

HP 125 Business Assistant



SYSTEM REFERENCE MANUAL



19410 Homestead Road, Cupertino, California 95014

Printing History

New editions of this manual will incorporate all material updated since the previous edition. Update packages may be issued between editions and contain replacement and additional pages to be merged into the manual by the user. Each updated page will be indicated by a revised date at the bottom of the page. Note that pages which are rearranged due to changes on a previous page are not considered revised.

The manual printing date and part number indicate its current edition. The printing date changes when a new edition is printed. (Minor corrections and updates which are incorporated at reprint do not cause the date to change.)

First Edition. July 1981

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Manual Plan

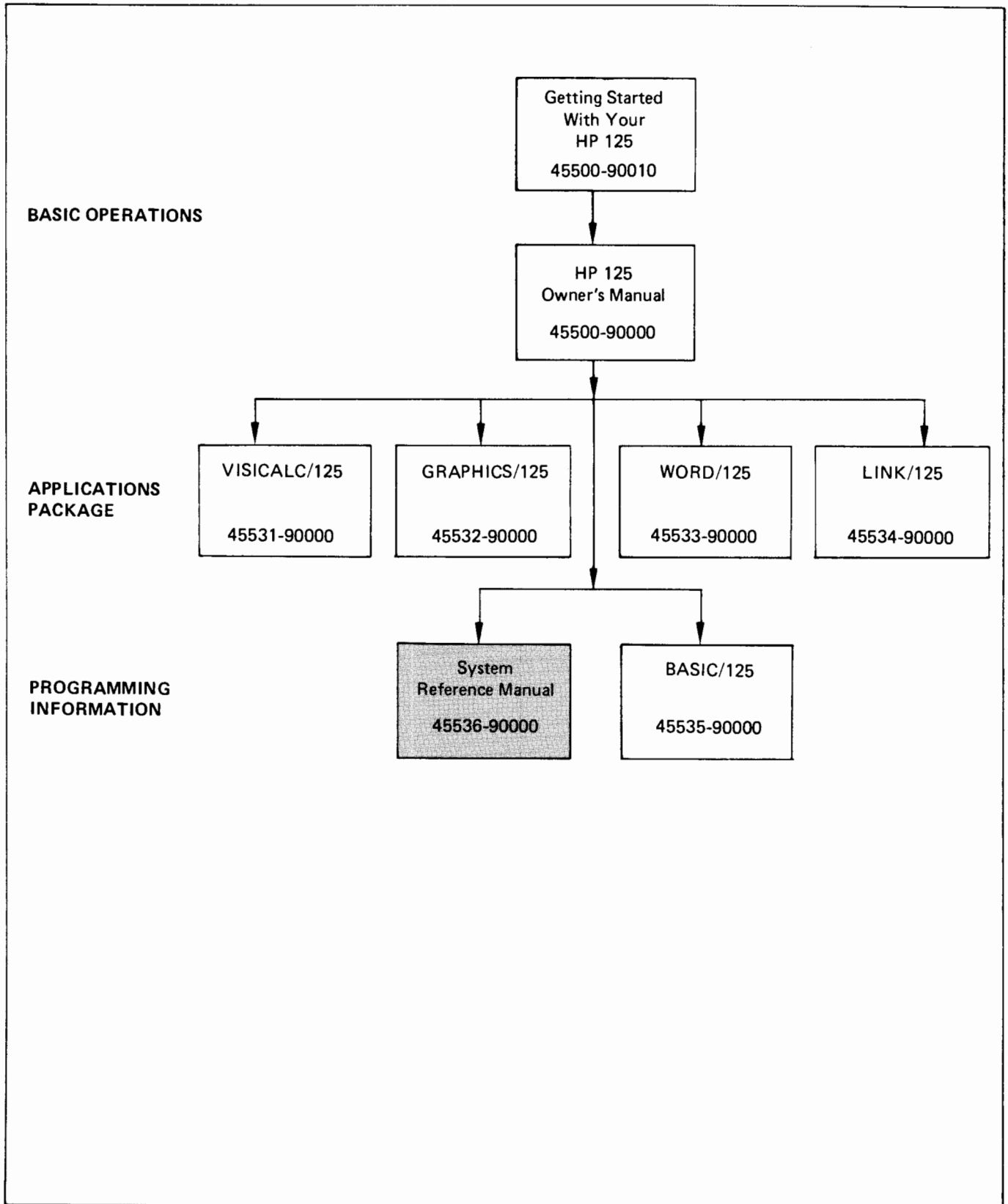


Table of Contents

Chapter 1 Overview of the HP 125

Introduction.....	1-1
The HP 125 as Terminal.....	1-2
Terminal Operation.....	1-2
The HP 125 as Computer.....	1-3
System Functioning.....	1-4
System Operation.....	1-4
Summary.....	1-5

Chapter 2 The HP 125 Terminal

Introduction.....	2-1
Programming the TPU.....	2-1
Two Character Escape Sequences.....	2-2
Multiple Character Escape Sequences.....	2-3
Escape Sequence Compatibility.....	2-4
Manual Organization.....	2-4
Summary.....	2-5

Chapter 3 Programmatic Configuration

Introduction.....	3-1
Keyboard Characteristic Control.....	3-2
Strap Configurations.....	3-3
Configuration Control.....	3-4
Programming the SPOW Latch.....	3-4
Summary.....	3-5

Chapter 4 Programming the Function Keys

Introduction.....	4-1
Function Control Sequences.....	4-1
Defining the User Keys.....	4-3
Defining Keys Under Program Control.....	4-3
Executing a User Key.....	4-6
Selecting the [USER KEYS] Menu.....	4-6
Summary.....	4-7

Table of Contents (Cont.)

Chapter 5 Display, Keyboard and Printer Control

Introduction.....	5-1
Cursor Positioning.....	5-1
Absolute Addressing.....	5-2
Screen Relative Addressing.....	5-2
Cursor Relative Addressing.....	5-3
Additional Cursor Features.....	5-3
Cursor Positioning Sequences.....	5-4
Display Enhancements.....	5-5
Margins/Tabs/Columns.....	5-6
Edit Control Sequences.....	5-8
Special Terminal Functions.....	5-9
Printer Control Functions.....	5-12
Internal Printer Page Format.....	5-12
Printer Logging.....	5-12
Data Transfers.....	5-12
Device Control Completion Codes.....	5-13
Summary.....	5-13

Chapter 6 Terminal and Device Status

Introduction.....	6-1
Interpreting Status.....	6-2
Terminal Status.....	6-3
Primary Terminal Status.....	6-3
Printer Status.....	6-11
HP-IB Printer Status.....	6-13
Internal Thermal Printer.....	6-13
Serial Printer Status.....	6-14
Terminal Identifier.....	6-14
Cursor Sensing.....	6-15
Summary.....	6-16

Chapter 7 Introduction to CP/M

The HP 125 as Computer.....	7-1
Introduction to CP/M.....	7-1
CP/M Commands.....	7-3
Built-In Commands.....	7-3
Transient Commands.....	7-4
Memory Organization.....	7-6
FDOS.....	7-6
CCP.....	7-7
TPA.....	7-7
Base Page.....	7-8
The File System.....	7-8
File Names.....	7-8
File Structure.....	7-9
File Access.....	7-10

Table of Contents (Cont.)

How the CCP Functions.....	7-10
Transient Programs.....	7-11
Summary.....	7-11

Chapter 8 CP/M Internal Organization

Introduction.....	8-1
File System Overview.....	8-1
File Control Block.....	8-3
Disc Directory.....	8-7
Default FCB's and the CCP.....	8-7
FCB Example.....	8-8
Base Page.....	8-10
Summary.....	8-12

Chapter 9 System Function Calls

Introduction.....	9-1
Accessing Function Calls.....	9-2
Summary.....	9-48

Chapter 10 Extended System Function Calls

Introduction.....	10-1
Summary.....	10-26

Chapter 11 Device Assignments

Introduction.....	11-1
Logical and Physical Devices.....	11-1
Device Mapping.....	11-4
Selecting Source and Destination Devices.....	11-6
Device Mapping Modes.....	11-7
Examples.....	11-10
Summary.....	11-11

Chapter 12 Advanced CP/M Features

Introduction.....	12-1
BIOS Entry Points.....	12-1
Disc Tables.....	12-5
Disc Parameter Header.....	12-5
Disc Parameter Block.....	12-6
Keycode Mode.....	12-8
Summary.....	12-12

Table of Contents (Cont.)

Chapter 13 CP/M Transient Utilities

Introduction.....	13-1
PIP.....	13-2
Special Devices.....	13-3
PIP Options.....	13-4
Examples.....	13-9
STAT.....	13-9
File Status.....	13-10
Setting Device Status.....	13-11
Device Status.....	13-11
Examples.....	13-13
Additional HP 125 Utilities.....	13-13
COPY.....	13-13
FORMAT.....	13-14
Batch Job Processing.....	13-15
Submit.....	13-15
XSUB.....	13-17
LOAD.....	13-18
DUMP.....	13-18
Other CP/M Utilities.....	13-18
Summary.....	13-18

Chapter 14 Using the CP/M Text Editor

Introduction.....	14-1
Memory Image.....	14-2
ED Commands.....	14-3
Sample ED Session.....	14-15
Summary.....	14-17

Chapter 15 The CP/M Assembler

Introduction.....	15-1
Using the Assembler.....	15-2
Assembly Language Programming.....	15-4
Program Format.....	15-4
The Value of a Label.....	15-5
Forming the Operand.....	15-6
Assembler Directives.....	15-10
Sample 8080 Source Program.....	15-15
Operation Codes.....	15-15
Program Control.....	15-16
Immediate Operand Instructions.....	15-17
Increment and Decrement Instructions.....	15-17
Data Movement Instructions.....	15-18
Arithmetic and Logical Operations.....	15-19
Control Instructions.....	15-19
Assembler Execution.....	15-20

Table of Contents (Cont.)



Error Messages.....	15-21
Fatal Errors.....	15-21
Source Code Error Messages.....	15-22
Summary.....	15-23

Chapter 16 Using the Dynamic Debugging Tool

Introduction.....	16-1
Starting DDT.....	16-1
Using DDT.....	16-2
DDT Commands.....	16-2
Implementation Notes.....	16-11
Notes on the DDT Assembler.....	16-12
Summary.....	16-13

Appendices

APPENDIX A SYSTEM REFERENCE TABLES

APPENDIX B ANNOTATED BIBLIOGRAPHY

CP/M General.....	B-1
Assembly Language Programming.....	B-2

APPENDIX C APPLICATION PROGRAM INSTALLATION

Introduction.....	C-1
Application Installation.....	C-3
The WEL File.....	C-3
Installation Header.....	C-3
File Count.....	C-3
Softkey Label.....	C-4
Application Size.....	C-4
Command String Length.....	C-4
Command String.....	C-4
Creating the '.WEL' File.....	C-4
Example.....	C-5
Master Files.....	C-7
Summary.....	C-8

APPENDIX D DISC ORGANIZATION

Introduction.....	D-1
Disc Data Organization.....	D-2
Reading Non-CP/M Discs.....	D-2
Hewlett-Packard Format Discs.....	D-2
Non-HP Discs.....	D-3
Summary.....	D-5

Table of Contents (Cont.)

APPENDIX E ERROR MESSAGES

Power-on Test.....	E-1
TPU Error Messages.....	E-4
CPU Error Messages.....	E-6
Summary.....	E-8

Illustrations

<u>Table</u>	<u>Page</u>
4-1	Function Group Control Sequences.....4-2
5-1	Display Enhancement Codes.....5-6
6-1	TPU Status Byte Values.....6-2
6-2	Primary Terminal Status Request.....6-4
6-3a	Primary Status Decoding.....6-5
6-3b	Primary Status Decoding.....6-6
6-3c	Primary Status Decoding.....6-7
6-4	Secondary TPU Status.....6-8
6-5a	Secondary Status Decoding.....6-9
6-5b	Secondary Status Decoding.....6-10
6-6	Printer Device Status.....6-11
6-7	Printer Status Decoding.....6-12
6-8	Cursor Sending Request.....6-16
7-1	CP/M Integral Command.....7-3
7-2	CP/M Transient Commands.....7-5
7-3	CP/M Memory Organization.....7-6
7-4	File Types.....7-9
8-1	File Control Block.....8-4
8-2	Example FCB Set-Up: DREM.....8-9
8-3	Command Line Buffer Format.....8-10
8-4	Base Page Organization.....8-11
9-1	System Function Call Summary.....9-2
9-2	Sample System Function Call.....9-3
10-1	HP Extended System Function Calls.....10-2
10-2	Sample HP Extended System Function Call.....10-2
10-3	Jump Vector Buffer.....10-5
10-4	Jump Vector Table Entries.....10-6
10-5	Terminal Message Buffer.....10-10
10-6	Chain Program Buffer.....10-12
10-7	Region Request Parameters.....10-15
10-8	Write Memory File Buffer.....10-17

Table of Contents (Cont.)

<u>Table</u>		<u>Page</u>
11-1	IOBYTE Format.....	11-2
11-2	IOBYTE LST: Device Value.....	11-3
11-3	Source and Destination Device Assignments.....	11-5
11-4	TPU View of Logical and Physical Devices.....	11-7
11-5	Mode Parameter Combinations.....	11-9
11-6	Pass Thru Devices.....	11-9
12-1	Jump Vector Table.....	12-2
12-2	Disc Parameter Header.....	12-5
12-3	Disc Parameter Block.....	12-7
13-1	PIP Command Examples.....	13-2
13-2	PIP Device Names.....	13-3
13-3	PIP Options.....	13-4
13-4	Sample PIP File Transfers.....	13-9
13-5	Valid STAT File Parameters.....	13-10
13-6	Sample STAT Device Status 'parm'.....	13-11
13-7	Physical List Device Names.....	13-12
13-8	Sample STAT Commands.....	13-13
14-1	ED Data Flow.....	14-2
14-2	Memory Image Buffer.....	14-3
14-3	ED Command Summary.....	14-3
15-1	ASM Execution Commands.....	15-3
15-2	Radix Indicators.....	15-6
15-3	Sample Valid Numeric Constants.....	15-7
15-4	Sample String Constants.....	15-8
15-5	Valid Arithmetic Operators.....	15-8
15-6	Sample Operator Precedence.....	15-9
15-7	Reserved Operand Words.....	15-10
15-8	Assembler Directives.....	15-11
15-9	Sample Use of the DB Directives.....	15-12
15-10	Sample DW Directives.....	15-13
15-11	Conventions.....	15-15
15-12	Sample ASM Messages.....	15-20
16-1	DDT Command Summary.....	16-3
16-2	Sample Display Command Listing.....	16-4
C-1	Sample Installation Menu.....	C-2
C-2	Installation File Field Locations.....	C-3
D-1	Disc Format and Organization.....	D-4
D-2	Disc Track Allocation Map.....	D-5

OVERVIEW OF THE HP 125

Introduction

The HP 125 is a terminal-sized device which combines the features of a traditional intelligent terminal with the functions of a stand-alone computer system. While there are terminals available which can serve as micro-computers in specific applications, and some small computer systems offer terminal emulation software, the HP 125 is one of the few products which contain all of the capabilities of both.

In fact, within the HP 125 package there are two totally separate 'computer systems'. The first, called the Terminal Processor Unit or TPU, provides control of the terminal portion of the system. The second is called the CP/M[®] Processor, or CPU. This processor gives the HP 125 its stand-alone computer personality within the framework of the CP/M Operating System.

To gain a better understanding of the entire system, we will take an in-depth look at each of the processors separately, and then consider the logical interconnection between the two. In this way, the total system can be understood through its component parts.

CP/M[®], which stands for Control Program for Microcomputers, is a registered trademark of Digital Research, Pacific Grove California.

The HP 125 as Terminal

The Terminal Processor Unit, or TPU, is that part of the HP 125 which functions as an intelligent terminal. The individual elements which are included in the TPU are:

- A Z80A processor chip which operates at a clock speed of 3.68MHz.
- 32K bytes of ROM which contains: the terminal firmware; firmware for communication with the CP/M processor; system defined softkey labels and definitions; and other fixed code such as the built-in self-test routines.
- 16K bytes of RAM memory used for the Z80A stack, softkey labels and definitions, data communications buffers, and 120 lines of CRT display memory.
- Two asynchronous data communications ports which provide external interface to RS-232 devices such as printers and other computer systems.
- An interface to the keyboard unit. This also provides a direct interrupt to the Z80A for 'hard reset' conditions.
- Video control logic for output to the CRT display with a variety of video enhancements, one per line.
- 256 bytes of non-volatile CMOS RAM memory with battery back-up for storage of the HP 125 configuration selections such as line frequency and data communication options.
- Control logic for the optional built-in thermal print mechanism (TPM).
- A parallel 'mailbox' for passing commands and data to and from the CP/M processor.

Terminal Operation

The TPU functions in essentially the same manner whether it is attached to a remote computer system via either RS-232 data communications port or to the CP/M processor via the internal 'mailbox'. The only differences involve 'trigger' characters which may be required from remote systems to initiate softkey transfers and terminal status transmissions. Refer to the HP 125 Owner's Manual for information on these remote trigger transfers.

The TPU, as an intelligent device, may be 'programmed' by the user. By this, we mean that most features of the HP 125 can be controlled by a program by means of 'escape sequences'. The functions which can be controlled are described in the

next several chapters, and include virtually every 'edit' and 'cursor control' key defined on the keyboard.

An 'escape sequence' is a series of one or more characters following the ASCII 'escape' character, decimal 27. In general, an escape sequence which includes only one character in addition to the 'escape' (ESC) character is associated with a keyboard function. Multiple character escape sequences are required for more advanced functions such as direct cursor addressing and programmatic system configuration.

Programming the TPU is discussed in greater depth in Chapter 3. Further, individual multiple character escape sequences are presented in depth in Chapters A and B, while simple two character escape sequences are summarized in Chapter C.

The HP 125 as Computer

In addition to the TPU, there is a second processor which makes the HP 125 a computer system in its own right. This second computer is referred to as the CP/M Processor, or CPU, because the Operating System used by this processor is CP/M.

CP/M is a widely used disc operating system for 8080 and Z80 based computer systems. Application programs written to execute with CP/M can generally be transported between a variety of different hardware as long as no hardware-dependent features are used. Since the HP 125 uses a superset of standard CP/M, most CP/M programs should execute on the HP 125 with little or no conversion effort.

Like the TPU, there are several components which make up the CPU. The major components are:

- A Z80A processor chip which operates at a clock speed of 3.68MHz.
- 8K bytes of ROM memory which contains the disc bootstrap loader, CPU diagnostic routines, and logic for mailbox communication with the TPU.
- 64K bytes of RAM memory used for the CP/M Disc Operating System and for user programs.
- An HP-IB controller chip which is used to drive all disc devices, plotters, and some printers.
- A parallel 'mailbox' for passing commands and data to and from the TPU.

System Functioning

While the HP 125 can perform as an intelligent terminal in a stand-alone fashion, a disc device must be attached in order to make effective use of the CPU. The minimum functional system includes the HP 125 System Unit and at least one disc drive. All other devices are technically optional, although a printer is generally considered necessary for a useful system.

As you have seen in the HP 125 Owner's Manual, the system can operate in any of three modes. Briefly, these are:

LOCAL OP SYS Mode. This mode, which requires a properly functioning disc device, logically connects the TPU to the CPU. The System Unit provides access to CP/M and to optional software application packages.

REMOTE Mode. When in this mode, the HP 125 TPU is logically connected to another computer system as an intelligent terminal. This connection can be via either of the two serial data communication ports, although the host system may require some action to complete the connection.

LOCAL Mode. The HP 125 is in this mode when it is not specifically in one of the other two modes. In this mode, the TPU is a stand-alone device which can perform some local typing and printing functions, but no data storage is available. Characters typed generally appear only on the CRT display.

System Operation

When you first turn on the System Unit, both the TPU and the CPU will perform a series of 'self-test' routines contained in the HP 125 ROM memory. If an error is discovered, a code is displayed on small light emitting diodes (LEDs) within the HP 125, which identifies the problem for authorized service representatives.

After several seconds, the TPU continues operation if possible by displaying a code on the HP 125 CRT. Refer to the HP 125 Owner's Manual for additional information on these 'power-on' errors.

If all components pass the self-test, the configuration will be just as it was when power was turned off. That is, if the HP 125 was in REMOTE Mode when it was turned off, it will be in that mode when again turned on. If the unit was left in LOCAL OP SYS Mode, the HP 125 will attempt to load the CP/M Operating System from the disc drive. This procedure is described in complete detail in the HP 125 Owner's Manual and will not be repeated here.

Summary

This concludes our quick introduction to the HP 125 system. Now we will look at the details of the TPU, and how the many features can be accessed under program control using 'escape sequences'. Remember that, for the most part, the TPU makes no distinction between the internal CP/M Processor and a remote host system. The TPU is programmed the same in both cases.



THE HP 125 TERMINAL

Introduction

The HP 125 Terminal Processor, the TPU, operates as a terminal device in both REMOTE and LOCAL OP SYS modes. In the first case, the TPU functions only as a smart terminal device under control of a remote host computer system via one of the two RS 232 data communications ports. In the second case, the TPU serves as the CP/M console device.

For the most part, the TPU makes no distinction between the remote host system and the local CPU: for this reason, we will refer only to the 'host system' throughout this manual. In the few cases where a distinction is made, examples will be provided for both remote and local programming.

Programming the TPU

Because the TPU is a smart terminal device, many of the features which can be altered locally by an operator can also be controlled by a program executing in the host system. These features range from controlling the display to changing the TPU configuration, but they all have one thing in common: all of these features are controlled by 'escape sequences'.

An 'escape sequence' is any ASCII string which begins with the 'escape' character, ASCII code 27 decimal. Depending upon what character follows the 'escape', the TPU will determine whether the sequence is a 'two character' sequence or a 'multiple character' sequence.

Two character sequences usually correspond to those functions available through a keystroke on the HP 125 keyboard. For example, the [CLEAR DSPLY] key corresponds to a two character sequence. Multiple character sequences are generally used for those functions which require one or more parameters. To direct the cursor to a specific row and column position requires a multiple character escape sequence.

Throughout this manual, two conventions will be used with regard to escape sequences. First, the 'escape' character will be shown as 'ESC'. Second, sample escape sequences will be shown with blank spaces between each character of the escape sequence. This is done for clarity only. ACTUAL ESCAPE SEQUENCES SHOULD NOT INCLUDE BLANK CHARACTERS.

While all of these escape sequences can be entered directly at the keyboard, the TPU will not 'echo' characters to the display. This can make it difficult to verify the accuracy of a particular sequence. Also, CP/M and other operating systems may interpret the 'escape' character as a special control character, and may make direct keyboard entry difficult except in LOCAL TERMINAL mode. For these reasons, most escape sequences are performed within a program executing at the host system.

Two Character Escape Sequences

As mentioned above, two character escape sequences are generally those which perform the same functions as can be performed by an operator at the HP 125 keyboard. In the case cited above, the [CLEAR DSPLY] key corresponds to the sequence:

ESC J

In this case, when the TPU receives this sequence, the display will be cleared just as it would be if the [CLEAR DSPLY] key had been pressed.

An important characteristic of two character escape sequences is that the second character must be specified exactly. A lower case 'j' would not perform the same function. In fact, 'ESC j' corresponds to the SHIFT-[USER KEYS] keystroke and causes the User Key Definition menu to be displayed.

Multiple Character Escape Sequences

Many of the terminal features require one or more parameters to be specified as part of the escape sequence. For example, to position the cursor at a particular screen address, you may wish to specify both a row number and a column number. Either or both may be specified within the same escape sequence, but the TPU must have some way of recognizing the end of the sequence.

For example, let's look at one of the 'cursor addressing' escape sequences described in depth in Chapter 5. The general form of specifying a cursor position uses the sequence:

```
ESC & a <row> r <col> C
```

Here, <row> and <col> represent the desired row and column values respectively. As you will see in Chapter 5, the values specified are absolute positions, so the upper left hand corner of the screen is row 0, column 0.

The order of the parameters is not important. The letter 'r' is used to indicate which parameter specifies the <row> address, while the letter 'C' is used to mark the <col> address.

For example, to position the cursor to column 20 in row 10, the escape sequence which might be used is:

```
ESC & a 1 0 r 2 0 C
```

Because the number of characters within a specific escape sequence depends upon the <row> and <col>, the TPU must have some way to identify the end of the sequence. The terminator in this and in all multiple character escape sequences is the first upper case character following the 'ESC' character. This is one of the principle differences between multiple character sequences and two character sequences discussed earlier: in multiple character sequences, only the final parameter character is upper case.

Because the sequence of parameters is not critical, the example above is equivalent to the sequence:

```
ESC & a 2 0 c 1 0 R
```

Notice that the <col> parameter is indicated by a lower case 'c' in this case. The 'R' at the end of the sequence indicates the end: the next character received by the TPU will be processed normally.

Since the TPU recognizes the first upper case character as the end of an escape sequence, you may elect to specify only one of the two possible parameters. To move the cursor to the left edge of the screen on the same line, regardless of the position of the left margin, you could use the sequence:

```
ESC & a 0 C
```

Because of the 'C', the TPU will not expect additional parameters as part of this escape sequence. Similarly, to move the cursor to row 5 without changing the column position, use the sequence:

```
ESC & a 5 R
```

Remember, the first upper case character which follow the 'ESC' character is treated as the end of the sequence.

Escape Sequence Compatibility

The escape sequences defined for the HP 125 are consistent with those defined on other Hewlett-Packard terminal devices. Of course, not all escape sequences are appropriate on all terminal devices. However, where a particular function is common to both terminal devices and the HP 125 TPU, the same escape sequence is used.

Because some escape sequences which might be received by the HP 125 are not valid, the TPU will ignore any invalid sequences rather than create an error condition.

Manual Organization

The escape sequences for the HP 125 are presented through the next several chapters. In Chapter 3, you will see the multiple character sequences which are used to program the configuration parameters discussed in the HP 125 Owner's Manual and available through the 'config' menu. While some parameters, such as the 'ReturnDef' field, can not be changed under program control, there is one parameter which can not be changed except through an escape sequence: disabling and enabling the terminal 'bell'.

Chapter 4 continues with multiple character sequences used to control the function and function control keys. With these sequences, you will see how to select the [AIDS], [MODES], or [USER KEYS] set of functions; how to define the labels and definitions of the [USER KEYS]; and even how to 'turn off' the softkey labels entirely.

The escape sequences which control the display, keyboard, and printer devices are presented in Chapter 5. Many of these are two character escape sequences which correspond to keyboard keystrokes available to the operator. Where this is the case, only a summary description is provided of each function. The detailed description can be found in the HP 125 Owner's Manual.

Finally, those sequences which require a response from the HP 125 TPU are included in Chapter 6. These sequences range from status requests and device identification to remote cursor position sensing. They result in a response from the TPU which can be read by the controlling program. In using these sequences, the TPU does differentiate between requests made from the CPU and those made from remote hosts. This is the only area of TPU operation in which the source of an escape sequence affects its operation.

Summary

In this chapter, you saw how to use escape sequences to program features of the HP 125 TPU. In the next several chapters, you will see just what escape sequences to use to perform many of the keyboard functions described in depth in the HP 125 Owner's Manual.



PROGRAMMATIC CONFIGURATION

Introduction

In the HP 125 Owner's Manual, you saw how an operator can use the 'config' menu to alter the characteristics of the HP 125 in both REMOTE and LOCAL OP SYS Modes. Under program control, you can access and modify many of those same parameters, as well as several which are not available to the operator. For example, the HP 125 can be configured so that the ASCII 'bell' will not sound. You can also 'lock' the configuration so the operator can not alter any of the configuration parameters.

As you will remember from the HP 125 Owner's Manual, a variety of terminal characteristics can be modified through the 'config' menu. These straps, which affect the way in which the TPU treats data communications and other system parameters, may be thought of as falling into one of three categories: Strap Configurations, Keyboard Characteristics, and Configuration Control. The HP 125 offers a unique escape sequence for controlling each type of category.

Keyboard Characteristic Control

The characteristics of the HP 125 keyboard can be controlled using the following sequences. Many are available to the operator through the 'Terminal Configuration' section of the 'config' menu, and others are accessible via the [MODES] key definitions. One, the 'Bell' control, is only available through this escape sequence.

The asterisk (*) indicates the default state.

AUTO LINE FEED

Enable: ESC & k 1 A (Auto line feed)
* Disable: ESC & k 0 A (No auto line feed)

CAPS LOCK MODE

Enable: ESC & k 1 C (All caps)
* Disable: ESC & k 0 C (Upper and lower case)

BELL

* Enable: ESC & k 1 D (Bell character rings bell)
Disable: ESC & k 0 D (No bell ring)

LINE FREQUENCY

50 Hz: ESC & k 1 J (European)
* 60 Hz: ESC & k 0 J (North American)

LOCAL ECHO

Enable: ESC & k 1 L (Half-duplex)
* Disable: ESC & k 0 L (Full duplex)

MODIFY ALL

Enable: ESC & k 1 M (Modify mode on)
* Disable: ESC & k 0 M (Modify mode off)

SPOW Latch

- Enable: ESC & k 1 N (Space bar won't destroy screen)
- * Disable: ESC & k 0 N (Space bar generates blanks)

(See the discussion of this latch near the end of this chapter)

CAPS MODE

- Enable: ESC & k 1 P (Reverse SHIFT key function)
- * Disable: ESC & k 0 P (SHIFT characters upper case)

KEY CLICK

- * Enable: ESC & k 1 Q (Keystrokes 'click')
- Disable: ESC & k 0 Q (Keystrokes 'quiet')

REMOTE MODE

- Enable: ESC & k 1 R (HP 125 is a remote terminal)
- * Disable: ESC & k 0 R (HP 125 is a local terminal)

LOCAL OP SYS MODE

- Enable: ESC & k 2 R (HP 125 is CP/M console)
- * Disable: ESC & k 0 R (HP 125 is a local terminal)

Each of the above Keyboard Control sequences will affect some characteristic of the HP 125, and each field is explained in detail in the HP 125 Owner's Manual. Refer to that document if additional information is required concerning a particular configuration option.

Strap Configurations

As you read in the Owner's Manual, many of the parameters related to data communications can be controlled by use of several straps in the TPU. These can be set or cleared under program control using this group of escape sequences.

TRANSMIT FUNCTION KEY

- Enable: ESC & s 1 A Strap = A
- * Disable: ESC & s 0 A Strap = a

SPACE OVERWRITE

- Enable: ESC & s 1 B Strap = B
- * Disable: ESC & s 0 B Strap = b

INHIBIT CURSOR WRAPAROUND

Enable: ESC & s 1 C Strap = C
* Disable: ESC & s 0 C Strap = c

INHIBIT HANDSHAKE

Enable: ESC & s 1 G Strap = G
* Disable: ESC & s 0 G Strap = g

INHIBIT DC2

Enable: ESC & s 1 H Strap = H
* Disable: ESC & s 0 H Strap = h

INHIBIT SELF-TEST

Enable: ESC & s 1 L Strap = L
* Disable: ESC & s 0 L Strap = l

Configuration Control

Many of the parameters included in the 'config' menu, and most features available via the keyboard, can be protected from user modification by using the following sequence.

ESC & q 1 L

This 'locks' the TPU configuration as it appears in the menu. The parameters will remain locked until the following sequence is received:

ESC & q 0 L

If locked, the configuration will remain locked, preventing user modification until this second sequence is received. Any escape sequences generated from the host system or the keyboard will be executed even when the configuration is locked. This allows a program to modify a configuration parameter even when it has locked such changes from the user.

Programming the SPOW Latch

The HP 125 Owner's Manual describes the effect of the space over-write, or B, strap. It also describes how it can be enabled or disabled under operator control.

If the SPOW feature is desired, it can be controlled programmatically. The state of the strap is set or cleared using the 'ESC & s' sequence described above. This is functionally equivalent to manually setting or clearing the 'B' strap within the 'config' menu.

Once the B strap is enabled, the space overwrite is turned on by using the 'ESC & K1N' escape sequence. This can be turned off with the 'ESC & KON' escape sequence. The SPOW latch will also be turned 'off' by a line feed or a TAB character, or by moving the cursor up one line. This is consistent with the operator SPOW latch. Of course, the latch can be enabled or disabled at any time by the operator as described in the HP 125 Owner's Manual.

Summary

This concludes the discussion on the programmatic control of the HP 125 system configuration control. In the next chapter, you will see how to use other escape sequences to control and define the function and function control keys.

PROGRAMMING THE FUNCTION KEYS



Introduction

In the previous chapter, you saw how to use escape sequences to control many of the configuration parameters available for the HP 125 TPU. In this chapter, you will learn how to use escape sequences to control the function and function control keys as described in the HP 125 Owner's Manual.

Function Control Sequences

From a program executing in a host computer you can control which function key labels are displayed on the screen, or even if any labels are to be displayed. This is done using the escape sequences described throughout the remainder of this chapter. The features to be described will allow you to:

- Remove the key labels from the screen entirely
- Enable the MODES Softkey Definitions
- Enable the USER KEYS Softkey Definitions
- Lock the currently displayed labels on the screen
- Re-enable the displayed keys

To perform the features described above, see the sequences described in Table 4-1. Of course, many of these features can be accomplished using keystroke sequences. However, keep in mind that if the escape sequence is entered locally the function will also be executed.

Table 4-1. Function Group Control Sequences

ESC & j @	Remove key labels from the screen
ESC & j A	Enable MODES softkeys
ESC & j B	Enable the USER KEYS
ESC & j S	Disable the AIDS, MODES, and USER KEYS keys.
ESC & j R	Re-enable the AIDS, MODES, and USER KEYS keys.

The sequence which removes the key labels from the screen, 'ESC & j @', is functionally equivalent to SHIFT-[AIDS].

The [MODES] label set is enabled with the sequence 'ESC & j A'. This sequence is functionally equivalent to pressing the [MODES] key.

The sequence which is equivalent to pressing the [USER KEYS] key 'ESC & j B'. Unlike pressing the [USER KEYS], however, this sequence will always display the [USER KEYS] and not toggle between the most recently selected set of [AIDS] or [MODES]. Refer to the HP 125 Owner's Manual for clarification on this toggling action.

To disable the function control keys, use the 'ESC & j S' sequence. This allows the function keys to be used, but prevents another function key set from being selected, either by the operator or by escape sequence.

The function control keys are re-enabled by using the last sequence shown, 'ESC & j R'. This can be entered either from the keyboard or transmitted by a host program.

Defining the User Keys

In addition to performing the system defined functions, all eight special function keys ([f1] through [f8]) can be assigned special functions definable by the user. These 'softkeys' can be assigned meanings by a person operating the HP 125. They can also be defined under program control, either from a remote host system or from the HP 125 system processor.

Some computer equipment features 'definable' keys which transmit a pre-defined set of characters. To use such keys, an application must be programmed to recognize this fixed set. The HP 125, however, like most Hewlett-Packard terminal products, features a much more comprehensive meaning of 'definable'. When we say a key is 'user definable', we mean:

- The user can assign any string of ASCII characters to be associated with each key.
- In addition, the user can specify whether those characters are to be transmitted to the host, with or without a 'RETURN' appended to the text, or whether the character string is to be performed at the TPU only.
- Finally, a screen label can be assigned to each key. These labels, which are automatically displayed at the lower edge of the screen, need not be identical to the text associated with a particular key. For example, a key might be labeled "DISC CATALOG" when, in fact, the string sent to the host system might be "DIR".

The 'RETURN' key itself is definable: however, no special label can be assigned to this key. Its definition is determined by the bytes specified in the 'RetrunDef' field within the 'config' menu.

Defining Keys Under Program Control

You can define the USER KEYS under program control as well. To do this, simply print to the screen an escape sequence as described below. Note that the same sequence, typed at the keyboard, will also define a key without entering the menu described above. Nonetheless, you will find it much easier to define keys locally using the menu!

To define a key from a host system, send a sequence in this format:

```
ESC & f <attr> <key> <len1> <len2> <label> <def>
```

where the various fields are defined as:

<attr>, the attribute field, is:

```
0 a for Normal
1 a for Local
or 2 a for Transmit
```

<key>, the key number, is:

```
1 k for key [f1]
2 k for key [f2]
thru 8 k for key [f8]
```

<len1>, the length of the label field, is:

```
0 d thru 16 d
```

<len2>, the length of the definition string, is:

```
0 l thru 80 l
```

<label>, the label string, is any string of ASCII characters of length <len1>.

<def>, the definition string, is any string of ASCII characters of length <len2>.

The <attr>, <key>, <len1>, and <len2> parameters may appear in any order, but must precede the <label> and <def> fields. You must use lowercase parameters in all except the last parameter defined: this is consistent with all HP 125 escape sequences. It is the uppercase letter which causes the sequence to be executed. Of course, both upper and lower case letters may be used in the label and definition fields.

The first <len1> characters following the last parameter are considered to be the label and are displayed in the label field. The next <len2> characters become the actual key definition.

As with many other multiple character escape sequences, not all of these parameters need be specified with each sequence. Any parameter not specified will default to the following values:

```
<attr> defaults to '0a', Normal
<key> defaults to '1k', Key [f1]
<len1> defaults to '0d', Null <label>
<len2> defaults to '1l', One byte <def>
```

A few words of caution are in order. You might conclude that the terminators for the <len1> and <len2> fields described on the previous page have been transposed. The proper letter to terminate the length of the label field really is 'd'. Likewise, the proper character to delineate the definition field is 'l'. This assures compatibility with other HP terminal products. The 'l' stands for 'length', and was defined before label definitions became definable.

One more word of caution. The actual label and definition characters follow the upper case terminator. This is unusual in the world of softkey definitions. Nonetheless, remember to make the last field terminator (usually the 'L') an upper case character, then provide the actual strings. Note the examples through the following pages.

Also, remember that the numeric values specified in both the label and definition length fields may exceed the maximum given. However, only the first 16 characters are used for the label length and only 80 characters are used for the definition field. Escape sequence characters are included in the count.

Let's take a look at a few examples of key definitions. For our purposes, we wish to define three softkeys as follows:

Key [f1] is to send a log-on message to a host HP 3000. The label is to read 'SAY HELLO', and the string to be sent is 'Hello MBK.HP125'.

Key [f2] is to be used as a typing aid to help delete several files all of which begin with the characters 'MMSG'. The label is to be 'DELETE FILES', and the definition is to be 'PURGE MMSG'. The user will supply the remaining two characters of each file name and press 'RETURN'.

Key [f3] is to be defined so that, with a single key, the entire HP 125 screen can be erased. The label is to read 'CLEAR SCREEN', and the definition is to be 'ESC h ESC J'. The host system is not to receive the sequence.

Key [f1] should be transmitted directly, so it will be assigned as a 'Transmit' key. Key [f2] requires additional input on each line, so it will have a 'Normal' attribute. Key [f3] is to effect the local display only, so an attribute of 'Local' is appropriate.

To define these same three keys programmatically, the following three sequences should be sent. This example is printed to the screen by a BASIC/125 program running locally. Note how the escape character is generated.

```
PRINT CHR$(27)+"&f1k2a9d15LSAY HELLOHello MBK.HP125"  
  
PRINT CHR$(27)+"&f2k0a12d10LDELETE FILESPURGE MSG"  
  
PRINT CHR$(27)+"&f3kla12d4LCLEAR SCREEN"+CHR$(27)+"h"  
      + CHR$(27)+"J"
```

The receipt of a correct 'ESC & f' sequence will result in a key label or definition being defined according to the remainder of the sequence. However, simply defining a key is not sufficient to display the [USER KEYS] labels. To cause your new labels to be displayed, you must send the 'ESC & j B' sequence described earlier in this chapter.

Executing a User Key

From a program executing in a host system, you can trigger the execution of any of the User Keys. The sequence which accomplishes this feat is of the form:

```
ESC &f <key> E
```

where <key> can be a numeric value in the range 1 through 8. Keys [f1] through [f8] correspond to values 1 through 8 respectively.

This sequence allows you to 'press' a softkey defined in the [USER KEYS] menu from a program, whether or not the [USER KEYS] labels are actively displayed.

Selecting the [USER KEYS] Menu

While not necessarily useful as such, there are two more escape sequences associated with the function and function control keys. These two sequences are those which select the [USER KEYS] menu, and which return to the previous key definitions upon exiting of the [USER KEYS] menu.

To display the [USER KEYS] menu, the TPU must receive the sequence:

```
ESC j
```

This is functionally equivalent to pressing the SHIFT-[USER KEYS] keys as described in the HP 125 Owner's Manual.

To exit the [USER KEYS] menu programmatically, use the sequence:

```
ESC k
```

This is functionally equivalent to pressing the [USER KEYS] key while in the [USER KEYS] definition menu.

Summary

In this chapter, you have seen all the escape sequences which control the function and function control keys. You now know how to define and select [USER KEYS], how to select [AIDS] and [MODES], and how to disable and enable the label display.

In the next chapter, you will see a variety of additional escape sequences which are associated with display and keyboard control functions. Many of these are functionally equivalent to keys on the HP 125 keyboard, and provide programmatic access to user features.



DISPLAY, KEYBOARD AND PRINTER CONTROL

Introduction

In this chapter, you will see both two character and multiple character escape sequences for control of the display and keyboard. Many of these sequences are equivalent to keystrokes available from the keyboard, while others must be entered as escape sequences, either from the keyboard or from a program.

Cursor Positioning

Since the HP 125 has five pages of screen 'display memory', it is possible that the first row of the display is not the first line in memory. For this reason, there are two ways to position the cursor at a particular row address: by specifying which line on the screen, or by specifying which line of display memory. Column addressing is treated identically in both cases.

When the cursor is to be located at one of the 120 lines of display memory, use Absolute Addressing. When the row is defined in relation to the top of the screen, use Screen Relative Addressing.

Absolute Addressing

This type of addressing refers to positioning the cursor by specifying the actual row and column to which the cursor is to move. To specify an absolute position in display memory, send the following sequence:

```
ESC & a <col> c <row> R
```

The <col> parameter is a decimal number between 0 and 79 and represents the desired column to which the cursor should be moved.

The <row> parameter is a decimal number between 0 and 119 and specifies the actual row of display memory to which the cursor should be moved. Note that, similar to the Roll Up and Roll Down keys, the cursor may be moved on the screen to meet the desired line at the bottom or top of the screen. This will happen if the specified row does not already appear on the screen.

Screen Relative Addressing

To specify a position relative to the top of the screen (rather than the top of display memory), send this sequence instead:

```
ESC & a <col> c <row> Y
```

The <col> parameter is as defined above, however the <row> parameter should be a decimal number between 0 and 23, the number of lines actually available on the display. If the value of <row> here is greater than 23, the cursor will be positioned on the bottom line of the display. No error is generated.

If you wish to change only the <row> or the <col> position, simply send the desired parameter, remembering to terminate the escape sequence with an upper-case character. For example, to move the cursor to the top line of the display without affecting the column position of the cursor, you could send:

```
ESC & a 0 R
```

Likewise, to move the cursor to the left edge of the screen, send:

```
ESC & a 0 C
```

and the cursor will change column position only.

Cursor Relative Addressing

Occasionally, you will want to move the cursor relative to the current position of the cursor. To perform this task, the escape sequence must take the format:

```
ESC & a <scol> c <srow> R  
or  
ESC & a <scol> c <srow> Y
```

In this case, <scol> must be a signed decimal number. That is, a '+' or '-' sign must precede the actual decimal value in order for the 'cursor relative' addressing to take effect.

Also, <srow> must be a signed decimal integer. Note, however, that the row address can be either a screen address or an actual row address depending upon whether a 'Y' or 'R' is used in the sequence.

As with Absolute Addressing above, you may specify only one of the parameters if you wish. That is, you may move the row address only by specifying only the 'R' portion of the sequence.

For example, if you need to move the cursor up two lines and five columns to the right, you could send:

```
ESC & a + 5 c - 2 R
```

Note that the range of row and column addresses discussed under 'Absolute Addressing' above remain valid here. If the valid bounds will be exceeded using cursor relative addressing, the cursor will move to the limit in the direction indicated by the cursor relative sequence. For example, if the cursor is located in column 5 and this sequence is received:

```
ESC & a - 1 0 C
```

the cursor will move to column 0 and no further.

Additional Cursor Features

In any of the escape sequences described above, you can mix the different modes of cursor addressing. For example, you can specify a cursor relative column address with a screen relative row address. Or, you could specify an absolute row with an unsigned column address. To move the cursor to row 50 of display memory, five columns to the right of the current cursor location, you could use the sequence:

```
ESC & a + 5 c 5 0 R
```

Cursor Positioning Sequences

This concludes our discussion of direct cursor addressing control sequences. There are other two character escape sequences which can be used to position the cursor. These functions correspond to the various keys on the HP 125 keyboard. The functioning of each is discussed in depth in the HP 125 Owner's Manual: refer to that document for a detailed explanation of any of these functions.

Name: Cursor Up
Function: ESC A

ESC A is equivalent to the 'up arrow' key.

Name: Cursor Down
Function: ESC B

ESC B is equivalent to the 'down arrow' key.

Name: Cursor Right
Function: ESC C

ESC C is equivalent to the 'right arrow' key.

Name: Cursor Left
Function: ESC D

ESC D is equivalent to the 'left arrow' key.

Name: Home Down
Function: ESC F

ESC F is equivalent to a SHIFT-Home Up keystroke.

Name: Return
Function: ESC G

This function returns the cursor to the logical left margin without any data transfer taking place.

ESC G should not be confused with 'RETURN' or 'ENTER': it does not effect the executing program. It simply moves the cursor to the left margin.

Name: Home Up
Function: ESC H
ESC h

ESC H is equivalent to the sequence ESC h and to the keystroke 'Home Up' of the cursor control group.

This concludes cursor addressing. Note that there are two additional sequences related to cursor positioning. These are used for cursor position sensing by a remote program. Since they result in transmission of a reply from the HP 125, these sequences are discussed in Chapter 6.

Display Enhancements

The terminal includes as a standard feature the following display enhancement capabilities:

- Inverse Video - Black characters are displayed against a white background.
- Underlined Text - Characters are underscored.
- Blinking Text - Characters blink on and off.
- Half-Bright Text - Characters are displayed at a lower intensity.

These enhancements can be enabled one at a time, or in any combination. However, only one enhancement or combination of enhancements can be enabled on any line of display memory at a time. Keep this in mind as you program various fields of enhancements.

From the keyboard, you can select and enable enhancements one character at a time. You must first select which combination of enhancements you wish to enable using the 'config' menu. Then, you must use the 'enhance select' key in the [AIDS] menu (key [f5]) to select which character(s) to enhance or reset. Again, only the most recently selected enhancement(s) may be included on a particular line.

Under program control, you can use a single escape sequence to both select and enhance one or more characters on the display. To do this, you must 'turn on' the desired enhancement, prior to printing your data. If you do not turn off your enhancements, the HP 125 will automatically turn it off for you when you move the cursor to another line of text (for example, with a 'carriage return' character).

The general form of the enhancement escape sequence is:

```
ESC & d <code>
```

In this case, <code> is an upper-case character from Table 5-1. The character you transmit will determine what enhancement is displayed.

Table 5-1. Display Enhancement Codes

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
BLINKING	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
INVERSE		X	X			X	X			X	X			X	X
UNDERLINE				X	X	X	X					X	X	X	X
HALF-BRIGHT								X	X	X	X	X	X	X	X

For example, to print a string of characters in underlined inverse video, send this sequence to turn on enhancements:

ESC & d F

To display characters normally once again, simply send:

ESC & d @

You may toggle these modes on and off several times on a line. Remember, however, if you use a different enhancement on the same line, all enhanced fields will be identical to the most recently selected one. This is done independently of any selection within the 'config' menu.

Margins/Tabs/Columns

The next set of escape sequences is similar to those functions available through the 'margins/tabs/col' function key available through the [AIDS] key. These functions affect the format of the screen in the definition of the left and right margins; the location of the tab 'stops'; and the use of the [TAB] and SHIFT-[TAB], or back-tab, keys.

Name: Set Tab
Sequence: ESC l

This function will set a 'tab stop' at the column in which the cursor is positioned when the sequence is received by the terminal. Normally, the cursor is positioned at the desired column prior to sending this sequence.



Name: Clear Tab
Sequence: ESC 2

This function will clear any 'tab stop' that might be active at the current cursor column. If no tab is defined at the current location, the sequence is ignored and no error is generated. The tab stop at the left margin cannot be cleared.

Name: Clear All Tabs
Sequence: ESC 3

The function clears all tabs active on the display line. If no tabs are defined, the sequence is ignored and no error is generated. Like the Clear Tab sequence above, the tab at the left margin cannot be cleared.

Name: Tab Forward
Sequence: ESC I

This function causes the cursor to advance to the right until the next tab stop is reached. If no tab stop is located between the cursor column and the right margin, the cursor is advanced to the left margin (default tab stop) on the next line of display.

This sequence is functionally identical to the [TAB] key.

Name: Back Tab
Sequence: ESC i

This function causes the cursor to move backwards to the previous tab stop on the current line. If the cursor is located at the logical left margin, this sequence moves the cursor to the rightmost tab stop defined on the previous line of display memory.

This function is equivalent to pressing the SHIFT-[TAB] key.

Name: Set Left Margin
Sequence: ESC 4

This function sets the left margin to the current cursor position. Normally, this sequence follows a cursor control sequence. An attempt to locate the left margin to the right of the right margin is considered an error: no action is taken, and the margin position remains unchanged.

Name: Set Right Margin

Sequence: ESC 5

This function sets the right margin to the current cursor position. Normally, this sequence follows a cursor control sequence. An attempt to locate the right margin to the left of the left margin is considered an error: no action is taken, and the right margin remains unchanged.

Name: Clear All Margins

Sequence: ESC 9

This function is used to reset both the left and right margins to the left and right edges of the screen, respectively. No cursor positioning is necessary, and the margins are automatically reset.

Edit Control Sequences

The keys included in the 'Edit Control Group' in the HP 125 Owner's Manual may also be effected under program control. The sequences used to perform these functions follow.

Name: Clear Display

Sequence: ESC J

ESC J is equivalent to the [CLEAR DSPLY] key.

Name: Clear Line

Sequence: ESC K

ESC K is equivalent to the [CLEAR LINE] key.

Name: Insert Line

Sequence: ESC L

ESC L is equivalent to the [INS LINE] key.

Name: Delete Line

Sequence: ESC M

ESC M is equivalent to the [DEL LINE] key.

Name: Delete Character

Sequence: ESC P

ESC P is equivalent to the [DEL CHAR] key.

Name: Insert Character Mode
Sequence: ESC Q

ESC Q is equivalent to the [INS CHAR] key when 'Insert Character' is not enabled. This sequence turns 'on' 'Insert Character'.

Name: Disable Insert Character
Sequence: ESC R

This function turns off the insert character mode described above.

ESC R is equivalent to striking the [INS CHAR] key when 'Insert Character' is enabled.

Name: Roll Up
Sequence: ESC S

ESC S is equivalent to the [ROLL UP] key.

Name: Roll Down
Sequence: ESC T

ESC T is equivalent to the [ROLL DOWN] key.

Name: Next Page
Sequence: ESC U

ESC U is equivalent to the [NEXT PAGE] key.

Name: Previous Page
Sequence: ESC V

ESC V is equivalent to the [PREV PAGE] key.

Special Terminal Functions

This section contains those remaining sequences useful in the remote operation of the HP 125. They are used just as the previously described sequences.

Name: Dump Display Memory
Sequence: ESC 0

This function, when received by the display portion of the HP 125 causes all of display memory to be printed to the currently selected printer device(s). If no printer is enabled, the TPU will display an error message stating that 'No "TO" Device' is selected. To clear the error, press RETURN, select a printer, and re-try this function. ESC 0 is similar to the keystrokes 'home-up' followed by a 'COPY ALL' key discussed in the HP 125 Owner's Manual. Pressing [ENTER] while in LOCAL mode performs the same function.

Name: Display Functions On
Sequence: ESC Y

ESC Y is equivalent to the 'DISPLAY FUNCTIONS' softkey accessible via the [MODES] key when display functions mode is disabled.

Name: Display Functions Off
Sequence: ESC Z

ESC Z is equivalent to the 'DISPLAY FUNCTIONS' softkey accessible via the [MODES] key when the 'display functions' mode is enabled. It is ignored if display functions mode is already enabled.

Name: Enable Keyboard
Sequence: ESC b

This function re-enables keyboard operation. This sequence is normally used after the keyboard disable function has been received by the terminal. If the keyboard is already enabled, this sequence is ignored. If the keyboard is disabled, this sequence may not be generated at the keyboard. Pressing [RESET] is the only option available to the operator to re-enable the keyboard.

ESC b has no equivalent keystroke.

Name: Disable Keyboard
Sequence: ESC c

This function causes all keyboard input to be ignored and a bell to be sounded whenever any key is struck. Only the [RESET] and SHIFT-CTRL-RESET are active. This is normally used when an application needs to insure that no user input will occur to disturb processing or output. Once disabled, the keyboard is re-enabled by the 'ESC b' sequence described above executed under program control, or by using the [RESET] key.

ESC c has no equivalent keystroke.

Name: Enter
Sequence: ESC d

This function performs a programmatic [ENTER], and operates only slightly differently from that key. 'ESC d' sends all data at and to the right of the current cursor position on the current line to the host system. Remember that the [ENTER] key sends all data starting at the 'StartCol' or the 'start-of-text' pointer discussed in the Owner's Manual. This sequence can not be used to print memory as can be done with the [ENTER] key in Local mode.

Name: Modem Disconnect
Sequence: ESC f

This sequence, when received by the HP 125 terminal, causes the 'carrier detect' signal on Datacomm Port #1 to 'go low' for 200 nanoseconds. The functional outcome of this operation is that the optional HP modem (HP 13265A) will perform a 'disconnect' and hang up the datacomm line in use.

ESC f has no equivalent keystroke.

Name: Soft Reset
Sequence: ESC g

ESC g is equivalent to the ['RESET'] key.

Name: Datacomm Selftest
Sequence: ESC x

ESC x is equivalent to the 'DATACOMM SELFTEST' key available via the [AIDS] softkey menu.

Name: Terminal Self-test
Sequence: ESC z

ESC z is equivalent to the 'SELF-TEST' key available via the [AIDS] and 'service keys' softkeys, or through the [MODES] softkey definitions.

Name: Terminal Pause
Sequence: ESC @

This sequence causes the TPU to pause for approximately one second. No TPU functions will be performed, including keyboard input or softkey processing.

Printer Control Functions

Many of the printer functions available to the operator through the 'printer control' and 'printer modes' keys within the [AIDS] set of key definitions may also be implemented under program control. The following sequences are used to control printer functioning.

Internal Printer Page Format

The optional internal thermal printer includes three mutually exclusive page printing formats. Selecting one, by definition, disables any other page format option. The possible sequences are:

ESC & p 1 7 C	Set REPORT PRINT page format
ESC & p 1 8 C	Set METRIC PRINT page format
ESC & p 1 9 C	Set Continuous Paging format

Printer Logging

The HP 125 TPU allows text from display memory to be 'logged', or transferred, to a local printer device using 'LOG BOTTOM' or 'LOG TOP'. These two modes are also mutually exclusive. The escape sequences to control logging are:

ESC & p 1 1 C	Enable 'LOG BOTTOM' Mode
ESC & p 1 2 C	Enable 'LOG TOP' Mode
ESC & p 1 3 C	Disable All Logging

Data Transfers

The TPU also allows data to be copied from display memory to one or more active printers. These data transfers are initiated using the sequences described below.

ESC & p B	Perform a COPY LINE
ESC & p F	Perform a COPY PAGE
ESC & p M	Perform a COPY ALL
ESC 0	Similar to COPY ALL

Device Control Completion Codes

When a printer operation is triggered under program control, the TPU reports the status of the transfer as a single byte code. The program must accept the status byte, which will also print to the display.

The terminal responds by sending an "S", "F", or "U". An "S" indicates successful completion, an "F" indicates that the operation failed, and a "U" indicates that the terminal operator interrupted the data transfer by pressing RETURN.

Note that these completion codes cannot be suppressed by configuration parameters or any other means. They are always transmitted and your programs should include input commands explicitly for accepting them.

Note that the terminal sends a CR (or a CRLF if AUTO LINE FEED mode is enabled) following the completion code.

If a data comm error occurs during the transmission of the data record, the device control completion code is unpredictable. Datacomm errors are detected using the TPU status bytes described in Chapter 6.

Summary

In this chapter you have seen a variety of escape sequences used to control the HP 125 TPU. The remaining TPU functions involve TPU status, and are described in depth in the next chapter.



**TERMINAL AND
DEVICE STATUS****Introduction**

This section contains information on how to obtain and interpret terminal status information. You will also see how to determine cursor position, and how to identify whether a remote terminal is an HP 125. In addition to this status information, you can also obtain status information on output devices used with the HP 125.

Status requests are made by sending an escape code sequence to the terminal to select the desired status information. All status requests are treated as block transfers. However, the terminator or 'trigger' character required to initiate status transfers to a remote system are not required for LOCAL OP SYS generated status requests. The HP 125 Owner's Manual contains a discussion of this block trigger using the G and H straps.

Interpreting Status

In response to a status request, the TPU returns an escape sequence followed by one or more bytes of data. Whether the request was initiated locally from CP/M or remotely from a host system, the status bytes are terminated by a 'carriage return' character. A 'line feed' character will also return if the TPU is configured for 'AUTO LINE FEED' mode.

The actual status information is contained in the lower four bits of each status byte. The upper four bits are set by the TPU to assure the byte represents a printable ASCII character. Each byte can be interpreted as one of the character values listed in Table 6-1. Remember, only the lower four bits are significant.

Table 6-1. TPU Status Byte Values

ASCII CHARACTER	BINARY STATUS
0	0011 0000
1	0011 0001
2	0011 0010
3	0011 0011
4	0011 0100
5	0011 0101
6	0011 0110
7	0011 0111
8	0011 1000
9	0011 1001
:	0011 1010
;	0011 1011
<	0011 1100
=	0011 1101
>	0011 1110
?	0011 1111

Terminal Status

Terminal status is made up of 14 status bytes, numbered 0 through 13. Each byte contains specific information about some aspect of the HP 125 terminal configuration. Memory size and strapping options are just two examples of some of the information available. The status is available in two sets of 7 bytes each. These two blocks are called primary and secondary status, respectively.

When you perform a terminal self-test from the keyboard, you will notice the line of 14 alphanumeric characters which appear across the last line of test information. These 14 bytes represent the TPU status, and reflect the same data which would be returned when status is requested via escape sequence.

Under program control you can request either primary or secondary TPU status separately. This is true from both remote systems and from CP/M running locally. In fact, the only difference between a local request and a remote request is that, a datacomm generated request may require a trigger if other than a Type 1 handshake is selected. See the HP 125 Owner's Manual for details on the types of data transfer available.

Primary Terminal Status

The first block of status, bytes 0 through 6 inclusive, is requested by sending the following escape sequence:

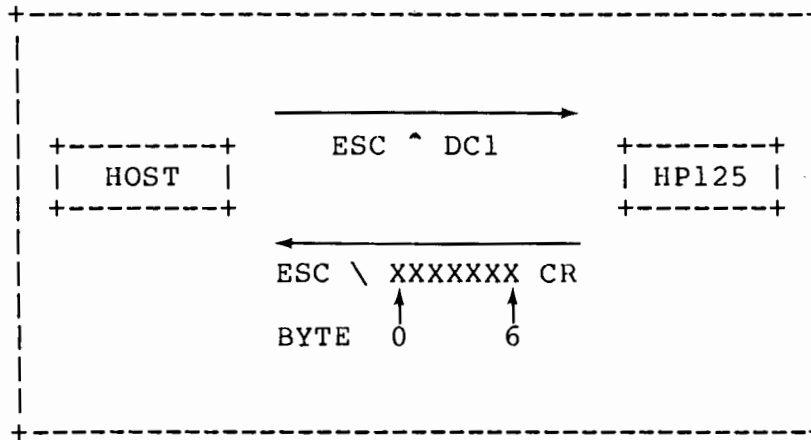
```
ESC ^
```

The second character is an 'up arrow', ASCII 94 decimal. No terminator (such as a 'carriage return') is required locally, although a 'trigger' may be required depending on the status of the G and H strap settings.

The TPU will respond with an 'ESC \' sequence, followed by seven bytes of primary status information. A 'carriage return' will be terminate the block, although a 'line feed' will also occur if the AUTO LINE FEED mode is enabled.

An example of a primary status request is shown in Table 6-2 below. The transfer shown reflects a remotely initiated request using a 'Type 2' trigger typical of an HP 3000. Remember that a locally initiated request would not have the DC1 trigger character to start the transfer.

Table 6-2. Primary Terminal Status Request



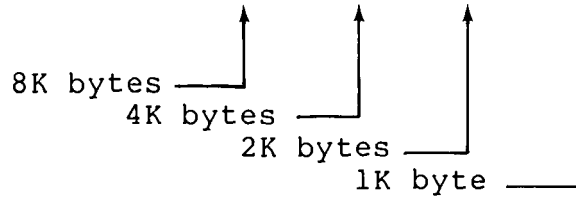
The meaning of each byte of primary status is shown in Table 6-3 which follows.

Table 6-3a. Primary Status Decoding

BYTE 0

DISPLAY MEMORY SIZE

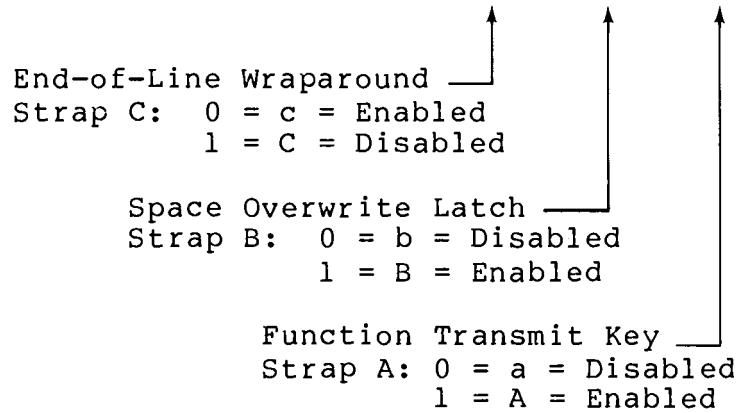
 | 0 | 0 | 1 | 1 | 0/1 | 0/1 | 0/1 | 0/1 |



BYTE 1

CONFIGURATION STRAPS A-C

 | 0 | 0 | 1 | 1 | 0 | 0/1 | 0/1 | 1/0 |



BYTE 2

CONFIGURATION STRAPS G-H

 | 0 | 0 | 1 | 1 | 0/1 | 0/1 | 0 | 0 |

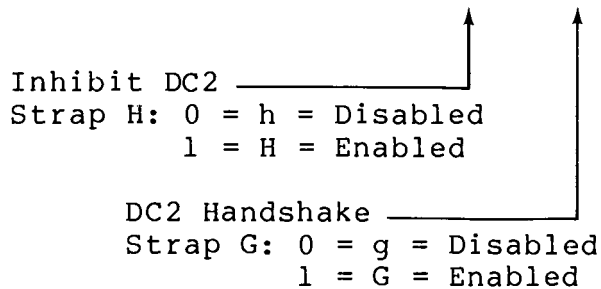


Table 6-3b. Primary Status Decoding

BYTE 3

LATCHING KEYS

 | 0 | 0 | 1 | 1 | 1 | 0/1 | 0 | 0/1 |

Auto Line Feed Mode _____
 0 = Disabled
 1 = Enabled

Caps Lock Key _____
 0 = Disabled (OFF)
 1 = Enabled (ON)

BYTE 4

TRANSFER PENDING FLAGS

 | 0 | 0 | 1 | 1 | 0/1 | 0/1 | 0/1 | 0/1 |

Secondary Status Request _____
 0 = Not Pending
 1 = Pending

ENTER Key _____
 0 = Not Pending
 1 = Pending

Function Key _____
 0 = Not Pending
 1 = Pending

Cursor Sense Request _____
 0 = Not Pending
 1 = Pending

Table 6-3c. Primary Status Decoding

BYTE 5

ERROR FLAGS

 | 0 | 0 | 1 | 1 | 0/1 | 0 | 0/1 | 1/0 |

Device Error
 0 = No Error
 1 = Error

Self-Test
 1 = No Error
 0 = Error

Data Comm Error
 0 = No Error
 1 = Error

BYTE 6

DEVICE TRANSFER PENDING FLAGS

 | 0 | 0 | 1 | 1 | 0 | 0 | 0/1 | 0/1 |

Device Operation
 0 = No Status Pending
 1 = Status Pending

Device Status
 0 = No Status Pending
 1 = Status Pending

Secondary Terminal Status

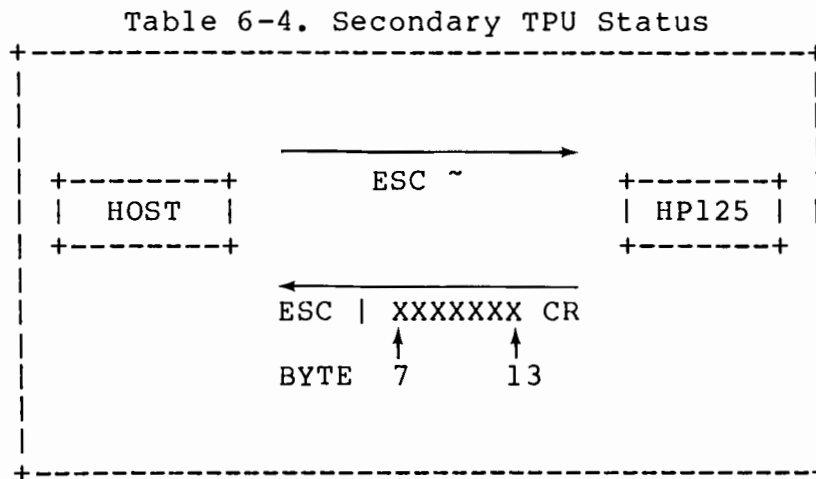
The second block of TPU status, bytes 7 through 13, is requested by using the following escape sequence:

```
ESC ~
```

The second character is a 'tilde', ASCII code 126. Again, any trigger for a remote request will depend on the status of the G and H straps in the TPU.

The TPU responds with an 'ESC |' and seven status bytes. Once again, the block is terminated by a 'carriage return'. If AUTO LINE FEED is enabled, a 'line feed' character will also be returned.

Table 6-4 illustrates how a request for secondary TPU status would be initiated from a program running locally under CP/M.



The meaning of each byte of secondary status is given in Table 6-5a and Table 6-5b.

Table 6-5a. Secondary Status Decoding

BYTE 7 UNUSED

```
-----
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
-----
```

BYTE 8 TERMINAL CONFIGURATION

```
-----
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |
-----
```

This byte is fixed. The various bits indicate that the HP 125 is a programmable terminal, that it will respond to a terminal identify request, that I/O drivers for additional devices are present in firmware, and that the TPU does not support APL.

BYTE 9 CONFIGURATION STRAP L

```
-----
| 0 | 0 | 1 | 1 | 0 | 0/1 | 0 | 0 |
-----
```


Inhibit Self-Test Latch 
 Strap L: 0 = 1 = Allow Test
 1 = L = Inhibit Test

Table 6-5b. Secondary Status Decoding

BYTE 10

UNUSED

| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

BYTE 11

UNUSED

| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

BYTE 12

UNUSED

| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

BYTE 13

UNUSED

| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

Printer Status

The TPU can report the status of any printer devices which are connected to the HP 125. This status, like terminal status, is requested via an escape sequence.

Typically, such a request might be made to verify the existence of a device, such as the built-in printer, or to confirm completion of an output operation. The device status bytes are decoded in much the same manner as the terminal status checks. Additionally, the trigger mechanism and return terminators are similar to those used with terminal status.

Device status is requested by sending the following escape sequence to the TPU:

```
ESC & p <code> ^
```

The final character of the sequence is an 'up arrow', ASCII code 94 decimal. The possible values for <code> are:

```
code    device
 4    -> Serial device on Port #2
 5    -> HP-IB device at address 1
 6    -> Internal thermal printer (TPM)
```



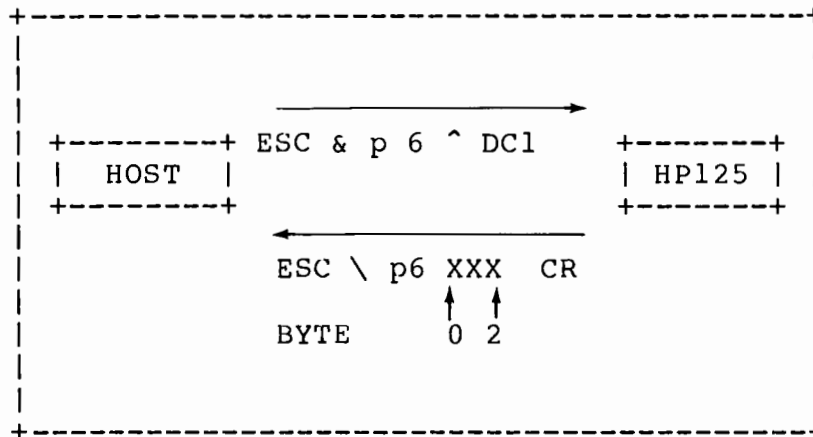
The TPU will return the four character sequence:

```
ESC \ p <code>
```

followed by three status bytes. A 'carriage return' will terminate the sequence, as was the case with TPU status.

An example of a printer status request is given in Table 6-6.

Table 6-6. Printer Device Status



An explanation of the three bytes of status is provided in Table 6-7 and in the text which follows. In general, each different type of printer uses each byte in a slightly unique manner. The device specific decodings follow.

Table 6-7. Printer Status Decoding

BYTE 0

```

-----
| 0 | 0 | 1 | 1 | 0 | 0 | 0/1 | 0/1 |
-----
  
```

Printer Error (0=No, 1=Yes)

Last Print Command
 0 = successful
 1 = failed

BYTE 1

```

-----
| 0 | 0 | 1 | 1 | 1/0 | 0 | 0 | 1/0 |
-----
  
```

Last Command Executed
 0 = No (Interrupted)
 1 = Yes (Completed)

Printer Busy (0=No, 1=Yes)

BYTE 2

```

-----
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1/0 |
-----
  
```

Printer Present (0 = No, 1 = Yes)

HP-IB Printer Status

A status request for the HP-IB printer returns a fixed set of data bytes when queried for status with a code of '5'. Because of the manner in which the CPU implements the HP-IB protocol, the actual status of the printer is not available to the TPU.

The status you will receive will be the ASCII characters

```
ESC \ p 5 0 8 1
```

Looking back at Table 10-7, you can see that this value indicates:

- No print error, last command successful (Byte 0)
- Last command performed, device not busy (Byte 1)
- Printer connected (Byte 2)

HP-IB Device address 1 should be reserved for HP supplied printer devices.

Internal Thermal Printer

The status returned for the internal printer does not use all of the possible values because the TPU handles the status independently. Specifically:

Byte 0 uses the low two bits.

Bit 0, 'Last Print Command', is set if a the last print command failed. It is cleared if command execution was successful.

Bit 1, 'Print Error', is set if the printer is out of paper or if the printer bail is open. It is cleared when paper is present and the bail is closed.

Byte 1 uses only bit 3, 'Last Command Executed'. The bit is set if the last print or transfer command was successful. The bit is cleared if the command was interrupted.

Byte 2 uses only bit 0, 'Printer Present'. If the unit includes a TPM, the bit is set. If no TPM is present, the bit is cleared.

Serial Printer Status

The device on Port #2 is considered the serial printer device by the TPU. The serial device status is interpreted as follows:

Byte 0 uses bit 0 only. 'Last Print Command' is set if the last print command failed, and cleared if the operation was successful.

Byte 1 uses bits 0 and 3.

Bit 0, 'Printer Busy', can be set by any of three conditions:

- SRRXmit is enabled and the reverse channel is busy;
- Transmit Handshake is enabled and 'Clear to Send' (CTS) is low. This means the 'CB' pin shows the device is 'busy'. A handshake menu of 'eTx' will allow this condition to occur; or
- Xon/Xoff handshaking is enabled and Xoff has been received from the device. A handshake menu of 'etX' will lead to this condition upon Xoff receipt.

Bit 3, 'Last Command Executed', is set if the last command completed normally, and cleared if the last command was interrupted.

Byte 2 uses bit 0, 'Printer Present', which is set when 'Clear to Send' is low. This indicates that the 'CB' line is 'busy'. The bit is cleared when 'Clear to Send' is high, indicating no device is present.

Terminal Identifier

The next sequence to be covered here is the Terminal Identifier request. A program executing on a host system can verify that the terminal device in use is an HP 125. This can be done from the LOCAL OP SYS mode as well; the TPU will always respond as an HP 125!

The following escape sequence initiates the terminal identifier sequence for the HP 125:

```
ESC * s ^
```

The last character is an 'up arrow' or exponentiation character, ASCII 94. It is not the same as the cursor control character! As with other sequences discussed above, some 'trigger' may be required depending on the state of the G and H datacomm straps.

Upon receipt of the sequence, the terminal will respond:

4 5 5 0 0

By using this sequence, a CP/M program can verify the HP 125 is 'mapped' as the console device. Much more is said about this device mapping in Chapter 16.



Cursor Sensing

There is another escape sequence which returns status information. This is the 'cursor sense' sequence, which allows you to determine the absolute row and column address of the cursor.

The current position of the cursor can be detected relative to the top of the screen (Screen Relative) or to the top of display memory (Screen Absolute).

To detect the position of the cursor relative to the top of the screen, Screen Relative sensing, is:

ESC `

The second character is a left quote, ASCII 96 decimal, and is the upper right key in the keyboard group of your HP 125 keyboard. Don't confuse it with the apostrophe, ASCII 39 decimal.

The sequence returned to the requesting program is the escape sequence string which, if sent, would position the cursor at the current location. See Table 6-9 for an example of using this request.

To request the absolute location of the cursor, relative to the top of display memory, send:

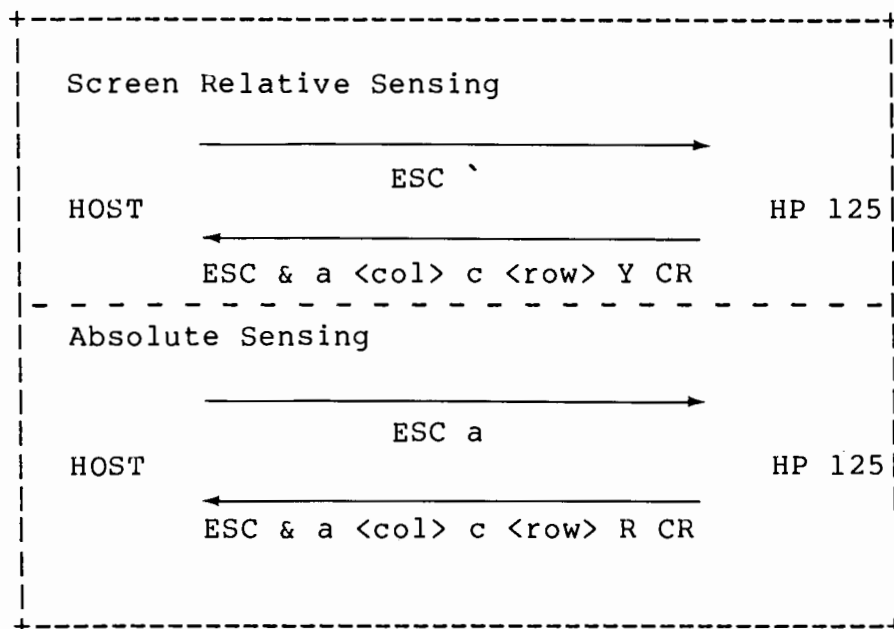
ESC a

This sequence will return the escape sequence string which would place the cursor (absolute) at the current position.

Both of these sequences follow the conventions presented above in regard to status information. Specifically, no trigger is necessary if initiated from the CPU, while a DC1 is necessary to trigger a remote request. Also, a carriage return character is appended to the returned address whether the request is initiated locally from CP/M or remotely.

Examples of both of these requests are provided in Table 6-8.

Table 6-8. Cursor Sending Request



Summary

This concludes our look at sequences which are used to determine TPU and device status. In the next chapters, we will begin our look at the CP/M Operating System and at programming in CP/M.

INTRODUCTION TO CP/M

The HP 125 as Computer

In previous chapters, you saw how the HP 125 functions as an intelligent terminal in both REMOTE and LOCAL OP SYS modes. There is another portion of your HP 125: the part that gives you all of the power of a stand-alone computer system. This other part of your HP 125 utilizes an operating system known as 'Control Program for Microcomputers', or CP/M. For this reason, we will refer to this 'computer' side of your system as the CP/M Processing Unit, or CPU.

The CP/M Operating system is loaded into the CPU memory when you first 'boot' your system from disc. It generally remains in memory, ready to work, until you turn off your computer. In the next few chapters, you will learn much more about the inner workings of CP/M. We will start our look at CP/M with a general overview of the features, then look at the more technical aspects of its operation.

Introduction to CP/M

The operating system on the HP 125 is Hewlett Packard's implementation of the popular microcomputer operating system, CP/M. Since the HP 125 has a superset of standard CP/M, any standard CP/M program should be usable on your new system. Note that programs designed specifically to use the HP 125's advanced features may not function properly on other CP/M computers.

CP/M is a small yet efficient memory resident program. It provides all the features necessary to implement a minimum useful operating system. However, its structure also allows access to expanded features within its standard configuration, so that the system can grow with your application.

CP/M contains six integral commands, discussed later in this chapter. It also allows user programs to load into memory, to access sequential and random access files, and to address a variety of physical and logical devices under program control. CP/M provides the common structure, fundamental I/O calls, and uniform memory organization. The user program may choose to utilize these features, or may implement all or part of the CP/M features independently.

Keep in mind, as you continue through this chapter, that many 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 HP 125. However, if you elect to program in assembly language, these CP/M features will become an integral part of your application.

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 start your HP 125. This process, known as 'bootstrap loading', involves some actions on your part. Specifically:

- a. Turn on your disc drive and insert a HP 125 operating system disc into drive 'A:'.
- b. Turn on the HP 125 mainframe. Allow approximately 15 seconds for warm-up and self-test.
- c. If you last used your HP 125 in 'LOCAL OP SYS' mode, your system will 'boot' automatically. Proceed to step 'e' below. If not, or if you have a start-up error or warning, configure your system for 'LOCAL OP SYS' and proceed to the next step.
- d. Press 'LOAD OP SYS': The 'A:' disc access light should come on, indicating disc activity. After a short period of time, the HP 125 should print a few lines of start-up information, and CP/M will attempt to execute a file named 'WELCOME.COM'. If such a file is present, that program will load and run.

If no such file is found, the CP/M prompt 'A>' will appear, and CP/M is ready to go!

- e. Your system is now 'up'.

If you have 'booted' a HP-supplied Applications Disc, you will see an 'Applications Menu' rather than the CP/M prompt. This is an example of the 'WELCOME' file described above. Refer to the various application manuals for the specific software products supplied by HP for more information on using the menu program.

CP/M Commands

Once you have the CP/M prompt, your HP 125 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 six 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 in this chapter.

Built-In Commands

The built-in CP/M commands are described in depth in the HP 125 Owner's Manual as well as in most of the commercially available CP/M books. A bibliography of several of the more popular reference manuals is contained in Appendix C. A brief description of each command is provided for your benefit in Table 7-1.

Table 7-1. CP/M Integral 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 and/or list device
USER	Log into a particular user area of a disc.

Transient Commands

As mentioned above, CP/M also provides several additional commands which are disc resident. These are called transient commands because they are called into memory only when a function is actually requested by the user. Built-in commands always reside in memory, and utilize memory space whether used or not. As you will discover later in this chapter, 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. Also remember that not every transient utility may exist on each disc!

The standard transient commands which extend the features of CP/M are listed here. Many of these are located on your HP 125 Operating System disc, while others are included only with the System Programming Package.

Table 7-2 indicates the names of each of these transient commands, and indicates whether the command is documented in the HP 125 Owner's Manual or in this manual. Generally, transients related to program development are documented here, while operator relevant transients are covered in depth in the Owner's Manual.

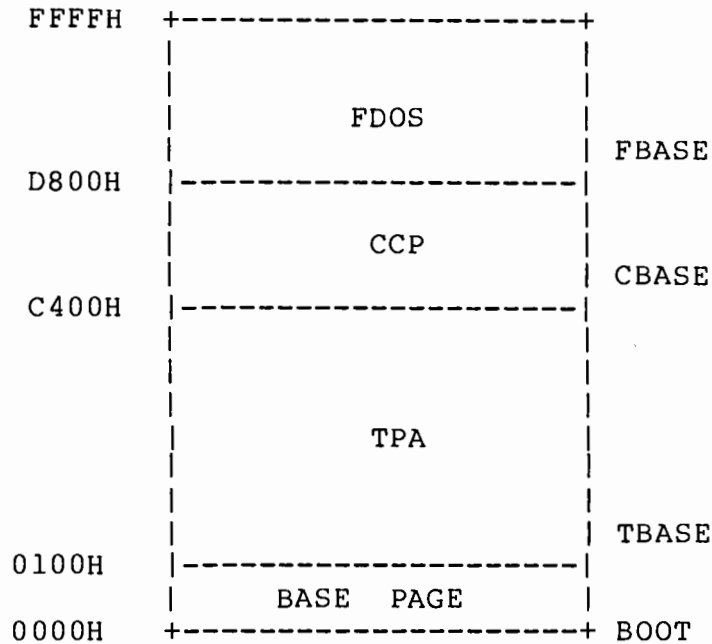
Table 7-2. CP/M Transient Commands

Name	Description
ASM	The CP/M 8080 Assembler Program for writing assembly language routines. See Chapter 15.
COPY	Permits the copying of files and/or the CP/M Operating System to another disc. See Chapter 13.
DDT	Dynamic Debugging Tool, an aid in verifying assembly language program execution. See Chapter 16.
DUMP	Produce a hex listing of a command file. See Chapter 13.
ED	The CP/M character oriented text editor program. See Chapter 14.
FORMAT	Performs a complete surface test and initializes disc media. See Chapter 13.
LOAD	The CP/M loader. This makes ASM output files executable. See Chapter 13.
PIP	The Peripheral Interchange Program which permits file and device transfers. See Chapter 13 and the Owner's Manual.
STAT	CP/M device status program. See the Owner's Manual for operator information, or Chapter 13 for device assignments.
SUBMIT	This CP/M transient permits batch job execution with no user interaction. See Chapter 13.
XSUB	This program extends the capabilities of SUBMIT to allow batch console input. See Chapter 13.

Memory Organization

The CP/M Operating System resides in approximately 12K bytes of high memory. This means that, for most user applications, up to 52K 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 7-3. CP/M Memory Organization



There are actually four logical areas of memory in CP/M. Table 9-3 indicates how memory is allocated in a typical HP 125 system. 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.

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. Part of the BIOS is actually reserved memory for use when ROM routines are executed by the CPU. The BIOS allows CP/M to communicate with the outside world.

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

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



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 six built-in commands discussed earlier. This is done by using the system function calls available to any program. The CCP simply accepts input and takes 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 re-loaded into memory once the transient has completed. This is done automatically during a 'warm boot'.

TPA

The Transient Program Area is where system and user programs are loaded for execution. This is roughly analogous to 'user memory' on other computer systems. Note that 'transient commands' are actually executed in the TPA while built-in commands reside in the CCP. This is why they are called 'transient'.

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

Base Page

Base page is the special name for the lowest 'page' of memory. On the HP 125, each page contains 256 bytes of memory, so the base page occupies memory from 0000H through 00FFH. Because base page contains important information for CP/M (and for TPA programs), all CP/M programs should start at address 0100H. This preserves the integrity of the base page, and allows re-entry to the CCP when a transient program has completed.

All of these sections of CP/M will be discussed in even greater depth in the following chapters, but this should serve as an introduction to the operating system.

The File System

In addition to the memory resident features described above, CP/M also includes a logical file structure on each disc device on your HP 125. This file system, fully accessible through system function calls, allows you to use random and sequential access data files and to store and retrieve program files. Again, it is this consistency of organization that allows the hardware independence of your programs.

File Names

Each disc file must have a name unique to the disc containing the file. A fully qualified file name is made up of three parts: a disc drive identifier, a file name, and a file type.

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 'H' inclusive, corresponds to the physical disc drive. Refer to the HP 125 Owner's Manual for further information.

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

File names and file types are generally assigned in an arbitrary manner. However, several types have special meanings to CP/M and its commands. These are presented in Table 7-4.

Table 7-4. File Types

ASM	ASM Source Code	LIN	GRAPH/125 Line File
BAK	Back-Up ED File	PIE	GRAPH/125 Piechart
BAR	GRAPH/125 Bar File	PRF	Visicalc Print File
BAS	BASIC Source File	PRN	Printer Listing File
COM	Transient Command	SLD	GRAPH/125 Slide File
DAT	ASCII Data File	TXT	ASCII Data File
DIF	Visicalc DIF File	VC	Visicalc Worksheet
HEX	ASM Output File	\$\$\$	Temporary File

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. More will be said about this in Chapter 8.

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. You will see much more on this in Chapter 8.

File Access

Data in files can be organized in either random or sequential fashion. There is no rigid distinction between the two in CP/M since both must have 128 byte logical records. However, random access permits directed access to data, while sequential files must be accessed serially.

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

ASCII files are considered to be a sequence of characters. Most applications consider the 'carriage-return line-feed' character as a logical record terminator, permitting more than one logical record per 128 byte disc record.

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.

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 'RETURN'.

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. See Chapter 8 for more information about base page and FCB's. All characters are shifted to upper case.

The CCP searches your input line for one of the built-in commands as the first non-blank characters. If one of the commands listed in Table 7-1 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 type 'COM' (see '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.

Transient Programs

By now you might have guessed that any compiled program you might write becomes a 'transient command' to CP/M. Your program is named with a filetype 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. The symbolic names used in the next several paragraphs refer to the memory map shown in Table 7-3.

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 're-boot' CP/M, and the user will have to do so on his own.

Summary

This concludes our first look at CP/M. In the next chapters, you will learn more about the internal functioning of CP/M, and how to use the operating system features to perform your application tasks.



CP/M INTERNAL ORGANIZATION

Introduction

In the previous chapter, you received a brief introduction to the features available through CP/M. In this chapter, you will learn more of the internal structure of the Operating System and its integrated file system. Once again, remember that most application programs usually don't require use of these advanced features. If system internals make you uncomfortable, rest assured that you probably won't need this knowledge. However, if you are curious about what makes your HP 125 tick, this chapter is for you.

File System Overview

We will begin this section by looking at the basic organization of data on the disc. Understanding this is essential in fully comprehending CP/M file access and system function calls (SFC).

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 mini-disc, and 4K bytes on the 8" 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 mini-disc, or 4 groups on the 1.2M byte 8" 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 mini-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:

$$\begin{array}{rcccl}
 16\text{K bytes} & & 32 \text{ extents} & & 512\text{K bytes} \\
 \text{-----} & \times & \text{-----} & = & \text{-----} \\
 \text{extent} & & \text{file} & & \text{file}
 \end{array}$$

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" 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:

$$\begin{array}{rcccl}
 32\text{K bytes} & & 64 \text{ entries} & & 2048\text{K bytes} \\
 \text{-----} & \times & \text{-----} & = & \text{-----} \\
 \text{entry} & & \text{file} & & \text{file}
 \end{array}$$

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.

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.

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 8-1, 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 above. 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 8-1. 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 Type: 3 bytes in length
10	0A	t2	
11	0B	t3	
12	0C	ex	Extent Number
13	0D	s1	Reserved for System Use (2 bytes)
14	0E	s2	
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 0 : dr

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

- 0 -> File on default drive
- 1 -> File on drive 'A:'
- 2 -> File on drive 'B:'
- · ·
- 8 -> File on drive 'H:'

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 8 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 3 bytes contain the optional file type. 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 0 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 80 hex.

BYTES 16 -> 31 : dl -> dl6

This group of bytes is reserved for system usage. The values which are stored here reflect the disc group address of allocated groups. As the file grows, additional groups are added to the list. On the mini-disc, this section is organized into 16 single byte entries; on the 9895A, 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 33 -> 35 : r0 -> r2 (FCB Only)

These three bytes are only used during random access disc I/O. Bytes r0 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 r0 contains the low-order 8 bits, while byte r1 contains the high order 8 bits or the actual record number. In the default FCB buffer, r0 is at address 007DH.

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 of 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 type, with the remainder of the fields set to zero.



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.

Default FCB's and the CCP

As you know, a user may specify one or two filenames in many of the CCP commands as well as in many of the transient commands. The CCP sets up one default FCB at address 005CH 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 d0 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 005DH through 006DH 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 buffer area at address 0080H 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, 0080H, contains the byte count of the line. Starting at byte 0081H you will find the parameters typed after the command.

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 consider a program which functions like the 'REN' command, except that it allows files to be 'renamed' from one disc to another. It is invoked like PIP, with the new name and the old name specified on the command line. The program name is 'DREN.COM', and the command format is:

```
A>dren b:new.txt a:old.txt
```

When the [RETURN] key is pressed, the CCP sets up a default FCB at address 005CH. The CCP will parse the characters which follow the program name, using a blank character to delimit the first parameter from the second parameter.

The CCP will treat each parameter, if provided, as a filename. The first parameter will be loaded into the first 16 bytes of the default FCB at 005CH. The second parameter, also treated as a filename, will be stored in the second 16 bytes of the FCB, starting at address 006CH.

Since the CCP considers any parameters to be filenames, a drive identifier, file name, and file type will be parsed. If no colon is provided, the default drive code is assumed. If no decimal point is provided, only the first 8 characters are considered as part of the file name.

In addition to the filename parsing, the CCP stores the 32 bytes which follow the program name into the DMA buffer at address 0080H. These bytes, called the 'tail' of the command line, are illustrated in Table 8-3.

In this 'tail' buffer, the CCP has not shifted characters to upper case as it has in building the FCB from the command line.

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, DREN will access the FCB at address 005CH to establish the FCBs for each file. One FCB can remain at that address, while the second must be established in the TPA. DREN can also access the command line stored at address 0080H to determine whether any additional parameters were entered. See Appendix F for the various commands and parameters available through DREN.

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

Table 8-2. Example FCB Set-Up: DREN

Hex	Byte	Chars	Description
00	dr	02H	Destination Drive Code (B:)
01	f1	N	Destination File Name: 'NEW'
02	f2	E	
03	f3	W	
04	f4		
05	f5		
06	f6		
07	f7		
08	f8		
09	t1	T	Destination File Type
0A	t2	X	
0B	t3	T	
0C	ex	00H	Destination Extent Number
0D	s1	00H	Reserved for System Use
0E	s2	00H	
0F	rc	00H	Destination Extent Rec Cnt
10	dr	01H	Source Drive Code (A:)
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 Type
1A	t2	X	
1B	t3	T	
1C	ex	10H	Source Extent Number
1D	s1	10H	Reserved for System Use
1E	s2	10H	
1F	rc	10H	Source Extent Record Count

Table 8-3. Command Line Buffer Format

Addr	Cnts	Comments
80H	13H	Length of buffer
81H	20H	Space after 'dren'
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	a	Source File Name
8DH	:	
8EH	o	
8FH	l	
90H	d	
91H	.	
92H	t	
93H	x	
93H	t	
94H	20H	Trailing Space(s)

Base Page

As mentioned earlier, base page is the name given to the lowest addressed 256 memory locations. These locations are typically used so that transient programs can access BDOS in a consistent manner. This is true regardless of the revision of CP/M you may have.

You've already seen one or two reserved locations in the discussion of file control blocks (FCB). Here you will see the organization of the entire base page, and which locations will be of use to your applications.

Base page includes all memory locations from address 0000H through 00FFH inclusive, for a total of 100H bytes. Table 8-4 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 0000H contains a 3 byte Z80 '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, see the section on 'BIOS Entry Points' in Chapter 12.

IOBYTE is stored at address 0003H and determines the physical devices CP/M maps onto its logical devices. See Chapter 11 for more information on IOBYTE.

DRIVE ID at address 0004H 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 the next chapter.

Table 8-4. 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	Interrupt Jump 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, and is discussed earlier in this chapter.

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.

Summary

In this chapter, you've seen some of the detailed functioning of CP/M. Much of it, to this point, is theoretical since you have not seen the system function calls through which all CP/M functions are implemented. In the next chapters, you will see the standard CP/M system function calls and the HP enhancements to CP/M. You will learn how to use these calls to take advantage of the features of CP/M in your applications.

SYSTEM FUNCTION CALLS

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 HP 125 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 Z80 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 9-1.

The number listed in Table 9-1 under the 'CALL' column is known as the 'system function number'. This number corresponds to the decimal value which 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 several extended calls which are useful in adapting the unique features of the HP 125 to CP/M. These calls are discussed in the Chapter 15.

Table 9-1. System Function Call Summary

CALL	OPERATION	CALL	OPERATION
0	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 I/O	26	SET DMA ADDRESS
7	GET IOBYTE	27	GET ADDR (ALLOC)
8	SET IOBYTE	28	READ-ONLY DISC
9	PRINT STRING	29	GET R/O VECTOR
10	READ CONSOLE BUFFER	30	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
20	READ SEQUENTIAL	40	WRITE W/ ZERO FILL

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 0005H. 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 contain the same value as register H.

Here is a sample program which illustrates how function calls are implemented. This short program reads one console character at a time. The function call used, SFC 1, will automatically echo the character to the console. It continues to accept input until an asterisk is encountered, whereupon it returns control to the CCP.

Table 9-2. Sample System Function Call

BDOS	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 re-load the CCP and re-initialize 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 0

FUNCTION 0: System Reset

Entry Parameters:

Register C: 00H

This 'System Reset' function is equivalent to a 'warm boot' in that program execution ends and the 'A:' 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. See the HP 125 Owner's Manual for a detailed description of these control features.

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 your program wishes 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, special HP 125 features such as device mapping and keycode mode often require a full eight bits. See Chapter 12 for a discussion of how this can be accomplished.

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      ;OUTPUT CHARACTER      |
|         . . .          ;CHARACTER NOW IN A     |
+-----+

```

Function 2

FUNCTION 2: CONSOLE OUTPUT

Entry Parameters:

Register C: 02H
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   0005H           ;MAIN CP/M ENTRY POINT |
|         . . .                |
|         MVI   E,'@'         ;MOVE ASCII @ INTO E   |
|         MVI   C,02H         ;MOVE SFC 2 INTO C     |
|         CALL  BDOS          ;OUTPUT CHARACTER      |
|         . . .                |
|         @ IS ON CONSOLE     |
+-----+
```

FUNCTION 3: READER INPUT**Entry Parameters:**

Register C: 03H

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. Standard CP/M does not have a call which checks the status of the reader, as SFC 11 does for the console. However, HP has provided a 'Reader Status' call as HP SFC 112. Look at that SFC description in the next chapter for more information.

Chapter 16, 'Device Assignments', contains a discussion of IOBYTE and other HP 125 device usage. Refer to that chapter to understand the CP/M device terminology, and how to assign the reader to a physical device.

EXAMPLE: Reader input

```

+-----+
| BDOS   EQU   0005H           ;MAIN CP/M ENTRY POINT
|       . . .
|       MVI   C,03H           ;MOVE SFC 3 INTO C
|       CALL  BDOS           ;READ CHARACTER
|       . . .
|                               ;CHARACTER NOW IN A
+-----+

```

Function 4

FUNCTION 4: PUNCH OUTPUT

Entry Parameters:

Register C: 04H
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.

Chapter 16, 'Device Assignments', contains a discussion of IOBYTE and other HP 125 device usage. Refer to that chapter to understand the CP/M device terminology, and how to assign the punch to a physical device.

EXAMPLE: Output @ to punch device

```
+-----+
|  BDOS   EQU   0005H           ;MAIN CP/M ENTRY POINT  |
|          . . .                |
|          MVI   E,'@'          ;MOVE ASCII @ INTO E   |
|          MVI   C,04H          ;MOVE SFC 4 INTO C   |
|          CALL  BDOS           ;OUTPUT CHAR TO PUNCH    |
|          . . .                |
|          . . .                |
|          . . .                |
+-----+
```

FUNCTION 5: LIST OUTPUT

Entry Parameters:

Register C: 05H
Register E: ASCII Character

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

Chapter 16, 'Device Assignments', contains a discussion of IOBYTE and other HP 125 device usage. Refer to that chapter to understand the CP/M device terminology, and how to assign the list device to a specific printer.

EXAMPLE: Output @ to IOBYTE defined list device

```
+-----+
| BDOS   EQU   0005H           ;MAIN CP/M ENTRY POINT
|       . . .
|       MVI   E, '@'         ;MOVE ASCII @ INTO E
|       MVI   C, 05H         ;MOVE SFC 5 INTO C
|       CALL  BDOS          ;PRINT CHARACTER TO LIST
|       . . .               ;CHARACTER IS PRINTED
+-----+
```

Function 6

FUNCTION 6: DIRECT CONSOLE I/O

Entry Parameters:

Register C: 06H
Register E: 0FFH (input) or
char (output)

Returned Value:

Register A: char or status
(no value)

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 0FFH. If a character is available for input, it is returned in register A. If no byte is ready, register A contains 00H upon return.

EXAMPLE: Read and echo a character on the console

```
+-----+
| BDOS   EQU   0005H           ;MAIN CP/M ENTRY POINT |
|       . . .                 |
| INPT   MVI   E,0FFH          ;MOVE INPUT CODE INTO E  |
|       MVI   C,06H           ;MOVE SFC 6 INTO C          |
|       CALL  BDOS            ;CHECK CONSOLE FOR INPUT       |
|       CPI   00H             ;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 125. Chapter 16, 'Device Assignments', contains a discussion of IOBYTE and other HP 125 device usage. Refer to that chapter to understand the CP/M device terminology.

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

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 125. Chapter 16, 'Device Assignments', contains a discussion of IOBYTE and other HP 125 device usage. Refer to that chapter to understand the CP/M device terminology.

EXAMPLE: Load HP-IB printer code in IOBYTE

```
+-----+
| BDOS   EQU   0005H           ;MAIN CP/M ENTRY POINT |
|         . . .               |
|         MVI   E,0C0H         ;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
|  MMSG   DB    'HELLO THERE$'
|          . . .          ;NOTE MMSG ENDS WITH $
|          LXI   D,MMSG    ;ADDR OF MMSG INTO DE
|          MVI   C,09H     ;MOVE SFC 9 INTO C
|          CALL  BDOS      ;PRINT STRING 'MMSG'
|          . . .          ;MESSAGE NOW AT CONSOLE
+-----+

```

Function 10

FUNCTION 10: READ CONSOLE BUFFER

Entry Parameters:

Register C: 0AH
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.

Before using this function, an application must first establish an input buffer. The format of the buffer is illustrated below. The address of this buffer should be loaded into the DE register pair prior to calling the BDOS.

```
-----  
|mx|nc|c1|c2|c3|c4|c5|c6|c7| . . . |??|  
-----  
DE: +0 +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 which 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

FUNCTION 11: GET CONSOLE STATUS

Entry Parameters:

Register C: 0BH

Returned Value:

Register A: Console Status

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, 01H is returned in register A. If no character is present, 00H 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   0005H           ;MAIN CP/M ENTRY POINT
|          . . .
|  STAT   MVI   C,0BH           ;MOVE SFC 11 INTO C
|          CALL  BDOS           ;GET STATUS
|          CPI   01H           ;IS CHAR READY?
|          JZ    READY          ;YES - GOTO READY
|  NOCHAR JMP   STAT           ;NO - START OVER
|          . . .
|          ;
+-----+
```

FUNCTION 12: RETURN VERSION NUMBER**Entry Parameters:**

Register C: 0CH

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 which 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 00H. L will contain a hex value in the range of 20H 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 which verify not only revision number, but also verify that the CP/M is running on an HP 125. See the discussion in Chapter 15.

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

```

+-----+
| BDOS   EQU   0005H           ;MAIN CP/M ENTRY POINT
|       . . .
|       MVI   C,0CH           ;MOVE SFC 12 INTO C
|       CALL  BDOS           ;READ REVISION NUMBER
|       MOV   A,L             ;MOVE REV INTO A
|       CPI   22H            ;IS REV = 2.2 OR > 2.2?
|       JNZ   REVOK          ;YES - GOTO REVOK
| EVERR   . . .               ;NO - ERROR
+-----+

```

Function 13

FUNCTION 13: RESET DISK SYSTEM

Entry Parameters:

Register C: 0DH

This function performs a 'disc system reset'. This means that the file system is restored to a 'read/write' state on all discs which are on-line. Drive 'A:' is selected, and the default disc buffer is set to 0080H.

Use of this function permits the application to programmatically request a change of discs without any need to 're-boot' to gain write access to the new disc. See SFC 28 and 29.

EXAMPLE: Make all discs read/write

```
+-----+
| BDOS   EQU   0005H           ;MAIN CP/M ENTRY POINT |
|         . . .               |
|         MVI   C,0DH         ;MOVE SFC 13 INTO C      |
|         CALL  BDOS         ;RESET DISC SYSTEM        |
|         . . .               |
|         ;CONTINUE          |
+-----+
```

FUNCTION 14: SELECT DISK

Entry Parameters:

Register C: 0EH
 Register E: Selected Disk



This function permits the selection of any disc drive as the default disc. This will set all subsequent disc operations which 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 00H indicates drive 'A:'; a value of 07H represents drive 'H:'.

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' 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 had been executing will need to be run again.

EXAMPLE: Select disc 'C:'

```

+-----+
|  BDOS  EQU  0005H      ;MAIN CP/M ENTRY POINT |
|          . . .        |
|          MVI  E,02H    ;CODE FOR DRIVE C: INTO E |
|          MVI  C,0EH    ;MOVE SFC 14 INTO C      |
|          CALL BDOS    ;SELECT DISC C:          |
|          . . .        |
+-----+
  
```


Function 15

FUNCTION 15: OPEN FILE

Entry Parameters:

Register C: 0FH
Registers DE: FCB Address

Returned Value:

Register A: Directory 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 0 is the disc specifier, and may contain a value in the range from 0 through 8. A value of 0 means the file is on the currently selected mass storage device; values from 1 through 8 indicate disc drives 'A:' through 'H:' respectively.

The file name and file type are loaded into byte positions 1 through 11. Byte positions 12 through 35 should be 00H.

You normally will search the directory for a specific file. 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 start of file after a successful open call.

This wild card feature is implemented by placing an ASCII question mark (03FH) 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 0FFH should be expected. If the open was successful, a return value in the range of 0 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

FUNCTION 16: CLOSE FILE

Entry Parameters:

Register C: 10H
Registers DE: FCB Address

Returned Value:

Register A: Directory 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 0 through 3 for the reasons described above under 'Open File'. An unsuccessful close operation, either because of a disc error or some other malfunction, will return a 0FFH.

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   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 IN DE
|      MVI   C,10H      ;MOVE SFC 16 INTO C
|      CALL  BDOS       ;CLOSE FILE
|      CPI   0FFH       ;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: Directory Code

This function is used to return the actual directory entry in the DMA 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 0FFH is returned in register A. If an entry is found, register A will be set to a value between 0 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 4 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 field, byte 0. This causes CP/M to search all disc drives which 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,'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,11H         ;MOVE SFC 17 INTO C      |
|         CALL BDOS          ;SEARCH FOR FILE SPEC    |
|         CPI   OFFH         ;ERROR/NO FILE?         |
|         JZ   ERR           ;YES - GOTO ERR          |
| OK     . . .               ;NO - PROCEED            |
+-----+

```

Function 18

FUNCTION 18: SEARCH FOR NEXT

Entry Parameters:

Register C: 12H

Returned Value:

Register A: Directory Code

This function is used in conjunction 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 0FFH when no additional qualifying entries are found in the directory.

EXAMPLE: Find next occurrence of specified filename

```
+-----+
|  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,12H             ;MOVE SFC 18 INTO C
|      CALL  BDOS              ;SEARCH FOR FILE
|      CPI   0FFH              ;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: Directory 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. This 'wild card' allows you to delete all files which 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 0 through 3 if the delete completes normally. A value of 0FFH is returned in case of an error (for example, no such file exists).

EXAMPLE: Delete 'B:PAYROLLZ.DAT'

```

+-----+
|  BDOS   EQU   0005H           ;MAIN CP/M ENTRY POINT
|  FCB    DB    00,'PAYROLLZ.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,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

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,'PAYROLLDAT',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: Directory 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 00H.

As with the 'read' 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,'PAYROLLDAT',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

FUNCTION 22: CREATE FILE

Entry Parameters:

Register C: 16H
Registers DE: FCB Address

Returned Value:

Register A: Directory 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 00H, 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 0 through 3. The value indicates the offset of the FCB within the physical record containing the directory entry for this file. A value of 0FFH 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:PAYROLLA.DAT'

```
+-----+
| BDOS   EQU   0005H      ;MAIN CP/M ENTRY POINT
| FCB    DB    01,'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
|          . . .        ;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

FUNCTION 23: RENAME FILE

Entry Parameters:

Register C: 17H
Registers DE: FCB Address

Returned Value:

Register A: Directory Code

This function is used to rename 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 0 and 3. An unsuccessful attempt will return a value of 0FFH in register A.

Wild card characters may exist in any of the file name or type 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 |
|  DBLOCKA DS    16             ;REMAINDER OF FCBA - GROUP |
|  CRA     DS     1             ;FCBA CURRENT RECORD |
|  RECA    DS     3             ;r0,r1, and r2 |
|  FCBB    DB     00,'NEWFILE TXT',00,00,00,00 |
|  DBLOCKB DS    16             ;REMAINDER OF FCBB - GROUP |
|  CRB     DS     1             ;FCBB CURRENT RECORD |
|  RECB    DS     3             ;r0,r1, and r2 |
|          . . . |
|          LXI    D,FCB         ;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 drives have been accessed or selected since the most recent re-load of CP/M. This could be useful in determining which drives have been placed 'on-line'. However, this function can not determine when the operator removes a disc after it has been logged here.

The 'login vector' is returned in the HL register pair. The H register is always 00H. Each bit in the L register corresponds to one of the possible 8 drives supported on your HP 125. The least significant bit of L contains the status bit for drive 'A:', while the most significant bit contains the status bit for drive 'H:'. This is illustrated in Table 9-3 below.

Table 9-3. Sample Return Login Vector Byte
L Register Only

```

+-----+
| H : G : F : E : D : C : B : A |
+-----+

```

If the status bit for a drive is '0', the drive has not been logged in, while a value of '1' signifies the drive is on-line. Attempts to access drives which 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   0005H      ;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   02H        ;IS BIT FOR B: SET? |
|      JZ    NOT.ON     ;NO - B: NOT ON-LINE |
| OK    . . . |
|      ;YES - DISC IS LOGGED |
+-----+

```

Function 25

FUNCTION 25: RETURN CURRENT DISK

Entry Parameters:

Register C: 19H

Returned Value:

Register A: Current Disk

This function returns the disc identifier for the currently selected mass storage device. The values in register A will be between 0 and 7, corresponding to drive 'A:' through 'H:' respectively.

EXAMPLE: Determine whether 'E:' is current disc

```
+-----+
|  BDOS    EQU    0005H      ;MAIN CP/M ENTRY POINT  |
|          . . .           |
|          MVI    C,19H     ;MOVE SFC 25 INTO C    |
|          CALL   BDOS      ;FIND CURRENT DISC       |
|          CPI    04H       ;IS 'E:' SELECTED?      |
|          JZ     EDISC     ;YES - GOTO EDISC        |
|  NOT.E   . . .           |
+-----+
```

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 the 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

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 Chapter 12 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 which 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. See Chapter 12 for additional information.

Notice that, if a user removes or changes a disc without an intervening warm-start or disc system reset, the information maintained within the system tables may be incorrect.

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 DISK

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 '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

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, function call number 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 H register contains 00F, while the 8 bits in L will reflect the status of drives 'A:' through 'H:'. The least significant bit reflects status of 'A:'.

Table 9-4. Sample Return Read Only Vector
L Register Only

```
+-----+
| H : G : F : E : D : C : B : A |
+-----+
```

A bit value of 0 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    0005H        ;MAIN CP/M ENTRY POINT |
|          . . .              |
|          MVI    C,1DH        ;MOVE SFC 29 INTO C     |
|          CALL   BDOS         ;RETURN VECTOR          |
|          MOV    A,L          ;MOVE VECTOR INTO A     |
|          CPI    00H          ;ARE ALL DISCS R/W?     |
|          JZ     ALL          ;YES - GOTO 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: Directory Code

The Set File Attributes function allows programmatic manipulation of permanent indicators attached to files. In particular, the R/O and System attributes (t1' and t2') can be set or reset. 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 filename, 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 file read/only

```

+-----+
|  BDOS    EQU    0005H        ;MAIN CP/M ENTRY POINT
|  FCB     DB     00,'PAYROLLDAT',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          ;READ 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

FUNCTION 31: GET ADDR (DISK 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 depth in Chapter 12.

One of the tables used by CP/M is the 'disc parameter block', 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.

Notice that, if a user removes or changes a disc without an intervening warm-start or disc system reset, the information maintained within the system tables may be incorrect.

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 an user number out of this range, the value of (E MOD 32) will be the user 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 USER NUMBER         |
| OK     . . .                 ;USER NUMBER NOW 1      |
+-----+
    
```

Function 33

FUNCTION 33: READ RANDOM

Entry Parameters:

Register C: 21H
Registers DE: FCB Address

Returned Value:

Register A: Return 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: r0, r1, and r2.

The r0,r1 byte pair is treated as a double-byte 'word' which contains the physical record number to be read. Byte r0 is the least significant 8 bits, while byte r1 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 00H) 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 r0,r1. 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 r0/r1 with the desired record number. Remember: r0 contains the least significant bits! Proceed with a 'CALL' to 0005H as with all other system function calls to execute the function.

If the read is successful, the A register will contain a '00H' 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 9-5. 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,'PAYROLLDAT',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

FUNCTION 34: WRITE RANDOM

Entry Parameters:

Register C: 22H
Registers DE: FCB Address

Returned Value:

Register A: Return 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    ERR5            ;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 r0, r1 and r2.

CP/M calculates the record number one greater than the last record in the file, and returns that value in the r0,r1 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,'PAYROLLDAT',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

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,'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           |
|
|      LHL    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: 00H

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

The D register should contain a 00H. Each bit within the E register represents the state to which each disc is to be set. The least significant bit represents drive 'A:', and the most significant bit represents 'H:'. If any bit is set to '1', the drive is to be reset to 'read/write' status. A value of '0' means no change should be made in the status. It is the login vector bit which is actually reset by this call.

EXAMPLE: Reset all discs

```

+-----+
|  BDOS  EQU   0005H      ;MAIN CP/M ENTRY POINT
|          . . .
|          LXI   D,0FFH   ;SETUP ALL DISC CODES
|          MVI  C,25H     ;MOVE SFC 37 INTO C
|          CALL  BDOS     ;RESET DISCS
|          . . .
+-----+

```

Function 38

FUNCTION 38: UNUSED

Function 39

FUNCTION 39: UNUSED

FUNCTION 40: WRITE RANDOM WITH
ZERO FILL

Entry Parameters:

Register C: 28H
Registers DE: FCB Address

Returned Value:

Register A: Return 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,'PAYROLLDAT',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
|      . . .
+-----+

```

Summary

In this chapter you have seen the 'standard' CP/M system function calls. In the next chapter, you will see the enhanced calls that have been implemented on your HP 125. By using these calls, you can use the advanced features of your HP 125 without losing all of the compatibility with most CP/M systems. This assures lasting value of your software.

EXTENDED SYSTEM FUNCTION CALLS

Introduction

In the previous chapter, you saw the many system function calls available through CP/M. You also saw that, by using those calls, your application did not need to handle the physical device I/O so often required with microcomputers. Access to the 'real' world is provided simply by using the function calls to the console, reader, punch, printer and disc. However, CP/M provides access to no other types of devices: on the HP 125, CP/M has been enhanced to overcome this limitation.

To address the added features of the HP 125, 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 HP 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 10-1. HP Extended System Function Calls

CP/M CALL	SUB FUNCTION	OPERATION DESCRIPTION
255	0	Return CP/M Identifier
255	112	Reader Status
255	113	Read Memory File
255	114	Set Pass Thru Datacomm Mode
255	115	Set 8 Bit Datacomm Mode
255	116	Set Default Datacomm Mode
255	117	Reserved for HP Use
255	118	HP-IB Device Output
255	119	Write Memory File
255	120	Return Region Bounds
255	121	Reserved for HP Use
255	122	Chain to Specified Program
255	123	Display Terminal Message
255	124	Disable Keycode Mode
255	125	Enable Keycode Mode
255	126	Interrogate and Alter Jump Vector
255	127	HP Revision Number Program

Table 10-2 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 10-2. Sample HP Extended System Function Call

BDOS	EQU	0005H	;BDOS Entry Point
HPFUN	EQU	0FFH	;HP Extended Call
SUBFN	EQU	74H	;Desired Sub-Function
	ORG	0100H	;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 chapter will introduce you to each, and will show you how to use these calls. In several cases, additional discussion is included in other chapters.

HP FUNCTION 0 : RETURN OEM NUMBER

Entry Parameters

Register Pair BC: 00FFH



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 125.

The return value passed in the HL register pair indicates the OEM number of the supplier of CP/M. The Hewlett-Packard OEM number is 209, or 0D1H. 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   0005H       ;MAIN CP/M ENTRY POINT |
| HPZERO EQU   00FFH       ;SUB-FUNCTION ZERO      |
|         . . .           |
|         LXI   B,HPZERO   ;SUB-FUNCTION IN BC     |
|         CALL  BDOS       ;RETURN NUMBER INTO HL  |
|         MOV  A,L         ;MOVE ID INTO A         |
|         CPI  0D1H        ;IS THIS HP CONTRACT ID? |
|         JNZ  NONHP       ;NO - GOTO NON-HP ROUTINE |
| OK      . . .           ;YES - PROCEED           |
+-----+

```


Sub-Function 127

HP FUNCTION 127 : Return HP CP/M Revision Number

Entry Parameters

Register Pair BC: 7FFFH

Return Parameters

Register Pair HL: 0001H

This sub-function is used to return the current revision of HP extended CP/M. 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 125, but that it is using a specific release version of HP CP/M.

This sub-function is accessed by passing 0FFH 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
| HPREV  EQU   7FFFH      ;SUB-FUNCTION 127
|
|      . . .
|      LXI   B,HPREV     ;SUB-FUNCTION IN BC
|      CALL  BDOS        ;RETURN REVISION IN HL
|      MOV   A,H         ;MOVE HIGH BYTE INTO A
|      CPI   00H         ;IS IT ZERO?
|      JNZ   REVERR      ;NO - GOTO REVERR
|      MOV   A,L         ;MOVE LOW BYTE INTO A
|      CPI   01H         ;IS IT ONE?
|      JZ    REV1        ;YES - OK
| REVERR . . .          ;NO - REVISION ERROR
+-----+
```

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 Chapter 12, 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 10-3. Byte 0 contains the 'jump vector number' which indicates which entry in the table is of interest. These JVNs are indicated in Table 10-4 which follows. Chapter 12 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 00H will indicate that this is a 'read', while a value of 01H will actually change the jump vector contents.

Table 10-3. Jump Vector Buffer

+-----+						
	+-----+					
	JVN	FLAG	JMP	LO	HI	
	+-----+					
	DE + 0	1	2	3	4	
+-----+						

If the request is a 'read', CP/M will access the jump vector table in BDOS 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 high byte of the address in byte 3 and the low 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 10-4. 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 CONOUT	;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 SECTTRAN	;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. SEE TABLE 15-4. |
| 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 |
| 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 |
| ; |
| LHL D,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 |
+-----+
```

Sub-Function 125

HP FUNCTION 125: Enable Keycode Mode

Entry Parameters

Register Pair BC: 7DFH

Return Parameters

Register Pair HL: Return Code

This sub-function is used to put the HP 125 into 'keycode' mode. In this mode, the TPU passes each key to CP/M for processing: no terminal processing is done. This feature is discussed in depth in Chapter 12. Normally, this sub-function is used in conjunction with HP sub-function number 124, 'Disable Keycode Mode'.

Note that, with Keycode Mode enabled, the TPU will transfer an eight bit ASCII value to CP/M. If no other action is taken, note that BDOS will return only the seven bit code via console input system function calls. Refer to the discussion in Chapter 12 for a detailed explanation of how the entire eight bits can be accepted.

To enable this feature, load the BC register pair with 7DFH. Upon return, HL will be non-zero. All subsequent key presses at the HP 125 keyboard will result in a 'keycode' byte being passed to the CPU. A list of these keycodes appears in Appendix A.

The return code is not significant: no errors should occur during this sub-function call.

It is suggested that the TPU is selected as both the console input and the console output devices. In this way, the keycode returned can be reliably decoded.

EXAMPLE: Enable Keycode Mode

```
+-----+
| BDOS   EQU   0005H      ;MAIN CP/M ENTRY POINT |
| KEYON  EQU   7DFH      ;SUB-FUNCTION 125          |
|        . . .          |
|        LXI   B,KEYON   ;SUB-FUNCTION IN BC       |
|        CALL  BDOS     ;ENABLE KEYCODE MODE       |
|        . . .          |
+-----+
```

HP FUNCTION 124: Disable Keycode Mode

Entry Parameters
Register Pair BC: 7CFFH

Return Parameters
Register Pair HL: Return Code

This sub-function disables keycode mode, and represents the default state for the HP 125. This sub-function is used in conjunction with sub-function 125, 'Enable Keycode Mode'.

Load the BC register pair with 7CFFH and call BDOS. The return will leave a non-zero value in HL, although its value is not significant.

For additional information about keycode mode, see Chapter 12.

EXAMPLE: Disable Keycode Mode

```
+-----+
| BDOS   EQU   0005H           ;MAIN CP/M ENTRY POINT |
| KEYOFF EQU   7CFFH           ;SUB-FUNCTION 124       |
|         . . .               |
|         LXI   B,KEYOFF       ;SUB-FUNCTION IN BC      |
|         CALL  BDOS           ;DISABLE KEYCODE MODE    |
|         . . .               |
+-----+
```

Sub-Function 123

HP FUNCTION 123: Display Terminal Message

Entry Parameters

Register Pair BC: 7BFFH

Register Pair DE: Address of Buffer

Return Parameters

Register Pair HL: Return Code

This sub-function permits your application to display an error or warning message on the HP 125 screen, and require the user to take some specific action to continue processing.

To set up this call, load 7BFFH into the BC register pair. The address of an information buffer is loaded into DE. The format of this buffer is illustrated in Table 10-5.

Table 10-5. Terminal Message Buffer

DE +		0	1	2	n
		LEN	FLAG	MESSAGE BUF					

Byte 0 of the buffer, LEN, contains the length of the 'message buffer' which follows. It should be equal to 'n', since it represents the length of FLAG plus the length of the message.

Byte 1, FLAG, indicates whether the message is an 'error' or a 'warning'. The difference will determine how a program may continue.

If Byte 1 contains a 00H, the message is considered an 'error'. To recover, the user must press the 'RETURN' key, which is absorbed by the TPU. That is, all keys are ignored until a 'RETURN' is pressed.

Actual program execution will continue after such an 'error' call is made. However, the TPU will 'wait' for RETURN to be pressed. This 'wait' causes the TPU to 'hang up' until RETURN is pressed, so any operation involving the TPU will also wait until the user presses RETURN. IF IT IS ESSENTIAL FOR YOUR PROGRAM TO PAUSE, PERFORM ANY TASK WHICH REQUIRES USE OF THE TPU. A simple example of such a task might be to output a null character to the console.

If Byte 1 contains a 01H, the message is considered a 'warning' or an 'informational' message. All other keys on the system are active, although the labels for the programmable softkeys are temporarily removed. The message is removed when 'RETURN' is pressed, and the TPU transmits the 'CR' character to the controlling program.

The bytes from 2 through the end of the buffer, MESSAGE BUF, are displayed on the CRT at line 25. If an 'error' message has been selected, line 26 will prompt 'Press RETURN to clear'. The softkey labels are, of course, disabled temporarily. Remember, however, that the keys remain active during 'warning' messages.

A maximum length of any message is 80 characters. This count does include any 'invisible' escape sequence characters that may be used to control display enhancements.

The Hewlett-Packard convention for these error/warning messages follows these guidelines as to the source of the message:

Terminal Firmware Messages are preceded by one asterisk CP/M messages are preceded by two asterisks Application program messages are preceded by three asterisks

This is only a suggested convention: no checking is performed on the contents of the message buffer.

EXAMPLE: Display the warning message 'HELLO SAM'

```

+-----+
|  BDOS  EQU   0005H      ;MAIN CP/M ENTRY POINT  |
|  CRTMSG EQU   7BFFH      ;SUB-FUNCTION 123    |
|  MESBUF DB    9,1,'HELLO SAM' ;MESSAGE BUFFER    |
|  LEN    EQU   MESBUF+0   ;LEN IS FIRST BYTE   |
|  FLAG   EQU   MESBUF+1   ;FLAG IS SECOND BYTE |
|  MSG    EQU   MESBUF+2   ;MESSAGE STARTS AT BYTE 3 |
|          . . .          |
|          LXI   B,CRTMSG   ;SUB-FUNCTION IN BC      |
|          LXI   D,MESBUF   ;ADDR OF BUFFER IN DE     |
|          CALL  BDOS       ;DISPLAY WARNING MESSAGE   |
|          CALL  GETCR      ;ROUTINE TO ACCEPT THE    |
|          . . .          ;EXPECTED CARRIAGE RETURN   |
+-----+

```



Sub-Function 122

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 highlevel 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 0000H), the CCP is re-loaded by CP/M. Once CCP is running, it will check for a file names '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 either the normal start-up program should be run, or that another program is to be executed instead.

To specify that the standard program is to be run, simply pass 0000H in the DE register pair. Unless your system has been altered, this call will execute 'WELCOME.COM'.

To specify some other program, or system command, simply load the DE register pair with the address of an information buffer. The buffer should be of the format shown below in Table 10-6.

Table 10-6. Chain Program Buffer

+-----+											
		+-----+									
		LEN	CCP	COM	M	A	N	D	BU	F	
		+-----+									
	DE +	0	1	2	n	
+-----+											

Byte 0, 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 |
+-----+
```

Sub-Function 121

HP FUNCTION 121: Reserved for HP Internal Use

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 10-7. Refer to Table 10-3 for an illustration of the CP/M memory map.

Table 10-7. 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. The Reserved RAM is used as a stack area for the 8K ROM which handles the communication with the TPU and other scratch pad storage required. 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

```
+-----+
| BOOT   EQU   0000H      ;CP/M WARM BOOT ENTRY
| BDOS   EQU   0005H      ;MAIN CP/M ENTRY POINT
| BOUNDS EQU   78FFH      ;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 119: Write Memory File

Entry Parameters

Register Pair BC: 77FFH

Register Pair DE: Address of Buffer



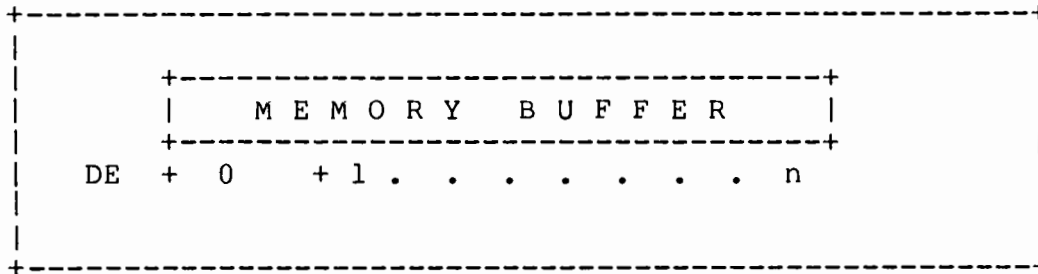
Return Parameters

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 above, 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 10-8.

Table 10-8. Write Memory File Buffer



The MEMORY BUFFER is of any reasonable length, and may contain any data which must be made available to the next program which is run, usually via a 'chain' sub-function described earlier. It is critical that any programs which share data via this memory file must expect a common data format, since CP/M makes no effort to distinguish between various items of data within the buffer.

When a buffer is written with this call, the address of the buffer is stored in BIOS, along with a flag indicating the 'age' of the buffer. When a warm boot is executed, the flag is incremented, indicating to BIOS that the buffer may be purged upon the next subsequent warm boot. This assures that a memory buffer is available only to the 'son' program, so that the buffer is not available to subsequent a 'grandson'.

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 120 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 0000H 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   0005H      ;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   5000H      ;RE-POSITION PC          |
|  MYNAME  DB    'FATHER'   ;LEAVE CURRENT PROGRAM   |
|          . . .          |
|          ;NAME IN MEMORY AT 5000H                 |
+-----+

```

HP FUNCTION 118: Output to HP-IB Device

Entry Parameters

Register Pair BC: 76FFH

Register D: HP-IB Device Address

Register E: Data Byte to be Output

Return Parameters

Register A: Return Code

This sub-function allows an application to output a byte of data to a specific HP-IB device. This is similar to the 'byte output' simple I/O function calls in standard CP/M, except that the data is sent to an HP-IB device. Be certain not to send data to HP-IB address 0 or 2, since these are reserved for Hewlett-Packard supplied disc devices.

The HP-IB device address is passed in register D, and the data byte to be output is passed in register E.

A value of 00H returned in register A indicates successful completion. A non-zero value indicates that an error has taken place: for some reason, the byte was not output. An incorrect device address, or a device that is turned off, could be the cause of the problem.

EXAMPLE: Output an 'X' to HP-IB device at address 5

```

+-----+
|  BDOS   EQU   0005H      ;MAIN CP/M ENTRY POINT |
|  HPIB   EQU   76FFH      ;SUB-FUNCTION 118      |
|  BADDR  EQU   5         ;HP-IN DEVICE ADDRESS      |
|          . . .          |
|          LXI   B,HPIB    ;SUB-FUNCTION IN BC      |
|          MVI  D,BADDR    ;BUS ADDRESS INTO D      |
|          MVI  E,'X'     ;MOVE 'X' INTO E          |
|          CALL BDOS      ;OUTPUT CHARACTER          |
|          CPI   0         ;ANY ERRORS ON OUTPUT?    |
|          JNZ  ERR       ;YES - GOTO ERR            |
|  OK     . . .          ;NO - PROCEED              |
+-----+

```


Sub-Function 117

HP FUNCTION 117: Reserved for HP Internal Use

HP FUNCTION 116: Set Default Datacomm Mode

Entry Parameters

Register Pair BC: 74FFH

Return Parameters

Register Pair HL: Return Code

This sub-function sets Datacomm Port #1 to the 'default' mode, clearing the effects of sub-functions 115 and 114.

Default mode implies 7 bit ASCII data, with the high order bit used for parity. Also, all handshaking is handled by the TPU, transparently to the user.

The return code will be non-zero upon return. No errors are anticipated and no significant data is returned.

EXAMPLE: Reset datacomm port to default mode

```

+-----+
| BOOT   EQU   0000H       ;CP/M WARM BOOT ENTRY  |
| BDOS   EQU   0005H       ;MAIN CP/M ENTRY POINT  |
| RESET  EQU   74FFH       ;SUB-FUNCTION 116      |
|         . . .           |
|         LXI   B,RESET    ;SUB-FUNCTION IN BC      |
|         CALL  BDOS       ;RE-SET DATACOMM PORT    |
|         . . .           |
+-----+

```

Sub-Function 115

HP FUNCTION 115: Set 8 Bit Pass-thru Datacomm Mode

Entry Parameters

Register Pair BC: 73FFH

Return Parameters

Register Pair HL: Return Code

This sub-function sets Datacomm Port #1 to '8 bit pass-thru' mode, which causes the TPU to stop processing the high-order bit as parity, and to stop processing datacomm handshaking and protocol.

Normally, the high-order bit is removed during transmission via datacomm, since that bit is used to indicate parity. To transmit binary data, however, the high-order bit is significant, and this call permits an application to turn off the automatic clearing of that bit.

The TPU also intercepts all datacomm control characters and any escape sequences which affect the terminal. By using this mode, an application can force the TPU to pass all control to the CPU. No handshaking or datacomm protocol is handled by TPU.

Simply pass 73FFH in the BC register pair. The return code will be non-zero, available in the HL register pair.

EXAMPLE: Put datacomm into 8-bit mode

```
+-----+
| BOOT   EQU   0000H      ;CP/M WARM BOOT ENTRY  |
| BDOS   EQU   0005H      ;MAIN CP/M ENTRY POINT  |
| BITS8  EQU   73FFH      ;SUB-FUNCTION 115      |
|        . . .           |
|        LXI   B,BITS8    ;SUB-FUNCTION IN BC      |
|        CALL  BDOS       ;SET 8 BIT MODE          |
|        . . .           |
+-----+
```

Sub-Function 114

HP FUNCTION 114: Set 7 Bit Pass-thru Datacomm Mode

Entry Parameters

Register Pair BC: 72FFH



Return Parameters

Register Pair HL: Return Code

This sub-function sets Datacomm Port #1 to 'pass-thru' mode. This is a unique feature of the HP 125, which allows the CPU to control the data communications protocol.

This sub-function performs similarly to sub-function 115, except that the TPU continues to process parity. Only 7 bits of each byte are available.

The return code will be non-zero, available in the HL register pair.

EXAMPLE: Turn off TPU handshake processing

```
+-----+
| BOOT   EQU   0000H      ;CP/M WARM BOOT ENTRY |
| BDOS   EQU   0005H      ;MAIN CP/M ENTRY POINT  |
| PASS   EQU   72FFH      ;SUB-FUNCTION 114        |
|         . . .          |
|       LXI   B,PASS      ;SUB-FUNCTION IN BC      |
|       CALL  BDOS        ;SET PASS THRU MODE      |
|         . . .          |
+-----+
```

Sub-Function 113

HP FUNCTION 113: Read Memory File

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
|  TEMP    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 112: Reader Status

Entry Parameters
 Register Pair BC: 70FFH

Return Parameters
 Register A: Reader Status

This sub-function allows an application to check whether a byte of data is available at the CP/M 'reader' device.

Remember that standard CP/M includes a status check for the console input, but not for the reader. If SFC #3, reader input, is called, no return is made until a byte is available. This can cause a program to 'hang'. To alleviate this problem, this sub-function permits a status check of the reader device.

The status is returned in register A. If A contains 00H, no character is available. A value of 01H indicates that a byte of data is ready, and can be accessed using SFC #3.

EXAMPLE: Accept a byte from the reader device

BOOT	EQU	0000H	;CP/M WARM BOOT ENTRY
BDOS	EQU	0005H	;MAIN CP/M ENTRY POINT
RDRST	EQU	70FFH	;SUB-FUNCTION 112
		. . .	
RETRY	LXI	B,RDRST	;SUB-FUNCTION IN BC
	CALL	BDOS	;CHECK STATUS
	CPI	00H	;IS CHAR READY?
	JZ	RETRY	;NO - TRY AGAIN
READY	MVI	C,3	;SFC 3 - READER INPUT
	CALL	BDOS	;READ CHARACTER
		. . .	

Summary

This concludes the list of HP extended CP/M sub-functions. In later chapters you will see more specific information on several of these calls.

In the next chapter, you will see how the HP 125 allows you to 'equate' various physical I/O devices to the four logical devices known to CP/M. With that data, you can begin to treat datacomm ports as reader/punch devices, or perform any other reasonable mapping your application may require.

DEVICE ASSIGNMENTS

Introduction

Standard CP/M provides for a distinction between logical and physical devices. In this manner, the operating system is capable of growing beyond the technology of a particular era, or of a particular set of hardware, so that software can remain compatible as your needs grow.

In this chapter, you will see how standard CP/M utilizes physical and logical devices; what enhancements are available because of HP 125 features; and how you can take advantage of standard CP/M devices with your HP 125.

Logical and Physical Devices

Standard CP/M 'knows about' four types of logical devices, which are described below. Later you will see how to 'equate' HP 125 physical devices with these logical device names.

CONSOLE	The principle I/O device for user interaction, typically a CRT device. The device name is 'CON:'.
READER	The principle serial input device, either paper or mag tape. The device name is 'RDR:'.

PUNCH	The principle serial output device, either paper or mag tape. The device name is 'PUN:'.
LIST	The principle list device, either line printer or teletype. The device name is 'LST:'.

Each of the above devices is called a 'logical' device because the name need not describe an actual (or physical) device. In other words, the logical punch device could, in fact, be any type of output device (such as a printer) which is treated by CP/M as if it were a paper tape punch.

IOBYTE

CP/M allows the dedication of a single byte of memory to describe the mapping between physical and logical devices. The mappings of each of the four logical devices is managed by allocating two bits of IOBYTE for each logical device.

Table 11-1 illustrates the format of IOBYTE.

Table 11-1. IOBYTE Format

+-----+-----+-----+-----+							
	+-----+-----+-----+-----+						
	LIST	PUNCH	READER	CONSOLE			
	+-----+-----+-----+-----+						
	Bit : 7	: 6	: 5	: 4	: 3	: 2	: 1 : 0 :
+-----+-----+-----+-----+							

Each logical device can assume values from 00 to 11 (Binary); thus four physical devices can be mapped by each logical device.

As you know by now, the HP 125 does not support either a paper tape reader or punch device. For that matter, no magnetic tape device is supported either. Because of the nature of the HP 125, and because of the powerful 'device mapping' available using escape sequence programming, IOBYTE is normally used only for the list device LST:. The escape sequence is described later in this chapter.

The list device is known as 'LST:' only to the PIP and STAT transient commands. It is selected by setting the two high order bits in IOBYTE to one of the values described in Table 11-2 below. The PIP and STAT logical names, as well as the actual devices, are described also.

Table 11-2. IOBYTE LST: Device Value

BIT VALUE		DEVICE	DESCRIPTION
BIT 7	BIT 6	NAME	
0	0	TTY:	GENERAL LIST DEVICE
0	1	CRT:	THERMAL PRINT MECH
1	0	LPT:	SERIAL PORT 2 DEV
1	1	UL1:	HP-IB PRINTER



The HP-IB printer at address 1 is selected whenever the value of IOBYTE is 0COH. The CPU can process this output directly since the HP-IB interface is a part of the CPU. Only minimal TPU action is necessary.

To specify the optional internal thermal printer as the LST: device, IOBYTE must contain a value of 40H. Since the thermal printer is connected to the TPU, data must be passed to the TPU for processing. This is handled automatically by the CPU.

The serial device on Datacomm Port #2 is selected as LST: when the value in IOBYTE is 80H. As with the thermal printer, the data must be passed to the TPU for processing.

When the value of IOBYTE is set to 00H, the 'General List Device' is selected.. Any output for 'General List' is passed to the TPU. To determine the actual device(s) to be used, the TPU will 'look at' the values specified in the terminal configuration menu. There, any of four devices can be enabled: the CRT display, the HP-IB printer, the thermal printer, or Datacomm Port #2. Since more than one can be 'on', list output can be directed to multiple printers. This allows the user to specify the printer device(s) while running. This is the additional flexibility of the HP 125.

For example, assume a letter is to be printed in final form on a letter-quality printer on Datacomm Port #2. Further, assume a 'draft' copy is to be printed on the HP-IB printer. This could be done by printing the letter twice, once on each device.

On the HP 125, both copies can be printed simultaneously specifying the 'General List' device code in IOBYTE. This is, incidently, the default value. Then, simply specify that both the HP-IB printer and the Datacomm Port #2 printer are 'on' in the 'config' menu.

When using 'Printer Echo' from the CCP, and the general list device is enabled, be certain to turn 'off' the console as one of the list devices in the 'config' menu. If this is not done, you will see two of every character printed: one as the console output, and a second as the 'printer echo'.

This concludes our discussion of the CP/M 'list' device. Now you will see how the HP 125 augments IOBYTE and the CP/M function calls to provide additional flexibility for those devices other than 'LST:'.

As you have seen, CP/M function calls provide simple device I/O to four device types: console, list, reader, and punch. You've also seen how to implement various list devices using IOBYTE. But the HP 125 offers additional power in handling these two devices, as well as allowing a unique method of defining the reader and punch devices.

Device Mapping

The HP 125 offers a variety of devices which were not available when CP/M was initially designed. For this reason, an escape sequence has been implemented on the HP 125 which allows CP/M non-list logical devices to be 'mapped' to various devices known to the HP 125.

In the escape sequence, you must specify the 'source' of data, the 'destination' of the data, and any special 'modes' of operation. The TPU will establish a logical link between the source device and the destination device(s) you specify.

The format of the escape sequence is:

```
ESC & i <source dev> s <dest dev> d <mode> M
```

Refer to Table 11-3 for valid source and destination codes, and to Table 11-5 for valid mode values.

Table 11-3. Source and Destination Device Assignments

SOURCE DEVICES		<source dev> code
Datacomm Port #1		0
Datacomm Port #2		2
Keyboard		7
CP/M Console Buffer		8
CP/M Punch Output		10
'Self'		32
DESTINATION DEVICES		<dest dev> code
Datacomm Port #1		16
Datacomm Port #2		18
HP 125 CRT Display		23
CP/M Console Buffer		24
CP/M Reader Input		25
'Bit Bucket'		26
HP-IB Printer Output		27
Serial Printer Output		28
Thermal Printer Output		29
Default Printer Output		30
'Self'		32

The rules which apply to other escape sequences remain valid here. Specifically, the parameters may be specified in any order and the sequence must be terminated with an upper case character. There are three specific restrictions which apply to this sequence only:

- Only one source device may be specified per sequence
- Up to eight destination devices may be selected per sequence
- Only one mode parameter may be included per sequence

Most of the devices included in Table 11-3 should be familiar to you by now. The only exception is the 'Bit Bucket', which is an unlimited output device which, in fact, has no physical device assignment. Any data sent to this 'bit bucket' is lost forever.

The 'Datacomm Port #2' and 'Serial Printer' destination codes both map to the same physical device. The two codes are there only for clarity: there is no difference in how the two device codes are processed.

The 'Default Printer' device, destination code 30, refers to the printer(s) enabled in the OPSYS GENERAL LIST DEVICE field in the TPU 'config' menu.

When 'Self' is specified as either a 'source' or 'destination' device, that device which initiated the 'ESC & i' sequence is the selected device. For example, if a program running on a computer connected remotely via Datacomm Port #2 initiates a device mapping sequence and specifies 'Self' as a source, you can consider that functionally identical to a source code of 2.

Upon examining Table 11-3, you might be surprised to find the CP/M Reader, an output device, to be among the 'destination' list. Or, you might have wondered about the CP/M Punch device being available as a 'source' of data. Obviously, the terms 'source' and 'destination' have some hidden meaning!

Selecting Source and Destination Devices

It will be helpful to remember to 'whom' the escape sequence is directed: the TPU. You may wish to think of the TPU as a 'little man' inside of your HP 125. It is his job to map logical devices, such as the reader, to physical devices, such as the thermal printer. From his point of view, the physical and logical devices appear as shown in Table 11-4. Those devices which are connected directly to the TPU are considered as 'outside' of the HP 125. Those devices which are connected or originate with the CPU are 'inside' to the TPU.

As you can see from Table 11-4, each device can be considered as a 'post office box'. Don't confuse this terminology with the 'mailbox' through which the two Z80 processors communicate. The 'source' and 'destination' terminology, directed to the TPU, goes like this:

When the 'ESC & i' escape sequence is used, the TPU establishes a logical connection, or mapping, from the 'source' device to the 'destination' device(s). Once the map is established, whenever a byte of data appears at the 'source' box, the TPU will transfer that byte to the 'destination' box.

Table 11-4. TPU View of Logical and Physical Devices

OUTSIDE WORLD		INSIDE WORLD
Keyboard		Console
CRT		Reader
DC Port #1		Punch
	T P U	
DC Port #2		HP-IB
Serial Prt		Bit Bucket
Int Prt		

Device Mapping Modes

The mode, or 'M', parameter of the 'ESC & i' sequence offers several different ways of mapping the various destination and source devices. For example, it is possible to add a particular mapping to those already existing, or to un-map all destinations so that a particular device will have no effect on the HP 125.

Only the least significant four bits of the mode parameter value are used in calculating the actual mode, so some care must be taken to assure the mode you want is the one which is enabled.

There are four possible effects of the mode parameter, depending upon the value of the least significant four bits of the ASCII character within the mode field. These four, along with the possible valid values, are:

- **Default State.** This option is selected whenever no bits are set in the parameter. A '0' will accomplish this task. Three specific actions are directed by this value of '<mode>':
 - The current state of 'escape sequence processing' remains in effect for the specified source.
 - All existing device assignments involving the specified source are un-mapped.
 - A mapping is established for the specified source to the specified destination(s)
- **Process Escape Sequences.** This mode, also known as 'Pass Thru mode off', results whenever bit 0 is on. A value of '1M' will accomplish this task. The TPU will process any escape sequences received from the specified source device, and will not pass the sequences through to the destination(s).
- **Do Not Process Escape Sequences.** This mode, known as 'Pass Thru mode on', results whenever bit 1 is on. A value of '2M' will accomplish this task. The TPU will not execute escape sequences from the source. Instead, they will be passed through to the specified destination(s) for possible processing.
- **Additional Mapping.** This mode is selected whenever bit 2 is on. A value of '4M' will accomplish this task. In this case, any existing mappings involving the source device remain active. Also, the specified destination(s) is/are added to the mapping.
- **Un-Map Specified Device.** This mode, selected whenever bit 3 is set, can be selected with a mode of '8M'. The mapping between the specified source and destination is cleared, and the relationship is ended.

You have seen the effect of selecting a single mode action. It is possible to combine more than one action in each mode parameter by adding the least significant four bits of the desired mode value. Table 11-5 illustrates all the possible valid combinations, and what value is required in the mode field. Note that, unlike the 'source' and 'destination' fields, this field can accept non-numeric ASCII values.

Table 11-5. Mode Parameter Combinations

BIT PATTERN	ASCII CHAR	DEFAULT STATE	PROCESS ESC SEQ	DO NOT PROCESS	ADD'L MAP	UN-MAP DEV
0000	0	X				
0001	1		X			
0010	2			X		
0011	INVALID					
0100	4				X	
0101	5		X		X	
0110	6			X	X	
0111	INVALID					
1000	8					X
1001	9		X			X
1010	:			X		X

The combinations which are listed as 'INVALID' are those bit combinations which would specify both 'process' and 'do not process' escape sequences.

The two modes which affect the processing of escape sequences assume significance only when certain devices are selected as the 'source' device. Table 11-6 illustrates which devices will be impacted by escape sequence processing.

There is one case among the combinations of modes which assumes a special significance. That is, when 'Un-map Specified Device' mode is selected and no destination is provided. In this case, all devices mapped to the specified source device are un-mapped.

When a source is mapped to no destinations, no escape sequences from that device are processed at all, so care should be taken not to totally un-map a device which may generate escape codes for processing. Note that, by mapping a device to the 'bit bucket', you can retain escape sequence processing without receiving any actual data from that source.

Table 11-6. Pass Thru Devices

Keyboard	CP/M Punch Device
Datacomm Port #1	CP/M Console Device
Datacomm Port #2	

When 'Pass Thru' mode is enabled, the only escape sequence which is processed is 'ESC & i 3 2 s 5 M', which causes the TPU to start processing escape sequences once again, without affecting the destination device selection(s). Note, however, that this sequence is sent through to the destination device(s).

Examples

Some examples of this sequence will save many additional pages of explanation. Let's assume for a moment that you wish to write a program which communicates with a remote host computer system for file transfer. Assume further that your program will use the system function calls to output to the punch and input from the reader to communicate with the host system. This means that we need to establish a relationship between Datacomm Port #1 and the CP/M punch and reader logical devices.

When your program sends an output byte to the TPU via a 'Punch Output' system function call, the TPU receives the actual request.

In this example, we want the TPU to send the output byte to Datacomm Port #1. In the terminology of the TPU, 'When CP/M Punch is the source of the data, the destination should be 'Datacomm Port 1'. Any escape sequence which may be transmitted will be handled by the remote host and are not to be acted upon by the TPU. The escape sequence which must be sent to establish this mapping is:

```
ESC & i 10 s 16 d 2 M
```

This establishes the datacomm port #1 as the CP/M punch device (PUN:), with 'Pass Thru' mode on. The TPU will not execute any escape sequences which may be transmitted.

Again, the CP/M punch device is the 'source', and Datacomm Port #1 is the 'destination'.

The second part of this example requires us to establish a relationship between Datacomm Port #1 and the CP/M reader. This will permit our application to 'read' data from the remote system. In this case, we will assume that any escape sequences received from the remote are to be processed by the TPU.

We instruct the TPU as follows: 'Whenever there is a byte of data available on Datacomm Port #1, send it to the CP/M reader. Also, if you receive any escape sequences, process them accordingly.

In the terminology of the TPU: 'When Datacomm Port #1 is the source, the CP/M reader is the destination. Pass thru mode should be 'off': process any escape sequences received'.

The escape sequence:

```
ESC & i 0 s 25 d 1 M
```

Summary

These examples illustrate just a few of the possible uses of the device assignment escape sequence. With some imagination, you can log data to a remote system as if it were a printer, or even use a remote terminal to diagnose problems within a CP/M application. For any of these situations, clearly analyze the 'source' and 'destination' devices, and the correct escape sequence will fall into place.

This concludes our look at the HP 125 CPU. In the next section, you will see several advanced features of the HP 125. While most applications can be written using information presented up to this point, these additional features provide that extra power generally unavailable in micro-computers.

**ADVANCED CP/M
FEATURES****Introduction**

In previous chapters, you have learned how to program the TPU using escape sequences, and how to use BDOS function calls in your applications. You have seen in those chapters 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 chapter will provide you with that knowledge.

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 12-1.

Table 12-1. 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 0 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

As you can see, the jump vectors implement three major groups of functions: system initialization and re-start; 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 re-loads BDOS and the CCP whenever called. This is the routine accessed when a user application performs a 'JMP' to address 0000H. 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 00H is returned in the A register. If a character is ready, the A register contains 0FFH 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 (01AH).

HOME returns the disc head to track 0 on the selected disc device. On the HP disc drives, this is accomplished by calling SETTRK with a track value of 0.

SELDSK selects the disc drive given by the C register as the current mass storage device. A value of 00H selects drive 'A:': a value of 07H selects drive 'H:'. Each call to SELDSK returns the address of the Disc Parameter Header (DPH) for the selected disc in the HL register pair. An invalid disc specifier will return a 0000H in the HL register pair.

SETTRK positions the disc head at the track address specified in the BC register pair. For HP 82901 media, the range should be between 0 and 65 decimal. HP 9895 media requires a track address between 0 and 153 decimal, and IBM format discs in the HP 9895 require an address between 0 and 76 decimal.

SETSEC positions the currently selected disc head to the sector specified in the BC register pair. HP 9895 media require a sector value between 0 and 59. When IBM format media is used on the HP 9895, the sector value must be between 1 and 26 decimal. Note, however, that before a SETSEC call is made for an IBM format disc, the SECTTRAN call described at the end of this section must be called first to perform the sector translation.

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 0080H.

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 01H will be returned in the A register. A value of 00H 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 01H if an error occurs, and a 00H if the write is successfully completed.

LISTST returns the status of the currently selected list device. A value of 00H is returned in the A register if the list device is not ready to accept data, while a value of 0FFH 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. IBM formatted discs utilize a interleave factor of 6, and will use SECTRAN to provide this compensation. 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 zero. 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. As discussed in Chapter 11, the HP 125 uses IOBYTE only for the LIST device. 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 a track and sector address 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-tries if an error is encountered during a READ or WRITE operation.

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 eight possible disc drives on the HP 125 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 12-2.

Table 12-2. Disc Parameter Header

Dec	Hex	Symbol	Description
00 01	00 01	XLT	Translation Vector
02 03	02 03	SCR	Scratch Area #1 Address
04 05	04 05	SCR	Scratch Area #2 Address
06 07	06 07	SCR	Scratch Area #3 Address
08 09	08 09	DIRBUF	Directory Buffer Address
10 11	0A 0B	DPB	Disc Parameter Blk Address
12 13	0C 0D	CSV	Checksum Vector Address
14 15	0E 0F	ALV	Disc Space Vector Address

The meaning of each of these fields follows.

XLT is the address of the translation vector used when a particular disc format requires a sector interleave or skew. HP media interleave is compensated for internally, so no XLT entry is required for HP formatted media. This address should be provided in the HL register prior to using the SECTTRAN call described earlier. That call, through a table lookup, will determine the interleave factor of 6 used for the IBM media.

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

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 125 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 three entries in the DPB: one each for the 82901 mini-disc; one for the 9895A with HP formatted discs; and one for the 9895A with IBM formatted discs. Notice that the 9895A can assume multiple 'styles' depending on the format.

The organization of each DPB entry is illustrated in Table 12-3.

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 the data allocation block mask.

Table 12-3. Disc Parameter Block

Dec	Hex	Symbol	Description
00	00	SPT	Sectors per Track
01	01		
02	02	BSH	Block Shift Factor
03	03	BLM	Block Mask
04	04	EXM	Extent Mask
05	05	DSM	Data Max in Groups
06	06		
07	07	DRM	Directory Entries
08	08		Maximum
09	09	ALO	Directory Data
10	0A	AL1	Directory Data
11	0B	CKS	Check Sum Vector
12	0C		
13	0D	OFF	OP SYS Reserved Tks
14	0E		



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 0, the total number of data blocks is 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.

AL0/AL1 are two single byte fields used to reserve directory blocks. Each bit of the AL0/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. A discussion of groups is included in Chapters 12 and 13. On the 82901 disc, BLS is 1024 bytes (1K); on the 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 DPH's can address the same DPB. On the HP 125, 8 entries exist in the DPH Tables, while three exist in the DPB Table. Each disc drive requires one DPH, while one DPB exists for each of the 82901; the 9895 in HP format; and the 9895 in IBM-compatible format.

Keycode Mode

A feature unique to the HP 125 is the ability for a CP/M program to 'intercept' a keystroke and optionally alter the TPU defined function that will result. For example, a CP/M program can redefine the functioning of the 'DEL LINE' key under program control.

The HP 125 normally operates with keycode mode 'disabled' or 'off'. To enable keycode mode, use HP system subfunction call #125. It is necessary that the CP/M console device is defined as the HP 125 CRT and keyboard. No device assignment involving the console should be active.

Remember that the BIOS CONIN routine returns a seven-bit ASCII value with the high-order parity bit set to zero. While the keycodes returned for all ASCII characters use only seven bits, any special keys on the HP 125 require use of the high-order bit.

Fortunately, CP/M provides the flexibility to deal with this apparent challenge. The BIOS CONIN routine returns low seven bit keycode value in the A register. However, when the keycode mode is enabled, the full 8 bit keycode is also available to your program in the B register. Once your program has returned from CONIN, take the value from the B register as the full keycode. Calls for console input through BDOS will destroy the B register before returning to your application. You should use the BIOS CONIN rather than system function call number 1.

The use of an application-implemented keycode input routine is illustrated in Table 12-4.

Table 12-4. Sample Keycode Mode Program

```

boot    equ    0000h        ;Warm boot entry
bdos    equ    0005h        ;Main BDOS entry point
sfcl    equ    01h         ;Console Input Call
sfc2    equ    02h         ;Console Output Call
sfc9    equ    09h         ;Console Buffer Output Call
;
aids    equ    98h         ;Keycode for [AIDS] Key
;
bell    equ    07h         ;ASCII Bell Character
lf      equ    0ah         ;ASCII Line Feed, decimal 10
cr      equ    0dh         ;ASCII Carriage Return, decimal 13
esc     equ    1bh         ;ASCII Escape Character, decimal 27
;
jvt     equ    7effh       ;Read/Write Jump Vector Call
keyon   equ    7dffh       ;Enable Keycode Mode Call
keyof   equ    7cffh       ;Disable Keycode Mode Call
;
; This program accepts input from the console via System
; Function Call 2. Whenever a normal ASCII character is
; typed, the program takes no special action because
; SFC 2 automatically echos the character to the screen.
; However, whenever a special HP 125 key is pressed, the
; program determines which key was pressed. If it was the
; AIDS key, the program prints a message to that effect.
; If it was any other special key, the program sounds the
; internal bell. In any case, the program returns for more
; data until the 'ESC' key is pressed.
;
; When called for character input, BDOS calls the BIOS CONIN
; routine. When the character is returned, the low seven bits
; are available in the A register while the full 8-bit keycode
; is available in the B register. However, standard CP/M
; will over-write the B register prior to returning to the
; user program. Thus, via BDOS, there is no way to access the
; entire 8-bit keycode.

```

```

; To remedy this CP/M 'feature', a special HP 125 capability
; is used to alter the address of the BIOS CONIN routine.
; By using HP Sub-function Call 126, the Jump Vector Table
; entry for CONIN is stored away, then over-written with the
; address of the user-supplied input routine.
;
; The user input routine will, in fact, call the normal CONIN
; routine. When that routine returns, the 8-bit keycode will
; be in the B register, the user routine will simply move that
; full code into the A register, then return to BDOS. In this
; way, the user program can read the full 8-bit keycode
;
; First, set up the Jump Vector Table entry for CONIN with the
; address of the user-supplied routine at 'MYIN'
;
    org    0100h        ;Start of TPA and program
;
    lxi    b,jvt        ;Sub-function 126
    lxi    d,jbuf        ;Addr of buffer
    call   bdos         ;Read Jump Table into 'jbuf'
    lhld   jump         ;Get address returned in 'jbuf'
    shld   real         ;and store it as the real CONIN
    lxi    h,myin       ;Get address of my input routine
    shld   jump         ;and store it in 'jbuf'
;
    lxi    b,keyon      ;Sub-function 125
    call   bdos         ;Enable Keycode Mode
;
    mvi    a,1          ;Move a 'write' code into A
    sta    flag         ;Store it in 'jbuf'
;
    lxi    b,jvt        ;Sub-function 126
    lxi    d,jbuf        ;Addr of buffer
    call   bdos         ;Write Jump Table from 'jbuf'
;
; Now MYIN will be accessed whenever SFC 2 is executed
;
    lxi    d,msg1       ;Addr of first message
    mvi    c,sfc9       ;Function 9 - Print Console String
    call   bdos         ;Print message to console
;
loop   mvi    c,sfc1     ;Function 1 - Console Input
       call   bdos         ;Accept console input through MYIN
;
; A register contains 8-bit value
;
    cpi    esc          ;Was it an 'ESC' character
    jz     last         ;Yes - exit program

```

```

        cpi    7fh          ;Is it a normal character?
        jm     loop        ;Yes - branch
;
; Not ASCII - must be a special key
;
        cpi    aids        ;Is keycode the [AIDS] key?
        jz     hit         ;Yes - branch
;
;                               ;No - continue
        mvi    e,bell      ;Move bell character for output
        mvi    c,sfc2      ;Console Output Call
        call   bdos        ;Ring bell at console
        jmp    loop        ;Accept more data
hit     lxi    d,msg2      ;Addr of second message
        mvi    c,sfc9      ;Output Buffer Call
        call   bdos        ;Print message to console
        jmp    loop        ;go for more data
;
; MYIN routine will call the real CONIN routine, which is
; stored as part of a JMP at location 'CHRIN' in my program
;
myin    call   chrin       ;Get character via actual BIOS
        mov    a,b         ;Put 8-bit code from A into B
        ret                ;And return to BDOS
;
; LAST lines of program must restore Jump Vector Table
; and disable Keycode Mode prior to exiting
;
last    mvi    a,1         ;Set up for Jump Table 'write'
        sta    flag        ;is not necessary in this program
        lhld   real        ;Get real value of CONIN
        shld   jump        ;Store in 'jbuf'
        lxi    b,jvt       ;Sub-function Call 126
        lxi    d,jbuf      ;Addr of buffer
        call   bdos        ;Write good CONIN in Jump Table
;
        lxi    b,keyof     ;Sub-function 124
        call   bdos        ;Disable Keycode Mode
;
        lxi    d,msg3      ;Addr of message 3
        mvi    c,sfc9      ;Print String Call
        call   bdos        ;Print message to console
;
        call   boot        ;Warm start CP/M
;
; Define data area
;
msg1    db     'Test of keycode mode and [AIDS]',cr,lf,'$'
msg2    db     cr,lf,esc,'&B[AIDS] pressed!',cr,lf,'$'
msg3    db     cr,lf,'End of program',7,cr,lf,'$'

```

```

; JMP command to 'real' CONIN address
;
chrin db 0c3h ;Op Code for JMP instruction
real ds 2 ;Reserve two bytes for address
;
jvn db 3 ;Code for CONIN
flag db 0 ;Set for 'Read' at first
opcode ds 1 ;Reserve one byte for JMP code
jump ds 2 ;and two bytes for address
;
jbuf equ jvn ;Name the entire buffer 'jbuf'
end ;Assembly

```

The sample above illustrates just one way to perform a keycode mode input. An application could be designed to perform a direct call to the user routine at 5000H to perform input, rather than use SFC 1 and patch the Jump Vector Table.

Summary

This concludes the discussion of advanced features of the HP 125 Computer System. Additional material presented in the next several chapters will assist you in writing, assembling, debugging, and using applications which will be of use to you in using the system for program development.

**CP/M TRANSIENT
UTILITIES****Introduction**

In this chapter, we will look at several of the CP/M 'Transient Commands' which are supplied with the HP 125, either as part of the standard Operating System disc or as part of the programming package which includes this manual. These utility programs are extensions of the standard CP/M command set, offering greater capability than is generally available via memory-resident commands.

While many of these utilities are documented in the HP 125 Owner's Manual, the documentation provided here is generally more in-depth and specific to the Systems Programmer. Consider this as supplementary material for the material included in the standard HP 125 documentation.

Both PIP and STAT are useful in determining device and file status and in general file management. The SUBMIT and XSUB utilities allow limited 'batch' processing for unattended CP/M operation. The two HP-provided utilities, COPY and FORMAT, are also mentioned here, as well as two utilities common to most CP/M systems which are not included with the HP 125. These two programs are MOVCPM and SYSGEN, which both provide features provided elsewhere on the HP 125.

PIP

PIP, The Peripheral Interchange Program, is a general-purpose file transfer program. As documented in the HP 125 Owner's Manual, PIP allows you to move files between disc devices and several 'logical devices' as defined through IOBYTE and through HP 125 device mapping. The general format of PIP command input is:

destination = source [options]

This format of the command line can be entered at each PIP prompt, an asterisk. When all desired PIP commands have been entered, exit by either pressing RETURN or by typing a Control-C (^C) as the first character of the line. The special form:

PIP destination = source [options]

invokes PIP, executes the single command specified, and re-boots CP/M.

Table 13-1. PIP Command Examples

Command	Action
*a: =b: file.txt	Copies B:file.txt to A:file.txt
b:file.=b:old.*	Copies all files named OLD of any type to FILE with the same type.
c:=a:.*	Copies all files from A: to C:

Valid source parameters include any existing disc file names, including an optional drive identifier; file names including 'wild card' specifiers and logical and physical input device names. Destination devices include valid filenames, or default names and types based on the corresponding source file specified.

Logical and physical output device names may also be specified. Several examples of valid PIP commands are provided below in Table 13-1. A list of source and destination device names, including several special devices names are given in Table 13-2.

Table 13-2. PIP Device Names

Source Device	Logical Name	Physical Name
HP 125 Console	CON:	TTY: CRT: UC1:
CP/M Reader	RDR:	PTR: UR1: UR2:
Destination Device	Logical Name	Physical Name
HP 125 Console	CON:	TTY: CRT: UC1:
CP/M Punch	PUN:	PTP: UP1: UP2:
CP/M List	LST:	LPT: UL1:

Special Devices

There are three special device names which are supported by PIP. These are used to enhance the file transfers which can be implemented from and to devices.

'NUL:' creates a sequence of 40 'null' characters, ASCII 00H. This is a source device often useful with punch output devices, as it creates a 'header'.

'EOF:' is replaced by the ASCII 'end-of-file' character, Control-Z or 1AH. Since a Control-Z is required to signal end-of-file, this device often must be provided as the final source 'file' to mark an end of transfer.

There is only one special destination device name.

'PRN:' is a special case of the list device defined as the serial printer on Datacomm Port #2. Further, TAB characters are expanded to eight columns, leading line numbers are printed, and page length is set to 60 lines. This is functionally equivalent to the option [T8NP60] described below.

PIP Options

Often during transfers from or to devices, and during special file transfers, you will want to provide PIP with additional information concerning the transfer. This can be done by specifying one or more 'options' from the list below. Table 13-3 contains a summary of these options.

Table 13-3. PIP Options

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 Lower-Case
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 Upper Case
V	Verify Copy
W	Write to Read/Only File
Z	Zero Parity Bit

B: Block Mode

This option specifies that the source device will provide Xon/Xoff handshaking with PIP. It is normally used with paper tape input to signal when the PIP buffer is full. On the HP 125, the TPU will normally process this handshake. Since PIP will not receive the handshake, this option should not be used.

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 'CR' character, PIP will transfer only n characters until the next 'CR' is received.

This can be used to eliminate trailing line numbers common with machine generated card images, or to shorten printer listing lines 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 'formfeed' characters as data is transferred from the source to the destination.

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

G: Access Other User Group

This option, when followed by a numeric value between 0 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.

H: Verify Hex File Format

This option is used to verify that the source file is in Intel Hex format. This allows you to verify that a file of type '.HEX' is actually a Hex file. Typically, this isn't required, since a visual inspection is usually sufficient.

I: Ignore Null Hex Records

This option is also used during '.HEX' file transfers. It causes any null records, those which begin with ':00', to be ignored during a data transfer.

L: Convert to Lower Case

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

N: Add Line Numbers

This option is used to automatically append line numbers to the beginning of each line of text in the destination file. There are actually two forms of this option. 'N' causes a 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

The option is used when transferring non-ASCII files from or to devices. Since PIP expects a Control-Z to be an 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 to device as well as disc files. In case, an ASCII form feed character is inserted after 'n' lines.

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 this value, 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 actual Control-Z character. The first occurrence of the 'string' characters 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

As you will remember, 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 can also be done with STAT.

This flag makes a file 'invisible' to the CCP, and protects them from being transferred with PIP. 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 which 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 is to be expanded. This is useful in printer listings and in creating formatted text files.

U: Convert to Upper Case

This option is similar to the 'L' option, except that all lower case characters from the source will be translated to upper case. No lower case 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. When the transfer is complete, PIP re-reads both the source and destination files, checking byte by byte that the files are identical.

The HP 125 performs this 'checkread' function on every disc write, so the V option is redundant.

W: Write to Read/Only File

As you will remember, the first byte of the file type can be set (via the high-order bit) as a Read/Only file. This can also be done using STAT as described later.

In copying to an (existing) filename, 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. This is usually the case for ASCII files, but may occasionally be useful to verify parity on a series of bytes within a file.

Examples

Rather than continue talking about the various options, here are several examples which illustrate how the options are specified as well as how several of them might be used. Note the last two examples which involve the RDR: and PUN: devices. Each of these assumes that the proper device mapping escape sequence has been entered locally (see Chapter 16) before CP/M can address either of the devices. In fact, when two HP 125 are interconnected, the sequences shown below using the PUN: and RDR: device are the proper method of transferring ASCII files between systems using PIP. The system to receive the data should read from the reader first, since PIP will wait until data is available.

Table 13-4. Sample PIP File Transfers

CP/M Command	Meaning
A:=B:*. *	Copy all files from B: to A:
LST:=PRINT.TXT[SFIRST^Z]	Copy PRINT.TXT to LST: starting with 'FIRST'
B:XYZ.BCD=E:BCD.TXT[LNV]	Copy BCD.TXT to XYZ.BCD in lower case, include line numbers, and verify the copy.
B:COPY.TXT=RDR:	Accept input from RDR: and create COPY.TXT. Control-Z signals EOF.
PUN:=ORIGINAL.TXT,EOF:	Copy ORIGINAL.TXT to the CP/M punch device. Append an end-of-file.

STAT

The STAT program is a useful transient command which is included as a part of standard HP 125 CP/M. By using STAT, a user can route printer output to any printer on the HP 125, produce limited reports on disc files, or check which user numbers have active files. It also permits the user to 'mark' a file as read/only, read/write, or 'system'. These features are discussed in the HP 125 Owner's Manual, and will not be discussed at length here.

File Status

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

STAT file parm

The 'file' parameter may be a simple file reference, or may be a fully qualified name including the drive code, file name, and file type. The ASCII '\$' character delimits the file name, and any of several parameters may follow. The valid parameters are listed in Table 13-5 below.

Table 13-5. Valid STAT File Parameters

Parameter	Meaning
\$R/O	Makes the file specified Read Only. This flag remains until specifically reset using the \$R/W option.
\$R/W	Set the file specified to Read/Write.
\$SYS	Set the file specified to 'system' status. See text for details.
\$DIR	Reset the \$SYS flag.

The \$R/O flag sets the specified file(s) to 'read-only'. This options actually changes the directory entry for the file(s): remember that the high order bit on the first byte of the file type is set to signify the file is read-only. This is the way to change that bit.

The \$R/O option, since it is recorded in the file directory, is maintained through both warm- and cold-starts.

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

When the \$SYS parameter is specified, the high-order bit of the second byte of the file type is set. You will remember that 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 re-set the effect of the \$SYS parameter. This is the default state for files.

Setting Device Status

There is a special case of the STAT program which allows all files on a specific disc to be marked as 'read only'. The format is:

```
STAT id=R/O
```

The 'id' represents a specific disc drive identifier. The files will remain read-only until the next warm-start.

Device Status

STAT can also be used to report on a variety of device status. The general form used to produce these reports is:

```
STAT parm
```

The various options are illustrated in Table 13-6. These options are explained in the text which follows.

Table 13-6. Sample STAT Device Status 'parm'

Parameter	Meaning
DEV:	Reports which physical devices are assigned to CP/M logical devices.
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:	Assign the CP/M logical unit 'log:' from the physical device 'phy:'.

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.

By specifying DEV:, STAT reports the current logical to physical mapping in effect in IOBYTE. You will remember that, on the HP 125, only the LST: device is utilized in IOBYTE, so the information reported here and by the VAL: parameter may not be meaningful.

TheUSR: 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.

By selecting the DSK: parameter, optionally specifying a drive code first, the user can receive a complete report on disc statistics.

The statistics include: the drive name; the drive capacity in both records and bytes; the number of directory entries; and physical drive characteristics such as records per extent and sectors per track.

Finally, STAT permits the user to select a physical to logical mapping by using the form 'log: = phy:', where each field is a valid logical or physical device name. The valid values for physical list devices are shown when the DEV: option is used. They are also shown in Table 13-7 below.

Table 13-7. Physical List Device Names

phy:	Actual Printer Device
TTY:	HP 125 General List Device
CRT:	Internal Thermal Printer
LST:	Serial List Device on Port # 2
UL1:	HP-IB Device Address 1

Examples

Table 13-8 below illustrates a variety of STAT commands and the effects each has upon system operation. You will see that, with practice, you can use STAT to keep track of device and file status.

Notice that the device assignment feature of STAT changes the value of IOBYTE. Since the HP 125 uses IOBYTE only for the LST: device, this device assignment will not be useful for non-list devices.

Table 13-8. Sample STAT Commands

CP/M Command	Meaning
STAT DEV:	Show current IOBYTE logical to physical mappings.
STAT LST:=CRT:	Set list device to the internal thermal printer.
STAT DSK:	Show drive statistics for all active drives.
STAT A:DSK:	Show statistics for A:
STATUSR:	Show user number statistics.

Additional HP 125 Utilities

Two other utilities are included with the standard HP 125 System disc, and are fully documented in the HP 125 Owner's Manual. A brief description is also included here.

COPY

COPY is used to copy both CP/M system tracks as well as file tracks. Its operation, discussed in the HP 125 Owner's Manual, is menu-driven and quite easy to use. In addition to the operation discussed there, COPY can be invoked as a single-line CCP command.

The general format of this command line is:

```
COPY src; dst; type; verify
```

The fields are defined below.

- The 'src' field is a single character letter designator of the source disc. For example, if the source disc is in the 'A:' drive, the src field would be 'A'.
- The 'dst' field is one or more single character letter designators specifying the destination drive(s). If more than one destination is to be specified, separate the devices with commas. For example, to copy to the B: drive, 'dst' should contain a 'B'; to copy to the 'C' disc also, specify 'B,C' as 'dst'.
- The 'type' field indicates the type of copy: 'System', 'data', or 'all' may be entered. 'System' copies the system tracks only; 'data' copies the file area; 'all' copies both 'system' and 'data'.

Note that specifying 'system' or 'all' when the source disc does not contain a CP/M system, will cause an error in COPY.

- The 'verify' field specifies whether the COPY program should verify the copy is accurate by re-reading the copy and verifying the copy is correct. The 'verify' field may contain a 'yes' or 'no'.

It is highly advisable to verify all copies.

The single line COPY command completes all the menu fields, but does require operator intervention to start the copy.

FORMAT

The FORMAT program is used to perform a complete test of the flexible disc. There are actually a few sub-programs available through FORMAT.

- INIT is used to initialize the surface of new discs, or to re-initialize discs for use in other applications. INIT will totally destroy any data which may exist on a disc.

- SEEK is used to test the ability of the disc controller to read a given disc.
- VERIFY can be used to verify that all tracks and sectors of the disc are readable.



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 which are repetitive in nature, which run with little or no human intervention, and which are frequently executed. 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, one, the 'Submit' program, allows only CCP command line input to execute. On utilizing the second program, XSUB, the capability of 'Anticipated Interaction' is added. This feature permits a job stream to execute as it would if an operator were present.

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

SUBMIT

As mentioned, Submit permits command line input to the CCP. By itself, Submit is useful only for this special case. This does not mean, however, that no global parameters can be included to Submit files. Several parameters can be specified with utilities such as PIP, and these can be supplied at the time a particular batch job is initiated.

To use Submit, an ASCII text file of the '.SUB' must be prepared. This file should contain all the CCP commands which make up the 'job' to be accomplished.

A file which copies all VisiCalc data and worksheet files from the 'A:' disc to the 'B:' disc would serve a useful function. Since the filetypes to be copied are 'PRN' and 'VC' respectively, the two commands to be executed are:

```
PIP B: = A:*.PRN
PIP B: = A:*.VC
```

For our example, assume that these two lines of text have been entered into a file called 'JOB1.SUB'. This file could be created easily using the text editor. To execute these statements, enter

```
SUBMIT JOB1
```

While the SUBMIT program may be on any disc, the JOB1.SUB file will execute only from the 'A:' drive. If submit is not on the default drive, the appropriate drive code must, of course, be specified.

The job defined above, while useful, is not as practical as it could be. For example, assume your source files are on the 'C' drive rather than the 'A' drive. Fortunately, Submit allows the certain argument fields to be specified when the job is submitted.

To use this feature, the 'SUB' file can use numeric parameters preceded by a dollar sign, ASCII 36. Using ED, create the file JOB2.SUB containing the following two lines:

```
PIP $2 = $1*.PRN  
PIP $2 = $1*.VC
```

The \$2 and \$1 will be replaced by the second and first parameters after the sub file name respectively to execute a copy from 'C:' to 'B:', enter:

```
SUBMIT JOB2 C: B:
```

Submit will create a file called \$\$\$\$.SUB which will look like this:

```
PIP B: = C:*.PRN  
PIP B: = C:*.VC
```

This file, on the A: disc, will automatically execute whenever a Warmstart is performed, and will be purged when the job is finished. In the event that some error causes submit to end prematurely, this file will execute whenever a warm boot occurs. There are two ways to avoid this possible problem.

The DEL key will abort any Submit file. This permits you to manually erase the \$\$\$\$.SUB file. The other way to terminate a SUBMIT file is to boot from another disc, and erase the \$\$\$\$.SUB file from the original disc. This second procedure should only be attempted when all else fails, as a disc might be unreadable if removed during a disc access. The DEL key should always work.

XSUB

Sometimes a program will require one or more repetitive operations. For example, assume that a series of source program files include revision information on the first line of text.

When a new revision is released, each file must be edited. However, the only change required is that the old revision data is to be replaced with new data. A sample SUB file is included below which will perform this task for one particular file. The task can be generalized by repeating this sample for each file in question.

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

```
XSUB
ED $1.ASM
#A          Append all lines
B          Start of Buffer
K          Kill 1st line
I Revision: 2 ^Z      Insert new line
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 94. Remember, this is not the same as the cursor control key. On the North American keyboard, pressing the SHIFT-[6] will generate the proper code.

To make the change shown above to the file called MBK.ASM, enter the following sequence to the CCP:

```
SUBMIT JOB3 MBK
```

The entire sequence will be executed to completion. Of course, both XSUB and SUBMIT must be on a disc which is present. XSUB must be on 'A:'.

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 (press LOAD CPSYS) or a disc system reset function call.

LOAD

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 file parm
```

The 'file' parameter is a 'HEX' type file.

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 0100' directive as the first line of assembly language code.

DUMP

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 parm
```

The 'file' parameter can be of any type, including '.HEX' and '.COM'.

Other CP/M Utilities

Most CP/M Systems are delivered with two additional transient utilities: MOVCPM and SYSGEN. The HP 125 does not alter either of these programs.

MOVCPM is used to copy CP/M System tracks to create additional system discs. You will remember that the COPY program permits this, as well as the copying of files. Because CP/M offers a superset of MOVCPM features, that utility is not necessary.

SYSGEN is used primarily to re-generate CP/M for a variety of different memory sizes. Since the HP 125 is available only with 64K bytes of RAM, SYSGEN is not necessary.

Summary

As you've seen, the HP 125 offers a variety of powerful features through the utility programs provided. These utilities can be useful in providing the functions necessary in developing your own applications, making your use of time more effective.

**USING THE CP/M
TEXT EDITOR****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 HP 125 console. There are many ways to create such files, including WORD/125, 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.ext
```

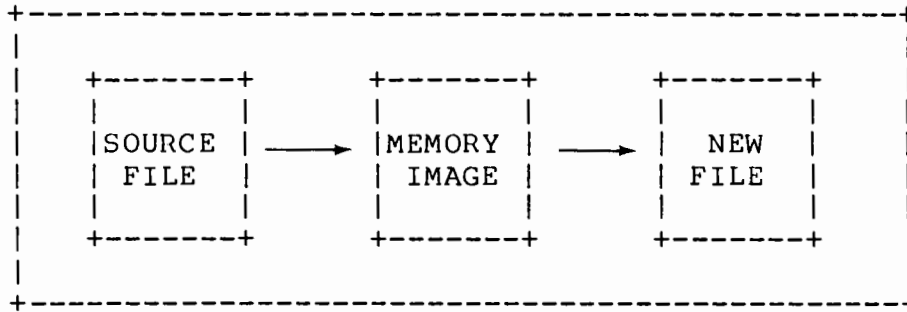
The filename and extension must be fully specified. The drive code is optional, unless the file resides on a disc other than the default drive.

If the specified file does not exist on the default or specified drive, ED will create an empty file by that name, and create a temporary work file. When the program is exited normally, the original file is renamed to have a file type of '.BAK', and the work file is renamed to reflect the original filename.

If the specified file does exist, ED will delete the '.BAK' file and create a temporary work file. As with editing a new file, when ED is exited normally, the original file is renamed to a file type of '.BAK' and the work file is renamed to the original filename.

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. Graphically this is illustrated in Table 14-1.

Table 14-1. ED Data Flow



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 14-2. Note that the 'CP' can be moved, using the commands described later in this chapter.

Table 14-2. 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 by the '^'.

ED Commands

The various commands available through ED to operate on the memory buffer are illustrated in Table 14-3 below.

Table 14-3. 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/Lower Case Shift
W	Write Lines of Text to Disc
X	Output to Temporary LIB File
Z	Sleep
<CR>	Move and Type Lines

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'.

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 '0A' 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 2 characters to skip over a CRLF.

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 CRLF 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 as the back-up file, and the work file is renamed to the original filename.

Command: F

Name: Find

The F command is used to search memory for a specified character string. The general form is:

nFstring^Z

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: if no such string is found, or the 'n'th' occurrence is not found, the 'cp' remains where it was.

The search string is terminated by a control-Z (^Z); the [RETURN] key is required and initiates the search.

To specify 'CRLF' 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:

- All lines in memory are written to disc;
- Any lines not yet loaded into memory are transferred to the output file;
- The original file becomes the back-up file;
- The output file is renamed to the original filename;
- ED starts again, using the same file name for input.

Since the disc I/O commands operate in a forward direction only (ie, 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 <CR>

Istring^Z

Istring<CR>

The first form is generally used when several lines of text are to be added to memory. All character, including CRLF, 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 CRLF 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 re-assigned as new text is inserted between existing lines.

Several special control characters are defined in the insert mode.

^H : Same as BACKSPACE, this deletes the last character typed

^L : Inserts a 'CRLF'

^M : Same as RETURN, this inserts a 'CRLF'

^R : Re-display Current Line

^U : Deletes Current Line

^X : Delete Current Line

DEL : Same as RUBOUT, this deletes and displays the previous character

REMEMBER: Insert mode considers the 'RETURN' key as just another character ('CRLF'). 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. Assuming the 'J' was successful, the 'cp' will be located at the end of 'str2'. The characters in 'str3' are not deleted.



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' CRLF characters have been deleted. This has the effect of killing lines of text.

A negative value of 'n' will cause previous characters to be erased until the 'nth' CRLF 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 CRLF. 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 at and after '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 'CRLF'. A negative value of 'n' will move 'up' in memory until 'n' CRLF characters are encountered.

The net effect is that the 'cp' will advance or go back 'n' lines. The special case of '0L' moves 'cp' to the beginning of the current line.

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 0 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 re-start 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. This is important only in remembering that the P command always begins displaying with a line number which is a multiple of 24 lines.

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 '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=0 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 have been deleted, so no '.BAK' file will exist.

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 filetype 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 effected 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 itself prints only those characters to the right of 'cp' through the CRLF. 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 together, 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 upper case. This is true, regardless of whether or not the characters are entered as lower case.

A negative sign before the 'U' will result in all text being left 'as is'. If entered in upper case, the text will remain upper case. If entered in lower case, the text will remain lower case.

Note that this command affects the text entered into the memory buffer only. The effect of upper or lower case commands is discussed in the text which 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.

This can be a convenient way to display the contents of a memory buffer exactly as it is. Also, by using one of the MODIFY modes, you can insert text which appears on the display, regardless of its source. Remember: Control-Z turns off insert mode, and will not be recognized in the MODIFY modes.

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

0V

ED prints a summary report indicating the number of bytes currently free followed by the total number of bytes available. Your filesize, 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 from the memory buffer, 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 written starting at the current position of 'cp', and 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.

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.

```
A>ED MILES.TXT <CR>           ;Start ED, creating a new
                                ;file named 'MILES.TXT' on
                                ;default drive, A:.
```

NEW FILE

```
  : *i <CR>
  1: This is line 1. <CR>       ;Enter lines of text
  2: While this is line 2. <CR>
  3: ^Z                          ;Control-Z ends insert
  : *b#t <CR>                   ;Top of buffer, type all
  1: This is line 1.           ;lines of text in buffer.
  2: While this is line 2.
  1: * <CR>                     ;cp stays at top of buffer
                                ;and <CR> types and advances
                                ;a line, similar to T command
  2: While this is line 2.     ;<CR> advances and types
  2: *i <CR>                   ;a line. Insert at cp.
  2: This is a new line between 1 and 2! <CR>
  3: ^Z                          ;Control-Z ends insert
  3: *b#t <CR>                 ;Top of buffer, type all
  1: This is line 1.
  2: This is a new line between 1 and 2!
  3: While this is line 2.

  1: *2L <CR>                   ;Advance 2 lines w/o typing
  3: *T <CR>                    ;Type a line.
  3: While this is line 2.     ;Current line 3
  3: *14CT <CR>                ;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 <CR>                       ;Top of buffer, type all
  1: This is line 1.
  2: This is a new line between 1 and 2!
  3: While this is the original line 2.
  1: *1L <CR>                   ;Advance 1 line
  2: *T <CR>                    ;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 <CR>                  ;Advance and type a line
  3: While this is the original line 2.
```



```

3: *b#t <CR> ;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.
1: *E <CR> ;End of edit. Save file.

WELCOME? ;Warm boot after exiting ED

A>ed miles.txt <CR> ;Re-edit file A:MILES.TXT
: *B#T <CR> ;Top of buffer, type all
;Nothing there - memory buffer
;is empty. Must append text.
: *#A <CR> ;Append all lines from file
1: *B#T <CR> ;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.
1: *-B <CR> ;Go to bottom of buffer
: *I <CR> ;Insert text at 'cp'
4: This is a new line inserted in upper case. <CR>
5: ^Z ;End insert with Control-Z
: *-LT <CR> ;Back a line and type
4: THIS IS A NEW LINE INSERTED IN UPPER CASE.
;This new line is upper cased
;because the insert command
;was upper cased. Note how
;the 'U' feature has been
;toggled.
4: *b#t <CR> ;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 <CR> ;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 <CR> ;Back a line, type 2
1: This is line 1.
2: This is an old line between 1 and 3!
1: *E <CR> ;End of edit. Save files
;destroying existing file

```

Summary

While this has not been a comprehensive example of using ED, it should serve as a starting point for you to experiment. As with most text editors, you will find a subset of commands which allow you to perform the editing functions you require. It is nice to know other commands are available, and you can refer to this reference to understand other commands as you need them.

**THE CP/M
ASSEMBLER****Introduction**

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/125. 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 on the HP 125. 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 offered on the HP 125 is the standard CP/M 8080 Assembler, which is used to generate machine language output for the HP 125.

Using the Assembler

The CP/M Assembler, ASM, accepts a source code input file prepared using the ED program, WORD/125 or some other text editor. The general format of the command which initiates ASM is:

```
ASM filename
```

The filename is required to have the file type '.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 type of '.HEX'. The second output file is called by the same file name, except the file is of type '.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 default 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 'B:', typing the command below will assume the source and output files are to be on 'A:'.

```
B:ASM MSB
```

To specify another drive for either the source or output files, ASM permits a three byte coded sequence to follow the filename in the command. While these three bytes appear to be a file type, remember that ASM requires type '.ASM' and the three bytes are coded instructions. These bytes direct ASM to other-than the default 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 15-1.

Table 15-1. ASM Execution Commands

COMMAND		FILE USAGE
ASM	MBK	Input: A: MBK.ASM Output: A: MBK.HEX A: MBK.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; WORD/125; or any other text editor which creates ASCII text files. Even BASIC/125 can be used by specifying the ASCII option when the file is saved!

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

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. 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, '!', 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. Neither ED nor WORD/125 do so; however BASIC/125 does include a leading line number. Generally, ASM source code does not include line numbers.

The LABEL FIELD is an optional field, and can include from one to sixteen 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 sixteen 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 not included within quotes are treated as upper case, 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 A, and the most commonly used are discussed later in the chapter. 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 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> <LF>' are considered to be part of the comment field and are ignored by ASM. This means that a comment can only be included following the last statement on a multiple-statement line.

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 chapter.

Forming the Operand

The operand field, as mentioned above, may contain a label, an expression, a data byte, or an expression. Which is 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 operand, 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 8080 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 or of 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 15-2. If no radix indicator is specified, the value is assumed to be base 10, decimal.

Table 15-2. 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 '0'. 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 10, 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 imbedded in a numeric value to improve readability. Refer to Chapter 2 earlier in this manual for additional discussion of non-decimal number systems.

Table 15-3 contains examples of valid numeric constants in the permitted bases. Notice that lower case radix indicators are treated as if they were upper case.

Table 15-3. Sample Valid Numeric Constants

3738	4382D	1111\$0101\$0000\$0110
3738h	0BCDH	177\$777Q 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 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 quotes side by side. Examples of valid string constants are illustrated in Table 15-4.

Table 15-4. Sample String Constants

'A'	'MK'	'MSB'	'Hello'
''''''	''If it works, don't fix it!''		
'The HP 125 Personal Business 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 15-5 below. 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 15-5. Valid Arithmetic Operators

OPERATION	EFFECT
op1 + op2	Unsigned arithmetic sum of op1 and op2
op1 - op2	Unsigned difference between op1 and op2
+ op2	Unary plus equivalent to op2
- op2	Unary minus equivalent to (0 - op2)
op1 * op2	Unsigned magnitude multiplication
op1 / op2	Unsigned magnitude division
op1 MOD op2	Remainder function of (op1 / op2)
NOT op2	Logical complement of op2
op1 AND op2	Bit-by-bit logical AND
op1 OR op2	Bit-by-bit logical OR
op1 XOR op2	Bit-by-bit logical exclusive OR
op1 SHL op2	Result of shifting op1 LEFT op2 bits
op1 SHR op2	Result of shifting op1 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 0FFFFH. Instead, an expression which zeros the high order bits can be used: 'ADI ((-1) AND OFFH)'. The 'OFFH' will force the high 8 bits to zero, making the expression acceptable.

As with most higher-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 of 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 15-6 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 15-6. 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 which assume special significance in the operand field. These reserved words are shown in Table 15-7 below, and include the names of each of the 8080 registers and the special ASM program counter.

Table 15-7. 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 can be 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. This will be demonstrated in the example later in the chapter.

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 15-8.

Table 15-8. 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 which 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 8 bits. This represents 0 through 255 decimal, or 0 through 0FF hex.

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 15-9.

Table 15-9. Sample Use of the DB Directives

CR	DB	13	;'Return' is ASCII 13
LF	DB	10	;'Line feed' is ASCII 10
HOMEUP	DB	27,'h'	;'ESC h' is home up
MESG	DB	'This is a line of text'	
LOG	DB	LF AND OFFH	

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 8-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, DS allows an expression, a numeric value, or a one- or two-byte string. Table 15-10 illustrates some valid examples of DS.

Table 15-10. Sample DW Directives

MAX	DW	0FFFFH	;Maximum 16 bit value
INTLS	DW	'MK'	;Two ASCII bytes
CRLF	DW	13 * 256 + 10	
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.

The use of the IF, ENDIF, and SET is illustrated along with the other directives at the end of the chapter.

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 used 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.

This is illustrated in the example included at the end of this chapter.

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 0100 hex.

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.

Sample 8080 Source Program

Operation Codes

While assembler directives are used within the instruction field in 8080 source code, it is the operation code mneumonics which actually direct the processor in performing any specific task. The mneumonics 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. Some of the more common are listed in the bibliography in Appendix C.

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

The conventions and standards used are summarized in Table 15-11.

Table 15-11. 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 0 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 with (A)
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 address in register pair
STAX rp	Store A into address in 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 port 'data'
OUT data	Store A register to CPU port 'data'
XTHL data	Exchange data from A register with port
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.
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.

Control Instructions

The remaining instructions specify direct control of the processor and its interrupt system. Except for the 'NOP' instruction, they should not ordinarily be used on the HP 125.

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

Assembler Execution

When the ASM program executes, it will display several message lines at the HP 125 console. The message will look something like the illustration in Table 15-12 below.

Table 15-12. Sample ASM Messages

```
+-----+
|      CP/M ASSEMBLER - VER 2.0      |
|      021F                          |
|      022H USE FACTOR                |
|      END OF ASSEMBLY                |
+-----+
```

The value on the second line, 021F in the example above, is the address of the next free location in memory following the assembled program. It is a hexadecimal number.

The 022H 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 000H through 0FFH. In the example above, 022H 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 byte on a line should either be blank or contain one of the error codes described later in this chapter. Refer to that section for an explanation of the error codes.

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' directives, 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 which 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.

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 program 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 filetype '.ASM' and that any 'filetype' specified indicates the disc code for ASM files.

Source Code Error Messages

If the ASM program ever finds a line it cannot properly syntax, it will report an error condition. This will be displayed both at the screen and in the '.PRN' file. The first byte 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, or a data error has been generated.
- 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 with 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.

Summary

The ASM program offers a way for the programmer to write 8080 source code programs. While 'macros' and other special options are not available, ASM can be useful in accessing all the features of the HP 125 System.

USING THE DYNAMIC DEBUGGING TOOL

Introduction

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 8080 registers; modify those registers or memory locations; and even 'assemble' new source code 'in-line'.

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.typ
```

Note that the full filename and file type must be specified. Only files of type 'HEX' or 'COM' may be specified.

Using DDT

Once the debugger is memory-resident, the program will display the revision number of the DDT program, which should be 2.0 at first release of the HP 125. If a filename 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 0100H.

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.

DDT Commands

The commands available are illustrated and briefly described in Table 16-1. They are described in detail following that table.

Table 16-1. DDT Command Summary

A	Assemble a program statement at the given address
D	Display contents of memory in hex
F	Fill memory with specified constant
G	Begin execution ('GO') of program in memory
I	Insert a filename 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
X	Examine the CPU registers

A: Assemble

DDT allows immediate assembly of Intel 8080 instructions by use of this command. The format is:

Axxxx

The user is prompted for a line 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 is entered at the HP 125 console. Note that the assembler, like the dis-assembler to be discussed later, is a part of DDT which can be overlaid by user code. This procedure, described later in this appendix, 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 format of the display is illustrated in Table 16-2. 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 '.'.

Table 16-2. Sample Display Command Listing

```

+-----+
| A>ddt                                     |
| DDT VERS 2.0                             |
| -d100,200                                |
| 0100 01 B6 0F C3 3D 01 43 4F 50 59 52 49 47 48 54 20 ....COPYRIGHT |
| 0110 28 43 29 20 31 39 37 38 2C 20 44 49 47 49 54 41 (C) 1978, DIGITA |
| 0120 4C 20 52 45 53 45 41 52 43 48 20 20 20 20 20 20 L RESEARCH      |
| 0130 44 44 54 20 56 45 52 53 20 32 2E 30 24 31 00 02 DDT VERS 2.0$1.. |
| 0140 C5 C5 11 30 01 0E 09 CD 05 00 C1 21 07 00 7E 3D ...0.....!..|= |
| 0150 90 57 1E 00 D5 21 00 02 78 B1 CA 65 01 0B 7E 12 .W...!..x..e...- |
| 0160 13 23 C3 58 01 D1 C1 E5 62 78 B1 CA 87 01 0B 7B .#.X...bx.....{ |
| 0170 E6 07 C2 7A 01 E3 7E 23 E3 6F 7D 17 6F D2 83 01 ...z...#.o}.o... |
| 0180 1A 84 12 13 C3 69 01 D1 2E 00 E9 2A 7C 1D EB 0E .....i.....*!... |
| 0190 1A CD 67 1B C9 3E 0C D3 01 3E 08 D3 01 DB 01 07 ..g.>...>..... |
| 01A0 07 07 1F DA A9 08 C3 9D 08 DB 03 E6 7F C9 21 83 .....!... |
| 01B0 1D 70 2B 71 2A 82 1D 44 4D CD A1 07 0E 3A CD 86 .p+q*.DM.....:.. |
| 01C0 07 0E 20 CD 86 07 3A 5F 1D 32 84 1D 3A 60 1D 21 .. ...:2...:;! |
| 01D0 84 1D BE DA F4 08 21 DE 1C 3A 84 1D BE D2 ED 08 .....!-:..... |
| 01E0 2A 84 1D 26 00 01 DF 1C 09 4E CD 86 07 21 84 1D *...&....N...!.. |
| 01F0 34 C2 CC 08 21 DE 1C 36 00 01 4A 01 00 C4 AD 13 4...!..6..J..... |
| 0200 C3 . |
+-----+

```

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 at the HP 125 Console.

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,yyy,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, cause DDT to begin execution. There are three 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 cause 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 0000H (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 re-set 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 filename to be inserted into the default FCB at address 005CH. This permits the file to be 'Read' using the 'R' command which is described later. The format is:

Ifilename.COM
Imyfile.HEX

The filename 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 005CH. The 'S' command allows this substitution.

If the file is to be loaded using the 'R' command, the type 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 type is arbitrary and optional.

L: Disassemble Memory

This command causes the contents of memory to be 'dis-assembled' 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 dis-assembled 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 dis-assembly at 'xxxx' and stops after dis-assembling the instruction at 'yyyy'.

If a location contains a value which 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 dis-assembler.

As with the 'A' instruction, that portion of DDT which contains the dis-assembler may be overlaid so that larger programs may be tested. This technique is described later in this chapter.

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 005CH. 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 the file being loaded.

If the file type is '.HEX', the load address is assumed to be 0000H, and the load address is taken from the Intel format Hex file.

If the file type is '.COM', the load address is assumed to be 0100H 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 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 [RETURN] 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
Tn

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 [DEL] key.

DDT will intervene between each instruction in the user program, so program execution can be up to 500 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 Trace 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 HC 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 dis-assembled operation code of the instruction about to be executed at the current value of P, the program counter. The second form allows the 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 [RETURN] 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.



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 dis-assembler, 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/dis-assembler. 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.

Notes on the DDT Assembler

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.

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 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. Convert this two-digit hex number to its decimal equivalent using the techniques described in Chapter 2.

If the low two digits in the original address were '00', 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 100H-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 0100H 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

```
GO
```

CP/M performs a 'warm-start', and control is returned to the CCP. However, my program is still in memory starting at 0100H! 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 0 !.COM
```

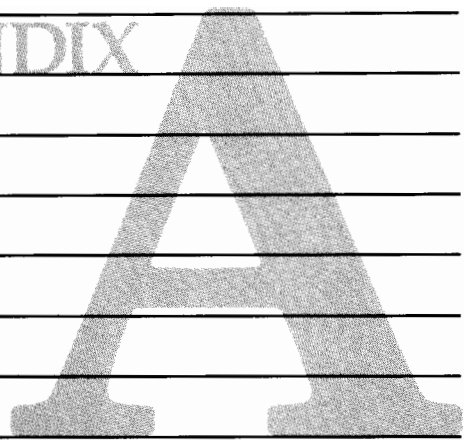
This causes a file named '!.COM' to be created with zero bytes.

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 0100H. The net effect is that the last program which 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 120: 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!

Summary

DDT is a useful tool in testing assembly language program files. While its use is not a replacement for thoughtful system analysis, it can help test limit conditions or perform spot checks on various CP/M functions. It can be an invaluable aid in designing and building custom software packages.

**SYSTEM
REFERENCE
TABLES**

ASCII CHARACTER CODES

NUL	Null	DC1	Device control 1
SOH	Start of heading	DC2	Device control 2
STX	Start of text	DC3	Device control 3
ETX	End of text	DC4	Device control 4
EOT	End of transmission	NAK	Negative acknowledge
ENG	Enquiry	SYN	Synchronous idle
ACK	Acknowledge	ETB	End of transmission block
BEL	Bell, or alarm	CAN	Cancel
BS	Backspace	EM	End of medium
HT	Horizontal tabulation	SUB	Substitute
LF	Line feed	ESC	Escape
VT	Vertical tabulation	FS	File separator
FF	Form feed	GS	Group separator
CR	Carriage return	RS	Record separator
SO	Shift out	US	Unit separator
SI	Shift in	SP	Space
DLE	Data link escape	DEL	Delete

ASCII CHARACTER SET
SEVEN BIT CODE

MSB → LSB ↙		0	1	2	3	4	5	6	7
		000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SP	0	@	P	'	P
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	\$	4	D	T	d	t
5	0101	ENG	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(8	H	X	h	x
9	1001	HT	EM)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[k	{
C	1100	FF	FS	,	<	L	\	l	
D	1101	CR	GS	-	=	M]	m	}
E	1110	SO	S	.	>	N	^	n	~
F	1111	SI	VS	/	?	O	_	o	DEL

MSB and LSB represent hexadecimal values. To determine the hex representation of a character, the hex most significant byte is above the top, while the least significant byte is on the left. For example, K is a 4B hex.

8080 CPU INSTRUCTIONS IN OPERATION CODE SEQUENCE

OP CODE	MNEMONIC	OP CODE	MNEMONIC	OP CODE	MNEMONIC	OP CODE	MNEMONIC	OP CODE	MNEMONIC	OP CODE	MNEMONIC
00	NOP	2B	DCX H	56	MOV D,M	81	ADD C	AC	XRA H	D7	RST 2
01	LXI B,D16	2C	INR L	57	MOV D,A	82	ADD D	AD	XRA L	D8	RC
02	STAX B	2D	DCR L	58	MOV E,B	83	ADD E	AE	XRA M	D9	—
03	INX B	2E	MVI L,D8	59	MOV E,C	84	ADD H	AF	XRA A	DA	JC Adr
04	INR B	2F	CMA	5A	MOV E,D	85	ADD L	B0	ORA B	DB	IN D8
05	DCR B	30	SIM	5B	MOV E,E	86	ADD M	B1	ORA C	DC	CC Adr
06	MVI B,D8	31	LXI SPD16	5C	MOV E,H	87	ADD A	B2	ORA D	DD	—
07	RLC	32	STA Adr	5D	MOV E,L	88	ADC B	B3	ORA E	DE	SBI D8
08	—	33	INX SP	5E	MOV E,M	89	ADC C	B4	ORA H	DF	RST 3
09	DAD B	34	INR M	5F	MOV E,A	8A	ADC D	B5	ORA L	E0	RPO
0A	LDAXB	35	DCR M	60	MOV H,B	8B	ADC E	B6	ORA M	E1	POP H
0B	DCX B	36	MVI M,D8	61	MOV H,C	8C	ADC H	B7	ORA A	E2	JPO Adr
0C	INR C	37	STC	62	MOV H,D	8D	ADC L	B8	CMP B	E3	XTHL
0D	DCR C	38	—	63	MOV H,E	8E	ADC M	B9	CMP C	E4	CPO Adr
0E	MVI C,D8	39	DAD SP	64	MOV H,H	8F	ADC A	BA	CMP D	E5	PUSH H
0F	RRC	3A	LDA Adr	65	MOV H,L	8G	SUB B	BB	CMP E	E6	ANI D8
10	—	3B	DCX SP	66	MOV H,M	91	SUB C	BC	CMP H	E7	RST 4
11	LXI D,D16	3C	INR A	67	MOV H,A	92	SUB D	BD	CMP L	E8	RPE
12	STAX D	3D	DCR A	68	MOV L,B	93	SUB E	BE	CMP M	E9	PCHL
13	INX D	3E	MVI A,D8	69	MOV L,C	94	SUB H	BF	CMP A	EA	JPE Adr
14	INR D	3F	CMC	6A	MOV L,D	95	SUB L	C0	RNZ	EB	XCHG
15	DCR D	40	MOV B,B	6B	MOV L,E	96	SUB M	C1	POP B	EC	CPE Adr
16	MVI D,D8	41	MOV B,C	6C	MOV L,H	97	SUB A	C2	JNZ Adr	ED	—
17	RAL	42	MOV B,D	6D	MOV L,L	98	SBB B	C3	JMP Adr	EE	XRI D8
18	—	43	MOV B,E	6E	MOV L,M	99	SBB C	C4	CNZ Adr	EF	RST 5
19	DAD D	44	MOV B,H	6F	MOV L,A	9A	SBB D	C5	PUSH B	F0	RP
1A	LDAXD	45	MOV B,L	70	MOV M,B	9B	SBB E	C6	ADI D8	F1	POP PSW
1B	DCX D	46	MOV B,M	71	MOV M,C	9C	SBB H	C7	RST 0	F2	JP Adr
1C	INR E	47	MOV B,A	72	MOV M,D	9D	SBB L	C8	RZ	F3	DI
1D	DRC E	48	MOV C,B	73	MOV M,E	9E	SBB M	C9	RET Adr	F4	CP Adr
1E	MVI E,D8	49	MOV C,C	74	MOV M,H	9F	SBB A*	CA	JZ	F5	PUSH PSW
1F	RAR	4A	MOV C,D	75	MOV M,L	A0	ANA B	CB	—	F6	ORI D8
20	RIM	4B	MOV C,E	76	HLT	A1	ANA C	CC	CZ Adr	F7	RST 6
21	LXI H,D16	4C	MOV C,H	77	MOV M,A	A2	ANA D	CD	CALL Adr	F8	RM
22	SHLD Adr	4D	MOV C,L	78	MOV A,B	A3	ANA E	CE	ACI D8	F9	SPHL
23	INX H	4E	MOV C,M	79	MOV A,C	A4	ANA H	CF	RST 1	FA	JM Adr
24	INR H	4F	MOV C,A	7A	MOV A,D	A5	ANA L	D0	RNC	FB	EI
25	DCR H	50	MOV D,B	7B	MOV A,E	A6	ANA M	D1	POP D	FC	CM Adr
26	MVI H,D8	51	MOV D,C	7C	MOV A,H	A7	ANA A	D2	JNC Adr	FD	—
27	DAA	52	MOV D,D	7D	MOV A,L	A8	XRA B	D3	OUT D8	FE	CPI D8
28	—	53	MOV D,E	7E	MOV A,M	A9	XRA C	D4	CNC Adr	FF	RST 7
29	DAD H	54	MOV D,H	7F	MOV A,A	AA	XRA D	D5	PUSH D		
2A	LHLD Adr	55	MOV D,L	80	ADD B	AB	XRA E	D6	SUI D8		

D8 = constant, or logical/arithmetic expression that evaluates to an 8 bit data quantity.

Adr = 16-bit address

D16 = constant, or logical/arithmetic expression that evaluates to a 16 bit data quantity

Escape Code Summary

The following is a summary of the escape codes available on the HP 125 system. Refer to the appropriate chapter for a detailed explanation of the function of each sequence.

Two character escape sequences:

Sequence	Description/Chapter
ESC 0	Print content of display memory to selected printer.
ESC 1	Set tab at cursor column.
ESC 2	Clear tab.
ESC 3	Clear all tabs.
ESC 4	Set left margin.
ESC 5	Set right margin.
ESC 9	Clear all margins.
ESC @	Delay one second.
ESC A	Cursor Up.
ESC B	Cursor Down.
ESC C	Cursor Right.
ESC D	Cursor Left.
ESC E	Hard reset TPU.
ESC F	Home Down.
ESC G	Return.
ESC H	Home Up cursor.
ESC I	Tab.
ESC J	Clear Display.
ESC K	Clear Line.
ESC L	Insert line.
ESC M	Delete line.
ESC P	Delete character.
ESC Q	Turn on insert character mode.
ESC R	Turn off insert character mode.
ESC S	Roll Up Display.
ESC T	Roll Down Display.
ESC U	Next Page.
ESC V	Previous Page.
ESC Y	Display Functions On.
ESC Z	Display Functions Off.

Two character escape sequences (continued):

Sequence	Description/Chapter
ESC ^	Send Primary Terminal Status.
ESC ~	Send Secondary TPU status.
ESC `	Relative Cursor Sense.
ESC a	Absolute Cursor Sense.
ESC b	Enable Keyboard.
ESC c	Disable Keyboard.
ESC d	Enter Line.
ESC f	Modem Disconnect.
ESC g	Soft Reset TPU.
ESC h	Home Up Cursor.
ESC i	Back Tab Cursor.
ESC j	Display Softkey Menu.
ESC k	Exit Softkey Menu.
ESC x	Perform datacomm self-test.
ESC z	Perform TPU self-test.



Multi-character escape sequences:

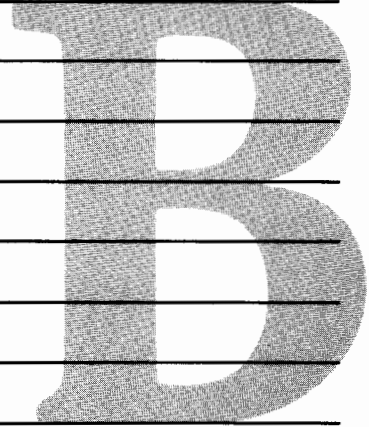
ESC & a	Cursor Addressing.
<col number> c	-> Set column
<row number> r	-> Set absolute row
<row number> y	-> Set relative row
ESC & d	Display Enhancement.
@	-> Clear enhancement
<A B... N O>	-> Set enhancement
ESC & f	Define User Softkeys.
< 0 1 2> a	-> Set key attribute
< len > d	-> Label length
<1 2 ... 8> E	-> Execute selected key
<1 2 ... 8> k	-> Select key to be defined
< len > l	-> Definition length
< string >	-> Display label definition
< string >	-> Softkey definition
ESC & i	Device Assignment.
<bufr numbr> d	-> Destination code
<bufr numbr> s	-> Source code
<mode numbr> m	-> Operating mode
ESC & j	Softkey Display Control.
@	-> Turn off softkey labels
A	-> Turn on softkey labels
B	-> Turn on [USER KEYS] labels
R	-> Unlock label set
S	-> Lock label set

Multi-character escape sequences (continued):

ESC & k		Set Terminal Straps.
< 0		1 > a -> Auto line feed mode
< 0		1 > c -> Caps lock mode
< 0		1 > d -> Bell
< 0		1 > j -> Screen refresh frequency
< 0		1 > l -> Local echo
< 0		1 > m -> Modify All mode
< 0		1 > n -> Space overwrite latch
< 0		1 > p -> Caps mode
< 0		1 > q -> Key click
< 0		1 > r -> Remote mode
ESC & p		Printer Device Control.
		^ -> Device status request
		b -> Print one line
		f -> Print page
		m -> Print memory
		<cntrl num> c -> Data log function
		(cntrl num) =11 -> Log bottom
		=12 -> Log top
		=13 -> Disable logging
		=17 -> Enable report print
		=18 -> Enable metric print
		=19 -> Disable report/metric print
ESC & q		TPU Configuration Control.
< 0		1 > L -> Lock configuration
ESC & s		Terminal Strap Control.
< 0		1 > a -> Transmit function key sequences
< 0		1 > b -> Space overwrite enable
< 0		1 > c -> Cursor wrap-around
< 0		1 > g -> Inhibit DC1 handshake
< 0		1 > h -> Inhibit DC2 handshake
< 0		1 > l -> Inhibit self-test
ESC * s ^		Terminal Identification.

KEYCODES

Hex Keycode	Keyboard Key	Hex Keycode	Keyboard Key
80	Home Up	A3	Clear All Tabs
81	Home Down	A4	Set Left Margin
82	Roll Down	A5	Set Right margin
83	Roll Up	A6	Clear All Margins
84	Cursor Up	A7	Config Menu
85	Cursor Left	A8	Enhance Character
86	Cursor Right	A9	Enhance Linergins
87	Cursor Down	AA	Remove Enhancement
88	Insert Char. Off	AB	Start Auto Repeat
89	Delete Character	AC	Stop Auto Repeat
8A	Use Softkey Menu	AD	Modify Line
8B	Enter	AE	Modify All
8C	Left Tab	AF	Terminal Test
8D	Unused	B0	Modem Dial
8E	Return	B1	Display Functions
8F	Blank Softkey Row	B2	Auto Line Feed
90	Clear Line	B3	Remote Mode
91	Break	B4	Local Op Sys Mode
92	Clear Display	B5	Log Bottom
93	Unused	B6	Log Top
94	Insert Char. On	B7	Report Print
95	Unused	B8	Metric Print
96	Unused	B9	Advance Page
97	Delete Line	BA	Advance Line
98	[AIDS]	BB	Copy All
99	[MODES]	BC	Copy Page
9A	User Softkeys	BD	Copy Line
9B	Next Page	D8	User Key [f1]
9C	Previous Page	D9	User Key [f2]
9D	Unused	DA	User Key [f3]
9E	Unused	DB	User Key [f4]
9F	Unused	DC	User Key [f5]
A0	Start Column	DD	User Key [f6]
A1	Set Tab	DE	User Key [f7]
A2	Clear Tab	DF	User Key [f8]



**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

An overview of CP/M, with both the user and programmer in mind. Includes a review of many software programs which can be used with CP/M.

Murtha, Syeven, and Waite, Mitchell. CP/M Primer. Indianapolis: Howard W. Sams and Company, Inc.

This book presents the beginner with an introduction to CP/M, written for the first-time CP/M user.

Zaks, Rodney. 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 Programming. 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

APPLICATION PROGRAM INSTALLATION

Introduction

Hewlett-Packard Applications Packages are made available on a 'Master Disc'; software must be 'installed' onto a 'Work Disc' in order to execute an application. Part of the installation includes entering a key label on the 'Welcome Menu' for the newly installed application.

To execute custom designed software from the same menu, you must 'install' the desired package in the same manner. This appendix will describe how that can be done.

The Welcome program, which performs the application installation, requires the 'Work Disc' to be mounted on drive 'A:', and the 'Master Disc' to be on drive 'B:'.

To begin an installation procedure, load the operating system so the Welcome Menu is displayed. If no 'WELCOME.COM' file exists on your 'Work Disc', you can use any file of the same name from any Hewlett-Packard Application Master.

For example, assume you have newly formatted a disc using FORMAT. Use the COPY program to move the SYSTEM from another disc to this new disc.

Next, load CP/M from this new work disc. No files should exist, CCP will display:

```
WELCOME?  
A>
```

Use PIP to copy all files from the master CP/M Operating System disc to the newly created work disc. Execute the program by typing 'WELCOME' or by pressing the 'LOAD OP SYS' softkey. You should see the 'Welcome Menu' on the screen, with only two of the eight keys defined. Key [f1] will lead to the HP 125 utilities, and key [f8] provides access to the CP/M Operating System.

When you are ready to install an application, place the application 'master' disc into drive 'B:', and press SHIFT-CTRL-[2]. The Welcome menu will be replaced by the 'Installation Menu' illustrated in Table C-1.

By pressing 'INSTALL APPL', the appropriate files from the master disc in 'B:' will be transferred to your work disc. This procedure is described later in this appendix.

The key labeled 'DISABLE WELCOME' can be used to prevent the Welcome Menu from executing when you EXIT most application programs. In fact, the WELCOME file will execute only when a program terminates with a cold-boot. No program which exits with a warm-boot will load the WELCOME Menu.

Table C-1. Sample Installation Menu

APPLICATION INSTALLATION MODE

Directions ---

- To install an application insert the application disc into the B disc drive and press INSTALL APPL.
- To exit installatin mode press EXIT. Modifications to the Application Interface will be saved automatically.
- To disable the WELCOME file auto-load feature press DISABLE WELCOME. Auto load will be disabled until next reload of the operating system.

Available space on disc A = 47

INSTALL APPL **EXIT** 1 1 **DISABLE WELCOME**

Application Installation

Once in the installation menu, you are ready to install your custom-written software. However, some special planning must be made to prepare your application for installation.

The WEL File

On the Master disc, the installation program must find a file with filetype '.wel'. This file contains the information necessary to complete the installation. Notice that the file type must be lower case!

The format of this '.wel' file, explained below, is illustrated in Table C-2.

Table C-2. Installation File Field Locations

Location	Field Description
0H -> 4FH	Installation Header
50H -> 50H	File Count
80H -> 8FH	Softkey Label
90H -> 91H	Application Size
92H -> 92H	Command String Length
93H -> 0E2H	Command String



Installation Header

This field, 80 bytes in length, contains the message which is displayed during the software installation. Normally, this message should indicate the application name, a revision number, and any other important information. The message is limited to 80 characters maximum.

File Count

This field is a single byte binary integer which indicates the number of files or programs which are to be copied to the 'A:' disc. This count does not include the '.wel' file.

If the number of files indicated here are not transferred, the installation is not considered successful and the softkey label will not be completed.

Softkey Label

The 16 bytes to be displayed in the softkey label field are stored in this field. The definition will appear exactly as shown here, so be certain to center the printing characters and to provide ASCII 'space' characters, 20H, in other positions.

Application Size

This field is a two byte integer field which specifies the number of 1K byte units of disc are required for installation of the complete application.

This value is stored with the least significant byte first, and the most significant byte second. This is consistent with the manner in which the Z80 processor stores integer 16-bit data.

Command String Length

This field is a single byte integer value which specifies the length of the command string used to invoke the main entry program of the application.

Command String

This field contains the string of ASCII characters which are used to invoke the application program. This is accomplished using Extended Function Call 122.

Note that, unlike user CCP input, a command may include one or more lower case characters. No 'upper case shift' takes place.

Creating the '.WEL' File

The installation file described above can be created in several ways, from 'writing a program' to creating a text file. Here, you will see a technique which offers a quick, easy to edit way of creating the file using DDT, the CP/M debugger utility.

Make sure that there is no 'WELCOME.COM' file present on your system disc in Drive 'A:'. Do not erase any such file there: use the 'REN' command to temporarily disable automatic execution of the Application Menu program.

Now that you have disabled WELCOME, execute DDT by loading the program into memory.

Next, fill the first 100H bytes of user memory with the ASCII 'space', a value of 20H. Do this by typing 'F 100,200,20' to DDT.

By using the 'S' command, substitute individual memory locations with the ASCII or binary values required, depending on your need. If you have any question on how this is done, refer to the example below for additional assistance.

Once you are satisfied with the contents of memory between address 0100H and 0200H, use the 'g0' command to warm-boot CP/M. Of course, memory will not be disturbed since you have no WELCOME file on the 'A:' drive: the data you want to save as the '.wel' file remains in memory.

Save one page of memory to disc, naming the file with a type of '.wel'. This creates the data file and writes it to disc. You are now ready to proceed in preparing the master disc.

Example

Assume you are preparing an installation file for an application called 'FINANCE'. Further, assume the package includes 12 programs, requires 200K bytes of disc for storage, and the command to start the package is 'finpack'.

The following is a DDT session which prepares such a file.

```
DDT
A>B:DDT

DDT VERS 2.0
F100,200,20      Fill memory with ASCII 'space'
-S100           Start substituting memory at 100H
0100 20 46      ASCII codes for 'FINANCE/125'
0101 20 49
0102 20 4E
0103 20 41
0104 20 4E
0105 20 43
0106 20 45
0107 20 2F
0108 20 31
0109 20 32
```



```

010A 20 35
010B 20
010C 20
010D 20
010E 20
010F 20
0110 20 52
0111 20 65
0112 20 76
0113 20 2E
0114 20
0115 20 31
0116 20 2E
0117 20 30
0118 20
0119 20
011A 20
011B 20
011C 20
011D 20 1B
011E 20 26
011F 20 64
0120 20 42
0121 20 4D
0122 20 42
0123 20 4B
0124 20 .
-S150
0150 20 0C
0151 20 .
-S180
0180 20 46
0181 20 49
0182 20 4E
0183 20 41
0184 20 4E
0185 20 43
0186 20 45
0187 20 2F
0188 20
0189 20
018A 20 31
018B 20 32
018C 20 35
018D 20
018E 20
018F 20
0190 20 C8
0191 20 00
0192 20 07
0193 20 66
0194 20 69
0195 20 6E
0196 20 70

```

Five ASCII spaces

ASCII codes for 'Rev. 1.0'

Five ASCII spaces

ASCII codes for inverse video 'MBK'
(Send ESC&dB first)

Done with Header Message
Enter number of files in package
This is '12' decimal

Enter Softkey Label
ASCII 'FINANCE/ 125'

Storage Size goes here (200K decimal)

Command Length is 7 bytes
ASCII Command 'finpack'

```

0197 20 61
0198 20 63
0199 20 6B
019A 20 .
-D100
0100 46 49 4E 41 4E 43 45 2F 31 32 35 20 20 20 20 20 FINANCE/125
0110 52 65 76 2E 20 31 2E 30 20 20 20 20 20 1B 26 64 Rev. 1.0 .&d
0120 42 4D 42 4B 20 20 20 20 20 20 20 20 20 20 20 BMBK
0130 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0140 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0150 0C 20 20 20 20 20 20 20 20 20 20 20 20 20 20 .
0160 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0170 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0180 46 49 4E 41 4E 43 45 2F 20 20 31 32 35 20 20 20 FINANCE/125
0190 C8 00 07 66 69 6E 70 61 63 6B 20 20 20 20 20 20 ...finpack
01A0 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
01B0 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20

```

```
-g0 Warm Boot CP/M
```

```
A>SAVE 1 FIN.WEL and save memory to file
```

```
A> Application File now ready!
```

Notice that this command will create a file which has an upper case file type. You may use a program such as BASIC/125 which allows you to specify a lower case file type (using the NAME AS command). You may also wish to write a small utility which will allow you to rename the 'WEL' file.

Master Files

The 'wel' file contains a variety of information for the installation menu, but does not indicate which files are to be copied to the work disc. However, the transfer always occurs from drive 'B:', which contains the master, to 'A:', which contains the Work Disc.

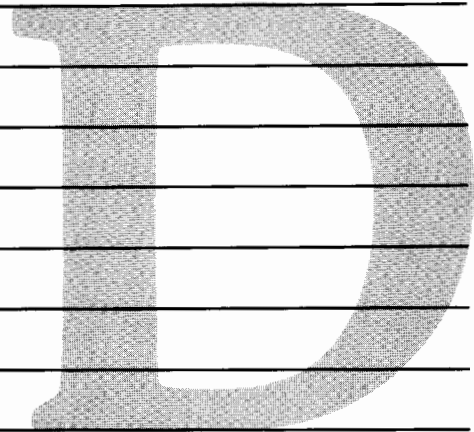
Any file on the Master Disc which is to be copied must be 'marked'. Specifically, the high-order bit on the first byte of a master file name must be set to '1' if it is to be copied.

For example, if a master file is called 'MASTER', the first byte in the directory entry must be set. The ASCII character 'M' is represented by a hex '4D' (see Appendix A). This must be changed to 'CD' in order to be transferred.

This can be done by preparing a custom written 'prep' utility which will read an FCB, add 80H to the first byte, and re-write the same FCB, possibly using function 23, 'Rename File'.

Summary

You have seen how to use the HP-supplied 'Welcome Menu' to install custom-written software. While Hewlett-Packard Applications use still other protection mechanisms, by following these guidelines you will be able to include your applications with the standard Welcome Menu. This will make your software as easy to access as HP applications.



DISC ORGANIZATION



Introduction

Before a disc can be used on the HP 125, it must be formatted. There are three types of disc format supported: HP format on 5 1/4" media from the HP 82901 disc drive; HP format on 8" media from the HP 9895 disc drive; and 'IBM 3740 format' on 8" media from the HP 9895 or other 'interchange format' compatible disc drives.

Any media which corresponds to one of the three formats specified above can conceivably be used on the HP 125. However, to use standard CP/M utilities (including the CCP), a disc must also contain a valid CP/M directory and conform to other basic assumptions about the organization of data on the media. These assumptions vary slightly between the three types of media, and will be explained in greater depth later in this appendix.

Occasionally, you will have data or programs on a non-CP/M disc which is nonetheless written in a hardware format supported by the HP 125. This would be the case when a particular media has been created by an HP-85, or by a key-to-disc '3740' type system.

When this is the case, the BDOS calls discussed in Chapter 9 can not be used: they require a CP/M file system. Instead, you will need a program which accesses BIOS directly through the Jump Vector Table as discussed in Chapter 12. By using these BIOS calls, you will be able to position the disc heads to a specific location on the disc, and read or write 128 bytes of data.

The method used to perform this BIOS level I/O will be discussed later in this appendix. Before doing so, however, let's take a look at the structure of data on each type of media.

Disc Data Organization

The disc is organized into several concentric recording tracks: the number of tracks per disc varies between the three types of disc format supported. Each track can be sub-divided into physical sectors of 256 bytes each, with a small 'inter-record gap' between each sector for use by the disc controller.

Part of the function performed by FORMAT is to organize the disc surface into these tracks and sectors, and to write information concerning each track and sector location within the inter-record gap. This gap information is accessed only by the disc controller, and is not available to CP/M.

You will remember that CP/M considers the disc as a collection of 128 byte 'logical' sectors. Because the 128 byte sector is so integral to CP/M, the BIOS device drivers are designed so that the actual 256 byte sector is totally transparent to the CP/M program, even at the BIOS level. That is, whether you access data through BDOS or through the BIOS, the disc driver will perform all the translation automatically. For all practical purposes, the disc contains 128 byte sectors.

Reading Non-CP/M Discs

When you have a disc which meets the disc drive hardware format requirements, but which does not contain data organized for CP/M, you can make use of the BIOS disc calls described in Chapter 12.

Hewlett-Packard Format Discs

Because Hewlett-Packard disc drives format in fixed physical sectors of 256 bytes each, you must perform a manual conversion between physical sectors and CP/M sectors when reading non-CP/M disc media on the HP 125. Each physical disc sector actually includes two CP/M sectors.

To calculate the actual sector address, determine the desired CP/M sector. Divide that sector number by two: the integer portion of the quotient is the physical sector number. The remainder (MOD function) of the division indicates which half of the actual sector contains the CP/M sector: remainder of zero means the CP/M sector is the first 128 bytes of the sector, while a remainder of 1 means the CP/M sector is the second 128 bytes of the sector.

Regardless of any 'stagger' in the sectors during formatting, the disc driver within BIOS handles the conversion automatically. You need not access the 'translation table' in the Disc Parameter Header for any Hewlett-Packard media.

Non-HP Discs

While no industry standard exists for 5-1/4" media, the 'IBM interchange format' is supported on the 8" drives. By using this format, you can read and write IBM 3740-type media with BIOS calls.

The scheme for accessing 3740 discs is similar to that used for non-CP/M HP disc media. However, you must use the 'SECTRAN' BIOS routine as well. The sequence is as follows:

Select the desired track address using the 'SETTRK' routine.

Call the 'SECTRAN' routine. You will pass that routine the desired logical sector and the address of the XLT from the Disc Parameter Header. It will return the actual physical sector number to you.

Using the physical sector number, call 'SETSEC'. This will position the disc head to the proper sector address.

Finally, use the 'READ' or 'WRITE' routine, depending on which operation you wish to perform.

You can now use the data in any way necessary. Remember that the HP 125 uses ASCII data on disc, while some 3740-type media use other coding schemes.

Table D-1. Disc Format and Organization

	HP 82901 5-1/4"	HP 9895 8"	'IBM' 8"
Bytes/Physical Sector	256	256	128
Physical Sectors/Track	16	30	26
Bytes per CP/M Sector	128	128	128
CP/M Sectors per Track	32	60	26
Tracks per side	33	77	77
Tracks per disc	66	154	77

Note that the IBM interchange format disc utilizes only a single side, so that the 'Tracks per Side' and 'Tracks per Disc' are the same.

Once the physical disc requirements are met, CP/M requires certain conditions to be met in order to use standard CP/M utilities and functions. The various requirements are illustrated in Table D-2.

In each case, the number following the 'T' indicates the track address in decimal, and the value following the 'S' is the sector or range of sectors. For example, the 'Bootstrap' Sector is located at track 0, sectors 6 through 7 on both types of HP media.

Table D-2. Disc Track Allocation Map

	82901	9895	
LIF Volume Information	T 0 S 0-5	T 0 S 0-5	n/a
Bootstrap Sector	T 0 S 6-7	T 0 S 6-7	n/a
Bootstrap Parameters	T 0 S 8-9	T 0 S 8-9	n/a
CP/M Operating	T 0 S 10-31	T 0 S 10-31	n/a
	T 1 S 0-31	T 1 S 0-59	n/a
	T 2 S 0-31		n/a
Directory	T 3 S 0-31	T 2 S 0-31	T 2 S 0-15
User file	T 4 - T 65	T 2 S 32-59	T 2 S 16-25 T 3 - T 153

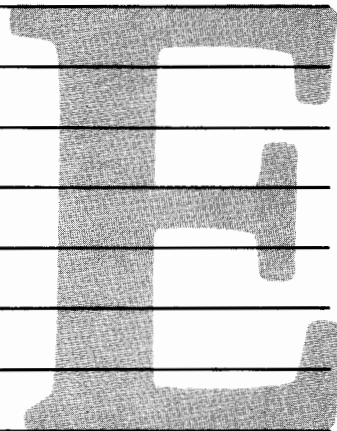
The LIF field above represents the 'Logical Interchange Format' used by many Hewlett-Packard products for interchangeability of media. The Bootstrap Sector contains the actual code which is initially loaded when a cold-start is performed, and the Boot Parameters specify various parameters required for the Boot ROM. The CP/M Operating System, when included on a particular media, is stored as indicated. If no system is included on a particular media, the space is nonetheless reserved.

The user file area begins with the directory sectors. Four file entries of 32 bytes each are packed into each 128 byte sector. Actual user files start immediately behind the directory tracks, and use the remainder of the disc.

Summary

Simple hardware compatibility is not sufficient for CP/M and its transient utilities to read and write data files. While the media must correspond to a supported format, the user can write an application utility to access data from non-CP/M discs.

When a properly formatted disc is on-line, directed track and sector access can be accomplished by using the BIOS calls which were described in Chapter 12.



ERROR MESSAGES



Error messages on the HP 125 can be classified into three general categories: System/Terminal Self-test Errors; TPU errors; and CPU errors. The first type can occur almost anytime a self-test is executed. Because there are so many possibilities, and because the HP 125 Owner's Manual contains a detailed explanation of all self-test outcomes, only the 'Power-On' self-test is described here.

The last two types of errors occur during normal processing, and indicate a possibly erroneous condition. In the final analysis, the operator must determine the cause, and the action to take. However, the HP 125 will attempt to 'recover' so that those tasks which can execute normally will do so. The HP 125 Owner's Manual contains some additional information on trouble-shooting.

Power-on Test

The power-on test is performed automatically whenever you turn on the system's power. It may also be executed manually by pressing the [AIDS] key followed by the service keys function key and then the POWER ON TEST function key. The power-on test normally takes 15 seconds to complete -- about the time it takes for the screen to warm up.

During those 15 seconds, the system performs a self-test of the electronic components that are critical to proper system operation, including the following integrated circuits ("chips"):

1. The two microprocessors
2. The five read-only memories (ROMs)
3. Non-volatile (configuration) memory (CMOS RAM)
4. Display memories (RAMs)
5. The operating system memories (RAMs)
6. Display electronics
7. The keyboard controller
8. The HP-IB controller
9. Internal printer electronics (if present)
10. Data Communication controllers
11. The interface between microprocessors

The system also tests the path from the System Processor to the disc drives through a "disc identify" test during the power-on test. Successful completion of the power-on test thus provides a very high degree of confidence that your system hardware is functioning properly.

One of three things may happen depending upon the results of the power-on test:

1. The test passes. Fifteen seconds after power is turned on, the display appears and the System Processor is ready for use.
2. The test fails. From 25 seconds to one minute after power is turned on, the display appears with the message

Power-on test failed nnnn

where nnnn has the following meaning.

0001	The System Processor is malfunctioning in some
0002	manner and may be unreliable.
0008	
0004	The System Processor should function as a remote
1000	terminal in all respects, however the local
8000	operating system is malfunctioning and may not be
	reliable.

0100	The operating system may not load or run
0200	correctly.
0400	
0800	
0010	The keyboard may not be functioning correctly.
0020	The configuration memory is malfunctioning and may not retain configuration settings. The System Processor may be unreliable.
0040	The data communications and/or serial printer interface is malfunctioning. Data communications and/or serial printer use may be unreliable.
0080	The internal printer is malfunctioning and may not be reliable.
2000	The HP-IB interface is malfunctioning and the HP-IB
4000	peripheral operations may be unreliable.

The values indicated above represent the hexadecimal values which are displayed on the internal LED's to indicate a failure. Because more than one error may occur, these values may appear in combination. For example, an HP-IB error and a keyboard error may display a value of '8010'. Remember: these are hex values, and represent individual bit settings. They are additive, but in hex. The values shown are hex representations of a binary flag word. If more than one bit is set, the hex representation of the binary sum will be displayed!

Note: If any of these power-on test failure messages is displayed contact your Service Representative.

3. The test fails. The display never appears or reappears but is visibly malfunctioning because the display electronics are faulty. Contact your Service Representative.

TPU Error Messages

These messages result when the TPU is requested to perform a task it is unable to accomplish. Generally, this is a result of some 'operator error' or a 'device mapping' problem. For example, when a 'COPY ALL' is performed and no printer has been selected, the TPU cannot complete the request and an error is displayed. When a program uses unconventional device mapping assignments, the same type of errors can result.

TPU error messages are indicated when a single asterisk (*) is the first character of an error message.

*Default config is used

The configuration set-up was lost. The system does not "remember" any changes you may have made to the default settings. All other parts of the system should operate correctly. If the message occurs repeatedly, contact your Service Representative.

*Device routing pending

An application program has specified a "device mapping" for its input/output operation. Certain terminal operations (e.g., configuration changes) are not allowed while this mapping is being used.

*Disc did not identify at address zero

The disc drive power was not turned on, the cable linking the disc drive and system is loose or disconnected, or there is a fault with your system or disc drives. Also, make sure that the disc from which you expect the operating system to be loaded is set to HP-IB address zero (0), as described in the "Getting Started with your HP 125 manual". If the problem persists, contact your Service Representative.

*Internal printer error

The internal printer is unable to print. Two possible causes are 1) the paper latch is not closed properly, or 2) the printer is out of paper. Otherwise the internal printer is malfunctioning and you should contact your Service Representative.

***No "TO" device**

A Printer logging or print operation tried to use a printer which CP/M has previously reserved, or vice versa, or no device was selected from the printer control function keys.

***Nonexistent printer**

An attempt was made to select a printer for output that is not configured into your system. Change the printer configuration using the Configuration Menu.



***Port1 error nnnnnn**

Failure response to DataComm Test function key. This result occurs if no test hood or cable is attached. The Data Comm Test function key is for use by service personnel.

***Port2 error nnnnnn**

Failure response to DataComm Test function key. This result occurs if no test hood or cable is attached. The Data Comm Test function key is for use by service personnel.

Press RETURN to clear

Occurs whenever a warning or error message is displayed.

***Printer(s) busy**

A logging or print operation tried to use a printer which CP/M has previously reserved or vice versa.

***Self-test inhibited**

The Terminal test, Power-on test, Internal Printer test or Datacomm test are inhibited when the Inhibit Self-test strap (L) is set to uppercase in the Configuration Menu.

***Test failed**

The Terminal test or Power-on test failed.

*Power-on test failed nnnn

Power-on test failure. See pages 11-8 11-9 for an explanation of the errors.

*This function LOCKED

The configuration is locked and an attempt was made to change it, or the [REMOTE MODE] or [LOCAL OP SYS] function keys were pressed while an unusual device mapping exists.

CPU Error Messages

These messages are displayed when the CP/M Operating System was unable to perform a requested task. This could be either an 'operator initiated' request, such as a disc directory ('DIR'), or because of a program file statement. In either case, such an error normally occurs when a disc problem of some type is indicated.

CPU error messages are always preceded by two asterisks (**) to differentiate them from TPU errors.

**OPSYS warmboot error

This error message should only occur if the system tries to partially re-load ("warmboot") an operating system that is of a different size than that which is currently in memory. After pressing the [RETURN] key, the HP 125 will "coldboot" the entire operating system from the disc in drive A. Check to see that you have the correct version of the operating system.

**Sector read/write error on drive x.

The system is unable to read or write to the flexible disc specified (A to H). The flexible disc is either 1) worn out, 2) not formatted, or 3) damaged by a magnet. After pressing the [RETURN] key to warmboot, make sure that the flexible disc is formatted and correctly placed in the drive. If you can read any part of it, attempt to copy as much as possible to another flexible disc, using COPY or PIP. Regardless of the success of the attempted copy, the flexible disc should be considered faulty and discarded. If a number of flexible discs seem to fail over a short period of time, and they are not old, contact your Service Representative.

****Disc select on drive x failed.**

The system cannot find the flexible disc specified. The disc door is open, there is no flexible disc in the drive, or you asked for a disc drive outside the range of A to H.

****Write to R/O disc error on drive x.**

An attempt was made to "write" to a flexible disc that the system was not "logged in". The first time a disc drive is accessed (with DIR, STAT, etc.) after a flexible disc reset, some information about the directory is copied into memory. If you change the flexible disc (and its contents are not IDENTICAL to the old one) without a reset (e.g., with a "warmboot"), then the next time you access the drive, CP/M "protects" the data on the flexible disc by placing it in a "read-only" status. Many application programs take care of this problem for you, but the standard CP/M utilities do not.

****Write to R/O file error on drive x.**

An attempt was made to write to a file that has been placed in a "read-only" status. This status indicates that the file cannot be updated or deleted unless it is "reset" to "read-write" (using a utility like STAT).

****Write to write-protected disc error in drive x.**

An attempt was made to write to a disc that was physically protected from update. On 5 1/4" discs, the tab is covered, and on 8" discs, the tab is open.

****DISC RETRY #n (Seek error on drive x).**

****DISC RETRY #n (Read error on drive x).**

****DISC RETRY #n (Write error on drive x).**

If the HP 125 cannot seek, read, or write a disc sector correctly, it automatically retries up to 9 more times. As it retries, the attempts are logged in this message for your information.

****Retry #n succeeded. Check disc in drive x for excessive wear.**

If any of the three types of retries described above are successful, the operating system displays this message, and the currently running program continues. Some programs may sense that this problem occurred, and prompt you to "press [RETURN] to continue", giving you a chance to check that the flexible disc is ok. Other programs and utilities will simply continue to completion.

****No OPSYS on disc in A:**

This message is displayed when the HP 125 cannot access an operating system on the flexible disc in the A drive. There may be no operating system on the flexible disc in A, the door may be open, or there may be no flexible disc in the drive. The system needs to access the operating system before loading some application programs, exiting most programs to return to CP/M, and when the operating system is partially or fully loaded into memory. To fix the problem, place a flexible disc with an operating system on it into drive A and press [SHIFT]-[CTRL]-[RESET].

- **Free OP SYS list device General printer.
- **Free OP SYS list device Internal printer.
- **Free OP SYS list device Port #2 printer.
- **Free OP SYS list device HPIB printer.

If there is a conflict between the local CPU and the terminal over which part of the system has control of a printer, this message may be displayed. Make sure that any previous use of a printer is finished, and press the [RESET] key to free the appropriate printer.

****Worn disc in drive x.**

It appears to the HP 125 that the flexible disc has worn out. Try to copy the contents of the flexible disc to another one. In any case you should consider the flexible disc as having failed. If a number of flexible discs seem to fail over a short period of time, and they are not old, contact your Service Representative.

****Disc parity error**

A hardware error occurred while accessing the flexible disc. If the condition persists, contact your Service Representative.

Summary

This concludes the section on error messages. Additional error information for system and terminal errors can be found in the HP 125 Owner's Manual. Error messages and recovery for the CP/M transient utilities such as PIP can be found with the other documentation on the specific utility in question.

Index



A

8080 Assembler.....15-1/15-23
Absolute Addressing.....5-2
Application
 Installation.....C-1/C-8
ASCII Codes.....A-1/A-2
ASM.....15-1/15-23
Assembler.....15-1/15-23
AUTO LINE FEED.....3-2

B

Back Tab.....5-7
Base Page.....7-8
BDOS.....7-6,7-7,7-11
BELL.....3-2
BIOS.....7-6/7-7
BIOS Entry Points.....12-1
Blinking Text.....5-5/5-6
Built in Commands.....7-3

C

CAPS LOCK Mode.....3-2
CAPS Mode.....3-3
CCP.....7-7,7-10,8-7
Clear All Margins.....5-8
Clear All Tabs.....5-7
Clear Display.....5-8
Clear Line.....5-8
Clear Tab.....5-7
CON:.....11-1
Configuration Locking.....3-4
Continuous Paging.....5-12
COPY.....13-13
COPY ALL.....5-12
COPY LINE.....5-12
COPY PAGE.....5-12
Cursor Control.....5-1
Cursor Movement.....5-4
Cursor Relative
 Addressing.....5-2
Cursor Sensing.....6-15
CURSOR WRAPAROUND.....3-4

D

Data Transfers.....5-12
DC2.....3-4
DDT.....16-1/16-13
Decoding Status Bytes.....6-2
Delete Character.....5-8
Delete Line.....5-8
Destination Device.....11-5
Device Assignments.....11-1
Device Mapping.....11-4
Disable Key Labels.....4-2
Disable Keyboard.....5-10
Disable Logging.....5-12
Disc Capacity.....8-2
Disc Directory.....8-7
Disc Drive Identifier.....7-8
Disc File Entry.....8-3
Disc Format.....D-1/D-5
Disc Parameter
 Block.....12-6
Disc Parameter
 Header.....12-5
Display Enhancements...5-5/5-6
Display Functions.....5-10
DPB.....12-6
DPH.....12-5
Dump.....13-18
Dump Display Memory.....5-10
Dynamic Debugging
 Tool.....16-1/16-13

E

ED.....14-1/14-17
Edit Control
 Sequences.....5-8/5-9
Editor.....14-1/14-17
Enable Key Labels.....4-2
Enable Keyboard.....5-10
Enter.....5-11
Escape Code Summary...A-4/A-6
Escape Sequence...1-3,2-1/2-24
Extent.....8-5

F

FCB.....8-3,8-8
 FDOS.....7-6,7-11
 File Access.....7-10
 File Control Block.....8-3
 File Names.....7-8
 File Structure.....7-9
 File System.....7-8,8-1
 File Type.....7-9
 FORMAT.....13-14,D-1
 Function Group Control.....4-2

H

Half-Bright Text.....5-5/5-6
 HANDSHAKE.....3-4
 HP Extended Function
 Calls.....10-1
 HPIB Printer Status.....6-13

I

INHIBIT CURSOR WRAPAROUND..3-4
 INHIBIT DC2.....3-4
 INHIBIT HANDSHAKE.....3-4
 INHIBIT SELF-TEST.....3-4
 IOBYTE.....8-11,11-2/11-4
 Insert Character.....5-9
 Insert Line.....5-8
 Internal Printer Status...6-13
 Inverse Video.....5-5/5-6

K

KEY CLICK.....3-3
 Keycode Mode.....12-8,A-7

L

LINE FREQUENCY.....3-2
 LOAD.....13-18
 LOCAL ECHO.....3-2
 LOCAL Mode.....1-4
 LOCAL OP Sys Mode.....1-4,3-3
 LOG BOTTOM.....5-12
 LOG TOP.....5-12
 Logging.....5-12
 LST:.....11-2

M

Mapping Modes.....11-9
 Margin Control.....5-6

Memory Map.....7-6
 METRIC PRINT.....5-12
 Modem Disconnect.....5-11
 [MODES] Softkeys.....4-2
 MODIFY MODE.....3-2
 MOVCPM.....13-9

N

Next Page.....5-9

P

Page Format.....5-12
 Pass Thru Mode.....11-9
 Physical List Device
 Names.....13-13
 PIP.....13-2
 Previous Page.....5-9
 Primary Terminal Status...6-3
 Printer Control.....5-12
 Printer Status.....6-11
 PUN:.....11-2

R

RDR:.....11-1
 REMOTE Mode.....1-4,3-3
 REPORT PRINT.....5-12
 Roll Down.....5-9
 Roll Up.....5-9

S

Screen Relative
 Addressing.....5-2
 Secondary Terminal Status..6-8
 SELF-TEST.....3-4
 Serial Printer Status.....6-14
 Set Left Margin.....5-7
 Set Right Margin.....5-8
 Set Tab.....5-7
 Soft Reset.....5-11
 Source Device.....11-5
 SPOW Latch.....3-3/3-5
 STAT.....13-9/13-13
 Status Bytes.....6-2
 SUBMIT.....13-15/13-16
 SYSGEN.....13-9
 System Function Calls.....9-1

T

Tab Back.....5-7
Tab Control.....5-7
Tab Forward.....5-6
Terminal Identifier.....6-14
Text Editor.....14-1/14-17
TPA.....7-7
TPU Pause.....5-11
TPU Self-Test.....5-11
Transient Commands....7-4,7-11
Transient Utilities 13-1/13-18
TRANSMIT FUNCTION KEY.....3-3

U

Underline Test.....5-5/5-6
[USER KEY]
 Definitions.....4-3/4-7
[USER KEYS] Softkeys.....4-2

X

XSUB.....13-17

