HEWLETT
PACKARD

HP 82900A CP/M® System
Reference Manual

HP-87

April 1982

00087-90001

Printed in U.S.A.

CONTENTS

Section		Page
1	GENERAL INFORMATION	
1.1	Introduction	1-1
1.2	CP/M Documentation	1-1
1.3	HP 82900A CP/M System	1-2
1.4	Bootstrap Loading	1-3
1.5	The CP/M Operating System	1-4
1.6	CP/M Commands	1-4
1.7	Reference Manual Organization	1-7
1.8	Entering CP/M Commands	1-7
1.9	The PIP Command	1-8
1.10	The STAT Command	1-17
1.11	Batch Job Processing	1-22
2	CP/M INTERNAL ORGANIZATION	
2.1	Memory Organization	2-1
2.2	FDOS	2-3
2.3	CCP	2-3
2.4	TPA	2-4
2.5	Base Page	2-4
2.6	How the CCP Functions	2-5
2.7	Transient Programs	2-6
3	CP/M FILE SYSTEM ORGANIZATION	
3.1	File References	3-1
3.2	File Structure	3-2
3.3	File Access	3-2
3.4	File System Overview	3-3
3.5	File Control Block	3-4
3.6	Disc Directory	3-7
3.7	Default FCBs and the CCP	3-8
3.8	FCB Example	3-8
4	SYSTEM FUNCTION CALLS	
4.1	Introduction	4-1
4.2	Accessing Function Calls	4-2
4.3	Extended System Function Calls	4-46

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

BIOS ROUTINES

5.1 Introduction 5-1
5.2 BIOS Entry Points 5-1
5.3 Disc Tables 5-4

6 THE CP/M TEXT EDITOR

6.1 Introduction 6-1
6.2 Memory Image 6-2
6.3 ED Commands 6-3
6.4 Sample ED Session 6-16

7 ASSEMBLY LANGUAGE UTILITIES

7.1 Introduction to ASM 7-1
7.2 Using the Assembler 7-1
7.3 Assembly Language Programming 7-3
7.4 Assembler Directives 7-10
7.5 Operation Codes 7-13
7.6 Assembler Execution 7-19
7.7 Error Messages 7-20
7.8 The LOAD Command 7-23
7.9 The DUMP Command 7-23
7.10 Introduction to DDT 7-23
7.11 Starting DDT 7-24
7.12 Using DDT 7-24
7.13 DDT Commands 7-24
7.14 Implementation Notes 7-33
7.15 The SAVE Command 7-33

Appendix A: DISC FILE ORGANIZATION

Appendix B: CP/M KEYBOARD FUNCTIONS

Appendix C: ASCII CHARACTER CODES

Appendix D: CP/M KEYCODES

Appendix E: CURSOR CONTROL CODES

Appendix F: CPU REGISTERS

Appendix G: CPU INSTRUCTION SET

Appendix H: CP/M ERROR MESSAGES

Appendix I: ASM ERROR MESSAGES

Appendix J: ANNOTATED BIBLIOGRAPHY

Section
I

GENERAL INFORMATION

1.1 Introduction

CP/M or Control Program for Microcomputers is an operating system developed by Digital Research, Inc. for use with 8080, 8085, or Z-80 based microcomputers. CP/M systems are available for several types of microcomputers using 5-1/4-inch and 8-inch flexible disc drives. Some of the CP/M system routines can be applied universally to systems based on the 8080, 8085, or Z-80 microprocessor. Other parts of CP/M must be customized in order to function with a specific type of microcomputer. The HP 82900A CP/M System is an implementation of CP/M for the HP-87 Personal Computer, and is designed to operate with either 5-1/4-inch or 8-inch, HP-formatted, flexible discs.

1.2 CP/M Documentation

In order to meet the needs of a widely varying audience, from the beginner to the experienced CP/M user, three manuals are provided with your CP/M system. Each has a different application and assumes a different level of CP/M experience. Before reading about or attempting to use your CP/M system you should be familiar with the operation of the HP-87. Installation and "getting started" instructions are presented in your computer's introductory manual, and a complete discussion of BASIC programming is presented in your computer's operating and programming manual.

The CP/M documentation provided with your system includes:

- Introduction to the HP 82900A CP/M System: The introductory manual is provided for the user with no prior experience with CP/M, and those who intend to use their system primarily for operating prepackaged software. Included, are instructions for installing the CP/M plug-in module and "booting-up" the CP/M system, a discussion of the CP/M built-in commands, and a look at some other CP/M utilities.

- HP 82900A CP/M System Reference Manual: The reference manual is designed for the audience of experienced CP/M users, and provides the detailed information necessary for program development. This manual includes a discussion of the CP/M transient commands, internal organization, file system, and other technical information relevant to programming. This manual is, in itself, not a complete guide to programming with CP/M. A bibliography is provided in appendix J which lists other CP/M references, including sources for assembly language programming.

- HP 82900A CP/M System Pocket Guide: The pocket guide provides a quick reference to the CP/M system details, in condensed form, for the experienced programmer or user.

In addition to these manuals, there are several books and user's guides which can provide you with additional information about CP/M. Some selected references are listed in appendix J.

1.3 HP 82900A CP/M System

Your CP/M system comprises a plug-in module and an associated 5-1/4-inch disc.

The CP/M Module

The central component of the module is a Z-80A microprocessor, which takes control of your computer while operating in "CP/M mode." Additional components include 64K bytes of RAM, a 2K "boot" ROM, and interfacing circuitry. CP/M occupies approximately 8K bytes of RAM, leaving 56K bytes of memory available to the user.

The CP/M Disc

The CP/M disc is HP formatted, and contains a copy of CP/M Version 2.2, modified for your computer. Also on the disc is a binary program named "CP/M", which enables your computer to exchange information with the Z-80A, and a BASIC autostart program which specifies the pagesize and printer address, and loads the binary. An 8-inch floppy disc is not provided, although the CP/M system is compatible with this size disc. Refer to Introduction to the HP 82900A CP/M System for instructions about copying the CP/M system from a 5-1/4-inch disc to an 8-inch disc.

5-1/4-Inch Disc Format

The 5-1/4-inch HP disc is initially formatted in LIF (Logical Interchange Format). The LIF directory is only one sector long, and includes three entries. The first entry is a large data file named "CP/MSYS" which includes the CP/M operating system, the CP/M directory, and the CP/M user's file space. This data file must not be purged or written to, or the entire CP/M contents of the disc will be lost. As a precaution, do not attempt to store HP BASIC programs on the CP/M disc, or CP/M programs on an HP BASIC disc. In other words, keep your CP/M and BASIC discs labeled, and in separate locations.

The second directory entry is a BASIC autostart program used to bootstrap the CP/M system. The third directory entry is the binary program that runs in your computer while CP/M runs in the module. A detailed sector map for the 5-1/4-inch disc is provided in appendix A.

8-Inch Disc Format

Your system is compatible with an 8-inch disc, although one is not provided. As mentioned, instructions for copying your CP/M system onto an 8-inch disc appear in Introduction to the HP 82900A CP/M System. The 8-inch disc will contain all the information residing on the 5-1/4-inch disc, but at different locations. A detailed sector map for the 8-inch disc is provided in appendix A.

1.4 Bootstrap Loading

Because the memory available to CP/M and its programs is cleared each time your system is turned off, the operating system must be brought into memory each time you turn on your computer. The CP/M operating system resides permanently on your system disc. Bringing the system from disc into the CP/M module RAM is known as "bootstrap loading" or a "cold start" and is implemented as follows:

a. Turn on your disc drive and insert the disc marked Hewlett-Packard CP/M System into the drive with the lowest HP-IB device address. This drive is identified as drive "A" by CP/M.

b. With the CP/M module installed in one of the four rear ports, turn on your computer. Allow approximately 20 seconds for warm-up and self-test.

c. The "A" disc access light should come on, indicating disc activity. After a short period of time, your computer should print a few lines of start-up information, the CP/M prompt "A>" will appear, and CP/M is ready to go. If the CP/M prompt does not appear, refer to Introduction to the HP 82900A CP/M System for instructions about maintenance and service.

CP/M remains in memory, ready to work, until you turn off or reset your computer.

Other start-up procedures such as identifying the system printer and specifying the alpha display pagesize are discussed in Introduction to the HP 82900A CP/M System.

1.5 The CP/M Operating System

CP/M contains seven "built-in" commands, which are discussed briefly in this section and in more detail in Introduction to the HP 82900A CP/M System. CP/M also enables you to load "transient" commands and user written programs into memory, access sequential and random access disc files, and address various physical and logical devices. CP/M provides the common structure, fundamental I/O calls, and uniform memory organization. User written programs can utilize all or part of these CP/M features independently.

Keep in mind, as you continue through this manual, that many programming applications can be implemented through a high-level language. Most of these language subsystems provide a simplified access to the CP/M features, so that a detailed knowledge of CP/M is not necessary to be productive with your computer. However, if you elect to program in assembly language, these CP/M features will become an integral part of your application.

1.6 CP/M Commands

Once you have the CP/M prompt, your computer is ready to accept a variety of commands and program names as your needs dictate. Several of the features of the operating system are implemented via the seven commands internal to CP/M. These internal, "built-in" commands are quite distinct from the extended set of "transient" commands which you will see later iafthis section. CP/M commands are executed by pressing the [END LINE] key after entering the command line, in much the same way as HP "native mode" commands. For more detailed information about entering commands, refer to Introduction to the HP 82900A CP/M System, and your computer's introductory manual.

Built-In Commands

The built-in commands are described in Introduction to the HP 82900A CP/M System as well as in most of the commercially available CP/M books. A brief description of each command is provided for your benefit in table 1-1. The SAVE command, which is used to store user developed programs on disc, is also discussed in section 7, with the assembly language utilities.

Table 1-1. CP/M Built-In Commands

Command	Function
DIR	List the directory of a specified flexible disc.
ERA	Erase (delete) one or more file name entries from a directory.
REN	Rename a specified file entry.
SAVE	Store an image of memory into a disc file.
TYPE	Type (list) a specified file to the CP/M console.
USER	Log into a particular user area of a disc.
x:	Changes the current logged disc to disc "x".

Transient Commands

As mentioned above, CP/M also provides several additional commands that are disc resident. These are called "transient" commands or "utilities" because they are called into memory only when requested by the user. Built-in commands always reside in memory, and utilize memory space whether active or not. CP/M transient commands are equivalent to utility programs on other computer systems. CP/M makes no distinction between standard CP/M transient commands and assembly language or compiled programs written by the user. Keep in mind that user written programs or transient commands may reside on a different disc than the CP/M system. CP/M looks for the requested transient command file only on the currently logged disc drive, unless a drive specifier is included with the command.

The standard CP/M transient commands provided with your system are listed in table 1-2. All of these are located on the CP/M System disc. As indicated, some of the commands are described in Introduction to the HP 82900A CP/M System, while others are discussed in this manual.

Generally, transients related to program development are documented in sections 6 and 7 of this manual. Four of the transients that have general uses as well as specific programming applications, are documented later in this section. Some of the general purpose transient commands for copying files and discs, and operating peripherals, are discussed in Introduction to the HP 82900A CP/M System. Generally, the material in the HP 82900A CP/M System Reference Manual is more in-depth and oriented towards the system programmer. Consider this as supplementary to the material included in the Introduction to the HP 82900A CP/M System.

Table 1-2. CP/M Transient Commands

Name	Description
ASM	The CP/M 8080 Assembler Program for writing assembly language routines. Refer to section 7.
DDT	Dynamic Debugging Tool, an aid in verifying assembly language program execution. Refer to section 7.
DUMP	Produce a hex listing of a file. Refer to section 7.
ED	The CP/M character oriented text editor program. Refer to section 6.
FORMAT	Performs a complete surface test and initializes disc media. Refer to Introduction to the HP 82900A CP/M System.
LOAD	The CP/M loader. This makes ASM output files executable. Refer to section 7.
PIP	The Peripheral Interchange Program which permits file and device transfers. Documented in this section and in Introduction to the HP 82900A CP/M System.
STAT	CP/M device status program. Refer to Introduction to the HP 82900A CP/M System for operator information, or read in this section about device assignments.
SUBMIT	This CP/M transient permits batch job execution with no user interaction. Documented in this section.
XSUB	This program extends the capabilities of SUBMIT to allow "interactive" input. Documented in this section.

1.7 Reference Manual Organization

The remainder of this section is devoted to the four general purpose transient commands PIP, STAT, SUBMIT, and XSUB. Both the PIP and STAT commands are useful in determining device and general file management. The SUBMIT and XSUB commands allow limited "batch" processing for unattended CP/M operation. Sections 2 and 3 discuss the more technical aspects of CP/M, including the internal system structure and disc file organization. Sections 4, 5, and 7 discuss CP/M transients and internal features related to assembly language programming. Section 6 discusses the text editor utility (ED) provided with CP/M. The ED command is useful for preparing assembly language source files, as well as other types of files. In addition, the appendices provide a summary of some technical information related to CP/M and your computer.

1.8 Entering CP/M Commands

CP/M commands are indicated in uppercase letters and enclosed in brackets throughout the manual. CP/M does not distinguish between lower and uppercase letters, treating everything as uppercase. For clarity and continuity in describing the CP/M commands, all commands appear in uppercase.

Usually, command descriptions are followed by one or more examples, which indicate how the command is entered, and the resulting action. The CP/M prompt (for example A>, if logged onto drive A) and other prompts invoked by the specific command are included in the examples.

Your computer's keys are also indicated in uppercase letters throughout the manual. For the sake of brevity, the [CTRL] key is indicated by an up arrow (^) when a keyboard operation requires that you press [CTRL] and another key simultaneously. For example, ^P indicates the [CTRL] key and the [P] key are pressed simultaneously. Recall that commands are entered by pressing the [END LINE] key after the command is typed; this action does not appear in the examples.

CP/M comes equipped with some command line editing features which aid the user in entering CP/M commands. These features, and other control character functions are discussed in Introduction to the HP 82900A CP/M System. Familiarize yourself with these features before attempting to enter commands.

To abort a CP/M command at any time, type in ^C. This operation, referred to as a "warm start," reloads the system from drive A and returns to the drive which was current before the ^C.

1.9 The PIP Command

PIP, The Peripheral Interchange Program, is a general-purpose file transfer program. As documented in Introduction to the HP 82900A CP/M System, PIP allows you to move files between disc devices and several "logical devices" as defined through IOBYTE device mapping (IOBYTE is discussed in section 2). The general format of the PIP command input is:

```
destination = source1 [options], source2 [options],...
```

Valid source parameters include any existing disc file reference, with an optional drive identifier, or any "wild card" specifier, and logical and physical input device names. Destination devices include valid file references, or default names and types based on the corresponding source file specified, and logical and physical output device names.

The PIP option parameter is used to specify any special operations to be performed during the PIP transfer. These PIP options can be left out of the command line if none are desired. First we'll discuss how the PIP command is entered, then, the use of device names, and finally the PIP option parameters.

There are two methods of entering the PIP command. Both methods use the above command input format. The first method is more convenient for entering a series of PIP commands; the second is better suited for a single PIP command.

1. PIP can be entered without destination and source inputs, by typing in PIP after the CP/M prompt, and pressing the [END LINE] key. PIP is loaded, and the system responds with an asterisk prompt.

   ```
   A>PIP
   *
   ```

   The PIP command input is typed in after the asterisk; pressing END LINE starts execution of the command. When the specified PIP operation is complete, the system responds with another asterisk prompt. Multiple PIP commands can be entered and executed in this fashion. When all the desired PIP commands have been executed, return control to the system by typing a ^C or an [END LINE], following the asterisk.

Examples:

Command	Action
A>PIP	Loads the PIP command.
*A:FILE3=B:FILE1.TXT, C:FILE2.TXT[E]	Copies B:FILE1.TXT followed by C:FILE2.TXT to A:FILE3.TXT. The E option is employed during the copying of C:FILE2.TXT.
B:FILE.=B:OLD.*	Copies all files named OLD of any extension to FILE with the same extension.

2. The PIP source and destination inputs can also be entered directly in the PIP command. The single PIP command is executed, and control is returned to the system.

   Example:

Command	Action
A>PIP C:=A:*.*	Copies all files from disc "A" to disc "C" except files designated as "System Files" (refer to STAT command).

Logical and physical device names may also be specified as source and destination parameters. When used in this way, the device name is followed by a colon, and entered in the command input.

Examples:

Command	Action
A>PIP CON:=A:GJOB.TXT	Copies A:GJOB.TXT to the CON: output device.
A>PIP *FILE.TXT=CON:	Loads the PIP command. Copies the CON: input to the file FILE.TXT.

A list of source and destination device names are given in table 1-3. Since there is only one input device (the console) and 2 output devices (the CRT and the system printer), some device names are redundant. The default power-on physical device maps are indicated.

Note: When using "CON:" as an input device in PIP, all input will go to the destination until a ^Z is typed, when control will be returned to either PIP or the operating system, depending on the way the PIP command was entered.

Table 1-3. PIP Device Names

Logical Device Name	Possible Physical Device Maps	Result
Source Devices		
CON: (Computer Console)	TTY: CRT: (default) BAT: UC1:	All physical input devices use keyboard input.
RDR: (CP/M Reader)	TTY: (default) PTR: UR1: UR2:	
Destination Devices		
CON: (Computer Console)	TTY: CRT: (default) BAT: UC1:	Output data is lost. Output to CRT. Output to printer. Output data is lost.
PUN: (CP/M Punch)	TTY: (default) PTP: UP1: UP2:	Output data is lost. Output data is lost. Output data is lost. Output data is lost.
LST: (CP/M List)	TTY: CRT: LPT: (default) UL1:	Output data is lost. Output to CRT. Output to printer. Output data is lost.

Special Devices

There are also three special PIP device names which implement specific functions during file transfers. These are provided as standard features of the CP/M system.

The special source device names are:

NUL: sends a sequence of 40 "null" characters (ASCII 00H) to the specified destination. This is normally used with punch output devices; it has little use in the HP 82900A CP/M System since punch devices are not supported.

EOF: sends an "end-of-file" character (Control-Z or ASCII 1AH) to the specified destination. Since this operation is automatic on your computer, this source device has little use.

There is only one special destination device name.

PRN: is a special case of the "LST:" device which automatically expands ASCII "TAB" characters to eight columns, prints leading line numbers, and paginates the text every 60 lines. This is functionally equivalent to specifying the option [T8NP60].

PIP Options

Often during transfers between devices, and during special file transfers, you will want to provide PIP with additional instructions concerning the transfer. This can be done by specifying one or more PIP command options. Table 1-4 contains a summary of these options.

Table 1-4. PIP Options

Option	Function
B	"Block Mode" Transfer
D	Delete Character Positions
E	Echo File Transfer to Console
F	Strip "Form Feed" Characters
G	Access Files in Other User Group
H	Verify Hex File Format
I	Ignore Null Hex Records
L	Convert to Lowercase
N	Add Line Numbers
O	Object Code (non-ASCII) Files
P	Printer Pagination
Q	Quit Copy
R	Read System Files
S	Start Copy
T	Set Tab Width
U	Convert to Uppercase
V	Verify Copy
W	Write to Read/Only File
Z	Zero Parity Bit

B: Block Mode

Designates block mode transfer of data. Data is sent in blocks; the end of each block is designated by the ^S character (ASCII 53H). It is normally used with paper tape input; since your only system input is CON:, this option is of little use.

D: Delete Character Positions

This option takes the form "Dn", where n is a numeric value. PIP will automatically truncate any characters beyond column "n" on each line of source text. That is, after each carriage return character, PIP will transfer only n characters until the next carriage return is received.

This is primarily used to narrow down wide-lined files for listing on narrow printer devices.

E: Echo File Transfer to Console

This causes each line of text transferred to be "echoed" to the console. This permits you to view a file or device transfer at the console as it occurs.

F: Strip "Form Feed" Characters

This option causes PIP to remove any ASCII "form feed" characters as data is transferred from the source to the destination.

By using this option, a printer file can be prepared for nonprinter use.

G: Access Other User Group

This option, when followed by a numeric value between Ø and 15 decimal, will access the specified source file in the user number specified. This is how files can be transferred from one user number to another.

Note: Copying files to a different user area (if it is necessary to have a copy of PIP in the destination user area) can be accomplished using the following commands:

```
USER Ø            (Select user Ø)
DDT PIP.COM       (Load PIP.COM into memory and note
                  size s.)
GØ                (Warmboot and return to CCP)
USER n            (Log in desired user area)
SAVE s PIP.COM    (Save copy of PIP.COM in new user area.)
```

H: Verify Hex File Format

This option is used to verify that the source file is in proper Intel Hex format. This is intended for use with paper tape readers where no other error checking is available.

I: Ignore Null Hex Records

This option is also used during ".HEX" file transfers. It causes any null records, those which begin with ":ØØ", to be ignored during a data transfer.

L: Convert to Lowercase

This option causes PIP to translate each source character to its respective lowercase equivalent. No uppercase characters will be in the destination file.

N: Add line Numbers

This option is used to automatically append line numbes to the beginning of each line of text in the destination file. There are actually two forms of this option. "N" generates line number padded with leading blanks, separated from the text by a colon. When "N2" is specified, the line number is printed with leading zeros, and a space separates the line number from the text.

O: Object Code (non-ASCII) Files

This option is used when transferring non-ASCII files between devices. Since PIP expects a Control-Z to mark the end-of-file, some mechanism must be provided when a non-ASCII file may have an embedded Control-Z (1AH) as valid data. When this option is specified, PIP will continue to process data until a true end-of-file is encountered.

P: Printer Pagination

This option, when followed by a numeric value "n," specifies the number of printed lines per page. This can be used while routing files to a device, or for direct formatting of a disc file. When the PIP destination is a disc file, an ASCII form feed character is inserted after "n" lines. If "n" is not specified or equals 1, 6Ø lines are printed on each page.

Three blank lines are assumed at both the top and bottom of the page, so the proper value for n can be calculated for a given page size as follows:

Page size in North America is usually 11 inches, or 66 lines. Since PIP assumes 6 lines per inch, and allows three lines at the top and at the bottom of each page, a total of 60 lines are actually printed on each page. The proper value of "n" is therefore 60. However, many other countries use a standard page size of 70 lines. In order to specify this page size, with three blank lines at the top and the bottom, specify an "n" of 64.

Q: Quit Copy

This option allows you to specify a character string which will terminate the PIP transfer. The format is:

[Qstring^Z]

The ^Z here represents the Control-Z character. The first occurence of the character "string" will cause PIP to stop the copy. If it is not found, PIP will print the message "QUIT NOT FOUND".

Note that with this option, the actual "string" is copied before the transfer ends.

R: Read System Files

Files can be defined as "System Files" by setting the high-order bit of the second byte of the file type in the disc directory. This is normally done with the STAT command.

This flag makes a file "invisible" to the CCP (the Console Command Processor or CCP is discussed in section 2) and protects it from a PIP instructed transfer. However, by specifying the "R" option, PIP will read and transfer a file regardless of its "System File" status.

S: Start Copy

This option is similar to the Q option discussed earlier, except that S permits you to specify the string that will start the copy. The format is:

[Sstring^Z]

The characters starting with "string" are transferred to the destination file.

T: Set Tab Width

This option, followed by a numeric value "n," specifies the number of spaces to which each "tab" character (^I or ASCII 09H) is to be expanded. This is useful in printer listings and in creating formatted text files.

U: Convert to Uppercase

This option is similar to the "L" option, except that all lowercase characters from the source will be translated to uppercase. No lowercase letters will exist in the destination file.

V: Verify Copy

This option, valid only on disc file transfers, causes PIP to copy the file as it normally would. After each block of data is written to the destination file, it is read again and compared byte-by-byte to verify that no errors occurred during writing.

W: Write to Read-Only File

The first byte of the file extension can be set (via the high-order bit) as a read-only file. This is normally done using STAT as described later.

In copying to an existing file reference, PIP will normally delete the file and create a new copy. If that file is flagged as read-only, PIP will ask for a verification to overwrite that file.

Z: Zero Parity Bit

This option is used to "force" the high-order, or parity bit on each byte of data, to zero.

Examples

The PIP options are entered after the source device, and enclosed with brackets. Spaces between multiple PIP options are ignored. Rather than continue talking about the various options, here are some examples which illustrate how the options are specified as well as how several of them might be used.

Table 1-5. Example PIP File Transfers

PIP Command Input	Action
A:=B:*.*[D40]	Copy all files from disc "B" to "A" deleting all characters after column 40.
LST:=PRINT.TXT[SFIRST^Z]	Copy PRINT.TXT to LST: starting with "FIRST."
B:XYZ.BCD=E:BCD.TXT[LEN]	Copy BCD.TXT on disc "E" to XYZ.BCD on disc "B", in lowercase, echo all lines to console, and add line numbers.
A:=B:EXP.DAT[V]	Copy file B:EXP.DAT to disc "A" and verify. Control-Z signals EOF.

1.10 The STAT Command

The STAT program is a useful transient which provides statistics on CP/M disc files and logical and physical devices. STAT is also used to implement logical and physical device mapping. First we will discuss how STAT is used with disc files, and then how it is used with devices.

File Status

In using STAT to set file status, the general form is:

```
STAT file reference $attribute
```

Valid file references include any existing disc file references, with optional drive identifier or "wild card" specifiers. The ASCII "$" character separates the file from the attribute parameter. The valid attribute parameters are listed in table 1-6.

Table 1-6. Valid STAT File Attributes

Attribute	Meaning
R/O	Sets the specified file(s) to "read-only" status.
R/W	Set the specified file(s) to "read/write" status.
SYS	Set the specified file(s) to "system" status. Refer to text for details.
DIR	Resets the $SYS flag.

The $R/O attribute sets the specified file(s) to "read-only." This option actually changes the directory entry for the file(s). The high order bit on the first byte of the file type is set to signify the file is read-only (refer to section 2).

The $R/O option, since it is recorded in the file directory, is maintained through both warm (^C) and cold starts.

The $R/W attribute provides a mechanism to reset the directory bit which marks a file as read only. This is the only way to reset the effect of the $R/O parameter.

When the $SYS attribute is specified, the high-order bit of the second byte of the file type is set. This byte specifies whether a file should appear in the DIR command report. If $SYS is specified for a file, it will not appear when a directory command is entered via the CCP.

The $DIR parameter is used to reset the effect of the $SYS parameter. This is the default state for files.

Device Status

STAT can also be used to report on logical and physical device status. The general form used to produce these reports is:

```
STAT device parameter
```

The various options are illustrated in table 1-7. These options are explained in the text which follows.

Table 1-7. STAT Device Status Parameters

Parameter	Meaning
DEV:	Reports which physical devices are assigned to CP/M logical devices and summary of available STAT commands.
VAL:	Show which physical devices may be assigned to logical devices.
USR:	Produce a brief report showing which user numbers are active.
DSK:	Report on disc characteristics.
log: = phy:	Assigns the CP/M physical device "phy:" to the logical device "log:".
x: = R/O	Assigns a temporary read-only status to drive "x:".

By specifying DEV:, STAT reports the current logical to physical device assignments.

Example

```
A>STAT DEV:
CON: is CRT:
RDR: is TTY:
PUN: is TTY:
LST: is LPT:
```

The VAL: parameter illustrates the default STAT input line form, as well as indicating which physical device names are valid for the four CP/M logical devices CON:, RDR:, PUN:, and LST:. It also includes a brief report on the format of each other command discussed here.

Example

```
A>STAT VAL:

Temp R/O Disk: d:=R/O
Set Indicator: d:filename.typ $R/O $R/W $SYS $DIR
Disk Status  : DSK: d:DSK:
User Status  : USR:
Iobyte Assign:
CON: = TTY: CRT: BAT: UC1:
RDR: = TTY: PTR: UR1: UR2:
PUN: = TTY: PTP: UP1: UP2:
LST: = TTY: CRT: LPT: UL1:
```

Recall that all physical input devices use the keyboard for input, and that output data is lost for some of the unsupported output devices. Refer to table 1-3 PIP Device Names.

The USR: parameter produces a brief report which tells you which is the currently logged user number. It also reports on all user numbers which have active files on the selected disc.

Example

```
A>STAT USR:

Active User : Ø
Active Files: Ø
```

By selecting the DSK: parameter, the user can receive a complete report on disc statistics. If the drive identifier is included before "DSK:" (for example STAT A:DSK:), statistics for the specified drive are reported. When the drive identifier is omitted, all accessible drives are reported. The statistics include: the drive name, the drive capacity in both records and bytes, the maximum allowable number of directory entries, and physical drive characteristics such as records per extent and sectors per track.

Example

```
A>STAT B:DSK

   B: Drive Characteristics
1952: 128 Byte Record Capacity
 244: Kilobyte Drive Capacity
 128: 32 Byte Directory Entries
 128: Checked Directory Entries
 128: Records/ Extent
   8: Records/ Block
  32: Sectors/ Track
   3: Reserved Tracks
```

STAT permits the user to select a physical to logical device mapping by using the form "log: = phy:", where each field is a valid logical or physical device name. The valid logical to physical device maps are shown when the VAL: option is used. Valid LST: devices are summarized in table 1-8 below.

Table 1-8. Physical List Device Names

phy:	Actual Printer Device
TTY:	None. Output data is lost.
CRT:	CRT.
LPT:	System printer.
UL1:	None. Output data is lost.

Example

```
A>STAT LST:=CRT:        Sets list device as the CRT.
```

Finally, STAT allows you to specify read-only status for a disc drive. The R/O status remains in effect until a warm start (^C) or cold start (power-on).

Example

```
A>STAT B: = R/O         The "B" drive is specified
                        "read-only" until the next
                        warm or cold start.
```

Note: Any time you change discs for a previously accessed drive, the disc becomes read-only until the next warm or cold start.

1.11 Batch Job Processing

Because CP/M is optimized for terminal use, most applications are interactive in nature. For example, most accounting programs require operator input, and each order is unique. However, there are some tasks that are repetitive in nature, which run with little or no human intervention, and are executed frequently. Posting of daily invoices is an example of such a task.

Two facilities are available through CP/M which permit such "batch" jobs to be performed. The SUBMIT program allows you to execute a sequence of commands by storing them in a ".SUB" extension command file. The CP/M Text Editor (ED) or another editor is used to create the SUBMIT file. The CCP executes the SUBMIT commands as if they were typed into the keyboard, and executed in sequence.

The XSUB command provides, as a subset of SUBMIT, the option of including special instructions during a SUBMIT batch operation. XSUB allows the system to make decisions regarding the batch operation, as if you were present.

The remainder of this section will describe these two powerful utilities and illustrate how each might be used.

The SUBMIT Command

As mentioned, SUBMIT "batches" command line input from a ".SUB" extension file to the CCP. The SUBMIT command can also specify information to be included in the batch operation through the optional SUBMIT parameters.

The general form of the SUBMIT command is:

```
SUBMIT file name parameter 1, parameter 2 ...
```

To use SUBMIT, an ASCII text file with the file extension ".SUB" must be prepared. This file should contain all the CCP commands which make up the "job" to be accomplished.

For example, suppose you want to copy all your backup files from disc "A" to disc "B" and produce a listing of the new "B" disc file directory on the system printer. The following sequence of commands performs the operation.

Table 1-9. Copy Files Commands

Command	Action
PIP B:=A:*.BAK	Copies all backup files from disc "A" to disc "B".
STAT CON:=BAT:	Assigns the system printer as the physical output device for the logical "CON:" device. (Refer to table 1-3 for a summary of possible maps.)
DIR B:	Produces a directory listing for disc "B".
STAT CON: = CRT:	Assigns the CRT as the physical output device.

For our example, assume that these four lines of text have been entered into a file called "JOB1.SUB". This file could be created easily using the text editor transient, ED. To execute these statements, enter the SUBMIT command as follows.

```
A>SUBMIT JOB1
```

The SUBMIT command creates a temporary file named "$$$.SUB" which contains the commands listed in "JOB1.SUB". The commands are then executed one by one in the order they appear in the file. After execution, each line is purged from the "$$$.SUB" file.

The SUBMIT program must be on the currently logged disc. The "JOB1.SUB" file can be located on another drive, if the drive identifier is included with the file name. The temporary "$$$.SUB" file is created on the current drive, where the SUBMIT program resides.

The job defined above, while useful, is not as practical as it could be. For example, assume your source files are on disc "C" rather than the disc "A", and that you wish to copy backup files to disc "D". The SUBMIT command allows you this flexibility, by accepting input parameters with the SUBMIT command line. When used in this way, the SUBMIT command takes the following form:

```
SUBMIT file name a b c ...
```

This format allows you to include incomplete CP/M command lines in your ".SUB" file; the parameters a b c ... sequentially "fill-in" the missing information. The parameters may be file references, numbers, letters, or any other information you might wish to include in your ".SUB" file.

To use this feature, the ".SUB" file uses numbers preceded by a "$" (ASCII 24H) to indicate the place where SUBMIT parameters are input. A "$1" in the ".SUB" file command line is replaced by parameter "a" in the SUBMIT command, a "$2" is replaced by parameter "b," and so on.

Using the ED transient (refer to section 6 for information about ED), create the file JOB2.SUB, containing the following three lines:

```
PIP $2:=$1:*.BAK
STAT CON:=BAT:
DIR $2:
STAT CON: = CRT:
```

The $2 and $1 will be replaced by the second and first parameters after the ".SUB" file name, respectively. To execute a copy from disc "C" to disc "D", enter:

```
A>SUBMIT JOB2 C D
```

SUBMIT creates a file called "$$$.SUB" on the currently logged disc which contains the following commands:

```
PIP D:=C:*.BAK
STAT CON:=BAT:
DIR D:
```

If the current drive is "A", the "$$$.SUB" file will automatically execute, line by line. Each line is purged from the file after it is executed. In the event that some error causes SUBMIT to end prematurely, or if the currently logged drive is not "A:", the "$$$.SUB" file remains on the disc (is not purged) and will execute when the next warm boot (^C) occurs after this disc is placed in the "A" drive.

To halt a SUBMIT operation at any time, press [CONT], [BACK SPACE], or ^C. Pressing the key(s) once halts SUBMIT at the current "$$$.SUB" file line; remaining lines are not purged, and will execute at the next warm boot. To purge the remainder of the "$$$.SUB" file, and avoid this situation, press the key(s) a second time.

Note: By using a SUBMIT command as the last line of the ".SUB" file, you can chain together multiple SUBMIT operations. You cannot use SUBMIT in the middle of the ".SUB" file to nest a SUBMIT operation within another SUBMIT operation.

The XSUB Command

When XSUB appears in the first line of a ".SUB" file, the scope of the SUBMIT operation is expanded to include general input. Without the XSUB command line, SUBMIT interprets each line of the ".SUB" file as a valid CP/M command, or a disc-resident program. XSUB allows you to also include any characters, names, or instructions in the ".SUB" that are consistent with the current operation. The ".SUB" file is executed as if you were present, inputting commands and instructions from the keyboard.

Unlike other transient commands that are typed in after the CP/M prompt, XSUB appears only as the first line of a ".SUB" file. The XSUB command is used only in conjunction with the SUBMIT command, and can appear only as the first line in the ".SUB" file.

For example, assume that you wish to revise a series of source program files. For each file you want to replace the first line with the text "Revision: 2". A sample ".SUB" file is listed below which can be used to edit the source file.

The file, JOB3.SUB, contains the following lines:

```
XSUB                Loads XSUB.
ED $1.ASM           Loads ED; file to edited is specified in SUBMIT.
#A                  Append all lines.
B                   Start of buffer.
K                   Kill first line.
I Revision: 2       Insert new line
^Z                  End of insert.
E                   Exit ED.
```

Note the ^Z which terminates the insert. Using ED, you cannot store the control character in a SUBMIT file; use the up arrow character, ASCII 5EH, instead.

To use the above ".SUB" file to the revise the file called "MBK.ASM", enter the following SUBMIT command.

```
SUBMIT JOB3 MBK
```

The entire sequence will be executed to completion. The SUBMIT command could be repeated with different parameters, specifying a different source program to be edited.

When the last line has been executed, control returns to the CCP as with SUBMIT. However, XSUB remains in memory until it is specifically over-written by a cold boot or a disc system reset function call.

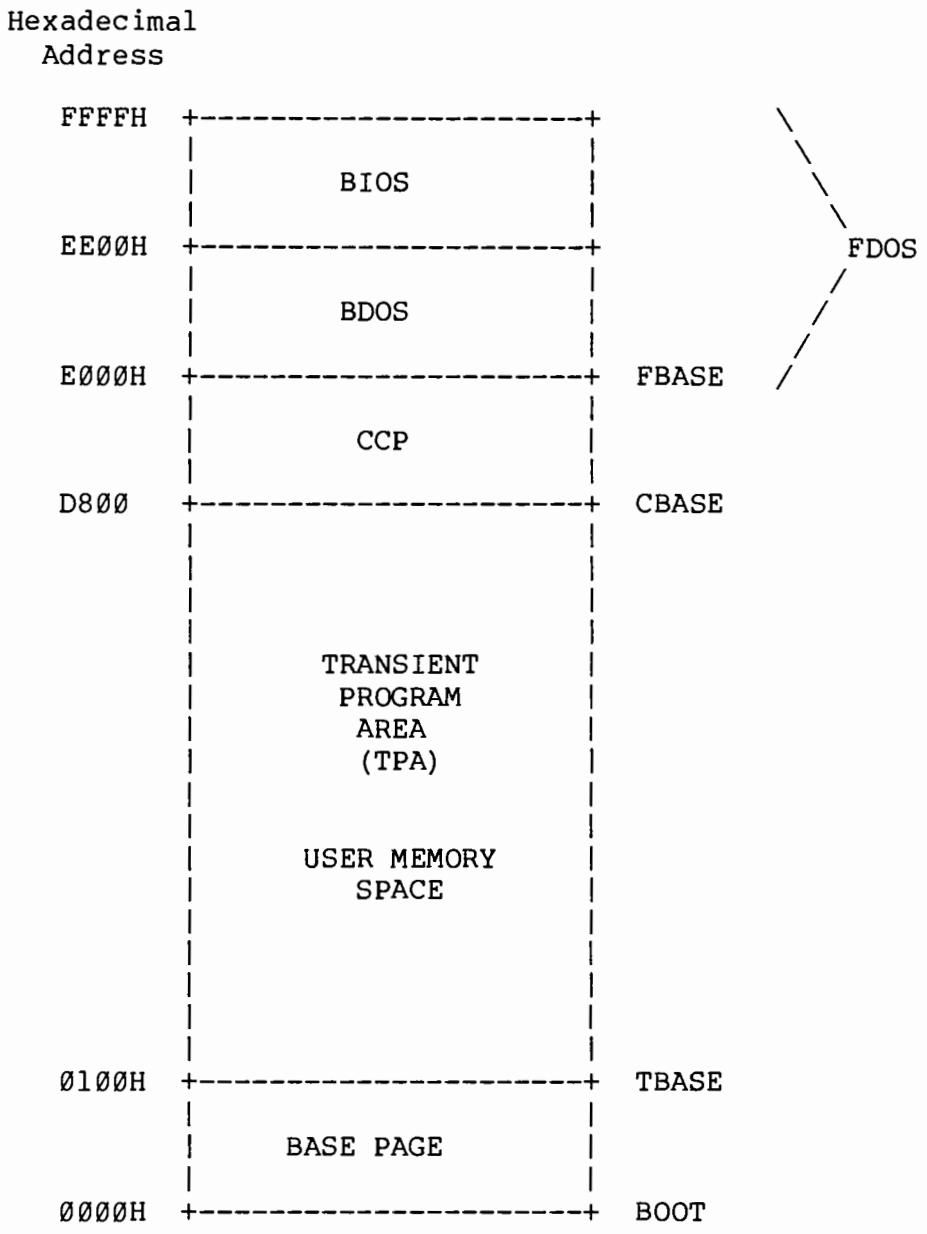
Section
II

CP/M INTERNAL ORGANIZATION

2.1 Memory Organization

The CP/M Operating System resides in approximately 8K bytes of high memory. This means that, for most user applications, up to 56K bytes of memory are available. Note that many interpretive languages use part of this memory as well, so memory available for your high-level source program may be somewhat less. Refer to the reference manual of the language you wish to use for more information.

Table 2-1. CP/M Memory Organization



There are actually four logical areas of memory in CP/M. Table 2-1 indicates how memory is allocated in a typical CP/M system application. The names along the right are the symbolic references given to the indicated addresses of memory. These will appear often through the remainder of the manual. Let's take a closer look at each area of memory and what task it performs.

2.2 FDOS

The highest area of memory is where the Fundamental Disc Operating System resides. FDOS is actually the heart of CP/M: all features of the operating system are implemented here. There are actually two areas which make up FDOS: the Basic I/O System (BIOS) and the Basic Disc Operating System (BDOS).

BIOS is that part of CP/M which is unique to each particular computer system. It contains all the routines necessary to communicate with physical I/O devices such as the disc and the CRT.

BDOS is generally fixed on every CP/M computer system, regardless of the manufacturer. BDOS implements all the system function calls that can be used by CP/M and your programs to perform input and output. It is BDOS that accesses routines in BIOS so your program can be written in a machine independent manner.

BDOS performs all calls to BIOS. Your program can access BDOS, pass one or more parameters, and request BDOS to perform I/O using BIOS. BDOS keeps track of where each routine in BIOS is located. Your program only needs to call BDOS. Fortunately, BDOS is accessed in the same manner regardless of machine manufacturer or CP/M revision. Your programs are, therefore, machine and revision independent.

2.3 CCP

The Console Command Program is the user interface to CP/M. If FDOS is the heart of CP/M, then CCP is its personality.

CCP is actually a program written to access the features and functions of FDOS. It is the CCP that provides you with the seven CP/M built-in commands. The CCP can access all the system function calls in FDOS (system function calls are discussed in section 4). The CCP simply accepts input and takes the appropriate action.

The CCP also loads transient programs into the Transient Program Area. Once it has done so, a transient program may expand into the area occupied by the CCP. Of course, the CCP must be reloaded into memory once the transient has completed. This is done automatically during a "warm start" (a warm start is implemented by typing [CTRL] [C], and is discussed in Introduction to the HP 82900A CP/M System).

2.4 TPA

The Transient Program Area is where the CP/M transient commands and user programs are loaded for execution, and is analogous to user memory in other computer systems. Note the distinction between transient commands, which are executed in the TPA, and built-in commands, which reside in the CCP.

Like the CCP, programs in the TPA may access all the system function calls in FDOS.

2.5 Base Page

Base page is the special name for the lowest page or 256 bytes of memory. On your computer, the base page occupies memory from ØØØØH through ØØFFH. Because the base page contains important information for CP/M (and for TPA programs), all CP/M programs should start at address Ø1ØØH. This preserves the integrity of the base page, and allows re-entry to the CCP when a transient program is completed.

Base page includes all memory locations from address ØØØØH through ØØFFH inclusive, for a total of 1ØØH bytes. Table 2-1 indicates which locations are reserved and which are used. The meaning of each section is given in the text which follows.

The jump to "Warm Start" command at address ØØØØH contains a 3-byte Z8Ø "JMP" instruction to the "Warm Start" entry in the Jump Vector Table in BDOS. The final result of this JMP is that BDOS performs a "reboot". For more information, refer to the section on BIOS Entry Points in section 5.

IOBYTE is stored at address ØØØ3H and determines the physical devices CP/M maps onto its logical devices. Refer to section 4 for more information on IOBYTE.

DRIVE ID at address ØØØ4H contains a numeric value representing the drive ID of the currently selected disc drive. Note that this information is informational only: to change mass storage device ID, use system function call 14 described in section 4.

Table 2-2. Base Page Organization

ADDRESS	DESCRIPTION
0000H - 0002H	Jump to Warm Start
0003H - 0003H	IOBYTE
0004H - 0004H	DRIVE ID
0005H - 0007H	Jump to BDOS
0008H - 0037H	Restart Locations
0038H - 003AH	Restart 7
003BH - 005BH	Reserved Scratch Locations
005CH - 007FH	Default FCB
0080F - 00FFH	Default DMA Buffer

The main entry into BDOS is stored in the form of a 3-byte "JMP" instruction stored at address 0005H. All entries to system function calls are made by performing a CALL to 0005H. This results in a jump to the lowest address of FDOS, and subsequent execution of the system function. Note that the address stored at 0006H and 0007H reflects the lowest address used by FDOS, so maximum user memory can be computed using this value.

The locations from 0008H through 0037H are used for restart interrupts available to the Z80 programmer. Interrupt location 6, or "Restart 6," addresses 0030H through 0037H should not be used since future versions of CP/M may require these locations.

"Restart 7" is interrupt location 7, addresses 0038H through 003AH. This restart is used by DDT and several other (unsupported) utilities in CP/M, but is not used by CP/M.

The reserved locations from 003BH through 005BH are used by CP/M for scratch areas and should not be used by your applications.

The default FCB is established by CCP at address 005CH.

CCP also establishes a default disc I/O buffer, called the DMA buffer, starting at address 0080H through the end of base page. These 128 bytes are where disc data is placed for I/O operations.

2.6 How the CCP Functions

As mentioned earlier, the CCP is the interface between the user and CP/M. CCP determines which disc is the current mass storage device, and prompts you with the disc letter identifier and a ">." It then accepts input from the console. Let's see what happens when you type a command and press the [END LINE] key.

First, the CCP allocates a section of base page to store the entire command line you entered. It also allocates part of the base page as a file control block (FCB) in case you need to access data files. Refer to section 3 for more information about base page and FCBs. All characters are shifted to upper case.

The CCP searches your input line for one of the built-in commands as the first nonblank characters. If one of the commands listed in table 1-1 is found, CCP executes that command. If no built-in command is found on the line, CCP proceeds to the next step.

The CCP will now search for a disc identifier as the first field on the line. If a disc identifier is found (for example, B:), subsequent checks will be made on the specified disc. Otherwise, all checks will be completed on the currently selected disc.

CCP now assumes the input line specifies the name of a file of file extension "COM" (refer to Transient Programs below). The specified (or default) disc is searched and, if such a program is found, the program is loaded. The CCP then performs a CALL to 0100H to begin execution.

If the CCP cannot recognize either a built-in command or a transient program name on your line, it will display the command it interpreted followed by a question mark. You will then be prompted for input once again.

2.7 Transient Programs

By now you might have guessed that any compiled or assembled program you might write becomes a transient command to CP/M. Your program is named with a file extension of "COM" by the LOAD program; the CCP recognizes it as an image of memory containing executable code; and your program has access to all the internal features of CP/M.

Your program, which begins at 0100H, occupies the TPA from TBASE through CBASE - 1. In fact, your program can expand beyond CBASE and overwrite the CCP. Memory up to FBASE - 1 can be used by an application and still leave FDOS intact. Once a program begins to overwrite FDOS, however, it can no longer access system functions available through FDOS.

FDOS is sometimes known as BDOS because it represents the primary entry into BDOS. Locations at and above FDOS should be considered sacred by your application. If you destroy any part of FDOS, your program will not be able to reboot CP/M, and the user will have to do so on his own.

CP/M FILE SYSTEM ORGANIZATION

3.1 File References

Each CP/M disc file is identified and accessed by a file reference. A fully qualified file reference is made up of three parts: a disc drive identifier, a file name, and a file extension.

The disc drive identifier is optional, and in fact will vary depending upon which drive contains the disc on which the file resides. If no drive identifier is included, CP/M assumes the identifier of the currently selected mass storage device.

The drive identifier is a single alphabetic character followed by a colon (:). The letter, which must be between "A" and "P" inclusive, corresponds to the physical disc drive. Refer to the Introduction to the HP 82900A CP/M System for information about how CP/M assigns the drive identifier.

The file name typically serves to describe the contents of a file. It is required, and may contain from one to eight alphanumeric characters. The only special characters which are not allowed in the file name are:

<	>	.	,	;	:	=	?	*	[	]

All other printing characters are permitted.

The file extension is an optional parameter which contains one to three alphanumeric characters. The restrictions on file extension characters are the same as for file names. Sometimes known as a file type, the file extension is used to identify what kind of data is contained in the file.

File names and file extensions are generally assigned in an arbitrary manner. However, several extensions have special meanings to CP/M and its commands.

These are presented in table 3-1.

Table 3-1. File Extensions

ASM	ASM Source Code	PRN	Printer Listing File
BAK	Back-Up ED File	TXT	ASCII Data File
BAS	BASIC Source File	$$$	Temporary File
COM	Transient Command		
DAT	ASCII Data File		
HEX	ASM Output File		

3.2 File Structure

Each file can be thought of as a sequence of up to 65535 records of 128 bytes each. By multiplying this out, you can see the maximum file size is 8 million bytes of data. However, there are some restrictions to this maximum. For example, a file must reside completely on a single disc.

When you create a file, you need not specify its maximum size. This is because CP/M reserves disc space incrementally, only as it is actually needed. Of course, this allocation on demand can mean your file may be located in many different areas of the disc.

Fortunately, CP/M manages this potential difficulty so that, from a programmatic point of view, every file can be treated as logically contiguous. This is done by means of several pointers maintained by the operating system.

3.3 File Access

Data files can be accessed in either random or sequential fashion. There is no rigid distinction between the two, but keep in mind that random access files must have a fixed logical record length. Sequential files can, of course, have variable length logical records. On the other hand, every record within a random access file can be accessed directly, without any need to access previous records. Both random and sequential data is transferred to and from the disc in 128-byte physical records.

CP/M will manage any possible segmentation of data files as mentioned earlier.

ASCII files are treated as a sequence of characters, where each logical record is terminated by a carriage return line feed sequence (CR/LF). This means that each 128-byte physical record from a disc can have one or more logical records imbedded.

For ASCII data, such as text files, the end-of-file (EOF) is marked by a control-Z character or a true end-of-file from CP/M. The latter is the case only when a text file actually ends on one of the 128-byte physical record boundaries. Binary files end only on physical record boundaries.

3.4 File System Overview

The user area of a CP/M disc is organized into physical records of 128 bytes each. CP/M keeps track of which sections of the disc are actually in use by program and data files. These sections, or groups, are allocated in segments of 1K bytes on the 5-1/4-inch disc and 4K bytes on the 8-inch disc. This represents the minimum file size on each respective disc.

When a file is created, an entry is made for that file in the disc directory. When the first record is written to that file, CP/M allocates one group of free space and enters that group number in the directory of the file. Note that changes are posted to the disc only when the file is subsequently closed or automatically whenever a group boundary is crossed.

As a file grows beyond one group, additional groups are added in segments of 1K or 4K respectively. This continues to happen until a total of 16K bytes are allocated. This represents 16 groups on the 5-1/4-inch disc, or 4 groups on the 8-inch disc. In either case, this cluster of 16K bytes is known as an extent and represents 128 physical records of 128 bytes each.

On the 5-1/4-inch disc, each directory entry can allocate up to 16 groups, so each directory entry represents an extent. Up to 32 directory entries or extents can exist for any particular file, so the maximum file size is:

$$\frac{16\text{K bytes}}{\text{extent}} \times \frac{32\text{ extents}}{\text{file}} = \frac{512\text{K bytes}}{\text{file}}$$

Of course, the maximum disc capacity is 256K bytes, so you would run out of disc before you run out of extents. In this case, the maximum can only be theoretical.

On the 8-inch disc, each directory entry has room for 8 groups of 4K bytes: this means that each directory entry can handle two extents for a total of 32K bytes. While the extent count cannot grow any larger than 31 here also, extra system bytes allow up to 64 directory entries permitting a maximum file size of:

$$\frac{32K\ bytes}{entry} \times \frac{64\ entries}{file} \times \frac{2048K\ bytes}{file}$$

Again in this case, the theoretical limit is larger than the actual disc capacity of 1.2M bytes.

You will see more about how these extents are managed by CP/M in the next few sections. First, let's see how dynamic information on the file is maintained by the operating system so your data is secure.

3.5 File Control Block

As with most other operating systems, CP/M requires the use of a File Control Block or FCB for each file that is open. The FCB is used by CP/M to keep track of the file name, the current extent, the current record number, and all other dynamic information for that file.

Most system function calls after function number 15 require the address of a FCB to be passed in the Z80 DE register pair. When a transient program is loaded, a default FCB is reserved in base page at address 005CH. Most programs which require use of a file will use this default FCB because of its convenience.

File I/O requires a location to use as a holding area, or buffer. CP/M establishes a default file buffer, 128 bytes in length, at address 0080H. This buffer location may be changed using function number 26, "Set DMA Address" but most programs are able to use the default. For more information, refer to the discussion of function 26 in section 4.

Regardless of its location in memory, the format of an FCB is fixed. The FCB for a sequential access file is always 33 bytes in length. Random access files require an additional 3 bytes appended to the sequential FCB for a total of 36 bytes. The format of such a FCB is illustrated in table 3-2, while discussion of the actual fields and their meanings follow the table.

Now we will look at a more detailed explanation of the meaning of each of the fields illustrated below. Note that bytes from 32 through 35 are not part of the disc directory entry: they exist only while an extent is actually in memory as a File Control Block.

Table 3-2. File Control Block

Dec	Hex	Name	Description
00	00	dr	Drive Code/User Number
01	01	f1	File Name: 8 bytes in length
02	02	f2	
03	03	f3	
04	04	f4	
05	05	f5	
06	06	f6	
07	07	f7	
08	08	f8	
09	09	t1	File Extension: 3 bytes in
10	0A	t2	length
11	0B	t3	
12	0C	ex	Extent Number
13	0D	s1	Reserved for system use
14	0E	s2	(2 bytes)
15	0F	rc	Extent Record Count
16	10	d0	Disc Group Allocation Blocks
17	11	d1	
18	12	d2	
19	13	d3	
20	14	d4	
21	15	d5	
22	16	d6	
23	17	d7	
24	18	d8	
25	19	d9	
26	1A	d10	
27	1B	d11	
28	1C	d12	
29	1D	d13	
30	1E	d14	
31	1F	d15	
32	20	cr	Current Record In Extent
33	21	r0	Record Number Low Byte
34	22	r1	Record Number Mid Byte
35	23	r2	Record Number Overflow

The fields are described in the next several pages. Remember that the last four bytes ("cr" through "r2") exist only in the memory copy of the FCB.

BYTE Ø : dr

This byte, known as the drive code, specifies the disc drive that contains the file. The possible values for "dr" are:

```
Ø  -> File on default drive.
1  -> File on drive "A"
2  -> File on drive "B"
. . .
16 -> File on drive "P"
```

This field will not change the currently selected drive; it will simply access the specified drive to access this file.

BYTES 1 -> 8 : f1 -> f8

This sequence of eight bytes should contain the ASCII file name. Remember that, at this system level, upper case characters are not automatically generated. If the file name is to be upper case, the proper ASCII values must be loaded.

BYTES 9 -> 11 : t1 -> t3

These three bytes contain the optional file extension. As with the name, the actual ASCII values must be loaded as no automatic upper casing takes place. The high order bit on the t1 and t2 bytes, normally zero for ASCII characters, is used to further classify this file. Specifically:

t1 high bit set to 1 defines the file as "read-only."

t2 high bit set defines the file as a "system file." This means the file will not appear in a directory listing.

BYTE 12 : ex

This byte determines the current extent being accessed, and can take values from Ø through 31 decimal.

BYTES 13 -> 14 : s1 - s2

Flags reserved for use by CP/M when the file is actually open.

BYTE 15 : rc

This field is the record count. The value reflects the actual number of physical 128-byte records referenced by the current extent. The maximum value of rc is 128 decimal or 8Ø hex.

BYTES 16 -> 31 : d1 -> d16

This group of bytes is reserved for system usage. The values that are stored here reflect the disc group address of allocated groups. As the file grows, additional groups are added to the list. On the 5-1/4-inch disc, this section is organized into 16 single-byte entries; on the 8-inch disc, the section is eight entries of two bytes each.

The following four bytes are not part of the disc directory entry: they appear only in the memory FCB.

BYTE 32 : cr (FCB Only)

This one-byte pointer contains the current record number for input or output. This value reflects the physical record within the current extent, and is between 1 and 128.

BYTES 32 -> 35 : rØ -> r2 (FCB Only)

These three bytes, reserved as part of the default FCB at ØØ5CH, are only used during random access disc I/O. Bytes rØ and r1 contain a 16-bit integer value which is the record number of the desired physical record. Byte r2 is an overflow byte. Byte rØ contains the low-order 8 bits, while byte r1 contains the high order 8 bits or the actual record number.

You will note that, with a maximum of 32 extents and only 128 records in each extent, it would be impossible to attain a file size larger than 512K bytes. CP/M manages this apparent conflict automatically for you by using some of the bits available in the various reserved bytes in the FCB. Those bits are used to keep track of which extent within a particular directory entry is being referenced by "rc," "cr," and "ex."

Each file being accessed through CP/M must have a corresponding FCB which provides the file name and allocation information for all subsequent file operations. When you first access a file, usually via an "open" or "make" function call, it is your responsibility to fill the lower 16 bytes of the FCB and the "cr" field with appropriate information. Normally, the first 12 bytes are set to ASCII character values for the disc, file name and file extension, with the remainder of the fields set to zero.

3.6 Disc Directory

The directory of each disc device contains an FCB for every file residing on the disc. Each directory entry has all the information found in the first 32 bytes of the FCB described above. The only difference is the first byte, "dr." It's clear that the drive code cannot be determined until a disc is actually mounted, so "dr" has no meaning in the disc directory. The byte is therefore available for other use: the User Number of the file creator is stored in that byte.

Before you can access a file, you must copy the FCB into memory. This is done for you by an "open" function call. CP/M manages the memory copy of the FCB while you have the file open, and updates the disc version of the FCB only when you close the file. This implies two points: first, remember to close your files so the disc FCB reflects the actual file structure; and second, whenever the memory FCB is destroyed before the disc FCB can be updated (for example, by a power failure), you should expect to recover your file from a backup and start again.

3.7 Default FCBs and the CCP

As you know, a user may specify one or two file names in many of the CCP commands as well as in many of the transient commands. The CCP sets up one default FCB at address ØØ5CH and fills the first 12 bytes from the first file name on the command line. This FCB may be used without any further action by your program. If a second file name is specified, the CCP places that information into the bytes designated as dØ through d16 in the default FCB. Before an application can use that information, these bytes must be moved into another valid FCB area according to the guidelines given above for FCB layout.

If no file names are specified on the command line, all fields from ØØ5DH through ØØ6DH contain blanks. In all cases, the CCP will translate all characters to ASCII upper case prior to loading any information into the buffers.

As an added convenience, the default file buffer area at address ØØ8ØH contains the tail end of the command line typed by the user. That is, all characters following the command or transient program name are loaded into the default buffer. The first byte, ØØ8ØH, contains the byte count of the line. Starting at byte ØØ81H you will find the parameters typed after the command.

3.8 FCB Example

As an example of the information presented above, let's take a look at the contents of the default FCB and the DMA buffer after a CCP command line is accepted.

In our example, we will use PIP to copy the file "OLD.TXT" from the currently selected disc to a file called "NEW.TXT" on the disc in drive "B." The command line is:

```
A>PIP B:NEW.TXT=OLD.TXT
```

After the [END LINE] key is pressed, CCP sets up a default FCB at ØØ5CH. The first 32 bytes are illustrated in table 3-3. Note that the first 16 bytes of each FCB are included in the buffer as established by the CCP. In this case, PIP must be responsible for moving the second 16 bytes, the FCB area for "OLD," into a second FCB area so both files can be accessed.

The default DMA buffer at address ØØ8ØH will contain the "tail" of the command line which invoked PIP. Table 3-4 illustrates how that buffer would appear for the example above.

In this "tail" buffer, CP/M has not acted to upper case any bytes. However, the CCP did upper case the file name characters when building the FCB from this buffer.

Further, notice that any delimiter characters entered on the command line are available in the tail buffer, but will be removed from the FCB.

In the example above, PIP will access the FCB at address ØØ5CH to establish the FCBs for each file. One FCB can remain at that address, while the second must be established in the TPA. PIP can also access the command line stored at address ØØ8ØH to determine whether any additional parameters were entered.

Just as PIP has access to these facilities, so do any assembly language programs you write.

Table 3-3. Example FCB Set-Up: PIP

Hex	Byte	Chars	Description
ØØ	dr	Ø2H	Destination Drive Code
Ø1	f1	N	Destination File Name: "NEW"
Ø2	f2	E	
Ø3	f3	W	
Ø4	f4		
Ø5	f5		
Ø6	f6		
Ø7	f7		
Ø8	f8		
Ø9	t1	T	Destination File Extension
ØA	t2	X	
ØB	t3	T	
ØC	ex	ØØH	Destination Extent Number
ØD	s1	ØØH	Reserved for system use
ØE	s2	ØØH	
ØF	rc	ØØH	Destination Extent Rec Count
1Ø	dr	ØØH	Source Drive Code (default)
11	f1	O	Source File Name: "OLD"
12	f2	L	
13	f3	D	
14	f4		
15	f5		
16	f6		
17	f7		
18	f8		
19	t1	T	Source File Extension
1A	t2	X	
1B	t3	T	
1C	ex	1ØH	Source Extent Number
1D	s1	1ØH	Reserved for System Use
1E	s2	1ØH	
1F	rc	1ØH	Source Extent Record Count

Table 3-4. Command Line Buffer Format

Addr	Cnts	Comments
80H	12H	Length of text
81H	20H	Space after "pip"
82H	b	Drive Code
83H	:	
84H	n	Destination File Name
85H	e	
86H	w	
87H	.	
88H	t	
89H	x	
8AH	t	
8BH	=	Assignment Symbol
8CH	o	Source File Name
8DH	l	
8EH	d	
8FH	.	
90H	t	
91H	x	
92H	t	
93H	20H	Trailing spaces(s)

Section
IV

SYSTEM FUNCTION CALLS

4.1 Introduction

You've seen the general structure of CP/M and what features are available. At this point, we will take a detailed look at how you can utilize all the features of CP/M on your computer under program control.

All of the features of the operating system are implemented by means of system function calls. These are subroutines called with a specific set of parameters corresponding to the data for the subprogram. Values are returned to the calling program once the subroutine terminates. The Z8Ø registers are used to pass data to, and receive data from, the system function routines.

The facilities available for access by transient programs fall into three general categories: simple device I/O, disc file I/O, and system control. The standard CP/M system function calls are summarized in table 4-1.

The number listed in table 4-1 under the CALL columns is known as the system function number. This number corresponds to the decimal value that is loaded into the C register to select a particular call.

In addition to the standard system function calls, or SFCs, Hewlett-Packard has added extended calls which are useful in adapting the unique features of your computer to CP/M. These calls are discussed later in this section.

Table 4-1. System Function Call Summary

CALL	OPERATION	CALL	OPERATION
Ø	SYSTEM RESET		
1	CONSOLE INPUT	21	WRITE SEQUENTIAL
2	CONSOLE OUTPUT	22	CREATE FILE
3	READER INPUT	23	RENAME FILE
4	PUNCH OUTPUT	24	DISC LOGIN VECTOR
5	LIST OUTPUT	25	CURRENT DISC ID
6	DIRECT CONSOLE I/O	26	SET DMA ADDRESS
7	GET IOBYTE	27	GET ADDR (ALLOC)
8	SET IOBYTE	28	WRITE PROTECT
9	PRINT STRING	29	GET R/O VECTOR
1Ø	READ CONSOLE BUFFER	3Ø	FILE ATTRIBUTES
11	CONSOLE STATUS	31	PARAMETER ADDRESS
12	CP/M VERSION #	32	GET/SET USER NUM
13	RESET DISC SYSTEM	33	READ RANDOM
14	SELECT DISC	34	WRITE RANDOM
15	OPEN FILE	35	COMPUTE FILE SIZE
16	CLOSE FILE	36	SET RANDOM RECORD
17	SEARCH FOR FIRST	37	RESET DRIVE
18	SEARCH FOR NEXT	38	UNUSED
19	DELETE FILE	39	UNUSED
2Ø	READ SEQUENTIAL	4Ø	WRITE W/ ZERO FILL

4.2 Accessing Function Calls

As mentioned above, access to all function calls is accomplished by passing a function number, and possibly additional information, to BDOS. The entry point for all CP/M system function calls is address ØØØ5H. In general, the function number is passed in register C with the address of additional information contained in registers D and E.

Single-byte values are returned in register A, with double byte values returned in registers H and L. In all cases, register A will contain the same value as register L, while register B will be the same as register H.

Here is a sample program that illustrates how function calls are implemented. This short program reads one console character at a time, echoing that character to the console. It continues to accept input until an asterisk is encountered, whereupon it returns control to CCP.

Table 4-2. Sample System Function Call

```
+-------------------------------------------------------------+
|    BDCS     EQU       0005H       ;BDOS ENTRY ADDR 0005H    |
|    CHARIN   EQU       1           ;READ CHAR FUNCTION       |
|             ORG       0100H       ;TPA STARTS HERE          |
|    NEXTC    MVI       C, CHARIN   ;FUNCTION # INTO C REG    |
|             CALL      BDOS        ;CALL DOS FUNCTIONS       |
|             CPI       '*'         ;IS CHAR A '*'?           |
|             JNZ       NEXTC       ;NO? GET ANOTHER CHAR     |
|             JMP       0000H       ;YES? WARM-BOOT SYSTEM    |
|             END                   ;END ASSEMBLY             |
+-------------------------------------------------------------+
```

Before CCP transfers control to a transient program, it builds an eight-level stack with the CCP return address pushed on the top of the stack. BDOS, during system function calls, creates its own stack, so overflow problems do not usually occur. DO NOT ASSUME THAT THE CONTENTS OF ANY REGISTER WILL REMAIN INTACT DURING ANY SYSTEM FUNCTION CALL!

Most programs terminate by performing a warm boot, which will reload the CCP and reinitialize the CCP stack. This is done by executing either a CALL or a JMP to 0000H. Because of this, stack management is not as important in CP/M as it is in many other systems.

Now we will look at each of the standard CP/M function calls in detail. The discussion of each SFC will begin on a new page, so that you can quickly locate a particular function.

FUNCTION Ø: SYSTEM RESET

Entry Parameters:
Register C: ØØH

The System Reset function is equivalent to a warm boot in that program execution ends and the previously current disc is selected as the mass storage device. The CCP is loaded, and begins execution.

This call is just one way to return control to CP/M.

EXAMPLE: System Reset

```
BDOS    EQU     0005H       ;CP/M ENTRY POINT
        . . .
        MVI     C,00H       ;MOVE SFC 0 INTO C
        CALL    BDOS        ;PERFORM SYSTEM RESET
        . . .
```

FUNCTION 1: CONSOLE INPUT

Entry Parameters:
    Register C: 01H

Returned Value:
    Register A: ASCII Character

The Console Input function reads the next console character into register A. Both alphanumeric characters and non-displayable control characters are echoed to the console. Special characters function as they normally would when entered as input to the CCP. Specifically, CTRL-I is expanded to a 8-column tab, and CTRL-P and CTRL-S function as printer echo and scroll controls respectively.

The FDOS does not return to the calling program until a character has been typed. If a character is not ready, execution suspends until a character becomes available. If you wish to avoid this wait state, it is suggested that you use SFC 11, Console Status, to verify that a character is ready. Then you can use this function to actually read the character.

ASCII data requires only seven bits, and this function returns only the least significant seven bits. BDOS specifically clears the high-order bit. Normally, this presents no problem whatsoever. However, custom applications could use the special keys which produce full eight-bit codes.

Executing this function is the only way to clear the console status.

EXAMPLE: Console Input

```
BDOS    EQU    0005H        ;MAIN CP/M ENTRY POINT
        . . .
        MVI    C,01H        ;MOVE SFC INTO C
        CALL   BDOS         ;INPUT AND ECHO CHARACTER
        . . .               ;CHARACTER NOW IN A
```

FUNCTION 2: CONSOLE OUTPUT

Entry Parameters:
    Register C: Ø2H
    Register E: ASCII Character

This function call causes the ASCII character value stored in register E to be output to the CP/M console device.

As with SFC 1, characters with special meanings in CP/M are functional. For example, tab characters (CTRL-I) are expanded into eight spaces. Start/stop scroll (CTRL-S/CTRL-Q) and printer echo (CTRL-P) retain their usual meaning.

EXAMPLE: Output @ to display

```
BDOS    EQU    ØØØ5H        ;MAIN CP/M ENTRY POINT
        . . .
        MVI    E, '@'       ;MOVE ASCII @ INTO E
        MVI    C, Ø2H       ;MOVE SFC 2 INTO C
        CALL   BDOS         ;OUTPUT CHARACTER
        . . .               ;@ IS ON CONSOLE
```

FUNCTION 3: READER INPUT

Entry Parameters:
    Register C: Ø3H

Returned Value:
    Register A: ASCII Character

This call is similar to SFC 1, Console Input, except that the byte of data is accepted from the currently defined CP/M reader device.

As with SFC 1, SFC 3 will not return to your program until a byte is available, so your program may "hang" in execution.

The reader device is always mapped to tthe keyboard.

EXAMPLE: Reader Input

```
BDOS    EQU    ØØØ5H        ;MAIN CP/M ENTRY POINT
        . . .
        MVI    C,Ø3H        ;MOVE SFC 3 INTO C
        CALL   BDOS         ;READ CHARACTER
        . . .               ;CHARACTER NOW IN A
```

FUNCTION 4: PUNCH OUTPUT

Entry Parameters:
    Register C: Ø4H
    Register E: ASCII Character

This call results in the ASCII character stored in register E being sent to the CP/M Punch device for output.

Data sent to the PUNCH device is always lost because no reader devices are supported.

EXAMPLE: Output @ to punch device

```
BDOS    EQU   ØØØ5H       ;MAIN CP/M ENTRY POINT
        . . .
        MVI   E,'@'       ;MOVE ASCII @ INTO E
        MVI   C,Ø4H       ;MOVE SFC 4 INTO C
        CALL  BDOS        ;OUTPUT CHAR TO PUNCH
        . . .             ;CHARACTER AT PUNCH
```

FUNCTION 5: LIST OUTPUT

Entry Parameters:
    Register C: Ø5H
    Register E: ASCII Character

This call causes the ASCII character stored in register E to be output to the currently defined list device.

EXAMPLE: Output @ to IOBYTE defined list device

```
BDOS    EQU    ØØØ5H        ;MAIN CP/M ENTRY POINT
        . . .
        MVI    E,'@'        ;MOVE ASCII @ INTO E
        MVI    C,Ø5H        ;MOVE SFC 5 INTO C
        CALL   BDOS         ;PRINT CHARACTER TO LIST
        . . .               ;CHARACTER IS PRINTED
```

FUNCTION 6: DIRECT CONSOLE I/O

Entry Parameters:
    Register C: 06H
    Register E: OFFH (input) or
                char (output)

Returned Value:
    Register A: char or status
                (no value)

This function implements a feature which permits I/O from and to the console device without any special treatment of special CP/M characters. That is, a tab character (CTRL-I) is accepted as a single byte, and is not expanded to eight spaces. This is the case for start/stop scroll and printer echo as well.

To output a character, load register E with the desired ASCII character. On input, the character accepted is not automatically echoed to the console as it is with SFC 1. Also, note that the console status (SFC 11) is not reset until SFC 1 executes. If you use this function to accept input, use this call for console status to assure consistent information.

To accept input, load register E with OFFH. If a character is available for input, it is returned in register A. If no byte is ready, register A contains OOH upon return.

EXAMPLE: Read and echo a character on the console

```
BDOS    EQU     0005H       ;MAIN CP/M ENTRY POINT
        . . .
INPT    MVI     E,OFFH      ;MOVE INPUT CODE INTO E
        MVI     C,06H       ;MOVE SFC 6 INTO C
        CALL    BDOS        ;CHECK CONSOLE FOR INPUT
        CPI     OOH         ;IS CHARACTER READY?
        JZ      INPT        ;NO - READ AGAIN
;                            CHARACTER NOW IN A
        MOV     E,A         ;MOVE CHARACTER INTO E
        MVI     C,06H       ;MOVE SFC 6 INTO C
        CALL    BDOS        ;ECHO CHARACTER AT CONSOLE
        . . .               ;CONTINUE
```

FUNCTION 7: GET IOBYTE

Entry Parameters:
Register C: 07H

Returned Value:
Register A: IOBYTE Value

This function returns the current value of IOBYTE, located at address 0003H.

IOBYTE is used to select the active list device on the HP-87.

EXAMPLE: Read current IOBYTE value

```
BDOS    EQU    0005H        ;MAIN CP/M ENTRY POINT
        . . .
        MVI    C,07H        ;MOVE SFC 7 INTO C
        CALL   BDOS         ;READ IOBYTE
        . . .               ;IOBYTE NOW IN A
```

FUNCTION 8: SET IOBYTE

Entry Parameters:
Register C: 08H
Register E: IOBYTE Value

This function is used to set the contents of IOBYTE to the value passed in register E.

IOBYTE is used to select the active list device on the HP-87.

EXAMPLE: Load HP-IB printer code in IOBYTE

```
BDOS    EQU    0005H        ;MAIN CP/M ENTRY POINT
        . . .
        MVI    E,COH        ;SELECT HP-IB IN IOBYTE
        MVI    C,08H        ;MOVE SFC 8 INTO C
        CALL   BDOS         ;WRITE TO IOBYTE
        . . .               ;IOBYTE NOW UPDATED
```

FUNCTION 9: PRINT STRING

Entry Parameters:
Register C: 09H
Registers DE: String Address

This function is used to print the contents of memory to the CP/M console device. It is a very useful function for simple output for prompts and information.

The buffer to be printed can be of any length, limited only by the amount of free memory. Regardless of the length, the printed string will terminate when the first ASCII "$" is encountered.

To use this call, the DE register pair must contain the address of the start of the buffer. Memory locations starting at that point are considered to be ASCII characters, and all subsequent memory locations are printed until a location with an ASCII "$" is encountered. The "$" is not printed to the console.

To print an ASCII "$" character, some other output function call must be used.

Data printed using this call is processed for special characters. Tabs, start/stop scroll, and printer echo are all treated normally as with SFC 2 above.

EXAMPLE: Print 'HELLO THERE' to console

```
BDOS    EQU    0005H        ;MAIN CP/M ENTRY POINT
MESG    DB     'HELLO THERE$'
        . . .               ;NOTE MESG ENDS WITH $
        LXI    D,MESG       ;ADDR OF MESG INTO DE
        MVI    C,09H        ;MOVE SFC 9 INTO C
        CALL   BDOS         ;PRINT STRING 'MESG'
        . . .               ;MESSAGE NOW AT CONSOLE
```

FUNCTION 1Ø: READ CONSOLE BUFFER

Entry Parameters:
    Register C: ØAH
    Registers DE: Buffer Address

Returned Value:
    Console Characters in Buffer

This function accepts a line of input from the CP/M console device. This data is not available to the calling program until the [RETURN] key is pressed or until the input buffer overflows.

To access this function, an application must first set up a buffer space into which to accept input. The DE register pair should contain the address of the start of the buffer. The buffer itself should take the form:

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

In this case, "mx" is the maximum number of characters to accept and should be set by your application. The "nc" parameter is returned by CP/M as the actual number of ASCII characters accepted, and the characters are stored from "c1" for "nc" characters.

This function does not return any data from CP/M until the [RETURN] key is pressed (or more than "mx" characters are typed). For this reason, all the line editing features of CP/M remain valid. These include:

DEL	erases the last character from the buffer and echoes the erased character to the console.
CTRL-C	causes a warm boot when typed as the first character on a line.
CTRL-E	performs a physical end-of-line without sending data to CP/M.
CTRL-H	backspaces one character, erasing it from both the screen and the input buffer.
CTRL-J	performs a line feed, and is functionally identical to [RETURN]. Data is returned to the calling program.
CTRL-M	performs a carriage return. Data is returned to the calling program.
CTRL-R	performs a CR/LF and re-types the current line. It is used to re-display a clean line.
CTRL-U	performs a CR/LF, erases all characters in the input buffer, and returns to input mode.
CTRL-X	performs multiple backspaces until all characters on the current line are erased.

Note also that those functions that return the carriage to the leftmost position (e.g., CTRL-X) do so only to the column position where the prompt ended. This convention makes operator data input and line correction more legible.

EXAMPLE: Accept console input buffer

```
BDOS    EQU    0005H       ;MAIN CP/M ENTRY POINT
BUF     DS     82          ;SAVE 82 BYTE BUFFER
MX      EQU    BUF+0       ;MX IS FIRST BYTE OF BUF
NC      EQU    BUF+1       ;NC IS NEXT BYTE OF BUF
INBUF   EQU    BUF+2       ;CONSOLE INPUT LINE
        . . .              ;
        MVI    A,80        ;MOVE 80 DECIMAL INTO A
        STA    MX          ;STORE A REGISTER IN MX
        LXI    D,BUF       ;MOVE BUF ADDR INTO DE
        MVI    C,0AH       ;MOVE SFC 10 INTO C
        CALL   BDOS        ;ACCEPT CONSOLE INPUT
        . . .              ;NC CONTAINS CHAR CNT
        . . .              ;INBUF CONTAINS LINE
```

FUNCTION 11: GET CONSOLE STATUS

Entry Parameters:
Register C: ØBH

Returned Value:
Register A: Console Staus

This function is used to determine whether a valid character is ready for input at the console device. This is useful in combination with SFC 1, Console Input.

If a character is ready at the console, Ø1H is returned in register A. If no character is present, ØØH is returned.

This is the only call which will clear the status of the console. For an alternate method of determining the console status, see SFC 6.

EXAMPLE: Read console status

```
BDOS    EQU    ØØØ5H       ;MAIN CP/M ENTRY POINT
        . . .
STAT    MVI    C,ØBH       ;MOVE SFC 11 INTO C
        CALL   BDOS        ;GET STATUS
        CPI    ØFFH        ;IS CHAR READY?
        JZ     READY       ;YES - GOTO READY
NOCHAR  JMP    STAT        ;NO - START OVER
        . . .              ;
```

FUNCTION 12: RETURN VERSION NUMBER

Entry Parameters:
    Register C: ØCH

Returned Value:
    Registers HL: Version Number

This function is used to determine which revision of CP/M is currently in memory. This permits an application that requires features of one revision of CP/M to verify the actual revision level.

The revision number will be returned in the HL register pair. In all cases, H will contain ØØH. L will contain a hex value in the range of 2ØH through 2FH. A 22H indicates revision 2.2.

Inclusion of this function is primarily for compatibility with other CP/M systems since the HP extended functions include a series of calls that verify not only revision number, but also verify that the CP/M is running on an HP-87.

EXAMPLE: Determine whether CP/M is Rev 2.2 or later

```
BDOS    EQU    ØØØ5H       ;MAIN CP/M ENTRY POINT
        . . .
        MVI    C,ØCH       ;MOVE SFC 12 INTO C
        CALL   BDOS        ;READ REVISION NUMBER
        MOV    A,L         ;MOVE REV INTO A
        CPI    22H         ;IS REV = 2.2
        JNZ    REVOK       ;YES - GOTO REVOK
REVERR  . . .              ;NO - ERROR
```

FUNCTION 13: RESET DISC SYSTEM

Entry Parameters:

Register C: ØDH

This function performs a disc system reset. This means that the file system is restored to a read/write state on all discs that are on-line. Drive "A:" is selected, and the default disc buffer is set to ØØ8ØH.

Use of this function permits the application to programmatically request a change of discs without any need to "reboot" to gain write access to the new disc. Refer to SFC 28 and 29.

EXAMPLE: Make all discs read/write

```
BDOS    EQU    ØØØ5H        ;MAIN CP/M ENTRY POINT
        . . .
        MVI    C,ØDH        ;MOVE SFC 13 INTO C
        CALL   BDOS         ;RESET DISC SYSTEM
        . . .               ;CONTINUE
```

FUNCTION 14: SELECT DISC

Entry Parameters:
Register C: ØEH
Register E: Selected Disc

This function permits the selection of any disc drive as the default disc. This will set all subsequent disc operations that specify the default mass storage device to this disc.

The E register should contain a value specifying which disc is to be selected. A value of ØØH indicates drive "A"; a value of ØFH represents drive "P".

The drive specified will be placed "on-line". Specifically, this activates the directory, which remains active until a re-boot or Reset Disc System (SFC 13) call is done.

If a disc is changed while on-line, it is treated as a read only disc by CP/M to prevent possible data loss from using an incorrect disc. A Reset Disc System call will reset the disc to read/write as mentioned in SFC 13.

If no disc is mounted in the selected drive, CP/M will print an error message. A warm-boot is the only recovery, and the program which has been executing will need to be run again.

EXAMPLE: Select disc "C"

```
BDOS    EQU    ØØØ5H       ;MAIN CP/M ENTRY POINT
        . . .
        MVI    E,Ø2H       ;CODE FOR DRIVE C: INTO E
        MVI    C,ØEH       ;MOVE SFC 14 INTO C
        CALL   BDOS        ;SELECT DISC C:
        . . .
```

FUNCTION 15: OPEN FILE

Entry Parameters:
Register C: ØFH
Registers DE: FCB Address

Returned Value:
Register A: Completion Code

This function is used to initially open a file for access by your application. A successful open will store the correct extent information from the disc directory into the FCB you have reserved for that file.

The address of the FCB should be passed in register pair DE. Your application must set up the FCB as follows: byte Ø is the disc, and may contain a value in the range from Ø through 16. A value of Ø means the file is on the currently selected mass storage device; values from 1 through 16 indicate disc drives "A" through "P," respectively.

The file name and file extension are loaded into byte positions 1 through 11. Byte positions 12 through 35 should be ØØH.

You normally will search the directory for a specific file, and there will be no question of the desired file name or extension. You may, however, choose to include a "wild card" specifier in any of the positions. A wild card character means that any character in that byte position will match. This wild card can also apply to the "ex" field, although care must be taken in doing this because you may not be at the start of file after a successful open call.

This wild card feature is implemented by placing an ASCII"?" (Ø3FH) in the desired byte position of the FCB. In this case, CP/M will return the FCB with data from the first file to match the wild card specification.

Upon return, the A register contains a completion code. If the file was not found, or could not be opened, a return of OFFH should be expected. If the open was successful, a return value in the range of Ø through 3 will occur. This number is used by the system to indicate which of the four directory entries per disc record contains the proper FCB. In any case, this value should not be necessary for your application.

EXAMPLE: Open file 'A:PAYROLLA.DAT'

```
;
BDOS    EQU     0005H       ;MAIN CP/M ENTRY POINT
FCB     DB      00,'PAYROLLADAT',00,00,00,00
DBLOCK  DS      16          ;REMAINDER OF FCB - GROUP
CR      DS      1           ;FCB CURRENT RECORD
REC     DS      3           ;r0,r1, and r2
        , , ,
        LXI     D,FCB       ;MOVE FCB ADDRESS INTO DE
        MVI     C,0FH       ;MOVE SFC 15 INTO C
        CALL    BDOS        ;OPEN FILE
        CPI     0FFH        ;ERROR/NO FILE?
        JZ      ERR         ;YES - GOTO ERR ROUTINE
OK      . . .               ;NO - PROCEED
;
```

FUNCTION 16: CLOSE FILE

Entry Parameters:
Register C: 1ØH
Registers DE: FCB Address

Returned Value:
Register A: Completion Code

This call is the inverse of SFC 15, Open File. When this call is executed, the FCB pointed to by register pair DE is written to the disc, which updates the current file information and assures your data is properly recorded.

The return value for a successful close is in the range of Ø through 3 for the reasons described above under Open File. An unsuccessful close operation, either because of a disc error or some other misfunction, will return a ØFFH.

Technically, a file need not be explicitly closed if no data has been written to that file. However, closing every active file prior to program termination is good programming practice.

EXAMPLE: Close file "A:PAYROLLA.DAT"

```
BDOS    EQU    ØØØ5H        ;MAIN CP/M ENTRY POINT
FCB     DB     ØØ,'PAYROLLADAT',ØØ,ØØ,ØØ,ØØ
DBLOCK  DS     16           ;REMAINDER OF FCB - GROUP
CR      DS     1            ;FCB CURRENT RECORD
REC     DS     3            ;rØ,rl, and r2
        . . .               ;
        LXI    D,FCB        ;FCB ADDRESS IN DE
        MVI    C,1ØH        ;MOVE SFC 16 INTO C
        CALL   BDOS         ;CLOSE FILE
        CPI    ØFFH         ;ERROR/NO FILE?
        JZ     ERR          ;YES - GOTO ERR ROUTINE
OK      . . .               ;NO - PROCEED
```

FUNCTION 17: SEARCH FOR FIRST

Entry Parameters:
Register C: 11H
Registers DE: FCB Address

Returned Value:
Register A: Completion Code

This function is used to return the actual directory entry in the DMA (Direct Memory Access) buffer. This information, while not frequently required by applications, can then be used to examine the actual directory entry. The DIR command is an example of how this call might be used.

The DE register pair should be set up to contain the address of the FCB. Follow the guidelines given in Open File, SFC 15, concerning wild cards and other set-up questions.

If no entry is found in the directory, a value of ØFFH is returned in register A. If an entry is found, register A will be set to a value between Ø and 3. As mentioned in Open File above, this value indicates which 32-byte directory entry within the 128 physical record contains the actual entry for this file.

The currently defined DMA buffer receives the entire disc directory record: the 128 bytes includes four possible files. By multiplying register A by 32, you can calculate how many bytes into the DMA buffer you will find the proper FCB entry.

In the case of this function, you can enter a question mark "wild card" in the drive code filed, byte Ø. This causes CP/M to search all disc drives that are on-line until the first file meeting the FCB name requirements is found.

EXAMPLE: Find first file matching "A:PAYROLL?.DAT"

```
+-----------------------------------------------------+
|                                                     |
|  BDOS    EQU    0005H       ;MAIN CP/M ENTRY POINT  |
|  FCB     DB     00,"PAYROLL?DAT",00,00,00,00        |
|  DBLOCK  DS     16          ;REMAINDER OF FCB - GROUP |
|  CR      DS     1           ;FCB CURRENT RECORD     |
|  REC     DS     3           ;r0,r1, and r2          |
|          . . .             ;                       |
|          LXI    D,FCB       ;FCB ADDRESS INTO DE    |
|          MVI    C,11H       ;MOVE SFC 17 INTO C     |
|          CALL   BDOS        ;SEARCH FOR FILE SPEC   |
|          CPI    0FFH        ;ERROR/NO FILE?         |
|          JZ     ERR         ;YES - GOTO ERR         |
|  OK      . . .              ;NO - PROCEED           |
|                                                     |
+-----------------------------------------------------+
```

FUNCTION 18: SEARCH FOR NEXT

Entry Parameters:
Register C: 12H

Returned Value:
Register A: Completion Code

This function is used in conjuction with the previous Search for First call, to continue scanning through the directory from the entry after that found as the first qualified entry.

Like the first function call, this call returns ØFFH when no additional qualifying entries are found in the directory.

EXAMPLE: Find next occurrence of specified file name

```
BDOS    EQU    ØØØ5H        ;MAIN CP/M ENTRY POINT
FCB     DB     ØØ,'PAYROLL?DAT',ØØ,ØØ,ØØ,ØØ
DBLOCK  DS     16           ;REMAINDER OF FCB - GROUP
CR      DS     1            ;FCB CURRENT RECORD
REC     DS     3            ;rØ,rl, and r2
        . . .
        LXI    D,FCB        ;FCB ADDRESS INTO DE
        MVI    C,12H        ;MOVE SFC  18 INTO C
        CALL   BDOS         ;SEARCH FOR FILE
        CPI    ØFFH         ;ERROR/NO FILE?
        JZ     ERR          ;YES - GOTO ERR
OK      . . .               ;NO - PROCEED
```

FUNCTION 19: DELETE FILE

Entry Parameters:
    Register C: 13H
    Registers DE: FCB Address

Returned Value:
    Register A: Completion Code

This call is used to delete a file entry from the specified disc directory. A question mark character (3FH) can be used in any position of the file name or extension. This "wild card" allows you to delete all files that meet the criteria specified. A "?" may not be specified in the drive code field.

CP/M will automatically delete all extents for the specified file(s), regardless of the value loaded into the "ex" field.

This function will return a value in the range of Ø through 3 if the delete completes normally. A value of ØFFH is returned in case of an error (for example, no such file exists).

EXAMPLE: Delete "B:PAYROLLA.DAT"

```
BDOS    EQU    0005H        ;MAIN CP/M ENTRY POINT
FCB     DB     00,"PAYROLL?DAT",00,00,00,00
DBLOCK  DS     16           ;REMAINDER OF FCB - GROUP
CR      DS     1            ;FCB CURRENT RECORD
REC     DS     3            ;r0,rl, and r2
        . . .
        LXI    D,FCB        ;FCB ADDRESS INTO DE
        MVI    C,13H        ;MOVE SFC 19 INTO C
        CALL   BDOS         ;DELETE FILE
        CPI    0FFH         ;ERROR/NO FILE?
        JZ     ERR          ;YES - GOTO ERR
OK      . . .               ;NO - PROCEED
```

FUNCTION 20: READ SEQUENTIAL

Entry Parameters:
    Register C: 14H
    Registers DE: FCB Address

Returned Value:
    Register A: Return Code

This function is used to input the next sequential record from a previously-opened file. The 128-byte physical record is placed into the currently selected DMA buffer, and a value of 00H is returned in the A register. If, for any reason, the read cannot be completed, a non-zero value is returned in A. Such an error could signify an attempt to read beyond the end of the file.

The record to be read will be the record number stored in the "cr" field of the FCB. After that record is read, the value will be incremented in anticipation of the subsequent read. If the data referenced by the new value of "cr" is not contained in the current extent, CP/M will close the current extent and locate and open the next sequential extent automatically. This assures that the FCB will be valid on subsequent operations.

While "cr" is normally calculated automatically, an application can alter the contents of "cr" as necessary. CP/M will attempt to access whatever record number is indicated by "cr".

EXAMPLE: Read next sequential record from file

```
BDOS    EQU    0005H        ;MAIN CP/M ENTRY POINT
FCB     DB     00,"PAYROLL?DAT",00,00,00,00
DBLOCK  DS     16           ;REMAINDER OF FCB - GROUP
CR      DS     1            ;FCB CURRENT RECORD
REC     DS     3            ;r0,r1, and r2
        . . .
        LXI    D,FCB        ;FCB ADDRESS INTO DE
        MVI    C,14H        ;MOVE SFC 20 INTO C
        CALL   BDOS         ;READ RECORD 'cr'
        CPI    00H          ;ERROR ON READ?
        JNZ    ERR          ;YES - GOTO ERR
OK      . . .               ;NO - PROCEED
```

FUNCTION 21: WRITE SEQUENTIAL

Entry Parameters:
Register C: 15H
Registers DE: FCB Address

Returned Value:
Register A: Completion Code

This function will write the contents of the 128-byte DMA buffer to the record pointed to by the "cr" field in the FCB. The contents of "cr" are incremented automatically after the data is written.

If an error occurs on output, a non-zero value returns in register A. This might happen if the disc is write-protected. A normal completion will return an A register value of ØØH.

As with the Read Sequential function, CP/M will automatically load the proper extent if, after the current "write", the next extent is required.

EXAMPLE: Write next sequential record to file

```
+-----------------------------------------------------+
|                                                     |
|  BDOS    EQU    0005H       ;MAIN CP/M ENTRY POINT  |
|  FCB     DB     00,"PAYROLL?DAT",00,00,00,00        |
|  DBLOCK  DS     16          ;REMAINDER OF FCB - GROUP |
|  CR      DS     1           ;FCB CURRENT RECORD     |
|  REC     DS     3           ;r0,r1, and r2          |
|          . . .                                      |
|          LXI    D,FCB       ;FCB ADDRESS INTO DE    |
|          MVI    C,15H       ;MOVE SFC 21 INTO C     |
|          CALL   BDOS        ;WRITE RECORD 'cr'      |
|          CPI    00H         ;ERROR ON WRITE?        |
|          JNZ    ERR         ;YES - GOTO ERR         |
|  OK      . . .              ;NO - PROCEED           |
|                                                     |
+-----------------------------------------------------+
```

FUNCTION 22: CREATE FILE

Entry Parameters:
    Register C: 16H
    Registers DE: FCB Address

Returned Value:
    Register A: Completion Code

This function is used to initially create a file on a specific disc drive. After the call, the file is open, so no call is necessary to additionally open the file.

The FCB must specify a file name which does not exist on the specified disc unit. The "dr" drive code may be ØØH, but this simply specifies the current default drive. No question marks are allowed in the FCB, since "wild card" characters do not make sense in a create environment.

The DE register pair should point to the FCB. If the file is successfully created, the A register contains a value in the range of Ø through 3. The value indicates the offset of the FCB within the physical record containing the directory entry for this file. A value of ØFFH will be in register A if the file could not be created for any reason.

It is the responsibility of the programmer to verify that no duplicate file name exists on the specified disc. CP/M will create a duplicate directory entry with the same name if requested, and this can make file integrity uncertain.

One way of detecting duplicate file names is to attempt an "open" function on the specified filename. If that call returns an error, then and only then do you attempt to create the file.

EXAMPLE: Create "B:PAYROLL?.DAT"

```
;
BDOS    EQU     0005H       ;MAIN CP/M ENTRY POINT
FCB     DB      01,"PAYROLL?DAT",00,00,00,00
DBLOCK  DS      16          ;REMAINDER OF FCB - GROUP
CR      DS      1           ;FCB CURRENT RECORD
REC     DS      3           ;r0,r1, and r2
        . . .               ;FIRST TEST TO SEE IF
                            ;FILE EXISTS ALREADY
                            ;THEN CREATE IT IF NEEDED
CHKOPN  LXI     D,FCB       ;FCB ADDRESS INTO DE
        MVI     C,0FH       ;MOVE SFC 15 INTO C
        CALL    BDOS        ;TRY TO OPEN FILE
        CPI     0FFH        ;DOES IT EXIST ALREADY?
        JNZ     EXISTS      ;YES - DON'T CREATE IT
CREATE  LXI     D,FCB       ;FCB ADDRESS INTO DE
        MVI     C,16H       ;MOVE SFC 22 INTO C
        CALL    BDOS        ;CREATE FILE
        CPI     0FFH        ;ERROR ON CREATE?
        JZ      ERR         ;YES - GOTO ERR
EXISTS  . . .               ;FILE EXISTS - PROCEED
;
```

FUNCTION 23: RENAME FILE

Entry Parameters:

Register C: 17H
Registers DE: FCB Address

Returned Value:
Register A: Completion Code

This function is used to renaming a file. In this case, the DE register pair contains the address of a buffer which contains 32 bytes. The first 16 bytes reflect the first bytes of the FCB for the existing file name, while the second 16 bytes are the FCB for the new file name.

If the re-name is successful, the A register will contain a value between Ø and 3. An unsuccessful attempt will return a value of ØFFH in register A.

Wild card characters may exist in any of the file name or extension locations, but should be used with care because several files on a given disc may meet the specified condition and be renamed erroneously.

EXAMPLE: Rename "A:OLDFILE.TXT" to "A:NEWFILE.TXT"

```
BDOS     EQU    0005H        ;MAIN CP/M ENTRY POINT
FCBA     DB     00,"OLDFILE TXT",00,00,00,00
DBLOCKS  DS     16           ;REMAINDER OF FCBA - GROUP
CRA      DS     1            ;FCBA CURRENT RECORD
RECA     DS     3            ;r0,rl, and r2
FCBB     DB     00,"NEWFILE TXT",00,00,0000
DBLOCKB  DS     16           ;REMAINDER OF FCBB - GROUP
CRB      DS     1            ;FCBB CURRENT RECORD
RECB     DS     3            ;r0,rl, and r2
         . . .
         LXI    D,FCBA       ;FCB ADDRESS INTO DE
         MVI    C,17H        ;MOVE SFC 23 INTO C
         CALL   BDOS         ;RENAME FILE
         CPI    0FFH         ;ERROR/NO FILE?
         JZ     ERR          ;YES - GOTO ERR
OK       . . .               ;NO - PROCEED
```

FUNCTION 24: RETURN LOGIN VECTOR

Entry Parameters:
Register C: 18H

Returned Value:
Registers HL: Login Vector

This call returns information about which disc drives are on-line. This is useful in determining which drives are available for specifying as drive codes in other system function calls.

The login vector is returned in the HL register pair. The H register is always ØØH. Each bit in the H and L register corresponds to one of the possible 16 drives supported on your HP-87. The least significant bit of L contains the status bit for drive "A," while the most significant bit of H contains the status bit for drive "P." This is illustrated in table 4-3 below.

Table 4-3. Sample Return Login Vector Byte
HL Register Pair

P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A

If the status bit for a drive is "Ø," the drive is not on-line. A value of "1" signifies that the drive is on-line. Attempts to access drives that are not on-line should be made with caution: if no disc is present, CP/M will halt, aborting the program. Warm booting the system is the only recovery available.

EXAMPLE: Find whether "B" is currently logged

```
BDOS    EQU     ØØØ5H       ;MAIN CP/M ENTRY POINT
        . . .
        MVI     C,18H       ;MOVE SFC 24 INTO C
        CALL    BDOS        ;GET LOGIN VECTOR
        MOV     A,L         ;MOVE VECTOR INTO A
        ANA     Ø2H         ;IS BIT FOR B SET?
        JZ      NOT.ON      ;NO - B; NOT ON-LINE
OK      . . .               :YES - DISC IS LOGGED
```

FUNCTION 25: RETURN CURRENT DISC

Entry Parameters:
    Register C: 19H

Returned Value:
    Register A: Current Disc

This function returns the disc identifier for the currently selected mass storage device. The values in register A will be between Ø and 15 corresponding to drive "A" through "P," respectively.

EXAMPLE: Determine whether "E" is current disc

```
BDOS    EQU     ØØØ5H       ;MAIN CP/M ENTRY POINT
        . . .
        MVI     C,19H       ;MOVE SFC 25 INTO C
        CALL    BDOS        ;FIND CURRENT DISC
        CPI     Ø5H         ;IS "E" SELECTED?
        JZ      EDISC       ;YES - GOTO EDISC
NOT.E   . . .               ;NO - PROCEED
```

FUNCTION 26: SET DMA ADDRESS

Entry Parameters:
Register C: 1AH
Registers DE: DMA Address

This function is used to specify the location of the disc buffer to be used in memory for subsequent I/O to disc.

On cold start, warm start, or Disc System Reset, the DMA address is set to 0080H, the default mass memory buffer. However, this call can be used to direct disc I/O to a 128-byte buffer anywhere in memory. To make this change, simply load the address of the new buffer into register pair DE and perform this function call. Since no error is possible, there is no return value.

EXAMPLE: Assign buffer at 5000H as DMA buffer

```
BDOS    EQU    0005H       ;MAIN CP/M ENTRY POINT
MYBUF   EQU    5000H       ;ARBITRARY ADDRESS IN TPA
        . . .
        LXI    D,MYBUF     ;ADDRESS INTO DE
        MVI    C,1AH       ;MOVE SFC 26 INTO C
        CALL   BDOS        ;CHANGE DMA ADDRESS
        . . .              ;DISC BUFFER NOW AT 5000H
```

FUNCTION 27: GET ADDR (ALLOC)

Entry Parameters:
    Register C: 1BH

Returned Value:
    Registers HL: ALLOC Address

This function returns information from BIOS to the application program. As you will see in section 5 of this manual, CP/M maintains several tables of information for each type of disc. Additionally, a table of information is maintained for each disc that is on-line.

If this information is necessary for your application, you can request the address of the "Allocation Vector" for the currently selected drive.

EXAMPLE: Determine allocation vector address

```
BDOS    EQU    0005H        ;MAIN CP/M ENTRY POINT
STORAL  DS     2            ;RESERVE 2 BYTES STORAGE
        . . .
        MVI    C,1BH        ;MOVE SFC 27 INTO C
        CALL   BDOS         ;GET ALLOCATION ADDRESS
        SHLD   STORAL       ;STORE HL IN MEMORY
        . . .
```

FUNCTION 28: WRITE PROTECT DISC

Entry Parameters:
Register C: 1CH

This function permits the currently selected disc to be write protected until the next disc system reset or warm boot. Any attempt to write to a disc after this will result in an error.

EXAMPLE: Write protect disc "A"

```
BDOS    EQU    0005H        ;MAIN CP/M ENTRY POINT
        . . .
        MVI    E,00H        ;CODE FOR DRIVE A: INTO E
        MVI    C,0EH        ;MOVE SFC 14 INTO C
        CALL   BDOS         ;SELECT DISC A:
        MVI    C,1CH        ;MOVE 28 INTO C
        CALL   BDOS         ;WRITE PROTECT DISC A:
        . . .
```

FUNCTION 29: GET READ/ONLY VECTOR

Entry Parameters:
Register C: 1DH

Returned Value:
Registers HL: R/O Vector Value

This function returns the current read/write status on each drive on-line. As you know, SFC 28 can make a disc read-only or write protected. Also, CP/M will force a disc to read-only if it is changed during processing. However, this call will report such a change only if some disc operation is performed on that drive prior to this function call. That is, CP/M does not mark the newly entered disc as read-only until it has some reason to perform an operation on that disc.

As in SFC 24, the HL register is returned with the status of each drive reflected in appropriate bits. The HL register pair will reflect the status of drives "A" through "P". The least significant bit reflects the status of the "A" drive.

Table 4-4. Sample Return Read-Only Vector
L Register Only

P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A

A bit value of "Ø" indicates the corresponding drive is not read-only, while a value of "1" indicates the drive is read-only.

EXAMPLE: Determine whether all disc are read/write

```
BDOS    EQU    ØØØ5H        ;MAIN CP/M ENTRY POINT
        . . .
        MVI    C,1DH        ;MOVE SFC 29 INTO C
        CALL   BDOS         ;RETURN VECTOR
        MOV    A,L          ;MOVE VECTOR INTO A
        OR     H            ;CHECK UPPER HALF OF VECTOR
        CPI    ØØH          ;ARE ALL DISCS R/W?
        KZ     ALL          ;YES - GOT ALL
NOTALL  . . .               ;NO - AT LEAST ONE IS R/O
```

FUNCTION 30: SET FILE ATTRIBUTES

Entry Parameters:
    Register C: 1EH
    Registers DE: FCB Address

Returned Value:
    Register A: Completion Code

The Set File Attributes function allows programmatic manipulation of permanent indicators attached to files. In particular, the R/O and system attributes (t1' and t2') can be set or reset. In the following explanation, a ' behind a FCB byte reference refers to the most significant bit of the specified byte. For example, f1' is the high-order bit of the first byte of the file name in the FCB.

The DE register pair addresses an FCB containing a unique file name with the appropriate attribute bits set as desired. That is, bytes f5 through f8 and t1 through t3 have their high order bits set to the desired new values. SFC 30 searches for a matching file name, and changes the existing directory entry to contain the selected indicators. Indicators f1' through f4' are not presently used, but may be useful for applications programs, since they are not involved in the matching process during file open and close operations. Indicators f5' through f8' and t3' are reserved for future system expansion.

EXAMPLE: Make 2 file read/only

```
BDOS    EQU    0005H        ;MAIN CP/M ENTRY POINT
FCB     DB     00,'PAYROLLADAT',00,00,00,00
ROFLAG  EQU    FCB+9        ;BYTE t1
SYSFLG  EQU    FCB+10       ;BYTE t2
DBLOCK  DS     16           ;REMAINDER OF FCB - GROUP
CR      DS     1            ;FCB CURRENT RECORD
REC     DS     3            ;r0,r1, and r2
        . . .
        LXI    H,ROFLAG     ;LOAD BYTE ADDRESS IN HL
        MOV    A,M          ;REQAD BYTE INTO A
        ADI    80H          ;SET HIGH BIT
        MOV    M,A          ;WRITE BYTE BACK TO FCB
        LXI    D,FCB        ;FCB ADDRESS INTO DE
        MVI    C,1EH        ;MOVE SFC 30 INTO C
        CALL   BDOS         ;SET FILE ATTRIBUTES
        CPI    0FFH         ;ERROR/NO FILE?
        JZ     ERR          ;YES - GOTO ERR
OK      . . .               ;NO - PROCEED
```

FUNCTION 31: GET ADDR (DISC PARMS)

Entry Parameters:
Register C: 1FH

Returned Value:
Registers HL: DPB Address

As mentioned in SFC 27, CP/M maintains a variety of tables in memory for BIOS. These are discussed in section 5.

One of the tables used by CP/M is the "disc parameter block" (DPB), which maintains information on each type of disc on the system.

This disc parameter block should be considered informational only, and not to be modified by your application.

EXAMPLE: Return address of disc parameter block

```
BDOS    EQU    0005H       ;MAIN CP/M ENTRY POINT
DPB     DS     2           ;RESERVE 2 BYTES
        . . .
        MVI    C,1FH       ;MOVE SFC 31 INTO C
        CALL   BDOS        ;GET DISC PARM ADDRESS
        SHLD   DPB         ;STORE HL IN MEMORY
        . . .
```

FUNCTION 32: SET/GET USER NUMBER

Entry Parameters:
    Register C: 20H
    Register E: 0FFH (get) or
                User Number (set)

Returned Value:
    Register A: Current User Number or
                (no value)

This function permits the application to programmatically read or write the current user number. The user number determines to which files a particular program (or user) has access.

To determine the current user ID, pass 0FFH in register E. The user number will be returned in register A.

To change to a new user ID, pass that user number in register E. In this case, the return value in A has no significance.

The user number should be in the range of 0 through 31. If a request is made to change to a user number out of this range, the value of (E MOD 32) will be the user number selected.

EXAMPLE: Read user number and set to 1

```
BDOS    EQU     0005H       ;MAIN CP/M ENTRY POINT
        . . .
        MVI     E,0FFH      ;SET UP READ CODE
        MVI     C,20H       ;MOVE SFC 32 INTO C
        CALL    BDOS        ;GET USER NUMBER
        CPI     01H         ;IS USER NUMBER = 1?
        JZ      OK          ;YES - GOTO OK
SET     MVI     E,01H       ;SET UP WRITE CODE
        MVI     C,20H       ;MOVE 32 INTO C
        CALL    BDOS        ;SET UER NUMBER
OK      . . .               ;USER NUMBER NOW 1
```

FUNCTION 33: READ RANDOM

Entry Parameters:
Register C: 21H
Registers DE: FCB Address

Returned Value:
Register A: Completion Code

This function, like the Sequential Read function above, reads a 128-byte physical record from a selected disc file. However, this function reads the record number stored in the last three bytes of the FCB rather than the record indicated in "cr".

You will remember that a file is considered a sequence of up to 65535 physical records of 128 bytes each. You will also remember that "random access" files require an additional three bytes in the FCB: rØ, rl, and r2.

The rØ, rl byte pair is treated as a double-byte word that contains the physical record number to be read. Byte rØ is the least significant 8 bits, while byte rl contains the most significant bits. Byte r2 is for system use, and should be set to zero. A non-zero value in r2 indicates overflow past end-of-file, and is an error.

In order to access a file with this call, the base extent (extent ØØH) must have been previously opened. This is true even if the base extent does not contain the desired record.

Upon each random read call, the logical extent ("ex") and current record ("cr") fields within the FCB are automatically set by CP/M. Unlike sequential read, a random read will not increment "cr" after each read, nor will it increment rØ,rl. Because CP/M uses the "cr" and "ex" pointers during random reads, it is possible for an application to use sequential reads and random reads within the same file. Remember, though, that a sequential read updates "cr" after it reads the record. This means that, if a sequential read is done immediately after a random read, both read calls will return the same buffer.

To set up a random read, load the address of the FCB in the DE register pair and load rØ/rl with the desired record number. Remember: rØ contains the least significant bits! Proceed with a call to ØØØ5H as with all other system function calls to execute the function.

If the read is successful, the A register will contain a ØØH and the DMA buffer will contain the physical record. If there is an error, the value returned in A will indicate the nature of the problem. The only errors associated with this function are indicated in the following table.

Table 4-3.
Random Access Error Codes

01H	Reading Un-Written Data.
03H	Cannot Close Current Extent.
04H	Seek to Un-Written Data.
06H	Seek past end-of-file.

Error codes 02H and 05H are not used in this call, and should not occur as a result of a random read.

Error codes 01H and 04H occur when a random read attempts to access a block of data which has not previously been written, or to an extent which has not been created. Error code 03H should never occur in a normally functioning system. Should it occur, either re-open the file or re-read the record. These two actions should correct any problem.

Error code 06H occurs whenever the value in r2 is non-zero.

EXAMPLE: Random read record in r0,r1

```
BDOS    EQU     0005H       ;MAIN CP/M ENTRY POINT
FCB     DB      00,'PAYROLLADAT',00,00,00,00
DBLOCK  DS      16          ;REMAINDER OF FCB - GROUP
CR      DS      1           ;FCB CURRENT RECORD
REC     DS      3           ;r0,r1, AND r2
        . . .
        LXI     D,FCB       ;FCB ADDRESS INTO DE
        MVI     C,21H       ;MOVE SFC 33 INTO C
        CALL    BDOS        ;READ RANDOM RECORD
        CPI     00H         ;ANY ERRORS?
        JZ      OK          ;NO - GOTO OK
        CPI     01H         ;ERROR 1?
        JZ      ERR1        ;YES - GOTO ERR1
        . . .               ;NO - CHECK NEXT ERROR
```

FUNCTION 34: WRITE RANDOM

Entry Parameters:
    Register C: 22H
    Registers DE: FCB Address

Returned Value:
    Register A: Completion Code

This function writes a record of data to a random access file.

This call is similar to the random read function discussed earlier. The data to be written is stored in the DMA buffer, and the record number to be written is stored in the r0,r1 byte pair. Finally, the DE register pair is loaded with the address of the FCB, and the call is made.

As with random read, the proper values of "cr" and "ex" are calculated by CP/M, but are not incremented after the record is written. Management of record pointers is left to the application.

A successful write operation returns a 00H in register A. The same error codes apply as with the random read above, with one additional code. A return value of 05H in register A indicates that a new extent cannot be created because of directory overflow.

As with random read, sequential file access can be mixed with random access in the same application. However, the proper management of the appropriate pointers is critical to insuring expected results.

EXAMPLE: Random write record r0,r1

```
BDOS    EQU     0005H       ;MAIN CP/M ENTRY POINT
FCB     DB      00,'PAYROLLADAT',00,00,00,00
DBLOCK  DS      16          ;REMAINDER OF FCB - GROUP
CR      DS      1           ;FCB CURRENT RECORD
REC     DS      3           ;r0,r1, and r2
        . . .
        LXI     D,FCB       ;FCB ADDRESS INTO DE
        MVI     C,22H       ;MOVE SFC 34 INTO C
        CALL    BDOS        ;WRITE RECORD
        CPI     00H         ;ERROR ON WRITE?
        JZ      OK          ;NO - PROCEED TO OK
        CPI     05H         ;YES - ERROR 5?
        JZ      ERRS        ;ERROR IS 5
        . . .
```

FUNCTION 35: COMPUTE FILE SIZE

Entry Parameters:
    Register C: 23H
    Registers DE: FCB Address

Returned Value:
    Random Record Field Set

This function call is used to determine the current file size. To calculate file size, this call returns the "next available record number."

To execute this call, the address of the FCB is loaded into the DE register pair. The FCB must include the random bytes rØ, rl and r2.

CP/M calculates the record number one greater than the last record in the file, and returns that value in the rØ,rl byte pair. If the highest record number is 65535, the record number will overflow into byte r2, signifying that no more records may be written to the file.

The size provided by this call is the physical size of the file assuming it was written sequentially. Note that a random access file may contain "holes," and the highest record number may not correspond to the actual number of records contained in the file. For example, if only record number 65536 is written to a random access file, this call will return a size of 65536 records, even though only one block has been allocated.

Data can be appended to the end of an existing file by simply making this call to set the random read bytes to the end of the file. Then any subsequent write operations will be appended onto the end.

EXAMPLE: Compute file size of specified file

```
BDOS    EQU    0005H        ;MAIN CP/M ENTRY POINT
HOLD    DS     2            ;RESERVE 2 BYTES
FCB     DB     00,"PAYROLL?DAT",00,00,00,00
DBLOCK  DS     16           ;REMAINDER OF FCB - GROUP
CR      DS     1            ;FCB CURRENT RECORD
REC     DS     3            ;r0,r1, and r2
        . . .
        LXI    D,FCB        ;FCB ADDRESS INTO DE
        MVI    C,23H        ;MOVE SFC 35 INTO C
        CALL   BDOS         ;COMPUTE FILESIZE
        LHLD   REC          ;LOAD HL FROM REC
        SHLD   HOLD         ;AND STORE IT IN MEMORY
        . . .
```

FUNCTION 36: SET RANDOM RECORD

Entry Parameters:
    Register C: 24H
    Registers DE: FCB Address

Returned Value:
    Random Record Field Set

This function call is used to set up an FCB for a random read or write following a sequential operation to the desired record.

To access this information, load the DE register pair with the address of the FCB. When the call is made, CP/M loads the actual record count into the r0,r1 byte pair. After this is done, the FCB is set up for random access.

EXAMPLE: Set record number to value from memory

```
BDOS    EQU    0005H        ;MAIN CP/M ENTRY POINT
HOLD    DS     2            ;RESERVE 2 BYTES
FCB     DB     00,"PAYROLL?DAT",00,00,00,00
DBLOCK  DS     16           ;REMAINDER OF FCB - GROUP
CR      DS     1            ;FCB CURRENT RECORD
REC     DS     3            ;r0,r1, and r2
        . . .
        LHLD   HOLD         ;LOAD RECORD FROM MEMORY
        SHLD   REC          ;MOVE INTO r0,r1
        LXI    D,FCB        ;FCB ADDRESS INTO DE
        MVI    C,24H        ;MOVE SFC 36 INTO C
        CALL   BDOS         ;SET RANDOM RECORD
        . . .
```

FUNCTION 37: RESET DRIVE

Entry Parameters:
Register C: 25H
Registers DE: Drive Vector

Returned Value:
Register A: ØØH

This function permits the status of any disc(s) to be reset with a single call.

Each bit within the DE register pair represents the state to which each disc is to be set. The least significant bit of E represents drive "A," and the most significant bit of D represents "P." If any bit is set to "1," the drive is to be reset to read/write status. A value of "Ø" means no change should be made in the status. It is the login vector bit that is actually reset by this call.

EXAMPLE: Reset all discs

```
BDOS    EQU    ØØØ5H       ;MAIN CP/M ENTRY POINT
        . . .
        LXI    D,ØFFFH     ;SETUP ALL DISC CODES
        MVI    C,25H       ;MOVE SFC 37 INTO C
        CALL   BDOS        ;RESET DISCS
        . . .
```

FUNCTION 38: UNUSED

FUNCTION 39: UNUSED

FUNCTION 40: WRITE RANDOM WITH ZERO FILL

Entry Parameters:
Register C: 28H
Registers DE: FCB Address

Returned Value:
Register A: Completion Code

This function is very similar to SFC 34 in that both calls result in a random write. Data is moved from the DMA buffer to the disc file. However, when the first record is written to any newly allocated group, SFC 34 does not alter any (old) existing data in that group. This call will do a zero fill, which assures that no "garbage" data exists in the currently unused areas of those groups allocated to your file.

EXAMPLE: Write record r0,r1 with zero fill

```
BDOS    EQU    0005H        ;MAIN CP/M ENTRY POINT
FCB     DB     00,'PAYROLLADAT',00,00,00,00
DBLOCK  DS     16           ;REMAINDER OF FCB - GROUP
CR      DS     1            ;FCB CURRENT RECORD
REC     DS     3            ;r0,r1, and r2
        . . .
        LXI    D,FCB        ;FCB ADDRESS INTO DE
        MVI    C,28H        ;MOVE SFC 40 INTO C
        CALL   BDOS         ;WRITE RANDOM
        . . .
```

4.3 Extended System Function Calls

To address the added features of HP Series 80 Personal Computers, several additional function calls have been added to standard CP/M. These functions allow you to utilize the power of your system within the CP/M environment, so your application can remain independent of the physical level device control.

Rather than add a variety of system function calls, and possibly interfere with future CP/M enhancements, all extended calls are treated as "sub-functions" within a single CP/M function call.

The HP extended calls follow all the conventions required by standard CP/M. Specifically, the HP function number 255 (OFFH) is loaded into the C register. The B register contains the subfunction number, and the other register pairs are used to pass information to the sub-function routine. In all cases, single byte returns are made in the A register and double byte returns are made through the HL register pair.

Table 4-4. HP Extended System Function Calls

CP/M CALL	SUB FUNCTION	OPERATION DESCRIPTION
255	Ø	RETURN CP/M IDENTIFIER
255	111	RETURN HP VERSION NUMBER
255	113	READ MEMORY BUFFER
255	119	WRITE MEMORY BUFFER
255	12Ø	RETURN REGION BOUNDS
255	122	CHAIN TO SPECIFIED PROGRAM
255	126	INTERROGATE AND ALTER JUMP VECTOR
255	127	RETURN MACHINE NUMBER

Table 4-5 illustrates a sample sub-program using the extended function call. Note that, instead of using two "MVI" commands, a single "LXI B,FUNUM" could be used to load register pair BC if "FUNUM" was equated to 74FFH.

Table 4-5. Sample HP Extended System Function Call

```
BDOS        EQU     ØØ5H        ;BDOS Entry Point
HPFUN       EQU     ØFFH        ;HP Extended Call
SUBFN       EQU     74H         ;Desired Sub-Function
            ORG     Ø1ØØH       ;Like a good program
START       MVI     B,SUBFN     ;Load sub-function
            MVI     C,HPFUN     ;Load HP function
            CALL    BDOS        ;Execute call
            RET                 ;To calling program
            END
```

Now you've seen a quick overview of the HP extended function calls. The rest of this section will introduce you to each, and will show you how to use these calls. In several cases, additional discussion is included in other sections.

HP FUNCTION Ø : RETURN OEM NUMBER

Entry Parameters
  Register Pair BC: ØØFFH

Return Parameters
  Register Pair HL: HP OEM Version

This sub-function is used to verify that the operating system which is currently loaded includes the HP extended system function calls as part of BDOS. This call should be used at the start of every program in order to verify that the system being used is an HP-87.

The return value passed in the HL register pair indicates the OEM number of the supplier of CP/M. The HP Series 8Ø OEM number is 517, or Ø1Ø5H. If the CP/M currently running has been provided by HP, this will be the value returned in the HL register pair. Any other value indicates the CP/M may not execute the other HP extended function calls.

EXAMPLE: Verify HP CP/M Identification

```
BDOS        EQU     ØØØ5H       ;MAIN CP/M ENTRY POINT
HPZERO      EQU     ØØFFH       ;SUB-FUNCTION ZERO
            . . .
            LXI     B,HPZERO    ;SB-FUNCTION IN BC
            CALL    BDOS        ;RETURN NUMBER INTO HL
            MOV     A,L         ;MOVE ID INTO A
            CPI     Ø5H         ;IS THIS HP CONTRACT ID?
            MOV     A,H
            CPI     Ø1H
            JNZ     NONHP       ;NO - GOTO NON-HP ROUTINE
OK          . . .               ;YES - PROCEED
```

HP FUNCTION 111: Return HP VERSION NUMBER

Entry Parameters
  Register Pair

Return Parameters
  Register Pair

This sub-function returns the HP version number of the operating system in the HL register pair. This version number is interpreted as follows. The high four bits of the H register are added to the ASCII value for the letter A to determine the version letter (A=Ø, B=1, ...). The low four bits of the H register and the high and low four bits of the L register each represent a BCD digit making a revision number of the form X.XX. For example, if the HL register pair contains Ø1ØØH, the operating system is version "A," revision 1.ØØ.

HP FUNCTION 113: READ MEMORY FILE BUFFER

Entry Parameters Register Pair BC: 71FFH

Return Parameters Register Pair HL: Address of Buffer

This sub-function permits a program to read a memory buffer left by a previous program. In this manner, programs can receive identifying data from a "father" program, or even from a previous portion of itself.

To access the buffer, load the BC register pair with 71FFH. If no buffer was left, the HL register pair will contain 0001H upon return. If a valid buffer is available, its address will be stored in HL upon return.

The "father" program must locate the buffer within the TPA, at a location which is not destroyed by the CCP or by the "son" program when a warm start is executed. IF THIS IS NOT DONE, THIS SUB-FUNCTION MAY INDICATE A VALID BUFFER ADDRESS WHICH CONTAINS POSSIBLY INVALID DATA.

If a buffer is available, it will be identical in format to the buffer described under sub-function number 119.

EXAMPLE: Determine which program called "me"

```
BDOS        EQU     0005H       ;MAIN CP/M ENTRY POINT
PASS        EQU     71FFH       ;SUB-FUNCTION 113
TMEP        DS      2           ;RESERVE 2 BYTE STORAGE
            . . .
            LXI     B,PASS      ;SUB-FUNCTION INTO BC
            CALL    BDOS        ;READ BUFFER ADDR INTO HL
            MOV     A,H         ;TEST H FOR ZERO
            CPI     00H         ;HIGH BYTE ZERO?
            JNZ     OK          ;NO - HL <> 0001, HENCE OK
            MOV     A,L         ;H = 0. IS L = 1
            CPI     01H         ;IS L = 1?
            JZ      NOBUF       ;HL = 0001. NO BUFFER
OK          SHLD    TEMP        ;KEEP ADDRESS IN TEMP
            . . .               ;ADDRESS NOW STORED
```

HP FUNCTION 119: WRITE MEMORY FILE BUFFER

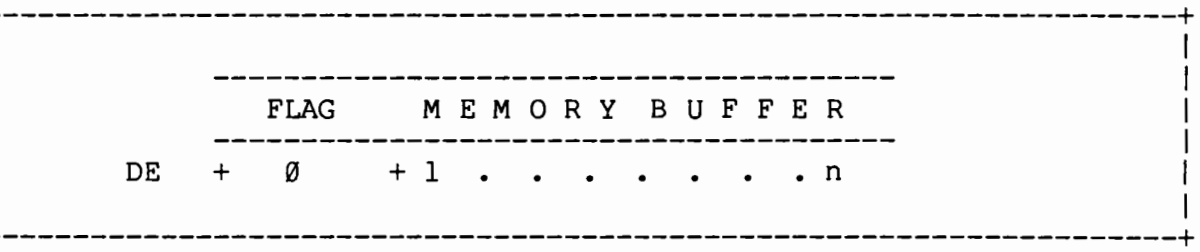
Entry Parameters
  Register Pair BC: 77FFH
  Register Pair DE: Address of Buffer

Return Parameters
  Register Pair HL: Non-zero

This sub-function allows a program to write information to a memory buffer to be read at a later time. The buffer, maintained by BIOS in the TPA, can be subsequently read by the same program, or by a "son" program initiated using the CHAIN sub-function described below, as long as the buffer area of the TPA is not destroyed.

The format of the buffer used in this sub-function is shown in table 4-6.

Table 4-6. Write Memory File Buffer



The FLAG byte is a numeric parameter used to ensure that memory buffers do not remain active indefinitely. Whenever a warm boot is performed, the FLAG is incremented by 2, and any buffer with a FLAG value greater than 3 is purged. By writing your original buffer with a value of Ø or 1, the buffer will remain through the next warm boot (or chain). However, the buffer will contain a value of 2 or 3 in the "son" program.

When the "son" terminates, the value will be incremented to 4 or 5 and will be deleted. The net effect of this FLAG is that, when initially set to Ø or 1, the buffer will remain valid for the "son" program only.

The MEMORY BUFFER can contain any data desired, although any programs which exchange data via this memory file should "know" the format of any data which was left.

Once the buffer is established, the program should load the address of the buffer into the DE register pair. When the sub-function call is made, the address of the buffer will be stored by BIOS. As with sub-function 122, CHAIN TO PROGRAM, the user must assume the responsibility to locate the buffer in an area of memory which will not be over-written by the CCP. In this case, there is the additional responsibility of placing the buffer in the TPA such that the program which will read the memory buffer will not over-write the buffer as it is loaded. To safely position the buffer, HP sub-function 12Ø can be used to determine the start of the CCP. Be certain to position the buffer such that the last byte of the buffer is at or below this boundary address. Then, make sure the "son" program is no larger than the starting address of the buffer.

If a program has performed this sub-function and subsequently needs to erase the buffer and not leave any data, the subfunction can be called a second time with a value of ØØØØH in the DE register pair. This has the effect of cancelling the buffer.

Careful use of this call can save time when small segments of data are to be exchanged within or between programs.

EXAMPLE: Leave current program name for "son"

```
BDOS    EQU     ØØØ5H       ;MAIN CP/M ENTRY POINT
MEMBUF  EQU     77FFH       ;SUB-FUNCTION 119
        . . .
        LXI     B,MEMBUF    ;SUB-FUNCTION IN BC
        LXI     D,MYNAME    ;ADDR OF BUFFER IN DE
        CALL    BDOS        ;'WRITE' BUFFER
        . . .
        ORG     5ØØØH       ;RE-POSITION PC
        DB      ØH          ;FLAG
MYNAME  DB      'FATHER'    ;LEAVE CURRENT PROGRAM
        . . .               ;NAME IN MEMORY AT 5ØØØH
```

HP FUNCTION 120: RETURN REGION BOUNDS

Entry Parameters
  Register Pair BC: 78FFH
  Register E: Request Code

Entry Parameters
  Register Pair HL: Return Address

This sub-function permits an application to determine the bounds of each region of CP/M. This can be useful in calculating the available memory space for buffers and other memory storage tables.

To request the bounds on a particular region, simply load the appropriate request code into register E. The codes and their meanings are shown in table 4-7. Refer to table 2-1 for an illustration of the CP/M memory map.

Table 4-7. Region Request Parameters

Request Code	Boundary Returned
0	Start of CCP
1	Top of CCP
2	Start of BDOS
3	Top of BDOS
4	Start of BIOS
5	Top of BIOS
6	Start of Reserved RAM
7	Top of Reserved RAM

Each of these regions should be familiar to you with the exception of the "Reserved RAM." This area is actually part of the BIOS, in that BIOS is defined as that portion of CP/M which is machine specific. No user application should use any locations in this area.

The address of the requested bound will be returned in the HL register pair upon return. The most significant byte is in register H, the least significant in register L.

EXAMPLE: Determine the last address of the TPA

```
;
BOO     EQU     0000H       ;CP/M WARM BOOT ENTRY
BDOS    EQU     0005H       ;MAIN CP/M ENTRY POINT
BOUNDS  EQU     7SFFH       ;SUB-FUNCTION 120
MEMTOP  DS      2           ;RESERVE TWO BYTES
        . . .
        LXI     B,BOUNDS    ;SUB-FUNCTION IN BC
        MVI     E,1         ;TOP OF CCP CODE IN E
        CALL    BDOS        ;RETURN TOP OF TPA IN HL
        SHLD    MEMTOP      ;STORE HL IN MEMTOP
        . . .
```

HP FUNCTION 122: CHAIN TO SPECIFIED PROGRAM

Entry Parameters
  Register Pair BC: 7AFFH
  Register Pair DE: Address of Buffer

Return Parameters
  Register Pair HL: Non-zero

This sub-function permits you to chain to another assembly language (.COM) file on disc rather than warm boot CP/M. This feature is similar to the chain feature available in many high level languages such as BASIC, but which is typically unavailable in most implementations of CP/M.

To understand this call, it is necessary to understand what is involved in performing a warm boot. Essentially, when you perform a warm boot (JMP ØØØØH), the CCP is reloaded by CP/M. Once CCP is running, it will check for a file named "WELCOME.COM" on the currently selected disc. If it finds that program, it will load and execute it as an auto-start program. If no such file is found, CCP displays "WELCOME?", signifying that no file with that name was found.

By using this sub-function call, you can specify to CP/M that another program is to be executed instead of returning to the CCP.

To specify that some program, or system command be executed, simply load the DE register pair with the address of an information buffer. The buffer should be of the format shown below in table 4-8.

Table 4-8. Chain Program Buffer

Byte Ø, LEN, contains the length of the string to be passed to the CCP in the Command Buffer. The value must be between 1 and 127. Any string longer than 127 is truncated.

The remaining bytes of the buffer should contain the actual string to be passed to CCP for evaluation. This buffer is shown above as the "CCP COMMAND BUFFER." The program which makes this sub-function call must build the appropriate buffer as indicated above. When this call is actually executed, the address of the buffer is saved in BIOS by CP/M. NOTE THAT THE BUFFER ITSELF IS NOT SAVED EXCEPT IN THE TPA! This means that the user must take the responsibility of locating the buffer in an area of memory which will not be destroyed by the CCP when the warm start is executed.

A program showing the use of this subfunction is shown in the example below. In the example, the PIP program is invoked to copy a file "OLD.TXT" to "NEW.TXT".

Example: Chain to PIP and copy OLD.TXT to NEW.TXT

```
BOOT    EQU     0000H       ;CP/M WARM BOOT ENTRY
BDOS    EQU     0005H       ;MAIN CP/M ENTRY POINT
CHAIN   EQU     7AFFH       ;SUB-FUNCTION 122
BUFFR   DB      19,"PIP NEW.TXT=OLD.TXT"
        . . .
        LXI     B,CHAIN     ;SUB-FUNCTION IN BC
        LXI     D,BUFFR     ;ADDR OF BUFFER IN DE
        CALL    BDOS        ;SET UP CHAIN ON BOOT
        CALL    BOOT        ;WARM-START AND CHAIN
        END                 ;END OF ASSEMBLY
```

HP FUNCTION 126: INTERROGATE AND ALTER JUMP VECTOR

Entry Parameters
  Register Pair BC: 7EFFH
  Register Pair DE: Address of Buffer

Return Parameters
  Register Pair DE: Address of Buffer

This sub-function permits an application to determine the contents of the Jump Vector table, and, if desired, to alter the table contents.

In section 5, you will learn more about the "Jump Vector Table" maintained by CP/M. Briefly, this table is a list of "jump" instructions which tells CP/M the addresses of routines to actually implement the standard system function calls. If, for any reason, your application needs to determine or alter the addresses in the jump table, this sub-function permits it to do so.

To make this call, load the BC register pair with 7EFFH. Then load the DE register Pair with the address of a buffer which includes additional information.

The buffer takes the format illustrated in table 4-9. Byte Ø contains the "jump vector number" which indicates which entry in the table is of interest. These JVNs are indicated in Table 15-4 which follows. Section 5 contains an extended discussion on the function of each of these jump vectors.

Byte 1 is a flag which indicates whether to read the jump vector entry into this buffer or whether to write to the jump vector table from this buffer. A value of ØØH will indicate that this is a read, while a value of Ø1H will actually change the jump vector contents.

Table 4-9. Jump Vector Buffer

	JVN	FLAG	JMP	LO	HI
DE +	Ø	1	2	3	4

If the request is a read, CP/M will access the jump vector table in BDCS and copy the table entry into byte 2 through 4 of this buffer. This is a jump command of the format:

"JMP XX XX"

Three bytes are returned in the buffer during an "interrogate" sub-function call. The single byte opcode for the "JMP" command is in byte 2, with the low byte of the address in byte 3 and the high byte in byte 4.

If the request is made to alter the jump vector, the new contents of the jump table should be loaded into byte 2 through 4. The call will exchange the contents of the jump table entry with the buffer contents and return a non-zero value in HL.

Table 4-10. Jump Vector Table Entries

JVN	Jump	Comments
00H	JMP BOOT	;Cold Boot Load Address
01H	JMP WBOOT	;Warm Boot Load Address
02H	JMP CONST	;Check Console Status
03H	JMP CONIN	;Read Console Character
04H	JMP CCNOUT	;Write Console Character
05H	JMP LIST	;Write List Character
06H	JMP PUNCH	;Write Punch Character
07H	JMP READER	;Read Reader Character
08H	JMP HOME	;Move to Track 0 on Disc
09H	JMP SELDSK	;Select Disc Drive
0AH	JMP SETTRK	;Set Track Number
0BH	JMP SETSEC	;Set Sector Number
0CH	JMP SETDMA	;Set DMA Buffer Address
0DH	JMP READ	;Read Selected Sector
0EH	JMP WRITE	;Write Selected Sector
0FH	JMP LISTST	;Check List Status
10H	JMP SECTRAN	;Sector Translate Routine

This call can be useful in the case where an application needs to replace or enhance a standard system function call although such changes should only be attempted by experienced CP/M and Z-80 programmers because of the complexities involved.

EXAMPLE: Replace CONIN with routine at address 5000H

```
BDOS    EQU     0005H           ;MAIN CP/M ENTRY POINT
JMPVEC  EQU     7EFFH           ;SUB-FUNCTION 126
MYADDR  EQU     5000H           ;ADDRESS OF MY 'CONIN'
HOLD    DS      2               ;TWO BYTE TEMP STORAGE
BUF     DS      5               ;RESERVE FIVE BYTE BUFFER
JVN     EQU     BUF+0           ;BUILD BUFFER ELEMENTS
FLAG    EQU     BUF+1           ;BY NAME.
CMD     EQU     BUF+2
ADDR    EQU     BUF+3           ;TWO BYTE ADDRESS (LO:HI)
        MVI     A,0             ;SET UP A 'READ' CODE
        STA     FLAG            ;STORE IN BUFFER
        MOV     M,A             ;SET UP A JVN 'READ'
        MVI     A,3             ;CONIN JVN NUMBER
        STA     JVN             ;STORE IN BUFFER
;                                NOW READ JVN TABLE
        LXI     B,JMPVEC        ;SUB-FUNCTION IN BC
        LXI     D,BUF           ;ADDR OF BUF INTO DE
        CALL    BDOS            ;INTERROGATE JUMP TABLE
;
        LHLD    H,ADDR          ;ADDR OF 'ADDR' INTO HL
        SHLD    HOLD            ;STORE 'ADDR' INTO 'HOLD'
        LXI     H,MYADDR        ;ADDR OF MY ROUTINE IN HL
        SHLD    ADDR            ;PLACE IT IN JVN BUFFER
;
        MVI     A,1             ;SET UP A 'WRITE' CODE
        STA     FLAG            ;STORE IN BUFFER
        LXI     B,JMPVEC        ;SUB-FUNCTION IN BC
        LXI     D,BUF           ;ADDR OF BUF IN DE
        CALL    BDOS            ;WRITE JUMP TABLE
```

HP FUNCTION 127: RETURN HP MACHINE NUMBER

Entry Parameters
  Register Pair BC: 7FFFH

Return Parameters
  Register Pair HL: 0057H (87)

This sub-function is used to return the model number of the HP computer on which CP/M is running. By using this call in conjunction with the "OEM Number" call described above you can verify not only that your application is running on an HP-87, but that it is using a specific release version of HP CP/M.

This sub-function is accessed by passing OFFH in register C and 7FH in register B. The revision number will be returned in the HL register pair.

EXAMPLE: Check for HP Revision 1 CP/M

```
BDOS     EQU     0005H          ;MAIN CP/M ENTRY POINT
HPMACH   EQU     7FFFH          ;SUB-FUNCTION 127
         . . .
         LXI     B,HPMACH       ;SUB-FUNCTION IN BC
         CALL    BDOS           :RETURN MACHINE # IN HL
         MOV     A,H            ;MOVE HIGH BYTE INTO A
         CPI     00H            ;IS IT ZERO?
         JNZ     MACERR         ;NO - GOTO MACERR
         MOV     A,L            ;MOVE LOW BYTE INTO A
         CPI     87H            ;IS IT ONE?
         JZ      REV1           ;YES - OK
MACERR   . . .                  ;NO - MACHINE ERROR
```

Section
V

BIOS ROUTINES

5.1 Introduction

In previous sections, you have learned how to use BDOS function calls in your applications. You have seen in those sections all of the information you will need for most tasks. However, you will occasionally want to know more about the internal functioning of the operating system, and this section will provide you with that knowledge.

5.2 BIOS Entry Points

As mentioned earlier in this manual, BDOS provides a uniform method of accessing physical devices for I/O, and, until now, all I/O was actually performed by BDOS through system function calls.

You would expect BDOS to access BIOS device drivers to perform the actual I/O, and this is, in fact, what happens, Within BIOS is a table of "jump" commands. This table starts at a known location, and each entry is exactly three bytes in length. The entries are in a specific order, so BDOS "knows" how to access each function based on the offset from the start of this Jump Vector Table. The actual jump table is illustrated in table 5-1.

Table 5-1. JUMP VECTOR TABLE

Jump	Comments
JMP BOOT	;Cold Boot Load Address
JMP WBOOT	;Warm Boot Load Address
JMP CONST	;Check Console Status
JMP CONIN	;Read Console Character
JMP CONOUT	;Write Console Character
JMP LIST	;Write List Character
JMP PUNCH	;Write Punch Character
JMP READER	;Read Reader Character
JMP HOME	;Move to Track Ø on Disc
JMP SELDSK	;Select Disc Drive
JMP SETTRK	;Set Track Number
JMP SETSEC	;Set Sector Number
JMP SETDMA	;Set DMA Buffer Address
JMP READ	;Read Selected Sector
JMP WRITE	;Write Selected Sector
JMP LISTST	;Check List Status
JMP SECTRAN	;Sector Translate Routine

System initialization and restart; simple device I/O; and disc I/O. Each "JMP" address corresponds to that portion of BIOS which performs the actual operation requested. Let's examine the functioning of the various entry points.

BOOT is called by the cold start loader and is responsible for system initialization. This routine prints the "logon" message, sets initial parameters for CP/M, and passes control to CCP.

WBOOT is called to perform a warm start. This routine reloads BDOS and the CCP whenever called. This is the routine accessed when a user application performs a "JMP" to address ØØØØH. At completion of the warm boot, BDOS passes control to CCP.

CONST checks the status of the console device. If a character is not ready, a value of ØØH is returned in the A register. If a character is ready, the A register contains Ø1H upon return.

CONIN reads the next console character into the A register and sets the high order bit to zero. If there is no character ready, the routine will wait until a character is available before returning.

CONOUT sends the character in the C register to the console output device. The character is expected to be ASCII data with the high order bit set to zero.

LIST sends the character in the C register to the currently selected list device. The character is expected to be ASCII data with the high order bit set to zero.

PUNCH sends the character in the C register to the currently selected punch device. The character is expected to be ASCII data with the high order bit set to zero.

READER reads the next character from the selected reader device into the A register. The high order bit is set to zero. An end-of-file is returned as an ASCII control-Z (Ø1AH).

HOME returns the disc head to track Ø on the selected disc device. On the HP disc drives, this is accomplished by calling SETTRK with a track value of Ø.

SELDSK selects the disc drive given by the C register as the current mass storage device. A value of ØØH selects drive "A": a value of ØFH selects drive "P." Each call to SELDSK returns the address of the Disc Parameter Header (DPH) for the selected disc in the HP register pair. An invalid disc specifier will return a ØØØØH in the HL register pair.

SETTRK positions the disc head at the track address specified in the HP register pair. The maximum track address for the HP 829Ø1 Disc is 42H (66 decimal); the maximum for HP-formatted 9895A discs is 96H (15Ø decimal).

SETSEC positions the currently selected disc head to the sector specified in the BC register pair. The range should be from ØØH through 1FH (31 decimal) for HP 829Ø1-type discs and from ØØH through 3BH (59 decimal) for HP 9895A-type discs. The sector can be selected either before or after the track selection.

SETDMA specifies the disc buffer address for all subsequent disc read or write operations. The address is passed in the BC register pair. The default value set during system initialization is ØØ8ØH.

READ will read data from the currently selected disc at the current track and sector address. The 128 bytes of data is stored into the selected DMA buffer. If an error prevents a successful completion, a value of Ø1H will be returned in the A register. A value of ØØH indicates a successful completion.

WRITE will output 128 bytes from the DMA buffer to the currently selected track and sector addresses. The A register will contain a ØlH if an error occurs, and a ØØH if the write is sucessfully completed.

LISTST returns the status of the currently selected list device. A value of ØØH is returned in the A register if the list device is not ready to accept data, while a value of ØlH indicates that data can be sent.

SECTRAN is used to compensate for the interleave factor on a particular disc. The controller on all HP discs automatically performs this for you: therefore, HP-formatted discs require no SECTRAN calls. SECTRAN will receive a logical sector number in the BC register pair and a translate table address in the HL register pair. The sector number is used as an index into the translate table, and the physical sector address will be returned in the HL register pair.

For simple device I/O, data is presumed to be ASCII. The high-order, or parity, bit is always set to "Ø". As with disc files, an ASCII control-Z is treated as an end-of-file.

In standard CP/M, all logical devices are mapped to physical devices through IOBYTE. The console, reader, and punch devices are established using escape sequence assignments.

Disc I/O is actually accomplished by using a sequence of calls to specific disc access subroutines which perform specific functions.

Typically, a call is made to select a specific disc; then track and sector addresses are selected; and a DMA buffer is specified. Once this has been done, the actual I/O can be performed. Note that this entire sequence is not necessary for each I/O request. For example, once a disc has been selected it will remain selected until another disc is selected using SELDSK. However, SECTRK and SETSEC should be used before each I/O request.

The READ and WRITE subroutines will actually perform the transfer of data to and from the disc. The controller will perform several re-entries if an error is encountered during a READ or WRITE operation.

5.3 Disc Tables

As with most disc operating systems, CP/M maintains tables of information on each type of disc drive as well as on each physical disc. These areas of memory are known as the Disc Parameter Tables and reside in BIOS.

Disc Parameter Header

Each of the 16 possible disc drives on the HP Series 8Ø Personal Computer

has an entry in the Disc Parameter Header (DPH) table. Each entry is 16 bytes in length. The format of a DPH entry is shown in table 5-2.

Table 5-2. Disc Parameter Header

Dec	Hex	Symbol	Description
ØØ Ø1	ØØ Ø1	XLT	Translation Vector
Ø2 Ø3	Ø2 Ø3	SCR	Scratch Area #1
Ø4 Ø5	Ø4 Ø5	SCR	Scratch Area #2
Ø6 Ø7	Ø6 Ø7	SCR	Scratch Area #3
Ø8 Ø9	Ø8 Ø9	DIRBUF	Directory Buffer
1Ø 11	ØA ØB	DPB	Disc Parameter Blk
12 13	ØC ØD	CSV	Checksum Vector
14 15	ØE ØF	ALV	Disc Space Vector

The meaning of each of these fields follows.

XLT is a translation vector which is used when a particular disc requires a sector interleave or skew. This is Ø for all HP-formatted discs.

SCR is a scratchpad area for use by BDOS. CP/M will store information in these bytes.

DIRBUF contains the address of a 128 byte block of memory used as a scratch area to hold directory information. BIOS uses the area during disc operations. Note that there is only one DIRBUF area system-wide, so all DPH entries point to the same buffer location.

DPB contains the address of the Disc Parameter Block for this type of disc. Each type of disc will have only a single DPB entry.

CSV holds the address of a scratchpad area of BIOS used to check whether a disc has been replaced by another disc. Note that, when CP/M determines that a disc has been changed, it is marked as "read-only" until a warm boot or reset disc call.

ALV is the address of another scratchpad area of BIOS. This area, unique for every DPH, maintains information on available space on the disc.

The SELDSK subroutine described above returns the address of the DPH for the disc being selected.

Disc Parameter Block

While each drive on the HP Series 80 Personal Computer has a DPH entry, a Disc Parameter Block (DPB) is maintained for each type of disc supported on the system.

At initial release, the CP/M operating system includes two entries in the DPB: one for the HP 82901 disc and one for the HP 9895A with HP-formatted discs.

The organization of each DPB entry is illustrated in table 5-3.

The values of the various fields are generally fixed by design, and should not be altered by the user. These values, and their significance, are given so you can better understand the internal operation of CP/M.

SPT is the total number of 128-byte sectors per track. It occupies two bytes.

BSH and BLM are two single byte fields which are fixed in relationship to the minimum data allocation, or group, size. BSH is the data allocation block shift factor, and BLM is the data allocation block mask.

Table 5-3. Disc Parameter Block

Dec	Hex	Symbol	Description
ØØ Ø1	ØØ Ø1	SPT	Sectors per Track
Ø2	Ø2	BSH	Block Shift Factor
Ø3	Ø3	BLM	Block Mask
Ø4	Ø4	EXM	Extent Mask
Ø5 Ø6	Ø5 Ø6	DSM	Data Max in Groups
Ø7 Ø8	Ø7 Ø8	DRM	Directory Entries Maximum
Ø9	Ø9	ALØ	Directory Data
1Ø	ØA	AL1	Directory Data
11 12	ØB ØC	CKS	Check Sum Vector
13 14	ØD ØE	OFF	OP SYS Reserved Tks

DSM is the maximum data block number on a particular disc. A block is BLS bytes in length, and corresponds to a "group." BLS, the minimum disc allocation size, will be discussed shortly. Since data blocks are numbered starting with Ø, the total number of data blocks in DSM+1. Two bytes are required for this field.

EXM is the extent mask, and is a function of DSM. It requires a single byte field.

DRM is the total number of directory entries (extents) which can be stored on the disc. Two bytes are required by this field.

ALØ/AL1 are two single-byte fields used to reserve directory blocks. Each bit of the ALØ/AL1 double word represents a group allocated to directory entries. Hence, a maximum of 16 groups can be reserved for directory space.

CKS is the size of the directory check vector (CSV) in bytes. It is a two byte field.

OFF is the number of tracks reserved at the beginning of the disc. Tracks are reserved for CP/M and for HP standard interchange format. The field is two bytes in length.

One additional factor, BLS, is calculated as a function of BSH and BLM. BLS is the size of a "group" on the particular disc. On the HP 82901 disc, BLS is 1024 bytes (1K); on the HP 9895 disc, the BLS is 4096 bytes (4K).

The CKS value is determined to be (DRM+1)/4 for removable media on both types of flexible disc drives.

As mentioned earlier, several DPHs can address the same DPB. On the HP 87, 16 entries exist in the DPH Tables, while two exist in the DPB Table. Each disc drive requires one DPH, while one DPB exists for each of the HP 82901 and the HP 9895.

Section
VI

THE CP/M TEXT EDITOR

6.1 Introduction

In order to prepare source language files for the assembler, or to create any other type of ASCII text file, there must be a way to input and edit data from the computer keyboard. There are many ways to create such files, but the primary tool intended for program development is the transient program ED.

The ED program is a character oriented text editor which provides the ability to create, modify, and store ASCII text files such as assembler source files. It is not a word processor, and should not be considered as such. It is simply a convenient way to create and edit text files.

To begin the ED, type:

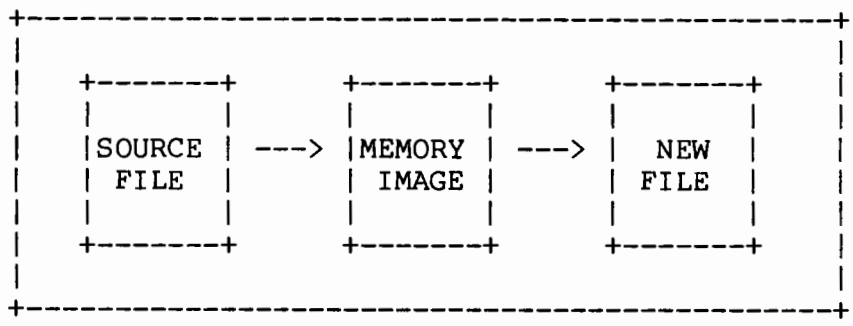
```
ED filename.extension
```

The file name and extension must be fully specified. The drive identifier is optional, unless the file resides on a disc other than the current drive.

If the specified file exists on the current (or specified) disc drive, a back-up copy will be created. This back-up will be named the same file name, with the file extension ".BAK". If the specified file cannot be located on disc, a back-up file containing zero records will be created, and ED will assume the file is new.

ED operates on a memory buffer image of the source file. Lines must be specifically appended into the memory buffer, operated upon, and written to the new source. This is illustrated graphically in table 6-1.

Table 6-1. ED Data Flow



6.2 Memory Image

Within memory, all operations occur at a location known as the "character pointer," or "cp." It will be useful to remember the "cp" whenever you are using the editor. In addition, ED maintains a similar pointer in the source file and in the destination file. The three pointers act independently, of course, so that ED can keep track of where text should be located.

When ED is first initiated, the memory buffer is empty. This is true whether the specified file exists or is new. Lines can then be appended into the buffer for existing files, or new lines can be inserted at "cp." In an empty buffer, the "cp" can be considered at the top of memory.

Lines of text within the memory buffer are illustrated in table 6-2. Note that the "cp" can be moved, using the commands described later in this chapter.

Table 6-2. Memory Image Buffer

```
This is the first line of text <CR> <LF>
This is line two of text <CR> <LF>
This is the "current" line <CR> <LF>
While this is the last line <CR> <LF>
```

The position of the "cp" is indicated below line three as a ^ character. <CR> and <LF> represent the carriage return and line feed characters, ASCII ØDH and ØAH, respectively.

6.3 ED Commands

The various commands available through ED to operate on the memory buffer are illustrated in table 6-3 below. The effect and use of each is described in the discussion which follows. Remember that the up arrow character (^) preceding a character signifies that the control key [CTRL] is pressed at the same time as the character. For example, ^C means control-C.

Table 6-3. ED Command Summary

A	Append text from disc.
B	Begin or bottom of buffer.
C	Move character positions.
D	Delete characters.
E	Exit ED and save file.
F	Find specified string.
H	Exit and re-start ED.
I	Insert text into memory buffer.
J	Juxtapose string.
K	Kill lines of text.
L	Move up or down lines.
M	Macro definition.
N	Find with autoscan.
O	Return to original file.
P	Move and print pages.
Q	Quit with no file changes.
R	Read library file.
S	Substitute string.
T	Type lines of memory buffer.
U	Upper/lowercase shift.
W	Write lines of text to disc.
X	Output to temporary LIB file.
Z	Sleep.
<CR>	Move and type lines.

Command: #

Name: Pound Sign

The pound sign is not technically an ED command. However, by specifying this character wherever a numeric value is permitted in other ED commands results in the value 65535 being substituted for the "#".

This has the effect of extending the command to all lines or occurrences, depending on the command being used.

Command: A

Name: Append

The append command copies lines of text from the source file into the memory buffer. The general form of the command is:

nA

ED maintains a next line pointer in both the input (source) and output (destination) files. Each occurrence of the "A" command loads one or more lines from the input file into memory starting at the next line pointer. Lines are appended into the memory buffer at the position of "cp." The next line pointer is, of course, updated after each append operation.

If no value is specified before the "A", a single line is read into memory from the input file. Specifying "#A" will load lines of text into memory until all lines are appended or until the memory buffer is full.

Specifying "ØA" acts like "#A", except that ED will stop appending lines when half of available memory is full.

Command: B

Name: Buffer Position

This command is used to move the location of "cp" to either the top or bottom of the memory text buffer. The format of this command is:

B/-B

To move to the top of the memory buffer (the beginning), specify the "B" command with no leading sign.

To move "cp" to the end of the memory buffer (the bottom), specify the command as "-B". Use this form to insert text after the last existing line in memory.

Command: C

Name: Character Position

The C command is used to move the "cp" one or more bytes "up" or "down" in the memory buffer. The general form of the command is:

+/-nC

Specifying a positive or unsigned value causes the "cp" to move to the right, proceeding to subsequent lines if necessary. A negative value moves "cp" to the left, and to previous lines as necessary.

Remember, a carriage return is a valid character, and "cp" must be moved two characters to skip over a CR/LF.

Compare the effect of this command with that of the "L" command, which moves "cp" lines at a time.

Command: D

Name: Delete Characters

The D command deletes one or more characters from the memory buffer relative to the character pointer "cp." The general format is:

+/-nD

A positive or unsigned value deletes characters at and to the right of "cp". A negative value causes those characters to the left of "cp" to be deleted.

Note that deleting a CR/LF has the effect of merging two lines of text!

Command: E

Name: Exit

The Exit command is the normal way to terminate an ED session. The format is:

E

All lines of text are written from memory into the output file. Any lines which have not been read into memory are automatically copied from the source file into the output file as well. The original source file is renamed to extension ".BAK", and the output file is renamed to the originally specified name.

Command: F

Name: Find

The F command is used to search memory for a specified character string. There are two forms of the F command:

nFstring^Z

nFstring[END LINE]

The command searches from "cp" through the end of the memory buffer. If no value of "n" is given, the first occurrence of "string" is found. If a value is given, that occurrence is found.

The "cp" will be positioned after the last character in "string" if it is found, or at the bottom of the buffer (-B) if no match is made.

The search string is terminated by a control-Z (^Z) or an [END LINE]. The ^Z terminator is used to separate multiple ED commands on a single line. The [END LINE] terminator initiates the ED command line.

To specify CR/LF as part of the string, use control-L (^L).

REMEMBER: The F command operates on text in the memory buffer only. Use the "N" command to search through the end of a document when part of it may not be in memory.

Command: H

Name: Restart ED

The "H" command is functionally identical to using the "E" (exit) command to copy all lines to the output file, then running ED once again using the newly created (output) file as input. The format of the command is:

```
H
```

The "H" command accomplishes the following tasks:

- o All lines in memory are written to disc.
- o Any lines not yet loaded into memory are transferred to the output file.
- o The original file is renamed to extension ".BAK".
- o The output file is renamed to the original file name.
- o ED starts again, using the same file name for input.

Since the disc I/O commands operate in a forward direction only (i.e., "A", "N", and "W"), "H" can be used to effectively move back in a file.

Also, whenever you use ED for an extended period of time, it is wise to use "H" often to write all changes you might make to a file to the disc. This helps minimize the loss of data in case of a power failure.

Command: I

Name: Insert Text

The insert command is used to insert (add) text into the memory buffer at the location of "cp." There are three general forms of the insert command:

```
I [END LINE]

Istring^Z

Istring[END LINE]
```

The first form is generally used when several lines of text are to be added to memory. All characters, including CR/LF, are inserted into the memory buffer until a ^Z (control-Z) is typed.

The second form is used to insert a short string with no CR/LF into the existing memory buffer at "cp." The ^Z is the string terminator.

The last form is a special case of the insert command. It is used to insert a single line of text into the memory buffer at "cp."

In all cases, the insert command causes all characters and lines in memory to be "pushed" down. Line numbers are automatically reassigned as new text is inserted between existing lines.

Several special control characters are defined in the insert mode.

^H : Same as [BACK SPACE], this deletes the last character typed.

^L : Inserts a "CR/LF."

^M : Same as [END LINE] this inserts a "CR/LF."

^R : Re-display current line.

^U : Deletes current line and lists a blank line below.

^X : Deletes current line by backspacing to beginning.

[CONT] : Deletes and displays the previous character.

REMEMBER: Insert mode considers the [END LINE] key as just another character ("CR/LF"). Insert mode is active until a control-Z is typed!

Command: J

Name: Juxtapose String

This command combines the effects of the "Insert" and "Delete" commands. The general format of the command is:

nJstr1^Zstr2^Zstr3^Z

This is how the command functions: ED searches from "cp" through the end of memory for the nth occurrence of "str1". To this point, "J" functions exactly as the "F" command. Next, "str2" is inserted into the memory buffer. This part of the command functions like the "I" command. Finally, all characters are deleted starting immediately after "str2" until the string "str3" is found. If "str3" cannot be found, no deletions are made. Assuming the "J" was successful, the "cp" will be located at the end of "str2".

Command: K

Name: Kill Lines

The kill command is used to delete entire lines from the memory buffer. The format is:

+/-nK

Specifying a positive value of "n" will delete all characters starting at "cp" until "n" CR/LF characters have been deleted. This has the effect of killing "n" lines of text.

A negative value of "n" will cause characters previous to the "cp" to be erased until the nth CR/LF is encountered. This has the effect of deleting the previous n lines of text.

Note that in the special case of n=1, characters on the current line are deleted up to the appropriate CR/LF. This means that, if "cp" is not at the beginning of a line, a "K" command will not delete all characters on the current line. Rather, only the characters that follow "cp" will be deleted. The same is true for a "-1K" command, except that previous characters are deleted.

Command: L

Name: Line Move

This command moves the "cp" up and down a line at a time in memory. The general format is:

+/-nL

The "L" command is to the "C" command as "K" is to "D". That is, specifying a positive or unsigned value of "n" will advance "cp" until n occurrences of "CR/LF." A negative value of "n" will move up in memory until n CR/LF characters are encountered.

The net effect is that the "cp" will advance or go back n lines.

Command: M

Name: Macro Definition

This command allows multiple commands to be executed as many times as desired, or until an error condition ends the macro. The general format is:

nM<cmd1><cmd2>...

In this case, each <cmdn> parameter represents a valid ED command string.

If the value of "n" is Ø or 1, the macro executes until an error is encountered. For example, the end of memory buffer is an error in the "Find" command. If any other value of "n" is entered, the macro is executed that number of times.

Command: N

Name: Find with Autoscan

This command operates on the memory buffer exactly as the "F" command does. However, this command will automatically scan through the end-of-file, loading additional text into memory, and writing lines to the output file, as necessary. The general format of the command is:

nNstring^Z

Except for its access to the disc, "N" functions exactly as the "F" command.

Command: O

Name: Return Original File

This command is used to restart the current ED session using the original input file as the source of data. The general format is:

O

The memory buffer is emptied, the output file is erased, and the line pointer in the source file is set to the beginning of that file.

This command functions like the "Q" command described below, except that the edit session continues in this case. The "Q" command terminates the edit immediately.

Command: P

Name: Print Pages

This command moves the "cp" and prints an entire page of the memory buffer to the display. The format of the command is:

+/-nP

A page in this case is a unit of 24 lines of text. Page boundaries exist invisibly in the memory buffer, and are moved about dynamically.

If a positive or unsigned value of "n" is given, the "P" command first advances the "cp" to the next page. Then, n consecutive 24-line screens are displayed.

If the value of "n" is negative, "cp" is moved back that number of pages, and all text from that point to the (original) "cp" location are displayed.

The special case where n=Ø results in the 23 lines which follow the "cp" being displayed to the screen.

Command: Q

Name: Quit

The Q command is used to quit the current session without making any changes to the file. The format is:

Q

The session is terminated immediately. No text is written from memory to the output file, since that file will be erased. The original file is left intact. Note, however, that the backup file will reflect the contents of the current file, and not reflect the previous backup file contents.

Command: R

Name: Read Library File

The "R" command allows text to be added from an existing disc file. The general format is:

```
Rname
```

The command illustrated above will result in the file "NAME.LIB" on the default disc drive being read into memory at "cp," in much the same way as the insert command operates.

The special case command "R" by itself will append the file "X$$$$$$$.LIB" into memory. This file is normally created using the "X" command, and the file is treated as a hold file by ED.

This command will only read library files, those with a file extension of ".LIB".

Command: S

Name: Substitute String

The "S" command is used to replace one string with another string. The general format of the command is:

```
nSstring1^Zstring2^Z
```

The string "string2" replaces "string1" for the first n occurrences of "string1".

Like the "F" command, "S" operates on the memory buffer only.

Command: T

Name: Type Lines

The T command types lines of the memory buffer to the screen. The general format of the command is:

+/-nT

A positive or unsigned value of "n" causes the next n lines to be displayed. A negative value of "n" prints the previous n lines of memory. The position of "cp" is not affected by the "T" command.

If "cp" is positioned at the beginning of a line, all characters on the line are typed. If "cp" is not at the beginning of a line, "T" by itslf prints only those characters to the right of "cp" through the CR/LF. A special case of "0T" causes the characters to the left of the "cp" on the current line to be displayed. By combining these two commands, the entire line may be seen by using the command "0TT".

Command: U

Name: Upper/Lower Case Translate

This command causes ED to perform automatic upper case conversion, or to end such conversion. The general form of the command is:

+/-U

If a positive or unsigned "U" is typed, all subsequent characters placed into the memory buffer will be shifted to uppercase. This is true, regardless of whether or not the characters are entered as lowercase.

A negative sign before the "U" will result in all text being left as is. If entered in uppercase, the text will remain uppercase. If entered in lowercase, the text will remain lowercase.

Note that this command affects the text entered into the memory buffer only. The effect of upper or lowercase commands is discussed in the text that follows these commands.

Command: V

Name: Verify Line Numbers

This command allows you to disable and re-enable automatic line numbering. The general format of the command is:

+/-V

The command typed with a leading plus sign or no sign turns on the line numbering. This is the default state when ED is first executed. If "-V" is typed, line numbering is disabled, and line numbers will not appear.

A special form of the "V" command allows you to determine the amount of memory available at any time. By typing:

ØV

ED prints a summary report indicating the number of bytes currently free followed by the total number of bytes available. Your file size, of course, is the second less the first. Both values are reported as decimal numbers.

Command: W

Name: Write Lines

This command writes lines of text from the memory buffer to the output file. The general format of the command is:

nW

This causes "n" lines of text to be written to the output file, starting at "cp". Only positive values of "n" are valid.

Using this command allows the user the flexibility to free part of the memory buffer so that additional lines of text can be appended from the input file.

Command: X

Name: Block Move

The X command performs an output of a block of text to a temporary disc file. The format is:

nX

This causes the "n" lines of text in memory to be written to a file called "X$$$$$$$.LIB" on the currently selected disc device. Lines are not deleted as they are written to the file.

The file remains active only for the current ED session. It is erased as part of a normal exit.

Command: Z

Name: Sleep

This command causes the ED program to wait. The general form of the command is:

nZ

As the value of "n" increases, so does the delay. When n = 10, the pause is approximately 3 seconds. A value of n = 100 delays almost 30 seconds.

This command can be used to delay the execution of a subsequent command. For example, this command can be inserted following a "P" command to create a pause between the printing of pages of memory. Generally, this function is more useful within macro definitions.

6.4 Sample ED Session

Since the ED program is one which must be used to be understood, a sample session with ED is illustrated below. Not all of the commands are shown here, but by working through this exercise you will begin to see which commands are most useful to your particular editing needs. Comments are separated from the ED session by semicolons. Note that the ED commands can be entered in either uppercase or lowercase letters.

```
A>ED TESTED.TXT  [END LINE]          ;Start ED, creating a new
                                     ;file named "TESTED.TXT" on
NEW FILE                             ;current drive, "A."
   : *i  [END LINE]
  1:  This is line 1.  [END LINE]    ;Enter lines of text.
  2:  While this is line 2.  [END LINE]
  3:  ^Z                             ;Control-Z ends insert.
   : *b#t  [END LINE]                ;Top of buffer, type all
  1:  This is line 1.                ;lines of text in buffer.
  2:  While this is line 2.
  1: *  [END LINE]                   ;cp stays at top of buffer
                                     ;and [END LINE] types and advances
                                     ;a line, similar to T command.
  2:  While this is line 2.          ;[END LINE] advances and types
  2: *i  [END LINE]                  ;a line.  Insert at "cp."
  2:  This is a new line between 1 and 2!
  3:  ^Z                             ;Control-Z ends insert
  3: *b#t  [END LINE]                ;Top of buffer, type all
  1:  This is line 1.                ;lines of text in buffer.
  2:  This is a new line between 1 and 2!
  3:  While this is line 2.

  1: *2L  [END LINE]                 ;Advance 2 lines w/o typing.
  3: *T  [END LINE]                  ;Type a line.
  3:  While this is line 2.          ;Current line 3.
  3: *14CT  [END LINE]               ;Advance 14 characters, type
     line 2.                         ;from "cp" to EOL.
  3: *ithe original ^Z               ;Insert at cp, between "is"
                                     ;and "line". End w/ Control-Z.
   : *b#t  [END LINE]                ;Top of buffer, type all
  1:  This is line 1.                ;lines of text in buffer.
  2:  This is a new line between 1 and 2!
  3:  While this is the original line 2.
  1: *1L  [END LINE]                 ;Advance 1 line.
  2: *T  [END LINE]                  ;Type the current line.
  2:  This is a new line between 1 and 2!
  2: *sa new^Zan old ^Z              ;Substitute new for old.
  2: *LT  [END LINE]                 ;Advance and type a line.
  3:  While this is the original line 2.
```

```
   3: *b#t  [END LINE]                ;Top of buffer, type all
   1:  This is line 1.                ;lines of text in buffer.
   2:  This is an old line between 1 and 2!
   3:  While this is the original line 2.
   1: *E  [END LINE]                  ;End of edit. Save file.

A>                                    ;Warm boot after exiting ED.

A>ED TESTED.TXT  [END LINE]           ;Re-edit file A:TESTED.TXT
   : *B#T  [END LINE]                 ;Top of buffer, type all
                                      ;lines of text in buffer.
                                      ;Nothing there - memory buffer
                                      ;is empty.  Must append text.
   : *#A  [END LINE]                  ;Append all lines from file.
   1: *B#T  [END LINE]                ;Top of buffer, type all
   1:  This is line 1.                ;lines of text in buffer.
   2:  This is an old line between 1 and 2!
   3:  While this is the original line 2.
   1: *-B  [END LINE]                 ;Go to bottom of buffer
    : *+U  [END LINE]                 ;Convert to upper case.
    : *I  [END LINE]                  ;Insert text at "cp."
   4:  This is a new line inserted in uppercase.  [END LINE]
   5:  ^Z                             ;End insert with Control-Z.
    : *-U  [END LINE]                 ;Preserve upper/lowercase.
    : *-LT  [END LINE]                ;Back a line and type.
   4:  THIS IS A NEW LINE INSERTED IN UPPERCASE.
                                      ;This new line is uppercased
                                      ;because the insert command
                                      ;was uppercased. Note how
                                      ;the "U" feature has been
                                      ;toggled.
   4: *b#t  [END LINE]                ;Top of buffer, type all.
   1:  This is line 1.
   2:  This is an old line between 1 and 2!
   3:  While this is the original line 2.
   4:  THIS IS A NEW LINE INSERTED IN UPPER CASE.

   1: *LT  [END LINE]                 ;Advance and type one line.
   2:  This is an old line between 1 and 2!
   2: *s2^Z3^Z                        ;Change first 2 to 3.
   2: *L2T  [END LINE]                ;Back a line, type 2.
   1:  This is line 1.
   2:  This is an old line betwen 1 and 3!
   1: *E  [END LINE]                  ;End of edit. Save files
                                      ;rename existing file.
```

Section
VII

ASSEMBLY LANGUAGE UTILITIES

The material in this section is devoted to the CP/M commands used for developing assembly language programs. These include the ASM, LOAD, DUMP, and DDT transient commands and the built-in SAVE command.

7.1 Introduction to ASM

In order to write computer programs in machine language, some tool must be utilized to translate English-like program lines into binary opcodes which are meaningful to the Z80 CPU. Such translation does not occur in interpretive languages such as BASIC. However, true compilers perform this task from high level languages such as COBOL and Pascal. The translator program for assembly language source statements is called an assembler.

Several different assemblers are available for 8080 and Z80 computers. Since the Z80 instruction set is a superset of the 8080 instructions, both Z80 and 8080 assemblers can be used with your CP/M system. All of these assemblers generate the same binary opcode for common instructions. However, the mnemonic commands understood by Z80 and 8080 assemblers is often different.

The assembler provided with your CP/M system is the standard CP/M 8080 Assembler.

7.2 Using the Assembler

The CP/M Assembler, ASM, accepts a source code input file prepared using the ED program or some other text editor. The general format of the command which initiates ASM is:

```
ASM filename
```

The file name is required to have the file extension ".ASM" in order to be processed by ASM. The assembler program will produce two output files as it executes. The first contains the assembled object code as a series of hexadecimal characters, and is given the same name as the source file with the file extension of ".HEX". The second output file is called by the same file name, except the file is of extension ".PRN". This file contains the printer listing of the assembly, and contains the source listing and comments along with the hex representation of the program. Addresses are also shown in the ".PRN" file.

If the ASM program is executed using the format above, the source file must be on the currently logged disc, and the two output files will be created on the same disc. This is true even if ASM is loaded from another drive. For example, if the "A" disc is the default drive and the ASM program is on disc "B," typing the command below will assume the source and output files are to be on "A."

```
B:ASM PROG1
```

To specify another drive for either the source or output files, ASM permits a three byte coded sequence to follow the file name in the command. While these three bytes appear to be a file extension, remember that ASM requires extension ".ASM" and the three bytes are coded instructions. These bytes direct ASM to other than the current drive. For example:

```
ASM filename.BCD
```

is coded as follows:

- The first byte, "B" in the above example, specifies the disc where the source file is located.
- The second byte, "C" above, specifies the disc where the hex file called "filename.HEX" will be placed.
- The third byte, "D" above, specifies the disc where the printer listing file "filename.PRN" will be placed.

If a "Z" is specified in the second or third byte positions, the respective file will not be created. This is useful for test assemblies or when a listing is not required.

Of course, if the currently selected disc is the desired drive for all three files, none of these parameters is required.

Some examples of using ASM are given in table 7-1.

Table 7-1. ASM Execution Commands

COMMAND	FILE USAGE	
ASM HBS	Input:	A: HBS.ASM
	Output:	A: HBS.HEX
		A: HBS.PRN
ASM PAY.BCD	Input:	B: PAY.ASM
	Output:	C: PAY.HEX
		D: PAY.PRN
ASM PAY.EAZ	Input:	E: PAY.ASM
	Output:	A: PAY.HEX
		(No PAY.PRN)

As mentioned earlier, the assembler source code can be entered into the ".ASM" file using ED, the CP/M Text Editor, or any other text editor that creates ASCII text files.

Now that you know how to enter source code, and how to use ASM, let`s look at what must be in an assembler source file.

7.3 Assembly Language Programming

Program Format

The ASM program expects a line of source code to consist of up to five fields: a line number, a label, an operation code mnemonic, an operand, and a comment. Not each line requires every field: which are required and which are optional often depends upon the operation code, or instruction, in question. Fields are separated by one or more spaces or tab (^I) characters. Often, however, a programmer will set and use tabs so that a source program is more readable. Using ED permits such tabbing without any special action on the part of the programmer.

The general format of a source line is:

```
line#  label  instruction  operand  ;comment
```

Each line of assembly language code is terminated by a "carriage return/line feed", which is automatically generated by ED. Optionally, multiple source statements may be included on each line by inserting an exclamation mark "!" between statements.

Let's take a closer look at each of these fields.

The LINE NUMBER is an optional field. If included, it should be a decimal value between 1 and 99999. Neither the sequence nor the interval between numbers is significant to the Assembler, and the field is ignored during assembly.

This field is permitted because some text processes create source files with leading line numbers. ED does not do so. Generally, ASM source code does not include line numbers.

The LABEL FIELD is an optional field, and can include from 1 to 16 characters. All characters are significant, except that dollar signs can be inserted within a label to improve readability. The "$" character is not included in the maximum 16 character count. If no line number is specified, the label may start in the first byte of a line. However, if a line number is included, at least one space must be included between the line number and the label.

A label may also be followed by a colon (":"), although a space must be included to separate the label from the instruction field.

All alphabetic characters are treated as uppercase, whether or not entered as such.

The OPERATION FIELD contains a valid Intel 8080 instruction mnemonic, or an assembler directive. The valid 8080 mnemonics are included in appendix G, and the most commonly used are discussed later in this section. An assembler directive is an instruction to ASM providing information for the duration of the Assembly. Typically, directives generate no direct output code.

The OPERAND FIELD, discussed in depth below, usually contains a numeric value, an address, a register designation, or a constant. The operand can also be an expression which evaluates to one of the above.

The COMMENT FIELD is included in the ASM listing, but is otherwise ignored by the Assembler. Of course, no comments are included in the hex output file, and require no memory in the fully assembled and loaded program.

All characters on a line following the semi-colon and preceding the "CR and LF" are considered to be part of the comment field and are ignored by ASM.

The Value of a Label

A label, as mentioned above, is used to reference an 8080 statement. During the assembly, a label is assigned a value. Whether that value is user specified or an address assigned by ASM depends on the instruction or directive.

If the instruction field is an instruction, such as "MVI", the label is assigned the address of the instruction. The label is the address of the first byte of the statement.

If the instruction field contains an assembler directive, the value assigned to the label will vary depending on the directive. This will be discussed in conjunction with each directive later in the section.

Forming the Operand

The operand field, as mentioned above, may contain a label, an expression, or a data byte. The type to be used in any particular statement is a function of the instruction specified and the required task. However, we can generalize about each of these types of operands, and when each might be useful.

When a LABEL is included in the operand field, it can be either an 8-bit value or a 16-bit address, depending upon the instruction labeled. If the label appears on a line with a Z8Ø instruction or a define memory instruction, the label is replaced with the address of the instruction or first data byte.

If the current labeled line contained a "EQU" or "SET" assembler directive, the label assumes the value of the operand field in that (labeled) statement. Whether the value assigned to the label in the current line will be an 8-bit or 16-bit instruction will depend on the instruction included on the current line. If the instruction requires an 8-bit value, only the least significant 8 bits of the labeled statement's operand will be used. When the instruction expects a 16-bit value, a full 16 bits are made available as the label value. Leading zeros are appended as necessary.

When a label appears in an operand field, its value is substituted by ASM. Remember: the VALUE of a label is an ADDRESS of or the label's OPERAND in an "EQU" or "SET" directive.

The operand field can contain a NUMERIC CONSTANT. This is always a 16-bit value, but may be specified in any one of several bases. The base, or radix, of a numeric constant is determined by the radix indicator which follows the value. Valid radix indicators are given in table 7-2. If no radix indicator is specified, the value is assumed to be base 1Ø, decimal.

Table 7-2. Radix Indicators

INDICATOR	RADIX	NUMBER BASE
B	Binary	Base 2
O	Octal	Base 8
Q	Octal	Base 8
D	Decimal	Base 10
H	Hexadecimal	Base 16

The letter "Q" is provided as an alternate specifier for base 8 since the letter "O" is so easily confused with the digit "Ø". Either can be used interchangeably.

A constant is therefore composed of a string of valid digits in the selected base. For example, in addition to numeric characters common to base 2, base 8, and base 1Ø, the following alphabetic characters are permitted in base 16: A, B, C, D, E, and F. However, the leading digit of a hex numeric constant must be numeric in order for the assembler to recognize the field as a value and not as a label. Placing a leading zero before a hex value will accomplish this task.

As with labels, a dollar sign may be embedded in a numeric value to improve readability.

Table 7-3 contains examples of valid numeric constants in the permitted bases. Notice that lowercase radix indicators are treated as if they were uppercase.

Table 7-3. Sample Valid Numeric Constants

3738	4382D	1111$Ø1Ø1$ØØØØ$Ø11Ø	
3738h	ØBCDH	177$77Q	177777o

Remember: a numeric constant, regardless of its specified base, must evaluate to a 16-bit binary value. If more than 16 bits are provided, ASM will truncate the number to the most significant 16 bits.

STRING CONSTANTS can also be specified in the operand field for both directives and instructions. Multiple character strings are formed by enclosing the desired string in single quotes ('). In most cases, the string length is restricted to one or two ASCII characters, depending on the instruction which will operate upon the string. The "DB" directive described later is an exception to this rule.

All strings must be fully contained on a single line of source text, thus permitting the "!" character within the string. Also, strings may never exceed 64 characters in length. The single quote character may be included in a string by placing two single quotes side by side. Examples of valid string constants are illustrated in table 7-4.

Table 7-4. Sample String Constants

```
'A'        'MK'          'MSB'      'Hello'

''''''     '''If it works, don't fix it!'''

          'The HP-87 Personal Computer'
```

ARITHMETIC EXPRESSIONS, including LOGICAL OPERATORS, may be part of the operand field. ASM recognizes and can evaluate several operators between any two arguments "op1" and "op2" as shown in table 7-5. In that table, "op1" and "op2" represent simple numeric constants, one or two character string constants, or reserved words discussed later in this chapter. The arguments may also be fully enclosed parenthetical expressions involving any of the above.

Table 7-5. Valid Arithmetic Operators

OPERATION	EFFECT
opl + op2	Unsigned arithmetic sum of opl and op2.
opl - op2	Unsigned difference between opl and op2.
+ op2	Unary plus equivalent to op2.
- op2	Unary minus equivalent to (Ø - op2).
* op2	Unsigned magnitude multiplication.
opl / op2	Unsigned magnitude division.
opl MOD op2	Remainder function of (opl / op2).
NOT op2	Logical complement of op2.
opl AND op2	Bit-by-bit logical AND.
opl XOR op2	Bit-by-bit logical exclusive OR.
opl SHL op2	Result of shifting opl LEFT op2 bits.
opl SHR op2	Result of shifting opl RIGHT op2 bits.

Note that all computations are performed by ASM at the time of assembly. The expression which results must be appropriate for the instruction used. For example, if the instruction expects the expression to evaluate to an 8-bit value, the high-order 8 bits must be zero or an error will result. Thus, for example, an "ADI -1" cannot be used since the "-1" evaluates to the 16-bit value OFFFFH. Instead, an expression that zeros the high order bits can be used: "ADI((-1 AND OFFH)". The "OFFH" will force the high eight bits to zero, making the expression acceptable.

As with most high-level programming languages, ASM assigns a precedence of operation so that the programmer can code expressions without the need for multiple levels of parentheses. The order of precedence or operators is:

Highest precedence:	*	/	MOD	SHL	SHR
	-	+			
	NOT				
	AND				
Lowest precedence:	OR	XOR			

Operators which appear on the same line above are of the same precedence. When two or more of these operators appear in the same expression, the expression is evaluated from left to right. Table 7-6 illustrates several pairs of equivalent expressions. The expressions on the left evaluate as if the parentheses were placed according to the expressions on the right.

Table 7-6. Sample Operator Precedence

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 c	a OR (b AND c)

Finally, there are several Reserved Words and expressions in ASM that assume special significance in the operand field. These reserved words are shown in table 7-7 below, and include the names of each of the 8080 registers and the special ASM program counter.

Table 7-7. Reserved Operand Words

Reserved Word	Significance
A	A Register
B	B Register or BC Pair
C	C Register
D	D Register or DE Pair
E	E Register
H	H Register or HL Pair
L	L Register
M	Data at address in HL Pair
$	Current PC Address
SP	Stack Pointer
PSW	Processor Status Word

The Processor Status Word is a 16-bit value made up from the A register and the 8-bit "Flag" byte, and is treated as a single unit for stack operations.

The current PC address, when it appears in an operand field not as a clarification character in expressions, is assigned the current value of the program counter maintained by ASM during the assembly.

7.4 Assembler Directives

Assembler directives are valid entries in the instruction field of 8080 assembly language source code. However, a directive is not an 8080 or Z80 instruction code. An assembler directive is a command to the ASM program to aid in the assembly.

Some directives are used to name, define, or reserve memory locations; others permit logical variables to be defined, thus allowing conditional assembly of parts of code; or to instruct ASM where in memory the source code should be located.

The directives to be discussed here are summarized in table 7-8.

Table 7-8. Assembler Directives

Directive	Meaning
DB	Define byte(s) of data.
DS	Define data storage.
DW	Define word(s) of data.
SET	Set logical values.
IF	Begin conditional assembly.
ENDIF	End conditional assembly.
EQU	Numeric "equate."
ORG	Specify program / data origin.
END	End of Assembly.

DB : Define Byte(s) of Memory

This directive is used to define one or more 8-bit bytes of data. Each byte, of course, occupies one byte of memory, so this directive can be used to define memory in either ASCII character strings or in numeric byte values. "DB" is also the only directive that allows the definition of more than two character strings.

The general form of the directive is:

```
DB    val
```

The "val" parameter may contain any numeric value which will fit in eight bits. This represents Ø through 255 decimal, or Ø through ØØFH.

If ASCII character codes are to be entered into memory, "val" may contain one or more ASCII characters enclosed in single quotes. Some examples of the "DB" command are included in table 7-9.

Table 7-9. Sample Use of the DB Directives

```
CR        DB      13      ;"Return" is ASCII 13

LF        DB      1Ø      ;"Line feed" is ASCII 1Ø

MESG      DB      'This is a line of text'

LOG       DB      LF AND ØFFH
```

As you can see, numeric and string values can be included within the same "DB" directive. In the case of the "MESG" line above, "MESG" will become the Address of the first byte of the string, the letter "t". The "h" is at address "MESG + 1", and so forth. Note that, as in the last line of the example, an expression may be used as long as it will evaluate to an eight-bit value.

DS : Define Data Storage

The "DS" directive is used to reserve a specific number of bytes of memory. If there is a label on the "DS" line, that label is the address of the first byte of memory reserved.

By using the "DS" directive, a programmer can hold a number of memory locations available for later data storage. This can also be done by using the "ORG" directive to simply advance the program counter to the desired location as you will see later.

The principle difference between the "DS" and the "DB" directives is that with "DS," the operand field specifies a number of bytes, not a value. Both the "DB" and the "DW" directives use the operand field to define value of memory locations.

DW : Define Word

This directive is similar to the "DB" directive discussed above, except that "DW" expects 16-bit values to be included in the operand. As with "DB," "DW" allows an expression, a numeric value, or a one or two-byte string. Table 7-1Ø illustrates some valid examples of "DW." Note that the least significant byte is stored first to be consistent with the 8Ø8Ø/Z8Ø.

Table 7-1Ø. Sample DW Directives

```
+-------------------------------------------------------+
|                                                       |
|    MAX     DW      OFFFFH     ;Maximum 16 bit value   |
|                                                       |
|    INTLS   DW      'MK'       ;Two ASCII bytes        |
|                                                       |
|    CRLF    DW      13 * 256 + 1Ø                      |
|                                                       |
|    DEC     DW      65535      ;Biggest decimal value  |
|                                                       |
+-------------------------------------------------------+
```

SET : Set Label Value

By using this directive, values can be dynamically assigned to label names during assembly. This, along with the "IF" directive, can permit certain parts of a program to be assembled based upon the value of labels.

The "SET" directive is the only type of statement which allows a label to appear on more than one line. The value assigned to the line label is valid only until the next "SET" directive assigns another value.

Examples of the "SET" directive are included below, under the "IF" directive.

IF : Begin Conditional Assembly
ENDIF: End Conditional Assembly

These two directives, which must always appear in matched pairs, are used to perform a conditional assembly.

The format is:

```
IF expression
  STATEMENT
  .....
  STATEMENT
  ENDIF
```

In the form shown above, the statements will be assembled only if the value of "expression" evaluates to non-zero.

If the expression evaluates to zero, the statements are listed in the ".PRN" file but are not assembled.

EQU : Equate

This directive permits a numeric value to be assigned to a specific label for the duration of the assembly. In this way, it is very similar to the "SET" directive. However, a label used in an "EQU" directive may only be defined once in a given assembly.

"EQU" is most often used to allow key numeric values to be referred to by label. Wherever the label of the "EQU" directive is used in a subsequent operand field, the value specified in the "EQU" operand is assembled.

ORG : Program/Data Origin

This directive is used to alter the location at which machine code generation will take place.

The value in the operand field is a constant or expression which evaluates to a 16-bit address.

Any number of "ORG" directives can be used in a program to control assembly. No checking is made to assure CP/M or any other part of memory is not over-written.

CP/M programs must initially "ORG" at address 0100H.

END : End Assembly

This directive is used to signal the end of an 8080 assembler program. While it is not required, it is good program practice to mark the end of the source program with the "END" directive.

7.5 Operation Codes

While assembler directives are used within the instruction field in 8080 source code, it is the operation code mnemonics which actually direct the processor in performing any specific task. The mnemonics used by the CP/M assembler are the standard Intel 8080 instructions, and are completely documented in a variety of well-written commercially-available texts.

A full description of each operation code is beyond the scope of this section. Nonetheless, a brief summary of the instructions is provided for your benefit.

The conventions and standards used are summarized in table 7-11.

Table 7-11. Conventions

Symbol	Description
data	An 8-bit value or expression.
addr	A 16-bit value, expression, or label.
r	Any 8-bit register or "M."
rp	Any 16-bit register or stack pointer. or program counter "S" or "P."
n	Contents of specified register or memory location.
M	Data at address (HL).

The instructions are grouped into arbitrary groups as follows:

Program Control - Jumps, Calls, and Returns
Immediate Operand Instructions - Data Specified
Increment and Decrement Instructions
Data Movement Instructions - Moves, Loads, and Exchanges
Arithmetic and Logical Instructions
Processor Control Instructions

Each group will be presented, showing the instructions and a brief description of the function of each.

Program Control

This group includes the jump, call, and return instructions which allow a variety of conditional program execution. The different forms of the instructions test the condition flags in the Flags register, and act accordingly. Except as noted, these instructions all reference a 16-bit address or line label in the operand.

The group includes:

```
JMP  addr      Unconditional Jump.
JNZ  addr      Jump on non-zero condition.
JZ   addr      Jump on zero condition.
JNC  addr      Jump on no carry condition.
JC   addr      Jump on carry condition.
JPO  addr      Jump on odd parity condition.
JPE  addr      Jump on even parity condition.
JP   addr      Jump on positive condition.
JM   addr      Jump on negative condition.

CALL addr      Unconditional call subroutine.
CNZ  addr      Call on non-zero condition.
CZ   addr      Call on zero condition.
CNC  addr      Call on no carry condition.
CC   addr      Call on carry condition.
CPO  addr      Call on odd parity condition.
CPE  addr      Call on even parity condition.
CP   addr      Call on positive condition.
CM   addr      Call on negative condition.

RET            Unconditional return from subroutine.
RNZ            Return on non-zero condition.
RZ             Return on zero condition.
RNC            Return on no carry condition.
RC             Return on carry condition.
RPO            Return on odd parity condition.
RPE            Return on even parity condition.
RP             Return on positive condition.
RM             Return on negative condition.

RST  n         Programmed "Restart."
```

The "Restart" instruction is a special single-byte call instruction. The "n" parameter is an integer value or expression which evaluates to a value between Ø and 7. It translates into a CALL to the address equivalent to 8 * n. By placing a JUMP command at that address, RST permits indirect addressing sometimes found in other processors.

Immediate Operand Instructions

The instructions in this group typically specify a one or two-byte value as part of the instruction: hence, the name immediate operand. The instructions either load a memory address or a register, or perform arithmetic or logical operations.

MVI	r,data	Move data byte into specified register.
ADI	data	Add operand to (A) without carry.
ACI	data	Add operand to (A) with carry.
SUI	data	Subtract operand from (A) without carry.
SBI	data	Subtract operand from (A) with borrow.
ANI	data	Logical AND operand with (A).
XRI	data	Logical exclusive OR operand with (A).
ORI	data	Logical OR operand.
CPI	data	Compare operand with (A) and set flags as if an SUI had been done.
LXI	rp,data	Move data bytes into register pair.

Notice that the borrow mentioned above is actually the state of the carry flag during subtraction operations. The CPI instruction affects the flag as if a "SUI" were the instruction, except that the contents of A are not changed: only the condition flags are set.

Increment and Decrement Instructions

This group of instructions permits a single register or a specific register pair to be incremented or decremented by one. This can be used in looping, or in arithmetic operations where the operand is "1."

INR	r	Increment specified register.
INX	rp	Increment specified register pair.
DCR	r	Decrement specified register.
DCX	rp	Decrement specified register pair.

Data Movement Instructions

The instructions in this group are used to move data from memory to the CPU registers and back again. Between-register moves are also included here.

```
MOV  r1,r2     Move contents of r2 into r1.

LDAX rp        Load A from memory addressed by register pair.
STAX rp        Store A into memory addressed by register pair.

LHLD addr      Load HL register pair from address.
SHLD addr      Store HL register pair to address.

LDA  addr      Load A register from address.
STA  addr      Store A register into address.

POP  rp        Load register pair from stack.
PUSH rp        Store register pair onto stack.

IN   data      Load A register from CPU I/O port "data."
OUT  data      Store A register to CPU I/O port "data."

XTHL data      Exchange data from top of stack with HL.
PCHL           Load program counter from HL.
SPHL           Load stack pointer from HL.

XCHG           Exchange DE and HL.
```

The "LDAX" and "STAX" instructions require the operand to be either the BC or the DE register pair ("B" or "D").

Both the "PUSH" and "POP" instructions affect the contents of the stack pointer after the data is stored or loaded. The operand must be "B", "D", or "H" to specify a register pair, or it must be "PSW" to specify the A register and the Flags word.

The special case of "MOV M,M" is not allowed, although other registers may be doubly specified.

Arithmetic and Logical Operations

These instructions act to perform single precision arithmetic or logical operations upon the A register. The Flag word will be affected by the result.

ADD	r	Add register to A without carry.
ADC	r	Add register to A with carry.
SUB	r	Subtract register without borrow.
SBB	r	Subtract register with borrow.
ANA	r	Logical AND register with A.
XRA	r	Exclusive OR register with A.
ORA	r	Logical OR register with A.
CMP	r	Compare register with A.
DAA		Decimal adjust A register.
CMA		Complement A register.
STC		Set carry flag.
CMC		Complement carry flag.
RLC		Rotate register A left. Carry is copy of LSB.
RRC		Rotate register A left. Carry is copy of LSB.
RAL		Rotate carry flag and register left.
RAR		Rotate carry flag and register right.
DAD	rp	Double precision add rp with HL.

In the "DAD" instruction, the contents of the specified register pair is added to the HL register pair, with the result stored in the HL register. The register pair (rp) must be B, D, H, or SP.

Control Instructions

The remaining instructions specify direct control of the processor and its interrupt system.

HLT	Halt operation of the CPU.
DI	Disable interrupts of the CPU.
EI	Enable interrupts of the CPU.
NOP	No operation, a place holder for memory locations.

7.6 Assembler Execution

When the ASM program executes, it will display several message lines. Assuming that there are no assembly errors,the message will look something like the illustration in table 7-12 below.

Table 7-12. Sample ASM Messages

```
+-------------------------------------------------------+
|                                                       |
|       CP/M ASSEMBLER - VER 2.Ø                        |
|       Ø21F                                            |
|       Ø12H USE FACTOR                                 |
|       END OF ASSEMBLY                                 |
|                                                       |
+-------------------------------------------------------+
```

The value on the second line, Ø21F in the example above, is the address of the next free location in memory following the assembled program. It is a hexadecimal number.

The Ø12H is a hexadecimal value which indicates the relative usage of the ASM Symbol Table. As the assembler executes, it has only a limited amount of space in which to store labels and their appropriate addresses for a program. This is called a Symbol Table Area. This value indicates how much of the maximum symbol table area was actually used during the assembly.

The value reported will be in the range of ØØØH through OFFH. In the example above, Ø22H is equivalent to 34 decimal. The percentage of the symbol table used during this assembly is

$$\frac{34}{255} = 13.3\%$$

Any value reported can be calculated similarly.

Note that the "END OF ASSEMBLY" message does not necessarily indicate a successful assembly. Error messages, discussed in the next few pages, will be summarized at the console between the first and second lines in the example above. To be certain that the assembly has no errors, you should check the ".PRN" file generated by ASM.

When you look at the ".PRN" file, either with the TYPE command or with a text editor, you will see some additional hexadecimal values to the left of each source code line. These values are the memory addresses and absolute hex code which represent the program you have written.

The first position on a line should either be blank or contain one of the error codes described later in this section.

Next you will see either an address or a value, depending upon the contents of the instruction field. If the instruction is the "EQU" or "SET" directive, the number will indicate the value of the label. You will also see an equal sign "=" following the value. Other instructions generally result in an address being printed in the field. In the case of instructions that generate more than one byte of code, the address is the location of the first byte of the instruction or data.

The next several columns before the source code line represent the actual hex code generated by ASM. It shows the contents of the address(es) indicated in the address column.

Note that, even in the ".PRN" file, source comments are maintained. Should your source file be destroyed, you can recover by using ED on the ".PRN" file and deleting the first several columns.

7.7 Error Messages

There are two classes of errors which may be generated during execution of ASM. The first, which are generally caused by disc or file errors, are fatal and will terminate execution. For example, if ASM cannot locate the specified source file to assemble, it cannot continue.

The second class of errors are due to syntax and source code statement mistakes. The assembler will attempt to complete if such errors are encountered, and will report the error line at the console and flag the line in the ".PRN" file. These types of errors are discussed below.

Fatal Errors

Message	Action
NO SOURCE FILE PRESENT	ASM cannot locate the source file you specified. Check the disc code or spelling and try again.
NO DIRECTORY SPACE	The disc directory is full: no more files may be created. You must erase some files to make room for the ".PRN" and ".HEX" files.
SOURCE FILE NAME ERROR	No wild card specifiers may be used in the ASM command.
SOURCE FILE READ ERROR	ASM cannot read the source file specified. Possibly the disc has a bad spot, or the file contains "garbage" data. Check to see whether the file contains what you think it contains using the TYPE command.
OUTPUT FILE WRITE ERROR	This means ASM cannot write to disc, either because it is write- protected or because it is full. Use the STAT command to determine which is the case.
CANNOT CLOSE FILE	ASM could locate the file but could not write anything to the disc. Check to see if the disc is write-protected.
DISC ERROR ON DRIVE "x"	You have specified an invalid drive code in the ASM command. Remember that ASM requires file extension ".ASM" and that any file extension specified indicates the disc code for ASM files.

Source Code Error Messages

If the ASM program ever finds a line it cannot assemble, it will report an error condition. This will be displayed both at the screen and in the ".PRN" file. The first position on a line which contains an error will be one of the letters shown below. The entire source line, along with its hex code equivalent, is also displayed.

The error codes which may appear are:

D — Data Error. The data specified does not fit into the number of bytes provided. Data may be truncated.

E — Expression Error. ASM cannot evaluate the expression properly.

L — Label Error. A label has been used incorrectly. This can happen when a label is duplicated in more than one source line, or when a label cannot be used as an operand.

N — Invalid Feature. You have used a feature of some other assembler which cannot be used in ASM.

O — Overflow Error. The expression is too complex for ASM to evaluate. Simplify it by using more than one statement.

P — Phase Error. A label changes value during an assembly, which is allowed only wth the "SET" directive.

R — Register Error. The register specified is not valid with the instruction used. Check to make sure the instruction permits the register specified.

U — Undefined Symbol. A label is used in an expression but has never been assigned a value.

V — Value Error. The operand is not correctly formed. Check for typing errors.

7.8 The LOAD Command

Once a program has been assembled using ASM, it is necessary to load the ".HEX" file into memory and convert the file to a ".COM" file. The LOAD command turns this file into a command file that is executable as a transient program by entering:

```
LOAD filename
```

The file name must correspond to a previously assembled ".HEX" extension file. An optional drive identifier plus colon may precede the file name parameter. No errors should occur when a properly assembled program is loaded. If an error occurs, check the ".PRN" file to determine what error has been made. The most common error, "INVERTED LOAD ADDRESS," occurs when there no "ORG Ø1ØØ" directive as the first line of assembly language code.

7.9 The DUMP Command

The DUMP command is used to display the contents of a file on the display screen in hexadecimal form. The format of this command is:

```
DUMP file
```

The file can be of any type, including ".HEX" and ".COM" extension files.

7.1Ø Introduction to DDT

DDT is the name of the CP/M "Dynamic Debugging Tool", a program designed to help test assembly language programs. By using DDT, you can single step through an assembled and loaded file just as you might trace a program in BASIC. At each breakpoint, you can examine the 8Ø8Ø registers; modify those registers or memory locations; and even assemble new source code in-line.

7.11 Starting DDT

As with many other CP/M transient utilities, DDT can be initiated in more than one way. The general form,

```
DDT
```

loads the DDT program from the disc. The program is now ready to accept user input. Any of the commands described in the next section are allowed.

Generally, DDT is loaded in order to work on a specific program file. This file can be either a fully assembled and loaded ".COM" file, or a ".HEX" file which has not yet been loaded.

While there are DDT commands to load a program to be tested, the name of the program to be debugged may be specified when DDT is first run. The format of this command is

```
DDT filename.extension
```

Note that the full file name and file extension must be specified. Only files of extension ".HEX" or ".COM" are meaningful.

7.12 Using DDT

Once the debugger is memory-resident, the program will display the revision number of the DDT program, which should be 2.Ø at first release of the HP-87. If a file name was specified when DDT was executed, two additional fields will be displayed. The first item, a column labeled "NEXT", indicates a four digit hex value. This represents the next free location in the TPA. The second item is a column labeled PC, and contains the four digit HEX value of the DDT program counter. This typically is initially ØlØØH.

Once DDT has been loaded and has reported the items described above, the DDT prompt "-" will appear in column 1. DDT is ready to accept input.

7.13 DDT Commands

The commands available are illustrated and briefly described in table 7-13. They are described in detail following that table.

Table 7-13. DDT Command Summary

A	Assemble a program statement at the given address.
D	Display contents of memory in hexadecimal and ASCII.
F	Fill memory with specified constant.
G	Begin execution ("GO") of program in memory with optional breakpoints.
I	Insert a file name into the FCB buffer.
L	Disassemble ("List") memory.
M	Move a block of memory from one location to another.
R	Read a file into memory for testing.
S	Display and edit ("Substitute") the contents of a memory location.
T	Trace the execution of a program.
U	Execute a specified number of program instructions (Untrace").
X	Examine the CPU registers.

A: Assemble

DDT allows immediate assembly of 8080 instructions by use of this command. The format is:

```
Axxxx
```

The user is prompted for a line of assembly language source code: line numbers and labels are ignored. The instruction mnemonic and the operand are assembled into memory starting at the specified address. The operand may contain any of the register names or hex constants. Labels may not, of course, be used as operands.

The next byte address for assembly will be displayed at the console. Once the DDT assembler has been invoked, it remains active until a blank line or period (".") is entered on the keyboard. Note that the assembler, like the disassembler to be discussed later, is a part of DDT which can be overlaid by user code. This procedure, described later in this section, causes a question mark to be printed in response to the "A" command input.

D: Display Memory

This command allows the contents of memory to be displayed in hex and ASCII. The first four bytes show the hexadecimal address of the first byte displayed on that line. The next sixteen bytes are the contents of memory locations starting at the indicated address. Finally, the ASCII representation of those sixteen bytes of memory. Non-displayable characters are represented by a decimal point ".".

There are three general forms of the command:

```
D
Dxxxx
Dxxxx,yyyy
```

In the first and most simple case, 192 bytes of memory are displayed starting at the current address contained in the DDT program counter.

The second case specifies that DDT should display the 192 bytes of memory starting at hexadecimal address "xxxx".

The final form causes memory locations between address "xxxx" and "yyyy" inclusive to be displayed on the CRT.

Displaying memory with this command increments the DDT program counter, so that on subsequent display commands no address need be specified to display contiguous segments of memory.

F: Fill Memory

This command allows the user to initialize a block of memory with a constant value. The format of the command is:

```
Fxxxx,yyyy,zz
```

DDT will fill all memory locations from "xxxx" through "yyyy" inclusive with the eight bit value specified by the hex value "zz".

G: Begin Execution

This command, the "Go" command, causes DDT to begin execution. There are four forms of this command:

```
G
Gxxxx
Gxxxx,yyyy
Gxxxx,yyyy,zzzz
```

Note that the first parameter "xxxx" may be omitted, so the form would be:

```
G,yyyy
G,yyyy,zzzz
```

This causes the current value of the DDT program counter to be used in place of the "xxxx" parameter.

The first form causes execution to begin at the current program counter address and to continue until a "RST 7", or reset-seven, is executed by the program under test. A jump or call to address ØØØØH (warm boot) will cause CP/M to restart, terminating DDT.

The second form is similar to the first, except that the DDT program counter is initially set to "xxxx".

The third form introduces a breakpoint at address "yyyy". Instructions beginning at address "xxxx" are executed, and will halt immediately prior to executing the instruction at "yyyy". Once the breakpoint is encountered, it is cleared, and must be reset if desired.

The final form allows a second breakpoint to be specified. This allows conditionally executed statements to break at either of two points. As with the previous form, encountering either breakpoint clears both.

By using the form in which no "xxxx" is specified, you can skip one or more statements. For example, while single stepping through a CP/M program, you will probably want to hop over all the statements executed in a system function call. By adding three to the current program counter, you can calculate the "yyyy" in the command

```
G,yyyy
```

The call to BDOS proceeds through its completion; then DDT resumes control. This makes disc functions much quicker, and makes console I/O much neater.

Once execution is started, it continues until: a breakpoint is encountered; a "RST 7" instruction is executed; or until a warm boot is performed. DDT cannot intervene otherwise.

When DDT does encounter a breakpoint, it displays an asterisk and a four-digit hex value. This represents the value of the DDT program counter, the address of the memory location about to be executed.

If either breakpoint is equivalent to the program counter, no instruction is executed and DDT intervenes immediately.

I: Insert FCB

This command permits the user to specify a file name to be inserted into the default FCB at address ØØ5CH. This permits the file to be Read using the "R" command which is described later. The format is:

```
Ifilename.COM
Imyfile.HEX
```

The file name must be specified although the drive code is assumed to be the current default drive. To specify another drive the drive code must be placed into address ØØ5CH. The "S" command allows this substitution.

If the file is to be loaded using the "R" command, the extension must be ".HEX" or ".COM". However, this command can also be used to initialize the FCB for file use by the program under test. In this case, the file extension is arbitrary and optional.

L: Disassemble Memory

This command causes the contents of memory to be disassembled back into 8080 mnemonic operation codes. This command is similar to the "D" command and includes the following three forms:

```
L
Lxxxx
Lxxxx,yyyy
```

The first form causes eleven lines of disassembled operation codes to be listed, starting at the current DDT program counter.

The second form is similar to the first, except that the DDT program counter is set to an initial value of "xxxx".

The final form starts the disassembly at "xxxx" and stops after disassembling the instruction at "yyyy".

If a location contains a value that cannot represent a valid 8080 operation code, DDT displays the address, question marks, and the contents of that address. This could mean that location is used for data storage by the program or that the value represents one of the valid Z80 opcodes unknown to the 8080 disassembler.

As with the "A" instruction, that portion of DDT that contains the disassembler may be overlaid so that larger programs may be tested.

M: Move Memory

This command permits a "block move" of memory. The format is:

```
Mxxxx,yyyy,zzzz
```

The contents of memory in the range from "xxxx" through "yyyy" inclusive are moved into the area of memory starting at "zzzz".

Note that, if a program is moved, it is not relocated. There is no effort to modify any addresses to correspond to the new starting address. This is a simple block move.

R: Read File

This command is used to load the contents of a file into memory. It is used in conjunction with the "I" command, since the name of the file to be loaded is taken from the default FCB buffer at address ØØ5CH. The "R" command can be specified in either of two forms:

```
R
Rxxxx
```

The first form selects a default local address which depends on the type of file being loaded.

If the file extension is".HEX", the load address is assumed to be Ø1ØØH and the file is loaded into the consecutive locations of memory.

When the second form of the command is used, the value "xxxx" is added to the default address as an offset.

When either form of this command is used, DDT displays either a question mark or a load message. The question mark indicates an error during the read operation: the file could not be opened, or a checksum error occurred in loading a ".HEX" file.

If the load message is displayed, it will be of the same format printed by DDT when it first executes. There will be a "NEXT" free address, and a current "PC" field containing the DDT program counter address.

S: Substitute Memory

This command is used to examine and optionally change one or more addresses in memory. The form is:

```
Sxxxx
```

The "xxxx" specifies the address which is to be examined. The address and the current hex contents are displayed. The user may press [END LINE] to leave that location unchanged and proceed to examine the next location.

If the user enters a valid two-digit hex value, that value is substituted for the initial value and the next address is examined. To end the substitute command, enter a decimal point ".". Entering an invalid value also terminates the "S" command.

T: Trace Execution

This command allows single step tracing of program execution for one to 65535 successive program steps. The format of the command is:

```
T
Txxxx
```

The first form is equivalent to the command "T1" and causes a single instruction to be executed. DDT displays the processor state as described for the "X" command described later.

The second form is similar, except that "xxxx" is assumed to be a hex value indicating the number of steps to be traced. A breakpoint can be forced during tracing by striking the [CONT] key.

DDT will intervene between each instruction in the user program, so program execution can be up to 5ØØ times slower during tracing.

After each trace, DDT sets the default program counter address to the contents of the register pair for the "D" command. It also sets the default value of the "L" command to the next byte of memory following the "D" display.

Since DDT receives control at each step by using the "RST 7" location, programs under test may not use this restart location.

U: Untrace

This command is similar to the "T" command, except that the processor status is not displayed before each instruction. The form is:

```
U
Uxxxx
```

The first form defaults to "U1". The number of steps specified by the hex value "xxxx" will be executed. Processor status is not automatically displayed.

All conditions relevant to the trace command apply to untrace as well.

X: Examine Program State

This command allows one or all registers to be examined and altered. The forms of this command are:

```
X
Xr
```

The "r" may be one of the Z80 CPU registers as follows:

```
C  Carry Flag
Z  Zero Flag
M  Memory Flag
E  Even Parity Flag
I  Intermediate Carry Flag
A  A Register
B  BC Register Pair
D  DE Register Pair
H  HL Register Pair
S  Stack Pointer
P  Program Counter
```

The first form displays all of the above fields as follows:

```
CxZxMxExIxA=yy B=zzzz D=zzzz H=zzzz S=zzzz P=zzzz instr
```

The "x", a zero or one, represents the value of the flag name which precedes it. The "yy" represents an eight-bit, two-digit hex value, the contents of the A register. The "zzzz", represents a sixteen-bit, four-digit hex value, the contents of the specified register pair or counter. The "instr" field contains the disassembled operation code of the instruction about to be executed at the current value of "P", the program counter. The second form allows the display and alteration of a flag, register, or register pair.

By typing one of the letters given above, the user selects which field is to be examined and edited. By pressing the [END LINE] key, nothing is changed. If an appropriate value is typed, that value is entered into the specified flag or register.

Note that, in entering any of the register pairs, both register values must be specified.

If the assembler/disassembler portion of DDT have been overlaid, the "instr" field contains the hex operation codes in place of the 8080 mnemonic.

7.14 Implementation Notes

When DDT first loads, it relocates itself into the portion of memory normally used by the CCP. Then, DDT changes the address stored at address 0006H so that all BDOS calls are directed through DDT. This allows programs which look at the address of the bottom of BDOS to see the base of DDT instead. DDT is now ready to accept input.

DDT is organized into two segments. The main nucleus, which can reside totally within the area used by the CCP, is always in memory and cannot be deleted. The second portion contains the DDT assembler and disassembler, and may be overlaid by user programs loaded into memory. Address 6 and 7 will contain the address of the base of the DDT program, which is the start of the DDT assembler/disassembler. That address will contain a jump to the start of the main nucleus.

As a user program grows toward the start of DDT, the assembler disassembler portion will be overwritten as it becomes necessary. Once DDT requires the memory between the start of this optional portion and the main nucleus, DDT will dynamically alter the address stored at address 6 and 7 and DDT will no longer be able to execute the "A" and "L" commands. Further, the "X" and "T" command will not display the 8080 mnemonics as it normally would. Instead, the machine operation code will be displayed.

If any portion of the assembler is overlaid, the "A", "L", "T" and "X" commands do not function as described above. Both "A" and "L", when entered at the console, cause a question mark "?" to be displayed indicating an improper DDT command. Where the "T" and "X" commands would normally display an assembly language mnemonic, the hex operation code is displayed instead.

7.15 The SAVE Command

When you have added or modified assembly language code with the DDT assembler, you will want to save the contents of memory on your disc as a ".COM" file. By using the SAVE command available as a standard part of CP/M, you can easily store a picture of memory into such a file for later execution.

The general form of the built-in SAVE command is:

```
SAVE n file
```

The "n" parameter is a decimal number referring to the number of pages of memory, starting at address 100H, to be saved on disc. One page of memory consists of 256 bytes of data. The "file" parameter must be a valid file reference consisting of a drive identifier (optional), file name, and extension.

Before you can save your program, however, you must calculate the size of the memory image. The size must be figured as the number of 256-byte (decimal) pages. Rather than work in decimal, which is clumsy here, let's see how to accomplish the task in hexadecimal.

First, determine the first free address above your program. You can discard the low two digits, and work with only the most significant two hex digits.

If the low two digits in the original address were "ØØ", subtract one from the number of pages you have just calculated. Otherwise, the number you have computed represents the exact number of 256-byte (or 1ØØH-byte) pages which must be saved.

Let's look at an example of this technique. Assume a program is in memory, loaded by DDT. The program uses memory from address ØlØØH through 2317H. The unused address above my program is 2318H: the most significant two digits are 23H.

The value 23H converts to 35 decimal. Since the low-order two digits are non-zero, the number of pages to save is 35. By exiting DDT with the command

```
GØ or ^C
```

CP/M performs a warm start, and control is returned to the CCP. However, my program is still in memory starting at ØlØØH! By typing the following command, my program will be stored on the disc ready to execute:

```
SAVE 35 MINE.COM
```

In fact, the last program executed can be re-entered by saving a program on disc using this command:

```
SAVE Ø !.COM
```

This causes a file named "!.COM" to be created with zero bytes.

When you type the file name as command input, CCP locates the file and loads it into memory. Of course, no data is loaded and CCP immediately calls address ØlØØH. The net effect is that the last program that executed is once again executed.

Of course, there are some qualifiers on these generalizations. First, the program must not use any memory above the beginning of the CCP. This can be determined using the HP sub-function number 12Ø: Return Region Bounds. If your program is larger than the start of CCP, each warm start will cause that part of memory to be overwritten by the CCP, destroying the program which was left there. However, if you are careful to avoid this limit condition, you can use this quirk of CP/M!

Disc File Organization

Standard File Extensions

Extension	File Type
ASM	ASM source code.
BAK	Back-up ED file.
BAS	BASIC source file.
COM	Transient command file.
DAT	ASCII data file.
DOC	Document file.
HEX	ASM object code.
LIB	Library file.
MAC	Relocatable assembler source code.
PRN	Printer list file.
REL	Relocatable object code.
SUB	SUBMIT command file.
SYM	Assembler symbol table.
XRF	Assembler absolute cross-reference table.
TXT	Text file.
$$$	Temporary file.

Disc Organization

Disc Format (Decimal values)	5-1/4-Inch Disc	8-Inch Disc
Bytes per physical sector	256	256
Physical sectors per track	16	30
Bytes per CP/M sector	128	128
CP/M sectors per track	32	60
Tracks per side	33	75
Tracks per disc	66	150

Each Winchester hard disc is equivalent to four 8-inch flexible discs.

Maximum File Size

5-1/4-inch disc:
16K bytes/extent x 32 extents/file = 512K bytes/file.

8-inch disc:
32K bytes/entry x 64 entries/file = 2048K bytes/file.

Sector Map

5-1/4" Disc Track	5-1/4" Disc Sector	8" Disc Track	8" Disc Sector	Description	CAT File Name and File Type
Ø	Ø-2	Ø	Ø-2	LIF volume information.	
Ø	3-4	Ø	3-4	Reserved.	
Ø	5-15	Ø	5-29	CP/M operating system.	
1-2	Ø-15	1	Ø-29	CP/M operating system.	
3	Ø-15	2	Ø-15	CP/M directory.	CP/MSYS DATA
4-63	Ø-15	2	16-29	CP/M user file space.	
---	---	3-147	Ø-29	CP/M user file space.	
---	---	148	Ø-27	CP/M user file space.	
64-65	Ø-15	148	28-29	BASIC and binary programs.	Autost Prog,
---	---	149	Ø-29	BASIC and binary programs.	CP/M BPGM

CP/M Keyboard Functions

Keystroke	Action
^[C]	Reloads and returns control to CP/M when at beginning of line.
^[E]	Performs carriage return/line feed without terminating input line.
^[H], [BACK SPACE]	Causes destructive back space.
^[I]	Writes tab character (HT) and tabs cursor.
^[J], ^[M], [END LINE]	Performs carriage return and executes input line.
^[P]	Routes console output to console and current list device. Pressed again, routes output to console only.
^[R]	Recalls current input line.
^[S]	Suspends console output. Console output resumes when any key is pressed.
^[U]	Deletes current input line and outputs carriage return/line feed.
^[X]	Causes destructive back spaces to beginning of input line.
[CONT]	Deletes and echoes preceding character.
^[[], [BACK SPACE]s, [TR/NORM]	Writes ESC character.
[PAUSE]	Pauses print buffer output. Pressed again, restarts print buffer output.
[STEP]	Erases print buffer.

^ [CTRL]

ASCII Character Codes

Character	Decimal Code	Hexadecimal Code	Keystroke	Character	Decimal Code	Hexadecimal Code
NULL	0	00	^[@]	space	32	20
SOH	1	01	^[A]	!	33	21
STX	2	02	^[B]	"	34	22
ETX	3	03	^[C]	#	35	23
ET	4	04	^[D]	$	36	24
ENQ	5	05	^[E]	%	37	25
ACK	6	06	^[F]	&	38	26
BEL	7	07	^[G]	'	39	27
BS	8	08	^[H]*	(	40	28
HT	9	09	^[I]	)	41	29
LF	10	0A	^[J]	*	42	2A
VT	11	0B	^[K]	+	43	2B
FF	12	0C	^[L]	,	44	2C
CR	13	0D	^[M]+	-	45	2D
SO	14	0E	^[N]	.	46	2E
SI	15	1F	^[O]	/	47	2F
DLE	16	10	^[P]	0	48	30
DC1	17	11	^[Q]	1	49	31
DC2	18	12	^[R]	2	50	32
DC3	19	13	^[S]	3	51	33
DC4	20	14	^[T]	4	52	34
NAK	21	15	^[U]	5	53	35
SYN	22	16	^[V]	6	54	36
ETB	23	17	^[W]	7	55	37
CAN	24	18	^[X]	8	56	38
EM	25	19	^[Y]	9	57	39
SUB	26	1A	^[Z]	:	58	3A
ESC	27	1B	^[[]++	;	59	3B
FS	28	1C	^[\]	<	60	3C
GS	29	1D	^[]]	=	61	3D
RS	30	1E	^[^]	>	62	3E
US	31	1F	^[_]	?	63	3F

^ [CTRL].
s [SHIFT].
* Also, [BACK SPACE].
+ Also, [END LINE].
++ Also, [BACK SPACE]s and [TR/NORM].

Character	Decimal Code	Hexadecimal Code	Character	Decimal Code	Hexadecimal Code
@	64	40	`	96	60
A	65	41	a	97	61
B	66	42	b	98	62
C	67	43	c	99	63
D	68	44	d	100	64
E	69	45	e	101	65
F	70	46	f	102	66
G	71	47	g	103	67
H	72	48	h	104	68
I	73	49	i	105	69
J	74	4A	j	106	6A
K	75	4B	k	107	6B
L	76	4C	l	108	6C
M	77	4D	m	109	6D
N	78	4E	n	110	6E
O	79	4F	o	111	6F
P	80	50	p	112	70
Q	81	51	q	113	71
R	82	52	r	114	72
S	83	53	s	115	73
T	84	54	t	116	74
U	85	55	u	117	75
V	86	56	v	118	76
W	87	57	w	119	77
X	88	58	x	120	78
Y	89	59	y	121	79
Z	90	5A	z	122	7A
[	91	5B	{	123	7B
\	92	5C	|	124	7C
]	93	5D	}	125	7D
^	94	5E	~	126	7E
-	95	5F	DEL*	127	7F

* [CONT].

CP/M Keycodes

When one of the following keys is pressed during a CONIN routine, the corresponding value will be returned in the B register.

Key	Code	
	Decimal	Hexa-decimal
[BACK SPACE]	8	Ø8
[END LINE]	13	ØD
[TR/NORM]	27	1B
[CONT]	127	7F
[k1]	128	8Ø
[k2]	129	81
[k3]	13Ø	82
[k4]	131	83
[k8]	132	84
[k9]	133	85
[k1Ø]	134	86
[k11]	135	87
[-CHAR]	136	88
[CLEAR]	137	89
[INIT]	14Ø	8C
[RUN]	141	8D
[CONT]	143	8F
[ROLL^]	145	91
[TEST]	146	92
[k14]	147	93
[LIST]	148	94
[P LST]	149	95
[KEY LABEL]	15Ø	96
[BACK SPACE]s	155	9B
[k7]	156	9C
[-LINE]	157	9D
[I/R]	158	9E
[<--]	159	9F
[E]	16Ø	AØ
[k5]	161	A1
[k6]	162	A2
[^]	163	A3
[v]	164	A4
[k12]	165	A5
[RESLT]	166	A6
[A/G]	168	A8
[ROLLv]	169	A9
[-->]	17Ø	AA
[k13]	172	AC

Cursor Control Codes

Control Sequence	Instruction	Description
Form Feed and ESC *	Clear All	Homes cursor and clears screen.
ESC &a col c row Y	Cursor to Address	Moves cursor to the specified column and row relative to the top of the screen, where col and row are 2-digit decimal numbers.
ESC = row col	Cursor to Address	Moves cursor to the specified row and column relative to the top of the screen, where row and col are characters whose codes are equal to 20H plus row or column number.
ESC &dv and ESC)	Reverse Video	Displays black characters against white background from cursor location, where v may be uppercase A through O.
ESC &d@ and ESC (	Normal Video	Displays white characters against black background from cursor location.
ESC T	Clear to End of Line	Clears the current display line from cursor location.
ESC Y	Clear to End of Screen	Clears the screen from cursor location.

CPU Registers

Reserved Operand	Reference
A	A register (accumlator).
B	B register or BC pair.
C	C register.
D	D register or DE pair.
E	E register.
H	H register or HL pair.
L	L register.
M	Data in memory addressed by HL pair.
$	Current address.
SP	Stack pointer.
PSW	A register and flags C (carry), Z (zero), M (minus), E (even parity), and I (intermediate carry).

CPU Instruction Set

Operand Abbreviations:

data — A constant or expression that evaluates to an 8-bit value.

addr — A constant, expression, or a label that evaluates to a 16-bit value.

r — Contents of an 8-bit register or M.

rp — Contents of a 16-bit register pair or stack pointer SP.

Opcode mnemonic and Operand(s)	Description
ACI data	Add operand to A with carry.
ADC r	Add register to A with carry.
ADD r	Add register to A without carry.
ADI data	Add operand to A without carry.
ANA r	Logical AND register with A.
ANI data	Logical AND operand with A.
CALL addr	Unconditional call subroutine.
CC addr	Call on carry condition.
CM addr	Call on negative condition.
CMA	Complement A register.
CMC	Complement carry flag.

Opcode mnemonic and Operand(s)	
CMP r	Compare register with A.
CNC addr	Call on no carry condition.
CNZ addr	Call on nonzero condition.
CP addr	Call on positive condition.
CPE addr	Call on even parity condition.
CPI data	Compare operand with A and set flags as if an SUI had been done.
CPO addr	Call on odd parity condition.
CZ addr	Call on zero condition.
DAA	Decimal adjust A register.
DAD rp	Double precision add rp to HL and store in HL.
DCR r	Decrement specified register.
DCX rp	Decrement specified register pair.
DI	Disable interrupts of the CPU.
EI	Enable interrupts of the CPU.
HLT	Halt operation of the CPU.
IN data	Load A register from CPU data port.
INR r	Increment specified register.
INX rp	Increment specified register pair.

Opcode Mnemonic and Operand(s)	Description
JC addr	Jump on carry condition.
JM addr	Jump on negative condition.
JMP addr	Unconditional jump.
JNC addr	Jump on no carry condition.
JNZ addr	Jump on nonzero condition.
JP addr	Jump on positive condition.
JPE addr	Jump on even parity condition.
JPO addr	Jump on odd parity condition.
JZ addr	Jump on zero condition.
LDA addr	Load A register from address.
LDAX rp	Load A from address in BC or DE register pair.
LHLD addr	Load HL register pair from address.
LXI rp, data	Move data bytes into register pair.
MVI r, data	Move data byte into specified register.
MOV r1,r2	Move contents of r2 into r1.
NOP	No operation; a place holder for memory locations.

Opcode mnemonic and Operand(s)	Description
ORA r	Logical OR register with A.
ORI data	Logical OR operand with A.
OUT data	Copy a Register to CPU data port.
PCHL	Load program counter from HL.
POP rp	Load register pair from stack.
PUSH rp	Store register pair onto stack.
RAL	Rotate carry flag and A register left.
RAR	Rotate carry flag and A register right.
RC	Return on carry condition.
RET	Unconditional return from subroutine.
RLC	Rotate register A left. Carry is copy of most significant bit.
RM	Return on negative condition.
RNC	Return on no carry condition.
RNZ	Return on nonzero condition.
RP	Return on positive condition.
RPE	Return on even parity condition.
RPO	Return on odd parity condition.

Opcode mnemonic and Operand(s)	Description
RRC	Rotate bits right. Carry is copy of least significant bit.
RST n	Programmed restart, where Ø < n < 7.
RZ	Return on zero condition.
SBB r	Subtract register with borrow.
SBI data	Subtract operand from (A) with borrow.
SHLD addr	Copy HL register pair to address.
SPHL	Load stack pointer from HL.
STA addr	Copy A register to memory at address.
STAX rp	Copy A to memory adressed by register pair.
STC	Set carry flag.
SUB r	Subtract register without borrow.
SUI data	Subtract operand from A without carry.
XCHG	Exchange DE and HL register pairs.
XRA r	Exclusive OR register with A.
XRI data	Exclusive OR operand with A.
XTHL data	Exchange data from top of stack with HL.

CP/M Error Messages

Message	Originator	Probable Cause
CP/M BOOT ERROR	CP/M module.	Bad disc, drive door open.
CP/M SYSTEM CARD SELF-TEST ERROR	CP/M BPGM.	Module not installed, module requires service.
Disk select error on drive x.*	BDOS.	Bad disc, drive door open.
No operating system in drive A:	CP/M module, BIOS warm boot routine.	Bad disc, drive door open, wrong disc in drive A.
Sector read error on drive x.*	BDOS.	Bad disc, drive door open, wrong drive identifier.
WARM BOOT FAILED	BIOS warm boot routine.	Bad disc, drive door open, wrong drive identifier.
Write to R/O disk error on drive x.*	BDOS.	Disc set to R/O; disc changed without warm boot.
Write to R/O file error on drive x.*	BDOS.	File set to R/O with STAT command.

* x is the appropriate drive identifier.

ASM Error Messages

Source Code Errors:

D	Data error.
E	Expression error.
L	Label error.
N	Not implemented in this version.
O	Overflow.
P	Phase error.
R	Register error.
V	Value error.

Assembly Errors:

CANNOT CLOSE FILE

NO SOURCE FILE PRESENT

NO DIRECTORY SPACE

SOURCE FILE NAME ERROR

SOURCE FILE READ ERROR

OUTPUT FILE WRITE ERROR

Annotated Bibliography

CP/M General:

Fernandez, Judi, and Ashley, Ruth, Using CP/M. New York: John Wiley Sons, 1980.

A self-teaching book which uses a question/answer format to convey user information. A primer of CP/M commands and syntax.

Hogan, Thom, Osborne CP/M User Guide. Berkeley: Osborne/McGraw-Hill, 1981.

An overview of CP/M, with both the user and progammer in mind. Includes a review of many software programs which can be used with CP/M.

Murtha, Stephen, and Waite, Mitchell, CP/M Primer. Indianapolis: Howard W. Sams and Company, Inc., 1980.

This book presents the beginner with an introduction to CP/M, written for the first-time CP/M user.

Zaks, Rodnay, The CP/M Handbook with MP/M. Berkeley: Sybex, 1980.

Discusses CP/M and MP/M commands, programs, and facilities.

Assembly Language Programming:

8080/8085 Assembly Language Programming Manual. Santa Clara, Calif.: Intel Corp., 1980.

The "source" for 8080 Assembly Language codes.

Leventhal, Lance, 8080A/8085 Assembly Language Programming. Berkeley: Osborne/McGraw-Hill, 1978.

Leventhal, Lance A., Z80 Assembly Language Programmng. Berkeley: Osborne/McGraw-Hill, 1979.

Santore, Ron, 8080 Machine Language Programming for Beginners. Portland: Dilithium Press, 1980.

Zaks, Rodnay, Programming the Z80. Berkeley: Sybex, 1979.

INDEX

A

A (DDT COMMAND), 7-26
A (ED COMMAND), 6-4
ALTER JUMP VECTOR, 4-59,4-60,4-61
ALV, 5-5,5-6
AL0, 5-7
AL1, 5-7
Append, 6-4
Arithmetic Instructions, 7-18
Arithmetic Operators, 7-8,7-9
ASCII Character Codes, C-1,C-2
ASM Command, 1-6,7-1
ASM File Extension, 3-2,7-2
Assemble, 7-26
Assembler, 7-1
Assembler Directives, 7-10

B

B (ED Command), 6-5
BAK File Extension, 3-2,6-1
BAS File Extension, 3-2
Base Page, 2-2,2-4,2-5
Basic Disc Operating System, 2-3
Basic I/O System, 2-3
Batch Job, 1-22
BAT:, 1-10
BDOS, 2-2,2-3,4-2
Begin Execution, 7-27
Bibliography, J-1
BIOS, 2-2,2-3,5-1
BIOS Entry Points, 5-1
BLM, 5-6,5-7
Block Mask, 5-6,5-7
Block Move, 6-15
Block Shift, 5-6,5-7
BLS, 5-7,5-8
BOOT, 5-2
Bootstrap Loading, 1-3
BSH, 5-6,5-7
Buffer Position, 6-5
Built-in Commands, 1-4,1-5

C

C (ED COMMAND), 6-5
CCP, 2-2,2-3,2-5,2-6
Chain, 4-57,4-58
CHAIN TO SPECIFIED PROGRAM, 4-57,4-58
Character Codes, C-1,C-2
Character Pointer, 6-2
Character Position, 6-5
Check Sum Vector, 5-5,5-7,5-8
CKS, 5-7,5-8
CLOSE FILE, 4-23
Cold Start, 1-3
COM File Extension, 3-2
Command Buffer, 4-57
Comment Field, 7-4
COMPUTE FILE SIZE, 4-45
CONIN, 5-2,D-1
CONOUT, 5-3
Console Command Program, 2-3
CONSOLE INPUT, 4-5
CONSOLE OUTPUT, 4-6
CONSOLE STATUS, 4-17
CONST, 5-2
Control Instructions, 7-18
CON:, 1-10
CP, 6-2
CP/M VERSION NUMBER, 4-18
CPU Instruction Set, G-1,7-13
CPU Registers, F-1
CREATE FILE, 4-30,4-31
CRT:, 1-10
CSV, 5-5,5-6
CURRENT DISC ID, 4-34
Current Record Number, 3-5,3-7
Cursor Control Codes, E-1

D

D (DDT Command), 7-26
D (ED Command), 6-5
DAT File Extension, 3-2

Data Movement Instruction, 7-17
DDT Command, 1-6,7-23
Default DMA Buffer, 2-5
Default FCB, 2-5,3-8
Default File Buffer, 3-4,3-8
Delete Characters, 6-5
DELETE FILE, 4-27
Destination Devices, 1-10
Device Assignments, 1-10,1-19
Device Mapping, 1-10,1-21
Device Names, 1-10,1-11
Device Status, 1-18
DEV:, 1-19
DIR Attribute, 1-18
DIR Command, 1-5
DIR BUF, 5-5
DIRECT CONSOLE I/O, 4-11
Direct Memory Access Buffer, 4-24
Disassemble Memory, 7-29
Disc Capacity, 3-3,3-4
Disc Directory, 3-7,3-8
Disc Drive Identifier, 1-8,3-1
Disc File Entry, 3-3,3-4
Disc Format, A-1,A-2
Disc Group Allocation Blocks, 3-5, 3-7
DISC LOGIN VECTOR, 4-33
Disc Organization, A-1,A-2
Disc Parameter Block, 4-40,5-5
Disc Parameter Header, 5-4,5-5
Disc Tables, 5-4
Display Memory, 7-26
DMA Buffer, 3-8,3-9,4-24,4-28, 4-29,4-35,5-3,5-4
DPB, 4-40,5-5,5-6,5-7
DPH, 5-4,5-5,5-6
Drive Code, 3-5,3-6
DRIVE ID, 2-4,2-5
Drive Identifier, 3-1
DRM, 5-7
DSM, 5-7
DSK:, 1-20,1-21
DUMP Command, 1-6,7-23
Dynamic Debugging Tool, 1-6,7-23

E

E (ED Command), 6-6
ED Command, 1-6,6-1
Editor, 6-1
EOF:, 1-11
ERA Command, 1-5
Error Messages, 7-20,7-21, 7-22,H-1,I-1
Examine Program State, 7-32
Exit, 6-6
EXM, 5-7
Extended System Function Calls, 4-48
Extent, 3-3,3-4,3-5
Extent Mask, 5-7
Extent Number, 3-5,3-6
Extent Record Count, 3-5,3-6

F

F (DDT Command), 7-27
F (ED Command), 6-6
FCB, 3-4,7-28
FDOS, 2-2,2-3
File Access, 3-2
FILE ATTRIBUTES, 4-39
File Control Block, 3-4
File Extension, 3-1,3-2,3-5,3-6, A-1
File Name, 3-1,3-5,3-6
File Reference, 3-1
File Size, 3-3,3-4,4-45,A-2
File Structure, 3-2
File Type, 3-1,A-1
Fill Memory, 7-27
Find, 6-6
Find With Autoscan, 6-10
Flag Byte, 4-53,4-54
FORMAT Command, 1-6
Fundamental Disc Operating System, 2-3

G

G (DDT Command), 7-27
GET ADDR (ALLOC), 4-36
GET IOBYTE, 4-12
GET R/O VECTOR, 4-38
GET/SET USER NUM, 4-41
GO, 7-34

H

H (ED Command), 6-7
HEX File Extension, 3-2,7-2
HOME, 5-3
HP Extended Function Calls, 4-48
HP Machine Number, 4-62
HP Version Number, 4-51

I

I (DDT Command), 7-28
I (ED Command), 6-7
Immediate Operand Instructions, 7-16
Increment and Decrement Insructions, 7-16
Insert FCB, 7-28
Insert Text, 6-7
Instruction Set, 7-13,G-1
IOBYTE, 2-4,2-5,4-12,4-13

J

J (ED Command), 6-8
Jump Vector Buffer, 4-59,4-60,4-61
Jump Vector Table, 4-60,5-1,5-2
Juxtapose String, 6-8

K

K (ED Command), 6-9
Keyboard Functions, B-1
Keycodes, D-1
Kill Lines, 6-9

L

L (DDT Command), 7-29
L (ED Command), 6-9
Label Field, 7-4
LIF Directory, 1-2
Line Move, 6-9
Line Number Field, 7-4
LIST, 5-3
List Device Names, 1-10,1-21
LIST OUTPUT, 4-10
LISTST, 5-4
LOAD Command, 1-6,7-23
Logical Device Names, 1-10
Logical Operation Instructions, 7-18
Logical Record, 3-2,3-3
LPT:, 1-10
LST:, 1-10

M

M (DDT Command), 7-29
M (ED Command), 6-10
Macro Definition, 6-10
Memory Addresses, 2-2
Memory Allocation, 2-2
Memory Buffer, 4-52,4-53,4-54
Memory Buffer Image, 6-1,6-2
Memory Organization, 2-1
Move Memory, 7-29

N

N (ED Command), 6-10
NUL:, 1-11
Numeric Constants, 7-6

O

O (ED Command), 6-10
Object Code, 7-2
OEM Number, 4-50
OFF, 5-7,5-8
Opcodes, 7-13,G-1
OPEN FILE, 4-21,4-22
Operand Field, 7-4
Operation Field, 7-4

P

P (ED Command), 6-11
PARAMETER ADDRESS, 4-40
Physical Device Names, 1-10
Physical Record, 3-2,3-3
PIP Command, 1-6,1-8
PIP Options, 1-11
Pound Sign (ED Command), 6-4
Print Pages, 6-11
PRINT STRING, 4-14
Program Control Instructions, 7-14,7-15
PRN File Extension, 3-2,7-2
PRN:, 1-11
PTP:, 1-10
PTR:, 1-10
PUNCH, 5-3
PUNCH OUTPUT, 4-9
PUN:, 1-10

Q

Q (ED Command), 6-11
Quit, 6-11

R

R (DDT Command), 7-30
R (ED Command), 6-12
Radix Indicators, 7-6
Random Record, 4-46
RDR:, 1-10
READ, 5-3
READ CONSOLE BUFFER, 4-15, 4-16
Read File, 7-30
Read Library File, 6-12
READ MEMORY BUFFER, 4-52
READ RANDOM, 4-42,4-43
READ SEQUENTIAL, 4-28
READER, 5-3
READER INPUT, 4-8
Read-only Status, 1-18,1-21, 4-38
Read-write Status, 1-18
Region Bounds, 4-55,4-56
REN Command, 1-5
RENAME FILE, 4-32
Reserved Operand Words, 7-9
RESET DISC SYSTEM, 4-19
RESET DRIVE, 4-47
Restart ED, 6-7
RETURN HP MACHINE NUMBER, 4-62
RETURN HP VERSION NUMBER, 4-51
RETURN OEM NUMBER, 4-50
Return Original File, 6-10

RETURN REGION BOUNDS, 4-55,4-56
R/O Attribute, 1-18
R/W Attribute,1-18

S

S (DDT Command), 7-30
S (ED Command), 6-12
SAVE Command, 1-5,7-33,7-34
SCR, 5-5
SEARCH FOR FIRST, 4-24,4-25
SEARCH FOR NEXT, 4-26
Sector Map, A-2
SELDSK, 5-3,5-6
SELECT DISC, 4-20
SET DMA ADDRESS, 4-35
SET IOBYTE, 4-13
SET RANDOM RECORD, 4-46
SETDMA, 5-3
SETSEC, 5-3
SECTRAN, 5-4
SETTRK, 5-3
SFC, 4-1
Sleep, 6-15
Source Code, 7-1,7-3
Source Devices, 1-10
SPT, 5-6,5-7
STAT Command, 1-6,1-17
STAT File Attributes, 1-18
String Constants, 7-7
SUBMIT Command, 1-22,1-23,1-24
Substitute Memory, 7-30
Substitute String, 6-12
SYS Attribute, 1-18
System Function Calls, 4-1
SYSTEM RESET, 4-4

T

T (DDT Command), 7-31
T (ED Command), 6-13
Text Editor, 6-1
TPA, 2-2,2-4
Trace Execution, 7-31
Transient Commands, 1-5,1-6
Transient Program Area, 2-2, 2-3,2-4
Transient Programs, 2-6
TTY:, 1-10
TXT File Extension, 3-2
TYPE Command, 1-5
Type Lines, 6-13

U

U (DDT Command), 7-31
U (ED Command), 6-13
UC1:, 1-10
UL1:, 1-10
Untrace, 7-31
Upper/Lower Case Translate, 6-13
UP1:, 1-10
UP2:, 1-10
UR1:, 1-10
UR2:, 1-10
USER Command, 1-5
User Memory Space, 2-2
User Number, 3-5,3-7,4-41
USR:, 1-20

V

V (ED Command), 6-14
VAL:, 1-2Ø
Verify Line Numbers, 6-14

W

W (ED Command), 6-14
Warm Start, 1-7,2-4,2-5
WBOOT, 5-2
WRITE, 5-4
Write Lines, 6-14
WRITE MEMORY BUFFER, 4-53,4-54
WRITE PROTECT DISC, 4-37
WRITE RANDOM, 4-44
WRITE SEQUENTIAL, 4-29
WRITE W/ZERO FILL, 4-48

X

X (DDT Command), 7-32
X (ED Command), 6-15
X:, 1-5
XLT, 5-5
XSUB Command, 1-6,1-25

Z

Z (ED Command), 6-15