.5 Memory Buffer Operation ...................... 97
   1.6 Command Strings ................................ 99
   1.7 Text Search and Alteration ................. 100
   1.8 Source Libraries ................................. 103
   1.9 Repetitive Command Execution ............. 104
2. ED Error Conditions ................................. 105
3. Control Characters And Commands ............ 106
1. ED TUTORIAL

1.1. Introduction to ED.

ED is the context editor for CP/M, and is used to create and alter CP/M source files. ED is initiated in CP/M by typing

```
ED { <filename>
    <filename>.<filetype>
}
```

In general, ED reads segments of the source file given by `<filename>` or `<filename>..<filetype>` into central memory, where the file is manipulated by the operator, and subsequently written back to disk after alterations. If the source file does not exist before editing, it is created by ED and initialized to empty. The overall operation of ED is shown in Figure 1.

1.2. ED Operation

ED operates upon the source file, denoted in Figure 1 by x.y, and passes all text through a memory buffer where the text can be viewed or altered (the number of lines which can be maintained in the memory buffer varies with the line length, but has a total capacity of about 6000 characters in a 16K CP/M system). Text material which has been edited is written onto a temporary work file under command of the operator. Upon termination of the edit, the memory buffer is written to the temporary file, followed by any remaining (unread) text in the source file. The name of the original file is changed from x.y to x.BAK so that the most recent previously edited source file can be reclaimed if necessary (see the CP/M commands ERASE and RENAME). The temporary file is then changed from x.$$ to x.y which becomes the resulting edited file.

The memory buffer is logically between the source file and working file as shown in Figure 2.

1.3. Text Transfer Functions

Given that n is an integer value in the range 0 through 65535, the following ED commands transfer lines of text from the source file through the memory buffer to the temporary (and eventually final) file:
Note: the ED program accepts both lower and upper case ASCII characters as input from the console. Single letter commands can be typed in either case. The U command can be issued to cause ED to translate lower case alphabets to upper case as characters are filled to the memory buffer from the console. Characters are echoed as typed without translation, however. The -U command causes ED to revert to "no translation" mode. ED starts with an assumed -U in effect.
Figure 2. Memory Buffer Organization

Figure 3. Logical Organization of Memory Buffer

Memory Buffer

- first line: 
  \text{\textless cr\textless lf}

- current line CL: 
  \text{\textless cr\textless lf}

- last line: 
  \text{\textless cr\textless lf}
nA<cr>* - append the next n unprocessed source
lines from the source file at SP to
the end of the memory buffer at MP.
Increment SP and MP by n.

nW<cr> - write the first n lines of the memory
buffer to the temporary file free space.
Shift the remaining lines n+1 through
MP to the top of the memory buffer.
Increment TP by n.

E<cr> - end the edit. Copy all buffered text
to temporary file, and copy all unprocessed source lines to the temporary
file. Rename files as described
previously.

H<cr> - move to head of new file by performing
automatic E command. Temporary file
becomes the new source file, the memory
buffer is emptied, and a new temporary
file is created (equivalent to issuing
an E command, followed by a reinvocation
of ED using x.y as the file to edit).

O<cr> - return to original file. The memory
buffer is emptied, the temporary file
id deleted, and the SP is returned to
position 1 of the source file. The
effects of the previous editing commands
are thus nullified.

Q<cr> - quit edit with no file alterations,
return to CP/M.

There are a number of special cases to consider. If the
integer n is omitted in any ED command where an integer is
allowed, then 1 is assumed. Thus, the commands A and W append
one line and write 1 line, respectively. In addition, if a
pound sign (#) is given in the place of n, then the integer
65535 is assumed (the largest value for n which is allowed).
Since most reasonably sized source files can be contained
entirely in the memory buffer, the command #A is often issued
at the beginning of the edit to read the entire source file
to memory. Similarly, the command #W writes the entire buffer
to the temporary file. Two special forms of the A and W

<cr> represents the carriage-return key
commands are provided as a convenience. The command OA fills the current memory buffer to at least half-full, while OW writes lines until the buffer is at least half empty. It should also be noted that an error is issued if the memory buffer size is exceeded. The operator may then enter any command (such as W) which does not increase memory requirements. The remainder of any partial line read during the overflow will be brought into memory on the next successful append.

1.4. Memory Buffer Organization

The memory buffer can be considered a sequence of source lines brought in with the A command from a source file. The memory buffer has an associated (imaginary) character pointer CP which moves throughout the memory buffer under command of the operator. The memory buffer appears logically as shown in Figure 3 where the dashes represent characters of the source line of indefinite length, terminated by carriage-return (<cr>) and line-feed (<lf>) characters, and \[\text{CP}\] represents the imaginary character pointer. Note that the CP is always located ahead of the first character of the first line, behind the last character of the last line, or between two characters. The current line CL is the source line which contains the CP.

1.5. Memory Buffer Operation

Upon initiation of ED, the memory buffer is empty (ie, CP is both ahead and behind the first and last character). The operator may either append lines (A command) from the source file, or enter the lines directly from the console with the insert command

\[\text{I<cr>}\]

ED then accepts any number of input lines, where each line terminates with a <cr> (the <lf> is supplied automatically), until a control-z (denoted by tz is typed by the operator. The CP is positioned after the last character entered. The sequence

\[\begin{align*}
\text{I<cr>} \\
\text{NOW IS THE<cr>} \\
\text{TIME FOR<cr>} \\
\text{ALL GOOD MEN<cr>} \\
\text{tz}
\end{align*}\]

leaves the memory buffer as shown below
Various commands can then be issued which manipulate the CP or display source text in the vicinity of the CP. The commands shown below with a preceding n indicate that an optional unsigned value can be specified. When preceded by +, the command can be unsigned, or have an optional preceding plus or minus sign. As before, the pound sign (#) is replaced by 65535. If an integer n is optional, but not supplied, then n=1 is assumed. Finally, if a plus sign is optional, but none is specified, then + is assumed.

±B<cr> - move CP to beginning of memory buffer if +, and to bottom if -.

±nC<cr> - move CP by ±n characters (toward front of buffer if +), counting the <cr><lf> as two distinct characters.

±nD<cr> - delete n characters ahead of CP if plus and behind CP if minus.

±nK<cr> - kill (ie remove) ±n lines of source text using CP as the current reference. If CP is not at the beginning of the current line when K is issued, then the characters before CP remain if + is specified, while the characters after CP remain if - is given in the command.

±nL<cr> - if n=0 then move CP to the beginning of the current line (if it is not already there) if n≠0 then first move the CP to the beginning of the current line, and then move it to the beginning of the line which is n lines down (if +) or up (if -). The CP will stop at the top or bottom of the memory buffer if too large a value of n is specified.
\[ \pm n<cr> \] - If \( n=0 \) then type the contents of the current line up to CP. If \( n=1 \) then type the contents of the current line from CP to the end of the line. If \( n>1 \) then type the current line along with \( n-1 \) lines which follow, if + is specified. Similarly, if \( n>1 \) and - is given, type the previous \( n \) lines, up to the CP. The break key can be depressed to abort long type-outs.

\[ \pm n<cr> \] - equivalent to \( \pm nLT \), which moves up or down and types a single line

1.6. Command Strings

Any number of commands can be typed contiguously (up to the capacity of the CP/M console buffer), and are executed only after the \(<cr>\) is typed. Thus, the operator may use the CP/M console command functions to manipulate the input command:

- Rubout: remove the last character
- Control-U: delete the entire line
- Control-C: re-initialize the CP/M System
- Control-E: return carriage for long lines without transmitting buffer (max 128 chars)

Suppose the memory buffer contains the characters shown in the previous section, with the CP following the last character of the buffer. The command strings shown below produce the results shown to the right:

<table>
<thead>
<tr>
<th>Command String</th>
<th>Effect</th>
<th>Resulting Memory Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. 5C0T&lt;cr&gt;</td>
<td>move CP 5 characters and type the beginning of the line &quot;NOW I&quot;</td>
<td>[NOW I&lt;CP&gt;S THE&lt;cr&gt;&lt;lf&gt;&lt;lf&gt;]</td>
</tr>
</tbody>
</table>
3. 2L-T<cr> move two lines down and type previous line "TIME FOR"
NOW IS THE<cr><lf>TIME FOR<cr><lf>ALL GOOD MEN<cr><lf>

4. -L#K<cr> move up one line, delete 65535 lines which follow
NOW IS THE<cr><lf>

5. I<cr> insert two lines of text
TIME TO<cr><lf>INSERT<cr><lf>tz
NOW IS THE<cr><lf>TIME TO<cr><lf>INSERT<cr><lf>

6. -2L#T<cr> move two lines, and type 65535 lines ahead of CP "NOW IS THE"
NOW IS THE<cr><lf>TIME TO<cr><lf>INSERT<cr><lf>

7. <cr> move down one line and type one line "INSERT"
NOW IS THE<cr><lf>TIME TO<cr><lf>INSERT<cr><lf>

1.7. Text Search and Alteration

ED also has a command which locates strings within the memory buffer. The command takes the form

\[ nF \ c_1 c_2 \ldots c_k \ {<cr>} \{tz\} \]

where \( c_1 \) through \( c_k \) represent the characters to match followed by either a \(<cr>\) or control -z. ED starts at the current position of CP and attempts to match all \( k \) characters. The match is attempted \( n \) times, and if successful, the CP is moved directly after the character \( c_k \). If the \( n \) matches are not successful, the CP is not moved from its initial position. Search strings can include \( \text{^}l \) (control-1), which is replaced by the pair of symbols \(<cr><lf>\).

*The control-z is used if additional commands will be typed following the \( \text{^}z \).
The following commands illustrate the use of the F command:

<table>
<thead>
<tr>
<th>Command String</th>
<th>Effect</th>
<th>Resulting Memory Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. B#T&lt;cr&gt;</td>
<td>move to beginning and type entire buffer</td>
<td>NOW IS THE&lt;cr&gt;&lt;lf&gt; TIME FOR&lt;cr&gt;&lt;lf&gt; ALL GOOD MEN&lt;cr&gt;&lt;lf&gt;</td>
</tr>
<tr>
<td>2. FS T&lt;cr&gt;</td>
<td>find the end of the string &quot;ST&quot;</td>
<td>NOW IS T&lt;cr&gt;&lt;lf&gt; NOW IS THE&lt;cr&gt;&lt;lf&gt;</td>
</tr>
<tr>
<td>3. Fi+zOTT</td>
<td>find the next &quot;I&quot; and type to the CP then type the remainder of the current line: &quot;TIME FOR&quot;</td>
<td>NOW IS THE&lt;cr&gt;&lt;lf&gt; ME FOR&lt;cr&gt;&lt;lf&gt; ALL GOOD MEN&lt;cr&gt;&lt;lf&gt;</td>
</tr>
</tbody>
</table>

An abbreviated form of the insert command is also allowed, which is often used in conjunction with the F command to make simple textual changes. The form is:

\[ I \ c_1 c_2 \ldots c_n +z \quad \text{or} \quad I \ c_1 c_2 \ldots c_n <cr> \]

where \( c_1 \) through \( c_n \) are characters to insert. If the insertion string is terminated by a +z, the characters \( c_1 \) through \( c_n \) are inserted directly following the CP, and the CP is moved directly after character \( c_n \). The action is the same if the command is followed by a <cr> except that a <cr><lf> is automatically inserted into the text following character \( c_n \). Consider the following command sequences as examples of the F and I commands:

<table>
<thead>
<tr>
<th>Command String</th>
<th>Effect</th>
<th>Resulting Memory Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>BITHIS IS +z&lt;cr&gt;</td>
<td>Insert &quot;THIS IS &quot; at the beginning of the text</td>
<td>THIS IS NOW THE &lt;cr&gt;&lt;lf&gt; TIME FOR&lt;cr&gt;&lt;lf&gt; ALL GOOD MEN&lt;cr&gt;&lt;lf&gt;</td>
</tr>
</tbody>
</table>
FTIME+z-4DIPLACE+z<cr>
find "TIME" and delete it; then insert "PLACE"

3FO+Z-3D5DICHANGES+cr>
find third occurrence of "0" (ie the second "0" in GOOD), delete previous 3 characters; then insert "CHANGES"

-8CISOURE<cr>
move back 8 characters and insert the line "SOURCE<cr><lf>"

ED also provides a single command which combines the F and I commands to perform simple string substitutions. The command takes the form

\[ n \ S \ c_1\ c_2 \ldots c_k \ t z \ d_1\ d_2 \ldots d_m \ \{ \ <cr> \} \]

and has exactly the same effect as applying the command string

\[ F \ c_1\ c_2 \ldots c_k \ t z \ -\ k \ D I d_1\ d_2 \ldots d_m \ \{ \ <cr> \} \]

a total of n times. That is, ED searches the memory buffer starting at the current position of CP and successively substitutes the second string for the first string until the end of buffer, or until the substitution has been performed n times.

As a convenience, a command similar to F is provided by ED which automatically appends and writes lines as the search proceeds. The form is

\[ n \ N \ c_1\ c_2 \ldots c_k \ \{ \ <cr> \} \]

which searches the entire source file for the nth occurrence of the string \(c_1c_2\ldots c_k\) (recall that F fails if the string cannot be found in the current buffer). The operation of the
The N command is precisely the same as F except in the case that the string cannot be found within the current memory buffer. In this case, the entire memory contents is written (i.e., an automatic \#W is issued). Input lines are then read until the buffer is at least half full, or the entire source file is exhausted. The search continues in this manner until the string has been found \( n \) times, or until the source file has been completely transferred to the temporary file.

A final line editing function, called the juxtaposition command takes the form

\[
\text{n J } c_1c_2\ldots c_k+z \ d_1d_2\ldots d_m+z \ e_1e_2\ldots e_q \{<cr><cr><cr>\}
\]

with the following action applied \( n \) times to the memory buffer: search from the current CP for the next occurrence of the string \( c_1c_2\ldots c_k \). If found, insert the string \( d_1d_2\ldots d_m \), and move CP to follow \( d_m \). Then delete all characters following CP up to (but not including) the string \( e_1e_2\ldots e_q \), leaving CP directly after \( d_m \). If \( e_1e_2\ldots e_q \) cannot be found, then no deletion is made. If the current line is

\[
\text{
\begin{array}{c}
\text{NOW IS THE TIME}<cr><lf>
\end{array}
}
\]

Then the command

\[
\text{JW } ^t z\text{WHAT}^t z+t l<cr>
\]

Results in

\[
\text{
\begin{array}{c}
\text{NOW WHAT}<cr><lf>
\end{array}
}
\]

(Recall that \( t l \) represents the pair \(<cr><lf>\) in search and substitute strings).

It should be noted that the number of characters allowed by ED in the F, S, N, and J commands is limited to 100 symbols.

1.8. Source Libraries

ED also allows the inclusion of source libraries during the editing process with the R command. The form of this command is
R $f_1 f_2 \ldots f_n^+ z$ or
R $f_1 f_2 \ldots f_n<cr>

where $f_1 f_2 \ldots f_n$ is the name of a source file on the disk with as assumed filetype of 'LIB'. ED reads the specified file, and places the characters into the memory buffer after CP, in a manner similar to the I command. Thus, if the command
RMACRO<cr>
is issued by the operator, ED reads from the file MACRO.LIB until the end-of-file, and automatically inserts the characters into the memory buffer.

1.9. Repetitive Command Execution

The macro command M allows the ED user to group ED commands together for repeated evaluation. The M command takes the form:

\[
n M c_1 c_2 \ldots c_k \{<cr>\}
\]

where $c_1 c_2 \ldots c_k$ represent a string of ED commands, not including another M command. ED executes the command string $n$ times if $n \geq 1$. If $n=0$ or 1, the command string is executed repetitively until an error condition is encountered (e.g., the end of the memory buffer is reached with an F command).

As an example, the following macro changes all occurrences of GAMMA to DELTA within the current buffer, and types each line which is changed:

MFGAMMA+z-5DIDELTA+z0TT<cr>
or equivalently

MSGAMMA+zDELTAz0TT<cr>
2. ED ERROR CONDITIONS

On error conditions, ED prints the last character read before the error, along with an error indicator:

- `?` unrecognized command
- `>` memory buffer full (use one of the commands D,K,N,S, or W to remove characters), F,N, or S strings too long.
- `#` cannot apply command the number of times specified (e.g., in F command)
- `0` cannot open LIB file in R command

Cyclic redundancy check (CRC) information is written with each output record under CP/M in order to detect errors on subsequent read operations. If a CRC error is detected, CP/M will type

```
PERM ERR DISK d
```

where d is the currently selected drive (A,B,...). The operator can choose to ignore the error by typing any character at the console (in this case, the memory buffer data should be examined to see if it was incorrectly read), or the user can reset the system and reclaim the backup file, if it exists. The file can be reclaimed by first typing the contents of the BAK file to ensure that it contains the proper information:

```
TYPE x.BAK<cr>
```

where x is the file being edited. Then remove the primary file:

```
ERA x.y<cr>
```

and rename the BAK file:

```
REN x.y=x.BAK<cr>
```

The file can then be re-edited, starting with the previous version.
3. CONTROL CHARACTERS AND COMMANDS

The following table summarizes the control characters and commands available in ED:

<table>
<thead>
<tr>
<th>Control Character</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>+c</td>
<td>system reboot</td>
</tr>
<tr>
<td>+e</td>
<td>physical &lt;cr&gt;&lt;lf&gt; (not actually entered in command)</td>
</tr>
<tr>
<td>+i</td>
<td>logical tab (cols 1, 8, 15, ...)</td>
</tr>
<tr>
<td>+l</td>
<td>logical &lt;cr&gt;&lt;lf&gt; in search and substitute strings</td>
</tr>
<tr>
<td>+u</td>
<td>line delete</td>
</tr>
<tr>
<td>+z</td>
<td>string terminator</td>
</tr>
<tr>
<td>rubout</td>
<td>character delete</td>
</tr>
<tr>
<td>break</td>
<td>discontinue command (e.g., stop typing)</td>
</tr>
<tr>
<td>Command</td>
<td>Function</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
</tr>
<tr>
<td>nA</td>
<td>append lines</td>
</tr>
<tr>
<td>±B</td>
<td>begin bottom of buffer</td>
</tr>
<tr>
<td>±nC</td>
<td>move character positions</td>
</tr>
<tr>
<td>±nD</td>
<td>delete characters</td>
</tr>
<tr>
<td>E</td>
<td>end edit and close files (normal end)</td>
</tr>
<tr>
<td>nF</td>
<td>find string</td>
</tr>
<tr>
<td>H</td>
<td>end edit, close and reopen files</td>
</tr>
<tr>
<td>I</td>
<td>insert characters</td>
</tr>
<tr>
<td>nJ</td>
<td>place strings in juxtaposition</td>
</tr>
<tr>
<td>±nK</td>
<td>kill lines</td>
</tr>
<tr>
<td>±nL</td>
<td>move down/up lines</td>
</tr>
<tr>
<td>nM</td>
<td>macro definition</td>
</tr>
<tr>
<td>nN</td>
<td>find next occurrence with autoscan</td>
</tr>
<tr>
<td>O</td>
<td>return to original file</td>
</tr>
<tr>
<td>±nP</td>
<td>move and print pages</td>
</tr>
<tr>
<td>Q</td>
<td>quit with no file changes</td>
</tr>
<tr>
<td>R</td>
<td>read library file</td>
</tr>
<tr>
<td>nS</td>
<td>substitute strings</td>
</tr>
<tr>
<td>±nT</td>
<td>type lines</td>
</tr>
<tr>
<td>± U</td>
<td>translate lower to upper case if U, no translation if -U</td>
</tr>
<tr>
<td>nW</td>
<td>write lines</td>
</tr>
<tr>
<td>nZ</td>
<td>sleep</td>
</tr>
<tr>
<td>±n&lt;cr&gt;</td>
<td>move and type (±nLT)</td>
</tr>
</tbody>
</table>
Appendix A: ED 1.4 Enhancements

The ED context editor contains a number of commands which enhance its usefulness in text editing. The improvements are found in the addition of line numbers, free space interrogation, and improved error reporting.

The context editor issued with CP/M 1.4 produces absolute line number prefixes when the "V" (Verify Line Numbers) command is issued. Following the V command, the line number is displayed ahead of each line in the format:

```
nnnnn:
```

where nnnnn is an absolute line number in the range 1 to 65535. If the memory buffer is empty, or if the current line is at the end of the memory buffer, then nnnnn appears as 5 blanks.

The user may reference an absolute line number by preceding any command by a number followed by a colon, in the same format as the line number display. In this case, the ED program moves the current line reference to the absolute line number, if the line exists in the current memory buffer. Thus, the command

```
345:T
```

is interpreted as "move to absolute line 345, and type the line." Note that absolute line numbers are produced only during the editing process, and are not recorded with the file. In particular, the line numbers will change following a deleted or expanded section of text.

The user may also reference an absolute line number as a backward or forward distance from the current line by preceding the absolute line number by a colon. Thus, the command

```
:400:T
```

is interpreted as "type from the current line number through the line whose absolute number is 400." Combining the two line reference forms, the command

```
345::400:T
```

for example, is interpreted as "move to absolute line 345, then type through absolute line 400." Note that absolute line references of this sort can precede any of the standard ED commands.

A special case of the V command, "$V", prints the memory buffer statistics in the form:

```
free/total
```

where "free" is the number of free bytes in the memory buffer (in decimal), and "total" is the size of the memory buffer.
ED 1.4 also includes a "block move" facility implemented through the "X" (Xfer) command. The form

\[ nX \]

transfers the next \( n \) lines from the current line to a temporary file called

\[ X\ldots \text{.LIB} \]

which is active only during the editing process. In general, the user can reposition the current line reference to any portion of the source file and transfer lines to the temporary file. The transferred line accumulate one after another in this file, and can be retrieved by simply typing:

\[ R \]

which is the trivial case of the library read command. In this case, the entire transferred set of lines is read into the memory buffer. Note that the \( X \) command does not remove the transferred lines from the memory buffer, although a \( K \) command can be used directly after the \( X \), and the \( R \) command does not empty the transferred line file. That is, given that a set of lines has been transferred with the \( X \) command, they can be re-read any number of times back into the source file. The command

\[ \#X \]

is provided, however, to empty the transferred line file.

Note that upon normal completion of the ED program through Q or E, the temporary LIB file is removed. If ED is aborted through ctl-C, the LIB file will exist if lines have been transferred, but will generally be empty (a subsequent ED invocation will erase the temporary file).

Due to common typographical errors, ED 1.4 requires several potentially disastrous commands to be typed as single letters, rather than in composite commands. The commands

\[ E \text{ (end)}, \quad H \text{ (head)}, \quad O \text{ (original)}, \quad Q \text{ (quit)} \]

must be typed as single letter commands.

ED 1.4 also prints error messages in the form

\[ \text{BREAK } "\text{x}" \text{ AT } c \]

where \( x \) is the error character, and \( c \) is the command where the error occurred.
CP/M DYNAMIC DEBUGGING TOOL (DDT): USER'S GUIDE
COPYRIGHT

Copyright (c) 1980 by Exidy Systems, Inc. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of Exidy Systems, Inc., 1234 Elko Drive, Sunnyvale, California 94086.

Since this manual is tutorial in nature, permission is granted to reproduce or abstract the example procedures and sample programs for the purposes of inclusion within the reader's programs.

DISCLAIMER

Exidy Systems, Inc. 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, Exidy Systems, Inc. reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Exidy Systems, Inc. to notify any person of such revision or changes.

TRADEMARKS

CP/M is a registered trademark of Digital Research.
CP/M DYNAMIC DEBUGGING TOOL (DDT): USER'S GUIDE
COPYRIGHT

Copyright (c) 1980 by Exidy Systems, Inc. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of Exidy Systems, Inc., 1234 Elko Drive, Sunnyvale, California 94086.

Since this manual is tutorial in nature, permission is granted to reproduce or abstract the example procedures and sample programs for the purposes of inclusion within the reader's programs.

DISCLAIMER

Exidy Systems, Inc. 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, Exidy Systems, Inc. reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Exidy Systems, Inc. to notify any person of such revision or changes.

TRADEMARKS

CP/M is a registered trademark of Digital Research.
Table Of Contents

CP/M Debugging Tool (DDT): User's Guide

I. Introduction ............................................. 111
II. DDT Commands ............................................ 113
   1. The A (Assemble) Command ......................... 113
   2. The D (Display) Command .......................... 114
   3. The F (Fill) Command .............................. 114
   4. The G (Go) Command .................................. 114
   5. The I (Input) Command .............................. 115
   6. The L (List) Command ............................... 116
   7. The M (Move) Command .............................. 116
   8. The R (Read) Command .............................. 116
   9. The S (Set) Command ................................ 117
  10. The T (Trace) Command ............................ 117
  11. The U (Untrace) Command ......................... 118
  12. The X (Examine) Command ......................... 118
III. Implementation Notes ................................. 119
IV. An Example .............................................. 120
CP/M Dynamic Debugging Tool (DDT)

User's Guide

I. Introduction.

The DDT program allows dynamic interactive testing and debugging of programs generated in the CP/M environment. The debugger is initiated by typing one of the following commands at the CP/M Console Command level:

DDT
DDT filename.HEX
DDT filename.COM

where "filename" is the name of the program to be loaded and tested. In both cases, the DDT program is brought into main memory in the place of the Console Command Processor (refer to the CP/M Interface Guide for standard memory organization), and thus resides directly below the Basic Disk Operating System portion of CP/M. The BDOS starting address, which is located in the address field of the JMP instruction at location 5H, is altered to reflect the reduced Transient Program Area size.

The second and third forms of the DDT command shown above perform the same actions as the first, except there is a subsequent automatic load of the specified HEX or COM file. The action is identical to the sequence of commands:

DDT
Ifilename.HEX or Ifilename.COM
R

where the I and R commands set up and read the specified program to test (see the explanation of the I and R commands below for exact details).

Upon initiation, DDT prints a sign-on message in the format:

nnK DDT-s VER m.m

where nn is the memory size (which must match the CP/M system being used), s is the hardware system which is assumed, corresponding to the codes:

D - Digital Research standard version
M - MDS version
I - IMSAI standard version
O - Omron systems
S - Digital Systems standard version

and m.m is the revision number.
Following the sign on message, DDT prompts the operator with the character "-" and waits for input commands from the console. The operator can type any of several single character commands, terminated by a carriage return to execute the command. Each line of input can be line-edited using the standard CP/M controls.

rubout remove the last character typed
ctl-U remove the entire line, ready for re-typing
ctl-C system reboot

Any command can be up to 32 characters in length (an automatic carriage return is inserted as the 33rd character), where the first character determines the command type:

A enter assembly language mnemonics with operands
D display memory in hexadecimal and ASCII
F fill memory with constant data
G begin execution with optional breakpoints
I set up a standard input file control block
L list memory using assembler mnemonics
M move a memory segment from source to destination
R read program for subsequent testing
S substitute memory values
T trace program execution
U untraced program monitoring
X examine and optionally alter the CPU state

The command character, in some cases, is followed by zero, one, two, or three hexadecimal values which are separated by commas or single blank characters. All DDT numeric output is in hexadecimal form. In all cases, the commands are not executed until the carriage return is typed at the end of the command.

At any point in the debug run, the operator can stop execution of DDT using either a ctl-C or G0 (jmp to location 0000H), and save the current memory image using a SAVE command of the form:

SAVE n filename.COM

where n is the number of pages (256 byte blocks) to be saved on disk. The number of blocks can be determined by taking the high order byte of the top load address and converting this number to decimal. For example, if the highest address in the Transient Program Area is 1234H then the number of pages is 12H, or 18 in decimal. Thus the operator could type a ctl-C during the debug run, returning to the Console Processor level, followed by:

SAVE 18 X.COM

The memory image is saved as X.COM on the diskette, and can be directly executed by simply typing the name X. If further testing is required, the memory image can be recalled by typing...
which reloads previously saved program from location 100H through page 18 (12FFH). The machine state is not a part of the COM file, and thus the program must be restarted from the beginning in order to properly test it.

II. DDT COMMANDS.

The individual commands are given below in some detail. In each case, the operator must wait for the prompt character (·) before entering the command. If control is passed to a program under test, and the program has not reached a breakpoint, control can be returned to DDT by executing a RST 7 from the front panel (note that the rubout key should be used instead if the program is executing a T or U command). In the explanation of each command, the command letter is shown in some cases with numbers separated by commas, where the numbers are represented by lower case letters. These numbers are always assumed to be in a hexadecimal radix, and from one to four digits in length (longer numbers will be automatically truncated on the right).

Many of the commands operate upon a "CPU state" which corresponds to the program under test. The CPU state holds the registers of the program being debugged, and initially contains zeroes for all registers and flags except for the program counter (P) and stack pointer (S), which default to 100H. The program counter is subsequently set to the starting address given in the last record of a HEX file if a file of this form is loaded (see the I and R commands).

1. The A (Assemble) Command. DDT allows inline assembly language to be inserted into the current memory image using the A command which takes the form

   As

where s is the hexadecimal starting address for the inline assembly. DDT prompts the console with the address of the next instruction to fill, and reads the console, looking for assembly language mnemonics (see the Intel 8080 Assembly Language Reference Card for a list of mnemonics), followed by register references and operands in absolute hexadecimal form. Each successive load address is printed before reading the console. The A command terminates when the first empty line is input from the console.

Upon completion of assembly language input, the operator can review the memory segment using the DDT disassembler (see the L command).

Note that the assembler/disassembler portion of DDT can be overlayed by the transient program being tested, in which case the DDT program responds with an error condition when the A and L commands are used (refer to Section IV).
2. The D (Display) Command. The D command allows the operator to view the contents of memory in hexadecimal and ASCII formats. The forms are

\[
\begin{align*}
D \\
Ds \\
Ds,f
\end{align*}
\]

In the first case, memory is displayed from the current display address (initially 100H), and continues for 16 display lines. Each display line takes the form shown below

\[
\begin{array}{cccccccccccccccc}
aaaa & bb & bb & bb & bb & bb & bb & bb & bb & bb & bb & bb & ccccccccccccccccc
\end{array}
\]

where aaaa is the display address in hexadecimal, and bb represents data present in memory starting at aaaa. The ASCII characters starting at aaaa are given to the right (represented by the sequence of c's), where non-graphic characters are printed as a period (.) symbol. Note that both upper and lower case alphabetics are displayed, and thus will appear as upper case symbols on a console device that supports only upper case. Each display line gives the values of 16 bytes of data, except that the first line displayed is truncated so that the next line begins at an address which is a multiple of 16.

The second form of the D command shown above is similar to the first, except that the display address is first set to address s. The third form causes the display to continue from address s through address f. In all cases, the display address is set to the first address not displayed in this command, so that a continuing display can be accomplished by issuing successive D commands with no explicit addresses.

Excessively long displays can be aborted by pushing the rubout key.

3. The F (Fill) Command. The F command takes the form

\[
Fs,f,c
\]

where s is the starting address, f is the final address, and c is a hexadecimal byte constant. The effect is as follows: DDT stores the constant c at address s, increments the value of s and tests against f. If s exceeds f then the operation terminates, otherwise the operation is repeated. Thus, the fill command can be used to set a memory block to a specific constant value.

4. The G (Go) Command. Program execution is started using the G command, with up to two optional breakpoint addresses. The G command takes one of the forms

\[
\begin{align*}
G \\
Gs \\
Gs,b
\end{align*}
\]
The first form starts execution of the program under test at the current value of the program counter in the current machine state, with no breakpoints set (the only way to regain control in DDT is through a RST 7 execution). The current program counter can be viewed by typing an X or XP command. The second form is similar to the first except that the program counter in the current machine state is set to address s before execution begins. The third form is the same as the second, except that program execution stops when address b is encountered (b must be in the area of the program under test). The instruction at location b is not executed when the breakpoint is encountered. The fourth form is identical to the third, except that two breakpoints are specified, one at b and the other at c. Encountering either breakpoint causes execution to stop, and both breakpoints are subsequently cleared. The last two forms take the program counter from the current machine state, and set one and two breakpoints, respectively.

Execution continues from the starting address in real-time to the next breakpoint. That is, there is no intervention between the starting address and the break address by DDT. Thus, if the program under test does not reach a breakpoint, control cannot return to DDT without executing a RST 7 instruction. Upon encountering a breakpoint, DDT stops execution and types

*d

where d is the stop address. The machine state can be examined at this point using the X (Examine) command. The operator must specify breakpoints which differ from the program counter address at the beginning of the G command. Thus, if the current program counter is 1234H, then the commands

G,1234

and

G400,400

both produce an immediate breakpoint, without executing any instructions whatsoever.

5. The I (Input) Command. The I command allows the operator to insert a file name into the default file control block at 5CH (the file control block created by CP/M for transient programs is placed at this location; see the CP/M Interface Guide). The default FCB can be used by the program under test as if it had been passed by the CP/M Console Processor. Note that this file name is also used by DDT for reading additional HEX and COM files. The form of the I command is

Ifilename

or
If the second form is used, and the filetype is either HEX or COM, then subsequent R commands can be used to read the pure binary or hex format machine code (see the R command for further details).

6. The L (List) Command. The L command is used to list assembly language mnemonics in a particular program region. The forms are

```
L
Ls
Ls,f
```

The first command lists twelve lines of disassembled machine code from the current list address. The second form sets the list address to s, and then lists twelve lines of code. The last form lists disassembled code from s through address f. In all three cases, the list address is set to the next unlisted location in preparation for a subsequent L command. Upon encountering an execution breakpoint, the list address is set to the current value of the program counter (see the G and T commands). Again, long typeouts can be aborted using the rubout key during the list process.

7. The M (Move) Command. The M command allows block movement of program or data areas from one location to another in memory. The form is

```
Ms,f,d
```

where s is the start address of the move, f is the final address of the move, and d is the destination address. Data is first moved from s to d, and both addresses are incremented. If s exceeds f then the move operation stops, otherwise the move operation is repeated.

8. The R (Read) Command. The R command is used in conjunction with the I command to read COM and HEX files from the diskette into the transient program area in preparation for the debug run. The forms are

```
R
Rb
```

where b is an optional bias address which is added to each program or data address as it is loaded. The load operation must not overwrite any of the system parameters from 0000H through 0FFH (i.e., the first page of memory). If b is omitted, then b=0000 is assumed. The R command requires a previous I command, specifying the name of a HEX or COM file. The load address for each record is obtained from each individual HEX record, while an assumed load address of 100H is taken for COM files. Note that any number of R commands can be issued following the I command to re-read the program under test,
assuming the tested program does not destroy the default area at 5CH. Further, any file specified with the filetype "COM" is assumed to contain machine code in pure binary form (created with the LOAD or SAVE command), and all others are assumed to contain machine code in Intel hex format (produced, for example, with the ASM command).

Recall that the command

```
DDT filename.filetype
```

which initiates the DDT program is equivalent to the commands

```
DDT
-Ifilename.filetype
-R
```

Whenever the R command is issued, DDT responds with either the error indicator "?" (file cannot be opened, or a checksum error occurred in a HEX file), or with a load message taking the form

```
NEXT PC
nnnn pppp
```

where nnnn is the next address following the loaded program, and pppp is the assumed program counter (100H for COM files, or taken from the last record if a HEX file is specified).

9. The S (Set) Command. The S command allows memory locations to be examined and optionally altered. The form of the command is

```
Ss
```

where s is the hexadecimal starting address for examination and alteration of memory. DDT responds with a numeric prompt, giving the memory location, along with the data currently held in the memory location. If the operator types a carriage return, then the data is not altered. If a byte value is typed, then the value is stored at the prompted address. In either case, DDT continues to prompt with successive addresses and values until either a period (.) is typed by the operator, or an invalid input value is detected.

10. The T (Trace) Command. The T command allows selective tracing of program execution for 1 to 65535 program steps. The forms are

```
T
Tn
```

In the first case, the CPU state is displayed, and the next program step is executed. The program terminates immediately, with the termination address
displayed as

*hhhh

where hhhh is the next address to execute. The display address (used in the D command) is set to the value of H and L, and the list address (used in the L command) is set to hhhh. The CPU state at program termination can then be examined using the X command.

The second form of the T command is similar to the first, except that execution is traced for n steps (n is a hexadecimal value) before a program breakpoint is occurs. A breakpoint can be forced in the trace mode by typing a rubout character. The CPU state is displayed before each program step is taken in trace mode. The format of the display is the same as described in the X command.

Note that program tracing is discontinued at the interface to CP/M, and resumes after return from CP/M to the program under test. Thus, CP/M functions which access I/O devices, such as the diskette drive, run in real-time, avoiding I/O timing problems. Programs running in trace mode execute approximately 500 times slower than real time since DDT gets control after each user instruction is executed. Interrupt processing routines can be traced, but it must be noted that commands which use the breakpoint facility (G, T, and U) accomplish the break using a RST 7 instruction, which means that the tested program cannot use this interrupt location. Further, the trace mode always runs the tested program with interrupts enabled, which may cause problems if asynchronous interrupts are received during tracing.

Note also that the operator should use the rubout key to get control back to DDT during trace, rather than executing a RST 7, in order to ensure that the trace for the current instruction is completed before interruption.

11. The U (Untrace) Command. The U command is identical to the T command except that intermediate program steps are not displayed. The untrace mode allows from 1 to 65535 (FFFFH) steps to be executed in monitored mode, and is used principally to retain control of an executing program while it reaches steady state conditions. All conditions of the T command apply to the U command.

12. The X (Examine) Command. The X command allows selective display and alteration of the current CPU state for the program under test. The forms are

X
Xr

where r is one of the 8080 CPU registers

C Carry Flag (0/1)
Z Zero Flag (0/1)
In the first case, the CPU register state is displayed in the format

\[
\text{CfZfMfEfIf A=bb B=dddd D=dddd H=dddd S=dddd P=dddd inst}
\]

where \( f \) is a 0 or 1 flag value, \( bb \) is a byte value, and \( dddd \) is a double byte quantity corresponding to the register pair. The "inst" field contains the disassembled instruction which occurs at the location addressed by the CPU state's program counter.

The second form allows display and optional alteration of register values, where \( r \) is one of the registers given above (C, Z, M, E, I, A, B, D, H, S, or P). In each case, the flag or register value is first displayed at the console. The DDT program then accepts input from the console. If a carriage return is typed, then the flag or register value is not altered. If a value in the proper range is typed, then the flag or register value is altered. Note that BC, DE, and HL are displayed as register pairs. Thus, the operator types the entire register pair when B, C, or the BC pair is altered.

III. IMPLEMENTATION NOTES.

The organization of DDT allows certain non-essential portions to be overlayed in order to gain a larger transient program area for debugging large programs. The DDT program consists of two parts: the DDT nucleus and the assembler/disassembler module. The DDT nucleus is loaded over the Console Command Processor, and, although loaded with the DDT nucleus, the assembler/disassembler is overlayable unless used to assemble or disassemble.

In particular, the BDOS address at location 6H (address field of the JMP instruction at location 5H) is modified by DDT to address the base location of the DDT nucleus which, in turn, contains a JMP instruction to the BDOS. Thus, programs which use this address field to size memory see the logical end of memory at the base of the DDT nucleus rather than the base of the BDOS.

The assembler/disassembler module resides directly below the DDT nucleus in the transient program area. If the A, L, T, or X commands are used during the debugging process then the DDT program again alters the address field at 6H to include this module, thus further reducing the logical end of memory. If a program loads beyond the beginning of the assembler/disassembler module, the A and L commands are lost (their use produces a "?" in response), and the
trace and display (T and X) commands list the "inst" field of the display in hexadecimal, rather than as a decoded instruction.

IV. AN EXAMPLE.

The following example shows an edit, assemble, and debug for a simple program which reads a set of data values and determines the largest value in the set. The largest value is taken from the vector, and stored into "LARGE" at the termination of the program.

```
ED SCAN.ASM

*END

ORG 100H  ;START OF TRANSIENT AREA
MVI B,LEN ;LENGTH OF VECTOR TO SCAN
MVI C,0  ;LARGEST VALUE SO FAR
LOOP P.O.D.L
LXI H,VECT ;BASE OF VECTOR
MOV A,M ;GET VALUE
SUB C ;LARGER VALUE IN C?
JNC NFOUND ;JUMP IF LARGER VALUE NOT FOUND
NEW LARGEST VALUE, STORE IT TO C
MOY C,A
NFOUND: INX H ;TO NEXT ELEMENT
DCR B ;MORE TO SCAN?
JNZ LOOP ;FOR ANOTHER?
END OF SCAN, STORE C
MOY A,C ;GET LARGEST VALUE
STA LARGE ;REBOOT
JMP B ;FOR ANOTHER?

TEST DATA

VECT:  DB 2,0,4,3,5,6,1,5
LEN EQU $-VECT ;LENGTH
LARGE: DS 1 ;LARGEST VALUE ON EXIT
END

BOP

ORG 100H ;START OF TRANSIENT AREA
MVI B,LEN ;LENGTH OF VECTOR TO SCAN
MVI C,0 ;LARGEST VALUE SO FAR
LXI H,VECT ;BASE OF VECTOR
LOOP: MOY A,M ;GET VALUE
SUB C ;LARGER VALUE IN C?
JNC NFOUND ;JUMP IF LARGER VALUE NOT FOUND
NEW LARGEST VALUE, STORE IT TO C
MOY C,A
NFOUND: INX H ;TO NEXT ELEMENT
DCR B ;MORE TO SCAN?
JNZ LOOP ;FOR ANOTHER
```

Create Source Program - underlined characters typed by programmer. ";" represents carriage return.
END OF SCAN, STORE C
MOY A, C ; GET LARGEST VALUE
STA LARGE
JMP 0 ; REBOOT

TEST DATA
VECT: DB 2, 0, 4, 3, 5, 6, 1, 5
LEN EQU $-VECT ; LENGTH
LARGE: DS 1 ; LARGEST VALUE ON EXIT
END

ASM SCAN, Start Assembler
CP/M ASSEMBLER - VER 1.0

0122
002H USE FACTOR
END OF ASSEMBLY

Assembly Complete - Look at Program Listing

Type SCAN.PRN

Code Address Machine Code (Source Program
0100 ORG 100H ; START OF TRANSIENT AREA
0100 0600 MYI B, LEN ; LENGTH OF VECTOR TO SCAN
0102 0E00 MYI C, 0 ; LARGEST VALUE SO FAR
0104 211901 LXI H, VECT ; BASE OF VECTOR
0107 7E LOOP: MOY A, M ; GET VALUE
0108 91 SUB C ; LARGER VALUE IN C?
0109 D20D01 JNC NFOUND ; JUMP IF LARGER VALUE NOT FOUND
010C 4F ; NEW LARGEST VALUE, STORE IT TO C
010D 23 MOY C, A
010E 05 NFOUND: INX H ; TO NEXT ELEMENT
010F C20701 DCR B ; MORE TO SCAN?
0112 79 JNZ LOOP ; FOR ANOTHER
0113 322101 END OF SCAN, STORE C
0116 C30000 MOY A, C ; GET LARGEST VALUE
0119 02004030 VECT: DB 2, 0, 4, 3, 5, 6, 1, 5
0008 = LEN EQU $-VECT ; LENGTH
0121 Value of LARGE: DS 1 ; LARGEST VALUE ON EXIT
0122 A> END
START Debugger using hex format machine code

16K DDT VER 1.0

Next PC 0121 0000

- [X] Last load address + 1

Next instruction to execute at PC = 0

COZMBE010 A=00 B=0000 D=0000 H=0000 S=0100 P=0000 OUT

-KP

Examines registers before debug run

P=0000 100

Change PC to 100

- [X], Look at registers again

PC changed.

COZMBE010 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI B, 08

-L100

Disassembled Machine

Next instruction to execute at PC = 100

A little more machine code

-NOTE that Program ends at location 116

-with a jmp to 0000

-enter inline assembly mode to change the jmp to 0000 into a RST 7, which

-will cause the program under test to return to DDT if 116H

-is ever executed.

(Single carriage return stops assemble mode)

-List code at 113H to check that RST 7 was properly inserted

0113 STA 0121

0116 RST 7

In place of JMP
- Look at registers

- Execute Program for one step. initial CPU state, before j is executed

- Trace one step again (note CPU in B)

- Trace again (Register C is cleared)

- Trace three steps

- Display memory starting at 113h.

- Current CPU state

- Trace 5 steps from current CPU state

- Trace without listing intermediate states

- CPU state at end of US,
Run Program from current PC until completion (in real-time)

Breakpoint at 1161 caused by executing RST 7 in machine code

CPU state at end of Program

COZIMBE111 A=00 B=0000 D=0000 H=0121 S=0100 P=0116 RST 07

Examine and change Program Counter
P=0116 100;

COZIMBE111 A=00 B=0000 D=0000 H=0121 S=0100 P=0100, MVI B, 08

Trace 10 (Hexadecimal) steps

COZIMBE111 A=00 B=0000 D=0000 H=0121 S=0100 P=0100 MVI B, 08
COZIMBE111 A=00 B=0000 D=0000 H=0121 S=0100 P=0102 MVI C, 00
COZIMBE111 A=00 B=0000 D=0000 H=0121 S=0100 P=0104 LXI H, 0119
COZIMBE111 A=00 B=0000 D=0000 H=0119 S=0100 P=0107 MOV A, M
COZIMBE011 A=02 B=0000 D=0000 H=0119 S=0100 P=0108 SUB C
COZIMBE011 A=02 B=0000 D=0000 H=0119 S=0100 P=0109 JNC 010D
COZIMBE011 A=02 B=0000 D=0000 H=0119 S=0100 P=010D INX H
COZIMBE011 A=02 B=0000 D=0000 H=011A S=0100 P=010E DCR B
COZIMBE011 A=02 B=0000 D=0000 H=011A S=0100 P=010F JNZ 0107
COZIMBE011 A=02 B=0000 D=0000 H=011A S=0100 P=0107 MOV A, M
COZIMBE011 A=00 B=0700 D=0000 H=011A S=0100 P=0108 SUB C
COZIMBE111 A=00 B=0700 D=0000 H=011A S=0100 P=0109 JNC 010D
COZIMBE111 A=00 B=0700 D=0000 H=011A S=0100 P=010D INX H
COZIMBE111 A=00 B=0700 D=0000 H=011A S=0100 P=010E DCR B
COZIMBE011 A=00 B=0680 D=0000 H=011B S=0100 P=0109 JNC 0107
COZIMBE011 A=00 B=0680 D=0000 H=011B S=0100 P=0107 MOV A, M*8108

Insert a "hot patch" into the machine code to change the value from A into C since A>C.

Program should have moved the value from A into C since A>C.

Since this code was not executed, it appears that the JNC should have been a JC instruction

Program resides on first page, so save 1 page.

Restart DDT with the saved memory image to continue testing

SAVE 1 SCAN.COM

A> DDT SCAN.COM

16K DDT VER 1.0
NEXT PC
0200 0100

List some code

0100 MVI B, 08
0102 MVI C, 00
0104 LXI H, 0119
0107 MOY A, M
0108 SUB C
0109 JC 010D

Previous Patch is present in X.COM
Trace to see how patched version operates  Data is moved from A to C

```
C0Z0M8E010 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI B, 08
C0Z0M8E010 A=00 B=0000 D=0000 H=0000 S=0100 P=0102 MVI C, 00
C0Z0M8E010 A=00 B=0000 D=0000 H=0000 S=0100 P=0104 LXI H, 0119
C0Z0M8E010 A=00 B=0000 D=0000 H=0119 S=0100 P=0107 MOY A, M
C0Z0M8E011 A=02 B=0000 D=0000 H=0119 S=0100 P=0108 SUB C
C0Z0M8E011 A=02 B=0000 D=0000 H=0119 S=0100 P=0109 JC 010D
C0Z0M8E011 A=02 B=0782 D=0000 H=011A S=0100 P=010F JNZ 0107
C0Z0M8E011 A=02 B=0782 D=0000 H=011A S=0100 P=0107 MOY A, M
C0Z0M8E011 A=00 B=0782 D=0000 H=011A S=0100 P=0108 SUB C
C1Z0M1E010 A=FE B=0782 D=0000 H=011A S=0100 P=0109 JC 010D
C1Z0M1E010 A=FE B=0782 D=0000 H=011A S=0100 P=010D INX H
C1Z0M1E010 A=FE B=0782 D=0000 H=011B S=0100 P=010E DCR B
C1Z0M8E111 A=FE B=0602 D=0000 H=011B S=0100 P=010F JNZ 0107*0107
C1Z0M8E111 A=FE B=0602 D=0000 H=011B S=0100 P=0107 MOY A, M
```

Run from current PC and breakpoint at 108H

```
*0108
C1Z0M8E111 A=04 B=0602 D=0000 H=011B S=0100 P=0108 SUB C
```

Single step for a few cycles
```
C1Z0M8E111 A=04 B=0602 D=0000 H=011B S=0100 P=0108 SUB C*0109
```

Run to completion
```
*0116
C0Z0M8E011 A=02 B=0602 D=0000 H=011B S=0100 P=0109 JC 010D*010C
C0Z0M8E011 A=02 B=0602 D=0000 H=011B S=0100 P=010C MOY C, A
```

```
C0Z0M8E111 A=03 B=0003 D=0000 H=0121 S=0100 P=0116 RST 07
```

Look at the value of "LAEGE"

0121 03, Wrong Value!
0122 00,
0123 22,
0124 21,
0125 00,
0126 02. End of the S Command
0127 7E, \[L100.\]

\[\begin{align*}
0100 & \text{ MVI } B, 08 \\
0102 & \text{ MVI } C, 00 \\
0104 & \text{ LXI } H, 0119 \\
0107 & \text{ MOV } A, M \\
0108 & \text{ SUB } C \\
0109 & \text{ JC } 010D \\
010C & \text{ MOV } C, A \\
010D & \text{ INX } H \\
010E & \text{ DCR } B \\
010F & \text{ JNZ } 0107 \\
0112 & \text{ MOV } A, C \\
0113 & \text{ STA } 0121 \\
0116 & \text{ RST } 07 \\
0117 & \text{ NOP } \\
0118 & \text{ NOP } \\
0119 & \text{ STAX } B \\
011A & \text{ NOP } \\
011B & \text{ INR } B \\
011C & \text{ INX } B \\
011D & \text{ DCR } B \\
011E & \text{ MVI } B, 01 \\
0120 & \text{ DCR } B \\
\end{align*}\]

\[\text{ \text{Review the code}}\]

\[\begin{align*}
P & = 0116 \text{ 100, Reset the PC} \\
-I_1 & \text{ Single step, and watch data values} \\
-C0Z1M0E111 \ A=03 \ B=0003 \ D=0000 \ H=0121 \ S=0100 \ P=0100 \ \text{MVI } B, 08*0102 \\
-I_2 & \text{ Count set} \ \text{labeled set} \\
-C0Z1M0E111 \ A=03 \ B=0003 \ D=0000 \ H=0121 \ S=0100 \ P=0102 \ \text{MVI } C, 00*0104 \\
-I_3 & \text{ base address of data set} \\
-C0Z1M0E111 \ A=03 \ B=0000 \ D=0000 \ H=0119*0107 \ S=0100 \ P=0104 \ \text{LXI } H, \text{ base address of data set} \\
-C0Z1M0E111 \ A=03 \ B=0000 \ D=0000 \ H=0119 \ S=0100 \ P=0107 \ \text{MOV } A, M*0108 \\
\end{align*}\]
-I₂
- first data item brought to A
C02110B111 A=02 B=0000 D=0000 H=0119 S=0100 P=0108 SUB C=0109

-2
C02110B011 A=02 B=0000 D=0000 H=0119 S=0100 P=0109 JC 010D+010C

-2
C02110B011 A=02 B=0000 D=0000 H=0119 S=0100 P=010C MOV C,A+010D

-2
C02110B011 A=02 B=0002 D=0000 H=0119 S=0100 P=010D INX H=010E

-2
C02110B011 A=02 B=0002 D=0000 H=0119 S=0100 P=010E DCR B=010F

-2
C02110B011 A=02 B=0702 D=0000 H=011A S=0100 P=010F JNZ 0107+0107

-2
C02110B011 A=02 B=0702 D=0000 H=011A S=0100 P=0107 MOV A,H+0108

-2
second data item brought to A
C02110B011 A=00 B=0702 D=0000 H=011A S=0100 P=0108 SUB C=0109

-2
subtract destroys data value which was loaded!!!
C1210B1010 A=FE B=0702 D=0000 H=011A S=0100 P=0109 JC 010D+010D

-2
C1210B1010 A=FE B=0702 D=0000 H=011A S=0100 P=010D INX H=010E

-100
0100 MVI B,08
0102 MVI C,00
0104 LXI H,0119
0107 MOV A,M
0108 SUB C
0109 JC 010D
010C MOV C,A
010D INX H
010E DCR B
010F JNZ 0107
0112 MOV A,C

This should have been a CMP so that register A
would not be destroyed.

-108
but patch at 1081 changes SUB to CMP

0108 CMP C

0109
-50, stop for SAVE
SAVE 1 SCAN.COM
Save memory image
ADD SCAN.COM Restart DDT
16K DDT VER 1.0
NEXT PC
0200 0100
-XP
P=0100
-L116
0116 RST 07
0117 NOP
0118 NOP
0119 STAX B
011A NOP
- (ubout)
-G.116 Run from 100H to completion
*0116
-XC Look at Carry (accidental type)
-C12
-X2 Look at CPU state
C12MBE111 A=06 B=0006 D=0000 H=0121 S=0100 P=0116 RST 07
-S1212 Look at "large" - it appears to be correct.
0121 06,
0122 00,
0123 22,
-G0 Stop DDT
ED SCAN.ASM Re-edit the source program, and make both changes
*NSUB
*BLT  ;LARGER VALUE IN C?
*SUB Z CMP LT  ;LARGER VALUE IN C?
*SNC Z LT  ;LARGER VALUE IN C?
*JNC NFOUNT ;JUMP IF LARGER VALUE NOT FOUND
*JC NFOUNT ;JUMP IF LARGER VALUE NOT FOUND
ASM SCAN.AAZ - Re-assemble, selecting source from disk A
CP/M ASSEMBLER - VER 1.0
Hex to disk A
Print to Z (selects no print file)

0122
002H USE FACTOR
END OF ASSEMBLY

DDT SCAN.HEX - Re-run debugger to check changes

16K DDT VER 1.0
NEXT PC
0121 0000
-L116-
0116 JMP 0000
check to ensure end is still at 116H
0119 STAX B
011A NOP
011B INR B
-(subplot)

-G100.116- Go from beginning with breakpoint at end
+0116 breakpoint reached

-D121-
Look at "LARGE" - Correct value computed

0121 05 00 22 21 00 02 7E EB 77 13 23 EB 08 78 B1 ... !... ~.W.O.X.
0130 C2 27 01 C3 03 29 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

-(subplot) aborts long typeout
-G0- Stop DDT, debug session complete
CP/M 2.2 INTERFACE GUIDE
COPYRIGHT

Copyright (c) 1980 by Exidy Systems, Inc. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of Exidy Systems, Inc., 1234 Elko Drive, Sunnyvale, California 94086.

Since this manual is tutorial in nature, permission is granted to reproduce or abstract the example procedures and sample programs for the purposes of inclusion within the reader's programs.

DISCLAIMER

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

TRADEMARKS

CP/M is a registered trademark of Digital Research.
COPYRIGHT

Copyright (c) 1980 by Exidy Systems, Inc. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of Exidy Systems, Inc., 1234 Elko Drive, Sunnyvale, California 94086.

Since this manual is tutorial in nature, permission is granted to reproduce or abstract the example procedures and sample programs for the purposes of inclusion within the reader's programs.

DISCLAIMER

Exidy Systems, Inc. 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, Exidy Systems, Inc. reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Exidy Systems, Inc. to notify any person of such revision or changes.

TRADEMARKS

CP/M is a registered trademark of Digital Research.
Table Of Contents

CP/M 2.2 Interface Guide

1. Introduction.................................131
2. Operating System Call Conventions........133
3. A Sample File-To-File Copy Program........159
4. A Sample File Dump Utility...............164
5. A Sample Random Access Program..........167
6. System Function Summary...................176
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:

![Memory Organization Diagram]

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

(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 FDOS 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 $005H ; STANDARD CP/M ENTRY
CONIN EQU 1 ; CONSOLE INPUT FUNCTION

ORG $100H ; BASE OF TPA
NEXC: MVI C,CONIN ; READ NEXT CHARACTER
CALL BDOS ; RETURN CHARACTER IN <A>
CPI "*" ; END OF PROCESSING?
JNZ NEXC ; 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:

<table>
<thead>
<tr>
<th>ASM</th>
<th>Assembler Source</th>
<th>PLI</th>
<th>PL/I Source File</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRN</td>
<td>Printer Listing</td>
<td>REL</td>
<td>Relocatable Module</td>
</tr>
<tr>
<td>HEX</td>
<td>Hex Machine Code</td>
<td>TEX</td>
<td>TEX Formatter Source</td>
</tr>
<tr>
<td>BAS</td>
<td>Basic Source File</td>
<td>BAK</td>
<td>ED Source Backup</td>
</tr>
<tr>
<td>INT</td>
<td>Intermediate Code</td>
<td>SYM</td>
<td>SID Symbol File</td>
</tr>
<tr>
<td>COM</td>
<td>CCP Command File</td>
<td>$$$</td>
<td>Temporary File</td>
</tr>
</tbody>
</table>

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:

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

- **tl,t2,t3**: contain the file type in ASCII upper case, with high bit = 0
  - tl', t2', and t3' denote the bit of these positions,
  - tl' = 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,rl,r2**: optional random record number in the range 0-65535, with overflow to r2, r0,rl constitute a 16-bit value with low byte r0, and high byte rl

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 do ... 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 alphabetics 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.
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.

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.

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.

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

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

(All Information Contained Herein is Proprietary to Digital Research.)
**FUNCTION I0: 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 current 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.

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

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

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). PCB'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.

(All Information Contained Herein is Proprietary to Digital Research.)
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 sl 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.
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 \( \text{OFFH} \) (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.
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.

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

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).

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

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

The Set File Attributes function allows programmatic manipulation of permanent indicators attached to files. In particular, the R/O and System attributes (tl' 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 fl' 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: 2FH
  * 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).

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

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

```assembly
; 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 = sfcb 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 0el0 mvi c,16 ; half an fcb
```

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

(All Information Contained Herein is Proprietary to Digital Research.)
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 and 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 = reaf 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 = fcbdn equ fcb+0 ;disk name
005d = fcbfn equ fcb+1 ;file name
0065 = fcbft equ fcb+9 ;disk file type (3 characters)
0068 = fcbrl equ fcb+12 ;file's current reel number
006b = fcbrc equ fcb+15 ;file's record count (0 to 128)
007c = fcbrl equ fcb+32 ;current (next) record number (0
007d = fcbln equ fcb+33 ;fcb length
;
; set up stack
0100 210000  lxi h,0
0103 39  dad sp
;
0104 221502 ;entry stack pointer in hl from the ccp
0107 315702 ;set sp to local stack area (restored at finis)
010a cdc101 ;read and print successive buffers
010d efef ;call setup ;set up input file
010f c21b01 ;jnz openok ;skip if open is ok
;
; file not there, give error message and return
0112 11f301 ;lxi d,opnmsg
0115 c9c01 ;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 cld1el pop b ! pop d ! pop h; environment restored

(All Information Contained Herein is Proprietary to Digital Research.)
; ; print a character
0165 e5d5c5  push h1 push d1 push b; saved
0168 0e02    mvi  c, typef
016a 5f       mov  e, a
016b c0500    call  bdos
016e c1dlel   pop  bl pop  dl 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     an1  ffh  ; 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     adi  'a' - 10
018b cd6501   prin:  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 c0500    call  bdos
01a1 c9  ret

; gnb: ; get next byte
01a2 3a1302   lda  ibp
01a5 fe80     cpi  80h
01a7 c2b301   jnz  g0
; read another buffer

(All Information Contained Herein is Proprietary to Digital Research.)
; call diskr
ora a ;zero value if read ok
jz g0 ;for another byte
; end of data, return with carry set for eof
stc
ret

; g0: ;read the byte at buff+reg a
mov e,a ;ls byte of buffer index
mov d,0 ;double precision index to de
inr a ;index=index+1
sta ibp ;back to memory
; pointer is incremented
; save the current file address
lxi h,buff
dad d
; absolute character address is in hl
mov a,m
; byte is in the accumulator
ora a ;reset carry bit
ret

; setup: ;set up file
; open the file for input
xra a ;zero to accum
sta fcbcr ;clear current record
lxi d,fcb
mvi c,.r.eadf
call bdos
pop bl pop dl pop h
ret

; diskr: ;read disk file record
push hl push dl push b
lxi d,fcb
mvi c,readf
call bdos
pop bl pop dl pop h
ret

; fixed message area
'db 'file dump version 2.0$'
db cr,lf,'no input file present on disk$'

; variable area
; input buffer pointer
ds 2
;entry sp value from ccp
; stack area
ds 64 ;reserve 32 level stack

; 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.

(All Information Contained Herein is Proprietary to Digital Research.)
;***************************************************
;* 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 3lbc0  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 l11b0  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 l15c0  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

(All Information Contained Herein is Proprietary to Digital Research.)
8122 0e16   mvi   c,makef
8124 115c0   lxi   d,fcb
8127 cd050   call  bdos
812a 3c      inr   a       ;err 255 becomes zero
812b c370   jnz   ready

; cannot create file, directory full
812e 113a0   lxi   d,nospace
8131 cda0   call  print
8134 c3000   jmp  reboot ;back to ccp

;*******************************
;*                           *
;*   loop back to "ready" after each command   *
;*                           *
;*******************************
;
ready:
    file is ready for processing
;
8137 cde50   call  readcom ;read next command
813a 227d0   shld  ranrec ;store input record#
813d 217f0   lxi   h,ranovf
8140 3600   mvi   m,0     ;clear high byte if set
8142 fe51   cpi   'Q'     ;quit?
8144 c2f0   jnz  notq

; quit processing, close file
8147 0e10   mvi   c,closef
8149 115c0   lxi   d,fcb
814c cd050   call  bdos
814f 3c      inr   a       ;err 255 becomes 0
8150 cab90   jz    error   ;error message, retry
8153 c3000   jmp  reboot ;back to ccp

;*******************************
;*                           *
;* end of quit command, process write     *
;*                           *
;*******************************

notq:
    not the quit command, random write?
8156 fe57   cpi   'W'
8158 c2f0   jnz  notw

; this is a random write, fill buffer until cr
815b 114d0   lxi   d,datmsg
815e cda0   call  print ;data prompt
8161 0e7f   mvi   c,127  ;up to 127 characters
8163 21800   lxi   h,buff ;destination
   rloop: ;read next character to buff
8166 c5     push  b      ;save counter
8167 e5     push  h      ;next destination
8168 cdc20  call  getchr ;character to a
816b el     pop   h      ;restore counter

(All Information Contained Herein is Proprietary to Digital Research.)
pop b ;restore next to fill
cli cr ;end of line?
je erloop ;not end, store character

mov m,a
inin h ;next to fill
dcr c ;counter goes down
jnz rloop ;end of buffer?

erloop:
; end of read loop, store 00

mvi m,0
;write the record to selected record number

mvi c,writer
lxi d,fcb
call bdos
ora a ;error code zero?
jnz error ;message if not

;jnz error ;for another record

;j***********************
*jend of write command, process read*
*j
;j***********************

notw:
;not a write command, read record?
cpi 'R'
jnz error ;skip if not
;
;read random record

mvi c,readr
lxi d,fcb
call bdos
ora a ;return code 00?
jnz error
;
;read was successful, write to console
call crlf ;new line
mvi c,128 ;max 128 characters
lxi h,buff ;next to get

wloop:
mov a,m ;next character
inin h ;next to get
ani 7fh ;mask parity
jz ready ;for another command if 00
push b ;save counter
push h ;save next to get
cpi ' ' ;graphic?
cnc putchr ;skip output if not
pop h
pop b
dcr c ;count=counter-1
jnz wloop
jmp ready

(All Information Contained Herein is Proprietary to Digital Research.)
;***************************************************************
;* end of read command, all errors end-up here                *
;***************************************************************
error:
01b9 11590     lxi   d,errmsg
01bc cdda0    call   print
01bf c3770     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 cdec80     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

(All Information Contained Herein is Proprietary to Digital Research.)
readc: lda x d

; next command character
inx d ; to next command position
ora a ; cannot be end of command

; not zero, numeric?
sui '0':
cpi 10 ; carry if numeric

jnc endrd

; add-in next digit

dad h ; *2
mov c, l
mov b, h ; bc = value * 2


dad h ; *4

dad h ; *8

dad b ; *2 + *8 = *10

add l ; +digit

mov l, a

jnc readc ; for another char

inr readc ; for another char

jnc readc ; for another char

endrd:

; end of read, restore value in a

adi '0'; command
cpi 'a'; translate case?
rc

; lower case, mask lower case bits

ani 101$1111b
ret

;******************************************************
; * ;
; * string data area for console messages ;
; *
;******************************************************

badver:

nospace:

prompt:

datmsg:

errmsg:

promt:
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!

(All Information Contained Herein is Proprietary to Digital Research.)
6. SYSTEM FUNCTION SUMMARY.

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

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

(All Information Contained Herein is Proprietary to Digital Research.)
CP/M 2.2 ALTERATION GUIDE
COPYRIGHT

Copyright (c) 1980 by Exidy Systems, Inc. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of Exidy Systems, Inc., 1234 Elko Drive, Sunnyvale, California 94086.

Since this manual is tutorial in nature, permission is granted to reproduce or abstract the example procedures and sample programs for the purposes of inclusion within the reader's programs.

DISCLAIMER

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

TRADEMARKS

CP/M is a registered trademark of Digital Research.
CP/M 2.2 ALTERATION GUIDE
COPYRIGHT

Copyright (c) 1980 by Exidy Systems, Inc. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of Exidy Systems, Inc., 1234 Elko Drive, Sunnyvale, California 94086.

Since this manual is tutorial in nature, permission is granted to reproduce or abstract the example procedures and sample programs for the purposes of inclusion within the reader's programs.

DISCLAIMER

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

TRADEMARKS

CP/M is a registered trademark of Digital Research.
# Table Of Contents

**CP/M Alteration Guide**

1. Introduction ........................................... 177
2. First Level System Regenerative ...................... 178
3. Second Level System Generation ....................... 182
4. Sample Getsys And Putsys Programs .................. 186
5. Diskette Organization ................................. 188
6. The BIOS Entry Points ................................ 190
7. A Sample BIOS .......................................... 197
8. A Sample Cold Start Loader ............................ 198
9. Reserved Locations In Page Zero ....................... 199
10. Disk Parameter Tables ................................ 201
11. The DISKDEF Macro Library ........................... 206
12. Sector Blocking And Deblocking ...................... 210
    Appendix A ............................................. 212
    Appendix B ............................................. 215
    Appendix C ............................................. 226
    Appendix D ............................................. 232
    Appendix E ............................................. 235
    Appendix F ............................................. 237
    Appendix G ............................................. 242
1. INTRODUCTION

The standard CP/M system assumes operation on an Intel MDS-800 microcomputer development system, but is designed so that the user can alter a specific set of subroutines which define the hardware operating environment. In this way, the user can produce a diskette which operates with any IBM-3741 format compatible drive controller and other peripheral devices.

Although standard CP/M 2.0 is configured for single density floppy disks, field-alteration features allow adaptation to a wide variety of disk subsystems from single drive minidisks through high-capacity "hard disk" systems. In order to simplify the following adaptation process, we assume that CP/M 2.0 will first be configured for single density floppy disks where minimal editing and debugging tools are available. If an earlier version of CP/M is available, the customizing process is eased considerably. In this latter case, you may wish to briefly review the system generation process, and skip to later sections which discuss system alteration for non-standard disk systems.

In order to achieve device independence, CP/M is separated into three distinct modules:

- **BIOS** - basic I/O system which is environment dependent
- **BDOS** - basic disk operating system which is not dependent upon the hardware configuration
- **CCP** - the console command processor which uses the BDOS

Of these modules, only the BIOS is dependent upon the particular hardware. That is, the user can "patch" the distribution version of CP/M to provide a new BIOS which provides a customized interface between the remaining CP/M modules and the user's own hardware system. The purpose of this document is to provide a step-by-step procedure for patching your new BIOS into CP/M.

If CP/M is being tailored to your computer system for the first time, the new BIOS requires some relatively simple software development and testing. The standard BIOS is listed in Appendix B, and can be used as a model for the customized package. A skeletal version of the BIOS is given in Appendix C which can serve as the basis for a modified BIOS. In addition to the BIOS, the user must write a simple memory loader, called GETSYS, which brings the operating system into memory. In order to patch the new BIOS into CP/M, the user must write the reverse of GETSYS, called PUTSYS, which places an altered version of CP/M back onto the diskette. PUTSYS can be derived from GETSYS by changing the disk read commands into disk write commands. Sample skeletal GETSYS and PUTSYS programs are described in Section 3, and listed in Appendix D. In order to make the CP/M system work automatically, the user must also supply a cold start loader, similar to the one provided with CP/M (listed in Appendixes A and B). A skeletal form of a cold start loader is given in Appendix E which can serve as a model for your loader.

(All Information Contained Herein is Proprietary to Digital Research.)
2. FIRST LEVEL SYSTEM REGENERATION

The procedure to follow to patch the CP/M system is given below in several steps. Address references in each step are shown with a following "H" which denotes the hexadecimal radix, and are given for a 20K CP/M system. For larger CP/M systems, add a "bias" to each address which is shown with a "+b" following it, where b is equal to the memory size - 20K. Values for b in various standard memory sizes are

- 24K: b = 24K - 20K = 4K = 1000H
- 32K: b = 32K - 20K = 12K = 3000H
- 40K: b = 40K - 20K = 20K = 5000H
- 48K: b = 48K - 20K = 28K = 7000H
- 56K: b = 56K - 20K = 36K = 9000H
- 62K: b = 62K - 20K = 42K = A800H
- 64K: b = 64K - 20K = 44K = B000H

Note: The standard distribution version of CP/M is set for operation within a 20K memory system. Therefore, you must first bring up the 20K CP/M system, and then configure it for your actual memory size (see Second Level System Generation).

1. Review Section 4 and write a GETSYS program which reads the first two tracks of a diskette into memory. The data from the diskette must begin at location 3380H. Code GETSYS so that it starts at location 100H (base of the TPA), as shown in the first part of Appendix D.

2. Test the GETSYS program by reading a blank diskette into memory, and check to see that the data has been read properly, and that the diskette has not been altered in any way by the GETSYS program.

3. Run the GETSYS program using an initialized CP/M diskette to see if GETSYS loads CP/M starting at 3380H (the operating system actually starts 128 bytes later at 3400H).

4. Review Section 4 and write the PUTSYS program which writes memory starting at 3380H back onto the first two tracks of the diskette. The PUTSYS program should be located at 200H, as shown in the second part of Appendix D.

5. Test the PUTSYS program using a blank uninitialized diskette by writing a portion of memory to the first two tracks; clear memory and read it back using GETSYS. Test PUTSYS completely, since this program will be used to alter CP/M on disk.

6. Study Sections 5, 6, and 7, along with the distribution version of the BIOS given in Appendix B, and write a simple version which performs a similar function for the customized environment. Use the program given in Appendix C as a model. Call this new BIOS by the name CBIOS (customized BIOS). Implement only the primitive disk operations on a single drive, and simple console input/output functions in this phase.

(All Information Contained Herein is Proprietary to Digital Research.)
(7) Test CBIOS completely to ensure that it properly performs console character I/O and disk reads and writes. Be especially careful to ensure that no disk write operations occur accidentally during read operations, and check that the proper track and sectors are addressed on all reads and writes. Failure to make these checks may cause destruction of the initialized CP/M system after it is patched.

(8) Referring to Figure 1 in Section 5, note that the BIOS is placed between locations 4A00H and 4FFFH. Read the CP/M system using GETSYS and replace the BIOS segment by the new CBIOS developed in step (6) and tested in step (7). This replacement is done in the memory of the machine, and will be placed on the diskette in the next step.

(9) Use PUTSYS to place the patched memory image of CP/M onto the first two tracks of a blank diskette for testing.

(10) Use GETSYS to bring the copied memory image from the test diskette back into memory at 3380H, and check to ensure that it has loaded back properly (clear memory, if possible, before the load). Upon successful load, branch to the cold start code at location 4A00H. The cold start routine will initialize page zero, then jump to the CCP at location 3400H which will call the BDOS, which will call the CBIOS. The CBIOS will be asked by the CCP to read sixteen sectors on track 2, and if successful, CP/M will type "A>>", the system prompt.

When you make it this far, you are almost on the air. If you have trouble, use whatever debug facilities you have available to trace and breakpoint your CBIOS.

(11) Upon completion of step (10), CP/M has prompted the console for a command input. Test the disk write operation by typing

```
SAVE 1 X.COM
```

(recall that all commands must be followed by a carriage return).

CP/M should respond with another prompt (after several disk accesses):

```
A>
```

If it does not, debug your disk write functions and retry.

(12) Then test the directory command by typing

```
DIR
```

CP/M should respond with

```
A: X.COM
```

(13) Test the erase command by typing

```
ERA X.COM
```

(All Information Contained Herein is Proprietary to Digital Research.)
CP/M should respond with the A prompt. When you make it this far, you should have an operational system which will only require a bootstrap loader to function completely.

(14) Write a bootstrap loader which is similar to GETSYS, and place it on track 0, sector 1 using PUTSYS (again using the test diskette, not the distribution diskette). See Sections 5 and 8 for more information on the bootstrap operation.

(15) Retest the new test diskette with the bootstrap loader installed by executing steps (11), (12), and (13). Upon completion of these tests, type a control-C (control and C keys simultaneously). The system should then execute a "warm start" which reboots the system, and types the A prompt.

(16) At this point, you probably have a good version of your customized CP/M system on your test diskette. Use GETSYS to load CP/M from your test diskette. Remove the test diskette, place the distribution diskette (or a legal copy) into the drive, and use PUTSYS to replace the distribution version by your customized version. Do not make this replacement if you are unsure of your patch since this step destroys the system which was sent to you from Digital Research.

(17) Load your modified CP/M system and test it by typing

```bash
DIR
```

CP/M should respond with a list of files which are provided on the initialized diskette. One such file should be the memory image for the debugger, called DDT.COM.

NOTE: from now on, it is important that you always reboot the CP/M system (ctl-C is sufficient) when the diskette is removed and replaced by another diskette, unless the new diskette is to be read only.

(18) Load and test the debugger by typing

```bash
DDT
```

(see the document "CP/M Dynamic Debugging Tool (DDT)" for operating procedures. You should take the time to become familiar with DDT, it will be your best friend in later steps.

(19) Before making further CBIOS modifications, practice using the editor (see the ED user's guide), and assembler (see the ASM user's guide). Then recode and test the GETSYS', PUTSYS', and CBIOS programs using ED, ASM, and DDT. Code and test a COPY program which does a sector-to-sector copy from one diskette to another to obtain back-up copies of the original diskette (NOTE: read your CP/M Licensing Agreement; it specifies your legal responsibilities when copying the CP/M system). Place the copyright notice

Copyright (c), 1979
Digital Research

(All Information Contained Herein is Proprietary to Digital Research.)
on each copy which is made with your COPY program.

(20) Modify your CBIOS to include the extra functions for punches, readers, signon messages, and so-forth, and add the facilities for an additional disk drives, if desired. You can make these changes with the GETSYS and PUTSYS programs which you have developed, or you can refer to the following section, which outlines CP/M facilities which will aid you in the regeneration process.

You now have a good copy of the customized CP/M system. Note that although the CBIOS portion of CP/M which you have developed belongs to you, the modified version of CP/M which you have created can be copied for your use only (again, read your Licensing Agreement), and cannot be legally copied for anyone else's use.

It should be noted that your system remains file-compatible with all other CP/M systems, (assuming media compatibility, of course) which allows transfer of non-proprietary software between users of CP/M.

(All Information Contained Herein is Proprietary to Digital Research.)
3. SECOND LEVEL SYSTEM GENERATION

Now that you have the CP/M system running, you will want to configure CP/M for your memory size. In general, you will first get a memory image of CP/M with the "MOVCPM" program (system relocator) and place this memory image into a named disk file. The disk file can then be loaded, examined, patched, and replaced using the debugger, and system generation program. For further details on the operation of these programs, see the "Guide to CP/M Features and Facilities" manual.

Your CBIOs and BOOT can be modified using ED, and assembled using ASM, producing files called CBIOs.HEX and BOOT.HEX, which contain the machine code for CBIOs and BOOT in Intel hex format.

To get the memory image of CP/M into the TPA configured for the desired memory size, give the command:

MOVCPM xx *

where "xx" is the memory size in decimal K bytes (e.g., 32 for 32K). The response will be:

CONSTRUCTING xxK CP/M VERS 2.0
READY FOR "SYSGEN" OR "SAVE 34 CPMxx.COM"

At this point, an image of a CP/M in the TPA configured for the requested memory size. The memory image is at location 990H through 227FH. (i.e., The BOOT is at 990H, the CCP is at 980H, the BDOS starts at 1180H, and the BIOS is at 1F80H.) Note that the memory image has the standard MDS-800 BIOS and BDOS on it. It is now necessary to save the memory image in a file so that you can patch your CBIOs and CBROOT into it:

SAVE 34 CPMxx.COM

The memory image created by the "MOVCPM" program is offset by a negative bias so that it loads into the free area of the TPA, and thus does not interfere with the operation of CP/M in higher memory. This memory image can be subsequently loaded under DDT and examined or changed in preparation for a new generation of the system. DDT is loaded with the memory image by typing:

DDT CPMxx.COM

Load DDT, then read the CPM image

DDT should respond with

NEXT PC
2300 0100

(The DDT prompt)

You can then use the display and disassembly commands to examine

(All Information Contained Herein is Proprietary to Digital Research.)
portions of the memory image between 900H and 227FB. Note, however, that to find any particular address within the memory image, you must apply the negative bias to the CP/M address to find the actual address. Track 00, sector 01 is loaded to location 900H (you should find the cold start loader at 900H to 97FH), track 00, sector 02 is loaded into 980H (this is the base of the CCP), and so-forth through the entire CP/M system load. In a 20K system, for example, the CCP resides at the CP/M address 3400H, but is placed into memory at 980H by the SYSGEN program. Thus, the negative bias, denoted by n, satisfies

\[ 3400H + n = 980H, \text{ or } n = 980H - 3400H \]

Assuming two's complement arithmetic, \( n = D580H \), which can be checked by

\[ 3400H + D580H = 10980H = 0980H \] (ignoring high-order overflow).

Note that for larger systems, \( n \) satisfies

\[ (3400H+b) + n = 980H, \text{ or } n = 980H - (3400H + b), \text{ or } n = D580H - b. \]

The value of \( n \) for common CP/M systems is given below

<table>
<thead>
<tr>
<th>Memory Size</th>
<th>Bias ( b )</th>
<th>Negative Offset ( n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>20K</td>
<td>0000H</td>
<td>D580H - 0000H = D580H</td>
</tr>
<tr>
<td>24K</td>
<td>0100H</td>
<td>D580H - 1000H = C580H</td>
</tr>
<tr>
<td>32K</td>
<td>0300H</td>
<td>D580H - 3000H = A580H</td>
</tr>
<tr>
<td>40K</td>
<td>0500H</td>
<td>D580H - 5000H = 8580H</td>
</tr>
<tr>
<td>48K</td>
<td>0700H</td>
<td>D580H - 7000H = 6580H</td>
</tr>
<tr>
<td>56K</td>
<td>0900H</td>
<td>D580H - 9000H = 4580H</td>
</tr>
<tr>
<td>62K</td>
<td>0A00H</td>
<td>D580H - A000H = 2D80H</td>
</tr>
<tr>
<td>64K</td>
<td>0B00H</td>
<td>D580H - B000H = 2580H</td>
</tr>
</tbody>
</table>

Assume, for example, that you want to locate the address \( x \) within the memory image loaded under DDT in a 20K system. First type

\[ Hx,n \]

Hexadecimal sum and difference

and DDT will respond with the value of \( x+n \) (sum) and \( x-n \) (difference). The first number printed by DDT will be the actual memory address in the image where the data or code will be found. The input

\[ H3400,D580 \]

for example, will produce 980H as the sum, which is where the CCP is located in the memory image under DDT.

Use the L command to disassemble portions the BIOS located at \( (4A00H+d) - n \) which, when you use the H command, produces an actual address of 1F80H. The disassembly command would thus be

(All Information Contained Herein is Proprietary to Digital Research.)
It is now necessary to patch in your CBOOT and CBIOS routines. The
BOOT resides at location 9990H in the memory image. If the actual
load address is “n”, then to calculate the bias (m) use the command:

\[
H900,n \quad \text{Subtract load address from target address.}
\]

The second number typed in response to the command is the desired bias
(m). For example, if your BOOT executes at 9080H, the command:

\[
H900,80
\]

will reply

\[
9080 0880 \quad \text{Sum and difference in hex.}
\]

Therefore, the bias “m” would be 0880H. To read-in the BOOT, give the
command:

\[
ICBOOT.HEX \quad \text{Input file CBOOT.HEX}
\]

Then:

\[
Rm \quad \text{Read CBOOT with a bias of}
\]

\[
m (=9080H-n)
\]

You may now examine your CBOOT with:

\[
L900
\]

We are now ready to replace the CBIOS. Examine the area at 1F80H
where the original version of the CBIOS resides. Then type

\[
ICBIOS.HEX \quad \text{Ready the “hex” file for loading}
\]

assume that your CBIOS is being integrated into a 20K CP/M system, and
thus is origined at location 4A00H. In order to properly locate the
CBIOS in the memory image under DDT, we must apply the negative bias n
for a 20K system when loading the hex file. This is accomplished by
typing

\[
RD580 \quad \text{Read the file with bias D580H}
\]

Upon completion of the read, re-examine the area where the CBIOS has
been loaded (use an “L1F80” command), to ensure that is was loaded
properly. When you are satisfied that the change has been made,
return from DDT using a control-C or “GO” command.

Now use SYSGEN to replace the patched memory image back onto a
diskette (use a test diskette until you are sure of your patch), as
shown in the following interaction.

(All Information Contained Herein is Proprietary to Digital Research.)
SYSGEN
SYSGEN VERSION 2.0
SOURCE DRIVE NAME (OR RETURN TO SKIP)
Respond with a carriage return to skip the CP/M read operation since the system is already in memory.

DESTINATION DRIVE NAME (OR RETURN TO REBOOT)
Respond with "B" to write the new system to the diskette in drive B.

DESTINATION ON B, THEN TYPE RETURN
Place a scratch diskette in drive B, then type return.

FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)

Place the scratch diskette in your drive A, and then perform a coldstart to bring up the new CP/M system you have configured.

Test the new CP/M system, and place the Digital Research copyright notice on the diskette, as specified in your Licensing Agreement:

Copyright (c), 1979
Digital Research
4. SAMPLE GETSYS AND PUTSYS PROGRAMS

The following program provides a framework for the GETSYS and
PUTSYS programs referenced in Section 2. The READSEC and WRITESEC
subroutines must be inserted by the user to read and write the
specific sectors.

```
; GETSYS PROGRAM - READ TRACKS 0 AND 1 TO MEMORY AT 3380H
; REGISTER USE
;   A (SCRATCH REGISTER)
;   B TRACK COUNT (0, 1)
;   C SECTOR COUNT (1, 2, ..., 26)
;   DE (SCRATCH REGISTER PAIR)
;   HL LOAD ADDRESS
;   SP SET TO STACK ADDRESS

START: LXI SP, 3380H ; SET STACK POINTER TO SCRATCH AREA
    LXI H, 3380H ; SET BASE LOAD ADDRESS
    MVI B, 0 ; START WITH TRACK 0
    RDTRK:
        MVI C, 1 ; READ NEXT SECTOR STARTING WITH SECTOR 1
        CALL READSEC ; USER-SUPPLIED SUBROUTINE
        LXI D, 128 ; MOVE LOAD ADDRESS TO NEXT 1/2 PAGE
        DAD D ; HL = HL + 128
        INR C ; SECTOR = SECTOR + 1
        MOV A, C ; CHECK FOR END OF TRACK
        CPI 27
        JC RDSEC ; CARRY GENERATED IF SECTOR < 27

        INR B
        MOV A, B ; TEST FOR LAST TRACK
        CPI 2
        JC RDTRK ; CARRY GENERATED IF TRACK < 2

        ARRIVE HERE AT END OF LOAD, HALT FOR NOW
        HLT

        USER-SUPPLIED SUBROUTINE TO READ THE DISK

READSEC:
    ENTER WITH TRACK NUMBER IN REGISTER B,
    SECTOR NUMBER IN REGISTER C, AND
    ADDRESS TO FILL IN HL

    PUSH B ; SAVE B AND C REGISTERS
    PUSH H ; SAVE HL REGISTERS

    perform disk read at this point, branch to
    label START if an error occurs

    POP H ; RECOVER HL
    POP B ; RECOVER B AND C REGISTERS
    RET ; BACK TO MAIN PROGRAM

END START
```

(All Information Contained Herein is Proprietary to Digital Research.)
Note that this program is assembled and listed in Appendix C for reference purposes, with an assumed origin of 100H. The hexadecimal operation codes which are listed on the left may be useful if the program has to be entered through your machine's front panel switches.

The PUTSYS program can be constructed from GETSYS by changing only a few operations in the GETSYS program given above, as shown in Appendix D. The register pair HL become the dump address (next address to write), and operations upon these registers do not change within the program. The READSEC subroutine is replaced by a WRITESEC subroutine which performs the opposite function: data from address HL is written to the track given by register B and sector given by register C. It is often useful to combine GETSYS and PUTSYS into a single program during the test and development phase, as shown in the Appendix.

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

The sector allocation for the standard distribution version of CP/M is given here for reference purposes. The first sector (see table on the following page) contains an optional software boot section. Disk controllers are often set up to bring track 0, sector 1 into memory at a specific location (often location 0000H). The program in this sector, called BOOT, has the responsibility of bringing the remaining sectors into memory starting at location 3400H+1. If your controller does not have a built-in sector load, you can ignore the program in track 0, sector 1, and begin the load from track 0 sector 2 to location 3400H+1.

As an example, the Intel MDS-800 hardware cold start loader brings track 0, sector 1 into absolute address 3000H. Upon loading this sector, control transfers to location 3000H, where the bootstrap operation commences by loading the remainder of tracks 0, and all of track 1 into memory, starting at 3400H+1. The user should note that this bootstrap loader is of little use in a non-MOS environment, although it is useful to examine it since some of the boot actions will have to be duplicated in your cold start loader.

(All Information Contained Herein is Proprietary to Digital Research.)
<table>
<thead>
<tr>
<th>Tracky Sector</th>
<th>Pagey</th>
<th>Memory Address</th>
<th>CP/M Module Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>01</td>
<td>(boot address)</td>
<td>Cold Start Loader</td>
</tr>
<tr>
<td>00</td>
<td>02</td>
<td>3400H+b</td>
<td>CCP</td>
</tr>
<tr>
<td>02</td>
<td>03</td>
<td>3800H+b</td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>04</td>
<td>3500H+b</td>
<td></td>
</tr>
<tr>
<td>04</td>
<td>05</td>
<td>3580H+b</td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>06</td>
<td>3600H+b</td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>07</td>
<td>3680H+b</td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>08</td>
<td>3700H+b</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>09</td>
<td>3780H+b</td>
<td></td>
</tr>
<tr>
<td>09</td>
<td>10</td>
<td>3800H+b</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>3880H+b</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>3900H+b</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>3980H+b</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>3A00H+b</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>15</td>
<td>3A80H+b</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>3B00H+b</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>3B80H+b</td>
<td>CCP</td>
</tr>
<tr>
<td>17</td>
<td>18</td>
<td>3C00H+b</td>
<td>BDOS</td>
</tr>
<tr>
<td>18</td>
<td>19</td>
<td>3C80H+b</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>20</td>
<td>3D00H+b</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>21</td>
<td>3D80H+b</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>22</td>
<td>3E00H+b</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>23</td>
<td>3E80H+b</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>24</td>
<td>3F00H+b</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>25</td>
<td>3F80H+b</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>26</td>
<td>4000H+b</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>01</td>
<td>4080H+b</td>
<td>BDOS</td>
</tr>
<tr>
<td>01</td>
<td>02</td>
<td>4100H+b</td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>03</td>
<td>4180H+b</td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>04</td>
<td>4200H+b</td>
<td></td>
</tr>
<tr>
<td>04</td>
<td>05</td>
<td>4280H+b</td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>06</td>
<td>4300H+b</td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>07</td>
<td>4380H+b</td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>08</td>
<td>4400H+b</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>09</td>
<td>4480H+b</td>
<td></td>
</tr>
<tr>
<td>09</td>
<td>10</td>
<td>4500H+b</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>4580H+b</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>4600H+b</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>4680H+b</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>4700H+b</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>15</td>
<td>4780H+b</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>4800H+b</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>4880H+b</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>18</td>
<td>4900H+b</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>19</td>
<td>4980H+b</td>
<td>BDOS</td>
</tr>
<tr>
<td>19</td>
<td>20</td>
<td>4A00H+b</td>
<td>BIOS</td>
</tr>
<tr>
<td>20</td>
<td>21</td>
<td>4A80H+b</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>22</td>
<td>4B00H+b</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>23</td>
<td>4B80H+b</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>24</td>
<td>4C00H+b</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>25</td>
<td>4C80H+b</td>
<td>BIOS</td>
</tr>
<tr>
<td>25</td>
<td>26</td>
<td>4D00H+b</td>
<td></td>
</tr>
</tbody>
</table>

(directory and data)

(All Information Contained Herein is Proprietary to Digital Research.)
6. THE BIOS ENTRY POINTS

The entry points into the BIOS from the cold start loader and BDOS are detailed below. Entry to the BIOS is through a "jump vector" located at 4A00H+b, as shown below (see Appendices B and C, as well). The jump vector is a sequence of 17 jump instructions which send program control to the individual BIOS subroutines. The BIOS subroutines may be empty for certain functions (i.e., they may contain a single RET operation) during regeneration of CP/M, but the entries must be present in the jump vector.

The jump vector at 4A00H+b takes the form shown below, where the individual jump addresses are given to the left:

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Function Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4A00H+b</td>
<td>JMP BOOT</td>
<td>ARRIVE HERE FROM COLD START LOAD</td>
</tr>
<tr>
<td>4A03H+b</td>
<td>JMP WBOOT</td>
<td>ARRIVE HERE FOR WARM START</td>
</tr>
<tr>
<td>4A06H+b</td>
<td>JMP CONST</td>
<td>CHECK FOR CONSOLE CHAR READY</td>
</tr>
<tr>
<td>4A09H+b</td>
<td>JMP CONIN</td>
<td>READ CONSOLE CHARACTER IN</td>
</tr>
<tr>
<td>4A0CH+b</td>
<td>JMP CONOUT</td>
<td>WRITE CONSOLE CHARACTER OUT</td>
</tr>
<tr>
<td>4A0FH+b</td>
<td>JMP LIST</td>
<td>WRITE LISTING CHARACTER OUT</td>
</tr>
<tr>
<td>4A12H+b</td>
<td>JMP PUNCH</td>
<td>WRITE CHARACTER TO PUNCH DEVICE</td>
</tr>
<tr>
<td>4A15H+b</td>
<td>JMP READER</td>
<td>READ READER DEVICE</td>
</tr>
<tr>
<td>4A18H+b</td>
<td>JMP HOME</td>
<td>MOVE TO TRACK 00 ON SELECTED DISK</td>
</tr>
<tr>
<td>4A1BH+b</td>
<td>JMP SELDSK</td>
<td>SELECT DISK DRIVE</td>
</tr>
<tr>
<td>4A1EH+b</td>
<td>JMP SETTRK</td>
<td>SET TRACK NUMBER</td>
</tr>
<tr>
<td>4A21H+b</td>
<td>JMP SETSEC</td>
<td>SET SECTOR NUMBER</td>
</tr>
<tr>
<td>4A24H+b</td>
<td>JMP SETDMA</td>
<td>SET DMA ADDRESS</td>
</tr>
<tr>
<td>4A27H+b</td>
<td>JMP READ</td>
<td>READ SELECTED SECTOR</td>
</tr>
<tr>
<td>4A2AH+b</td>
<td>JMP WRITE</td>
<td>WRITE SELECTED SECTOR</td>
</tr>
<tr>
<td>4A2DH+b</td>
<td>JMP LISTST</td>
<td>RETURN LIST STATUS</td>
</tr>
<tr>
<td>4A30H+b</td>
<td>JMP SECTRAN</td>
<td>SECTOR TRANSLATE SUBROUTINE</td>
</tr>
</tbody>
</table>

Each jump address corresponds to a particular subroutine which performs the specific function, as outlined below. There are three major divisions in the jump table: the system (re)initialization which results from calls on BOOT and WBOOT, simple character I/O performed by calls on CONST, CONIN, CONOUT, LIST, PUNCH, READER, and LISTST, and diskette I/O performed by calls on HOME, SELDSK, SETTRK, SETSEC, SETDMA, READ, WRITE, and SECTRAN.

All simple character I/O operations are assumed to be performed in ASCII, upper and lower case, with high order (parity bit) set to zero. An end-of-file condition for an input device is given by an ASCII control-z (IAH). Peripheral devices are seen by CP/M as "logical" devices, and are assigned to physical devices within the BIOS.

In order to operate, the BDOS needs only the CONST, CONIN, and CONOUT subroutines (LIST, PUNCH, and READER may be used by PIP, but not the BDOS). Further, the LISTST entry is used currently only by DESPOOL, and thus, the initial version of CBIOS may have empty subroutines for the remaining ASCII devices.

(All Information Contained Herein is Proprietary to Digital Research.)
The characteristics of each device are:

**CONSOLE**

The principal interactive console which communicates with the operator, accessed through CONST, CONIN, and CONOUT. Typically, the CONSOLE is a device such as a CRT or Teletype.

**LIST**

The principal listing device, if it exists on your system, which is usually a hard-copy device, such as a printer or Teletype.

**PUNCH**

The principal tape punching device, if it exists, which is normally a high-speed paper tape punch or Teletype.

**READER**

The principal tape reading device, such as a simple optical reader or Teletype.

Note that a single peripheral can be assigned as the LIST, PUNCH, and READER device simultaneously. If no peripheral device is assigned as the LIST, PUNCH, or READER device, the CBIOS created by the user may give an appropriate error message so that the system does not "hang" if the device is accessed by PIP or some other user program. Alternately, the PUNCH and LIST routines can just simply return, and the READER routine can return with a LAH (ctl-Z) in reg A to indicate immediate end-of-file.

For added flexibility, the user can optionally implement the "IOBYTE" function which allows reassignment of physical and logical devices. The IOBYTE function creates a mapping of logical to physical devices which can be altered during CP/M processing (see the STAT command). The definition of the IOBYTE function corresponds to the Intel standard as follows:

A single location in memory (currently location 0003H) is maintained, called IOBYTE, which defines the logical to physical device mapping which is in effect at a particular time. The mapping is performed by splitting the IOBYTE into four distinct fields of two bits each, called the CONSOLE, READER, PUNCH, and LIST fields, as shown below:

<table>
<thead>
<tr>
<th>IOBYTE AT 0003H</th>
<th>LIST</th>
<th>PUNCH</th>
<th>READER</th>
<th>CONSOLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>bits 6,7</td>
<td>bits 4,5</td>
<td>bits 2,3</td>
<td>bits 0,1</td>
<td></td>
</tr>
</tbody>
</table>

The value in each field can be in the range 0-3, defining the assigned source or destination of each logical device. The values which can be assigned to each field are given below.

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

191
CONSOLE field (bits 0,1)
0 - console is assigned to the console printer device (TTY:)
1 - console is assigned to the CRT device (CRT:)
2 - batch mode: use the READER as the CONSOLE input,
and the LIST device as the CONSOLE output (BAT:)
3 - user defined console device (UC1:)

READER field (bits 2,3)
0 - READER is the Teletype device (TTY:)
1 - READER is the high-speed reader device (RDR:)
2 - user defined reader # 1 (UR1:)
3 - user defined reader # 2 (UR2:)

PUNCH field (bits 4,5)
0 - PUNCH is the Teletype device (TTY:)
1 - PUNCH is the high speed punch device (PUN:)
2 - user defined punch # 1 (UP1:)
3 - user defined punch # 2 (UP2:)

LIST field (bits 6,7)
0 - LIST is the Teletype device (TTY:)
1 - LIST is the CRT device (CRT:)
2 - LIST is the line printer device (LPT:)
3 - user defined list device (UL1:)

Note again that the implementation of the IOBYTE is optional, and affects only the organization of your CBIO S. No CP/M systems use the IOBYTE (although they tolerate the existence of the IOBYTE at location 0003H), except for PIP which allows access to the physical devices, and STAT which allows logical-physical assignments to be made and/or displayed (for more information, see the "CP/M Features and Facilities Guide"). In any case, the IOBYTE implementation should be omitted until your basic CBIO S is fully implemented and tested; then add the IOBYTE to increase your facilities.

Disk I/O is always performed through a sequence of calls on the various disk access subroutines which set up the disk number to access, the track and sector on a particular disk, and the direct memory access (DMA) address involved in the I/O operation. After all these parameters have been set up, a call is made to the READ or WRITE function to perform the actual I/O operation. Note that there is often a single call to SELDSK to select a disk drive, followed by a number of read or write operations to the selected disk before selecting another drive for subsequent operations. Similarly, there may be a single call to set the DMA address, followed by several calls which read or write from the selected DMA address before the DMA address is changed. The track and sector subroutines are always called before the READ or WRITE operations are performed.

(All Information Contained Herein is Proprietary to Digital Research.)
Note that the READ and WRITE routines should perform several retries (10 is standard) before reporting the error condition to the BDOS. If the error condition is returned to the BDOS, it will report the error to the user. The HOME subroutine may or may not actually perform the track 00 seek, depending upon your controller characteristics; the important point is that track 00 has been selected for the next operation, and is often treated in exactly the same manner as SETTRK with a parameter of 00.

The exact responsibilities of each entry point subroutine are given below:

**BOOT**

The BOOT entry point gets control from the cold start loader and is responsible for basic system initialization, including sending a signon message (which can be omitted in the first version). If the IOBYTE function is implemented, it must be set at this point. The various system parameters which are set by the WBOOT entry point must be initialized, and control is transferred to the CCP at 3400H+b for further processing. Note that reg C must be set to zero to select drive A.

**WBOOT**

The WBOOT entry point gets control when a warm start occurs. A warm start is performed whenever a user program branches to location 0000H, or when the CPU is reset from the front panel. The CP/M system must be loaded from the first two tracks of drive A up to, but not including, the BIOS (or CBIOS, if you have completed your patch). System parameters must be initialized as shown below:

- **location 0,1,2** set to JMP WBOOT for warm starts (0000H: JMP 4A03H+b)
- **location 3** set initial value of IOBYTE, if implemented in your CBIOS
- **location 5,6,7** set to JMP BDOS, which is the primary entry point to CP/M for transient programs. (0005H: JMP 3C06H+b)

(see Section 9 for complete details of page zero use)

Upon completion of the initialization, the WBOOT program must branch to the CCP at 3400H+b to (re)start the system. Upon entry to the CCP, register C is set to the drive to select after system initialization.

**CONST**

Sample the status of the currently assigned console device and return 0FFH in register A if a character is ready to read, and 00H in register A if no console characters are ready.

**CONIN**

Read the next console character into register A, and

(All Information Contained Herein is Proprietary to Digital Research.)
set the parity bit (high order bit) to zero. If no console character is ready, wait until a character is typed before returning.

**CONOUT**

Send the character from register C to the console output device. The character is in ASCII, with high order parity bit set to zero. You may want to include a time-out on a line feed or carriage return, if your console device requires some time interval at the end of the line (such as a TI Silent 700 terminal). You can, if you wish, filter out control characters which cause your console device to react in a strange way (a control-z causes the Lear Seigler terminal to clear the screen, for example).

**LIST**

Send the character from register C to the currently assigned listing device. The character is in ASCII with zero parity.

**PUNCH**

Send the character from register C to the currently assigned punch device. The character is in ASCII with zero parity.

**READER**

Read the next character from the currently assigned reader device into register A with zero parity (high order bit must be zero), an end of file condition is reported by returning an ASCII control-z (lAH).

**HOME**

Return the disk head of the currently selected disk (initially disk A) to the track 00 position. If your controller allows access to the track 0 flag from the drive, step the head until the track 0 flag is detected. If your controller does not support this feature, you can translate the HOME call into a call on SETTRK with a parameter of 0.

**SELDSDK**

Select the disk drive given by register C for further operations, where register C contains 0 for drive A, 1 for drive B, and so-forth up to 15 for drive P (the standard CP/M distribution version supports four drives). On each disk select, SELDSK must return in HL the base address of a 16-byte area, called the Disk Parameter Header, described in the Section 10. For standard floppy disk drives, the contents of the header and associated tables does not change, and thus the program segment included in the sample BIOS performs this operation automatically. If there is an attempt to select a non-existent drive, SELDSK returns HL=0000H as an error indicator. Although SELDSK must return the header address on each call, it is advisable to postpone the actual physical disk select operation until an I/O function (seek, read or write) is actually performed, since disk selects often occur without ultimately performing any disk I/O, and many controllers will unload the head of the current disk.
before selecting the new drive. This would cause an excessive amount of noise and disk wear.

SETTRK
Register BC contains the track number for subsequent disk accesses on the currently selected drive. You can choose to seek the selected track at this time, or delay the seek until the next read or write actually occurs. Register BC can take on values in the range 0-76 corresponding to valid track numbers for standard floppy disk drives, and 0-65535 for non-standard disk subsystems.

SETSEC
Register BC contains the sector number (1 through 26) for subsequent disk accesses on the currently selected drive. You can choose to send this information to the controller at this point, or instead delay sector selection until a read or write operation occurs.

SETDMA
Register BC contains the DMA (disk memory access) address for subsequent read or write operations. For example, if B = 00H and C = 80H when SETDMA is called, then all subsequent read operations read their data into 80H through 0FFH, and all subsequent write operations get their data from 80H through 0FFH, until the next call to SETDMA occurs. The initial DMA address is assumed to be 80H. Note that the controller need not actually support direct memory access. If, for example, all data is received and sent through I/O ports, the CB IOS which you construct will use the 128 byte area starting at the selected DMA address for the memory buffer during the following read or write operations.

READ
Assuming the drive has been selected, the track has been set, the sector has been set, and the DMA address has been specified, the READ subroutine attempts to read one sector based upon these parameters, and returns the following error codes in register A:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>no errors occurred</td>
</tr>
<tr>
<td>1</td>
<td>non-recoverable error occurred</td>
</tr>
</tbody>
</table>

Currently, CP/M responds only to a zero or non-zero value as the return code. That is, if the value in register A is 0 then CP/M assumes that the disk operation completed properly. If an error occurs, however, the CB IOS should attempt at least 10 retries to see if the error is recoverable. When an error is reported the BDOS will print the message "BDOS ERR ON x: BAD SECTOR". The operator then has the option of typing <cr> to ignore the error, or ctl-C to abort.

WRITE
Write the data from the currently selected DMA address to the currently selected drive, track, and sector. The data should be marked as "non deleted data" to
maintain compatibility with other CP/M systems. The error codes given in the READ command are returned in register A, with error recovery attempts as described above.

**LISTST**

Return the ready status of the list device. Used by the DESPOOL program to improve console response during its operation. The value 00 is returned in A if the list device is not ready to accept a character, and 0FFH if a character can be sent to the printer. Note that a 00 value always suffices.

**SECTRAN**

Performs sector logical to physical sector translation in order to improve the overall response of CP/M. Standard CP/M systems are shipped with a "skew factor" of 6, where six physical sectors are skipped between each logical read operation. This skew factor allows enough time between sectors for most programs to load their buffers without missing the next sector. In particular computer systems which use fast processors, memory, and disk subsystems, the skew factor may be changed to improve overall response. Note, however, that you should maintain a single density IBM compatible version of CP/M for information transfer into and out of your computer system, using a skew factor of 6. In general, SECTRAN receives a logical sector number in BC, and a translate table address in DE. The sector number is used as an index into the translate table, with the resulting physical sector number in HL. For standard systems, the tables and indexing code is provided in the CBIOS and need not be changed.

(All Information Contained Herein is Proprietary to Digital Research.)
7. A SAMPLE BIOS

The program shown in Appendix C can serve as a basis for your first BIOS. The simplest functions are assumed in this BIOS, so that you can enter it through the front panel, if absolutely necessary. Note that the user must alter and insert code into the subroutines for CONST, CONIN, CONOUT, READ, WRITE, and WAITIO subroutines. Storage is reserved for user-supplied code in these regions. The scratch area reserved in page zero (see Section 9) for the BIOS is used in this program, so that it could be implemented in ROM, if desired.

Once operational, this skeletal version can be enhanced to print the initial sign-on message and perform better error recovery. The subroutines for LIST, PUNCH, and READER can be filled-out, and the IOBYTE function can be implemented.
8. A SAMPLE COLD START LOADER

The program shown in Appendix D can serve as a basis for your cold start loader. The disk read function must be supplied by the user, and the program must be loaded somehow starting at location 0000. Note that space is reserved for your patch so that the total amount of storage required for the cold start loader is 128 bytes. Eventually, you will probably want to get this loader onto the first disk sector (track 0, sector 1), and cause your controller to load it into memory automatically upon system start-up. Alternatively, you may wish to place the cold start loader into ROM, and place it above the CP/M system. In this case, it will be necessary to originate the program at a higher address, and key-in a jump instruction at system start-up which branches to the loader. Subsequent warm starts will not require this key-in operation, since the entry point 'WBOOT' gets control, thus bringing the system in from disk automatically. Note also that the skeletal cold start loader has minimal error recovery, which may be enhanced on later versions.

(All Information Contained Herein is Proprietary to Digital Research.)
9. RESERVED LOCATIONS IN PAGE ZERO

Main memory page zero, between locations 00H and 0FFH, contains several segments of code and data which are used during CP/M processing. The code and data areas are given below for reference purposes.

<table>
<thead>
<tr>
<th>Locations from</th>
<th>to</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000H - 0002H</td>
<td></td>
<td>Contains a jump instruction to the warm start entry point at location 4A03H+b. This allows a simple programmed restart (JMP 0000H) or manual restart from the front panel.</td>
</tr>
<tr>
<td>0003H - 0003H</td>
<td></td>
<td>Contains the Intel standard IOBYTE, which is optionally included in the user's CBIOS, as described in Section 6.</td>
</tr>
<tr>
<td>0004H - 0004H</td>
<td></td>
<td>Current default drive number (0=A,...,15=P).</td>
</tr>
<tr>
<td>0005H - 0007H</td>
<td></td>
<td>Contains a jump instruction to the BDOS, and serves two purposes: JMP 0005H provides the primary entry point to the BDOS, as described in the manual &quot;CP/M Interface Guide,&quot; and LHLD 0006H brings the address field of the instruction to the HL register pair. This value is the lowest address in memory used by CP/M (assuming the CCP is being overlayed). Note that the DDT program will change the address field to reflect the reduced memory size in debug mode.</td>
</tr>
<tr>
<td>0008H - 0027H</td>
<td></td>
<td>(interrupt locations 1 through 5 not used)</td>
</tr>
<tr>
<td>0030H - 0037H</td>
<td></td>
<td>(interrupt location 6, not currently used - reserved)</td>
</tr>
<tr>
<td>0038H - 003AH</td>
<td></td>
<td>Restart 7 - Contains a jump instruction into the DDT or SID program when running in debug mode for programmed breakpoints, but is not otherwise used by CP/M.</td>
</tr>
<tr>
<td>003BH - 003FH</td>
<td></td>
<td>(not currently used - reserved)</td>
</tr>
<tr>
<td>0040H - 004FH</td>
<td></td>
<td>16 byte area reserved for scratch by CBIOS, but is not used for any purpose in the distribution version of CP/M</td>
</tr>
<tr>
<td>0050H - 005BH</td>
<td></td>
<td>(not currently used - reserved)</td>
</tr>
<tr>
<td>005CH - 007CH</td>
<td></td>
<td>default file control block produced for a transient program by the Console Command Processor.</td>
</tr>
<tr>
<td>007DH - 007FH</td>
<td></td>
<td>Optional default random record position</td>
</tr>
</tbody>
</table>

(All Information Contained Herein is Proprietary to Digital Research.)
88888 - 00FFH  default 128 byte disk buffer (also filled with the command line when a transient is loaded under the CCP).

Note that this information is set-up for normal operation under the CP/M system, but can be overwritten by a transient program if the BDOS facilities are not required by the transient.

If, for example, a particular program performs only simple I/O and must begin execution at location 0, it can be first loaded into the TPA, using normal CP/M facilities, with a small memory move program which gets control when loaded (the memory move program must get control from location 0100H, which is the assumed beginning of all transient programs). The move program can then proceed to move the entire memory image down to location 0, and pass control to the starting address of the memory load. Note that if the BIOS is overwritten, or if location 0 (containing the warm start entry point) is overwritten, then the programmer must bring the CP/M system back into memory with a cold start sequence.

(All Information Contained Herein is Proprietary to Digital Research.)
Tables are included in the BIOS which describe the particular characteristics of the disk subsystem used with CP/M. These tables can be either hand-coded, as shown in the sample CBIOS in Appendix C, or automatically generated using the DISKDEF macro library, as shown in Appendix B. The purpose here is to describe the elements of these tables.

In general, each disk drive has an associated (16-byte) disk parameter header which both contains information about the disk drive and provides a scratchpad area for certain BDOS operations. The format of the disk parameter header for each drive is shown below:

```
<table>
<thead>
<tr>
<th>Disk Parameter Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>XLT</td>
</tr>
<tr>
<td>16b</td>
</tr>
</tbody>
</table>
```

where each element is a word (16-bit) value. The meaning of each Disk Parameter Header (DPH) element is:

- **XLT**: Address of the logical to physical translation vector, if used for this particular drive, or the value 0000H if no sector translation takes place (i.e., the physical and logical sector numbers are the same). Disk drives with identical sector skew factors share the same translate tables.

- **0000**: Scratchpad values for use within the BDOS (initial value is unimportant).

- **DIRBUF**: Address of a 128 byte scratchpad area for directory operations within BDOS. All DPH's address the same scratchpad area.

- **DPB**: Address of a disk parameter block for this drive. Drives with identical disk characteristics address the same disk parameter block.

- **CSV**: Address of a scratchpad area used for software check for changed disks. This address is different for each DPH.

- **ALV**: Address of a scratchpad area used by the BDOS to keep disk storage allocation information. This address is different for each DPH.

Given n disk drives, the DPH's are arranged in a table whose first row of 16 bytes corresponds to drive 0, with the last row corresponding to drive n-1. The table thus appears as:

(All Information Contained Herein is Proprietary to Digital Research.)
where the label DPBASE defines the base address of the DPH table.

A responsibility of the SELDSK subroutine is to return the base address of the DPH for the selected drive. The following sequence of operations returns the table address, with a 0000H returned if the selected drive does not exist.

\[
\begin{align*}
\text{SELDSK:} & \quad ; \text{SELECT DISK GIVEN BY BC} \\
& \text{LXI H,0000H} \quad ; \text{ERROR CODE} \\
& \text{MOV A,C} \quad ; \text{DRIVE OK?} \\
& \text{CPI NDISKS} \quad ; \text{CY IF SO} \\
& \text{RNC} \quad ; \text{RET IF ERROR} \\
& \text{; NO ERROR, CONTINUE} \\
& \text{MOV L,C} \quad ; \text{LOW(DISK)} \\
& \text{MOV H,B} \quad ; \text{HIGH(DISK)} \\
& \text{DAD H} \quad ; *2 \\
& \text{DAD H} \quad ; *4 \\
& \text{DAD H} \quad ; *8 \\
& \text{DAD H} \quad ; *16 \\
& \text{LXI D,DPBASE} \quad ; \text{FIRST DPH} \\
& \text{DAD D} \quad ; \text{DPH(DISK)} \\
& \text{RET}
\end{align*}
\]

The translation vectors (XLT 00 through XLTn-l) are located elsewhere in the BIOS, and simply correspond one-for-one with the logical sector numbers zero through the sector count-l. The Disk Parameter Block (DPB) for each drive is more complex. A particular DPB, which is addressed by one or more DPH's, takes the general form

\[
\begin{array}{|c|c|c|c|c|c|c|c|c|c|}
\hline
| SPT | BSH | LBM | EXM | DSM | DRM | AL0 | AL1 | CKS | OFF |
\hline
| 16b | 8b | 8b | 8b | 16b | 16b | 8b | 8b | 16b | 16b |
\hline
\end{array}
\]

where each is a byte or word value, as shown by the "8b" or "16b" indicator below the field.

- **SPT** is the total number of sectors per track
- **BSH** is the data allocation block shift factor, determined by the data block allocation size.

(All Information Contained Herein is Proprietary to Digital Research.)
EXM is the extent mask, determined by the data block allocation size and the number of disk blocks.

DSM determines the total storage capacity of the disk drive.

DRM determines the total number of directory entries which can be stored on this drive. AL0, AL1 determine reserved directory blocks.

CKS is the size of the directory check vector.

OFF is the number of reserved tracks at the beginning of the (logical) disk.

The values of BSH and BLM determine (implicitly) the data allocation size BLS, which is not an entry in the disk parameter block. Given that the designer has selected a value for BLS, the values of BSH and BLM are shown in the table below.

<table>
<thead>
<tr>
<th>BLS</th>
<th>BSH</th>
<th>BLM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,024</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>2,048</td>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>4,096</td>
<td>5</td>
<td>31</td>
</tr>
<tr>
<td>8,192</td>
<td>6</td>
<td>63</td>
</tr>
<tr>
<td>16,384</td>
<td>7</td>
<td>127</td>
</tr>
</tbody>
</table>

where all values are in decimal. The value of EXM depends upon both the BLS and whether the DSM value is less than 256 or greater than 255, as shown in the following table.

<table>
<thead>
<tr>
<th>BLS</th>
<th>DSM &lt; 256</th>
<th>DSM &gt; 255</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,024</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>2,048</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4,096</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>8,192</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>16,384</td>
<td>15</td>
<td>7</td>
</tr>
</tbody>
</table>

The value of DSM is the maximum data block number supported by this particular drive, measured in BLS units. The product BLS times (DSM+1) is the total number of bytes held by the drive and, of course, must be within the capacity of the physical disk, not counting the reserved operating system tracks.

The DRM entry is the one less than the total number of directory entries, which can take on a 16-bit value. The values of AL0 and AL1, however, are determined by DRM. The two values AL0 and AL1 can together be considered a string of 16-bits, as shown below.

(All Information Contained Herein is Proprietary to Digital Research.)
where position 00 corresponds to the high order bit of the byte labelled AL0, and 15 corresponds to the low order bit of the byte labelled AL1. Each bit position reserves a data block for number of directory entries, thus allowing a total of 16 data blocks to be assigned for directory entries (bits are assigned starting at 00 and filled to the right until position 15). Each directory entry occupies 32 bytes, resulting in the following table:

<table>
<thead>
<tr>
<th>BLS</th>
<th>Directory Entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,024</td>
<td>32 times # bits</td>
</tr>
<tr>
<td>2,048</td>
<td>64 times # bits</td>
</tr>
<tr>
<td>4,096</td>
<td>128 times # bits</td>
</tr>
<tr>
<td>8,192</td>
<td>256 times # bits</td>
</tr>
<tr>
<td>16,384</td>
<td>512 times # bits</td>
</tr>
</tbody>
</table>

Thus, if DRM = 127 (128 directory entries), and BLS = 1024, then there are 32 directory entries per block, requiring 4 reserved blocks. In this case, the 4 high order bits of AL0 are set, resulting in the values AL0 = 0F0H and AL1 = 00H.

The CKS value is determined as follows: if the disk drive media is removable, then CKS = (DRM+1)/4, where DRM is the last directory entry number. If the media is fixed, then set CKS = 0 (no directory records are checked in this case).

Finally, the OFF field determines the number of tracks which are skipped at the beginning of the physical disk. This value is automatically added whenever SETTRK is called, and can be used as a mechanism for skipping reserved operating system tracks, or for partitioning a large disk into smaller segmented sections.

To complete the discussion of the DPB, recall that several DPH's can address the same DPB if their drive characteristics are identical. Further, the DPB can be dynamically changed when a new drive is addressed by simply changing the pointer in the DPH since the BDOS copies the DPB values to a local area whenever the SELDSK function is invoked.

Returning back to the DPH for a particular drive, note that the two address values CSV and ALV remain. Both addresses reference an area of uninitialized memory following the BIOS. The areas must be unique for each drive, and the size of each area is determined by the values in the DPB.

The size of the area addressed by CSV is CKS bytes, which is sufficient to hold the directory check information for this particular drive. If CKS = (DRM+1)/4, then you must reserve (DRM+1)/4 bytes for directory check use. If CKS = 0, then no storage is reserved.

(All Information Contained Herein is Proprietary to Digital Research.)
The size of the area addressed by ALV is determined by the maximum number of data blocks allowed for this particular disk, and is computed as \((\text{DSM}/8)+1\).

The CBIOS shown in Appendix C demonstrates an instance of these tables for standard 8" single density drives. It may be useful to examine this program, and compare the tabular values with the definitions given above.
A macro library is shown in Appendix P, called DISKDEF, which greatly simplifies the table construction 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.

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 the in-line fixed data tables described in the previous section, 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 in memory above your BIOS.

The form of the DISKDEF macro call is

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

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
- `[Ø]` is an optional 1.4 compatibility flag

The value "dn" is the drive number being defined with this DISKDEF

(All Information Contained Herein is Proprietary to Digital Research.)
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). As stated in the previous section, 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 space or to simulate several logical drives on a single large capacity physical drive. Finally, the [0] 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:

(All Information Contained Herein is Proprietary to Digital Research.)
with all disks having the same parameter values of 26 sectors per track (numbered 1 through 26), with 6 sectors skipped between each access, 1024 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 DISKS macro generates n Disk Parameter Headers (DPH's), starting at the DPH table address DPBASE generated by the macro. Each disk header block contains sixteen bytes, as described above, and correspond one-for-one 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  $
DPE0:  DW  XLT0, 0000H, 0000H, 0000H, DIRBUF, DPBO, CSV0, ALV0
DPE1:  DW  XLT0, 0000H, 0000H, 0000H, DIRBUF, DPBO, CSV1, ALV1
DPE2:  DW  XLT0, 0000H, 0000H, 0000H, DIRBUF, DPBO, CSV2, ALV2
DPE3:  DW  XLT0, 0000H, 0000H, 0000H, DIRBUF, DPBO, CSV3, ALV3
```

where the DPH labels are included for reference purposes to show the beginning table addresses for each drive 0 through 3. The values contained within the disk parameter header are described in detail in the previous section. The check and allocation vector addresses are generated by the ENDEF macro in the ram area following the BIOS code and tables.

Note that if the "skf" (skew factor) parameter is omitted (or equal to 0), the translation table is omitted, and a 0000H value is inserted in the XLT position of the disk parameter header for the disk. In a subsequent call to perform the logical to physical translation, SECTRAN receives a translation table address of DE = 0000H, and simply returns the original logical sector from BC in the HL register pair. A translate table is constructed when the skf parameter is present, and the (non-zero) table address is placed into the corresponding DPH's. 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
```

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

(All Information Contained Herein is Proprietary to Digital Research.)
which indicates that uninitialized RAM begins at location 4C72H, ends at 4DB0H-1, and occupies 013CH bytes. You must ensure that these addresses are free for use after the system is loaded.

After modification, you can use the STAT program to check your drive characteristics, since STAT uses the disk parameter block to decode the drive information. The STAT command form

```
STAT d:DSK:
```

decodes the disk parameter block for drive d (d=A,...,P) and displays the values shown below:

- r: 128 Byte Record Capacity
- k: Kilobyte Drive Capacity
- d: 32 Byte Directory Entries
- c: Checked Directory Entries
- e: Records/ Extent
- b: Records/ Block
- s: Sectors/ Track
- t: Reserved Tracks

Three examples of DISKDEF macro invocations are shown below with corresponding STAT parameter values (the last produces a full 8-megabyte system).

```
DISKDEF 0,1,58,,2048,256,128,128,2
r=4096, k=512, d=128, c=128, e=256, b=16, s=58, t=2

DISKDEF 0,1,58,,2048,1024,300,0,2
r=16384, k=2048, d=300, c=0, e=128, b=16, s=58, t=2

DISKDEF 0,1,58,,16384,512,128,128,2
r=65536, k=8192, d=128, c=128, e=1024, b=128, s=58, t=2
```

(All Information Contained Herein is Proprietary to Digital Research.)
12. SECTOR BLOCKING AND DEBLOCKING.

Upon each call to the BIOS WRITE entry point, the CP/M BDOS includes information which allows effective sector blocking and deblocking where the host disk subsystem has a sector size which is a multiple of the basic 128-byte unit. The purpose here is to present a general-purpose algorithm which can be included within your BIOS which uses the BDOS information to perform the operations automatically.

Upon each call to WRITE, the BDOS provides the following information in register C:

\[
\begin{align*}
0 &= \text{normal sector write} \\
1 &= \text{write to directory sector} \\
2 &= \text{write to the first sector of a new data block}
\end{align*}
\]

Condition 0 occurs whenever the next write operation is into a previously written area, such as a random mode record update, when the write is to other than the first sector of an unallocated block, or when the write is not into the directory area. Condition 1 occurs when a write into the directory area is performed. Condition 2 occurs when the first record (only) of a newly allocated data block is written. In most cases, application programs read or write multiple 128-byte sectors in sequence, and thus there is little overhead involved in either operation when blocking and deblocking records since pre-read operations can be avoided when writing records.

Appendix G lists the blocking and deblocking algorithms in skeletal form (this file is included on your CP/M disk). Generally, the algorithms map all CP/M sector read operations onto the host disk through an intermediate buffer which is the size of the host disk sector. Throughout the program, values and variables which relate to the CP/M sector involved in a seek operation are prefixed by "sek," while those related to the host disk system are prefixed by "hst." The equate statements beginning on line 29 of Appendix G define the mapping between CP/M and the host system, and must be changed if other than the sample host system is involved.

The entry points BOOT and WBOOT must contain the initialization code starting on line 57, while the SELDSK entry point must be augmented by the code starting on line 65. Note that although the SELDSK entry point computes and returns the Disk Parameter Header address, it does not physically selected the host disk at this point (it is selected later at READHST or WRITEHST). Further, SETTRK, SETTRK, and SETDMA simply store the values, but do not take any other action at this point. SECTRAN performs a trivial function of returning the physical sector number.

The principal entry points are READ and WRITE, starting on lines 110 and 125, respectively. These subroutines take the place of your previous READ and WRITE operations.

The actual physical read or write takes place at either WRITEHST or READHST, where all values have been prepared: hstdsk is the host

(All Information Contained Herein is Proprietary to Digital Research.)
disk number, hstrrk is the host track number, and hstsec is the host sector number (which may require translation to a physical sector number). You must insert code at this point which performs the full host sector read or write into, or out of, the buffer at hstbuf of length hstsiz. All other mapping functions are performed by the algorithms.

This particular algorithm was tested using an 80 megabyte hard disk unit which was originally configured for 128 byte sectors, producing approximately 35 megabytes of formatted storage. When configured for 512 byte host sectors, usable storage increased to 57 megabytes, with a corresponding 400% improvement in overall response. In this situation, there is no apparent overhead involved in deblocking sectors, with the advantage that user programs still maintain the (less memory consuming) 128-byte sectors. This is primarily due, of course, to the information provided by the BDOS which eliminates the necessity for pre-read operations to take place.
APPENDIX A: THE MDS COLD START LOADER

; MDS-800 Cold Start Loader for CP/M 2.0
;
; Version 2.0 August, 1979
;
0000 = false equ 0
ffff = true equ not false
0000 = testing equ false
;
if testing
bias equ 03400h
endif

if not testing
0000 = bias equ 0000h
endif

0000 = cpmb equ bias ;base of dos load
0806 = bdos equ 806h+bias ;entry to dos for calls
1880 = bdose equ 1880h+bias ;end of dos load
1600 = boot equ 1600h+bias ;cold start entry point
1603 = rboot equ boot+3 ;warm start entry point
;
3000 org 0300h ;loaded here by hardware

1880 = bdosl equ bdose-cpmb
0002 = ntrks equ 2 ;tracks to read
0031 = bdoss equ bdosl/128 ;# sectors in bdos
0019 = bdos0 equ 25 ;# on track 0
0018 = bdosl equ bdoss-bdos0 ;# on track 1

f800 = mon80 equ 0f800h ;intel monitor base
ff0f = rmon80 equ 0ff0fh ;restart location for mon80
0078 = base equ 078h ;'base' used by controller
0079 = rtype equ base+1 ;result type
007b = rbyte equ base+3 ;result byte
007f = reset equ base+7 ;reset controller
;
0078 = dstat equ base ;disk status port
0079 = ilow equ base+1 ;low iopb address
007a = ihigh equ base+2 ;high iopb address
00ff = bsw equ 0ffh ;boot switch
0003 = recal equ 3h ;recalibrate selected drive
0004 = readf equ 4h ;disk read function
0100 = stack equ 100h ;use end of boot for stack
;
rstart:
3000 310001 lxi sp,stack;in case of call to mon80
;
cold start
3003 db79 in rtype
3005 db7b in rbyte
;
coldstart:
3007 dbff in bsw
3008 e20730 ani 02h ;switch on?
3008 e20730 jnz coldstart

212
clear the controller
out reset ;logic cleared

mvi b,ntrks ;number of tracks to read
lxi h,iopb0

start:

read first/next track into cpmb
mov a,l
out ilow
mov a,h
out ihigh
wait0: in dstat
ani 4
jz wait0

check disk status
in rtype
ani 1lb
cpi 2

if testing
cnc rmon80 ;go to monitor if 11 or 10
endif

if not testing
rnc rstart ;retry the load
endif

rbyte ;i/o complete, check status
if not ready, then go to mon80
ral
cc rmon80 ;not ready bit set
rar ;restore
ani 1lll0b ;overrun/addr err/seek/crc
if testing
cnz rmon80 ;go to monitor
endif
if not testing
jnz rstart ;retry the load
endif

lxi d,iopb1 ;length of iopb
dad d ;addressing next iopb
dcr b ;count down tracks
jnz start

jmp boot, print message, set-up jmps
jmp boot

parameter blocks
iopb0: db 80h ; iocw, no update
readf ; read function
bdos0 ; # sectors to read trk 0
0 ; track 0
2 ; start with sector 2, trk 0
1 ; track 1
1 ; sector 1
cpmb+bdos0*128 ; base of second rd

end
APPENDIX B: THE MDS BASIC I/O SYSTEM (BIOS)

; mds-800 I/O drivers for cp/m 2.0
; (four drive single density version)
; version 2.0 august, 1979

0014 vers equ 20 ;version 2.0

; copyright (c) 1979
digital research
box 579, pacific grove
california, 93950

4a00 org 4a00h ;base of bios in 20k system
3400 cpmb equ 3400h ;base of cpm ccp
3c06 bdos equ 3c06h ;base of bdos in 20k system
1600 cpm1 equ $-cpmb ;length (in bytes) of cpm system
002c nsects equ cpm1/128;number of sectors to load
0002 offset equ 2 ;number of disk tracks used by cp
0004 cdisk equ 0004h ;address of last logged disk
0080 buff equ 0080h ;default buffer address
000a retry equ 10 ;max retries on disk i/o before e

perform following functions
boot cold start
wboot warm start (save i/o byte)
(boot and wboot are the same for mds)
const console status
reg-a = 00 if no character ready
reg-a = ff if character ready
conin console character in (result in reg-a)
conout console character out (char in reg-c)
list list out (char in reg-c)
punch punch out (char in reg-c)
reader paper tape reader in (result to reg-a)
home move to track 00

(the following calls set-up the io parameter bloc
mds, which is used to perform subsequent reads an
seldisk select disk given by reg-c (0,1,2...)
settrk set track address (0,...,76) for sub r/w
setsec set sector address (1,...,26)
setdma set subsequent dma address (initially 80h

read/write assume previous calls to set i/o parms
read read track/sector to preset dma address
write write track/sector from preset dma address

jump vector for individual routines

4a00 c3b34a jmp boot
4a03 c3c34a wboote: jmp wboot
4a06 c3614b jmp const
4a09 c3644b jmp conin
4a0c c36a4b jmp conout

215
maclib diskdef ;load the disk definition library
disks 4 ;four disks
4a33+= dpbase equ $ ;base of disk parameter blocks
4a33+824a00 dpe0: dw xlt0,0000h ;translate table
4a37+000000 dw 0000h,0000h ;scratch area
4a3b+6e4c73 dw dirbuf,dpb0 ;dir buff,parm block
4a3f+0d4d4e dw csv0,alv0 ;check, alloc vectors
4a43+824a00 dpel: dw xlt1,0000h ;translate table
4a47+000000 dw 0000h,0000h ;scratch area
4a4b+6e4c73 dw dirbuf,dpbl ;dir buff,parm block
4a4f+3c4d1d dw csv1,alvl ;check, alloc vectors
4a53+824a00 dpe2: dw xlt2,0000h ;translate table
4a57+000000 dw 0000h,0000h ;scratch area
4a5b+6e4c73 dw dirbuf,dpb2 ;dir buff,parm block
4a5f+6d4d4c dw csv2,alv2 ;check, alloc vectors
4a63+824a00 dpe3: dw xlt3,0000h ;translate table
4a67+000000 dw 0000h,0000h ;scratch area
4a6b+6e4c73 dw dirbuf,dpb3 ;dir buff,parm block
4a6f+9a4d7b dw csv3,alv3 ;check, alloc vectors
diskdef 0,1,26,6,1024,243,64,64,offset
4a73+= dpb0 equ $ ;disk parm block
4a73+1a00 dw 26 ;sec per track
4a75+03 db 3 ;block shift
4a76+07 db 7 ;block mask
4a77+00 db 0 ;extnt mask
4a78+f200 dw 242 ;disk size-l
4a7a+3f00 dw 63 ;directory max
4a7c+c0 db 192 ;alloc0
4a7d+00 db 0 ;alloc1
4a7e+1000 dw 16 ;check size
4a80+0200 dw 2 ;offset
4a82+= xlt0 equ $ ;translate table
4a82+01 db 1
4a83+07 db 7
4a84+0d db 13
4a85+13 db 19
4a86+19 db 25
4a87+05 db 5
4a88+0b db 11
4a89+11 db 17
4a8a+17 db 23
4a8b+03 db 3
; assembly code

; equivalent parameters
4a73+ = dpbl equ dpb0
001f+ = als1 equ als0
0010+ = cssl equ css0
4a82+ = xltl equ xlt0

diskdef 1,0

; equivalent parameters
4a73+ = dpb2 equ dpb0
001f+ = als2 equ als0
0010+ = css2 equ css0
4a82+ = xlt2 equ xlt0

diskdef 2,0

; equivalent parameters
4a73+ = dpb3 equ dpb0
001f+ = als3 equ als0
0010+ = css3 equ css0
4a82+ = xlt3 equ xlt0

diskdef 3,0

; equivalent parameters
endef occurs at end of assembly

; end of controller-independent code, the remaini
; are tailored to the particular operating environm
; be altered for any system which differs from the

; the following code assumes the mds monitor exists
; and uses the i/o subroutines within the monitor

; we also assume the mds system has four disk drive

00fd = revrt equ 0fdh ;interrupt revert port
00fc = intc equ 0fch ;interrupt mask port
00f3 = icon equ 0f3h ;interrupt control port
007e = inte equ 0111$110b;enable rst 0(warm boot),rst 7

; mds monitor equates

f800 = mon80 equ 0f000h ;mds monitor
ff0f = rmon80 equ 0ff0fh ;restart mon80 (boot error)
f803 = ci equ 0f803h ;console character to reg-a
f806 = ri equ 0f806h ;reader in to reg-a
f809 = co equ 0f809h ;console char from c to console o
f80c = po equ 0f80ch ;punch char from c to punch devic
f80f = lo equ 0f80fh ;list from c to list device
f812 = csts equ 0f812h ;console status 00/ff to register
; disk ports and commands
0078 = base equ 78h  ; base of disk command io ports
0078 = dstat equ base  ; disk status (input)
0079 = rtype equ base+1  ; result type (input)
007b = rbyte equ base+3  ; result byte (input)
0079 = ilow equ base+1  ; iopb low address (output)
007a = ihigh equ base+2  ; iopb high address (output)
0004 = readf equ 4h  ; read function
0006 = writf equ 6h  ; write function
0003 = recal equ 3h  ; recalibrate drive
0004 = iordy equ 4h  ; i/o finished mask
000d = cr equ 0dh  ; carriage return
000a = lf equ 0ah  ; line feed

; signon: ; signon message: xxk cp/m vers y.y
4a9c 0d0a0a db cr,lf,lf
4a9f 3230 db '20'  ; sample memory size
4aaf 6b2043f db 'k cp/m vers'
4aad 322e30 db vers/l0+'0','.',vers mod 10+'0'
4ab0 0d0a00 db cr,lf,0

; boot: ; print signon message and go to ccp
; (note: mds boot initialized iobyte at 0003h)
4ab3 310001 lxi sp,buff+80h
4ab6 219c4a lxi h,signon
4ab9 cdd34b call prmsg  ; print message
4abc af xra a  ; clear accumulator
4abd 320400 sta cdisk  ; set initially to disk a
4ac0 c30f4b jmp gocpm  ; go to cp/m

; wboot: ; loader on track 0, sector 1, which will be skippe
; read cp/m from disk - assuming there is a 128 byt
; start.

4ac3 318000 lxi sp,buff  ; using dma - thus 80 thru ff ok f
4ac6 0e0a mvi c,retry  ; max retries
4ac8 c5 push b

; wboot0: ; enter here on error retries
4ac9 010034 lxi b,cpmb  ; set dma address to start of disk
4acc cdbb4b call setdma
4acf 0e00 mvi c,0  ; boot from drive 0
4ad1 cd7d4b call seldsk
4ad4 0e00 mvi c,0
4ad6 cda74b call settrk  ; start with track 0
4ad9 0e02 mvi c,2  ; start reading sector 2
4adb cdac4b call setsec

; read sectors, count nsects to zero
4ade cl pop b  ; 10-error count
4adf 062c mvi b,nsects
rdsec: ; read next sector
  push b ; save sector count
  call read
  jnz booterr ; retry if errors occur
  lhld iod ; increment dma address
  lxi d,128 ; sector size
  dad d ; incremented dma address in hl
  mov b,h
  mov c,l ; ready for call to set dma
  call setdma
  lda ios ; sector number just read
  cpi 26 ; read last sector?
  lda iot ; get track to register a
  mov c,a ; ready for call
  call settrk
  xra a ; clear sector number
  inr a ; to next sector
  mov c,a ; ready for call
  call setsec
  pop b ; recall sector count
  dcr b ; done?
  jnz rdsec ; must be sector 26, zero and go to next track
  lda iot ; get track to register a
  mov c,a ; ready for call
  call settrk
  xra a ; clear sector number
  inr a ; to next sector
  mov c,a ; ready for call
  call setsec
  pop b ; recall sector count
  dcr b ; done?
  jnz rdsec ; done with the load, reset default buffer address

gocpm: ; (enter here from cold start boot)
; enable rst0 and rst7
    di
    mvi a,12h ; initialize command
    out revrt
    xra a
    out intc ; cleared
    mvi a,inte ; rst0 and rst7 bits on
    out intc
    xra a
    out icon ; interrupt control

; set default buffer address to 80h
    lxi b,buff
    call setdma

; reset monitor entry points
    mvi a,jmp
    sta 0
    lxi h,wboote
    shld 1 ; jmp wboot at location 00
    sta 5
    lxi h,bdos
    shld 6 ; jmp bdos at location 5
    sta 7*8+1 ; jmp to mon80 (may have been chan
    lxi h,mon80
    shld 7*8+1 ; leave iobyte set
4b41 3a0400
4b44 4f
4b45 fb
4b46 c30034

; previously selected disk was b, send parameter to
; last logged disk number
lda cdisk ;send to ccp to log it in
mov c,a
ei
jmp cpmb

; error condition occurred, print message and retry
booterr:
4b49 cl
pop b ;recall counts
4b4a 0d
dcr c
4b4b ca524b
jz booter0
; try again
4b4e c5
push b
4b4f c3c94a
jmp wboot0

; booter0:
; otherwise too many retries
4b52 215b4b
lxi h,bootmsg
4b55 cdd34b
call prmsg
4b58 c30fff
jmp rmon80 ;mds hardware monitor

; bootmsg:
4b5b 3f626f4
db '?boot',0

; const: ;console status to reg-a
; (exactly the same as mds call)
4b61 c312f8
jmp csts

; conin: ;console character to reg-a
4b64 cd03f8
call ci
4b67 e67f
ani 7fh ;remove parity bit
4b69 c9
ret

; conout: ;console character from c to console out
4b6a c309f8
jmp co

; list: ;list device out
; (exactly the same as mds call)
4b6d c30ff8
jmp lo

; listst:
4b70 af
xra a
4b71 c9
ret ;always not ready

; punch: ;punch device out
; (exactly the same as mds call)
4b72 c30cf8
jmp po

; reader: ;reader character in to reg-a
; (exactly the same as mds call)
4b75 c306f8
jmp ri

; home: ;move to home position
; treat as track 00 seek
4b78 0e00     mvi    c,0
4b7a c3a74b   jmp    settrak
;
seldsk: ;select disk given by register c
4b7d 210000   lxi    h,0000h ;return 0000 if error
4b80 79       mov    a,c
4b81 fe04     cpi    ndisks ;too large?
4b83 d0       rnc    ;leave hl = 0000

4b84 e602     ani    lb    ;00 00 for drive 0,1 and 10 10 fo
4b86 32664c   sta    dbank ;to select drive bank
4b89 79       mov    a,c    ;00, 01, 10, 11
4b8a e601     ani    lb    ;mds has 0,1 at 78, 2,3 at 88
4b8c b7       ora    a    ;result 00?
4b8d ca924b   jz     setdrive
4b90 3e30     mvi    a,00110000b ;selects drive 1 in bank

setdrive:
4b92 47       mov    b,a    ;save the function
4b93 21684c   lxi    h,iof ;io function
4b96 7e       mov    a,m
4b97 e6cf     ani    11001111b ;mask out disk number
4b99 b0       ora    b    ;mask in new disk number
4b9a 77       mov    m,a    ;save it in iopb
4b9b 69       mov    h,6 ;hl=disk number
4b9e 29       dad    h    ;*2
4b9f 29       dad    h    ;*4
4ba0 29       dad    h    ;*8
4ba1 29       dad    h    ;*16
4ba2 11334a   lxi    d,dpbase
4ba5 19       dad    d    ;hl=disk header table address
4ba6 c9       ret

;
;
settrak: ;set track address given by c
4ba7 216a4c   lxi    h,iot
4baa 71       mov    m,c
4bab c9       ret

;
;
setsec: ;set sector number given by c
4bac 216b4c   lxi    h,ios
4baf 71       mov    m,c
4bb0 c9       ret

sectran:     ;translate sector bc using table at de
4bbl 0600     mvi    b,0    ;double precision sector number i
4bb3 eb       xchg   ;translate table address to hl
4bb4 09       dad    b    ;translate(sector) address
4bb5 7e       mov    a,m    ;translated sector number to a
4bb6 326b4c   sta    ios
4bb9 6f       mov    l,a    ;return sector number in l
4bbac9       ret

;
;
setdma: ;set dma address given by regs b,c

221
mov l,c
mov h,b
shld iod
ret

; read:
mov c,readf ; set to read function
call setfunc
call waitio ; perform read function
ret ; may have error set in reg-a

; write:
mov c,writf ; set to write function
call setfunc
call waitio
ret ; may have error set

; utility subroutines

prmsg: ; print message at h,l to 0
mov a,m
ora a ; zero?
rz
push h
mov c,a
call conout
pop h
inx h
jmp prmsg

setfunc:
set function for next i/o (command in reg-c)
lxi h,iof ; io function address
mov a,m ; get it to accumulator for maskin
ani 111110b0b ; remove previous command
ora c ; set to new command
mov m,a ; replaced in iopb

; the mds-800 controller req's disk bank bit in sec
; mask the bit from the current i/o function
ani 00100000b ; mask the disk select bit
lxi h,ios ; address the sector selec
ora m ; select proper disk bank
mov m,a ; set disk select bit on/o
ret

waitio:
mvi c,retry ; max retries before perm error

rewait:

mov l,c
mov h,b
shld iod
ret

; read:
mov c,readf ; set to read function
call setfunc
call waitio ; perform read function
ret ; may have error set in reg-a

; write:
mov c,writf ; set to write function
call setfunc
call waitio
ret ; may have error set

; utility subroutines

prmsg: ; print message at h,l to 0
mov a,m
ora a ; zero?
rz
push h
mov c,a
call conout
pop h
inx h
jmp prmsg

setfunc:
set function for next i/o (command in reg-c)
lxi h,iof ; io function address
mov a,m ; get it to accumulator for maskin
ani 111110b0b ; remove previous command
ora c ; set to new command
mov m,a ; replaced in iopb

; the mds-800 controller req's disk bank bit in sec
; mask the bit from the current i/o function
ani 00100000b ; mask the disk select bit
lxi h,ios ; address the sector selec
ora m ; select proper disk bank
mov m,a ; set disk select bit on/o
ret

waitio:
mvi c,retry ; max retries before perm error

rewait:

mov l,c
mov h,b
shld iod
ret

; read:
mov c,readf ; set to read function
call setfunc
call waitio ; perform read function
ret ; may have error set in reg-a

; write:
mov c,writf ; set to write function
call setfunc
call waitio
ret ; may have error set

; utility subroutines

prmsg: ; print message at h,l to 0
mov a,m
ora a ; zero?
rz
push h
mov c,a
call conout
pop h
inx h
jmp prmsg

setfunc:
set function for next i/o (command in reg-c)
lxi h,iof ; io function address
mov a,m ; get it to accumulator for maskin
ani 111110b0b ; remove previous command
ora c ; set to new command
mov m,a ; replaced in iopb

; the mds-800 controller req's disk bank bit in sec
; mask the bit from the current i/o function
ani 00100000b ; mask the disk select bit
lxi h,ios ; address the sector selec
ora m ; select proper disk bank
mov m,a ; set disk select bit on/o
ret

waitio:
mvi c,retry ; max retries before perm error

rewait:

mov l,c
mov h,b
shld iod
ret

; read:
mov c,readf ; set to read function
call setfunc
call waitio ; perform read function
ret ; may have error set in reg-a

; write:
mov c,writf ; set to write function
call setfunc
call waitio
ret ; may have error set

; utility subroutines

prmsg: ; print message at h,l to 0
mov a,m
ora a ; zero?
rz
push h
mov c,a
call conout
pop h
inx h
jmp prmsg

setfunc:
set function for next i/o (command in reg-c)
lxi h,iof ; io function address
mov a,m ; get it to accumulator for maskin
ani 111110b0b ; remove previous command
ora c ; set to new command
mov m,a ; replaced in iopb

; the mds-800 controller req's disk bank bit in sec
; mask the bit from the current i/o function
ani 00100000b ; mask the disk select bit
lxi h,ios ; address the sector selec
ora m ; select proper disk bank
mov m,a ; set disk select bit on/o
ret

waitio:
mvi c,retry ; max retries before perm error

rewait:

mov l,c
mov h,b
shld iod
ret
ora a, a, iopb and 0ffh; low address for iopb
mvi a, iopb and 0ffh; low address for iopb
mvi b, iopb shr 8; high address for iopb
jnz iodrl; drive bank 1?
out ilow; low address to controller
mov a, b
out ilow; low address to controller
jmp wait0; to wait for complete
iodrl:; drive bank 1
out ilow+10h; 88 for drive bank 10
mov a, b
out ilow+10h
wait0:; wait for completion
call instat
ani iordy; ready?
call instat; wait for completion
mov a, b
out ihigh+10h
; check io completion ok
call intype; must be io complete (00)
00 unlinked i/o complete, 01 linked i/o complete
10 disk status changed 11 (not used)
cpi l0b; ready status change?
call instat
ani iordy; ready?
call intype; must be 00 in the accumulator
ora a
jnz werror; some other condition, re
ja
; check i/o error bits
call instat
ral
jc wready; unit not ready
rar
ani 1111110b; any other errors?
call instat
jnz werror
;read or write is ok, accumulator contains zero
ret
; not ready, treat as error for now
jnz werror; return hardware malfunction (crc, track, seek, e
the mds controller has returned a bit in each pos
of the accumulator, corresponding to the conditio
0 - deleted data (accepted as ok above)
1 - crc error
2 - seek error
3 - address error (hardware malfunction)
4 - data over/under flow (hardware malfunction)
5 - write protect (treated as not ready)
6 - write error (hardware malfunction)
7 - not ready
(accumulator bits are numbered 7 6 5 4 3 2 1 0)

it may be useful to filter out the various conditions, but we will get a permanent error message if it is uncorrectable. In any case, the not ready condition is handled separately. For trycount:

register c contains retry count, decrement 'til zero

```
4c38 0d  dcr   c
4c39 c2f24b  jnz  rewait ; for another try
```

cannot recover from error

```
4c3c 3e01  mvi   a, 1 ; error code
4c3e c9  ret
```

intype, inbyte, instat read drive bank 00 or 10

```
4c3f 3a664c  intype:  lda   dbank
4c42 b7  ora   a
4c43 c2494c  jnz  intypel ; skip to bank 10
4c46 db79  in   rtype
4c48 c9  ret
4c49 db89  intypel:  in   rtype+10h  ; 78 for 0,1  88 for 2,3
4c4b c9  ret
```

```
4c4c 3a664c  inbyte:  lda   dbank
4c4f b7  ora   a
4c50 c2564c  jnz  inbytl
4c53 db7b  in   rbyte
4c55 c9  ret
4c56 db8b  inbytl:  in   rbyte+10h
4c58 c9  ret
```

```
4c59 3a664c  instat:  lda   dbank
4c5c b7  ora   a
4c5d c2634c  jnz  instal
4c60 db78  in   dstat
4c62 c9  ret
4c63 db88  instal:  in   dstat+10h
4c65 c9  ret
```

data areas (must be in ram)

```
4c66 00  dbank:  db   0  ; disk bank 00 if drive 0,1
; 10 if drive 2,3
4c67 80  db   80h  ; normal i/o operation
4c68 04  iof:  db   readf  ; i/o function, initial read
4c69 01  ion:  db   l  ; number of sectors to read
4c6a 02  iot:  db   offset  ; track number
4c6b 01  ios:  db   l  ; sector number
4c6c 8000  iod:  dw   buff  ; i/o address
```

define ram areas for bdos operation
endef
begdat equ $  ;directory access buffer

dirbuf: ds 128
alv0: ds 31
csv0: ds 16
alv1: ds 31
csv1: ds 16
alv2: ds 31
csv2: ds 16
alv3: ds 31
csv3: ds 16
enddat equ $
datsiz equ $-begdat
end
APPENDIX C: A SKELETAL CBIOS

; skeletal cbios for first level of cp/m 2.0 altera

0014 =
msize equ   20 ;cp/m version memory size in kilo

; "bias" is address offset from 3400h for memory sy
; than 16k (referred to as "b" throughout the text)

0000 =
bias equ (msize-20)*1024

3400 =
ccp equ 3400h+bias ;base of ccp

3C06 =
bdos equ ccp+806h ;base of bdos

4A00 =
bios equ ccp+1600h ;base of bios

0004 =
cdisk equ 0004h ;current disk number 0=a,...,15=p

0003 =
iobyte equ 0003h ;intel i/o byte

4a00 =
org bios ;origin of this program

002c =
nsects equ ($-ccp)/128 ;warm start sector count

; jump vector for individual subroutines

4a00 c39c4a
jmp boot ;cold start

4a03 c3a64a wboote: jmp wboot ;warm start

4a06 c3114b jmp const ;console status

4a09 c3244b jmp conin ;console character in

4a0c c3374b jmp conout ;console character out

4a0f c3494b jmp list ;list character out

4a12 c34d4b jmp punch ;punch character out

4a15 c34f4b jmp reader ;reader character out

4a18 c3544b jmp home ;move head to home position

4a1b c35a4b jmp seldsk ;select disk

4a1e c37d4b jmp settrk ;set track number

4a21 c3924b jmp setsec ;set sector number

4a24 c3ad4b jmp setdma ;set dma address

4a27 c3c34b jmp read ;read disk

4a2a c3d64b jmp write ;write disk

4a2d c34b4b jmp listst ;return list status

4a30 c3a74b jmp sectran ;sector translate

; fixed data tables for four-drive standard
; ibm-compatible 8" disks

disk parameter header for disk 00

4a33 734a00 dpbase: dw trans,0000h
4a37 000000 dw 0000h,0000h
4a3b f04c8d dw dirbf,dpblk
4a3f ec4d70 dw chk00,all00

disk parameter header for disk 01

4a43 734a00 dw trans,0000h
4a47 000000 dw 0000h,0000h
4a4b f04c8d dw dirbf,dpblk
4a4f fc4d8f dw chk01,all01

disk parameter header for disk 02

4a53 734a00 dw trans,0000h
4a57 000000 dw 0000h,0000h
4a5b f04c8d dw dirbf,dpblk
4a5f 0c4eae dw chk02,all02
; disk parameter header for disk 03
4a63 734a00          dw trans,0000h
4a67 000000          dw 0000h,0000h
4a6b f04c8d          dw dirbf,dpblk
4a6f 1c4ecd          dw chk03,all03

; sector translate vector
4a73 10700d          trans:   db 1,7,13,19 ;sectors 1,2,3,4
4a77 10500b          db 25,5,11,17 ;sectors 5,6,7,8
4a7b 170309          db 23,3,9,15 ;sectors 9,10,11,12
4a7f 150208          db 21,2,8,14 ;sectors 13,14,15,16
4a83 141a06          db 20,26,6,12 ;sectors 17,18,19,20
4a87 121804          db 18,24,4,10 ;sectors 21,22,23,24
4a8b 1016            db 16,22 ;sectors 25,26

; dpblk: disk parameter block, common to all disks
4a8d 1a00            dw 26 ;sectors per track
4a8f 03             db 3 ;block shift factor
4a90 07             db 7 ;block mask
4a91 00             db 0 ;null mask
4a92 f200             dw 242 ;disk size-l
4a94 3f00             dw 63 ;directory max
4a96 c0             db 192 ;alloc 0
4a97 00             db 0 ;alloc 1
4a98 1000            dw 16 ;check size
4a9a 0200            dw 2 ;track offset

; end of fixed tables

; individual subroutines to perform each function
boot: simplest case is to just perform parameter initia
4a9c af             xra a ;zero in the accum
4a9d 320300           sta iobyte ;clear the iobyte
4aa0 320400           sta cdisk ;select disk zero
4aa3 c3ef4a           jmp gocpm ;initialize and go to cp/

wboot: simplest case is to read the disk until all sect
4aa6 318000           lxi sp,80h ;use space below buffer f
4aa9 0e00            mvi c,0 ;select disk 0
4aab cd5a4b          call seldsk
4aae cd544b          call home ;go to track 00

4ab1 062c            mvi b,nsects ;b counts # of sectors to
4ab3 0e00            mvi c,0 ;c has the current track
4ab5 1602            mvi d,2 ;d has the next sector to
 ; note that we begin by reading track 0, sector 2 s ; contains the cold start loader, which is skipped
4ab7 210034           lxi h,ccp ;base of cp/m (initial 10

load1: load one more sector
4aba c5             push b ;save sector count, current track
4abb d5             push d ;save next sector to read
4abc e5             push h ;save dma address
4abd 4a             mov c,d ;get sector address to register c
4abe cd924b          call setsec ;set sector address from register
4acl cl             pop b ;recall dma address to b,c

227
; replace on stack for later recall
push b
; drive set to 0, track set, sector set, dma address
; call setdma ; set dma address from b,c
call setdma
; call read
; cpi 00h ; any errors?
cpi 00h
; no error, move to next sector
; pop h ; recall dma address
pop h
; new dma address is in h,l
; dad d
; recall sector address
; cpi 00h
; pop d
; recall number of sectors remaini
; pop b
; sectors=sectors-l
; jnz wboot ; retry the entire boot if an erro
jnz wboot
; more sectors remain to load, check for track chan
; inr d
; sector=27?, if so, change tracks
; mov a,d
; new dma address is in h,l
; dad d
; recall sector address
; cpi 00h
; pop d
; recall number of sectors remaini
; pop b
; sectors=sectors-l
; jc loadl ; carry generated if sector<27
jc loadl
; end of current track, go to next track
; mvi d,l ; begin with first sector of next
mvi d,l
; save register state, and change tracks
; pop c
; track=track+1
; jmp loadl ; for another sector
; end of load operation, set parameters and go to c
jmp loadl

; entry point
mvi a,0c3h ;c3 is a jmp instruction
; for jmp to wboot
mvi a,0c3h
sta 0
; entry point
sta 0
; wboot entry point
lxi h,wboot
; address field for jmp at 0
lxi h,wboot
sta 0
; bdos entry point
lxi h,bdos
sta 0
; address field of jump at 5 to bd
lxi h,bdos
sta 0
; default dma address is 80h
lxi b,80h
; send to the ccp
; mov c,a
; go to cp/m for further processin
mov c,a
; simple i/o handlers (must be filled in by user)
; in each case, the entry point is provided, with space to insert your own code

; const: ;console status, return 0ffh if character ready,
ds 10h ;space for status subroutine
4b21 3e00
mvi a,00h
4b23 c9
ret

; conin: ;console character into register a
ds 10h ;space for input routine
4b34 e67f
ani 7fh ;strip parity bit
4b36 c9
ret

; conout: ;console character output from register c
mov a,c ;get to accumulator
ds 10h ;space for output routine
4b37 79
4b38 ret
4b48 c9

; list: ;list character from register c
mov a,c ;character to register a
ret ;null subroutine
4b49 79
4b4a c9

; listst: ;return list status (0 if not ready, 1 if ready)
xra a ;0 is always ok to return
4b4b af
4b4c c9
ret

; punch: ;punch character from register c
mov a,c ;character to register a
ret ;null subroutine
4b4d 79
4b4e c9

; reader: ;read character into register a from reader device
mvi a,lah ;enter end of file for now (replace)
4b51 e67f
ani 7fh ;remember to strip parity bit
4b53 c9
ret

; i/o drivers for the disk follow
; for now, we will simply store the parameters away
; in the read and write subroutines

; home: ;move to the track 00 position of current drive
; translate this call into a settrk call with param
mov c,0 ;select track 0
4b5a 0e00
mvi
4b56 cd7d4b
call settrk
4b59 c9
ret ;we will move to 00 on first read

; seldsk: ;select disk given by register c
4b54 0e00
lxi h,0000h ;error return code
4b56 cd7d4b
mov a,c
4b5e 32ef4c
sta diskno
4b61 fe04
cpi 4 ;must be between 0 and 3
4b63 d0
   ; disk number is in the proper range
4b64   ds   l0   ; space for disk select
   ; compute proper disk parameter header address
4b6e 3ae4fc
4b71 6f
   mov 1,a        ; l = disk number 0,1,2,3
4b72 2600
   mvi h,0       ; high order zero
4b74 29
4b75 29
4b76 29
4b77 29
   dad  h        ; *2
4b78 11334a
4b79 19
   dad  h        ; *4
4b7c 19
   dad  h        ; *8
4b7c 29
   dad  h        ; *16 (size of each header)
4b7c 19
4b7c 1a
   lxi  d,dpbase
4b7c 19
4b7c 19
   dad  d        ; hl = .dpbase(diskno*16)
4b7c c9
   ret

   ; settrack: ; set track given by register c
4b7d 79
   mov  a,c
4b7e 32e94c
   sta  track
4b81   ds   l0h   ; space for track select
4b81 c9
   ret

   ; setsec: ; set sector given by register c
4b82 79
   mov  a,c
4b83 32eb4c
   sta  sector
4b86   ds   l0h   ; space for sector select
4b86 c9
   ret

   ; sectran:
   ; translate the sector given by bc using the
   ; translate table given by de
4bac c9
4bad 69
4bae 60
4baf 22ed4c
4bb2   ds   l0h   ; space for setting the dma address
4bb2 c9
   ret

   ; setdma: ; set dma address given by registers b and c
4bc2 c9
   jmp  waitio   ; to perform the actual i/o

4bc3   ds   l0h   ; set up read command
4bd3 c3e64b
   mov  h,b       ; high order address
4baf 22ed4c
   shld  dmaad    ; save the address
4bb2   ds   l0h   ; space for setting the dma address
4bb2 c9
   ret

   ; read: ; perform read operation (usually this is similar
   ; so we will allow space to set up read command, th
   ; common code in write)
4bc3   ds   l0h   ; set up read command
4bd3 c3e64b
   jmp  waitio   ; to perform the actual i/o

   ; write: ; perform a write operation
4bd6   ds   l0h   ; set up write command

   ; waitio: ; enter here from read and write to perform the ac
   ; operation. return a 00h in register a if the ope
   ; properly, and 01h if an error occurs during the r
in this case, we have saved the disk number in 'diskno' (8-15)
the track number in 'track' (0-76)
the sector number in 'sector' (1-8)
the dma address in 'dmaad' (0-655)

4be6 ds 256 ; space reserved for i/o drivers
4ce6 3e01 mvi a,1 ; error condition
4ce8 c9 ret ; replaced when filled-in

the remainder of the cbios is reserved uninitiali
data area, and does not need to be a part of the
system memory image (the space must be available, however, between "begdat" and "enddat").

4ce9 track: ds 2 ; two bytes for expansion
4ceb sector: ds 2 ; two bytes for expansion
4ced dmaad: ds 2 ; direct memory address
4cef diskno: ds 1 ; disk number 0-15

scratch ram area for bdos use
4cf0 = begdat equ $ ; beginning of data area
4cf0 dirbf: ds 128 ; scratch directory area
4d70 all00: ds 31 ; allocation vector 0
4d8f all01: ds 31 ; allocation vector 1
4dae all02: ds 31 ; allocation vector 2
4dcd all03: ds 31 ; allocation vector 3
4dec chk00: ds 16 ; check vector 0
4dfc chk01: ds 16 ; check vector 1
4e0c chk02: ds 16 ; check vector 2
4e1c chk03: ds 16 ; check vector 3

4e2c = enddat equ $ ; end of data area
013c = datsiz equ $-begdat ; size of data area
4e2c end
APPENDIX D: A SKELETAL GETSYS/PUTSYS PROGRAM

; combined getsys and putsys programs from Sec 4.
; Start the programs at the base of the TPA

0100  org 0100h

0014 = msize equ 20         ; size of cp/m in Kbytes

; "bias" is the amount to add to addresses for > 20k
; (referred to as "b" throughout the text)

0000 = bias equ (msize-20)*1024
3400 = ccp equ 3400h+bias
3c00 = bdos equ ccp+0000h
4a00 = bios equ ccp+1600h

; getsys programs tracks 0 and 1 to memory at
; 3880h + bias

; register usage
; a  (scratch register)
; b  track count (0...76)
; c  sector count (1...26)
; d,e (scratch register pair)
; h,l load address
; sp set to stack address

gstart:

0100 318033 lxi sp,ccp-0080h ; start of getsys
0103 218033 lxi h,ccp-0080h ; convenient plac
0106 0600 mvi b,0 ; set initial loa
rdstrk:
0108 0e01 mvi c,1 ; start with trac
rdsect:
010a cd0003 call read$sec ; each track star
010d 118000 lxi d,128 ; get the next se
0110 19 dad d ; offset by one s
0111 0c inr c ; (hl=hl+128)
0112 79 mov a,c ; fetch sector nu
0113 f6b cpi 27 ; next sector
0115 da0a01 jc rd$sec ; and see if la

; arrive here at end of track, move to next track

0118 04 inr b ; track = track+1
0119 78 mov a,b ; check for last
011a fe02 cpi 2 ; track = 2 ?
011c da0801 jc rd$trk ; <, do another

; arrive here at end of load, halt for lack of anything b

011f fb ei
0120 76 hlt
; putsys program, places memory image starting at
; 3800h + bias back to tracks 0 and 1
; start this program at the next page boundary

0200 org ($+0100h) and 0ff00h

put$sys:
0200 318033 lxi sp, ccp-0080h ; convenient plac
0203 218033 lxi h, ccp-0000h ; start of dump
0206 0600 mvi b, 0 ; start with trac

wr$trk:
0208 0e01 mvi c, 1 ; start with sect

wr$sec:
020a cd0004 call write$sec ; write one secto
020d 118000 lxi d, 128 ; length of each
0210 19 dad d ; <hl>=<hl> + 128
0211 0c inr c ; <c> = <c> + 1
0212 79 mov a, c ; see if
0213 felb cpi 27 ; past end of t
0215 da0a02 jc wr$sec ; no, do another

; arrive here at end of track, move to next track
0218 04 inr b ; track = track+1
0219 78 mov a, b ; see if
021a fe02 cpi 2 ; last track
021c da0802 jc wr$trk ; no, do another

; done with putsys, halt for lack of anything bette
021f fb ei
0220 76 hlt

; user supplied subroutines for sector read and write

; move to next page boundary
0300 org ($+0100h) and 0ff00h

read$sec:
; read the next sector
; track in <b>,
; sector in <c>
; dmaaddr in <hl>

0300 c5 push b
0301 e5 push h

; user defined read operation goes here
0302 ds 64

0342 e1 pop h
0343 cl pop b
org ($+0100h) and 0ff00h ; another page bo

write$sec:

; same parameters as read$sec

push b
push h

; user defined write operation goes here

ds 64

pop h
pop b
ret

; end of getsys/putsys program

end
APPENDIX E: A SKELETAL COLD START LOADER

; this is a sample cold start loader which, when modified
; resides on track 00, sector 01 (the first sector on the
; diskette). we assume that the controller has loaded
; this sector into memory upon system start-up (this pro-
; gram can be keyed-in, or can exist in read/only memory
; beyond the address space of the cp/m version you are
; running). the cold start loader brings the cp/m system
; into memory at "loadp" (3400h + "bias"). in a 20k
; memory system, the value of "bias" is 0000h, with large
; values for increased memory sizes (see section 2). afte
; loading the cp/m system, the cold start loader branches
; to the "boot" entry point of the bios, which begins at
; "bios" + "bias." the cold start loader is not used un-
; till the system is powered up again, as long as the bios
; is not overwritten. the origin is assumed at 0000h, an
; must be changed if the controller brings the cold start
; loader into another area, or if a read/only memory area
; is used.

 movements

0000 org 0 ; base of ram in cp/m
0014 = msize equ 20 ; min mem size in kbytes
0000 = bias equ (msize-20)*1024 ; offset from 20k system
3400 = ccp equ 3400h+bias ; base of the ccp
4a00 = bios equ ccp+1600h ; base of the bios
0300 = biosl equ 0300h ; length of the bios
4a00 = boot equ bios
1900 = size equ bios+biosl-ccp ; size of cp/m system
0032 = sects equ size/128 ; # of sectors to load

; begin the load operation

cold:

0000 010200 lxi b,2 ; b=0, c=sector 2
0003 1632 mvi d,sects ; d=# sectors to load
0005 210034 lxi h,ccp ; base transfer address

lsect: ; load the next sector

; insert inline code at this point to
; read one 128 byte sector from the
; track given in register b, sector
; given in register c,
; into the address given by <hl>
;
; branch to location "cold" if a read error occurs
paste$patch:
; go to next sector if load is incomplete
006b 15   dcr    d        ; sects=sects-1
006c ca004a jz     boot     ; head for the bios

; more sectors to load

; we aren't using a stack, so use <sp> as scratch register
; to hold the load address increment

006f 318000  lxi    sp,128    ; 128 bytes per sector
0072 39     dad    sp        ; <hl> = <hl> + 128
0073 0c     inr    c        ; sector = sector + 1
0074 79     mov    a,c
0075 felb   cpi    27       ; last sector of track?
0077 da0800 jc     lsect    ; no, go read another

; end of track, increment to next track

007a 0e01   mvi    c,1       ; sector = 1
007c 04     inr    b        ; track = track + 1
007d c30800 jmp    lsect    ; for another group
0080
end
CP/M 2.0 disk re-definition library

Copyright (c) 1979
Digital Research
Box 579
Pacific Grove, CA
93950

CP/M logical disk drives are defined using the macros given below, where the sequence of calls is:

disks n
diskdef parameter-list-0
diskdef parameter-list-1
...
diskdef parameter-list-n
endef

where n is the number of logical disk drives attached to the CP/M system, and parameter-list-i defines the characteristics of the ith drive (i=0,1,...,n-1)

each parameter-list-i takes the form
dn,fsc,lsc,[skf],bls,dks,dir,cks,ofs,[0]

where
dn is the disk number 0,1,...,n-1
fsc is the first sector number (usually 0 or 1)
lsc is the last sector number on a track
skf is optional "skew factor" for sector translate
bls is the data block size (1024,2048,...,16384)
dks is the disk size in bls increments (word)
dir is the number of directory elements (word)
cks is the number of dir elements to checksum
ofs is the number of tracks to skip (word)
[0] is an optional 0 which forces 16K/directory en

for convenience, the form
dn,dm

defines disk dn as having the same characteristics as a previously defined disk dm.

a standard four drive CP/M system is defined by
disks 4
diskdef 0,1,26,6,1024,243,64,64,2
dsk set 0
rept 3
dsk set dsk+1
diskdef &dsk,0
endm
endef

the value of "begdat" at the end of assembly defines t
beginning of the uninitialized ram area above the bios,
while the value of "enddat" defines the next location
following the end of the data area, the size of this
area is given by the value of "datsiz" at the end of t
assembly, note that the allocation vector will be qui
large if a large disk size is defined with a small blo
size.

52: ;
53: ;
54: dskhdr macro dn
55: ; define a single disk header list
56: dpe&dn: dw xlt&dn,0000h ;translate table
57: dw 0000h,0000h ;scratch area
58: dw dirm,dpb&dn ;dir buff,parm block
59: dw csv&dn,alv&dn ;check, alloc vectors
60: endm
61: ;
62: disks macro nd
63: ; define nd disks
64: ndisks set nd ;for later reference
65: dpbase equ $ ;base of disk parameter blocks
66: ; generate the nd elements
67: dsknxt set 0
68: rept nd
69: dskhdr &dsknxt
70: dsknxt set dsknxt+1
71: endm
72: endm
73: ;
74: dpbhdr macro dn
75: dpb&dn equ $ ;disk parm block
76: endm
77: ;
78: ddb macro data,comment
79: db data comment
80: endm
81: ;
82: ddw macro data,comment
83: dw data comment
84: endm
85: ;
86: gcd macro m,n
87: ; greatest common divisor of m,n
88: ; produces value gcdn as result
89: ; (used in sector translate table generation)
90: gcdm set m ;variable for m
91: gcdn set n ;variable for n
92: gcdr set 0 ;variable for r
93: rept 65535
94: gcdx set gcdm/gcdn
95: gcdr set gcdm - gcdx*gcdn
96: if gcdr = 0
97: endif
109: gcdm set gcdn
110: gcdn set gcdr
111: endm
112: endm
113: ;
114: diskdef macro dn,fsc,lsc,skf,bls,dks;dir,cks,bfs,k16
115: ; generate the set statements for later tables
116: if nul lsc
117: ; current disk dn same as previous fsc
118: dpb&dn equ dpb&fsc ;equivalent parameters
119: als&dn equ als&fsc ;same allocation vector size
120: css&dn equ css&fsc ;same checksum vector size
121: xlt&dn equ xlt&fsc ;same translate table
122: else
123: seccmax set lsc-(fsc) ;;sectors 0...seccmax
124: sectors set seccmax+1; number of sectors
125: als&dn set (dks)/8 ;;size of allocation vector
126: if ((dks) mod 8) ne 0
127: als&dn set als&dn+1
128: endif
129: css&dn set (cks)/4 ;;number of checksum elements
130: ; generate the block shift value
131: blkval set bls/128 ;;number of sectors/block
132: blkshf set 0 ;;counts right 0's in blkval
133: blkmsk set 0 ;;fills with 1's from right
134: rept 16 ;;once for each bit position
135: if blkval=1
136: exitm
137: endif
138: ; otherwise, high order 1 not found yet
139: blkshf set blkshf+1
140: blkmsk set (blkmsk shl 1) or 1
141: blkval set blkval/2
142: endm
143: ; generate the extent mask byte
144: blkval set bls/1024 ;;number of kilobytes/block
145: extmsk set 0 ;;fill from right with 1's
146: rept 16
147: if blkval=1
148: exitm
149: endif
150: ; otherwise more to shift
151: extmsk set (extmsk shl 1) or 1
152: blkval set blkval/2
153: endm
154: ; may be double byte allocation
155: if (dks) > 256
156: extmsk set (extmsk shr 1)
157: endif
158: ; may be optional [0] in last position
159: if not nul k16
160: extmsk set k16
161: endif
162: ; now generate directory reservation bit vector
163: dirrem set dir ;# remaining to process
164: dirbks set bks/32 ;;number of entries per block
165: dirblk set 0 ;;fill with 1's on each loop
166: rept 16
167: if dirrem=0
168: endif
170: ;; not complete, iterate once again
171: ;; shift right and add 1 high order bit
172: dirblk set (dirblk shr i) or 8000h
173: if dirrem > dirbks
174: dirrem set dirrem-dirbks
175: else
176: dirrem set 0
177: endif
178: endm
179: dpbhdr dn ;;generate equ $
180: ddw %sectors,<;sec per track>
181: ddb %blkshf,<;block shift>
182: ddb %blkmsk,<;block mask>
183: ddb %extmsk,<;extnt mask>
184: ddw %dks-1,<;disk size-1>
185: ddw %dir-1,<;directory max>
186: ddb %dirblk shr 8,<;alloc0>
187: ddb %dirblk and 0ffh,<;alloc1>
188: ddw %cks/4,<;check size>
189: ddw %ofs,<;offset>
190: ;; generate the translate table, if requested
191: if nul skf
192: xlt&dn equ 0 ;;no xlate table
193: else
194: if skf = 0
195: xlt&dn equ 0 ;;no xlate table
196: else
197: ;; generate the translate table
198: nxtsec set 0 ;;next sector to fill
199: nxtbas set 0 ;;moves by one on overflow
200: gcd %sectors,skf
201: ;; gcdn = gcd(sectors,skew)
202: neltst set sectors/gcdn
203: ;; neltst is number of elements to generate
204: ;; before we overlap previous elements
205: nelts set neltst ;;counter
206: xlt&dn equ $ ;;translate table
207: rept sectors ;;once for each sector
208: if sectors < 256
209: ddb %nxtsec+(fsc)
210: else
211: ddw %nxtsec+(fsc)
212: endif
213: nxtsec set nxtsec+(skf)
214: if nxtsec >= sectors
215: nxtsec set nxtsec-sectors
216: endif
217: nelts set nelts-1
218: if nelts = 0

240
219: nxbas  set  nxbas+1
220: nxssec set  nxbas
221: neltst set  neltst
222: endif
223: endm
224: endif  ;;end of nul fac test
225: endif  ;;end of nul bls test
226: endm
227:
228: defds macro lab,space
229: lab:  ds  space
230: endm
231:
232: lds macro lb,dn,val
233: defds lb&dn,%val&dn
234: endm
235:
236: endif macro
237: ;; generate the necessary ram data areas
238: begdat equ $      
239: ddirbuf: ds 128   ;directory access buffer
240: dsknxt set 0      
241: rept ndisks ;; once for each disk
242: lds alv,%dsknxt,als
243: lds csv,%dsknxt,css
244: dsknxt set dsknxt+1
245: endm
246: enddat equ $    
247: datsiz equ $-begdat
248: ;; db 0 at this point forces hex record
249: endm
APPENDIX G: BLOCKING AND DEBLOCKING ALGORITHMS.

1: ;********************************************************************************
2: ;* Sector Deblocking Algorithms for CP/M 2.0 *
3: ;* ********************************************************************************
4: ;* utility macro to compute sector mask *
5: ;*
6: ;
7: ;
8: smask macro hblk
9: ;
10: ;
11: ;
12: ;
13: ;
14: ;
15: ;
16: ;
17: ;
18: ;
19: ;
20: ;
21: ;
22: ;
23: ;
24: ;********************************************************************************
25: ;*
26: ;* CP/M to host disk constants *
27: ;*
28: ;********************************************************************************
29: blksiz equ 2048 ;CP/M allocation size
30: hstsiz equ 512 ;host disk sector size
31: hstspt equ 20 ;host disk sectors/trk
32: hstblk equ hstsiz/128 ;CP/M sects/host buff
33: cpmspt equ hstblk * hstspt ;CP/M sectors/track
34: secmsk equ hstblk-1 ;sector mask
35: smask equ hstblk ;compute sector mask
36: secshf equ @x ;log2(hstblk)
37: ;
38: ;********************************************************************************
39: ;*
40: ;* BDOS constants on entry to write *
41: ;*
42: ;********************************************************************************
43: wrall equ 0 ;write to allocated
44: wrdir equ 1 ;write to directory
45: wural equ 2 ;write to unallocated
46: ;
47: ;********************************************************************************
48: ;*
49: ;* The BDOS entry points given below show the *
50: ;* code which is relevant to deblocking only. *
51: ;*
52: ;********************************************************************************
53: ;
DISKDEF macro, or hand coded tables go here

dpbase equ $ ; disk param block base

; enter here on system boot to initialize
xra a ; 0 to accumulator
sta hstact ; host buffer inactive
sta unacnt ; clear unalloc count
ret

; select disk
mov a,c ; selected disk number
sta sekdisk ; seek disk number
mov l,a ; disk number to HL
mvi h,0
rept 4 ; multiply by 16
dad h
endm
1xi d,dpbase ; base of parm block
dad d ; hl=.dpb(curdsk)
ret

; set track given by registers BC
mov h,b
mov l,c
shld sektrak ; track to seek
ret

; set sector given by register c
mov a,c
sta seksec ; sector to seek
ret

; set dma address given by BC
mov h,b
mov l,c
shld dmaadr
ret

; translate sector number BC
mov h,b
mov l,c
ret
104: ;******************************************************************************
105: ;*
106: ;*   The READ entry point takes the place of *
107: ;*   the previous BIOS definition for READ.  *
108: ;*  
109: ;******************************************************************************
110: ;read:
111: ;read the selected CP/M sector
112: mvi a,l
113: sta readop ;read operation
114: sta rsflag ;must read data
115: mvi a,wrual
116: sta wrtype ;treat as unalloc
117: jmp wrtype ;to perform the read
118: ;
119: ;******************************************************************************
120: ;*
121: ;*   The WRITE entry point takes the place of *
122: ;*   the previous BIOS definition for WRITE.  *
123: ;*  
124: ;******************************************************************************
125: ;write:
126: ;write the selected CP/M sector
127: xra a ;0 to accumulator
128: sta readop ;not a read operation
129: mov a,c ;write type in c
130: sta wrtype
131: cpi wrual ;write unallocated?
132: jnz chkuna ;check for unalloc
133: ;
134: ;write to unallocated, set parameters
135: mvi a,blksziz/128 ;next unalloc recs
136: sta unacnt
137: lda sekdske ;disk to seek
138: sta unadsk ;unadsk = sekdske
139: lhld sektkr
140: shld unatrk ;unatrk = sektkr
141: lda seksec
142: sta unasec ;unasec = seksec
143: ;
144: chkuna:
145: ;check for write to unallocated sector
146: lda unacnt ;any unalloc remain?
147: ora a
148: jz alloc ;skip if not
149: ;
150: ;more unallocated records remain
151: dcr a ;unacnt = unacnt-1
152: sta unacnt
153: lda sekdske ;same disk?
154: lxi h,unadsk
155: cmp m ;sekdske = unadsk?
156: jnz alloc ;skip if not
157: ;
158: ;disks are the same
159:  lxi h,unatrk
160:  call sektrkcmp  ;sektrk = unatrk?
161:  jnz alloc  ;skip if not
162:  ;
163:  ; tracks are the same
164:  lda seksec  ;same sector?
165:  lxi h,unasec
166:  cmp m  ;seksec = unasec?
167:  jnz alloc  ;skip if not
168:  ;
169:  match, move to next sector for future ref
170:  inr m  ;unasec = unasec+1
171:  mov a,m  ;end of track?
172:  cpi cpmspt  ;count CP/M sectors
173:  jc noovf  ;skip if no overflow
174:  ;
175:  ; overflow to next track
176:  mvi m,0  ;unasec = 0
177:  lhlh unatrk
178:  inx h
179:  shld unatrk  ;unatrk = unatrk+1
180:  ;
181:  noovf:
182:  ;match found, mark as unnecessary read
183:  xra a  ;0 to accumulator
184:  sta rsflag  ;rsflag = 0
185:  jmp rwoper  ;to perform the write
186:  ;
187:  alloc:
188:  ;not an unallocated record, requires pre-read
189:  xra a  ;0 to accum
190:  sta unacnt  ;unacnt = 0
191:  inr a  ;1 to accum
192:  sta rsflag  ;rsflag = 1
193:  ;
194:  ;******************************************************************************
195:  ;*
196:  ;* Common code for READ and WRITE follows  *
197:  ;*
198:  ;******************************************************************************
199:  rwoper:
200:  ; enter here to perform the read/write
201:  xra a  ;zero to accum
202:  sta erflag  ;no errors (yet)
203:  lda seksec  ;compute host sector
204:  rept secshf
205:  ora a  ;carry = 0
206:  rar  ;shift right
207:  endm
208:  sta sekhst  ;host sector to seek
209:  ;
210:  ; active host sector?
211:  lxi h,hstact  ;host active flag
212:  mov a,m
213:  mvi m,l  ;always becomes 1
214: ora a ;was it already?
215: jz filhst ;fill host if not
216: ;
217: ; host buffer active, same as seek buffer?
218: lda sekdsk
219: lxi h,hstdsk ;same disk?
220: cmp m ;sekdsk = hstdsk?
221: jnz nomatch
222: ;
223: ; same disk, same track?
224: lxi h,hsttrak
225: call sektrkcmp ;sektrak = hsttrak?
226: jnz nomatch
227: ;
228: ; same disk, same track, same buffer?
229: lda sekhst
230: lxi h,hstsec ;sekhst = hstsec?
231: cmp m
232: jz match ;skip if match
233: ;
234: nomatch:
235: ;proper disk, but not correct sector
236: lda hstwrt ;host written?
237: ora a
238: cnz writehst ;clear host buff
239: ;
240: filhst:
241: ;may have to fill the host buffer
242: lda sekdsk
243: sta hstdsk
244: lhld sektrak
245: shld hsttrak
246: lda sekhst
247: sta hstsec
248: lda rsflag ;need to read?
249: ora a
250: cnz readhst ;yes, if 1
251: xra a ;0 to accum
252: sta hstwrt ;no pending write
253: ;
254: match:
255: ;copy data to or from buffer
256: lda seksec ;mask buffer number
257: ani secmsk ;least signif bits
258: mov l,a ;ready to shift
259: mvi h,0 ;double count
260: rept 7 ;shift left 7
261: dad h
262: endm
263: ; hl has relative host buffer address
264: lxi d,hstbuf
265: dad d ;hl = host address
266: xchg ;now in DE
267: lhld dmaadr ;get/put CP/M data
268: mvi c,128 ;length of move
lda readop ;which way?
ora a
jnz rwmv ;skip if read

write operation, mark and switch direction
mvi a,l
sta hstwrt ;hstwrt = 1
xchg ;source/dest swap

rwmv:
;C initially 128, DE is source, HL is dest
ldax d ;source character
inx d
mov m,a ;to dest
inx h
dcr c ;loop 128 times
jnz rwmv

data has been moved to/from host buffer
lda wrtype ;write type
cpi wrdir ;to directory?
lda erflag ;in case of errors
rnz ;no further processing

clear host buffer for directory write
ora a ;errors?
rnz ;skip if so
xra a ;0 to accum
sta hstwrt ;buffer written
call writehst
lda erflag
ret

;***************************************************** *
; Utility subroutine for 16-bit compare *
;***************************************************** *
sektrkcmp:
;HL = .unatr or .hstrk, compare with sektrk
xchg
lxi h,sektrk
ldax d ;low byte compare
cmp m ;same?
rnz ;return if not

low bytes equal, test high ls
inx d
inx h
ldax d
cmp m ;sets flags
ret
**WRITEHST performs the physical write to the host disk, READHST reads the physical disk.**

WRITEHST:

```
; hstdsk = host disk #, hsttrak = host track #,
; hstsec = host sect #. write "hstsiz" bytes

; from hstbuf and return error flag in erflag.
; return erflag non-zero if error

ret
```

READHST:

```
; hstdsk = host disk #, hsttrak = host track #,
; hstsec = host sect #. read "hstsiz" bytes
; into hstbuf and return error flag in erflag.

ret
```

**Uninitialized RAM data areas**

```
; seek disk number
; seek track number
; seek sector number

; host disk number
; host track number
; host sector number

; seek shr secshf
; host active flag
; host written flag

; unalloc rec cnt
; last unalloc disk
; last unalloc track
; last unalloc sector

; error reporting
; read sector flag
; l if read operation
; write operation type
; last dma address
; host buffer
```
EXIDY SYSTEMS' CBIOS USERS GUIDE

VERSION 1.0
FOR
CP/M 2.2

COPYRIGHT (C) 1981
EXIDY SYSTEMS, INC.

AUGUST 1981
COPYRIGHT

Copyright (c) 1980 by Exidy Systems, Inc. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of Exidy Systems, Inc., 1234 Elko Drive, Sunnyvale, California 94086.

Since this manual is tutorial in nature, permission is granted to reproduce or abstract the example procedures and sample programs for the purposes of inclusion within the reader's programs.

DISCLAIMER

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

TRADEMARKS

CP/M is a registered trademark of Digital Research.
COPYRIGHT

Copyright (c) 1980 by Exidy Systems, Inc. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of Exidy Systems, Inc., 1234 Elko Drive, Sunnyvale, California 94086.

Since this manual is tutorial in nature, permission is granted to reproduce or abstract the example procedures and sample programs for the purposes of inclusion within the reader's programs.

DISCLAIMER

Exidy Systems, Inc. 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, Exidy Systems, Inc. reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Exidy Systems, Inc. to notify any person of such revision or changes.

TRADEMARKS

CP/M is a registered trademark of Digital Research.
Table Of Contents

Exidy Systems' CBios User's Guide:
Version 1.0 For CP/M 2.2

1. Introduction..............................249
2. Configuration and System Generation......251
   A. Hardware for the DDS and the FDS......251
   B. System Generation.......................252
   C. Options................................253
   D. Incompatibilities......................254
   E. Sector Skew Pattern....................254
   F. Special Video Display Interface.......255
3. Features..................................256
   A. Error Recovery.........................256
   B. CP/M Programming......................257
4. Error Messages............................259
1. INTRODUCTION

The CBIOS program provides the interface between CP/M (TM) and the Sorcerer's hardware. The term BIOS was coined by Digital Research, the creators of CP/M. Their term CBIOS stands for Customized Basic Input/Output System, in which the BIOS is customized to the user's hardware. The Exidy program is a CBIOS in this sense.

Portability is the most valuable attribute of CP/M. The clean separation of logical and physical I/O enables it to run on many 8080/Z80 based systems. Digital Research provides the logical I/O in CP/M's Basic Disk Operating System (BDOS). This routes all physical I/O through a BIOS vector. See the "CP/M System Alteration Guide" for a description of the BIOS vector and its functions.

Each hardware system CP/M is used on requires a separate BIOS program. The Exidy interface is tailored for the Sorcerer and the hardware the Sorcerer supports. Because the disk drive is the most complicated piece of hardware CP/M uses, this document focuses on the CBIOS disk interface for the Display Disk System (DDS) double sided drives for soft-sectored 77 track diskettes and the Floppy Disk System (FDS) single sided drives for soft-sectored 77 track diskettes.

For clarification, a brief explanation of physical and logical units may benefit the user in our discussion. A physical unit is the actual input/output device and its hardware recording medium. A diskette, for example, physically has 77 tracks, each of which has 16 sectors of 256-bytes. However, an interface between the physical hardware and the user may translate or break up this size into any number of combinations. This level, used by the programmer, is the logical level. That is, the very same diskette may be dealt with by the programmer as a different size from the physical size (say, 77 tracks each of which has 32 sectors of 128-bytes). The software interface makes all necessary adjustments for the input to be understood on the differing physical diskette size.

The Digital Research interface of the BDOS to the BIOS defines the logical CP/M diskette with a logical sector size of 128-bytes. The Exidy physical diskette systems, however, have a 256-byte sector. To compensate for this difference, Exidy splits each physical 256-byte sector in half to form two CP/M logical 128-byte sectors. The CBIOS is responsible for mapping 128-byte logical CP/M sectors to the proper half of a 256-byte physical sector. The physical sectors are skewed or interleaved on the diskette to minimize rotational delay. This skewing pattern is described in detail later.
The disk buffer cache improves performance by buffering reads and writes of the disk in a RAM cache storage area. When CP/M requests a 128-byte sector read from a sector not within the cache buffer, a 256-byte sector from disk must be read. Thus, the cache returns the requested sector to the user, keeping track of the other 128-byte sector half within the 256-byte cache buffer. Should a read request be made for that sector at this point, no disk I/O is required because the sector already exists in the memory cache. This same principle applies to cache buffer writing. That is, only one 256-byte physical sector I/O is written for two CP/M 128-byte logical I/O requests on the same physical sector.
2. CONFIGURATION AND SYSTEM GENERATION

A. Hardware for the DDS and the FDS

The Exidy CBIOS runs on both the Display Disk System (DDS) and the Floppy Disk Subsystem (FDS). A DDS consists of a Sorcerer II Computer (with keyboard), Display Disk Unit containing a video screen, and two soft sectored Micropolis drives. The Exidy CBIOS may actually support three disk drives connected to the soft sectored disk controller. However, a controller and only two drives are supplied with the DDS. The DDS may be augmented by other peripherals such as a printer, cassettes, etc.. The Exidy CBIOS assigns the logical CP/M devices, Punch and Reader, to serial write and serial read respectively. The List device is assigned to the Sorcerer Centronics parallel printer interface.

On cold boot, CP/M on the DDS outputs one of the following messages:

CP/M on the Exidy Sorcerer for 77 Track Disk
77 Track Single Sided Disks
32K CP/M VERS 2.2/1F
Copyright (c) 1981 Exidy Systems, Inc.
A>

or

CP/M on the Exidy System 80
77 Track Double Sided Disks
32K CP/M VERS 2.2/2F
Copyright (c) 1981 Exidy Systems, Inc.
A>

The Exidy CBIOS also runs on the Floppy Disk System (FDS). An FDS consists of a Sorcerer II Computer (with keyboard), and a Floppy Disk Subsystem containing an MPI floppy disk drive and controller. The Exidy CBIOS may actually support three disk drives connected to the soft sectored disk controller. However, a controller and only one drive are supplied with the standard Floppy Disk Subsystem. The FDS may be augmented by other peripherals such as printers, cassettes, etc.. The Exidy CBIOS assigns the logical CP/M devices Punch and Reader, to serial write and serial read respectively. The List device is assigned to the Sorcerer Centronics parallel printer interface.

Version 2.2/1F CBIOS operates on the FDS and Version 2.2/2F operates on the DDS.
B. System Generation

Two programs, MOVCPM and SYSGEN, either create a new CP/M system or change its location in RAM. The MOVCPM program obtains a CP/M system image sized appropriately, for example, 32K, 48K, etc. and performs system relocation. SYSGEN takes the system output from MOVCPM and writes it to tracks 0 and 1 of the target diskette. The simplest method of doing this is:

A> MOVCPM 32 * <carriage return>
MOVEPM PROGRAM VERSION 2.0
CONSTRUCTING 32K CP/M VERS 2.2
READY FOR "SYSGEN" OR
"SAVE 40 CPM32.COM"
A>

Notice that "MOVCPM 32 *" is the only user input. At this point MOVCPM has created a 32K CP/M system image in memory and the user may either proceed immediately to SYSGEN or he may save the image on disk as a COM file by typing SAVE 40 CPM32.COM <carriage return> in response to the "A>" prompt. The latter procedure provides the user the option of modification with the DDT utility.

With a MOVCPM image in memory, as it is after the last prompt, the user may do a memory image SYSGEN. In the next example, the system is created on drive B. Note the response to the source drive name prompt is a carriage return. This indicates the source system already exists in memory.

A> SYSGEN <carriage return>
SYSGEN VER 2.0

SOURCE DRIVE NAME (OR RETURN TO SKIP) <carriage return>
DESTINATION DRIVE NAME (OR RETURN TO REBOOT) B
DESTINATION ON B, THEN TYPE RETURN <carriage return>

The actual disk writing occurs and when complete this message signs on:

FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)

At this point the system has been created on disk. The user should do a cold boot (RESET) on the new diskette (after placing it in drive A) to verify this. For more details on MOVCPM and SYSGEN, see "An Introduction to CP/M Features and Facilities."
MOVCPM may also be performed without the second "**" parameter MOVCPM 29. In this case, MOVCPM attempts to create and execute in memory a new system of the specified size. However, the system may be destroyed if the given memory size causes the new target system to use memory in either the executing MOVCPM program or the executing CP/M system. Exidy suggests always specifying the second "**" parameter for MOVCPM and using SYSGEN to create a new disk system.

C. Options

Two options come with the CBIOS: Diagnostic Error Messages and Read After Write Data Verification. The default settings are 1) no Diagnostic Error Messages and 2) Read After Write Verification.

With the Diagnostic Error Message option turned off, only fatal errors are reported to the user and recovered soft errors are not. The user selects this option if he wants all errors reported. A Diagnostic Error message shows for any error encountered.

The Read After Write Verification option allows all data to be reread and compared to the write buffer and CRC after the user has written to a physical disk sector. The user may turn off this option, increasing the speed of disk writes by 50 per cent. This may, however, decrease data reliability. We suggest leaving this option set to the default value to assure disk writes are being done successfully.

Only someone familiar with CP/M programming should attempt changing option values. To make these changes, the user creates a disk file with his CP/M system on it. That is, he does a MOVCPM, followed by a SAVE, as described in Section 2.B of this manual. The user then DDT's the CP/M system into memory, altering the contents of absolute location 1F02 hex to reflect the options he wishes, as shown below:

bit 0 = Read After Write Option (hex 01)
bit 2 = Diagnostic Error Message Option (hex 04)

If the value of the bit is 1, then the option is asserted. A bit of 0 turns off the option. Thus the default value is 01 hex for the Read After Write option without Diagnostic Error Messages.
D. Incompatibilities

Some CP/M disk formats, including Exidy's, are incompatible with other CP/M formats. All disk formats, are incompatible with the Micropolis Disk Operating System (MDOS). This incompatibility, especially evident with MDOS, is a result of different sector skewing arrangements. All Exidy disk-based software products only run on Exidy's CP/M(TM).

The Exidy skewing pattern for the FDS version follows for those interested in developing translation programs. The DDS version has a one-for-one sector correspondence and therefore no skewing.

E. Sector Skew Pattern

<table>
<thead>
<tr>
<th>CP/M logical 128-byte sector</th>
<th>Exidy 256-byte Sector (Physical Sector, first/last half)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16, first</td>
</tr>
<tr>
<td>2</td>
<td>16, last</td>
</tr>
<tr>
<td>3</td>
<td>13, first</td>
</tr>
<tr>
<td>4</td>
<td>13, last</td>
</tr>
<tr>
<td>5</td>
<td>10, first</td>
</tr>
<tr>
<td>6</td>
<td>10, last</td>
</tr>
<tr>
<td>7</td>
<td>7, first</td>
</tr>
<tr>
<td>8</td>
<td>7, last</td>
</tr>
<tr>
<td>9</td>
<td>4, first</td>
</tr>
<tr>
<td>10</td>
<td>4, last</td>
</tr>
<tr>
<td>11</td>
<td>1, first</td>
</tr>
<tr>
<td>12</td>
<td>1, last</td>
</tr>
<tr>
<td>13</td>
<td>14, first</td>
</tr>
<tr>
<td>14</td>
<td>14, last</td>
</tr>
<tr>
<td>15</td>
<td>11, first</td>
</tr>
<tr>
<td>16</td>
<td>11, last</td>
</tr>
<tr>
<td>17</td>
<td>8, first</td>
</tr>
<tr>
<td>18</td>
<td>8, last</td>
</tr>
<tr>
<td>19</td>
<td>5, first</td>
</tr>
<tr>
<td>20</td>
<td>5, last</td>
</tr>
<tr>
<td>21</td>
<td>2, first</td>
</tr>
<tr>
<td>22</td>
<td>2, last</td>
</tr>
<tr>
<td>23</td>
<td>15, first</td>
</tr>
<tr>
<td>24</td>
<td>15, last</td>
</tr>
<tr>
<td>25</td>
<td>12, first</td>
</tr>
<tr>
<td>26</td>
<td>12, last</td>
</tr>
<tr>
<td>27</td>
<td>9, first</td>
</tr>
<tr>
<td>28</td>
<td>9, last</td>
</tr>
<tr>
<td>29</td>
<td>6, first</td>
</tr>
<tr>
<td>30</td>
<td>6, last</td>
</tr>
<tr>
<td>31</td>
<td>3, first</td>
</tr>
<tr>
<td>32</td>
<td>3, last</td>
</tr>
</tbody>
</table>
Note that Exidy physical sectors are numbered 1 to 16, and are 256 physical bytes long. CP/M logical sectors are numbered 1-32 and are 128-bytes long. Thus, two CP/M sectors fit in one Exidy physical sector.

F. Special Video Display Interface

The video display on the Sorcerer II is a memory mapped device which is not standard with any other on the market. CBIOS supports the following TTY interface for standardized full screen operations.

<table>
<thead>
<tr>
<th>Operation</th>
<th>HEX</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form Feed</td>
<td>(0C)</td>
<td>Clear screen, home cursor</td>
</tr>
<tr>
<td>Clear Screen</td>
<td>(1A)</td>
<td>Same as above</td>
</tr>
<tr>
<td>Home Cursor</td>
<td>(1E)</td>
<td>Place cursor at row 0, col 0</td>
</tr>
</tbody>
</table>

Direct Cursor Addressing:

ESC "=" Row Column
(1B) (3D) Row/1-29 + 31(1F) Col/1-64 + 31(1F)

Using the above cursor addressing information the following sequence will position the cursor at row 1, col 2:
ESC "=" 32(20) 33(21)
A. Error Recovery

The CBIOS includes extensive automatic error detection, recovery, and reporting facilities. The Read After Write option, active by default, is the only error detection function controlled by the user. When a disk I/O error occurs, recovery is fully automatic in the following steps:

1. CBIOS retries operation up to 5 times until successful.
2. If the error still exists, it steps one track in/out alternately for a total of 6 times and repeats step 1 again.
3. If error still exists, then it deselects/reselects drive and then homes to track 0, up to 2 times, repeating 1 and 2. If error still exists, the error is treated as "permanent" and unrecoverable and the operation is aborted.

These error recovery steps are performed in nested fashion. That is, a separate counter is maintained for each error retry state, 1, 2, and 3. If step 1 fails, (its counter reaching 5), then step 2 is performed and its counter incremented. Meanwhile, the step 1 counter is reset, and its process again performed. If successive errors cause the step 2 counter to reach its maximum, then step 3 is performed, and its counter incremented. Both the first and second counters are reset, and step 1 is reinitiated. Thus a total of sixty (5x6x2) retry steps are performed before the error is declared non-recoverable. This retry process can take up to 75 seconds.

If the error is non-recoverable, the CBIOS issues an error message stating:

n DRV: ERR CODE= D

n here identifies the drive A, B, or C. Further identification of the error code follows the message.

The CBIOS then returns the error to its caller, the CP/M Basic Disk Operating System (BDOS). BDOS reports the error to the user, in less descriptive terms than the CBIOS in the following message:

BDOS ERR ON n: BAD SECTOR
The BDOS operation is suspended until the user hits any key except control-C. When any other key is hit, the BDOS retries the I/O. If the user wishes to end error processing, he must hit control-C or reset to the Sorcerer Monitor and perform either a warm boot (GO 0) or a cold boot.

If the user chooses the Diagnostic Error Message option, each error issues an I/O error message even if recovered by the CBIOS. In the event of a nonrecoverable error, the CBIOS prints 60 diagnostic error messages before declaring the error nonrecoverable and issuing the above error message. This procedure slows down recovery considerably. Only technicians diagnosing disk-related hardware errors should use this option.

One peculiar "error" of CP/M systems is the write protect error. The CBIOS shows this error message to the user:

n DRV: ERR CODE=B

However, the CBIOS doesn't report the error to the BDOS. Thus the BDOS thinks it is writing to a disk, but cannot because it is a write protected disk. The BDOS discovers the error only after it reads back the directory data and it does not agree with what it remembered having "written". This usually results in the following error message:

BDOS ERR ON n: R/O

The write protect error occurs when CP/M performs "token" directory writes upon reading each new extent of a file. Thus if a PIP (Peripheral Interchange Program) is performed on a large (>16K) file from a write protected diskette to a writable diskette, the token directory writes cause write protect errors on the write protected source diskette. If the CBIOS returned the write protect error to the BDOS, the user could never copy files from a write protected diskette (even though only reads are to be done). These write protect errors on a diskette used only for input can be ignored as a peculiarity in CP/M.

B. CP/M Programming

Cache. BIOS does not immediately, upon user request, execute disk writes. At any given moment there may be "dirty" buffers in the cache, that is, buffers which should be written to disk. Writing such buffers to disk is called "flushing the cache". The typical user who interfaces to the CBIOS through the BDOS, that is, does
logical file I/O, documented in the "Interface Guide", does not need to be aware of the flushing mechanism. The cache is automatically flushed upon BDOS file closing. Only the user who performs direct CBIOS I/O through the vector needs to be aware of cache flushing. The cache is flushed when:

1. Console output
2. A write to the directory track occurs
3. A CBIOS disk select occurs
4. Warm Boot

Programmers using the non-standard CBIOS I/O functions and not the standard BDOS ones should be careful and account for caching processes.

The best guide to CP/M programming is Digital Research's "CP/M Interface Guide." Although the guide is accurate and informative, additional information may help the user overcome any problems he may encounter, as listed below.

1. The BDOS search commands (function numbers 17 and 18) do not work as indicated in the "CP/M Interface Guide". The following provides accurate information
   a. The Search command (17) does not return a byte pointer. Instead, it returns the index of the found file (within the directory) to register A, or to 255 if a match is not found. The index of the file is within the range of 0 to 127, since the Exidy CP/M contains up to 128 directory entries. Directory entries are 32 bytes per entry, thus there are four entries per sector. The BDOS, searching for the desired file, reads directory sectors into its DMA buffer, located from 80 to FF hex. These facts provide the basis for the following formula. The File Control Block (FCB) for the found file is located at:

   \[ 80H + \text{MOD} (\text{index}, 4) \times 32 \]

   The BDOS returns "index" and MOD is the modulus function which returns the remainder of "index" divided by 4.

   b. Only after the initial search (17), may search (18) occur. The Interface Guide incorrectly states that an FCB parameter is required. Actually, the FDB from the previous search call (17) is used. The parameter returns in the A register and is a directory index exactly as described above in la.
The following is the error message format with an explanation of the various error codes:

x DRV: ERR CODE=c

where x represents the drive on which the error occurred (A, B, or C) and c represents one of the following:

A - Disk Select Error- The disk selected was not drive "A", "B", or "C".

B - Write Protect Error- The attempt to write to a write protected diskette was not reported to the BDOS as an error. See Section 3.B., CP/M Programming.

C - Disk Track Out of Range- A track number was detected past the end of the disk, indicating that the CP/M is very sick.

D - Non-Recoverable Disk I/O Error- All retries have failed to eliminate a read or write error. A more complete description follows this section.

E - Insufficient Memory for Disk Cache Buffers- At initialization, cache buffers are allocated. This error occurs if there is not sufficient space for one cache buffer.

F - Error on Cache Flush- An error occurred while the cache was flushed. The last CP/M job or command should be redone.

A non-recoverable disk I/O error (Code D) has a few possible causes, which is one of the following:

(I/O Type):TRK=ttH, SCTR=ssH: STAT=bbH *

Where I/O Type is of the following:

READ ERR - Error occurred during read operation

WRITE ERR - Error occurred during write operation

RDAFTWR ERR - The read after write option was selected, and the verify did not agree. The write however, was done successfully.

Note that track and sector values (tt and ss) are expressed in hexadecimal.
The status value (bb) is eight bits of status flags from the disk controller, expressed in hexadecimal. Only the status flags listed below are used for error indicators.

04H (bit 2) - Lost data (data overrun/underrun)
08H (bit 3) - CRC error
10H (bit 4) - Record Not Found (RNF)
20H (bit 5) - Write Fault for Write command
40H (bit 6) - Write Protect Flag
80H (bit 7) - Not Ready Flag

These codes show up in messages like this:

A DRV: ERR CODE = D

READ ERR: TRK=02, SIDE=00, SCTR=05: STAT=10H

This message would tell the user that he has a non-recoverable disk I/O error (ERR CODE = D) on the second track of sector 5 on side 0, and its status, 10H refers to bit 4, Record Not Found.
COPYRIGHT

Copyright (c) 1980 by Exidy Systems, Inc. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of Exidy Systems, Inc., 1234 Elko Drive, Sunnyvale, California 94086.

Since this manual is tutorial in nature, permission is granted to reproduce or abstract the example procedures and sample programs for the purposes of inclusion within the reader's programs.

DISCLAIMER

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

TRADEMARKS

CP/M is a registered trademark of Digital Research.
COPYRIGHT

Copyright (c) 1980 by Exidy Systems, Inc. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of Exidy Systems, Inc., 1234 Elko Drive, Sunnyvale, California 94086.

Since this manual is tutorial in nature, permission is granted to reproduce or abstract the example procedures and sample programs for the purposes of inclusion within the reader's programs.

DISCLAIMER

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

TRADEMARKS

CP/M is a registered trademark of Digital Research.
Table Of Contents


1. Introduction ........................................... 262
2. Use ......................................................... 263
3. Sample Run ............................................... 264
   A. Excopy With Two Multiple Drives .......... 264
   B. Excopy With One Drive ...................... 264
   C. Format Only ....................................... 265
4. Error Messages ......................................... 266
   A. Can Not Format, Try Again ................. 266
   B. Destination Is Write Protected ........... 266
   C. Write Error On Track #XX ................. 267
   D. Read Back Error On Track #XX ............ 267
   E. Additional Messages ......................... 267
5. Recovery ................................................. 268
1. INTRODUCTION

EXCOPY (TM) is a CP/M (TM) program which formats or copies formatted, soft-sectored diskettes on the Exidy Floppy Disk Subsystem (FDS) and Display Disk System (DDS). The copying operation automatically formats the destination diskette on either a single or a multiple drive configuration. The copy program minimizes the number of read/write cycles performed by determining the amount of RAM available as a copy buffer, using as much of it as possible. Disk formatting without copying may also be performed by the program. The user must not violate any software licensing agreements when copying diskettes.
2. **USE**

The practical user will want to secure back-up copies of all his important files, protecting against any possibility of losing data and enabling the user to read and write to the disk. Also, the user may wish to format only, that is, create on a new disk proper tracks and sectors to read and write to.

**EXCOPY** is called from CP/M by simply entering "EXCOPY" on the CP/M command line. The program signs on and requests an indication of one of these responses: "C" (or carriage return) for copy, "F" for format only, or "E" (or control-C) for exit back to CP/M.

If copying is requested, the program asks if more than one disk drive is configured and available for the copying operation. The response "Y" or "N" indicates yes or no. If more than one drive is indicated, a message instructs you to place the source diskette in drive A and the destination diskette in drive B. Copying automatically occurs after striking any key.

If only one drive is indicated, a message instructs you to first insert the destination diskette in the A drive for formatting. Hitting any key triggers this process. Then, alternately place source and destination diskettes in the A drive as requested by the console messages.

**WARNING:** Copying and Formatting destroys any previous information on the destination diskette. **BE SURE THERE ARE NO IMPORTANT FILES ON THE DESTINATION DISKETTE, AS THEY WILL BE LOST. ALSO, PAY CAREFUL ATTENTION TO THE DRIVE YOU PLACE YOUR SOURCE AND DESTINATION DISKETTE IN.** Any mixup will lose all information on your source diskette. Exidy suggests putting write-protect tabs on your source diskette to guard against any such mixup.

After copying, the console asks if more is desired. If your response is "Y" (yes), the cycle is repeated. Otherwise, the program directs the operator to place a system diskette in the A drive, hitting any key to re-boot the system.

Formatting without copying may also be requested. In this case, the program asks the user to specify which drive he wishes to format, "A", "B", or "C". After completion, the program asks if more is desired, and repeats the cycle if "Y" is entered.
3. **SAMPLE RUN:**

A. **EXCOPY With Multiple Drives**

Here is a sample of the console I/O when using EXCOPY.
For clarification, user input is underlined to differentiate from program output:

A>EXCOPY (return)

Exidy Disk Copy and Format Program
For 77 Track Single Sided diskettes. Ver 2.0
Copyright (C) 1981 Exidy Systems, Inc.

or

Exidy Disk Copy and Format Program
For 77 Track Double Sided diskettes. Ver 2.0
Copyright (C) 1981 Exidy Systems, Inc.

Format only, Copy, or Exit (F/C/E)?...C

Do you have more than one drive configured in this system (Y/N)? Y

Put source diskette in drive A and destination diskette in drive B then
Hit any key when ready. (any key)

(copying commences)

Good Copy.

More (Y/N)? N

Place system diskette in drive A and Hit any key when ready to reboot. any key

(CP/M reboots)

B. **EXCOPY with One Drive**

The following is the console I/O when invoking EXCOPY with only one drive configured on the Sorcerer. An asterisk (*) indicates the point where the program waits until any key is hit. Track numbers indicated will vary depending on size of RAM. In this example, RAM is 32K. Once again, be sure to begin by placing the DESTINATION diskette in the drive. Should you confuse it with the source diskette, all information will be permanently lost.
A>EXCOPY (return)

Exidy Disk Copy and Format Program
For 77 Track Single Sided diskettes. Ver 2.0
Copyright (C) 1981 Exidy Systems, Inc.

or

Exidy Disk Copy and Format Program
For 77 Track Double Sided diskettes. Ver 2.0
Copyright (C) 1981 Exidy Systems, Inc.

Format only, Copy, or Exit (F/C/E)?...(return)

Do you have more than one drive configured in this system (Y/N)? N

Place destination diskette in drive A
For initial formatting and
Hit any key when ready. (any key)

(formatting commences)

<table>
<thead>
<tr>
<th>Tracks</th>
<th>Tracks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source:* 0-- 6</td>
<td>Destination:* 0-- 6</td>
</tr>
<tr>
<td>Source:* 7--13</td>
<td>Destination:* 7--13</td>
</tr>
<tr>
<td>Source:*14--20</td>
<td>Destination:*14--20</td>
</tr>
<tr>
<td>Source:*21--27</td>
<td>Destination:*21--27</td>
</tr>
<tr>
<td>Source:*28--34</td>
<td>Destination:*28--34</td>
</tr>
<tr>
<td>Source:*35--41</td>
<td>Destination:*35--41</td>
</tr>
<tr>
<td>Source:*42--48</td>
<td>Destination:*42--48</td>
</tr>
<tr>
<td>Source:*49--55</td>
<td>Destination:*49--55</td>
</tr>
<tr>
<td>Source:*56--62</td>
<td>Destination:*56--62</td>
</tr>
<tr>
<td>Source:*63--69</td>
<td>Destination:*63--69</td>
</tr>
<tr>
<td>Source:*70--76</td>
<td>Destination:*70--76</td>
</tr>
</tbody>
</table>

Good Copy.

More (Y/N)? [N]

Place system diskette in drive A and
Hit any key when ready to reboot. [any key]

(cold boots system diskette)

C. Format Only

The console I/O for a Format Only operation would appear as follows:
Exidy Disk Copy and Format Program
For 77 Track Single Sided diskettes. Ver 2.0
Copyright (C) 1981 Exidy Systems, Inc.

or

Exidy Disk Copy and Format Program
For 77 Track Double Sided diskettes. Ver 2.0
Copyright (C) 1981 Exidy Systems, Inc.

Format only, Copy, or Exit (F/C/E)?...F
Select drive (A,B, or .C)... B

Place diskette in drive B for formatting then
Hit any key when ready, any key

(formating commences)

More (Y/N)? N

(etc.)

4. ERROR MESSAGES

Several conditions may display error messages on the console, as follows:

A. Cannot format, try again

This message appears when the disk controller fails either to write a track with the formatting data, or after writing, the controller cannot read back each sector on the track in question. An improper destination diskette (such as a hard-sectored diskette) or a worn or damaged one may cause this. Also, malfunctioning disk drive hardware which prevents formatting may return this message. Try fresh media and double-check the hardware. The program automatically restarts after each error to allow another attempt.

B. Destination is write protected.

This message is given if the destination diskette has a write protect tab covering its write protect notch. Either remove the tab or use an unprotected diskette.
C. Write error on track # xx

This message is returned during formatting if the write operation cannot be performed. The track number (xx) is specified.

D. Read back error on track # xx

This message is displayed if a track can be written to but cannot be read back after repeatedly attempting to do so. Again, the track number is specified as (xx).

E. Additional Message

I can't find your boot address!!!
Please enter it...[hhhh]

This occurs if a non-Exidy boot-strap controller prom is used and the copy program can't determine where the cold boot program is addressed. In this event, simply enter the hexadecimal address of your boot-strap prom. This should not occur if Exidy hardware and software is used.
5. Recovery

If the copy program has errors while reading the source, writing the destination, or verifying the destination diskette, the program repeatedly attempts the operation until successful. If, after many attempts, the operation still has errors, a message is displayed and the error retry continues indefinitely.

These messages are:
Recovering from read errors, track # xx
Recovering from write errors, track # xx

If it appears that the copy program cannot recover from an error, reset the computer to end processing and check ground and other connections between computer and disk drive. Also check the diskette for wear.