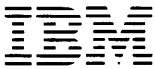


Personal Computer
Computer Language
Series

CP/M-86™ Operating System

First Edition (March 1982)

Changes are periodically made to the information herein; these changes will be incorporated in new editions of this publication.

Products are not stocked at the address below. Requests for copies of this product and for technical information about the system should be made to your authorized IBM Personal Computer Dealer.

A Product Comment Form is provided at the back of this publication. If this form has been removed, address comment to IBM Corp., Personal Computer, P.O. Box 1328-C, Boca Raton, Florida 33432. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligations whatever.

Portions of this material are licensed from Digital Research, Inc.

CONTENTS

Preface

Welcome to the world of microcomputers opened to you by your IBM Personal Computer. Welcome also to the world of application software accessible through Digital Research CP/M-86™. Digital Research designed this version of CP/M-86 especially for the Intel™ 8088 microprocessor that is the heart of your IBM Personal Computer.

What CP/M-86 does for you

CP/M-86 manages information stored magnetically on your diskettes by grouping this information into files of programs and data. CP/M-86 can copy files from a diskette to the IBM Personal Computer's memory, or to a peripheral device such as a printer. CP/M-86 performs these and other tasks by executing various programs according to commands you enter at your keyboard.

Once in memory, a program runs through a set of steps that instruct your computer to perform a certain task. The advantage of using CP/M-86 on your IBM Personal Computer is that you can create your own CP/M-86 programs to entertain, educate, or solve your own commercial or scientific problems.

What you need to run CP/M-86 on the IBM Personal Computer

The minimum IBM Personal Computer consists of a System Unit with 16K of memory, a keyboard and a screen device. However, CP/M-86 needs some additional features to operate properly.

To start, CP/M-86 needs at least one diskette drive. You'll find a second drive particularly handy for making quick copies of your own programs and data. CP/M-86 can support up to four logical diskette drives. You'll also need at least 32K of memory. To run DDT-86 you must have 48K, and to run ASM-86 and many of the application programs that run under CP/M-86 you must have 64K.

If you expand your system beyond these minimums, you will appreciate that CP/M-86 supports many other features you can add to your IBM Personal Computer. For example, CP/M-86 can support the maximum amount of memory you can install. CP/M-86 supports both low and high resolution, color and monochrome displays. CP/M-86 can also accept input from a light pen.

How this book is organized

This book introduces you to CP/M-86 and tells you how to use it. The book assumes you have read the *Guide to Operations* that accompanied your IBM Personal Computer, and are familiar with the parts of your computer, how to set it up and turn it on, and how to handle, insert and store diskettes. However, it does not assume you have had a great deal of experience with computers. It starts with the basics, then gradually advances to more technical detail.

Chapter 1 tells how to start CP/M-86, enter a command and make a back-up diskette. Chapter 2 discusses diskettes and files. Chapter 3 develops the CP/M-86 command concepts you need to understand the command summary in Chapter 4. The command summary describes every command and program supplied with CP/M-86. Because some of these programs are too complex to be adequately described in a few pages, they are singled out for complete discussion in later chapters.

Chapter 5 tells you how to use ED, the CP/M-86 file editor. With ED you can create and edit program, text and data files.

Chapters 6 through 10 describe ASM-86, the CP/M-86 assembler for your IBM Personal Computer. You won't need ASM-86 until you decide to write assembly language programs and become more familiar with your IBM Personal Computer's 8088 microprocessor instruction set. When you do, you'll find that ASM-86 simplifies writing 8088 microprocessor programs.

Chapter 11 gives you the operating details for DDT-86, the CP/M-86 debugging program. You can use DDT-86 to find errors in programs written in high level languages as well as in ASM-86.

Appendix A lists the messages CP/M-86 displays when it encounters special conditions. If the condition requires correction, Appendix A can also tell you what actions you should take before you proceed.

Appendixes B through F provide you with details not required for day-to-day operation of CP/M-86. Appendix B tells how application programs are loaded for operation with CP/M-86, while Appendix C shows you how to create new CP/M-86 commands.

Appendix D tells how a program can call on CP/M-86 to perform tasks like reading or writing to diskettes. You'll need to read this appendix when you write your own programs that use CP/M-86 facilities. A sample program is included in Appendix E.

Finally, Appendix F shows you how to use a color monitor and light pen when you connect them to your IBM Personal Computer.

If you are new to computers, you may find these last topics a bit confusing, right? But half the fun of an IBM Personal Computer is learning more about how to use it in your home, business, research, or school. It's the purpose of this book to proceed step-by-step so you can readily understand each CP/M-86 operation.

CHAPTER 1. INTRODUCTION

Contents

Introduction

This chapter discusses the fundamentals of your IBM Personal Computer and CP/M-86. It starts by describing CP/M-86 start-up procedures and initial messages. Then it shows how to enter a CP/M-86 command and make a back-up copy of your CP/M-86 distribution diskette.

How to get CP/M-86 started

Starting or loading CP/M-86 means reading a copy of CP/M-86 from your CP/M-86 system diskette into your computer's memory. You can start CP/M-86 in one of two ways, depending on whether your computer is powered on or off.

If power is off, insert your CP/M-86 system diskette with the label facing upward into drive A, the built-in drive on the left side of the System Unit. Close the drive door. When you turn the power on, your IBM Personal Computer automatically loads CP/M-86 into memory after a few moments of self-testing. (If you've just powered off but want to use your computer again, you must count slowly to five between turning the power off and then back on.)

If power is on and you want to restart CP/M-86, first make sure your CP/M-86 system diskette is in drive A and then hold down the Ctrl and Alt keys and press the Del key. Release all three keys. This sequence is called System Reset.

In either start sequence, the light on drive A goes on and the drive clicks and whirs as CP/M-86 is loaded into memory. The first thing CP/M-86 does after it is loaded into memory is display the following message on your screen:

CP/M-86 for the IBM Personal Computer Version V.V
Copyright (c) 1982 Digital Research Inc.

The version number, represented above by V.V, tells you the major and minor revision level of the CP/M-86 version that you own. After displaying this message, CP/M-86 takes inventory of your computer to find out some of its components and reports what it finds. For example, if your computer has two disk drives, a printer, a serial communications card (Asynchronous Communications Adapter) and 64K of memory, CP/M-86 writes the message:

Hardware supported:

Diskette(s): 2
Printer(s): 1
Serial Port(s): 1
Memory (Kb): 64

Then CP/M-86 writes a "status line" at the bottom line of your screen. The status line has space for a status message, written by CP/M-86, followed by the current user number and actual or elapsed time and date. The initial status message is blank and the initial time is all zeros. The first date displayed is the date your version of CP/M-86 was created. However, you can use the TOD command described in Chapter 4 to set the date and time fields to the current date and time. Otherwise, CP/M-86 keeps track of the time elapsed from the instant CP/M-86 begins operation.

Finally, you'll get the two character message:

A>

This "system prompt" tells you CP/M-86 is ready to read a command from your keyboard. It also telis you that drive A is your "default" drive. That means that until you tell CP/M-86 to do otherwise, it looks for program and data files on the diskette in drive A.

A sample command

CP/M-86 performs certain tasks according to specific commands that you type at your keyboard. A CP/M-86 command line is composed of a command keyword, an optional command tail, and an "Enter" keystroke. The command keyword identifies a command (program) to be executed. The command tail can contain extra information for the command such as a filename or parameter. To end the command line, you must press the Enter key (↵).

As you type characters at the keyboard, they appear on your screen and the cursor moves to the right. If you make a mistake in typing, push the Backspace key (←) to move the cursor to the left and correct the error.

You can type the keyword and command tail in any combination of upper-case and lower-case letters. CP/M-86 treats all letters in the command line as upper case.

Generally, you type a command line directly after the system prompt. However, CP/M-86 does allow spaces between the prompt and the command keyword.

A command keyword identifies one of two different types of commands: Built-in commands and Transient Utility commands. Built-in commands reside in memory as a part of CP/M-86 and can be executed immediately. Transient Utility commands are stored on diskette as program files. They must be loaded into memory to perform their task. You can recognize Transient Utility program files in a diskette's directory because their filenames end with CMD.

For Transient Utilities, CP/M-86 checks only the command keyword. If you include a command tail, CP/M-86 passes it to the utility without checking it because many utilities require unique command tails.

Let's use one Built-in command to demonstrate how CP/M-86 reads command lines. The DIR command tells CP/M-86 to display the names of diskette files on your screen. Type the DIR keyword after the system prompt, omit the command tail, and press Enter.

```
A>DIR
```

CP/M-86 responds to this command by writing the names of all the files you have stored on the diskette in drive A. For example, if you have your CP/M-86 system diskette in drive A, these filenames, among others, appear on your screen:

```
COPYDISK    CMD
PROTOCOL    CMD
NEWDISK     CMD
```

CP/M-86 recognizes only correctly spelled command keywords. If you make a typing error and press Enter before correcting your mistake, CP/M-86 echoes the command line with a question mark at the end. For example, if you accidentally mistype the DIR command, CP/M-86 responds

```
A>DJR
DJR?
```

to tell you that it does not understand the command keyword.

DIR accepts a filename as a command tail. You can use DIR with a filename to see if a specific file is on the disk. For example, to check that the Transient Utility program COPYDISK.CMD is on your system diskette, type:

A>DIR COPYDISK.CMD

CP/M-86 performs this task by writing either the name of the file you specified or the message NO FILE.

Be sure to type at least one space after DIR to separate the command keyword from the command tail. If you don't, CP/M-86 responds as shown below.

A>DIRCOPYDISK.CMD
DIRCOPYDISK.CMD?

Some of the utility programs display messages requiring a response. When you type in your answer you must press the Enter key to send the response to the program.

CP/M-86 line editing control characters

You can correct simple typing mistakes with the Backspace (←) key. However, CP/M-86 supports the following control character commands to help you edit more efficiently. You can use these these control characters to edit command lines or input lines to most programs. To type a control character, hold down the Ctrl key and press the required letter key. Release both keys.

Ctrl-E — moves the cursor to the beginning of the following line without erasing your previous input.

Ctrl-H — moves the cursor left one character position and deletes the character–the same as the backspace (←) key.

Ctrl-I — moves the cursor to the next tab stop, where tab stops are automatically placed at each eighth column–same as the (→) key.

Ctrl-J	moves the cursor to the left of the current line and sends the command line to CP/M-86–same as an Enter (↵) keystroke.
Ctrl-M	moves the cursor to the left of the current line and sends the command line to CP/M-86–same as an Enter (↵) keystroke.
Ctrl-R	types a # at the current cursor location, moves the cursor to the next line and retypes any partial command you have typed so far.
Ctrl-U	discards all the characters in the command line that you've typed so far, types a # at the current cursor position and moves the cursor to the next command line.
Ctrl-X	discards all the characters in the command line that you've typed so far and moves the cursor back to the beginning of the current line.

You probably noticed that some control characters have the same meaning. For example, the Ctrl-J and Ctrl-M keystrokes have the same effect as pressing the Enter key: all three send the command line to CP/M-86 for processing. Also, Ctrl-H has the same effect as pressing the backspace (←) key.

Why you should back up your files

Humans have faults, and so do computers. Human or computer errors sometimes destroy valuable programs or data files. By mistyping a command, for example, you could accidentally erase a program that you just created. A similar disaster could result from an electronic component failure.

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

Professionals also make frequent copies of new programs or data files during the time they are being developed. The frequency of making copies varies with each programmer, but as a general rule, make a copy at the point where it takes ten to twenty times longer to reenter the information than it takes to make the copy.

You can make back-ups in two ways. You can back up files one at a time, or you can make a complete copy of the entire diskette. The choice is usually made based on the number of files on the diskette that need to be backed up. It takes less than a minute to make a copy of one file, but it takes only two or three minutes to copy an entire diskette.

So far, we haven't discussed any commands that change information recorded on a CP/M-86 system diskette. Before we do, let's make a few working copies of your original CP/M-86 diskette.

How to make a copy of your CP/M-86 diskette

CP/M-86 supports two kinds of drives for your IBM Personal Computer: single-sided or double-sided. Before copying your CP/M-86 system diskette, you need to know whether your computer has single-sided or double-sided drives. If your system has single-sided drives, you need to prepare your back-up diskette with a single-sided format so CP/M-86 can write to it properly. If your system has double-sided drives, you can prepare your back-up diskette with a single-sided or a double-sided format. If you prepare the diskette with a double-sided format, be sure to use the procedure for double-sided diskettes described below. CP/M-86 is distributed on single-sided diskettes only.

To back up your CP/M-86 diskette, you will use one or more diskettes for the back-ups, the NEWDISK and COPYDISK or PIP Transient Utility programs, and of course your CP/M-86 diskette. The back-up diskettes may be factory-fresh or used. If the diskettes are factory-fresh, you must format them with the NEWDISK transient utility program. If the diskettes are used, make sure they do not contain any information you may need again! COPYDISK copies everything from a source diskette to a destination diskette–including blank space–and writes over any information that may already be stored on the destination diskette.

This section shows how to use NEWDISK and COPYDISK or PIP on an IBM Personal Computer with two diskette drives. The next section describes the differences involved if you are working with one drive. Whether you have one drive or two, you may need to use NEWDISK to format a factory-fresh diskette for your back-up. Before starting NEWDISK, make sure your CP/M-86 diskette is in drive A and the factory-fresh

disk is in drive B. Then type the following command line and press the Enter key.

A>newdisk b: $n

Note: If you are preparing double-sided diskettes you must use the form

A>newdisk b: $ds

to tell CP/M-86 to format the disk on both sides. In response to either of the command lines shown above, CP/M-86 loads NEWDISK into memory and runs it. NEWDISK sends the following display to the screen:

NEWDISK vn 1.0 mm/dd/yy

Disk B will be formatted.

ALL DATA WILL BE ERASED FROM THE DISK.

Is this what you want (y/n)?

If you press n, NEWDISK returns control to CP/M-86 and the system prompt reappears. If you press y, NEWDISK continues and displays the message:

Disk format in progress.
000

NEWDISK tells you which track of the diskette it is formatting by "counting" from 0 to 39 (or 79 for double-sided diskettes). When NEWDISK is finished, the following message is displayed:

Format Complete.
Press Control-C to exit, or
ENTER to format another disk.

Press the Enter key if you have more factory-fresh diskettes to format. Press Ctrl-C to return to the CP/M-86 system prompt. See the description of NEWDISK in Chapter 4 for a complete discussion of NEWDISK's capabilities.

To make a copy of your single-sided CP/M-86 diskette, use the COPYDISK utility described below. To make a double-sided CP/M-86 system diskette, use the PIP utility described afterwards.

First make sure that your system diskette is in drive A and a single-sided formatted diskette is inserted in drive B.

Note: In general, when using COPYDISK both diskettes must have the same format. You must use two single-sided or two double-sided diskettes. In this case, because your CP/M-86 system diskette is single-sided, you must use two single-sided diskettes.

Enter the following command to the system prompt:

A>copydisk

CP/M-86 loads COPYDISK into memory and runs it. COPYDISK displays the following messages on your screen and prompts you for the source and destination drive names, then verifies that you do indeed want to make the copy:

CP/M-86 Full Disk Copy Utility
Version 1.0

Enter Source Disk Drive (A-D)? A

Destination Disk Drive (A-D)? B

Copying disk A: to disk B:
Is this what you want to do (Y/N)? Y
Copy started

Reading track nn (after read, new text appears)
Writing track nn (after write, next message is)
Verifying track nn
Copy completed.

Copy another disk (Y/N)? N
Copy program exiting

To make a copy of your CP/M-86 system on a double-sided diskette, use the PIP utility. First make sure that your CP/M-86 diskette is inserted in drive A and a diskette formatted by NEWDISK to double-sided is in drive B. Enter the following command to the system prompt or press the F6 function key, and then press Enter.

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

CP/M-86 loads PIP into memory and runs it. PIP displays the prompt "COPYING–" on your sreen, followed by the name of each diskette file as it is copied to the back-up diskette. When PIP finishes copying all of the files from the source diskette to the back-up diskette, it displays the CP/M-86 system prompt.

Now you have an exact copy of the original CP/M-86 diskette in drive B. Remove the original from drive A and store it in a safe place. If your original remains safe and unchanged, you can easily restore your CP/M-86 program files if something happens to your working copy.

Remove the copy from drive B and insert it in drive A. Press Ctrl-C. Use the copy as your CP/M-86 system diskette to make more back-ups, to try the examples shown in the rest of this book, and to start CP/M-86 the next time you power up your IBM Personal Computer.

f you only have one drive

Perhaps you purchased only one diskette drive and plan to add another later. If so, you must shuffle diskettes in and out of your single drive to copy programs or data from one diskette to another. However, CP/M-86 simplifies this task by acting like a two-drive system. It keeps track of your source and destination diskettes and tells you when to take out one and insert the other. To keep things clear, CP/M-86 refers to the diskette that would reside in drive A as Diskette A, and the diskette that would reside in drive B as Diskette B. You may wish to label your diskettes A and B to help you keep track during a long copy operation.

As an example, let's make a back-up copy of your original CP/M-86 diskette. Assume your drive is single-sided. To begin, enter the same command you would if you had two drives.

A>copydisk

CP/M-86 loads and executes COPYDISK, which at first displays its normal messages. However, after it has read a track from the source diskette into memory, it beeps and displays the following message in the status line:

Put Disk B in A; ENTER to continue

This message tells you to insert the destination diskette into your only drive, drive A. After you insert the diskette, push Enter to tell COPYDISK to write the information from memory to the diskette. When COPYDISK finishes writing, it sounds a beep and types the following message on the status line:

Put Disk A in A; ENTER to continue

Return the source diskette to the drive and press Enter to continue. Switch diskettes as COPYDISK requests until the copy is complete.

The CP/M-86 utilities that copy information from one diskette to another, such as PIP and COPYDISK, or perform other diskette functions, such as NEWDISK, simulate multiple drives in this way. They display the same messages in the status line when it is time for you to change diskettes. See the descriptions in Chapter 4 for a detailed discussion of how each command and utility works on a single-drive system.

CHAPTER 2. FILES, DISKETTES, DRIVES AND DEVICES

Contents

iles, Diskettes, Drives and Devices

CP/M-86's most important task is to access and maintain files on your diskettes. It can create, read, write, and erase program and data files. This chapter tells you what a file is, how to create, name and access a file, and how files are stored on your diskettes. It also tells how to indicate to CP/M-86 that you've changed diskettes or that you want to change your default drive.

Vhat is a file?

A CP/M-86 file is a collection of related information stored on a diskette. Every file must have a unique name because that name is used to access that file. A directory is also stored on each diskette. The directory contains a list of the filenames stored on that diskette and the locations of each file on the diskette.

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

A data file may contain the source code for a program. Generally, a program source file must be processed by an assembler or compiler before it becomes an executable program file. In most cases, an executing program processes a data file. However, there are times when an executing program processes an executable program file. For example, DDT-86 can both edit and execute a command file.

Iow are files created?

There are many ways to create a file. You can create a file by copying an existing file to a new location, perhaps renaming it in the process. Under CP/M-86, you can use the Transient Utility PIP to copy and rename files. The second way to create a file is to use a text editor. The CP/M-86 text editor ED can create a file and assign it the name you specify. Finally, some

programs such as ASM-86 create output files as they process input files.

Naming files–what's in a name?

CP/M-86 identifies every file by its unique file specification. A file specification can have three parts: a drive specification, a filename and a filetype. We recommend that you create file specifications made up of letters and numbers. Because the CP/M-86 command processor recognizes the following special characters as delimiters, they must not be a part of a file specification:

< > . , ; : = ? * [ ]

A file specification can be simply a one- to eight-character filename, such as:

MYFILE2

When you make up a filename, try to let the name tell you something about what the file contains. For example, if you have a list of customer names for your business, you could name the file

CUSTOMER

so that the name is eight or fewer characters and also gives you some idea of what's in the file.

As you begin to use your IBM Personal Computer with CP/M-86, you'll find that files fall naturally into families. To keep file families separated, CP/M-86 allows you to add a family name, called a filetype, to the filename. For example, you could add the following filetype to the file that contains a list of customer names:

CUSTOMER.NAM

The executable program files that CP/M-86 loads into memory from a diskette have different filenames, but are in the family of 8088 programs that run with CP/M-86. The filetype CMD identifies this family of executable programs.

A filetype may be up to three characters long. Try to use three letters that tell something about the file's family. When you type a file specification for CP/M-86 to read, separate the filetype from the filename with a period. When CP/M-86 lists file specifications for you to read in response to a DIR command, it separates the filename from the filetype with blanks so that you can compare filetypes quickly.

CP/M-86 has already established several file families. Here's a list of their filetypes with a short description of each family:

CMD	8088 Machine Language Program
LST	Printable output from ASM-86
$$$	Temporary
A86	ASM-86 Source Program
H86	Assembled ASM-86 Program in hexadecimal format
SUB	List of commands to be executed by SUBMIT

Accessing files—do you have the correct drive?

When you type a file specification in a command tail, the Built-in or Transient Utility looks for the file on the diskette in the drive named by the system prompt. For example, if you type the command

A>dir copydisk.cmd

CP/M-86 looks in the directory of the diskette in drive A for COPYDISK.CMD. But if you have another drive, B for example, you need a way to tell CP/M-86 to access the diskette in drive B instead. For this reason, CP/M-86 lets you to precede a filename with a drive specification, which is the drive letter followed by a colon. For example, in response to the command

A>dir b:myfile2.lib

CP/M-86 looks for the file MYFILE2.LIB in the directory of the diskette in drive B.

You can also precede an executable program filename with a disk specification, even if you are using the program filename as a command keyword. For example, if you type the command

A>b:pip

CP/M-86 looks in the directory of the diskette in the B drive for the file PIP.CMD. If CP/M-86 finds PIP on drive B, it loads PIP into memory and executes it.

Unlike the filename and filetype which are stored in the diskette directory, the drive specification for a file changes as you move the diskette from one drive to another. Therefore a file has a different file specification when you change its diskette from one drive to another.

Accessing more than one file

Certain CP/M-86 Built-in and Transient Utilities can select and process several files when special "wildcard" characters are included in the filename or filetype. A file specification containing wildcards can refer to more than one file because it gives CP/M-86 a pattern to match: CP/M-86 searches the diskette directory and selects any file whose filename or filetype matches the pattern.

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

- A ? matches any character in a name, including a space character.
- A * must be the last, or only, character in the filename or filetype. CP/M-86 internally replaces a * with ? characters to the end of the filename or filetype.
- When the filename to match is shorter than eight characters, CP/M-86 treats the name as if it ends with spaces.
- When the filetype to match is shorter than three characters, CP/M-86 treats the filetype as if it ends with spaces.

Suppose, for example, you have a diskette with six files named A.CMD, AA.CMD, AAA.CMD, B.CMD, A.A86, and B.A86.

Several cases are listed below where a name with wildcards matches all, or a portion of, these files:

* *	is treated as ????????.???
????????.???	matches all six names
*.CMD	is treated as ????????.CMD
????????.CMD	matches the first four names
?.CMD	matches A.CMD and B.CMD
?.*	is treated as ?.???
?.???	matches A.CMD, B.CMD, A.A86, and B.A86
A?.CMD	matches A.CMD and AA.CMD
A*.CMD	is treated as A???????.CMD
A???????.CMD	matches A.CMD, AA.CMD, and AAA.CMD

Remember that CP/M-86 uses wildcard patterns only while searching a diskette directory, and therefore wildcards are valid only in filenames and filetypes. You cannot use a wildcard in a drive specification.

How can I organize and protect my files?

Under CP/M-86 you can organize your files into groups, protect your files from accidental change, and specify how your files are displayed in response to a DIR command. CP/M-86 supports these features by assigning user numbers and attributes to files and recording them in the diskette's directory.

You can use user numbers to separate your files into 16 file groups. All files are identified by a user number which ranges from 0 to 15. CP/M-86 assigns a user number to a file when the file is created. Generally, the file is assigned the current user

number, the one displayed in the Status Line at the bottom of your screen. You can change the current user number with the Built-in USER command described in Chapter 4.

Most commands can access only those files that have the current user number. For example, if the current user number is 7, a DIR command displays only the files that were created under user number 7. The exception to this is the PIP command. PIP can copy a file with one user number and give the copy another user number. See the discussion of PIP in Chapter 4.

File attributes control how a file can be accessed. There are two kinds of file attributes. The first attribute can be either DIR (Directory) or SYS (System). When you create a file, it is automatically marked with the DIR attribute. You can display the name of a file marked with the DIR attribute only if the file has the current user number. If you give a file the SYS attribute, it is not displayed in response to a DIR command; you must use DIRS as described in Chapter 4.

If you give a file with user number 0 the SYS attribute, you can read and execute that file from any user number on the same drive. This feature gives you a convenient way to make your commonly used programs available under any user number. However, note that a user 0 SYS file does not appear in response to a DIRS command unless 0 is the current user number.

The second file attribute can be set to either R/W (Read Write) or R/O (Read Only). If a file is marked R/O, any attempt to write data to that file produces a Read-Only error message. Therefore you can use the R/O attribute to protect important files. A file with the R/W attribute can be read or written to at any time unless there is a tab over the write-protect notch on the diskette, or the drive containing the diskette is set to Read Only. You can use the STAT Transient Utility program to assign attributes to a file.

How are files stored on a diskette?

CP/M-86 stores files on a single-sided or double-sided diskette in the same manner. CP/M-86 records the filename, filetype, user number and attributes of each file in a special area of the diskette called the directory. In the directory, CP/M-86 records which diskette sectors belong to which file. Whether

the diskette is single-sided or double-sided, the directory is large enough to store this data for up to sixty-four files.

CP/M-86 allocates directory and storage space for a file only as the file grows. When you erase a file, CP/M-86 reclaims storage in two ways: it makes the file's directory space available to catalog a different file, and frees the file's storage space for later use. It's this "dynamic allocation" feature that makes CP/M-86 powerful. You don't have to tell CP/M-86 how big your file will become because CP/M-86 automatically allocates more storage for a file as it is needed, and releases the storage for reallocation when the file is erased.

Changing diskettes

CP/M-86 cannot, of course, do anything to a file unless the diskette that holds the file is inserted into a drive and the drive door is closed. When a diskette is in a drive, it is "on-line" and CP/M-86 can access its directory.

At some time, you'll have to take a diskette out of a drive and insert another that contains different files. You can replace an on-line diskette whenever you see the system prompt at your console. However, you must tell CP/M-86 that you have changed a diskette by typing Ctrl-C directly after the system prompt. In response, CP/M-86 "logs in" the new diskette.

If you forget to type Ctrl-C after you change a diskette, CP/M-86 automatically protects the new diskette by setting it to Read-Only (R/O). You can run a text editor or copying program and try to write to the new diskette, but when you do, CP/M-86 notices that the original diskette is no longer in the drive and writes the message:

Bdos err on d: R/O

where d: is the drive specification of the new diskette. If you get this message, you must type one Ctrl-C to return to the system prompt and another Ctrl-C to log in the new diskette.

Note: After changing between single-sided and double-sided diskettes, you must type a Ctrl-C immediately following the system prompt for CP/M-86 to read or write to the changed diskette properly. To distinguish between a single-sided and a double-sided diskette, refer to the STAT DSK: command described in Chapter 4.

There are times when it is appropriate to load a command program into memory and then change a diskette while the program in memory has paused for the change. At these times, the new diskette that you insert in the drive MUST be in the same format as the diskette you have removed from the drive. If it is not of the same format, CP/M-86 cannot read it properly. It is not possible to use the Ctrl-C to properly log in the new diskette because the Ctrl-C causes the program to stop executing and return to the operating system prompt.

To sum up, there are two things to note when changing a diskette after loading a program into memory:

- You must replace the diskette you removed with a diskette in the same format, either single-sided or double-sided.

- CP/M-86 can read from the changed diskette but cannot write to a diskette unless it has been logged in at system start-up or with a Ctrl-C following the system prompt.

Recall that CP/M-86 protects a diskette that you have changed and forgotten to log in by setting it to Read-Only status. This prevents data meant for one diskette from being written on the wrong diskette.

Changing the default drive

At any given time during operation of CP/M-86, there is one drive called the default drive. Unless you put a drive specification in your command line, CP/M-86 and the utilities look in the directory of the diskette in the default drive for all program and data files. You can tell the default drive from the CP/M-86 system prompt. For example, the message

A>

tells you that the A drive is the default drive. When you give commands to CP/M-86, you should remember which diskette is the default drive. Then you will know which files an

application program can access if you do not add a drive specification.

Drive A is the default drive when you start CP/M-86. If you have more than one drive, you may want to change the default drive. Do this by typing the drive specification of the desired default drive next to the system prompt and pressing the Enter key.

A>B:

This command, for example, changes the default drive to B. Unless you change the default drive again, all system prompt messages appear as:

B>

The system prompt now indicates that CP/M-86 and its utilities will check in the directory of the diskette in drive B for any file that does not have a drive specification included in the file specification.

More CP/M-86 drive features

Under CP/M-86, drives can be marked R/O just as files can be given the R/O attribute. The default state of a drive is R/W, but CP/M-86 marks a drive R/O whenever you change the diskette in the drive. To return the drive to R/W you must type a Ctrl-C to the system prompt or a Ctrl-Break. You can give a drive the R/O attribute by using the STAT Transient Utility described in Chapter 4.

As mentioned in the Preface, CP/M-86 can support up to four logical diskette drives. This means you can use the drive specifications C: and D: in CP/M-86 command lines even when you have only one or two drives. If you do, CP/M-86 keeps track of three or four diskettes. It displays messages similar to those it uses when you specify B: in a single drive system. That is, it asks you to insert Diskette C or D when it would normally be accessing drive C or D. For example, during a copy operation, CP/M-86 prints the following message in the status line when it is time to swap diskettes:

Put Diskette C in A; ENTER to continue

Other CP/M-86 Devices

CP/M-86 manages all the peripheral devices attached to your IBM Personal Computer. Peripheral devices are actual physical pieces of hardware such as disk drives, input devices such as keyboards, light pens or serial devices, and output devices such as printers, serial devices, and screens.

A "logical device" is a name that CP/M-86 uses to keep track of input and output. The table below shows CP/M-86 logical device names and indicates whether the device is input or output.

CON:	Console input and output
AXI:	Auxiliary input
AXO:	Auxiliary output
LST:	List output

The user communicates with the physical devices attached to the computer through CP/M-86 logical devices. The ASSIGN command allows the user to assign the physical devices to the CP/M-86 logical device names. For example, the default console input device is the keyboard and the default console output device is the screen. If you want CP/M-86 to manage an optional peripheral, you must use the ASSIGN command to assign an alternate peripheral to the logical device name. For example, an ASSIGN command can change the console input device from the keyboard to other serial devices.

A logical input device can be assigned only one physical device, but you can assign several peripherals to a logical output device. For example, you can send list output to the screen, to the serial ports, and to up to three printers. See the description of the ASSIGN command in Chapter 4 for more detail.

CHAPTER 3. CP/M-86 COMMAND CONCEPTS

Contents

CP/M-86 Command Concepts

As we discussed in Chapter 1, a CP/M-86 command line consists of a command keyword, an optional command tail, and an Enter keystroke. This chapter describes the two different kinds of programs the command keyword can identify, and tells how CP/M-86 searches for command files on a diskette. It also introduces the control characters and function keys that direct CP/M-86 to perform various tasks.

Two Types of Commands

A command keyword identifies a program that resides either in memory as part of CP/M-86, or on a diskette as a program file. If a command keyword identifies a program in memory, it is called a Built-in command. If a command keyword identifies a program file on a diskette, it is called a Transient Utility or simply a utility.

Six Built-in commands and fifteen Transient Utilities are included with CP/M-86. If you are an experienced programmer, you can also write your own utilities that operate with CP/M-86.

Built-in Commands

Built-in commands are part of CP/M-86 and are always available for your use regardless of which diskettes you have in which drives. Built-in commands reside in memory as a part of CP/M-86 and therefore execute more quickly than the utilities. Chapter 4 gives you the operating details for the Built-in commands listed below:

DIR	displays a list of filenames from a diskette directory.
DIRS	displays a filename list of those files marked with the SYS attribute.
ERA	erases a filename from a diskette directory and releases the storage occupied by the file.

REN	lets you rename a file.
TYPE	writes the content of a character file to your console output device.
USER	lets you change from one user number to another.

Transient Utility Commands

A program that performs a Transient Utility command comes into memory only when you request it. Chapter 4 gives you operating details for the standard CP/M-86 Utilities listed below:

ASM86	translates 8086 assembly language programs into machine code form.
ASSIGN	lets you direct input and output to the peripheral devices attached to your computer.
COPYDISK	creates a copy of a diskette that may contain CP/M-86, program files, and data files.
DDT86	helps you check out your programs and interactively correct "bugs" and programming errors.
ED	lets you create and alter character files for access by various commands and application programs.
FUNCTION	programs the Function Keys and the Numeric Keypad on your keyboard.
GENCMD	uses the output of ASM86 to produce an executable command file.
HELP	displays information on how to use CP/M-86 commands.
NEWDISK	formats a factory-fresh diskette for use with CP/M-86.

PIP	combines and copies files.
PROTOCOL	establishes the "handshaking protocols" used by the serial interfaces.
SPEED	sets up your computer's asynchronous communications adaptor to communicate with other machines.
STAT	lets you examine and alter file status.
SUBMIT	sends a file of commands to CP/M-86 for execution.
TOD	sets the date and time displayed in the status line at the bottom of your display.

How CP/M-86 Searches for Commands

If a command keyword does not identify a Built-in command, CP/M-86 looks on the default or specified disk for a program file. It looks for a filename equal to the keyword and a filetype of CMD. For example, suppose you type the command line:

A>ED MYPROG.TXT

CP/M-86 goes through these steps to execute the command:

1. CP/M-86 first finds that the keyword ED does not identify one of the Built-in commands.

2. CP/M-86 searches for the utility program file ED.CMD in the directory of the default drive. If it does not find the file under the current user number, it looks under user number 0 for ED.CMD with the SYS attribute.

3. When CP/M-86 locates ED.CMD, it copies the program to memory and passes control to ED.

4. ED runs in memory until you enter a command to exit ED.

5. CP/M-86 types the system prompt and waits for you to type another command line.

If CP/M-86 cannot find either a Built-in or a Transient Utility, it reports a keyword error by repeating the command line you typed on your screen, followed by a question mark. This tells you that:

- The keyword is not a Built-in command and
- No corresponding .CMD file appears under the current user number or with the SYS attribute under user 0 on the default or specified drive.

For example, suppose your default diskette contains only standard CP/M-86 utilities and you type the command line:

A>EDIT MYPROG.A86

Here are the steps that CP/M-86 goes through to report the error:

1. CP/M-86 first examines the keyword EDIT and finds that it is not one of the Built-in commands.

2. CP/M-86 then searches the directory of the default diskette, first under the current user number for EDIT.CMD and then under user 0 for EDIT.CMD with the SYS attribute.

3. When the file cannot be found, CP/M-86 writes the message

   EDIT?

   at the screen to tell you that the command cannot be executed.

4. CP/M-86 displays the system prompt and waits for you to type another command line.

Control Character and Function Key Commands

You can direct CP/M-86 to perform certain functions just by striking a special key. Using the Control Character commands, you can tell CP/M-86 to start and stop screen scrolling, suspend current operations, or echo the screen display at the printer. The table below summarizes Control Character commands.

Ctrl-C	ends the currently operating program, or, if typed after the system prompt, initializes the system and default drives and sets all drives to R/W status.
Ctrl-P	tells CP/M-86 to send screen output characters to the printer too, or, if the printer is already echoing the screen, tells CP/M-86 to stop sending screen output characters to the printer.
Ctrl-S	toggles screen scrolling. If a display at your screen rolls by too quickly for you to read it, press Ctrl-S. Press any key or Ctrl-S again to continue the display.
Ctrl-Break	exits the current program, initializes all the drives and sets them to R/W, and clears printer echo if Ctrl-P has been pressed. Also interrupts and terminates a list of commands being executed by SUBMIT.
Ctrl-numlock	suspends current operation, similar to Ctrl-S. Press any key to continue.
⇧ Prntsc	lists the contents of the screen at the printer.

When you start CP/M-86, the Function Keys at the left of your keyboard are programmed with the commands shown in the table below. In most cases you can simply press one key to execute a complete CP/M-86 command. However, because F6 can damage data, you must press the Enter key to complete the command. In the table below, the Enter keystroke is represented by the symbol <CR>. You can reprogram these keys using the FUNCTION utility.

F1	dir<CR>
F2	dir b:<CR>
F3	stat<CR>
F4	stat b:<CR>
F5	pip<CR>
F6	pip b:=a:*.*[v]
F7	stat *.*<CR>
F8	stat b:*.*<CR>

When you press the Numlock key, you can use numeric keypad keys as special function keys. Listed below are the default functions of these keys. Note that a few of the keys, PgUp, PgDn and Ins, are unprogrammed. You can program them and change the functions of the other keys with the FUNCTION utility.

Home	<ESC>H
↑	<ESC>A
PgUp	
<—	<ESC>D
—>	<ESC>C
End	END
↓	<ESC>B
PgDn	
Ins	
Del	<DEL>

CHAPTER 4. COMMAND SUMMARY

Contents

ommand Summary

This chapter describes how we show the parts of a file specification in a command line. It also describes the notation used to indicate optional parts of a command line and other format notation. The remainder of the chapter provides a handy reference for all standard CP/M-86 commands.

Built-in and Transient Utility commands are intermixed in alphabetical order. Each command is listed, followed by a short explanation of its operation with examples. More complicated commands are described later in detail in separate chapters.

et's get past the formalities

You can see that there are several parts in a file specification that we must distinguish. To avoid confusion, we give each part a formal name that is used when we discuss command lines.

The three parts of a file specification are:

drive specifier	the optional diskette drive, A, B, C, or D that contains the file or group of files to which you are referring. If a drive specification is included in your command line, it must be followed by a colon.
filename	the one to eight character first name of a file or group of files.
filetype	the optional one to three character family name of a file or group of files. If the filetype is present, it must be separated from the filename by a period.

We use the following form to write the general form of a file specification:

d:filename.typ

In the above form, "d:" represents the optional drive specification, "filename" represents the one to eight character filename, and ".typ" represents the optional one to three character filetype. Valid combinations of the elements of a CP/M-86 file specification are shown below.

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

If you do not include a drive specification, CP/M-86 automatically supplies the default drive. If you omit the period and the filetype, CP/M-86 again automatically includes a filetype of three blanks.

We call this general form a "file specification." A file specification names a particular file or group of files in the directory of the on-line diskette given by the drive specifier. For example,

B:MYFILE.A86

is a file specification that indicates drive "B:", filename "MYFILE", and filetype "A86". We abbreviate "file specification" as simply

filespec

in the command format statements.

Some CP/M-86 command keywords accept wildcards in the filename and filetype parts of the command tail. For example,

B:MY*.A??

is a file specification with drive-name "B:", filename "MY*", and filetype "A??". This file specification may match several files in the directory.

You now understand command keywords, command tails, control characters, default drives, on-line drives, and wildcards. You also see how we use the formal names filespec, drive specification, filename, and filetype. These concepts give you the background necessary to compose complete command lines on your own. We devote the next section to introducing the various command line forms.

How commands are described

Chapter 4 lists the Built-in and Transient Utility commands in alphabetical order. Each command description is given in a specific form.

The description begins with the command keyword in upper-case.

When appropriate, an English phrase that is more descriptive of the command's purpose follows the keyword, in parentheses.

The "Format" section gives you one or more general forms to follow when you compose the command line.

The "Type" section tells you if the keyword is a Built-in or Transient Utility command. Built-in commands are always available for your use, while Transient Utility commands must be present on an on-line diskette as a CMD program file.

The "Purpose" section defines the general use of the command keyword.

The "Remarks" section points out exceptions and special cases.

The "Examples" section lists a number of valid command lines that use the command keyword.

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

Words in capital letters must be typed by you and spelled as shown, but you may use any combination of upper- or lower-case letters.

A lower-case word has a general meaning that is defined further in the text for that command. When you see the word "option", for example, you may choose from a given list of options.

You can substitute a number for "n."

The symbolic notation "d:", "filename", ".typ" and "filespec" has the general meanings described in the previous section.

You must include one or more space characters where a space is shown, unless otherwise specified. For example, the PIP options do not need to be separated by spaces.

Items enclosed within curly braces { } are optional. You can enter a command without the optional items. The optional items add effects to your command line.

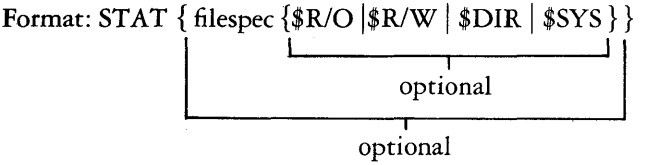
An ellipsis (...) tells you that the previous item can be repeated any number of times.

When you can enter one or more alternative items in the format line, a vertical bar, |, separates the alternatives. Think of this vertical bar as the "or" bar.

All other punctuation must be included in the command line.

Let's look at some examples of format notation. The CP/M-86 Transient Utility command STAT (status) displays the amount of free space in kilobytes for all on-line drives. It also displays the amount of space in kilobytes used by individual files. STAT can also assign the Read-Only (R/O) or Read-Write (R/W) and the System (SYS) or Directory (DIR) attributes to a file.

The Format section of the STAT command shows how the command line format notation is used:



This tells you that the command tail following the command keyword STAT is optional. STAT alone is a valid command, but you can include a file specification in the command line. Therefore, STAT filespec is a valid command. Furthermore, the file specification can be followed by another optional value selected from one of:

$R/O $R/W $DIR $SYS

Therefore,

STAT filespec $R/O

is a valid command.

Recall that in Chapter 3 you learned about wildcards in filenames and filetypes. The STAT command accepts wildcards in the file specification.

Using this format, we can construct several valid command lines:

STAT
STAT X.A86
STAT X.A86 $R/O
STAT X.A86 $SYS
STAT *.A86
STAT *.* $R/W
STAT X.* $DIR

The CP/M-86 command PIP (Peripheral Interchange Program) is the file copy program. PIP can copy information from your screen to the disk or printer. PIP can combine two or more files into one longer file. PIP can also rename files after copying them. Let's look at one of the formats of the PIP command line for another example of how to use command line notation.

Format: PIP dest-filespec = source-filespec{,filespec...}

For this example, "dest-filespec" is further defined as a destination file specification or peripheral device (printer, for example) that receives data. Similarly, "source-filespec" is a file specification or peripheral device (keyboard, for example) that transmits data. PIP accepts wildcards in the filename and filetype. (See the PIP command summary for details regarding other capabilities of PIP.) There are, of course, many valid command lines that come from this format. Some of them are shown below.

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

The ASM86 (Assembler) Command

Format: ASM86 filespec { $parameter-list }

Type: Transient Utility

Purpose: The ASM86 Utility converts 8088 and 8086 assembly language source statements into machine code form.

The operation of the ASM86 assembler is described in detail in Chapters 6 through 10.

Remarks: The filespec names the character file that contains an 8086 assembly language program to translate. If you omit the filetype, a filetype of A86 is assumed. The assembler uses the drive specification portion of the filespec as the destination drive for output files unless you include a parameter in the command tail to override this default.

The three output files produced by the assembler are given the filetypes listed below.

LST	contains the annotated source listing.
H86	contains the 8086 machine code in "hex" format.
SYM	contains all programmer-defined symbols with their program relative addresses.

The assembler assigns the same filename as the source filename to the LST, H86 and SYM files.

You control the assembly process by including optional parameters in the parameter-list. Each parameter is a single parameter letter followed by a single letter device name. The parameters can be separated by blanks, but each parameter letter must be followed immediately by the device name.

The parameter letters are A, H, P, S, and F. The device names are the letters A, B, C, and D, corresponding to the four drive letters. The letters X, Y, and Z have special meaning when used as device names:

X is the Screen.

Y is the Printer.

Z is zero output.

Use the A parameter letter to override the default drive specification to obtain the source file. The valid parameters are AA through AD.

Use the H parameter letter to override the default drive specification to receive the H86 file. Valid parameters are HA through HD, and HX, HY, and HZ.

Use the P parameter letter to override the default drive specification to receive the LST file. Valid parameters are PA through PD, PX, PY, and PZ.

Use the S parameter letter to override the default drive specification to receive the SYM file. Valid parameters are SA through SD, SX, SY, and SZ.

Use the F parameter letter to select the format of the "hex" output file. Valid parameters are FI and FD. The FI parameter selects Intel format "hex" file output. The FD parameter selects Digital Research format "hex" file output. FD is assumed if neither FI nor FD appear as a parameter. Use FI when the program is going to be combined with a program generated by an Intel compiler or assembler.

When conflicting parameters appear on the command line, the rightmost parameter prevails.

Examples: **A>ASM86 X**

The ASM86.CMD file must be on drive A. The source file X.A86 is read from drive A, and X.LST, X.H86, and X.SYM are written to drive A.

B>ASM86 X.A86 $PX

The ASM86.CMD file must be on drive B. The source file X.A86 is read from drive B. The listing is written to the screen, and the X.H86 and X.SYM files are placed on drive B.

A>ASM86 B:MYPROG $PY HC

The source file MYPROG.A86 is read from drive B, the listing is sent to the printer, the file MYPROG.H86 is written to drive C, and file MYPROG.SYM is placed on drive B.

A>B:ASM86 X $SZ

The ASM86.CMD file must be on drive B. The X.A86 file is read from drive A. The X.LST and X.H86 files are written to drive A. No X.SYM file is generated.

The ASSIGN (Assign Physical to Logical Device) Command

Format: ASSIGN logical-device type {physical-device}

Type: Transient Utility

Purpose: The ASSIGN Utility lets you assign a CP/M-86 logical device to one or more physical devices. For example, if you want a printout of the messages currently on your screen, you can use ASSIGN to send the messages to the printer as well as the screen. Specify a logical device and a type in the command tail when you want ASSIGN to display the current assignments for logical devices.

CP/M-86 supports four logical devices. The table below shows the CP/M-86 logical device names, followed by the formal name ASSIGN recognizes for each logical device. The last column lists the possible types (input and/or output) for each device.

CON:	CONSOLE	(INPUT and OUTPUT)
AXI:	AUXILIARY	(INPUT)
AXO:	AUXILIARY	(OUTPUT)
LST:	LIST	(OUTPUT)

CP/M-86 for the IBM Personal Computer supports eight physical devices. The table below lists each physical device with the formal name that ASSIGN recognizes as the name of each device. Then it shows the type (input or output) of logical device to which each physical device can be assigned.

Keyboard	KEYBOARD	(INPUT)
Screen	SCREEN	(OUTPUT)
Serial Port #0	SERIAL-0	(INPUT and OUTPUT)
Serial Port #1	SERIAL-1	(INPUT and OUTPUT)
Printer #0	PRINTER-0	(OUTPUT)
Printer #1	PRINTER-1	(OUTPUT)
Printer #2	PRINTER-2	(OUTPUT)
Dummy Device	DUMMY	(OUTPUT)

Normally, you assign the IBM Personal Computer keyboard tc CONSOLE INPUT and you assign the screen to CONSOLE OUTPUT. One purpose of the ASSIGN command might be to help you connect a remote console to your IBM Personal Computer through a telephone line and one of the serial ports. Use an ASSIGN command to assign CONSOLE INPUT and CONSOLE OUTPUT to SERIAL-0 or -1. If you do this, the remote operator takes full control of your IBM Personal Computer. However, if you assign only CONSOLE OUTPUT to the serial port, the remote operator can only monitor what you do with your IBM Personal Computer.

Remarks: A logical device can accept input from only one source, so you can assign only one physical device to a logical input device. However, you can direct the output of any logical device to any one or all of the physical output devices.

When processing a command tail, ASSIGN reads only the first character of the logical device, the first character of the type, and the first and last characters of the physical device. Therefore, you may abbreviate logical devices, types and physical devices as follows:

C	CONSOLE
A	AUXILIARY
L	LIST
I	INPUT
O	OUTPUT
KD	KEYBOARD
SN	SCREEN
S0	SERIAL-0
S1	SERIAL-1
P0	PRINTER-0
P1	PRINTER-1
P2	PRINTER-2
DY	DUMMY

If you do not specify a physical device in the command tail, ASSIGN displays the name of the physical device currently assigned to the logical device and type you did specify.

Use DUMMY as an OUTPUT device whenever you need to test a program but do not want to use its output. This can speed up program testing; for example, if a program normally sends output to the printer, you can eliminate the time required to print the output by assigning LIST OUTPUT to DUMMY.

xamples:
```
A>ASSIGN CONSOLE INPUT KEYBOARD
A>ASSIGN CONSOLE INPUT SERIAL-0
A>ASSIGN C I KD
A>ASSIGN C I S0
```

Use the first command when you want CP/M-86 to take its console input from the keyboard, or the second command to take it from Serial Port #0. The third and fourth commands are abbreviations of the first and second.

```
A>ASSIGN CONSOLE OUTPUT SCREEN
A>ASSIGN CONSOLE OUTPUT SCREEN
      SERIAL-0 PRINTER-1
A>ASSIGN C O SN
A>ASSIGN C O SN S0 P1
```

Use these commands to change the console output device assignment. Whenever an ASSIGN command specifies logical output, you can assign multiple physical devices as in the second command. After the second command is executed, whatever CP/M-86 sends to the console will also be sent to every device you specified. The third and fourth commands are abbreviations of the first and second.

```
A>ASSIGN LIST OUTPUT PRINTER-0
A>ASSIGN LIST OUTPUT SCREEN SERIAL-1 PRINTER-2
A>ASSIGN L O P0
A>ASSIGN L O SN S1 P2
```

Use these commands to change the physical device CP/M-86 uses for printouts. Because the type of the logical device is output, you can specify more than one output device. The third and fourth commands are abbreviations of the first and second.

```
A>ASSIGN CONSOLE INPUT
A>ASSIGN C I
```

The first command checks the current assignment of the CONSOLE INPUT logical device. In response to this command, ASSIGN sends a message similar to the following to your screen:

ASSIGN vn. 1.0 mm/dd/yy

Console Input assigned to Keyboard

The second command is an abbreviation of the first.

The COPYDISK (Copy Diskette) Command

ormat: COPYDISK

'ype: Transient Utility

'urpose: The COPYDISK Utility copies all the information on one diskette to another diskette, including the CP/M-86 system tracks if they are present on the source diskette.

Before copying to a brand-new diskette, you must first prepare it with the NEWDISK utility. If you copy to a used diskette, COPYDISK writes all the information from the source diskette over the information on the destination diskette.

Note: When using the COPYDISK Utility, both diskettes must be formatted to the same type. They must both be single-sided or double-sided diskettes.

ɬemarks: To display instructions on how to use COPYDISK, enter the keyword HELP with the command tail COPYDISK.

To successfully copy from one disk to another, you must make sure that your destination diskette is not write-protected. Check that there is no foil tab covering the write protect notch on the edge of your diskette before inserting the diskette into the destination drive.

If your IBM Personal Computer has only one diskette drive, COPYDISK keeps track of whether the source or the destination diskette should be in the drive. COPYDISK sends messages to the screen when it needs you to remove one diskette and insert the other.

COPYDISK is an exact track-for-track, sector-for-sector copy utility, and is the fastest way to copy an entire diskette. However, if many files have been created and erased on the source diskette, the records belonging to a particular file may be randomly placed on the diskette. In this case, it may be more efficient (although slower) to use PIP to copy the files and thus to put all the records in sequential order on the new diskette.

Examples: **A>COPYDISK**

Invoke COPYDISK and it prompts you for the source and destination disk. Use this form even if you have a single drive system and do not have drive B:. In this case, you should think of A: and B: as diskette names rather than as drive names. COPYDISK gives you instructions in the status line on your screen when it is time to change diskettes. In our example below, COPYDISK copies from your master diskette (diskette A:) to the new diskette (diskette B:). When invoked, COPYDISK displays the information in the first line of our example:

CP/M-86 Full Disk Copy Utility
Version 1.0

Enter Source Disk Drive (A-D) ? A

Destination Disk Drive (A-D) ? B

Copying disk A: to disk B:
Is this what you want to do (Y/N) ? Y
Copy started
Reading track nn (After read, new text appears)
Writing track nn (After write, next message is)
Verifying track nn
Copy completed.

Copy another disk (Y/N) ? N
Copy program exiting

A>

The DDT86 (Dynamic Debugging Tool) Command

Format: DDT86 { filespec }

Type: Transient Utility

Purpose: The DDT-86 Utility allows you to monitor and test programs developed for the 8086 and the 8088 processors.

DDT-86 responds to several single letter commands:

A	(Assemble)	Enter Assembly Language Statements
B	(Block Cmp)	Compare Blocks of Memory
D	(Display)	Display Memory in Hexadecimal and ASCII
E	(Execution)	Load Program for Execution
F	(Fill)	Fill Memory Block
G	(Go)	Begin Execution
H	(Hex)	Hexadecimal Sum and Difference
I	(Input)	Set Up Input Command Line
L	(List)	List Memory in Mnemonic Form
M	(Move)	Move Memory Block
R	(Read)	Read Disk File to Memory
S	(Set)	Set Memory Values
T	(Trace)	Trace Program Execution
U	(Untrace)	Monitor Execution without Trace

V	(Verify)	Show Memory Layout after Disk Read
W	(Write)	Write Content of Memory Block to Disk
X	(Examine)	Examine and Modify CPU Registers

The overall operation of DDT-86, along with each single letter command, is described in detail in Chapter 11.

Remarks: If the file specification is not included, DDT-86 is loaded into User Memory without a test program. You must not use the DDT-86 commands G, T, or U until you have first loaded a test program. The test program is usually loaded using E command.

If the file specification is included, both DDT-86 and the test program file specified by filespec are loaded into User Memory. Use G, T, or U to begin execution of the test program under supervision of DDT-86.

If the filetype is omitted from the file specification, a filetype of CMD is assumed.

DDT-86 cannot directly load test programs in hexadecimal (H86) format. You must first convert to command file form (CMD) using the GENCMD Utility.

To exit from DDT-86, press Ctrl-C.

Examples: **A>DDT86**

The DDT-86 Utility is loaded from drive A to User Memory. DDT-86 displays the "-" prompt when it is ready to accept commands.

A>B:DDT86 TEST.CMD

The DDT-86 Utility is loaded from drive B to User Memory. The program file TEST.CMD is then loaded to User Memory from drive A. DDT-86 displays the address at which the file was loaded and the "-" prompt.

The DIR (Directory) Built-in Command

Format:

DIR {d:}
DIR {filespec}

DIRS {d:}
DIRS {filespec}

Type: Built-in

Purpose: The DIR and DIRS Built-in commands display the names of files cataloged in the directory of an on-line diskette. The DIR Built-in lists the names of files in the current user number that have the Directory (DIR) attribute. DIR accepts wildcards in the file specification.

The DIRS command displays the names of files with the current user number that have the System (SYS) attribute. Therefore, even though you can access System (SYS) files that are stored in user 0 from any other user number on the same drive, DIRS only displays those user 0 files if the current user number is 0. DIRS accepts wildcards in the file specification.

Remarks: If the drive and file specifications are omitted, the DIR command displays the names of all files with the DIR attribute on the diskette in the default drive and current user number. Similarly, DIRS displays the SYS files.

If the drive specification is included, but the filename and filetype are omitted, the DIR command displays the names of all DIR files in the current user on the diskette in the specified drive. DIRS displays the SYS files.

If the file specification contains wildcard characters, all file names that satisfy the match are displayed on the screen.

If no filenames match the file specification, or if no files are cataloged in the directory of the diskette in the named drive, the DIR command displays the message:

NO FILE

If system (SYS) files reside on the specified drive, DIR displays the message:

SYSTEM FILE(S) EXIST

If non-system (DIR) files reside on the specified drive, DIRS displays the message:

NON-SYSTEM FILE(S) EXIST

You cannot use a wildcard character in a drive specification.

Examples: **A>DIR**

All DIR files cataloged in the current user number in the directory of the diskette mounted in drive A are displayed on the screen.

A>DIR B:

All DIR files in the current user number on the diskette in drive B are displayed on the screen.

A>DIR B:X.A86

If the file X.A86 is present on the diskette in drive B, the DIR command displays the name X.A86 on the screen.

A>DIR *.A86

All DIR files with filetype A86 in the current user number on the diskette in drive A are displayed on the screen.

B>DIR A:X*.C?D

All DIR files in the current user number on the diskette in drive A whose filename begins with the letter X, and whose three character filetype contains the first character C and last character D are displayed on the screen.

A>DIRS

Displays all files in the current user number on drive A that have the system (SYS) attribute.

A>DIRS *.CMD

Displays all files with the current user number on drive A with a filetype of CMD that have the system (SYS) attribute.

The ED (Character File Editor) Command

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

Type: Transient Utility

Purpose: The ED Utility lets you create and edit a diskette file.

The ED Utility is a "line-oriented" and "context" editor. This means that you create and change character files line-by-line, or by referencing individual characters within a line.

The ED Utility lets you create or alter the file named in the file specification.

The ED Utility uses a portion of your User Memory as the active text "Buffer" where you add, delete, or alter the characters in the file. You use the A command to read all or a portion of the file into the Buffer. You use the W or E command to write all or a portion of the characters from the Buffer back to the file.

An imaginary "character pointer," called CP, is at the beginning of the Buffer, between two characters in the Buffer, or at the end of the Buffer.

You interact with the ED Utility in either "command" or "insert" mode. ED displays the "*" prompt on the screen when ED is in command mode. When the "*" appears, you can enter the single letter command that reads text from the Buffer, moves the CP, or changes the ED mode of operation.

a (Append)	Load lines of text to the buffer.
b (Begin/Bottom)	Move CP to beginning or bottom of the buffer.
c (Character)	Move CP to right or left by characters.
d (Delete)	Delete characters to the right or left of the CP.
e (End)	End edit session and write buffer.

f (Find)	Move CP to a character sequence.
h (Head)	Write buffer, move to beginning of the file.
i (Insert)	Change from command to insert mode.
j (Juxtapose)	Place characters next to each other.
k (Kill Lines)	Remove full lines above or below the CP
l (Move Lines)	Move CP up or down by full lines.
m (Macro)	Repetitive evaluation of a command group.
n (Next)	Find next occurrence, automatic buffer fill.
o (Original)	Discard current edit, restart with original.
p (Page)	Move CP up or down by 23 full lines.
q (Quit)	Discard current edit, no changes to the file.
r (Read)	Read LIB or transferred text.
s (Substitute)	Substitute one character sequence for another.
t (Type)	Type full lines on the display.
u (Upper Case)	Translate all input to upper case.
v (Verify)	Set line number mode, or show buffer space.
w (Write)	Write lines of text from the buffer.
x (Transfer)	Transfer lines to or from a temporary file.
z (Sleep)	Delay command execution.

Chapter 5 gives a detailed description of the overall operation of the ED Utility and the use of each command.

Remarks: Include the second file specification only if the file named by the first file specification is already present and you do not want the original file replaced. The file named by the second file specification receives the altered text from the first file, which remains unchanged.

If the second file specification contains only the drive specification, the second filename and filetype become the same as the first filename and filetype.

If the file given by the first file specification is not present, the ED Utility creates the file and writes the message:

NEW FILE

If the second filespec is omitted, the original file is preserved by renaming its filetype to BAK before it is replaced. If you issue an ED command line that contains a filespec with filetype BAK, ED creates and saves your new edited version of the BAK file, but ED deletes your source file, leaving no back-up. If you want to save the original BAK file, use the REN command first to change the filetype from BAK, so that ED can rename it to BAK.

If you include the optional second filespec and give it the same name as the first filespec, ED again creates and saves your new edited version of the output filespec, but has to delete the original input filespec because it has the same name as the output file. You cannot, of course, have two files with the same name in the same user number on the same drive.

If the file given by the first filespec is already present, you must issue the A command to read portions of the file to the Buffer. If the size of the file does not exceed the size of the Buffer, the command:

#a

reads the entire file to the Buffer.

The i (Insert) command places the ED Utility in insert mode. In this mode, any characters you type are stored in sequence in the Buffer starting at the current CP.

Any single letter commands typed in insert mode are not interpreted as commands, but are simply stored in the Buffer. You return from insert mode to command mode by typing Ctrl-Z.

The single letter commands are normally typed in lower-case. The commands that must be followed by a character sequence end with Ctrl-Z if they are to be followed by another command letter.

Any single letter command typed in upper-case tells ED to internally translate to upper-case all characters up to the Ctrl-Z that ends the command.

When enabled, line numbers that appear on the left of the screen take the form:

nnnnn:

where nnnnn is a number in the range 1 through 65535. Line numbers are displayed for your reference and are not contained in either the Buffer or the character file. The screen line starts with

when the CP is at the beginning or end of the Buffer.

Examples: **A>ED MYPROG.A86**

If not already present, this command line creates the file MYPROG.A86 on drive A. The command prompt

:*

appears on the screen. This tells you that the CP is at the beginning of the Buffer. If the file is already present, issue the command

:*#a

to fill the Buffer. Then type the command

:*0p

to fill the screen with the first 23 lines of the Buffer. Type the command

:*e

to stop the ED Utility when you are finished changing the character file. The ED Utility leaves the original file unchanged as MYPROG.BAK and the altered file as MYPROG.A86.

A>ED MYPROG.A86 B:NEWPROG.A86

The original file is MYPROG.A86 on the default drive A. The original file remains unchanged when the ED Utility finishes, with the altered file stored as NEWPROG.A86 on drive B.

A>B:ED MYPROG.A86 B:

The ED.CMD file must be on drive B. The original file is MYPROG.A86 located on Drive A. It remains unchanged, with the altered program stored on drive B as MYPROG.A86.

The ERA (Erase) Built-in Command

Format: ERA filespec

Type: Built-In

Purpose: The ERA Built-in removes one or more files from the directory of a diskette. Wildcard characters are accepted in the command tail. Directory and data space are automatically reclaimed for later use by another file.

Remarks: Use the ERA command with care since all files that satisfy the file specification are removed from the diskette directory.

Command lines that take the form

ERA {d:}* *

require your acknowledgment since they reclaim all file space. You'll see the message:

All (Y/N)?

Respond with "y" if you want to remove all files, and "n" if you want to avoid erasing any files.

You will see the message

NO FILE

on the screen if no files match the file specification.

Examples: **A>ERA X.A86**

This command removes the file X.A86 from the diskette in drive A.

A>ERA *.PRN

All files with the filetype PRN are removed from the diskette in drive A.

B> ERA A:MY *.*

Each file on drive A with a filename that begins with MY is removed from the diskette.

A>ERA B:*.*

All files on drive B are removed from the diskette. To complete the operation, you must respond with a "y" when th ERA command displays the message:

All (Y/N)?

The FUNCTION (User-Assigned Function Keys) Command

Format: FUNCTION

.

.

Function key?: key
—> command line {Ctrl-G} <cr>

Type: Transient Utility

Purpose: The FUNCTION Utility lets you assign any function you want to any of your IBM Personal Computer function and numeric keypad keys. For example, if you want the Function Key labelled F2 to display the directory for drive B when you press F2, then you must assign the directory command to the F2 key using the FUNCTION Utility. When invoked, FUNCTION first displays a list of all the function keys and any current function key assignments. It then displays the prompt "Function key?:". You must press the function key you wish to program or reprogram. Next, FUNCTION displays an —> prompt. You must enter the exact command you want the specified function key to reproduce. Press Enter to finish programming the key. The screen changes to show the new function key setting, and FUNCTION displays "Function key?:" again and waits for you to press another function key. Use Ctrl-C after the "Function key?:" or —> prompt to exit the FUNCTION program.

Remarks: To include a carriage return into your actual command, type a Ctrl-G before pressing the Enter key.

To include a line feed into your command, type a Ctrl-L before pressing the Enter key.

The specified command cannot include dollar signs ($).

Use CP/M-86 editing characters to correct mistakes in the command line.

The specified command for function keys one through ten can be up to eighteen characters long.

The specified commands for the numeric keypad can be up to four characters long.

In the FUNCTION screen display, a <CR> means there is a carriage return embedded in the command, put there by a Ctrl-G.

If you press a key other than a function key in response to the prompt, FUNCTION displays the following message:

Not a programmable key–try again.

To erase a function key assignment, press ENTER in response to the —> prompt.

Examples:

```
A>Function <cr>
FUNCTION utility, v 1.0 mm/dd/yy
 F1:dir <CR>
 F2:dir b:<CR>
 F3:stat<CR>
 F4:stat b:<CR>
 F5:pip<CR>
 F6:pip b:=a:*.*[v]
 F7:stat *.* <CR>
 F8:stat b:*.*<CR>
 F9:
F10:

   Home:  <ESC>H
      ↑:  <ESC>A
   PgUp:
    <—:  <ESC>D
    —>:
    End:  END
      ↓:  <ESC>B
   PgDn:
    Ins:
    Del:  <DEL>B

Function key?: F7
—>STAT *.* Ctrl-G ↵
Function key?: Ctrl-C
A>
```

In response to the FUNCTION command, FUNCTION displayed the existing function key assignments on the screen, followed by the "Function key?" prompt for a new function key assignment. The user entered the F7 key. FUNCTION then displayed the —> , prompting for the command string for the F7 key to represent. The user typed "STAT *.* Ctrl-G" and ↵. FUNCTION then displayed the new assignment on the screen. The new assignment for the F7 key is STAT *.* <CR>. Note that before completing the function key assignment by pressing the Enter key, the user typed a Ctrl-G to cause a carriage return to be included in the STAT *.* command.

The GENCMD (Generate CMD File) Command

Format: GENCMD filespec {8080 CODE[An,Bn,Mn,Xn] DATA[An,Bn,Mn,Xn] STACK[An,Bn,Mn,Xn] EXTRA[An,Bn,Mn,Xn]}

Type: Transient Utility

Purpose: The GENCMD Utility produces an executable CMD file that can be used as a Transient Utility command. The input to GENCMD is an H86 file produced by ASM86 or the Intel OH86 Utility. The operation of GENCMD is described in detail in the Appendixes. Included in the CMD file is a header containing information used at load time to determine memory requirements and segment register initialization.

An optional parameter list follows the file specification. In the parameter list, n represents a hexadecimal value up to four digits long.

The parameter list consists of up to five keywords with their corresponding list of values. The keywords are:

8080 CODE DATA STACK EXTRA

The keyword 8080 identifies the CMD file as an "8080 Memory Model" where segment registers are initialized to the same value. The remaining keywords define segment groups which have specific memory requirements. The values that define the memory requirements are separated by commas and enclosed in square brackets ([]) following each keyword. The bracketed keywords and related values must be separated from other keywords by at least one blank.

The values included in brackets are defined below, where n represents a hexadecimal constant of from one to four digits. The value n represents a "paragraph" value where each paragraph is 16 bytes long. The paragraph value corresponds to the byte value n * 16, or hhhh0 in hexadecimal.

An	Load group at absolute location n.
Bn	Begin group at address n in the hexadecimal file.
Mn	The group requires a minimum of n * 16 bytes.
Xn	The group can address up to n * 16 bytes.

Remarks: Use the 8080 keyword for programs from 8-bit microprocessors that are a direct conversion to CP/M-86 so that the program loads into an area with overlapping code and data segments. The code segment in the program must begin at location 100H.

Use An for any group that must be loaded at an absolute location in memory. Don't use an A value in the command tail unless you know that the requested absolute area will be available when the program runs.

Use Bn when your input Hex file does not contain information that identifies the segment groups. This value is not necessary when your H86 file is the output from the Digital Research ASM-86 assembler, unless the ASM-86 parameter FI was included.

Use the Mn value when you include a data segment that has an uninitialized data area at the end of the segment.

Use Xn when your program can use a larger data area, if available, than the minimum given by Mn.

Examples: **A>GENCMD MYFILE**

The file MYPROG.H86 is read from drive A. The output file MYPROG.CMD is written back to drive A. The input H86 file includes information that marks the program as operating with a particular memory model.

B>GENCMD MYFILE CODE[A40]DATA[M30,XFFF]

The file MYFILE.H86 is read from drive B. The MYFILE.CMD output file is written to drive B. The code group must be loaded at location 400 hexadecimal. The data group requires a minimum of 300 hexadecimal bytes, but if available, the program can use up to FFF0 bytes.

The HELP (Help) Command

Format: HELP {topic} {subtopic1 subtopic2 . . . subtopic8}{[P]}

Type: Transient Utility

Purpose: The HELP command provides summarized information for all of the CP/M-86 commands described in this manual. HELP with no command tail displays a list of all the available topics. HELP with a topic in the command tail displays information about that topic, followed by any available additional subtopics. HELP with a topic and a subtopic displays information about the specific subtopic.

Remarks: After HELP displays the information for your specified topic, it displays the special prompt HELP> on your screen. You can continue to specify topics for additional information, or simply press the Enter key to return to the CP/M-86 system prompt.

You can abbreviate the names of topics and subtopics. Usually one or two letters is enough to specifically identify the topics.

HELP with the [P] option prevents the screen display from stopping every 23 lines.

Examples: **A>HELP**

The above command displays a list of topics for which help is available.

A>HELP STAT

The above command displays general information about the STAT command. It also displays any available subtopics.

A>HELP STAT OPTIONS

The above command includes the subtopic "options". In response, HELP displays information about options associated with the STAT command.

A>HELP ED

The above command displays general information about the ED Utility.

HELP>ED

The above example shows how to enter a topic and subtopic following the program's internal prompt, HELP>

Note: You must type the topic AND the subtopic; otherwise the HELP program cannot know which main topic you are referencing.

The NEWDISK (Create New Diskette) Command

Format: NEWDISK d: $S | $DS
NEWDISK d: $N | $DN

Type: Transient Utility

Purpose: The NEWDISK Utility prepares a new diskette for use in your IBM Personal Computer. NEWDISK initializes the new diskette by writing a known pattern of information on every sector of the diskette and tests diskette surface usability before you try to store data on it.

If you enter the $S parameter, NEWDISK creates a new system diskette on the drive named in the command by copying the CP/M-86 operating system onto the new diskette. NEWDISK assumes that CP/M-86 is on the disk in drive A. Therefore, to create a new system diskette, NEWDISK requires that the diskette in drive A: be an existing system diskette and that the new diskette be on a different drive. Even if you have a single drive, enter the command: NEWDISK B: $S. NEWDISK prompts you to change diskettes when necessary. When you change disks, NEWDISK treats the second disk as if it were in drive B. In this case, it is useful to think of the diskettes as diskette A and diskette B.

Enter the $N parameter if you want to create a normal or non-system diskette.

Use $DN to format a double-sided diskette without the CP/M-86 operating system. Use $DS to create a double-sided diskette with the CP/M-86 operating system on it.

Note: After changing between single-sided and double-sided diskettes, be sure to type a Ctrl-C immediately following the system prompt for CP/M-86 to read the changed diskette properly.

Remarks: To display information on how to use NEWDISK, enter the keyword HELP with NEWDISK as the command tail.

If you run NEWDISK on a used diskette, NEWDISK destroys all information previously written on the diskette.

Examples: **A>NEWDISK B: $S**

In response to this command, NEWDISK displays the following messages as it formats the diskette in drive B and copies the CP/M-86 operating system from drive A to drive B. You must enter a "y" to start formatting. As NEWDISK formats the diskette, it displays each track number, represented by 039 in the messages below.

```
NEWDISK vn 1.0 mm/dd/yy

Disk B will be formatted.

ALL DATA WILL BE ERASED FROM THE DISK.

Is this what you want (y/n)? y

Disk format in progress.
039

Format complete.
Press Control-C to exit, or
ENTER to format another disk.
--
```

If you have a single drive system, NEWDISK sends a message to the screen when it is time to change diskettes. The message appears on the Status Line and looks like this:

```
Put Disk B in A; ENTER to continue
```

A>NEWDISK B: $DS

The command shown above formats a double-sided diskette on drive B and puts the CP/M-86 operating system on the formatted diskette. As NEWDISK formats the diskette, it displays each track number, terminating with a 079.

The PIP (Peripheral Interchange Program–Copy File) Command

Format: PIP dest-file{[Gn]} | dev = src-file{[options]} | dev{[options]}

Type: Transient Utility

Purpose: The PIP Utility copies one or more files from one disk and/or user number to another. PIP can rename a file after copying it. PIP can combine two or more files into one file. PIP can also copy a character file from disk to the printer or other auxiliary logical output device. PIP can create a file on disk from input from the console or other logical input device. PIP can transfer data from a logical input device to a logical output device. Hence the name Peripheral Interchange Program.

Note: PIP can copy from a single-sided diskette to a double-sided diskette or vice versa. However, before attempting to read or write to a diskette of a different format, you must log in the new diskette with a Ctrl-C immediately following the system prompt. If you forget to log in the diskette, PIP reads from the source diskette improperly.

Single File Copy

Format: PIP d:{[Gn]} = source-filespec{[options]}

PIP dest-filespec{[Gn]} = d:{[options]}

PIP dest-filespec{[Gn]} = source-filespec{[options]}

Purpose: The first form shows the simplest way to copy a file. PIP looks for the file named by source-filespec on the default or optionally specified drive. PIP copies the file to the drive specified by d: and gives it the same name as source-filespec. If you want, you can use the [Gn] option to place your destination file (dest-filespec) in the user number specified by n. The only option recognized for the destination file is [Gn]. Several options can be combined together for the source file specification (source-filespec). See the section on PIP options.

The second form is a variation of the first. PIP looks for the file named by dest-filespec on the drive specified by d:, copies it to the default or optionally specified drive, and gives it the same name as dest-filespec.

The third form shows how to rename the file after you copy it. You can copy it to the same drive and user number, or to a different drive and/or user number. Rules for options are the same. PIP looks for the file specified by source-filespec, copies it to the location specified in dest-filespec, and gives it the name indicated by dest-filespec.

Remarks: Before you start PIP, be sure that you have enough free space in kilobytes on your destination diskette to hold the entire file or files that you are copying. Even if you are replacing an old copy on the destination diskette with a new copy, PIP still needs enough room for the new copy before it deletes the old copy. (See the STAT Utility.)

Data is first copied to a temporary file to ensure that the entire data file can be constructed within the space available on the diskette. PIP gives the temporary file the filename specified for the destination, with the filetype $$$. If the copy operation is successful, PIP changes the temporary filetype $$$ to the filetype specified in the destination.

If the copy operation succeeds and a file with the same name as the destination file already exists, the old file with the same name is erased before renaming the temporary file.

File attributes (SYS,DIR,R/W,R/O) are transferred with the files.

If the existing destination file is set to Read-Only (R/O), PIP asks you if you want to delete it. Answer Y or N. Use the W option to write over Read-Only files.

You can include PIP options following each source name (see the section on PIP Options). There is one valid option ([Gn]–go to user number n) for the destination file specification. Options are enclosed in square brackets. Several options can be included for the source files. They can be packed together or separated by spaces. Options can verify that a file was copied correctly, allow PIP to read a file with the system (SYS) attribute, cause PIP to write over Read-Only files, cause PIP to put a file into or copy it from a specified user number, transfer from lower to upper case, and much more.

Examples: **A>PIP B:=A:oldfile.dat**

A>PIP B:oldfile.dat = A:

Both forms of this command cause PIP to read the file oldfile.dat from drive A and put an exact copy of it onto drive B. This is called the short form of PIP, because the source or destination names only a drive and does not include a filename. When using this form you cannot copy a file from one drive and user number to the same drive and user number. You must put the destination file on a different drive or in a different user number. See the section on PIP options, and the section on the USER Utility. The second short form produces exactly the same result as the first one. PIP simply looks for the file oldfile.dat on drive A, the drive specified as the source.

A>PIP B:newfile.dat = A:oldfile.dat

This command copies the file oldfile.dat from drive A to drive B and renames it to newfile.dat. The file remains as oldfile.dat on drive A. This is the long form of the PIP command, because it names a file on both sides of the command line.

A>PIP newfile.dat = oldfile.dat

Using this long form of PIP, you can copy a file from one drive and user number (usually user 0 because CP/M-86 automatically starts out in user 0–the default user number) to the same drive and user number. This effectively gives you two copies of the same file on one drive and user number, each with a different name.

A>PIP B:PROGRAM.BAK = A:PROGRAM.DAT[G1]

The command above copies the file PROGRAM.DAT from user 1 on drive A to the currently selected user number on drive B (usually user 0), and renames the filetype on drive B to BAK.

B>PIP program2.dat = A:program1.dat[E V G3]

In this command, PIP copies the file named program1.dat on drive A and echoes [E] the transfer to the console, verifies [V] that the two copies are exactly the same, and gets [G3] the file program1.dat from user 3 on drive A. Because there is no drive specified for the destination, PIP automatically copies the file to the default drive and user number, in this case drive B.

Multiple File Copy

Format: PIP d:{[Gn]} = {d:}wildcard-filespec{[options]}

Purpose: When you use a wildcard in the source specification, PIP copies qualifying files one-by-one to the destination drive, retaining the original name of each file. PIP displays the message "COPYING" followed by each file name as the copy operation proceeds. PIP issues an error message and stops the copy operation if the destination drive and user number are the same as those specified in the source.

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

This command causes PIP to copy all the files on drive A with the filetype CMD to drive B.

A>PIP B:=A:*.*

This command causes PIP to copy all the files on drive A to drive B. You can use this command to make a back-up copy of your distribution disk. Note, however, that this command does not copy the CP/M-86 system from the system tracks. COPYDISK copies the system for you.

A>PIP B:=A:PROG????.*

The above command causes PIP to copy all files beginning with PROG and having any filetype at all from drive A to drive B.

A>PIP B:[G1]=A:*.BAK

This command causes PIP to copy all the files with a filetype of BAK on drive A in the default user number (user 0 unless you have changed the user number with the USER command) to drive B in user number 1. Remember that the DIR, TYPE, ERA and other commands only access files in the same user number from which they were invoked. (See the USER Utility.)

Combining Files

Format: PIP dest-file{[Gn]}=src-file{[opt]},file{[opt]}{,file{[opt]}...}

Purpose: This form of the PIP command lets you specify two or more files in the source. PIP copies the files specified in the source from left to right and combines them into one file with the name indicated by the destination file specification. This procedure is called file concatenation. You can use the [Gn] option after the destination file to place it in the user number specified by n. You can specify one or more options for each source file.

Remarks: Most of the options force PIP to copy files character by character. In these cases PIP looks for a Ctrl-Z character to determine where the end of the file is. All of the PIP options force a character transfer except the following:

Gn,O,R,V, and W.

Copying data to or from logical devices also forces a character transfer.

During character transfers, you can terminate a file concatenation operation by striking any key on your keyboard.

When concatenating files, PIP searches only the last record of a file for the Ctrl-Z end-of-file character. However, if PIP is doing a character transfer, it stops when it encounters a Ctrl-Z character.

Use the [O] option if you are concatenating machine code files. The [O] option causes PIP to ignore embedded Ctrl-Z (end-of-file) characters, normally used to indicate the end of character files.

Examples: **A>PIP NEWFILE=FILE1,FILE2,FILE3**

The three files named FILE1, FILE2, and FILE3 are joined from left to right and copied to NEWFILE.$$$. NEWFILE.$$$ is renamed to NEWFILE upon successful completion of the copy operation. All source and destination files are on the diskette in the default drive A.

A>PIP B:X.A86 = Y.A86, B:Z.A86

The file Y.A86 on drive A is joined with Z.A86 from drive B and placed in the temporary file X.$$$ on drive B. The file X.$$$ is renamed to X.A86 on drive B when PIP runs to successful completion.

Copy Files to and from Auxiliary Devices

Format:

PIP dest-filespec {[Gn]}	= source-filespec {[options]}
AXO:	AXI: {[options]}
CON:	CON: {[options]}
PRN:	NUL:
LST:	EOF:

Purpose: This form is a special case of the PIP command line that lets you copy a file from a disk to a device, from a device to a disk or from one device to another. The files must contain printable characters. Each peripheral device has a "logical" name that identifies a source device that can transmit data or a destination device that can receive data. A colon (:) follows each logical device name so it cannot be confused with a filename. Strike any key to stop a copy operation that uses a logical device in the source or destination.

The logical device names are:

CON: Console: the physical device assigned to CON:. When used as a source, usually the keyboard; when used as a destination, usually the screen.

AXI: Auxiliary Input Device

AXO: Auxiliary Output Device

LST: The destination device assigned to LST: usually the printer.

There are three device names that have special meaning:

NUL: A source device that produces 40 hexadecimal zeros.

EOF: A source device that produces a single Ctrl-Z (the CP/M end-of-file mark).

PRN: The printer device with tab expansion to every eighth column, line numbers, and page ejects every 60th line.

Examples: **B>PIP PRN:=CON:,MYDATA.DAT**

Characters are first read from the console input device, generally the keyboard, and sent directly to your printer device. You type a Ctrl-Z character to tell PIP that keyboard input is complete. At that time, PIP continues by reading character data from the file MYDATA.DAT on drive B. Since PRN: is the destination device, tabs are expanded, line numbers are added, and page ejects occur every 60 lines.

A>PIP B:FUNFILE.SUE = CON:

If CON: is assigned to input, whatever you type at the console is written to the file FUNFILE.SUE on drive B. End the keyboard input by typing a Ctrl-Z.

A>PIP LST:=CON:

If CON: is assigned as input, whatever you type at the keyboard is written to the list device, generally the printer. Terminate input with a Ctrl-Z.

A>PIP LST: = B:DRAFT.TXT[T8]

The file DRAFT.TXT on drive B is written to the printer device. Any tab characters are expanded to the nearest column that is a multiple of 8.

A>PIP PRN: = B:DRAFT.TXT

The command above causes PIP to write the file DRAFT.TXT to the list device. It automatically expands the tabs, adds line numbers, and ejects pages after sixty lines.

Multiple Command Mode

Format: PIP

Purpose: This form of the PIP command starts the PIP Utility and lets you type multiple command lines while PIP remains in User Memory.

Remarks: PIP writes an asterisk (*) on your screen when ready to accept input command lines.

You can type any valid command line described under previous PIP formats following the asterisk prompt.

Terminate PIP by pushing only the Enter key following the asterisk prompt. The empty command line tells PIP to discontinue operation and return to the CP/M-86 system prompt.

Note: This form of PIP lets you change SOURCE diskettes of the SAME FORMAT between commands. You cannot, however, change from a single-sided to a double-sided source diskette, or vice versa. If you do, PIP reads the source file improperly. You cannot change the destination diskette at all. If you do, CP/M-86 displays a Read-Only error message.

Examples:

```
A>PIP
*NEWFILE=FILE1,FILE2,FILE3
*APROG.CMD=BPROG.CMD
*A:=B:X.A86
*B:=*.*
*
```

This command loads the PIP program. The PIP command input prompt (*) tells you that PIP is ready to accept commands. The effects of this sequence of commands are the same as shown in the previous examples, where the command line is included in the command tail. PIP is not loaded into memory for each command.

Using Options With PIP

Purpose: Options enable you to process your source file in special ways. You can expand tab characters, translate from upper- to lower-case, extract portions of your text, verify that the copy is correct, and much more.

The PIP options are listed below, using "n" to represent a number and "s" to represent a sequence of characters terminated by a Ctrl-Z. An option must immediately follow the file or device it affects.The option must be enclosed in square brackets [ ]. For those options that require a numeric value, no blanks can occur between the letter and the value.

You can include the [Gn] option after a destination file specification. You can include a list of options after a source file or source device. An option list is a sequence of single letters and numeric values that are optionally separated by blanks and enclosed in square brackets [ ].

Dn Delete any characters past column n. This parameter follows a source file that contains lines too long to be handled by the destination device; for example, an 80-character printer or narrow console. The number n should be the maximum column width of the destination device.

E Echo transfer at console. When this parameter follows a source name, PIP displays the source data at the console as the copy is taking place. The source must contain character data.

F Filter form-feeds. When this parameter follows a source name, PIP removes all form-feeds embedded in the source data. To change form-feeds set for one page length in the source file to another page length in the destination file, use the F command to delete the old form-feeds and a P command to simultaneously add new form-feeds to the destination file.

Gn Get source from or Go to user number n. When this parameter follows a source name, PIP searches the directory of user number n for the source file. When it follows the destination name, PIP places the destination file in the user number specified by n. The number must be in the range 0 to 15.

H Hex data transfer. PIP checks all data for proper Intel hexadecimal file format. The console displays error messages when errors occur.

I Ignore :00 records in the transfer of Intel hexadecimal format file. The I option automatically sets the H option.

L Translate upper-case alphabetics in the source file to lower-case in the destination file. This parameter follows the source device or filename.

N Add line numbers to the destination file. When this parameter follows the source filename, PIP adds a line number to each line copied, starting with 1 and incrementing by one. A colon follows the line number. If N2 is specified, PIP adds leading zeroes to the line number and inserts a tab after the number. If the T parameter is also set, PIP expands the tab.

O Object file transfer for machine code (non-character and therefore non-printable) files. PIP ignores any Ctrl-Z ends-of-file during concatenation and transfer. Use this option if you are combining object code files.

Pn Set page length. n specifies the number of lines per page. When this parameter modifies a source file, PIP includes a page eject at the beginning of the destination file and at every n lines. If n = 1 or is not specified, PIP inserts page ejects every 60 lines. When you also specify the F option, PIP ignores form-feeds in the source data and inserts new form-feeds in the destination data at the page length specified by n.

Qs Quit copying from the source device after the string s. When used with the S parameter, this parameter can extract a portion of a source file. The string argument must be terminated by Ctrl-Z.

R Read system (SYS) files. Normally, PIP ignores files marked with the system attribute in the disk directory. But when this parameter follows a source filename, PIP copies system files, including their attributes, to the destination.

Ss Start copying from the source device at the string s. The string argument must be terminated by Ctrl-Z. When used with the Q parameter, this parameter can extract a portion of a source file. Both start and quit strings are included in the destination file.

Tn Expand tabs. When this parameter follows a source filename, PIP expands tab (Ctrl-I) characters in the destination file. PIP replaces each Ctrl-I with enough spaces to position the next character in a column divisible by n.

U Translate lower-case alphabetic characters in the source file to upper-case in the destination file. This parameter follows the source device or filename.

V Verify that data has been copied correctly. PIP compares the destination to the source data to ensure that the data has been written correctly. The destination must be a disk file.

W Write over files with R/O (Read-Only) attribute. Normally, if a PIP command tail includes an existing R/O file as a destination, PIP sends a query to the console to make sure you want to write over the existing file. When this parameter follows a source name, PIP overwrites the R/O file without a console exchange. If the command tail contains multiple source files, this parameter need follow only the last file in the list.

Z Zero the parity bit. When this parameter follows a source name, PIP sets the parity bit of each data byte in the destination file to zero. The source must contain character data.

Examples:

A>PIP NEWPROG.A86 = CODE.A86[L], DATA.A86[U]

This command constructs the file NEWPROG.A86 on drive A by joining the two files CODE.A86 and DATA.A86 from drive A. During the copy operation, CODE.A86 is translated to lower-case, while DATA.A86 is translated to upper-case.

A>PIP CON: = WIDEFILE.A86[D80]

This command writes the character file WIDEFILE.A86 from drive A to the console device, but deletes all characters following the 80th column position.

A>PIP B: = LETTER.TXT[E]

The file LETTER.TXT from drive A is copied to LETTER.TXT on drive B. The LETTER.TXT file is also written to the screen as the copy operation proceeds.

A>PIP LST: = B:LONGPAGE.TXT[FP65]

This command writes the file LONGPAGE.TXT from drive B to the printer device. As the file is written, form-feed characters are removed and reinserted at the beginning and every 65th line thereafter.

B>PIP LST: = PROGRAM.A86[NT8U]

This command writes the file PROGRAM.A86 from drive B to the printer device. The N parameter tells PIP to number each line. The T8 parameter expands tabs to every eighth column. The U parameter translates lower-case letters to upper-case as the file is printed.

A>PIP PORTION.TXT = LETTER.TXT[SDear Sir^Z QSincerely^Z]

This command abstracts a portion of the LETTER.TXT file from drive A by searching for the character sequence "Dear Sir" before starting the copy operation. When found, the characters are copied to PORTION.TXT on drive A until the sequence "Sincerely" is found in the source file.

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

This command copies all files with filetype CMD from drive A to drive B. The V parameter tells PIP to read the destination files to ensure that data were correctly transferred. The W parameter lets PIP overwrite any destination files that are marked as R/O (Read-Only). The R parameter tells PIP to read files from drive A that are marked with the SYS (System) attribute.

The PROTOCOL (Set Communications Protocol) Command

ɔrmat: PROTOCOL SERIAL-x protocol [message-length]

ype: Transient Utility

urpose: The PROTOCOL Utility lets you change the communications protocol for the serial ports of your computer. For example, if you have a slow printer connected to one of your computer's serial ports, you can prevent CP/M-86 from sending data to the printer faster than the printer can handle the data.

To use PROTOCOL, replace x in the format line above with 0 or 1. Use Serial-0 to change the communications protocol for Serial Port 0. Use Serial-1 to change the communications protocol for Serial Port 1. Then specify one of the three protocols described below: XON, ETX or NONE.

XON This protocol uses two special characters in the ASCII character set, XON and XOFF, as signals for "Transmission On" and "Transmission Off." Before each character is output from the computer to the peripheral device, the computer checks to see whether there is any incoming data from the peripheral. If the incoming character is XOFF, the computer suspends all further output until it receives an XON from the device, indicating that the device is again ready to receive more data.

ETX This protocol uses two other ASCII characters, ETX and ACK, which stand for "End of Transmission" and "Acknowledge." With this protocol you can define a message length as the last item in the command line. Use a value that is half the peripheral's buffer size (the default message length is 127 decimal). When the ETX/ACK protocol is in force, CP/M-86 sends the number of characters defined by message-length followed by an ETX character. CP/M-86 then waits for an ACK to be sent back from the peripheral before sending any further data.

NONE This option eliminates all protocols. CP/M-86 sends data to the device whether or not the device is ready tc receive it.

Remarks: You can use abbreviations and either upper- or lower-case in the above format. Use S0 or S1 for the serial device, and the beginning character X, E, or N for the protocol.

Enter PROTOCOL followed by Serial-0 or Serial-1 to display the current protocol setting for the specified port.

Enter HELP PROTOCOL to display information about the PROTOCOL Utility.

Examples: **A>PROTOCOL Serial-0 XON**

The above command assigns XON/XOFF protocol to Serial Port 0.

A>PROTOCOL Serial-1 ETX 2048

The above command assigns ETX/ACK protocol with a message length of 2048 to Serial Port 1.

A>PROTOCOL Serial-1 NONE

The above command eliminates all protocols from Serial Port 1.

A>PROTOCOL Serial-0 ETX 200

A>PROTOCOL S0 E 200

The two commands shown above set Serial Port 0 to ETX/ACK protocol, with a message length of 200 characters. The command may be abbreviated as shown in the second form of the command.

A>PROTOCOL Serial-1

Serial-1 set to Xon/Xoff Protocol

A>PROTOCOL S1

Serial-1 set to Xon/Xoff Protocol

Use the PROTOCOL command without any specified attributes to display the current protocol for a particular port. The two examples above show how to display the current assignment for Serial Port 1. The second form of the command line is an abbreviation of the first form.

The REN (Rename) Built-in Command

Format: REN {d:}newname{.typ} = oldname{.typ}

Type: Built-in

Purpose: The REN Built-in lets you change the name of a file that is cataloged in the directory of a diskette.

The filename oldname identifies an existing file on the diskette. The filename newname is not in the directory of the diskette. The REN command changes the file named by oldname to the name given as newname.

Remarks: REN does not make a copy of the file. REN changes only the name of the file.

If you omit the drive specifier, REN assumes the file to rename is on the default drive.

You can include a drive specification as a part of the newname. If both file specifications name a drive, it must be the same drive.

If the file given by oldname does not exist, REN displays the following message on the screen:

NO FILE

If the file given by newname is already present in the directory, REN displays the following message on the screen:

FILE EXISTS

Examples: **A>REN NEWASM.A86 = OLDFILE.A86**

The file OLDFILE.A86 changes to NEWASM.A86 on drive A.

B>REN A:X.PAS = Y.PLI

The file Y.PLI changes to X.PAS on drive A.

A>REN B:NEWLIST=B:OLDLIST

The file OLDLIST changes to NEWLIST on drive B. Since the second drive name, B: is implied by the first one, it is unnecessary in this example. The command line above has the same effect as the following:

A>REN B:NEWLIST=OLDLIST

The SPEED (Set Serial Port Attributes) Command

Format: SPEED SERIAL-x {attribute-1 attribute-2 ...}

Type: Transient Utility

Purpose: The SPEED Utility sets the attributes of a serial port. Normally you would use a serial port to set up communication between your IBM Personal Computer and some other equipment; for example, a remote terminal. Use a SPEED command to make sure that CP/M-86 sends and receives data the way the external equipment expects. You must know the external equipment's requirements for baud rate, parity, number of data bits and number of stop bits before you can effectively use the SPEED Utility.

The command tail for a speed command specifies the serial port number (Serial-0 or Serial-1) and zero or more attributes. To specify a serial port, replace x in the format line above with 0 for Serial Port 0 or 1 for Serial Port 1. If you do not specify an attribute, SPEED displays the current attributes for the specified port.

To specify a BAUD (Bits Per Second) RATE, enter one of the following baud rate values as an attribute:

9600
4800
1200
600
300
150
110

To specify PARITY, enter one of the following as an attribute:

PARITY-NONE
PARITY-EVEN
PARITY-ODD

To specify the number of STOP BITS, enter one of the following as an attribute:

STOP-1
STOP-2

To specify the number of DATA BITS, enter one of the following as an attribute:

BITS-7
BITS-8

Examples: **A>SPEED SERIAL-0 1200 PARITY-ODD**

The command line shown above sets the baud rate for Serial Port 0 to 1200, the parity to odd, and does not set stop bits or data bits.

A>SPEED SERIAL-1 150 STOP-1 BITS-7

The above example shows how to set the stop bits to 1, the data bits to 7, and the baud rate to 150 for Serial Port 1.

A>SPEED SERIAL-1

Serial-1 set to 9600 Parity-None
1 Stop Bits, 8 Data Bits

When you do not specify any attributes, SPEED displays the current attributes of the specified serial port, as shown above. The baud rate is set to 9600, the parity is set to none, the stop bits are set to 1, and the data bits are set to 8.

The STAT (Status) Command

Format: STAT d: = R/O
STAT {filespec {$R/O | R/W | SYS | DIR | SIZE}}
STAT {d:}DSK: | USR:

Type: Transient Utility

Purpose: The various forms of the STAT Utility command give you information about the disk drives and files associated with your IBM Personal Computer. STAT also lets you change the attributes of files and drives.

Remarks: The notation "R/W" tells you the drive is in a Read-Write state so that data can be both read from and written to the diskette.

The notation "R/O" tells you the drive is in a Read-Only state so that data can only be read from, but not written to, the diskette.

Drives are in a Read-Write state by default, and become Read-Only when you set the drive to R/O or when you change a diskette and forget to type a Ctrl-C.

Note that the STAT options following filespec can be preceded by a square bracket [, no delimiter or a dollar sign $ as shown above. Note also that the slash / can be omitted from RO and RW.

Set a Drive to Read-Only Mode

Format: STAT d: = R/O

Purpose: You may use this form of the STAT command to set the drive to Read-Only mode. Use Ctrl-C to reset a drive to Read-Write.

Example: **STAT B:= R/O**

The command line shown above sets drive B to Read-Only mode.

Free Space on Disk

Format: STAT {d:}

Purpose: STAT with no command tail reports the amount of free storage space that is available on all on-line diskettes. This form of the STAT command reports free space for only those diskettes that have been accessed since CP/M-86 was last started or reloaded. You can find the amount of free space on a particular diskette by including the drive specification in the command tail.

Remarks: If the drive specifier names a drive that is not on-line, CP/M-86 places the drive in an on-line status.

This form of the STAT command displays information on your screen in the following form:

d: RW, Free Space: nnK

where d is the drive specifier, and n is the number of kilobytes of storage remaining on the diskette in the drive specified by d.

Examples: Suppose you have two disk drives containing active diskettes. Suppose also that drive A has 16K (16,384) bytes of free space, while drive B has 32K (32,768) bytes of free space. Assume that drive A is marked R/W, and drive B is marked R/O. A STAT command displays the following messages on your screen:

A>STAT
A: RW, Free Space: 16K
B: RO, Free Space: 32K

Suppose drive B is set to Read-Only and has 98 kilobytes of storage that is free for program and data storage. A STAT command displays the following message is displayed on your screen:

```
A>STAT B:
B: RO, Free Space: 98K
```

Files–Display Space Used and Access Mode

Format: STAT filespec {$SIZE}

Purpose: This form of the STAT command displays the amount of space in kilobytes used by the specified file. It also displays the Access Mode of the file. STAT accepts wildcards in the filename and filetype part of the command tail. When you include a wildcard in your file specification, the STAT command displays a list of qualifying files from the default or specified drive, with their file characteristics, in alphabetical order.

CP/M-86 supports four file Access Modes:

R/O The file has the Read-Only attribute that allows data to come from the file, but the file cannot be altered.

R/W The file has the Read-Write attribute that allows data to move either to or from the file.

SYS The file has the "system" attribute. System files do not appear in DIR (directory) displays. Use DIRS to show System (SYS) files. Use the STAT command to display all files including those with the System attribute. The STAT command shows System files in parentheses.

DIR The file has the "directory" attribute and appears in DIR (directory) displays.

A file has either the R/O or R/W attribute, and either the SYS or DIR attribute. By default, and unless changed by the STAT command, a file has the R/W and DIR attributes.

This format for the STAT command produces a list of file characteristics under five headings.

The first column displays the number of records used by the file, where each record is 128 bytes in length. This value is listed on your screen under the column marked "Recs."

The second column displays the number of kilobytes used by the file, where each kilobyte contains 1,024 bytes. This value is listed under "Bytes."

The third column displays the number of directory entries used by the file. This value appears under the "FCBs" column. FCB (File Control Block) is another name for a directory entry.

The Access Modes are displayed under the "Attributes" column.

The file specification, consisting of the drive specification, filename, and filetype of the file appears under "Name" on your screen.

Remarks: If the drive specifier is included, and the corresponding drive is not active, CP/M-86 places the drive in an active status.

Use $SIZE to tell STAT to compute the "virtual file size" of each file. The virtual and real file size are identical for sequential files, but can differ for files written in random mode. When you use $SIZE, the additional column, marked "Size", is displayed on the screen. The value in this column represents the number of filled and unfilled records allotted to the file.

When you enter the command STAT *.*, STAT performs a directory verification to ensure that two files do not share the same disk space allocation. This means that the indicated file shares a portion of the disk with another file in the directory. If STAT finds a duplicate space allocation, it displays the following message:

Bad Directory on d:
Space Allocation Conflict:
d:filename.typ

STAT prints the name of the file containing doubly allocated space. More than one file can be listed. The recommended solution is to erase the listed files.

STAT does a complete directory verification whenever a wildcard character appears in the command tail.

Examples: **A>STAT MY*.***

This command tells STAT to display the characteristics of all files that begin with the letters MY, with any filetype at all. Assume that the following three files satisfy the file specification. The screen could display the following:

Recs	Bytes	FCBs	Attributes	Name
16	2K	1	Dir RW	B:MYPROG.A86
8	1K	1	Dir RO	B:MYTEST.DAT
32	18K	2	Sys RO	B:MYTRAN.CMD
Total:	21K	4		

B: RW, Free Space: 90K

A>STAT MY*.* $SIZE

This command causes the same action as the previous command, but includes the "Size" column in the display. Assume that MYTEST.DAT was written using random access from record numbers 8 through 15, leaving the first eight records empty. The virtual file size is 16 records, although the file only consumes only eight records. The screen appears as follows:

Size	Recs	Bytes	FCBs	Attributes	Name
16	16	2K	1	Dir RW	B:MYPROG.A86
16	8	1K	1	Dir RO	B:MYTEST.DAT
32	32	18K	2	Sys RO	B:MYTRAN.CMD
Total:		21K	4		

B: RW, Free Space: 90K

Set File Access Modes (Attributes)

Format: STAT filespec $R/O | $R/W | $SYS | $DIR

Purpose: This form of the STAT command lets you set the Access Mode for one or more files. The four Access Modes, described above, are:

R/O R/W SYS DIR

Remarks: If the drive named in the file specification corresponds to an inactive drive, CP/M-86 first places the drive in an on-line state.

A file can have either the R/O or R/W Access Mode, but not both. Similarly, a file can have either the SYS or DIR Access Mode, but not both.

Examples: **A>STAT LETTER.TXT $R/O**

This command sets the Access Mode for the file LETTER.TXT on the default drive to R/O. The following message appears on your screen if the file is present:

LETTER.TXT set to R/O

B>STAT A:*.CMD $SYS

This command sets the Access Mode for all files on drive A, with filetype CMD, to SYS. Given that the three command files PIP, ED, and ASM86 are present on drive A, the following message appears on your screen:

PIP.CMD set to SYS
ED.CMD set to SYS
ASM86.CMD set to SYS

Display Disk Status

Format: STAT {d:}DSK:

Purpose: This form of the STAT command displays internal information about your disk system for all on-line diskettes.

If a drive is specified, it is placed in an on-line status.

The information provided by this command is useful for more advanced programming, and is not necessary for your everyday use of CP/M-86.

Remarks: The third entry in the display, "Kilobyte Drive Capacity," shows the total amount of storage in kilobytes available on the specified diskette. After logging in a new diskette with a Ctrl-C following the system prompt, you can use this command to determine whether the diskette is single-sided or double-sided. Single-sided diskettes have a total of 156 kilobytes of disk storage. Double-sided diskettes have a total of 316 kilobytes of storage.

Examples: **A>STAT DSK:**

This STAT command displays information about drive A in the following form. STAT supplies numbers for n.

```
A:   Drive Characteristics
nnnn: 128 Byte Record Capacity
nnnn: Kilobyte Drive Capacity
nnnn: 32 Byte Directory Entries
nnnn: Checked Directory Entries
nnnn: 128 Byte Records/Directory Entry
nnnn: 128 Byte Records/Block
nnnn: 128 Byte Records/Track
nnnn: Reserved Tracks
```

A>STAT B:DSK:

This command produces the information shown in the previous example for drive B.

Display User Numbers With Active Files

Format: STAT {d:}USR:

Purpose: CP/M-86 assigns the current user number to files created under it. Except when using the [Gn] option with a PIP command, or accessing a file in user 0 that has been set to SYS with the STAT command, you can access only that file from the user number under which it was created. This form of the STAT command displays the current user number and the user numbers containing files on the diskette in the default or specified drive.

Examples: **A>STAT USR:**

```
A: Active User:          0
A: Active Files:         0 15
```

The STAT USR: command with no drive specification displays information for the default drive. This command displays the current user number and the user numbers that contain files in the form shown above. The display begins with the default or specified drive. STAT displays the current user number following "Active User." In this case it is user 0. The user numbers containing files are displayed following "Active Files." In the example above user 0 and user 15 contain files.

The SUBMIT (Batch Processing) Command

Format: SUBMIT filespec { parameters... }

Type: Transient Utility

Purpose: The SUBMIT Utility lets you group a set of commands together for automatic processing by CP/M-86.

Normally, you enter commands one line at a time. If you must enter the same sequence of commands several times, you'll find it easier to "batch" the commands together using the SUBMIT Utility. To do this, create a file and list your commands in this file. The file is identified by the filename, and must have a filetype of SUB. When you issue the SUBMIT command, SUBMIT reads the file named by filespec and prepares it for interpretation by CP/M-86.

The file of type SUB can contain any valid CP/M-86 commands. A command line cannot exceed 125 characters. The file of type SUB can contain a maximum of 128 command lines. The SUBMIT buffer allows up to 2048 characters in the input file.

If you want, you can include SUBMIT parameters within the SUB file that are filled in by values that you include in the command tail.

SUBMIT parameters take the form of a dollar sign ($), followed by a number in the range 0 through 9:

$0 $1 $2 $3 $4 $5 $6 $7 $8 $9

You can put these parameters anywhere in the command lines in your file of type SUB.

The SUBMIT Utility reads the command line following SUBMIT filespec and substitutes the items you type in the command tail for the parameters that you included in the file of type SUB. When the substitutions are complete, SUBMIT sends the file to CP/M-86 line by line as if you were typing each command.

Remarks: Each item in the command tail is a sequence of alphabetic, numeric, and/or special characters. The items are separated by one or more blanks.

The first word in the command tail takes the place of $1 in the SUB file, the second word replaces $2, and so forth, through the last item in the command tail. The filename of the SUB file replaces any $0 parameters in the SUB file.

SUBMIT creates a file named $$$.SUB that contains the command lines resulting from the substitutions.

If you type fewer items in the command tail than parameters in the SUB file, remaining parameters are not included in the temporary file.

If you type more items in the command tail than parameters in the SUB file, the items remaining in the command tail are ignored.

Batch command processing stops after reading the last line of the $$$.SUB file. Ctrl-Break stops the SUBMIT process. You can also stop batch processing before reaching the end of the SUB file by pressing any key after CP/M-86 issues the command input prompt, A>.

The file $$$.SUB is automatically removed when CP/M-86 stops reading the batched commands.

SUB files cannot contain nested SUBMIT commands. However, the last command in a SUB file can be a SUBMIT command that "chains" to another SUB file.

To include an actual dollar sign ($) in your file of type SUB, type two dollar signs ($$). The SUBMIT Utility replaces them with a single dollar sign when it substitutes a command tail item for a $ parameter in the SUB file.

Examples: **A>SUBMIT SUBFILE**

Assume the file SUBFILE.SUB is on the diskette in drive A, and that it contains the lines shown below.

```
DIR *.COM
ASM86 X $$SB
PIP LST: = X.PRN[T8D80]
```

The SUBMIT command shown above sends the sequence of commands contained in SUBFILE.SUB to CP/M-86 for processing. CP/M-86 first performs the DIR command and then assembles X.A86. When ASM-86 finishes, the PIP command line is executed.

```
A>SUBMIT B:ASMCOM   X   8  D80 SZ <— these command
                                    tail items

                    $1  $2  $3  $4 <— are assigned
                                    to these SUB
                                    file $n
                                    parameters
```

Assume that ASMCOM.SUB is present on drive B and that it contains the commands:

```
ERA $1.BAK
ASM86 $1 $$4

PIP LST: = $1.PRN[T$2 $3 $5]
```

The SUBMIT Utility reads this file and substitutes the items in the command tail for the parameters in the SUB file as follows:

```
ERA X.BAK
ASM86 X $SZ
PIP LST: = X.PRN[T8 D80]
```

These commands are executed from top to bottom by CP/M-86.

The TOD (Display and Set Time of Day) Command

Format: TOD {time-specification | P }

Type: Transient Utility

Purpose: The TOD Utility lets you examine and set the time of day.

The bottom line of your screen, called the Status Line, provides you with continuous date and time information. When you start CP/M-86, the date and time are set to the creation date of the system. Use TOD to change this initial value, at your option, to the current date and actual time.

A date is represented as a month value in the range 1 to 12, a day value in the range 1 to 31, depending upon the month, and a two-digit year value relative to 1900.

Time is represented as a twenty-four hour clock, with hour values from 00 to 11 for the morning, and 12 to 23 for the afternoon.

Use the command

TOD

to obtain the current date and time in the format:

weekday month/day/year hour:minute:second

For example, the screen might appear as

12/06/81 09:15:37

in response to the TOD command.

Use the command form

TOD time-specification

to set the date and time, where the time-specification takes the form:

month/day/year hour:minute:second

A command line in this form is:

TOD 02/09/81 10:30:00

To let you accurately set the time, the TOD Utility writes the message:

Press any key to set time

When the time that you give in the command tail occurs, press any key. TOD begins timing from that instant, and responds with a display in the form:

02/09/81 10:30:00

Use the command form

TOD P

to continuously print the date and time on the screen. This display appears in the status line also. However, you can assign the screen to another logical device on which you might want to display the date and time.

You can stop the continuous display by pressing any key.

Remarks: TOD checks to ensure that the time-specification represents a valid date and time.

You need not set the time-of-day for proper operation of CP/M-86.

Examples: **A>TOD**

This command writes the current date and time on the screen.

A>TOD 12/31/82 23:59:59

This command sets the current date and time to the last second of 1982.

The TYPE (Display File) Built-in Command

Format: TYPE filespec

Type: Built-in

Purpose: The TYPE Built-in displays the content of a character file on your screen.

Remarks: Tab characters occurring in the file named by filespec are expanded to every eighth column position of your screen.

Press any key on your keyboard to discontinue the TYPE command.

Make sure the file specification identifies a file containing character data.

If the file named by filespec is not present on an on-line diskette, TYPE displays the following message on your screen:

NO FILE

To list the file at the printer as well as on the screen, type a Ctrl-P before entering the TYPE command line. To stop echoing keyboard input at the printer, type a second Ctrl-P.

Examples: **A>TYPE MYPROG.A86**

This command displays the content of the file MYPROG.A86 on your screen.

A>TYPE B:THISFILE

This command displays the content of the file THISFILE from drive B on your screen.

The USER (Display and Set User Number) Built-in Command

Format: USER {number}

Type: Built-in

Purpose: The USER Built-in command displays and changes the current User Number. The diskette directory can be divided into distinct areas according to a "User Number."

Remarks: The default User Number is 0. When CP/M-86 starts, it assumes that 0 is the current User Number. Your IBM Personal Computer displays the current User Number in the status line, in the form

U = number

where "number" is a number in the range 0 through 15. Any files you create under this User Number are not accessible under any other User Number except through the PIP command. (See the G parameter of the PIP Utility.)

Use the command

USER

to display the current User Number.

Use the command form

USER number

where "number" is a number in the range 0 through 15, to change the current User Number.

See the command form

STAT USR:

to get a list of user numbers that have files associated with them.

Examples: **A>USER**
0

This command displays the current User Number.

A>USER 3

This command changes the current User Number to 3.

CHAPTER 5. ED, THE CP/M-86 CONTEXT EDITOR

Contents

ED, the CP/M-86 Context Editor

To do almost anything with a computer you need some way to enter data, some way to give the computer the information you want it to process. The programs most commonly used for this task are called "editors." They transfer your keystrokes at the keyboard to a disk file. CP/M-86's editor is named ED. Using ED, you can easily create and alter CP/M-86 text files.

The correct command format for invoking the CP/M-86 editor is given in the first section, "Starting ED." After starting ED, you issue commands that transfer text from a disk file to memory for editing. "ED Operation" details this operation and describes the basic text transfer commands that allow you to easily enter and exit the editor.

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

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

Starting ED

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

To start ED, enter its name after the CP/M-86 prompt. The command ED must be followed by a file specification, one that contains no wildcard characters, such as:

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

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

A>ED MYFILE.TEX B:

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

Optionally, you can send the edited material to a file with a different filename, as shown in the example below.

A>ED MYFILE.TEX YOURFILE.TEX

The file with the different filename cannot already exist or ED prints the following message and terminates.

Output File Exists, Erase It

The ED prompt, *, appears at the screen when ED is ready to accept a command, as shown below.

A>ED MYFILE.TEX
:*

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

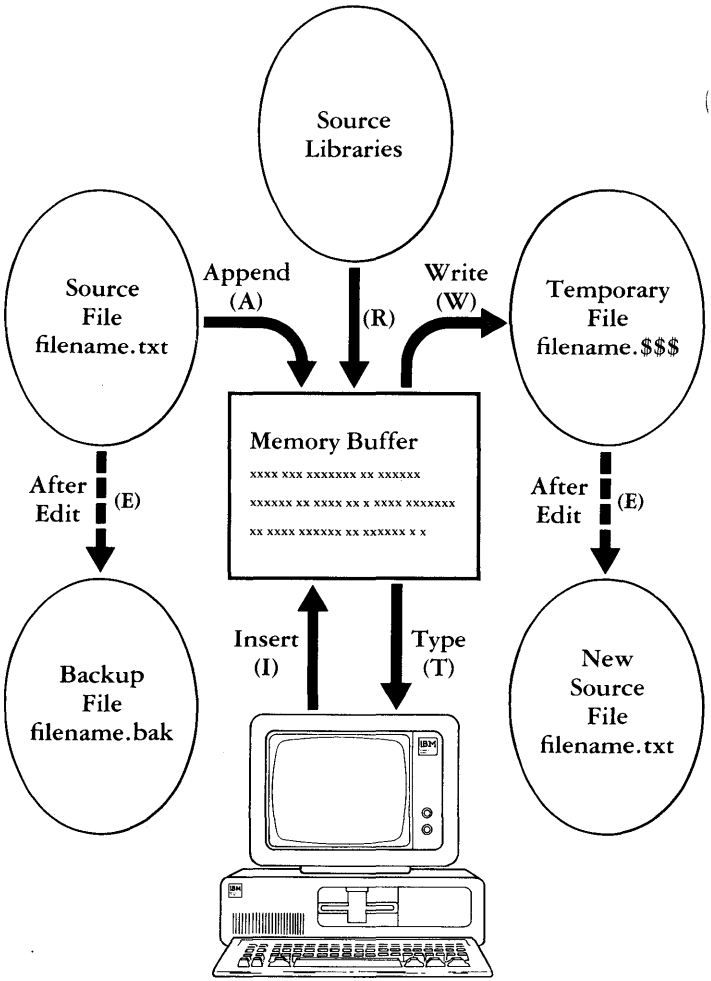
NEW FILE
:*

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

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

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

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



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

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

Appending Text into the Buffer

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

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

The V (Verify Line Numbers) Command

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

V, −V, 0V

Initially, the line number display is on. Use −V to turn it off. If the memory buffer is empty, or if the current line is at the end of the memory buffer, ED represents the line number as five blanks.

The 0V command prints the memory buffer statistics in the form:

free/total

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

```
46652/48253
```

The A (Append) Command

The A command appends (copies) lines from an existing source file into the memory buffer. The form of the A command is:

```
nA
```

where n is the number of unprocessed source lines to append into the memory buffer. If a pound sign, #, is given in place of n, then the integer 65535 is assumed. Because the memory buffer can contain most reasonably sized source files, it is often possible to issue the command #A at the beginning of the edit to read the entire source file into memory.

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

ED Exit

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

The W (Write) Command

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

```
nW
```

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

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

The E (Exit) Command

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

```
E
```

followed by a carriage return.

When you enter an E command, ED first writes all data lines from the buffer and the original source file to the .$$$ file. If a .BAK file exists, ED deletes it, then renames the original file with the .BAK filetype. Finally, ED renames the .$$$ file from filename.$$$ to the original filetype and returns control to CP/M-86.

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

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

Basic Editing Commands

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

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

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

- If the numeric argument is omitted, ED assumes an argument of 1.

- Use a negative number if the command is to be executed backwards through the memory buffer. (The B command is an exception.)

- If you enter a pound sign, #, in place of a number, ED uses the value 65535 as the argument. A pound sign argument can be preceded by a minus sign, $-$ # to cause the command to execute backwards through the memory buffer.

- ED accepts 0 as a numeric argument only in certain commands. In some cases, 0 causes the command to be executed approximately half the possible number of times, while in other cases it prevents the movement of the CP.

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

Basic Editing Commands

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

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

Moving the Character Pointer

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

The B (Beginning/Bottom) Command

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

B, −B

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

The C (Character) Command

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

nC, −nC

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

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

The L (Line) Command

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

nL, −nL

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

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

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

The n (Number) Command

The n command moves the CP and displays the destination line. The forms of the n command are:

n, −n

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

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

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

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

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

Displaying Memory Buffer Contents

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

The T (Type) Command

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

nT, −nT

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

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

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

"I find ec^stasy in living −

Whenever ED is displaying text with the T command, you can enter a Ctrl-S to stop the display, then press any key when you're ready to continue scrolling. Enter a Ctrl-C to abort long type-outs.

Deleting Characters

The D (Delete) Command

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

nD, −nD

where n is the number of characters to be deleted. If no number is specified, ED deletes the character to the right of the CP. A positive number deletes multiple characters to the right of the CP, towards the bottom of the file. A negative number deletes characters to the left of the CP, towards the top of the file. If the character pointer is positioned in the memory buffer as shown below,

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

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

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

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

The K (Kill) Command

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

nK, −nK

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

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

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

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

If the CP is in the middle of a line, a K command kills only the characters from the CP to the end of the line and concatenates the characters before the CP with the next line. A −K command deletes all the characters between the beginning of the previous line and the CP. A 0K command deletes the characters on the line up to the CP.

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

The I (Insert) Command

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

```
I
Istring^Z
```

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

```
A>ED B:QUOTE.TEX

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

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

In insert or command mode, you can use CP/M-86 line editing control characters to edit your input. Note, however, that you cannot use Ctrl-E in insert mode. The table below lists these control characters.

CP/M-86 Line Editing Controls

Command	Result
Ctrl-E	Return carriage for long lines without transmitting command line to the buffer.
Ctrl-H	Delete the last character typed on the current line.
Ctrl-U	Delete the entire line currently being typed.
Ctrl-X	Delete the entire line currently being typed. Same as Ctrl-U.
Backspace	Remove the last character.

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

The Istring^Z (Insert String) Command

The second form of the I command does not enter insert mode. It inserts the character string into the memory buffer and returns immediately to the ED prompt. You can use CP/M-86's line editing control characters to edit the command string.

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

iIn 1870, ^Z

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

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

Replacing Characters

The S (Substitute) Command

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

nSsearch string^Znew string

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

sEmily Dickinson^ZThe poet

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

The poet^ said,<cr><lf>

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

Combining ED Commands

It saves keystrokes and editing time to combine the editing and display commands. You can type any number of ED commands on the same line. ED executes the command string only after you press the Enter key. Use CP/M-86's line editing controls to manipulate ED command strings.

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

- The combined-command line must not exceed CP/M-86's 128-character maximum.
- If the combined-command line contains a character string, the line must not exceed 100 characters.
- Commands to terminate an editing session must not appear in a combined-command line.
- Commands, such as the I, S, J, X and R commands, that require character strings or filespecs must be either the last command on a line or must be terminated with a Ctrl-Z or Esc character, even if no character string or filespec is given.

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

Moving the Character Pointer

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

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

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

Displaying Text

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

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

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

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

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

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

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

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

Combine the edit command K with other ED commands to delete entire lines and verify the correction quickly, as shown below.

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

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

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

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

Moving the CP and Displaying Text

The P (Page) Command

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

nP, −nP

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

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

The n: (Line Number) Command

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

n:

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

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

The :n (Through Line Number) Command

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

```
:ncommand
```

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

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

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

Finding and Replacing Character Strings

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

The F command performs the simplest find function. Its form is:

nFstring

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

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

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

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

It makes a difference whether you enter the F command in upper- or lower-case. If you enter F, ED internally translates the argument string to upper-case. If you specify f, ED looks for an exact match. For example, FCp/m-86 searches for CP/M-86 but fCp/m-86 searches for Cp/m-86, and cannot find CP/M-86 or cp/m-86.

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

```
BREAK "#" AT
```

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

The N Command

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

nNstring

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

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

BREAK "#" AT

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

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

The J (Juxtapose) Command

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

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

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

If you combine this command with other commands, you must terminate the delete-to string with a Ctrl-Z or Esc. (This is shown in the following example.) If an upper-case J command letter is specified, ED looks for upper-case search and delete-to strings and inserts an upper-case insert string.

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

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

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

Note: If long strings make your command longer than your screen line length, enter a Ctrl-E to cause a physical carriage return at the screen. A Ctrl-E returns the cursor to the left edge of the screen, but does not send the command line to ED. Remember that no ED command line containing strings can exceed 100 characters. When you finish your command, press the carriage-return key to send the command to ED.

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

nMcommand string

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

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

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

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

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

To stop a macro command, strike a Ctrl-C at the keyboard.

The Z (Sleep) Command

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

nZ

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

Generally, the Z command has no real effect unless you use it with a macro command. The example below shows you how you can use the Z command to cause a ten-second pause each time ED finds the word "text" in a file.

```
1: zmfliving^Z0tt10z
```

Moving Text Blocks

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

The X (Transfer) Command

The X command takes the forms:

nX
nX filespec^Z

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

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

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

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

The R (Read) Command

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

```
R
Rfilepsec
```

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

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

Saving or Abandoning Changes: ED Exit

You can save or abandon editing changes with the following three commands: H, O, and Q.

The H (Head of File) Command

An H command saves the contents of the memory buffer without ending the ED session, but it returns to the "head" of the file. It saves the current changes and lets you reedit the file without exiting ED. The form of the H command is:

```
H
```

followed by a carriage return.

To execute an H command, ED first finalizes the new file, transferring all lines remaining in the buffer and the source file to the new file. Then ED closes the new file, erases any .BAK file that has the same file specification as the original source file, and renames the original source file filename.BAK. ED then renames the new file, which has had the filetype .$$$, with the original file specification. Finally, ED opens the newly renamed file as the new source file for a new edit, and opens a new .$$$ file. When ED returns the * prompt, the CP is at the beginning of an empty memory buffer.

If you want to send the edited material to a file other than the original file, use a command of the following form:

```
A>ED filespec different-filespec
```

If you then restart the edit with the H command, ED renames the file different-filename.$$$ to different-filename.BAK and creates a new file of different-filespec when you finish editing.

The O (Original) Command

An O command abandons changes made since the beginning of the edit and allows you to return to the original source file and begin reediting without ending the ED session. The form of the O command is:

O

followed by a carriage return. When you enter an O command, ED confirms that you want to abandon your changes by asking:

O (Y/N)?

You must respond with either a Y or an N; if you press any other key, ED repeats the question. When you enter Y, ED erases the temporary file and the contents of the memory buffer. When the * prompt returns, the character pointer is pointing to the beginning of an empty memory buffer, just as it is when you start ED.

The Q (Quit) Command

A Q command abandons changes made since the beginning of the ED session and exits ED. The form of the Q command is:

Q

followed by a carriage return.

When you enter a Q command, ED verifies that you want to abandon the changes by asking:

Q (Y/N)?

You must respond with either a Y or an N; if you press any other key, ED repeats the question. When you enter Y, ED erases the temporary file, closes the source file, and returns control to CP/M-86.

Note: You can enter a Ctrl-Break or a Ctrl-C to immediately return control to CP/M-86. This does not give ED a chance to close the source or new files, but it prevents ED from deleting any temporary files.

ED Error Messages

ED returns one of two types of error messages: an ED error message if ED cannot execute an edit command, or a CP/M-86 error message if ED cannot read or write to the specified file.

The form of an ED error message is:

BREAK "x" AT c

where x is one of the symbols defined in the following table and c is the command letter where the error occurred.

ED Error Symbols

Symbol	Meaning
#	Search failure. ED cannot find the string specified in an F, S, or N command.
?c	Unrecognized command letter c. ED does not recognize the indicated command letter; or an E, H, Q, or O command is not alone on its command line.
0	No .LIB file. ED did not find the .LIB file specified in an R command.
>	Buffer full. ED cannot put any more characters in the memory buffer, or string specified in an F, N, or S command is too long.
E	Command aborted. A keystroke at the keyboard aborted command execution.
F	File error. Followed by either DISK FULL or DIRECTORY FULL.

The following examples show how to recover from common editing error conditions. For example:

BREAK ">" AT A

means that ED filled the memory buffer before completing the execution of an A command. When this occurs, the character pointer is at the end of the buffer and no editing is possible. Use the 0W command to write out half the buffer or use an O or H command and reedit the file.

BREAK "#" AT F

means that ED reached the end of the memory buffer without matching the string in an F command. At this point, the character pointer is at the end of the buffer. Move the CP with a B or n: line number command to resume editing.

BREAK "F" AT F DISK FULL

Use the 0X command to erase an unnecessary file on the disk or a B#Xd:buffer.sav command to write the contents of the memory buffer onto another disk.

BREAK "F" AT n DIRECTORY FULL

Use the same commands described in the previous message to recover from this file error.

The following table defines the disk file error messages ED returns when it cannot read or write a file.

ED Disk File Error Messages

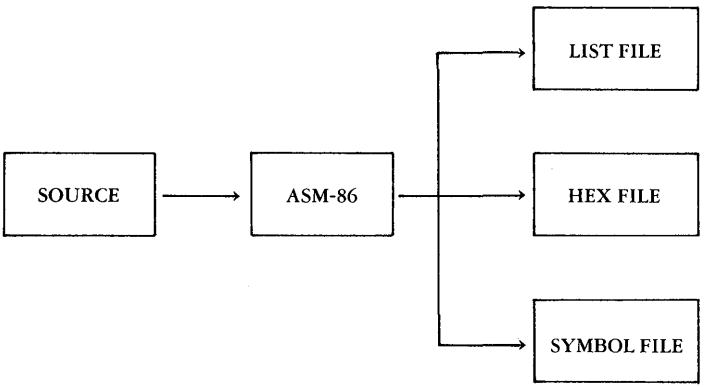
Message	Meaning
BDOS ERR ON d: R/O	Disk d: has read-only attribute. This occurs if a different disk has been inserted in the drive since the last cold or warm boot.
**** FILE IS READ ONLY ****	The file specified in the command to invoke ED has the R/O attribute. ED can read the file so that the user can examine it, but ED cannot change a Read-Only file.

CHAPTER 6. INTRODUCTION TO ASM-86

Contents

Assembler Operation

ASM-86 processes an 8086 or 8088 assembly language source file in three passes and produces three output files, including an 8086/8088 machine language file in hexadecimal format. This object file can be in either Intel or Digital Research hex format. These formats are described in Appendix C. ASM-86 is designed to run under CP/M-86 on an Intel 8086 or 8088 based system. ASM-86 typically produces three output files from one input file as shown in the figure below.



filename.A86 – contains source
filename.LST – contains listing
filename.H86 – contains assembled program in hexadecimal format
filename.SYM – contains all user-defined symbols

ASM-86 Source and Object Files

The figure above also lists ASM-86 filetypes. ASM-86 accepts a source file with any three-letter filetype, but if the type is omitted from the invoking command, it looks for the specified filename with the filetype .A86 in the directory. If the file has a type other than .A86 or has no type at all, ASM-86 returns an error message.

The other filetypes listed in the figure identify ASM-86 output files. The .LST file contains the assembly language listing with any error messages. The .H86 file contains the machine language program in either Digital Research or Intel hexadecimal format. The .SYM file lists any user-defined symbols.

Start ASM-86 by entering a command of the following form:

ASM86 source filespec { $ optional parameters }

Specify the source file in the following form:

{d:}filename{.typ}

where

d:	is a valid drive letter specifying the source file's location. Not needed if source is on current drive.
filename	is a valid CP/M-86 filename of 1 to 8 characters.
filetype	is a valid filetype of 1 to 3 characters, usually .A86.

Some examples of valid ASM-86 commands are:

A>ASM86 B:BIOS88

A>ASM86 BIOS88.A86 $FI AA HB PB SB

A>ASM86 D:TEST

Note that if you assemble an empty source file, ASM-86 produces three empty output files.

Once invoked, ASM-86 responds with the message:

CP/M-86 8086 ASSEMBLER VER x.x

where x.x is the ASM-86 version number. ASM-86 then attempts to open the source file. If the file does not exist on the designated drive, or does not have the correct filetype as described above, the assembler displays the message:

NO FILE

If an invalid parameter is given in the optional parameter list, ASM-86 displays the message:

PARAMETER ERROR

After opening the source, the assembler creates the output files. Usually these are placed on the default diskette drive, but they can be redirected by optional parameters, or by a drive specifier in the source file specification. In the latter case, ASM-86 directs the output files to the drive specified in the source file specification.

During assembly, ASM-86 terminates if an error condition such as diskette full or symbol table overflow is detected. When ASM-86 detects an error in the source file, it places an error message line in the listing file in front of the line containing the error. Each error message has a number and gives a brief explanation of the error. See Appendix A for a list of ASM-86 error messages. Also in the list file, the value of the absolute address generated by relative instructions, such as calls, jumps and loops, appears in a field to the left of the source line. When the assembly is complete, ASM-86 displays the message:

END OF ASSEMBLY. NUMBER OF ERRORS: n. USE FACTOR: pp%

The use factor indicates how much of the available symbol table space was actually used during the assembly; pp is a decimal percentage ranging from 0 to 99.

Optional Run-time Parameters

The dollar-sign character, $, flags an optional string of run-time parameters. A parameter is a single letter followed by a single letter device name specification. The parameters are shown in the table below.

Run-time Parameter Summary

Parameter	To Specify	Valid Arguments
A	source file device	A, B, C, D
H	hex output file device	A, B, C, D, X, Y, Z
P	list file device	A, B, C, D, X, Y, Z
S	symbol file device	A, B, C, D, X, Y, Z
F	format of hex output file	I, D

All parameters are optional, and can be entered in the command line in any order. Enter the dollar sign only once at the beginning of the parameter string. Spaces may separate parameters, but are not required. No space is permitted, however, between a parameter and its device name.

A device name must follow parameters A, H, P and S. The devices are labeled:

A, B, C, D or X, Y, Z

Device names A through D, respectively, specify disk drives A through D. X specifies the screen, Y specifies the printer, and Z suppresses output.

If output is directed to the screen, it can be temporarily stopped at any time by typing a Ctrl-S. Restart the output by typing a second Ctrl-S or any other character.

The F parameter requires either an I or a D argument. When I is specified, ASM-86 produces an object file in Intel hex format. A D argument requests Digital Research hex format. Appendix C discusses these formats in detail. If the F parameter is not entered in the command line, ASM-86 produces Digital Research hex format.

Run-time Parameter Examples

Command Line	Result
ASM86 IO	Assemble file IO.A86, produce IO.H86, IO.LST and IO.SYM, all on the default drive.
ASM86 IO.ASM $ AD SZ	Assemble file IO.ASM on drive D, produce IO.LST and IO.H86, no symbol file.
ASM86 IO $ PY SX	Assemble file IO.A86, produce IO.H86, route listing directly to printer, output symbols on screen.
ASM86 IO $ FD	Produce Digital Research hex format.
ASM86 IO $ FI	Produce Intel hex format.

nding ASM-86

You can end ASM-86 execution at any time by hitting any key on the keyboard. When you press a key, ASM-86 responds with the question:

USER BREAK. OK(Y/N)?

A Y response ends the assembly and returns to the operating system. An N response continues the assembly.

CHAPTER 7. ELEMENTS OF ASM-86 ASSEMBLY LANGUAGE

Contents

ASM-86 Character Set

ASM-86 recognizes a subset of the ASCII character set. The valid characters are the alphanumerics, special characters, and non-printing characters shown below:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9

+  -  *  /  =  (  )  [  ]  ;  '
!  ,  _  :  @  $
```

space, tab, carriage-return, and line-feed

Lower-case letters are treated as upper-case except within strings. Only alphanumerics, special characters, and spaces may appear within a string.

Tokens and Separators

A token is the smallest meaningful unit of an ASM-86 source program, much as a word is the smallest meaningful unit of an English composition. Adjacent tokens are commonly separated by a blank character or space. Any sequence of spaces may appear wherever a single space is allowed. ASM-86 recognizes horizontal tabs as separators and interprets them as spaces. Tabs are expanded to spaces in the list file. The tab stops are at each eighth column.

Delimiters

Delimiters mark the end of a token and add special meaning to the instruction, as opposed to separators, which merely mark the end of a token. When a delimiter is present, separators need not be used. However, separators after delimiters can make your program easier to read.

The table below describes ASM-86 separators and delimiters. Some delimiters are also operators and are explained in greater detail in the section "Operators."

Separators and Delimiters

Character	Name	Use
20H	space	separator
09H	tab	legal in source files, expanded in list files
CR	carriage return	terminates source lines
LF	line feed	legal after CR; if within source lines, it is interpreted as a space
;	semicolon	starts comment field
:	colon	identifies a label, used in segment override specification
.	period	forms variables from numbers
$	dollar sign	notation for "present value of location pointer"
+	plus	arithmetic operator for addition
−	minus	arithmetic operator for subtraction
*	asterisk	arithmetic operator for multiplication
/	slash	arithmetic operator for division
@	"at" sign	legal in identifiers
_	underscore	legal in identifiers
!	exclamation point	logically terminates a statement, thus allowing multiple statements on a single source line
'	apostrophe	delimits string constants

A constant is a value known at assembly time that does not change while the assembled program is executed. A constant may be either an integer or a character string.

Numeric Constants

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

Radix Indicators for Constants

Indicator	Constant Type	Base
B	binary	2
O	octal	8
Q	octal	8
D	decimal	10
H	hexadecimal	16

ASM-86 assumes that any numeric constant not terminated with a radix indicator is a decimal constant. Radix indicators may be upper- or lower-case.

A constant is thus a sequence of digits followed by an optional radix indicator, where the digits are in the range for the radix. Binary constants must be composed of 0's and 1's. Octal digits range from 0 to 7; decimal digits range from 0 to 9. Hexadecimal constants contain decimal digits as well as the hexadecimal digits A (10D), B (11D), C (12D), D (13D), E (14D), and F (15D). Note that the leading character of a hexadecimal constant must be a decimal digit so that ASM-86 cannot confuse a hex constant with an identifier. The following are valid numeric constants:

1234	1234D	1100B	1111000011110000B
1234H	0FFEH	3377O	13772Q
33770	0FE3H	1234d	0FFFFH

Character Strings

ASM-86 treats an ASCII character string delimited by apostrophes as a string constant. All instructions accept only one- or two-character constants as valid arguments. Instructions treat a one-character string as an 8-bit number. A two-character string is treated as a 16-bit number with the value of the second character in the low-order byte, and the value of the first character in the high-order byte.

The numeric value of a character is its ASCII code. Both upper- and lower-case letters can be used because ASM-86 does not translate case within character strings. Note that only alphanumerics, special characters, and spaces are allowed within strings.

A DB assembler directive is the only ASM-86 statement that may contain strings longer than two characters. The string may not exceed 255 bytes. Include any apostrophe to be printed within the string by entering it twice. ASM-86 interprets the two keystrokes ('') as a single apostrophe. The examples below show valid strings and how they appear after processing.

```
          'a' —> a
      'Ab''Cd' —> Ab'Cd
          '''' —> '
'ONLY UPPER CASE' —> ONLY UPPER CASE
'only lower case' —> only lower case
```

Identifiers

Identifiers are character sequences which have a special, symbolic meaning to the assembler. All identifiers in ASM-86 must obey the following rules:

1. The first character must be alphabetic (A, . . . Z, a, . . . z).

2. Any subsequent characters can be either alphabetical or a numeral (0, 1, ... 9). ASM-86 ignores the special characters @ and _, but they are still legal. For example, a__b becomes ab.

3. Identifiers may be of any length up to the limit of the physical line.

Identifiers are of two types. The first are keywords, which have predefined meanings to the assembler. The second are symbols, which are defined by the user. The following are all valid identifiers:

NOLIST
WORD
AH
Third__street
How__are__you__today
variable@number@1234567890

Keywords

A keyword is an identifier that has a predefined meaning to the assembler. Keywords are reserved; the user cannot define an identifier identical to a keyword. The following table gives a complete list of reserved words.

Reserved Words

Predefined Numbers

BYTE	WORD	DWORD

Operators

EQ	GE	GT	LE	LT
NE	OR	AND	MOD	NOT
PTR	SEG	SHL	SHR	XOR
LAST	TYPE	LENGTH	OFFSET	

Assembler Directives

DB	DD	DW	IF	RB
RS	RW	END	EQU	ORG
CSEG	DSEG	ENDM	ESEG	LIST
SSEG	EJECT	ENDIF	TITLE	IFLIST
NOLIST	INCLUDE	SIMFORM	NOIFLIST	PAGESIZE
CODEMACRO	PAGEWIDTH			

Code-macro directives

DB	DD	DW	DBIT	RELB
RELW	MODRM	SEGFIX	NOSEGFIX	

8086 Registers

AH	AL	AX	BH	BL
BP	BX	CH	CL	CS
CX	DH	DI	DL	DS
DX	ES	SI	SP	SS

ASM-86 recognizes five types of keywords: instructions, directives, operators, registers and predefined numbers. 8086 instruction mnemonic keywords and the actions they initiate are defined in Chapter 9. Directives are discussed in Chapter 8.

Three keywords are predefined numbers: BYTE, WORD, and DWORD. The values of these numbers are 1, 2 and 4, respectively. In addition, a Type attribute is associated with each of these numbers. The keyword's Type attribute is equal to the keyword's numeric value. See the section "Symbols and Their Attributes" for a complete discussion of Type attributes. The following table gives a complete list of register keywords.

Register Keywords

Register Symbol	Size	Numeric Value	Meaning
AH	1 byte	100 B	Accumulator-High-Byte
BH	1 byte	111 B	Base-Register-High-Byte
CH	1 byte	101 B	Count-Register-High-Byte
DH	1 byte	110 B	Data-Register-High-Byte
AL	1 byte	000 B	Accumulator-Low-Byte
BL	1 byte	011 B	Base-Register-Low-Byte
CL	1 byte	001 B	Count-Register-Low-Byte
DL	1 byte	010 B	Data-Register-Low-Byte
AX	2 bytes	000 B	Accumulator (full word)
BX	2 bytes	011 B	Base-Register (full word)
CX	2 bytes	001 B	Count-Register (full word)
DX	2 bytes	010 B	Data-Register (full word)
BP	2 bytes	101 B	Base Pointer
SP	2 bytes	100 B	Stack Pointer
SI	2 bytes	110 B	Source Index
DI	2 bytes	111 B	Destination Index
CS	2 bytes	01 B	Code-Segment-Register
DS	2 bytes	11 B	Data-Segment-Register
SS	2 bytes	10 B	Stack-Segment-Register
ES	2 bytes	00 B	Extra-Segment-Register

Symbols and Their Attributes

A symbol is a user-defined identifier that has attributes which specify what kind of information the symbol represents. Symbols fall into three categories:

- variables
- labels
- numbers

Variables identify data stored at a particular location in memory. All variables have the following three attributes:

- Segment–tells which segment was being assembled when the variable was defined.

- Offset–tells how many bytes there are between the beginning of the segment and the location of this variable.

- Type–tells how many bytes of data are manipulated when this variable is referenced.

A Segment may be a code-segment, a data-segment, a stack-segment or an extra-segment depending on its contents and the register that contains its starting address (see "Segment Start Directives" in Chapter 8). A segment may start at any address divisible by 16. ASM-86 uses this boundary value as the Segment portion of the variable's definition.

The Offset of a variable may be any number between 0 and 0FFFFH or 65535D. A variable must have one of the following Type attributes:

- BYTE
- WORD
- DWORD

BYTE specifies a one-byte variable, WORD a two-byte variable and DWORD a four-byte variable. The DB, DW, and DD directives, respectively, define variables as these three types. For example, a variable is defined when it appears as the name for a storage directive:

```
VARIABLE DB 0
```

A variable may also be defined as the name for an EQU directive referencing another variable, as shown below:

```
VARIABLE  EQU  ANOTHER_VARIABLE
```

Labels identify locations in memory that contain instruction statements. They are referenced with jumps or calls. All labels have two attributes:

- Segment
- Offset

Label segment and offset attributes are essentially the same as variable segment and offset attributes. Generally, a label is defined when it precedes an instruction. A colon, :, separates the label from instruction; for example:

```
LABEL:  ADD AX,BX
```

A label may also appear as the name for an EQU directive referencing another label; for example:

```
LABEL  EQU  ANOTHER_LABEL
```

Numbers may also be defined as symbols. A number symbol is treated as if you had explicitly coded the number it represents. For example:

```
Number_five  EQU  5
MOV  AL,Number_five
```

is equivalent to:

```
MOV  AL,5
```

The following section describes operators and their effects on numbers and number symbols.

Operators

ASM-86 operators fall into the following categories: arithmetic, logical, and relational operators, segment override, variable manipulators and creators. The following table defines ASM-86 operators. In this table, a and b represent two elements of the expression. The validity column defines the type of operands the operator can manipulate, using the "or" bar character, |, to separate alternatives.

ASM-86 Operators

Syntax	Result	Validity
	Logical Operators	
a XOR b	bit-by-bit logical EXCLUSIVE OR of a and b.	a, b = number
a OR b	bit-by-bit logical OR of a and b.	a, b = number
a AND b	bit-by-bit logical AND of a and b.	a, b = number
NOT a	logical inverse of a: all 0's become 1's, 1's become 0's.	a = 16-bit number
a EQ b	returns 0FFFFH if a = b, otherwise 0.	a, b = unsigned number
a LT b	returns 0FFFFH if a < b, otherwise 0.	a, b = unsigned number
a LE b	returns 0FFFFH if a <= b, otherwise 0.	a, b = unsigned number
a GT b	returns 0FFFFH if a > b, otherwise 0.	a, b = unsigned number
a GE b	returns 0FFFFH if a >= b, otherwise 0.	a, b = unsigned number
a NE b	returns 0FFFFH if a <> b, otherwise 0.	a, b = unsigned number

ASM-86 Operators (continued)

Syntax	Result	Validity
	Arithmetic Operators	
a + b	arithmetic sum of a and b.	a = variable, label or number b = number
a − b	arithmetic difference of a and b.	a = variable, label or number b = number
a * b	does unsigned multiplication of a and b.	a, b = number
a / b	does unsigned division of a and b.	a, b = number
a MOD b	returns remainder of a/b.	a, b = number
a SHL b	returns the value which results from shifting a to the left by an amount b.	a, b = number
a SHR b	returns the value which results from shifting a to the right by an amount b.	a, b = number
+ a	gives a.	a = number
− a	gives 0 − a.	a = number
	Segment Override	
<seg reg>: <addr exp>	overrides assembler's choice of segment register.	<seg reg> = CS, DS, SS or ES

ASM-86 Operators (continued)

Syntax	Result	Validity
	Variable Manipulators, Creators	
SEG a	creates a number whose value is the segment value of the variable or label a. The variable or label a must be declared in an absolute segment (i.e. CSEG 1234H); otherwise the SEG operator is undefined.	a = label | variable
OFFSET a	creates a number whose value is the offset value of the variable or label a.	a = label | variable
TYPE a	creates a number. If the variable a is of type BYTE, WORD or DWORD, the value of the number will be 1, 2 or 4, respectively.	a = label | variable
LENGTH a	creates a number whose value is the LENGTH attribute of the variable a. The length attribute is the number of bytes associated with the variable.	a = label | variable
LAST a	if LENGTH a > 0, then LAST a = LENGTH a − 1; if LENGTH a = 0, then LAST a = 0.	a = label | variable
a PTR b	creates virtual variable or label with type of a and attributes of b	a = BYTE | WORD, | DWORD b = <addr exp>

Syntax	Result	Validity
	Variable Manipulators, Creators	
.a	creates variable with an offset attribute of a. Segment attribute is current segment.	a = number
$	creates label with offset equal to current value of location counter; segment attribute is current segment.	no argument

Operator Examples

Logical operators accept only numbers as operands. They perform the boolean logic operations AND, OR, XOR, and NOT. For example:

```
  00FC      MASK      EQU   0FCH
  0080      SIGNBIT   EQU   80H
0000 B180             MOV   CL,MASK AND SIGNBIT
0002 B003             MOV   AL,NOT MASK
```

Relational operators treat all operands as unsigned numbers. The relational operators are EQ (equal), LT (less than), LE (less than or equal), GT (greater than), GE (greater than or equal), and NE (not equal). Each operator compares two operands and returns all ones (0FFFFH) if the specified relation is true and all zeros if it is not. For example:

```
000A          LIMIT1    EQU   10
0019          LIMIT2    EQU   25
                        .
                        .
                        .
0004 B8FFFF             MOV   AX,LIMIT1 LT LIMIT2
0007 B80000             MOV   AX,LIMIT1 GT LIMIT2
```

Addition and subtraction operators compute the arithmetic sum and difference of two operands. The first operand may be a variable, label, or number, but the second operand must be a number. When a number is added to a variable or label, the result is a variable or label whose offset is the numeric value of the second operand plus the offset of the first operand. Subtraction from a variable or label returns a variable or label whose offset is that of first operand decremented by the number specified in the second operand. For example:

```
0002                COUNT  EQU  2
0005                DISP1  EQU  5
000A FF             FLAG   DB   0FFH
                           .
                           .
                           .
000B 2EA00B00              MOV  AL,FLAG+1
000F 2E8A0E0F00            MOV  CL,FLAG+DISP1
00014 B303                 MOV  BL,DISP1-COUNT
```

The multiplication and division operators *, /, MOD, SHL, and SHR accept only numbers as operands. * and / treat all operators as unsigned numbers. For example:

```
0016 BE5500    MOV          SI,256/3
0019 B310      MOV          BL,64/4
0050           BUFFERSIZE   EQU     80
001B B8A000    MOV          AX,BUFFERSIZE * 2
```

Unary operators accept both signed and unsigned operators as shown below:

```
001E B123      MOV          CL,+35
0020 B007      MOV          AL,2-5
0022 B2F4      MOV          DL,-12
```

When manipulating variables, the assembler decides which segment register to use. You may override the assembler's choice by specifying a different register with the segment override operator. The syntax for the override operator is segment-register : address-expression where the segment-register is CS, DS, SS, or ES. For example:

```
0024 368B472D  MOV          AX,SS:WORDBUFFER[BX]
0028 268B0E5B00 MOV         CX,ES:ARRAY
```

A variable manipulator creates a number equal to one attribute of its variable operand. SEG extracts the variable's segment value, OFFSET its offset value, TYPE its type value (1, 2, or 4), and LENGTH the number of bytes associated with the variable. LAST compares the variable's LENGTH with 0 and if greater, then decrements LENGTH by one. If LENGTH equals 0, LAST leaves it unchanged. Variable manipulators accept only variables as operators. For example:

```
 1234                                    DSEG 1234H
002D 000000000000      WORDBUFFER       DW   0,0,0
0033 0102030405        BUFFER           DB   1,2,3,4,5
                                        .
                                        .
                                        .
0038 B80500            MOV     AX,LENGTH BUFFER
003B B80400            MOV     AX,LAST BUFFER
003E B80100            MOV     AX,TYPE BUFFER
0041 B80200            MOV     AX,TYPE WORDBUFFER
0044 B83412            MOV     AX,SEG BUFFER
```

The PTR operator creates a virtual variable or label, one valid only during the execution of the instruction. It makes no changes to either of its operands. The temporary symbol has the same Type attribute as the left operator, and all other attributes of the right operator as shown below.

```
0044 C60705            MOV     BYTE PTR [BX], 5
0047 8A07              MOV     AL,BYTE PTR [BX]
0049 FF04              INC     WORD PTR [SI]
```

The Period operator, ., creates a variable in the current data segment. The new variable has a segment attribute equal to the current data segment and an offset attribute equal to its operand. Its operand must be a number. For example:

```
004B A10000            MOV     AX, .0
004E 268B1E0040        MOV     BX, ES: .4000H
```

The dollar-sign operator, $, creates a label with an offset attribute equal to the current value of the location counter. The label's segment value is the same as the current segment. This operator takes no operand. For example:

```
0053 E9FDFF            JMP     $
0056 EBFE              JMPS    $
0058 E9FD2F            JMP     $+3000H
```

Operator Precedence

Expressions combine variables, labels or numbers with operators. ASM-86 allows several kinds of expressions which are discussed in the section "Expressions." However, this section defines the order in which operations are executed, should more than one operator appear in an expression.

In general, ASM-86 evaluates expressions left to right, but operators with higher precedence are evaluated before operators with lower precedence. When two operators have equal precedence, the left-most is evaluated first. The table below presents ASM-86 operators in order of increasing precedence.

Parentheses can override normal rules of precedence. The part of an expression enclosed in parentheses is evaluated first. If parentheses are nested, the innermost expressions are evaluated first. Only five levels of nested parentheses are legal. For example:

```
15/3 + 18/9 = 5 + 2 = 7
15/(3 + 18/9) = 15/(3 + 2) = 15/5 = 3
```

Precedence of Operations in ASM-86

Order	Operator Type	Operators
1	Logical	XOR, OR
2	Logical	AND
3	Logical	NOT
4	Relational	EQ, LT, LE, GT, GE, NE
5	Addition/subtraction	+, −
6	Multiplication/division	*, /, MOD, SHL, SHR
7	Unary	+, −
8	Segment override	<segment override>:

Precedence of Operations in ASM-86 (continued)

Order	Operator Type	Operators
9	Variable manipulators, creators	SEG, OFFSET, PTR, TYPE, LENGTH, LAST
10	Parentheses/brackets	( ), [ ]
11	Period and Dollar	., $

Expressions

ASM-86 allows address, numeric, and bracketed expressions. An address expression evaluates to a memory address and has three components:

- A segment value
- An offset value
- A type

Both variables and labels are address expressions. An address expression is not a number, but its components are. Numbers may be combined with operators such as PTR to make an address expression.

A numeric expression evaluates to a number. It does not contain any variables or labels, only numbers and operands.

Bracketed expressions specify base- and index-addressing modes. The base registers are BX and BP, and the index registers are DI and SI. A bracketed expression may consist of a base register, an index register, or a base register and an index register. Use the + operator between a base register and an index register to specify both base- and index-register addressing. For example:

```
MOV variable[bx],0
MOV AX,[BX+DI]
MOV AX,[SI]
```

Statements

Just as "tokens" in this assembly language correspond to words in English, so are statements analogous to sentences. A statement tells ASM-86 what action to perform. Statements are of two types: instructions and directives. Instructions are translated by the assembler into 8086 machine language instructions. Directives are not translated into machine code but instead direct the assembler to perform certain clerical functions.

Terminate each assembly language statement with a carriage return (CR) and line-feed (LF), or with an exclamation point, !, which ASM-86 treats as an end-of-line. Multiple assembly language statements can be written on the same physical line if separated by exclamation points.

The ASM-86 instruction set is defined in Chapter 9. The syntax for an instruction statement is:

[label:] [prefix] mnemonic [operand(s)] [;comment]

where the fields are defined as:

label:

> A symbol followed by ":" defines a label at the current value of the location counter in the current segment. This field is optional.

prefix

> Certain machine instructions such as LOCK and REP may prefix other instructions. This field is optional.

mnemonic

> A symbol defined as a machine instruction, either by the assembler or by an EQU directive. This field is optional unless preceded by a prefix instruction. If it is omitted, no operands may be present, although the other fields may appear. ASM-86 mnemonics are defined in Chapter 9.

operand(s)

An instruction mnemonic may require other symbols to represent operands to the instruction. Instructions may have zero, one or two operands.

comment

Any semicolon (;) appearing outside a character string begins a comment, which is ended by a carriage return. Comments improve the readability of programs. This field is optional.

ASM-86 directives are described in Chapter 8. The syntax for a directive statement is:

[name] directive operand(s) [;comment]

where the fields are defined as:

name

Unlike the label field of an instruction, the name field of a directive is never terminated with a colon. Directive names are legal for only DB, DW, DD, RS and EQU. For DB, DW, DD and RS the name is optional; for EQU it is required.

directive

One of the directive keywords defined in Chapter 8.

operand(s)

Analogous to the operands of the instruction mnemonics. Some directives, such as DB, DW, and DD, allow any operand while others have special requirements.

comment

Exactly as defined for instruction statements.

CHAPTER 8. ASSEMBLER DIRECTIVES

Contents

Assembler Directives

Directive statements cause ASM-86 to perform housekeeping functions such as assigning portions of code to logical segments, requesting conditional assembly, defining data items, and specifying listing file format. General syntax for directive statements appears in the preceding chapter.

In the sections that follow, the specific syntax for each directive statement is given under the heading and before the explanation. These syntax lines use special symbols to represent possible arguments and other alternatives. Braces, {}, enclose optional arguments. User-supplied arguments are described in lower-case, hyphenated phrases. Do not include these symbols or phrases when coding a directive.

Segment Start Directives

At run-time, every 8086 memory reference must have a 16-bit segment base value and a 16-bit offset value. These are combined to produce the 20-bit effective address needed by the CPU to physically address the location. The 16-bit segment base value or boundary is contained in one of the segment registers CS, DS, SS, or ES. The offset value gives the offset of the memory reference from the segment boundary. A 16-byte physical segment is the smallest relocatable unit of memory.

ASM-86 predefines four logical segments: the Code Segment, Data Segment, Stack Segment, and Extra Segment, which are respectively addressed by the CS, DS, SS, and ES registers. All ASM-86 statements must be assigned to one of the four segments so that they can be referenced by the CPU. A segment directive statement, CSEG, DSEG, SSEG, or ESEG, specifies that the statements following it belong to a specific segment. The statements are then addressed by the corresponding segment register. ASM-86 assigns statements to the specified segment until it encounters another segment directive.

Instruction statements must be assigned to the Code Segment. Directive statements may be assigned to any segment. ASM-86 uses these assignments to change from one segment register to another. For example, when an instruction accesses a memory variable, ASM-86 must know which segment contains the variable so it can generate a segment override prefix byte if necessary.

The CSEG Directive

```
CSEG        numeric-expression
CSEG
CSEG        $
```

This directive tells the assembler that the following statements belong in the Code Segment. All instruction statements must be assigned to the Code Segment. All directive statements are legal within the Code Segment.

Use the first form when the location of the segment is known at assembly time; the code generated is not relocatable. Use the second form when the segment location is not known at assembly time; the code generated is relocatable. Use the third form to continue the Code Segment after it has been interrupted by a DSEG, SSEG, or ESEG directive. The continuing Code Segment starts with the same attributes, such as location and instruction pointer, as the previous Code Segment.

The DSEG Directive

```
DSEG        numeric-expression
DSEG
DSEG        $
```

This directive specifies that the following statements belong to the Data Segment. The Data Segment primarily contains the data allocation directives DB, DW, DD and RS, but all other directive statements are also legal. Instruction statements are illegal in the Data Segment.

Use the first form when the location of the segment is known at assembly time; the code generated is not relocatable. Use the second form when the segment location is not known at assembly time; the code generated is relocatable. Use the third form to continue the Data Segment after it has been interrupted by a CSEG, SSEG, or ESEG directive. The continuing Data Segment starts with the same attributes as the previous Data Segment.

The SSEG Directive

```
SSEG        numeric-expression
SSEG
SSEG        $
```

The SSEG directive indicates the beginning of source lines for the Stack Segment. Use the Stack Segment for all stack operations. All directive statements are legal in the Stack Segment, but instruction statements are illegal.

Use the first form when the location of the segment is known at assembly time; the code generated is not relocatable. Use the second form when the segment location is not known at assembly time; the code generated is relocatable. Use the third form to continue the Stack Segment after it has been interrupted by a CSEG, DSEG, or ESEG directive. The continuing Stack Segment starts with the same attributes as the previous Stack Segment. Refer to the stack segment initialization example in Appendix B.

The ESEG Directive

```
ESEG        numeric-expression
ESEG
ESEG        $
```

This directive initiates the Extra Segment. Instruction statements are not legal in this segment, but all directive statements are.

Use the first form when the location of the segment is known at assembly time; the code generated is not relocatable. Use the second form when the segment location is not known at assembly time; the code generated is relocatable. Use the third form to continue the Extra Segment after it has been interrupted by a DSEG, SSEG, or CSEG directive. The continuing Extra Segment starts with the same attributes as the previous Extra Segment.

The ORG Directive

```
ORG         numeric-expression
```

The ORG directive sets the offset of the location counter in the current segment to the value specified in the numeric expression. Define all elements of the expression before the ORG directive because forward references may be ambiguous.

In most segments, an ORG directive is unnecessary. If no ORG is included before the first instruction or data byte in a segment, assembly begins at location zero relative to the beginning of the segment. A segment can have any number of ORG directives.

The IF and ENDIF Directives

```
IF          numeric-expression
            source-line-1
            source-line-2
                .
                .
                .
            source-line-n
ENDIF
```

The IF and ENDIF directives allow a group of source lines to be included or excluded from the assembly. Use conditional directives to assemble several different versions of a single source program.

When the assembler finds an IF directive, it evaluates the numeric expression following the IF keyword. If the expression evaluates to a non-zero value, then source-line-1 through source-line-n are assembled. If the expression evaluates to zero, the lines are not assembled, but are listed unless a NOIFLIST directive is active. All elements in the numeric expression must be defined before they appear in the IF directive. IF directives may be nested to a maximum depth of five levels.

The INCLUDE Directive

```
INCLUDE     filespec
```

This directive includes another ASM-86 file in the source text. For example:

```
INCLUDE  EQUALS.A86
```

Use INCLUDE when the source program resides in several different files. INCLUDE directives may not be nested; a source file called by an INCLUDE directive may not contain another INCLUDE statement. If file specification does not contain a filetype, the filetype is assumed to be .A86. If the file specification does not include a drive specification, ASM-86 assumes the file resides on the drive containing the source file.

The END Directive

END

An END directive marks the end of a source file. Any subsequent lines are ignored by the assembler. END is optional. If not present, ASM-86 processes the source until it finds an End-Of-File character (1AH).

The EQU Directive

```
symbol   EQU   numeric-expression
symbol   EQU   address-expression
symbol   EQU   register
symbol   EQU   instruction-mnemonic
```

The EQU (equate) directive assigns values and attributes to user-defined symbols. The required symbol name may not be terminated with a colon. The symbol cannot be redefined by a subsequent EQU or another directive. Any elements used in numeric or address expressions must be defined before the EQU directive appears.

The first form assigns a numeric value to the symbol, the second a memory address. The third form assigns a new name to an 8086 register. The fourth form defines a new instruction (sub)set. The following are examples of these four forms:

```
0005          FIVE        EQU   2*2+1
0033          NEXT        EQU   BUFFER
0001          COUNTER     EQU   CX
              MOVVV       EQU   MOV
                                .
                                .
                                .
005D 8BC3                 MOVVV AX,BX
```

{symbol} DB
numeric-expression{,numeric-expression..}
{symbol} DB string-constant{,string-constant...}

The DB directive defines initialized storage areas in byte format. Numeric expressions are evaluated to 8-bit values and sequentially placed in the hex output file. String constants are placed in the output file according to the rules defined in the section "Constants" in Chapter 7. A DB directive is the only ASM-86 statement that accepts a string constant longer than two bytes. There is no translation from lower- to upper-case within strings. Multiple expressions or constants, separated by commas, may be added to the definition, but may not exceed the physical line length.

Use an optional symbol to reference the defined data area throughout the program. The symbol has four attributes: the Segment and Offset attributes determine the symbol's memory reference, the Type attribute specifies single bytes, and Length tells the number of bytes (allocation units) reserved.

The following statements show DB directives with symbols:

```
005F  43502F4D2073    TEXT    DB    'CP/M system',0
      797374656D00
006B  E1              AA      DB    'a' + 80H
006C  0102030405      X       DB    1,2,3,4,5
                                    .
                                    .
                                    .
0071  B90C00                  MOV   CX,LENGTH TEXT
```

The DW Directive

{symbol} DW
numeric-expression{,numeric-expression..}
{symbol} DW string-constant{,string-constant...}

The DW directive initializes two-byte words of storage. String constants longer than two characters are illegal. Otherwise, DW uses the same procedure to initialize storage as DB. The following are examples of DW statements:

```
0074   0000            CNTR     DW   0
0076   63C166C169C1    JMPTAB   DW   SUBR1,SUBR2,SUBR3
007C   010002000300             DW   1,2,3,4,5,6
       040005000600
```

The DD Directive

{symbol} DD
numeric-expression{,numeric-expression..}

The DD directive initializes four bytes of storage. The Offset attribute of the address expression is stored in the two lower bytes, the Segment attribute in the two upper bytes. Otherwise, DD follows the same procedure as DB. For example:

```
1234                              CSEG   1234H
                                  .
                                  .
                                  .
0000   6CC134126FC1 LONG_JMPTAB   DD     ROUT1,ROUT2
       3412
0008   72C1341275C1               DD     ROUT3,ROUT4
       3412
```

The RS Directive

{symbol} RS numeric-expression

The RS directive allocates storage in memory but does not initialize it. The numeric expression gives the number of bytes to be reserved. An RS statement does not give a byte attribute to the optional symbol. For example:

```
0010                    BUF     RS      80
0060                            RS      4000H
4060                            RS      1
```

If an RS statement is the last statement in a segment, you must follow it with a DB statement in order for GENCMD to allocate the memory space.

The RB Directive

{symbol} RB numeric-expression

The RB directive allocates byte storage in memory without any initialization. This directive is identical to the RS directive except that it does give the byte attribute.

The RW Directive

{symbol} RW numeric-expression

The RW directive allocates two-byte word storage in memory but does not initialize it. The numeric expression gives the number of words to be reserved. For example:

```
4061                    BUF     RW      128
4161                            RW      4000H
C161                            RW      1
```

The TITLE Directive

TITLE string-constant

ASM-86 prints the string constant defined by a TITLE directive statement at the top of each printout page in the listing file. The title character string should not exceed 30 characters. For example:

```
TITLE 'CP/M-86 monitor'
```

If the title is too long, the ASM-86 page number overwrites the title.

The PAGESIZE Directive

PAGESIZE numeric-expression

The PAGESIZE directive defines the number of lines to be included on each printout page. The default pagesize is 66.

The PAGEWIDTH Directive

PAGEWIDTH numeric-expression

The PAGEWIDTH directive defines the number of columns printed across the page when the listing file is output. The default pagewidth is 120 unless the listing is routed directly to the terminal; then the default pagewidth is 79.

The EJECT Directive

EJECT

The EJECT directive performs a page eject during printout. The EJECT directive itself is printed on the first line of the next page.

The SIMFORM Directive

SIMFORM

The SIMFORM directive replaces a form-feed (FF) character in the print file with the correct number of line-feeds (LF). Use this directive when printing out on a printer unable to interpret the form-feed character.

The NOLIST and LIST Directives

NOLIST
LIST

The NOLIST directive blocks the printout of the following lines. Restart the listing with a LIST directive.

The IFLIST and NOIFLIST Directives

IFLIST
NOIFLIST

The NOIFLIST directive suppresses the printout of the contents of IF-ENDIF blocks that are not assembled. The IFLIST directive resumes printout of IF-ENDIF blocks.

CHAPTER 9. THE ASM-86 INSTRUCTION SET

Contents

ASM-86 Instruction Set Summary

The ASM-86 instruction set includes all 8086 machine instructions. The general syntax for instruction statements is given at the end of Chapter 7. The following table lists all ASM-86 instructions alphabetically.

ASM-86 Instruction Summary

Mnemonic	Description
AAA	ASCII Adjust for Addition
AAD	ASCII Adjust for Division
AAM	ASCII Adjust for Multiplication
AAS	ASCII Adjust for Subtraction
ADC	Add with Carry
ADD	Add
AND	And
CALL	Call (intra segment)
CALLF	Call (inter segment)
CBW	Convert Byte to Word
CLC	Clear Carry
CLD	Clear Direction
CLI	Clear Interrupt
CMC	Complement Carry
CMP	Compare
CMPS	Compare Byte or Word (of string)
CMPSB	Compare Byte (of string)
CMPSW	Compare Word (of string)
CWD	Convert Word to Double Word
DAA	Decimal Adjust for Addition
DAS	Decimal Adjust for Subtraction
DEC	Decrement
DIV	Divide
ESC	Escape
HLT	Halt
IDIV	Integer Divide
IMUL	Integer Multiply
IN	Input Byte or Word
INC	Increment
INT	Interrupt
INTO	Interrupt on Overflow
IRET	Interrupt Return
JA	Jump on Above
JAE	Jump on Above or Equal

Mnemonic	Description
JB	Jump on Below
JBE	Jump on Below or Equal
JC	Jump on Carry
JCXZ	Jump on CX Zero
JE	Jump on Equal
JG	Jump on Greater
JGE	Jump on Greater or Equal
JL	Jump on Less
JLE	Jump on Less or Equal
JMP	Jump (intra segment)
JMPF	Jump (inter segment)
JMPS	Jump (8-bit displacement)
JNA	Jump on Not Above
JNAE	Jump on Not Above or Equal
JNB	Jump on Not Below
JNBE	Jump on Not Below or Equal
JNC	Jump on Not Carry
JNE	Jump on Not Equal
JNG	Jump on Not Greater
JNGE	Jump on Not Greater or Equal
JNL	Jump on Not Less
JNLE	Jump on Not Less or Equal
JNO	Jump on Not Overflow
JNP	Jump on Not Parity
JNS	Jump on Not Sign
JNZ	Jump on Not Zero
JO	Jump on Overflow
JP	Jump on Parity
JPE	Jump on Parity Even
JPO	Jump on Parity Odd
JS	Jump on Sign
JZ	Jump on Zero
LAHF	Load AH with Flags
LDS	Load Pointer into DS
LEA	Load Effective Address
LES	Load Pointer into ES
LOCK	Lock Bus
LODS	Load Byte or Word (of string)
LODSB	Load Byte (of string)
LODSW	Load Word (of string)
LOOP	Loop
LOOPE	Loop while Equal
LOOPNE	Loop while Not Equal

Mnemonic	Description
LOOPNZ	Loop while Not Zero
LOOPZ	Loop while Zero
MOV	Move
MOVS	Move Byte or Word (of string)
MOVSB	Move Byte (of string)
MOVSW	Move Word (of string)
MUL	Multiply
NEG	Negate
NOT	Not
OR	Or
OUT	Output Byte or Word
POP	Pop
POPF	Pop Flags
PUSH	Push
PUSHF	Push Flags
RCL	Rotate through Carry Left
RCR	Rotate through Carry Right
REP	Repeat
REPE	Repeat while Equal
REPNE	Repeat while Not Equal
REPNZ	Repeat while Not Zero
REPZ	Repeat while Zero
RET	Return (intra segment)
RETF	Return (inter segment)
ROL	Rotate Left
ROR	Rotate Right
SAHF	Store AH into Flags
SAL	Shift Arithmetic Left
SAR	Shift Arithmetic Right
SBB	Subtract with Borrow
SCAS	Scan Byte or Word (of string)
SCASB	Scan Byte (of string)
SCASW	Scan Word (of string)
SHL	Shift Left
SHR	Shift Right
STC	Set Carry
STD	Set Direction
STI	Set Interrupt
STOS	Store Byte or Word (of string)
STOSB	Store Byte (of string)
STOSW	Store Word (of string)
SUB	Subtract
TEST	Test

Mnemonic	Description
WAIT	Wait
XCHG	Exchange
XLAT	Translate
XOR	Exclusive Or

The following sections define the specific syntax and required operand types for each instruction, without reference to labels or comments. The instruction definitions are presented in tables for easy reference. For a more detailed description of each instruction, see Intel's *MCS-86 Assembly Language Reference Manual.* For descriptions of the instruction bit patterns and operations, see Intel's *MCS-86 User's Manual.*

The instruction-definition tables present ASM-86 instruction statements as combinations of mnemonics and operands. A mnemonic is a symbolic representation for an instruction, and its operands are its required parameters. Instructions can take zero, one or two operands. When two operands are specified, the left operand is the instruction's destination operand, and the two operands are separated by a comma.

The instruction-definition tables organize ASM-86 instructions into functional groups. Within each table, the instructions are listed alphabetically. The table below shows the symbols used in the instruction-definition tables to define operand types.

Operand Type Symbols

Symbol	Operand Type
numb	any NUMERIC expression
numb8	any NUMERIC expression which evaluates to an 8-bit number
acc	accumulator register, AX or AL
reg	any general purpose register, not segment register
reg16	a 16-bit general purpose register, not segment register
segreg	any segment register: CS, DS, SS, or ES
mem	any ADDRESS expression, with or without base- and/or index-addressing modes, such as: variable variable+3 variable[bx] variable[SI] variable[BX+SI] [BX] [BP+DI]
simpmem	any ADDRESS expression WITHOUT base- and index-addressing modes, such as: variable variable+4
mem|reg	any expression symbolized by "reg" or "mem"
mem|reg16	any expression symbolized by "mem|reg", but must be 16 bits
label	any ADDRESS expression which evaluates to a label
lab8	any "label" which is within ± 128 bytes distance from the instruction

The 8086 CPU has nine single-bit Flag registers which reflect the state of the CPU. The user cannot access these registers directly, but can test them to determine the effects of an executed instruction upon an operand or register. The effects of instructions on Flag registers are also described in the instruction-definition tables, using the symbols shown in the table below to represent the nine Flag registers.

Flag Register Symbols

AF	Auxiliary-Carry-Flag
CF	Carry-Flag
DF	Direction-Flag
IF	Interrupt-Enable-Flag
OF	Overflow-Flag
PF	Parity-Flag
SF	Sign-Flag
TF	Trap-Flag
ZF	Zero-Flag

Data Transfer Instructions

There are four classes of data transfer operations: general purpose, accumulator specific, address-object, and flag. Only SAHF and POPF affect flag settings. Note in the following table that if acc = AL, a byte is transferred, but if acc = AX, a word is transferred.

Data Transfer Instructions

Syntax		Result
IN	acc,numb8	Transfer data from input port given by numb8 (0–255) to accumulator.
IN	acc,DX	Transfer data from input port given by DX register (0-0FFFFH) to accumulator.
LAHF		Transfer flags to the AH register.

Data Transfer Instructions (continued)

Syntax		Result
LDS	reg16,mem	Transfer the segment part of the memory address (DWORD variable) to the DS segment register; transfer the offset part to a general purpose 16-bit register.
LEA	reg16,mem	Transfer the offset of the memory address to a (16-bit) register.
LES	reg16,mem	Transfer the segment part of the memory address to the ES segment register; transfer offset part to a 16-bit general purpose register.
MOV	reg,mem|reg	Move memory or register to register.
MOV	mem|reg,reg	Move register to memory or register.
MOV	mem|reg,numb	Move immediate data to memory or register.
MOV	segreg,mem|reg16	Move memory or register to segment register.
MOV	mem|reg16,segreg	Move segment register to memory or register.
OUT	numb8,acc	Transfer data from accumulator to output port (0–255) given by numb8.
OUT	DX,acc	Transfer data from accumulator to output port (0–0FFFFH) given by DX register.
POP	mem|reg16	Move top stack element to memory or register.

Data Transfer Instructions (continued)

Syntax		Result
POP	segreg	Move top stack element to segment register; note that CS segment register not allowed.
POPF		Transfer top stack element to flags.
PUSH	mem|reg16	Move memory or register to top stack element.
PUSH	segreg	Move segment register to top stack element.
PUSHF		Transfer flags to top stack element.
SAHF		Transfer the AH register to flags.
XCHG	reg,mem|reg	Exchange register and memory or register.
XCHG	mem|reg,reg	Exchange memory or register and register.
XLAT	mem|reg	Perform table lookup translation, table given by "mem|reg", which is always BX. Replaces AL with AL offset from BX.

Arithmetic, Logic, and Shift Instructions

The 8086 CPU performs the four basic mathematical operations in several different ways. It supports both 8- and 16-bit operations and also signed and unsigned arithmetic.

Six of the nine flag bits are set or cleared by most arithmetic operations to reflect the result of the operation. The following table summarizes the effects of arithmetic instructions on flag bits. Subsequent tables define arithmetic instructions and logical and shift instructions.

Effects of Arithmetic Instructions on Flags

CF is set if the operation resulted in a carry out of (from addition) or a borrow into (from subtraction) the high-order bit of the result; otherwise CF is cleared.

AF is set if the operation resulted in a carry out of (from addition) or a borrow into (from subtraction) the low-order four bits of the result; otherwise AF is cleared.

ZF is set if the result of the operation is zero; otherwise ZF is cleared.

SF is set if the result is negative.

PF is set if the modulo 2 sum of the low-order eight bits of the result of the operation is 0 (even parity); otherwise PF is cleared (odd parity).

OF is set if the operation resulted in an overflow; the size of the result exceeded the capacity of its destination.

Syntax		Result
AAA		adjust unpacked BCD (ASCII) for addition–adjusts AL
AAD		adjust unpacked BCD (ASCII) for division–adjusts AL
AAM		adjust unpacked BCD (ASCII) for multiplication–adjusts AX
AAS		adjust unpacked BCD (ASCII) for subtraction–adjusts AL
ADC	reg,mem|reg	add (with carry) memory or register to register
ADC	mem|reg,reg	add (with carry) register to memory or register
ADC	mem|reg,numb	add (with carry) immediate data to memory or register
ADD	reg,mem|reg	add memory or register to register
ADD	mem|reg,reg	add register to memory or register
ADD	mem|reg,numb	add immediate data to memory or register
CBW		convert byte in AL to word in AH by sign extension
CWD		convert word in AX to double word in DX/AX by sign extension
CMP	reg,mem|reg	compare register with memory or register
CMP	mem|reg,reg	compare memory or register with register

Syntax		Result
CMP	mem|reg,numb	compare data constant with memory or register
DAA		decimal adjust for addition, adjusts AL
DAS		decimal adjust for subtraction, adjusts AL
DEC	mem|reg	subtract 1 from memory or register
INC	mem|reg	add 1 to memory or register
DIV	mem|reg	divide (unsigned) accumulator (AX or AL) by memory or register: if byte results, AL = quotient, AH = remainder, if word results, AX = quotient, DX = remainder
IDIV	mem|reg	divide (signed) accumulator (AX or AL) by memory or register–quotient and remainder stored as in DIV
IMUL	mem|reg	multiply (signed) memory or register by accumulator (AX or AL)–if byte, results in AH, AL; if word, results in DX, AX
MUL	mem|reg	multiply (unsigned) memory or register by accumulator (AX or AL)–results stored as in IMUL
NEG	mem|reg	two's complement memory or register
SBB	reg,mem|reg	subtract (with borrow) memory or register from register

Arithmetic Instructions (continued)

Syntax		Result
SBB	mem|reg,reg	subtract (with borrow) register from memory or register
SBB	mem|reg,numb	subtract (with borrow) immediate data from memory or register
SUB	reg,mem|reg	subtract memory or register from register
SUB	mem|reg,reg	subtract register from memory or register
SUB	mem|reg,numb	subtract data constant from memory or register

Logic and Shift Instructions

Syntax		Result
AND	reg,mem|reg	perform bitwise logical "and" of a register and memory register
AND	mem|reg,reg	perform bitwise logical "and" of memory register and register
AND	mem|reg,numb	perform bitwise logical "and" of memory register and data constant
NOT	mem|reg	form one's complement of memory or register
OR	reg,mem|reg	perform bitwise logical "or" of a register and memory register
OR	mem|reg,reg	perform bitwise logical "or" of memory register and register

Logic and Shift Instructions (continued)

Syntax		Result
OR	mem|reg,numb	perform bitwise logical "or" of memory register and data constant
RCL	mem|reg,1	rotate memory or register 1 bit left through carry flag
RCL	mem|reg,CL	rotate memory or register left through carry flag, number of bits given by CL register
RCR	mem|reg,1	rotate memory or register 1 bit right through carry flag
RCR	mem|reg,CL	rotate memory or register right through carry flag, number of bits given by CL register
ROL	mem|reg,1	rotate memory or register 1 bit left
ROL	mem|reg,CL	rotate memory or register left, number of bits given by CL register
ROR	mem|reg,1	rotate memory or register 1 bit right
ROR	mem|reg,CL	rotate memory or register right, number of bits given by CL register
SAL	mem|reg,1	shift memory or register 1 bit left, shift in low-order zero bits
SAL	mem|reg,CL	shift memory or register left, number of bits given by CL register, shift in low-order zero bits

Logic and Shift Instructions (continued)

Syntax		Result
SAR	mem|reg,1	shift memory or register 1 bit right, shift in high-order bits equal to the original high-order bit
SAR	mem|reg,CL	shift memory or register right, number of bits given by CL register, shift in high-order bits equal to the original high-order bit
SHL	mem|reg,1	shift memory or register 1 bit left, shift in low-order zero bits –note that SHL is a different mnemonic for SAL
SHL	mem|reg,CL	shift memory or register left, number of bits given by CL register, shift in low-order zero bits–note that SHL is a different mnemonic for SAL
SHL	mem|reg,1	shift memory or register 1 bit right, shift in high-order zero bits
SHR	mem|reg,CL	shift memory or register right, number of bits given by CL register, shift in high-order zero bits
TEST	reg,mem|reg	perform bitwise logical "and" of a register and memory or register–set condition flags but do not change destination
TEST	mem|reg,reg	perform bitwise logical "and" of memory register and register–set condition flags but do not change destination

Logic and Shift Instructions (continued)

Syntax		Result
TEST	mem|reg,numb	perform bitwise logical "and"–test of memory register and data constant–set condition flags but do not change destination
XOR	reg,mem|reg	perform bitwise logical "exclusive OR" of a register and memory or register
XOR	mem|reg,reg	perform bitwise logical "exclusive OR" of memory register and register
XOR	mem|reg,numb	perform bitwise logical "exclusive OR" of memory register and data constant

String Instructions

String instructions take zero, one or two operands. The operands specify only the operand type, determining whether operation is on bytes or words. If there are two operands, the source operand is addressed by the SI register and the destination operand is addressed by the DI register. The DI and SI registers are always used for addressing. Note that for string operations, destination operands addressed by DI must always reside in the Extra Segment (ES).

String Instructions

Syntax	Result
CMPS mem|reg,mem|reg	subtract source from destination, affect flags, but do not return result
CMPSB	an alternate mnemonics for CMPS which assumes a byte operand
CMPSW	an alternate mnemonics for CMPS which assumes a word operand
LODS mem|reg	transfer a byte or word from the source operand to the accumulator
LODSB	an alternate mnemonic for LODS which assumes a byte operand
LODSW	an alternate mnemonic for LODS which assumes a word operand
MOVS mem|reg,mem|reg	move 1 byte (or word) from source to destination
MOVSB	an alternate mnemonic for MOVS which assumes a byte operand

Syntax	Result
MOVSW	an alternate mnemonic for MOVS which assumes a word operand
SCAS mem|reg	subtract destination operand from accumulator (AX or AL), affect flags, but do not return result
SCASB	an alternate mnemonic for SCAS which assumes a byte operand
SCASW	an alternate mnemonic for SCAS which assumes a word operand
STOS mem|reg	transfer a byte or word from accumulator to the destination operand
STOSB	an alternate mnemonic for STOS which assumes a byte operand
STOSW	an alternate mnemonic for STOS which assumes a word operand

The following table defines prefixes for string instructions. A prefix repeats its string instruction the number of times contained in the CX register, which is decremented by 1 for each iteration. Prefix mnemonics precede the string instruction mnemonic in the statement line as shown in "Statements in Chapter 7."

Prefix Instructions

Syntax	Result
REP	repeat until CX register is zero
REPZ	repeat until CX register is zero and zero flag (ZF) is not zero
REPE	equal to "REPZ"
REPNZ	repeat until CX register is zero and zero flag (ZF) is zero
REPNE	equal to "REPNZ"

Control Transfer Instructions

There are four classes of control transfer instructions:

- calls, jumps, and returns
- conditional jumps
- iterational control
- interrupts

All control transfer instructions cause program execution to continue at some new location in memory, possibly in a new code segment. The transfer may be absolute or depend upon a certain condition. The following table defines control transfer instructions. In the definitions of conditional jumps, "above" and "below" refer to the relationship between unsigned values, and "greater than" and "less than" refer to the relationship between signed values.

Control Transfer Instructions

Syntax		Result
CALL	label	push the offset address of the next instruction on the stack, jump to the target label
CALL	mem|reg16	push the offset address of the next instruction on the stack, jump to location indicated by contents of specified memory or register
CALLF	label	push CS segment register on the stack, push the offset address of the next instruction on the stack (after CS), jump to the target label
CALLF	mem	push CS register on the stack, push the offset address of the next instruction on the stack, jump to location indicated by contents of specified double word in memory
INT	numb8	push the flag registers (as in PUSHF), clear TF and IF flags, transfer control with an indirect call through any one of the 256 interrupt-vector elements–uses three levels of stack
INTO		if OF (the overflow flag) is set, push the flag registers (as in PUSHF), clear TF and IF flags, transfer control with an indirect call through interrupt-vector element 4 (location 10H)–if the OF flag is cleared, no operation takes place

Syntax		Result
IRET		transfer control to the return address saved by a previous interrupt operation, restore saved flag registers, as well as CS and IP–pops three levels of stack
JA	lab8	jump if "not below or equal" or "above" ((CF or ZF) = 0)
JAE	lab8	jump if "not below" or "above or equal" (CF = 0)
JB	lab8	jump if "below" or "not above or equal" (CF = 1)
JBE	lab8	jump if "below or equal" or "not above" ((CF or ZF) = 1)
JC	lab8	same as "JB"
JCXZ	lab8	jump to target label if CX register is zero
JE	lab8	jump if "equal" or "zero" (ZF = 1)
JG	lab8	jump if "not less or equal" or "greater" (((SF xor OF) or ZF) = 0)
JGE	lab8	jump if "not less" or "greater or equal" ((SF xor OF) = 0)
JL	lab8	jump if "less" or "not greater or equal" ((SF xor OF) = 1)
JLE	lab8	jump if "less or equal" or "not greater" (((SF xor OF) or ZF) = 1)
JMP	label	jump to the target label

Syntax		Result
JMP	mem|reg16	jump to location indicated by contents of specified memory or register
JMPF	label	jump to the target label possibly in another code segment
JMPS	lab8	jump to the target label within ± 128 bytes from instruction
JNA	lab8	same as "JBE"
JNAE	lab8	same as "JB"
JNB	lab8	same as "JAE"
JNBE	lab8	same as "JA"
JNC	lab8	same as "JNB"
JNE	lab8	jump if "not equal" or "not zero" (ZF = 0)
JNG	lab8	same as "JLE"
JNGE	lab8	same as "JL"
JNL	lab8	same as "JGE"
JNLE	lab8	same as "JG"
JNO	lab8	jump if "not overflow" (OF = 0)
JNP	lab8	jump if "not parity" or "parity odd"
JNS	lab8	jump if "not sign"
JNZ	lab8	same as "JNE"

Control Transfer Instructions (continued)

Syntax		Result
JO	lab8	jump if "overflow" (OF = 1)
JP	lab8	jump if "parity" or "parity even" (PF = 1)
JPE	lab8	same as "JP"
JPO	lab8	same as "JNP"
JS	lab8	jump if "sign" (SF = 1)
JZ	lab8	same as "JE"
LOOP	lab8	decrement CX register by one, jump to target label if CX is not zero
LOOPE	lab8	decrement CX register by one, jump to target label if CX is not zero and the ZF flag is set–"loop while zero" or "loop while equal"
LOOPNE	lab8	decrement CX register by one, jump to target label if CX is not zero and ZF flag is cleared–"loop while not zero" or "loop while not equal"
LOOPNZ	lab8	same as "LOOPNE"
LOOPZ	lab8	same as "LOOPE"
RET		return to the return address pushed by a previous CALL instruction, increment stack pointer by 2
RET	numb	return to the address pushed by a previous CALL, increment stack pointer by 2+numb

Control Transfer Instructions (continued)

Syntax		Result
RETF		return to the address pushed by a previous CALLF instruction, increment stack pointer by 4
RETF	numb	return to the address pushed by a previous CALLF instruction, increment stack pointer by 4+numb

Processor Control Instructions

Processor control instructions manipulate the flag registers. Moreover, some of these instructions can synchronize the 8086 CPU with external hardware.

Processor Control Instructions

Syntax		Results
CLC		clear CF flag
CLD		clear DF flag, causing string instructions to auto-increment the operand pointers
CLI		clear IF flag, disabling maskable external interrupts
CMC		complement CF flag
ESC	numb8,mem|reg	do no operation other than compute the effective address and place it on the address bus (ESC is used by the 8087 numeric co-processor), "numb8" must be in the range 0 to 63

Processor Control Instructions (continued)

Syntax	Results
HLT	cause 8086 processor to enter halt state until an interrupt is recognized
LOCK	PREFIX instruction, cause the 8086 processor to assert the "bus-lock" signal for the duration of the operation caused by the following instruction–the LOCK prefix instruction may precede any other instruction–buslock prevents co-processors from gaining the bus; this is useful for shared-resource semaphores
NOP	no operation is performed
STC	set CF flag
STD	set DF flag, causing string instructions to auto-decrement the operand pointers
STI	set IF flag, enabling maskable external interrupts
WAIT	cause the 8086 processor to enter a "wait" state if the signal on its "TEST" pin is not asserted

Mnemonic Differences

The CP/M-86 8086 assembler uses the same instruction mnemonics as the INTEL 8086 assembler except for explicitly specifying far and short jumps, calls and returns. The following table shows the four differences:

Mnemonic Differences

Mnemonic Function	CP/M-86	Intel
Intra segment short jump:	JMPS	JMP
Inter segment jump:	JMPF	JMP
Inter segment return:	RETF	RET
Inter segment call:	CALLF	CALL

CHAPTER 10. ASM-86 ERROR MESSAGES

Contents

ASM-86 Fatal Error Messages

There are two types of error messages produced by ASM-86: fatal errors and diagnostic errors. Fatal errors occur when ASM-86 is unable to continue assembling. Diagnostic error messages report problems with the syntax and semantics of the program being assembled. The following messages indicate fatal errors encountered by ASM-86 during assembly:

NO FILE

The indicated source or include file could not be found on the indicated drive.

DISK FULL

There is not enough disk space for the output files (LST, H86 and SYM). You should either erase some unnecessary files or get another diskette with more room and run ASM-86 again.

DIRECTORY FULL

There is not enough directory space for the output files. You should either erase some unnecessary files or get another diskette with more directory room and run ASM-86 again.

DISK READ ERROR–{filespec}

A source or include file could not be read properly. This is usually the result of an unexpected end of file. Correct the problem in your source file.

CANNOT CLOSE

An output file cannot be closed. This is a fatal error that terminates ASM-86 execution. The user should take appropriate action after checking to see if the correct diskette is in the drive and that the diskette is not write-protected.

SYMBOL TABLE OVERFLOW

There is not enough memory for the symbol table. Either reduce the length and/or number of symbols, or reassemble on a system with more memory available.

PARAMETER ERROR

A parameter in the command tail of the ASM-86 command was specified incorrectly. Example:

```
ASM86 TEST $S;
```

ASM-86 Diagnostic Error Messages

ASM-86 reports semantic and syntax errors by placing a numbered message in front of the erroneous source line. If there is more than one error in the line, only the first one is reported. The following messages indicate diagnostic errors encountered by ASM-86.

** ERROR NO: 0 ILLEGAL FIRST ITEM

The first item on a source line is not a valid identifier, directive or mnemonic. Example:

```
1234H
```

** ERROR NO: 1 MISSING PSEUDO INSTRUCTION

The first item on a source line is a valid identifier and the second item is not a valid directive which may be preceded by an identifier. Example:

```
THIS IS A MISTAKE
```

** ERROR NO: 2 ILLEGAL PSEUDO INSTRUCTION

Either a required identifier in front of a pseudo instruction is missing, or an identifier appears before a pseudo instruction which doesn't allow an identifier.

** ERROR NO: 3 DOUBLE DEFINED VARIABLE

An identifier used as the name of a variable is used elsewhere in the program as the name of a variable or label. Example:

```
X       DB  5
        ...
X       DB  123H
```

** ERROR NO: 4 DOUBLE DEFINED LABEL

An identifier used as a label is used elsewhere in the program as a label or variable name. Example:

```
LAB3:  MOV   BX,5
       ...
LAB3:  CALL  MOVE
```

** ERROR NO: 5 UNDEFINED INSTRUCTION

The item following a label on a source line is not a valid instruction. Example:

```
DONE:  BAD     INSTR
```

** ERROR NO: 6 GARBAGE AT END OF LINE–IGNORED

Additional items were encountered on a line when ASM-86 was expecting an end of line. Examples:

```
NOLIST  4
MOV     AX,4RET
```

** ERROR NO: 7 OPERAND(S) MISMATCH INSTRUCTION

Either an instruction has the wrong number of operands, or the types of the operands don't match. Examples:

```
        MOV   CX,1,2
X       DB    0
        MOV   AX,X
```

** ERROR NO: 8 ILLEGAL INSTRUCTION OPERANDS

An instruction operand is improperly formed. Examples:

```
MOV   [BP+SP],1234
CALL  BX|1
```

** ERROR NO: 9 MISSING INSTRUCTION

A prefix on a source line is not followed by an instruction. Example:

```
REPNZ
```

** ERROR NO: 10 UNDEFINED ELEMENT OF EXPRESSION

An identifier used as an operand is not defined or has been illegally forward referenced. Examples:

```
        JMP   X
A       EQU   B
B       EQU   5
        MOV   AL,B
```

** ERROR NO: 11 ILLEGAL PSEUDO OPERAND

The operand in a directive is invalid. Examples:

```
X       EQU   0AGH
        TITLE UNQUOTED STRING
```

**** ERROR NO: 12 NESTED "IF" ILLEGAL–"IF" IGNORED**

The maximum nesting level for IF statements has been exceeded.

**** ERROR NO: 13 ILLEGAL "IF" OPERAND–"IF" IGNORED**

Either the expression in an IF statement is not numeric, or it contains a forward reference.

**** ERROR NO: 14 NO MATCHING "IF" FOR "ENDIF"**

An ENDIF statement was encountered without a matching IF statement.

**** ERROR NO: 15 SYMBOL ILLEGALLY FORWARD REFERENCED – NEGLECTED**

The indicated symbol was illegally forward referenced in an ORG, RS, EQU or IF statement.

**** ERROR NO: 16 DOUBLE DEFINED SYMBOL–TREATED AS UNDEFINED**

The identifier used as the name of an EQU directive is used as a name elsewhere in the program.

**** ERROR NO: 17 INSTRUCTION NOT IN CODE SEGMENT**

An instruction appears in a segment other than a CSEG.

**** ERROR NO: 18 FILE NAME SYNTAX ERROR**

The filename in an INCLUDE directive is improperly formed. Example:

```
INCLUDE FILE.A86X
```

**** ERROR NO: 19 NESTED INCLUDE NOT ALLOWED**

An INCLUDE directive was encountered within a file already being included.

**** ERROR NO: 20 ILLEGAL EXPRESSION ELEMENT**

An expression is improperly formed. Examples:

```
X   DB    12X
    DW    (4 * )
```

**** ERROR NO: 21 MISSING TYPE INFORMATION IN OPERAND(S)**

Neither instruction operand contains sufficient type information. Example:

```
MOV     [BX],10
```

**** ERROR NO: 22 LABEL OUT OF RANGE**

The label referred to in a call, jump or loop instruction is out of range. The label may be defined in a segment other than the segment containing the instruction. In the case of short instructions (JMPS, conditional jumps and loops), the label is more than 128 bytes from the location of the following instruction.

** ERROR NO: 23 MISSING SEGMENT INFORMATION IN OPERAND

The operand in a CALLF or JMPF instruction (or an expression in a DD directive) does not contain segment information. The required segment information can be supplied by including a numeric field in the segment directive as shown:

```
        CSEG   1000H
X:
        . . .
        JMPF   X
        DD     X
```

** ERROR NO: 24 ERROR IN CODEMACRO BUILDING

Either a codemacro contains invalid statements, or a codemacro directive was encountered outside a codemacro.

CHAPTER 11. DDT-86

Contents

DDT-86 Operation

The DDT-86 program allows the user to test and debug programs interactively in a CP/M-86 environment. You should be familiar with the 8088 processor, ASM-86 and the CP/M-86 operating system before using DDT-86.

Invoking DDT-86

Invoke DDT-86 by entering one of the following commands:

DDT86
DDT86 filespec

The first command simply loads and executes DDT-86. After displaying its sign-on message and prompt character, "-", DDT-86 is ready to accept operator commands. The second command is similar to the first, except that after DDT-86 is loaded, it loads the file specified by file specification. If the filetype is omitted from file specification, .CMD is assumed. Note that DDT-86 cannot load a file of type .H86. The file must be in the CMD file format produced by the GENCMD utility. The second form of the invoking command is equivalent to the sequence:

A>DDT86
DDT86 x.x
-Efilespec

At this point, the program that was loaded is ready for execution.

DDT-86 Command Conventions

When DDT-86 is ready to accept a command, it prompts the operator with a hyphen, "-". In response, the operator can type a command line or a Ctrl-C to end the debugging session. A command line can have up to 64 characters, and must be terminated with a carriage return. While entering the command, use standard CP/M-86 line-editing functions (Ctrl-X, Ctrl-H, Ctrl-R, etc.) to correct typing errors. DDT-86 does not process the command line until a carriage return is entered.

The first character of each command line determines the command action. The table below summarizes DDT-86 commands. DDT-86 commands are defined individually in the following sections.

A	enter assembly language statements
B	compare blocks of memory
D	display memory in hexadecimal and ASCII
E	load program for execution
F	fill memory block with a constant
G	begin execution with optional breakpoints
H	hexadecimal arithmetic
I	set up file control block and command tail
L	list memory using 8086 mnemonics
M	move memory block
R	read disk file into memory
S	set memory to new values
T	trace program execution
U	untraced program monitoring
V	show memory layout of disk file read
W	write contents of memory block to disk
X	examine and modify CPU state

The command character may be followed by one or more arguments, which may be hexadecimal values, file specifications or other information, depending on the command. Arguments are separated from each other by commas or spaces. No spaces are allowed between the command character and the first argument. Note that if the first character of a DDT-86 command line is a semicolon, ;, the entire line is treated as a comment and is ignored.

Specifying a 20-Bit Address

Most DDT-86 commands require one or more addresses as operands. Because the 8088 can address up to 1 megabyte of memory, addresses must be 20-bit values. Enter a 20-bit address as follows:

```
ssss:oooo
```

where ssss represents an optional 16-bit segment number and oooo is a 16-bit offset. DDT-86 combines these values to produce a 20-bit effective address as follows:

```
  ssss0
+ oooo
------
  eeeee
```

The optional value ssss may be a 16-bit hexadecimal value or the name of a segment register. If a segment register name is specified, the value of ssss is the contents of that register in the user's CPU state, as displayed by the X command. If omitted, a default value appropriate to the command being executed is used as described in the following section.

Terminating DDT-86

Terminate DDT-86 by typing a Ctrl-C in response to the hyphen prompt. This returns control to CP/M-86. If you use DDT-86 to patch a file, write the file to disk using the W command before exiting DDT-86.

DDT-86 Operation With Interrupts

DDT-86 operates with interrupts enabled or disabled, and preserves the interrupt state of the program being executed under DDT-86. When DDT-86 has control of the CPU, either when it is initially invoked or when it regains control from the program being tested, the condition of the interrupt flag is the same as it was when DDT-86 was invoked. While the program being tested has control of the CPU as the result of a G command, the user's CPU state determines the state of the interrupt flag. When the program is being traced using T or U commands, interrupts are always disabled during the execution of the traced instruction. This allows normal tracing of programs in systems where interrupts occur frequently; for example, from a timer.

DDT-86 Commands

This section defines DDT-86 commands and their arguments. DDT-86 commands give the user control of program execution and allow the user to display and modify system memory and the CPU state.

The A (Assemble) Command

The A command assembles 8086 mnemonics directly into memory. The form is:

```
As
```

where s is the 20-bit address where assembly is to start. DDT-86 responds to the A command by displaying the address of the memory location where assembly is to begin. At this point the operator enters assembly language statements as described in "Assembly Language Syntax." When a statement is entered, DDT-86 converts it to machine code, places the value(s) in memory, and displays the address of the next available memory location. This process continues until the user enters a blank line or a line containing only a period.

DDT-86 responds to invalid statements by displaying a question mark, ?, and redisplaying the current assembly address.

The B (Block Compare) Command

The B command compares two blocks of memory and displays any differences on the screen. The form is:

```
Bs1,f1,s2
```

where s1 is the 20-bit address of the start of the first block; f1 is the offset of the final byte of the first block, and s2 is the 20-bit address of the start of the second block. If the segment is not specified in s2, the same value is used that was used for s1.

Any differences in the two blocks are displayed at the screen in the following form:

```
s1:o1 b1      s2:o2 b2
```

where s1:o1 and s2:o2 are the addresses in the blocks; b1 and b2 are the values at the indicated addresses. If no differences are displayed, the blocks are identical.

The D (Display) Command

The D command displays the contents of memory as 8-bit or 16-bit hexadecimal values and in ASCII. The forms are:

```
D
Ds
Ds,f
DW
DWs
DWs,f
```

where s is the 20-bit address where the display is to start, and f is the 16-bit offset within the segment specified in s where the display is to finish.

Memory is displayed on one or more display lines. Each display line shows the values of up to 16 memory locations. For the first three forms, the display line appears as follows:

```
ssss:oooo bb bb . . . bb cc . . . c
```

where ssss is the segment being displayed and oooo is the offset within segment ssss. The bb's represent the contents of the memory locations in hexadecimal, and the c's represent the contents of memory in ASCII. Any non-graphic ASCII characters are represented by periods.

In response to the first form shown above, DDT-86 displays memory from the current display address for 12 display lines. The response to the second form is similar to the first, except that the display address is first set to the 20-bit address s. The third form displays the memory block between locations s and f. The next three forms are analogous to the first three, except that the contents of memory are displayed as 16-bit values, rather than 8-bit values, as shown below:

```
ssss:oooo wwww wwww . . . wwww cccc . . . cc
```

During a long display, you can terminate the D command by typing any character at the keyboard.

The E (Load for Execution) Command

The E command loads a file generated by the GENCMD utility into memory so that a subsequent G, T or U command can begin program execution. The E command takes the form:

```
E filespec
```

where filespec is the name of the file to be loaded. If no filetype is specified, .CMD is assumed. The contents of the user segment registers and IP register are altered according to the information in the header of the file loaded.

An E command releases any blocks of memory allocated by any previous E or R commands or by programs executed under DDT-86. Thus only one file at a time can be loaded for execution.

When the load is complete, DDT-86 displays the start and end addresses of each segment in the file loaded. Use the V command to redisplay this information at a later time.

If the file does not exist or cannot be successfully loaded in the available memory, DDT-86 issues an error message.

The F (Fill) Command

The F command fills an area of memory with a byte or word constant. The forms are:

```
Fs,f,b
FWs,f,w
```

where s is a 20-bit starting address of the block to be filled, and f is a 16-bit offset of the final byte of the block within the segment specified in s.

In response to the first form, DDT-86 stores the 8-bit value b in locations s through f. In the second form, the 16-bit value w is stored in locations s through f in standard form, low 8 bits first followed by high 8 bits.

If s is greater than f or the value b is greater than 255, DDT-86 responds with a question mark. DDT-86 issues an error message if the value stored in memory cannot be read back successfully, indicating faulty or non-existent RAM at the location indicated.

The G (Go) Command

The G command transfers control to the program being tested, and optionally sets one or two breakpoints. The forms are:

```
G
G,b1
G,b1,b2
Gs
Gs,b1
Gs,b1,b2
```

where s is a 20-bit address where program execution is to start, and b1 and b2 are 20-bit addresses of breakpoints. If no segment value is supplied for any of these three addresses, the segment value defaults to the contents of the CS register.

In the first three forms, no starting address is specified, so DDT-86 derives the 20-bit address from the user's CS and IP registers. The first form transfers control to the user's program without setting any breakpoints. The next two forms set one and two breakpoints, respectively, before passing control to the user's program. The next three forms are analogous to the first three, except that the user's CS and IP registers are first set to s.

Once control has been transferred to the program under test, it executes in real time until a breakpoint is encountered. At this point, DDT-86 regains control, clears all breakpoints, and indicates the address at which execution of the program under test was interrupted as follows:

```
*ssss:oooo
```

where ssss corresponds to the CS and oooo corresponds to the IP where the break occurred. When a breakpoint returns control to DDT-86, the instruction at the breakpoint address has not yet been executed.

The H (Hexadecimal Math) Command

The H command computes the sum and difference of two 16-bit values. The form is:

```
Ha,b
```

where a and b are the values whose sum and difference are to be computed. DDT-86 displays the sum (ssss) and the difference (dddd) truncated to 16 bits on the next line as shown below:

```
ssss dddd
```

The I (Input Command Tail) Command

The I command prepares a file control block and command tail buffer in DDT-86's base page, and copies this information into the base page of the last file loaded with the E command. The form is:

```
I command tail
```

where command tail is a character string which usually contains one or more file specifications. The first file specification is parsed into the default file control block at 005CH. The optional second file specification (if specified) is parsed into the second part of the default file control block beginning at 006CH. The characters in the command tail are also copied into the default command buffer at 0080H. The length of the command tail is stored at 0080H, followed by the character string terminated with a binary zero.

If a file has been loaded with the E command, DDT-86 copies the file control block and command buffer from the base page of DDT-86 to the base page of the program loaded. The location of DDT-86's base page can be obtained from the SS register in the user's CPU state when DDT-86 is invoked. The location of the base page of a program loaded with the E command is the value displayed for DS upon completion of the program load.

The L (List) Command

The L command lists the contents of memory in assembly language. The forms are:

```
L
Ls
Ls,f
```

where s is a 20-bit address where the list is to start, and f is a 16-bit offset within the segment specified in s where the list is to finish.

The first form lists twelve lines of disassembled machine code from the current list address. The second form sets the list address to s and then lists twelve lines of code. The last form lists disassembled code from s through f. In all three cases, the list address is set to the next unlisted location in preparation for a subsequent L command. When DDT-86 regains control from a program being tested (see G, T and U commands), the list address is set to the current value of the CS and IP registers.

Terminate displays by typing any key during the list process. Or, enter Ctrl-S to halt the display temporarily.

The section "Assembly Language Syntax" discusses the syntax of statements produced by the L command.

The M (Move) Command

The M command moves a block of data values from one area of memory to another. The form is:

```
Ms,f,d
```

where s is the 20-bit starting address of the block to be moved; f is the offset of the final byte to be moved within the segment described by s, and d is the 20-bit address of the first byte of the area to receive the data. If the segment is not specified in d, the same value is used that was used for s. Note that if d is between s and f, part of the block being moved is overwritten before it is moved, because data is transferred starting from location s.

The R (Read) Command

The R command reads a file into a contiguous block of memory. The form is:

```
R filespec
```

where filespec is the name and type of the file to be read.

DDT-86 reads the file into memory and displays the start and end addresses of the block of memory occupied by the file. A V command can redisplay this information at a later time. The default display pointer (for subsequent D commands) is set to the start of the block occupied by the file.

The R command does not free any memory previously allocated by another R or E command. Thus a number of files can be read into memory without overlapping. The number of files that can be loaded is limited to seven, which is the number of memory allocations allowed by the BDOS, minus one for DDT-86 itself.

If the file does not exist or there is not enough memory to load the file, DDT-86 issues an error message.

The S (Set) Command

The S command can change the contents of bytes or words of memory. The forms are:

```
Ss
SWs
```

where s is the 20-bit address where the change is to occur.

DDT-86 displays the memory address and its current contents on the following line. In response to the first form, the display format is:

```
ssss:oooo bb
```

and in response to the second form:

```
ssss:oooo wwww
```

where bb and wwww are the contents of memory in byte and word formats, respectively.

In response to one of the above displays, you can choose to alter the memory location or to leave it unchanged. If a valid hexadecimal value is entered, the contents of the byte (or word) in memory is replaced with the value. If no value is entered, the contents of memory are unaffected and the contents of the next address are displayed. In either case, DDT-86 continues to display successive memory addresses and values until either a period or an invalid value is entered.

DDT-86 issues an error message if the value stored in memory cannot be read back successfully, indicating faulty or non-existent RAM at the location indicated.

The T (Trace) Command

The T command traces program execution for 1 to 0FFFFH program steps. The forms are:

```
T
Tn
TS
TSn
```

where n is the number of instructions to execute before returning control to the screen.

Before DDT-86 traces an instruction, it displays the current CPU state and the disassembled instruction. In the first two forms, the segment registers are not displayed, which allows the entire CPU state to be displayed on one line. The next two forms are analogous to the first two, except that all the registers are displayed, which forces the disassembled instruction to be displayed on the next line as in the X command.

In all of the forms, control transfers to the program under test at the address indicated by the CS and IP registers. If n is not specified, one instruction is executed. Otherwise DDT-86 executes n instructions, displaying the CPU state before each step. A long trace can be terminated before n steps have been executed by typing any character at the keyboard.

After a T command, the list address used in the L command is set to the address of the next instruction to be executed.

Note that DDT-86 does not trace through a system instruction, since DDT-86 itself makes operating system calls and the operating system is not reentrant. Instead, the entire sequence of instructions from the system interrupt through the return from the operating system is treated as one traced instruction.

The U (Untrace) Command

The U command is identical to the T command except that the CPU state is displayed only before the first instruction is executed, rather than before every step. The forms are:

```
U
Un
US
USn
```

where n is the number of instructions to execute before returning control to the screen. Terminate the U command by striking any key at the keyboard.

The V (Value) Command

The V command displays information about the last file loaded with the E or R commands. The form is:

```
V
```

If the last file was loaded with the E command, the V command displays the start and end addresses of each of the segments contained in the file. If the last file was read with the R command, the V command displays the start and end addresses of the block of memory where the file was read. If neither the R nor E commands have been used, DDT-86 responds to the V command with a question mark, ?.

The W (Write) Command

The W command writes the contents of a contiguous block of memory to disk. The forms are:

```
W filespec
W filespec,s,f
```

where filespec is the filename and filetype of the disk file to receive the data, and s and f are the 20-bit first and last addresses of the block to be written. If the segment is not specified in f, DDT-86 uses the same value that was used for s.

If the first form is used, DDT-86 assumes the s and f values from the last file read with an R command. If no file was read with an R command, DDT-86 responds with a question mark, ?. This first form is useful for writing out files after patches have been installed, assuming the overall length of the file is unchanged.

In the second form where s and f are specified as 20-bit addresses, the low four bits of s are ignored. Thus the block being written must always start on a paragraph boundary.

If a file by the name specified in the W command already exists, DDT-86 deletes it before writing a new file.

The X (Examine CPU State) Command

The X command allows you to examine and alter the CPU state of the program under test. The forms are:

```
X
Xr
Xf
```

where r is the name of one of the 8086 CPU registers and f is the abbreviation of one of the CPU flags. The first form displays the CPU state in the format:

```
          AX  BX  CX  . . .  SS  ES  IP
--------  xxxx xxxx xxxx . . . xxxx xxxx xxxx
instruction
```

The nine hyphens at the beginning of the line indicate the state of the nine CPU flags. Each position may be either a hyphen, indicating that the corresponding flag is not set (0), or a one-character abbreviation of the flag name, indicating that the flag is set (1). The abbreviations of the flag names are shown in the table below. The instruction is the disassembled instruction at the next location to be executed, which is indicated by the CS and IP registers.

O	Overflow
D	Direction
I	Interrupt Enable
T	Trap
S	Sign
Z	Zero
A	Auxiliary Carry
P	Parity
C	Carry

The second form allows you to alter the registers in the CPU state of the program being tested. The r following the X is the name of one of the 16-bit CPU registers. DDT-86 responds by displaying the name of the register followed by its current value. If a carriage return is typed, the value of the register is not changed. If a valid value is typed, the contents of the register are changed to that value. In either case, the next register is then displayed. This process continues until a period or an invalid value is entered, or the last register is displayed.

The third form allows the operator to alter one of the flags in the CPU state of the program being tested. DDT-86 responds by displaying the name of the flag followed by its current state. If a carriage return is typed, the state of the flag is not changed. If a valid value is typed, the state of the flag is changed to that value. Only one flag may be examined or altered with each Xf command. Set or reset flags by entering a value of 1 or 0.

Default Segment Values

DDT-86 internally keeps track of the current segment value, making segment specification an optional part of a DDT–86 command. DDT-86 divides the command set into two types of commands, according to which segment a command defaults if no segment value is specified in the command line.

The first type of command pertains to the code segment: A (Assemble), L (List Mnemonics) and W (Write). These commands use the internal type-1 segment value if no segment value is specified in the command.

When invoked, DDT-86 sets the type-1 segment value to 0, and changes it when one of the following actions is taken:

- When a file is loaded by an E command, DDT-86 sets the type-1 segment value to the value of the CS register.
- When a file is read by an R command, DDT-86 sets the type-1 segment value to the base segment where the file was read.
- When an X command changes the value of the CS register, DDT-86 changes the type-1 segment value to the new value of the CS register.
- When DDT-86 regains control from a user program after a G, T or U command, it sets the type-1 segment value to the value of the CS register.
- When a segment value is specified explicitly in an A or L command, DDT-86 sets the type-1 segment value to the segment value specified.

The second type of command pertains to the data segment: B (Block Compare), D (Display), F (Fill), M (Move) and S (Set). These commands use the internal type-2 segment value if no segment value is specified in the command.

When invoked, DDT-86 sets the type-2 segment value to 0, and changes it when one of the following actions is taken:

- When a file is loaded by an E command, DDT-86 sets the type-2 segment value to the value of the DS register.

- When a file is read by an R command, DDT-86 sets the type-2 segment value to the base segment where the file was read.

- When an X command changes the value of the DS register, DDT-86 changes the type-2 segment value to the new value of the DS register.

- When DDT-86 regains control from a user program after a G, T or U command, it sets the type-2 segment value to the value of the DS register.

- When a segment value is specified explicitly in a B, D, F, M or S command, DDT-86 sets the type-2 segment value to the segment value specified.

When evaluating programs that use identical values in the CS and DS registers, all DDT-86 commands default to the same segment value unless explicitly overridden.

Note that the G (Go) command does not fall into either group, since it defaults to the CS register.

The table below summarizes DDT-86's default segment values.

Command	type-1	type-2
A	x	
B		x
D		x
E	u	u
F		x
G	u	u
H		
I		
L	x	
M		x
R	u	u
S		x
T	u	u
U	u	u
V		
W	x	
X	u	u

x – use this segment default if none specified; change default if specified explicitly
u – update this segment default

Assembly Language Syntax for A and L Commands

In general, the syntax of the assembly language statements used in the A and L commands is standard 8086/8088 assembly language. Several minor exceptions are listed below.

- DDT-86 assumes that all numeric values entered are hexadecimal.

- Up to three prefixes (LOCK, repeat, segment override) may appear in one statement, but they all must precede the opcode of the statement. Alternately, a prefix may be entered on a line by itself.

- The distinction between byte and word string instructions is made as follows:

byte	word
LODSB	LODSW
STOSB	STOSW
SCASB	SCASW
MOVSB	MOVSW
CMPSB	CMPSW

- The mnemonics for near and far control transfer instructions are as follows:

short	normal	far
JMPS	JMP	JMPF
	CALL	CALLF
	RET	RETF

- If the operand of a CALLF or JMPF instruction is a 20-bit absolute address, it is entered in the form:

```
ssss:oooo
```

  where ssss is the segment and oooo is the offset of the address.

- Operands that could refer to either a byte or word are ambiguous, and must be preceded either by the prefix "BYTE" or "WORD". These prefixes may be abbreviated to "BY" and "WO". For example:

```
INC     BYTE [BP]
NOT     WORD[1234]
```

  Failure to supply a prefix when needed results in an error message.

- Operands which address memory directly are enclosed in square brackets to distinguish them from immediate values. For example:

```
ADD     AX,5      ;add 5 to register AX
ADD     AX,[5]    ;add the contents of 5 to AX
```

- The forms of register indirect memory operands are:

  [pointer register]
  [index register]
  [pointer register + index register]

  where the pointer registers are BX and BP, and the index registers are SI and DI. Any of these forms may be preceded by a numeric offset. For example:

```
ADD     BX,[BP+SI]
ADD     BX,3[BP+SI]
ADD     BX,1D47[BP+SI]
```

DDT-86 Sample Session

In the following sample session, the user interactively debugs a simple sort program. Comments in square brackets explain the steps involved.

[Source file of program to test.]

```
A>type sort.a86
;
;       simple sort program
;
sort:
        mov     si,0                    ;initialize index
        mov     bx,offset nlist         ;bx = base of list
        mov     sw,0                    ;clear switch flag
comp:
        mov     al,[bx+si]              ;get byte from list
        cmp     al,1[bx+si]             ;compare with next byte
        jna     inci                    ;don't switch if in order
        xchg    al,1[bx+si]             ;do first part of switch
        mov     [bx+si],al              ;do second part
        mov     sw,1                    ;set switch flag
inci:
        inc     si                      ;increment index
        cmp     si,count                ;end of list?
        jnz     comp                    ;no, keep going
```

```
        test    sw,1                    ;done - any switches?
        jnz     sort                    ;yes, sort some more
done:
        jmp     done                    ;get here when list ordered
;
        dseg
        org     100h                    ;leave space for base page
;
nlist   db      3,8,4,6,31,6,4,1
count   equ     offset $ - offset nlist
sw      db      0
        end
```

[Assemble program.]

```
A>asm86 sort

CP/M 8086 ASSEMBLER VER 1.1
END OF PASS 1
END OF PASS 2
END OF ASSEMBLY. NUMBER OF ERRORS:  0
```

[Type listing file generated by ASM-86.]

```
A>type sort.lst
```

```
                        ;
                        ;       simple sort program
                        ;
                        sort:
0000  BE0000                    mov  si,0               ;initialize index
0003  BB0001                    mov  bx,offset nlist    ;bx = base of list
0006  C606080100                mov  sw,0               ;clear switch flag
                        comp:
000B  8A00                      mov  al,[bx+si]         ;get byte from list
000D  3A4001                    cmp  al,1[bx+si]        ;compare with next byte
0010  760A                      jna  inci               ;don't switch if in order
0012  864001                    xchg al,1[bx+si]        ;do first part of switch
0015  8800                      mov  [bx+si],al         ;do second part
0017  C606080101                mov  sw,1               ;set switch flag
                        inci:
001C  46                        inc  si                 ;increment index
001D  83FE08                    cmp  si,count           ;end of list?
0020  75E9                      jnz  comp               ;no, keep going
0022  F606080101                test sw,1               ;done-any switches?
0027  75D7                      jnz  sort               ;yes, sort some more
                        done:
0029  E9FDFF                    jmp  done               ;get here when list ordered
                        ;
                                dseg
                                org  100h               ;leave space for base page
```

```
                        ;
0100  030804061F06      nlist   db    3,8,4,6,31,6,4,1
      0401
  0008                  count   equ   offset $-offset nlist
0108  00                sw      db    0
                              end

END OF ASSEMBLY. NUMBER OF ERRORS:  0
```

[Type symbol table file generated by ASM-86.]

```
A>type sort.sym
0000  VARIABLES
0100  NLIST         0108 SW

0000  NUMBERS
0008  COUNT

0000  LABELS
000B  COMP          0029 DONE       001C INCI           0000 SORT
```

[Type hex file generated by ASM-86.]

```
A>type sort.h86
:040000030000000F9
:1B000081BE0000BB0001C6060801008A003A4001760A8640018800C60608016C
:11001B81014683FE0875E9F60608010175D7E9FDFFEE
:09010082030804061F0604010035
```

```
:00000001FF
```

[Generate CMD file from .H86 file.]

```
A>gencmd sort

BYTES READ       0039
RECORDS WRITTEN 04
```

[Invoke DDT-86 and load SORT.CMD.]

```
A>ddt86 sort
DDT86 1.0
     START      END
CS 047D:0000  047D:002F
DS 0480:0000  0480:010F
```

[Display initial register values.]

```
-x
                AX     BX     CX     DX     SP     BP     SI     DI     CS     DS     SS     ES     IP
---------       0000   0000   0000   0000   119E   0000   0000   0000   047D   0480   0491   0480   0000
MOV     SI, 0000
```

[Disassemble the beginning of the code segment.]

```
                -l
047D:0000 MOV   SI,0000
047D:0003 MOV   BX,0100
047D:0006 MOV   BYTE [0108],00
047D:000B MOV   AL,[BX+SI]
047D:000D CMP   AL,01[BX+SI]
047D:0010 JBE   001C
047D:0012 XCHG  AL,01[BX+SI]
047D:0015 MOV   [BX+SI],AL
047D:0017 MOV   BYTE [0108],01
047D:001C INC   SI
047D:001D CMP   SI,0008
047D:0020 JNZ   000B
```

[Display the start of the data segment.]

```
-d100,10f
0480:0100       03   08   04   06   1F   06   04   01   00   00   00   00   00   00   00   00 ................
```

[Disassemble the rest of the code.]

```
                -l
047D:0022 TEST  BYTE [0108],01
047D:0027 JNZ   0000
047D:0029 JMP   0029
047D:002C ADD   [BX+SI],AL
```

```
047D:002E ADD     [BX+SI],AL
047D:0030 DAS
047D:0031 ADD     [BX+SI],AL
047D:0033 ??=     6C
047D:0034 POP     ES
047D:0035 ADD     [BX],CL
047D:0037 ADD     [BX+SI],AX
047D:0039 ??=     6F
```

[Execute program from IP (=0) setting breakpoint at 29H.]

```
-g,29
```

[Breakpoint encountered.]

```
*047D:0029
```

[Display sorted list.]

```
-d100,10f
0480:0100      00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   ................
```

[Doesn't look good; reload file.]

```
-esort
     START      END
CS 047D:0000  047D:002F
```

```
DS 0480:0000  0480:010F
```

[Trace 3 instructions.]

```
-t3
                AX   BX   CX   DX   SP   BP   SI   DI   IP
-----Z-P-       0000 0100 0000 0000 119E 0000 0008 0000 0000 MOV SI,0000
-----Z-P-       0000 0100 0000 0000 119E 0000 0000 0000 0003 MOV BX,0100
---- Z-P-       0000 0100 0000 0000 119E 0000 0000 0000 0006 MOV BYTE [0108],00
*047D:000B
```

[Trace some more.]

```
-t3
                AX   BX   CX   DX   SP   BP   SI   DI   IP
-----Z-P-       0000 0100 0000 0000 119E 0000 0000 0000 000B MOV AL,[BX+SI]
-----Z-P-       0003 0100 0000 0000 119E 0000 0000 0000 000D CMP AL,01[BX+SI]
----S-A-C       0003 0100 0000 0000 119E 0000 0000 0000 0010 JBE 001C
*047D:001C
```

[Display unsorted list.]

```
-d100,10f
0480:0100       03   08   04   06   1F   06   04   01   00   00   00   00   00   00   00   00 ................
```

[Display next instructions to be executed.]

```
-I
047D:001C INC      SI
047D:001D CMP      SI,0008
047D:0020 JNZ      000B
047D:0022 TEST     BYTE [0108],01
047D:0027 JNZ      0000
047D:0029 JMP      0029
047D:002C ADD      [BX+SI],AL
047D:002E ADD      [BX+SI],AL
047D:0030 DAS
047D:0031 ADD      [BX+SI],AL
047D:0033 ??=      6C
047D:0034 POP      ES
```

[Trace some more.]

```
-t3
              AX   BX   CX   DX   SP   BP   SI   DI   IP
----S-A-C     0003 0100 0000 0000 119E 0000 0000 0000 001C  INC SI
--------C     0003 0100 0000 0000 119E 0000 0001 0000 001D CMP SI,0008
----S-APC     0003 0100 0000 0000 119E 0000 0001 0000 0020  JNZ 000B
*047D:000B
```

[Display instructions from current IP.]

```
-I
047D:000B MOV      AL,[BX+SI]
```

```
047D:000D CMP     AL,01[BX+SI]
047D:0010 JBE     001C
047D:0012 XCHG    AL,01[BX+SI]
047D:0015 MOV     [BX+SI],AL
047D:0017 MOV     BYTE [0108],01
047D:001C INC     SI
047D:001D CMP     SI,0008
047D:0020 JNZ     000B
047D:0022 TEST    BYTE [0108],01
047D:0027 JNZ     0000
047D:0029 JMP     0029

-t3
              AX   BX   CX   DX   SP   BP   SI   DI   IP
----S-APC     0003 0100 0000 0000 119E 0000 0001 0000 000B MOV AL,[BX+SI]
----S-APC     0008 0100 0000 0000 119E 0000 0001 0000 000D CMP AL,01[BX+SI]
---------     0008 0100 0000 0000 119E 0000 0001 0000 0010 JBE 001C
*047D:0012

-l
047D:0012 XCHG    AL,01[BX+SI]
047D:0015 MOV     [BX+SI],AL
047D:0017 MOV     BYTE [0108],01
047D:001C INC     SI
047D:001D CMP     SI,0008
047D:0020 JNZ     000B
047D:0022 TEST    BYTE [0108],01
```

```
047D:0027 JNZ     0000
047D:0029 JMP     0029
047D:002C ADD     [BX+SI],AL
047D:002E ADD     [BX+SI],AL
047D:0030 DAS
```

[Go until switch has been performed.]

```
-g,20
*047D:0020
```

[Display list.]

```
-d100,10f
0480:0100      03   04   08   06   1F   06   04   01   01   00   00   00   00   00   00   00 ................
```

[Looks like 4 and 8 were switched okay. (And toggle is true.)]

```
-t
              AX   BX   CX   DX   SP   BP   SI   DI   IP
----S-APC     0004 0100 0000 0000 119E 0000 0002 0000 0020 JNZ      000B
*047D:000B
```

[Display next instructions.]

```
-l
047D:000B MOV     AL,[BX+SI]
```

```
047D:000D CMP     AL,01[BX+SI]
047D:0010 JBE     001C
047D:0012 XCHG    AL,01[BX+SI]
047D:0015 MOV     [BX+SI],AL
047D:0017 MOV     BYTE [0108],01
047D:001C INC     SI
047D:001D CMP     SI,0008
047D:0020 JNZ     000B
047D:0022 TEST    BYTE [0108],01
047D:0027 JNZ     0000
047D:0029 JMP     0029
```

[Since switch worked, let's reload and check boundary conditions.]

```
-esort
     START      END
CS 047D:0000  047D:002F
DS 0480:0000  0480:010F
```

[Make it quicker by setting list length to 3. (Could also have used s47d:1e = 3 to patch.)]

```
-a1d
047D:001D cmp si,3
```

[Display unsorted list.]

```
-d100
0480:0100   03  08  04  06  1F  06  04  01  00  00  00  00  00  00  00  00 ................
0480:0110   00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00 ................
0480:0120   00  00  00  00  00  00  00  00  00  00  00  00  00  00  20  20  20 .............
```

[Set breakpoint when first 3 elements of list should be sorted.]

```
-g,29
*047D:0029
```

[See if list is sorted.]

```
-d100,10f
0480:0100   03  04  06  08  1F  06  04  01  00  00  00  00  00  00  00  00 ................
```

[Interesting, the fourth element seems to have been sorted in.]

```
-esort
      START       END
CS 047D:0000  047D:002F
DS 0480:0000  0480:010F
```

[Let's try again with some tracing.]

```
-a1d
047D:001D cmp si,3
047D:0020 .
```

```
-t9
                AX   BX   CX   DX   SP   BP   SI   DI   IP
-----Z-P-       0006 0100 0000 0000 119E 0000 0003 0000 0000 MOV SI,0000
-----Z-P-       0006 0100 0000 0000 119E 0000 0000 0000 0003 MOV BX,0100
-----Z-P-       0006 0100 0000 0000 119E 0000 0000 0000 0006 MOV BYTE [0108],00
-----Z-P-       0006 0100 0000 0000 119E 0000 0000 0000 000B MOV AL,[BX+SI]
-----Z-P-       0003 0100 0000 0000 119E 0000 0000 0000 000D CMP AL,01[BX+SI]
----S-A-C       0003 0100 0000 0000 119E 0000 0000 0000 0010 JBE 001C
----S-A-C       0003 0100 0000 0000 119E 0000 0000 0000 001C INC SI
--------C       0003 0100 0000 0000 119E 0000 0001 0000 001D CMP SI,0003
----S-A-C       0003 0100 0000 0000 119E 0000 0001 0000 0020 JNZ 000B
*047D:000B

-l
047D:000B MOV     AL,[BX+SI]
047D:000D CMP     AL,01[BX+SI]
047D:0010 JBE     001C
047D:0012 XCHG    AL,01[BX+SI]
047D:0015 MOV     [BX+SI],AL
047D:0017 MOV     BYTE [0108],01
047D:001C INC     SI
047D:001D CMP     SI,0003
047D:0020 JNZ     000B
047D:0022 TEST    BYTE [0108],01
047D:0027 JNZ     0000
047D:0029 JMP     0029
```

```
-t3
                AX   BX   CX   DX   SP   BP   SI   DI   IP
----S-A-C       0003 0100 0000 0000 119E 0000 0001 0000 000B MOV AL,[BX+SI]
----S-A-C       0008 0100 0000 0000 119E 0000 0001 0000 000D CMP AL,01[BX+SI]
---------       0008 0100 0000 0000 119E 0000 0001 0000 0010 JBE 001C
*047D:0012

-I
047D:0012 XCHG    AL,01[BX+SI]
047D:0015 MOV     [BX+SI[],AL
047D:0017 MOV     BYTE [0108],01
047D:001C INC     SI
047D:001D CMP     SI,0003
047D:0020 JNZ     000B
047D:0022 TEST    BYTE [0108],01

-t3
                AX   BX   CX   DX   SP   BP   SI   DI   IP
---------       0008 0100 0000 0000 119E 0000 0001 0000 0012 XCHGAL,01[BX+SI]
---------       0004 0100 0000 0000 119E 0000 0001 0000 0015 MOV [BX+SI],AL
---------       0004 0100 0000 0000 119E 0000 0001 0000 0017 MOV BYTE [0108],01
*047D:001C

-d100,10f
0480:0100       03   04   08   06   1F   06   04   01   01   00   00   00   00   00   00   00 ................
```

[So far, so good.]

```
-t3
                AX   BX   CX   DX   SP   BP   SI   DI   IP
---------       0004 0100 0000 0000 119E 0000 0001 0000 001C  INC SI
---------       0004 0100 0000 0000 119E 0000 0002 0000 001D  CMP SI,0003
----S-APC       0004 0100 0000 0000 119E 0000 0002 0000 0020  JNZ 000B
*047D:000B

-l
047D:000B MOV     AL,[BX+SI]
047D:000D CMP     AL,01[BX+SI]
047D:0010 JBE     001C
047D:0012 XCHG    AL,01[BX+SI]
047D:0015 MOV     [BX+SI],AL
047D:0017 MOV     BYTE [0108],01
047D:001C INC     SI
047D:001D CMP     SI,0003
047D:0020 JNZ     000B
047D:0022 TEST    BYTE [0108],01
047D:0027 JNZ     0000
047D:0029 JMP     0029

-t3
                AX   BX   CX   DX   SP   BP   SI   DI   IP
----S-APC       0004 0100 0000 0000 119E 0000 0002 0000 000B  MOV AL,[BX+SI]
----S-APC       0008 0100 0000 0000 119E 0000 0002 0000 000D  CMP AL,01[BX+SI]
---------       0008 0100 0000 0000 119E 0000 0002 0000 0010  JBE 001C
*047D:0012
```

[Sure enough, it's comparing the third and fourth elements of the list. Reload program.]

```
-esort
      START             END
CS 047D:0000        047D:002F
DS 0480:0000        0480:010F

-l
047D:0000 MOV       SI,0000
047D:0003 MOV       BX,0100
047D:0006 MOV       BYTE [0108],00
047D:000B MOV       AL,[BX+SI]
047D:000D CMP       AL,01[BX+SI]
047D:0010 JBE       001C
047D:0012 XCHG      AL,01[BX+SI]
047D:0015 MOV       [BX+SI],AL
047D:0017 MOV       BYTE [0108],01
047D:001C INC       SI
047D:001D CMP       SI,0008
047D:0020 JNZ       000B
```

[Patch length.]

```
-a1d
047D:001D           cmp si,7
047D:0020
```

[Try it out.]

```
-g,29
*047D:0029
```

[See if list is sorted.]

```
-d100,10f
0480:0100      01  03  04  04  06  06  08  1F  00  00  00  00  00  00  00  00 ................
```

[Looks better; let's install patch in disk file. To do this, we must read CMD file including header, so we use R command.]

```
-rsort.cmd
  START       END
2000:0000   2000:01FF
```

[First 80h bytes contain header, so code starts at 80h.]

```
-l80
2000:0080 MOV     SI,0000
2000:0083 MOV     BX,0100
2000:0086 MOV     BYTE [0108],00
2000:008B MOV     AL,[BX+SI]
2000:008D CMP     AL,01[BX+SI]
2000:0090 JBE     009C
2000:0092 XCHG    AL,01[BX+SI]
```

```
2000:0095 MOV     [BX+SI],AL
2000:0097 MOV     BYTE [0108],01
2000:009C INC     SI
2000:009D CMP     SI,0008
2000:00A0 JNZ     008B
```

[Install patch.]

```
-a9d
2000:009D         cmp si,7
```

[Write file back to disk. (Length of file assumed to be unchanged since no length specified.)]

```
-wsort.cmd
```

[Reload file.]

```
-esort

     START      END
CS 047D:0000  047D:002F
DS 0480:0000  0480:010F
```

[Verify that patch was installed.]

```
-l
047D:0000 MOV     SI,0000
047D:0003 MOV     BX,0100
```

```
047D:0006 MOV     BYTE [0108],00
047D:000B MOV     AL,[BX+SI]
047D:000D CMP     AL,01[BX+SI]
047D:0010 JBE     001C
047D:0012 XCHG    AL,01[BX+SI]
047D:0015 MOV     [BX+SI],AL
047D:0017 MOV     BYTE [0108],01
047D:001C INC     SI
047D:001D CMP     SI,0007
047D:0020 JNZ     000B
```

[Run it.]

```
-g,29
*047D:0029
```

[Still looks good. Ship it!]

```
-d100,10f
0480:0100      01   03   04   04   06   06   08   1F   00   00   00   00   00   00   00   00 ................
-^C
A>
```

APPENDIX A. MESSAGES

Contents

Appendix A. Messages

CP/M-86 communicates with you through messages displayed on your screen. CP/M-86 keeps you informed of the date, time, and status of your IBM Personal Computer by maintaining a "Status Line" on the last line of your screen. CP/M-86 responds to a mistyped command or other error by displaying a message directly beneath the command line that contained or caused the error. This Appendix describes Status Line Messages and CP/M-86 Error Messages.

Status Line Messages

CP/M-86 maintains a Status Line at the bottom of your screen. Normally the Status Line looks like this:

[|U = 10| mm/dd/yy | 05:42:27 |]

The part of the screen to the left of the Status Line display is generally blank. The second field of the Status Line shows the current user number. In the example above, the user number is 10. When you first bring CP/M-86 into memory, the second field tells the date your CP/M-86 system was created and the third field shows the time elapsed since CP/M-86 was initiated. You can set either of these fields to the current date and time with a TOD command.

Sometimes CP/M-86 replaces the normal messages in the Status Line with information about light pen input, diskette/drive errors and printer errors.

When a program requires that you make a selection from the screen with the light pen, the Status Line displays the message, "Waiting for Light Pen Input" until you depress the tip of the pen against the screen's surface. When the program receives your input, the Status Line returns to its normal state.

Diskette/Drive Error Status Line Messages

If CP/M-86 detects an error while operating on a diskette or drive, it retries the operation five times before sending a diskette/drive error message to the Status Line and ringing the bell. A diskette/drive error message has the following format:

Disk d: message R/I/C/D?

In an actual error message, CP/M-86 replaces d: with the drive specification of the drive where the error occurred, and message with one of the messages listed below. When you receive this error message, CP/M-86 asks you to type a character and thereby select one of four options. The character you type is not echoed at the screen. To retry the operation that caused the error, type R. To ignore the error, type I. To cancel and return to the operating system, type C. To display details of the error, type D. The detail message, which appears in the Status Line if you press D, has the following format:

Operation Trk 00 Sec 00 R/I/C/D?

In an actual detail message, CP/M-86 replaces Operation with the name of the operation it was performing when the error took place, and also inserts the track and sector numbers where the error occurred. The following list defines the operation names that can appear in a detail message.

Status	error occurred while trying to obtain diskette status
Read	error occurred while trying to read from diskette
Write	error occurred while trying to write to diskette
Verify	error occurred while comparing data on diskette with data stored in memory
Unknown Op	error occurred during an unknown operation

The following messages can appear in the diskette/drive error Status Line. These messages can indicate problems with hardware. If the error persists contact your point of sale.

ADDRESS MARK MISSING

Diskette is worn or improperly formatted. Reformat diskette with NEWDISK or retry with new diskette.

BAD DISK COMMAND

This could be a hardware problem, or could indicate that the part of memory in which CP/M-86 resides has been violated. Try starting CP/M-86 from another system disk.

CONTROLLER FAILED

This is a hardware problem; retry.

DATA ERROR

Check detail message. If the error occurred during a read operation, you could ignore the error and read what you can from the disk. If the error occurred during a write operation, you should retry with a fresh diskette.

DMA ADDRESS ERROR

A program has instructed the DMA chip to read data into a memory area that straddles a 64K boundary. Take careful note of the situation that caused the error. If the running program was an application program you have written then you must correct your error in attempting to instruct the DMA chip to read data into a memory area that straddles a 64K boundary. If a standard utility program was executing at the time of the error, it may indicate a hardware problem. Retry the operation.

DMA CHIP FAILURE

This is a hardware problem; retry the operation.

FAILED TO RESPOND

There is no diskette in the drive, the diskette is improperly inserted, or the drive latch is not closed. Correct the problem and press R for retry.

SECTOR NOT FOUND

Diskette is worn or improperly formatted. Reformat the diskette with NEWDISK or try a new diskette.

SEEK FAILED

This could be a hardware problem or a symptom of a worn diskette. Retry the operation with a new diskette.

WRITE-PROTECTED

Diskette has a foil tab over the write-protect notch. Check that you have the correct diskette. If you want to write on the diskette, remove the foil tab and retry.

Printer Error Status Line Messages

If you have an IBM 80 cps Matrix Printer added to your IBM Personal Computer, CP/M-86 displays printer error messages in the Status Line in the format shown below:

BUSY

Check cabling and retry.

NOT ONLINE

Check that the green light next to Online is illuminated. If not, press Online switch and retry. If the light is illuminated, check cabling and retry.

INPUT/OUTPUT ERROR

There is no power to the printer. Apply power to the printer and retry. Check cabling.

OUT OF PAPER

Your printer has a tiny switch that is depressed when continuous-feed paper is threaded through the printer correctly. This error message appears when the printer is out of paper or the paper is incorrectly threaded. Correct the problem and retry.

PRINTER #: MESSAGE R/I/C?

In an actual printer error message, CP/M-86 replaces # with the printer number (0, 1 or 2), and message with one of the messages defined below.

STATUS = 0000 0000

This message shows the printer error status byte in binary as two groups of four digits. The Status Code is displayed for diagnostic purposes; note the code value and the circumstances that caused the error.

CP/M-86 Command Error Messages

CP/M-86 command error messages can occur when you type a program name at the console. The program can be one supplied with the CP/M-86 system diskette or it can be from other software designed to run under the CP/M-86 system. Of course, this list does not include all the error messages that can occur when running other software, but it is a good idea to check the list if you think the error message might have originated from CP/M-86. If CP/M-86 cannot find the program you typed, it simply repeats what you typed followed by a question mark. You can easily tell if you made a typing error. If not, you might have the wrong diskette inserted in the drive.

Command error messages are not displayed on the Status Line. They are printed on the line below the current cursor position. In this Appendix the error messages are organized alphabetically.

COMMAND NAME?

If CP/M-86 cannot find the command you specified, it returns the command name you entered followed by a question mark. Check that you have typed the command name correctly. Check that the command you requested exists as a .CMD file with the current user number on the default or specified diskette.

GENCMD. An invalid GENCMD command line was entered. If the command was mistyped then try again; otherwise, review GENCMD operation (Appendix C).

ALL DATA WILL BE ERASED FROM THE DISK. IS THIS WHAT YOU WANT? Y/N

NEWDISK. This message is displayed by NEWDISK to verify that the operator understands that the program erases all the data from the disk in the process of formatting it.

AMBIGUOUS OPERAND

DDT-86. An attempt was made to assemble a command with an ambiguous operand. Precede the operand with the prefix "BYTE" or "WORD".

ATTRIBUTE INCORRECTLY SPECIFIED

SPEED. You have made an error in specifying an attribute. Refer to the description of the SPEED command to correct your error.

BAD DIRECTORY ON D:
SPACE ALLOCATION CONFLICT

STAT. This message is followed by a list of one or more filenames. The files listed contain data blocks that are already allocated to another file on the diskette. The error can be caused by a hardware or software failure. The error can be corrected by erasing the file(s) displayed and rebooting CP/M-86. Note that if CP/M-86 is not rebooted after erasing the files, the problem will reappear.

BAD PARAMETER

PIP. An illegal parameter has been entered in a PIP command. Retype the entry correctly.

BDOS ERR ON d:

CP/M-86 replaces d: with the drive specification of the drive where the error occurred.

BDOS ERR ON d: BAD SECTOR

This could indicate a hardware problem or a worn or improperly formatted diskette. Press Ctrl-C to terminate the program and return to CP/M-86, or press the Enter key to ignore the error.

BDOS ERR ON d: FILE R/O

An erase, rename or set file attributes operation was attempted on a Read-Only file. The file should first be set to Read-Write (R/W) with the command: "STAT filespec $R/W".

BDOS ERR ON d: SELECT

CP/M-86 has received a request specifying a non-existent drive, or diskette in drive is improperly formatted. CP/M-86 terminates the current program as soon as you press any key.

BDOS ERR ON d: R/O

Drive has been assigned Read-Only status with a STAT command, or the diskette in the drive has been changed without being initialized with a Ctrl-C. CP/M-86 terminates the current program as soon as you press any key.

BREAK "x" AT c

ED. "x" is one of the symbols described below and c is the command letter being executed when the error occurred.

#	Search failure. ED cannot find the string specified in an F, S, or N command.
?	Unrecognized command letter c. ED does not recognize the indicated command letter, or an E, H, Q, or O command is not alone on its command line.
O	The file specified in an R command could not be found.
>	Buffer full. ED cannot put any more characters in the memory buffer, or the string specified in an F, N, or S command is too long.
E	Command terminated. A keystroke at the console terminated command execution.
F	Disk or directory full. This error is followed by either the disk or directory full message. Refer to the recovery procedures listed under these messages.

CANNOT CLOSE
CANNOT CLOSE FILE

DDT-86. The disk file written by a W command cannot be closed. This is a fatal error that terminates DDT-86 execution. The user should take appropriate action after checking to see if the correct diskette is in the drive and that the diskette is not write-protected.

GENCMD. The CMD file written by GENCMD cannot be closed. This is a fatal error that terminates GENCMD execution. The user should take appropriate action after checking to see if the correct diskette is in the drive and that the diskette is not write-protected.

CANNOT CLOSE, READ/ONLY?

SUBMIT. The $$$.SUB file could not be closed. Check to see if the correct system diskette is in the "A" drive and that the diskette is not write-protected. The SUBMIT job can be restarted after rebooting CP/M-86.

CANNOT OPEN SOURCE

GENCMD. The hex file specified in the GENCMD command line could not be found. The hex file must have the filetype ".H86". Check to see that the correct diskette was specified and try again.

CHECKSUM ERROR

GENCMD. A hex record checksum error was encountered. The hex record that produced the error must be corrected, probably by recreating the hex file.

CLOSE FILE - {filespec}

PIP. An output file cannot be closed. The user should take appropriate action after checking to see if the correct diskette is in the drive and that the diskette is not write-protected.

COMMAND BUFFER OVERFLOW

SUBMIT. The SUBMIT buffer allows up to 2048 characters in the input file.

COMMAND TOO LONG

SUBMIT. A command in the SUBMIT file cannot exceed 125 characters.

DEFECTIVE DISKETTE

NEWDISK. The diskette could not be formatted. The diskette should be discarded and a new one tried. If the problem persists, the diskettes may be the wrong type or the drive may need servicing.

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

PIP. The destination file specified in a PIP command already exists and it is Read-Only. If you type "Y" the destination file will be deleted before the file copy is done.

DEVICE IS NOT ON-LINE

ASSIGN, PROTOCOL & SPEED. The ASSIGN, PROTOCOL or SPEED command that you have entered specifies a device that is not currently on-line on your computer. If you do have the device on-line, the message can indicate that the device has failed. If the error persists, you should contact your point of sale.

DIRECTORY FULL

ED. There is not enough directory space for the file being written. You can use the "0Xfilespec" command to erase any unnecessary files on the diskette without leaving the editor. Alternatively, you can save the contents of the memory buffer on another diskette with the command "B#Xfilespec", where filespec is a file on a different drive. You can then quit the edit. If you reedit the file the output should be placed on a different drive with the command "ED filespec d:", where d: is a valid drive name other than the drive containing the source file. You can read the file saved with the "Rfilespec" command. Caution: Part of the file may not be in the memory buffer when you save it (if you have not appended the whole file or if you have issued any "W" commands).

SUBMIT. There is not enough directory space to write the $$$.SUB file used for processing SUBMITS. Erase some files or select a new disk, and retry.

DISK DRIVE DOES NOT EXIST ON THIS SYSTEM

NEWDISK. You have specified a drive which does not exist on your system. Try again with a correct drive specification.

DISK FORMAT IN PROGRESS

NEWDISK. The NEWDISK program is currently formatting a diskette. Wait until it is complete before taking any further action.

DISK FULL

ED. There is not enough disk space for the output file. This error may occur on the W, E, H, or X commands. If it occurs with the X command you can repeat the command prefixing the filename with a different drive. Otherwise you can try the recovery methods described under the Directory Full error.

DISK READ ERROR
DISK READ - {filespec}

DDT-86. The disk file specified in an R command could not be read properly. This is usually the result of an unexpected end of file. Correct the problem in your file.

GENCMD. The specified hex file could not be read properly. This is usually the result of an unexpected end of file. Correct the problem by regenerating the H86 file.

PIP. The input disk file specified in a PIP command could not be read properly. This is usually the result of an unexpected end of file. Correct the problem in your file.

DISK WRITE ERROR
DISK WRITE - {filespec}

DDT-86. A disk write operation could not be successfully performed during a W command, probably due to a full disk. You should either erase some unnecessary files or get another diskette with more room.

PIP. A disk write operation could not be successfully performed during a PIP command, probably due to a full disk. You should either erase some unnecessary files or get another diskette with more room and run PIP again.

SUBMIT. The SUBMIT program could not write the $$$.SUB file to the disk. Erase some files or select a new disk; retry.

DIVISION BY ZERO TRAP
**** PROGRAM ABORTED ****

CP/M-86 prints this message when the user program executes a DIV or IDIV instruction and the CPU generates a divide error interrupt.

ERROR: error message

Refer to the error message following the word "ERROR:".

PIP. All of the messages generated by the PIP program are displayed in the format shown above.

ERROR ON LINE nnn MESSAGE

SUBMIT. The SUBMIT program displays its messages in the format shown above, where nnn represents the line number of the SUBMIT file. Refer to the message following the line number.

ERROR READING HELP.HLP INDEX

HELP. The HELP.HLP file used by the HELP command is invalid. This may be caused by an unexpected end of file. The distributed HELP.HLP file should be copied to the diskette from the CP/M-86 system diskette.

FILE EXISTS

REN. You have tried to rename a file to a name already assigned to another file. Either delete the existing file or rename the file.

FILE IS READ/ONLY

ED. A Read-Only file cannot be edited with the ED command "ED filespec". The command "ED inputfilespec outputfilespec" should be used instead.

FILE NOT FOUND
FILE NOT FOUND - {filespec}

ED. ED could not find the specified file. Check that you have entered the correct drive specification or that you have the correct diskette in the drive.

PIP. An input file which you have specified does not exist.

STAT. STAT could not find the specified file; check to see if the correct diskette is in the drive.

FILENAME REQUIRED

ED. The ED command was typed without a filename. Reenter the ED command followed by the name of the file you wish to edit or create.

HEX RECORD CHECKSUM - {filespec}

PIP. A hex record checksum error was encountered during the transfer of a hex file. The hex file with the checksum error should be corrected, probably by recreating the hex file.

HELP.HLP READ ERROR

HELP. An error occurred in reading the HELP.HLP help file. Usually this error is caused by an unexpected end of file. A new copy of the HELP.HLP file should be copied from the CP/M-86 system diskette.

ILLEGAL DISK NAME
ILLEGAL DISKETTE DRIVE

NEWDISK. The drive must be A:, B:, C: or D:. Reenter the command correctly.

COPYDISK. An invalid or non-existent drive was specified.

ILLEGAL TYPE COMBINATION

ASSIGN. You have probably specified an output device as an input device, or an input device as an output device. Re-enter the ASSIGN command correctly.

ILLEGAL TYPE SPECIFIED

NEWDISK. Only type specifications of $S, $N, $DS or $DN are allowed.

INCORRECT LOGICAL DEVICE SPECIFICATION

ASSIGN. You have specified a logical device that is incorrect.

INCORRECT PHYSICAL DEVICE SPECIFICATION

ASSIGN. You have specified a physical device that is incorrect.

INCORRECT TYPE DELIMITER

NEWDISK. A "$" must be used to delimit the type ($S, $N, $DS, or $DN). An example of a valid command is "NEWDISK B: $S".

INPUT CANNOT BE ASSIGNED TO MORE THAN ONE DEVICE AT A TIME

ASSIGN. Specify only one device for input.

IS THIS WHAT YOU WANT TO DO (Y/N)?

COPYDISK. If the displayed COPYDISK function is what you want performed, then type "Y".

INSUFFICIENT MEMORY

DDT-86. There is not enough memory to load the file specified in an R or E command.

INSUFFICIENT MEMORY AVAILABLE FOR COPY

COPYDISK. There is not enough memory available to copy a track from the specified diskette.

INSUFFICIENT MEMORY TO CREATE CMD FILE

GENCMD. There is not enough memory to create a CMD file from the H86 file specified.

INVALID ASSIGNMENT

STAT. An invalid drive or file assignment was attempted. This error message may be followed by a list of the valid file assignments which can follow a filename. If an invalid drive assignment was attempted, the message "Use: d: = RO" is displayed showing the proper syntax for drive assignments.

INVALID CONTROL CHARACTER

SUBMIT. The only valid control characters in the SUBMIT files of type SUB are ^A through ^Z. Note that in a SUBMIT file, the control character is represented by typing the circumflex, ^, not by depressing the Control Key.

INVALID DESTINATION

PIP. The destination specified in your PIP command is illegal. You have probably specified an input device as a destination.

INVALID FORMAT

PIP. The format of your PIP command is illegal. See the description of the PIP command.

INVALID HEX DIGIT
INVALID HEX DIGIT - {filespec}

GENCMD & PIP. An invalid hex digit has been encountered while reading a hex file. The hex file with the invalid hex digit should be corrected, probably by recreating the hex file.

INVALID SEPARATOR

PIP. You have placed an invalid character for a separator between two input filenames.

INVALID SOURCE

PIP. The source specified in your PIP command is illegal. You have probably specified an output device as a source.

INVALID USER NUMBER

PIP. You have specified a user number greater than 15. User numbers are in the range 0 to 15.

MEMORY REQUEST DENIED

DDT-86. A request for memory during an R command could not be fulfilled. Up to eight blocks of memory can be allocated at a given time.

MEMORY NOT AVAILABLE

CP/M-86. There is not enough memory available for loading the program specified.

MESSAGE LENGTH INCORRECTLY SPECIFIED

PROTOCOL. Enter a correct message length.

NO DEVICES SPECIFIED

ASSIGN. Devices must be specified in an ASSIGN command.

NO DIRECTORY SPACE - {filespec}

PIP. There is not enough directory space for the output file. You should either erase some unnecessary files or get another diskette with more directory room and run PIP again.

NO DISK NAME OR TYPE SPECIFIED

NEWDISK. Enter a correct disk name or type. For example, to format a new double-sided system diskette on the B drive, you use the command "NEWDISK B: $DS".

NO FILE

DIR, ERA & REN. CP/M-86 could not find the specified file, or no files exist.

DDT-86. The file specified in an R or E command could not be found on the disk.

NO .HLP FILE ON THE DEFAULT DRIVE

HELP. The HELP.HLP file was not found on the default drive. It should be copied from the CP/M-86 system diskette.

NO MORE DIRECTORY SPACE

GENCMD. There is insufficient directory space for creating the output file. A new diskette should be selected or unnecessary files erased.

NO SERIAL PORT SPECIFIED

PROTOCOL. Enter a correct serial port.

SPEED. Enter a correct serial port.

NO SPACE

DDT-86. There is no space in the directory for the file being written by a W command.

NO SUB FILE PRESENT

SUBMIT. For SUBMIT to operate properly, you must create a file with a filetype of SUB. The SUB file contains normal CP/M-86 commands. Use one command per line.

NO TYPE SPECIFIED

NEWDISK. Enter a NEWDISK command with a type specified. Valid types are system ($S or $DS), or normal ($N or $DN). System means that the diskette is formatted and the CP/M-86 system is copied from the system diskette. Normal means that the diskette is formatted for data only.

NOT A PROGRAMMABLE KEY – TRY AGAIN

FUNCTION. If you are trying to set up a programmable key, you should try again. If you are trying to exit the FUNCTION program by entering a carriage return, you should press Ctrl-Break.

OUTPUT FILE EXISTS, ERASE IT

ED. The destination filename already exists when you are placing the destination file on a different diskette than the source. It should be erased or another diskette selected to receive the output file.

PARAMETER ERROR

SUBMIT. Within the SUBMIT file of type SUB, valid parameters are $0 through $9.

PERMANENT ERROR ON TRACK n
PERMANENT ERROR, SECTOR n

COPYDISK. n is the track or sector number. A bad sector exists on the source diskette if the error occurred during a track read. Otherwise, the bad sector is on the destination diskette. If the destination diskette has the error it can be reformatted with NEWDISK; if the error persists it should be discarded.

** PROGRAM ABORTED **

The program has been terminated due to one of the following conditions: a Ctrl-Break has been entered, the C option has been selected after a disk error message, or a division by zero trap has occurred.

PROTOCOL INCORRECTLY SPECIFIED

PROTOCOL. Your PROTOCOL command is incorrect; refer to the description of the PROTOCOL command.

QUIT NOT FOUND

PIP. The string argument to a Q parameter was not found in your input file.

READ ERROR

TYPE. An error occurred when reading the file specified in the TYPE command. Check the diskette and try again. The "STAT filespec" command can be helpful in diagnosing trouble.

SERIAL PORT INCORRECTLY SPECIFIED

PROTOCOL. Enter a correct serial port.

SPEED. Enter a correct serial port.

SOURCE AND DESTINATION CANNOT BE THE SAME

COPYDISK. The source and destination drives must be different, although drives A, B, C and D can all be mapped to the same physical drive. The system will prompt for you to change diskettes when the drive changes.

SOURCE AND DESTINATION DISKS MUST BE THE SAME TYPE

COPYDISK. Both the source and destination diskettes must have the same characteristics. (The "STAT DSK:" command displays the disk characteristics.)

START NOT FOUND

PIP. The string argument to an S parameter could not be found in the source file.

** SUBMIT FILE ABORTED **

The SUBMIT program has been terminated due to one of the following conditions: a Ctrl-Break has been entered, the C option has been selected after a disk error message, or a division by zero trap has occurred.

TOO MANY FILES
TOO MANY ENTRIES IN INDEX TABLE

HELP. There is not enough memory available to run the HELP utility.

STAT. There is not enough memory for STAT to sort the files specified or more than 512 files were specified.

TOPIC NOT FOUND

HELP. The topic requested does not exist in the HELP.HLP file. A topic should be selected from the menu displayed.

UNABLE TO FIND FILE HELP.HLP

HELP. The HELP.HLP file could not be found on the default drive. Copy it to the default drive from the CP/M-86 system diskette.

UNEXPECTED END OF HEX FILE - {filespec}

PIP. An end of file was encountered prior to a termination hex record. The hex file without a termination record should be corrected, probably by recreating the hex file.

UNKNOWN ID

FUNCTION. The internal code for this key is not known. This key cannot be programmed.

USER ABORTED

PIP. The user has terminated a PIP operation by pressing a key.

VERIFY - {filespec}

PIP. When copying with the V option, PIP found a difference when rereading the data just written and comparing it to the data in its memory buffer. Usually this indicates a failure of either the destination diskette or drive.

VERIFY ERROR AT s:o

DDT-86. The value placed in memory by a Fill, Set, Move, or Assemble command could not be read back correctly, indicating bad user memory or attempting to write to ROM or non-existent memory at the indicated location.

APPENDIX B. COMMAND SETUP AND EXECUTION UNDER CP/M-86

Contents

Command Setup and Execution Under CP/M-86

The 8086 microprocessor uses segment registers to reference memory. CP/M-86 provides three different memory organizations in which programs may execute. Each memory organization is called a memory model. CP/M-86 uses a 256-byte area of memory as a base page to store information about the program being executed. This area is also used by a program to communicate with CP/M-86.

Transient Program Execution Models

The initial values of the segment registers are determined by one of three "memory models" used by the transient program, and described in the CMD file header. The three memory models are summarized in the table below.

CP/M-86 Memory Models

Model	*Group Relationships*
8080 Model	Code and Data Groups Overlap
Small Model	Independent Code and Data Groups
Compact Model	Three or More Independent Groups

The 8080 Model supports programs which are directly translated from CP/M-80 when code and data areas are intermixed. The 8080 Model consists of one group which contains all the code, data, and stack areas. Segment registers are initialized to the starting address of the region containing this group. The segment registers can, however, be managed by the application program during execution so that multiple segments within the group can be addressed.

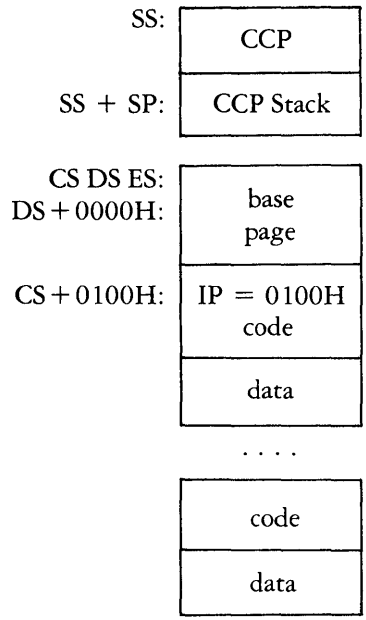
The Small Model is similar to that defined by Intel, where the program consists of an independent code group and a data group. The Small Model is suitable for use by programs where code and data is easily separated. Note again that the code and data groups often consist of, but are not restricted to, 64K byte segments.

The Compact Model occurs when any of the extra, stack, or auxiliary groups are present in programs. Each group may consist of one or more segments, but if any group exceeds 64K in size, or if auxiliary groups are present, then the application program must manage its own segment registers during execution in order to address all code and data areas.

The three models differ primarily in the manner in which segment registers are initialized upon transient program loading. The operating system program load function determines the memory model used by a transient program by examining the program group usage, as described in the following sections.

The 8080 Memory Model

The simplest memory model is the 8080 model. The name is derived from the similarity to the 8080 microprocessor memory organization. It provides only 64K for all the program's code and data. In this case, the CS, DS, and ES registers are initialized to the beginning of the code group, while the SS and SP registers remain set to a 96-byte stack area in the CCP. The Instruction Pointer Register (IP) is set to 100H to allow base page values at the beginning of the code group. Following program load, the 8080 Model appears as shown in the figure below where low addresses are shown at the top of the diagram:



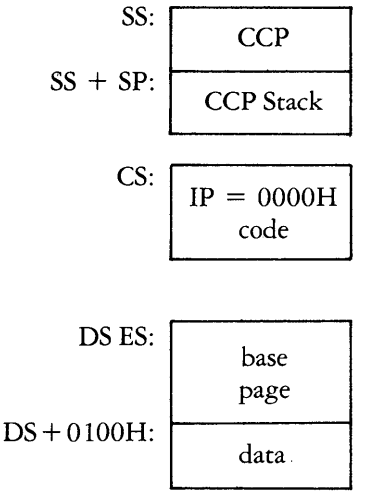
CP/M-86 8080 Memory Model

The intermixed code and data regions are indistinguishable. The "base page" values, described in the "Base Page Initialization" section, are similar to CP/M-80, allowing translation from 8080, 8085, or Z80 code into the 8086 and 8088 environment. The following ASM-86 example shows how to code an 8080 model transient program.

```
        cseg
        org     100h
        .
        .       (code)
endcs   equ     $
        dseg
        org     offset endcs
        .
        .       (data)
endds   equ     $
        cseg
        org     offset endds
        end
```

The Small Memory Model

The Small Model is assumed when the transient program contains both a code and data group. (In ASM-86, all code is generated following a CSEG directive, while data is defined following a DSEG directive with the origin of the data segment independent of the code segment.) In this model, CS is set to the beginning of the code group, the DS and ES are set to the start of the data group, and the SS and SP registers remain in the CCP's stack area as shown in the figure below.



CP/M-86 Small Memory Model

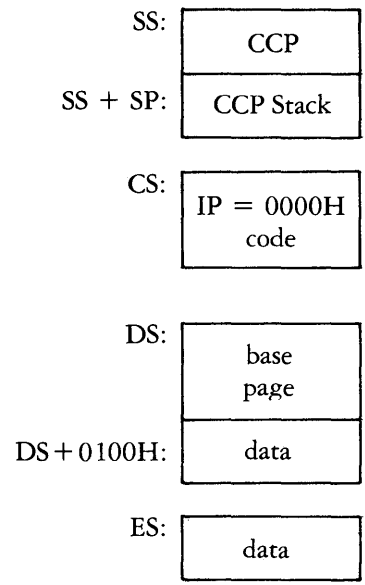
The machine code begins at CS + 0000H, the "base page" values begin at DS + 0000H, and the data area starts at DS + 0100H. The following ASM-86 example shows how to code a small model transient program.

```
cseg
.
.       (code)
dseg
org     100h
.
.       (data)
end
```

The Compact Memory Model

The Compact Model is assumed when code and data groups are present, along with one or more of the remaining stack, extra, or auxiliary groups. In this case, the CS, DS, and ES registers are set to the base addresses of their respective areas. The figure below shows the initial configuration of segment registers in the Compact Model. The values of the various segment registers can be programmatically changed during execution by loading from the initial values placed in base page by the CCP, thus allowing access to the entire memory space.

If the transient program intends to use the stack group as a stack area, the SS and SP registers must be set upon entry. The SS and SP registers remain in the CCP area, even if a stack group is defined. Although it may appear that the SS and SP registers should be set to address the stack group, there are two contradictions. First, the transient program may be using the stack group as a data area. In that case, the Far Call instruction used by the CCP to transfer control to the transient program could overwrite data in the stack area. Second, the SS register would logically be set to the base of the group, while the SP would be set to the offset of the end of the group. However, if the stack group exceeds 64K the address range from the base to the end of the group exceeds a 16-bit offset value.



CP/M-86 Compact Memory Model

The following ASM-86 example shows how to code a compact model transient program.

```
cseg
.
.           (code)
dseg
org         100h
.
.           (data)
eseg
.
.           (more data)
sseg
.
.           (stack area)
end
```

The following example shows how to initialize the stack segment register from the values contained in the base page.

```
                        Cseg
                Sst:
0000 9c                 Pushf
0001 5B                 Pop     Bx                      ;save flags in Bx
0002 FA                 Cli
0003 8E161500           Mov     Ss, SSpage
0007 BC4000             Mov     Sp, Offset Stackbase
000A 53                 Push    Bx
000B 9D                 Popf
                ;               .
                ;               .
                ;               .
                        Dseg
                        Org     015H                    ;Location of the stack
                                                        ;segment in the base page
0015            Sspage          Rw      1
0017                            Rs      100H - (15H + word)     ;Reserve this area for
                                                        ;base page values
0100 00         End_Data_Area   Db      0               ;For Gencmd to determine
                                                        ;length of group
                        Sseg
0000            StackSeg        Rs      40H             ;Stack Area
0040 00         Stackbase       Db      0               ;For Gencmd to determining
                                                        ;length of group
                End
```

END OF ASSEMBLY. NUMBER OF ERRORS: 0. USE FACTOR: 0%

Base Page Initialization

Similar to CP/M-80, the CP/M-86 base page contains default values and locations initialized by the CCP and used by the transient program. The base page occupies a region of user memory from offset 0000H through 00FFH relative to the DS register. The values stored in the base page are as follows:

DS + 0000:	LC0	LC1	LC2
DS + 0003:	BC0	BC1	M80
DS + 0006:	LD0	LD1	LD2
DS + 0009:	BD0	BD1	xxx
DS + 000C:	LE0	LE1	LE2
DS + 000F:	BE0	BE1	xxx
DS + 0012:	LS0	LS1	LS2
DS + 0015:	BS0	BS1	xxx
DS + 0018:	LX0	LX1	LX2
DS + 001B:	BX0	BX1	xxx
DS + 001E:	LX0	LX1	LX2
DS + 0021:	BX0	BX1	xxx
DS + 0024:	LX0	LX1	LX2
DS + 0027:	BX0	BX1	xxx
DS + 002A:	LX0	LX1	LX2
DS + 002D:	BX0	BX1	xxx
DS + 0030:	Not Currently Used		
...			
DS + 005B:			
DS + 005C:	Default FCB		
DS + 0080:	Default Buffer		
DS + 0100:	Begin User Data		

CP/M-86 Base Page Values

Each byte is indexed by 0, 1, and 2, corresponding to the standard Intel storage convention of low, middle, and high-order (most significant) byte. "xxx" in the figure above marks unused bytes. LC is the last code group location (24-bits, where the 4 high-order bits equal zero).

BC is the base paragraph address of the code group (16-bits). LD and BD provide the last position and paragraph base of the data group. The last position is one byte less than the group length. The M80 byte is equal to 1 when the 8080 Memory Model is in use. LE and BE provide the length and paragraph base of the optional extra group, while LS and BS give the optional stack group length and base. The bytes marked LX and BX correspond to a set of four optional independent groups which may be required for programs which execute using the Compact Memory Model. The initial values for these descriptors are derived from the header record in the memory image file, described in Appendix C.

Transient Program Load and Exit

The CCP parses up to two filenames following the command and places the properly formatted FCB's at locations 005CH and 006CH in the base page relative to the DS register. Therefore, the default DMA base is initialized to the value of DS, and the default DMA offset is initialized to 0080H. CP/M-86 assumes the default DMA buffer occupies the second half of the base page.

The CCP transfers control to the transient program through an 8086 "Far Call." The transient program may choose to use the 96-byte CCP stack and return directly to the CCP upon program termination by executing a "Far Return." Optionally, the user may set his/her own stack using the example of stack segment initialization shown in the previous section on the CP/M-86 Compact Memory Model. The program will also terminate when BDOS function zero is executed. The operator may terminate program execution by typing a single Ctrl-C during line edited input. This has the same effect as executing BDOS function zero.

APPENDIX C. COMMAND (CMD) FILE GENERATION

Contents

Command (CMD) File Generation

As mentioned previously, a utility program is provided with CP/M-86, called GENCMD, which is used to produce CMD memory image files suitable for execution under CP/M-86. GENCMD accepts Intel 8086 "hex" format files as input. GENCMD is used to process output from the Digital Research ASM-86 assembler or Intel's OH86 utility.

Intel 8086 Hex File Format

GENCMD input is in Intel "hex" format produced by both the Digital Research ASM-86 assembler and the standard Intel OH86 utility program (see Intel document #9800639-03 entitled *"CS-86 Software Development Utilities Operating Instructions for ISIS-II Users"*). The CMD file produced by GENCMD contains a header record which defines the memory model and memory size requirements for loading and executing the CMD file.

An Intel "hex" file consists of the traditional sequence of ASCII records in the following format:

:llaaaattddd...dcc

where the beginning of the record is marked by an ASCII colon, and each subsequent digit position contains an ASCII hexadecimal digit in the range 0–9 or A–F. The fields are defined below.

Intel Hex Field Definitions

Field	Contents
ll	Record Length 00–FF (0–255 in decimal)
aaaa	Load Address
tt	Record Type:
	Record Types Generated When $FI Switch Is Used With ASM86:

Intel Hex Field Definitions (continued)

Field	Contents
	00 data record, loaded starting at offset aaaa from current base paragraph 01 end of file 02 extended address, aaaa is paragraph base for subsequent data records 03 start address is aaaa (ignored, IP set according to memory model in use) Record Types Generated When $FD Switch (Default) Is Used With ASM-86: 01 end of file 81H data belongs to code segment 82H data belongs to data segment 83H data belongs to stack segment 84H data belongs to extra segment 85H paragraph address for absolute code segment 86H paragraph address for absolute data segment 87H paragraph address for absolute stack segment 88H paragraph address for absolute extra segment
dd...d	Data Byte
cc	Check Sum (00–Sum of Previous Digits)

All characters preceding the colon for each record are ignored. (Additional hex file format information is included in Chapter 11, and in Intel's document #9800821A entitled *MCS-86 Absolute Object File Formats*.)

Operation of GENCMD

The GENCMD utility is invoked at the command level by typing

GENCMD filename parameter-list

where the filename corresponds to the hex input file with an assumed (and unspecified) file type of H86. GENCMD accepts optional parameters to specifically identify the 8080 Memory Model and to describe memory requirements of each segment group. The GENCMD parameters are listed following the filename, as shown in the command line above where the parameter-list consists of a sequence of keywords and values separated by commas or blanks. The keywords are:

8080 CODE DATA EXTRA STACK

The 8080 keyword forces a single code group so that the BDOS load function sets up the 8080 Memory Model for execution, thus allowing intermixed code and data within a single segment. The form of this command is

GENCMD filename 8080

The remaining keywords follow the filename or the 8080 option and define specific memory requirements for each segment group, corresponding one-to-one with the segment groups defined in the previous section. In each case, the values corresponding to each group are enclosed in square brackets and separated by commas. Each value is a hexadecimal number representing a paragraph address or segment length in paragraph units denoted by hhhh, prefixed by a single letter which defines the meaning of each value:

Ahhhh	Load the group at absolute location hhhh
Bhhhh	The group starts at hhhh in the hex file
Mhhhh	The group requires a minimum of hhhh * 16 bytes
Xhhhh	The group can address a maximum of hhhh * 16 bytes

Generally, the CMD file header values are derived directly from the hex file and the parameters shown above need not be included. The following situations, however, require the use of GENCMD parameters.

- 8080 Keyword–The 8080 keyword is included whenever ASM-86 is used in the conversion of 8080 programs to the 8086/8088 environment when code and data are intermixed within a single segment, regardless of the use of CSEG and DSEG directives in the source program.

- Absolute Address–An absolute address (A value) must be given for any group which must be located at an absolute location. Normally, this value is not specified since CP/M-86 cannot generally ensure that the required memory region is available, in which case the CMD file cannot be loaded.

- Beginning Address of Groups–The B value is used when GENCMD processes a hex file produced by Intel's OH86, or similar utility program that contains more than one group. The output from OH86 consists of a sequence of data records with no information to identify code, data, extra, stack, or auxiliary groups. In this case, the B value marks the beginning address of the group named by the keyword, causing GENCMD to load data following this address to the named group (see the examples below). Thus, the B value is normally used to mark the boundary between code and data segments when no segment information is included in the hex file. Files produced by ASM-86 do not require the use of the B value since segment information is included in the hex file.

- Minimum Memory Value–The M value (minimum memory value) is included only when the hex records do not define the minimum memory requirements for the named group. Generally, the code group size is determined precisely by the data records loaded into the area. That is, the total space required for the group is defined by the range between the lowest and highest data byte addresses. The data group, however, may contain uninitialized storage at the end of the group and thus no data records are present in the hex file which define the highest referenced data item. The highest address in the data group can be defined within the source program by including a "DB 0" as the last data item. Alternatively, the M value can be included to allocate the additional space at the end of the group. Similarly, the stack, extra, and auxiliary group sizes must be defined using the M value unless the highest addresses within the groups are implicitly defined by data records in the hex file.

- Maximum Memory Size–The maximum memory size, given by the X value, is generally used when additional free memory may be needed for such purposes as I/O buffers or symbol tables. If the data area size is fixed, then the X parameter need not be included. In this case, the X value is assumed to be the same as the M value. The value XFFFF allocates the largest memory region available but, if used, the transient program must be aware that a three-byte length field is produced in the base page for this group where the high-order byte may be non-zero. Programs converted directly from CP/M-80 or programs that use a 2-byte pointer to address buffers should restrict this value to XFFF or less, producing a maximum allocation length of 0FFF0H bytes.

The following GENCMD command line transforms the file X.H86 into the file X.CMD with the proper header record:

gencmd x code[a40] data[m30,xfff]

In this case, the code group is forced to paragraph address 40H, or equivalently, byte address 400H. The data group requires a minimum of 300H bytes, but can use up to 0FFF0H bytes, if available.

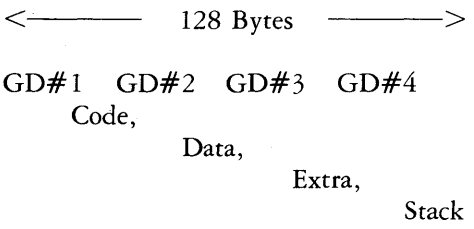
Assuming a file Y.H86 exists on drive B containing Intel hex records with no interspersed segment information, the command

gencmd b:y data[b30,m20] extra[b50] stack[m40]

produces the file Y.CMD on drive B by selecting records beginning at address 0000H for the code segment, with records starting at 300H allocated to the data segment. The extra segment is filled from records beginning at 500H, while the stack segment is an uninitialized area requiring a minimum of 400H bytes. In this example, the data area requires a minimum of 200H bytes. Note again, that the B value need not be included if the Digital Research ASM-86 assembler is used.

Command (CMD) File Format

The CMD file produced by GENCMD consists of the 128-byte header record followed immediately by the memory image. Under normal circumstances, the format of the header record is of no consequence to a programmer. For completeness, however, the various fields of this record are shown below.



CMD File Header Format

In the figure above, GD#1 through GD#4 represent "Group Descriptors." Each Group Descriptor corresponds to an independently loaded program unit and has the following fields:

8-bit	16-bit	16-bit	16-bit	16-bit
G-Form	G-Length	A-Base	G-Min	G-Max

where G-Form describes the group format, or has the value zero if no more descriptors follow. If G-Form is non-zero, then the 8-bit value is parsed as two fields:

G-Form:

4-bit	4-bit
x x x x	G-Type

The G-Type field determines the Group Descriptor type. The valid Group Descriptors have a G-Type in the range 1 through 4, as shown in the table below.

Group Descriptors

G-Type	Group Type
1	Code Group
2	Data Group
3	Extra Group
4	Stack Group
5–14	Unused,but Reserved

All remaining values in the Group Descriptor are given in increments of 16-byte paragraph units with an assumed low-order 0 nibble to complete the 20-bit address. G-Length gives the number of paragraphs in the group. Given a G-Length of 0080H, for example, the size of the group is 00800H = 2048D bytes. A-Base defines the base paragraph address for a non-relocatable group while G-Min and G-Max define the minimum and maximum size of the memory area to allocate to the group.

The memory model described by a header record is implicitly determined by the Group Descriptors. The 8080 Memory Model is assumed when only a code group is present, since no independent data group is named. The Small Model is implied when both a code and data group are present, but no additional Group Descriptors occur. Otherwise, the Compact Model is assumed when the CMD file is loaded.

APPENDIX D. BASIC DISK OPERATING SYSTEM (BDOS) FUNCTIONS

Basic Disk Operating System (BDOS) Functions

This section presents the interface conventions which allow transient program access to CP/M-86 BDOS and BIOS functions. The BDOS calls correspond closely to CP/M-80 Version 2 in order to simplify translation of existing CP/M-80 programs for operation under CP/M-86. BDOS entry and exit conditions are described first, followed by a presentation of the individual BDOS function calls.

BDOS Parameters and Function Codes

Entry to the BDOS is accomplished through the 8086 software interrupt #224, which is reserved by Intel Corporation for use by CP/M-86. The function code is passed in register CL with byte parameters in DL and word parameters in DX. Single byte values are returned in AL, word values in both AX and BX, and double word values in ES and BX. All segment registers, except ES, are saved upon entry and restored upon exit from the BDOS (corresponding to PL/M-86 conventions). The table below summarizes input and output parameter passing.

BDOS Parameter Summary

BDOS Entry Registers	BDOS Return Registers
CL Function Code	Byte value returned in AL
DL Byte Parameter	Word value returned in both AX and BX
DX Word Parameter	Double-word value returned with offset in BX and segment in ES
DS Data Segment	

Note that the CP/M-80 BDOS requires an "information address" as input to various functions. This address usually provides buffer or File Control Block information used in the system call. In CP/M-86, however, the information address is derived from the current DS register combined with the offset given in the DX register. That is, the DX register in CP/M-86 performs the same function as the DE pair in CP/M-80, with the assumption that DS is properly set. This poses no

particular problem for programs which use only a single data segment (as is the case for programs converted from CP/M-80), but when the data group exceeds a single segment, you must ensure that the DS register is set to the segment containing the data area related to the call. It should also be noted that zero values are returned for function calls which are out-of-range.

A list of CP/M-86 calls is given below with an asterisk following functions which differ from or are added to the set of CP/M-80 Version 2 functions.

CP/M-86 BDOS Functions

F#	Result	F#	Result
0*	System Reset	24	Return Login Vector
1	Console Input	25	Return Current Disk
2	Console Output	26	Set DMA Address
3	Reader Input	27*	Get Addr(Alloc)
4	Punch Output	28	Write Protect Disk
5	List Output	29	Get Addr(R/O Vector)
6*	Direct Console I/O	30	Set File Attributes
7	Get I/O Byte	31*	Get Addr(Disk Parms)
8	Set I/O Byte	32	Set/Get User Code
9	Print String	33	Read Random
10	Read Console Buffer	34	Write Random
11	Get Console Status	35	Compute File Size
12	Return Version Number	36	Set Random Record
		37*	Reset Drive
13	Reset Disk System	40	Write Random With Zero Fill
14	Select Disk	50*	Direct BIOS Call
15	Open File	51*	Set DMA Segment Base
16	Close File	52*	Get DMA Segment Base
17	Search for First	53*	Get Max Memory Available
18	Search for Next	54*	Get Max Mem at Abs Location
19	Delete File	55*	Get Memory Region
20	Read Sequential	56*	Get Absolute Memory Region
21	Write Sequential	57*	Free Memory Region
22	Make File	58*	Free All Memory
23	Rename File	59*	Program Load

The individual BDOS functions are described below in three sections which cover the simple functions, file operations, and extended operations for memory management and program loading.

Simple BDOS Calls

The first set of BDOS functions cover the range 0 through 12, and perform simple functions such as system reset and single character I/O.

Function 0: System Reset

Entry	*Return*
CL: 00H	(none)
DL: Abort Code	

The System Reset Function returns control to the CP/M operating system at the CCP command level. The abort code in DL has two possible values: if DL = 00H then the currently active program is terminated and control is returned to the CCP; if DL is a 01H, the program remains in memory and the memory allocation state remains unchanged.

Function 1: Console Input

Entry	*Return*
CL: 01H	AL: ASCII Character

The Console Input function reads the next character from the logical console device (CONSOLE) to register AL. Graphic characters, along with carriage return, line-feed, and backspace (Ctrl-H) are echoed to the console. Tab characters (Ctrl-I) are expanded in columns of eight characters. The BDOS does not return to the calling program until a character has been typed, thus suspending execution if a character is not ready.

Function 2: Console Output

Entry	*Return*
CL: 02H	(none)
DL: ASCII Character	

The ASCII character from DL is sent to the logical console. Tab characters expand in columns of eight characters. In addition, a check is made for start/stop scroll (Ctrl-S).

Function 3: AXI: Input

Entry	*Return*
CL: 03H	AL: ASCII Character

The AXI: Input function reads the next character from the logical AXI: device into register AL. Control does not return until the character has been read.

Function 4: AXO: Output

Entry	*Return*
CL: 04H	(none)
DL: ASCII Character	

The AXO: Output function sends the character from register DL to the logical AXO: device.

Function 5: List Output

Entry	*Return*
CL: 05H	(none)
DL: ASCII Character	

The List Output function sends the ASCII character in register DL to the logical list device (LIST).

Function 6: Direct Console I/O

Entry	*Return*
CL: 06H	AL: char or status
DL: 0FFH (input/status) or 0FEH (status) or char (output)	(no value)

Direct Console I/O is supported under CP/M-86 for those specialized applications where unadorned console input and output is required. Use of this function should, in general, be avoided since it bypasses all of CP/M-86's normal control character functions (e.g., Ctrl-S and Ctrl-P). Programs which perform direct I/O through the BIOS under previous releases of CP/M-80, however, should be changed to use direct I/O under the BDOS so that they can be fully supported under future releases of CP/M.

Upon entry to Function 6, register DL either contains (1) a hexadecimal FF, denoting a CONSOLE input/status request, or (2) a hexadecimal FE, denoting a CONSOLE status request, or (3) an ASCII character to be output to CONSOLE where CONSOLE is the logical console device. If the input value is FF, then Function 6 checks to see if a character is ready. If a character is ready, Function 6 with FF returns the character; otherwise it returns a zero. You cannot use Function 6 with FF or FE in combination with Function 1 or Function 11. Function 6 must be used independently.

The next console input character is returned in AL. If the input value is FE, then Function 6 returns AL = 00 if no character is ready and AL = FF otherwise. If the input value in DL is not FE or FF, then Function 6 assumes that DL contains a valid ASCII character which is sent to the console.

Function 7: Get I/O Byte

Entry	*Return*
CL: 07H	AL: I/O Byte Value

The Get I/O Byte function returns the current value of IOBYTE in register AL. The IOBYTE contains the current assignments for the logical devices CONSOLE, READER. PUNCH, and LIST, provided the IOBYTE facility is implemented in the BIOS.

Function 8: Set I/O Byte

Entry	*Return*
CL: 08H	(none)
DL: I/O Byte Value	

The Set I/O Byte function changes the system IOBYTE value to that given in register DL. This function allows transient program access to the IOBYTE in order to modify the current assignments for the logical devices CONSOLE, READER, PUNCH, and LIST.

Function 9: Print String

Entry	*Return*
CL: 09H	(none)
DX: String Offset	

The Print String function sends the character string stored in memory at the location given by DX to the logical console device (CONSOLE), until a "$" is encountered in the string. Tabs are expanded as in Function 2, and checks are made for start/stop scroll and printer echo.

Function 10: Read Console Buffer

Entry	*Return*
CL: 0AH	Console Characters
DX: Buffer Offset	in Buffer

The Read Buffer function reads a line of edited console input into a buffer addressed by register DX from the logical console device (CONSOLE). Console input is terminated when either the input buffer is filled or when a return (Ctrl-M) or a line-feed (Ctrl-J) character is entered. The input buffer addressed by DX takes the form:

DX:	+0	+1	+2	+3	+4	+5	+6	+7	+8	...	+n
	mx	nc	c1	c2	c3	c4	c5	c6	c7	...	??

where "mx" is the maximum number of characters which the buffer will hold, and "nc" is the number of characters placed in the buffer.

The characters entered by the operator follow the "nc" value. The value "mx" must be set prior to making a Function 10 call and may range in value from 1 to 255. Setting mx to zero is equivalent to setting mx to one. The value "nc" is returned to the user and may range from 0 to mx. If nc < mx, then uninitialized positions follow the last character, denoted by "??" in the above figure. Note that a terminating return or line-feed character is not placed in the buffer and not included in the count "nc".

A number of editing control functions are supported during console input under Function 10. These are summarized in the table below.

Line Editing Controls

Keystroke	Result
Ctrl-C	reboots when at the beginning of line
Ctrl-E	causes physical end of line
Ctrl-H	backspaces one character position
Ctrl-J	(line-feed) terminates input line
Ctrl-M	(return) terminates input line
Ctrl-R	retypes the current line after new line
Ctrl-U	removes current line after new line
Ctrl-X	backspaces to beginning of current line

Certain 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.

Function 11: Get Console Status

Entry	*Return*
CL: 0BH	AL: Console Status

The Console Status function checks to see if a character has been typed at the logical console device (CONSOLE). If a character is ready, the value 01H is returned in register AL. Otherwise a 00H value is returned.

Function 12: Return Version Number

Entry	*Return*
CL: 0CH	BX: Version Number

Function 12 provides information which allows version independent programming. A two-byte value is returned, with BH = 00 designating the CP/M release and BL = 00 for all releases previous to 2.0. CP/M 2.0 returns a hexadecimal 20 in register BL, with subsequent Version 2 releases in the hexadecimal range 21, 22, through 2F. To provide version number compatibility, the initial release of CP/M-86 returns a 2.2.

BDOS File Operations

Functions 12 through 52 are related to disk file operations under CP/M-86. In many of these operations, DX provides the DS-relative offset to a file control block (FCB). The File Control Block (FCB) data area consists of a sequence of 33 bytes for sequential access, or a sequence of 36 bytes in the case that the file is accessed randomly. The default file control block normally located at offset 005CH from the DS register can be used for random access files, since bytes 007DH, 007EH, and 007FH are available for this purpose. Here is the FCB format, followed by definitions of each of its fields:

```
dr f1 f2 // f8 t1 t2 t3 ex s1 s2 rc d0 // dn cr r0 r1 r2
00 01 02 ... 08 09 10 11 12 13 14 15 16 ... 31 32 33 34 35
```

where

dr — drive code (0–16)
0 —> use default drive for file
1 —> auto disk select drive A,
2 —> auto disk select drive B,
...
16—> auto disk select drive P. (Note that CP/M-86 on the IBM Personal Computer supports drives 0–4, corresponding to A–D.)

f1...f8 — contain the file name in ASCII upper case, with high bit = 0

t1,t2,t3 — contain the file type in ASCII upper-case, with high-bit = 0
t1', t2', and t3' denote the high-bit of these positions,
t1' = 1 —> Read/Only file,
t2' = 1 —> SYS file, no DIR list

ex	contains the current extent number, normally set to 00 by the user, but in range 0–31 during file I/O
s1	reserved for internal system use
s2	reserved for internal system use, set to zero on call to OPEN, MAKE, SEARCH
rc	record count for extent "ex," takes on values from 0–128
d0...dn	filled-in by CP/M, reserved for system use
cr	current record to read or write in a sequential file operation, normally set to zero by user
r0,r1,r2	optional random record number in the range 0–65535, with overflow to r2, r0,r1 constitute a 16-bit value with low byte r0, and high byte r1

For users of earlier versions of CP/M, it should be noted that both CP/M Version 2 and CP/M-86 perform directory operations in a reserved area of memory that does not affect write buffer content, except in the case of Search and Search Next where the directory record is copied to the current DMA address.

There are three error situations that the BDOS may encounter during file processing, initiated as a result of a BDOS File I/O function call. When one of these conditions is detected, the BDOS issues a message of the following form:

BDOS ERR ON x: error

where x is the drive name of the drive selected when the error condition is detected, and "error" is one of the three messages:

BAD SECTOR SELECT R/O

These error situations are trapped by the BDOS, and thus the executing transient program is temporarily halted when the error is detected. No indication of the error situation is returned to the transient program.

The "BAD SECTOR" error is issued as the result of an error condition returned to the BDOS from the BIOS module. The BDOS makes BIOS sector read and write commands as part of the execution of BDOS file-related system calls. If the BIOS read or write routine detects a hardware error, it returns an error code to the BDOS resulting in this error message. The operator may respond to this error in two ways: a Ctrl-C terminates the executing program.

The "SELECT" error is also issued as the result of an error condition returned to the BDOS from the BIOS module. The BDOS makes a BIOS disk select call prior to issuing any BIOS read or write to a particular drive. If the selected drive is not supported in the BIOS module, it returns an error code to the BDOS resulting in this error message. CP/M-86 terminates the currently running program and returns to the command level of the CCP following any input from the console.

The "R/O" message occurs when the BDOS receives a command to write to a drive that is in Read-Only status. Drives may be placed in Read-Only status explicitly as the result of a STAT command or BDOS function call, or implicitly if the BDOS detects that a diskette has been changed without performing a "warm start." The ability to detect changed media is optionally included in the BIOS, and exists only if a checksum vector is included for the selected drive. Upon entry of any character at the keyboard, the transient program is terminated, and control returns to the CCP.

Function 13: Reset Disk System

Entry	*Return*
CL: 0DH	(none)

The Reset Disk function is used to programmatically restore the file system to a reset state where all disks are set to Read/Write (see functions 28 and 29); only disk drive A is selected. This function can be used, for example, by an application program which requires disk changes during operation. Function 37 (Reset Drive) can also be used for this purpose.

Function 14: Select Disk

Entry	*Return*
CL: 0EH	(none)
DL: Selected Disk	

The Select Disk function designates the disk drive named in register DL as the default disk for subsequent file operations, with DL = 0 for drive A, 1 for drive B, and so forth through 15 corresponding to drive P in a full 16-drive system. In addition, the designated drive is logged-in if it is currently in the reset state. Logging-in a drive places it in "on-line" status which activates the drive's directory until the next cold start, warm start, disk system reset, or drive reset operation. FCB's that specify drive code zero (dr = 00H) automatically reference the currently selected default drive. Drive code values between 1 and 16, however, ignore the selected default drive and directly reference drives A through P.

Function 15: Open File

Entry	*Return*
CL: 0FH	AL: Return Code
DX: FCB Offset	

The Open File function is used to activate an FCB specifying a file which currently exists in the disk directory for the currently active user number. The BDOS scans the disk directory of the drive specified by byte 0 of the FCB referenced by DX for a match in positions 1 through 12 of the referenced FCB, where positions 1–8 specify the filename, positions 9–11 specify the filetype, and position 12 specifies the extent. Normally, the extent byte is set to zero.

If a directory element is matched, the relevant directory information is copied into bytes d0 through dn of the FCB, thus allowing access to the files through subsequent read and write operations. Note that an existing file must not be accessed until a successful open operation is completed. Further, an FCB not activated by either an open or make function must not be used in BDOS read or write commands. Upon return, the open function returns a "directory code" with the value 0 through 3 if the open was successful, or 0FFH (255 decimal) if the file cannot be found. With the exception of the BDOS search functions, Directory Code values (0–3) have no significance other than to indicate a successful result. However, for the search functions, a successful Directory Code identifies the relative starting position of the directory element in the calling process's current DMA buffer. Note that the current record ("cr") must be zeroed by the program if the file is to be accessed sequentially from the first record.

Function 16: Close File

Entry	*Return*
CL: 10H	AL: Return Code
DX: FCB Offset	

The Close File function performs the inverse of the Open File function. Given that the FCB addressed by DX has been previously activated through an Open or Make function (see functions 15 and 22), the Close function permanently records the new FCB in the referenced disk directory. The FCB matching process for the close is identical to the Open function. The directory code returned for a successful close operation is 0, 1, 2, or 3, while a 0FFH (255 decimal) is returned if the file name cannot be found in the directory. A file need not be closed if only read operations have taken place. If write operations have occurred, however, the close operation is necessary to permanently record the new directory information.

Function 17: Search For First

Entry	*Return*
CL: 11H	AL: Directory Code
DX: FCB Offset	

Search for First scans the directory for a match with the file given by the FCB addressed by DX. The value 255 (hexadecimal FF) is returned if the file is not found, otherwise 0, 1, 2, or 3 is returned indicating the file is present. In the case that the file is found, the buffer at the current DMA address is filled with the record containing the directory entry, and its relative starting position is AL * 32 (i.e., rotate the AL register left 5 bits). Although not normally required for application programs, the directory information can be extracted from the buffer at this position.

An ASCII question mark (63 decimal, 3F hexadecimal) in any position from "f1" through "ex" matches the corresponding field of any directory entry on the default or auto-selected disk drive. If the "dr" field contains an ASCII question mark, then the auto disk select function is disabled, the default disk is searched, with the search function returning any matched entry, allocated or free, belonging to any user number. This latter function is not normally used by application programs, but does allow complete flexibility to scan all current directory values. If the "dr" field is not a question mark, the "s2" byte is automatically zeroed.

Function 18: Search For Next

Entry	*Return*
CL: 12H	AL: Directory Code

The Search for Next function is similar to the Search for First function, except that the directory scan continues from the last matched entry. Similar to Function 17, Function 18 returns the decimal value 255 in A when no more directory items match. In terms of execution sequence, a Function 18 call must follow either a Function 17 or Function 18 call with no other intervening BDOS disk-related function calls.

Function 19: Delete File

Entry	*Return*
CL: 13H	AL: Return Code
DX: FCB Offset	

The Delete File function removes files which match the FCB addressed by DX. The filename and type may contain ambiguous references (i.e., question marks in various positions), but the drive select code cannot be ambiguous, as in the Search for First and Search for Next functions. Function 19 returns a 0FFH (decimal 255) if the referenced file or files cannot be found, otherwise a value of zero is returned.

Function 20: Read Sequential

Entry	*Return*
CL: 14H	AL: Return Code
DX: FCB Offset	

Given that the FCB addressed by DX has been activated through an open or make function (numbers 15 and 22), the Read Sequential function reads the next 128-byte record from the file into memory at the current DMA address. The record

is read from position "cr" of the extent, and the "cr" field is automatically incremented to the next record position. If the "cr" field overflows then the next logical extent is automatically opened and the "cr" field is reset to zero in preparation for the next read operation. The "cr" field must be set to zero following the open call by the user if the intent is to read sequentially from the beginning of the file. The value 00H is returned in the AL register if the read operation was successful, while a value of 01H is returned if no data exists at the next record position of the file. Normally, the no data situation is encountered at the end of a file. However, it can also occur if an attempt is made to read a data block which has not been previously written, or an extent which has not been created. These situations are usually restricted to files created or appended by use of the BDOS Write Random commmand (Function 34).

Function 21: Write Sequential

Entry	*Return*
CL: 15H	AL: Return Code
DX: FCB Offset	

Given that the FCB addressed by DX has been activated through an open or make function (numbers 15 and 22), the Write Sequential function writes the 128-byte data record at the current DMA address to the file named by the FCB. The record is placed at position "cr" of the file, and the "cr" field is automatically incremented to the next record position. If the "cr" field overflows then the next logical extent is automatically opened and the "cr" field is reset to zero in preparation for the next write operation. Write operations can take place into an existing file, in which case newly written records overlay those which already exist in the file. The "cr" field must be set to zero following an open or make call by the user if the intent is to write sequentially from the beginning of the file. Register AL = 00H upon return from a successful write operation, while a non-zero value indicates an unsuccessful write due to one of the following conditions:

01 No available directory space–This condition occurs when the write command attempts to create a new extent that requires a new directory entry and no available directory entries exist on the selected disk drive.

02 No available data block–This condition is encountered when the write command attempts to allocate a new data block to the file and no unallocated data blocks exist on the selected disk drive.

Function 22: Make File

Entry	*Return*
CL: 16H	AL: Return Code
DX: FCB Offset	

The Make File operation is similar to the Open File operation except that the FCB must name a file which does not exist in the currently referenced disk directory (i.e., the one named explicitly by a non-zero "dr" code, or the default disk if "dr" is zero). The BDOS creates the file and initializes both the directory and main memory value to an empty file. The programmer must ensure that no duplicate filenames occur, and a preceding delete operation is sufficient if there is any possibility of duplication. Upon return, register A = 0, 1, 2, or 3 if the operation was successful and 0FFH (255 decimal) if no more directory space is available. The Make function has the side-effect of activating the FCB and thus a subsequent open is not necessary.

Function 23: Rename File

Entry	*Return*
CL: 17H	AL: Return Code
DX: FCB Offset	

The Rename File function uses the FCB addressed by DX to change all directory entries of the file specified by the filename in the first 16 bytes of the FCB to the filename in the second 16 bytes. It is the user's responsibility to ensure that the filenames specified are valid CP/M unambiguous filenames. The drive code "dr" at position 0 is used to select the drive, while the drive code for the new filename at position 16 of the FCB is ignored. Upon return, register AL is set to a value of zero if the rename was successful, and 0FFH (255 decimal) if the first filename could not be found in the directory scan.

Function 24: Return Login Vector

Entry	*Return*
(none)	BX: Login Vector

The login vector value returned by CP/M-86 is a 16-bit value in BX, where the least significant bit corresponds to the first drive A, and the high-order bit corresponds to the sixteenth drive, labelled P. A "0" bit indicates that the drive is not on-line, while a "1" bit marks a drive that is actively on-line due to an explicit disk drive selection, or an implicit drive select caused by a file operation which specified a non-zero "dr" field.

Function 25: Return Current Disk

Entry	*Return*
CL: 19H	AL: Current Disk

Function 25 returns the currently selected default disk number in register AL. The disk numbers range from 0 through 15 corresponding to drives A through P.

Function 26: Set DMA Address

Entry	*Return*
CL: 1AH	(none)
DX: DMA Offset	

"DMA" is an abbreviation for Direct Memory Address, which is often used in connection with disk controllers which directly access the memory of the mainframe computer to transfer data to and from the disk subsystem. Although many computer systems use non-DMA access (i.e., the data is transferred through programmed I/O operations), the DMA address has, in CP/M, come to mean the address at which the 128-byte data record resides before a disk write and after a disk read. In the CP/M-86 environment, the Set DMA function is used to specify the offset of the read or write buffer from the current DMA base. Therefore, to specify the DMA address, both a Function 26 call and a Function 51 call are required. Thus, the DMA address becomes the value specified by DX plus the DMA base value until it is changed by a subsequent Set DMA or Set DMA base function.

Function 27: Get ADDR (ALLOC)

Entry	*Return*
CL: 1BH	BX: ALLOC Offset
	ES: Segment Base

An "allocation vector" is maintained in main memory for each on-line disk drive. Various system programs use the information provided by the allocation vector to determine the amount of remaining storage (see the STAT program). Function 27 returns the segment base and the offset address of the allocation vector for the currently selected disk drive. The allocation information may, however, be invalid if the selected disk has been marked Read/Only.

Function 28: Write Protect Disk

Entry *Return*

CL: 1CH (none)

The Write Protect Disk function provides temporary write protection for the currently selected disk. Any attempt to write to the disk, before the next cold start, warm start, disk system reset, or drive reset operation produces a message of the following form:

Bdos Err on d: R/O

Function 29: Get Read/Only Vector

Entry *Return*

CL: 1DH BX: R/O Vector Value

Function 29 returns a bit vector in register BX which indicates drives which have the temporary read/only bit set. Similar to Function 24, the least significant bit corresponds to drive A, while the most significant bit corresponds to drive P. The R/O bit is set either by an explicit call to Function 28, or by the automatic software mechanisms within CP/M-86 which detect changed disks.

Function 30: Set File Attributes

Entry *Return*

CL: 1EH AL: Return Code

DX: FCB
Offset

The Set File Attributes function allows programmatic manipulation of permanent indicators attached to files. In particular, the R/O, System and Archive attributes (t1', t2', and t3') can be set or reset. The DX pair addresses an FCB containing a filename with the appropriate attributes set or reset. It is the user's responsibility to ensure that an ambiguous filename is not specified. Function 30 searches the default disk drive directory area for directory entries that

belong to the current user number and that match the FCB specified name and type fields. All matching directory entries are updated 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' are reserved for future system expansion. The currently assigned attributes are defined as follows:

t1': The R/O attribute indicates if set that the file is in Read/Only status. BDOS will not allow write commands to be issued to files in R/O status.

t2': The System attribute is referenced by the CP/M DIR utility. If set, DIR will not display the file in a directory display.

t3': The Archive attribute is reserved but not actually used by CP/M-86. If set it indicates that the file has been written to back up storage by a user-written archive program. To implement this facility, the archive program sets this attribute when it copies a file to back up storage; any programs updating or creating files reset this attribute. Further, the archive program backs up only those files that have the Archive attribute reset. Thus, an automatic back-up facility restricted to modified files can be easily implemented.

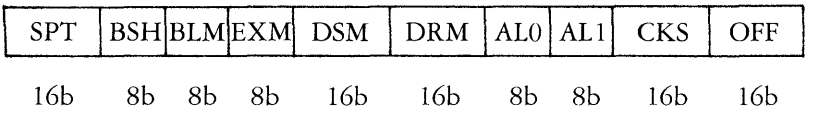
Function 30 returns with register AL set to 0FFH (255 decimal) if the referenced file cannot be found, otherwise a value of zero is returned.

Function 31: Get ADDR (Disk Parms)

Entry	*Return*
CL: 1FH	BX: DPB Offset
	ES: Segment Base

The offset and the segment base of the BIOS resident disk parameter block of the currently selected drive are returned in BX and ES as a result of this function call. This control block can be used for either of two purposes. First, the disk parameter values can be extracted for display and space

computation purposes, or transient programs can dynamically change the values of current disk parameters when the disk environment changes, if required. A disk parameter block has the following form:



The disk parameter block fields are defined in the table below.

Disk Parameter Block Fields

Field	*Description*
SPT	is the total number of sectors per track
BSH	determines data block allocation size
BLM	mask used by operating system
EXM	mask used by operating system
DSM	total storage capacity of disk drive
DRM	total number of directory entries
AL0, AL1	reserved directory allocation blocks
CKS	size of directory check vectors
OFF	is the number of reserved tracks at the beginning of the logical disk

Normally, application programs will not require this facility.

Function 32: Set/Get User Code

Entry	*Return*
CL: 20H	AL: Current Code or no value
DL: 0FFH(get) or User Code (set)	

An application program can change or interrogate the currently active user number by calling Function 32. If register DL = 0FFH, then the value of the current user number is returned in register AL, where the value is in the range 0 to 15. If register DL is not 0FFH, then the current user number is changed to the value of DL (modulo 16).

Function 33: Read Random

Entry	*Return*
CL: 21H	AL: Return Code
DX: FCB Offset	

The Read Random function is similar to the sequential file read operation of previous releases, except that the read operation takes place at a particular record number, selected by the 24-bit value constructed from the three-byte field following the FCB (byte positions r0 at 33, r1 at 34, and r2 at 35). Note that the sequence of 24 bits is stored with least significant byte first (r0), middle byte next (r1), and high byte last (r2). CP/M does not reference byte r2, except in computing the size of a file (Function 35). Byte r2 must be zero, however, since a non-zero value indicates overflow past the end of file.

Thus, the r0,r1 byte pair is treated as a double-byte, or "word" value, which contains the record to read. This value ranges from 0 to 65535, providing access to any particular record of any size file. In order to access a file using the Read Random function, the base extent (extent 0) must first be opened. Although the base extent may or may not contain any allocated data, this ensures that the FCB is properly initialized for subsequent random access operations. The selected record number is then stored into the random record field (r0,r1), and the BDOS is called to read the record. Upon return from the call, register AL either contains an error code, as listed below, or the value 00 indicating the operation was successful. In the latter case, the buffer at the current DMA address contains the randomly accessed record. Note that contrary to the sequential read operation, the record number is not advanced. Thus, subsequent random read operations continue to read the same record.

Upon each random read operation, the logical extent and current record values are automatically set. Thus, the file can be sequentially read or written, starting from the current randomly accessed position. Note, however, that in this case, the last randomly read record will be reread as you switch from random mode to sequential read, and the last record will be rewritten as you switch to a sequential write operation. You can, of course, simply advance the random record position following each random read or write to obtain the effect of a sequential I/O operation.

Error codes returned in register AL following a random read are listed in the table below.

Function 33–Read Random Error Codes

Code	*Meaning*
01	Reading unwritten data–This error code is returned when a random read operation accesses a data block which has not been previously written.
02	(not returned by the Random Read command)
03	Cannot close current extent–This error code is returned when BDOS cannot close the current extent prior to moving to the new extent containing the record specified by bytes r0,r1 of the FCB. This error can be caused by an overwritten FCB or a read random operation on an FCB that has not been opened.
04	Seek to unwritten extent–This error code is returned when a random read operation accesses an extent that has not been created. This error situation is equivalent to error 01.
05	(not returned by the Random Read command)
06	Random record number out of range–This error code is returned whenever byte r2 of the FCB is non-zero.

Normally, non-zero return codes can be treated as missing data, with zero return codes indicating operation complete.

Function 34: Write Random

Entry	*Return*
CL: 22H	AL: Return Code
DX: FCB Offset	

The Write Random operation is initiated similar to the Read Random call, except that data is written to the disk from the current DMA address. Further, if the disk extent or data block which is the target of the write has not yet been allocated, the allocation is performed before the write operation continues. As in the Read Random operation, the random record number is not changed as a result of the write. The logical extent number and current record positions of the file control block are set to correspond to the random record which is being written. Sequential read or write operations can commence following a random write, with the note that the currently addressed record is either read or rewritten again as the sequential operation begins. You can also simply advance the random record position following each write to get the effect of a sequential write operation. In particular, reading or writing the last record of an extent in random mode does not cause an automatic extent switch as it does in sequential mode.

In order to access a file using the Write Random function, the base extent (extent 0) must first be opened. As in the Read Random function, this ensures that the FCB is properly initialized for subsequent random access operations. If the file is empty, a Make File function must be issued for the base extent. Although the base extent may or may not contain any allocated data, this ensures that the file is properly recorded in the directory, and is visible in DIR requests.

Upon return from a Write Random call, register AL either contains an error code, as listed below, or the value 00 indicating the operation was successful.

Function 34–Write Random Error Codes

Code	*Meaning*
01	(not returned by the Random Write command)
02	No available data block–This condition is encountered when the Write Random command attempts to allocate a new data block to the file and no unallocated data blocks exist on the selected disk drive.
03	Cannot close current extent–This error code is returned when BDOS cannot close the current extent prior to moving to the new extent containing the record specified by bytes r0,r1 of the FCB. This error can be caused by an overwritten FCB or a write random operation on an FCB that has not been opened.
04	(not returned by the Random Write command)
05	No available directory space–This condition occurs when the write command attempts to create a new extent that requires a new directory entry and no available directory entries exist on the selected disk drive.
06	Random record number out of range–This error code is returned whenever byte r2 of the FCB is non-zero.

Function 35: Compute File Size

Entry	*Return*
CL: 23H	Random Record Field Set
DX: FCB Offset	

When computing the size of a file, the DX register addresses an FCB in random mode format (bytes r0, r1, and r2 are present). The FCB contains an unambiguous filename which is used in the directory scan. Upon return, the random record bytes contain the "virtual" file size which is, in effect, the record address of the record following the end of the file. If,

following a call to Function 35, the high record byte r2 is 01, then the file contains the maximum record count 65536. Otherwise, bytes r0 and r1 constitute a 16-bit value (r0 is the least significant byte, as before) which is the file size.

Data can be appended to the end of an existing file by simply calling Function 35 to set the random record position to the end of file, then performing a sequence of random writes starting at the preset record address.

The virtual size of a file corresponds to the physical size when the file is written sequentially. If, instead, the file was created in random mode and "holes" exist in the allocation, then the file may in fact contain fewer records than the size indicates. If, for example, a single record with record number 65535 (CP/M's maximum record number) is written to a file using the Write Random function, then the virtual size of the file is 65536 records, although only one block of data is actually allocated.

Function 36: Set Random Record

Entry	*Return*
CL: 24H	Random Record Field Set
DX: FCB Offset	

The Set Random Record function causes the BDOS to automatically produce the random record position of the next record to be accessed from a file which has been read or written sequentially to a particular point. The function can be useful in two ways.

First, it is often necessary to initially read and scan a sequential file to extract the positions of various "key" fields. As each key is encountered, Function 36 is called to compute the random record position for the data corresponding to this key. If the data unit size is 128 bytes, the resulting record position minus one is placed into a table with the key for later retrieval. After scanning the entire file and tabularizing the keys and their record numbers, you can move instantly to a particular keyed record by performing a random read using the corresponding random record number which was saved earlier. The scheme is easily generalized when variable record lengths are involved

since the program need only store the buffer-relative byte position along with the key and record number in order to find the exact starting position of the keyed data at a later time.

Second, you can use Function 36 to switch from a sequential read or write over to random read or write. A file is sequentially accessed to a particular point in the file, Function 36 is called which sets the record number, and subsequent random read and write operations continue from the next record in the file.

Function 37: Reset Drive

Entry	*Return*
CL: 25H	AL: 00H
DX: Drive Vector	

The Reset Drive function is used to programmatically restore specified drives to the reset state (a reset drive is not logged-in and is in read/write status). The passed parameter in register DX is a 16-bit vector of drives to be reset, where the least significant bit corresponds to the first drive, A, and the high-order bit corresponds to the sixteenth drive, labelled P. Bit values of "1" indicate that the specified drive is to be reset. CP/M returns a zero value for this function.

Function 40: Write Random With Zero Fill

Entry	*Return*
CL: 28H	AL: Return Code
DX: FCB Offset	

The Write Random With Zero Fill function is similar to the Write Random function (Function 34) with the exception that a previously unallocated data block is initialized to records filled with zeros before the record is written. If this function has been used to create a file, records accessed by a read random operation that contain all zeros identify unwritten random record numbers. Unwritten random records in allocated data blocks of files created using the Write Random function contain uninitialized data.

Function 50: Direct BIOS Call

Entry	*Return*
CL: 32H	(none)
DX: BIOS Descriptor	

Function 50 provides a direct BIOS call and transfers control through the BDOS to the BIOS. The DX register addresses a five-byte memory area containing the BIOS call parameters:

8-bit	16-bit	16-bit
Func	value(CX)	value(DX)

where Func is a BIOS function number (see the table below), and value (CX) and value (DX) are the 16-bit values which would normally be passed directly in the CX and DX registers with the BIOS call. The CX and DX values are loaded into the 8086 registers before the BIOS call is initiated.

BIOS Jump Vector

Offset from Beginning of BIOS	Suggested Instruction	BIOS F#	Description
2500H	JMP INIT	0	Arrive Here from Cold Boot
2503H	JMP WBOOT	1	Arrive Here for Warm Start
2506H	JMP CONST	2	Check for Console Character Ready
2509H	JMP CONIN	3	Read Console Character In
250CH	JMP CONOUT	4	Write Console Character Out
250FH	JMP LIST	5	Write Listing Character Out
2512H	JMP PUNCH	6	Write Char to Punch Device
2515H	JMP READER	7	Read Reader Device
2518H	JMP HOME	8	Move to Track 00
251BH	JMP SELDSK	9	Select Disk Drive
251EH	JMP SETTRK	10	Set Track Number
2521H	JMP SETSEC	11	Set Sector Number
2524H	JMP SETDMA	12	Set DMA Offset Address
2527H	JMP READ	13	Read Selected Sector
252AH	JMP WRITE	14	Write Selected Sector
252DH	JMP LISTST	15	Return List Status
2530H	JMP SECTRAN	16	Sector Translate
2533H	JMP SETDMAB	17	Set DMA Segment Address
2536H	JMP GETSEGB	18	Get MEM DESC Table Offset
2539H	JMP GETIOB	19	Get I/O Mapping Byte
253CH	JMP SETIOB	20	Set I/O Mapping Byte

Function 51: Set DMA Base

Entry	*Return*
CL: 33H	(none)
DX: Base Address	

Function 51 sets the base register for subsequent DMA transfers. The word parameter in DX is a paragraph address and is used with the DMA offset to specify the address of a 128-byte buffer area to be used in the disk read and write functions. Note that upon initial program loading, the default DMA base is set to the address of the user's data segment (the initial value of DS) and the DMA offset is set to 0080H, which provides access to the default buffer in the base page.

Function 52: Get DMA Base

Entry	*Return*
CL: 34H	BX: DMA Offset
	ES: DMA Segment

Function 52 returns the current DMA Base Segment address in ES, with the current DMA Offset in DX.

BDOS Memory Management and Load

Memory is allocated in two distinct ways under CP/M-86. The first is through a static allocation map, located within the BIOS, that defines the physical memory which is available on the host system. In this way, it is possible to operate CP/M-86 in a memory configuration which is a mixture of up to eight non-contiguous areas of RAM or ROM, along with reserved, missing, or faulty memory regions. In a simple RAM-based system with contiguous memory, the static map defines a single region, usually starting at the end of the BIOS and extending up to the end of available memory.

Once memory is physically mapped in this manner, CP/M-86 performs the second level of dynamic allocation to support transient program loading and execution. CP/M-86 allows dynamic allocation of memory into, again, eight regions. A request for allocation takes place either implicitly, through a program load operation, or explicitly through the BDOS calls given in this section. Programs themselves are loaded in two ways: through a command entered at the CCP level, or through the BDOS Program Load operation (Function 59). Multiple programs can be loaded at the CCP level, as long as each program executes a System Reset (Function 0) and remains in memory (DL = 01H). Multiple programs of this type receive control only by intercepting interrupts, and thus under normal circumstances there is only one transient program in memory at any given time. If, however, multiple programs are present in memory, then Ctrl-C characters entered by the operator delete these programs in the opposite order in which they were loaded no matter which program is actively reading the console.

Any given program loaded through a CCP command can, itself, load additional programs and allocate data areas. Suppose four regions of memory are allocated in the following order: A, B, C, and D. A program is loaded at the CCP level through an operator command. The CMD file header is read, and the entire memory image consisting of the program and its data is loaded into region A, and execution begins. This program, in turn, calls the BDOS Program Load function (59) to load another program into region B, and transfers control to the loaded program. The region B program then allocates an additional region C, followed by a region D. The order of allocation is shown in the figure below:

Region A

Region B

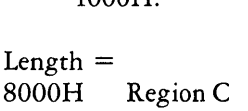
Region C

Region D

Example Memory Allocation

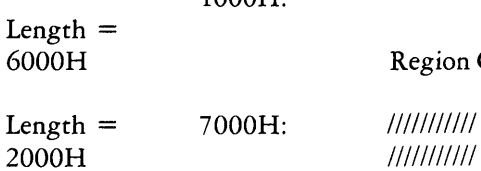
There is a hierarchical ownership of these regions: the program in A controls all memory from A through D. The program in B also controls regions B through D. The program in A can release regions B through D, if desired, and reload yet another program. DDT-86, for example, operates in this manner by executing the Free Memory call (Function 57) to release the memory used by the current program before loading another test program. Further, the program in B can release regions C and D if required by the application. It must be noted, however, that if either A or B terminates by a System Reset (BDOS Function 0 with DL = 00H) then all four regions A through D are released.

A transient program may release a portion of a region, allowing the released portion to be assigned on the next allocation request. The released portion must, however, be at the beginning or end of the region. Suppose, for example, the program in region B above receives 800H paragraphs at paragraph location 100H following its first allocation request. The result is as shown in the figure below.



Example Memory Region

Suppose further that region D is then allocated. The last 200H paragraphs in region C can be returned without affecting region D by releasing the 200H paragraphs beginning at paragraph base 700H, resulting in the memory arrangement shown in the figure below.



Example Memory Regions

The region beginning at paragraph address 7000H is now available for allocation in the next request. Note that a memory request will fail if eight memory regions have already been allocated. Normally, if all program units can reside in a contiguous region, the system allocates only one region.

Memory management functions beginning at 53 reference a Memory Control Block (MCB), defined in the calling program, which takes the form:

	16-bit	16-bit	8-bit
MCB:	M-Base	M-Length	M-Ext

where M-Base and M-Length are either input or output values expressed in 16-byte paragraph units, and M-Ext is a returned byte value, as defined specifically with each function code. An error condition is normally flagged with a 0FFH returned value in order to match the file error conventions of CP/M.

Function 53: Get MAX MEM

Entry	*Return*
CL: 35H	AL: Return Code
DX: Offset of MCB	

Function 53 finds the largest available memory region which is less than or equal to M-Length paragraphs. If successful, M-Base is set to the base paragraph address of the available area, and M-Length to the paragraph length. AL has the value 0FFH upon return if no memory is available, and 00H if the request was successful. M-Ext is set to 1 if there is additional memory for allocation, and 0 if no additional memory is available.

Function 54: Get ABS MAX

Entry	*Return*
CL: 36H	AL: Return Code
DX: Offset of MCB	

Function 54 is used to find the largest possible region at the absolute paragraph boundary given by M-Base, for a maximum of M-Length paragraphs. M-Length is set to the actual length if successful. AL has the value 0FFH upon return if no memory is available at the absolute address, and 00H if the request was successful.

Function 55: ALLOC MEM

Entry	*Return*
CL: 37H	AL: Return Code
DX: Offset of MCB	

The Allocate Memory function allocates a memory area according to the MCB addressed by DX. The allocation request size is obtained from M-Length. Function 55 returns in the user's MCB the base paragraph address of the allocated region. Register AL contains a 00H if the request was successful and a 0FFH if the memory could not be allocated.

Function 56: ALLOC ABS MEM

Entry	*Return*
CL: 38H	AL: Return Code
DX: Offset of MCB	

The Allocate Absolute Memory function allocates a memory area according to the MCB addressed by DX. The allocation request size is obtained from M-Length and the absolute base address from M-Base. Register AL contains a 00H if the request was successful and a 0FFH if the memory could not be allocated.

Function 57: Free MEM

Entry	*Return*
CL: 39H	(none)
DX: Offset of MCB	

Function 57 is used to release memory areas allocated to the program. The value of the M-Ext field controls the operation of this function: if M-Ext = 0FFH then all memory areas allocated by the calling program are released. Otherwise, the memory area of length M-Length at location M-Base given in the MCB addressed by DX is released (the M-Ext field should be set to 00H in this case). As described above, either an entire allocated region must be released, or the end of a region must be released; the middle section cannot be returned under CP/M-86.

Function 58: Free All MEM

Entry	*Return*
CL: 3AH	(none)

Function 58 is used to release all memory in the CP/M-86 environment (normally used only by the CCP upon initialization).

Function 59: Program Load

Entry	*Return*
CL: 3BH	AX: Return Code/ Base Page Addr
DX: Offset of FCB	BX: Base Page Addr

Function 59 loads a CMD file. Upon entry, register DX contains the DS relative offset of a successfully opened FCB which names the input CMD file. AX has the value 0FFFFH if the program load was unsuccessful. Otherwise, AX and BX both contain the paragraph address of the base page belonging to the loaded program. The base address and segment length of each segment are stored in the base page. Note that upon program load at the CCP level, the DMA base address is initialized to the base page of the loaded program, and the DMA offset address is initialized to 0080H. However, this is a function of the CCP, and a Function 59 does not establish a default DMA address. It is the responsibility of the program which executes Function 59 to execute Function 51 to set the DMA base and Function 26 to set the DMA offset before passing control to the loaded program.

Appendix E. Sample Random Access Program

This appendix contains an extensive and complete example of random access operation. The program listed here performs the simple function of reading or writing random records upon command from the terminal. Given that the program has been created, assembled, and placed into a file labelled RANDOM.CMD, the command:

RANDOM X.DAT

starts the test program. The program looks for a file by the name X.DAT (in this particular case) and, if found, proceeds to prompt the console for input. If not found, the file is created before the prompt is given. Each prompt takes the form

next command?

and is followed by operator input, terminated by a carriage return. The input commands take the form

nW nR Q

where n is an integer value in the range 0 to 65535, and W, R, and Q are simple command characters corresponding to random write, random read, and quit processing, respectively. If the W command is issued, the RANDOM program issues the prompt

type data:

The operator then responds by typing up to 127 characters, followed by a carriage return. RANDOM then writes the character string into the X.DAT file at record n. If the R command is issued, RANDOM reads record number n and displays the string value at the console. If the Q command is issued, the X.DAT file is closed, and the program returns to the console command processor. The only error message is

error, try again

The program begins with an initialization section where the input file is opened or created, followed by a continuous loop at the label "ready" where the individual commands are interpreted. The default file control block at offset 005CH and the default buffer at offset 0080H are used in all disk operations. The utility subroutines then follow, which contain the principal input line processor, called "readc." This particular program shows the elements of random access processing, and can be used as the basis for further program development. In fact, with some work, this program could evolve into a simple data base management system.

One could, for example, assume a standard record size of 128 bytes, consisting of arbitrary fields within the record. A program, called GETKEY, could be developed which first reads a sequential file and extracts a specific field defined by the operator. For example, the command

GETKEY NAMES.DAT LASTNAME 10 20

would cause GETKEY to read the data base file NAMES.DAT and extract the LASTNAME field from each record, starting at position 10 and ending at character 20. GETKEY builds a table in memory consisting of each particular LASTNAME field, along with its 16-bit record number location within the file. The GETKEY program then sorts this list, and writes a new file, called LASTNAME.KEY, which is an alphabetical list of LASTNAME fields with their corresponding record numbers. (This list is called an "inverted index" in information retrieval parlance.)

Rename the program shown above as QUERY, and enhance it a bit so that it reads a sorted key file into memory. The command line might appear as:

QUERY NAMES.DAT LASTNAME.KEY

Instead of reading a number, the QUERY program reads an alphanumeric string which is a particular key to find in the NAMES.DAT data base. Since the LASTNAME.KEY list is sorted, you can find a particular entry quite rapidly by performing a "binary search," similar to looking up a name in the telephone book. That is, starting at both ends of the list, you examine the entry halfway in between and, if not matched, split either the upper half or the lower half for the next search. You'll quickly reach the item you're looking for (in log2(n) steps) where you'll find the corresponding record

number. Fetch and display this record at the console, just as we have done in the program shown above.

At this point you're just getting started. With a little more work, you can allow a fixed grouping size which differs from the 128-byte record shown above. This is accomplished by keeping track of the record number as well as the byte offset within the record. Knowing the group size, you randomly access the record containing the proper group, offset to the beginning of the group within the record read sequentially until the group size has been exhausted.

Finally, you can improve QUERY considerably by allowing boolean expressions which compute the set of records which satisfy several relationships, such as a LASTNAME between HARDY and LAUREL, and an AGE less than 45. Display all the records which fit this description. Finally, if your lists are getting too big to fit into memory, randomly access your key files from the disk as well.

```
1: ;
2: ;*****************************************************************************
3: ;*                                                                           *
4: ;* Sample Random Access Program for CP/M-86                                  *
5: ;*                                                                           *
6: ;*****************************************************************************
7: ;
8: ;       BDOS Functions
9: ;
10: coninp      equ     1               ;console input function
11: conout      equ     2               ;console output function
12: pstring     equ     9               ;print string until '$'
13: rstring     equ     10              ;read console buffer
14: version     equ     12              ;return version number
15: openf       equ     15              ;file open function
16: closef      equ     16              ;close function
17: makef       equ     22              ;make file function
18: readr       equ     33              ;read random
19: writer      equ     34              ;write random
20: ;
21: ;       Equates for non graphic characters
22: cr          equ     0dh             ;carriage return
23: lf          equ     0ah             ;line feed
24: ;
25: ;
26: ;       load SP, ready file for random access
27: ;
```

```
28:             cseg
29:             pushf                   ;push flags in CCP stack
30:             pop     ax              ;save flags in AX
31:             cli                     ;disable interrupts
32:             mov     bx,ds           ;set SS register to base
33:             mov     ss,bx           ;set SS, SP with interrupt
34:             mov     sp,offset stack         ;      for 8088
35:             push    ax              ;restore the flags
36:             popf
37: ;
38: ;           CP/M-86 initial release returns the file
39: ;           system version number of 2.2: check is
40: ;           shown below for illustration purposes.
41: ;
42:             mov     cl,version
43:             call    bdos
44:             cmp     al,20h          ;version 2.0 or later?
45:             jnb     versok
46:             ;       bad version, message and go back
47:             mov     dx,offset badver
48:             call    print
49:             jmp     abort
50: ;
51: versok:
52: ;           correct version for random access
53:             mov     cl,openf        ;open default fct
54:             mov     dx,offset fcb
```

```
55:                 call        bdos
56:                 inc         al              ;err 255 becomes zero
57:                 jnz         ready
58: ;
59: ;               cannot open file, so create it
60:                 mov         cl,makef
61:                 mov         dx,offset fcb
62:                 call        bdos
63:                 inc         al              ;err 255 becomes zero
64:                 jnz         ready
65: ;
66: ;               cannot create file, directory full
67:                 mov         dx,offset nospace
68:                 call        print
69:                 jmp         abort           ;back to ccp
70: ;
71: ;       loop back to "ready" after each command
72: ;
73: ready:
74: ;               file is ready for processing
75: ;
76:                 call        readcom         ;read next command
77:                 mov         ranrec,dx       ;store input record#
78:                 mov         ranovf,0h       ;clear high byte if set
79:                 cmp         al,'Q'          ;quit?
80:                 jnz         notq
81: ;
```

```
82: ;               quit processing, close file
83:                 mov     cl,closef
84:                 mov     dx,offset fcb
85:                 call    bdos
86:                 inc     al              ;err 255 becomes 0
87:                 jz      error           ;error message, retry
88:                 jmps    abort           ;back to ccp
89: ;
90: ;
91: ;       end of quit command, process write
92: ;
93: ;
94: notq:
95: ;               not the quit command, random write?
96:                 cmp     al,'W'
97:                 jnz     notw
98: ;
99: ;               this is a random write, fill buffer until cr
100:                mov     dx,offset datmsg
101:                call    print           ;data prompt
102:                mov     cx,127          ;up to 127 characters
103:                mov     bx,offset buff  ;destination
104: rloop:         ;read next character to buff
105:                push    cx              ;save loop control
106:                push    bx              ;next destination
107:                call    getchr          ;character to AL
108:                pop     bx              ;restore destination
```

```
109:            pop     cx              ;restore counter
110:            cmp     al,cr           ;end of line?
111:            jz      erloop
112: ;          not end, store character
113:            mov     byte ptr [bx],al
114:            inc     bx              ;next to fill
115:            loop    rloop           ;decrement cx ..loop if
116: erloop:
117: ;          end of read loop, store 00
118:            mov     byte ptr [bx],0h
119: ;
120: ;          write the record to selected record number
121:            mov     cl,writer
122:            mov     dx,offset fcb
123:            call    bdos
124:            or      al,al           ;error code zero?
125:            jz      ready           ;for another recor
126:            jmps    error           ;message if not
127: ;
128: ;
129: ;
130: ;          end of write command, process read
131: ;
132: ;
133: notw:
134: ;          not a write command, read record?
135:            cmp     al,'R'
```

```
136:            jz          ranread
137:            jmps        error           ;skip if not
138: ;
139: ;          read random record
140: ranread:
141:            mov         cl,readr
142:            mov         dx,offset fcb
143:            call        bdos
144:            or          al,al           ;return code 00?
145:            jz          readok
146:            jmps        error
147: ;
148: ;          read was successful, write to console
149: readok:
150:            call        crlf            ;new line
151:            mov         cx,128          ;max 128 characters
152:            mov         si,offset buff  ;next to get
153: wloop:
154:            lods        al              ;next character
155:            and         al,07fh         ;mask parity
156:            jnz         wloop1
157:            jmp         ready           ;for another command if
158: wloop1:
159:            push        cx              ;save counter
160:            push        si              ;save next to get
161:            cmp         al,' '          ;graphic?
162:            jb          skipw           ;skip output if not graphic
```

```
163:                call        putchr          ;output character
164: skipw:
165:                pop         si
166:                pop         cx
167:                loop        wloop           ;decrement CX and check
168:                jmp         ready
169: ;
170: ;
171: ;      end of read command, all errors end-up here
172: ;
173: ;
174: error:
175:                mov         dx,offset errmsg
176:                call        print
177:                jmp         ready
178: ;
179: ;      BDOS entry subroutine
180: bdos:
181:                int         224             ;entry to BDOS if by INT
182:                ret
183: ;
184: abort:                                     ;return to CCP
185:                mov         cl,0
186:                call        bdos            ;use function 0 to end e
187: ;
188: ;      utility subroutines for console i/o
189: ;
```

```
getchr:
        ;read next console character to a
        mov     cl,coninp
        call    bdos
        ret
;
putchr:
        ;write character from a to console
        mov     cl,conout
        mov     dl,al           ;character to send
        call    bdos            ;send character
        ret
;
crlf:
        ;send carriage return line feed
        mov     al,cr           ;carriage return
        call    putchr
        mov     al,lf           ;line feed
        call    putchr
        ret
;
print:
        ;print the buffer addressed by dx until $
        push    dx
        call    crlf
        pop     dx              ;new line
        mov     cl,pstring
```

```
217:                call        bdos            ;print the string
218:                ret
219: ;
220: readcom:
221:                ;read the next command line to the conbuf
222:                mov         dx,offset prompt
223:                call        print           ;command?
224:                mov         cl,rstring
225:                mov         dx,offset conbuf
226:                call        bdos            ;read command line
227: ;              ;command line is present, scan it
228:                mov         ax,0            ;start with 0000
229:                mov         bx,offset conlin
230: readc:         mov         dl,[bx]         ;next command character
231:                inc         bx              ;to next command position
232:                mov         dh,0            ;zero high byte for add
233:                or          dl,dl           ;check for end of command
234:                jnz         getnum
235:                ret
236: ;              not zero, numeric?
237: getnum:
238:                sub         dl,'0'
239:                cmp         dl,10           ;carry if numeric
240:                jnb         endrd
241:                mov         cl,10
242:                mul         cl              ;multiply accumulator by
243:                add         ax,dx           ; + digit
```

```
244:                jmps        readc           ;for another char
245: endrd:
246: ;              end of read, restore value in a and return value
247:                mov         dx,ax           ;return value in DX
248:                mov         al,-1[bx]
249:                cmp         al,'a'          ;check for lower case
250:                jnb         transl
251:                ret
252: transl:        and         al,5fH          ;translate to upper case
253:                ret
254: ;
255: ;
256: ;      Template for Page 0 of Data Group
257: ;      Contains default FCB and DMA buffer
258: ;
259:                dseg
260:                org         05ch
261: fcb            rb          33              ;default file control block
262: ranrec         rw          1               ;random record position
263: ranovf         rb          1               ;high order (overflow) b
264: buff           rb          128             ;default DMA buffer
265: ;
266: ;      string data area for console messages
267: badver         db                          'sorry, you need cp/m version 2$'
268: nospace        db                          'no directory space$'
269: datmsg         db                          'type data: $'
270: errmsg         db                          'error, try again.$'
```

```
271: prompt     db                      'next command? $'
272: ;
273: ;
274: ;      fixed and variable data area
275: ;
276: conbuf     db          conlen      ;length of console buffer
277: consiz     rs          1           ;resulting size after read
278: conlin     rs          32          ;length 32 buffer
279: conlen     equ         offset      $-offset consiz
280: ;
281:            rs          31          ;16 level stack
282: stack      rb          1
283:            db          0           ;end byte for GENCMD
284:            end
```

APPENDIX F. LIGHT PEN AND ESCAPE CODE SEQUENCES

ight Pen and Escape Code Sequences

This appendix describes the operation of the light pen and the format of the data it sends to an application program. This appendix also describes the escape code sequences you can use in a program to control the cursor, change colors on your display, and assign logical device input and output.

ight Pen

If a program requests input from the Light Pen and there is no data waiting, CP/M-86 displays a status line message: "Waiting for Light Pen Input". When CP/M-86 obtains a data set from the Pen, it resets the status line and places the data into the console status stream.

The Light Pen sends data to the console input (or auxiliary input) stream in the following format:

Byte 1 – Character Row + 20H
Byte 2 – Character Column + 20H
Byte 3 – Dot Row + 20H
Byte 4 – Most Significant 6 bits of Dot Column + 20H
Byte 5 – Least Significant 6 bits of Dot Column + 20H
Byte 6 – Terminator, always 00 (ASCII NUL)

To avoid misinterpreting characters, the driver converts all binary values to graphic ASCII characters by adding 20H (32 decimal) to each byte. The character row, column and the raster scan line values can be obtained by simply subtracting 20H from the respective Light Pen input bytes. The dot column is more difficult to obtain because it is actually a 12-bit binary value which has been broken into two bytes. The first byte contains the most significant 6 bits of the dot column, offset by adding 20H as in the case of the first three bytes from the Light Pen. The second byte contains the least significant 6 bits of the dot column, also offset by adding 20H. The dot column can be obtained from these two bytes in the following manner: first, subtract 20H from the most significant byte and then shift the result left six places forming a 16-bit value; and second, subtract 20H from the least significant byte and add it to the 16-bit value from the first step. This technique will reconstruct the dot column value from the input bytes.

Escape Code Sequences

You can use Escape codes to control the cursor, change display colors, redirect logical input and output, and program the Function keys and Cursor keypad keys. An ASCII Escape character (hex 1B) triggers Escape sequence processing. The Escape character can be followed by one or more characters depending on the function required. The character immediately following the Escape character indicates which function is to be performed. You must send Escape character sequences directly to the operating system, for example, through an assembly language routine. When Escape sequences are read from the input stream, it is necessary to read two characters for each Escape sequence. You cannot send the Escape character sequence through the Console Command Processor (CCP). The following table summarizes the Escape codes and their functions.

ESC a	Set Console Mode
ESC b	Set Foreground Color
ESC c	Set Background Color
ESC d	Redirect Console Input
ESC e	Redirect Console Output
ESC f	Redirect Auxiliary Input
ESC g	Redirect Auxiliary Output
ESC h	Redirect List Output
ESC i	Enable/Disable Transparent Mode
ESC j	Save Cursor Position
ESC k	Restore Cursor Position
ESC l	Enable/Disable Console Status Mode
ESC A	Cursor Up
ESC B	Cursor Down
ESC C	Cursor Forward
ESC D	Cursor Backward (non-destructive)
ESC E	Clear Screen (and Home Cursor)
ESC H	Home Cursor
ESC K	Clear to End of Line
ESC Y	Position Cursor
ESC /	Set Color Palette
ESC ?	Get Time, Date, Background Message
ESC :	Program Function/Cursor Keys
ESC ESC	Display Escape Graphics Character

ESC a–Set Console Mode

The console mode selects the number of rows and columns on the CRT screen, as well as Color or Monochrome display. Escape-a must be followed by a number from 0 to 7. This number selects a mode according to the table below:

0 – 40 × 25 Black & White on the Color Board
1 – 40 × 25 Color
2 – 80 × 25 Black & White on the Color Board
3 – 80 × 25 Color
4 – 320 × 200 Color Graphics
5 – 320 × 200 Monochrome Graphics
6 – 640 × 200 Monochrome Graphics
7 – 80 × 25 Monochrome (IBM Monochrome Display)

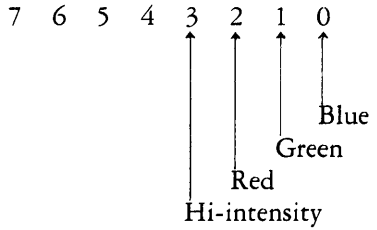
Only the least significant three bits of the mode number are used.

ESC b–Set Foreground Color

The Foreground Color displays the character. Associated with the foreground color is an intensity selection bit, although many color monitors do not support high and low intensity characters.

Escape-b must be followed by a color selection character. Only the four least significant bits of the color character are used, with the individual bits having the following significance:

Bit Pattern of Control Byte:



Here are some examples of color selection:

Sample Byte Values for Various Colors:

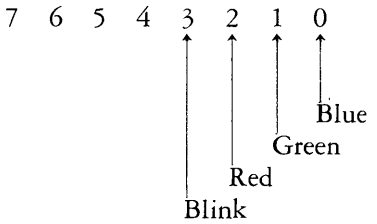
0 – Black (Used with Non-black Backgrounds)
1 – Blue
2 – Green
3 – Blue + Green (Cyan)
4 – Red
5 – Red + Blue (Magenta)
6 – Red + Green (Yellow)
7 – Red + Green + Blue (White)

ESC c–Set Background Color

This function selects Background Color, the color of the screen "behind" the characters. In addition, this function can make individual characters blink on and off.

Escape-c must be followed by a color selection character. Only the four least significant bits of the color character are used, with the individual bits having the following significance:

Bit Pattern of Control Byte:



The background color selection characters are the same as for foreground. Note that White Background combined with Black Foreground is effectively Inverse Video.

To combine color selection with blink, use the following characters:

h – Black
i – Blue
j – Green
k – Blue + Green (Cyan)
l – Red
m – Red + Blue (Magenta)
n – Red + Green (Yellow)
o – Red + Green + Blue (White)

ESC d,e,f,g,h–I/O Redirection

These Escape Sequences can redirect input and output between logical and physical devices. The function letter (d, e, f, g, or h) must be followed by two bytes. CP/M-86 uses the most significant seven bits of each byte. Therefore, you should set the most significant bit to 1 so that CP/M-86 never mistakes the values you are outputting for carriage return or tab characters.

To understand how the I/O redirection works you must view the two bytes as a set of bits. Each bit is associated with a physical device driver. When you are specifying Input Source (such as Console Input–ESC d) you can specify only a single physical device. Output Destinations (such as Console Output–ESC e) can have several output devices.

The bit values for each of the physical devices are shown below. Note that Byte 1 is output after the selection letter; Byte 2 follows Byte 1.

Byte 1		Byte 2		Physical Device
Binary	*Hex*	*Binary*	*Hex*	
10000001	81H	10000000	80H	Keyboard
10000010	82H	10000000	80H	Screen
10000100	84H	10000000	80H	Serial Port #0
10001000	88H	10000000	80H	Serial Port #1
10010000	90H	10000000	80H	Printer #0
10100000	A0H	10000000	80H	Printer #1
11000000	C0H	10000000	80H	Printer #2
10000000	80H	10000001	81H	Light Pen
10000000	80H	10000010	82H	Reserved for Cassette I/O
10000000	80H	10000100	84H	Reserved for Game Card I/O
10000000	80H	10001000	88H	Dummy Device

The function selection letters are:

d – Console Input
e – Console Output
f – Auxiliary Input
g – Auxiliary Output
h – List Output

ESC i–Enable/Disable Transparent Mode

When transparent mode is enabled, the following characters are output to the screen without special processing:

Carriage Return
Line Feed
Backspace
Bell

When transparent mode is disabled (which is the default state), these characters cause their appropriate function to occur. Enable transparent mode when you want the special symbols assigned to these characters to appear on the screen.

The least signficant bit of the character following the "i" enables or disables transparent mode. The recommended values are:

0 – Disable transparent mode
1 – Enable transparent mode

ESC j–Save Cursor Position

This sequence preserves the current cursor position. You can restore the cursor to the previously saved position with Escape-k.

ESC k–Restore Cursor Position

This sequence restores the cursor to a previously saved position. If you use this sequence without having previously saved the cursor position, then the cursor is moved "home" to the top left-hand corner of the screen.

ESC l–Enable/Disable Console Status Mode

This sequence enables or disables the special feature in the console status routine that alters whether CONST reports any keyboard characters waiting. When the console status mode is enabled, CONST reports only that logical input is waiting for physical keyboard input. When disabled, CONST reports not only keyboard input, but also input from internal character strings such as function keys or the light pen.

Programs that poll the console to check for incoming data characters simply do not work if you have enabled console status mode.

ESC A–Cursor Up

This moves the cursor up one line. If the cursor is already on the top line of the screen, this Escape sequence has no effect.

ESC B–Cursor Down

This moves the cursor down one line. If the cursor is already on the last line of the screen, that is, the one above the status line, then this Escape sequence has no effect.

ESC C–Cursor Forward

This moves the cursor one position to the right. If this function would move the cursor off the screen, this Escape sequence has no effect.

ESC D–Cursor Backward

This moves the cursor one position to the left. This is a "non-destructive" move because the character over which the cursor now rests is not replaced by a blank. If the cursor is already in column 0, this Escape sequence has no effect.

ESC E–Clear Screen (and Home Cursor)

This moves the cursor to column 0, row 0 (the top left-hand corner of the screen), and clears all characters from the screen.

ESC H–Home Cursor

This moves the cursor to column 0, row 0. The screen is NOT cleared.

ESC K–Clear to End of Line

This clears the line from the current cursor position to the end of the line.

ESC Y–Position Cursor

The two characters that follow the "Y" specify the row and column to which the cursor is to be moved. The first character specifies the row, the second specifies the column. Rows number from 0 to 23 (24 being the status line), column numbers from 0 to 79.

To avoid confusing row and column values with control characters, row and column values have 20H (32 decimal) added to them. For example, to move the cursor to the home position (0,0), the two characters following the "Y" would be ASCII spaces (20H).

ESC /–Set Color Palette

The character following the "/" sets the color palette for the display. CP/M-86 uses the least significant 7 bits of this character.

ESC ?–Get Time, Date, Background Message

This Escape sequence causes a string of characters to be injected into the console input stream. The exact character sequence is shown below:

MM/DD/YY,HH:MM:SS, ...blanks...

The character string is exactly as shown above; the commas and carriage return are included in the data stream.

Note: The interrupt handler in the ROM BIOS provides approximately 18.2 timer interrupts per second. CP/M-86 has reprogrammed the 8253 timer to provide approximately 19 ticks per second.

ESC : –Program Function Keys

This sequence programs the function keys, F1 to F10, and the cursor control keys on the number pad. The overall format of this escape sequence is:

ESC : kid string 00H

"kid" is a key indentifier and tells CP/M-86 which function key/cursor control key you want to program. "string" is an arbitrary string of characters; for function keys this can be up to 18 characters long. For cursor control keys this can be up to 4 characters. "00H" is a byte of hexadecimal 0 and terminates the string.

The valid key identifiers and their default string settings are shown below. Note that the symbol "<cr>" represents the Enter key.

; – F1	dir<cr>
< – F2	dir b:<cr>
= – F3	stat<cr>
> – F4	stat b:<cr>
? – F5	pip<cr>
@ – F6	pip b:=a:*.*[v]
A – F7	stat *.*<cr>
B – F8	stat b:*.*<cr>
C – F9	(not programmed)
D – F10	(not programmed)
G – Home	ESC H (Home)
H – Up Arrow	ESC A (Cursor Up)
I – Page Up	(not programmed)
K – Left Arrow	ESC D (Cursor Left)
M – Right Arrow	ESC C (Cursor Right)
O – End	'END'
P – Down Arrow	ESC B (Cursor Down)
Q – Page Down	(not programmed)
R – Ins	(not programmed)
S – Del	DEL (ASCII delete)

INDEX

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

X

Personal Computer
Computer Language Series

Product Comment Form

CP/M™-86

6936616

Your comments assist us in improving our products. IBM may use and distribute any of the information you supply in anyway it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Comments:

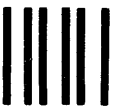
If you wish a reply, provide your name and address in this space.

Name_____

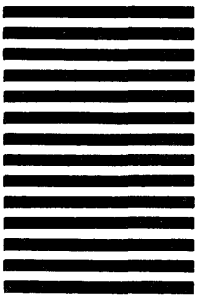
Address _____

City _____ State_____

Zip Code _____



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 123 BOCA RATON, FLORIDA 33432

POSTAGE WILL BE PAID BY ADDRESSEE

IBM PERSONAL COMPUTER
SALES & SERVICE
P.O. BOX 1328-C
BOCA RATON, FLORIDA 33432

Fold here

Continued from inside front cover

SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM STATE TO STATE.

IBM does not warrant that the functions contained in the program will meet your requirements or that the operation of the program will be uninterrupted or error free.

However, IBM warrants the diskette(s) or cassette(s) on which the program is furnished, to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of delivery to you as evidenced by a copy of your receipt.

LIMITATIONS OF REMEDIES

IBM's entire liability and your exclusive remedy shall be:

1. the replacement of any diskette(s) or cassette(s) not meeting IBM's "Limited Warranty" and which is returned to IBM or an authorized IBM PERSONAL COMPUTER dealer with a copy of your receipt, or
2. if IBM or the dealer is unable to deliver a replacement diskette(s) or cassette(s) which is free of defects in materials or workmanship, you may terminate this Agreement by returning the program and your money will be refunded.

IN NO EVENT WILL IBM BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE SUCH PROGRAM EVEN IF IBM OR AN AUTHORIZED IBM PERSONAL COMPUTER DEALER HAS BEEN ADVISED OF THE POSSIBLITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY.

SOME STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU.

GENERAL

You may not sublicense, assign or transfer the license or the program except as expressly provided in this Agreement. Any attempt otherwise to sublicense, assign or transfer any of the rights, duties or obligations hereunder is void.

This Agreement will be governed by the laws of the State of Florida.

Should you have any questions concerning this Agreement, you may contact IBM by writing to IBM Personal Computer, Sales and Service, P.O. Box 1328-W, Boca Raton, Florida 33432.

YOU ACKNOWLEDGE THAT YOU HAVE READ THIS AGREEMENT, UNDERSTAND IT AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. YOU FURTHER AGREE THAT IT IS THE COMPLETE AND EXCLUSIVE STATEMENT OF THE AGREEMENT BETWEEN US WHICH SUPERSEDES ANY PROPOSAL OR PRIOR AGREEMENT, ORAL OR WRITTEN, AND ANY OTHER COMMUNICATIONS BETWEEN US RELATING TO THE SUBJECT MATTER OF THIS AGREEMENT.

International Business Machines Corporation

P.O. Box 1328-W
Boca Raton, Florida 33432

6936616

Printed in United States of America