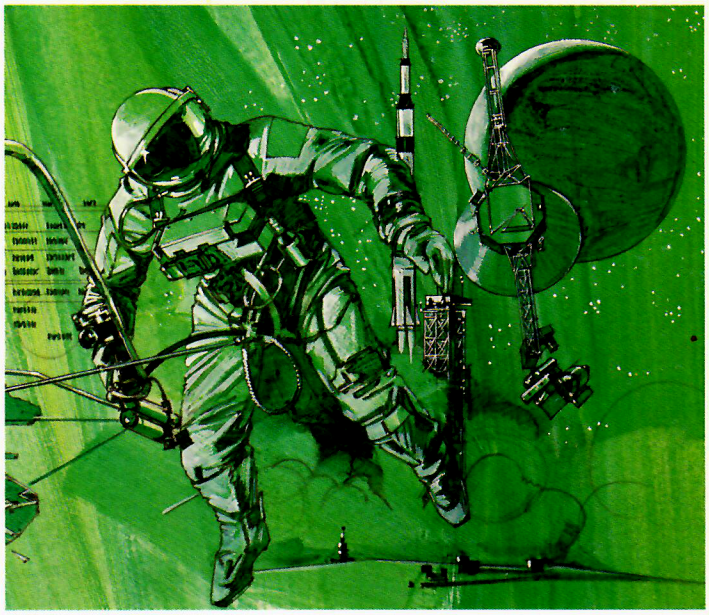
Personal Computer

MZ-800

Personal CP/M™ MANUAL



SHARP®

Personal Computer

MZ-800

Personal CP/M™ MANUAL

Introduction

Congratulation on your purchase of Sharp Personal CP/M™ (MZ-2Z047).
In this manual, Personal CP/M shall be represented by P-CP/M™.
Read this manual before use.

The P-CP/M is compatible with the CP/M V2.2 and has been developed as an operating system (OS) for use at home, in schools, and in small businesses. For this purpose the P-CP/M incorporates a VCCP (Visual Console Command Processor) system that stresses easy operation virtually without the use of a manual, so just about anyone can operate the system using menu selection.
The basic features of the P-CP/M are as follows:

- "Auto-logged in" allows free exchange of media during OS execution. (The CP/M V2.2 requires CTRL + C.)
- Support of AUXIN/AUXOUT devices for increased communication functions.
- Elimination of I/O bytes reduces dependence on peripheral devices.
- Display functions increased and BDOS call added.
- Use of a VCCP system.

In addition to the above features, when used with the MZ-800, the P-CP/M also has the following additional functions.

- Use of IBM format.
- High-speed file access.
- Support of ANSI escape sequence.
- Also single-drive compatibility.
- Wide array of utilities.

How To Use MZ-800 P-CP/M Documentation

The MZ-800 P-CP/M documentation set includes four manuals:

- Extension of MZ-800 P-CP/M
- Personal CP/M 8-Bit Operating System User's Guide
- Personal CP/M 8-Bit Operating System Programmer's Guide
- Personal CP/M 8-Bit Operating System System Guide

Extension of MZ-800 P-CP/M

This manual contains explanations of the parts of the User's Guide, Programmer's Guide, and System Guide which change when the P-CP/M is used with the MZ-800, and information on the additional utilities available with the MZ-800. We recommend that you read and use it together with the basic manuals (the three manuals mentioned).

Personal CP/M 8-Bit Operating System User's Guide

The Personal CP/M 8-Bit Operating System User's Guide (cited as the Personal CP/M User's Guide) introduce you to the Personal CP/M operating system and tells you how to use it.
The User's Guide assumes that the version of Personal CP/M delivered to you is ready to run on your computer.
To use this manual, you must be familiar with the parts of your computer, know how to set it up and turn it on, and how to handle, insert, and store disk.
However, you do not need a great deal of experience with computers.
Explanations of a LOAD command, an assembler, and a debugging tool are included for the use of a user with previous experience who might wish to use an assembler.

Personal CP/M 8-Bit Operating System Programmers Guide

The Personal CP/M 8-Bit Operating System Programmers's Guide (cited as the Personal CP/M Programmer's Guide) presents information for application programmers who are creating or adapting programs to urn under Personal CP/M.

Personal CP/M 8 Bit Operating System System Guide

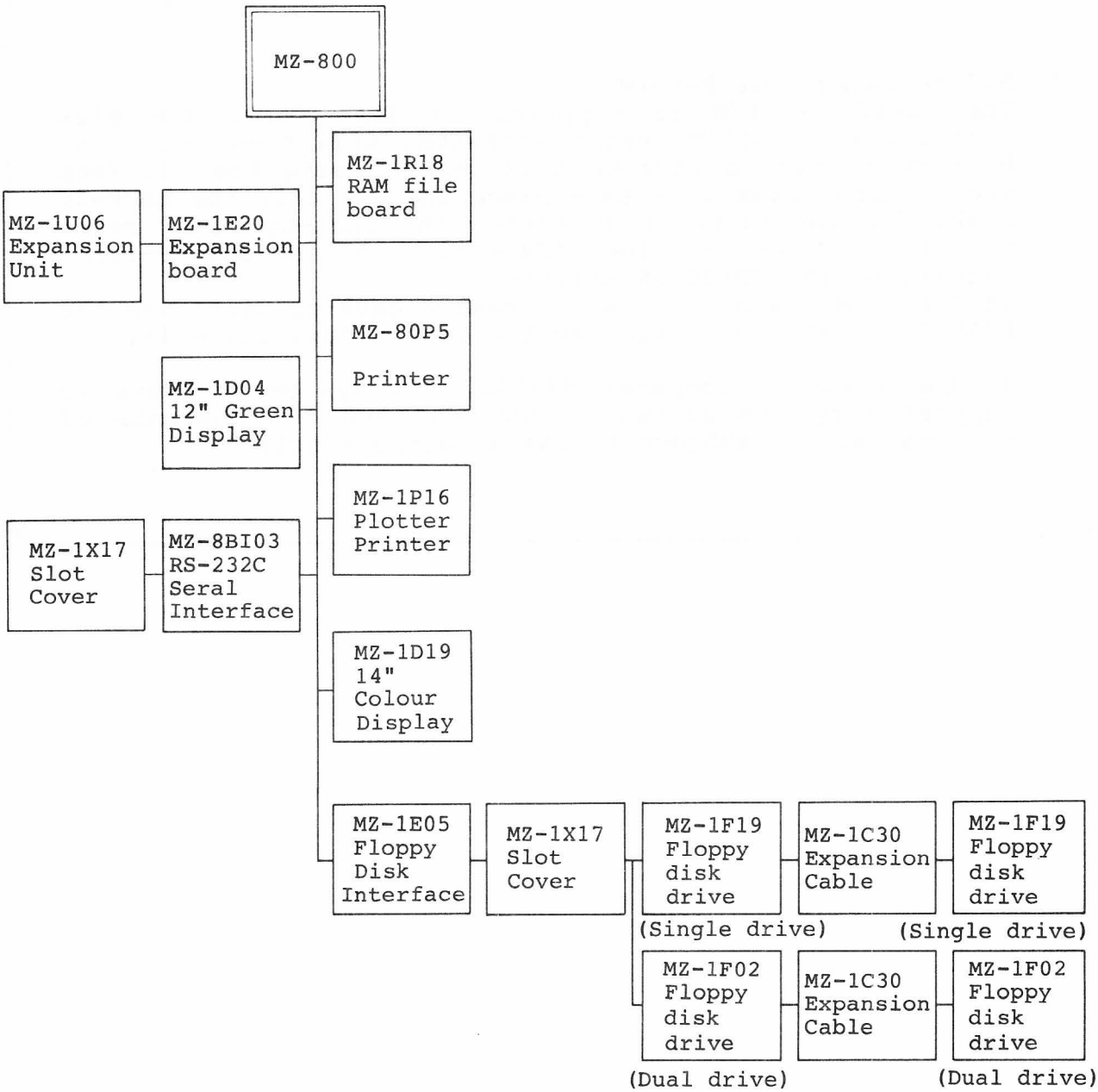
The Personal CP/M 8-Bit Operating System System Guide describes the steps necessary to creat or modify a Personal CP/M Basic Input/Output System tailored for a specific hardware environment.
Since Personal CP/M has been tailored by your computer manufacturer for your system, it is highly unlikely that you will need to modify Personal CP/M.

NOTE

- Before using your P-CP/M
  The MZ-800 P-CP/M is supplied in disk form. The disk contains the P-CP/M system software, utilities, etc., so be sure to make a back-up disk before using the OS. Keep the original disk in a safe place and use only the back-up disk. For instructions on making the back-up disk, refer to the section of the "Extension of MZ-800 P-CP/M" describing the COPYDISK utility.
  If for some reason you cannot make a back-up disk, use the FORMAT command to initialize the disk before using it.

- In the Personal Computer MZ-800, all system software is supported by the software component and the contents of this manual are subject to change without notice.

MZ-800 System Configuration under P-CP/M



Note: One of the models described in this manual may not be available in some countries.

Assignment of Storage Devices When Using P-CP/M For the MZ-800

When activating P-CP/M for the MZ-800, the storage devices and their respective numbers are assigned as follows.

Logical Drive number	A	B	C	D	E
Device name	Mini-floppy disk drive				Memory disk
Physical drive number	1	2	3	4	—

Personal CP/M™

8-Bit Operating System

Extension of MZ-800 P-CP/M

EXTENSION OF THE MZ-800 P-CP/M

1. LOADING AND EXECUTION OF THE P-CP/M

(1) Insert the disk containing the P-CP/M disk into drive A.
(2) Switch ON the power source of the MZ-800.
(3) The following display will appear.

```
Select a Command                    VCCP V1.0A

                Drive=A:   User=0   Search string=*.*

PCPM.SYS        ASM.COM         COPY.ASM        COPYDISK.COM
COPYSYS.COM     DDT.COM         DEL.COM         DISKDEF.COM
DISKEDIT.COM    DUMP.ASM        ED.COM          EJECT.COM
FILES.COM       FORMAT.COM      LOAD.COM        PIP.COM
RANDOM.ASM      SETUP.COM       STAT.COM        SUBMIT.COM
TERMINAL.ASM    TIME.COM        VCCP.CFG        VCCP.COM
XSUB.COM

Help
Dir
Drive
User
Run
Search
Rename
Erase
Type
Print
Make
Quit

Command: Help, display command help text
DIR↓   DIR B:↓   TYPE   REN   < NEXT >   / 12:34:56 /   CAPS
```

VCCP* Main Menu

*VCCP is an abbreviation for Visual Console Command Processor.
In this Main Menu, you select the operations (commands) you want performed, and you select the files upon which the operation will be performed.
The Main Menu is composed of three parts: command window, file directory window and prompt/status line.

NOTE: VCCP is actually an application that runs, like all other applications and programs on your computer, under Personal CP/M. If you prefer, you may use Personal CP/M without running VCCP.

Command Window
Commands are key words identifying operations. The command window (the right hand portion of the menu) lists the 12 commands.

File Directory Window
The file directory window (the central portion of the menu) displays a directory of files on the disk in the active drive, that is, the drive currently being accessed.

Prompt/Status Line
The prompt/status line (at the 24th line) serves two purposes. First, it displays a capsule summary of the highlighted command. Second, when you select a command, it displays short messages and prompts regarding the command.

System Line
The bottom of the screen displays the contents of function keys. (Refer to 1.1)

(4) Selecting Commands
To select a command, move the highlight box to the command you want, then press the [CR] key.

When you start P-CP/M, the highlight box is on the HELP command. You can move the highlight box in either of two ways. To move the highlight box up and down the command list one command at a time, press the up or down arrow keys or the space bar. Also, you can type the first one letter of the command you want; the highlight box then jumps to that command.

The 12 VCCP commands are listed below.

command	description
Help	displays the help message for the Main Menu
Dir	changes the active drive; changes the active user number
Drive	changes drive number
User	changes user number
Run	runs program files
Search	sets the search string for the file directory
Rename	changes file name
Erase	erases files
Type	displays the contents of files
Print	prints files on a printer attached to your compeuter
Make	creates files
Quit	exits VCCP to the Personal CP/M A > prompt

How to add commands to the VCCP
VCCP.CFG files are made by ED as described below.

(1) Set the number of commands (max. 9) to be added on the first line.
(2) On the second and subsequent lines, define the file names (8 characters or less).
Example: PIP [CR] for PIP COM.
(3) For each line, from ";" to [CR] is considered to be the comment, and is ignored.
(4) Make the file size 2k byte or less.
(5) For each line, ignore 20H (spaces).
Example:
3 ; max file [CR]
Files [CR]
PIP [CR]
Format [CR]

1.1 Function keys

The content of the function keys ([F1] - [F4]) consists of 4 pages; each page can be selected by the [F5] key. The function keys are originally set as follows; they can be changed by the SETUP utility.

	F1	F2	F3	F4
Page 0	DIR [↓]	DIR B: [↓]	TYPE ␣	REN ␣
Page 1	ERA ␣	STAT ␣	PIP ␣	PIP A:=B:*.* [V]
Page 2	FORMAT	COPYSYS	SETUP	FILES [↓]
Page 3	ED ␣	ASM ␣	DDT ␣	DISKDEF

" [↓] " is code for the CR key and " ␣ " is the code for the space bar.
When [F5] key is pressed, the next page is displayed.
Therefore, assigning the dir [↓] to a key means that the following command will be executed when the key is pressed.

dir [CR]

1.2 CLOCK

The clock is always set to 00:00:00 when the power is switched on.
It contains a 6-digit valve which is the time of the 24-hour built-in clock.
Time setting can be done using the ANSI ESC sequence or the TIME utility.

1.3 CAPS (Capital lock)

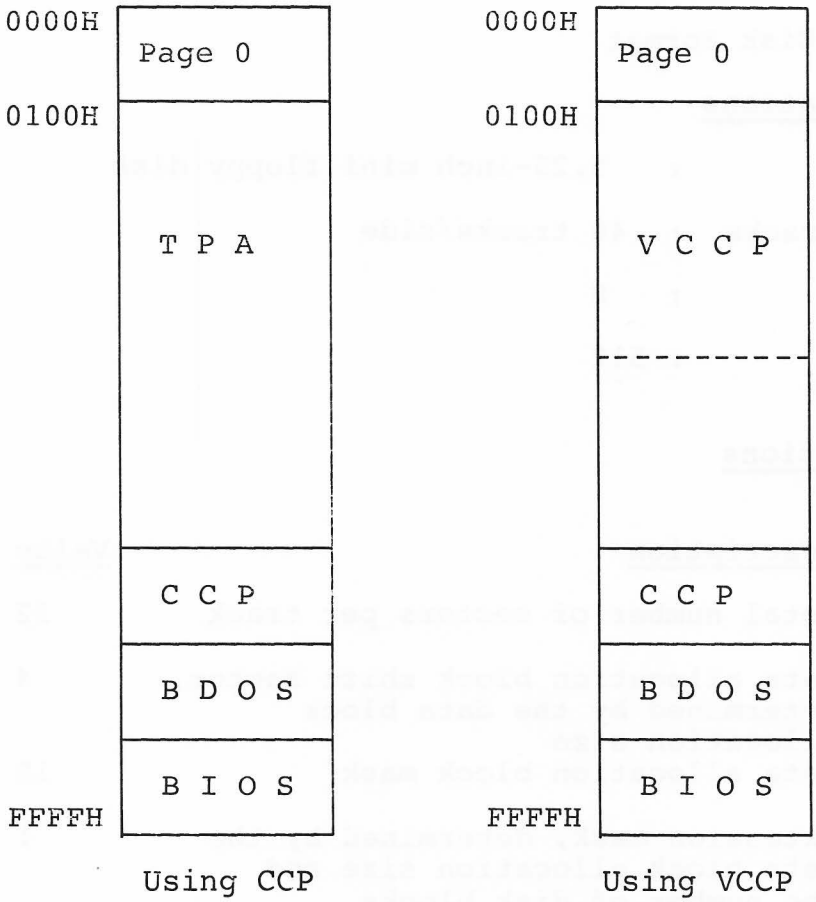
Indicates the present key condition. Because the MZ-800 does not have a CAPS key, this indication appears on the screen.

CAPS: Lower-case

[CAPS]: Upper-case
↑—Reverse

It functions as a toggle key: when pressed once, it displays CAPS in reverse video and enters upper-case letters; when pressed again, it cancels the reverse mode of CAPS and enters lower-case letters.

2. MEMORY MAP OF MZ-800 UNDER P-CP/M



3. DISK FORMAT

As many as four 5.25-inch mini-floppy disk drives can be connected to an MZ-800 using P-CP/M.

3.1 Mini Floppy Disk Format

Physical specifications

Media	:	5.25-inch mini floppy disk
Total number of tracks	:	40 tracks/side
Sectors/track	:	8
Bytes/sector	:	512

Logical specifications

Word value	Description	Value
SPT	total number of sectors per track	32
BSF	data allocation block shift factor, determined by the data block allocation size	4
BLM	data allocation block mask	15
EXM	extension mask, determined by the data block allocation size and the number of disk blocks	1
DSM	total storage capacity of the disk drive	155
DRM	total number of directory entries that can be stored on this drive	63
CKS	size of the directory check sector	16
OFF	number of reserved tracks at the beginning of the (logical) disk	1

4. FILES ON THE P-CP/M DISK

The following files are already written on a P-CP/M master disk.

* ASM	.COM	.....	Assembler
* COPY	.ASM	.....	Sample File-to File Copy Program
* COPYDISK	.COM	.....	Copy disk program
* COPYSYS	.COM	.....	System file copy program
* DDT	.COM	.....	Dynamic debugging tool
* DEL	.COM	.....	File delete with check program
* DISKDEF	.COM	.....	Disk convert program
* DISKEDIT	.COM	.....	Patch program
* DUMP	.ASM	.....	File dump program (source file)
* ED	.COM	.....	Editor
* EJECT	.COM	.....	System printer form feed command
* FILES	.COM	.....	Extended sorted directory display
* FORMAT	.COM	.....	Disk initialization program
* LOAD	.COM	.....	HEX to COM conversion program
* PCPM	.SYS	.....	P-CP/M system program
* PIP	.COM	.....	Data transfer program between peripheral units
* RANDOM	.ASM	.....	Sample Random Access Program
* SETUP	.COM	.....	Set-up program
* STAT	.COM	.....	Status information program
* SUBMIT	.COM	.....	Program for batch processing
* TERMINAL	.ASM	.....	Sample Full duplex Terminal Emulator
* TIME	.COM	.....	Time set/display program
* VCCP	.CFG	.....	VCCP Configuration file
* VCCP	.COM	.....	Visual Console Command Processor
* XSUB	.COM	.....	Extension program for batch processing

5. OPERATION OF MZ-800 P-CP/M ON A SINGLE FLOPPY DISK DRIVE SYSTEM

MZ-800 P-CP/M will function with a single drive system, regarding the single physical drive as two logical drives. When diskettes must be exchanged during on operation, a message is displayed in the system display area.
NOTE:
The number of connected disk drives must be set at the SETUP (DEVICE ASSIGN) utility beforehand. Note that the initial value is 2.

Example:

A > PIP A:=B:*.*[V]

The 25th line is as follows in the system message area.

Set disk B: /Push any key

6. USING RS-232C FUNCTIONS

The AUXIN/AUXOUT functions of the BIOS may be used when the optional RS-232C Interface Card (MZ-8B103) is installed. This interface card has two channels, both are supported by P-CP/M. The AUXIN/AUXOUT function is assigned to channel A by default, this may be changed by using the SETUP utility.

7. FUNCTION OF EXTENDED BIOS

7.1 Standard BIOS Functions

BIOS functions differ from the CP/M V2.2 as decribed below.

BOOT : Set 0003H to 00H to default to the standard 'A >' prompt CCP, or to 01H to default to visual CCP. Jump to CCP+0003H for the standard CCP, or to CCP+0000H for the Visual CCP.

WBOOT : If 0003H equals 00H, then jump to CCP+00003H, otherwise jump to CCP+0000H.

AUXOUT: This function sends an 8-bit character from register C to the currently assigned auxiliary output device.

AUXIN : This function reads the next 8-bit character from the AUXIN device into register A.

7.2 PUBLIC BIOS Functions

* MZ-800 P-CP/M does not support the ?BYTBC and ?BYTBA PUBLIC BIOS subroutines.

* The following function is added to the PUBLIC BIOS subroutines of MZ-800 P-CP/M.

Function	Input	Output
PRTSW	C=0 (normal printer) C=1 (MZ printer)	None

PRTSW functions as a switch for selection of the MZ type of printer and another ASCII-standard printer. When 0 is set to 0 register, the ASCII-standard printer is selected; when 1 is set, the MZ-printer is selected.
This is to provide printer selection by the user application program. Setting is also possible via the SETUP utility.

* Direct Screen Subfunction
* The following addition had been made to the direct screen subfunction.

Subfunction	Description
3	IDENTIFY TERMINAL Return Value: HL=Pointer to null-terminated identifier string ANSI: ESC, '[', 'n', NULL VT52: ESC, '/', 'Z', NULL For example, a VT-52 type terminal would return the bytes; ESCape, '/', 'K', NULL
17 18 19 20	Not supported by MZ-800 P-CP/M
23	ENTER REVERSE VIDEO MODE
24	EXIT REVERSE VIDEO MODE For the MZ-800 P-CP/M, when subfunction 23 is executed, all subsequently displayed characters are reversed and CONOUT occurs. Subfunction 24 is for return to reversed characters to the normal display.

8. FUNCTION OF BDOS

Comparing the MZ-800 P-CP/M with the CP/M with the CP/M V2.2, more system calls have been added and changed. For details, please refer to the P-CP/M Programmer's Guide.

* Functions not supported by MZ-800 P-CP/M are as follows.

Subfunctions for Function 113

Subfunction	Description	
17:	ENTER GRAPHICS MODE	Not supported by the MZ-800 P-CP/M
18:	EXIT GRAPHICS MODE	
19:	ENTER ALTERNATE KEYPAD MODE	
20:	EXIT ALTERNATE KEYPAD MODE	

9. FUNCTION OF CONTROL CODES AND ESC SEQUENCES

9.1 Control Code

Key	Function
CTRL + G	Bell
CTRL + H	Cursor left (backspace) Moves the cursor one column to the left. When cursor is in the first column in a line, it is moved to the last column in the preceding line. When the cursor is at the home position, this code is ignored.
CTRL + I	Skip to next TAB stop. Moves the cursor to the next TAB stop.
CTRL + J	Cursor down Moves the cursor down one line. When the cursor is on the last line, the screen is scrolled up one line.
CTRL + K	Cursor up
CTRL + L	Cursor right Moves the cursor one column to the right. When the cursor is at the last column in a line, it is moved to the first column in the following line. When the cursor is at the last column in the last line, the screen is scrolled up one line and the cursor is moved to the first column in the last line.
CTRL + M	Carriage return
CTRL + Z	Clear screen Clears the entire screen. The cursor is not moved.
CTRL + [	ESCAPE Inputs an escape code (1BH).

Key	Function
CTRL + \	Inputs code (1CH).
CTRL +]	Inputs code (1DH).
CTRL + ↑	Inputs code (1EH). Cursor home.

9.2 ESC sequence

In this table, ESC stands for code 1BH, and designations Pn, Pl, Pc, and Ps indicate decimal numbers.

Escape sequence		Function
ANSI	VT52	
ESC [? 2 1	--------	Sets the VT52 mode.
--------	ESC <	Sets the ANSI mode.
ESC [Pn A	ESC A	Cursor up This escape sequence moves the cursor upward by the number of lines specified for n (where n=pn). Here, pn=1 is assumed if pn is omitted or set to equal 0.
ESC [Pn B	ESC B	Cursor down This escape sequence moves the cursor downward by the number of lines specified for n (where n=pn). Here, pn=1 is assumed if pn=1 is omitted or set to equal 0.
ESC [Pn C	ESC C	Cursor forward This escape sequence moves the cursor to the right by the number of columns specified for n (where n=pn). If the number of columns specified is greater than the number of columns to the right side of the screen, the cursor is positioned in the column on the far right.

Escape sequence		Function
ANSI	VT52	
ESC [Pn D	ESC D	Cursor backward This escape sequence moves the cursor to the left by the number of columns specified for n (where n=pn). If the number of columns specified is greater than the number of columns to the left side of the screen, the cursor is positioned in the column on the far left.
ESC [0 J	--------	Erase from cursor to end of screen This escape sequence erases the screen from the position of the cursor to the end of the screen. (The 0 parameter may be omitted.)
ESC [2 J	ESC E	Erase entire screen This escape sequence erases the entire screen.
ESC [0 K	ESC K	Erase from cursor to end of line This escape sequence erases the screen from the cursor to the end of that line.
ESC [1 K	--------	Erase from beginning of line cursor This escape sequence erases the screen from the beginning of the line in which the cursor is located to the cursor's current position.
ESC [2 K	--------	Erase entire line with cursor This escape sequence erases the entire line in which the cursor is located.

Escape sequence		Function
ANSI	VT52	
ESC [Pl;Pc H	ESC Y line column	Cursor position This escape sequence determines the position of the cursor on the screen. When pl=m, the cursor is positioned to the mth line. If m is greater than the greatest line, the cursor is positioned to the last line. When pl=0 or pl is not specified, the cursor is positioned to line 0. When pc=n, the cursor is positioned to the nth column. If n is greater than the greatest column, the cursor is positioned to the last column. When pc=0 or pc is not specified, the cursor is positioned to column 1.
ESC [Pl;Pc f	--------	Same as above
ESC = line column	--------	Same as above
--------	ESC H	This escape sequence moves the cursor to the home position.
ESC [s	ESC j	Save cursor position The current cursor position is saved.
ESC [u	ESC l	Restore cursor position This sequence restores the cursor position.
ESC [6 n	--------	Please report active position Posts the cursor position for an immediately following console input call. The format is ESC [pl;pc R.

Escape sequence		Function
ANSI	VT52	
ESC [ps;..; psm	--------	Character attribute Sets the character display attribute as follows. 0 or nothing: Previously determined attribute 7 : Reverse
ESC [HH;MM; SS t	--------	Set time Sets the time, the format for specification of the time is as follows. HH:MM:SS (Hour:Minute:Second)
ESC [Pn T	--------	Scroll up
ESC [Pn S	--------	Scroll down
ESC D	--------	Cursor one line down
ESC M	--------	Cursor one line up
ESC [? 7h	--------	Sets wrap-around mode.
ESC [? 71	--------	Resets warp-around mode.
--------	ESC I	Reverse line feed Moves the active position upward one position without altering the column position. If the active position is at the top margin, a scroll down is performed.

10. UTILITY COMMANDS

These specifications describe the utility program which have been added to P-CP/M for the MZ-800 (the following programs).

FORMAT. COM

COPYDISK. COM

COPYSYS. COM

SETUP. COM

DISKEDIT. COM

DISKDEF. COM

TIME. COM

FILES. COM

DEL. COM

EJECT. COM

FORMAT Command

Syntax: FORMAT

FORMAT { d: } {/S } {/N }
d: drive number

Explanation: The FORMAT command clears all the data stored in the floppy disk in drive B or the specified drive and initializes the disk. When you use a new floppy disk or a disk used for a machine other than the MZ-800, always use the disk after initializing it with the FORMAT command. When no drive is specified, drive B is selected.
With the option switch's /S or /N, you can specify whether or not you will copy the P-CP/M system of the MZ-800 after formatting.
If you specify /S, the P-CP/M system is copied.
If you specify /N, the P-CP/N system is not copied.
If you do not specify an option switch, the P-CP/M system is copied in the same way as specifying /S.
You cannot specify both /S and /N at the same time.
Specify /N when you use a floppy disk already formatted as a data disk or when you execute the COPYDISK command after formatting.

Examples:

```
A >FORMAT
Format version 1.0A
Drive B: will be formatted, then type <CR> (^C or
ESC to reboot)? ■
Now formatting <++++++++++++++++++++++++++++++++++++++++>
Copy system
Function complete
Format next disk (Y/N)? N
A >
```

(1) Enter "FORMAT".
(2) Put a new floppy disk into drive B.
(3) Enter CR as shown on the screen; formatting then starts.
(4) When formatting is complete, a message is displayed.
Then the P-CP/M system is copied from drive A.
(5) When copying is complete, a message is displayed.
To format the next floppy disk, enter "Y"; to end formatting, enter "N".

The following example specifies the drive name and an option switch with the FORMAT command.

A >FORMAT C: /N

This means that the floppy disk in drive C will be formatted and the system will not be copied.
In this case, and error occurs if no disk drive unit is connected as drive C.

NOTE:
The FORMAT command clears all the floppy disk completely. Be careful not to erase any important contents by mistake. Formatting is stopped by pressing the C key while holding down the CTRL key.

COPYDISK Command

Syntax: copydisk

copydisk ds: dd:

ds: Source disk drive
dd: Destination disk drive

Explanation: The COPYDISK command copies all the contents of the floppy disk to another floppy disk. It copies not only the files on the floppy disk but also the P-CP/M system and provides a floppy disk which is exactly the same as the original one.
Before copying to a new disk, be sure to initialize the new disk with the FORMAT command. Also ensure that the destination disk has no write-protect seal on it. Copying to a disk with a write-protect seal will cause an error.
The copy disk utility is the fastest procedure available for copying the entire contents of one disk to another.
However, when the source disk has been extensively used (when files have been created and deleted many times), the records making up those files will be scattered around the disk in many different locations. In such cases, the efficiency of subsequent processing can be increased (at the expense of lower speed during copying) by using the PIP utility to copy each individual file. The result is that records making up each individual file are located adjacent to each other on the disk.

The COPYDISK command also enables a single drive to copy a disk.

Examples: **A >COPYDISK**

Copy disk version 1.0A

Copy Disk from A: to B:
Push any key when ready

Funcition complete
Copy next (Y/N)? ■

1. Copying with two disk drives

(1) Enter "COPYDISK".
(2) The copy program begins to run and displays the names of the source floppy disk and the destination floppy disk. Put the specified disks to each drive and enter CR.
(3) When copying is completed, a message is displayed.
(4) To copy another disk, enter "Y"; to end copying, enter "N".

2. Copying with one disk drive

A >COPYDISK

(1) Enter "COPYDISK".

Copy disk version 1.0A

Copy Disk from A: to B:
Push any key when neady

(2) The copy program begins to run, displays that it is in the single drive mode, and displays the names of the source drive and the destination drive.
Put a source disk with the write-protect seal into the drive, and Press any key.
(3) The contents of the source disk put into the disk drive are read as far as the memory capacity allows.
(4) When reading is finished, a message telling you to put the disk for drive B is displayed in the system message area. Take out the source disk, set the disk you want to copy to, and press any key.

Set disk B: /Push any key

(5) The contents of the source disk previously read are written into the new disk.
(6) When writing is completed, the contents written are verified.
(7) If verifying succeeds a message telling you to put the disk for drive A (the source disk) is displayed in the system message area. Take out the destination disk, put the source disk, and press any key.

Set disk A: /Push any key

Then return to step (3).
In a single drive system, this procedure is repeated until all the contents of the source disk are transfered to the destination disk.
Normally, it is necessary to repeat this procedure seven or eight times in order to copy a 320k-byte, 5.25-inch floppy disk.
(8) When copying is completed, a message is displayed asking if you wish to continue copying.
(9) To copy another disk, enter "Y"; to end copying, enter "N".

3. Specifying the drive names

With the COPYDISK command, if the drive names are not specified copying is executed drive A as source disk drive and drive B as destination disk drive. You can specify the drive name when entering the command if required.

A >COPYDISK B: A:

Source disk drive .... drive B
Destination disk drive .... drive A

A >COPYDISK C: D:

Source disk drive .... drive C
Destination disk drive .... drive D

COPYSYS Command

Syntax: COPYSYS d:

(A: Source disk drive)
d: Destination disk drive

Explanation: Copies the system program of the P-CP/M system disk for the MZ-800 (tracks including the bootstrap sector and the system file named "PCPM. SYS") to floppy disks formatted for P-CP/M for the MZ-800.
It is used only to write a system program to a floppy disk formatted with the /N option by the FORMAT command or a floppy disk from which all files are erased.

The COPYSYS command copies the following files:

BOOT CODE (physical copy)
P-CP/M LOADER (physical copy)
PCPM. SYS (file copy)

PCPM.SYS is a P-CP/M system file which does not appear in the directory. It can be anywhere on the disk, but it has to be in the entry position of the directory.
Therefore the disk to be copied must be either of the following:

1. A disk completely initialized by the FORMAT command
2. A disk for the MZ-800 already including the P-CP/M system

Examples: 1. Copying with two disk drives

A >COPYSYS

(1) Enter "COPYSYS".

Copy system version 1.0A

Copy system from A: to B:
Push any key when ready

(2) The copy program begins to run and displays the names of the source disk drive and the destination drive.
Insert each disk, then press any key.
(3) The COPYSYS program reads the system file from the source disk in drive A and writes it to the destination disk in drive B.

Set disk B: /push any key

(4) When copying is completed, a message is displayed.
(5) To copy to another disk, enter "T"; to end copying and return to P-CP/M, enter "N".

2. Copying with one disk drive

A > COPYSYS

(1) Enter "COPYSYS".

Copy system version 1.0A

Copy Disk from A: to B:
Push any key when neady

(2) The copy program begins to run, displays that it is in the single drive mode, and displays the names of the source drive and the destination drive.
Put a source disk with the write-protect seal into the drive, and Press any key.
(3) The contents of the source disk put into the disk drive are read as far as the memory capacity allows.
(4) When reading is finished, a message telling you to put the disk for drive B is displayed in the system message area. Take out the source disk, set the disk you want to copy to, and press any key.

Set disk B: /Push any key

(5) The contents of the source disk previously read are written into the new disk.
(6) When writing is completed, the contents written are verified.
(7) If verifying succeeds a message telling you to put the disk for drive A (the source disk) is displayed in the system message area. Take out the destination disk, put the source disk, and press any key.

Set system A: /Push any key

Then return to step (3).
In a single drive system, this procedure is repeated until all the contents of the P-CP/M system is transfered to the destination disk.
Normally, it is necessary to repeat this procedure a couple of times in order to copy the P-CP/M system.
(8) When copying is completed, a message is displayed asking if you wish to continue copying.
(9) To copy another disk, enter "Y"; to end copying, enter "N".

* Error messages
When the FORMAT, COPYDISK or COPYSYS command is executed, the following messages are displayed in the system message area when necessary.

Set disk B: /push any key

For a single drive system, set the disk for drive A (normally the source disk).

Set disk A: /push any key

For a single drive system, set the disk for drive B (normally the destination disk).

Read error: on A: R)etry, A)bort, I)gnore

An error is detected while the content of the disk in drive A is being read.

Write error: on B: R)etry, A)bort, I)gnore

An error is detected while the content of the disk in drive B is being written.

verify error: R)etry, A)bort, I)gnore

An error is detected while the written data is being verified.

When an error is detected, enter one of the following keys.

R (Retry)
→ This will re-attempt the operation

A (Abort)
→ Process is aborted, with return to condition immediately prior to COPYDISK start.

I (Ignore)
→ Ignore the error and continue the operation

SETUP Command

Syntax: SETUP

Explanation: The SETUP Command functions to set up or change the configuration of the P-CP/M system for the MZ-800. Its functions are as follows.

1. Specification of the file to be execute automatically
2. Specification of the colour of characters and background of the screen
3. Assignment of physical device to the logical device
4. Specification of Floppy Read After Write
5. Specification of ON/OFF of key click sound
6. Specification of the MSB* of the console output characters
7. Specification of the code system for the printer
8. Definition of parameters of the RS-232C Ports.
9. Definition of the definable keys

* MSB is the abbreviation for Most Significant Bit.

Using to the VCCP of P-CP/M, specification or change is done via the menu with the cursor.

Examples: (1) Enter "SETUP".

(SETUP main menu)

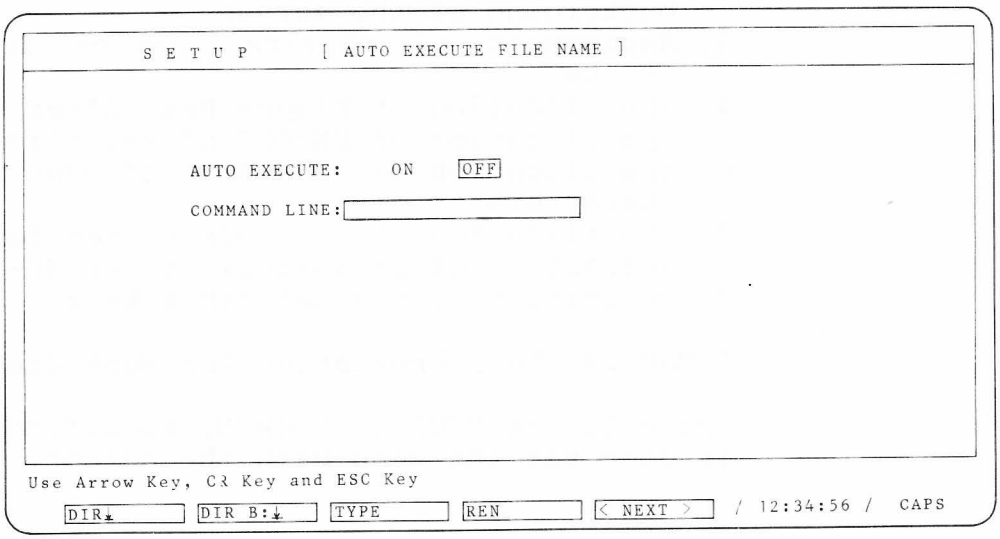
```
S E T U P  for MZ-800 P-CP/M [ V1.0A ] (C) SHARP Corp.

                    1  AUTO EXECUTE FILE
                    2  CHARACTER COLOUR
                    3  DEVICE ASSIGN
                    4  FLOPPY DISK
                    5  KEY CLICK SOUND
                    6  MSB MASK
                    7  PRINTER MODE
                    8  RS-232C PARAMETERS
                    9  USER DEFINABLE KEY
                    0  END OF SETUP

Use Arrow key to select and CR or ESC key to exit
DIR↓   DIR B:↓   TYPE   REN   < NEXT >   / 12:34:56 /   CAPS
```

(2) The menu is displayed as shown above.
Move the cursor to the item which you want to verify or change. Then press the CR key.
The display changes to the item you select.

Main menu

Displayed initialialy after the SETUP command.
Select the item you want to set up or change from the menu the cursor or from numeric keys and press the CR key. Then the option menu appears where you can set up the item you select. The available keys in the main menu are [↑] , [↓] , 0 to 9, and CR.

(1) SETUP [AUTO EXECUTE]



You can set up this function to execute the specific command file when the power is switched ON (when the P-CP/M begins to run). When AUTO EXECUTE is ON, this function is effective; when AUTO EXECUTE is OFF, nothing is executed after the P-CP/M begins to run; only the prompt (A >) is displayed.
When AUTO EXECUTE is ON, a command of maximum 20 characters specified at COMMAND LINE is executed as entered into the command line of the P-CP/M.
When you wish to execute a specific program automatically at the POWER ON, set AUTO EXECUTE ON and enter the file name of the program for automatic execution.
To enter characters to COMMAND LINE: use the [↑] and [↓] keys to move the cursor to COMMAND LINE: and then use the [CR] key to enter the command name.
EXAMPLE: When you set up VCCP (Visual Console Command Processor) to be executed the POWER ON, enter "VCCP" into the COMMAND LINE. If OFF is entered at AUTO EXECUTE line, automatic execution does not function regardless of the content of the COMMAND LINE area. After setting, press the CR key or the ESC key; the main menu is displayed again.

(2) SETUP [CHARACTER COLOUR]

```
S E T U P     [ CHARACTOR COLOR ]

FOREGROUND: BLACK   BLUE    RED     MAGENTA GREEN   SYAN    YELLOW  [WHITE]
 INTENSITY: LOW     [HIGH]

BACKGROUND: BLACK   [BLUE]  RED     MAGENTA GREEN   SYAN    YELLOW  WHITE
 INTENSITY: [LOW]   HIGH

Use arrow key,CR key and ESC key.
[DIR↓] [DIR B:↓] [TYPE] [REN] [< NEXT >] / 12:34:56 / CAPS
```

When a colour CRT is connected to the MZ-800, you can select the background colour from 16 colours when the P-CP/M begins to run.
With this display, you can specify the FOREGROUND COLOUR and BACKGROUND COLOUR by selecting the colours from the 16 colours displayed by the cursor. The set-up result is applied to the display instantly, so you can view the combination of colours.
Use the cursor key or the space bar to set up, and the CR key or the ESC key to return to the main menu.

(3) SETUP [DEVICE ASSIGN]

```
S E T U P      . [ DEVICE ASSIGN ]

CONSOLE IN:    [KEY]  RS1   RS2
CONSOLE OUT:   [CRT]  RS1   RS2   PRN
AUX IN:        KEY    [RS1] RS2
AUX OUT:       CRT    [RS1] RS2   PRN
LIST OUT:      CRT    RS1   RS2   [PRN]
DISK DRIVES:   1      [2]   3     4

Use Arrow Key, CR Key and ESC Key
[DIR*] [DIR B:*] [TYPE] [REN] [< NEXT >] / 12:34:56 / CAPS
```

The MZ-800 can deal with five logical devices: CONSOLE-IN, CONSOLE-OUT, AUX-IN, AUX-OUT and LIST-OUT.
You can assign to those logical devices one of the physical device such as RS-232C, the printer, as well as the keyboard and CRT.
You set up by moving the cursor to the physical devices you want to assign by using the cursor key or the space bar.
The number of connected disk drives is specified at the first line.
When copying, etc. is executed by the COPYDISK command by only one disk drive, 1 must here be specified as the number of drives.
Pressing the CR key or the ESC key will fix the setting and return the display to the main menu.

(4) SETUP [FLOPPY DISK]

```
S E T U P   [ FLOPPY DISK ]

READ AFTER WRITE:  ON   [OFF]
   STEPPING RATE:  [6ms] 12ms 20ms 30ms

Use Arrow Key, CR Key and ESC Key
[DIR↓]  [DIR B:↓]  [TYPE]  [REN]  [< NEXT >]  / 12:34:56 /  CAPS
```

* Enabling read after write operation results in somewhat slower processing speed, but greater reliability. If an error is detected during a read operation following a write, that write operation is repeated.
  At this menu, you can set up ON/OFF of the READ AFTER WRITE function.

* Stepping rate
  With MZ-800 P-CP/M, the stepping rate (seek time) of the drive can be changed.
  The usual setting is to 6ms, but, if an application program, etc. on a disk of a format other than the MZ-800 P-CP/M format cannot be accessed, it may be possible to access it by changing this (stepping rate).
  To change the setting, move the cursor position by pressing either the cursor key or the space bar.
  Pressing the CR key or the ESC key will renew the setting and return the display to the main menu.

(5) SETUP [KEY CLICK SOUND]

```
S E T U P      [ KEY CLICK SOUND ]

KEY CLICK SOUND :  [ON]  OFF

Use Arrow Key, CR Key and ESC Key
[DIR*] [DIR B:*] [TYPE] [REN] [< NEXT >] / 12:34:56 / CAPS
```

This function enables you to select wether or not a clicking sound accompanies each key depression.
Use the cursor key or the space bar to set up, and the CR key or the ESC key to return to the main menu.

(6) SETUP [MSB MASK]

```
S E T U P    [ MSB MASK ]

CONSOLE IN:     MASK [UNMASK]
CONSOLE OUT:    MASK [UNMASK]
AUX IN:         MASK [UNMASK]
AUX OUT:        MASK [UNMASK]
LIST OUT:       MASK [UNMASK]

Use Arrow Key, CR Key and ESC Key
[DIR:] [DIR B::] [TYPE] [REN] [< NEXT >] / 12:34:56 / CAPS
```

MZ-800 P-CP/M has the ability to mask the most significant bit (MSB) of the BIOS I/O data so that it can function normally even when you use application programs adding parity bits to 7-bit ASCII code in the console I/O.
You can specify whether or not you mask the MSB.
When the "MSB MASK:" is ON, the MSB is ignored and only the other 7 bits are entered as the effective data.
When the "MSB MASK:" is OFF, all 8 bits are regarded as effective data.
Use the cursor key or the space bar to set up, and the CR key or the ESC key to return to the main menu.

(7) SETUP [PRINTER MODE]

```
S E T U P   [ PRINTER MODE ]

PRINTER CODE:   [MZ-CODE]   ASCII-CODE
CR/LF CANCEL:   ALL OUT     CR OFF   [LF OFF]

Use Arrow Key, CR Key and ESC Key
[DIR↓] [DIR B:↓] [TYPE] [REN] [< NEXT >] / 12:34:56 / CAPS
```

With the MZ-800 it is possible to connect either of the following two types of printers.
One is the group of printers developed only for the MZ-800, such as the MZ-1P16 and MZ-80P5, and the other is the group of printers of the Centronics standard which uses regular ASCII characters.
Regarding these two groups, there are some differences in the character codes such as ASCII small letters. Therefore, the MZ-800 P-CP/M changes the output code system according to the printer type used.
At this menu, you can specify either of the two code systems. They are defined as the MZ-CODE and ASCII CODE.

Regarding the new line, either of 3 designations can be made: CR + LF, LF only or CR only.
(CR: Carriage Return; LF: Line Feed)
Use the cursor key or the space bar to set up, and the CR key or the ESC key to return to the main menu.

NOTE:
If the TAB code (09H) is output to the MZ-80KP5, MZ-1P16 printer, etc., note that this may operate as a function code, so care should be taken.

(8) SETUP [RS-232C PARAMETERS]

```
S E T U P    [ RS232C PARAMETERS ]

[ R S - 1 ]
     WORD LENGTH:    5    6    7    [8]
          PARITY:    EVEN ODD  [NON]
        STOP BIT:    1    1.5  [2]
[ R S - 2 ]
     WORD LENGTH:    5    6    7    [8]
          PARITY:    EVEN ODD  [NON]
        STOP BIT:    1    1.5  [2]

Use Arrow Key, CR Key and ESC Key
[DIR:] [DIR B:*] [TYPE] [REN] [< NEXT >] / 12:34:56 / CAPS
```

At this menu, you can assign the two RS-232C interfaces RS-1 (CH.A) and RS-2 (CH.B) and set up the parameters for the system to which the optional RS-232C interface board (MZ-8BIO3) is connected.
The items you can specify in this display are WORD LENGTH, PARITY (EVEN, ODD or NON) and the length of the STOP BIT.
Some parameters such as BAUD RATE are set up by hardware switchs on the MZ-8BIO3 board. For setting those parameters, refer to the manual for the optional RS-232C interface board (MZ-8BIO3).
At this menu, like at the others, select the item you want to set up with the cursor keys (↑ or ↓), and then select the parameter value with the cursor keys (← or →) or the space bar. The parameter is set to the value displayed in reverse video.
After all settings are completed, return the display to the main menu by pressing the CR key or the ESC key.

(9) SETUP [USER DEFINABLE KEY]

```
S E T U P     [ USER DEFINABLE KEYS ]

SYSTEM LINE DISPLAY:     ON     OFF

F 1:  DIR↓                                  BLANK: H
F 2:  DIR B:↓                     SHIFT+BLANK: H
F 3:  TYPE                              T A B: I
F 4:  REN                         SHIFT+T A B: I
F 5:  ERA                               INST: R
F 6:  STAT                        SHIFT+ INST: Z
F 7:  PIP                               D E L: H
F 8:  PIP A:=B:*.*[V]             SHIFT+D E L: ^
F 9:  FORMAT                                ↑ : K
F10:  COPYSYS                     SHIFT+    ↑ : K
F11:  SETUP                                 ↓ : J
F12:  FILES↓                      SHIFT+    ↓ : J
F13:  ED                                    → : L
F14:  ASM                         SHIFT+    → : L
F15:  DDT                                   ← : H
F16:  DISKDEF                     SHIFT+    ← : H

Use Arrow Key, CR Key and ESC Key

DIR↓   DIR B:↓   TYPE   REN   < NEXT >   / 12:34:56 /   CAPS
```

At this menu, you can specify the functions of the 16 user definable keys of the MZ-800 and can also specify ON/OFF for display of the system line on the 25th line of the CRT.
If you specify ON at SYSTEM LINE DISPLAY, the system line on the lowest line (the 25th line) of the CRT screen is displayed and the contents of the five function keys, the clock, and the condition CAPS LOCK are displayed.
When you specify OFF, this line is not displayed. (However, this line is preserved as the system message area for P-CP/M, so the user cannot use it, it is used by P-CP/M so that if an error occurs, or if it is necessary to exchange a floppy disk in a single drive system, the messages from the P-CP/M appear here.) Under the SYSTEM LINE DISPLAY: the line is displayed in the setting of the 32 user definable keys.
To change the contents of these key settings, move the cursor to the position of the relerant key with the cursor keys (↓, ↑, → and ←), and enter the content to be set on the 24th line of the CRT from the keyboard. The available keys are alphabet keys, symbol keys, the CR key, the control key + alphabet keys, etc. You cannot enter the cursor keys, user definable keys or graphic characters.

EXAMPLE:

ENTER STRING xx

At this prompt you can keyin the character string you wish to be assigned to key xx

The valid keys are described on the previous page and may be edited or terminated using the keys below

[←] or ^H	deletes one character (if no characters are input, this acts as the [F4] key.)
[CR]	this signifies the end of input for key xx
[↓]	this input is the [CR] code
[INST]	Hexadecimal input mode*
[ESC]	Abort current input

Control codes are displayed in reverse video, and the CR code is displayed as [↓]

* In hexadecimal mode only 0 ~ 9, A ~ F, and a ~ f are valid, characters other than this will cause a warning buzzer to sound, and the MZ-800 will exit hexadecimal mode.
"[?]" is displayed by pressing the [INST] key.

When all settings are complete, press the [ESC] key to return to the SETUP main menu.

(10) SETUP [END OF PROCESS]

```
S E T U P    [ END OF PROCESS ]

        End of SETUP process

    1  UPDATE SYSTEM DISK AND RUN
    2  RUN UNDER NEW SYSTEM (NO UPDATE SYSTEM DISK)
    3  NO UPDATE (EXIT TO P-CP/M)
    4  RETURN TO MAIN MENU

Use Arrow Key and CR Key or 1,2,3,4 Key
DIR↓  DIR B:↓  TYPE  REN  < NEXT >  / 12:34:56 /  CAPS
```

When you are using the SETUP command, you can setup or change the various parameters P-CP/M on the nine menus.
However, it is not until you select either
1 UPDATE SYSTEM DISK AND RUN
or
2 RUN UNDER NEW SYSTEM (NO UPDATE SYSTEM DISK)
on this END OF PROCESS screen that the new contents you setup on each screen become effective.
If you select 1, the new system will be written the MZ-800 system disk inserted into drive A and will be effective. If you select 2, the new system will also be available. However, as it will not be written to disk, it will be cleared with POWER OFF.
If you select
3 NO UPDATE (EXIT TO P-CP/M)
all contents you set up will be ignored and the system will return to P-CP/M.
If you select
4 RETURN TO MAIN MENU
the display will return to the main menu.

DISKEDIT Command

Syntax: DISKEDIT {d:}

Note:
{d:} is the entered drive name (target drive) of the disk to edit. If omitted, the target drive is the currently logged disk drive.

Explanation: When a command is input, the target drive disk parameters are read in, displayed on the screen, and we go to the DISKEDIT command stand-by condition.

```
D I S K - E D I T  utility for MZ-800 P-CP/M [V1.CA] (C)SHARP Corporation
Target disk                Sector/track :  32          Disk  size:    312 kbytes
  A: 2DMZ800               Directory max:  64 entries  Block size:   2048 bytes

COMMAND:  D/ir  B/lock  F/ile  S/ector  T/arget  [ESC]Q/uit: B

[E D I T] [RANDOM] [        ] [Q U I T] [        ]  / 00:11:22 /  CAPS
```

There are 6 types of DISKEDIT commands, as shown on the screen at the time of start; they are as described below.

Command	Contents
[D]	Display and edit directory.
[B]	Display and edit disk content as designated by block no.
[F]	Display and edit disk content as described by file name.
[S]	Display and edit disk content as designated by track sector.
[T]	Target, this command specified target drive.
[ESC] [Q]	Finish DISKEDIT and return to system.

(1) DIR command

This command is for the display and change of CP/M format directory data.
In the DISKEDIT command stand-by condition, press [D].
The directory's initial sector's data is then displayed in hexadecimal and ASCII characters.

```
D I S K - E D I T  utility for MZ-800 P-CP/M [V1.CA] (C)SHARP Corporation

Target disk                Sector/track :  32          Disk  size:    312 kbytes
  A: 2DMZ800               Directory max:  64 entries  Block size:   2048 bytes

SECTOR:  /Next  /Last  E/dit  R/andom  [ESC]/Quit

Drive: A Track:  1    Sector:  0
         00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

0000     00 50 43 50 4D 20 20 20 20 53 D9 53 00 00 00 70     .PCPM....SYS...p
0010     01 02 03 04 05 06 07 00 00 00 00 00 00 00 00 00     ................
0020     00 50 49 50 20 20 20 20 20 43 4F 4D 00 00 00 3A     .PIP.....COM...:
0030     00 09 0A 0B 08 00 00 00 00 00 00 00 00 00 00 00     ................
0040     00 41 53 4D 20 20 20 20 20 43 4F 4D 00 00 00 40     .ASM.....COM...@
0050     0C 0D 0E 0F 00 00 00 00 00 00 00 00 00 00 00 00     ................
0060     00 45 44 20 20 20 20 20 20 43 4F 4D 00 00 00 34     .ED......COM...4
0070     10 11 12 13 00 00 00 00 00 00 00 00 00 00 00 00     ................

[E D I T] [RANDOM] [        ] [Q U I T] [        ]   / 00:11:22 /  CAPS
```

This is now the sector data display change mode, it is possible to change the sector displayed at will. At this stage, commands are input as follows. Note, however, that care should be taken regarding changes of directory data.

SECTOR: [↓] /Next [↑] /Last E/dit R/andom [ESC] Q/uit:

When [N], [CR] or [↑] is pressed, the next sector is displayed.
When [L] or [↓] is pressed, the previous sector is displayed.
When [E] or [F1] is pressed, the mode changes to the screen edit mode.
When [Q] , [ESC] or [F4] is pressed, the condition will return to the DISKEDIT command stand-by menu.

For information concerning the sector display change mode, refer to (8).

(2) BLOCK Command

This command is used to edit, by hexadecimal and ASCII character display, the disk are a designated by the CP/M block number.
In the DISKEDIT command stand-by condition, press [B].

COMMAND: D/ir B/lock F/ile S/ector T/arget [ESC]Q/uit: B
A: BLOCK NUMBER:

When that is done, the block number is requested as shown in the above, key in the block number as a decimal value and then press [CR]. The data of the first record of the designated block number is then displayed.

```
D I S K - E D I T  utility for MZ-800 P-CP/M [V1.0A] (C)SHARP Corporation

Target disk                 Sector/track :  32            Disk  size:    312 kbytes
  A: 2DMZ800                Directory max:  64 entries    Block size:   2048 bytes

COMMAND:  D/ir  B/lock  F/ile  S/ector  T/arget  [ESC]Q/uit: B

FILE: [↓]/Next [↑]/Last  E/dit  R/andom  [ESC]/Quit

A: BLOCK DUMP

        00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F     [ RECORD:    0 ]

0000    0D 0A 0D 0A 20 20 20 20 20 20 20 20 20 20 20 20     ....
0010    20 20 20 20 3B 09 44 75 6D 70 20 70 72 6F 67 72         ;.Dump progr
0020    61 6D 2C 20 72 65 61 64 73 20 69 6E 70 75 74 20     am,reads input
0030    66 69 6C 65 20 61 6E 64 20 64 69 73 70 6C 61 79     file and display
0040    73 20 68 65 78 20 64 61 74 61 0D 0A 20 20 20 20     s hex data..
0050    20 20 20 20 20 20 20 20 20 20 20 20 3B 0D 0A 20                 ;..
0060    30 31 30 30 20 20 20 20 20 20 20 20 20 20 20 09     0100            .
0070    6F 72 67 09 31 30 30 68 0D 0A 20 30 30 30 35 20     org.100h.. 0005

[E D I T] [RANDOM] [        ] [Q U I T] [        ]          / 00:11:22 /  CAPS
```

You are now in the block record data display change mode, it is now possible to change the data display as required.
In this condition, commands are input as follows.

FILE: [↓]/Next [↑]/Last E/dit R/andom [ESC]/Quit:

When [N], [CR] or [↓] is pressed, the next record will be displayed.
When [L] or [↑] is pressed, the previous record will be displayed.
When [E] or [F1] is pressed, the mode will change to the screen edit mode.
When [R] or [F2] is pressed, the record number can be designated.
When [Q] , [ESC] or [F4] is pressed, the condition will return to DISKEDIT command stand-by.

For information concerning the block record display change mode, refer to "7. Block record display change mode."

(3) FILE Command

This command is for the display and alteration of a CP/M file content.
In the DISKEDIT command stand-by condition, press [F].
Next, the filename will be requested, so input the filename to be displayed. The file's header record will then be displayed.

COMMAND: D/ir B/lock F/ile S/ector T/arget [ESC]Q/uit:B
File name:

The file's header record will then be displayed.

```
D I S K - E D I T  utility for MZ-800 P-CP/M [V1.0A] (C)SHARP Corporation
Target disk          Sector/track :  32          Disk  size:    312 kbytes
  A: 2DMZ800         Directory max:  64 entries  Block size:   2048 bytes

FILE: [↓]/Next [↑]/Last  E/dit  R/andom  [ESC]/Quit
A: DUMP    ASM

        00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F     [ RECORD:    0 ]

0000    3B 09 44 75 6D 70 20 70 72 6F 67 72 61 6D 2C 20     ;.Dump.program,
0010    72 65 61 64 73 20 69 6E 70 75 74 20 66 69 6C 65     records input
0020    20 61 6E 64 20 64 69 73 70 6C 61 79 73 20 68 65      and displays he
0030    78 20 64 61 74 61 0D 0A 3B 0D 0A 09 6F 72 67 09     x data..;...org.
0040    31 30 30 68 0D 0A 62 64 6F 73 09 65 71 75 09 30     100h..bdos.equ.0
0050    30 30 35 68 09 3B 64 6F 73 20 65 6E 74 72 79 20     005h.;dos entry
0060    70 6F 69 6E 74 0D 0A 63 6F 6E 73 09 65 71 75 09     point..cons.equ.
0070    31 09 3B 72 65 61 64 20 63 6F 6E 73 6F 6C 65 0D     1.;read console.

[E D I T] [RANDOM] [        ] [Q U I T] [        ]  / 00:11:22 /  CAPS
```

You are now in the block record data display change mode, so it is now possible to change the data displayed as required. At this point, commands are input as follows.

* When [N], [CR] or [↓] is pressed, the next record will be displayed.
* When [L] or [↑] is pressed, the previous record will be displayed.
* When [E] or [F1] is pressed, the mode will change to the screen edit mode.
* When [R] or [F2] is pressed, the record number can be designated.
* When [Q], [ESC] or [F4] is pressed, the condition will return to DISKEDIT command stand-by.

For information concerning the block record data display change mode, refer to "(7) Block record display change mode."

(4) SECTOR Command

This command is a command which can be used to directly designate the logical sector of the disk and display that sector. In the DISKEDIT command stand-by condition, press [S] .

COMMAND: D/ir B/lock F/ile S/ector T/arget [ESC]Q/uit:

Track :1
Sector:2

Next, the track number and sector will be requested, input them as decimal values.
When the track and sector values are input, the sector content will be displayed.

```
D I S K - E D I T  utility for MZ-800 P-CP/M [V1.0A] (C)SHARP Corporation

Target disk          Sector/track :  32          Disk  size:    312 kbytes
  A: 2DMZ800         Directory max:  64 entries  Block size:   2048 bytes

SECTOR:  /Next  /Last  E/dit  R/andom  [ESC]/Quit

Drive: A Track:  1    Sector:  0
         00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

0000     00 50 43 50 4D 20 20 20 20 53 D9 53 00 00 00 70     .PCPM....SYS...p
0010     01 02 03 04 05 06 07 00 00 00 00 00 00 00 00 00     ................
0020     00 50 49 50 20 20 20 20 20 43 4F 4D 00 00 00 3A     .PIP.....COM...:
0030     00 09 0A 0B 08 00 00 00 00 00 00 00 00 00 00 00     ................
0040     00 41 53 4D 20 20 20 20 20 43 4F 4D 00 00 00 40     .ASM.....COM...@
0050     0C 0D 0E 0F 00 00 00 00 00 00 00 00 00 00 00 00     ................
0060     00 45 44 20 20 20 20 20 20 43 4F 4D 00 00 00 34     .ED......COM...4
0070     10 11 12 13 00 00 00 00 00 00 00 00 00 00 00 00     ................

[E D I T] [RANDOM] [        ] [Q U I T] [        ]  / 00:11:22 /  CAPS
```

You are now in the sector data display change mode, so it is now possible to change the data displayed as required.
In this condition, commands are input as follows.

SECTOR: [↓]/Next [↑]/Last E/dit R/andom [ESC]Q/uit:

* When [N] or [↓] is pressed, the next sector will be displayed.

* When [L] or [↑] is pressed, the previous sector will be displayed.

* When [E] or [F1] is pressed, the mode will changed to the screen edit mode.

* When [Q] , [ESC] or [F4] is pressed, the mode will return to the DISKEDIT command stand-by mode.

For information concerning the sector data display change mode, refer to "(8) Sector display change mode."

(5) TARGET Command

This command is to specify the destination drive. The target drive will be requested after selection of the T command.

COMMAND:

D/ir B/lock F/ile S/ector T/arget [ESC]Quit: T

TARGET DRIVE (A/B/E):

(6) QUIT Command

This command is to Quit DISKEDIT and to return to the system (P-CP/M).
At the DISKEDIT command stand-by condition, press [Q], [ESC] or [F5].

COMMAND: D/ir B/lock F/ile S/ector T/arget [ESC]Quit: Q

The display will be clear and the MZ-800 will return to the system (P-CP/M).

(7) Block record display change mode

When data designation by the BLOCK command or the FILE command is completed, the mode changes to the block record display change mode.

```
D I S K - E D I T  utility for MZ-800 P-CP/M [V1.0A] (C)SHARP Corporation

Target disk          Sector/track :  32          Disk  size:    312 kbytes
  A: 2DMZ800         Directory max:  64 entries  Block size:   2048 bytes

FILE: ↓/Next ↑/Last  E/dit  R/andom  ESC/Quit
A: DUMP    ASM

       00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F     [ RECORD:    0 ]

0000   3B 09 44 75 6D 70 20 70 72 6F 67 72 61 6D 2C 20     ;.Dump.program,
0010   72 65 61 64 73 20 69 6E 70 75 74 20 66 69 6C 65     records input
0020   20 61 6E 64 20 64 69 73 70 6C 61 79 73 20 68 65      and displays he
0030   78 20 64 61 74 61 0D 0A 3B 0D 0A 09 6F 72 67 09     x data..;...org.
0040   31 30 30 68 0D 0A 62 64 6F 73 09 65 71 75 09 30     100h..bdos.equ.0
0050   30 30 35 68 09 3B 64 6F 73 20 65 6E 74 72 79 20     005h.;dos entry
0060   70 6F 69 6E 74 0D 0A 63 6F 6E 73 09 65 71 75 09     point..cons.equ.
0070   31 09 3B 72 65 61 64 20 63 6F 6E 73 6F 6C 65 0D     1.;read console.

E D I T  RANDOM                  Q U I T                   / 00:11:22 /  CAPS
```

In this condition, there are 5 commands:

* When N or ↓ is pressed, the next record will be displayed.

* When L or ↑ is pressed, the previous record will be displayed.

* When E or F1 is pressed, the mode will change to the screen edit mode.

* When R or F2 is pressed, the record number can be designated.

* When Q or F4 is pressed, the condition will return to DISKEDIT command stand-by.

For the N and L commands, the command is executed when the key is pressed; for the E and R commands however, the operation continues after the commend; the following is an explanation of the E and R commands.

(7-1) E command (screen edit mode)

When [E] is pressed during the file record display change mode, the mode changes to the screen edit mode.
Record data is displayed in hexadecimal and characters, and editing can then be performed.

EDIT: [→]/forward [←]/backward [↑]/up [↓]/down [INST]/hex [DEL]/char [ESC]/exit:

Editing is by the screen-edit format, and cursor movement is controled by using the following control keys.
The editor is a screen editor, controlled using the keys below.

[→] Moves the cursor to the right

[←] Moves the cursor to the left

[↑] Moves the cursor upward

[↓] Moves the cursor downward

[SHIFT] + [←] Hexadecimal input mode

[SHIFT] + [→] Character input mode

When data is changed, both displays change, no matter whether the editing is done in the character or hexadecimal fields.
Note that data is two digits if hexadecimal, and one character if the input is character input.
When editing is finished, press the [ESC] key.
Then you will be asked what you want to do with the finalized data.

End of edit: [INST]/Write [DEL]/Return to edit [ESC]/Quit:

Then:

If [W] is pressed, the amended data will be written to disk and the mode will return to the screen edit mode.
If [R] is pressed, the mode will return to the screen edit mode.
If [Q] is pressed, the mode will return to the block record data display change mode, but the record data will not be written to disk.

(7-2) R command (to designate random record)

When [R] or [F2] is pressed during the file record display change mode, it is possible to designate a random record for editing.

Record number:

(8) Sector display change mode

When data designation by the DIR command or the SECTOR command is completed, the mode changes to the sector display change mode.

```
D I S K - E D I T  utility for MZ-800 P-CP/M [V1.0A] (C)SHARP Corporation

Target disk          Sector/track :  32          Disk  size:    312 kbytes
  A: 2DMZ800         Directory max:  64 entries  Block size:   2048 bytes

SECTOR:  /Next  /Last  E/dit  R/andom  [ESC]/Quit

Drive: A Track:  1    Sector:  0
        00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

0000    00 50 43 50 4D 20 20 20 20 53 D9 53 00 00 00 70    .PCPM....SYS...p
0010    01 02 03 04 05 06 07 00 00 00 00 00 00 00 00 00    ................
0020    00 50 49 50 20 20 20 20 20 43 4F 4D 00 00 00 3A    .PIP.....COM...:
0030    00 09 0A 0B 08 00 00 00 00 00 00 00 00 00 00 00    ................
0040    00 41 53 4D 20 20 20 20 20 43 4F 4D 00 00 00 40    .ASM.....COM...@
0050    0C 0D 0E 0F 00 00 00 00 00 00 00 00 00 00 00 00    ................
0060    00 45 44 20 20 20 20 20 20 43 4F 4D 00 00 00 34    .ED......COM...4
0070    10 11 12 13 00 00 00 00 00 00 00 00 00 00 00 00    ................

[E D I T] [RANDOM] [        ] [Q U I T] [        ]   / 00:11:22 /  CAPS
```

In this mode, there are 4 commands:

SECTOR: [↓]/Next [↑]/Last E/dit R/andom [ESC]Q/uit:

* When [N] or [↓] is pressed, the next sector will be displayed.

* When [L] or [↑] is pressed, the previous sector will be displayed.

* When [E] or [F1] is pressed, the mode will change to the screen edit mode.

* When [Q] or [F4] is pressed, the MZ-800 will return to DISKEDIT command stand-by.

The operation of these commands is exactly the same as for the block record display change mode, so please refer to that section.

DISKDEF Command

Syntax: DISKDEF

Explanation: The DISKDEF command enables the MZ-800 to read disks with formats other than that of the MZ-800 P-CP/M.

Example: A CP/M disk for the MZ-3500 in drive B may be read as follows.
The DISKDEF utility is started by typing

A > DISKDEF

The following screen is then displayed.

```
D I S K D E F   MZ-800 P-CP/M DISK DEFINE UTILITY Ver 1.0A (C) SHARP Corp.

Drive                                CP/M format

 A:   2DMZ800

 B:   2DMZ800   1DIBMPC   2DIBMPC   2DMZ80B   2DMZ3500 2DMZ5500

Use Arrow key to select and CR or ESC key to exit
 DIR↓   DIR B:↓   TYPE   REN   < NEXT >   / 12:34:56 /   CAPS
```

Here, 1D indicates single-sided, double-density disks and 2D indicates double-sided, double-density disks.
Disks which may be read are as shown above.
Drive A must be the P-CP/M format of the MZ-800; it can not be used in any other format video.
The CP/M format now set is displayed in reverse video.
When the CP/M disk format used is changed, the cursor is moved to the name of the format required.

TIME Command

Syntax: TIME

Explanation: The TIME command is used to check or set the time.

Example:

```
A > TIME
Current time is 01:49:57
Enter new time:
```

To set the time, the format for specification of the time is as follows;

HH:MM:SS (Hour:Minutes:Second)

Minutes and seconds can be omitted; when omitted, the time in set to 00 minutes 00 seconds.
If you enter CR only, the time will not be changed.

If the time is not correctly specified, the following message will be displayed.

```
Enter new time:666666
Invalid time

Enter new time:
```

FILES Command

Syntax: FILES {filespec}

Explanation: This displays the names and size of files satisfying the specified filespec, then displays the number of such files, the amount of disk space used, and the amount of free disk space. The FILES command differs from the DIR command in that files are displayed in alphabetical order, and that the number of bytes used is displayed.

Examples: **A > FILES B:**

```
                    Directory  of  disk B:
ASM      COM   8k  :DISKEDIT COM  10k  :EJECT    COM   2k  :SETUP    COM  10k
COPYDISK COM   4k  :DUMP     ASM   4k  :FILES    COM   2k  :STAT     COM   6k
COPYSYS  COM   2k  :DUMP     COM   2k  :FORMAT   COM   2k  :TIME     COM   4k
DDT      COM   6k  :DUMP     HEX   2k  :LOAD     COM   2k  :
DEL      COM   2k  :DUMP     PRN   8k  :PCPM     SYS  16k  :
DISKDEF  COM   6k  :ED       COM   8k  :PIP      COM   8k

Used:  21File(s) ,  116Kbytes        Free:  43Directory space ,  196 K bytes
```

(Example)

* The number of the total bytes used is always displayed even if the type of files is limited.

DEL Command

Syntax: DEL {filespec}

Explanation: This command deletes files in the same manner as the ERA command (see the P-CP/M user's guide); however, it differs in that it always requests confirmation before actually deleting the specified file.
When this is executed, the file specification for each applicable file is displayed on the CRT screen to request confirmation that it is to be deleted.
If a file is to be deleted, press the Y key; if it is not to be deleted, press the space bar. After this has been done for all applicable files, the DEL program displays the final request for confirmation. At this time, pressing Y causes all specified files to be deleted from the disk.

Examples:

```
A>DEL B:SAMPLE.*

B:SAMPLE.COM? Y
B:SAMPLE.TXT?
B:SAMPLE.LIB? Y
B:SAMPLE.BSD?
B:SAMPLE.BTX? Y
******** Are you sure? (Y/N):Y
```

The example above results in deletion of SAMPLE.LIB and SAMPLE.BTX from the disk in drive B:.

EJECT Command

Syntax: EJECT

Explanation: This command outputs a form feed code (OCH) to the printer or other LIST device.

Examples: **A > EJECT**

11. MEMORY DISK

The memory disk function enables the use of the RAM FILE (MZ-1R18, optional) in the same manner as a floppy disk.
The memory disk is accessed as logical drive "E:."
Since access to memory does not involve any of the waiting which is involved in access to a disk drive, using the memory disk enables much faster file access.

Before the memory disk can be used, the file to be processed must be transferred to drive E: from a real disk.

A > PIP E:=A:TEST.*

Data must also be transferred from the memory disk to a real disk after processing has been completed.

A > PIP A:=E:TEST.*

If the RAM FILE is not connected, the following message will be displayed.

CP/M Error On E: Invalid drive

NOTE:
When using the memory disk, remember that all data in the RAM is lost when the power is switched OFF.

12. SPECIAL KEYS

Key	Function
TAB	Tab key This key moves the cursor in units of 8 characters.
SHIFT	Shift key If the SHIFT key is pressed when CAPS is displayed in reverse video, lower-case letters are entered. If the SHIFT key is pressed when CAPS is displayed normally, upper-case letters are entered.
ALPHA	Functions as the CAPS LOCK key. It functions as a toggle key: when pressed once, it displays CAPS is reverse video and enters upper-case letters; when pressed again, it cancels the reverse mode of CAPS and enters lower-case letters.
CTRL	Control key When pressed together with other keys, this key inputs ASCII control codes or CRT control codes from the keyboard.
GRAPH	GRAPH key Inputs an escape code (1BH).
BREAK ESC	Same as GRAPH key. Inputs an escape code (1BH). Inputs CTRL-C (code 03H) when the SHIFT key is depressed.
INST	Insert key Inputs CTRL-R (code 12H).
DEL	Delete key Same as the cursor left-key.

Key	Function
SHIFT + INST	Clear screen Inputs CTRL-Z (code 1AH).
SHIFT + DEL	Cursor home Inputs CTRL-↑(code 1EH).
↑	Cursor-up key Inputs CTRL-K. The cursor does not move during the input of commands.)
↓	Cursor-down key Inputs CTRL-J.
→	Cursor-right key Inputs CTRL-L. (The cursor does not move during the input of commands.)
←	Cursor-Left key Inputs CTRL-H (backspace key).
□	Blank key Same as backspace key.

Personal CP/M ™

8-Bit Operating System

User's Guide

NOTICE TO USER

From time to time changes are made in the file names and in the files actually included with the distribution disk. This manual should not be construed as a representation or warranty that such files or facilities exist on the distribution disk or as part of the materials and programs distributed. Most distribution disks include a "READ.ME" file, which explains variations from the manual and which do constitute modification of the manual and the items included therewith. Be sure to read that file before using the software.

Foreword

Welcome to the world of microcomputers opened to you by your eight-bit microprocessor. Welcome also to the world of application software accessible with your Digital Research Personal CP/M™ operating system. Digital Research designed Personal CP/M especially for the Z80® or equivalent microprocessor that is the heart of your computer.

What Personal CP/M Does For You

Personal CP/M manages and supervises your computer's resources, including memory and disk storage, the console (screen and keyboard), printer, and communications devices. It also manages information stored magnetically on disks by grouping this information into files of programs or data. Personal CP/M can copy files from a disk to your computer's memory, or to a peripheral device such as a printer. To do this, Personal CP/M places various programs in memory and executes them in response to commands you enter at your console.

Once in memory, a program executes through a set of steps that instruct your computer to perform a certain task. You can use Personal CP/M to create your own programs, or you can choose from the wide variety of Personal CP/M application programs that entertain you, educate you, and help you solve commercial and scientific problems.

Distribution of Personal CP/M

Your Personal CP/M operating system can be contained in ROM (fixed in your system or on a plug-in capsule), or it can be contained on floppy disk. This depends entirely on how the manufacturer of your computer has decided to deliver it. To find out how your Personal CP/M has been delivered to you, read the instructions supplied by the manufacturer of your computer.

How This Guide Is Organized

This guide begins with simple examples, proceeds with basic concepts, then presents a detailed reference section on commands. The first four sections describe Personal CP/M operation for the first-time user. Section 1 introduces Personal CP/M and tells you how to start the operating system, enter commands and edit the command line. Section 2 explains files, disks, and drives. Section 3 describes how Personal CP/M manages your printer and console. Section 4 develops the concepts you need to use Personal CP/M commands. If you are new to CP/M, read the first four sections carefully for a general understanding of how to use Personal CP/M before you go on to the specific command descriptions.

Section 5 provides detailed information on each Personal CP/M utility program, arranged alphabetically. You will not use many of these programming utilities until you start writing your own Personal CP/M programs. Section 6 tells you how to use ED, the Personal CP/M file editor. With ED, you can create and edit program source codes, text, and some data files.

Section 7 discusses ASM operation and the various assembler options which may be enabled when invoking ASM.

Section 8 discusses the DDT program, which allows the user to test and debug programs interactively in the CP/M environment. Section 8 includes a DDT sample debugging session.

Appendix A lists Personal CP/M error messages and describes corrective action where necessary. Appendix B provides an ASCII to hexadecimal conversion table. Appendix C lists the filetypes associated with Personal CP/M. Appendix D lists and defines the Personal CP/M control characters. This guide concludes with a glossary of commonly used computer terms.

If you are new to computers, you might find some of the topics, such as the programming utilities, difficult to understand at first. Learning to use your computer is a challenge, and we hope you will find it fun. This book proceeds step by step so you can quickly proceed from setting up your system to mastering Personal CP/M's powerful facilities.

Table of Contents

Table of Contents
(continued)

Table of Contents
(continued)

Appendixes

Table and Figures

Tables

Figure

Section 1
Introduction to Personal CP/M

This section tells you how to start Personal CP/M. It describes the command line and tells you how to edit it; also, it tells you why you should back up your files.

HOW TO START Personal CP/M

If your computer's manufacturer has built Personal CP/M into your system, when you turn on your computer, Personal CP/M starts immediately. If your computer's manufacturer did not build Personal CP/M into the system, you must start Personal CP/M from your system disk, ROM, or cartridge--or from the device or media on which your computer manufacturer delivered Personal CP/M.

Starting or loading Personal CP/M means reading Personal CP/M from the device or media supplied with your computer into your computer's main memory.

In the following discussion it is assumed that Personal CP/M is supplied to you on disk. Consult the manual for your specific computer to determine the exact form in which Personal CP/M has been supplied to you.

First, check that your computer's power is on. Next, insert the Personal CP/M system disk into your initial drive. In this section, assume that the initial drive is A and the disk is removable. Close the drive door. Then restart your system. (In many cases this means pressing the RESET or RESTART button. But consult the manual for your computer to be sure. Your manual might also refer to restarting as resetting, cold booting, cold starting, or simply loading the system.) This automatically loads Personal CP/M into memory.

After Personal CP/M is loaded into memory, a message similar to the following is displayed on your screen:

```
P-CP/M80 for SHARP MZ-800

Copyright (C) 1984 Digital Research Inc. / SHARP Corporation
```

Note: Visual CP/M is a visually oriented version of Personal CP/M that prompts for commands and parameters--much as menu-driven programs prompt for names of programs and data to be entered. If Visual CCP automatically loads with your system, the screen formats and the manner of entering command lines are different.

The version number, represented by V.V, identifies the version of Personal CP/M that you own. After this display, the following two-character message appears on your screen:

A>

This is the Personal CP/M system prompt. The system prompt tells you that Personal CP/M is ready to read a command from your keyboard. In this example, the prompt also tells you that drive A is your default drive. This means that until you tell Personal CP/M to do otherwise, it looks for program and data files on the disk in drive A. Also, the absence of a user number tells you that you are logged in as user 0.

THE COMMAND LINE

Personal CP/M performs tasks according to commands you type at your keyboard. A Personal CP/M command line is composed of a command keyword, an optional command tail, and a carriage return keystroke. The command keyword identifies a command (program) to execute. The command tail can contain extra information for the command, such as a filename or parameters. To end the command line, press the carriage return or ENTER key. The following example shows a command line entered by a user.

A>**DIR MYFILE**

In this manual, the characters you type are shown in boldface to distinguish them from characters the system displays. In this example, DIR is the command keyword, and MYFILE is the command tail. The carriage return keystroke does not appear on the screen or in the example. You must remember to press the carriage return key to send a command line to Personal CP/M for processing. Note that the carriage return key might be marked ENTER, RETURN, CR, or something similar on your keyboard. In this guide, RETURN signifies the carriage return key.

As you type characters at the keyboard, they appear on your screen. The single-character position indicator, called the cursor, moves to the right as you type characters. If you make a typing error, press the BACKSPACE key (if your keyboard has one) or CTRL-H to move the cursor to the left and correct the error. CTRL is the abbreviation for the Control key. To type a control character, hold down the Control key and press the required letter key. For example, to move the cursor to the left, hold down CTRL and press the H key.

You can type the keyword and command tail in any combination of uppercase and lowercase letters. Personal CP/M treats all letters in the command line as uppercase.

You type a command line directly after the system prompt. However, Personal CP/M does allow spaces between the prompt and the command keyword.

Personal CP/M recognizes two types of commands: built-in commands and transient utility commands. Built-in commands execute programs that reside in memory as part of the Personal CP/M operating system. CP/M executes built-in commands immediately. Transient utility commands are stored on disk as program files. They must be loaded from disk to perform their task. You can recognize transient utility program files when a directory is displayed on the screen because their filenames are followed by COM. Section 4 includes lists of the Personal CP/M built-in and transient program commands.

For transient programs, Personal CP/M checks only the command keyword. If you include a command tail, Personal CP/M passes it to the utility without checking it because many utilities require unique command tails. A command tail cannot contain more than 128 characters. Personal CP/M cannot read either the command keyword or the command tail until you press the RETURN key.

The following command demonstrates how Personal CP/M reads command lines. The DIR (Directory) command tells Personal CP/M to display a directory of disk files on your screen. Type DIR after the system prompt, omit the command tail, and press RETURN.

```
A>DIR
```

Personal CP/M responds to this command by writing the names of all the files stored on the disk in drive A (with the exception SYS (System) files, which are explained later on). For example, if you have your Personal CP/M system disk in drive A, these filenames, among others, appear on your screen:

```
PIP     COM
STAT    COM
```

Personal CP/M recognizes only correctly spelled command keywords. If you make a typing error and press RETURN before correcting your mistake, Personal CP/M echoes the command line followed by a question mark. If you type the DIR command incorrectly, as in the following example, Personal CP/M responds

```
A>DJR
DJR?
```

to tell you that it cannot find the command keyword. To correct simple typing errors, use the BACKSPACE key, or hold down the CTRL key and press H to move the cursor to the left. Personal CP/M supports other control characters that help you efficiently edit command lines. Section 3 tells how to use control characters to edit command lines and other information you enter at your console.

DIR accepts a filename as a command tail. You can use DIR with a filename to see if a specific file is on the disk. For example, to check that the transient utility program STAT.COM is on your system disk, type

```
A>DIR STAT.COM
```

Personal CP/M performs this task by displaying the name of the file you specified, or the message, No File.

Be sure you type at least one space after DIR to separate the command keyword from the command tail. If you do not, Personal CP/M responds as follows:

```
A>DIRSTAT.COM
DIRSTAT.COM?
```

WHY YOU SHOULD BACK UP YOUR FILES

Humans make mistakes, and so do computers. Human or computer errors sometimes destroy valuable programs or data files. By typing a command incorrectly, for example, you can accidentally erase a program that you just created or a data file that has been months in the making. A similar disaster can result from an electronic component failure.

Data processing professionals avoid losing programs and data by copying (backing up) valuable files. Always make a working copy of any new program you purchase and save the original. If the program is accidentally erased from the working copy, you can easily restore it from the original.

It is also wise to make frequent copies of new programs or data files as you develop them. The frequency of making copies varies with each programmer. However, as a general rule, make a copy whenever it takes 10 to 20 times longer to reenter the information than it takes to make the copy.

So far we have not discussed commands that change recorded information on disks or other media. Before we do, if you received Personal CP/M on a disk, make a copy of it. To make a copy of a disk, follow the instructions provided by the manufacturer of your computer. If your system includes one or more disk drives, disk formatting and copy programs are included along with instructions on how to run them.

End of Section 1

Section 2
Files, Disks, and Drives

Your system might contain mass storage devices other than disks. It can contain bubble, RAM, or ROM memory supplied on fixed boards in the system or in plug-in capsules. If you can access the data on such a device in the same way you can access that data on a disk (that is, in groups of data, often called "blocks"), then the device is said to be a disk-like device. In the following discussion, you can substitute "disk-like device" wherever the word "disk" occurs.

Personal CP/M's most important task is to access and maintain files on your disks (or disk-like devices). With Personal CP/M you can create, read, write, copy, and erase files. This section tells you what a file is, how to create, name, and access a file, and how files are stored on your disks. It also explains how to change disks and change the default drive.

Although this section describes how files are stored on disks, you can apply the principles to files stored on any disk-like device.

WHAT IS A FILE?

A Personal CP/M file is a collection of related information stored on a disk. Every file must have a unique name because Personal CP/M accesses files by name. A directory is also stored on each disk. The directory contains a list of the filenames stored on the disk and the locations of each file on the disk.

Basically, there are two kinds of files: program (command) files and data files. A program file contains an executable program--a series of instructions that the computer follows step by step. A data file is usually a collection of information: a list of names and addresses, the inventory of a store, the accounting records of a business, the text of a document, or similar related information. For example, your computer cannot execute names and addresses, but it can execute a program that prints names and addresses on mailing labels.

A data file can also contain the source code for a program. A program source file must be processed by an assembler or compiler before it becomes a program file. In most cases, an executing program processes a data file. However, some executing programs can process a program file. For example, the copy program PIP can copy one or more program files.

HOW ARE FILES CREATED?

There are many ways to create a file. One way is to use a text editor. The Personal CP/M text editor ED (described in Section 6)

can create a file and assign it the name you specify. You can also create a file by copying an existing file to a new location, perhaps renaming it in the process. Under Personal CP/M, you can use the PIP command to copy and rename files. Finally, some programs such as ASM create output files as they process input files.

HOW ARE FILES NAMED?

Personal CP/M identifies every file by its unique file specification. A file specification can be a one- to eight-character filename, such as the following:

MYFILE

A file specification can have three parts: a drive specifier, a filename, and a filetype.

The drive specifier is a single letter (A-P) followed by a colon. Each drive in your system is assigned a letter. When you include a drive specifier as part of the file specification, you are telling Personal CP/M that the file is stored on the disk currently in that drive. For example, if you enter

B:MYFILE

Personal CP/M looks in drive B for the file MYFILE.

When you make up a filename, try to use a name that tells you something about the file's contents. For example, you might name a file containing a list of customer names for your business

CUSTOMER

As you begin to use your computer with Personal CP/M, your files will fall naturally into categories. To help you identify files belonging to the same category, Personal CP/M allows you to add an optional one- to three-character extension, called a filetype, to the filename. When you add a filetype to the filename, separate the filetype from the filename with a period. Use three letters that indicate the file's category. For example, you might add the following filetype to the file that contains a list of customer names:

CUSTOMER.NAM

When Personal CP/M displays file specifications in response to a DIR command, it adds blanks to short filenames so you can compare filetypes quickly. The program files Personal CP/M loads into memory from a disk have different filenames, but all have the filetype COM.

Create filenames and filetypes from letters and numbers. You must not use the following characters in filenames or filetypes because they have special meanings for Personal CP/M:

< > . , ; : = ? * [ ] - % | ( ) / \

A complete file specification containing all possible elements consists of a drive specification, a primary filename, and a filetype, each separated by its appropriate delimiter, as in the following example:

A:DOCUMENT.LAW

DO YOU HAVE THE CORRECT DRIVE?

When you type a file specification in a command tail without a drive specifier, the program looks for the file in the drive named by the system prompt, called the default drive. For example, if you type the command

A>**DIR STAT.COM**

DIR looks in the directory of the disk in drive A for STAT.COM. If you have another drive, B, for example, you must tell Personal CP/M to access the disk in B instead. For this reason, Personal CP/M lets you precede a filename with a drive specifier. For example, in response to the command

A>DIR B:**MYFILE.LIB**

Personal CP/M looks for the file MYFILE.LIB in the directory of the disk in drive B. When you give a command to Personal CP/M, note which disk is in the default drive. Many application programs require that the data files they access be stored in the default drive.

You can also precede a program filename with a drive specifier, even if you use the program filename as a command keyword. For example, if you type

A>**B:PIP**

Personal CP/M looks in the directory of the disk in drive B for the file PIP.COM. If Personal CP/M finds PIP on drive B, it loads PIP into memory and executes it.

To access many files on the same drive, you might find it convenient to change the default drive so that you need not repeatedly enter a drive specifier. To change the default drive, enter the drive specifier next to the system prompt and press RETURN. In response, Personal CP/M changes the system prompt to display the new default drive:

```
A>B:
B>
```

Unlike the filename and filetype, which are stored in the disk directory, the drive specifier for a file changes as you move the disk from one drive to another. So a file has a different file specification when you move a disk from one drive to another. Section 4 details how Personal CP/M locates program and data files.

DO YOU HAVE THE CORRECT USER NUMBER?

Personal CP/M further identifies all files by assigning each one a user number ranging from 0 to 15. Personal CP/M assigns the user number to a file when the file is created. User numbers allow you to separate your files into 16 file groups.

When you use a Personal CP/M utility to create a file, the file is assigned the current user number unless you use PIP to copy the file to another user number. You can determine the current user number by looking at the system prompt.

4A>	User number 4, drive A
A>	User number 0, drive A
2B>	User number 2, drive B

The user number always precedes the drive identifier. User 0, however, is the default user number and is not displayed in the prompt.

You can use the built-in command USER to change the current user number.

```
A>USER 3
3A>
```

Most commands can access only files that have the current user number. For example, if the current user number is 7, a DIR command displays only the files created under user number 7.

ACCESSING MORE THAN ONE FILE

Certain Personal CP/M built-in and transient utilities can select and process several files when special wildcard characters are included in the filename or filetype. A file specification containing wildcards is called an ambiguous filespec and can refer to more than one file because it gives Personal CP/M a pattern to match. Personal CP/M searches the disk directory and selects any file with a filename or filetype that matches the pattern.

The two wildcard characters are ?, which matches any single letter in the same position, and *, which matches any character at that position, and any other characters remaining in the filename or filetype. The following list presents the rules for using wildcards.

- A ? matches any character in a name, including a space character.
- An * must be the last, or only, character in the filename or filetype. Personal CP/M internally replaces an * with ? characters to the end of the filename or filetype.
- When the filename to match is shorter than eight characters, Personal CP/M treats the name as though it ends with spaces.
- When the filetype to match is shorter than three characters, Personal CP/M treats the filetype as though it ends with spaces.

Suppose, for example, you have a disk that contains the following six files:

A.COM, AA.COM, AAA.COM, B.COM, A.ASM, and B.ASM

The following wildcard specifications match all, or a portion of, these files:

.	is treated as ????????.???
????????.???	matches all six names
*.COM	is treated as ????????.COM
????????.COM	matches the first four names
?.COM	matches A.COM and B.COM
?.*	is treated as ?.???
?.???	matches A.COM, B.COM, A.ASM, and B.ASM
A?.COM	matches A.COM and AA.COM
A*.COM	is treated as A???????.COM
A???????.COM	matches A.COM, AA.COM, and AAA.COM

Remember, Personal CP/M uses wildcard patterns only when searching a disk directory, so wildcards are valid only in filenames and filetypes. You cannot use a wildcard character in a drive

specifier. Nor can you use a wildcard character as part of a filename or filetype when you create a file.

HOW TO PROTECT YOUR FILES

Under Personal CP/M you can organize your files into groups to protect them from accidental change. You can also specify how your files are displayed in response to a DIR command. Personal CP/M supports these features by assigning a user number and attributes to each file.

All of this information is recorded in the disk directory. File attributes control how programs access files. When you create a file, Personal CP/M gives it two attributes. You can change the attributes with a STAT command.

You can set the first attribute to DIR (Directory) or SYS (System). This attribute controls whether Personal CP/M displays the file's name in response to a DIR command. When you create a file, Personal CP/M automatically sets this attribute to DIR. You can display the name of a file marked with the DIR attribute with a DIR command. If you give a file the SYS attribute, the DIR command will not display the filename.

Note: To display a SYS file, use the STAT command with the command tail *.*. The DIR command displays only the filenames created under the current user number.

A file with the SYS attribute has a special advantage when it is created under user 0. When you give a file with user number 0 the SYS attribute, you can read and execute that file from any user number. This feature makes your commonly used programs available under any user number.

The second file attribute can be set to either R/W (Read/Write) or R/O (Read/Only). If a file is marked R/O, attempting to write data to that file produces a Read/Only error message. Therefore, you can use the R/O attribute to protect important files. A file with the R/W attribute can be read, written to, or erased at any time, unless the disk is physically write-protected.

HOW ARE FILES STORED ON A DISK?

Personal CP/M records the filename, filetype, user number, and attributes of each file in a special area of the disk called the directory. The Personal CP/M directory also records the location of each file on the disk.

Personal CP/M allocates directory and storage space for a file as you add records to the file. When you erase a file, Personal CP/M reclaims storage in two ways: it makes the file's directory space available to catalog a different file, and it frees the file's storage space for later use. This dynamic allocation feature makes Personal CP/M powerful. You need not tell Personal CP/M how big your file will become, because Personal CP/M automatically allocates more storage for a file as needed, and releases the storage for reallocation when the file is erased. Use the STAT command to find out how much space remains on the disk.

CHANGING FLOPPY DISKS

Personal CP/M cannot do anything to a file unless the disk that holds the file is inserted into a drive and the drive is ready. When a disk is in a drive, it is online and Personal CP/M can access its directory and files.

At some time, you must take a disk out of a drive and insert another that contains different files. You can replace an online disk whenever the system prompt appears on your console. The system prompt indicates that no program is reading or writing to the drive.

You can also remove a disk and insert a new one when an application program prompts you to do so. This can occur, for example, when the data that the program uses does not fit on one floppy disk.

Note: Never remove a disk while a program is reading or writing to it.

You can change disks on the drive without sending any special signals to Personal CP/M. You can insert a different disk at a program's request and read files from or create files on the new disk.

PROTECTING A DRIVE

Under Personal CP/M, drives can be marked R/O (Read/Only) just as files can be given the R/O attribute. The default state of a drive is R/W (Read/Write). You can give a drive the R/O attribute by using the STAT command described in Section 5. To return the drive to R/W, use the STAT command or press CTRL-C to return to the system prompt.

End of Section 2

Section 3
Console and Printer

This section describes how Personal CP/M communicates with your console and printer. It tells how to start and stop console and printer output, and how to edit commands you enter at your console.

CONTROLLING CONSOLE OUTPUT

Sometimes Personal CP/M displays information on your screen too quickly for you to read it. Sometimes an especially long display scrolls off the top of your screen before you have a chance to study it. To ask the system to wait while you read the display, hold down the Control (CTRL) key and press S. A CTRL-S keystroke causes the display to pause. When you are ready, press any other key to resume the display.

CONTROLLING PRINTER OUTPUT

You can also use a control command to echo console output to the printer. To start printer echo, press CTRL-P. To stop, press CTRL-P again. While printer echo is in effect, characters that appear on your screen are listed at your printer.

You can use printer echo with a DIR command to make a list of files stored on a floppy disk. You can also use CTRL-P with CTRL-S to make a hard copy of part of a file. Use a TYPE command to start a display of the file at the console. When the display reaches the part you want to print, press CTRL-S to stop the display and CTRL-P to enable printer echo. Then press any key to resume the display and start printing. Use another CTRL-S/CTRL-P sequence to terminate printer echo.

CONSOLE LINE EDITING

You can correct simple typing errors with the BACKSPACE key. Personal CP/M also supports additional line-editing functions that you perform with control characters. You can use the control characters to edit command lines or input lines to most programs.

Personal CP/M allows you to edit your command line using the characters listed in Table 3-1. To edit a command line in Personal CP/M, use control characters to delete characters left of the cursor, then replace them with new characters.

In the following example command line, the command keyword PIP is incorrectly typed. (The underbar represents the cursor.)

```
A>POP A:=B:*.*_
```

To move the cursor to the letter O, hold down the CTRL key and press the letter H 11 times. CTRL-H deletes characters as it moves the cursor left, leaving the following command line:

```
A>P_
```

Now type the correct letters and press RETURN, sending the command to Personal CP/M.

```
A>PIP A:=B:*.*_
```

Table 3-1 describes Personal CP/M control characters.

Table 3-1. Personal CP/M Control Characters

Character	Meaning
CTRL-C	Warm boot (restarts) the Personal CP/M operating system when typed at the beginning of a line.
CTRL-E	Forces a physical carriage return but does not send the command line to Personal CP/M. Moves the cursor to the beginning of the next line without erasing your previous input.
CTRL-H	Deletes a character and moves the cursor left one character position.
CTRL-J	Sends the command line to Personal CP/M and returns the cursor to the left of the current line. Has the same effect as a RETURN or a CTRL-M.
CTRL-M	Sends the command line to Personal CP/M and returns the cursor to the left of the current line. Has the same effect as a RETURN or a CTRL-J.
CTRL-R	Places a # sign at the current cursor location, moves the cursor to the next line, and displays any partial command you typed so far.
CTRL-U	Discards all the characters in the command line (but leaves them displayed), places a # at the current cursor position, and moves the cursor to the next command line.
CTRL-X	Discards all the characters in the command line (actually removes them from display), and moves the cursor to the beginning of the current line.
RUBOUT	Deletes the last character typed and echoes it at the console.

You probably noticed that some control characters have the same meaning. For example, the CTRL-J and CTRL-M keystrokes have the same effect as pressing the RETURN key; all three send the command line to Personal CP/M for processing. Also, CTRL-H has the same effect as pressing the BACKSPACE key.

End of Section 3

Section 4
Personal CP/M Command Concepts

As explained in Section 1, a Personal CP/M command line consists of a command keyword, an optional command tail, and a carriage return keystroke. This section describes the two kinds of programs the command keyword can identify and tells how Personal CP/M searches for a program file on a disk. This section also explains how to execute multiple Personal CP/M commands and how to terminate programs and reset the disk system.

TWO KINDS OF COMMANDS

A command keyword identifies a program that resides in memory as part of Personal CP/M, or on a disk as a program file. Commands that identify programs in memory are called built-in commands. Commands that identify program files on a disk are called transient commands.

Personal CP/M has six built-in commands and eight transient program commands. You can add programs to your system by purchasing Personal CP/M-compatible application programs. If you are an experienced programmer, you can also write your own programs that operate with Personal CP/M.

Built-in Commands

Built-in commands are part of Personal CP/M. You can always use them regardless of which disk you have in which drive. Built-in commands reside in memory as a part of Personal CP/M and therefore execute more quickly than the transient programs.

Section 5 explains in detail the built-in commands listed in Table 4-1.

Table 4-1. Built-in Commands

Command	Function
DIR	Displays filenames of all files in the directory except those marked with the SYS attribute.
ERA	Erases a filename from the disk directory and releases storage space occupied by the file.
REN	Renames a disk file.
SAVE	Stores a portion of main memory in a disk file.
TYPE	Displays contents of an ASCII (TEXT) file at your screen.
USER	Allows you to change to a different user number.

Transient Program Commands

When you enter a command keyword that identifies a transient program, Personal CP/M loads the program file from the disk and passes it any filenames, data, or parameters you entered in the command tail. Section 5 provides the operating details for the Personal CP/M transient programs listed in Table 4-2. These utilities are used only by experienced programmers.

Seven other transient programs are available: COPY, DUMP, RANDON, TERMINAL, ASM, LOAD, and DDT, COPY, DUMP, RANDOM, and TERMINAL are described in the Personal CP/M Programmer's Guide are included in this manual as sample programs supplied in source code only. If you assemble the source code, you can run these programs. ASM (the assembler program), LOAD (the loader), and DDT (the debugger) are described in this Manual.

Table 4-2. Transient Program Commands

Command	Action
ED	Loads and executes the CP/M text editor program; creates and alters character files.
PIP	Loads and executes the Peripheral Interchange Program, which copies, combines, or transfers files.
STAT	Displays statistical information including space in kilobytes occupied by a file; file attributes; disk status (Read/Only or Read/Write). Also allows you to set file attributes, and disk status.
SUBMIT	Executes a list of commands contained in a file.
XSUB	This additional utility program is used in conjunction with SUBMIT. XSUB extends the power of the SUBMIT facility to include line input to programs as well as to the console command processor.

HOW Personal CP/M SEARCHES FOR FILES

If Personal CP/M cannot find a program file you specified in a command line, Personal CP/M might not be looking on the drive on

which the file is stored. This section explains how Personal CP/M searches for program and data files.

Finding Data Files

When you enter a command line, Personal CP/M passes the command tail to the program identified by the command keyword. If the command tail contains a file specification, the program calls Personal CP/M to search for the data file. If Personal CP/M cannot find the data file, the program displays an error message at the console. Typically, this message is "File not found" or "No File," but the exact message depends on the program the command keyword identifies.

If you do not include a drive specifier with the filename in a command tail, Personal CP/M searches the directory of the current user number on the default drive. If the file is not there, Personal CP/M looks for the file with the SYS attribute in the directory of user 0 on the default drive. If Personal CP/M finds the file under user 0, it allows the program Read/Only access to the file. For example, if you enter the following command line:

```
3A>TYPE MYFILE.TXT
```

Personal CP/M first searches the directory for user 3 on drive A. If it does not find MYFILE.TXT there, it searches the directory of user 0 on drive A for MYFILE.TXT marked with the SYS attribute. If the file is not in either directory, Personal CP/M returns control to TYPE, which then displays "No File."

Some Personal CP/M utilities, such as PIP and DIR, restrict their file search to the current user number. Because Personal CP/M does not allow Read/Write access to SYS files, ERA and REN also restrict their search to the current user number.

The search procedure is basically the same if you include a drive specifier with the filename. Personal CP/M first looks in the directory of the current user number on the specified drive. Then, if it does not find the file, it looks in the directory for user 0 on the specified drive for the file with the SYS attribute. If Personal CP/M does not find the data file after these two searches, it displays an error message.

Finding Program Files

If a command keyword identifies a transient program, Personal CP/M looks for that program file on the default or specified drive. It looks under the current user number, and then under user 0 for the same file marked with the SYS attribute. At any point in the search process, Personal CP/M stops the search if it finds the program file. Personal CP/M then loads the program into memory and executes it. When a program terminates, Personal CP/M displays the system

prompt and waits for your next command. However, if Personal CP/M does not find the command file, it repeats the command line followed by a question mark, and waits for your next command.

When you include a drive specifier before the command keyword, you tell Personal CP/M to look on that drive for the program file. Personal CP/M then searches two locations: the directory for the current user on the specified drive, and then for user 0 on the specified drive, before it repeats the command line with a question mark. For example, if you enter

```
4C>A:STAT SPACE
```

Personal CP/M looks on drive A, user 4 and then user 0 for the file STAT.COM.

EXECUTING MULTIPLE COMMANDS

Personal CP/M can execute a sequence of commands. You can put a frequently needed sequence of commands into a disk file. Once you have stored the sequence in a disk file, you can execute the entire sequence with a single SUBMIT command.

Store command sequences you execute frequently in a disk file. To create this file, use ED or another character file editor. The file must have a filetype of SUB, and each command in the file must start on a new line. For example, an UPDATE.SUB file might look like this:

```
DIR A:*.COM
ERA B:*.COM
PIP B:=A:*.COM
```

To execute this list, enter the following command:

```
A>SUBMIT UPDATE
```

The SUBMIT utility passes each command to Personal CP/M for sequential execution. While SUBMIT executes, the commands usually echo at the console. When one command completes, the system prompt reappears with the next command in the SUB file or else reappears by itself, when the SUB file is exhausted. SUBMIT then waits for your next command from the keyboard.

The SUBMIT command is detailed in Section 5.

TERMINATING PROGRAMS

The two-keystroke command CTRL-C terminates program execution or resets the disk system. To enter a CTRL-C command, hold down the CTRL key and press C.

Not all application programs that run under CP/M terminate with a CTRL-C. However, you can terminate most of the transient programs supplied with Personal CP/M immediately with a CTRL-C keystroke. If you want to terminate a program that is sending a display to the screen, you might have to press CTRL-S to halt the display before entering CTRL-C.

CTRL-C also resets the disk system. This is called a warm boot or warm start. When you press CTRL-C and the cursor is at the system prompt, Personal CP/M logs out all the active drives, then logs in the default drive. The active drives are any drives you have accessed since the last cold or warm start. A STAT command displays the remaining space on all active drives. In the following example, STAT indicates that three drives are active. However, if you press CTRL-C immediately after this display and then enter another STAT command, only the space for the default drive, A, is displayed.

```
A>STAT
A: R/W,  Space:    9,488k
B: R/O,  Space:    2,454k
C: R/O,  Space:    1,665k
A>^C
A>STAT
A: R/W,  Space:    9,488k
```

End of Section 4

Section 5
Command Summary

This section describes the commands and programs supplied with your Personal CP/M operating system. The commands are listed alphabetically along with short explanations and examples.

ED is described in greater detail in Section 6. ASM (the assembler), LOAD (the loader), and DDT (the debugger) are described in the CP/M Manual. Five other programs--BLTMEMO, COPY, DUMP, RANDOM, and TERMINAL--are supplied only in source code. See the Personal CP/M Programmer's Guide for their descriptions.

LET'S GET PAST THE FORMALITIES

This section describes the parts of a file specification in a command line. A file specification names a file or group of files in the directory of the on-line disk given by the drive specifier. For example,

B:MYFILE.DAT

is a file specification that indicates drive B:, filename MYFILE, and filetype DAT. File specification is abbreviated

filespec

in the command syntax statements. The three parts of a file specification are

- drive specifier--the optional disk drive A, B, C, through P that contains the file or group of files to which you refer. If you include a drive specifier in your command line, a colon must follow it.
- filename--the one- to eight-character first name of a file or group of files.
- filetype--the optional one- to three-character category name of a file or group of files. A period must separate the filetype from the filename.

If you do not include a drive specifier, Personal CP/M automatically uses the default drive. If you omit the period and the filetype, Personal CP/M automatically includes a filetype of three blanks.

Some Personal CP/M commands accept wildcards in the filename and filetype parts of the command tail. For example,

B:MY*.A??

is a file specification with drive specifier B:, filename MY*, and filetype A??. This ambiguous file specification might match several files in the directory.

Put together, the parts of a file specification are represented in the following general form:

d:filename.typ

In the preceding form, d: represents the optional drive specifier, filename represents the one- to eight-character filename, and typ represents the optional one- to three-character filetype. The syntax descriptions in this section use the term filespec to indicate any valid combination of the elements included in the file specification. The following list shows valid combinations of the elements of a Personal CP/M file specification.

- filename
- filename.typ
- d:filename
- d:filename.typ

The following characters have special meaning in Personal CP/M, so do not use them except as specified in a description:

< > . , ; : = ? * [ ] - % | () /

Personal CP/M has established several file groups. Table 5-1 lists some of their filetypes with a short description of each family. Appendix C provides the complete list.

Table 5-1. Personal CP/M Filetypes

Filetype	Meaning
ASM	Assembler source file
BAS	CBASIC® source program
COM	Machine language program
HLP	HELP message file
SUB	List of commands to be executed by SUBMIT
$$$	Temporary file

In some commands, descriptive qualifiers are used with filespecs to further define the type of filespec accepted by the commands. For example, wildcard-filespec denotes wildcard specifications, dest-filespec denotes a destination filespec, and src-filespec denotes a source filespec.

You now understand command keywords, command tails, control characters, default drives, and wildcards. You also see how to use the formal names filespec, drive specifier, filename, and filetype. These concepts give you the background necessary to compose complete command lines.

HOW COMMANDS ARE DESCRIBED

Personal CP/M commands are presented in alphabetical order by command keyword. The command description format is as follows:

- The command keyword appears in uppercase.
- The syntax section gives you one or more general forms to follow when you compose the command line.
- The explanation section defines the command keyword and points out exceptions and special cases. Some explanations include tables or lists of options you can use in the command line.
- The examples section lists a number of valid command lines. To clarify examples of interactions between you and the operating system, the characters you enter are shown in boldface.

The notation in the syntax lines describes the general command form using these rules:

- Words in capital letters must be spelled as shown, but you can use any combination of upper- or lowercase letters.
- Words italicized in the syntax line are defined in the text.
- The symbolic notation d:, filename, typ, and filespec have the general meanings described earlier in this section.
- You must include one or more space characters where a space is shown, unless otherwise specified. For example, the PIP options need not be separated by spaces.

Table 5-2 defines the special symbols and abbreviations used in syntax lines.

Table 5-2. Syntax Notation

Symbol	Meaning
DIR	Directory attribute.
n	You can substitute a number for n.
o	Indicates an option or an option list.
R/O	Read/Only. Indicates a file or disk that can only be read.
R/W	Read/Write. Indicates a file that can be read and written.
s	You can substitute a string, which consists of a group of characters, for s.
SYS	System attribute.
{ }	Items within braces are optional. You can enter a command without optional items. The optional items add effects to your command line.
[ ]	Items in square brackets are options or an option list. When you use an option specified within the brackets, you must enclose the option in brackets. If the right bracket is the last character on the command line, you can omit it.
( )	Items in parentheses indicate a range of options. If you use a range from an option list, you must enclose the range in parentheses.
...	The item preceding the ellipses can be repeated any number of times.
|	The OR bar separates alternative items in a command line. You can select any or all of the alternatives specified. Mutually exclusive options are indicated in additional syntax lines or are specifically noted in the text.
^ or CTRL	Represents the CTRL key on your keyboard. (CTRL characters appear as ^ on your screen.)
<cr>	Indicates a carriage return keystroke.

Table 5-2. (continued)

Symbol	Meaning
*	Wildcard character--any valid group of characters can take the place of the *.
?	Wildcard character--any valid character can take the place of ?.

Let's look at some examples of syntax notation. The Personal CP/M DIR (Directory) command displays the names of files cataloged in the disk directory.

The syntax of the DIR command is

```
Syntax:  DIR {d:}    {filespec}
               |          |
           optional   optional
```

The braces indicate that the command tail following the command keyword DIR is optional. DIR alone is a valid command, but you can include a file specification, a drive specification, or both. Thus the following forms of the DIR command are valid:

```
DIR
DIR d:
DIR filename.typ
DIR d:filename.typ
```

Recall that in Section 2 you learned about wildcards in filenames and filetypes. The DIR command accepts wildcards in the file specification. So command lines like the following are valid:

```
DIR B:*.C?M
```

The Personal CP/M command PIP (Peripheral Interchange Program) calls the file copy program. PIP copies information from the disk to the screen or printer. PIP combines two or more files into one longer file. PIP also renames files after copying them and copies files from disk to disk. Look at one of the formats of the PIP command line for another example of command line notation.

```
Syntax:  PIP dest-filespec=src-filespec{,filespec...}
```

In the preceding example, dest-filespec is further defined as a destination file specification or peripheral device (printer, for example) that receives data. Similarly, src-filespec is a source file specification or peripheral device (keyboard, for example) that transmits data. PIP accepts wildcards in the filename and filetype.

(See the PIP command description for other capabilities of PIP.) Many valid command lines come from this syntax. Some examples follow:

```
PIP NEWFILE.DAT=OLDFILE.DAT
PIP B:=A:THISFILE.DAT
PIP B:X.BAS=Y.BAS,Z.BAS
PIP X.BAS=A.BAS,B.BAS,C.BAS
PIP B:=A:*.BAK
PIP B:=A:*.*
```

A complete description of each Personal CP/M utility follows. The descriptions are arranged alphabetically.

DIR Command

Syntax: DIR {d:} {filename.typ}

Explanation: The DIR (Directory) command displays the names of all directory (DIR) files in the current user number denoted by the drive and file specifications.

The drive and the filename.typ specifications are optional; either or both can appear. Both filename and filetype can contain wildcard characters. If no drive specification is set, DIR assumes the currently logged drive. If you omit the filename.typ specification, DIR displays the names of all files with the DIR attribute on the currently logged or specified drive. DIR by itself is equivalent to

DIR *.*

where the drive is the currently logged drive.

The DIR command displays only files with the DIR attribute. Use the STAT command to display files with the SYS attribute. Only files under the current user number are displayed.

If no file meets the drive/file/user specifications, DIR displays the message

NO FILE

Examples: A>**DIR**

Displays all DIR files in user 0 on the default drive A.

A>**DIR** B:

Displays all DIR files in user 0 on drive B.

4B>**DIR *.NAM**

For User 4 on drive B, displays all DIR files with filetype NAM.

4B>**DIR C:*.***

For user 4 on drive C, displays all DIR files.

ED Command

Syntax: ED {d1:} filename.typ {d2:}

Explanation: ED allows you to create and edit disk files.

Drive specifications d1 and d2 are optional. When d1 appears, Personal CP/M looks for the source file on drive d1. When d2 appears, Personal CP/M places the temporary edit file (used during actual file editing) on d2. When editing is complete, the edited file appears on d2. The temporary edit file is called filename.$$$. When editing is complete, the source file, called filename.typ, is renamed filename.BAK, and the temporary edit file (filename.$$$) is named filename.typ.

The filename and filetype cannot contain wildcard characters; the filetype is optional.

ED uses a portion of main memory as a buffer for sections of text being edited. You move text into this buffer with the A (Append) command. You write text from the buffer to the temporary edit file (filename.$$$) with the W (Write) command. The E (Exit) command functions like the W command. You also rename filename.typ to filename.BAK and filename.$$$ to filename.typ and return to the Personal CP/M prompt. (A full command list for ED appears in Section 6.)

If no file called filename.typ exists when you type

A>**ED filename.typ**

ED displays the message

NEW FILE

and opens a file for you to edit.

You interact with the ED utility in command or insert mode. ED displays the prompt * on the screen when ED is in command mode, and you can enter a variety of command characters (described in Section 6) that allow you to manipulate text. In insert mode you can insert new text into the file. In insert mode line numbers nnnnn appear at the beginning of each line. These numbers are displayed for reference only and

are not contained in the buffer or in any of the disk files. You can disable line numbers with the -V command.

Examples: **A>ED X.TXT**

Creates a temporary work file called X.$$$. Allows you to edit source file in a buffer in main memory and store edited text in the temporary file. At completion of editing, source is renamed X.BAK and X.$$$ is renamed X.TXT. All operations occur on the currently logged disk.

A>ED C:Z.TXT B:

Creates a temporary work file called Z.$$$ on drive B, allowing you to edit source in a buffer in main memory. At completion of editing, ED renames Z.TXT on drive C Z.BAK and renames Z.$$$ on drive B Z.TXT.

See Section 6 for a complete description of ED.

ERA Command

Syntax: ERA {d:}filename.typ

Explanation: This command erases files from a disk.

The disk specification d: is optional. If no disk specification is given, Personal CP/M deletes the file or files from the currently logged disk. The filename.typ can contain wildcard characters. The typ is optional.

If no file or set of files matches filename.typ, the following message appears:

NO FILE

Use ERA with care, because ERA erases every file that matches filename.typ. When using wilcards in the specification, be sure you mean to delete all files denoted by the specification.

This command takes place in the currently logged user number. Directory and data space are immediately reclaimed for use by other files.

Examples: 2B>**ERA X.TXT**

Erases the file X.TXT in user area 2 of drive B.

3C>**ERA *.TXT**

Erases all files with typ TXT in user area 3 of drive C.

PIP COMMAND

Syntax: PIP
PIP 'command line'

Explanation: PIP is a transient program that copies one or more files from one disk and or user number to another. PIP can rename a file after copying it. PIP can combine two or more files into one file. PIP can also copy a character file from disk to the printer or other auxiliary logical output device. PIP can create a file on disk from input from the console or other logical input device. PIP can transfer data from a logical input device to a logical output device, thus the name Peripheral Interchange Program.

To initiate PIP, type one of the preceding forms. Both forms load PIP into the TPA and execute. When you use the first form, PIP reads command lines directly from the console, prompting you with the "*" character, until you enter an empty command (that is, until you press a single carriage return). Each successive command line causes a media conversion to take place. The form PIP 'command line' is equivalent to the form PIP, except that the single command line you type with PIP automatically executes, and PIP terminates immediately. The form of each command line is

destination = source1, source2, ... , sourcen

where destination is the file or peripheral device to receive the data and source1,... is a series of files or devices that are copied from left to right to the destination.

When the command line specifies multiple files (that is, when n is greater than 1 in the preceding form), PIP assumes the files contain ASCII characters with a CP/M end-of-file character (CTRL-Z) at the end of each file. (See the O parameter to override this assumption.) PIP internally translates lowercase ASCII alphabetics to uppercase for consistency with CP/M file and device name conventions. Finally, the total command line length must not exceed 128 characters. Use CTRL-E to force a physical carriage return for lines that exceed 128 characters.

The destination and source elements are unambiguous references to CP/M source files with or without a preceding disk drive name. You can reference any

file with a preceding drive name (A: through P:) that defines the drive on which the file is stored. When you do not include a drive, PIP assumes the currently logged disk. The destination file can also appear as one or more source files; in this case, PIP does not alter the source file until the entire concatenation is complete. If the destination file already exists, it is removed if the command line is properly formed.

The destination file is not removed if an error condition arises.

Examples: A>**PIP X=Y**

Copies to file X from file Y, where X and Y are unambiguous filenames; Y remains unchanged.

A>**PIP X.ASM=Y.ASM,Z.ASM,FIN.ASM**

Creates the file X.ASM from the concatenation of the Y, Z, and FIN files with type ASM.

A>**PIP B:A.U=B:B.V,A:C.W,D.X**

Concatenates file B.V from drive B with C.W from drive A and D.X from the logged disk; creates the file A.U on drive B.

With Abbreviations

Syntax: Destination=Source

```
PIP d:=filename.typ          (wildcards allowed)
PIP d1:=d2:filename.typ      (wildcards allowed)
PIP filename.typ=d2:         (wildcards not allowed)
PIP d1:filename.typ=d2:      (wildcards not allowed)
```

PIP allows the foregoing abbreviated commands for transferring files between disk drives.

The first form copies all files that satisfy filename.typ from the currently logged disk to the same files on drive d (d = drive A through drive P). The second form copies all files that satisfy filename.typ from drive d2 (the second drive) to the same files on drive d1 (the first drive). The third form copies all files that satisfy filename.typ on drive d2 to the same files on the currently logged

disk. The fourth form copies all files that satisfy filename.typ on drive d2 to the same files on drive d1.

The source and destination disks must be different in all these cases. An ambiguous filename is a filename that contains wildcard characters. An unambiguous filename is a filename that contains no wildcard characters. If an ambiguous filename is specified, PIP lists each unambiguous filename that satisfies that ambiguous filename as it is being copied. If a file of the same name as the destination file exists, it is removed on successful completion of the copy and replaced by the copied file.

Examples: **A>PIP B:=*.COM**

Copy all files that have the filetype COM to drive B from the current drive.

A>PIP A:=B:ZAP.*

Copy all files that have the filename ZAP to drive A from drive B.

A>PIP ZAP.ASM=B:

Equivalent to ZAP.ASM=B:ZAP.ASM

A>PIP B:ZOT.COM=A:

Equivalent to B:ZOT.COM=A:ZOT.COM

Command Line with Disk or Character Devices

Syntax: device0:filename0.typ0=
devicel:filenamel.typl, device2:filename2.typ2, ...
devicen:filenamen.typn

Explanation: PIP allows reference to the various character devices attached to the CP/M system. The character devices you can reference are listed in Table 5-3.

Table 5-3. Personal CP/M Character Devices

Device	Explanation
CON:	Console. Input/output device.
LST:	List. Output device.

Table 5-3. (Continued)

Device	Explanation
PRN:	Same as LST: with options [t8np60]. Expands tabs to eight spaces, numbers the lines, and puts 60 lines on a page before a form-feed.
EOF:	End of file. Sends a CP/M end-of-file (ASCII CTRL-Z) to the destination device. Input device.
NUL:	Null. Sends 40 "nulls" (ASCII 0's) to the destination device. Input device.
INP:	Input. Special input source that can be patched into the PIP program by your computer manufacturer. See the user's manual for your computer for more information about this device. Input device.
OUT:	Output. Special output destination that can be patched into the PIP program by your computer manufacturer. See the user's manual for your computer for more information about this device. Output device.
AUX:	Auxiliary device. Typically used for a serial I/O device such as a printer or modem. Input/Output device.

Note that the destination device must be capable of receiving data, and the source devices must be capable of generating data. For example, the LST: device cannot be read.

You can intersperse file and device names in the PIP commands. In each case, PIP reads the specific device until end-of-file (CTRL-Z for ASCII files, and end-of-data for non-ASCII disk files). PIP concatenates data from each device or file 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 (CTRL-Z) is appended to the result for ASCII files. If the destination is a disk file, PIP creates a temporary file (filetype = $$$) that changes to the actual filename only on successful completion of the copy. PIP assumes files with the extension COM to be non-ASCII.

You can abort a copy operation in two ways: If you press CTRL-Z, PIP treats this like an end-of-file from the device; that is, it treats it like a normal ("clean") termination. If you press any other key, the operation aborts and the message "ABORTED" appears on the console. If an operation aborts, or if an error occurs during processing, PIP removes pending commands that were set up while using the SUBMIT command.

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, PIP checks to ensure that the source file contains a properly formed hex file, with legal hexadecimal values and checksum records.

When PIP finds an invalid input record, 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 reread, press RETURN once, and PIP attempts another read. If PIP cannot read the tape position properly, continue the read by pressing RETURN following the error message. Enter the record manually with the ED program after the disk file is constructed. PIP allows you to enter the end-of-file from the console. As noted above, when you type CTRL-Z at the keyboard, the read operation terminates normally.

Examples: **A>PIP LST:=X.PRN**

Copies X.PRN to the list device and terminates the PIP program.

A>PIP CON:=X.ASM,Y.ASM,A.ASM

Concatenates three ASM files on the currently logged disk and copies them to the console device.

Command Line with Parameters

Syntax: source n{[parameters]}

You can also specify one or more PIP parameters. Enclose the parameters in square brackets and separate them with zero or more blanks. Each parameter affects the copy operation, and the enclosed list of parameters must immediately follow the affected file or device. An optional decimal integer value typically follows each parameter. (The S and Q parameters are exceptions.) Valid PIP parameters are listed in Table 5-4.

Table 5-4. Valid PIP Parameters

Parameter	Explanation
B	Block mode transfer: PIP buffers data until an ASCII x-off character (CTL-S) is received by 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 that can be buffered depends on the memory size of the host system. PIP issues an error message when the buffer overflows.
Dn	Delete characters that extend past column n in the transfer of data to the destination from the character source. This parameter truncates long lines that are sent to a (narrow) printer or console device.
E	Echoes all transfer operations to the console as they are being performed.
F	Filters form feeds from the file. All imbedded form feeds are removed. The P parameter can simultaneously insert new form feeds.
Gn	Get File from user number n (n in the range 0-15).

Table 5-4. (continued)

Parameter	Explanation
H	Hex data transfer. Checks all data for proper Intel hex file format. Removes nonessential characters between hex records during the copy operation. The console prompts for corrective action in case errors occur.
I	Ignores :00 records in the transfer of Intel hex format file. (The I parameter automatically sets the H parameter.)
L	Translates uppercase alphabetics to lowercase.
N	Adds line numbers to each line transferred to the destination, starting at 1 and incrementing by 1. Leading zeroes are suppressed, and a colon follows the number. If N2 is specified, leading zeroes are included, and a tab is inserted following the number. The tab is expanded if T is set.
O	Object file (non-ASCII) transfer: the CP/M end-of-file is ignored.
Pn	Includes page ejects at every n lines (with an initial page eject). If n = 1 or is excluded, page ejects occur every 60 lines. If you use the F parameter, Pn suppresses form feed before the new page ejects are inserted.
Qs^z	Quits copying from the source device or file when the string s (terminated by a CTRL-Z) is encountered.
R	Reads system files.

Table 5-4. (continued)

Parameter	Explanation
Ss^z	Starts copying from the source device when the string s (terminated by CTRL-Z) is encountered. You can use the S and Q parameters to abstract a section of a file such as a subroutine. The copy operation always includes the start and quit strings. If you select the PIP 'command form' syntax, the CCP translates strings following the S and Q parameters to uppercase. The command form PIP does not perform the automatic uppercase translation.
Tn	Expands tabs (CTRL-I characters) to every nth column during the transfer of characters to the destination from the source.
U	Translates lowercase alphabetics to uppercase during the copy operation.
V	Verifies that data has been copied correctly by rereading after the write operation.
W	Writes over R/O (Read/Only) files without console interrogation.
Z	Zeroes the parity bit for each ASCII character.

Examples:

A>PIP X.ASM=B:[v]

Copies X.ASM from drive B to the current drive and verifies that the data were properly copied.

A>PIP LST:=X.ASM[nt8u]

Copies X.ASM to the list device, numbers each line, expands tabs to every eighth column, and translates lowercase alphabetics to uppercase.

A>PIP X.LIB=Y.ASM[sSUBRI:^zqJMP L3^z]

Copies from the file Y.ASM into the file X.LIB. Starts the copy when PIP finds the string SUBRI: and quits copying when the string JMP L3 is encountered.

A>**PIP PRN:=X.ASM[p50]**

Sends X.ASM to the list device with line numbers, tabs expanded to every eighth column, and page ejects at every 50th line. The assumed parameter list for a PRN file is nt8p60; p50 overrides the default value.

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

PIP does not overwrite a file set to a permanent R/O status. If you attempt to overwrite an R/O file, PIP responds

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

If you press the character Y, the file is over written. Otherwise, PIP responds

** NOT DELETED **

PIP skips the file transfer and continues with the next operation in sequence. To avoid the prompt and response for R/O file overwrite, include the W parameter in the command line, shown in the two preceding examples. In the first example, PIP copies all nonsystem COM files from drive B to drive A, overwriting any R/O files in the process. If the operation involves several concatenated files, you must include the W parameter only with the last file in the list, as shown in the second example.

REN Command

Syntax: REN {d:}newfile.typ={d:}oldfile.typ

Explanation: The REN (Rename) command allows you to change the names of files on disk.

Oldfile.typ changes to newfile.typ. The d: is optional in the command. It can precede newfile.typ or oldfile.typ or both. If the drive specification precedes both, then d: must reference the same drive in both cases. If d: precedes only one of the filenames, REN assumes the renaming operation takes place on that drive. If no drive is specified, then the REN assumes the currently logged drive.

The filename and filetype must not contain wildcards. The filetype is optional.

If newfile.typ is already present, REN responds with the error message FILE EXISTS and makes no change. If oldfile.typ does not exist on the specified disk, the message NO FILE prints at the console.

Examples: A>**REN X=Y**

Changes the name of file X to Y.

A>**REN B:TWO.ASM=ONE.ASM**

Changes the name of file ONE.ASM to TWO.ASM on drive B.

SAVE Command

Syntax: SAVE n {d:}filename.typ

Explanation: The SAVE command allows you to save a portion of memory in a disk file. The SAVE command writes n pages (256-byte blocks) of the TPA (Transient Program Area, located in main memory) to a file on disk d, and names the file filename.typ.

The disk specification d: is optional. If no disk is specified, the currently logged disk is assumed. The filetype is also optional.

The TPA starts at location 100H (100 hexadecimal=256). If your program occupies memory locations 100H through 2FFH (256 through 767), you must specify 2 pages in order to save it (256 + 2 X 256 - 1=767).

Examples: A>**SAVE 3 X**

Copies memory locations 100H (256) through 3FFH (1023) to a file called X on the currently logged drive.

A>**SAVE 20 C:X.SAV**

Copies memory locations 100H through 15FFH (5119) to a file called X.SAV on drive C.

STAT Command

Syntax: STAT
STAT 'command line'

Explanation: The STAT command provides general statistical information about file storage and status. Initiate STAT by typing one of the preceding command forms.

Special forms of the command line allow you to examine and alter the current device assignment. The various command lines you can specify are shown and explained below.

Examples: **A>STAT**

When you type an empty command line, the STAT transient calculates the storage remaining on all active drives, and prints a message.

d: R/W,SPACE: nnnK

or

d: R/O,SPACE: nnnK

for each active drive d:, where R/W indicates the drive can be read or written, and R/O indicates the drive is Read-Only (a drive becomes R/O by setting it to Read-Only as shown below). The space remaining on the disk in drive d:, in kilobytes, is denoted by nnn.

Display Remaining Bytes on Drive

Syntax: STAT d:

Explanation: When you specify a drive STAT selects the drive before computing the storage. Thus, you can issue the command STAT B: while logged into drive A; the following message results:

BYTES REMAINING ON B: nnnK

Specify Files

Syntax: STAT filename.typ

Explanation: The filename and filetype can contain wildcard characters.

The command line can also specify a set of files for STAT to scan. The files that satisfy the filename and filetype specified are listed in alphabetical order, with storage requirements for each file under the heading

```
RECS  BYTS  EX  D:FILENAME.TYP
rrrr  bbbK  ee  d:filename.typ
```

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 extents (ee=bbb/16); d is the drive name containing the file (A...P), filename is the primary filename (up to eight characters long); and typ is the filetype, which can be up to three characters long. After listing the individual files, STAT summarizes the storage usage.

Specify Drive and Files

Syntax: STAT d: filename.typ

Explanation: This syntax gives the drive name before the filename and filetype. STAT selects the specified drive, then executes.

Set Drive Status

Syntax: STAT d:=R/O

Explanation: This form sets drive d: to Read/Only. Read/Only status remains in effect until the next warm or cold start. When a disk is set for Read/Only, the message

CP/M Error on d: Read/Only Disk

appears when you try to write to the Read-Only disk d:. CP/M waits until you depress a key before performing an automatic warm start, at which time the disk becomes R/W.

For Available Status Commands

Syntax: STAT VAL:

Explanation: This command produces an instant summary of the possible STAT commands, output as follows:

```
Temp R/O Disk:  d:=R/O
Set Indicator:  filename.typ $R/O $R/W $SYS $DIR
Disk Status:  DSK: d:DSK
User Status:  USR:
```

Display File Information

Syntax: STAT d:filename.typ $S

Explanation: d: is an optional drive name; the filename and filetype can contain wildcard characters. This form of the STAT command produces the output display format

Size	Recs	Bytes	Ext	Acc	
48	55	6k	1	R/O	A:ED.COM
55	55	12k	1	R/O	(A:PIP.COM)
65536	128	16k	2	R/W	A:X.DAT

where the $S parameter causes the Size field to be displayed. Without the $S, STAT skips the Size field and displays the remaining fields. The Size field lists the virtual file size in records; 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 number of bytes allocated to a file. The system configuration determines the minimum allocation unit at configuration time. 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. For random access files, 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, can contain unallocated holes even though they are added into the record count.

The Ext field counts the number of physical extents allocated to the file. The Ext count corresponds to the file's number of directory entries. Depending on allocation size, a single directory entry can directly address up to 128K bytes (8 logical extents). (A physical extent can address up to 256K bytes.)

The Acc field gives the R/O or R/W file indicator; the following commands can change the indicator. Similarly, the parentheses enclosing the PIP.COM filename indicate that the file's system indicator is set, so it is not listed in response to DIR commands.

Set File Indicators

Syntax:

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

Explanation: The four preceding command forms set or reset permanent file indicators. The R/O indicator places the file (or set of files) in a Read-Only status. A subsequent STAT command can change this status. The R/O status is recorded in the directory with the file, so the file remains R/O through intervening cold start operations. The R/W indicator places the file in permanent Read/Write status. The SYS indicator attaches the system indicator to the file. The DIR command removes the system indicator. The filename and filetype can contain wildcard characters, but files whose attributes are changed are listed at the console when the change occurs. The drive name denoted by d: is optional.

Attempts to erase or write into a file marked R/O result in the BDOS message

CP/M Error on d: Read/Only File

The BDOS waits for console input before performing a subsequent warm start. (A RETURN is sufficient.)

Display Drive Characteristics:

Syntax: STAT d:DSK

Explanation: This command form displays the drive characteristics of the disk named d:. STAT lists the drive characteristics in the following form:

```
    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 total record capacity (65536 for an eight-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, because this mechanism detects changed media during CP/M operation without an intervening warm start. For fixed media, the number is usually zero, because the media are 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 128K in the previous example). 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 number of physical sectors per track and the number of reserved tracks follows the listing. For

logical drives that share the same physical disk, the number of reserved tracks can be quite large because this mechanism skips lower-numbered disk areas allocated to other logical disks.

If you omit d: in the preceding form, STAT produces a drive characteristic table for all currently active drives.

Display Users

Syntax: STAT USR:

Explanation: This command produces a list of user numbers that 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 this example, the active user number is 0 (default at cold start), with three user numbers that 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 or USER 3 commands, followed by a DIR command at the CCP level.

SUBMIT Command

Syntax: SUBMIT filename.SUB parm#1 ... parm#n

Explanation: The SUBMIT command allows you to batch CP/M commands for automatic processing. The SUBMIT command must use the filename of an existing file on the currently logged disk and the filetype SUB. The SUB file contains CP/M prototype commands with possible parameter substitution. SUB substitutes the actual parameters parm#1 ... parm#n into the prototype commands, and, if no errors occur, CP/M processes the file of substituted comands sequentially.

You can create a prototype command file with the ED program, interspersing "$" parameters of the form

$1 $2 $3 ... $n

corresponding to the number of actual parameters to be included when you submit the file for execution. When the SUBMIT transient executes, the actual parameters parm#1 ... parm#n are paired with the formal parameters $1 ... $n in the prototype commands. If the numbers of formal and actual parameters do not correspond, the submit function aborts with an error message at the console. The SUBMIT function creates a file of substituted commands on the logged disk with the name

$$$.SUB

The system reboots when SUBMIT terminates. Then the CCP reads this command file as a source of input rather than the console. If you perform the SUBMIT function on any disk other than drive A, the commands are not processed until you insert the disk into drive A and reboot the system. The SUBMIT function can access a SUB file on an alternate drive when you precede the filename with a drive name. Because SUBMIT files are acted upon only when they appear on drive A, you can create a SUBMIT file on drive B to execute at a later time, when inserted in drive A.

You can abort command processing at any time by pressing a RUBOUT when the command is read and echoed. In this case the $$$.SUB file is removed and subsequent commands come from the console. Command processing also aborts when the CCP detects an error

in any command. Programs that execute under CP/M abort processing command files when error conditions occur by erasing any existing $$$.SUB file.

To introduce dollar signs into a SUBMIT file, type a $$, which reduces to a single $ within the command file. An up-arrow symbol preceding an alphabetic character x produces a single CTRL-X character within the file.

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

The utility program called XSUB extends the power of the SUBMIT facility to include line input to programs as well as to the CCP. The XSUB command is the first line of the submit file. When it executes, XSUB self-relocates directly below the CCP. XSUB processes all subsequent submit command lines so that programs that read buffered console input (BDOS function 10) receive their input directly from the submit file.

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 a cold start has intervened. Note that you must load XSUB after the optional CP/M DESPOOL™ utility, if both are to run simultaneously.

Examples: 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
```

You issue the command

A>SUBMIT ASMBL X PRN

The SUBMIT program reads the ASMBL.SUB file, substituting X for all occurrences of $1 and PRN for all of occurrences of $2. This results in a $$$.SUB file containing the commands

```
ASM X
DIR X.*
ERA *.BAK
PIP PRN:=X.PRN
ERA X.PRN
```

The CCP executes these commands in sequence.

The file SAVER.SUB contains the submit lines

```
XSUB
DDT
|$1.COM
R
GO
SAVE 1 $2.COM
```

with a subsequent SUBMIT command

A>**SUBMIT SAVER PIP Y**

that substitutes X for $1 and Y for $2 in the command stream. The XSUB program loads, followed by DDT, which is sent to the command lines PIP.COM, R, and GO, thus returning to the CCP. The CCP processes the final command SAVE 1 Y.COM.

TYPE Command

Syntax: TYPE d: filename.typ

Explanation: The drive name is optional and filename and filetype must contain no wildcard characters.

This command displays the contents of the ASCII source file filename.typ at the console device. filename.typ must be on d: if the disk is specified, or on the currently logged disk if no disk is specified.

The TYPE command expands tabs (CTRL-I characters), assuming tab positions are set at every eighth column.

Examples: A>**TYPE B: X.PRN**

Displays the file X.PRN on drive B.

USER Command

Syntax: USER n

Explanation: n is an integer value in the range 0 to 15.

This command allows you to maintain separate groups of files (that is, separate user areas) in the same directory. User numbers range from 0 to 15.

On the cold boot, you are automatically logged on as user 0. Issue the USER command to move to another user area within the same directory.

Files that are active when you are logged onto one user area remain active when you log onto another user area.

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

Examples: A>USER 2

Allows you access to all files in user area 2.

End of Section 5

LOAD Command

Syntax: LOAD d:filename.HEX

Explanation: The LOAD command reads the file d:filename.HEX, which is assumed to contain "HEX" format machine code, and produces a memory image file that can subsequently be executed. The file name d:filename.HEX is assumed to be of the form

X.HEX
and only the filename X need be specified in the command. The LOAD command creates a file named

X.COM
that marks it as containing machine executable code. The file is actually loaded into memory and executed when the user types the filename X immediately after the prompting character ">" printed by the CCP.
Generally the CCP reads the filename 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 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 are deleted at the user's option.) The operation takes 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.
The user should note that the BETA.HEX file must contain valid Intel format hexadecimal machine code records (as produced by the ASM program, for example) that begin at 100H of the TPA. 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 that operate in the TPA. Programs that occupy regions of memory other than the TPA are loaded under DDT.

Example: LOAD TEST

Section 6
ED, The Personal CP/M Context Editor

INTRODUCTION TO ED

To do almost anything with a computer you need some way to enter data, a way to give the computer the information you want it to process. The programs most commonly used for this task are called editors. They transfer your keystrokes at the keyboard to a disk file. Personal CP/M's editor is named ED. Using ED, you can easily create and alter Personal CP/M text files.

The correct command format for invoking the Personal CP/M editor is given in "Starting ED." After starting ED, you issue commands that transfer text from a disk file to memory for editing. "ED Operation" details this operation and describes the basic text transfer commands that allow you to easily enter and exit the editor.

"Basic Editing Commands" details the commands that edit a file. "Combining ED Commands" describes how to combine the basic commands to edit more efficiently. Although you can edit any file with the basic ED commands, ED provides several more commands that perform more complicated editing functions, as described in "Advanced ED Commands."

During an editing session, ED can return two types of error messages. "ED Error Messages" lists these messages and provides examples that indicate how to recover from common editing error conditions.

STARTING ED

Syntax:

ED input-filespec {d: | output-filespec}

To start ED, enter its name after the Personal CP/M prompt. The command ED must be followed by a file specification, one that contains no wildcard characters, such as:

A>ED MYFILE.TEX

The file specification, MYFILE.TEX in the preceding example, specifies a file to be edited or created. The file specification can be preceded by a drive specification, but a drive specification is unnecessary if the file to be edited is on your default drive. Optionally, the file specification can be followed by a drive specification, as shown in the following example:

```
A>ED MYFILE.TEX B:
```

In response to this command, ED opens the file to be edited, MYFILE.TEX, on drive A, but sends all the edited material to a file on drive B.

Optionally, you can send the edited material to a file with a different filename, as in the following example:

```
A>ED MYFILE.TEX YOURFILE.TEX
```

If the file with the different filename already exists, ED prints the following message and terminates.

```
Output File Exists, Erase It
```

The ED prompt, *, appears at the screen when ED is ready to accept a command, as follows:

```
A>ED MYFILE.TEX
     : *
```

If no previous version of the file exists on the current disk, ED automatically creates a new file and displays the following message:

```
NEW FILE
     : *
```

Note: Before starting an editing session, use the SET command to check the amount of free space on your disk. Make sure that the unused portion of your disk is at least as large as the file you are editing, or larger if you plan to add characters to the file. When ED finds a disk or directory full, ED has only limited recovery mechanisms. These are explained in "ED Error Messages."

ED OPERATION

With ED, you change portions of a file that pass through a memory buffer. When you start ED with one of the preceding commands, this memory buffer is empty. At your command, ED reads segments of the source file, for example MYFILE.TEX, into the memory buffer for you to edit. If the file is new, you must insert text into the file before you can edit. During the edit, ED writes the edited text onto a temporary work file, MYFILE.$$$.

When you end the edit, ED writes the memory buffer contents to the temporary file, followed by any remaining text in the source file. ED then changes the name of the source file from MYFILE.TEX to MYFILE.BAK, so you can reclaim this original material from the back-up file if necessary. ED then renames the temporary file, MYFILE.$$$, to MYFILE.TEX, the new edited file. The following figure illustrates the relationship between the source file, the temporary work file, and the new file.

Note: When you invoke ED with two filespecs, an input file and an output file, ED does not rename the input file to type BAK; therefore, the input file can be Read/Only or on a write-protected disk if the output file is written to another disk.

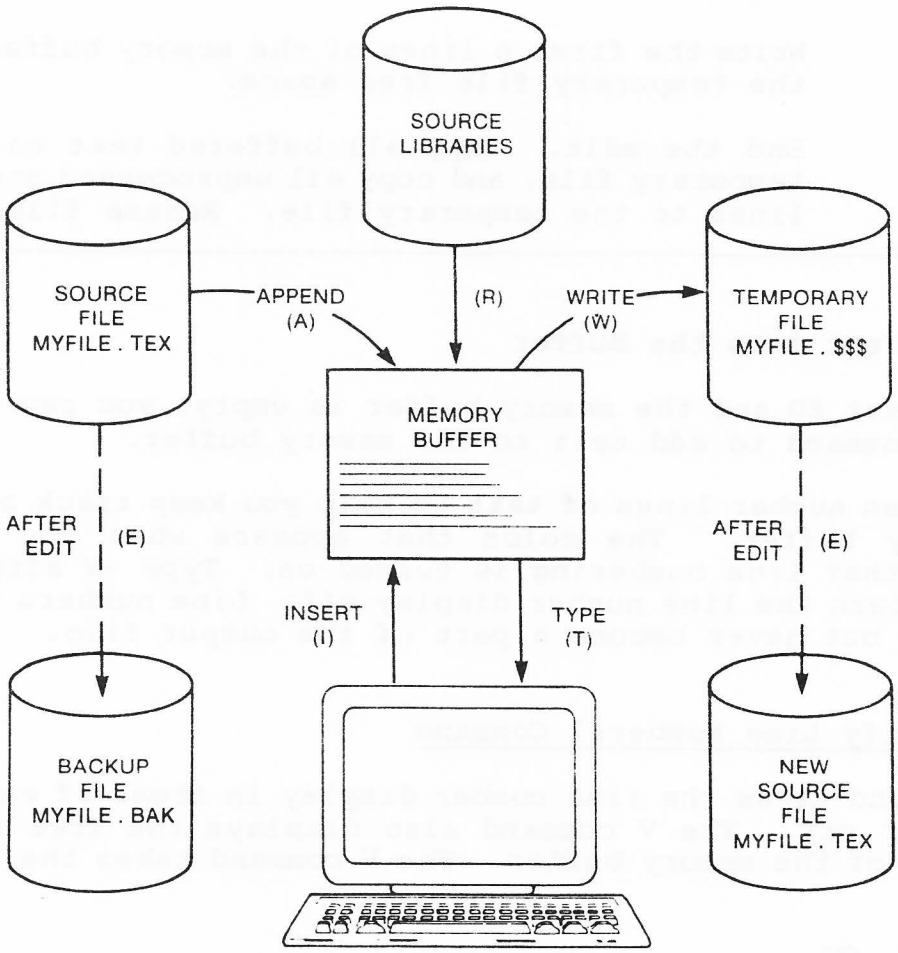


Figure 6-1. Overall ED Operation

In the preceding figure, the memory buffer is logically between the source file and the temporary work file. ED supports several commands that transfer lines of text between the source file, the memory buffer, and the temporary, and eventually final, file. The following table lists the three basic text transfer commands that allow you to easily enter the editor, write text to the temporary file, and exit the editor.

Table 6-1. Text Transfer Commands

Command	Result
nA	Append the next n unprocessed source lines from the source file to the end of the memory buffer.
nW	Write the first n lines of the memory buffer to the temporary file free space.
E	End the edit. Copy all buffered text to the temporary file, and copy all unprocessed source lines to the temporary file. Rename files.

Appending Text into the Buffer

When you start ED and the memory buffer is empty, you can use the A (append) command to add text to the memory buffer.

Note: ED can number lines of text to help you keep track of data in the memory buffer. The colon that appears when you start ED indicates that line numbering is turned on. Type -V after the ED prompt to turn the line number display off. Line numbers appear on the screen but never become a part of the output file.

The V (Verify Line Numbers) Command

The V command turns the line number display in front of each line of text on or off. The V command also displays the free bytes and total size of the memory buffer. The V command takes the following forms:

V, -V, 0V

Initially, the line number display is on. Use -V to turn it off. If the memory buffer is empty, or if the current line is at the end of the memory buffer, ED represents the line number as five blanks. The 0V command prints the memory buffer statistics in the form:

free/total

where free is the number of free bytes in the memory buffer, and total is the size of the memory buffer. For example, if you have a total of 48,253 bytes in the memory buffer and 46,652 of them are free, the 0V command displays this information as follows:

46652/48253

If the buffer is full, the first field, which indicates free space, is blank.

The A (Append) Command

The A command appends, that is, copies, lines from an existing source file into the memory buffer. The A command takes the following form:

nA

where n is the number of unprocessed source lines to append into the memory buffer. If a pound sign, #, is given in place of n, then the integer 65,535 is assumed. Because the memory buffer can contain most reasonably sized source files, it is often possible to issue the command #A at the beginning of the edit to read the entire source file into memory.

When n is 0, ED appends the unprocessed source lines into the memory buffer until the buffer is approximately half full. If you do not specify n, ED appends one line from the source file into the memory buffer.

ED Exit

You can use the W (Write) command and the E (Exit) command to save your editing changes. The W command writes lines from the memory buffer to the new file without ending the ED session. An E command saves the contents of the buffer and any unprocessed material from the source file and exits ED.

The W (Write) Command

The W command writes lines from the buffer to the new file. The W command takes the form:

nW

where n is the number of lines to be written from the beginning of the buffer to the end of the new file. If n is greater than 0, ED writes n lines from the beginning of the buffer to the end of the new file. If n is 0, ED writes lines until the buffer is half empty. The 0W command is a convenient way of making room in the memory buffer for more lines from the source file. If the buffer is full, you can use the 0W command to write half the contents of the memory buffer to the new file. You can use the #W command to write the entire contents of the buffer to the new file. Then you can use the 0A command to read in more lines from the source file.

Note: After a W command is executed, you must enter the H command to reedit the saved lines during the current editing session.

The E (Exit) Command

An E command performs a normal exit from ED. The E command takes the form:

```
E
```

followed by a carriage return.

When you enter an E command, ED first writes all data lines from the buffer and the original source file to the $$$ file. If a BAK file exists, ED deletes it, then renames the original file with the BAK filetype. Finally, ED renames the $$$ file from filename.$$$ to the original filetype and returns control to the operating system.

The operation of the E command makes it unwise to edit a back-up file. When you edit a BAK file and exit with an E command, ED erases your original file because it has a BAK filetype. To avoid this, always rename a backup file to some other filetype before editing it with ED.

Note: Any command that terminates an ED session must be the only command on the line.

BASIC EDITING COMMANDS

The text transfer commands discussed previously allow you to easily enter and exit the editor. This section discusses the basic commands that edit a file.

ED treats a file as a long chain of characters grouped together in lines. ED displays and edits characters and lines in relation to an imaginary device called the character pointer (CP). During an edit session, you must mentally picture the CP's location in the memory buffer and issue commands to move the CP and edit the file.

The following commands move the character pointer or display text in the vicinity of the CP. These ED commands consist of a numeric argument and a single command letter and must be followed by a carriage return. The numeric argument, n, determines the number of times ED executes a command; however, there are four special cases to consider in regard to the numeric argument:

- If the numeric argument is omitted, ED assumes an argument of 1.
- Use a negative number if the command is to be executed backwards through the memory buffer. The B command is an exception.

- If you enter a pound sign, #, in place of a number, ED uses the value 65,535 as the argument. A pound sign argument can be preceded by a minus sign to cause the command to execute backwards through the memory buffer, -#.
- ED accepts 0 as a numeric argument only in certain commands. In some cases, 0 causes the command to be executed approximately half the possible number of times, while in other cases it prevents the movement of the CP.

The following table alphabetically summarizes the basic editing commands and their valid arguments.

Table 6-2. Basic Editing Commands

Command	Action
B, -B	Move CP to the beginning (B) or end (-B) of the memory buffer.
nC, -nC	Move CP n characters forward (nC) or backward (-nC) through the memory buffer.
nD, -nD	Delete n characters before (-nD) or after (nD) the CP.
I	Enter insert mode.
Istring CTRL-Z	Insert a string of characters.
nK, -nK	Delete (kill) n lines before the CP (-nK) or after the CP (nK).
nL, -nL	Move the CP n lines forward (nL) or backward (- nL) through the memory buffer.
nT, -nT	Type n lines before the CP (-nT) or after the CP (nT).
n, -n	Move the CP n lines before the CP (-n) or after the CP (n) and display the destination line.

The following sections discuss ED's basic editing commands in more detail. The examples in these sections illustrate how the commands affect the position of the character pointer in the memory buffer. Later examples in "Combining ED Commands" illustrate how the commands appear at the screen. For these sections, however, the symbol ^ in command examples represents the character pointer, which you must imagine in the memory buffer.

Moving the Character Pointer

This section describes commands that move the character pointer in useful increments but do not display the destination line. Although ED is used primarily to create and edit program source files, the following sections present a simple text as an example to make ED easier to learn and understand.

The B (Beginning/Bottom) Command

The B command moves the CP to the beginning or bottom of the memory buffer. The B command takes the following forms:

B, -B

-B moves the CP to the end or bottom of the memory buffer; B moves the CP to the beginning of the buffer.

The C (Character) Command

The C command moves the CP forward or backward the specified number of characters. The C command takes the following forms:

nC, -nC

when n is the number of characters the CP is to be moved. A positive number moves the CP towards the end of the line and the bottom of the buffer. A negative number moves the CP towards the beginning of the line and the top of the buffer. You can enter an n large enough to move the CP to a different line. However, each line is separated from the next by two invisible characters: a carriage return and a line feed, represented by <cr><lf>. You must compensate for their presence. For example, if the CP is pointing to the beginning of the line, the command 30C moves the CP to the next line:

```
Emily Dickinson said,<cr><lf>
"I fin^d ecstasy in living -<cr><lf>
```

The L (Line) Command

The L command moves the CP the specified number of lines. After an L command, the CP always points to the beginning of a line. The L command takes the following forms:

```
nL, -nL
```

where n is the number of lines the CP is to be moved. A positive number moves the CP towards the end of the buffer. A negative number moves the CP back toward the beginning of the buffer. The command 2L moves the CP two lines forward through the memory buffer and positions the character pointer at the beginning of the line.

```
 "I find ecstasy in living -<cr><lf>
 the mere sense of living<cr><lf>
^is joy enough."<cr><lf>
```

The command -L moves the CP to the beginning of the previous line, even if the CP originally points to a character in the middle of the line. Use the special character 0 to move the CP to the beginning of the current line.

The n (Number) Command

The n command moves the CP and displays the destination line. The n command takes the following forms:

```
n, -n
```

where n is the number of lines the CP is to be moved. In response to this command, ED moves the CP forward or backward the number of lines specified, then prints only the destination line. For example, the command -2 moves the CP back two lines.

```
 Emily Dickinson said,<cr><lf>
^"I find ecstasy in living -<cr><lf>
 the mere sense of living<cr><lf>
 is joy enough."<cr><lf>
```

A further abbreviation of this command is to enter no number at all. In response to a carriage return without a preceding command, ED assumes an n command of 1 and moves the CP down to the next line and prints it, as follows

```
 Emily Dickinson said,<cr><lf>
 "I find ecstasy in living -<cr><lf>
^the mere sense of living<cr><lf>
```

Also, a minus sign without a number moves the CP back one line.

Displaying Memory Buffer Contents

ED does not display the contents of the memory buffer until you specify the part of the text you want to see. The T command displays text without moving the CP.

The T (Type) Command

The T command types a specified number of lines from the CP at the screen. The T command takes the forms:

```
nT, -nT
```

where n specifies the number of lines to be displayed. If you enter a negative number, ED displays n lines before the CP. A positive number displays n lines after the CP. If no number is specified, ED types from the character pointer to the end of the line. The CP remains in its original position no matter how many lines are typed. For example, if the character pointer is at the beginning of the memory buffer, and you instruct ED to type four lines (4T), four lines are displayed at the screen, but the CP stays at the beginning of line 1.

```
^Emily Dickinson said,<cr><lf>
 "I find ecstasy in living -<cr><lf>
 the mere sense of living<cr><lf>
 is joy enough."<cr><lf>
```

If the CP is between two characters in the middle of the line, a T command with no number specified types only the characters between the CP and the end of the line, but the character pointer stays in the same position, as in the following memory buffer example:

```
"I find ec^stasy in living -
```

When ED is displaying text with the T command, you can enter a CTRL-S to stop the display, then press any key when you are ready to continue scrolling. Enter a CTRL-C to abort long type-outs.

Deleting Characters

The D (Delete) Command

The D command deletes a specified number of characters and takes the forms:

```
nD, -nD
```

where n is the number of characters to be deleted. If no number is specified, ED deletes the character to the right of the CP. A

positive number deletes multiple characters to the right of the CP, towards the bottom of the file. A negative number deletes characters to the left of the CP, towards the top of the file. If the character pointer is positioned in the memory buffer as follows

```
Emily Dickinson said,<cr><lf>
"I find ecstasy in living -<cr><lf>
the mere sense of living<cr><lf>
is joy ^enough."<cr><lf>
```

the command 6D deletes the six characters after the CP, and the resulting memory buffer looks like this:

```
Emily Dickinson said,<cr><lf>
"I find ecstasy in living -<cr><lf>
the mere sense of living<cr><lf>
is joy ^."<cr><lf>
```

You can also use a D command to delete the <cr><lf> between two lines to join them together. Remember that the <cr> and <lf> are two characters.

The K (Kill) Command

The K command kills or deletes whole lines from the memory buffer and takes the forms:

```
nK, -nK
```

where n is the number of lines to be deleted. A positive number kills lines after the CP. A negative number kills lines before the CP. When no number is specified, ED kills the current line. If the character pointer is at the beginning of the second line,

```
 Emily Dickinson said,<cr><lf>
^"I find ecstasy in living -<cr><lf>
 the mere sense of living<cr><lf>
 is joy enough."<cr><lf>
```

then the command -K deletes the previous line and the memory buffer changes:

```
^"I find ecstasy in living -<cr><lf>
 the mere sense of living<cr><lf>
 is joy enough."<cr><lf>
```

If the CP is in the middle of a line, a K command kills only the characters from the CP to the end of the line and concatenates the

characters before the CP with the next line. A -K command deletes all the characters between the beginning of the previous line and the CP. A 0K command deletes the characters on the line up to the CP.

You can use the special # character to delete all the text from the CP to the beginning or end of the buffer. Be careful when using #K because you cannot reclaim lines after they are removed from the memory buffer.

Inserting Characters into the Memory Buffer

The I (Insert) Command

To insert characters into the memory buffer from the screen, use the I command. If you enter the command in uppercase, ED automatically converts the string to uppercase. The I command takes the forms:

```
I
Istring^Z
```

When you type the first command, ED enters insert mode. In this mode, all keystrokes are added directly to the memory buffer. ED enters characters in lines and does not start a new line until you press the enter key.

```
A>ED B:QUOTE.TEX

  NEW FILE
     : *i
    1:  Emily Dickinson said,
    2:  "I find ecstasy in living -
    3:  the mere sense of living
    4:  is joy enough."
    5:  ^Z
     : *
```

Note: To exit from insert mode, press CTRL-Z or ESC. When the ED prompt, *, appears on the screen, ED is not in insert mode.

In command mode, you can use Personal CP/M command line-editing control characters. In insert mode, you can use the control characters listed in Table 6-3.

Table 6-3. Personal CP/M Line-editing Controls

Command	Result
CTRL-H	Deletes the last character typed on the current line.
CTRL-U	Deletes the entire line currently being typed.
CTRL-X	Deletes the entire line currently being typed. Same as CTRL-U.
Backspace	Removes the last character.

When entering a combination of numbers and letters, you might find it inconvenient to press a caps-lock key if your terminal translates the uppercase of numbers to special characters. ED provides two ways to translate your alphabetic input to uppercase without affecting numbers. The first is to enter the insert command letter in uppercase: I. All alphabetics entered during the course of the capitalized command, either in insert mode or as a string, are translated to uppercase. If you enter the insert command letter in lowercase, all alphabetics are inserted as typed. The second method is to enter a U command before inserting text. Uppercase translation remains in effect until you enter a -U command.

The Istring^Z (Insert String) Command

The second form of the I command does not enter insert mode. It inserts the character string into the memory buffer and returns immediately to the ED prompt. You can use Personal CP/M's line-editing control characters to edit the command string.

To insert a string, first use one of the commands that position the CP. You must move the CP to the place where you want to insert a string. For example, if you want to insert a string at the beginning of the first line, use a B command to move the CP to the beginning of the buffer. With the CP positioned correctly, enter an insert string, as follows:

iIn 1870, ^Z

This inserts the phrase "In 1870," at the beginning of the first line, and returns immediately to the ED prompt. In the memory buffer, the CP appears after the inserted string, as follows:

In 1870, ^Emily Dickinson said,<cr><lf>

Replacing Characters

The S (Substitute) Command

The S command searches the memory buffer for the specified string, but when it finds it, automatically substitutes a new string for the search string. When you enter a command in uppercase, ED automatically converts the string to uppercase. The S command takes the form:

```
nSsearch string^Znew string
```

where n is the number of substitutions to make. If no number is specified, ED searches for the next occurrence of the search string in the memory buffer. For example, the command:

```
sEmily Dickinson^ZThe poet
```

searches for the first occurrence of "Emily Dickinson" and substitutes "The poet." In the memory buffer, the CP appears after the substituted phrase, as follows:

```
The poet^ said,<cr><lf>
```

If uppercase translation is enabled by a capital S command letter, ED looks for a capitalized search string and inserts a capitalized insert string. Note that if you combine this command with other commands, you must terminate the new string with a CTRL-Z.

COMBINING ED COMMANDS

You save keystrokes and editing time by combining the editing and display commands. You can type any number of ED commands on the same line. ED executes the command string only after you press the carriage return key. Use Personal CP/M's line-editing controls to manipulate ED command strings.

When you combine several commands on a line, ED executes them in the same order they are entered, from left to right on the command line. There are four restrictions to combining ED commands:

- The combined-command line must not exceed Personal CP/M's 128-character maximum.
- If the combined-command line contains a character string, the line must not exceed 100 characters.
- Commands to terminate an editing session must not appear in a combined-command line.

- Commands, such as the I, J, R, S, and X commands, that require character strings or filespecs must be either the last command on a line or must terminate with a CTRL-Z or ESC character, even if no character string or filespec is given.

The examples in the previous section show the memory buffer and the position of the character pointer. The examples in this section show how the screen looks during an editing session. Remember that the character pointer is imaginary, but you must picture its location because ED's commands display and edit text in relation to the character pointer.

Moving the Character Pointer

To move the CP to the end of a line without calculating the number of characters, combine an L command with a C command, L-2C. This command string accounts for the <cr><lf> sequence at the end of the line.

Change the C command in this command string to move the CP more characters to the left. You can use this command string if you must make a change at the end of the line and you do not want to calculate the number of characters before the change, as in the following example:

```
    1: *T
    1:  Emily Dickinson said,
    1: *L-7CT
said,
    1: *
```

Displaying Text

A T command types from the CP to the end of the line. To see the entire line, you can combine an L command and a T command. Type 0lt to move the CP from the middle to the beginning of the line and then display the entire line. In the following example, the CP is in the middle of the line. 0L moves the CP to the beginning of the line. T types from the CP to the end of the line, allowing you to see the entire line.

```
    3: *T
sense of living
    3: *0LT
    3:  the mere sense of living
    3: *
```

The command 0TT displays the entire line without moving the CP.

To verify that an ED command moves the CP correctly, combine the command with the T command to display the line. The following example combines a C command and a T command.

```
    2: *8CT
ecstasy in living -
    2: *
```

```
    4: *B#T
    1:  Emily Dickinson said,
    2:  "I find ecstasy in living -
    3:  the mere sense of living
    4:  is joy enough."
    1: *
```

Editing

To edit text and verify corrections quickly, combine the edit commands with other ED commands that move the CP and display text. Command strings like the one that follows move the CP, delete specified characters, and verify changes quickly.

```
1: *15C5D0LT
1:  Emily Dickinson,
1: *
```

Combine the edit command K with other ED commands to delete entire lines and verify the correction quickly, as follows:

```
1: *2L2KB#T
1:  Emily Dickinson said,
2:  "I find ecstasy in living -
1: *
```

The abbreviated form of the I (insert) command makes simple textual changes. To make and verify these changes, combine the I command string with the C command and the 0LT command string as follows. Remember that the insert string must be terminated by a CTRL-Z.

```
1: *20Ci to a friend^Z0LT
1:  Emily Dickinson said to a friend,
1: *
```

ADVANCED ED COMMANDS

The basic editing commands discussed previously allow you to use ED for all your editing. The following ED commands, however, enhance ED's usefulness.

Moving the CP and Displaying Text

The P (Page) Command

Although you can display any amount of text at the screen with a T command, it is sometimes more convenient to page through the buffer, viewing whole screens of data and moving the CP to the top of each new screen at the same time. To do this, use ED's P command. The P command takes the following forms:

nP, -nP

where n is the number of pages to be displayed. If you do not specify n, ED types the 23 lines following the CP and then moves the CP forward 23 lines. This leaves the CP pointing to the first character on the screen.

To display the current page without moving the CP, enter 0P. The special character 0 prevents the movement of the CP. If you specify a negative number for n, P pages backwards towards the top of the file.

The n: (Line Number) Command

When line numbers are being displayed, ED accepts a line number as a command to specify a destination for the CP. The line number command takes the following form:

n:

where n is the number of the destination line. This command places the CP at the beginning of the specified line. For example, the command 4: moves the CP to the beginning of the fourth line.

Remember that ED dynamically renumbers text lines in the buffer each time a line is added or deleted. Therefore, the number of the destination line you have in mind can change during editing.

The :n (Through Line Number) Command

The inverse of the line number command executes a command through a certain line number. You can use this command with only three ED commands: the K (kill) command, the L (line) command, and the T (type) command. The :n command takes the following form:

:ncommand

where n is the line number through which the command is to be executed. The :n part of the command does not move the CP, but the command that follows it might.

You can combine n: with :n to specify a range of lines through which a command should be executed. For example, the command 2::4T types the second, third, and fourth lines:

```
1: *2::4T
2:  "I find ecstasy in living -
3:  the mere sense of living
4:  is joy enough."
2: *
```

Finding and Replacing Character Strings

ED supports a find command, F, that searches through the memory buffer and places the CP after the word or phrase you want. The N command allows ED to search through the entire source file instead of just the buffer. The J command searches for and then juxtaposes character strings.

The F (Find) Command

The F command performs the simplest find function; it takes the form:

nFstring

where n is the occurrence of the string to be found. Any number you enter must be positive because ED can only search from the CP to the bottom of the buffer. If you enter no number, ED finds the next occurrence of the string in the file. In the following example, the second occurrence of the word living is found.

```
1: *2fliving
3: *
```

The character pointer moves to the beginning of the third line where the second occurrence of the word "living" is located. To display the line, combine the find command with a type command. Note that if you follow an F command with another ED command on the same line, you must terminate the string with a CTRL-Z, as follows

```
1: *2fliving^Z0lt
3: *the mere sense of living
```

It makes a difference whether you enter the F command in upper or lowercase. If you enter F, ED internally translates the argument string to uppercase. If you specify f, ED looks for an exact match. For example, Fcp/m searches for Personal CP/M, but fcp/m searches for cp/m, and cannot find Personal CP/M.

If ED does not find a match for the string in the memory buffer, it issues the message,

BREAK "#" AT

where the symbol # indicates that the search failed during the execution of an F command.

The N Command

The N command extends the search function beyond the memory buffer to include the source file. If the search is successful, it leaves the CP pointing to the first character after the search string. The N command takes the form:

nNstring

where n is the occurrence of the string to be found. If no number is entered, ED looks for the next occurrence of the string in the file. The case of the N command has the same effect on an N command as it does on an F command. Note that if you follow an N command with another ED command, you must terminate the string with a CTRL-Z.

When an N command is executed, ED searches the memory buffer for the specified string, but if ED does not find the string, it does not issue an error message. Instead, ED automatically writes the searched data from the buffer into the new file. Then ED performs a 0A command to fill the buffer with unsearched data from the source file. ED continues to search the buffer, write out data, and append new data until it either finds the string or reaches the end of the source file. If ED reaches the end of the source file, ED issues the following message:

BREAK "#" AT

Because ED writes the searched data to the new file before looking for more data in the source file, ED usually writes the contents of the buffer to the new file before finding the end of the source file and issuing the error message.

Note: You must use the H command to continue an edit session after the source file is exhausted and the memory buffer is emptied.

The J (Juxtapose) Command

The J command inserts a string after the search string, then deletes any characters between the end of the inserted string to the beginning of the a third delete-to string. This juxtaposes the string between the search and delete-to strings with the insert string. The J command takes the form:

```
nJsearch string^Zinsert string^Zdelete-to string
```

where n is the occurrence of the search string. If no number is specified, ED searches for the next occurrence of the search string in the memory buffer. In the following example, ED searches for the word "Dickinson", inserts the phrase "told a friend" after it, and then deletes everything up to the comma.

```
1: *#T
1:  Emily Dickinson said,
2:  "I find ecstasy in living -
3:  the mere of living
4:  is joy enough."
1: *jDickinson^Z told a friend^Z,
1: *0lt
1:  Emily Dickinson told a friend,
1: *
```

If you combine this command with other commands, you must terminate the delete-to string with a CTRL-Z or ESC, as in the following example. If an uppercase J command letter is specified, ED looks for uppercase search and delete-to strings and inserts an uppercase insert string.

The J command is especially useful when revising comments in assembly language source code, as follows

```
236:  SORT    LXI    H, SW    ;ADDRESS TOGGLE SWITCH
236: *j;^ZADDRESS SWITCH TOGGLE^Z^L^Z0LT
236:  SORT    LXI    H, SW    ;ADDRESS SWITCH TOGGLE
236: *
```

In this example, ED searches for the first semicolon and inserts ADDRESS SWITCH TOGGLE after the mark and then deletes to the <cr><lf> sequence, represented by CTRL-L. In any search string, you can use CTRL-L to represent a <cr><lf> when the phrase that you want extends across a line break. You can also use a CTRL-I in a search string to represent a tab.

Note: If long strings make your command longer than your screen line length, enter a CTRL-E to cause a physical carriage return at the screen. A CTRL-E returns the cursor to the left edge of the screen, but does not send the command line to ED. Remember that no

ED command line containing strings can exceed 100 characters. When you finish your command, press the carriage return key to send the command to ED.

The M (Macro) Command

An ED macro command, M, can increase the usefulness of a string of commands. The M command allows you to group ED commands together for repeated execution. The M command takes the following form:

```
nMcommand string
```

where n is the number of times the command string is to be executed. A negative number is not a valid argument for an M command. If no number is specified, the special character # is assumed, and ED executes the command string until it reaches the end of data in the buffer or the end of the source file, depending on the commands specified in the string. In the following example, ED executes the four commands repetitively until it reaches the end of the memory buffer:

```
        1: *mfliving^Z-6diLiving^Z0lt
        2:  "I find ecstasy in Living -
        3:  the mere sense of Living

    BREAK "#" AT ^Z
3: *
```

The terminator for an M command is a carriage return; therefore, an M command must be the last command on the line. Also, all character strings that appear in a macro must be terminated by CTRL-Z or ESC. If a character string ends the combined-command string, it must be terminated by CTRL-Z, then followed by a <cr> to end the M command.

The execution of a macro command always ends in a BREAK "#" message, even when you have limited the number of times the macro is to be performed, and ED does not reach the end of the buffer or source file. Usually the command letter displayed in the message is one of the commands from the string and not M.

To abort a macro command, press a CTRL-C at the keyboard.

The Z (Sleep) Command

Use the Z command to make the editor pause between operations. The pauses give you a chance to review what you have done. The Z command takes the form:

```
nZ
```

where n is the number of seconds to wait before proceeding to the next instruction.

Usually, the Z command has no real effect unless you use it with a macro command. The following example shows you how you can use the Z command to cause a brief pause each time ED finds the word TEXT in a file.

```
A>*mfliving^Z0ttl0z
```

Moving Text Blocks

To move a group of lines from one area of your data to another, use an X command to write the text block into a temporary LIB file, then a K command to remove these lines from their original location, and finally an R command to read the block into its new location.

The X (Transfer) Command

The X command takes the forms:

```
nX
nXfilespec^Z
```

where n is the number of lines from the CP towards the bottom of the buffer that are to be transferred to a file. Therefore, n must always be a positive number. The nX command with no file specified creates a temporary file named X$$$$$$$.LIB. This file is erased when you terminate the edit session. The nX command with a file specified creates a file of the specified name. If no filetype is specified, LIB is assumed. This file is saved when you terminate the edit session. If the X command is not the last command on the line, the command must be terminated by a CTRL-Z or ESC. In the following example, just one line is transferred to the temporary file:

```
1: *X
1: *t
1: *Emily Dickinson said,
1: *kt
1: *"I find ecstasy in living -
1: *
```

If no library file is specified, ED looks for a file named X$$$$$$$.LIB. If the file does not exist, ED creates it. If a previous X command already created the library file, ED appends the specified lines to the end of the existing file.

Use the special character 0 as the n argument in an X command to delete any file from within ED.

The R (Read) Command

The X command transfers the next n lines from the current line to a library file. The R command can retrieve the transferred lines. The R command takes the forms:

```
R
Rfilepsec
```

If no filename is specified, X$$$$$$$ is assumed. If no filetype is specified, LIB is assumed. R inserts the library file in front of the CP; therefore, after the file is added to the memory buffer, the CP points to the same character it did before the read, although the character is on a new line number. If you combine an R command with other commands, you must separate the filename from subsequent command letters with a CTRL-Z as in the following example where ED types the entire file to verify the read.

```
1: *4l
 : *R^ZB#T
1:  "I find ecstasy in living -
2:  the mere sense of living
3:  is joy enough."
4:  Emily Dickinson said,
1: *
```

Saving or Abandoning Changes: ED Exit

You can save or abandon editing changes with the following three commands.

The H (Head of File) Command

An H command saves the contents of the memory buffer without ending the ED session, but it returns to the head of the file. It saves the current changes and lets you reedit the file without exiting ED. The H command takes the following form:

```
H
```

followed by a carriage return.

To execute an H command, ED first finalizes the new file, transferring all lines remaining in the buffer and the source file to the new file. Then ED closes the new file, erases any BAK file that has the same file specification as the original source file, and renames the original source file filename.BAK. ED then renames the new file, which has had the filetype $$$, with the original file specification. Finally, ED opens the newly renamed file as the new source file for a new edit, and opens a new $$$ file. When ED returns the * prompt, the CP is at the beginning of an empty memory buffer.

If you want to send the edited material to a file other than the original file, use the following command form:

ED filespec differentfilespec

If you then restart the edit with the H command, ED renames the file differentfilename.$$$ to differentfilename.BAK and creates a new file of differentfilespec when you finish editing.

The O (Original) Command

An O command abandons changes made since the beginning of the edit and allows you to return to the original source file and begin reediting without ending the ED session. The O command takes the form:

O

followed by a carriage return. When you enter an O command, ED confirms that you want to abandon your changes by asking

O (Y/N)?

You must respond with either a Y or an N; if you press any other key, ED repeats the question. When you enter Y, ED erases the temporary file and the contents of the memory buffer. When the * prompt returns, the character pointer is pointing to the beginning of an empty memory buffer, just as it is when you start ED.

The Q (Quit) Command

A Q command abandons changes made since the beginning of the ED session and exits ED. The Q command takes the form:

Q

followed by a carriage return.

When you enter a Q command, ED verifies that you want to abandon the changes by asking

Q (Y/N)?

You must respond with either a Y or an N; if you press any other key, ED repeats the question. When you enter Y, ED erases the temporary file, closes the source file, and returns control to Personal CP/M.

Note: You can enter a CTRL-Break or a CTRL-C to return control immediately to Personal CP/M. This does not give ED a chance to close the source or new files, but it prevents ED from deleting any temporary files.

ED ERROR MESSAGES

ED returns one of two types of error messages: an ED error message if ED cannot execute an edit command, or a Personal CP/M error message if ED cannot read or write to the specified file. An ED error message takes the form:

BREAK "x" AT c

where x is one of the symbols defined in the following table and c is the command letter where the error occurred.

Table 6-4. ED Error Symbols

Symbol	Meaning
#	Search failure. ED cannot find the string specified in a F, S, or N command.
?c	Unrecognized command letter c. ED does not recognize the indicated command letter, or an E, H, O, or Q command is not alone on its command line.
0	No .LIB file. ED did not find the LIB file specified in an R command.
>	Buffer full. ED cannot put anymore characters in the memory buffer, or string specified in an F, N, or S command is too long.
E	Command aborted. A keystroke at the keyboard aborted command execution.
F	File error. Followed by either disk FULL or DIRECTORY FULL.

The following examples show how to recover from common editing error conditions. For example,

BREAK ">" AT A

means that ED filled the memory buffer before completing the execution of an A command. When this occurs, the character pointer is at the end of the buffer and no editing is possible. Use the OW command to write out half the buffer or use an O or H command and reedit the file.

BREAK "#" AT F

means that ED reached the end of the memory buffer without matching the string in an F command. At this point, the character pointer is at the end of the buffer. Move the CP with a B or n: line number command to resume editing.

```
BREAK "F" AT F
DISK FULL
```

Use the 0X command to erase an unnecessary file on the disk or a B#Xd:buffer.sav command to write the contents of the memory buffer onto another disk.

```
BREAK "F" AT n
DIRECTORY FULL
```

Use the same commands described in the previous message to recover from this file error.

The following table defines the disk file error messages ED returns when it cannot read or write a file.

Table 6-5. ED Disk File Error Messages

Message	Meaning
Bdos Err on d: Function NNN	R/O File: FILENAME.TYP Disk d: has Read/Only attribute. This occurs if a different disk has been inserted in the drive since the last cold or warm boot.
** FILE IS READ ONLY **	The file specified in the command to invoke ED has the R/O attribute. ED can read the file so that the user can examine it, but ED cannot change a Read/Only file.

End of Section 6

Section 7
CP/M Assembler

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 form ASM filename parms 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 that specifies the origin of the source file, the destination of the hex file, and the destination of the print file. The form is

```
filename.p1p2p3
```

where p1, p2, and p3 are single letters

P1: A,B, ..., P designates the disk name that contains the source file

p2: A,B, ..., P designates the disk name that will receive the hex file

Z skips the generation of the hex file

p3: A,B, ..., P designates the disk name that will receive the print file

X places the listing at the console

Z skips generation of the print file

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 also to be created 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 to be 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 the Intel 8080 assembler (macros are not implemented in ASM; see the optional MAC macro assembler). There are certain extensions in the CP/M assembler that make it somewhat easier to use. These extensions are described below.

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 assembly 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 treated as an end-of-line by the assembler (thus, multiple assembly language statements can be written on the same physical line if separated by exclamation mark symbols).

The line# is an optional decimal integer value representing the source program line number, and 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 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 are treated as if they were upper case. The following are all valid instances of labels

```
x       xy      long$name
x:      yxl:    longer$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. The CP/M assembler also treats statements that begin with an * in column one as comment statements, which are listed and ignored in the assembly process.

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

FORMING THE OPERAND

To describe the operation codes and pseudo-operations completely, it is necessary first to 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, the most significant 8 bits of the expression must be zero. The restriction on the expression significance is given with the individual instructions.

Lables

As discussed above, a label is an identifier that occurs on a particular statement. In general, the label is given a value determined by the type of statement that it precedes. If the label occurs on a statement that generates machine code or reserves memory space (e.g., a MOV instruction or a DS pseudo-operation), the label is given the value of the program address that it labels. If the label precedes an EQU or SET, the label is given the value that 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.

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 that 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). The user should note that the leading digit of a hexadecimal constant must be a decimal digit 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 that can be contained within a 16-bit counter, otherwise it is truncated on the right by the assembler. Similar to identifiers, imbedded $ signs 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	3377O	33$77$22Q
3377o	0fe3h	1234d	0ffffh

Reserved Words

There are several reserved character sequences that have predefined meanings in the

operand field of a statement. The names of 8080 registers are given below. When they are encountered, they produce the values 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 that 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.

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. The user should note, however, that only graphic (printing) ASCII characters are allowed within strings.

Valid strings are	which represent
'A' 'AB' 'ab' 'c'	A AB ab c
'' 'a''' '''' ''''	a' ' '
'Walla Walla Wash.'	Walla Walla Wash.
'She said "Hello" to me.'	She said "Hello" to me
'I said "Hello" to her.'	I said "Hello" to her

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

a + b	unsigned arithmetic sum of a and b
a - b	unsigned arithmetic difference between a and b
+ b	unary plus (produces b)
- b	unary minus (identical to 0 - b)
a * b	unsigned magnitude multiplication of a and b
a / b	unsigned magnitude division of a by b
a MOD b	remainder after a / b
NOT b	logical inverse of b (all 0s become 1s, 1s become 0s), where b is considered a 16-bit value
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 that results from shifting a to the left by an amount b, with zero fill
a SHR b	the value that 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   LI /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., all1s). The resulting expression must fit the operation code in which it is used. For example, if the expression is used in an ADI (add immediate) instruction, the high order 8 bits of the expression must be zero. As a result, the operation ADI -1 produces an error message (-1 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.

Precedence of Operators

As a convenience to the programmer, ASM assumes that operators have a relative precedence of application that allows the programmer to write expressions without nested levels of parentheses. The resulting expression has assumed parentheses that are defined by the relative precedence. The order of application of operators in unparenthesized 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 parenthesized expressions shown to the right

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

Balanced parenthesized subexpressions can always be used to override the assumed parentheses; 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))
```

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

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 that appears in the operation field of the line. The acceptable pseudo-operations are

ORG	set the program or data origin
END	end program, optional start address
EQU	numeric "equate"
SET	numeric "set"
IF	begin conditional assembly
ENDIF	end of conditional assembly
DB	define data bytes

DW	define data words
DS	define data storage area

The individual directives are detailed below.

The ORG Directive

The ORG statement takes the form

```
label  ORG  expression
```

where "label" is an optional program identifier and expression is a 16-bit expression, consisting of operands that are defined before 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. The user should 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, 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).

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 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 100H (beginning of the transient program area).

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 that describes the value in a more human-oriented manner. Further, this name is used throughout the program to "parameterize" certain functions. Suppose data received from a teletype appear on a particular input port and data are 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 that access the teletype could appear as

```
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.

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 that 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.

The IF and ENDIF Directives

The IF and ENDIF statements define a range of assembly language statements that 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 nonzero value, then statement#l through statement#n are assembled; if the expression evaluates to zero, the statements are listed but not assembled. Conditional assembly is often used to write a single "generic" program that 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 that communicates with either a teletype or a CRT console (but not both) by selecting a particular value for TTY before the assembly begins.

```
TRUE      EQU      0FFFFH      ;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

The DB directive allows the programmer to define initialized 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 that evaluate to 8-bit values (the high order bit 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. The user should note that ASCII characters are always placed in memory with the parity bit reset (0). Also, 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,5,377Q,1+2+3+4
sign-on:  DB     'please type your name',cr,lf,0
          DB     'AB' SHR 8, 'C', 'DE' AND 7FH
```

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 that evaluate to 16-bit results. The user should note that ASCII strings of one or two characters are allowed, but strings longer than two characters are 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 llb.
```

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
```

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 Intel's "8080 Assembly Language Programming Manual." Labels are optional on each input line. 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 the following tables,

e3	represents 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.
e8	represents an 8-bit value in the range 0-255.
e16	represents a 16-bit value in the range 0-65535.

These expressions can 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 that follow, each operation code is listed in its most general form, along with a specific example, with a short explanation and special restrictions.

Jumps, Calls, and Returns

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

JMP	e16	JMP L1	Jump unconditionally to label
JNZ	e16	JNZ L2	Jump on nonzero condition to label
JZ	e16	JZ 100H	Jump on zero condition to label
JNC	e16	JNC L1+4	Jump no carry to label
JC	e16	JC L3	Jump on carry to label
JPO	e16	JPO $+8	Jump on parity odd to label
JPE	e16	JPE L4	Jump on even parity to label
JP	e16	JP GAMMA	Jump on positive result to label
JM	e16	JM al	Jump on minus to label.
CALL	e16	CALL S1	Call subroutine unconditionally
CNZ	e16	CNZ S2	Call subroutine on nonzero condition

CZ	e16	CZ 100H	Call subroutine on zero condition
CNC	e16	CNC S1+4	Call subroutine if no carry set
CC	e16	CC S3	Call subroutine if carry set
CPO	e16	CPO $+8	Call subroutine if parity odd
CPE	e16	CPE S4	Call subroutine if parity even
CP	e16	CP GAMMA	Call subroutine if positive result
CM	e16	CM b1$c2	Call subroutine if minus flag.
RST	e3	RST 0	Programmed restart, equivalent to CALL 8*e3, except one byte call.
RET			Return from subroutine
RNZ			Return if nonzero flag set
RZ			Return if zero flag set
RNC			Return if no carry
RC			Return if carry flag set
RPO			Return if parity is odd
RPE			Return if parity is even
RP			Return if positive result
RM			Return if minus flag is set.

Immediate Operand Instructions

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

MVI e3,e8	MVI B,255	Move immediate data to register A, B, C, D, E, H, L, or M (memory)
ADI e8	ADI 1	Add immediate operand to A without carry
ACI e8	ACI 0FFH	Add immediate operand to A with carry
SUI e8	SUI L + 3	Subtract from A without borrow (carry)
SBI e8	SBI L AND 11B	Subtract from A with borrow (carry)
ANI e8	ANI $ AND 7FH	Logical "and" A with immediate data
XRI e8	XRI 1111$0000B	"Exclusive or" A with immediate data
ORI e8	ORI L AND 1+1	Logical "or" A with immediate data

CPI e8	CPI 'a'	Compare A with immediate data (same as SUI except register A not changed).
LXI e3,e16	LXI B,100H	Load extended immediate to register pair (e3 must be equivalent to B,D,H, or SP).

Increment and Decrement Instructions

The 8080 provides instructions for incrementing or decrementing single and double precision registers. The instructions are

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

Data Movement Instructions

Instructions that move data from memory to the CPU and from CPU to memory are given below.

MOV e3,e3	MOV A,B	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
LDAX e3	LDAX B	Load register A from computed address (e3 must produce either B or D)
STAX e3	STAX D	Store register A to computed address (e3 must produce either B or D)
LHLD e16	LHLD L1	Load HL direct from location e16 (double precision load to H and L)
SHLD e16	SHLD L5+x	Store HL direct to location e16 (double precision store from H and L to memory)
LDA e16	LDA Gamma	Load register A from address e16

STA e16	STA X3-5	Store register A into memory at e16
POP e3	POP PSW	Load register pair from stack, set SP (e3 must produce one of B, D, H, or PSW)
PUSH e3	PUSH B	Store register pair into stack, set SP (e3 must produce one of B, D, H, or PSW)
IN e8	IN 0	Load register A with data from port e8
OUT e8	OUT 255	Send data from register A to port e8
XTHL		Exchange data from top of stack with HL
PCHL		Fill program counter with data from HL
SPHL		Fill stack pointer with data from HL
XCHG		Exchange DE pair with HL pair

Arithmetic Logic Unit Operations

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

ADD e3	ADD B	Add register given by e3 to accumulator without carry (e3 must produce one of A, B, C, D, E, H, or L)
ADC e3	ADC L	Add register to A with carry, e3 as above
SUB e3	SUB H	Subtract reg e3 from A without carry, e3 is defined as above
SBB e3	SBB 2	Subtract register e3 from A with carry, e3 defined as above
ANA e3	ANA 1+1	Logical "and" reg with A, e3 as above
XRA e3	XRA A	"Exclusive or" with A, e3 as above
ORA e3	ORA B	Logical "or" with A, e3 defined as above
CMP e3	CMP H	Compare register with A, e3 as above
DAA		Decimal adjust register A based upon last arithmetic logic unit operation
CMA		Complement the bits in register A

STC		Set the carry flag to 1
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).

Control Instructions

The four remaining instructions categorized as control instructions are

HLT	Halt the 8080 processor
DI	Disable the interrupt system
EI	Enable the interrupt system
NOP	No operation.

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 that 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 be computed and should be simplified.
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.
S	Syntax error: statement is not properly formed.
V	Value error: operand encountered in expression is improperly formed.

Several error messages are printed that 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 that 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.

A SAMPLE SESSION

The following session shows interaction with the assembler and debugger in the development of a simple assembly language program. The ↓ arrow represents a carriage return keystroke.

```
A>ASM SORT↓        Assemble SORT.ASM

CP/M ASSEMBLER - VER 1.0

015C       Next free address
003H USE FACTOR      Percent of table used 00 to ff (hexadecimal)
END OF ASSEMBLY

A>DIR SORT.*↓

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.PRN↓
                    Source line
                ;   SORT PROGRAM IN CP/M ASSEMBLY LANGUAGE
                ;   START AT THE BEGINNING OF THE TRANSIENT
                    PROGRAM AREA

Machine code location
 0100                ORG          100H
```

```
Generated machine code
0100 214601 SORT:   LXI  H,SW  ;ADDRESS SWITCH TOGGLE
0103 3601           MVI  M,1   ;SET TO 1 FOR FIRST ITERATION
0105 214701         LXI  H,I   ;ADDRESS INDEX
0108 3600           MVI  M,0   ;I = 0
             ;
             ;      COMPARE I WITH ARRAY SIZE
010A 7E     COMPL:  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)
             ;
             ;      END OF ONE PASS THROUGH DATA
0110 214601         LXI  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 REB
             ;
             ;      CONTINUE THIS PASS
Truncated    ;      ADDRESSING I, SO LOAD AV(I) INTO REGISTERS
0119
  5F16002148CONT:   MOV E, A! MVI D, 0! LXI H, AV! 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
             ;
             ;      MOV H AND L TO ADDRESS AV(I+1)
0125 23             INX H
             ;
             ;      COMPARE VALUE WITH REGS CONTAINING AV (I)
0126 965778239E     SUB M! MOV D, A! MOV A, B! INX H! SBB M   ;SUBTRACT
             ;
             ;      BORROW SET IF AV(I+1) > AV(I)
012B DA3F01         JC   INCI  ;SKIP IF IN PROPER ORDER
             ;
             ;      CHECK FOR EQUAL VALUES
012E B2CA3F01       ORA D! JZ INCI ;SKIP IF AV(I) = AV(I+1)
0132 56702B5E       MOV D, M! MOV M, B! DCX H! MOV E, M
0136 712B722B73     MOV M, C! DCX H! MOV M, D! DCX H! MOV M, E
             ;
             ;      INCREMENT SWITCH COUNT
013B 21460134       LXI H,SW!  INR M
             ;
             ;      INCREMENT I
013F 21470134C3INCI: LXI H,I!  INR M! JMP COMP
             ;
             ;      DATA DEFINITION SECTION
0146 00     SW:     DB  0      ;RESERVE SPACE FOR SWITCH COUNT
0147        I:      DS  1      ;SPACE FOR INDEX
0148 050064001EAV:  DW  5, 100, 30, 50, 20, 7, 1000, 300, 100, -32767
000A =      N       EQU ($-AV)/2      ;COMPUTE N INSTEAD OF PRE
015C                END
A>TYPE SORT.HEX     Equate value

:100100002146013601214701360 07EFE09D2190140
:100110002146017EB7C20001FF5F16002148011988
:10012000194E79234623965778239EDA3F01B2CAA7
```

Machine code in HEX format

```
:100130003F0156702B5E712B722B732146013421C7
:07014000470134C30A01006E
:10014800050064001E00320014000700E8032C01BB
:0401580064000180BE
:0000000000
A>DDT SORT.HEX↯      Start debug run

16K DDT VER 1.0
NEXT PC
015C 0000      Default address (no address on END statement)
-XP↯

P=0000 100↯      Change PC to 100

-UFFFF↯      Untrace for 65535 steps
                                                    Abort with rubout
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 LXI H,0146*0100
-T10↯      Trace 10₁₆ steps

C0Z0M0E0I0 A=01 B=0000 D=0000 H=0146 S=0100 P=0100 LXI H, 0146
C0Z0M0E0I0 A=01 B=0000 D=0000 H=0146 S=0100 P=0103 MVI M, 01
C0Z0M0E0I0 A=01 B=0000 D=0000 H=0146 S=0100 P=0105 LXI H, 0147
C0Z0M0E0I0 A=01 B=0000 D=0000 H=0147 S=0100 P=0108 MVI M, 00
C0Z0M0E0I0 A=01 B=0000 D=0000 H=0147 S=0100 P=010A MOV A, M
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=010B CPI 09
C1Z0M1E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=010D JNC 0119
C1Z0M1E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=0110 LXI H, 0146
C1Z0M1E0I0 A=00 B=0000 D=0000 H=0146 S=0100 P=0113 MOV A, M
C1Z0M1E0I0 A=01 B=0000 D=0000 H=0146 S=0100 P=0114 ORA A
C0Z0M0E0I0 A=01 B=0000 D=0000 H=0146 S=0100 P=0115 JNZ 0100
C0Z0M0E0I0 A=01 B=0000 D=0000 H=0146 S=0100 P=0100 LXI H, 0146
C0Z0M0E0I0 A=01 B=0000 D=0000 H=0146 S=0100 P=0103 MVI M, 01
C0Z0M0E0I0 A=01 B=0000 D=0000 H=0146 S=0100 P=0105 LXI H, 0147
C0Z0M0E0I0 A=01 B=0000 D=0000 H=0147 S=0100 P=0108 MVI M, 00
C0Z0M0E0I0 A=01 B=0000 D=0000 H=0147 S=0100 P=010A MOV A, M*010B
-A10D                                                Stopped at 10BH

010D JC 119↯      Change to a jump on carry
0110↯

-XP↯

P=010B 100↯      Reset program counter back to beginning of program

-T10↯      Trace execution for 10H steps
                                                    Altered instruction
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=0100 LXI H,0146
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0146 S=0100 P=0103 MVI M,01
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0146 S=0100 P=0105 LXI H,0147
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=0108 MVI M,00
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=010A MOV A,M
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=010B CPI 09
C1Z0M1E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=010D JC 0119
C1Z0M1E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=0119 MOV E,A
C1Z0M1E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=011A MVI D,00
```

Machine code in HEX format

```
C1Z0M1E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=011C LXI H,0148
C1Z0M1E0I0 A=00 B=0000 D=0000 H=0148 S=0100 P=011F DAD D
C0Z0M1E0I0 A=00 B=0000 D=0000 H=0148 S=0100 P=0120 DAD D
C0Z0M1E0I0 A=00 B=0000 D=0000 H=0148 S=0100 P=0121 MOV C,M
C0Z0M1E0I0 A=00 B=0005 D=0000 H=0148 S=0100 P=0122 MOV A,C
C0Z0M1E0I0 A=05 B=0005 D=0000 H=0148 S=0100 P=1023 INX H
C0Z0M1E0I0 A=05 B=0005 D=0000 H=0149 S=0100 P=0124 MOV B,M*0125
-L100↯                                   Automatic breakpoint

 0100    LXI  H,0146 ⎫
 0103    MVI  M,01   ⎪
 0105    LXI  H,0147 ⎪
 0108    MVI  M,00   ⎪
 010A    MOV  A,M    ⎪ List some code
 010B    CPI  09     ⎪ from 100H
 010D    JC   0119   ⎪
 0110    LXI  H,0146 ⎪
 0113    MOV  A,M    ⎪
 0114    ORA  A      ⎪
 0115    JNZ 0100    ⎭
 -L↯

 0118    RST 07      ⎫
 0119    MOV  E,A    ⎪ List more
 011A    MVI  D,00   ⎪
 011C    LXI  H,0148 ⎭
-Abort list with rubout
-G,11B↯    Start program from current PC (0125H) and run in real time to 11BH


*0127      Stopped with an external interrupt 7 from front panel (program was
                                                          looping indefinitely)
-T4↯  Look at looping program in trace mode
                                   ↓
C0Z0M0E0I0 A=38 B=0064 D=0006 H=0156 S=0100 P=0127 MOV D,A
C0Z0M0E0I0 A=38 B=0064 D=3806 H=0156 S=0100 P=0128 MOV A,B
C0Z0M0E0I0 A=00 B=0064 D=3806 H=0156 S=0100 P=0129 INX H
C0Z0M0E0I0 A=00 B=0064 D=3806 H=0157 S=0100 P=012A SBB M*012B
-D148
                                   Data are sorted, but program does not stop.
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.............
-G0 ↯  Return to CP/M

A>DDT SORT. HEX↯    Reload the memory image

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

P=0000 100↯    Set PC to beginning of program
```

```
-L10D↵      List bad OPCODE

 010D    JNC 0119
 0110    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 00

0103  MVI M,0↵

0105↵

-^C     Return to CP/M with ctl-C (G0 works as well)

SAVE 1 SORT.COM↵     Save 1 page (256 bytes, from 100H to 1ffH) on disk in case
                     there is need 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
-G↵     Run the program from PC=100H

*0118       Programmed stop (RST 7) encountered
-D148
                                  Data properly sorted
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................
0170  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00................

-G0 ↵ Return to CP/M
```

```
A>ED SORT.ASM↵      Make changes to original program

*N,0^Z0TT↵          Find next ",0"
      MVI           M, 0        ;I = 0

*-↵   Up one line in text
      LXI           H, I        ;ADDRESS INDEX

*-↵   Up another line
      MVI           M, 1        ;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

*NJNC^Z0T↵
      JNC*T↵
      CONT          ;CONTINUE IF I <= (N-2)

*-2DIC^Z0LT↵
      JC            CONT        ;CONTINUE IF I <= (N-2)

*E↵                 Source from disk A
                    HEX to disk A
A>ASM SORT.AAZ↵     Skip PRN file

CP/M ASSEMBLER - VER 1.0

015C      Next address to assemble
003H USE FACTOR
END OF ASSEMBLY

A>DDT SORT.HEX↵     Test program changes

16K DDT VER 1.0
NEXT PC
015C 0000
-G100↵

*0118
-D148↵
                                Data sorted
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 ................

-Abort with rubout

-G0↵      Return to CP/M—program checks OK.
```

Section 8
CP/M Dynamic Debugging Tool

INTRODUCTION

The DDT program allows dynamic interactive testing and debugging of programs generated in the CP/M environment. Invoke the debugger with a command of one of the following forms:

```
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 place of the Console Command Processor (the user should refer to Chapter 5 for standard memory organization), and resides directly below the Basic Disk Operating System portion of CP/M. The BDOS starting address, 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 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. (The user should see the explanation of the I and R commands below for exact details.)

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

```
DDT VER m.m
```

where 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 retyping
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. 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 by using either a ctl-C or G0 (jmp to location 0000H), and save the current memory image by 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 is determined by taking the high order byte of the address in the TPA and converting this number to decimal. For example, if the highest address in the Transient Program Area is 1234H, the number of pages is 12H or 18 in decimal. The operator could type a ctl-C during the debug run, returning to the Console Command 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 typing the name X. If further testing is required, the memory image can be recalled by typing

DDT X.COM

which reloads the previously saved program from location 100H through page 18 (23FFH). The CPU state is not a part of the COM file; thus, the program must be restarted from the beginning to test it properly.

DDT COMMANDS

The individual commands are detailed below. 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. In the explanation of each command, the command letter is shown in some cases with numbers separated by commas, and 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" that 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).

The A (Assembly) Command

DDT allows in-line assembly language to be inserted into the current memory image using the A command, that 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).

The user should note that the assembler/disassembler portion of DDT can be overlaid 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.

The D (Display) Command

The D command allows the operator to view the contents of memory in hexadecimal and ASCII formats. The forms are

```
D
Ds
Ds,f
```

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

```
aaaa bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb cccccccccccccccc
```

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 to the right (represented by the sequence of c's), where nongraphic characters are printed as a period (.). The user should note that both upper and lower case alphabetics are displayed, and 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, with the first line truncated so that the next line begins at an address that is a multiple of 16.

The second form of the D command 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 return key.

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. DDT stores the constant c at address s, increments the value of s and tests against f. If s exceeds f, 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.

The G (Go) Command

A program is executed using the G command, with up to two optional breakpoint addresses. The G command takes the forms

```
G
Gs
Gs,b
Gs,b,c
G,b
G,b,c
```

The first form executes the program 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 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. There is no intervention between the starting address and the break address by DDT. 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 that 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.

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 Chapter 5). The default FCB can be used by the program under test as if it had been passed by the CP/M Console Processor. The user should 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

```
Ifilename.typ
```

If the second form is used and the filetype is either HEX or COM, subsequent R commands can be used to read the pure binary or hex format machine code. (Section 4.2.8 gives further details.)

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 form 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 (G and T commands). Again, long typeouts can be aborted using the return key during the list process.

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, and d is the destination address. Data are first removed from s to d, and both addresses are incremented. If s exceeds f, the move operation stops; otherwise, the move operation is repeated.

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 that is added to each program or data address as it is loaded. The load operation must not overwrite any of the system parameters from 000H 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 used for COM files. The user should note that any number of R commands can be issued following the I command to reread the program under test, assuming the tested program does not destroy the default area at 5CH. 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).

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 memory. If the operator types a carriage return, the data are not altered. If a byte value is typed, 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.

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 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.

The user should note that program tracing is discontinued at the CP/M interface and resumes after return from CP/M to the program under test. Thus, CP/M functions that 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 commands that use the breakpoint facility (G, T, and U) accomplish the break using an 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.

The operator should use the return key to get control back to DDT during trace, rather than executing an RST 7, to ensure that the trace for current instruction is completed before interruption.

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 (0FFFFH) 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.

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)
M	Minus flag	(0/1)
E	Even parity flag	(0/1)
I	Interdigit carry	(0/1)
A	Accumulator	(0-FF)
B	BC register pair	(0-FFFF)
D	DE register pair	(0-FFFF)
H	HL register pair	(0-FFFF)
S	Stack pointer	(0-FFFF)
P	Program counter	(0-FFFF)

In the first case, the CPU register state is displayed in the format

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, the flag or register value is not altered. If a value in the proper range is typed, the flag or register value is altered. The user should 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.

IMPLEMENTATION NOTES

The organization of DDT allows certain nonessential portions to be overlaid 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 that 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, the DDT program again alters the address field at 6H to include this module, 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.

AN EXAMPLE

The following example shows an edit, assemble, and debug for a simple program that 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

```
A>ED SCAN.ASM                Create source program;
                             "↓" represents carriage return.
*I↓
                ORG          1-00H          ;START  OF  TRANSIENT
                                            ;AREA↓
                MVI          B, LEN         ;LENGTH OF VECTOR TO SCAN↓
                MVI          C, 0           ;LARGER_RST VALUE SO FAR↓
LOOP            LXI          H, VECT        ;BASE OF VECTOR↓
LOOP:           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↓
                MOV          C, A
NFOUND          INX          H              ;TO NEXT ELEMENT↓
                DCR          B              ;MORE TO SCAN?↓
                JNZ          LOOP           ;FOR ANOTHER↓
;
;               END OF SCAN, STORE C↓
                MOV          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↓

↑-Z
*B0P↓
                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:           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
                MOV          C,A
NFOUND:         INX          H              ;TO NEXT ELEMENT
                DCR          B              ;MORE TO SCAN?
                JNZ          LOOP           ;FOR ANOTHER
;               END OF SCAN, STORE C
                MOV          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
*E↯ ← End of edit

A>ASM SCAN↯    Start Assembler

CP/M ASSEMBLER - VER 1.0

0122
002H USE FACTOR
END OF ASSEMBLY    Assembly complete; lock at program listing

A>TYPE SCAN.PRN↯
 Code address    Source program
 0100                          ORG  100H     ;START OF TRANSIENT AREA
 0100   0608                   MVI  B,LEN    ;LENGTH OF VECTOR TO SCAN
 0102   0E00  Machine code     MVI  C,0      ;LARGEST VALUE SO FAR
 0104   211901                 LXI  H,VECT.  ;BASE OF VECTOR
 0107   7E            LOOP:    MOV  A,M      ;GET VALUE
 0108   91                     SUB C         ;LARGER VALUE IN C?
 0109   D20D01                 JNC NFOUND    ;JUMP IF LARGER VALUE NOT
                               ;FOUND
                      ;        NEW LARGEST VALUE, STORE IT TO C
 010C   4F                     MOV  C, A
 010D   23            NFOUND:  INX  H        ;TO NEXT ELEMENT
 010E   05                     DCR  B        ;MORE TO SCAN?
 010F   C20701                 JNZ LOOP      ;FOR ANOTHER
                      ;
                      ;        END OF SCAN, STORE C
 0112   79                     MOV  A, C     ;GET LARGEST VALUE
 0113   322101                 STA LARGE
 0116   C30000                 JMP 0         ;REBOOT
    Code—data listing ;
    truncated         ;        TEST DATA
 0119   0200040305    VECT:    DB  2,0,4,3,5,6,1,5
 0008   = Value of    LEN      EQU$-VECT     ;LENGTH
 0121     equate      LARGE:   DS  1         ;LARGEST VALUE ON EXIT
 0122                          END

A>DDT SCAN.HEX↯    Start debugger using hex format machine code

DDT VER 1.0
NEXT PC                                                  Next instruction
0121 0000                                                to execute at
-X↯    Last load address + 1                             PC=0

C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0000 OUT 7F
-XP↯           Examine registers before debug run

P=0000 100↯    Change PC to 100

-X↯    Look at registers again
```

```
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI B,08
-L100↵
```

PC changed. Next instruction to execute at PC=100

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

Disassembled machine code at 100H (see source listing for comparison)

```
 0113    STA     0121
 0116    JMP     0000
 0119    STAX    B
 011A    NOP
 011B    INR     B
 011C    INX     B
 011D    DCR     B
 011E    MVI     B,01
 0120    DCR     B
 0121    LXI     D,2200
 0124    LXI     H,0200
```

A little more machine code. Note that program ends at location 116 with a JMP to 0000. Remainder of listing is assembly of data.

-A116↵ Enter in-line 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.

```
0116 RST 7

0117↵
```

(Single carriage return stops assemble mode)

-L113↵ List code at 113H to check that RST 7 was properly inserted

```
 0113    STA     0121
 0116    RST     07      in place of JMP
 0117    NOP
 0118    NOP
 0119    STAX    B
 011A    NOP
 011B    INR     B
 011C    INX     B
-
```

-X↵ Look at registers

```
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI B,08
-T↵
```

Execute Program for one stop. Initial CPU state, before is executed

```
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI B,08*0102
-T↵
```

Automatic breakpoint

Trace one step again (note O8H in B)

```
C0Z0M0E0I0 A=00 B=0800 D=0000 H=0000 S=0100 P=0102 MVI C,00*0104
-T↵
```

Trace again (Register C is cleared)

```
C0Z0M0E0I0 A=00 B=0800 D=0000 H=0000 S=0100 P=0104 LXI H,0119*0107
-T3↵ Trace three steps
C0Z0M0E0I0 A=00 B=0800 D=0000 H=0119 S=0100 P=0107 MOV A,M
C0Z0M0E0I0 A=02 B=0800 D=0000 H=0119 S=0100 P=0108 SUB C
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=0109 JNC 010D*010D
-D119↵
```

Display memory starting at 119H. Automatic breakpoint at 10DH

Program data

Lowercase x

```
0119 02 00 04 03 05 06 01 . . . . . .
0120 05 11 00 22 21 00 02 7E EB 77 13 23 EB 0B 78 B1 . . . " ! . . . W . # . . X
0130 C2 27 01 C3 03 29 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 . . . . . .
0150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . . .
0160 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 . . . . . .
0180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . . .
0190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . . .
01A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . . .
01B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . . .
01C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . . .
-X↵
```

Data are displayed in ASCII with a "." in the position of nongraphic characters

Current CPU state

```
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=010D INX H
-T5↵
```

Trace 5 steps from current CPU state

```
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=010D INX H
C0Z0M0E0I1 A=02 B=0800 D=0000 H=011A S=0100 P=010E DCR B
C0Z0M0E0I1 A=02 B=0700 D=0000 H=011A S=0100 P=010F JNZ 0107
C0Z0M0E0I1 A=02 B=0700 D=0000 H=011A S=0100 P=0107 MOV A,M
C0Z0M0E0I1 A=00 B=0700 D=0000 H=011A S=0100 P=0108 SUB C*0109
U5↵
```

Automatic breakpoint

Trace without listing intermediate states

```
C0Z1M0E1I1 A=00 B=0700 D=0000 H=011A S=0100 P=0109 JNC 010D*0108
-X↵
```

CPU state at end of U5

```
C0Z0M0E1I1 A=04 B=0600 D=0000 H=011B S=0100 P=0108 SUB C
-G↵ Run program from current PC until completion (in real-time)

*0116 breakpoint at 116H, caused by executing RST 7 in machine code.
-X↵
```

CPU state at end of program

```
C0Z1M0E1I1 A=00 B=0000 D=0000 H=0121 S=0100 P=0116 RST 07
-XP↵
```

Examine and change program counter

```
P=0116 100↵

-X↵

C0Z1M0E1I1 A=00 B=0000 D=0000 H=0121 S=0100 P=0100 MVI B,08
-T10↵
```

First data element

Current largest value

Subtract for comparison, C

Trace 10 (hexadecimal) steps

```
C0Z1M0E1I1 A=00 B=0800 D=0000 H=0121 S=0100 P=0100 MVI B,08
C0Z1M0E1I1 A=00 B=0000 D=0000 H=0121 S=0100 P=0102 MVI C,00
C0Z1M0E1I1 A=00 B=0800 D=0000 H=0121 S=0100 P=0104 LXI H,0119
C0Z1M0E1I1 A=00 B=0800 D=0000 H=0119 S=0100 P=0107 MOV A,M
C0Z1M0E1I1 A=02 B=0800 D=0000 H=0119 S=0100 P=0108 SUB C
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=0109 JNC 010D
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=010D INX H
C0Z0M0E0I1 A=02 B=0800 D=0000 H=011A S=0100 P=010E DCR B
C0Z0M0E0I1 A=02 B=0700 D=0000 H=011A S=0100 P=010F JNZ 0107
C0Z0M0E0I1 A=02 B=0700 D=0000 H=011A S=0100 P=0107 MOV A,M
C0Z0M0E0I1 A=00 B=0700 D=0000 H=011A S=0100 P=0108 SUB C
C0Z1M0E1I1 A=00 B=0700 D=0000 H=011A S=0100 P=0109 JNC 010D
C0Z1M0E1I1 A=00 B=0700 D=0000 H=011A S=0100 P=010D INX H
C0Z1M0E1I1 A=00 B=0700 D=0000 H=011B S=0100 P=010E DCR B
C0Z0M0E1I1 A=00 B=0600 D=0000 H=011B S=0100 P=010F JNZ 0107
C0Z0M0E1I1 A=00 B=0600 D=0000 H=011B S=0100 P=0107 MOV A,M*0108
-A109↵
```

Insert a "hot patch" into the machine code to change the JNC to JC

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

```
0109 JC 10D↵

010C↵

-G0↵
```

Stop DDT so that a version of the patched program can be saved

```
A>SAVE 1 SCAN.COM↵
```

Program resides on first page, so save 1 page.

```
A>DDT SCAN.COM↵
```

Restart DDT with the save memory image to continue testing

```
DDT VER 1.0
NEXT PC
0200 0100
-L100↵
```

List some code

```
 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
 -XP↵

P=0100↵
```

Previous patch is present in X.COM

```
-T10↵
    Trace to see how patched version operates                Data is moved from A to C
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI  B,08
C0Z0M0E0I0 A=00 B=0800 D=0000 H=0000 S=0100 P=0102 MVI  C,00
C0Z0M0E0I0 A=00 B=0800 D=0000 H=0000 S=0100 P=0104 LXI  H,0119
C0Z0M0E0I0 A=00 B=0800 D=0000 H=0119 S=0100 P=0107 MOV  A,M
C0Z0M0E0I0 A=02 B=0800 D=0000 H=0119 S=0100 P=0108 SUB  C
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=0109 JC   010D
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=010C MOV  C,A
C0Z0M0E0I1 A=02 B=0802 D=0000 H=0119 S=0100 P=010D INX  H
C0Z0M0E0I1 A=02 B=0802 D=0000 H=011A S=0100 P=010E DCR  B
C0Z0M0E0I1 A=02 B=0702 D=0000 H=011A S=0100 P=010F JNZ  0107
C0Z0M0E0I1 A=02 B=0702 D=0000 H=011A S=0100 P=0107 MOV  A,M
C0Z0M0E0I1 A=00 B=0702 D=0000 H=011A S=0100 P=0108 SUB  C
C1Z0M1E0I0 A=FE B=0702 D=0000 H=011A S=0100 P=0109 JC   010D
C1Z0M1E0I0 A=FE B=0702 D=0000 H=011A S=0100 P=010D INX  H
C1Z0M1E0I0 A=FE B=0702 D=0000 H=011B S=0100 P=010E DCR  B
C1Z0M0E1I1 A=FE B=0602 D=0000 H=011B S=0100 P=010F JNZ  0107*0107
-X↵                                                Breakpoint after 16 steps

C1Z0M0E1I1 A=FE B=0602 D=0000 H=011B S=0100 P=0107 MOV  A,M
-G,108↵    Run from current PC and breakpoint at 108H

*0108
-X↵
                    Next data item
C1Z0M0E1I1 A=04 B=0602 D=0000 H=011B S=0100 P=0108 SUB  C
-T↵
    Single step for a few cycles
C1Z0M0E1I1 A=04 B=0602 D=0000 H=011B S=0100 P=0108 SUB  C*0109
-T↵

C0Z0M0E0I1 A=02 B=0602 D=0000 H=011B S=0100 P=0109 JC   010D*010C
-X↵

C0Z0M0E0I1 A=02 B=0602 D=0000 H=011B S=0100 P=010C MOV  C,A
-G↵    Run to completion

*0116
-X↵

C0Z1M0E1I1 A=03 B=0003 D=0000 H=0121 S=0100 P=0116 RST  07
-S121↵    Look at the value of "LARGE"

 0121    03↵    Wrong value!

 0122    00↵

 0123    22↵

 0124    21↵
```

```
 0125    00↯

 0126    02↯

 0127    7E↯ .      End of the S command

-L100↯

 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
 -L↯                                  Review the code

 0113    STA        0121
 0116    RST        07
 0117    NOP
 0118    NOP
 0119    STAX       B
 011A    NOP
 011B    INR        B
 011C    INX        B
 011D    DCR        B
 011E    MVI        B,01
 0120    DCR        B
 -XP↯

P=0116  100↯    Reset the PC

-T↯
      Single step, and watch data values
C0Z1M0E1I1  A=03  B=0003  D=0000  H=0121  S=0100  P=0100  MVI  B,08*0102
-T↯

C0Z1M0E1I1  A=03  B=0803  D=0000  H=0121  S=0100  P=0102  MVI  C,00*0104
-T↯
            Count set ↘   ↙ "Largest" set
C0Z1M0E1I1  A=03  B=0800  D=0000  H=0121  S=0100  P=0104  LXI  H,0119*0107
-T↯
                                          ↙ Base address of data set
C0Z1M0E1I1  A=03  B=0800  D=0000  H=0119  S=0100  P=0107  MOV  A,M*0108
-T↯
                  ↙ First data item brought to A
C0Z1M0E1I1  A=02  B=0800  D=0000  H=0119  S=0100  P=0108  SUB  C*0109
-T↯

C0Z0M0E0I1  A=02  B=0800  D=0000  H=0119  S=0100  P=0109  JC  010D*010C
-T↯
```

```
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=010C MOV C,A*010D
-T↯
                          First data item moved to C correctly
C0Z0M0E0I1 A=02 B=0802 D=0000 H=0119 S=0100 P=010D INX H*010E
-T↯

C0Z0M0E0I1 A=02 B=0802 D=0000 H=011A S=0100 P=010E DCR B*010F
-T↯

C0Z0M0E0I1 A=02 B=0702 D=0000 H=011A S=0100 P=010F JNZ 0107*0107
-T↯

C0Z0M0E0I1 A=02 B=0702 D=0000 H=011A S=0100 P=0107 MOV A,M*0108
-T↯
                          Second data item brought to A
C0Z0M0E0I1 A=00 B=0702 D=0000 H=011A S=0100 P=0108 SUB C*0109
-T↯
                          Subtract destroys data value that was loaded!
C1Z0M1E0I0 A=FE B=0702 D=0000 H=011A S=0100 P=0109 JC 010D*010D
-T↯

C1Z0M1E0I0 A=FE B=0702 D=0000 H=011A S=0100 P=010D INX H*010E
-L100↯

 0100    MVI     B,08
 0102    MVI     C,00
 0104    LXI     H,0119
 0107    MOV     A,M
 0108    SUB     C ← This should have been a CMP so that register A
 0109    JC      010D  would not be destroyed.
 010C    MOV     C,A
 010D    INX     H
 010E    DCR     B
 010F    JNZ     0107
 0112    MOV     A,C
 -A108↯

0108 CMP C↯      Hot patch at 108H changes SUB to CMP

0109

-G0↯     Stop DDT for SAVE

A> SAVE 1 SCAN.COM↯     Save memory image

A>DDT SCAN.COM↯     Restart DDT

DDT VER 1.0
NEXT  PC
0200 0100
-XP↯

P=0100

-L116↯
```

```
0116    RST     07
0117    NOP
0118    NOP
0119    STAX    B
011A    NOP
-
```

Look at code to see if it was properly loaded (long typeout aborted with rubout)

```
-G,116↯
```

Run from 100H to completion

```
*0116
-XC↯
```

Look at carry (accidental typo)

```
C1↯
-X↯
```

Look at CPU state

```
C1Z1M0E1I1 A=06 B=0006 D=0000 H=0121 S=0100 P=0116 RST 07
-S121↯
```

Look at "large"—it appears to be correct.

```
0121 06↯

0122 00↯

0123 22

-G0↯
```

Stop DDT

```
A>ED SCAN.ASM↯
```

Re-edit the source program, and make both changes

```
*NSUB↯
*0LT↯
      ctl-Z↘      SUB     C          ;LARGER VALUE IN C?
*SSUB↑ZCMP↑Z0LT↯
                  CMP     C          ;LARGER VALUE IN C?
*
                  JNC     NFOUND     ;JUMP IF LARGER VALUE NOT FOUND
*SNC↑ZC↑Z0LT↯
                  JC      NFOUND     ;JUMP IF LARGER VALUE NOT FOUND
*E↯
```

Re-assemble, selecting source from disk A

```
A>ASM SCAN.AAZ↯ ←
```

Hex to disk A
Print to Z (selects no print file)

```
CP/M ASSEMBLER    VER 1.0

0122
002H USE FACTOR
END OF ASSEMBLY
```

```
A>DDT SCAN.HEX↯     Re-run debugger to check changes

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
 - (rubout)

-G100,116↯    Go from beginning with breakpoint at end

*0116         Breakpoint reached
-D121↯        Look at "LARGE"
                          Correct value computed
0121  06 00 22 21 00 02 7E EB 77 13 23 EB 0B 78 B1 . . '!...W.#..X.
0130  C2 27 01 C3 03 29 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 ................

- (rubout)  Aborts long type-out

G0↯    Stop DDT, debug session complete.
```

Appendix A
Personal CP/M Messages

Messages come from several different sources. Personal CP/M displays error messages when there are errors in calls to the Basic Disk Operating System (BDOS). Personal CP/M also displays messages when there are errors in command lines. Each utility supplied with Personal CP/M has its own set of messages. The following lists Personal CP/M messages and utility messages. You might see messages other than those listed here if you are running an application program. Check the application program's documentation for explanations of those messages.

Table A-1. Personal CP/M Error Messages

Message	Meaning
ABORTED	PIP. You stopped a PIP operation by pressing a key.
BAD DELIMETER	STAT. Check command line for typing errors.
Bad Load	CCP error message, or SAVE error message.
CP/M Error on d: Disk I/O	The disk I/O error results from an error condition returned to the BDOS from the BIOS module. The file system makes BIOS read and write calls to execute file-related BDOS calls. If the BIOS read or write routine detects an error, it returns an error code to the BDOS resulting in this error.

Table A-1. (continued)

Message	Meaning
CP/M Error on d: Invalid Drive	The invalid drive error results from an error condition returned to the BDOS from the BIOS module. The BDOS makes a BIOS SELECT DISK call prior to accessing a drive to perform a requested BDOS function. If the BIOS does not support the selected disk, the BDOS returns an error code resulting in this error message.
CP/M Error on d: Read/Only File	The Read/Only file error is returned when a program attempts to write to a file that is marked with the Read/Only attribute. It is also returned to a program that attempts to write to a system file opened under user zero from a nonzero user number.
CP/M Error on d: Read/Only Disk	The Read/Only Disk error is returned when a program writes to a disk that is in Read/Only status. A drive can be placed in Read/Only status explicitly with the BDOS WRITE PROTECT DISK function.
Break "x" at c	ED. "x" is one of the symbols described below and c is the command letter being executed when the error occurred. # Search failure. ED cannot find the string specified in an F, S, or N command. ? Unrecognized command letter c. ED does not recognize the indicated command letter, or an E, H, Q, or O command is not alone on its command line. O The file specified in an R command cannot be found.

Table A-1. (continued)

Message	Meaning
	> Buffer full. ED cannot put any more characters in the memory buffer, or the string specified in an F, N, or S command is too long. E Command aborted. A keystroke at the console aborted command execution. F Disk or directory full. This error is followed by either the disk or directory full message. Refer to the recovery procedures listed under these messages.
CANNOT CLOSE DESTINATION FILE--{filespec}	PIP. An output file cannot be closed. Take appropriate action after checking to see if the correct disk is in the drive and that the disk is not write protected.
Cannot close, R/O CANNOT CLOSE FILES	CP/M cannot write to file. This usually occurs because the disk is write protected. SUBMIT. This error can occur during SUBMIT file processing. Check if the correct system disk is in the A drive and that the disk is not write protected. The SUBMIT job can be restarted after rebooting Personal CP/M.
CANNOT READ	PIP. PIP cannot read the specified source. Reader might not be implemented.
CANNOT WRITE	PIP. The destination specified in the PIP command is illegal. You probably specified an input device as a destination.

Table A-1. (continued)

Message	Meaning
Checksum error	PIP. A hex record checksum error was encountered. The hex record that produced the error must be corrected, probably by recreating the hex file.
CHECKSUM ERROR LOAD ADDRESS hhhh ERROR ADDRESS hhhh BYTES READ: hhhh:	LOAD. File contains incorrect data. Regenerate hex file from the source.
Command Buffer Overflow	SUBMIT. The SUBMIT buffer allows up to 2048 characters in the input file.
Command too long	SUBMIT. A command in the SUBMIT file cannot exceed 125 characters.
CORRECT ERROR, TYPE RETURN OR CTL-Z	PIP. A hex record checksum was encountered during the transfer of a hex file. Correct the hex file with the checksum error, probably by recreating the hex file.
DESTINATION IS R/O, DELETE (Y/N)?	PIP. The destination file specified in a PIP command already exists and it is Read/Only. If you type Y, the destination file is deleted before the file copy is done.

Table A-1. (continued)

Message	Meaning
Directory full	ED. There is not enough directory space for the file being written to the destination disk. You can use the OXfilespec command to erase unnecessary files on the disk without leaving the editor. SUBMIT. There is not enough directory space to write the $$$.SUB file used for processing SUBMITs. Erase some files or select a new disk and retry.
Disk full	ED. There is not enough disk space for the output file. This error can occur on the W, E, H, or X commands. If it occurs with X command, you can repeat the command prefixing the filename with a different drive.
DISK READ ERROR-{filespec}	PIP. The input disk file specified in a PIP command cannot be read properly. This is usually the result of an unexpected end-of-file. Correct the problem in your file.
DISK WRITE ERROR-{filespec}	PIP. A disk write operation cannot be successfully performed during a PIP command, probably due to a full disk. Erase some unnecessary files, or get another disk with more space and execute PIP again. SUBMIT. The SUBMIT program cannot write the $$$.SUB file to the disk. Erase some files, or select a new disk and try again.

Table A-1. (continued)

Message	Meaning
ERROR: BAD PARAMETER	PIP. You entered an illegal parameter in PIP command. Retype the entry correctly.
ERROR: CANNOT OPEN SOURCE, LOAD ADDRESS hhhh	LOAD. Displayed if LOAD cannot find the specified file or if no filename is specified.
ERROR: CANNOT CLOSE FILE, LOAD ADDRESS hhhh	LOAD. Caused by an error code returned by a BDOS function call. Disk might be write protected.
ERROR: CANNOT OPEN SOURCE, LOAD ADDRESS hhhh	LOAD. Cannot find source file. Check disk directory.
ERROR: DISK READ, LOAD ADDRESS hhhh	LOAD. Caused by an error code returned by a BDOS function call.
ERROR: DISK WRITE, LOAD ADDRESS hhhh	LOAD. Destination Disk is full.
ERROR: INVERTED LOAD ADDRESS, LOAD ADDRESS hhhh	LOAD. The address of a record was too far from the address of the previously processed record. This is an internal limitation of LOAD, but it can be circumvented. Use DDT to read the hexfile into memory, then use a SAVE command to store the memory image file on disk.

Table A-1. (continued)

Message	Meaning
ERROR: NO MORE DIRECTORY SPACE, LOAD ADDRESS hhhh	LOAD. Disk directory is full.
Error on line nnn message	SUBMIT. The SUBMIT program displays its messages in the format shown above, where nnn represents the line number of the SUBMIT file. Refer to the message following the line number.
FILE ERROR	ED. Disk or directory full, and ED cannot write anything more on the disk. This is a fatal error, so make sure there is enough space on the disk to hold a second copy of the file before invoking ED.
FILE EXISTS	You have asked Personal CP/M to create or rename a file using a file specification that is already assigned to another file. Delete the existing file or use another file specification. REN. The new name specified is the name of a file that already exists. You cannot rename a file with the name of an existing file. If you want to replace an existing file with a newer version of the same file, either rename or erase the existing file, or use the PIP utility.
File exists, erase it	ED. The destination filename already exists when you are placing the destination file on a disk different from the source. Erase the file or select another disk to receive the output file.

Table A-1. (continued)

Message	Meaning
** FILE IS READ/ONLY **	ED. The file specified in the command to invoke ED has the Read/Only attribute. ED can read the file, so the user can examine it, but ED cannot change a Read/Only file.
File Not Found	Personal CP/M cannot find the specified file. Check that you have entered the correct drive specification or that you have the correct disk in the drive. ED. ED cannot find the specified file. Check that you have entered the correct drive specification or that you have the correct disk in the drive. STAT. STAT cannot find the specified file. The message might appear if you omit the drive specification. Check to see if the correct disk is in the drive.
FILE NOT FOUND-{filespec}	PIP. You specified an input file that does not exist.
Filename required	ED. You typed the ED command without a filename. Reenter the ED command followed by the name of the file you want to edit or create.

Table A-1. (continued)

Message	Meaning
Invalid Assignment	STAT. You specified an invalid drive or file assignment, or misspelled a device name. This error message might be followed by a list of the valid file assignments that can follow a filename. If an invalid drive assignment was attempted, the message "Use: d:=R/O" is displayed, showing the proper syntax for drive assignments.
Invalid control character	SUBMIT. The only valid control characters in the SUBMIT files of the type SUB are ^A through ^Z. Note that in a SUBMIT file the control character is represented by typing the circumflex, ^, not by pressing the control key.
INVALID DIGIT-{filespec}	PIP. An invalid hex digit has been encountered while reading a hex file. Correct the hex file with the invalid hex digit by recreating the hex file.
Invalid Disk Assignment	STAT. Might appear if you follow the drive specification with anything except =R/O.
Invalid File Indicator	STAT. Appears if you do not specify R/O, R/W, DIR, or SYS.
INVALID FORMAT	PIP. The format of your PIP command is illegal. See the description of PIP.

Table A-1. (continued)

Message	Meaning
INVALID HEX DIGIT LOAD ADDRESS hhhh ERROR ADDRESS hhhh BYTES READ: hhhh	LOAD. File contains incorrect hex digit.
INVALID SEPARATOR	PIP. You have placed an invalid character for a separator between two input filenames.
INVALID USER NUMBER	PIP. You have specified a user number greater than 15. User numbers are in the range 0 to 15.
n?	USER. You specified a user area number greater than 15. For example, if you type USER 18<cr>, the screen displays "18?"
NO DIRECTORY SPACE-{filespec}	PIP. There is not enough directory space for the output file. Erase some unnecessary files, or get another disk with more directory space and execute PIP again.
NO FILE-{filespec}	DIR, ERA, REN, PIP. Personal CP/M cannot find the specified file, or no files exist.

Table A-1. (continued)

Message	Meaning
NO INPUT FILE PRESENT ON DISK	DUMP. The file you requested does not exist.
No memory	There is not enough memory available for loading the program specified.
NO SPACE	SAVE. Too many files are already on the disk, or no room is left on the disk to save the information.
No SUB file present	SUBMIT. For SUBMIT to operate properly, you must create a file with filetype of SUB. The SUB file contains usual Personal CP/M commands. Use one command per line.
NOT A CHARACTER SOURCE	PIP. The source specified in your PIP command is illegal. You have probably specified an output device as a source.
** NOT DELETED **	PIP. PIP did not delete the file, which might have had the R/O attribute.
NOT FOUND	PIP. PIP cannot find the specified file.
Parameter error	SUBMIT. Within the SUBMIT file of type sub, valid parameters are $0 through $9.

Table A-1. (continued)

Message	Meaning
QUIT NOT FOUND	PIP. The string argument to a Q parameter was not found in your input file.
Read error	TYPE. An error occurred when reading the file specified in the type command. Check the disk and try again. The STAT filespec command can diagnose trouble.
READER STOPPING	PIP. Reader operation interrupted.
Record Too Long	PIP. PIP cannot process a record longer than 128 bytes.
Requires CP/M 2.0 or later	XSUB. XSUB requires the facilities of CP/M 2.0 or newer version.
Requires Personal CP/M 1.0 or newer for operation	PIP. This version of PIP requires the facilities of Personal CP/M 1.0 or newer version.
START NOT FOUND	PIP. The string argument to an S parameter cannot be found in the source file.
"SYSTEM" FILE NOT ACCESSIBLE	You tried to access a file set to SYS with the STAT command.

Table A-1. (continued)

Message	Meaning
** TOO MANY FILES **	STAT. There is not enough memory for STAT to sort the files specified, or more than 512 files were specified.
UNEXPECTED END OF HEX FILE-{FILESPEC}	PIP. An end-of-file was encountered prior to a termination hex record. The hex file without a termination record should be corrected, probably by recreating the hex file.
Unrecognized Destination	PIP. Check command line for valid destination.
Use: STAT d=R/O	STAT. An invalid STAT drive command was given. The only valid drive assignment in STAT is STAT d:R/O.
VERIFY ERROR:-{filespec}	PIP. When copying with the V option, PIP found a difference when rereading the data just written and comparing it to the data in its memory buffer. Usually this indicates a failure of either the destination disk or drive.

Table A-1. (continued)

Message	Meaning
WRONG CP/M VERSION (REQUIRES 2.0)	XSUB ACTIVE SUBMIT. XSUB has been invoked.
XSUB ALREADY PRESENT	SUBMIT. XSUB is already active in memory.
YOUR INPUT?	If CP/M cannot find the command you specified, it returns the command name you entered followed by a question mark. Check that you have typed the command line correctly, or that the command you requested exists as a COM file on the default or specified disk.

End of Appendix A

Appendix B
ASCII and Hexadecimal Conversions

ASCII stands for American Standard Code for Information Interchange. The code contains 96 printing and 32 nonprinting characters used to store data on a disk. Table B-1 defines ASCII symbols; Table B-2 lists the ASCII and hexadecimal conversions. Table B-2 includes binary, decimal, hexadecimal, and ASCII conversions.

Table B-1. ASCII Symbols

Symbol	Meaning	Symbol	Meaning
ACK	acknowledge	FS	file separator
BEL	bell	GS	group separator
BS	backspace	HT	horizontal tabulation
CAN	cancel	LF	line feed
CR	carriage return	NAK	negative acknowledge
DC	device control	NUL	null
DEL	delete	RS	record separator
DLE	data link escape	SI	shift in
EM	end of medium	SO	shift out
ENQ	enquiry	SOH	start of heading
EOT	end of transmission	SP	space
ESC	escape	STX	start of text
ETB	end of transmission	SUB	substitute
ETX	end of text	SYN	synchronous idle
FF	form feed	US	unit separator
		VT	vertical tabulation

Table B-2. ASCII Conversion Table

Binary	Decimal	Hexadecimal	ASCII
0000000	0	0	NUL
0000001	1	1	SOH (CTRL-A)
0000010	2	2	STX (CTRL-B)
0000011	3	3	ETX (CTRL-C)
0000100	4	4	EOT (CTRL-D)
0000101	5	5	ENQ (CTRL-E)
0000110	6	6	ACK (CTRL-F)
0000111	7	7	BEL (CTRL-G)
0001000	8	8	BS (CTRL-H)
0001001	9	9	HT (CTRL-I)
0001010	10	A	LF (CTRL-J)
0001011	11	B	VT (CTRL-K)
0001100	12	C	FF (CTRL-L)
0001101	13	D	CR (CTRL-M)
0001110	14	E	SO (CTRL-N)
0001111	15	F	SI (CTRL-O)
0010000	16	10	DLE (CTRL-P)
0010001	17	11	DC1 (CTRL-Q)
0010010	18	12	DC2 (CTRL-R)
0010011	19	13	DC3 (CTRL-S)
0010100	20	14	DC4 (CTRL-T)
0010101	21	15	NAK (CTRL-U)
0010110	22	16	SYN (CTRL-V)
0010111	23	17	ETB (CTRL-W)
0011000	24	18	CAN (CTRL-X)
0011001	25	19	EM (CTRL-Y)
0011010	26	1A	SUB (CTRL-Z)
0011011	27	1B	ESC (CTRL-[)
0011100	28	1C	FS (CTRL-\)
0011101	29	1D	GS (CTRL-])
0011110	30	1E	RS (CTRL-^)
0011111	31	1F	US (CTRL-_)
0100000	32	20	(SPACE)
0100001	33	21	!
0100010	34	22	"
0100011	35	23	#
0100100	36	24	$
0100101	37	25	%
0100110	38	26	&
0100111	39	27	'
0101000	40	28	(
0101001	41	29	)
0101010	42	2A	*
0101011	43	2B	+
0101100	44	2C	,
0101101	45	2D	-
0101110	46	2E	.
0101111	47	2F	/

Table B-2. (continued)

Binary	Decimal	Hexadecimal	ASCII
0110000	48	30	0
0110001	49	31	1
0110010	50	32	2
0110011	51	33	3
0110100	52	34	4
0110101	53	35	5
0110110	54	36	6
0110111	55	37	7
0111000	56	38	8
0111001	57	39	9
0111010	58	3A	:
0111011	59	3B	;
0111100	60	3C	<
0111101	61	3D	=
0111110	62	3E	>
0111111	63	3F	?
1000000	64	40	@
1000001	65	41	A
1000010	66	42	B
1000011	67	43	C
1000100	68	44	D
1000101	69	45	E
1000110	70	46	F
1000111	71	47	G
1001000	72	48	H
1001001	73	49	I
1001010	74	4A	J
1001011	75	4B	K
1001100	76	4C	L
1001101	77	4D	M
1001110	78	4E	N
1001111	79	4F	O
1010000	80	50	P
1010001	81	51	Q
1010010	82	52	R
1010011	83	53	S
1010100	84	54	T
1010101	85	55	U
1010110	86	56	V
1010111	87	57	W
1011000	88	58	X
1011001	89	59	Y
1011010	90	5A	Z
1011011	91	5B	[
1011100	92	5C	\
1011101	93	5D	]
1011110	94	5E	^

Table B-2. (continued)

Binary	Decimal	Hexadecimal	ASCII
1011111	95	5F	<
1100000	96	60	'
1100001	97	61	a
1100010	98	62	b
1100011	99	63	c
1100100	100	64	d
1100101	101	65	e
1100110	102	66	f
1100111	103	67	g
1101000	104	68	h
1101001	105	69	i
1101010	106	6A	j
1101011	107	6B	k
1101100	108	6C	l
1101101	109	6D	m
1101110	110	6E	n
1101111	111	6F	o
1110000	112	70	p
1110001	113	71	q
1110010	114	72	r
1110011	115	73	s
1110100	116	74	t
1110101	117	75	u
1110110	118	76	v
1110111	119	77	w
1111000	120	78	x
1111001	121	79	y
1111010	122	7A	z
1111011	123	7B	{
1111100	124	7C	|
1111101	125	7D	}
1111110	126	7E	~
1111111	127	7F	DEL

End of Appendix B

Appendix C
Filetypes

Personal CP/M identifies every file by a unique file specification, which consists of a drive specification, a filename, a filetype, and an optional password. The filetype is an optional three-character ending separated from the filename by a period. The filetype indicates a special kind of file. Table C-1 lists common filetypes and their meanings.

Table C-1. Common Filetypes

Type	Meaning
ASM	Assembly language source file; the Personal CP/M assemblers assemble or translate a type ASM file into machine language.
BAK	Backup file created by text editor; the editor renames the source file with this filetype to indicate that the original file has been processed. The original file stays on disk as the backup file, so you can refer to it.
BAS	CBASIC program source file.
COM	Machine lanquage program (Z80, 8080, or 8085).
ERL	Pascal/MT+™ relocatable file.
HEX	Program file in hexadecimal format.
INT	CBASIC program intermediate language file.
IRL	Indexed REL file produced by LIB.
LIB	Used by MAC™ and RMAC™ for macro libraries. The ED R command reads files of type LIB. The ED X command writes files of type LIB. Printable file displayable on console or printer.
OVL	Program overlay file. PL/I-80™ compiler overlays files; you can create overlay files with LINK-80™ .
PAS	Pascal/MT+ source program filetype.
PLI	PL/I-80 source program filetype.

Table C-1. (continued)

Type	Meaning
PRL	Page Relocatable file; a file that does not require an absolute segment. It can be relocated in any page boundary (256 Bytes).
PRN	Printable file displayable on console or printer.
REL	Relocatable file produced by RMAC and PL/I-80 that can be linked by LINK-80.
SPR	System Page Relocatable file; system files required to generate Personal CP/M, such as BNKBDOS.SPR, BDOS.SPR, BIOS.SPR, and RESBDOS.SPR.
SUB	Filetype required for submit file containing one or more Personal CP/M commands. The SUBMIT program executes commands in files of type SUB, providing a batch execution mode for Personal CP/M.
SYM	Symbol table file. MAC, RMAC, and LINK-80 output files of type SYM. SID™ and ZSID™ read files of type SYM.
SYS	System file for Personal CP/M.
TEX	Source file for TEX™ , the Digital Research text formatter.
TOK	Pascal/MT+ intermediate language file.
XRF	Cross-reference file produced by XREF.
$$$	Temporary file.

End of Appendix C

Appendix D
Personal CP/M Control Character Summary

Table D-1. Personal CP/M Control Characters

Character	Meaning
CTRL-C	Warm boot (restarts) the CP/M operating system when typed at the beginning of a line.
CTRL-E	Forces a physical carriage return but does not send the command line to Personal CP/M. Moves the cursor to the beginning of the next line without erasing your previous input.
CTRL-H	Deletes a character and moves the cursor left one character position.
CTRL-J	Sends the command line to Personal CP/M and returns the cursor to the left of the current line. Has the same effect as a RETURN or a CTRL-M.
CTRL-M	Sends the command line to Personal CP/M and returns the cursor to the left of the current line. Has the same effect as a RETURN or a CTRL-J.
CTRL-R	Places a # sign at the current cursor location, moves the cursor to the next line, and displays any partial command you typed so far.
CTRL-U	Discards all the characters in the command line (but leaves them displayed), places a # at the current cursor position, and moves the cursor to the next command line.
CTRL-X	Discards all the characters in the command line (actually removes them from display), and moves the cursor to the beginning of the current line.
RUBOUT	Deletes the last character typed and echoes it at the console.

End of Appendix D

Index

W

X

Z

Glossary

ambiguous filename: Filename that contains either of the Personal CP/M wildcard characters, ? or *, in the primary filename, the filetype, or both. When you use wildcard characters, you create an ambiguous filespec and can easily reference more than one Personal CP/M file. See Section 2 of this manual.

applications program: Program that solves a specific problem. Typical applications programs are business accounting packages, word processing (editing) programs, and mailing list programs.

argument: Symbol indicating a place into which you can substitute a number, letter, or name to give an appropriate meaning to a command line.

ASCII: The American Standard Code for Information Interchange is a standard code for representation of numbers, letters, and symbols. An ASCII text file is a file that can be intelligibly displayed on the video screen or printed on paper.

attribute: File characteristic that can be set to on or off.

backup: Copy of a disk or file made for safe keeping, or the creation of the backup disk or file.

bit: Switch in memory that can be set to on (1) or off (0). Bits are grouped into bytes.

block: Area of disk.

bootstrap: Process of loading an operating system into memory. Bootstrap procedures vary from system to system. The boot for an operating system must be customized for the memory size and hardware environment that the operating system manages. Typically, the boot is loaded automatically and executed at power up or when the computer is reset. Sometimes called a cold start.

buffer: Area of memory that temporarily stores data during the transfer of information.

built-in commands: Commands that permanently reside in memory. They respond quickly because they are not accessed from a disk.

byte: Unit of memory or disk storage containing eight bits.

character string: Any combination of letters, numbers, or special characters on your keyboard.

command: Elements of a Personal CP/M command line. In general, a Personal CP/M command has three parts: the command keyword, the command tail, and a carriage return keystroke.

command file: Series of coded machine executable instructions stored on disk as a program file, invoked in Personal CP/M by typing the command keyword next to the system prompt on the console. Personal CP/M command files generally have a filetype of COM. Files are either command files or data files. Same as a command program.

command keyword: Name that identifies an Personal CP/M command, usually the primary filename of a file of type COM, or a built-in command. The command keyword precedes the command tail and the carriage return in the command line.

command syntax: Statement that defines the correct way to enter a command. The correct structure generally includes the command keyword, the command tail, and a carriage return. A syntax line usually contains symbols that you should replace with actual values when you enter the command.

command tail: Part of a command that follows the command keyword in the command line. The command tail can include a drive specification, a filename and/or filetype, and options or parameters, but cannot exceed 128 characters. Some commands do not require a command tail.

concatenate: Term that describes one of PIP's operations that combines two or more separate files into one new file in the specified sequence.

console: Primary input/output device. The console consists of a listing device such as a screen and a keyboard through which the user communicates with the operating system or applications program.

control character: Nonprinting character combination that sends a simple command to Personal CP/M. Some control characters perform line editing functions. To enter a control character, hold down the CTRL key on your terminal and strike the character key specified. See Appendix D.

cursor: One-character symbol that can appear anywhere on the console screen. The cursor indicates the position where the next keystroke at the console will have an effect.

data file: Nonexecutable collection of similar information that generally requires a command file to manipulate it.

default: Currently selected disk drive and/or user number. Any command that does not specify a disk drive or a user number references the default disk drive and user number. When Personal CP/M is first invoked, the default disk drive is drive A, and the default user number is 0, until changed with the USER command.

delimiter: Special characters that separate different items in a command line. For example, in Personal CP/M, a colon separates the drive spec from the filename. A period separates the filename from the filetype. Brackets separate any options from their command or filespec. Commas separate one item in an option list from another. All of the preceding special characters are delimiters.

directory: Portion of a disk that contains descriptions of each file on the disk. In response to the DIR command, Personal CP/M displays the filenames stored in the directory.

DIR attribute: File attribute. A file with the DIR attribute can be displayed by a DIR command. The file can be accessed from the default user number only.

disk, diskette: Magnetic media used to store information. Programs and data are recorded on the disk in the same way that music is recorded on a cassette tape. The term diskette refers to smaller capacity removable floppy diskettes. Disk can refer to a diskette, a removable cartridge disk, or a fixed hard disk.

disk drive: Peripheral device that reads and writes on hard or floppy disks. Personal CP/M assigns a letter to each drive under its control. For example, Personal CP/M can refer to the drives in a four-drive system as A, B, C, and D.

editor: Utility program that creates and modifies text files. An editor can be used for creation of documents or creation of code for computer programs. The Personal CP/M editor is invoked by typing the command ED next to the system prompt on the console.

executable: Ready to be run by the computer. Executable code is a series of instructions that can be carried out by the computer. For example, the computer cannot execute names and addresses, but it can execute a program that prints all those names and addresses on mailing labels.

execute a program: Start a program executing. When a program is running, the computer is executing a sequence of instructions.

FCB: See **File Control Block.**

file: Collection of characters, instructions or data stored on a disk. The user can create files on a disk.

File Control Block: Structure used for accessing files on disk. Contains the drive, filename, filetype and other information describing a file to be accessed or created on the disk.

filename: Name assigned to a file. A filename can include a primary filename of 1-8 characters and a filetype of 0-3 characters. A period separates the primary filename from the filetype.

file specification: Unique file identifier. A complete Personal CP/M file specification includes a disk drive specification followed by a colon (d:), a primary filename of 1 to 8 characters, a period, and a filetype of 0 to 3 characters. For example, b:example.tex is a complete Personal CP/M file specification.

filetype: Extension to a filename. A filetype can be from 0 to 3 characters and must be separated from the primary filename by a period. A filetype can tell something about the file. Certain programs require that files to be processed have certain filetypes.

floppy disk: Flexible magnetic disk used to store information. Floppy disks come in 5 1/4- and 8-inch diameters.

hard disk: Rigid, platter-like, magnetic disk sealed in a container. A hard disk stores more information than a floppy disk.

hardware: Physical components of a computer.

hex file: ASCII-printable representation of a command (machine language) file.

hexadecimal notation: Notation for the base 16 number system using the symbols 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F to represent the sixteen digits. Machine code is often converted to hexadecimal notation because it can be easily represented by ASCII characters and therefore printed on the console screen or on paper (see Appendix B).

input: Data going into the system, usually from an operator typing at the terminal or by a program reading from the disk.

interface: Object that allows two independent systems to communicate with each other, as an interface between hardware and software in a microcomputer.

I/O: Abbreviation for input/output.

keyword: See **command keyword.**

kilobyte: 1024 bytes denoted as 1K. 32 kilobytes equal 32K. 1024 kilobytes equal one megabyte, or over one million bytes.

list device: Device such as a printer onto which data can be listed or printed.

logical: Representation of something that might or might not be the same in its actual physical form. For example, a hard disk can occupy one physical drive, and yet you can divide the available storage on it to appear to the user as if it were in several different drives. These apparent drives are the logical drives.

megabyte: Over one million bytes; 1024 kilobytes. See **byte** and **kilobyte.**

microprocessor: Silicon chip that is the Central Processing Unit (CPU) of the microcomputer.

operating system: Collection of programs that supervises the running of other programs and the management of computer resources. An operating system provides an orderly input/output environment between the computer and its peripheral devices.

option: One of many parameters that can be part of a command tail. Use options to specifiy additional conditions for a command's execution.

output: Data that the system sends to the console or disk.

parameter: Value in the command tail that provides additional information for the command. Technically, a parameter is a required element of a command.

peripheral devices: Devices external to the CPU. For example, terminals, printers, and disk drives are common peripheral devices that are not part of the processor, but are used in conjunction with it.

physical: Actual hardware of a computer. The physical environment varies from computer to computer.

primary filename: First 8 characters of a filename. The primary filename is a unique name that helps the user identify the file contents. A primary filename contains 1 to 8 characters and can include any letter or number and some special characters. The primary filename follows the optional drive specification and precedes the optional filetype.

program: Series of specially coded instructions that performs specific tasks when executed by a computer.

prompt: Characters displayed on the screen to help the user decide what the next appropriate action is. A system prompt is a special prompt displayed by the operating system. The system prompt indicates to the user that the operating system is ready to accept input. The Personal CP/M system prompt is an alphabetic character followed by an angle bracket. The alphabetic character indicates the default drive. Some applications programs have their own special system prompts.

Read/Only: Attribute that can be assigned to a disk file or a disk drive. When assigned to a file, the Read/Only attribute allows you to read from that file but not change it. When assigned to a drive, the Read/Only attribute allows you to read any file on the disk, but prevents you from adding a new file, erasing or changing a file, renaming a file, or writing on the disk. The SET command can set a file or a drive to Read/Only. Every file and drive is either Read/Only or Read/Write. The default setting for drives and files is Read/Write, but an error in resetting the disk or changing media automatically sets the drive to Read/Only until the error is corrected. Files and disk drives can be set to either Read/Only or Read/Write.

Read/Write: Attribute that can be assigned to a disk file or a disk drive. The Read/Write attribute allows you to read from and write to a specific Read/Write file or to a any file on a disk that is in a drive set to Read/Write. A file or drive can be set to either Read/Only or Read/Write.

record: Collection of data. A file consists of one or more records stored on disk. An Personal CP/M record is 128 bytes long.

RO: See **Read/Only.**

RW: See **Read/Write.**

sector: Portion of a disk track. There are a specified number of sectors on each track.

software: Specially coded programs that transmit machine-readable instructions to the computer, as opposed to hardware, which is the actual physical components of a computer.

source file: ASCII text file that is an input file for a processing program, such as an editor, text formatter, or assembler.

string: See **character string.**

syntax: Format for entering a given command.

system attribute: File attribute. You can give a file the system attribute by using the SYS option in the SET command. A file with the SYS attribute is not displayed in response to a DIR command; you must use DIRS. If you give a file with user number 0 the SYS attribute, you can read and execute that file from any user number on the same drive. Use this feature to make your commonly used programs available under any user number.

system prompt: Symbol displayed by the operating system indicating that the system is ready to receive input. See **prompt.**

terminal: See **console.**

track: Concentric rings dividing a disk. There are 77 tracks on a typical eight-inch floppy disk.

turn-key application: Application designed for the noncomputer-oriented user. For example, a typical turn-key application is designed so that the operator needs only to turn on the computer, insert the proper program disk, and select the desired procedure from a selection of functions (menu) displayed on the screen.

upward compatible: Term meaning that a program created for the previously released operating system (or compiler, etc.) runs under the newly released version of the same operating system.

user number: Number from 0 to 15 assigned to a file when it is created. User numbers can organize files into sixteen file groups.

Utility Tool: Program that enables the user to perform certain operations, such as copying files, erasing files, and editing files. Utilities are created for the convenience of programmers and users.

wildcard characters: Special characters that give Personal CP/M a pattern to match when it searches the directory for a file. Personal CP/M recognizes two wildcard characters, ? and *. The ? can be substituted for any single character in a filespec, and the * can be substituted for the primary filename or the filetype or both. By placing wildcard characters in a filespec, you create an ambiguous filespec and can quickly reference one or more files.

End of Glossary

Personal CP/M™

8-Bit Operating System

Programmer's Guide

Foreword

Personal CP/M™ is a microcomputer operating system designed for the Zilog® Z80® or any compatible microprocessor. To run Personal CP/M, your computer must have an ASCII console, which includes a keyboard, video display screen, or another display device, 1 to 16 disk drives, and a munimum or 32K of Random Access Memory (RAM).

This manual describes the Basic Disk Operating System (BDOS) functions of Personal CP/M, and how to call the functions using Zilog Z80 assembler language. It is written for experienced programmers who are writing application software in the Personal CP/M environment. It assumes you are familiar with the system features and utilities described in the Personal CP/M Operating System User's Guide (cited as Personal CP/M User's Guide), the Personal CP/M Operating System System Guide (cited as Personal CP/M System Guide), and the Programmer's Utilities Guide for the CP/M Family of Operating Systems (cited as Programmer's Utilities Guide).

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

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

Section 3 presents five example programs.

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

This manual displays computer output in second color and user output in boldface second color.

Table of Contents

Table of Contents
(continued)

Appendixes

Tables, Figures, and Listing

Tables

Tables, Figures, and Listing (continued)

Figures

Listing

Section 1
Introduction to Personal CP/M

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

1.1 Components of Personal CP/M

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

1.2 Personal CP/M Memory Organization

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

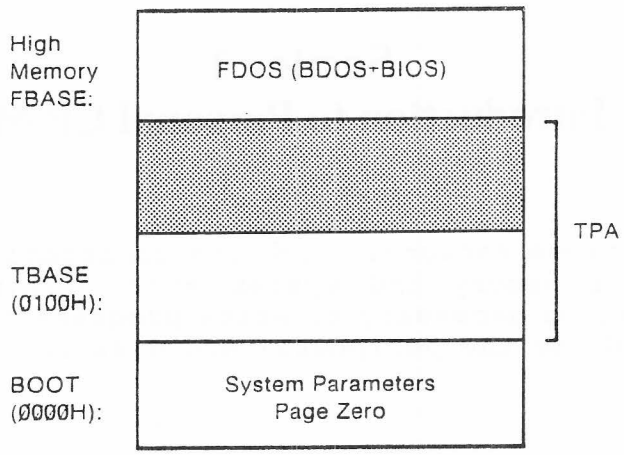


Figure 1-1. Personal CP/M Memory Organization

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

1.3 Program Execution

Transient programs are loaded into the TPA and executed through the CCP by typing command lines following each prompt. The CCP is capable of parsing the following general form of the command line:

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

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

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

```
command.COM
```

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

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

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

1.4 Calling a Function

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

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

SCREEN FUNCTIONS) are present in Personal CP/M to improve console I/O performance. Function 112 (LIST BLOCK) is included to improve printer I/O performance.

End of Section 1

Section 2
Operating System Call Conventions

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

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

The simple device operations include the following:

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

The following FDOS operations perform disk I/O:

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

The high performance video operations are as follows:

- direct screen functions

2.1 FDOS Operation

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

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

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

Listing 2-1. Assembly Language Program Segment

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

2.2 File Structure

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

Table 2-1. Personal CP/M Filetypes

Filetype	Meaning
ASM	assembler source
PRN	printer listing
HEX	Hex machine code
BAS	basic source file
INT	intermediate code
COM	command file
PLI	PL/I source file
REL	relocatable module
TEX	TEX Formatter source
BAK	ED source backup
SYM	SID™ symbol file
$$$	temporary file

2.2.1 File Records

Files in Personal CP/M can be thought of as a sequence of up to 65,536 logical sectors of 128 bytes each, numbered from 0 through 65,535, thus allowing a maximum of eight megabytes for each file. Note, however, that although the logical sectors may be considered logically contiguous, they may not be physically contiguous in the disk data area. Internally, all files are divided into 16K-byte segments called logical extents, so that counters are easily maintained as eight-bit values. The division into extents is discussed in Section 2.2.2, "File Control Block"; however, they are not particularly significant for the programmer because each extent is automatically accessed in both sequential and random access modes.

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

2.2.2 File Control Block

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

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

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

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

Figure 2-1. File Control Block Format

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

Table 2-2. File Control Block Fields

Field	Definition
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 filename in ASCII uppercase, with high bit = 0
t1, t2, t3	contain the filetype in ASCII uppercase, with high bit = 0. t1', t2', and t3' denote the bit of these positions. t1' = 1 => Read/Only file, t2' = 1 => SYS file, no DIR list
ex	contains the current extent number, normally set to 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 0-127
d0...dn	filled in by Personal CP/M; reserved for system use
cr	current record to read or write in a sequential file operation; normally set to zero by user
r0, r1, r2	optional random record number in the range 0-65535 (0-FFFF); r0, r1, r2 constitute an 16-bit value with low byte r0, high byte r1, and byte r2 = 0.

Each file being accessed through Personal CP/M must have a corresponding FCB, which provides the name and allocation information for all subsequent file operations. Bytes 1 through 11 are set by the CCP to the ASCII character values for the filename

and filetype. Byte 0 is set to the drive code. All other fields are set to zero. When constructing your own FCB, it is your responsibility to fill the lower 12 bytes of the FCB and initialize the cr field to zero.

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

The CCP constructs the first 12 bytes of two optional FCBs for a transient command by scanning the remainder of the command line following the transient name, denoted by file1 and file2 in the prototype command line described in Section 1.3, "Program Execution," with unspecified fields set to ASCII blanks. If no filenames are specified in the original command, the fields beginning at BOOT+005DH and BOOT+006DH contain blanks. In all cases, the CCP translates lowercase letters to uppercase to be consistent with the Personal CP/M file-naming conventions. The first FCB is constructed at location BOOT+005CH and can be used as is for subsequent file operations. The second FCB occupies the d0...dn portion of the first FCB and must be moved to another area of memory before use. For example, assume the command line shown below is typed; then, the CCP loads the file PROGNAME.COM into the TPA and initializes the default FCB at BOOT+005CH to drive code 2, filename X, and filetype ZOT:

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

The drive code for file 2 takes the default value 0, which the CCP places at BOOT+006CH, with the filename Y placed into location BOOT+006DH and filetype ZAP located eight bytes later at BOOT+0075H. The CCP sets all remaining fields through cr to zero. Note again that it is your responsibility to move this second filename and filetype to another area, usually a separate file control block that you create, before opening the file that begins at BOOT+005CH, because the open operation overwrites the second name and type.

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

```
BOOT+0080H:

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

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

2.3 BDOS Function Calls

Individual functions are described in detail in the following pages.

FUNCTION 0: SYSTEM RESET
Entry Parameters: Register C: 00H Returned Value: none

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

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

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

FUNCTION 2: CONSOLE OUTPUT	
Entry Parameters:	
Register C:	02H
Register E:	ASCII character
Returned Value:	none

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

FUNCTION 3: AUXILIARY INPUT
Entry Parameters: Register C: 03H Returned Value: Register A: ASCII character

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

FUNCTION 4: AUXILIARY OUTPUT
Entry Parameters: Register C: 04H Register E: ASCII character Returned Value: none

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

FUNCTION 5: LIST OUTPUT
Entry Parameters: Register C: 05H Register E: ASCII character Returned Value: none

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

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

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

Table 2-3. Function 6 Entry Parameters

Register E Value	Meaning
0FFH	Console input/status command returns an input character; if no character is ready, a value of zero is returned.
0FEH	Console status command (on return, register A contains 00H if no character is ready; otherwise, it contains FFH.)
ASCII character	Function 6 assumes register E contains a valid ASCII character and sends it to the console.

FUNCTION 7: AUXILIARY INPUT STATUS
Entry Parameters: Register C: 07H Returned Value: Register A: Auxiliary Input Status 00H = no character for input 0FFH = character ready for input

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

FUNCTION 8: AUXILIARY OUTPUT STATUS
Entry Parameters: Register C: 08H Returned Value: Register A: Auxiliary Output Status 00H = device not ready for output 0FFH = device ready

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

FUNCTION 9: PRINT STRING
Entry Parameters: Register C: 09H Registers DE: String address Returned Value: none

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

FUNCTION 10: READ CONSOLE BUFFER
Entry Parameters: Register C: 0AH Registers DE: Buffer address Returned Value: Console characters in buffer

The READ CONSOLE BUFFER function reads a line of edited console input into a buffer addressed by registers DE. Console input is terminated when either input buffer overflows or a carriage return or line feed is typed. Function 10 takes the following form, where mx is the maximum number of characters that the buffer will hold, 1 to 255, and nc is the number of characters read (set by FDOS upon return) followed by the characters read from the console.

```
DE:+0 +1 +2 +3 +4 +5 +6 +7 +8 . . .+n

   mx nc c1 c2 c3 c4 c5 c6 c7  ... ??
```

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

Table 2-4. Edit Control Characters

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

FUNCTION 11: GET CONSOLE STATUS	
Entry Parameters:	
Register C:	0BH
Returned Value:	
Register A:	Console status 00H = no character ready 0FFH = character ready

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

FUNCTION 12: RETURN VERSION NUMBER
Entry Parameters: Register C: 0CH Returned Value: Registers HL: Version number 0028H

The RETURN VERSION NUMBER function provides information that allows version independent programming. A two-byte value is returned, with H = 00 designating the CP/M release. Return version numbers 20H through 27H are designated for all previous CP/M Release 2 versions; 0028H is designated for Personal CP/M Release 1.0. Personal CP/M returns a hexadecimal 28 in register L. Function 12 is useful for writing application programs that must run on multiple CP/M products.

FUNCTION 13: RESET DISK SYSTEM
Entry Parameters: Register C: 0DH Returned Value: none

The RESET DISK SYSTEM function is used to restore the file system to a reset state where all disks are set to Read/Write (see Functions 28 and 29). 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 that requires a disk change without a system reboot.

FUNCTION 14: SELECT DISK
Entry Parameters: Register C: 0EH Register E: Selected disk Returned Value: A: Error flag 00H if successful, or 0FFH if failed H: Physical error

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 on through 15, corresponding to drive P in a full 16-drive system. The drive is placed in an on-line status, which activates its directory until the next cold start, warm start, or RESET DISK SYSTEM operation. FCBs that specify drive code zero (dr = 00H) automatically reference the currently selected default drive. Drive code values between 1 and 16 ignore the selected default drive and directly reference drives A through P.

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

01 : Disk I/O error
04 : Invalid drive

FUNCTION 15: OPEN FILE	
Entry Parameters:	
Register C:	0FH
Registers DE:	FCB address
Returned Value:	
Register A:	00H if successful, or 0FFH if failed
H:	00H if successful, or Physical error (see below)

The OPEN FILE operation is used to activate a file that currently exists in the disk directory for the currently active user number. The FDOS scans the referenced disk directory for a match in positions 1 through 14 of the FCB referenced by DE (byte s2 is automatically zeroed). Normally, bytes ex and s1 of the FCB are zero.

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

Function 15 opens a file under user 0 when the current user number is nonzero, if two conditions exist:

- the file is not present under the current user number, and
- the file under user 0 has the system attribute T2' set

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

Upon return, the OPEN FILE function returns a 00H in register A if the open was successful, or 0FFH, 255 decimal, if the file was not found. Register H is set to zero in both of these cases. If a physical error was encountered the Function 15 performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is in the default mode, a message identifying the

error is displayed at the console (see Appendix B), and the program is terminated. Otherwise, the OPEN FILE function returns to the calling program with register A set to 0FFH, and register H set to one of the following physical error codes:

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

FUNCTION 16: CLOSE FILE	
Entry Parameters:	
Register C:	10H
Registers DE:	FCB address
Returned Value:	
Register A:	00H if successful, or 0FFH if failed
H:	00H if successful, or Physical error (see below)

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 FILE or MAKE FILE function, the CLOSE FILE function permanently records the new FCB in the reference disk directory (see Functions 15 and 22). The FCB matching process for the CLOSE FILE is identical to the OPEN FILE function. The directory code returned for a successful CLOSE FILE operation is 0, while a 0FFH (255 decimal) is returned if the filename cannot be found in the directory. If write operations have occurred, the CLOSE FILE operation is necessary to record the new directory information permanently.

Upon return, the CLOSE FILE function returns a 00H in register A if the close was successful, or FFH, 255 decimal, if the file was not found. Register H is set to zero in both cases. If a physical error is encountered, the CLOSE FILE function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is in the default mode, a message identifying the error is displayed at the console, and the calling program is terminated. Otherwise, the CLOSE FILE function returns to the calling program with register A set to 0FFH and register H set to one of the following physical error codes:

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

FUNCTION 17: SEARCH FOR FIRST	
Entry Parameters:	
Register C:	11H
Registers DE:	FCB address
Returned Value:	
Register A:	Directory code 00 - 03 if successful, or 0FFH if failed H: 00H if successful, or Physical error (see below)

SEARCH FOR FIRST scans the directory for a match with the file given by the FCB addressed by DE. The value 255 (hexadecimal FF) is returned if the file is not found; otherwise, 0, 1, 2, or 3 is returned indicating the file is present. When the file is found, the current DMA address is filled with the record containing the directory entry, and the relative starting position is A *32 (that is, rotate the A register left five bits, or ADD A five times). Although not normally required for application programs, the directory information can be extracted from the buffer at this position. Byte 0 of a returned directory entry contains the file's user number.

An ASCII question mark (63 decimal, 3F hexadecimal) in any position from f1 through ex matches the corresponding field of any directory entry on the default or auto-selected disk drive. If the dr field contains an ASCII question mark, the auto disk select function is disabled and the default disk is searched, with the SEARCH FOR FIRST function returning any matched entry, allocated or free, belonging to any user number. This latter function is not normally used by application programs, but it allows complete flexibility to scan all current directory values. If the dr field is not a question mark, the s2 byte is automatically zeroed.

If a physical error is encountered, the SEARCH FOR FIRST function performs actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is in the default mode, a message identifying the error is displayed at the console, and the calling program is terminated. Otherwise, Function 17 returns to the calling program with register A set to 0FFH, and register H set to one of the following physical error codes:

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

FUNCTION 18: SEARCH FOR NEXT	
Entry Parameters:	
Register C:	12H
Returned Value:	
Register A:	Directory code 00 - 03 if successful, or 0FFH if failed
H:	00H if successful or Physical error (see Function 17)

The SEARCH FOR NEXT function is similar to the SEARCH FOR FIRST function, except that the directory scan continues from the last matched entry. Similar to Function 17, Function 18 returns the 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:	00H if successful, or 0FFH if failed
H:	00H if successful, or Physical error (see below)

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

Upon return, the DELETE FILE function returns a 00H in register A if successful, or 0FFH, 255 decimal, if no file that matches the referenced FCB is found. Register H is set to zero in both cases. If a physical error is encountered, Function 19 performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is the default mode, a message identifying the error is displayed at the console and the calling program is terminated. Otherwise, the DELETE FILE function returns to the calling program with register A set to 0FFH and register H set to one of the following physical error codes:

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

FUNCTION 20: READ SEQUENTIAL	
Entry Parameters:	
Register C:	14H
Registers DE:	FCB address
Returned Value:	
Register A:	Error code 00H if successful, or 01, 10, or 255 if failed
H:	00H if successful, or Physical error if failed

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

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

01 : Reading unwritten data (end-of-file)
10 : Media change occurred
255: Physical error, refer to register H

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

Error Code 10 is returned if a media change occurs on the drive after the referenced FCB is activated by a BDOS OPEN FILE or MAKE CALL.

Error Code 255 is returned if a physical error is encountered and the BDOS error mode is return error mode, or return and display error mode (see Function 45). If the error mode is the default mode, a message identifying the physical error is displayed at the console, and the calling program is terminated. When a physical error is returned to the calling program, register H contains one of the following error codes:

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

FUNCTION 21: WRITE SEQUENTIAL	
Entry Parameters:	
Register C:	15H
Registers DE:	FCB address
Returned Value:	
Register A:	Error code 00H if successful, or 01, 02, 10, or 255 if failed
H:	00H if successful, or Physical error if failed

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

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

01 : No available directory space
02 : No available data block
10 : Media change occurred
255: Physical error, refer to register H

Error Code 01 is returned when Function 21 attempts to create a new extent that requires a new directory entry, and no available directory entries exist on the selected disk drive.

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

Error Code 10 is returned if a media change occurs on the drive after the referenced FCB is activated by a BDOS OPEN FILE or MAKE FILE call.

Error Code 255 is returned if a physical error is encountered and the BDOS error mode is return error mode, or return and display error mode (see Function 45). If the error mode is the default mode, a message identifying the physical error is displayed at the console, and the calling program is terminated. When a physical error is returned to the calling program, register H contains one of the following error codes:

```
01 : Disk I/O error
02 : Read/Only disk
03 : Read/Only file or
     File open from user 0 when the
     current user number is nonzero
04 : Invalid drive error
```

FUNCTION 22: MAKE FILE	
Entry Parameters:	
Register C:	16H
Registers DE:	FCB address
Returned Value:	
Register A:	00H if successful, or 0FFH if failed
H:	00H if successful, or Physical error (see below)

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

Upon return, Function 22 returns a 00H in register A the operation is successful, or 0FFH, 255 decimal, if no directory space is available. Register H is set to zero in both of these cases. If a physical error is encountered, the MAKE FILE function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is the default mode, a message identifying the error is displayed at the console, and the calling program is terminated. Otherwise, the MAKE FILE function returns to the calling program with register A set to 0FFH, and register H set to one of the following physical error codes:

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

FUNCTION 23: RENAME FILE	
Entry Parameters:	
Register C:	17H
Registers DE:	FCB address
Returned Value:	
Register A:	00H if successful, or 0FFH if failed
H:	00H if successful, or Physical error (see below)

The RENAME FILE function uses the FCB addressed by DE to change 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 filename at position 16 of the FCB is assumed to be zero.

Upon return, the RENAME FILE function returns a 00H in register A if the operation is successful, or 0FFH, 255 decimal, if the file named by the first filename in the FCB is not found. Register H is set to zero in both cases. If a physical error is encountered, the RENAME FILE function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is the default mode, a message identifying the error is displayed at the console and the program is terminated. Otherwise, Function 23 returns to the calling program with register A set to 0FFH and register H set to one of the following physical error codes:

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

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

The RETURN LOGIN VECTOR value returned by Personal 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 16th drive, labeled P. A 0 bit indicates that the drive is not on-line, while a 1 bit marks a drive that is actively on-line as a result of an explicit disk drive selection or an implicit drive select caused by a file operation that specified a nonzero dr field. The user should note that compatibility is maintained with previous CP/M products, because registers A and L contain the same values upon return.

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

The RETURN CURRENT DISK function returns the currently selected default disk number in register A. The disk numbers range from 0 through 15, corresponding to drives A through P.

FUNCTION 26: SET DMA ADDRESS
Entry Parameters: Register C: 1AH Registers DE: DMA address Returned Value: none

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

FUNCTION 27: GET ADDR (ALLOC)	
Entry Parameters:	
Register C:	1BH
Returned Value:	
Registers HL:	ALLOC address if successful, or 0FFFFH if failed

An allocation vector is maintained in main memory for each on-line disk drive. Various system programs use the information provided by the allocation vector to determine the amount of remaining storage (see the STAT program). The GET ADDR (Alloc) function returns the base address of the allocation vector for the currently selected disk drive. However, the allocation information might 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 Personal CP/M System Guide.

If a physical error is encountered when the BDOS error mode is one of the return modes (see Function 45), Function 27 returns the value 0FFFFH in the register pair HL.

FUNCTION 28: WRITE PROTECT DISK
Entry Parameters: Register C: 1CH Returned Value: none

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

CP/M error on d: Read/Only Disk

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

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

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

FUNCTION 30: SET FILE ATTRIBUTES	
Entry Parameters:	
Register C:	1EH
Registers DE:	FCB address
Returned Value:	
Register A:	00H if successful, or 0FFH if failed
H:	00H if successful, or Physical error (see below)

The SET FILE ATTRIBUTES function allows the permanent indicators attached to files to be modified by a program. In particular, the RO and system attributes (t1' and t2') can be set or reset. The DE pair addresses an unambiguous filename with the appropriate attributes set or reset. Function 30 searches for a match and changes the matched directory entry to contain the selected indicators. Indicators f1' through f8' are not currently used in Personal CP/M. Indicators f1' through f4' may be used by application programs since they are not involved in the matching process during OPEN FILE and CLOSE FILE operations. Indicators f5' through f8' and t3' are reserved.

Upon return, Function 30 returns a 00H in register A if the function is successful, or 0FFH, 255 decimal, if the file specified by the referenced FCB is not found. Register H is set to zero in both of these cases. If a physical error is encountered, the SET FILE ATTRIBUTES function performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is the default mode, a message identifying the error is displayed at the console, and the program is terminated. Otherwise, Function 30 returns to the calling program with register A set to 0FFH, and register H set to one of the following physical error codes:

01 : Disk I/O error
02 : Read/Only disk
04 : Select error

FUNCTION 31: GET ADDR (DISK PARMS)	
Entry Parameters:	
Register C:	1FH
Returned Value:	
Registers HL:	DPB address if successful, or 0FFFFH if failed

The GET ADDR (DISK PARMS) function returns the address of the BIOS resident disk parameter block in HL. 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 do not require this facility.

If a physical error is encountered when the BDOS error mode is one of the return modes (see Function 45), Function 31 returns the value 0FFFFH in register pair HL.

FUNCTION 32: SET/GET USER CODE
Entry Parameters: Register C: 20H Register E: OFFH (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, the value of the current user number is returned in register A, where the value is in the range of 0 to 15. If register E is not 0FFH, the current user number is changed to the value of E, modulo 16.

FUNCTION 33: READ RANDOM	
Entry Parameters:	
Register C:	21H
Returned Value:	
Register A:	Error code 00H if successful, or nonzero if failed (see below)
H:	00H if successful, or Physical error (see below)

The READ RANDOM function is similar to the READ SEQUENTIAL operation of previous CP/M products, except that the read operation takes place at a particular record number, selected by the 24-bit value constructed from the 3-byte field following the FCB (byte positions r0 at 33, r1 at 34, and r2 at 35). You should note that the sequence of 24 bits is stored with least significant byte first (r0), middle byte next (r1), and high byte last (r2). Personal CP/M does not reference byte r2, except in computing the size of a file (Function 35). Byte r2 must be zero, however, since a nonzero value indicates overflow past the end of file.

Thus, the r0, r1 byte pair is treated as a double byte, or word value, that contains the record to read. This value ranges from 0 to 65,535, providing access to any particular record of the eight-megabyte file. To process a file using random access, the base extent (extent 0) must first be opened. Although the base extent may or may not contain any allocated data, this step ensures that the file is properly recorded in the directory and is visible in DIR requests. The selected record number is then stored in the random record field (r0, r1), and the BDOS is called to read the record.

Upon return from the call, register A either contains an error code, as listed below, or the value 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 READ RANDOM operations continue to read the same record.

Upon each READ RANDOM operation, the logical extent and current record values are automatically set. Thus, the file can be sequentially read or written, starting from the current randomly accessed position. In this case, the last randomly read record will be reread as one switches from random mode to SEQUENTIAL READ and

the last record will be rewritten as one switches to a SEQUENTIAL WRITE operation. The user can advance the random record position following each READ RANDOM or WRITE RANDOM to obtain the effect of sequential I/O operation.

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

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

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

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

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

Error Code 06 is returned when byte 35, r2, of the referenced FCB is nonzero.

Error Code 10 is returned if a media change occurs on the drive after the referenced FCB is activated by a BDOS OPEN FILE or MAKE FILE call.

Error Code 255 is returned if a physical error is encountered, and the BDOS error mode is one of the return modes (see Function 45). If the error mode is the default mode, a message identifying the physical error is displayed at the console, and the calling program is terminated. When a physical error is returned to the calling program, register H contains one of the following error codes:

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

FUNCTION 34: WRITE RANDOM	
Entry Parameters:	
Register C:	22H
Registers DE:	FCB address
Returned Value:	
Register A:	Error code 00H if successful, or nonzero if failed (see below)
H:	00H if successful, or Physical error (see below)

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

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

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

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

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

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

Error Code 06 is returned when byte 35, r2, of the referenced FCB is nonzero.

Error Code 10 is returned if a media change occurs on the drive after the referenced FCB is activated by a BDOS OPEN FILE or MAKE FILE call.

Error Code 255 is returned if a physical error is encountered and the BDOS error mode is one of the return modes (see Function 45). If the error mode is the default mode, a message identifying the physical error is displayed at the console, and the calling program is terminated. When a physical error is returned to the calling program, it is identified by register H as shown below:

```
01 : Disk I/O error
02 : Read/Only disk
03 : Read/Only file or
     File open from user 0 when the
     current user number is nonzero
04 : Invalid drive error
```

FUNCTION 35: COMPUTE FILE SIZE	
Entry Parameters:	
Register C:	23H
Registers DE:	FCB Address
Returned Value:	
Register A:	00H if successful, or 0FFH if failed
H:	00H if successful, or Physical error (see below)

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 filename is used in the directory scan. Upon return, the random record bytes contain the virtual file size, which is, in effect, the record address of the record following the end of the file. Following a call to Function 35, if the high record byte r2 is 01, the file contains the maximum record count 65,536. Otherwise, bytes r0 and r1 constitute a 16-bit value as before (r0 is the least significant byte), which is the file size.

Data can be appended to the end of an existing file by calling Function 35 to set the random record position to the end of file and then performing a sequence of WRITE RANDOM operations starting at the preset record address.

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

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

Upon return, Function 35 returns a zero in register A if the file specified by the referenced FCB is found, or an 0FFH in register A if the file is not found. Register H is set to zero in both of these cases. If a physical error is encountered, Function 35 performs different actions depending on the BDOS error mode (see Function 45). If the BDOS error mode is the default mode, a message identifying the error is displayed at the console and the program is terminated. Otherwise, Function 35 returns to the calling program with register A set to 0FFH, and register H set to one of the following physical errors:

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

FUNCTION 36: SET RANDOM RECORD
Entry Parameters: Register C: 24H Registers DE: FCB address Returned Value: Random record field set

The SET RANDOM RECORD function returns the random record number of the next record to be accessed from a file that has been read or written sequentially to a particular point. This value is returned in the random record field, bytes r0, r1, and r2 of the FCB addressed by the register pair DE. Function 36 can be useful in two ways.

First, it is often necessary initially to 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 number minus one is placed into a table with the key for later retrieval. After scanning the entire file and tabularizing the keys and their record numbers, you can move directly to a particular record by performing a RANDOM READ using the corresponding random record number that you saved earlier. The scheme is easily generalized when variable record lengths are involved because the program need only store the buffer-relative byte position along with the key and record number to find the exact starting position of the keyed data at a later time.

A second use of Function 36 occurs when switching from a SEQUENTIAL READ or SEQUENTIAL WRITE over to RANDOM READ or RANDOM WRITE. A file is sequentially accessed to a particular point in the file; then Function 36 is called to set the record number, and subsequent RANDOM READ and RANDOM WRITE operations continue from the next record in the file.

FUNCTION 37: RESET DRIVE
Entry Parameters: Register C: 25H Registers DE: Drive vector Returned Value: none

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

FUNCTION 40: WRITE RANDOM WITH ZERO FILL	
Entry Parameters:	
Register C:	28H
Registers DE:	FCB address
Returned Value:	
Register A:	Error code 00H if successful, or nonzero if failed (see Function 34)
H:	00H if successful, or Physical error (see Function 34)

The WRITE RANDOM WITH ZERO FILL operation is similar to Function 34, with the exception that a previously unallocated block is filled with zeros before the data is written.

FUNCTION 45: SET BDOS ERROR MODE
Entry Parameters: Register C: 2DH E: BDOS error mode 0FFH - Return error 0FEH - Return and display Any other - Default mode Returned Value: none

Function 45 sets the BDOS error mode for the calling program to the mode specified in register E. If register E is set to 0FFH, 255 decimal, the error mode is set to return error mode. If register E is set to 0FEH, 254 decimal, the error mode is set to return and display mode. If register E is set to any other value, the error mode is set to the default mode.

The SET BDOS ERROR MODE function determines how physical errors are handled for a program. The operation can exist in three modes: the default mode, return error mode, and return and display error mode. In the default mode, the BDOS displays a system message at the console that identifies the error and terminates the calling program. In the return modes, the BDOS sets register A to 0FFH, 255 decimal, places an error code that identifies the physical error in register H, and returns to the calling program. In return and display mode, the BDOS displays the system message before returning to the calling program. No system messages are displayed, however, when the BDOS is in return error mode.

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

Table 2-5. Messages for Physical Errors Returned

Physical Error Codes Returned	Corresponding Personal CP/M Message
1	Disk I/O error
2	Read/Only disk
3	Read/Only file
4	Invalid drive error

FUNCTION 48: FLUSH BUFFERS	
Entry Parameters:	
Register C:	30H
Returned Value:	
Register A:	00H if successful, or Error flag if failed
Register H:	Physical error

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

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

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

FUNCTION 109: GET/SET CONSOLE MODE	
Entry Parameters:	
Register C:	6DH
Register DE:	0FFFFH (Get)or Console mode (Set)
Returned Value:	
Register HL:	Console mode or (no value)

A program can set or interrogate the console mode by calling Function 109. If register pair DE = 0FFFFH, the current console mode is returned in register HL. Otherwise, Function 109 sets the console mode to the value contained in register pair DE.

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

Table 2-6. GET/SET CONSOLE MODE Description

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

Note that the GET/SET CONSOLE MODE bits are numbered from right to left. The CCP initializes the GET/SET CONSOLE MODE to zero when it loads a program.

FUNCTION 110: GET/SET OUTPUT DELIMITER	
Entry Parameters:	
Register C:	6EH
Register DE:	0FFFFH (Get) or
E:	Output delimiter (Set)
Returned Value:	
Register A:	Output delimiter or (no value)

A program can set or interrogate the current output delimiter by calling Function 110. If register pair DE = 0FFFFH, the current output delimiter is returned in register A. Otherwise, Function 110 sets the output delimiter to the value contained in register E. Function 110 sets the string delimiter for Function 9, PRINT STRING. The default delimiter value is a dollar sign, $. The CCP sets the GET/SET OUTPUT DELIMITER to the default value when a transient program is loaded.

FUNCTION 111: PRINT BLOCK
Entry Parameters: Register C: 6FH Register DE: CCB address Returned Value: none

The PRINT BLOCK function sends the character string located by the character control block, CCB, addressed in register pair DE to the logical console, CONOUT:. If the console mode is in the default state, Function 111 expands tab characters, CTRL-I, in columns of eight characters. It also checks for CTRL-S (stop/start scroll) and echoes to the logical list device, LST:, if printer echo, CTRL-P, has been invoked. The CCB format is as follows:

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

FUNCTION 112: LIST BLOCK
Entry Parameters: Register C: 70H Register DE: CCB address Returned Value: none

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

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

FUNCTION 113: DIRECT SCREEN FUNCTIONS
Entry Parameters: Register C: 71 H Register DE: Address of SFB byte 0 : Subfunction number byte 1-2: Address of extended information OR byte 1 : Column value byte 2 : Row value Returned Value: none

The DIRECT SCREEN FUNCTIONS call provides direct access to cursor movement and screen editing functions typically used by video intensive applications, such as word processing and spreadsheets. This direct access is important primarily on computer systems with memory-mapped displays. While most BIOS display drivers provide some terminal emulation for these functions, the overhead involved in interpreting an ESCAPE sequence into the corresponding screen function can slow a video intensive application to an unacceptable degree. The DIRECT SCREEN FUNCTIONS call not only allows direct access to these screen functions, but also can return information to the calling program about whether a specific function executes fast or slowly on a particular system (see Subfunction 1). Table 2-7 lists the subfunctions for Function 113.

Table 2-7. Subfunctions for Function 113

Subfunction	Description
0: SUBFUNCTIONS SUPPORTED	Returned value: Register HL: Pointer to bit vector stored as

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

Corresponding bit is on if subfunction is supported.

1: SUBFUNCTIONS EMULATED	Returned value: Register HL: Pointer to bit vector stored as above. Bit is on if subfunction is emulated (and therefore slower).
2: DISPLAY SIZE	Returned value: Register H: Number of columns (1-n). Register L: Number of rows (1-n).
3: IDENTIFY TERMINAL	Returned value: Register HL: Pointer to null terminated identifier string. For a VT-52 type terminal, it would return a pointer to the byte string (ESCAPE, /, K, NULL).

Table 2-7. (continued)

Subfunction	Description
4: CURSOR UP	Move cursor up but does not scroll screen down if cursor was at top of page.
5: CURSOR DOWN	Move cursor down but does not scroll screen up if cursor was at bottom of page.
6: CURSOR LEFT	Wrap depends on mode set up by subfunction 26 or 27.
7: CURSOR RIGHT	Wrap depends on mode set by subfunction 26 or 27.
8: CURSOR HOME	Move cursor to top left-hand corner of screen.
9: CURSOR ON	Make cursor visible.
10: CURSOR OFF	Make cursor invisible.
11: DIRECT CURSOR ADDRESSING	Move cursor to absolute column and row in SFB.

Table 2-7. (continued)

Subfunction	Description
12: CLEAR DISPLAY	Move cursor to top left-hand corner of screen and erase screen.
13: ERASE TO END OF LINE	
14: ERASE TO END OF SCREEN	
15: ENTER ANSI MODE	
16: ENTER VT-52 MODE	
17: ENTER GRAPHICS MODE	Not supported by the MZ-800 P-CP/M
18: EXIT GRAPHICS MODE	
19: ENTER ALTERNATE KEYPAD MODE	
20: EXIT ALTERNATE KEYPAD MODE	
21: ENTER HOLD SCREEN MODE	
22: EXIT HOLD SCREEN MODE	
23: ENTER REVERSE VIDEO MODE	
24: EXIT REVERSE VIDEO MODE	
25: REVERSE LINE-FEED	

Table 2-7. (continued)

Suofunction	Description
26:	ENABLE WRAP-AROUND AT END OF LINE
27:	TRUNCATE CHARACTERS AT END OF LINE
28-255:	(Reserved)

Section 3
Sample Programs

3.1 Sample File-to-File Copy Program

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

At this point, the source and destination FCBs are ready for processing, because the SFCB 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 creating the destination file. If all this is successful, the program loops at the label COPY until each 128-byte record is read from the source file and placed into the destination file. When the data transfer is complete, the destination file is closed and the program returns to the CCP command level by jumping to BOOT.

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

3.2 Sample File Dump Utility

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

3.3 Sample Random Access Program

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

A>**RANDOM X.DAT**

The program looks for a file named X.DAT 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 following form and is followed by operator input, followed by a carriage return.

next command?

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

nW nR Q

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

type data:

The operator then responds by typing up to 127 characters, followed by a carriage return. 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 CCP. For brevity, the only error message is

error, try again.

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

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

A>**GETKEY NAMES.DAT LASTNAME 10 20**

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

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

A>**QUERY NAMES.DAT LASTNAME.KEY**

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

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

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

3.4 Full Duplex Terminal Emulator

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

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

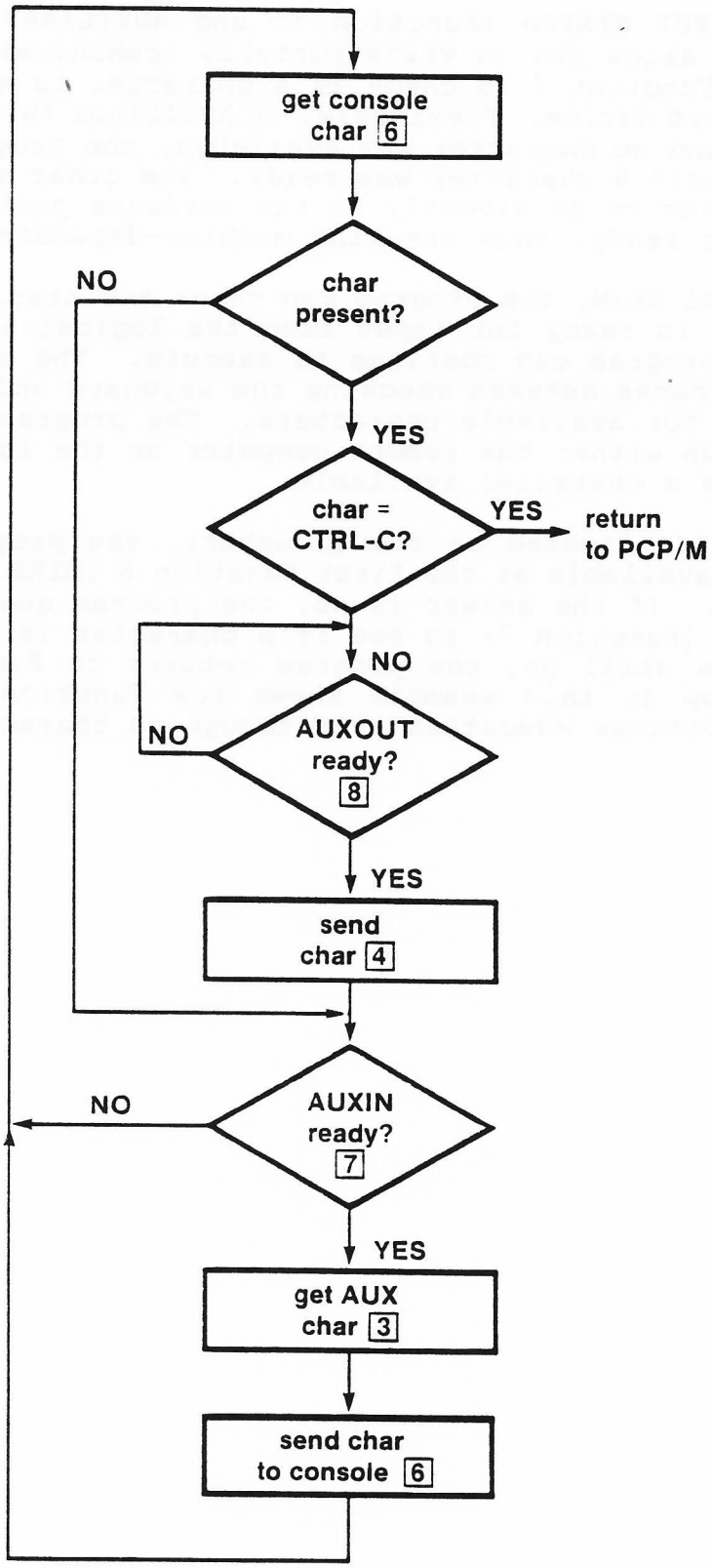


Figure 3-1. Full Duplex Terminal Emulator Flowchart

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

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

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

Appendix A
System Function Summary

Table A-1. System Function Summary

Dec	Hex	Name	Input Parameters	Returned Value
0	0	System Reset	none	none
1	1	Console Input	none	A = ASCII char
2	2	Console Output	E = ASCII char	none
3	3	Auxiliary Input	none	A = ASCII char
4	4	Auxiliary Output	E = ASCII char	none
5	5	List Output	E = ASCII char	none
6	6	Direct Console I/O	E = 0FFH/ 0FEH/ char	A = char/status/ no value
7	7	Auxiliary Input Status	none	A = AUXIN status
8	8	Auxiliary Output Status	none	A = AUXOUT status
9	9	Print String	DE = String Addr	none
10	A	Read Console Buffer	DE = Buffer Addr	Chars in buffer
11	B	Get Console Status	none	A = 00H/0FFH
12	C	Return Version Number	none	HL: Version (0028H)
13	D	Reset Disk System	none	none
14	E	Select Disk	E = Selected Disk	A = Err Flag/ 00H/0FFH H = Phys err
15	F	Open File	DE = FCB Addr	A = 00H/0FFH H = 00H/Phys err
16	10	Close File	DE = FCB Addr	A = 00H/0FFH H = 00H/Phys err
17	11	Search For First	DE = FCB Addr	A = Dir Code/00-03/0FFH H = 00H/Phys err
18	12	Search For Next	none	A = Dir Code/00-03/0FFH H = 00H/Phys err
19	13	Delete File	DE = FCB Addr	A = 00H/0FFH H = 00H/Phys err
20	14	Read Sequential	DE = FCB Addr	A = Err Code/00H/ 01, 10, or 255 H = 00H/Phys err
21	15	Write Sequential	DE = FCB Addr	A = Err Code/00H/ 01, 02, 10, or 255 H = 00H/Phys err

Table A-1. (continued)

Dec	Hex	Name	Input Parameters	Returned Value
22	16	Make File	DE = FCB Addr	A = 00H/0FFH H = 00H/Phys err
23	17	Rename File	DE = FCB Addr	A = 00H/0FFH H = 00H/Phys err
24	18	Return Login Vector	none	HL = Login Vector*
25	19	Return Current Disk	none	A = Current Disk
26	1A	Set DMA Address	DE = DMA Addr	none
27	1B	Get ADDR (ALLOC)	none	HL = Alloc Addr*/ 0FFFFH
28	1C	Write Protect Disk	none	none
29	1D	Get Read/Only Vector	none	HL = R/O Vector Value*
30	1E	Set File	DE = FCB Addr	A = 00H/0FFH H = 00H/Phys err
31	1F	Get ADDR (Disk Parms)	none	HL = DPB Add/ 0FFFFH
32	20	Set/Get User Code	E = 0FFH for Get E = User Code for Set	A = Curr Code/no value
33	21	Read Random	DE = FCB Addr	A = Err Code/00H/ nonzero H = 00H/Phys err
34	22	Write Random	DE = FCB Addr	A = Err Code/00H/ nonzero H = 00H/Phys err
35	23	Compute File Size	DE = FCB Addr	A = 00H/0FFH H = 00H/Phys err
36	24	Set Random Record	DE = FCB Addr	Random Record Field Set
37	25	Reset Drive	DE = Drive Vector	none
40	28	Write Random with Zero Fill	DE = FCB	A = Err Code/00H/ nonzero H = 00H/Phys err
45	2D	Set BDOS Error Mode	E = BDOS Err Mode	none
48	30	Flusn Buffers	none	A = Err Flag/00H H = Phys err
109	6D	Get/Set Console Mode	DE = 0FFFFH/Mode	HL = Console Mode/no value
110	6E	Get/Set Output Delimiter	DE = 0FFFFH/ E = Delimiter	A = Output Delimiter/ no value
111	6F	Print Block	DE = CCB Addr	none
112	70	List Block	DE = CCB Addr	none

Table A-1. (continued)

Dec	Hex	Name	Input Parameters	Returned Value
113	71	Direct Screen Functions	DE = SFB Addr	none
124	7C	Byte BLT Copy	DE = BCB Addr	A = 00H/0FFH
125	7D	Byte BLT Alter	DE = BCB Addr	A = 00H/0FFH

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

End of Appendix A

Appendix B
BDOS Error Handling

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

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

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

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

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

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

CP/M Error on d: error message

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

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

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

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

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

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

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

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

20. READ SEQUENTIAL
21. WRITE SEQUENTIAL
33. READ RANDOM
34. WRITE RANDOM
40. WRITE RANDOM WITH ZERO FILL

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

Table B-1. Register A BDOS Error Codes

Code	Meaning
00 :	Function successful
255 :	Physical error, refer to register H
01 :	Reading unwritten data or no available directory space (Write Sequential)
02 :	No available data block
03 :	Cannot close current extent
04 :	Seek to unwritten extent
05 :	No available directory space
06 :	Random record number out of range
10 :	Media changed (a media change was detected on the FCB's drive after the FCB was opened)

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

17. SEARCH FOR FIRST
18. SEARCH FOR NEXT

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

Table B-2. BDOS Directory Codes

Code	Meaning
00 - 03 :	Successful function
255 :	Unsuccessful function

A successful directory code identifies the relative starting position of the directory entry in the calling program's current DMA buffer. If the SET BDOS ERROR MODE function is used to place the BDOS in return error mode, the following functions return an error flag on physical errors:

14. SELECT DISK
15. OPEN FILE
16. CLOSE FILE
19. DELETE FILE
22. MAKE FILE
23. RENAME FILE
30. SET FILE ATTRIBUTES
35. COMPUTE FILE SIZE
48. FLUSH BUFFERS

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

Table B-3. BDOS Error Flags

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

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

Table B-4. BDOS Physical Errors

Code	Meaning
00 :	No error, or not a physical error
01 :	Disk I/O error
02 :	Read/Only disk
03 :	Read/Only file or file opened under user zero from another user number
04 :	Invalid Drive : drive select error

The following two functions represent a special case because they return an address in registers H and L.

27. GET ADDR (ALLOC)
31. GET ADDR (DISK PARMS)

When the BDOS is in return error mode and it detects a physical error for these functions, it returns to the calling program with registers A, H, and L all set to 255. Otherwise, they return no error code.

End of Appendix B

Appendix C
User Number Conventions

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

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

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

User 0 has special properties under Personal CP/M. When the current user number is not equal to zero and if a requested file is not present under the current user number, the file system automatically attempts to open the file under user zero. If the file exists under user zero and if it has the system attribute, t2', set, the file is opened from user zero. Note, however, that files opened in this way cannot be written to; they are available only for read access. This procedure allows utilities that may include overlays and any other commonly accessed files to be placed on user zero, but also be available for access from other user numbers. As a result, commonly needed utilities need not be copied to all user numbers on a directory, and you can control which user zero files are directly accessible from other user numbers.

End of Appendix C

Index

Personal CP/M™

8-Bit Operating System

System Guide

Foreword

Personal CP/M™ is a single-user operating system for 8-bit computers that use the Zilog® Z80® microprocessor. Personal CP/M is upward-compatible with its predecessor, CP/M® release 2, and offers more features and higher performance than CP/M release 2. This manual describes the procedures required to adapt Personal CP/M for a custom hardware environment.

Personal CP/M Documentation Set

The Personal CP/M documantation set includes the following manuals:

- Extension of MZ-800 P-CP/M
- Personal CP/M 8-Bit Operating System User's Guide (cited as Personal CP/M User's Guide)
- Personal CP/M 8-Bit Operating System Programmer's Guide (cited as Personal CP/M Programmer's Guide)
- Personal CP/M 8-Bit Operating System System Guide (cited as Personal CP/M System Guide)

The Extension of MZ-800 P-CP/M manual contains explanation of the parts of the User's Guide, Programmer's Guide, and System Guide which change when the P-CP/M is used with the MZ-800, and information on the addtional utilities available with the MZ-800.

The Personal CP/M User's Guide introduces the Personal CP/M operating system and tells how to use it. The Personal CP/M Programmer's Guide presents information for application programmers who are creating or adapting programs to run under Personal CP/M.

This manual, the Personal CP/M System Guide, describes the steps necessary to create or modify a Personal CP/M Basic Input/Output System tailored for a specific hardware environment. This manual assumes you are familiar with systems programming in Z80 assembly language and that you have access to a CP/M release 2 system. It also assumes you understand the target hardware and that you have functioning disk I/O drivers.

You should be familiar with the Personal CP/M Programmer's Guide, which describes the system calls used by the application programmer to interface with the operating system. The Programmer's Utilities Guide for the CP/M Family of Operating Systems documents the assembling and debugging utilities.

How the System Guide Is Organized

Section 1 of the Personal CP/M System Guide is an overview of the component modules of the Personal CP/M operating system. Section 2 provides a description of system generation for all-RAM and ROM/RAM systems.

Section 3 describes bootstrapping procedures for Personal CP/M. Section 4 describes the entry points and the required input and returned parameters of all the modules of the BIOS. Section 5 describes the diskette header and other pertinent data.

In this manual, boldface type represents user input.

Table of Contents

Table of Contents
(continued)

Tables and Figures

Tables

Figures

Section 1
System Overview

1.1 Introduction

This section is an overview of the Personal CP/M operating system, with a description of the system components and how they relate to each other. Included is a discussion of memory configurations and supported hardware. The last portion summarizes the creation of a customized version of the Personal CP/M Basic Input Output System (BIOS).

Personal CP/M provides an environment for program execution on computer systems that use the Zilog Z80 microprocessor chip. Personal CP/M provides rapid access to data and programs through a file structure that supports dynamic allocation of space for sequential and random access files.

Personal CP/M supports a maximum of 16 logical floppy or hard disks, or disk-like devices, with a storage capacity of up to 512 megabytes each. The maximum file size supported is 32 megabytes. You can configure the number of directory entries and block size to satisfy various needs.

Personal CP/M is supplied for user memory sizes up to 64 kilobytes. The operating system requires about 6 kilobytes of memory plus that needed for the BIOS.

1.2 Personal CP/M Organization

Personal CP/M is composed of three system modules: the Console Command Processor (CCP), the Basic Disk Operating System (BDOS), and the Basic Input/Output System (BIOS). These modules are linked together to form the operating system. They are discussed individually in this section.

1.2.1 Memory Layout

The Personal CP/M operating system is designed to reside in the top of available memory. Figure 1-1 illustrates two types of memory configurations: ROM/RAM and an all-RAM system. All or part of the operating system code can reside in ROM, with the remaining portion (data areas) at the top of available RAM. In this event, a gap in memory between RAM and ROM can exist. For systems with all RAM, the entire operating system will be at the top of the available memory (typically 64 kilobytes maximum).

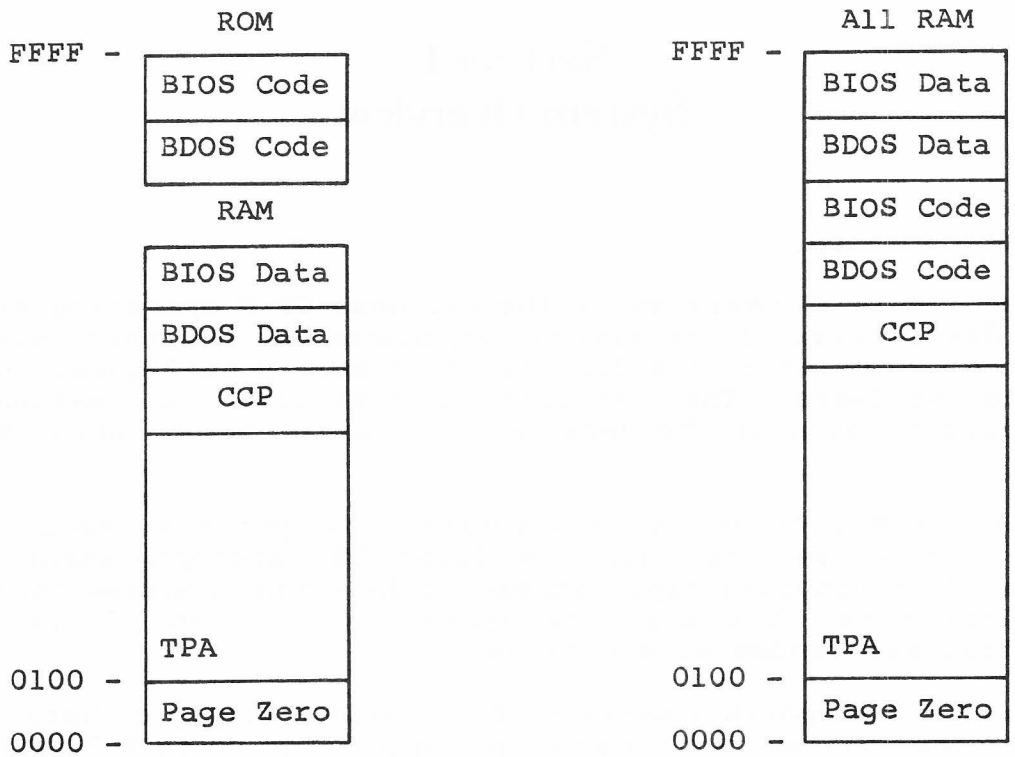


Figure 1-1. Typical Personal CP/M Memory Layout

1.2.2 Console Command Processor

The Console Command Processor (CCP) provides the user interface to Personal CP/M. The CCP uses the BDOS to read user commands and load programs, and provides several built-in user commands. It also provides parsing of command lines entered at the console. Typically, the standard CCP autoloads the Visual CCP (VCCP).

1.2.3 Basic Disk Operating System

The Basic Disk Operating System (BDOS) provides operating system services to applications programs and to the CCP. These include character I/O, disk file I/O (the BDOS disk I/O operations comprise the Personal CP/M file system), and others.

1.2.4 Basic Input/Output System

The Basic Input/Output System (BIOS) is the interface between Personal CP/M and its hardware environment. All physical input and output is done by the BIOS. It includes all physical device drivers, tables defining disk characteristics, and other hardware-specific functions and tables. The CCP and BDOS do not change for

different hardware environments because all hardware dependencies have been concentrated in the BIOS. Each hardware configuration needs its own BIOS. Section 4 describes the BIOS functions in detail. Section 5 discusses the disk parameter and formatting data and blocking/deblocking algorithms.

1.3 Input/Output Devices

Personal CP/M recognizes two basic types of I/O devices: character devices and disk drives. Character devices are serial devices that handle one character at a time. Disk devices handle data in units of 128 bytes, called logical sectors, and provide a large number of physical sectors which can be accessed in random, nonsequential order. Logical and physical sector sizes can be different. In fact, real systems might have devices with characteristics different from disks, such as a block-accessible, random-access tape cassette device. It is the BIOS's responsibility to resolve differences between the logical device models and the actual physical devices.

1.3.1 Character Devices

Character devices are input/output devices that accept or supply streams of ASCII characters to the computer. Typical character devices are consoles, printers, and modems. In Personal CP/M, operations on character devices are done one character at a time.

1.3.2 Disk Devices

Disk devices are used for file storage. They are organized into sectors and tracks. Each logical sector contains 128 bytes of data. (If physical sector sizes other than 128 bytes are used on the actual disk, then the BIOS must do a logical-to-physical mapping to simulate 128-byte logical sectors to the rest of the system.) All disk I/O in Personal CP/M is done in one-sector units. Usually, a track or cylinder of a disk is a group of physical sectors. The number of sectors on a track is a constant depending on the particular device. (The characteristics of a disk device are specified in the Disk Parameter Block for that device. See Section 5 for more information.)

To locate a particular physical sector, the disk, track number, and sector number must all be specified.

1.4 System Generation and Cold Start Operation

Generating a Personal CP/M system is done by linking together the CCP, BDOS, and BIOS to create the operating system. Section 2 discusses how to create the operating system. The bootstrap process is discussed in Section 3.

End of Section 1

Section 2
System Generation

2.1 Overview

This section describes how to build a custom version of Personal CP/M by combining your BIOS with the BDOS supplied by Digital Research. Section 3 describes how to boot the system.

This section assumes that you have access to a working 8-bit CP/M system capable of reading the standard single-sided, single-density 8-inch disk on which Personal CP/M is distributed. You should also be able to create the media (disks, disk-like devices, or ROMs) that the target system will use. It is also assumed that the BIOS is written with an assembler that generates a REL format relocatable object file compatible with the Digital Research LINK-80..linker.

The Personal CP/M operating system is generated by using the linker to resolve external label references between the BDOS and BIOS, and to bind them and the CCP to absolute memory locations.

2.2 Creating a Personal CP/M System File

The CCP and the BDOS for Personal CP/M are distributed on the following three files:

- CCP.REL - for use with all systems
- BDOSH.REL - for use with systems in which the BDOS and BIOS are loaded into and executed from RAM
- BDOSL.REL - for use with systems in which the BDOS and BIOS are executed in ROM

You must link your BIOS with one of the two BDOS files. The BDOSH.REL file is used in systems in which the data segment is linked to a higher address than the code segment, as is typical of systems that execute out of RAM. The BDOSL.REL file is used in systems in which the data segment, which must reside in RAM, is linked to a lower address than the code segment, as is the case in a system where BDOS and BIOS execute out of a ROM at the top end of the address space.

Each of the Personal CP/M elements, CCP, BDOS, and BIOS, must begin on a page boundary; that is, at an address that is a multiple of 100H. The BDOS contains linkage information that automatically forces the BIOS to begin on a page boundary.

For systems in which the BDOS and BIOS execute out of ROM, the BIOS data segment should consist of 'define storage' pseudo-ops only. Any data that must be initialized at cold or warm boot time should be transferred from read-only images of the data in the BIOS code segment.

2.2.1 All-RAM Systems

To generate a Personal CP/M operating system image file that can be loaded into RAM at or near the top of the memory address space, the following procedure should be used:

1. Determine the highest page boundary on which the BDOS can be located. This is done by adding the size of the BDOS code segment (1100H) and the BDOS data segment (00BFH for BDOSH.REL), plus the size of your BIOS code and data segments. For example, if your BIOS code segment is 0A23H bytes, and the data segment is 0280H bytes, then the following memory map represents the logical arrangement of the Personal CP/M system within memory:

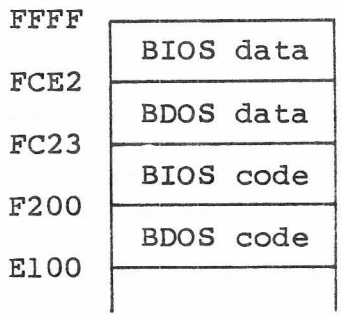


Figure 2-1. All-RAM System Configuration

2. Link the BDOS and BIOS together with the following command:

   A>**link pcpm[le100]=bdosh,bios**

   See the Programmer's Utilities Guide for LINK-80 command line options. This creates the file PCPM.COM, which contains an absolute image of the object code to be loaded at E100H, rather than the standard COM file, which contains an image of the object code to be loaded at 0100H. Note that the BDOS data segment is not required to start on a page boundary in this case.

3. Link the CCP to reside at 0800H less than the load address used in the previous LINK command:

   A>**link ccp[ld900]**

4. The CCP.COM and PCPM.COM files, together with a Cold Boot Loader, can now be written to the system area of the storage media for the target computer. A typical computer system executes a small loader program from ROM, that loads the Cold Boot Loader in from the system area of the storage medium. The Cold Boot Loader then loads the CCP and BDOS/BIOS to the addresses that they are linked to, and finally transfers control to the cold boot entry point of the BIOS.

2.2.2 RAM/ROM Systems

To generate a Personal CP/M operating system image file that can execute from ROM at or near the top of the memory address space, the following procedure should be used:

1. Determine a page boundary in ROM at which to locate BDOS. Do this by adding the size of the BDOS code segment (1100H) and the size of your BIOS code segment. From the size of the BDOS data segment (00CCH for BDOSL.REL) plus the size of your BIOS data segment, determine a page boundary near the top of RAM at which to locate the data segments. Assuming an 8K byte ROM at the top of the address space, and a BIOS with the same size segments as in the "all-RAM" example, then the following memory map represents the logical arrangement of the Personal CP/M system within memory:

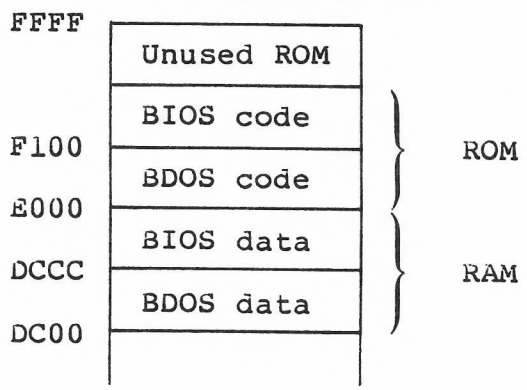


Figure 2-2. ROM/RAM System Configuration

2. Link the BDOS and BIOS together with the command:

   A>**link pcpm[ldc00,ddc00,pe000]=bdosl,bios**

   This creates the file PCPM.COM, which contains an absolute image of the object data and code to be loaded at DC00H, rather than the standard COM file, which contains an image of the object code to be loaded at 0100H. Note that the BDOS data segment is required to start on a page boundary in this case

3. Link the CCP to reside at 0800H less than the load address used in the previous LINK command:

   A>**link ccp[ld400]**

4. The first part of the PCPM.COM file contains an image of the data segments of BDOS and BIOS. The first 2*n sectors of the file, where n is the number of pages difference between the data address and the program address in the LINK command for PCPM.COM, must be discarded by your utility program that creates the ROM. This is because the data segments reside in RAM, and must be treated as uninitialized (see Section 2.2). In the example above, 8 sectors (2*4 pages) would be discarded. The remainder of the PCPM.COM file is then programmed into the ROM.

5. The CCP.COM file, which needs to be reloaded at every Warm Boot, can now be written to the system area of the storage media for the target computer. Another possibility, provided that there is sufficient room (0800H bytes) left over in the ROM, is to store a copy of the CCP in ROM and move it to its execution address at Cold and Warm Boot times.

End of Section 2

Section 3
Bootstrap Procedures

The bootstrap process involves the following three procedures:

1. Do any necessary preliminary hardware initialization.
2. Get the executable object code of the Personal CP/M operating system into memory for execution.
3. Transfer control to the BOOT entry point of the BIOS.

If Personal CP/M is executing out of RAM, the cold boot loader must load the CCP, BDOS, and BIOS into memory at the addresses to which they were linked from the system area of the computer's disk, or disk-like storage media.

If Personal CP/M is executing out of ROM, the BIOS has the responsibility of loading the CCP into memory at cold and warm boot. As mentioned in Section 2, the BIOS is also responsible for initializing any RAM data areas necessary to its operation.

End of Section 3

Section 4
BIOS Functions

4.1 Introduction

All Personal CP/M hardware dependencies are concentrated in subroutines that are collectively referred to as the Basic Input/Output System (BIOS). A Personal CP/M system implementor can tailor Personal CP/M to fit nearly any Z80 operating environment. This section describes the calling conventions and parameters of each BIOS function, and the actions it must perform.

4.2 BIOS Entry Points

Entry to the BIOS is through a jump table located at the beginning of BIOS and labels declared PUBLIC. For Personal CP/M, there are 17 fixed jump vectors, with additional functions being defined as PUBLIC. The 17 jump vectors are listed in Table 4-1, and the PUBLIC routines are listed in Table 4-2. The BIOS subroutines can be empty for certain functions (contain a single RET instruction) during reconfiguration of Personal CP/M, but the entries must be present in the jump vector and PUBLIC declarations as well.

Table 4-1. Standard BIOS Functions

Function	Input	Output
BOOT	None	None
WBOOT	None	None
CONST	None	A=0FFH if ready A=00H if not ready
CONIN	None	A=Character
CONOUT	C=Character	None
LIST	C=Character	None
AUXOUT	C=Character	None
AUXIN	None	A=Character
HOME	None	None
SELDSK	C=Drive 0-15 E=initial select flag	HL=DPH address HL=000H if invalid drive

Table 4-1. (continued)

Function	Input	Output
SETTRK	BC=Track No	None
SETSEC	BC=Sector No	None
SETDMA	BC=DMA Address	None
READ	None	A=00H if no Error A=01H if Nonrecoverable Error
WRITE	C=deblocking code	A=00H if no Error A=01H if Nonrecoverable Error
LISTST	None	A=00H if not ready A=0FFH if ready
SECTRN	BC=Logical Sector Number	HL=Physical Sector Number DE=Translation Table Address

Table 4-2. PUBLIC BIOS Subroutines

Function	Input	Output
?AUXIS	None	A=00H if not ready A=0FFH if ready
?AUXOS	None	A=00H if not ready A=0FFH if ready
?FLUSH	None	A=000H if no error A=001H if physical error A=002H if disk R/O
?DISCD	None	None
?MOV	HL=destination address DE=source address	HL & DE point to next bytes following MOVE
?DSCRF	DE=SFB address	None
?BYTBC*	DE=COPY block address	A=00H implemented copy A=0FFH not implemented
?BYTBA*	DE=ALTER block address	A=00H successful alter A=0FFH not implemented

* Not supported by the MZ-800 P-CP/M

All simple character I/O operations are assumed to be performed in ASCII, both uppercase and lowercase. With some programs an end-of-file condition for an input device is given by an ASCII ^Z (1AH). Peripheral devices are seen by Personal CP/M as logical devices, and are assigned physical devices within the BIOS.

To operate, BDOS needs the CONST, CONIN, CONOUT, ?FLUSH, and ?MOV subroutines (LIST, AUXIN, and AUXOUT may be used by PIP, but not the BDOS). The initial version of BIOS may have empty subroutines for the remaining ASCII devices.

The characteristics of each device are as follows:

CONSOLE	The principal interactive console that communicates with the user. Typically, the CONSOLE is a memory-mapped video display.
LIST	The principal listing device, if it exists in your system. This is an output-only function.
AUXILIARY INPUT	An auxiliary input device, such as serial I/O, paper tape reader, modem, or tape storage peripheral. This is an input only function.
AUXILIARY OUTPUT	An auxiliary output device, such as serial I/O, paper tape punch, modem, or tape storage peripheral. This is an output-only function.

A single peripheral can be simultaneously the LIST, AUXIN, or AUXOUT device. If no peripheral device is assigned as the LIST, AUXIN, or AUXOUT device, the BIOS you create should give an appropriate error message. This prevents the system from hanging if the device is accessed by PIP or some user program.

When the BDOS calls a BIOS function, certain registers will contain information (entry parameters), and are described in the folowing paragraphs. Also, specific registers are used to return information to the BDOS (returned values). The BIOS returns single-byte results in register A, and double-byte values in register pair H and L. For reasons of compatibility, register A = L and register B = H upon return in all cases. The size of the result depends on the particular function.

4.3 BDOS Entry Points

The BDOS contains three PUBLIC entry points: ?bdosc, ?bdosw, and ?bdos. The ?bdosc entry point is called by the BIOS Cold Boot code (see the description of the BOOT entry point). The ?bdosw entry point is called by the BIOS Warm Boot code (see the description of the WBOOT entry point). Finally, the ?bdos entry point is used as the address of the jump instruction written to location 0005h at both Cold and Warm Boot time.

4.4 BIOS Entry Descriptions

BIOS Function: BOOT
Get control from Cold Boot Loader and initialize system.
Entry Parameters: None Returned Values: None

The BOOT entry point gets control from the Cold Start Loader or Power On/Reset code, and is responsible for the following actions:

1. Do any remaining system hardware initialization.
2. Load the CCP, if it was not loaded by the Cold Start Loader.
3. Display a sign-on message (optional).
4. Set 0000H to jump to BIOS WBOOT entry point.
5. Set 0003H to 00H to default to the standard 'A>' CCP, or to 01H to default to the Visual CCP.
6. Set 0005H to jump to ?bdos.
7. Call the ?bdosc entry point in BDOS.
8. Load register C with the default user number in the high nibble, and the default drive number in the low nibble.
9. Jump to CCP+0003H for the standard CCP, or to CCP+0000H for the Visual CCP.

Table 4-3 gives a description of the locations in page zero (0000H through 00FFH) that are used by BOOT and other portions of Personal CP/M.

Table 4-3. Memory Page Zero Definitions

Locations	Contents
00 - 02H	Contains a jump instruction to the warm start entry point. This permits a programmed restart (JMP 0000H).
03H	Used as the VCCP flag: if clear (0), then jump to standard CCP; if set (1), then load and execute the VCCP.
04H	Current default user number (high nibble) and current default drive number (low nibble).
05H - 07H	Contains a jump instruction to the BDOS. A CALL 0005H provides the primary entry point to the BDOS described in the Personal CP/M Programmer's Guide.
08H - 0027H	Interrupt locations 1 through 5 not used.
030H - 037H	Interrupt location 6, not currently used, but reserved.
038H - 03AH	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 Personal CP/M.
03BH - 03FH	Not currently used; reserved.
040H - 04FH	A 16-byte area reserved for scratch by BIOS, but is not used for any purpose in the distribution version of Personal CP/M.
050H - 05BH	Not currently used; reserved.
05CH - 07CH	Default file control block produced for a transient program by the CCP.
07DH - 07FH	Optional default random record position.
080H - 0FFH	Default 128-byte disk buffer. Also filled with the command line when a transient is loaded under the CCP.

BIOS Function: WBOOT
Get control when a warm start occurs.
Entry Parameters: None Returned Value: None

The WBOOT entry point gets control whenever a Warm Boot occurs. That is, a user program jumps to 0000H or calls BDOS with register C set equal to 0, and is responsible for the following actions:

1. Load the CCP.
2. Set 0000H to jump to BIOS WBOOT entry point.
3. Set 0005H to jump to ?bdos.
4. Call the ?bdosw entry point in BDOS.
5. Load register C with the contents of 0004H.
6. If 0003H equals 00H, then jump to CCP+0003H, otherwise jump to CCP+0000H.

BIOS Function: CONST
Sample the status of the console input device.
Entry Parameters: None Returned Value: A=0FFH if a console character is ready to be read A=00H if no console character is ready to be read

Read the status of the currently assigned console device and return 0FFH in register A if a character is ready to be read, or 00H if a character is not ready.

BIOS Function: CONIN
Read a character from the console.
Entry Parameters: None Returned Value: A=console character

Read the next console character into register A with no parity. If no console character is ready, wait until a character is available before returning.

BIOS Function: CONOUT
Output a character to console.
Entry Parameters: C=console character Returned Value: None

This function sends the character from register C to the console output device. The character is in ASCII. You might need to include a delay or filler characters for a linefeed or carriage return if your console device requires some time interval at the end of the line.

BIOS Function: LIST
Output character to list device.
Entry Parameters: C=character Returned Values: None

This function sends an ASCII character from register C to the currently assigned listing device. If your list device requires some communication protocol, it must be handled here.

BIOS Function: AUXOUT
Output a character to the auxiliary output device.
Entry Parameters: C=character Returned Values: None

This function sends an 8-bit character from register C to the currently assigned auxiliary output device.

BIOS Function: AUXIN
Read a character from the auxiliary input device.
Entry Parameters: None Returned Value: A=character

This function reads the next 8-bit character from the AUXIN device into register A.

BIOS Function: HOME
Select track 00 of the specified drive.
Entry Parameters: None Returned Values: None

This function positions the disk head of the currently selected disk to the track 00 position. Usually, you can translate the HOME call into a call on SETTRK with a parameter of 0.

BIOS Function: SELDSK
Select the specified disk drive.
Entry Parameters: C=disk drive (0-15) E=initial select flag Returned Values: HL=address of the Disk Parameter Header if drive exists =0000H if drive does not exist

This function selects the disk drive specified in register C for further operations. Register C contains 0 for drive A, 1 for drive B, and so on up to 15 for drive P.

On each disk select, SELDSK must return in HL the base address of a 16-byte area, called the Disk Parameter Header (DPH), as described in Section 5. For standard floppy disk drives, the contents of the header and associated tables do not change. The program segment included in the sample BIOS performs this operation automatically.

If there is an attempt to select a nonexistent drive, SELDSK returns HL=0000H as an error indicator. Although the function must return the header address on each call, it may be advisable to postpone the physical disk select operation until an I/O function (seek, read, or write) is actually performed. Disk select operations can occur without performing any disk I/O, and many controllers will unload the head of the current disk before selecting the new drive. This could waste time, and cause an excessive amount of noise and head wear. The least-significant bit of register E is zero if this is the first occurrence of the drive select since the last cold or warm start.

BIOS Function: SETTRK
Set specified track number.
Entry Parameters: BC=track number Returned Values: None

Register pair BC contains the track number for a subsequent disk access on the currently selected drive. The sector number in BC is the same as the number returned from the SECTRN entry point. 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 nonstandard disk subsystems.

BIOS Function: SETSEC
Set specified sector number.
Entry Parameters: BC=sector number Returned Values: None

Register pair BC contains the sector number for the subsequent disk access on the currently selected drive. This number is the value returned by SECTRN. Usually, actual sector selection is delayed until a READ or WRITE operation occurs. This number remains in effect until another SETSEC Function is performed.

BIOS Function: SETDMA
Set address for subsequent disk I/O.
Entry Parameters: BC=Direct Memory Access Address Returned Values: None

Register pair BC contains the Direct Memory Access (DMA) address for the subsequent READ or WRITE operation. For example, if B=00H and C=80H when BDOS calls SETDMA, then the subsequent write operation gets its data from 80H through 0FFH, until the next call to SETDMA occurs. The initial DMA address is assumed to be 80H. The controller need not actually support Direct Memory Access. If, for example, all data transfers are through I/O ports, the BIOS that is constructed uses the 128-byte area starting at the selected DMA address for the memory buffer during the subsequent read or write operations.

BIOS Function: READ
Read a sector from the specified drive.
Entry Parameters: None Returned Values: A=000H if no errors occurred A=001H if nonrecoverable error condition occurred

Assuming that the drive has been selected, the track and sector have been set, and the DMA address has been specified, the READ subroutine will attempt to read one sector. The following error codes will be returned in register A:

Zero=no errors detected
Nonzero=nonrecoverable error condition detected

Personal CP/M responds only to a zero or nonzero value. If an error occurs, BIOS should attempt at least ten retries to see if the error is recoverable. When an error is reported, the BDOS will output the message "BDOS ERR ON x: BAD SECTOR". The operator then has the option of typing a RETURN to ignore the error, or ^C to abort.

BIOS Function: WRITE
Write a sector to the specified drive.
Entry Parameters: None Returned Values: A=000H if no error occurred A=001H if nonrecoverable error occurred

Write the data from the currently selected DMA address to the currently selected drive, track, and sector. Upon each call to WRITE, the BDOS provides the same error codes as the READ function.

As in READ, the BIOS should attempt several retries before reporting an error.

BIOS Function: LISTST
Return the ready status of the list device.
Entry Parameters: None Returned Values: A=00H if list device is not ready to accept a character A=0FFH if list device is ready to accept a character

The BIOS LISTST function returns the ready status of the list device.

BIOS Function: SECTRN
Translate sector number given translate table.
Entry Parameters: BC=logical sector number DE=translate table address Returned Values: HL=physical sector number

The user performs logical-to-physical sector translation to improve the overall response of Personal CP/M. Standard Personal CP/M is shipped on a single-sided, single-density 8-inch disk 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 that use fast processors, memory, and disk subsystems, the skew factor can be changed to improve overall response. However, you should maintain a single-density IBM®-compatible version of Personal CP/M for information transfer into and out of the computer system, using a skew factor of 6.

In general, SECTRN receives a logical sector number relative to zero in register BC, and a translate table address in register DE. The sector number is used as an index into the translate table. Registers HL returns the resulting physical sector number. For standard systems, the table and indexing code are provided in the sample BIOS and need not be changed.

For the rest of this section, the BIOS entry points are defined as PUBLICs.

BIOS Function: ?AUXIS
Return input status of auxiliary port.
Entry Parameters: None Returned Values: A=0FFH if ready A=00H if not ready

The AUXIS routine checks the input status of the auxiliary port. This entry point allows full polled handshaking for communications support using an auxiliary port.

BIOS Function: ?AUXOS
Return the output status of auxiliary port.
Entry Parameters: None Returned Values: A=0FFH if ready to accept a character for transmission A=00H if not ready

The AUXOS routine checks the output status of the auxiliary port. This entry point allows full polled handshaking for communications support using an auxiliary port.

BIOS Function: ?FLUSH
Force physical buffer flushing for user-supported deblocking.
Entry Parameters: None Returned Values: A=00H if no error occurred A=01H if physical error occurred A=02H if disk is read-only

The FLUSH buffer entry point allows the system to force physical sector buffer flushing when your BIOS is performing its own record blocking and deblocking.

The BDOS calls the FLUSH routine to ensure that no dirty buffers remain in memory. The BIOS should immediately write any buffers that contain unwritten data.

Note: If you do not implement FLUSH, the routine must return a zero in register A. This can be accomplished by:

```
xra  a
ret
```

BIOS Function: ?DISCD
Discard deblocking buffers.
Entry Parameters: E=drive (0=A, 1=B,...15=P) Returned Values: None

This function must discard the contents of the deblocking buffers for the specified drive, or set a flag indicating that the buffer contents are not valid.

BIOS Function: ?MOV
Move a block of bytes from one location in memory to another.
Entry Parameters: HL=destination address DE=source address BC=byte count Returned Values: HL and DE must point to next bytes following move operation

The BDOS calls the MOVE routine to perform memory-to-memory block moves. This allows use of the Z80 LDIR instruction or special DMA hardware, if available. Note that arguments in HL and DE are reversed from the Z80 machine insruction, necessitating the use of XCHG instructions on either side of the LDIR. The BDOS uses this routine for all large memory copy operations. On return, the HL and DE registers are expected to point to the next bytes following the move.

BIOS Function: ?DSCRF	
Perform direct screen functions.	
Entry Parameters:	DE points to:
byte 0:	Subfunction number
byte 1-2:	Pointer to extended information
OR	
byte 1:	Column value
byte 2:	Row value
Returned Values:	Depends upon subfunction (described below)

The Direct Screen Function routines provide direct access to cursor movement and screen editing functions for video-intensive applications, such as word processing and electronic spreadsheets. Direct access is important in systems with memory-mapped displays. This call not only permits direct access to these functions, but can also return information to the calling program about whether a specific function executes quickly or slowly on a particular system. If a particular function is emulated by BIOS display drivers, the system response will be slower than the direct screen access.

Upon entry to this BIOS function, register DE points to a three-byte block containing the following:

Byte 0: Subfunction number
Bytes 1-2: Pointer to extended information

or

Byte 1: Column value
Byte 2: Row value

It is the responsibility of the BIOS to report in the bit map returned by subfunction 0 whether the subfunction is supported. The subfunctions supported by DSCRF are described in the following paragraphs.

Table 4-4. Direct Screen Subfunctions

Subfunction	Description
0	SUBFUNCTIONS SUPPORTED Returned value: HL=Pointer to a four byte block of memory as follows: byte 0: 07 06 05 04 03 02 01 00 byte 1: 15 14 13 12 11 10 09 08 byte 2: 23 22 21 20 19 18 17 16 byte 3: 27 26 25 24 The corresponding bit is set if a particular subfunction is supported in the BIOS.
1	SUBFUNCTIONS EMULATED Returned value: HL=Pointer to a four-byte block of memory as in Sub-function 0, above.
2	DISPLAY SIZE Returned Value: H=number of columns (n-1) L=number of rows (n-1)
3	IDENTIFY TERMINAL Returned Value: HL=Pointer to null-terminated identifier string. For example, a VT-52-type terminal would return the bytes: ESCape, '/', 'K', NULL.
4	CURSOR UP Does not scroll screen down if the cursor is at the top of screen.
5	CURSOR DOWN Does not scroll screen up if the cursor is at the bottom of screen.

Table 4-4. (continued)

Subfunction	Description
6	CURSOR LEFT Wrap depends on the mode set by Subfunction 26 or 27.
7	CURSOR RIGHT Wrap depends on the mode set by Subfunction 26 or 27.
8	CURSOR HOME Move the cursor to the top left corner of screen.
9	CURSOR ON Make the cursor visible.
10	CURSOR OFF Make the cursor invisible.
11	DIRECT CURSOR ADDRESSING Move the cursor to absolute column and row indicated by the second and third bytes pointed to by DE upon entry to the DSCRF function.
12	CLEAR DISPLAY Move the cursor to the top left corner of the screen, and erase the screen.
13	ERASE TO END OF LINE Erase all characters to the right of the cursor.
14	ERASE TO END OF SCREEN Erase all characters to the right of the cursor to the end of the screen.
15	ENTER ANSI MODE Place the display hardware in the ANSI mode.

Table 4-4. (continued)

Subfunction	Description
16	ENTER VT52 MODE Place the display hardware in the VT52 mode.
17*	ENTER GRAPHICS MODE Place the display hardware in the graphics mode.
18*	EXIT GRAPHICS MODE Return the display hardware to the current terminal mode, either ANSI or VT52.
19*	ENTER ALTERNATE KEYPAD MODE
20*	EXIT ALTERNATE KEYPAD MODE
21	ENTER HOLD SCREEN MODE
22	EXIT HOLD SCREEN MODE
23	ENTER REVERSE VIDEO MODE
24	EXIT REVERSE VIDEO MODE
25	REVERSE LINE FEED
26	ENABLE WRAP-AROUND AT END OF LINE
27	TRUNCATE CHARACTERS AT END OF LINE
28 to 255	Reserved

* Not supported by the MZ-800 P-CP/M

Section 5
Disk Definition Information

5.1 Introduction

The BIOS provides a standard interface to the physical input/output devices in your system. The BIOS interface is defined by the functions described in Section 4. Those functions, taken together, constitute a model of the hardware environment. Each BIOS is responsible for mapping that model onto the real hardware.

In addition, the BIOS contains disk definition tables which define the characteristics of the disk devices which are present, and provides some storage for use by the BDOS in maintaining disk directory information.

Section 4 describes the functions that must be performed by the BIOS, and the external interface to those functions. This Section contains additional information describing the reserved locations in page zero, and the structure and significance of the disk definition tables and information about sector blocking and deblocking. Careful choices of disk parameters and disk buffering methods are necessary if you are to achieve the best possible performance from Personal CP/M. Therefore, this section should be read thoroughly before writing a custom BIOS.

5.2 Disk Definition Tables

As in other CP/M systems, Personal CP/M uses a set of tables to define disk device characteristics. This section describes each of these tables and discusses choices of certain parameters.

5.2.1 Disk Parameter Header

Each disk drive has an associated 16-byte Disk Parameter Header (DPH) that contains information about the disk drive and also provides a scratch pad area for certain BDOS operations. Each drive must have its own unique DPH. The format of a Disk Parameter Header is shown in Figure 5.1.

XLT	0000	0000	0000	DIRBUF	DPB	CSV	ALV
16b	16b	16b	16b	16b	16b	16b	16b

Figure 5-1. Disk Parameter Header

Each element of the DPH is a word (16-bit) value and is described in Table 5-1.

Table 5-1. Disk Parameter Header Elements

Address	Description
XLT	Address of the logical-to-physical sector translation table, if used for this particular drive. Otherwise, the value of 0000H if there is no translation table for this drive (that is, the physical and logical sector numbers are the same). Disk drives with identical sector translation can share the same translate table.
0000	Three scratch pad words for use within the BDOS. The initial value is unimportant.
DIRBUF	Address of a 128-byte scratch pad area for directory operations within BDOS. All DPHs address the same scratch pad area.
DPB	Address of a disk parameter block for this drive. Drives with identical disk characteristics can address the same disk parameter block.
CSV	Address of a scratch pad area used for software check for changed disks. This address is different for each DPH.
ALV	Address of a scratch pad area used by the BDOS to keep disk storage allocation information. This address is different for each DPH.

Given n disk drives, the DPHs are arranged in an array. The first row of 16 bytes corresponds to drive 0, with the last row corresponding to drive n-1. The array appears in Figure 5-2.

DPBASE:

00	XLT 00	0000	0000	0000	DIRBUF	DBP 00	CSV 00	ALV 00
01	XLT 01	0000	0000	0000	DIRBUF	DBP 01	CSV 01	ALV 01
	(and so on through)							
n-1	XLT n-1	0000	0000	0000	DIRBUF	DBPn-1	CSVn-1	ALVn-1

Figure 5-2. Array of DPH Entries

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.

```
NDISKS      EQU     4           ;NUMBER OF DISK DRIVES
....
SELDSK:     ;SELECT DISK GIVEN BY BC
            LXI     H,0000H     ;ERROR CODE
            MOV     A,C         ;DRIVE OK?
            CPI     NDISKS      ;CY IF SO
            RNC                 ;RET IF ERROR

            ;NO ERROR, CONTINUE
            MOV     L,C         ;LOW (DISK)
            MOV     H,B         ;HIGH(DISK)
            DAD     H           ;*2
            DAD     H           ;*4
            DAD     H           ;*8
            DAD     H           ;*16
            LXI     D,DPBASE    ;FIRST DPH
            DAD     D           ;DPH(DISK)
            RET
```

Figure 5-3. SELDSK Example

The translation vectors (XLT 00 through XLTn-1) are located elsewhere in the BIOS, and simply correspond one-for-one with the logical sector numbers zero through the sector count 1.

5.2.2 Disk Parameter Block

The Disk Parameter Block (DPB), which is addressed by one or more DPHs, takes this general form:

SPT	BSH	BLM	EXM	DSM	DRM	AL0	AL1	CKS	OFF
16b	8b	8b	8b	16b	16b	8b	8b	16b	16b

where each is a byte or word value, as shown by the 8b or 16b indicator below the field. Table 5-2 gives a description of the Disk Parameter Block.

Table 5-2. Disk Parameter Block Description

Word Value	Description
SPT	total number of sectors per track
BSH	data allocation block shift factor, determined by the data block allocation size
BLM	data allocation block mask ((2^BSH)-1)
EXM	extent mask, determined by the data block allocation size and the number of disk blocks
DSM	total storage capacity of the disk drive
DRM	total number of directory entries that can be stored on this drive. (AL0,AL1 determine reserved directory blocks.)
CKS	size of the directory check vector
OFF	number of reserved tracks at the beginning of the (logical) disk

The values of BSH and BLM implicitly determine the data allocation size, BLS, which is not an entry in the DPB. Given that the designer has selected a value for BLS, the values of BSH and BLM are shown in the following tabulation:

BLS	BSH	BLM
1024	3	7
2048	4	15
4096	5	31
8192	6	63
16384	7	127

All values are decimal. The value of EXM depends upon both the BLS and whether the DSM value is less than 256 or greater than 255. For DSM less than 256, the value of EXM is given by:

BLS	EXM
1024	0
2048	1
4096	3
8192	7
16384	15

For DSM greater than 255, the value of EXM is given by:

BLS	EXM
1024	N/A
2048	0
4096	1
8192	3
16384	7

The value of DSM is the maximum data block number measured in BLS units supported by this particular drive. The product BLS * (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 one less than the total number of directory entries that can take on a 16-bit value. The values of AL0 and AL1 are determined by DRM. AL0 and AL1 values together can be considered a string of 16-bits, as shown below:

AL0								AL1							
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15

Position 00 corresponds to the high-order bit of the byte AL0, and position 15 corresponds to the low-order bit of the byte AL1. Each bit position reserves a data block for a 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 through position 15). Each directory entry occupies 32 bytes, resulting in the following tabulation:

BLS	Directory Entries
1024	32 times # bits
2048	64 times # bits
4096	128 times # bits
8192	256 times # bits
16384	512 times # bits

If DRM = 127 (128 directory entries) and BLS = 1024, there are 32 directory entries per block, requiring four reserved blocks. In this case, the four high-order bits of AL0 are set, resulting in the values AL0 = 0F0H and AL1 = 00H.

The CKS value is determined as follows:

1. If the disk drive media is removable, then CKS = (DRM + 1)/4, where DRM is the last directory entry number.

2. If the media are fixed, then CKS = 0. No directory records are checked in this case.

Finally, the OFF field determines the number of tracks that are skipped at the beginning of the physical disk. This value is automatically added whenever SETTRK is called. It 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, several DPHs can address the same DPB if their drive characteristics are identical. Further, the DPB can be dynamically changed when a new drive is addressed. Since the BDOS copies the DPB values to a local area whenever the SELDSK function is called, simply change the pointer in the DPH.

Returning back to the DPH for a particular drive, the two address values, CSV and ALV, reference areas of uninitialized memory in the BIOS data segment. 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, you must reserve (DRM + 1)/4 bytes for directory check use. If CKS = 0, no storage is reserved.

The size of the area addressed by ALV is determined by the maximum number of data blocks allowed for this particular disk, and is equal to 2*(DSM/8+1). Two copies of the allocation map for the disk are kept in this area: the first vector stores temporarily allocated blocks resulting from write operations, the second stores permanently allocated blocks resulting from CLOSE FILE operations.

5.3 The DISKDEF Macro Library

A macro library which is on the destribution disks, called DISKDEF, greatly simplifies the table construction process. Of course, you must have access to the MAC™ macro assembler to use the DISKDEF facility.

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

```
MACLIB    DISKDEF

.....
DISKS     n
DISKDEF   0,...
DISKDEF   1,...

.....
DISKDEF   n-1

.....
ENDEF
```

The MACLIB statement loads the DISKDEF.LIB file (on the same disk as the BIOS) into MAC's internal tables. The DISKS macro call follows, which specifies the number of drives to be configured with the user's system, where n is an integer from 1 to 16. A series of DISKDEF macro calls then follow that define the characteristics of each logical disk, 0 through n-1 (corresponding to logical drives A through P). The DISKS and DISDEF macros generate the in-line fixed data tables described in the previous section, and must be placed in a nonexecutable portion of the BIOS, typically directly following the BIOS jump vector.

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

The form of the DISKDEF macro call is as follows:

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

where

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

The value dn is the drive number being defined with this DISKDEF macro invocation. Parameter fsc accounts for differing sector numbering systems is usually zero or one. The last numbered sector on the track is defined by lsc. 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 1-byte table is created. Otherwise, each translation table element occupies two bytes. No sector translation table is created if the skf parameter is either omitted or equal to zero. 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. Also, each directory entry addresses more data, and the BIOS-resident data space is reduced.

The dks parameter specifies the total disk size in bls units. That is, if the bls = 2048 and dks = 1000, the total disk capacity is 2,048,000 bytes. If dks is greater than 255, 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. It is used internally to detect changed disks during system operation, where an intervening cold or warm boot has not occurred. When a disk is removed, Personal CP/M automatically marks the disk as read-only.

As mentioned earlier, the value of cks=dir when the medium is easily changed, as in a floppy disk subsystem. If the disk is permanently mounted, the value of cks is typically zero, since the probability of changing disks without a warm start is low.

The value of ofs determines the number of tracks to skip when this particular drive is addressed. This permits reserving a number of tracks for the operating system, or for simulating a number of drives on a single large capacity physical drive.

Finally, the [0] parameter is included when file compatibility is required with versions of CP/M 1.4 that have been modified for higher density disks. This parameter ensures that only 16K bytes is allocated for each directory record, as was the case for earlier CP/M 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 common 4-drive, single-density system, which is compatible with CP/M 1.4, is defined using the following macro invocations:

```
DISKS     4
DISKDEF   0,1,26,6,1024,243,64,64,2
DISKDEF   1,0
DISKDEF   2,0
DISKDEF   3,0

....
ENDEF
```

with all disks having the same parameter values of 26 sectors for each track (numbered 1 through 26), 6 sectors skipped between each access, 1024 bytes for each 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 DPHs, starting at the DPH table address DPBASE generated by the macro. Each disk header block contains 16 bytes, as described earlier, and correspond one-for-one to each of the defined drives. For example, in a 4-drive system, the DISKS macro generates a table of the form:

```
DPBASE    EQU $
DPE0:     DW XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV0,ALV0
DPE1:     DW XLT0,0000H,0000H,0000H,DIRBUF,DPB1,CSV1,ALV1
DPE2      DW XLT0,0000H,0000H,0000H,DIRBUF,DPB2,CSV2,ALV2
DPE3      DW XLT0,0000H,0000H,0000H,DIRBUF,DPB3,CSV3,ALV3
```

where the DPH labels are included for reference purposes to show the beginning table addresses for each drive, zero through three. The values contained within the DPH are described in detail in Section 5.2.1. The check and allocation vector addresses are generated by the ENDEF macro in the RAM area following the BIOS code and tables.

You should note that if the skf (skew factor) parameter is omitted (or equal to zero), the translation table is omitted, and a 0000H value is inserted in the XLT position of the DPH 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 (nonzero) table address is placed into the corresponding DPHs. For example, the following tabulation 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 that 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 this:

```
4C72 =    BEGDAT EQU $
          (data areas)
4DB0 =    ENDDAT EQU $
013C =    DATSIZ EQU $-BEGDAT
```

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 utilize the STAT program to check drive characteristics, because STAT uses the disk parameter block to decode the drive information:

```
STAT d:DSK:
```

This command decodes the disk parameter block for drive d (d=A through P) and displays the following values:

```
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 example produces an 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, c128, e=1024, b=128, s=58, t=2
```

5.4 Sector Blocking and Deblocking

Upon each call to the BIOS WRITE entry point, the Personal CP/M BDOS includes information that allows effective sector blocking and deblocking where the host subsystem has a sector size that is a multiple of the basic 128-byte unit. The purpose here is to present a general-purpose algorithm that can be included within the BIOS and that uses the BDOS information to perform the operations automatically.

On each call to WRITE, the BDOS provides the following information in register C:

```
0=normal sector write
1=write to directory sector
2=write to the first sector of a new data block
```

Condition zero occurs whenever the next write operation is into a previously written area, such as a random mode record update, or 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 one occurs when a write into the directory area is performed.

Condition two 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; there is little overhead involved in either operation when blocking or deblocking records, since preread operations can be avoided when writing records.

End of Section 5

Index

Printed in Japan
Gedruckt in Japan
Imprimé au Japon
Stampato in Giappone

4M 0.5-I(TINSE1288ACZZ)②

SHARP CORPORATION
Osaka, Japan