Chapter I

CP/M-86

User's Guide

Canon AS-100 Series

CP/M-86™ User's Guide

Digital Research
P.O. Box 579
160 Central Avenue
Pacific Grove, CA 93950
(408) 649-3896
TWX 910 360 5001

COPYRIGHT

DISCLAIMER

TRADEMARKS

Foreword

CP/M-86™ is an operating system designed by Digital Research for the 8086 and 8088 sixteen bit microprocessor. CP/M-86 is distributed with its accompanying utility programs on two eight-inch single sided, single density floppy disks.

CP/M-86 file structure is compatible with the file structure of Digital Research's CP/M® operating system for computers based on the 8080 or Z80® microprocessor chips. This means that if the disk formats are the same, as in standard single density format, CP/M-86 can read the same data files as CP/M. The system calls are as close to CP/M as possible to provide a familiar assembly language programming environment. This allows application programs to be easily converted to execute under CP/M-86.

The minimum hardware requirement for CP/M-86 consists of a computer system based on an 8086 or 8088 microprocessor, 32K (kilobytes) of random access memory, a keyboard and a screen device, and generally, two eight-inch floppy disk drives with diskettes. The CP/M-86 operating system itself, excluding the utility programs supplied with it, uses approximately 12 kilobytes of memory. To run DDT-86™, you must have 48K of memory, and to run ASM-86™ and many of the application programs that run under CP/M-86, you must have 64K of memory.

If you expand your system beyond these minimums, you will appreciate that CP/M-86 supports many other features you can add to your computer. For example, CP/M-86 can support up to one megabyte of Random Access Memory (RAM), the maximum allowed by your 8086 or 8088 microprocessor. CP/M-86 can support up to sixteen logical disk drives of up to eight megabytes of storage each, allowing up to 128 megabytes of on-line storage.

This manual introduces you to CP/M-86 and tells you how to use it. The manual assumes your CP/M-86 system is up and running. (The interface between the hardware and the software must be configured in the Basic Input Output System (BIOS) according to the instructions in the CP/M-86 System Guide.) The manual also assumes you are familiar with the parts of your computer, how to set it up and turn it on, and how to handle, insert and store disks. However, it does not assume you have had a great deal of experience with computers.

Section 1 tells how to start CP/M-86, enter a command and make a back-up disk. Section 2 discusses disks and files. Section 3 develops the CP/M-86 command concepts you need to understand the command summary in Section 4. The command summary describes all of the user programs supplied with CP/M-86.

Section 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.

Appendix A supplies an ASCII to Hexadecimal conversion table. Appendix B lists the filetypes associated with CP/M-86. Appendix C lists the CP/M-86 Control Characters. Appendix D lists the messages CP/M-86 displays when it encounters special conditions. If the condition requires correction, Appendix D can also tell you what actions you should take before you proceed. Appendix E provides a simple glossary of commonly used computer terms for the convenience of the user.

The more complex programs are described in the CP/M-86 Programmer's and System Guides. ASM-86 is the CP/M-86 assembler for your computer. You won't need ASM-86 until you decide to write assembly language programs and become more familiar with your computer's 8086 or 8088 microprocessor instruction set. When you do, you'll find that ASM-86 simplifies writing 8086 or 8088 microprocessor programs. DDT-86 is 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.

Table of Contents

Table of Contents (continued)

Table of Contents
(continued)

Appendixes

Section 1
Introduction

This section discusses the fundamentals of your computer and CP/M-86. It describes CP/M-86 start-up procedures and initial messages. Then it shows you how to enter a CP/M-86 command and make a back-up copy of your CP/M-86 distribution disk.

CP/M-86 manages information stored magnetically on disks by grouping this information into files of programs or data. CP/M-86 can copy files from a disk to your 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. You can use CP/M-86 to create your own CP/M-86 programs, or you can choose from the wide variety of CP/M-86 application programs that entertain, educate, and solve commercial and scientific problems.

1.1 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 distribution system disk into your computer's memory. For AS-100 series you can start CP/M-86 under the following procedure.

Starting with five-inch mini-floppy disk drive:

- insert five-inch CP/M-86 system disk into drive A (the lower drive)
- close the drive door
- turn on the power of the main unit

This automatically loads CP/M-86 into memory.

Starting with eight-inch standard floppy disk drive:

- turn on the power of eight-inch floppy disk unit
- insert eight-inch CP/M-86 system disk into drive A (the lefthand side drive)
- close the drive door
- turn on the power of the main unit

This automatically loads CP/M-86 into memory.

If power is on and you want to restart CP/M-86, first make sure your CP/M-86 system disk is in drive A and turn off the power of the main unit, and then turn on the power again after ten seconds or so. Or press the system reset switch (the left-side hole of the lower side of the display) with the like pen. This causes restarting CP/M-86. This is called System Restart, or "booting the system".

At System Reset, 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:

```
Canon AS-100 CP/M-86 Version V.V
Copyright (C) 1981, Digital Research Inc.

BIOS (A) Vn.mm by Canon 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. Vn.mm indicates the version number of BIOS presented by Canon.

This display is followed by the two character message:

```
A>
```

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

Note: Your AS-100 CP/M-86 system disk may execute several commands automatically under the control of the START.SUB function before the system prompt message is displayed. See the volume four AS-100 CP/M-86 user's guide for more detail.

1.2 The Command Line

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 a carriage return keystroke. The carriage return key might be marked RETURN or CR on your particular terminal. The command keyword identifies a command (program) to be executed by the AS-100. The command tail can contain extra information for the command such as a filename, option or parameter. To end the command line, you must press the ENTER Key or RETURN (↵) Key.

As you type characters at the keyboard, they appear on your screen and the cursor (position indicator) moves to the right. If you make a typing mistake, press the DEL key or CTRL-H characters to 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 disk as program files. They must be loaded into memory to perform their task. You can recognize Transient Utility program files in a disk'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 disk 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 that are stored on the disk in drive A. For example, if you have your CP/M-86 system disk in drive A, these filenames, among others, appear on your screen:

```
COPYDISK CMD
PIP      CMD
STAT     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 accidently mistype the DIR command, CP/M-86 responds:

```
A>DJR
DJR?
```

to tell you that it can not find 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 disk, 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?
```

1.3 CP/M-86 Line Editing Control Characters

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

Table 1-1. Control Character Commands

Command	Meaning
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.
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 effect as a 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 a ENTER keystroke.
CTRL-R	displays a # at the current cursor location, moves the cursor to the next line and redisplays any partial command you have typed so far.
CTRL-U	discards all the characters in the command line that you've typed so far, displays 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 - same as a DELETE-LINE key.

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.

Note: The DEL key and CTRL-H have the same function but the different movement of the cursor occurs on the display. When you type CTRL-H, the letters before the cursor are deleted and the cursor moves to the left position. When you press the DEL key the letters before the cursor are displayed again (echoback) and the cursor moves to the right position.

1.4 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 accidently 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 disk. The choice is usually made based on the number of files on the disk that need to be backed up. It might take less than a minute to make a copy of one file, but it only takes two or three minutes to copy an entire disk.

Note: Other than these two ways that presented by the standard CP/M-86 system, AS-100 CP/M-86 prepares high-speed back-up command "VOL COPY". See the volume four AS-100 CP/M-86 user's guide for more detail.

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

1.5 How to Make a Copy of Your CP/M-86 Disk

To back-up your CP/M-86 disk, you will use one or more floppy disks for the back-ups, the COPYDISK Transient Utility program, and of course your CP/M-86 disk.

The back-up disks can be factory-fresh or used. Some eight-inch disks come with a notch cut out of the lower right hand side. This notch prevents data from being written to the disk. It is called a "write-protect" notch. To copy data to these disks, you have to "write-enable" them by placing a small foil tab over the write-protect notch. These tabs are supplied with the disks.

Note: Five-inch disks have a notch on the upper righthand side. Placing a small foil tab in the opposite way of the eight-inch disks, you have "write-protect" status.

You might want to format new or reformat used disks with the disk formatting program (FORMAT, see volume four AS-100 CP/M-86 user's guide). If the disks are used, make sure they do not contain any information you might need again! COPYDISK copies everything from a source disk to a destination disk - including blank space - and writes over any information that might already be stored on the destination disk.

To make a copy of your CP/M-86 disk, use the COPYDISK utility. First make sure that your system disk is in drive A and a formatted disk is inserted in drive B. Then enter the following command to the system prompt, terminated by a carriage return keystroke.

A>**COPYDISK**

CP/M-86 loads COPYDISK into memory and runs it. COPYDISK displays the following messages on your screen:

```
CP/M-86 Full Disk COPY Utility
      Version 2.0

Enter Source Disk Drive (A-P) ?A

Destination Disk Drive (A-P) ?B

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

Copy started
Reading Track 0...
Copy completed.

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

A>
```

Now you have an exact copy of the original CP/M-86 disk 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. Use it as your CP/M-86 system disk to make more back-ups, to try the examples shown in the rest of this manual and to start CP/M-86 the next time you power up your computer.

Section 2
Files, Disks, Drives and Devices

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

2.1 What is a File?

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

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 can also 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, the executable copy program PIP can copy one or more command program files.

2.2 How 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.

2.3 Naming Files - What's in a Name?

CP/M-86 identifies every file by its unique file specification. A file specification (filespec) can have three parts:

d:	drive specifier	one character	optional
filename	filename	1-8 characters	
typ	filetype	0-3 characters	optional

We recommend that you create file specifications from letters and numbers. Because the CP/M-86 command processor recognizes the following special characters as delimiters (separators), they must not be included within a filename or filetype.

< > . , ; : = ? * [ ]

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

MYFILE

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 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 an optional one to three character family name, called a filetype, to the filename. When you add a filetype to the filename, separate the filetype from the filename with a period. Try to use three letters that tell something about the file's family. For example, you could add the following filetype to the file that contains a list of customer names:

CUSTOMER.NAM

When CP/M-86 displays file specifications in response to a DIR command, it fills in short filenames and filetypes with blanks so that you can compare filetypes quickly.

The executable program files that CP/M-86 loads into memory from a disk have different filenames, but are in the family of 8086 or 8088 programs that run with CP/M-86. The filetype CMD identifies this family of executable programs.

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

Table 2-1. CP/M-86 Filetypes

Filetype	Meaning
CMD	8086 or 8088 Machine Language Program
$$$	Temporary File
A86	ASM-86 Source File
H86	Assembled ASM-86 Program in hexadecimal format
SUB	List of commands to be executed by SUBMIT

2.4 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 disk 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 disk 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 disk in drive B instead. For this reason, CP/M-86 lets you to preceed a filename with a drive specifier which is the drive letter followed by a colon. For example, in response to the command

A>**dir b:myfile.lib**

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

You can also preceed an executable program filename with a drive specifier, even if you are using the program filename as a command keyword. For example, if you type the following command

A>**b:pip**

CP/M-86 looks in the directory of the disk 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 that are stored in the disk directory, the drive specifier for a file changes as you move the disk from one drive to another. Therefore a file has a different file specification when you change its disk from one drive to another.

2.5 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 disk 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 disk with the following six files:

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 disk directory, and therefore wildcards are valid only in filenames and filetypes. You cannot use a wildcard in a drive specifier.

2.6 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 disk'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. Unless you use the command program PIP to copy the file to another user number, the file is assigned the "current" user number. You can use the Built-in command USER to display and change the current user number.

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. With the [Gn] option, PIP can copy a file with one user number and give the copy another user number.

File attributes control how a file can be accessed. There are two kinds of file accessing attributes. The DIR/SYS attribute can be set to either DIR (Directory) or SYS (System). When you create a file, it is automatically marked with the DIR attribute. The DIR command only displays files that are in the current user area, whether that is user number 0,1,2,3 or 15.

You can use the STAT Transient Utility command to assign the SYS or DIR attribute to a file. The DIR command does not display files that are marked with the SYS attribute. You must use the DIRS command to display SYS files. Remember that DIRS only displays the system files that are in the current user number. The STAT command also displays files marked with the SYS attribute. Again, STAT displays files from the current user number only.

It is very useful to assign the SYS attribute to files that are in user number 0. They should be command files, files with a filetype of CMD. If you give a command file in 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, without having to maintain a copy of each command program in every user number.

The RW/RO file accessing attribute can be set to either RW (Read-Write) or RO (Read-Only). A file with the RW attribute can be read or written to at any time unless the disk is write-protected, or the drive containing the disk is set to Read-Only. If a file is marked RO, any attempt to write data to that file produces a Read-Only error message. Therefore you can use the RO attribute to protect important files.

You can use the STAT Transient Utility program to assign the Read-Write or Read-Only attribute to a file or group of files. STAT can also assign the Read-Only attribute to a drive. CTRL-C resets all logged-in drives to Read-Write.

2.7 How Are Files Stored on a Disk?

CP/M-86 records the filename, filetype, user number and attributes of each file in a special area of the disk called the directory. In the directory, CP/M-86 also records which disk sectors belong to which file. The directory is large enough to store this data for up to sixty-four files.

Note: AS-100 CP/M-86 five-inch and eight-inch floppy disks use double-sided and double-density media. The directory is up to one hundred and twenty-eight files. The eight-inch, single-sided and single-density media, the directory is up to sixty-four files - the standard CP/M-86.

CP/M-86 allocates directory and storage space for a file as records are added to the file. 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.

2.8 Changing Disks

CP/M-86 cannot, of course, do anything to a file unless the disk that holds the file is inserted into a drive and the drive is in ready status. When a disk 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 disk out of a drive and insert another that contains different files. You can replace an on-line disk whenever you see the system prompt at your console. However, if you are going to write on the disk, you must tell CP/M-86 that you have changed a disk by typing CTRL-C directly after the system prompt. In response, CP/M-86 resets the drive for the new disk.

If you forget to type CTRL-C after you change a disk, CP/M-86 automatically protects the new disk. You can run a text editor or copy program and try to write to the new disk, but when you do, CP/M-86 notices that the original disk is no longer in the drive and writes the message:

```
Bdos err on d: RO
```

where d: is the drive specifier of the new disk. 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 disk.

2.9 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 specifier in your command line, CP/M-86 and the utilities look in the directory of the disk 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 disk is the default drive. Then you will know which files an application program can access if you do not add a drive specifier.

Drive A is usually the default drive when you start CP/M-86. If you have more than one drive, you might want to change the default drive. Do this by typing the drive specifier of the desired default drive next to the system prompt and pressing the RETURN 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 disk in drive B for any file that does not have a drive specifier included in the file specification.

2.10 More CP/M-86 Drive Features

Under CP/M-86, drives can be marked RO just as files can be given the RO attribute. The default state of a drive is RW, but CP/M-86 marks a drive RO whenever you change the disk in the drive. You can give a drive the RO attribute by using the STAT Transient Utility described in Section 4. To return the drive to RW you must type a CTRL-C to the system prompt.

2.11 Other CP/M-86 Devices

CP/M-86 manages all the peripheral devices attached to your computer. These can include storage devices such as disk drives, input devices such as keyboards, or modems, and output devices such as printers, modems, and screens.

To keep track of input and output devices, CP/M-86 uses "logical" devices. The table below shows CP/M-86 logical device names and indicates whether the device is input or output.

Table 2-2. CP/M-86 Logical Devices

Device Name	Device Type
CON:	Console input and output
AXI:	Auxiliary input
AXO:	Auxiliary output
LST:	List output

CP/M-86 associates physical devices with the 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 STAT command to assign an alternate peripheral to the logical device name. For example, a STAT command can change the console input device from the keyboard to a teletype. STAT can assign a printer to the LST: logical output device name.

A logical input device can be assigned only one physical device. A logical output device can be assigned only one physical device. See the description of the STAT command in Section 4 for more detail.

Note: See the volume four AS-100 CP/M-86 user's guide for the description of the input/output device of AS-100 options.

Section 3
CP/M-86 Command Concepts

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

3.1 Two Types of Commands

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

You can add utilities to your system by purchasing various CP/M-86-compatible application programs. If you are an experienced programmer, you can also write your own utilities that operate with CP/M-86.

Note: Some Transient Utilities are included with AS-100 CP/M-86. See the volume four.

3.2 Built-In Commands

Built-in commands are part of CP/M-86 and are always available for your use regardless of which disks 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. Section 4 gives you the operating details for the Built-in commands listed in the table below.

Table 3-1. Built-In Commands

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

Table 3-1. (continued)

Command	Meaning
REN	lets you rename a file.
TYPE	writes the content of a character file at your screen.
USER	lets you change from one user number to another.

3.3 Transient Utility Commands

A program that executes a Transient Utility command comes into memory only when you request it. Section 5 gives you operating details for the standard CP/M-86 Utilities listed in the table below.

Table 3-2. CP/M-86 Utilities

Command	Meaning
ASM86	translates 8086 assembly language programs into machine code form.
COPYDISK	creates a copy of a disk that can 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 programs.
GENCMD	uses the output of ASM-86 to produce an executable command file.
HELP	displays information on how to use each CP/M-86 command.
PIP	combines and copies files.
STAT	lets you examine and alter file and disk status, and assign physical I/O devices to CP/M-86 logical devices.
SUBMIT	sends a file of commands to CP/M-86 for execution.
TOD	sets and displays the system date and time.

3.4 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 drive 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.BAS**

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 remains operational 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 one of four errors has occurred:

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

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

A>**EDIT MYPROG.BAS**

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 disk, 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.

3.5 Control Character 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.

Table 3-3. Control Character Commands

Command	Meaning
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 RW status.
CTRL-P	echoes all console activity at the printer; a second CTRL-P ends printer echo. This only works if your system is connected to a 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.

Section 4
Command Summary

This section 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 syntax notation. The remainder of the section 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. For example, ED is described in Section 5 while ASM-86, DDT-86 and GENCMD are described in the CP/M-86 System Guide.

4.1 Let'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 disk drive, A, B, C, or D that contains the file or group of files to which you are referring. If a drive specifier 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 specifier, "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 in the following list.

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

If you do not include a drive specifier, CP/M-86 automatically supplies the default drive. If you omit the period and the filetype, CP/M-86 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 disk 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 syntax statements.

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

    B:MY*.A??

is a file specification with drive-specifier "B:", filename "MY*", and filetype "A??". This file specification might 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 specifier, filename, and filetype. These concepts give you the background necessary to compose complete command lines.

4.2 How Commands Are Described

This section 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 "Syntax" 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 disk 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. To clarify examples of interactions between the user and the operating system, the characters entered by the user are shown in **boldface.** CP/M-86's responses are shown in normal type.

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

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

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

- You can substitute a number for n.

- The symbolic notation "d:", "filename", ".typ" and "filespec" have 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 Syntax line, a vertical bar | separates the alternatives. Think of this vertical bar as the "or" bar.

- An up-arrow ↑ or CTRL represent the Control Key on your keyboard.

- All other punctuation must be included in the command line.

Let's look at some examples of syntax 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 (RO) or Read-Write (RW), and the System (SYS) or Directory (DIR) attributes to a file.

The Syntax section of the STAT command shows how the command line syntax notation is used:

Syntax:

```
STAT { filespec  {RO | RW | DIR | SYS } }
     |           |                     | |
     |           ----------optional------ |
    --------------- optional -------------
```

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 the following:

```
RO
RW
DIR
SYS
```

Therefore,

```
STAT filespec RO
```

is a valid command.

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

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

```
STAT
STAT X.A86
STAT X.A86 RO
STAT X.A86 SYS
STAT *.A86
STAT *.* RW
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.

Syntax:

```
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 syntax. Some of them are shown below.

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

4.3 The ASM-86 (Assembler) Command

Syntax:

ASM86 filespec { $parameter-list }

Type:

Transient Utility

Purpose:

The ASM-86 Utility converts 8088 and 8086 assembly language source statements into machine code form.

The operation of the ASM-86 assembler is described in detail in the CP/M-86 Programmer's Guide.

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 specifier 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 through P, corresponding to the 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 specifier to obtain the source file. The valid parameters are AA through AP.

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

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

Use the S parameter letter to override the default drive specifier to receive the SYM file. Valid parameters are SA through SP, 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.ASM $PX**

The ASM86.CMD file must be on drive B. The source file X.ASM 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.

4.4 The COPYDISK (Copy Disk) Command

Syntax:

COPYDISK

Type:

Transient Utility

Purpose:

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

Before copying to a brand-new disk, you might first have to prepare it with the disk formatting program that should accompany your computer. If you copy to a used disk, COPYDISK writes all the information from the source disk over the information on the destination disk, including blank space.

Note: See the volume four for the disk formatting program.

Remarks:

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 disk is not write-protected. Check that there is a foil tab covering any existing write-protect notch on the edge of your disk before inserting the disk into the destination drive.

Note: If the foil tab covers a notch on the five-inch mini-floppy disk of AS-100 system, it is write-protected.

COPYDISK is an exact track-for-track, sector-for-sector copy utility, and is the fastest way to copy an entire disk. However, if many files have been created and erased on the source disk, the records belonging to a particular file might be randomly placed on the disk. In this case, it might 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 disk.

Note: AS-100 CP/M-86 supplies high-speed media copy program VOL COPY. See the volume four AS-100 CP/M-86 user's guide for more detail.

Examples:

A>**COPYDISK**

Invoke COPYDISK and it prompts you for the source and destination disk. In our next example, COPYDISK copies from your master disk (disk A:) to the new disk (disk B:). When invoked, COPYDISK displays the information in the first line of our example:

```
CP/M-86 Full Disk Copy Utility
     Version 2.0

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
opy 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>
```

4.5 The DDT-86 (Dynamic Debugging Tool) Command

Syntax:

DDT86 { filespec }

Type:

Transient Utility

Purpose:

The DDT-86 Utility allows you to monitor and test programs developed for the 3086 and the 8088 processors.

The DDT-86 single letter commands, their parameters and options are described in Table 4-1. The actual command letter is printed in boldface. The parameters are in lower-case and follow the command letter. Optional items are in curly brackets. Replace the arguments with the appropriate values as described in the following list Table 4-1.

Table 4-1. DDT-86 Commands

Command	Meaning	
As	(Assemble)	Enter Assembly Language Statements
Bs,f,sl	(Block Cmp)	Compare Blocks of Memory
D{W}{s{,f}}	(Display)	Display Memory in Hex and ASCII
Efilespec	(Execution)	Load Program for Execution
Fs,f,bc	(Fill)	Fill Memory Block - Byte
FWs,f,wc	(Fill)	Fill Memory Block - Word
G{s}{,bl{,b2}}	(Go)	Begin Execution
Hwcl,wc2	(Hex)	Hexadecimal Sum and Difference
Icommand tail	(Input)	Set Up Input Command Line
L{s{,f}}	(List)	List Memory in Mnemonic Form
Ms,f,d	(Move)	Move Memory Block
Rfilespec	(Read)	Read Disk File to Memory
S{W}s	(Set)	Set Memory Values

Table 4-1. (continued)

Command		Meaning
T{n}	(Trace)	Trace Program Execution
TS{n}	(Trace)	Trace and Show All Registers
U{n}	(Untrace)	Monitor Execution without Trace
US{n}	(Untrace)	Monitor and Show All Registers
V	(Verify)	Show Memory Layout after Disk Read
Wfilespec{,s,f}	(Write)	Write Content of Block to Disk
X{r}	(Examine)	Examine and Modify CPU Registers

Parameter	Replace with
bc	byte constant
b1	breakpoint one
b2	breakpoint two
d	destination for data
f	final address
n	number of instructions to execute
r	register or flag name
s	starting address
s1	second starting address
W	word 16-bit
wc	word constant

The overall operation of DDT-86, along with each single letter command, is described in detail in the CP/M-86 Programmer's Guide.

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.

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 where the file was loaded and the "-" prompt.

4.6 The DIR (Directory) Built-in

Syntax:

```
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 disk. 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 in 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 disk in the default drive and current user number. Similarly, DIRS displays the SYS files.

If the drive specifier 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 disk in the specified drive. DIRS displays the SYS files.

If the file specification contains wildcard characters, all filenames 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 disk 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 specifed drive, DIRS displays the message:

    NON-SYSTEM FILES(S) EXIST

You cannot use a wildcard character in a drive specifier.

Examples:

A>**DIR**

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

A>**DIR B:**

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

A>**DIR B:X.A86**

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

A>**DIR *.BAS**

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

B>**DIR A:X*.C?D**

All DIR files in the current user number on the disk 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 in the current user number on drive A with a filetype of CMD that have the system (SYS) attribute.

4.7 The ED (Character File Editor) Command

Syntax:

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

Type:

    Transient Utility

Purpose:

The ED Utility lets you create and edit a disk 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.

Table 4-2. ED Command Summary

Command	Action
nA	append n lines from original file to memory buffer
0A	append file until buffer is one half full
#A	append file until buffer is full (or end of file)

Table 4-2. (continued)

Command	Action
B, -B	move CP to the beginning (B) or bottom (-B) of buffer
nC, -nC	move CP n characters forward (C) or back (-C) through buffer
nD, -nD	delete n characters before (-D) or from (D) the CP
E	save new file and return to CP/M-86
Fstring{↑Z}	find character string
H	save the new file, then reedit, using the new file as the original file
I	enter insert mode; use ↑Z to exit insert mode
Istring{↑Z}	insert string at CP
Jsearch_str^Zins_str^Zdel_to_str{↑Z}	juxtapose strings
nK, -nK	delete (kill) n lines from the CP
nL, -nL, 0L	move CP n lines
nMcommands	execute commands n times
n, -n	move CP n lines and display that line
n:	move to line n
:ncommand	execute command through line n
Nstring{↑Z}	extended find string

Table 4-2. (continued)

Command	Action
O	return to original file
nP, -nP	move CP 23 lines forward and display 23 lines at console
Q	abandon new file, return to CP/M-86
R	read X$$$$$$$.LIB file into buffer
Rfilespec{↑Z}	read filespec into buffer
Sdelete string^Zinsert string{↑Z}	substitute string
nT, -nT, 0T	type n lines
U, -U	upper-case translation
V, -V, 0V	line numbering on/off, display free buffer space
nW	write n lines to new file
nX	write or append n lines to X$$$$$$$.LIB
nXfilespec{↑Z}	write n lines to filespec or append if previous x command applied to the same file
0X	delete file X$$$$$$$.LIB
0Xfilespec{↑Z}	delete filespec
nZ	wait n seconds

Section 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 specifier 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 it's 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.

4.8 The ERA (Erase) Built-in

Syntax:

    ERA filespec

Type:

    Built-in

Purpose:

The ERA Built-in removes one or more files from the directory of a disk. 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 disk 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 disk in drive A.

A>**ERA *.PRN**

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

B>**ERA A:MY*.***

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

A>**ERA B:*.***

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

All (Y/N)?

4.9 The GENCMD (Generate CMD File) Command

Syntax:

```
GENCMD filespec {8080 CODE[An,Bn,Mn,Xn] DATA[An,Bn,Mn,Xn]
                STACK[An,Bn,Mn,Xn] EXTRA[An,Bn,Mn,Xn] X1[...]}
```

Type:

Transient Utility

Purpose:

The GENCMD Utility uses the hex output of ASM-86 and other language processors to produce a CMD file. An optional parameter list follows the file specification.

You need to know how to use GENCMD when you write assembly language programs that become Transient Utility commands.

The operation of GENCMD is described in detail in the CP/M-86 System Guide.

The parameter-list consists of up to nine keywords with a corresponding list of values. The keywords are:

8080 CODE DATA STACK EXTRA X1 X2 X3 X4

The keyword 8080 identifies the CMD file as an "8080 Memory Model" where code and data groups overlap. The remaining keywords define segment groups that 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 converted from 8-bit microprocessors to CP/M-86. The programs load 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** MYPROG

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.

4.10 The HELP (Help) Command

Syntax:

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

A>**HELP STAT**

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

A>**HELP STAT OPTIONS**

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

A>**HELP ED**

The command above displays general information about the ED Utility.

A>**HELP ED COMMANDS**

This form of HELP displays information about commands internal to ED.

CP/M-86 is distributed with two related HELP files: HELP. CMD and HELP.HLP. The HELP.CMD file is the command file that processes the text of the HELP.HLP file and displays it on the screen. The HELP.HLP file is a text file to which you can add customized information, but you cannot edit the HELP.HLP file. You must use the HELP.CMD file to convert HELP.HLP to a file named HELP.DAT before you can edit or add your own text.

Use the following forms of the HELP command to change HELP.HLP to HELP.DAT and change HELP.DAT back to HELP.HLP.

HELP [E]

HELP [C]

The HELP [E] command accesses the file HELP.HLP on the default drive, removes the header record, and creates a file called HELP.DAT on the default drive. You can now invoke a word-processing program to edit or add your own text to the HELP.DAT file.

The HELP [C] command accesses your edited HELP.DAT file on the default drive, generates a new index for the entries record, and buids a revised HELP.HLP file on the default drive. HELP.CMD can now display your new HELP.HLP file.

You must add topics and subtopics to the HELP.DAT file in a specific format. The general format of a topic heading in the HELP.DAT file is shown below.

///nTOPICNAME<cr>

The three back slashes are the topic delimiters and must begin in column one. In the format statement above, n is a number in the range from 1 through 9 that signifies the level of the topic. A main topic always has a level number of 1. The first subtopic has a level number of 2. The next subtopic has a level number of 3, and so forth up to a maximum of nine levels. TOPIC-NAME is the name of your topic, and allows a maximum of twelve characters. The entire line is terminated with a carriage return.

Use the following guidelines to properly edit and insert text into the HELP.DAT file.

- Topics should be ordered in ascending alphabetical order.
- Subtopics should be ordered in ascending alphabetical order within their respective supertopic.
- Levels must be indicated by a number 1 - 9.

Some examples of topic and subtopic lines in the HELP.HLP file are shown below.

```
///1NEW UTILITY<cr>

///2COMMANDS<cr>

///3EXAMPLES<cr>
```

The first example shown above illustrates the format of a main topic line. The second example shows how to number the first subtopic of that main topic. The third example shows how the next level subtopic should be numbered. Any topicname with a level number of 1 is a main topic. Any topicname with a level number of 2 is a subtopic within its main topic.

When you are executing the HELP.CMD file, you need only enter enough letters of the topic to unambiguously identify the topic name. When referencing a subtopic, you must type the topic name.and the subtopic, otherwise the HELP program cannot determine which main topic you are referencing. You can also enter a topic and subtopic following the program's internal prompt, HELP>, as shown below.

```
HELP>ED COMMANDS
```

This form of HELP displays information about commands internal to the editing program, ED.

4.11 PIP (Peripheral Interchange Program - Copy File) Command

Syntax:

```
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.

4.11.1 Single File Copy

Syntax:

```
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 the the location specified in dest-filespec, and gives it the name indicated by dest-filespec.

Remember that PIP always "goes to" and "gets from" the current user number unless you specify otherwise with the [Gn] option.

Remarks:

Before you start PIP, be sure that you have enough free space in kilobytes on your destination disk to hold the entire file or files that you are copying. Even if you are replacing an old copy on the destination disk 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 disk. 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, RW, RO) are transferred with the files.

If the existing destination file is set to Read-Only (RO), 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 PIP Options, below). 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 Command. 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 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. Since there is no drive specified for the destination, PIP automatically copies the file to the default user number and drive, in this case drive B.

4.11.2 Multiple File Copy

Syntax:

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 filename as the copy operation proceeds. PIP issues an error message and aborts 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 command above 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:*.A86
```

This command causes PIP to copy all the files with a filetype of A86 on drive A in the default user number (user ZERO 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.)

4.11.3 Combining Files

Syntax:

```
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,K,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 only searches 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-file character in 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 disk 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.

4.11.4 Copy Files to and from Auxiliary Devices

Syntax:

```
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 can be assigned to 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 abort 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 or Output 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 zeroes.
EOF:	A source device that produces a single CTRL-Z, (The CP/M-86 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 CRT: is assigned to CON: 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 CRT: is assigned to CON: 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.

4.11.5 Multiple Command Mode

Syntax:

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 ENTRY key following the asterisk prompt. The empty command line tells PIP to discontinue operation and return to the CP/M-86 system prompt.

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.

4.11.6 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 [].

Table 4-3. PIP Options

Option	Function
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.

Table 4-3. (continued)

Option	Function
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.

Table 4-3. (continued)

Options	Function
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.

Table 4-3. (continued)

Option	Function
W	Write over files with RO (Read-Only) attribute. Normally, if a PIP command tail includes an existing RO 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 RO 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 re-inserted 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[VWR]**

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 was correctly transferred. The W parameter lets PIP overwrite any destination files that are marked as RO (Read-Only). The R parameter tells PIP to read files from drive A that are marked with the SYS (System) attribute.

4.12 The REN (Rename) Built-in

Syntax:

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 disk.

The filename oldname identifies an existing file on the disk. The filename newname is not in the directory of the disk. 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 specifier 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 specifier, 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**

4.13 The STAT (Status) Command

Syntax:

```
STAT
STAT  d:=RO
STAT filespec [RO|RW|SYS|DIR|SIZE}
STAT [d:}DSK: | USR:
STAT VAL: | DEV:
```

Type:

Transient Utility

Purpose:

The various forms of the STAT Utility command give you information about the disk drives, files and devices associated with your computer. STAT lets you change the attributes of files and drives. You can also assign physical devices to the STAT logical device names.

Note that the options following filespec can be enclosed in square brackets [], or be preceeded by a dollar $ sign or by no delimiter as shown in the syntax section above.

Remarks:

The notation "RW" tells you the drive is in a Read-Write state so that data can be both read from and written to the disk.

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

Drives are in a Read-Write state by default, and become Read-Only when you set the drive to RO or when you change a disk and forget to type a CTRL-C.

4.13.1 Set a Drive to Read-Only Status

Syntax:

```
STAT d:= RO
```

Purpose:

You can use this form of the STAT command to set the drive to Read-Only status. Use CTRL-C to reset a drive to Read-Write.

Example:

A>**STAT B:= RO**

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

4.13.2 Free Space on Disk

Syntax:

STAT {d:}

Purpose:

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

Examples:

A>**STAT**

Suppose you have two disk drives containing active disks. Suppose also that drive A has 16K (16,384) bytes of free space, while drive B has 32K (32,728) bytes of free space. Assume that drive A is marked RW, and drive B is marked RO. The STAT command displays the following messages on your screen:

```
A: RW, Free Space: 16K
B: RO, Free Space: 32K
```

A>**STAT B:**

Suppose drive B is set to Read-Only and has 98 Kilobytes of storage that is free for program and data storage. The following message is displayed on your screen:

B: RO, Free Space: 98K

4.13.3 Files - Display Space Used and Access Mode

Syntax:

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.

Note that the S option following the filespec can be enclosed in square brackets [], or be preceeded by a dollar $ sign, or by no delimiter as shown in the syntax line above.

CP/M-86 supports four file Access Modes:

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

RW 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 RO or RW attribute, and either the SYS or DIR attribute. By default, and unless changed by the STAT command, a file has the RW 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 specifier, 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:
User nn  d:filename.typ
```

STAT prints the user number and 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, and then type a CTRL-C.

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:

```
Drive B:                                     User 0
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 MYFILE.DAT was written using random access from record number 8 through 15, leaving the first 8 records empty. The virtual file size is 16 records, although the file only consumes eight records. The screen appears as follows:

```
Drive B:                                            User 0
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
```

4.13.4 Set File Access Modes (Attributes)

Syntax:

STAT filespec RO |RW |SYS |DIR

Purpose:

This form of the STAT command lets you set the Access Mode for one or more files. Note that the option following filespec can be enclosed in square brackets [], be preceeded by a dollar $ sign or by no delimiter as shown above.

The four Access Modes, described above, are:

RO
RW
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 RO or RW 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 RO**

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

LETTER.TXT set to RO

The command:

B>**STAT A:*.COM SYS**

sets the Access Mode for all files on drive A, with filetype COM, to SYS. Given that the three command files PIP, ED, and ASM-86 are present on drive A, the following message appears on your screen:

PIP.COM set to SYS
ED.COM set to SYS
ASM86 set to SYS

4.13.5 Display Disk Status

Syntax:

STAT {d:}DSK:

Purpose:

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

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.

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.

4.13.6 Display User Numbers With Active Files

Syntax:

STAT {d:}USR:

Purpose:

This form of the STAT command lets you determine the User Numbers that have files on the disk in the specified drive.

User Numbers are assigned to files that are created under CP/M-86. Use this form of the STAT command to determine the active User Numbers on a disk.

Examples:

A>**STAT USR:**

This command displays the User Numbers containing active files on the disk in drive A.

4.13.7 Display STAT Commands and Device Names

Syntax:

```
STAT VAL:
```

Purpose:

STAT VAL: displays the general form of the STAT commands. It also displays the possible physical device names that you can assign to each of the four CP/M-86 logical device names.

Examples:

The STAT VAL: display is shown below:

```
A>STAT VAL:
STAT 2.1

Read Only Disk: d:=RO
Set Attribute: d:filename.typ [ro] [rw] [sys] [dir]
Disk Status  : DSK: d:DSK:
User Status  : USR: d:USR:
Iobyte Assign:
CON: = TTY: CRT: BAT: UC1:
AXI: = TTY: PTR: UR1: UR2:
AXO: = TTY: PTP: UP1: UP2:
LST: = TTY: CRT: LPT: UL1:
A>
```

4.13.8 Display and Set Physical to Logical Device Assignments

Syntax:

```
STAT DEV:
STAT logical device: = physical device:
```

Purpose:

STAT DEV: displays the current assignments for the four CP/M-86 logical device names, CON:, AXI:, AXO: and LST:. Use the second form of the above STAT command to change these current assignments. The command STAT VAL: displays the possible physical device names that you can assign to each logical device name. Refer to the part of the STAT VAL: display entitled "Iobyte Assign" shown above.

When you assign a physical device to a logical device, STAT assigns a value from 0 to 3 to the logical device name in what is called the IObyte.

You can assign any of the listed physical device names to their appropriate logical device names. However, the assignment does not work unless you are using the proper Input-Output Port on your computer, with the proper cable to connect the computer to the device, and the proper IO (Input-Output) driver routine for the particular physical device.

The physical device drivers have to be implemented in the BIOS (Basic Disk Operating System). The IObyte must be read and interpreted. The appropriate drivers must be jumped to for the logical output routine. Refer to the CP/M-86 System Guide for further information on handling external physical devices.

Examples:

A>**STAT CON: = CRT:**

The command above assigns the physical device name CRT: to the logical input device name CON:, which generally refers to the console.

A>**STAT LST: = LPT:**

The command above assigns the physical device name LPT: to the logical output device name LST:, which generally refers to the list device of the printer.

4.14 The SUBMIT (Batch Processing) Command

Syntax:

    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. 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 1 through 9:

    $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, the second word replaces $2, and so-forth, through the last parameter.

If you type fewer items in the command tail than parameters in the SUB file, remaining parameters are removed from the command line.

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

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

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 has processed all command lines.

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.

<u>Examples:</u>

A>**SUBMIT SUBFILE**

Assume the file SUBFILE.SUB is on the disk 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 are assigned
                  $1 $2  $3  $4  <--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.

4.15 The TOD (Display and Set Time of Day) Command

Syntax:

    TOD {time-specification | P }

Type:

    Transient Utility

Purpose:

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

When you start CP/M-86, the date and time are set to the creation date of the BDOS. 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:

    month/day/year (weekday), hour:minute:second

For example, the screen might appear as:

    12/06/81 (WED), 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. 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 12/31/81 23:59:59**

This command sets the current date and time to the last second of 1981.

Note: In AS-100 system, when you have an internal timer as an option the current date and time are set automatically. If you don't have the internal timer, when you start CP/M-86 the date that the system is generated and 0 hour:0 minute: 0 second are set as an initial value.

4.16 The TYPE (Display File) Built-in

Syntax:

```
TYPE {d:}filename{.typ}
```

Type:

Built-in

Purpose:

The TYPE built-in displays the contents of a character file on your screen.

Remarks:

Tab characters occurring in the file named by the file specification 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 the file specification is not present on an on-line disk, 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 contents of the file MYPROG.A86 on your screen.

A>**TYPE B:THISFILE**

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

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

Syntax:

    USER { number }

Type:

    Built-in

Purpose:

The USER Built-in command displays and changes the current user number. The disk directory can be divided into distinct groups according to a "User Number."

Remarks:

When CP/M-86 starts, 0 is the current User Number. Any files you create under this User Number are not generally accessible under any other User Number except through the PIP command or the System (SYS) attribute as assigned with the STAT command. (See the G parameter of the PIP Utility.)

Use the command

    USER

to display the current User Number.

Use the command

    USER number

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

See the command

    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.

Section 5
ED, The CP/M-86 Editor

5.1 Introduction to ED

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 syntax 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.

5.2 Starting ED

Syntax:

```
ED filespec 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 specifier but a drive specifier is unnecessary if the file to be edited is on your default drive. Optionally, the file specification can be followed by a drive specifier, as shown in the following example.

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

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

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

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."

5.3 ED Operation

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

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

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

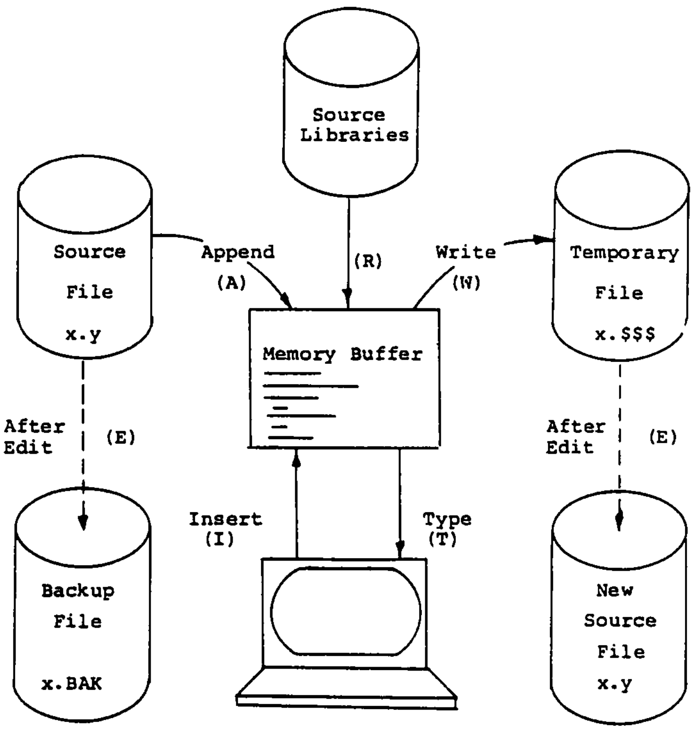


Figure 5-1. Overall ED Operation

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.

Table 5-1. Text Transfer Commands

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

5.3.1 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 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, than 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.

If 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.

5.3.2 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. You can determine the number of lines to write out by executing a 0V command to check the amount of free space in the buffer, as shown below:

```
1: *0V
25000/30000
1: *
```

The above display indicates that the total size of the memory buffer is 30,000 bytes and there are 25,000 free bytes in the memory buffer.

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 new file. If a .BAK file exists, ED deletes it, then renames the original file with the .BAK filetype. Finally, ED renames the new file from filename.$$$ to the original filetype and returns control to the CCP.

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.

5.4 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 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.

Table 5-2. Basic Editing Commands

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

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

5.4.1 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, 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.

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

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

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

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

5.4.2 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
 is joy enough."
```

If the CP is between two characters in the middle of the line, 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 a CTRL-Q when you're ready to continue scrolling. Enter a CTRL-C to abort long type-outs.

5.4.3 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.

5.4.4 Inserting Characters into the Memory Buffer

The I (Insert) Command

To insert characters into the memory buffer from the screen, use the I command. 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 command mode, you can use CP/M-86 line editing control characters to edit your input. The table below lists these control characters.

Table 5-3. CP/M-86 Line Editing Controls

Command	Result
CTRL-C	Abort the editor and return to the CP/M-86 system.
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.
DEL	Remove the last character and echo deleted character at the screen.

Note: in insert mode, the same line editing controls exist except for CTRL-C and CTRL-E.

When entering a combination of numbers and letters, you might find it inconvenient to press a caps-lock key if your terminal translates caps-locked 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>
```

5.4.5 Replacing Characters

<u>The S (Substitute) Command</u>

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. 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:

```
Emily 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.

5.5 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 carriage-return 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.

5.5.1 Moving the Character Pointer

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

Change the C command in this command string to move the CP more characters to the left. You can use this command string if you must make a change at the end of the line and you 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: *
```

5.5.2 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: *
```

5.5.3 Editing

To edit text and verify corrections quickly, combine the edit commands with other ED commands that move the CP and display text. Command strings like the one 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: *
```

5.6 Advanced ED Commands

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

5.6.1 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 only use this command with 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: *
```

5.6.2 Finding and Replacing Character Strings

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

The F (Find) Command

The F command performs the simplest find function. 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 only search from the CP to the bottom of the buffer. If you enter no number, ED finds the next occurrence of the string in the file. In the following example, the second occurrence of the word living is found.

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

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

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

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

```
1: *#T
1:  Emily Dickinson said,
2:  "I find ecstasy in living -
3:  the mere 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:  SORT    LXI    H, SW    ;ADDRESS TOGGLE SWITCH
236: *j;^ZADDRESS SWITCH TOGGLE^Z^L^Z0LT
236:  SORT    LXI    H, SW    ;ADDRESS SWITCH TOGGLE
236: *
```

In this example, ED searches for the first semicolon and inserts ADDRESS SWITCH TOGGLE after the mark and then deletes to the <cr><lf> sequence, represented by CTRL-L. (In any search string, you can use CTRL-L to represent a <cr><lf> when 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.

The M (Macro) Command

An ED macro command, M, can increase the usefulness of a string of commands. The M command allows you to group ED commands together for repeated execution. The 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 abort a macro command, strike a CTRL-C at the keyboard.

5.6.3 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 (Xfer) 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 temporary file; therefore, n must always be a positive number. If no filename is specified, X$$$$$$$ is assumed. If no filetype is specified, .LIB is assumed. 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
Rfilespec
```

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: *
```

5.6.4 Saving or Abandoning Changes: ED Exit

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

The H (Head of File) Command

An H command saves the contents of the memory buffer without ending the ED session, but it returns to the "head" of the file. It saves the current changes and lets you reedit the file without exiting ED. The 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 the following command:

A>**ED filespec differentfilespec**

If you then restart the edit with the H command, ED renames the file differentfilename.$$$ to differentfilename.BAK and creates a new file of differentfilespec when you finish editing.

The O (Original) Command

An O command abandons changes made since the beginning of the edit and allows you to return to the original source file and begin reediting without ending the ED session. The 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-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.

5.7 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.

Table 5-4. ED Error Symbols

Symbol	Meaning
#	Search failure. ED cannot find the string specified in an F, S, or N command.
?c	Unrecoginzed 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	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.

Section 1
CP/M-86 System Overview

1.1 CP/M-86 General Characteristics

CP/M-86 contains all facilities of CP/M-80 with additional features to account for increased processor address space of up to a megabyte (1,048,576) of main memory. Further, CP/M-86 maintains file compatibility with all previous versions of CP/M. The file structure of version 2 of CP/M is used, allowing as many as sixteen drives with up to eight megabytes on each drive. Thus, CP/M-80 and CP/M-86 systems may exchange files without modifying the file format.

CP/M-86 resides in the file CPM.SYS, which is loaded into memory by a cold start loader during system initialization. The cold start loader resides on the first two tracks of the system disk. CPM.SYS contains three program modules: the Console Command Processor (CCP), the Basic Disk Operating System (BDOS), and the user-configurable Basic I/O System (BIOS). The CCP and BDOS portions occupy approximately 10K bytes, while the size of the BIOS varies with the implementation. The operating system executes in any portion of memory above the reserved interrupt locations, while the remainder of the address space is partitioned into as many as eight non-contiguous regions, as defined in a BIOS table. Unlike CP/M-80, the CCP area cannot be used as a data area subsequent to transient program load; all CP/M-86 modules remain in memory at all times, and are not reloaded at a warm start.

Similar to CP/M-80, CP/M-86 loads and executes memory image files from disk. Memory image files are preceded by a "header record," defined in this document, which provides information required for proper program loading and execution. Memory image files under CP/M-86 are identified by a "CMD" file type.

Unlike CP/M-80, CP/M-86 does not use absolute locations for system entry or default variables. The BDOS entry takes place through a reserved software interrupt, while entry to the BIOS is provided by a new BDOS call. Two variables maintained in low memory under CP/M-80, the default disk number and I/O Byte, are placed in the CCP and BIOS, respectively. Dependence upon absolute addresses is minimized in CP/M-86 by maintaining initial "base page" values, such as the default FCB and default command buffer, in the transient program data area.

Utility programs such as ED, PIP, STAT and SUBMIT operate in the same manner under CP/M-86 and CP/M-80. In its operation, DDT-86 resembles DDT supplied with CP/M-80. It allows interactive debugging of 8086 and 8088 machine code. Similarly, ASM-86 allows assembly language programming and development for the 8086 and 8088 using Intel-like mnemonics.

The GENCMD (Generate CMD) utility replaces the LOAD program of CP/M-80, and converts the hex files produced by ASM-86 or Intel utilities into memory image format suitable for execution under CP/M-86. Further, the LDCOPY (Loader Copy) program replaces SYSGEN, and is used to copy the cold start loader from a system disk for replication. In addition, a variation of GENCMD, called LMCMD, converts output from the Intel LOC86 utility into CMD format. Finally, GENDEF (Generate DISKDEF) is provided as an aid in producing custom disk parameter tables. ASM-86, GENCMD, LMCMD, and GENDEF are also supplied in "COM" file format for cross-development under CP/M-80.

Several terms used throughout this manual are defined in Table 1-1 below:

Table 1-1. CP/M-86 Terms

Term	Meaning
Nibble	4-bit half-byte
Byte	8-bit value
Word	16-bit value
Double Word	32-bit value
Paragraph	16 contiguous bytes
Paragraph Boundary	An address divisible evenly by 16 (low order nibble 0)
Segment	Up to 64K contiguous bytes
Segment Register	One of CS, DS, ES, or SS
Offset	16-bit displacement from a segment register
Group	A segment-register-relative relocatable program unit
Address	The effective memory address derived from the composition of a segment register value with an offset value

A group consists of segments that are loaded into memory as a single unit. Since a group may consist of more than 64K bytes, it is the responsibility of the application program to manage segment registers when code or data beyond the first 64K segment is accessed.

CP/M-86 supports eight program groups: the code, data, stack and extra groups as well as four auxiliary groups. When a code, data, stack or extra group is loaded, CP/M-86 sets the respective segment register (CS, DS, SS or ES) to the base of the group. CP/M-86 can also load four auxiliary groups. A transient program manages the location of the auxiliary groups using values stored by CP/M-86 in the user's base page.

1.2 CP/M-80 and CP/M-86 Differences

The structure of CP/M-86 is as close to CP/M-80 as possible in order to provide a familiar programming environment which allows application programs to be transported to the 8086 and 8088 processors with minimum effort. This section points out the specific differences between CP/M-80 and CP/M-86 in order to reduce your time in scanning this manual if you are already familiar with CP/M-80. The terms and concepts presented in this section are explained in detail throughout this manual, so you will need to refer to the Table of Contents to find relevant sections which provide specific definitions and information.

Due to the nature of the 8086 processor, the fundamental difference between CP/M-80 and CP/M-86 is found in the management of the various relocatable groups. Although CP/M-80 references absolute memory locations by necessity, CP/M-86 takes advantage of the static relocation inherent in the 8086 processor. The operating system itself is usually loaded directly above the interrupt locations, at location 0400H, and relocatable transient programs load in the best fit memory region. However, you can load CP/M-86 into any portion of memory without changing the operating system (thus, there is no MOVCPM utility with CP/M-86), and transient programs will load and run in any non-reserved region.

Three general memory models are presented below, but if you are converting 8080 programs to CP/M-86, you can use either the 8080 Model or Small Model and leave the Compact Model for later when your addressing needs increase. You'll use GENCMD, described in Section 3.2, to produce an executable program file from a hex file. GENCMD parameters allow you to specify which memory model your program requires.

CP/M-86 itself is constructed as an 8080 Model. This means that all the segment registers are placed at the base of CP/M-86, and your customized BIOS is identical, in most respects, to that of CP/M-80 (with changes in instruction mnemonics, of course). In fact, the only additions are found in the SETDMAB, GETSEGB, SETIOB, and GETIOB entry points in the BIOS. Your warm start subroutine is simpler since you are not required to reload the CCP and BDOS under CP/M-86. One other point: if you implement the IOBYTE facility, you'll have to define the variable in your BIOS. Taking these changes into account, you need only perform a simple translation of your CP/M-80 BIOS into 8086 code in order to implement your 8086 BIOS.

If you've implemented CP/M-80 Version 2, you already have disk definition tables which will operate properly with CP/M-86. You may wish to attach different disk drives, or experiment with sector skew factors to increase performance. If so, you can use the new GENDEF utility which performs the same function as the DISKDEF macro used by MAC under CP/M-80. You'll find, however, that GENDEF provides you with more information and checks error conditions better than the DISKDEF macro.

Although generating a CP/M-86 system is generally easier than generating a CP/M-80 system, complications arise if you are using single-density floppy disks. CP/M-86 is too large to fit in the two-track system area of a single-density disk, so the bootstrap operation must perform two steps to load CP/M-86: first the bootstrap must load the cold start loader, then the cold start loader loads CP/M-86 from a system file. The cold start loader includes a LDBIOS which is identical to your CP/M-86 BIOS with the exception of the INIT entry point. You can simplify the LDBIOS if you wish because the loader need not write to the disk. If you have a double-density disk or reserve enough tracks on a single-density disk, you can load CP/M-86 without a two-step boot.

To make a BDOS system call, use the reserved software interrupt #244. The jump to the BDOS at location 0005 found in CP/M-80 is not present in CP/M-86. However, the address field at offset 0006 is present so that programs which "size" available memory using this word value will operate without change. CP/M-80 BDOS functions use certain 8080 registers for entry parameters and returned values. CP/M-86 BDOS functions use a table of corresponding 8086 registers. For example, the 8086 registers CH and CL correspond to the 8080 registers B and C. Look through the list of BDOS function numbers in Table 4-2. and you'll find that functions 0, 27, and 31 have changed slightly. Several new functions have been added, but they do not affect existing programs.

One major philosophical difference is that in CP/M-80, all addresses sent to the BDOS are simply 16-bit values in the range 0000H to 0FFFFH. In CP/M-86, however, the addresses are really just 16-bit offsets from the DS (Data Segment) register which is set to the base of your data area. If you translate an existing CP/M-80 program to the CP/M-86 environment, your data segment will be less than 64K bytes. In this case, the DS register need not be changed following initial load, and thus all CP/M-80 addresses become simple DS-relative offsets in CP/M-86.

Under CP/M-80, programs terminate in one of three ways: by returning directly to the CCP, by calling BDOS function 0, or by transferring control to absolute location 0000H. CP/M-86, however, supports only the first two methods of program termination. This has the side effect of not providing the automatic disk system reset following the jump to 0000H which, instead, is accomplished by entering a CONTROL-C at the CCP level.

You'll find many new facilities in CP/M-86 that will simplify your programming and expand your application programming capability. But, we've designed CP/M-86 to make it easy to get started: in short, if you are converting from CP/M-80 to CP/M-86, there will be no major changes beyond the translation to 8086 machine code. Further, programs you design for CP/M-86 are upward compatible with MP/M-86, our multitasking operating system, as well as CP/NET-86 which provides a distributed operating system in a network environment.

Section 2
Command Setup and Execution Under CP/M-86

This section discusses the operation of the Console Command Processor (CCP), the format of transient programs, CP/M-86 memory models, and memory image formats.

2.1 CCP Built-in and Transient Commands

The operation of the CP/M-86 CCP is similar to that of CP/M-80. Upon initial cold start, the CP/M sign-on message is printed, drive A is automatically logged in, and the standard prompt is issued at the console. CP/M-86 then waits for input command lines from the console, which may include one of the built-in commands

DIR ERA REN TYPE USER

(note that SAVE is not supported under CP/M-86 since the equivalent function is performed by DDT-86).

Alternatively, the command line may begin with the name of a transient program with the assumed file type "CMD" denoting a "command file." The CMD file type differentiates transient command files used under CP/M-86 from COM files which operate under CP/M-80.

The CCP allows multiple programs to reside in memory, providing facilities for background tasks. A transient program such as a debugger may load additional programs for execution under its own control. Thus, for example, a background printer spooler could first be loaded, followed by an execution of DDT-86. DDT-86 may, in turn, load a test program for a debugging session and transfer control to the test program between breakpoints. CP/M-86 keeps account of the order in which programs are loaded and, upon encountering a CONTROL-C, discontinues execution of the most recent program activated at the CCP level. A CONTROL-C at the DDT-86 command level aborts DDT-86 and its test program. A second CONTROL-C at the CCP level aborts the background printer spooler. A third CONTROL-C resets the disk system. Note that program abort due to CONTROL-C does not reset the disk system, as is the case in CP/M-80. A disk reset does not occur unless the CONTROL-C occurs at the CCP command input level with no programs residing in memory.

When CP/M-86 receives a request to load a transient program from the CCP or another transient program, it checks the program's memory requirements. If sufficient memory is available, CP/M-86 assigns the required amount of memory to the program and loads the program. Once loaded, the program can request additional memory from the BDOS for buffer space. When the program is terminated, CP/M-86 frees both the program memory area and any additional buffer space.

2.2 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 Table 2-1 below.

Table 2-1. 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 code 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 taken from CP/M-80 where code and data is easily separated. Note again that the code and data groups often consist of, but are not restricted to, single 64K byte segments.

The Compact Model occurs when any of the extra, stack, or auxiliary groups are present in program. Each group may consist of one or more segments, but if any group exceeds one segment 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.

2.3 The 8080 Memory Model

The 8080 Model is assumed when the transient program contains only a code group. 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, similar to CP/M-80, thus allowing base page values at the beginning of the code group. Following program load, the 8080 Model appears as shown in Figure 2-1, where low addresses are shown at the top of the diagram:

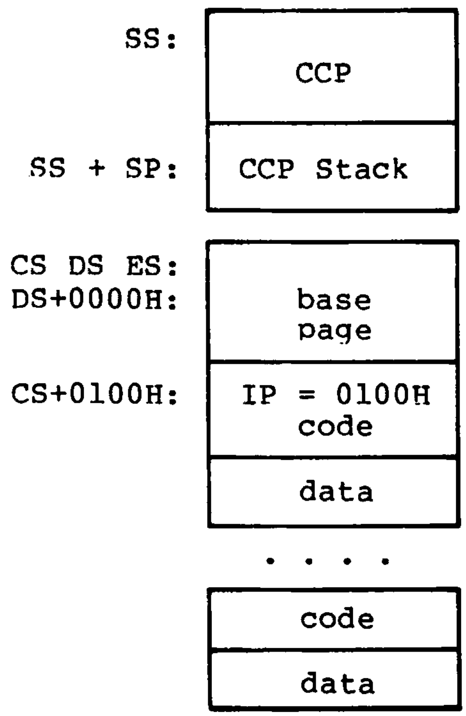


Figure 2-1. CP/M-86 8080 Memory Model

The intermixed code and data regions are indistinguishable. The "base page" values, described below, are identical to CP/M-80, allowing simple 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.

```
        eseg
        org     100h
        .
        .       (code)
endcs   equ     $
        dseg
        org     offset endcs
        .
        .     (data)
        end
```

2.4 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 Figure 2-2.

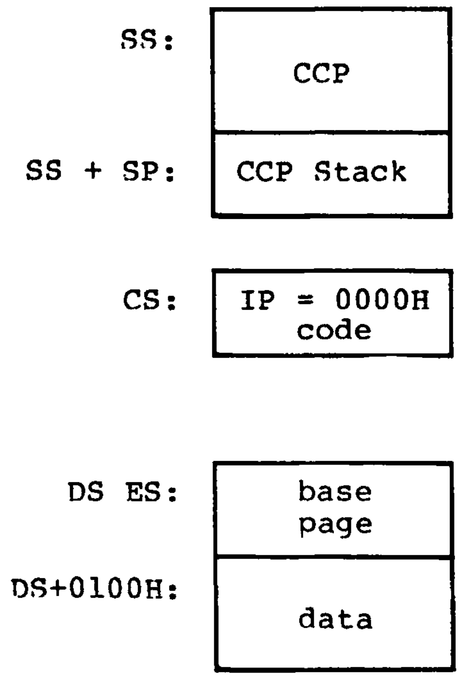


Figure 2-2. CP/M-86 Small Memory Model

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
```

2.5 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. Figure 2-3 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.

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
```

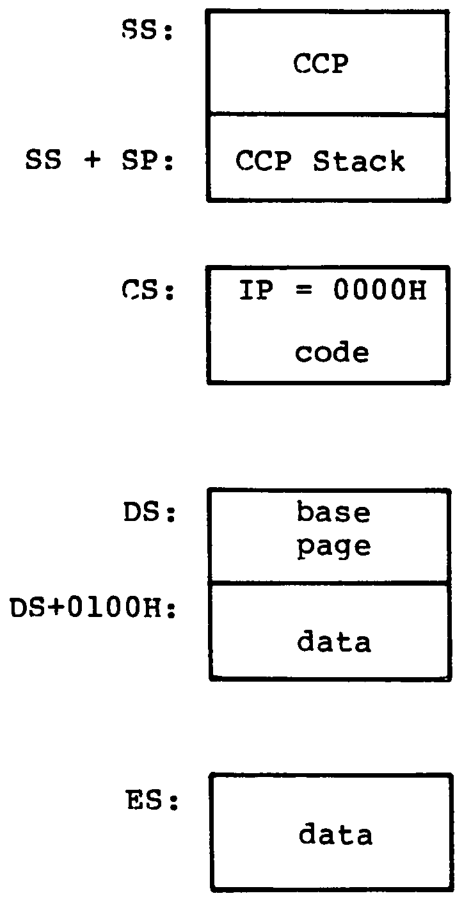


Figure 2-3. CP/M-86 Compact Memory Model

2.6 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 the regions from offset 0000H through 00FFH relative to the DS register. The values in the base page for CP/M-86 include those of CP/M-80, and appear in the same relative positions, as shown in Figure 2-4.

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: . . . DS + 005B:	Not Currently Used		
DS + 005C:	Default FCB		
DS + 0080:	Default Buffer		
DS + 0100:	Begin User Data		

Figure 2-4. 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 Figure 2-4 marks unused bytes. LC is the last code group location (24-bits, where the 4 high-order bits equal zero).

In the 8080 Model, the low order bytes of LC (LC0 and LC1) never exceed 0FFFFH and the high order byte (LC2) is always zero. BC is 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. It should be noted that bytes LD0 and LD1 appear in the same relative positions of the base page in both CP/M-80 and CP/M-86, thus easing the program translation task. 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 the following section.

2.7 Transient Program Load and Exit

Similar to CP/M-80, 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. Under CP/M-80, the default DMA address is initialized to 0080H in the base page. Due to the segmented memory of the 8086 and 8088 processors, the DMA address is divided into two parts: the DMA segment address and the DMA offset. Therefore, under CP/M-86, the default DMA base is initialized to the value of DS, and the default DMA offset is initialized to 0080H. Thus, CP/M-80 and CP/M-86 operate in the same way: both assume 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 optionally return directly to the CCP upon program termination by executing a "Far Return." Program termination also occurs when BDOS function zero is executed. Note that function zero can terminate a program without removing the program from memory or changing the memory allocation state (see Section 4.2). The operator may terminate program execution by typing a single CONTROL-C during line edited input which has the same effect as the program executing BDOS function zero. Unlike the operation of CP/M-80, no disk reset occurs and the CCP and BDOS modules are not reloaded from disk upon program termination.

Section 3
Command (CMD) File Generation

As mentioned previously, two utility programs are provided with CP/M-86, called GENCMD and LMCMD, which are used to produce CMD memory image files suitable for execution under CP/M-86. GENCMD accepts Intel 8086 "hex" format files as input, while LMCMD reads Intel L-module files output from the standard Intel LOC86 Object Code Locator utility. GENCMD is used to process output from the Digital Research ASM-86 assembler and Intel's OH86 utility, while LMCMD is used when Intel compatible developmental software is available for generation of programs targeted for CP/M-86 operation.

3.1 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 "MCS-86 Software Development Utitities 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:

:	l	l	a	a	a	a	t	t	d	d	d	. . .	d	c	c

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 in Table 3-1.

Table 3-1. Intel Hex Field Definitions

Field	Contents
ll	Record Length 00-FF (0-255 in decimal)
aaaa	Load Address
tt	Record Type: 00 data record, loaded starting at offset aaaa from current base paragraph 01 end of file, cc = FF 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) The following are output from ASM-86 only: 81 same as 00, data belongs to code segment 82 same as 00, data belongs to data segment 83 same as 00, data belongs to stack segment 84 same as 00, data belongs to extra segment 85 paragraph address for absolute code segment 86 paragraph address for absolute data segment 87 paragraph address for absolute stack segment 88 paragraph address for absolute extra segment
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 the ASM-86 User's Guide, and in Intel's document #9800821A entitled "MCS-86 Absolute Object File Formats.")

3.2 Operation of GENCMD

The GENCMD utility is invoked at the CCP 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  X1  X2  X3  X4
```

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.

- 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 64K segment, regardless of the use of CSEG and DSEG directives in the source program.

- 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.

- 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.

- The minimum memory value (M 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.

- 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] xl[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 and auxiliary segment #1 are uninitialized areas requiring a minimum of 400H bytes each. 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.

3.3 Operation of LMCMD

The LMCMD utility operates in exactly the same manner as GENCMD, with the exception that LMCMD accepts an Intel L-module file as input. The primary advantage of the L-module format is that the file contains internally coded information which defines values which would otherwise be required as parameters to GENCMD, such the beginning address of the group's data segment. Currently, however, the only language processors which use this format are the standard Intel development packages, although various independent vendors will, most likely, take advantage of this format in the future.

3.4 Command (CMD) File Format

The CMD file produced by GENCMD and LMCMD 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 in Figure 3-1.

```
+---------------------- 128 Bytes ----------------------+
| GD#1 | GD#2 | GD#3 | GD#4 | GD#5-GD#8. . .            |
+-------------------------------------------------------+
|     Code,                                             |
|          Data,                                        |
|               Extra,                                  |
|                    Stack,                             |
|                         Auxiliary                     |
+-------------------------------------------------------+
```

Figure 3-1. CMD File Header Format

In Figure 3-1, GD#2 through GD#8 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 9, as shown in Table 3-2 below.

Table 3-2. Group Descriptors

G-Type	Group Type
1	Code Group
2	Data Group
3	Extra Group
4	Stack Group
5	Auxiliary Group #1
6	Auxiliary Group #2
7	Auxiliary Group #3
8	Auxiliary Group #4
9	Shared Code Group
10 - 14	Unused, but Reserved
15	Escape Code for Additional Types

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. G-Type 9 marks a "pure" code group for use under MP/M-86 and future versions of CP/M-86. Presently a Shared Code Group is treated as a non-shared Program Code Group under CP/M-86.

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.

Section 4
Basic Disk Operating System 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.

4.1 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 and MP/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). Table 4-1 summarizes input and output parameter passing:

Table 4-1. 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 in Table 4-2 with an asterisk following functions which differ from or are added to the set of CP/M-80 Version 2 functions.

Table 4-2. 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
13	Reset Disk System	37*	Reset drive
14	Select Disk	40	Write Random with Zero Fill
15	Open File	50*	Direct BIOS Call
16	Close File	51*	Set DMA Segment Base
17	Search for First	52*	Get DMA Segment Base
18	Search for Next	53*	Get Max Memory Available
19	Delete File	54*	Get Max Mem at Abs Location
20	Read Sequential	55*	Get Memory Region
21	Write Sequential	56*	Get Absolute Memory Region
22	Make File	57*	Free memory region
23	Rename File	58*	Free all memory
		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.

4.2 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.

Entry		Return
CL: 00H	FUNCTION 0	
DL: Abort Code	SYSTEM RESET	

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.

Entry		Return
CL: 01H	FUNCTION 1	AL: ASCII Character
	CONSOLE INPUT	

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 (CONTROL-H) are echoed to the console. Tab characters (CONTROL-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.

Entry		Return
CL: 02H	FUNCTION 2	
DL: ASCII Character	CONSOLE OUTPUT	

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 (CONTROL-S).

```
     Entry                                      Return
  ---------->   +------------------+       ---------->
  CL: 03H       |  FUNCTION   3    |      AL: ASCII Character
                |                  |
                |  READER INPUT    |
                +------------------+
```

The Reader Input function reads the next character from the logical reader (READER) into register AL. Control does not return until the character has been read.

```
     Entry                                      Return
  ---------->   +------------------+       ---------->
  CL: 04H       |  FUNCTION   4    |
                |                  |
  DL: ASCII     |  PUNCH OUTPUT    |
      Character +------------------+
```

The Punch Output function sends the character from register DL to the logical punch device (PUNCH).

```
     Entry                                      Return
  ---------->   +------------------+       ---------->
  CL: 05H       |  FUNCTION   5    |
                |                  |
  DL: ASCII     |  LIST OUTPUT     |
      Character +------------------+
```

The List Output function sends the ASCII character in register DL to the logical list device (LIST).

Entry

CL: 06H

DL: 0FFH (input/ status)
or 0FEH (status)
or char (output)

FUNCTION 6

DIRECT CONSOLE I/O

Return

AL: char or status

(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., CONTROL-S and CONTROL-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 MP/M and CP/M.

Upon entry to Function 6, register DL contains either (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 returns the character in AL; otherwise Function 6 returns a zero in AL. If the input value is FE and no character is ready, then Function 6 returns AL = 00; otherwise, AL = FF. 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.

You cannot use Function 6 with FF or FE in combination with either Function 1 or Function 11. Function 1 is used in conjunction with Function 11. Function 6 must be used independently.

Entry

CL: 07H

FUNCTION 7

GET I/O BYTE

Return

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.

Entry		Return
CL: 08H	FUNCTION 8	
DL: I/O Byte Value	SET I/O BYTE	

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.

Entry		Return
CL: 09H	FUNCTION 9	
DX: String Offset	PRINT STRING	

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.

Entry		Return
CL: 0AH	FUNCTION 10	Console Characters
DX: Buffer Offset	READ CONSOLE BUFFER	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 (CONTROL-M) or a line feed (CONTROL-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 Table 4-3.

Table 4-3. Line Editing Controls

Keystroke	Result
rub/del	removes and echoes the last character
CONTROL-C	reboots when at the beginning of line
CONTROL-E	causes physical end of line
CONTROL-H	backspaces one character position
CONTROL-J	(line feed) terminates input line
CONTROL-M	(return) terminates input line
CONTROL-R	retypes the current line after new line
CONTROL-U	removes current line after new line
CONTROL-X	backspaces to beginning of current line

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

Entry		Return
CL: 0BH	FUNCTION 11 GET CONSOLE STATUS	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.

Entry		Return
CL: 0CH	FUNCTION 12 RETURN VERSION NUMBER	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 (BH = 01 for MP/M), 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.

4.3 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.
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 in passing 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 the following message to the console:

```
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 CONTROL-C terminates the executing program, while a RETURN instructs CP/M-86 to ignore the error and allow the program to continue execution.

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 disk media 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 aborted, and control returns to the CCP.

Entry: CL: 0DH

FUNCTION 13
RESET DISK SYSTEM

Return

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.

Entry: CL: 0EH
DL: Selected Disk

FUNCTION 14
SELECT DISK

Return

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 sixteen 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 which specify drive code zero (dr = 00H) automatically reference the currently selected default drive. Drive code values between 1 and 16, however, ignore the selected default drive and directly reference drives A through P.

Entry: CL: 0FH
DX: FCB Offset

FUNCTION 15
OPEN FILE

Return: AL: Return Code

The Open File operation is used to activate a 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 an ASCII question mark (3FH) matches any directory character in any of these positions. Normally, no question marks are included and, further, byte "ex" of the FCB is set to zero before making the open call.

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. If question marks occur in the FCB then the first matching FCB is activated. Note that the current record ("cr") must be zeroed by the program if the file is to be accessed sequentially from the first record.

Entry		Return
CL: 10H	FUNCTION 16	AL: Return Code
DX: FCB Offset	CLOSE FILE	

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.

Entry		Return
CL: 11H	FUNCTION 17	AL: Directory Code
DX: FCB Offset	SEARCH FOR FIRST	

Search 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.

Entry		Return
CL: 12H	FUNCTION 18	AL: Directory Code
	SEARCH FOR NEXT	

The Search Next function is similar to the Search First function, except that the directory scan continues from the last matched entry. Similar to function 17, function 18 returns the decimal value 255 in A when no more directory items match. 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.

```
    Entry                                     Return
  ----------->  +---------------------+    ----------->
    CL: 13H     |     FUNCTION 19     |    AL: Return Code
                |                     |
    DX: FCB     |     DELETE FILE     |
        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 and Search 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.

```
    Entry                                     Return
  ----------->  +---------------------+    ----------->
    CL: 14H     |     FUNCTION 20     |    AL: Return Code
                |                     |
    DX: FCB     |   READ SEQUENTIAL   |
        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).

Entry		Return
CL: 15H	FUNCTION 21	AL: Return Code
DX: FCB Offset	WRITE SEQUENTIAL	

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.

Entry		Return
CL: 16H	FUNCTION 22	AL: Return Code
DX: FCB Offset	MAKE FILE	

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 file names occur, and a preceding delete operation is sufficient if there is any possibility of duplication. Upon return, register A = 0, 1, 2, or 3 if the operation was successful and 0FFH (255 decimal) if no more directory space is available. The make function has the side-effect of activating the FCB and thus a subsequent open is not necessary.

Entry		Return
CL: 17H	FUNCTION 23	AL: Return Code
DX: FCB Offset	RENAME FILE	

The Rename function uses the FCB addressed by DX to change all directory entries of the file specified by the file name in the first 16 bytes of the FCB to the file name in the second 16 bytes. It is the user's responsibility to insure that the file names specified are valid CP/M unambiguous file names. The drive code "dr" at position 0 is used to select the drive, while the drive code for the new file name at position 16 of the FCB is ignored. Upon return, register AL is set to a value of zero if the rename was successful, and 0FFH (255 decimal) if the first file name could not be found in the directory scan.

Entry		Return
CL: 18H	FUNCTION 24	BX: Login Vector
BX: Login Vector	RETURN 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 an drive that is actively on-line due to an explicit disk drive selection, or an implicit drive select caused by a file operation which specified a non-zero "dr" field.

Entry		Return
CL: 19H	FUNCTION 25	AL: Current Disk
	RETURN 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.

Entry
CL: 1AH
DX: DMA Offset

FUNCTION 26
SET DMA ADDRESS

Return

"DMA" is an acronym for Direct Memory Address, which is often used in connection with disk controllers which directly access the memory of the mainframe computer to transfer data to and from the disk subsystem. Although many computer systems use non-DMA access (i.e., the data is transfered 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.

Entry
CL: 1BH

FUNCTION 27
GET ADDR(ALLOC)

Return
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.

Entry
CL: 1CH

FUNCTION 28
WRITE PROTECT DISK

Return

The disk write protect function provides temporary write protection for the currently selected disk. Any attempt to write to the disk, before the next cold start, warm start, disk system reset, or drive reset operation produces the message:

```
Bdos Err on d: R/O
```

```
    Entry                                        Return
 ----------->   +------------------+  ----------->
  CL: 1DH       |   FUNCTION 29    |  BX: R/O Vector Value
                |                  |
                | GET READ/ONLY    |
                |     VECTOR       |
                +------------------+
```

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.

```
    Entry                                        Return
 ----------->   +------------------+  ----------->
  CL: 1EH       |   FUNCTION 30    |  AL: Return Code
                |                  |
  DX: FCB       |    SET FILE      |
      Offset    |   ATTRIBUTES     |
                +------------------+
```

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 a FCB containing a file name with the appropriate attributes set or reset. It is the user's responsibility to insure that an ambiquous file name 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.

```
     Entry                                        Return
 ----------->                                 ----------->
 CL: 1FH              FUNCTION 31             BX: DPB Offset

                       GET ADDR               ES: Segment Base
                     (DISK PARMS)
```

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. Normally, application programs will not require this facility. Section 6.3 defines the BIOS disk parameter block.

```
     Entry                                        Return
 ----------->                                 ----------->
 CL: 20H              FUNCTION 32             AL: Current Code
                                                  or no value
 DL: 0FFH(get)         SET/GET
        or            USER CODE
     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).

Entry		Return
CL: 21H	FUNCTION 33	AL: Return Code
DX: FCB Offset	READ RANDOM	

The Read Random function is similar to the sequential file read operation of previous releases, except that the read operation takes place at a particular record number, selected by the 24-bit value constructed from the three byte field following the FCB (byte positions r0 at 33, r1 at 34, and r2 at 35). Note that the sequence of 24 bits is stored with least significant byte first (r0), middle byte next (r1), and high byte last (r2). CP/M 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 re-read as you switch from random mode to sequential read, and the last record will be re-written as you switch to a sequential write operation. You can, of course, simply advance the random record position following each random read or write to obtain the effect of a sequential I/O operation.

Error codes returned in register AL following a random read are listed in Table 4-4, below.

Table 4-4. 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.

Entry		Return
CL: 22H	FUNCTION 34	AL: Return Code
DX: FCB Offset	WRITE RANDOM	

The Write Random operation is initiated similar to the Read Random call, except that data is written to the disk from the current DMA address. Further, if the disk extent or data block which is the target of the write has not yet been allocated, the allocation is performed before the write operation continues. As in the Read Random operation, the random record number is not changed as a result of the write. The logical extent number and current record positions of the file control block are set to correspond to the random record which is being written. 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 in Table 4-5 below, or the value 00 indicating the operation was successful.

Table 4-5. 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.

Table 4-5. (continued)

Code	Meaning
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.

Entry		Return
CL: 23H	FUNCTION 35	Random Record Field Set
DX: FCB Offset	COMPUTE FILE SIZE	

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 file name which is used in the directory scan. Upon return, the random record bytes contain the "virtual" file size which is, in effect, the record address of the record following the end of the file. If, following a call to function 35, the high record byte r2 is 01, then the file contains the maximum record count 65536. Otherwise, bytes r0 and r1 constitute a 16-bit value (r0 is the least significant byte, as before) which is the file size.

Data can be appended to the end of an existing file by simply calling function 35 to set the random record position to the end of file, then performing a sequence of random writes starting at the preset record address.

The virtual size of a file corresponds to the physical size when the file is written sequentially. If, instead, the file was created in random mode and "holes" exist in the allocation, then the file may in fact contain fewer records than the size indicates. If, for example, 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.

Entry		Return
CL: 24H	FUNCTION 36	Random Record Field Set
DX: FCB Offset	SET RANDOM RECORD	

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.

A second use of function 36 occurs when switching from a sequential read or write over to random read or write. A file is sequentially accessed to a particular point in the file, function 36 is called which sets the record number, and subsequent random read and write operations continue from the next record in the file.

Entry		Return
CL: 25H	FUNCTION 37	AL: 00H
DX: Drive Vector	RESET DRIVE	

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.

In order to maintain compatibility with MP/M, CP/M returns a zero value for this function.

Entry		Return
CL: 28H	FUNCTION 40	AL: Return Code
DX: FCB Offset	WRITE RANDOM WITH ZERO FILL	

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.

Entry		Return
CL: 2FH	FUNCTION 47	
DMA buffer: Command Line	CHAIN TO PROGRAM	

The CHAIN TO PROGRAM function provides a means of chaining from one program to the next without operator intervention. Although there is no passed parameter for this call, the calling process must place a command line terminated by a null byte in the default DMA buffer.

Under CP/M-86™, the CHAIN TO PROGRAM function releases the memory of the calling function before executing the command. The command line is parsed and placed in the Base Page of the new program. The Console Command Processor (CCP) then executes the command line.

Entry		Return
CL: 031H	**FUNCTION 49** GET SYSDAT ADDRESS	BX: SYSDAT Address Offset ES: SYSDAT Address Segment

The GET SYSDAT function returns the address of the System Data Area. The system data area includes the following information:

```
dmaad           equ    word ptr 0     ;user DMA address
dmabase         equ    word ptr 2     ;user DMA base
curdsk          equ    byte ptr 4     ;current user disk
usrcode         equ    byte ptr 5     ;current user number
control_p_flag  equ    byte ptr 22    ;listing toggle...
                                      ;set by ctrl-p

console_width   equ    byte ptr 64
printer_width   equ    byte ptr 65
console_column  equ    byte ptr 66
printer_column  equ    byte ptr 67
```

The following list provides an explanation of system data area parameters.

- dmaad means current user DMA address.
- dmabase means current user DMA base.
- curdsk means current user disk, 0-15 (A-P).
- usrcode means current user area, 0-15.
- control_p_flag, 0 means do not echo console output to the printer. FF means echo to the printer.

Entry		Return
CL: 32H DX: BIOS Descriptor	FUNCTION 50 DIRECT BIOS CALL	

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 Table 5-1), 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.

Entry		Return
CL: 33H	FUNCTION 51	
DX: Base Address	SET DMA BASE	

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.

Entry		Return
CL: 34H	FUNCTION 52	BX: DMA Offset
	GET DMA BASE	ES: DMA Segment

Function 52 returns the current DMA Base Segment address in ES, with the current DMA Offset in DX.

4.4 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 only receive control 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 CONTROL-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 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 Figure 4-1 below:

Region A
Region B
Region C
Region D

Figure 4-1. 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 as shown in Figure 4-2 below.

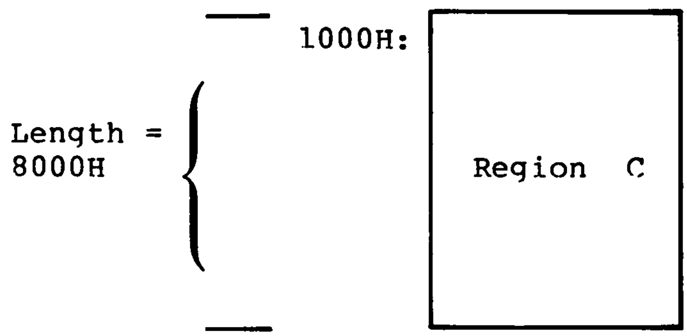


Figure 4-2. Example Memory Region

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 Figure 4-3.

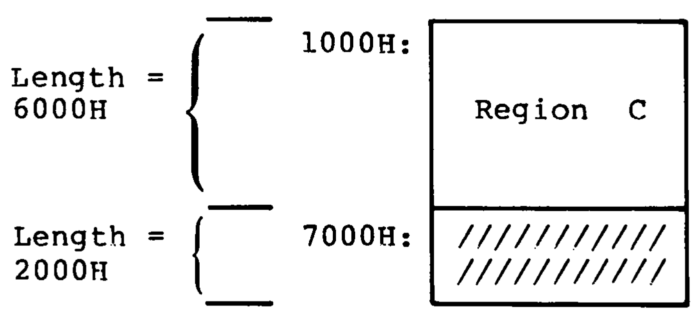


Figure 4-3. Example Memory Regions

The region beginning at paragraph address 700H 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

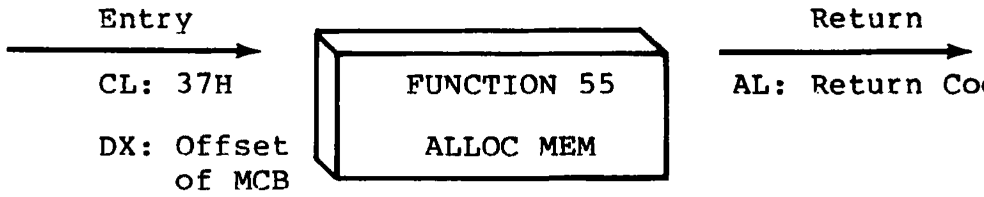
where M-Base and M-Length are either input or output values expressed in 16-byte paragraph units, and M-Ext is a returned byte value, as defined specifically with each function code. An error condition is normally flagged with a 0FFH returned value in order to match the file error conventions of CP/M.

Entry		Return
CL: 35H	FUNCTION 53	AL: Return Code
DX: Offset of MCB	GET MAX MEM	

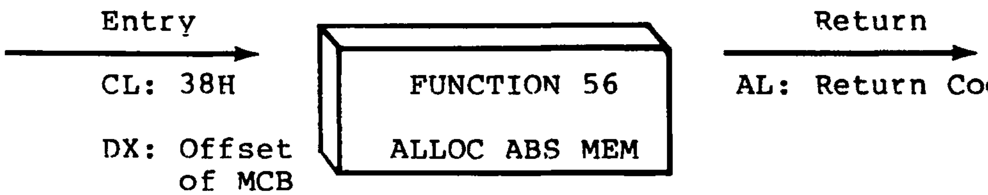
Function 53 finds the largest available memory region which is less than or equal to M-Length paragraphs. If successful, M-Base is set to the base paragraph address of the available area, and M-Length to the paragraph length. AL has the value 0FFH upon return if no memory is available, and 00H if the request was successful. M-Ext is set to 1 if there is additional memory for allocation, and 0 if no additional memory is available.

Entry		Return
CL: 36H	FUNCTION 54	AL: Return Code
DX: Offset of MCB	GET ABS MAX	

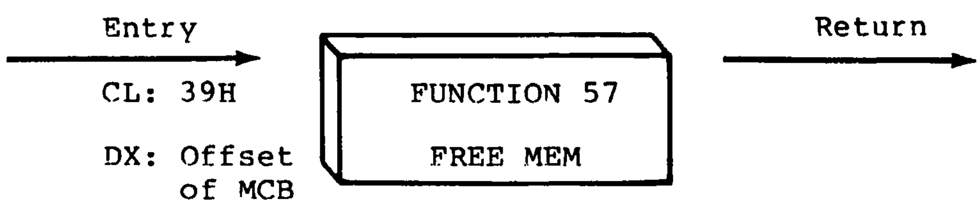
Function 54 is used to find the largest possible region at the absolute paragraph boundary given by M-Base, for a maximum of M-Length paragraphs. M-Length is set to the actual length if successful. AL has the value 0FFH upon return if no memory is available at the absolute address, and 00H if the request was successful.



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.



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 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.

Entry		Return
CL: 3AH	FUNCTION 58 FREE ALL MEM	

Function 58 is used to release all memory in the CP/M-86 environment (normally used only by the CCP upon initialization).

Entry		Return
CL: 3BH DX: Offset of FCB	FUNCTION 59 PROGRAM LOAD	AX: Return Code/ Base Page Addr 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 is 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.

Section 5
Basic I/O System (BIOS) Organization

The distribution version of CP/M-86 is setup for operation with the Intel SBC 86/12 microcomputer and an Intel 204 diskette controller. All hardware dependencies are, however, concentrated in subroutines which are collectively referred to as the Basic I/O System, or BIOS. A CP/M-86 system implementor can modify these subroutines, as described below, to tailor CP/M-86 to fit nearly any 8086 or 8088 operating environment. This section describes the actions of each BIOS entry point, and defines variables and tables referenced within the BIOS. The discussion of Disk Definition Tables is, however, treated separately in the next section of this manual.

5.1 Organization of the BIOS

The BIOS portion of CP/M-86 resides in the topmost portion of the operating system (highest addresses), and takes the general form shown in Figure 5-1, below:

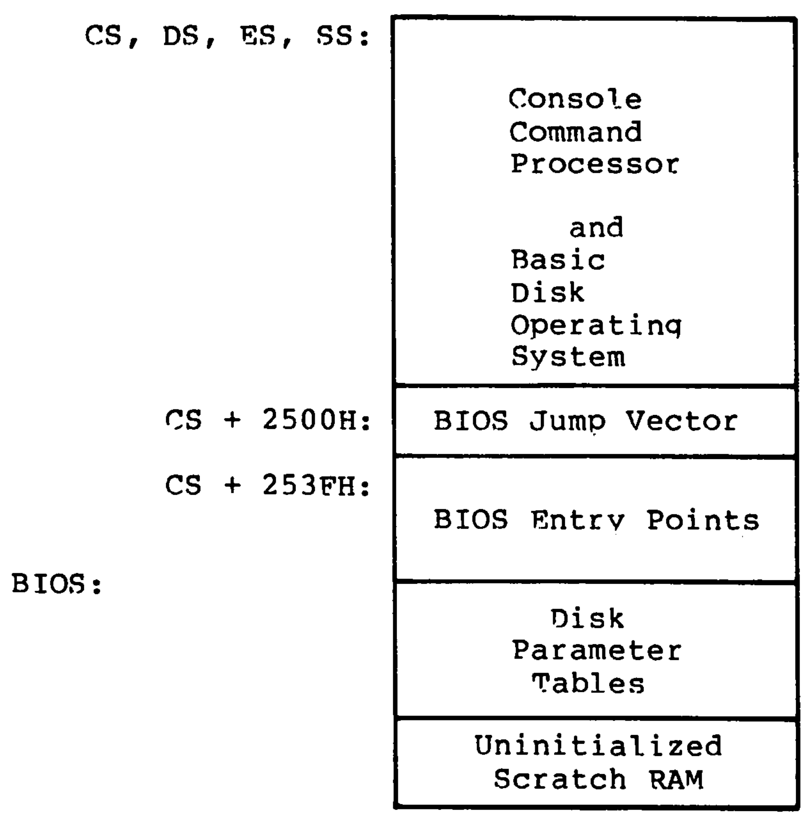


Figure 5-1. General CP/M-86 Organization

As described in the following sections, the CCP and BDOS are supplied with CP/M-86 in hex file form as CPM.H86. In order to implement CP/M-86 on non-standard hardware, you must create a BIOS which performs the functions listed below and concatenate the resulting hex file to the end of the CPM.H86 file. The GENCMD utility is then used to produce the CPM.SYS file for subsequent load by the cold start loader. The cold start loader that loads the CPM.SYS file into memory contains a simplified form of the BIOS, called the LDBIOS (Loader BIOS). It loads CPM.SYS into memory at the location defined in the CPM.SYS header (usually 0400H). The procedure to follow in construction and execution of the cold start loader and the CP/M-86 Loader is given in a later section.

Appendix D contains a listing of the standard CP/M-86 BIOS for the Intel SBC 86/12 system using the Intel 204 Controller Board. Appendix E shows a sample "skeletal" BIOS called CBIOS that contains the essential elements with the device drivers removed. You may wish to review these listings in order to determine the overall structure of the BIOS.

5.2 The BIOS Jump Vector

Entry to the BIOS is through a "jump vector" located at offset 2500H from the base of the operating system. The jump vector is a sequence of 21 three-byte jump instructions which transfer program control to the individual BIOS entry points. Although some non-essential BIOS subroutines may contain a single return (RET) instruction, the corresponding jump vector element must be present in the order shown below in Table 5-1. An example of a BIOS jump vector may be found in Appendix D, in the standard CP/M-86 BIOS listing.

Parameters for the individual subroutines in the BIOS are passed in the CX and DX registers, when required. CX receives the first parameter; DX is used for a second argument. Return values are passed in the registers acco ding to type: Byte values are returned in AL. Word values (16 bits) are returned in BX. Specific parameters and returned values are described with each subroutine.

Table 5-1. 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 Char 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

There are three major divisions in the BIOS jump table: system (re)initialization subroutines, simple character I/O subroutines, and disk I/O subroutines.

5.3 Simple Peripheral Devices

All simple character I/O operations are assumed to be performed in ASCII, upper and lower case, with high order (parity bit) set to zero. An end-of-file condition for an input device is given by an ASCII control-z (1AH). Peripheral devices are seen by CP/M-86 as "logical" devices, and are assigned to physical devices within the BIOS. Device characteristics are defined in Table 5-2.

Table 5-2. CP/M-86 Logical Device Characteristics

Device Name	Characteristics
CONSOLE	The principal interactive console which communicates with the operator, accessed through CONST, CONIN, and CONOUT. Typically, the CONSOLE is a device such as a CRT or Teletype.
LIST	The principal listing device, if it exists on your system, which is usually a hard-copy device, such as a printer or Teletype.
PUNCH	The principal tape punching device, if it exists, which is normally a high-speed paper tape punch or Teletype.
READER	The principal tape reading device, such as a simple optical reader or teletype.

Note that a single peripheral can be assigned as the LIST, PUNCH, and READER device simultaneously. If no peripheral device is assigned as the LIST, PUNCH, or READER device, your CBIOS should give an appropriate error message so that the system does not "hang" if the device is accessed by PIP or some other transient program. Alternately, the PUNCH and LIST subroutines can just simply return, and the READER subroutine can return with a 1AH (ctl-Z) in reg A to indicate immediate end-of-file.

For added flexibility, you can optionally implement the "IOBYTE" function which allows reassignment of physical and logical devices. The IOBYTE function creates a mapping of logical to physical devices which can be altered during CP/M-86 processing (see the STAT command). The definition of the IOBYTE function corresponds to the Intel standard as follows: a single location in the BIOS is maintained, called IOBYTE, which defines the logical to physical device mapping which is in effect at a particular time. The mapping is performed by splitting the IOBYTE into four distinct fields of two bits each, called the CONSOLE, READER, PUNCH, and LIST fields, as shown below:

	most significant			least significant
IOBYTE	LIST	PUNCH	READER	CONSOLE
	bits 6,7	bits 4,5	bits 2,3	bits 0,1

The value in each field can be in the range 0-3, defining the assigned source or destination of each logical device. The values which can be assigned to each field are given in Table 5-3, below.

Table 5-3. IOBYTE Field Definitions

```
CONSOLE field (bits 0,1)
    0  - console is assigned to the console printer (TTY:)
    1  - console is assigned to the CRT device (CRT:)
    2  - batch mode: use the READER as the CONSOLE input,
         and the LIST device as the CONSOLE output (BAT:)
    3  - user defined console device (UC1:)

READER field (bits 2,3)
    0  - READER is the Teletype device (TTY:)
    1  - READER is the high-speed reader device (RDR:)
    2  - user defined reader # 1 (UR1:)
    3  - user defined reader # 2 (UR2:)

PUNCH field (bits 4,5)
    0  - PUNCH is the Teletype device (TTY:)
    1  - PUNCH is the high speed punch device (PUN:)
    2  - user defined punch # 1 (UP1:)
    3  - user defined punch # 2 (UP2:)

LIST field (bits 6,7)
    0  - LIST is the Teletype device (TTY:)
    1  - LIST is the CRT device (CRT:)
    2  - LIST is the line printer device (LPT:)
    3  - user defined list device (UL1:)
```

Note again that the implementation of the IOBYTE is optional, and affects only the organization of your CBIOS. No CP/M-86 utilities use the IOBYTE except for PIP which allows access to the physical devices, and STAT which allows logical-physical assignments to be made and displayed. In any case, you should omit the IOBYTE implementation until your basic CBIOS is fully implemented and tested, then add the IOBYTE to increase your facilities.

5.4 BIOS Subroutine Entry Points

The actions which must take place upon entry to each BIOS subroutine are given below. It should be noted that disk I/O is always performed through a sequence of calls on the various disk access subroutines. These setup the disk number to access, the track and sector on a particular disk, and the direct memory access (DMA) offset and segment addresses involved in the I/O operation. After all these parameters have been setup, a call is made to the READ or WRITE function to perform the actual I/O operation. Note that there is often a single call to SELDSK to select a disk drive, followed by a number of read or write operations to the selected disk before selecting another drive for subsequent operations. Similarly, there may be a call to set the DMA segment base and a call to set the DMA offset followed by several calls which read or write from the selected DMA address before the DMA address is changed. The track and sector subroutines are always called before the READ or WRITE operations are performed.

The READ and WRITE subroutines should perform several retries (10 is standard) before reporting the error condition to the BDOS. The HOME subroutine may or may not actually perform the track 00 seek, depending upon your controller characteristics; the important point is that track 00 has been selected for the next operation, and is often treated in exactly the same manner as SETTRK with a parameter of 00.

Table 5-4. BIOS Subroutine Summary

Subroutine	Description
INIT	This subroutine is called directly by the CP/M-86 loader after the CPM.SYS file has been read into memory. The procedure is responsible for any hardware initialization not performed by the bootstrap loader, setting initial values for BIOS variables (including IOBYTE), printing a sign-on message, and initializing the interrupt vector to point to the BDOS offset (0B06H) and base. When this routine completes, it jumps to the CCP offset (0H). All segment registers should be initialized at this time to contain the base of the operating system.
WBOOT	This subroutine is called whenever a program terminates by performing a BDOS function #0 call. Some re-initialization of the hardware or software may occur here. When this routine completes, it jumps directly to the warm start entry point of the CCP (06H).
CONST	Sample the status of the currently assigned console device and return 0FFH in register AL if a character is ready to read, and 00H in register AL if no console characters are ready.

Table 5-4. (continued)

Subroutine	Description
CONIN	Read the next console character into register AL, and set the parity bit (high order bit) to zero. If no console character is ready, wait until a character is typed before returning.
CONOUT	Send the character from register CL to the console output device. The character is in ASCII, with high order parity bit set to zero. You may want to include a time-out on a line feed or carriage return, if your console device requires some time interval at the end of the line (such as a TI Silent 700 terminal). You can, if you wish, filter out control characters which have undesirable effects on the console device.
LIST	Send the character from register CL to the currently assigned listing device. The character is in ASCII with zero parity.
PUNCH	Send the character from register CL to the currently assigned punch device. The character is in ASCII with zero parity.
READER	Read the next character from the currently assigned reader device into register AL with zero parity (high order bit must be zero). An end of file condition is reported by returning an ASCII CONTROL-Z (1AH).
HOME	Return the disk head of the currently selected disk to the track 00 position. If your controller does not have a special feature for finding track 00, you can translate the call into a call to SETTRK with a parameter of 0.

Table 5-4. (continued)

Subroutine	Description
SELDSK	Select the disk drive given by register CL for further operations, where register CL contains 0 for drive A, 1 for drive B, and so on up to 15 for drive P (the standard CP/M-86 distribution version supports two drives). On each disk select, SELDSK must return in BX the base address of the selected drive's Disk Parameter Header. For standard floppy disk drives, the content of the header and associated tables does not change. The sample BIOS included with CP/M-86 called CBIOS contains an example program segment that performs the SELDSK function. If there is an attempt to select a non-existent drive, SELDSK returns BX=0000H as an error indicator. Although SELDSK must return the header address on each call, it is advisable to postpone the actual physical disk select operation until an I/O function (seek, read or write) is performed. This is due to the fact that disk select operations may take place without a subsequent disk operation and thus disk access may be substantially slower using some disk controllers. On entry to SELDSK it is possible to determine whether it is the first time the specified disk has been selected. Register DL, bit 0 (least significant bit) is a zero if the drive has not been previously selected. This information is of interest in systems which read configuration information from the disk in order to set up a dynamic disk definition table.
SETTRK	Register CX contains the track number for subsequent disk accesses on the currently selected drive. You can choose to seek the selected track at this time, or delay the seek until the next read or write actually occurs. Register CX can take on values in the range 0-76 corresponding to valid track numbers for standard floppy disk drives, and 0-65535 for non-standard disk subsystems.
SETSEC	Register CX contains the translated sector number for subsequent disk accesses on the currently selected drive (see SECTRAN, below). You can choose to send this information to the controller at this point, or instead delay sector selection until a read or write operation occurs.

Table 5-4. (continued)

Subroutine	Description
SETDMA	Register CX contains the DMA (disk memory access) offset for subsequent read or write operations. For example, if CX = 80H when SETDMA is called, then all subsequent read operations read their data into 80H through 0FFH offset from the current DMA segment base, and all subsequent write operations get their data from that address, until the next calls to SETDMA and SETDMAB occur. Note that the controller need not actually support direct memory access. If, for example, all data is received and sent through I/O ports, the CBIOS which you construct will use the 128 byte area starting at the selected DMA offset and base for the memory buffer during the following read or write operations.
READ	Assuming the drive has been selected, the track has been set, the sector has been set, and the DMA offset and segment base have been specified, the READ subroutine attempts to read one sector based upon these parameters, and returns the following error codes in register AL: 0 no errors occurred 1 non-recoverable error condition occurred Currently, CP/M-86 responds only to a zero or non-zero value as the return code. That is, if the value in register AL is 0 then CP/M-86 assumes that the disk operation completed properly. If an error occurs, however, the CBIOS should attempt at least 10 retries to see if the error is recoverable. When an error is reported the BDOS will print the message "BDOS ERR ON x: BAD SECTOR". The operator then has the option of typing RETURN to ignore the error, or CONTROL-C to abort.
WRITE	Write the data from the currently selected DMA buffer to the currently selected drive, track, and sector. The data should be marked as "non-deleted data" to maintain compatibility with other CP/M systems. The error codes given in the READ command are returned in register AL, with error recovery attempts as described above.
LISTST	Return the ready status of the list device. The value 00 is returned in AL if the list device is not ready to accept a character, and 0FFH if a character can be sent to the printer.

Table 5-4. (continued)

Subroutine	Description
SECTRAN	Performs logical to physical sector translation to improve the overall response of CP/M-86. Standard CP/M-86 systems are shipped with a "skew factor" of 6, where five physical sectors are skipped between sequential read or write operations. This skew factor allows enough time between sectors for most programs to load their buffers without missing the next sector. In computer systems that use fast processors, memory and disk subsystems, the skew factor may be changed to improve overall response. Note, however, that you should maintain a single density IBM compatible version of CP/M-86 for information transfer into and out of your computer system, using a skew factor of 6. In general, SECTRAN receives a logical sector number in CX. This logical sector number may range from 0 to the number of sectors -1. Sectran also receives a translate table offset in DX. The sector number is used as an index into the translate table, with the resulting physical sector number in BX. For standard systems, the tables and indexing code is provided in the CBIOS and need not be changed. If DX = 0000H no translation takes place, and CX is simply copied to BX before returning. Otherwise, SECTRAN computes and returns the translated sector number in BX. Note that SECTRAN is called when no translation is specified in the Disk Parameter Header.
SETDMAB	Register CX contains the segment base for subsequent DMA read or write operations. The BIOS will use the 128 byte buffer at the memory address determined by the DMA base and the DMA offset during read and write operations.
GETSEGB	Returns the address of the Memory Region Table (MRT) in BX. The returned value is the offset of the table relative to the start of the operating system. The table defines the location and extent of physical memory which is available for transient programs.

Table 5-4. (continued)

Subroutine	Description
	Memory areas reserved for interrupt vectors and the CP/M-86 operating system are not included in the MRT. The Memory Region Table takes the form:

	8-bit	
MRT:	R-Cnt	
0:	R-Base	R-Length
1:	R-Base	R-Length
	. . .	
n:	R-Base	R-Length
	16-bit	16-bit

	where R-Cnt is the number of Memory Region Descriptors (equal to n+1 in the diagram above), while R-Base and R-Length give the paragraph base and length of each physically contiguous area of memory. Again, the reserved interrupt locations, normally 0-3FFH, and the CP/M-86 operating system are not included in this map, because the map contains regions available to transient programs. If all memory is contiguous, the R-Cnt field is 1 and n = 0, with only a single Memory Region Descriptor which defines the region.
GETIOB	Returns the current value of the logical to physical input/output device byte (IOBYTE) in AL. This eight-bit value is used to associate physical devices with CP/M-86's four logical devices.
SETIOB	Use the value in CL to set the value of the IOBYTE stored in the BIOS.

The following section describes the exact layout and construction of the disk parameter tables referenced by various subroutines in the BIOS.

Section 6
BIOS Disk Definition Tables

Similar to CP/M-80, CP/M-86 is a table-driven operating system with a separate field-configurable Basic I/O System (BIOS). By altering specific subroutines in the BIOS presented in the previous section, CP/M-86 can be customized for operation on any RAM-based 8086 or 8088 microprocessor system.

The purpose of this section is to present the organization and construction of tables within the BIOS that define the characteristics of a particular disk system used with CP/M-86. These tables can be either hand-coded or automatically generated using the GENDEF utility provided with CP/M-86. The elements of these tables are presented below.

6.1 Disk Parameter Table Format

In general, each disk drive has an associated (16-byte) disk parameter header which both contains information about the disk drive and provides a scratchpad area for certain BDOS operations. The format of the disk parameter header for each drive is shown below.

Disk Parameter Header							
XLT	0000	0000	0000	DIRBUF	DPB	CSV	ALV
16b	16b	16b	16b	16b	16b	16b	16b

where each element is a word (16-bit) value. The meaning of each Disk Parameter Header (DPH) element is given in Table 6-1.

Table 6-1. Disk Parameter Header Elements

Element	Description
XLT	Offset of the logical to physical translation vector, if used for this particular drive, or the value 0000H if no sector translation takes place (i.e, the physical and logical sector numbers are the same). Disk drives with identical sector skew factors share the same translate tables.
0000	Scratchpad values for use within the BDOS (initial value is unimportant).

Table 6-1. (continued)

Element	Description
DIRBUF	Offset of a 128 byte scratchpad area for directory operations within BDOS. All DPH's address the same scratchpad area.
DPB	Offset of a disk parameter block for this drive. Drives with identical disk characteristics address the same disk parameter block.
CSV	Offset of a scratchpad area used for software check for changed disks. This offset is different for each DPH.
ALV	Offset of a scratchpad area used by the BDOS to keep disk storage allocation information. This offset is different for each DPH.

Given n disk drives, the DPH's are arranged in a table whose first row of 16 bytes corresponds to drive 0, with the last row corresponding to drive n-1. The table thus appears as

DPBASE

00	XLT 00	0000	0000	0000	DIRBUF	DBP 00	CSV 00	ALV 00
01	XLT 01	0000	0000	0000	DIRBUF	DBP 01	CSV 01	ALV 01

(and so-forth through)

n-1	XLTn-1	0000	0000	0000	DIRBUF	DBPn-1	CSVn-1	ALVn-1

where the label DPBASE defines the offset of the DPH table relative to the beginning of the operating system.

A responsibility of the SELDSK subroutine, defined in the previous section, is to return the offset of the DPH from the beginning of the operating system for the selected drive. The following sequence of operations returns the table offset, with a 0000H returned if the selected drive does not exist.

```
NDISKS    EQU    4      ;NUMBER OF DISK DRIVES
......
SELDSK:
          ;SELECT DISK N GIVEN BY CL
          MOV    BX,0000H  ;READY FOR ERR
          CPM    CL,NDISKS ;N BEYOND MAX DISKS?
          JNB    RETURN    ;RETURN IF SO
                           ;0 <= N < NDISKS
          MOV    CH,0      ;DOUBLE (N)
          MOV    BX,CX     ;BX = N
          MOV    CL,4      ;READY FOR * 16
          SHL    BX,CL     ;N = N * 16
          MOV    CX,OFFSET DPBASE
          ADD    BX,CX     ;DPBASE + N * 16
RETURN:   RET              ;BX - .DPH (N)
```

The translation vectors (XLT 00 through XLTn-1) are located elsewhere in the BIOS, and simply correspond one-for-one with the logical sector numbers zero through the sector count-1. The Disk Parameter Block (DPB) for each drive is more complex. A particular DPB, which is addressed by one or more DPH's, takes the general form:

SPT	BSH	BLM	EXM	DSM	DRM	AL0	AL1	CKS	OFF
16b	8b	8b	8b	16b	16b	8b	8b	16b	16b

where each is a byte or word value, as shown by the "8b" or "16b" indicator below the field. The fields are defined in Table 6-2.

Table 6-2. Disk Parameter Block Fields

Field	Definition
SPT	is the total number of sectors per track
BSH	is the data allocation block shift factor, determined by the data block allocation size.
BLM	is the block mask which is also determined by the data block allocation size.
EXM	is the extent mask, determined by the data block allocation size and the number of disk blocks.
DSM	determines the total storage capacity of the disk drive
DRM	determines the total number of directory entries which can be stored on this drive

Table 6-2. (continued)

Field	Definition
AL0,AL1	determine reserved directory blocks.
CKS	is the size of the directory check vector
OFF	is the number of reserved tracks at the beginning of the (logical) disk.

Although these table values are produced automatically by GENDEF, it is worthwhile reviewing the derivation of each field so that the values may be cross-checked when necessary. The values of BSH and BLM determine (implicitly) the data allocation size BLS, which is not an entry in the disk parameter block. Given that you have selected a value for BLS, the values of BSH and BLM are shown in Table 6-3 below, where all values are in decimal.

Table 6-3. BSH and BLM Values for Selected BLS

BLS	BSH	BLM
1,024	3	7
2,048	4	15
4,096	5	31
8,192	6	63
16,384	7	127

The value of EXM depends upon both the BLS and whether the DSM value is less than 256 or greater than 255, as shown in the following table.

Table 6-4. Maximum EXM Values

BLS	DSM < 256	DSM > 255
1,024	0	N/A
2,048	1	0
4,096	3	1
8,192	7	3
16,384	15	7

The value of DSM is the maximum data block number supported by this particular drive, measured in BLS units. The product BLS times (DSM+1) is the total number of bytes held by the drive and, of course, must be within the capacity of the physical disk, not counting the reserved operating system tracks.

The DRM entry is one less than the total number of directory entries, which can take on a 16-bit value. The values of AL0 and AL1, however, are determined by DRM. The two values AL0 and AL1 can together be considered a string of 16-bits, as shown below.

AL0								AL1							
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15

where position 00 corresponds to the high order bit of the byte labeled AL0, and 15 corresponds to the low order bit of the byte labeled AL1. Each bit position reserves a data block for a number of directory entries, thus allowing a total of 16 data blocks to be assigned for directory entries (bits are assigned starting at 00 and filled to the right until position 15). Each directory entry occupies 32 bytes, as shown in Table 6-5.

Table 6-5. BLS and Number of Directory Entries

BLS	Directory Entries
1,024	32 times # bits
2,048	64 times # bits
4,096	128 times # bits
8,192	256 times # bits
16,384	512 times # bits

Thus, if DRM = 127 (128 directory entries), and BLS = 1024, then there are 32 directory entries per block, requiring 4 reserved blocks. In this case, the 4 high order bits of AL0 are set, resulting in the values AL0 = 0F0H and AL1 = 00H.

The CKS value is determined as follows: if the disk drive media is removable, then CKS = (DRM+1)/4, where DRM is the last directory entry number. If the media is fixed, then set CKS = 0 (no directory records are checked in this case).

Finally, the OFF field determines the number of tracks which are skipped at the beginning of the physical disk. This value is automatically added whenever SETTRK is called, and can be used as a mechanism for skipping reserved operating system tracks, or for partitioning a large disk into smaller segmented sections.

To complete the discussion of the DPB, recall that several DPH's can address the same DPB if their drive characteristics are identical. Further, the DPB can be dynamically changed when a new drive is addressed by simply changing the pointer in the DPH since the BDOS copies the DPB values to a local area whenever the SELDSK function is invoked.

Returning back to the DPH for a particular drive, note that the two address values CSV and ALV remain. Both addresses reference an area of uninitialized memory following the BIOS. The areas must be unique for each drive, and the size of each area is determined by the values in the DPB.

The size of the area addressed by CSV is CKS bytes, which is sufficient to hold the directory check information for this particular drive. If CKS = (DRM+1)/4, then you must reserve (DRM+1)/4 bytes for directory check use. If CKS = 0, then no storage is reserved.

The size of the area addressed by ALV is determined by the maximum number of data blocks allowed for this particular disk, and is computed as (DSM/8)+1.

The BIOS shown in Appendix D demonstrates an instance of these tables for standard 8" single density drives. It may be useful to examine this program, and compare the tabular values with the definitions given above.

6.2 Table Generation Using GENDEF

The GENDEF utility supplied with CP/M-86 greatly simplifies the table construction process. GENDEF reads a file

```
x.DEF
```

containing the disk definition statements, and produces an output file

```
x.LIB
```

containing assembly language statements which define the tables necessary to support a particular drive configuration. The form of the GENDEF command is:

```
GENDEF x parameter list
```

where x has an assumed (and unspecified) filetype of DEF. The parameter list may contain zero or more of the symbols defined in Table 6-6.

Table 6-6. GENDEF Optional Parameters

Parameter	Effect
$C	Generate Disk Parameter Comments
$O	Generate DPBASE OFFSET $
$Z	Z80, 8080, 8085 Override
$COZ	(Any of the Above)

The C parameter causes GENDEF to produce an accompanying comment line, similar to the output from the "STAT DSK:" utility which describes the characteristics of each defined disk. Normally, the DPBASE is defined as

```
DPBASE  EQU  $
```

which requires a MOV CX,OFFSET DPBASE in the SELDSK subroutine shown above. For convenience, the $O parameter produces the definition

```
DPBASE  EQU  OFFSET $
```

allowing a MOV CX,DPBASE in SELDSK, in order to match your particular programming practices. The $Z parameter is included to override the standard 8086/8088 mode in order to generate tables acceptable for operation with Z80, 8080, and 8085 assemblers.

The disk definition contained within x.DEF is composed with the CP/M text editor, and consists of disk definition statements identical to those accepted by the DISKDEF macro supplied with CP/M-80 Version 2. A BIOS disk definition consists of the following sequence of statements:

```
DISKS    n
DISKDEF  0,...
DISKDEF  1,...
......
DISKDEF  n-1
......
ENDEF
```

Each statement is placed on a single line, with optional embedded comments between the keywords, numbers, and delimiters.

The DISKS statement defines the number of drives to be configured with your system, where n is an integer in the range 1 through 16. A series of DISKDEF statements then follow which define the characteristics of each logical disk, 0 through n-1, corresponding to logical drives A through P. Note that the DISKS and DISKDEF statements generate the in-line fixed data tables described in the previous section, and thus must be placed in a non-executable portion of your BIOS, typically at the end of your BIOS, before the start of uninitialized RAM.

The ENDEF (End of Diskdef) statement generates the necessary uninitialized RAM areas which are located beyond initialized RAM in your BIOS.

The form of the DISKDEF statement is

```
DISKDEF  dn,fsc,lsc,[skf],bls,dks,dir,cks,ofs,[0]
```

where

```
dn      is the logical disk number, 0 to n-1
fsc     is the first physical sector number (0 or 1)
lsc     is the last sector number
skf     is the optional sector skew factor
bls     is the data allocation block size
dks     is the disk size in bls units
dir     is the number of directory entries
cks     is the number of "checked" directory entries
ofs     is the track offset to logical track 00
[0]     is an optional 1.4 compatibility flag
```

The value "dn" is the drive number being defined with this DISKDEF statement. The "fsc" parameter accounts for differing sector numbering systems, and is usually 0 or 1. The "lsc" is the last numbered sector on a track. When present, the "skf" parameter defines the sector skew factor which is used to create a sector translation table according to the skew. If the number of sectors is less than 256, a single-byte table is created, otherwise each translation table element occupies two bytes. No translation table is created if the skf parameter is omitted or equal to 0.

The "bls" parameter specifies the number of bytes allocated to each data block, and takes on the values 1024, 2048, 4096, 8192, or 16384. Generally, performance increases with larger data block sizes because there are fewer directory references. Also, logically connected data records are physically close on the disk. Further, each directory entry addresses more data and the amount of BIOS work space is reduced. The "dks" specifies the total disk size in "bls" units. That is, if the bls = 2048 and dks = 1000, then the total disk capacity is 2,048,000 bytes. If dks is greater than 255, then the block size parameter bls must be greater than 1024. The value of "dir" is the total number of directory entries which may exceed 255, if desired.

The "cks" parameter determines the number of directory items to check on each directory scan, and is used internally to detect changed disks during system operation, where an intervening cold start or system reset has not occurred (when this situation is detected, CP/M-86 automatically marks the disk read/only so that data is not subsequently destroyed). As stated in the previous section, the value of cks = dir when the media is easily changed, as is the case with a floppy disk subsystem. If the disk is permanently mounted, then the value of cks is typically 0, since the probability of changing disks without a restart is quite low.

The "ofs" value determines the number of tracks to skip when this particular drive is addressed, which can be used to reserve additional operating system space or to simulate several logical drives on a single large capacity physical drive. Finally, the [0] parameter is included when file compatibility is required with versions of CP/M-80, version 1.4 which have been modified for higher density disks (typically double density). This parameter ensures that no directory compression takes place, which would cause incompatibilities with these non-standard CP/M 1.4 versions. Normally, this parameter is not included.

For convenience and economy of table space, the special form

```
DISKDEF    i,j
```

gives disk i the same characteristics as a previously defined drive j. A standard four-drive single density system, which is compatible with CP/M-80 Version 1.4, and upwardly compatible with CP/M-80 Version 2 implementations, is defined using the following statements:

```
DISKS      4
DISKDEF    0,1,26,6,1024,243,64,64,2
DISKDEF    1,0
DISKDEF    2,0
DISKDEF    3,0
ENDEF
```

with all disks having the same parameter values of 26 sectors per track (numbered 1 through 26), with a skew of 6 between sequential accesses, 1024 bytes per data block, 243 data blocks for a total of 243K byte disk capacity, 64 checked directory entries, and two operating system tracks.

The DISKS statement generates n Disk Parameter Headers (DPH's), starting at the DPH table address DPBASE generated by the statement. Each disk header block contains sixteen bytes, as described above, and corresponds one-for-one to each of the defined drives. In the four drive standard system, for example, the DISKS statement generates a table of the form:

```
DPBASE  EQU  $
DPE0    DW   XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV0,ALV0
DPE1    DW   XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV1,ALV1
DPE2    DW   XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV2,ALV2
DPE3    DW   XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV3,ALV3
```

where the DPH labels are included for reference purposes to show the beginning table addresses for each drive 0 through 3. The values contained within the disk parameter header are described in detail earlier in this section. The check and allocation vector addresses are generated by the ENDEF statement for inclusion in the RAM area following the BIOS code and tables.

Note that if the "skf" (skew factor) parameter is omitted (or equal to 0), the translation table is omitted, and a 0000H value is inserted in the XLT position of the disk parameter header for the disk. In a subsequent call to perform the logical to physical translation, SECTRAN receives a translation table address of DX = 0000H, and simply returns the original logical sector from CX in the BX register. A translate table is constructed when the skf parameter is present, and the (non-zero) table address is placed into the corresponding DPH's. The table shown below, for example, is constructed when the standard skew factor skf = 6 is specified in the DISKDEF statement call:

```
XLT0    EQU    OFFSET $
        DB     1,7,13,19,25,5,11,17,23,3,9,15,21
        DB     2,8,14,20,26,6,12,18,24,4,10,16,22
```

Following the ENDEF statement, a number of uninitialized data areas are defined. These data areas need not be a part of the BIOS which is loaded upon cold start, but must be available between the BIOS and the end of operating system memory. The size of the uninitialized RAM area is determined by EQU statements generated by the ENDEF statement. For a standard four-drive system, the ENDEF statement might produce

```
1C72 =       BEGDAT EQU OFFSET $
             (data areas)
1DB0 =       ENDDAT EQU OFFSET $
013C =       DATSIZ EQU OFFSET $-BEGDAT
```

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

After modification, you can use the STAT program to check your drive characteristics, since STAT uses the disk parameter block to decode the drive information. The comment included in the LIB file by the $C parameter to GENCMD will match the output from STAT. The STAT command form

```
STAT d:DSK:
```

decodes the disk parameter block for drive d (d=A,...,P) and displays the values shown below:

```
r: 128 Byte Record Capacity
k: Kilobyte Drive  Capacity
d: 32  Byte Directory Entries
c: Checked  Directory Entries
e: Records/ Extent
b: Records/ Block
s: Sectors/ Track
t: Reserved Tracks
```

6.3 GENDEF Output

GENDEF produces a listing of the statements included in the DEF file at the user console (CONTROL-P can be used to obtain a printed listing, if desired). Each source line is numbered, and any errors are shown below the line in error, with a "?" beneath the item which caused the condition. The source errors produced by GENCMD are listed in Table 6-7, followed by errors that can occur when producing input and output files in Table 6-8.

Table 6-7. GENDEF Source Error Messages

Message	Meaning
Bad Val	More than 16 disks defined in DISKS statement.
Convert	Number cannot be converted, must be constant in binary, octal, decimal, or hexadecimal as in ASM-86.
Delimit	Missing delimiter between parameters.
Duplic	Duplicate definition for a disk drive.
Extra	Extra parameters occur at the end of line.
Length	Keyword or data item is too long.
Missing	Parameter required in this position.
No Disk	Referenced disk not previously defined.
No Stmt	Statement keyword not recognized.
Numeric	Number required in this position
Range	Number in this position is out of range.
Too Few	Not enough parameters provided.
Quote	Missing end quote on current line.

Table 6-8. GENDEF Input and Output Error Messages

Message	Meaning
Cannot Close ".LIB" File	LIB file close operation unsuccessful, usually due to hardware write protect.
"LIB" Disk Full	No space for LIB file.
No Input File Present	Specified DEF file not found.
No ".LIB" Directory Space	Cannot create LIB file due to too many files on LIB disk.
Premature End-of-File	End of DEF file encountered unexpectedly.

Given the file TWO.DEF containing the following statements

```
disks   2
diskdef 0,1,26,6,2048,256,128,128,2
diskdef 1,1,58,,2048,1024,300,0,2
endef
```

the command

```
gencmd two $c
```

produces the console output

```
DISKDEF Table Generator, Vers 1.0
1             DISKS    2
2             DISKDEF 0,1,58,,2048,256,128,128,2
3             DISKDEF 1,1,58,,2048,1024,300,0,2
4             ENDEF
No Error(s)
```

The resulting TWO.LIB file is brought into the following skeletal assembly language program, using the ASM-86 INCLUDE directive. The ASM-86 output listing is truncated on the right, but can be easily reproduced using GENDEF and ASM-86.

```
                                ;       Sample Program Including TWO.LI
                                ;
                                SELDSK:
                                ;       ....
  0000 B9 03 00                         MOV     CX,OFFSET DPBASE
                                ;       ....
=                                       INCLUDE TWO.LIB
=                               ;               DISKS   2
=   0003                        dpbase  equ     $               ;Base o
= 0003 32 00 00 00              dpe0    dw      xlt0,0000h      ;Transl
= 0007 00 00 00 00                      dw      0000h,0000h     ;Scratc
= 000B 5B 00 23 00                      dw      dirbuf,dpb0     ;Dir Bu
= 000F FB 00 DB 00                      dw      csv0,alv0       ;Check,
= 0013 00 00 00 00              dpe1    dw      xlt1,0000h      ;Transl
= 0017 00 00 00 00                      dw      0000h,0000h     ;Scratc
= 001B 5B 00 4C 00                      dw      dirbuf,dpb1     ;Dir Bu
= 001F 9B 01 1B 01                      dw      csv1,alv1       ;Check,
=                               ;               DISKDEF 0,1,26,6,2048,2
=                               ;
=                               ;       Disk 0 is CP/M 1.4 Single Densi
=                               ;        4096:  128 Byte Record Capacit
=                               ;         512:  Kilobyte Drive  Capacit
=                               ;         128:  32 Byte Directory Entri
=                               ;         128:  Checked Directory Entri
=                               ;         256:  Records / Extent
=                               ;          16:  Records / Block
=                               ;          26:  Sectors / Track
=                               ;           2:  Reserved  Tracks
=                               ;           6:  Sector Skew Factor
=                               ;
=   0023                        dpb0    equ     offset $        ;Disk P
= 0023 1A 00                            dw      26              ;Sector
= 0025 04                               db      4               ;Block
= 0026 0F                               db      15              ;Block
= 0027 01                               db      1               ;Extnt
= 0028 FF 00                            dw      255             ;Disk S
= 002A 7F 00                            dw      127             ;Direct
= 002C C0                               db      192             ;Alloc0
= 002D 00                               db      0               ;Alloc1
= 002E 20 00                            dw      32              ;Check
= 0030 02 00                            dw      2               ;Offset
=   0032                        xlt0    equ     offset $        ;Transl
= 0032 01 07 0D 13                      db      1,7,13,19
= 0036 19 05 0B 11                      db      25,5,11,17
= 003A 17 03 09 0F                      db      23,3,9,15
= 003E 15 02 08 0E                      db      21,2,8,14
= 0042 14 1A 06 0C                      db      20,26,6,12
= 0046 12 18 04 0A                      db      18,24,4,10
= 004A 10 16                            db      16,22
=   0020                        als0    equ     32              ;Alloca
=   0020                        css0    equ     32              ;Check
=                               ;               DISKDEF 1,1,58,,2048,10
=                               ;
=                               ;       Disk 1 is CP/M 1.4 Single Densi
=                               ;       16384:  128 Byte Record Capacit
```

```
=                           ;          2048:   Kilobyte Drive  Capacit
=                           ;           300:   32 Byte Directory Entri
=                           ;             0:   Checked Directory Entri
=                           ;           128:   Records / Extent
=                           ;            16:   Records / Block
=                           ;            58:   Sectors / Track
=                           ;             2:   Reserved  Tracks
=                           ;
=   004C                    dpbl     equ       offset $          ;Disk P
= 004C 3A 00                         dw        58                ;Sector
= 004E 04                            db        4                 ;Block
= 004F 0F                            db        15                ;Block
= 0050 00                            db        0                 ;Extnt
= 0051 FF 03                         dw        1023              ;Disk S
= 0053 2B 01                         dw        299               ;Direct
= 0055 F8                            db        248               ;Alloc0
= 0056 00                            db        0                 ;Alloc1
= 0057 00 00                         dw        0                 ;Check
= 0059 02 00                         dw        2                 ;Offset
=   0000                    xltl     equ       0                 ;No Tra
=   0080                    alsl     equ       128               ;Alloca
=   0000                    cssl     equ       0                 ;Check
=                           ;                  ENDEF
=                           ;
=                           ;        Uninitialized Scratch Memory Fo
=                           ;
=   005B                    begdat   equ       offset $          ;Start
= 005B                      dirbuf   rs        128               ;Direct
= 00DB                      alv0     rs        als0              ;Alloc
= 00FB                      csv0     rs        css0              ;Check
= 011B                      alvl     rs        alsl              ;Alloc
= 019B                      csvl     rs        cssl              ;Check
=   019B                    enddat   equ       offset $          ;End of
=   0140                    datsiz   equ       offset $-begdat ;Size o
= 019B 00                            db        0                 ;Marks
                                     END
```

Section 7
CP/M-86 Bootstrap and Adaptation Procedures

This section describes the components of the standard CP/M-86 distribution disk, the operation of each component, and the procedures to follow in adapting CP/M-86 to non-standard hardware.

CP/M-86 is distributed on a single-density IBM compatible 8" diskette using a file format which is compatible with all previous CP/M-80 operating systems. In particular, the first two tracks are reserved for operating system and bootstrap programs, while the remainder of the diskette contains directory information which leads to program and data files. CP/M-86 is distributed for operation with the Intel SBC 86/12 single-board computer connected to floppy disks through an Intel 204 Controller. The operation of CP/M-86 on this configuration serves as a model for other 8086 and 8088 environments, and is presented below.

The principal components of the distribution system are listed below:

- The 86/12 Bootstrap ROM (BOOT ROM)
- The Cold Start Loader (LOADER)
- The CP/M-86 System (CPM.SYS)

When installed in the SBC 86/12, the BOOT ROM becomes a part of the memory address space, beginning at byte location 0FF000H, and receives control when the system reset button is depressed. In a non-standard environment, the BOOT ROM is replaced by an equivalent initial loader and, therefore, the ROM itself is not included with CP/M-86. The BOOT ROM can be obtained from Digital Research or, alternatively, it can be programmed from the listing given in Appendix C or directly from the source file which is included on the distribution disk as BOOT.A86. The responsibility of the BOOT ROM is to read the LOADER from the first two system tracks into memory and pass program control to the LOADER for execution.

7.1 The Cold Start Load Operation

The LOADER program is a simple version of CP/M-86 that contains sufficient file processing capability to read CPM.SYS from the system disk to memory. When LOADER completes its operation, the CPM.SYS program receives control and proceeds to process operator input commands.

Both the LOADER and CPM.SYS programs are preceded by the standard CMD header record. The 128-byte LOADER header record contains the following single group descriptor.

G-Form	G-Length	A-Base	G-Min	G-Max
1	xxxxxxxxx	0400	xxxxxxx	xxxxxxx
8b	16b	16b	16b	16b

where G-Form = 1 denotes a code group, "x" fields are ignored, and A-Base defines the paragraph address where the BOOT ROM begins filling memory (A-Base is the word value which is offset three bytes from the beginning of the header). Note that since only a code group is present, an 8080 memory model is assumed. Further, although the A-Base defines the base paragraph address for LOADER (byte address 04000H), the LOADER can, in fact be loaded and executed at any paragraph boundary that does not overlap CP/M-86 or the BOOT ROM.

The LOADER itself consists of three parts: the Load CPM program (LDCPM), the Loader Basic Disk System (LDBDOS), and the Loader Basic I/O System (LDBIOS). Although the LOADER is setup to initialize CP/M-86 using the Intel 86/12 configuration, the LDBIOS can be field-altered to account for non-standard hardware using the same entry points described in a previous section for BIOS modification. The organization of LOADER is shown in Figure 7-1 below:

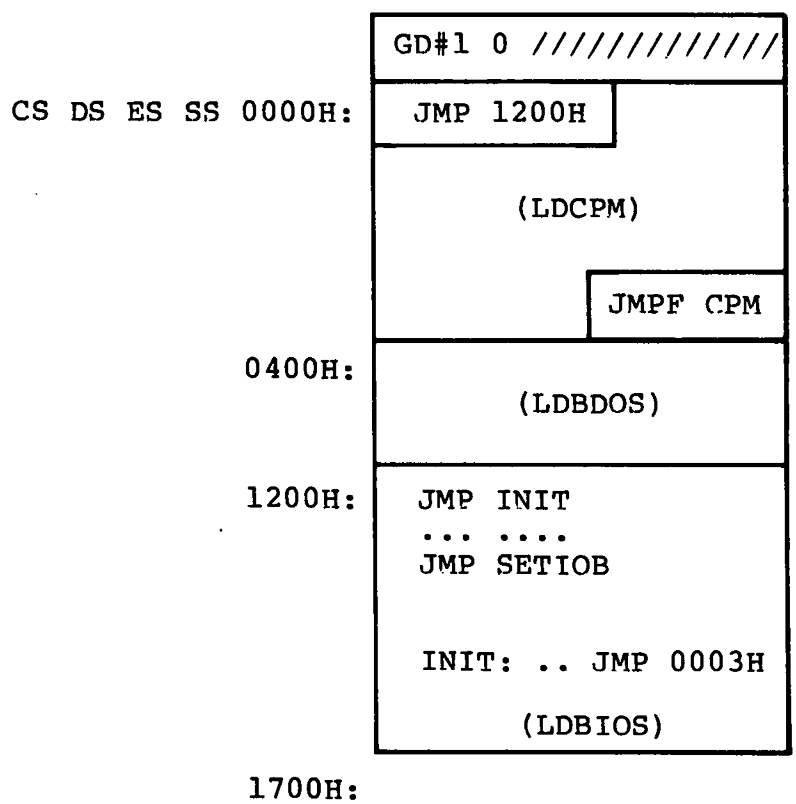


Figure 7-1. LOADER Organization

Byte offsets from the base registers are shown at the left of the diagram. GD#1 is the Group Descriptor for the LOADER code group described above, followed immediately by a "0" group terminator. The entire LOADER program is read by the BOOT ROM, excluding the header record, starting at byte location 04000H as given by the A-Field. Upon completion of the read, the BOOT ROM passes control to location 04000H where the LOADER program commences execution. The JMP 1200H instruction at the base of LDCPM transfers control to the beginning of the LDBIOS where control then transfers to the INIT subroutine. The subroutine starting at INIT performs device initialization, prints a sign-on message, and transfers back to the LDCPM program at byte offset 0003H. The LDCPM module opens the CPM.SYS file, loads the CP/M-86 system into memory and transfers control to CP/M-86 through the JMPF CPM instruction at the end of LDCPM execution, thus completing the cold start sequence.

The files LDCPM.H86 and LDBDOS.H86 are included with CP/M-86 so that you can append your own modified LDBIOS in the construction of a customized loader. In fact, BIOS.A86 contains a conditional assembly switch, called "loader_bios," which, when enabled, produces the distributed LDBIOS. The INIT subroutine portion of LDBIOS is listed in Appendix C for reference purposes. To construct a custom LDBIOS, modify your standard BIOS to start the code at offset 1200H, and change your initialization subroutine beginning at INIT to perform disk and device initialization. Include a JMP to offset 0003H at the end of your INIT subroutine. Use ASM-86 to assemble your LDBIOS.A86 program:

```
ASM86 LDBIOS
```

to produce the LDBIOS.H86 machine code file. Concatenate the three LOADER modules using PIP:

```
PIP LOADER.H86=LDCPM.H86,LDBDOS.H86,LDBIOS.H86
```

to produce the machine code file for the LOADER program. Although the standard LOADER program ends at offset 1700H, your modified LDBIOS may differ from this last address with the restriction that the LOADER must fit within the first two tracks and not overlap CP/M-86 areas. Generate the command (CMD) file for LOADER using the GENCMD utility:

```
GENCMD LOADER 8080 CODE[A400]
```

resulting in the file LOADER.CMD with a header record defining the 8080 Memory Model with an absolute paragraph address of 400H, or byte address 4000H. Use DDT to read LOADER.CMD to location 900H in your 8080 system. Then use the 8080 utility SYSGEN to copy the loader to the first two tracks of a disk.

```
A>DDT
-ILOADER.CMD
-R800
-^C
A>SYSGEN
SOURCE DRIVE NAME (or return to skip) <cr>
DESTINATION DRIVE NAME (or return to skip) B
```

Alternatively, if you have access to an operational CP/M-86 system, the command

```
LDCOPY LOADER
```

copies LOADER to the system tracks. You now have a diskette with a LOADER program which incorporates your custom LDBIOS capable of reading the CPM.SYS file into memory. For standardization, we assume LOADER executes at location 4000H. LOADER is statically relocatable, however, and its operating address is determined only by the value of A-Base in the header record.

You must, of course, perform the same function as the BOOT ROM to get LOADER into memory. The boot operation is usually accomplished in one of two ways. First, you can program your own ROM (or PROM) to perform a function similar to the BOOT ROM when your computer's reset button is pushed. As an alternative, most controllers provide a power-on "boot" operation that reads the first disk sector into memory. This one-sector program, in turn, reads the LOADER from the remaining sectors and transfers to LOADER upon completion, thereby performing the same actions as the BOOT ROM. Either of these alternatives is hardware-specific, so you'll need to be familiar with the operating environment.

7.2 Organization of CPM.SYS

The CPM.SYS file, read by the LOADER program, consists of the CCP, BDOS, and BIOS in CMD file format, with a 128-byte header record similar to the LOADER program:

G-Form	G-Length	A-Base	G-Min	G-Max
1	xxxxxxxxx	040	xxxxxxx	xxxxxxx
8b	16b	16b	16b	16b

where, instead, the A-Base load address is paragraph 040H, or byte address 0400H, immediately following the 8086 interrupt locations. The entire CPM.SYS file appears on disk as shown in Figure 7-2.

```
                                  +------+--+----------------+
                                  | GD#1 |0 |////////////////|
                                  +------+--+----------------+
(0040:0) CS DS ES SS 0000H:       |                          |
                                  |      (CCP and BDOS)      |
                                  |                          |
                                  |                          |
             (0040:) 2500H:       | JMP INIT                 |
                                  | ... ....                 |
                                  | JMP SETIOB               |
                                  |          (BIOS)          |
                                  |  INIT: .. JMP 0000H      |
                                  +--------------------------+
             (0040:) 2A00H:
```

Figure 7-2. CPM.SYS File Organization

where GD#1 is the Group Descriptor containing the A-Base value followed by a "0" terminator. The distributed 86/12 BIOS is listed in Appendix D, with an "include" statement that reads the SINGLES.LIB file containing the disk definition tables. The SINGLES.LIB file is created by GENDEF using the SINGLES.DEF statements shown below:

```
disks 2
diskdef 0,1,26,6,1024,243,64,64,2
diskdef 1,0
endef
```

The CPM.SYS file is read by the LOADER program beginning at the address given by A-Base (byte address 0400H), and control is passed to the INIT entry point at offset address 2500H. Any additional initialization, not performed by LOADER, takes place in the INIT subroutine and, upon completion, INIT executes a JMP 0000H to begin execution of the CCP. The actual load address of CPM.SYS is determined entirely by the address given in the A-Base field which can be changed if you wish to execute CP/M-86 in another region of memory. Note that the region occupied by the operating system must be excluded from the BIOS memory region table.

Similar to the LOADER program, you can modify the BIOS by altering either the BIOS.A86 or skeletal CBIOS.A86 assembly language files which are included on your source disk. In either case, create a customized BIOS which includes your specialized I/O drivers, and assemble using ASM-86:

```
ASM86 BIOS
```

to produce the file BIOS.H86 containing your BIOS machine code.

Concatenate this new BIOS to the CPM.H86 file on your distribution disk:

```
PIP CPMX.H86 = CPM.H86,BIOS.H86
```

The resulting CPMX hex file is then converted to CMD file format by executing

```
GENCMD CPMX 8080 CODE[A40]
```

in order to produce the CMD memory image with A-Base = 40H. Finally, rename the CPMX file using the command

```
REN CPM.SYS = CPMX.CMD
```

and place this file on your 8086 system disk. Now the tailoring process is complete: you have replaced the BOOT ROM by either your own customized BOOT ROM, or a one-sector cold start loader which brings the LOADER program, with your custom LDBIOS, into memory at byte location 04000H. The LOADER program, in turn, reads the CPM.SYS file, with your custom BIOS, into memory at byte location 0400H. Control transfers to CP/M-86, and you are up and operating. CP/M-86 remains in memory until the next cold start operation takes place.

You can avoid the two-step boot operation if you construct a non-standard disk with sufficient space to hold the entire CPM.SYS file on the system tracks. In this case, the cold start brings the CP/M-86 memory image into memory at the location given by A-Base, and control transfers to the INIT entry point at offset 2500H. Thus, the intermediate LOADER program is eliminated entirely, although the initialization found in the LDBIOS must, of course, take place instead within the BIOS.

Since ASM-86, GENCMD and GENDEF are provided in both COM and CMD formats, either CP/M-80 or CP/M-86 can be used to aid the customizing process. If CP/M-80 or CP/M-86 is not available, but you have minimal editing and debugging tools, you can write specialized disk I/O routines to read and write the system tracks, as well as the CPM.SYS file.

The two system tracks are simple to access, but the CPM.SYS file is somewhat more difficult to read. CPM.SYS is the first file on the disk and thus it appears immediately following the directory on the diskette. The directory begins on the third track, and occupies the first sixteen logical sectors of the diskette, while the CPM.SYS is found starting at the seventeenth sector. Sectors are "skewed" by a factor of six beginning with the directory track (the system tracks are sequential), so that you must load every sixth sector in reading the CPM.SYS file. Clearly, it is worth the time and effort to use an existing CP/M system to aid the conversion process.

Appendix A
Sector Blocking and Deblocking

Upon each call to the BIOS WRITE entry point, the CP/M-86 BDOS includes information that allows effective sector blocking and deblocking where the host disk subsystem has a sector size which is a multiple of the basic 128-byte unit. This appendix presents a general-purpose algorithm that can be included within your BIOS and that uses the BDOS information to perform the operations automatically.

Upon each call to WRITE, the BDOS provides the following information in register CL:

0	=	normal sector write
1	=	write to directory sector
2	=	write to the first sector of a new data block

Condition 0 occurs whenever the next write operation is into a previously written area, such as a random mode record update, when the write is to other than the first sector of an unallocated block, or when the write is not into the directory area. Condition 1 occurs when a write into the directory area is performed. Condition 2 occurs when the first record (only) of a newly allocated data block is written. In most cases, application programs read or write multiple 128-byte sectors in sequence, and thus there is little overhead involved in either operation when blocking and deblocking records since pre-read operations can be avoided when writing records.

This appendix lists the blocking and deblocking algorithm in skeletal form (the file is included on your CP/M-86 disk). Generally, the algorithms map all CP/M sector read operations onto the host disk through an intermediate buffer which is the size of the host disk sector. Throughout the program, values and variables which relate to the CP/M sector involved in a seek operation are prefixed by "sek," while those related to the host disk system are prefixed by "hst." The equate statements beginning on line 24 of Appendix F define the mapping between CP/M and the host system, and must be changed if other than the sample host system is involved.

The SELDSK entry point clears the host buffer flag whenever a new disk is logged-in. Note that although the SELDSK entry point computes and returns the Disk Parameter Header address, it does not physically select the host disk at this point (it is selected later at READHST or WRITEHST). Further, SETTRK, SETSEC, and SETDMA simply store the values, but do not take any other action at this point. SECTRAN performs a trivial function of returning the physical sector number.

The principal entry points are READ and WRITE. These subroutines take the place of your previous READ and WRITE operations.

The actual physical read or write takes place at either WRITEHST or READHST, where all values have been prepared: hstdsk is the host disk number, hsttrk is the host track number, and hstsec is the host sector number (which may require translation to a physical sector number). You must insert code at this point which performs the full host sector read or write into, or out of, the buffer at hstbuf of length hstsiz. All other mapping functions are performed by the algorithms.

```
;*****************************************************
;*                                                   *
;*          Sector Blocking / Deblocking             *
;*                                                   *
;* This algorithm is a direct translation of the     *
;* CP/M-80 Version, and is included here for refer-  *
;* ence purposes only.  The file DEBLOCK.LIB is in-  *
;* cluded on your CP/M-86 disk, and should be used   *
;* for actual applications.  You may wish to contact *
;* Digital Research for notices of updates.          *
;*                                                   *
;*****************************************************
;
;*****************************************************
;*                                                   *
;*          CP/M to host disk constants              *
;*                                                   *
;* (This example is setup for CP/M block size of 16K *
;* with a host sector size of 512 bytes, and 12 sec- *
;* tors per track.  Blksiz, hstsiz, hstspt, hstblk   *
;* and secshf may change for different hardware.)    *
;*****************************************************
una     equ     byte ptr [BX]   ;name for byte at BX
;
blksiz  equu    16384           ;CP/M allocation size
hstsiz  equ     512             ;host disk sector size
hstspt  equ     12              ;host disk sectors/trk
hstblk  equ     hstsiz/128      ;CP/M sects/host buff
;
;*****************************************************
;*                                                   *
;* secshf is log2(hstblk), and is listed below for   *
;* values of hstsiz up to 2048.                      *
;*                                                   *
;*          hstsiz    hstblk    secshf               *
;*             256         2         1               *
;*             512         4         2               *
;*            1024         8         3               *
;*            2048        16         4               *
;*                                                   *
```

```
;*****************************************************
secshf  equu    2                   ;log2(hstblk)
cpmspt  equ     hstblk * hstspt ;CP/M sectors/track
secmsk  equ     hstblk-1            ;sector mask
;
;*****************************************************
;*                                                   *
;*        BDOS constants on entry to write           *
;*                                                   *
;*****************************************************
wrall   equ     0                   ;write to allocated
wrdir   equ     1                   ;write to directory
wrual   equ     2                   ;write to unallocated
;
;*****************************************************
;*                                                   *
;*      The BIOS entry points given below show the   *
;*      code which is relevant to deblocking only.   *
;*                                                   *
;*****************************************************
seldsk:
        ;select disk
        ;is this the first activation of the drive?
        test DL,1                   ;lsb = 0?
        jnz selset
        ;this is the first activation, clear host buff
        mov hstact,0
        mov unacnt,0
selset:
        mov al,cl ! cbw             ;put in AX
        mov sekdsk,al               ;seek disk number
        mov cl,4 ! shl al,cl        ;times 16
        add ax,offset dpbase
        mov bx,ax
        ret
;
home:
        ;home the selected disk
        mov al,hstwrt               ;check for pending write
        test al,al
        jnz homed
        mov hstact,0                ;clear host active flag
homed:
        mov cx,0                    ;now, set track zero
;       (continue HOME routine)
        ret
;
settrk:
        ;set track given by registers CX
        mov sektrk,CX               ;track to seek
        ret
;
setsec:
        ;set sector given by register cl
        mov seksec,cl               ;sector to seek
```

```
        ret
;
setdma:
        ;set dma address given by CX
        mov dma_off,CX
        ret
;
setdmab:
        ;set segment address given by CX
        mov dma_seg,CX
        ret
;
sectran:
        ;translate sector number CX with table at [DX]
        test DX,DX      ;test for hard skewed
        jz notran       ;(blocked must be hard skewed)
        mov BX,CX
        add BX,DX
        mov BL,[BX]
        ret
no_tran:
        ;hard skewed disk, physical = logical sector
        mov BX,CX
        ret
;
read:
        ;read the selected CP/M sector
        mov unacnt,0                ;clear unallocated counter
        mov readop,1                ;read operation
        mov rsflag,1                ;must read data
        mov wrtype,wrual            ;treat as unalloc
        jmp rwoper                  ;to perform the read
;
write:
        ;write the selected CP/M sector
        mov readop,0                ;write operation
        mov wrtype,cl
        cmp cl,wrual                ;write unallocated?
        jnz chkuna                  ;check for unalloc
;
;       write to unallocated, set parameters
;
        mov unacnt,(blksiz/128) ;next unalloc recs
        mov al,sekdsk               ;disk to seek
        mov unadsk,al               ;unadsk = sekdsk
        mov ax,sektrk
        mov unatrk,ax               ;unatrk = sektrk
        mov al,seksec
        mov unasec,al               ;unasec = seksec
;
chkuna:
        ;check for write to unallocated sector
;
        mov bx,offset unacnt    ;point "UNA" at UNACNT
        mov al,una ! test al,al ;any unalloc remain?
```

```
        jz alloc                ;skip if not
;
;       more unallocated records remain
        dec al                  ;unacnt = unacnt-1
        mov una,al
        mov al,sekdsk           ;same disk?
        mov BX,offset unadsk
        cmp al,una              ;sekdsk = unadsk?
        jnz alloc               ;skip if not
;
;       disks are the same
        mov AX, unatrk
        cmp AX, sektrk
        jnz alloc               ;skip if not
;
;       tracks are the same
        mov al,seksec           ;same sector?
;
        mov BX,offset unasec    ;point una at unasec
;
        cmp al,una              ;seksec = unasec?
        jnz alloc               ;skip if not
;
;       match, move to next sector for future ref
        inc una                 ;unasec = unasec+1
        mov al,una              ;end of track?
        cmp al,cpmspt           ;count CP/M sectors
        jb noovf                ;skip if below
;
;       overflow to next track
        mov una,0               ;unasec = 0
        inc unatrk              ;unatrk=unatrk+1
;
noovf:
        ;match found, mark as unnecessary read
        mov rsflag,0            ;rsflag = 0
        jmps rwoper             ;to perform the write
;
alloc:
        ;not an unallocated record, requires pre-read
        mov unacnt,0            ;unacnt = 0
        mov rsflag,1            ;rsflag = 1
                                ;drop through to rwoper
;
;*****************************************************
;*                                                   *
;*      Common code for READ and WRITE follows       *
;*                                                   *
;*****************************************************
rwoper:
        ;enter here to perform the read/write
        mov erflag,0            ;no errors (yet)
        mov al, seksec          ;compute host sector
        mov cl, secshf
        shr al,cl
```

```
        mov sekhst,al           ;host sector to seek
;
;       active host sector?
        mov al,1
        xchg al,hstact          ;always becomes 1
        test al,al              ;was it already?
        jz filhst               ;fill host if not
;
;       host buffer active, same as seek buffer?
        mov al,sekdsk
        cmp al,hstdsk           ;sekdsk = hstdsk?
        jnz nomatch
;
;       same disk, same track?
        mov ax,hsttrk
        cmp ax,sektrk           ;host track same as seek track
        jnz nomatch
;
;       same disk, same track, same buffer?
        mov al,sekhst
        cmp al,hstsec           ;sekhst = hstsec?
        jz match                ;skip if match
nomatch:
        ;proper disk, but not correct sector
        mov al, hstwrt
        test al,al              ;"dirty" buffer ?
        jz filhst               ;no, don't need to write
        call writehst           ;yes, clear host buff
;       (check errors here)
;
filhst:
        ;may have to fill the host buffer
        mov al,sekdsk ! mov hstdsk,al
        mov ax,sektrk ! mov hsttrk,ax
        mov al,sekhst ! mov hstsec,al
        mov al,rsflag
        test al,al              ;need to read?
        jz filhstl
;
        call readhst            ;yes, if 1
;       (check errors here)
;
filhstl:
        mov hstwrt,0            ;no pending write
;
match:
        ;copy data to or from buffer depending on "readop"
        mov al,seksec           ;mask buffer number
        and ax,secmsk           ;least signif bits are masked
        mov cl, 7 ! shl ax,cl   ;shift left 7 (* 128 = 2**7)
;
;       ax has relative host buffer offset
;
        add ax,offset hstbuf    ;ax has buffer address
        mov si,ax               ;put in source index register
```

```
        mov di,dma_off          ;user buffer is dest if readop
;
        push DS ! push ES       ;save segment registers
;
        mov ES,dma_seg          ;set destseg to the users seg
                                ;SI/DI and DS/ES is swapped
                                ;if write op
        mov cx,128/2            ;length of move in words
        mov al,readop
        test al,al              ;which way?
        jnz     rwmove          ;skip if read
;
;       write operation, mark and switch direction
        mov hstwrt,1            ;hstwrt = 1 (dirty buffer now)
        xchg si,di              ;source/dest index swap
        mov ax,DS
        mov ES,ax
        mov DS,dma_seg          ;setup DS,ES for write
;
rwmove:
        cld ! rep movs AX,AX    ;move as 16 bit words
        pop ES ! pop DS         ;restore segment registers
;
;       data has been moved to/from host buffer
        cmp wrtype,wrdir        ;write type to directory?
        mov al,erflag           ;in case of errors
        jnz return_rw           ;no further processing
;
;       clear host buffer for directory write
        test al,al              ;errors?
        jnz return_rw           ;skip if so
        mov hstwrt,0            ;buffer written
        call writehst
        mov al,erflag
return_rw:
        ret
;
;*****************************************************
;*                                                   *
;* WRITEHST performs the physical write to the host  *
;* disk, while READHST reads the physical disk.      *
;*                                                   *
;*****************************************************
writehst:
        ret
;
readhst:
        ret
;
;*****************************************************
;*                                                   *
;* Use the GENDEF utility to create disk def tables  *
;*                                                   *
;*****************************************************
dpbase  equ     offset $
```

```
;       disk parameter tables go here
;
;*****************************************************
;*                                                   *
;* Uninitialized RAM areas follow, including the     *
;* areas created by the GENDEF utility listed above. *
;*                                                   *
;*****************************************************
sek_dsk rb      1               ;seek disk number
sek_trk rw      1               ;seek track number
sek_sec rb      1               ;seek sector number
;
hst_dsk rb      1               ;host disk number
hst_trk rw      1               ;host track number
hst_sec rb      1               ;host sector number
;
sek_hst rb      1               ;seek shr secshf
hst_act rb      1               ;host active flag
hst_wrt rb      1               ;host written flag
;
una_cnt rb      1               ;unalloc rec cnt
una_dsk rb      1               ;last unalloc disk
una_trk rw      1               ;last unalloc track
una_sec rb      1               ;last unalloc sector
;
erflag  rb      1               ;error reporting
rsflag  rb      1               ;read sector flag
readop  rb      1               ;1 if read operation
wrtype  rb      1               ;write operation type
dma_seg rw      1               ;last dma segment
dma_off rw      1               ;last dma offset
hstbuf  rb      hstsiz          ;host buffer
        end
```

Appendix B
Sample Random Access Program

This appendix contains a rather 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 CCP level command:

    RANDOM X.DAT

starts the test program. The program looks for a file by the name X.DAT (in this particular case) and, if found, proceeds to prompt the console for input. If not found, the file is created before the prompt is given. Each prompt takes the form

    next command?

and is followed by operator input, terminated by a carriage return. The input commands take the form

    nW   nR   Q

where n is an integer value in the range 0 to 65535, and W, R, and Q are simple command characters corresponding to random write, random read, and quit processing, respectively. If the W command is issued, the RANDOM program issues the prompt

    type data:

The operator then responds by typing up to 127 characters, followed by a carriage return. RANDOM then writes the character string into the X.DAT file at record n. If the R command is issued, RANDOM reads record number n and displays the string value at the console. If the Q command is issued, the X.DAT file is closed, and the program returns to the console command processor. 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.

```
;
;*****************************************************
;*                                                   *
;*    Sample Random Access Program for CP/M-86       *
;*                                                   *
;*****************************************************
;
;    BDOS Functions
;
coninp  equu    1        ;console input function
conout  equ     2        ;console output function
pstring equ     9        ;print string until '$'
rstring equ     10       ;read console buffer
version equ     12       ;return version number
openf   equ     15       ;file open function
closef  equ     16       ;close function
makef   equ     22       ;make file function
readr   equ     33       ;read random
writer  equ     34       ;write random
;
;   Equates for non graphic characters
cr      equ     0dh      ;carriage return
lf      equ     0ah      ;line feed
;
;
;  load SP, ready file for random access
;
        cseg
        pushf                   ;push flags in CCP stack
        pop     ax              ;save flags in AX
        cli                     ;disable interrupts
        mov     bx,ds           ;set SS register to base
        mov     ss,bx           ;set SS, SP with interru
        mov     sp,offset stack ;    for 80888
        push    ax              ;restore the flags
        popf
;
;       CP/M-86 initial release returns the file
;       system version number of 2.2: check is
;       shown below for illustration purposes.
;
        mov     cl,version
        call    bdos
        cmp     al,20h          ;version 2.0 or later?
        jnb     versok
        ;       bad version, message and go back
        mov     dx,offset badver
        call    print
        jmp     abort
;
versok:
;       correct version for random access
        mov     cl,openf        ;open default fct
        mov     dx,offset fcb
        call    bdos
```

```
        inc     al                  ;err 255 becomes zero
        jnz     ready
;
;       cannot open file, so create it
        mov     cl,makef
        mov     dx,offset fcb
        call    bdos
        inc     al                  ;err 255 becomes zero
        jnz     ready
;
;       cannot create file, directory full
        mov     dx,offset nospace
        call    print
        jmp     abort               ;back to ccp
;
;   loop back to "ready" after each command
;
ready:
;       file is ready for processing
;
        call    readcom             ;read next command
        mov     ranrec,dx           ;store input record#
        mov     ranovf,0h           ;clear high byte if set
        cmp     al,'Q'              ;quit?
        jnz     notq
;
;       quit processing, close file
        mov     cl,closef
        mov     dx,offset fcb
        call    bdos
        inc     al      ;err 255 becomes 0
        jz      error   ;error message, retry
        jmps    abort   ;back to ccp
;
;
;   end of quit command, process write
;
;
notq:
;       not the quit command, random write?
        cmp     al,'W'
        jnz     notw
;
;       this is a random write, fill buffer until cr
        mov     dx,offset datmsg
        call    print               ;data prompt
        mov     cx,127              ;up to 127 characters
        mov     bx,offset buff      ;destination
rloop:  ;read next character to buff
        push    cx                  ;save loop conntrol
        push    bx                  ;next destination
        call    getchr              ;character to AL
        pop     bx                  ;restore destination
        pop     cx                  ;restore counter
        cmp     al,cr               ;end of line?
```

```
          jz      erloop
;         not end, store character
          mov     byte ptr [bx],al
          inc     bx                  ;next to fill
          loop    rloop               ;decrement cx ..loop if
erloop:
;         end of read loop, store 00
          mov     byte ptr [bx],0h
;
;         write the record to selected record number
          mov     cl,writer
          mov     dx,offset fcb
          call    bdos
          or      al,al               ;error code zero?
          jz      ready   ;for another record
          jmps     error   ;message if not
;
;
;
;  end of write command, process read
;
;
notw:
;         not a write command, read record?
          cmp     al,'R'
          jz      ranread
          jmps     error   ;skip if not
;
;         read random record
ranread:
          mov     cl,readr
          mov     dx,offset fcb
          call    bdos
          or      al,al               ;return code 00?
          jz      readok
          jmps     error
;
;         read was successful, write to console
readok:
          call    crlf                ;new line
          mov     cx,128              ;max 128 characters
          mov     si,offset buff      ;next to get
wloop:
          lods    al                  ;next character
          and     al,07fh             ;mask parity
          jnz     wloop1
          jmp     ready               ;for another command if
wloop1:
          push    cx                  ;save counter
          push    si                  ;save next to get
          cmp     al,' '              ;graphic?
          jb      skipw               ;skip output if not grap
          call    putchr              ;output character
skipw:
          pop     si
```

```
        pop     cx
        loop    wloop           ;decrement CX and check
        jmp     ready
;
;
;  end of read command, all errors end-up here
;
;
error:
        mov     dx,offset errmsg
        call    print
        jmp     ready
;
;  BDOS entry subroutine
bdos:
        int     224             ;entry to BDOS if by INT
        ret
;
abort:                          ;return to CCP
        mov     cl,0
        call    bdos            ;use function 0 to end e
;
;  utility subroutines for console i/o
;
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
        call    bdos            ;print the string
        ret
;
readcom:
```

BREAK "F" AT n
DIRECTORY FULL

Use the same commands described in the previous message to recover from this file error.

The following table defines the disk file error messages ED returns when it cannot read or write a file.

Table 5-5. ED Disk File Error Messages

Message	Meaning
BDOS ERR ON d: RO	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 RO attribute. ED can read the file so that you can examine it, but ED cannot change a Read-Only file.

Appendix A
ASCII and Hexadecimal Conversion

ASCII stands for American Standard Code for Information Interchange. The code contains 96 printing and 32 non-printing characters used to store data on a disk. Table A-1 defines ASCII symbols, then Table A-2 lists the ASCII and hexadecimal conversions. The table includes binary, decimal, hexadecimal, and ASCII conversions.

Table A-1. ASCII Symbols

Symbol	Meaning	Symbol	Meaning
ACK	acknowledge	FS	file separator
BEL	bell	GS	group separator
BS	backspace	HT	horizontal tabulation
CAN	cancel	LF	line-feed
CR	carriage return	NAK	negative acknowledge
DC	device control	NUL	null
DEL	delete	RS	record separator
DLE	data link escape	SI	shift in
EM	end of medium	SO	shift out
ENQ	enquiry	SOH	start of heading
EOT	end of transmission	SP	space
ESC	escape	STX	start of text
ETB	end of transmission	SUB	substitute
ETX	end of text	SYN	synchronous idle
FF	form-feed	US	unit separator
		VT	vertical tabulation

Table A-2. ASCII Conversion Table

Binary	Decimal	Hexadecimal	ASCII
0000000	0	0	NUL
0000001	1	1	SOH (CTRL-A)
0000010	2	2	STX (CTRL-B)
0000011	3	3	ETX (CTRL-C)
0000100	4	4	EOT (CTRL-D)
0000101	5	5	ENQ (CTRL-E)
0000110	6	6	ACK (CTRL-F)
0000111	7	7	BEL (CTRL-G)
0001000	8	8	BS (CTRL-H)
0001001	9	9	HT (CTRL-I)
0001010	10	A	LF (CTRL-J)
0001011	11	B	VT (CTRL-K)
0001100	12	C	FF (CTRL-L)
0001101	13	D	CR (CTRL-M)
0001110	14	E	SO (CTRL-N)
0001111	15	F	SI (CTRL-O)
0010000	16	10	DLE (CTRL-P)
0010001	17	11	DC1 (CTRL-Q)
0010010	18	12	DC2 (CTRL-R)
0010011	19	13	DC3 (CTRL-S)
0010100	20	14	DC4 (CTRL-T)
0010101	21	15	NAK (CTRL-U)
0010110	22	16	SYN (CTRL-V)
0010111	23	17	ETB (CTRL-W)
0011000	24	18	CAN (CTRL-X)
0011001	25	19	EM (CTRL-Y)
0011010	26	1A	SUB (CTRL-Z)
0011011	27	1B	ESC (CTRL-[)
0011100	28	1C	FS (CTRL-\)
0011101	29	1D	GS (CTRL-])
0011110	30	1E	RS (CTRL-^)
0011111	31	1F	US (CTRL-_)
0100000	32	20	(SPACE)
0100001	33	21	!
0100010	34	22	"
0100011	35	23	#
0100100	36	24	$
0100101	37	25	%
0100110	38	26	&
0100111	39	27	'
0101000	40	28	(
0101001	41	29	)
0101010	42	2A	*
0101011	43	2B	+
0101100	44	2C	,
0101101	45	2D	-
0101110	46	2E	.
0101111	47	2F	/
0110000	48	30	0
0110001	49	31	1
0110010	50	32	2

Table A-2. (continued)

Binary	Decimal	Hexadecimal	ASCII
0110011	51	33	3
0110100	52	34	4
0110101	53	35	5
0110110	54	36	6
0110111	55	37	7
0111000	56	38	8
0111001	57	39	9
0111010	58	3A	:
0111011	59	3B	;
0111100	60	3C	<
0111101	61	3D	=
0111110	62	3E	>
0111111	63	3F	?
1000000	64	40	@
1000001	65	41	A
1000010	66	42	B
1000011	67	43	C
1000100	68	44	D
1000101	69	45	E
1000110	70	46	F
1000111	71	47	G
1001000	72	48	H
1001001	73	49	I
1001010	74	4A	J
1001011	75	4B	K
1001100	76	4C	L
1001101	77	4D	M
1001110	78	4E	N
1001111	79	4F	O
1010000	80	50	P
1010001	81	51	Q
1010010	82	52	R
1010011	83	53	S
1010100	84	54	T
1010101	85	55	U
1010110	86	56	V
1010111	87	57	W
1011000	88	58	X
1011001	89	59	Y
1011010	90	5A	Z
1011011	91	5B	[
1011100	92	5C	\
1011101	93	5D	]
1011110	94	5E	^
1011111	95	5F	_
1100000	96	60	`
1100001	97	61	a
1100010	98	62	b
1100011	99	63	c
1100100	100	64	d

Table A-2. (continued)

Binary	Decimal	Hexadecimal	ASCII
1100101	101	65	e
1100110	102	66	f
1100111	103	67	g
1101000	104	68	h
1101001	105	69	i
1101010	106	6A	j
1101011	107	6B	k
1101100	108	6C	l
1101101	109	6D	m
1101110	110	6E	n
1101111	111	6F	o
1110000	112	70	p
1110001	113	71	q
1110010	114	72	r
1110011	115	73	s
1110100	116	74	t
1110101	117	75	u
1110110	118	76	v
1110111	119	77	w
1111000	120	78	x
1111001	121	79	y
1111010	122	7A	z
1111011	123	7B	{
1111100	124	7C	|
1111101	125	7D	}
1111110	126	7E	~
1111111	127	7F	DEL

Appendix B
CP/M-86 File Types

CP/M-86 identifies every file by a unique file specification, which consists of a drive specifier, a filename, and a filetype. The filetype is an optional three character ending separated from the filename by a period. The filetype generally indicates a special kind of file. The following table lists common filetypes and their meanings.

Table B-1. Filetypes

Filetype	Indication
A86	Assembly language source file; the CP/M-86 assembler, ASM-86, assembles or translates a file of type .A86 into machine language.
BAK	Back-up file created by a text editor; an editor renames the source file with this filetype to indicate that the original file has been processed. The original file stays on the disk as the back-up file, so you can refer to it.
CMD	Command file that contains instructions in machine executable code.
COM	8080 executable file.
H86	Program file in hexadecimal format.
LST	Printable file that can be displayed on a console or printer.
PRN	Printable file that can be displayed on a console or printer.
SUB	Filetype required for SUBMIT input file containing one or more CP/M-86 commands. The SUBMIT program executes the commands in the file of type SUB providing a batch mode for CP/M-86.
SYM	Symbol table file.
$$$	Temporary file created by PIP.

Appendix C
CP/M-86 Control Characters

Table C-1. CP/M-86 Control Characters

Keystroke	Action
CTRL-C	prompts to abort a program currently running at a given console.
DEL	deletes character to the left of cursor; echoes character deleted - cursor moves right.
CTRL-E	forces a physical carriage return, but does not send command to CP/M-86.
CTRL-H	moves cursor back one space, erases previous character.
CTRL-J	line-feed, terminates input at the console.
CTRL-M	same as carriage return.
CTRL-P	echoes all console activity at the printer; a second CTRL-P ends printer echo. This only works if your system is connected to a printer.
CTRL-R	retypes current command line; useful after using RUB or DEL key.
RETURN	carriage return. (ENTER or ↵ in AS-100)
CTRL-S	stops console listing temporarily; CTRL-S resumes the listing.
CTRL-U	cancels line, displays #, cursor moves down one line and awaits a new command.
CTRL-X	deletes all characters in command line.
CTRL-Z	string or field separator.

Appendix D
CP/M-86 Error Messages

Table D-1. CP/M-86 Command Messages

Message	Meaning
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".
Bad Directory on d: Space Allocation Conflict: User n d:filename.typ	STAT has detected a space allocation conflict in which one data block is assigned to more than one file. One or more filenames might be listed. Each of the files listed contain a data block already allocated to another file on the disk. You can correct the problem by erasing the files listed. After erasing the conflicting file or files, press ↑C to regenerate the allocation vector. If you do not, the error might repeat itself.
BDOS err on d:	CP/M-86 replaces d: with the drive specifier of the drive where the error occurred. This message appears when CP/M-86 finds no disk in the the drive, when the disk is improperly formatted, when the drive latch is open, or when power to the drive is off. Check for one of these situations and retry.

Table D-1. (continued)

Message	Meaning
BDOS err on d: bad sector	This could indicate a hardware problem or a worn or improperly formatted disk. 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: select	CP/M-86 has received a request specifying a non-existent drive, or disk in drive is improperly formatted. CP/M-86 terminates the current program as soon as you press any key.
BDOS err on d: RO	Drive has been assigned Read-Only status with a STAT command, or the disk 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.
Cannot close	ASM-86. 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 disk is in the drive and that the disk is not write protected. 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 disk is in the drive and that the disk is not write protected.

Table D-1. (continued)

Message	Meaning
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, or that the command you requested exists as a .CMD file on the default or specified disk.
DESTINATION IS R/O, DELETE (Y/N)?	PIP. The destination file specified in a PIP command already exists and it is Read-Only. If you type Y, the destination file is deleted before the file copy is done.
Directory full	ASM-86. There is not enough directory space for the output files. You should either erase some unnecessary files or get another disk with more directory space and execute ASM-86 again.
Disk full	ASM-86. There is not enough disk space for the output files (LST, H86 and SYM). You should either erase some unnecessary files or get another disk with more space and execute ASM-86 again.
Disk read error	ASM-86. 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.

Table D-1. (continued)

Message	Meaning
	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.
Disk write error	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 disk with more space and execute ASM-86 again.

Double defined variable	ASM-86. 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`

Double defined label	ASM-86. 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`

Double defined symbol - treated as undefined	ASM-86. The identifier used as the name of an EQU directive is used as a name elsewhere in the program.

Table D-1. (continued)

Message	Meaning
ERROR: BAD PARAMETER	PIP. An illegal parameter has been entered in a PIP command. Retype the entry correctly.
ERROR: CLOSE FILE - {filespec}	PIP. An output file cannot be closed. The user should take appropriate action after checking to see if the correct disk is in the drive and that the disk is not write protected.
ERROR: DISK READ - {filespec}	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.
ERROR: DISK WRITE - {filespec}	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 disk with more space and execute PIP again.
ERROR: FILE NOT FOUND - {filespec}	PIP. An input file that you have specified does not exist.
ERROR: HEX RECORD CHECKSUM - {filespec}	PIP. A hex record checksum 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.

Table D-1. (continued)

Message	Meaning
Error in codemacro building	ASM-86. Either a codemacro contains invalid statements, or a codemacro directive was encountered outside a codemacro.
ERROR: INVALID DESTINATION	PIP. The destination specified in your PIP command is illegal. You have probably specified an input device as a destination.
ERROR: INVALID FORMAT	PIP. The format of your PIP command is illegal. See the description of the PIP command.
ERROR: INVALID HEX DIGIT - {filespec}	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.
ERROR: INVALID SEPARATOR	PIP. You have placed an invalid character for a separator between two input filenames.
ERROR: INVALID SOURCE	PIP. The source specified in your PIP command is illegal. You have probably specified an output device as a source.

Table D-1. (continued)

Message	Meaning
ERROR: INVALID USER NUMBER	PIP. You have specified a User Number greater than 15. User Numbers are in the range 0 to 15.
ERROR: 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 disk with more directory space and execute PIP again.
ERROR: QUIT NOT FOUND	PIP. The string argument to a Q parameter was not found in your input file.
ERROR: START NOT FOUND	PIP. The string argument to an S parameter could not be found in the source file.
ERROR: 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.
ERROR: USER ABORTED	PIP. The user has aborted a PIP operation by pressing a key.

Table D-1. (continued)

Message	Meaning
ERROR: 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 disk or drive.
File exists	You have asked CP/M-86 to create a new file using a file specification that is already assigned to another file. Either delete the existing file or use another file specification.
File name syntax error	ASM-86. The filename in an INCLUDE directive is improperly formed. Example: INCLUDE FILE.A86X
File not found	CP/M-86 could not find the specified file. Check that you have entered the correct drive specification or that you have the correct disk in the drive.
Garbage at end of line - ignored	ASM-86. Additional items were encountered on a line when ASM-86 was expecting an end of line. Examples: NOLIST 4 MOV AX,4 RET

Table D-1. (continued)

Message	Meaning

Illegal expression element	ASM-86. An expression is improperly formed. Examples:
	`X DB 12X`
	`DW (4 * )`

Illegal first item	ASM-86. The first item on a source line is not a valid identifier, directive or mnemonic. Example: 1234H
Illegal "IF" operand - "IF" ignored	ASM-86. Either the expression in an IF statement is not numeric, or it contains a forward reference.
Illegal pseudo instruction	ASM-86. Either a required identifier in front of a pseudo instruction is missing, or an identifier appears before a pseudo instruction that doesn't allow an identifier.

Illegal pseudo operand	ASM-86. The operand in a directive is invalid. Examples:
	`X EQU 0AGH`
	`TITLE UNQUOTED STRING`

Instruction not in code segment	ASM-86. An instruction appears in a segment other than a CSEG.

Table D-1. (continued)

Message	Meaning
Is this what you want to do (Y/N)?	COPYDISK. If the displayed COPYDISK function is what you want performed, type Y.
Insufficient memory	DDT-86. There is not enough memory to load the file specified in an R or E command.
Invalid Assignment	STAT. An invalid device was specified in a STAT device assignment. Use the STAT val: display to list the valid assignments for each of the four logical STAT devices: CON:, AXI:, AXO: and LST:.
Label out of range	ASM-86. The label referred to in a call, jump or loop instruction is out of range. The label can 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.
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.

Table D-1. (continued)

Message	Meaning
Missing instruction	ASM-86. A prefix on a source line is not followed by an instruction. Example: REPNZ
Missing pseudo instruction	ASM-86. The first item on a source line is a valid identifier and the second item is not a valid directive that can be preceded by an identifier. Example: THIS IS A MISTAKE

Missing segment information in operand	ASM-86. 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`

Missing type information in operand(s)	ASM-86. Neither instruction operand contains sufficient type information. Example: `MOV [BX],10`
Nested "IF" illegal - "IF" ignored	ASM-86. The maximum nesting level for IF statements has been exceeded.

Table D-1. (continued)

Message	Meaning
Nested INCLUDE not allowed	ASM-86. An INCLUDE directive was encountered within a file already being included.
No file	CP/M-86 could not find the specified file, or no files exist. ASM-86. The indicated source or include file could not be found on the indicated drive. DDT-86. The file specified in an R or E command could not be found on the disk.
No matching "IF" for "ENDIF"	ASM-86. An ENDIF statement was encountered without a matching IF statement.
No space	DDT-86. There is no space in the directory for the file being written by a W command.

Operand(s) mismatch instruction	ASM-86. Either an instruction has the wrong number of operands, or the types of the operands do not match. Examples:
	
	`        MOV    CX,1,2`
	`X       DB     0`
	`        MOV    AX,X`

Table D-1. (continued)

Message	Meaning
Parameter error	ASM-86. A parameter in the command tail of the ASM-86 command was specified incorrectly. Example: `ASM86 TEST $S;`
Symbol illegally forward referenced - neglected	ASM-86. The indicated symbol was illegally forward referenced in an ORG, RS, EQU or IF statement.
Symbol table overflow	ASM-86. 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.
Undefined element of expression	ASM-86. 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`
Undefined instruction	ASM-86. The item following a label on a source line is not a valid instruction. Example: `DONE: BAD INSTR`

Table D-1. (continued)

Message	Meaning
Use: [size] [ro] [rw] [sys] or [dir]	STAT. This message results from an invalid set file attributes command. These are the only options valid in a STAT filespec [option] command.
Use: STAT d:=RO	STAT. An invalid STAT drive command was given. The only valid drive assignment in STAT is STAT d:=RO.
Too Many Files	STAT. A STAT wildcard command matched more files in the directory than STAT can sort. STAT can sort a maximum of 512 files.
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 E
User's Glossary

ambiguous filename: Filename that contains either of the CP/M-86 wildcard characters, ? or *, in the primary filename or the filetype or both. When you replace characters in a filename with these wildcard characters, you create an ambiguous filename and can easily reference more than one CP/M-86 file in a single command line. See Section 2 of this manual.

applications program: Program that needs an operating system to provide an environment in which to execute. Typical applications programs are business accounting packages, word processing (editing) programs and mailing list programs.

argument: Symbol, usually a letter, indicating a place into which you can substitute a number, letter or name to give an appropriate meaning to the formula in question.

ASCII: The American Standard Code for Information Interchange is a standard code for representation of numbers, letters, and symbols. An ASCII text file is a file that can be intelligibly displayed on the video screen or printed on paper. See Appendix A.

attribute: File characteristic that can be set to on or off.

back-up: Copy of a disk or file made for safe keeping, or the creation of the disk or file.

bit: "Switch" in memory that can be set to on (1) or off (0). Bits are grouped into bytes.

block: Area of disk reserved for a specific use.

bootstrap: Process of loading an operating system into memory. Bootstrap procedures vary from system to system. The boot for an operating system must be customized for the memory size and hardware environment that the operating system manages. Typically, the boot is loaded automatically and executed at power up or when the computer is reset. Sometimes called a "cold start."

buffer: Area of memory that temporarily stores data during the transfer of information.

built-in commands: Commands that permanently reside in memory. They respond quickly because they are not accessed from a disk.

byte: Unit of memory or disk storage containing eight bits.

command: Elements of a CP/M-86 command line. In general, a CP/M-86 command has three parts: the command keyword, the command tail, and a carriage return.

command file: Series of coded machine executable instructions stored on disk as a program file, invoked in CP/M-86 by typing the command keyword next to the system prompt on the console. The CP/M-86 command files generally have a filetype of CMD. Files are either command files or data files. Same as a command program.

command keyword: Name that identifies an CP/M-86 command, usually the primary filename of a file of type CMD, or a built in command. The command keyword precedes the command tail and the carriage return in the command line.

command syntax: Statement that defines the correct way to enter a command. The correct structure generally includes the command keyword, the command tail, and a carriage return. A syntax line usually contains symbols that you should replace with actual values when you enter the command.

command tail: Part of a command that follows the command keyword in the command line. The command tail can include a drive specification, a filename and/or filetype, and options or parameters. Some commands do not require a command tail.

concatenate: Term that describes one of PIP's operations that copies two or more separate files into one new file in the specified sequence.

console: Primary input/output device. The console consists of a listing device such as a screen and a keyboard through which the user communicates with the operating system or applications program.

control character: Non-printing character combination that sends a simple command to CP/M-86. Some control characters perform line editing functions. To enter a control character, hold down the CONTROL key on your terminal and strike the character key specified. See Appendix C.

cursor: One-character symbol that can appear anywhere on the console screen. The cursor indicates the position where the next keystroke at the console will have an effect.

data file: Non-executable collection of similar information that generally requires a command file to manipulate it.

default: Currently selected disk drive and user number. Any command that does not specify a disk drive or a user number references the default disk drive and user number. When CP/M-86 is first invoked, the default disk drive is drive A, and the default user number is 0, until changed with the USER command.

delimiter: Special characters that separate different items in a command line. For example, in CP/M-86, a colon separates the drive specification from the filename. A period separates the filename from the filetype. Brackets separate any options from their command or file specification. Commas separate one item in an option list from another. All of the above special characters are delimiters.

directory: Portion of a disk that contains entries for each file on the disk. In response to the DIR command, CP/M-86 displays the filenames stored in the directory.

DIR attribute: File attribute. A file with the DIR attribute can be displayed by a DIR command. The file can be accessed from the default user number and drive only.

disk, diskette: Magnetic media used to store information. Programs and data are recorded on the disk in the same way that music is recorded on a cassette tape. The term "diskette" refers to smaller capacity removable floppy diskettes. "Disk" can refer to a diskette, a removable cartridge disk or a fixed hard disk.

disk drive: Peripheral device that reads and writes on hard or floppy disks. CP/M-86 assigns a letter to each drive under its control. For example, CP/M-86 may refer to the drives in a four-drive system as A, B, C, and D.

editor: Utility program that creates and modifies text files. An editor can be used for creation of documents or creation of code for computer programs. The CP/M-86 editor is invoked by typing the command ED next to the system prompt on the console. (See ED in Section 5 of this manual).

executable: Ready to be run by the computer. Executable code is a series of instructions that can be carried out by the computer. For example, the computer cannot "execute" names and addresses, but it can execute a program that prints all those names and addresses on mailing labels.

execute a program: Start a program executing. When a program is running, the computer is executing a sequence of instructions.

FCB: File Control Block.

file: Collection of characters, instructions or data stored on a disk. The user can create files on a disk.

File Control Block: Structure used for accessing files on disk. Contains the drive, filename, filetype and other information describing a file to be accessed or created on the disk.

filename: Name assigned to a file. A filename can include a primary filename of 1-8 characters and a filetype of 0-3 characters. A period separates the primary filename from the filetype.

file specification: Unique file identifier. A complete CP/M-86 file specification includes a disk drive specification followed by a colon (d:), a primary filename of 1 to 8 characters, a period and a filetype of 0 to 3 characters. For example, b:example.tex is a complete CP/M-86 file specification.

filetype: Extension to a filename. A filetype can be from 0 to 3 characters and must be separated from the primary filename by a period. A filetype can tell something about the file. Certain programs require that files to be processed have certain filetypes (see Appendix B).

floppy disk: Flexible magnetic disk used to store information. Floppy disks come in 5 1/4 and 8 inch diameters.

hard disk: Rigid, platter-like, magnetic disk sealed in a container. A hard disk stores more information than a floppy disk.

hardware: Physical components of a computer.

hex file: ASCII-printable representation of a command (machine language) file.

hexadecimal notation: Notation for the base 16 number system using the symbols 0,1,2,3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F to represent the sixteen digits. Machine code is often converted to hexadecimal notation because it can be easily represented by ASCII characters and therefore printed on the console screen or on paper (see Appendix A).

input: Data going into the computer, usually from an operator typing at the terminal or by a program reading from the disk.

interface: Object that allows two independent systems to communicate with each other, as an interface between hardware and software in a microcomputer.

I/O: Abbreviation for input/output.

keyword: See **command keyword.**

kilobyte: 1024 bytes denoted as 1K. 32 kilobytes equal 32K. 1024 kilobytes equal one megabyte, or over one million bytes.

list device: Device such as a printer onto which data can be listed or printed.

logged in: Made known to the operating system, in reference to drives. A drive is logged in when it is selected by the user or an executing process, and remains selected or logged in until you change disks in a floppy disk drive or enter ↑C at the command level.

logical: Representation of something that may or may not be the same in its actual physical form. For example, a hard disk can occupy one physical drive, and yet you can divide the available storage on it to appear to the user as if it were in several different drives. These apparent drives are the logical drives.

megabyte: Over one million bytes; 1024 kilobytes (see byte, kilobyte).

microprocessor: Silicon chip that is the Central Processing Unit (CPU) of the microcomputer.

operating system: Collection of programs that supervises the running of other programs and the management of computer resources. An operating system provides an orderly input/output environment between the computer and its peripheral devices. It enables user written programs to execute safely.

option: One of many parameters that can be part of a command tail. Use options to specifiy additional conditions for a command's execution.

output: Data that the processor sends to the console or disk.

parameter: Value in the command tail that provides additional information for the command. Technically, a parameter is a required element of a command.

peripheral devices: Devices external to the CPU. For example, terminals, printers and disk drives are common peripheral devices that are not part of the processor, but are used in conjunction with it.

physical: Actual hardware of a computer. The physical environment varies from computer to computer.

primary filename: First 8 characters of a filename. The primary filename is a unique name that helps the user identify the file contents. A primary filename contains 1 to 8 characters and can include any letter or number and some special characters. The primary filename follows the optional drive specification and precedes the optional filetype.

program: Series of specially coded instructions that performs specific tasks when executed by a computer.

prompt: Any characters displayed on the video screen to help the user decide what the next appropriate action is. A system prompt is a special prompt displayed by the operating system. The system prompt indicates to the user that the operating system is ready to accept input. The CP/M-86 system prompt is an alphabetic character followed by an angle bracket. The alphabetic character indicates the default drive. Some applications programs have their own special "system" prompts.

Read-Only: Attribute that can be assigned to a disk file or a disk drive. When assigned to a file, the Read-Only attribute allows you to read from that file but not write any changes to it. When assigned to a drive, the Read-Only attribute allows you to read any file on the disk, but prevents you from adding a new file, erasing or changing a file, renaming a file, or writing on the disk. The STAT command can set a file or a drive to Read-Only. Every file and drive is either Read-Only or Read-Write. The default setting for drives and files is Read-Write, but an error in resetting the disk or changing media automatically sets the drive to Read-Only until the error is corrected. Files and disk drives may be set to either Read-Only or Read-Write.

Read-Write: Attribute that can be assigned to a disk file or a disk drive. The Read-Write attribute allows you to read from and write to a specific Read-Write file or to a any file on a disk that is in a drive set to Read-Write. A file or drive can be set to either Read-Only or Read-Write.

record: Collection of data. A file consists of one or more records stored on disk. An CP/M-86 record is 128 bytes long.

RO: Abbreviation for Read-Only.

RW: Abbreviation for Read-Write.

sector: Portion of a disk track. There are a specified number of sectors on each track.

software: Specially coded programs that transmit machine readable instructions to the computer, as opposed to hardware, which is the actual physical components of a computer.

source file: ASCII text file that is an input file for a processing program, such as an editor, text formatter, or assembler.

syntax: Format for entering a given command.

system attribute: A file attribute. You can give a file the system attribute by using the SYS option in the STAT command. A file with the SYS attribute is not displayed in response to a DIR command; you must use DIRS (see Section 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. Use this feature to make your commonly used programs available under any user number.

system prompt: Symbol displayed by the operating system indicating that the system is ready to receive input. See prompt.

terminal: See console.

track: Concentric rings dividing a disk. There are 77 tracks on a typical eight inch floppy disk.

turn-key application: Application designed for the non computer-oriented user. For example, a typical turn-key application is designed so that the operator needs only to turn on the computer, insert the proper program disk and select the desired procedure from a selection of functions (menu) displayed on the screen.

upward-compatible: Term meaning that a program created for the previously released operating system (or compiler, etc.) runs under the newly released version of the same operating system.

user number: Number assigned to files in the disk directory so that different users need only deal with their own files and have their "own" directories even though they are all working from the same disk. In CP/M-86, files can be divided into 16 user groups.

utility: "Tool." Program that enables the user to perform certain operations, such as copying files, erasing files, and editing files. Utilities are created for the convenience of programmers and users.

wildcard characters: Special characters that match certain specified items. In CP/M-86 there are two wildcard characters, ? and *. The ? can be substituted for any single character in a filename, and the * can be substituted for the primary filename or the filetype or both. By placing wildcard characters in filenames, the user creates an ambiguous filename and can quickly reference one or more files.

Chapter II

CP/M-86

System Guide

Canon AS-100 Series

CP/M-86™
System Guide

Digital Research
P.O. Box 579
801 Lighthouse Avenue
Pacific Grove, CA 93950
(408) 649-3896
TWX 910 360 5001

COPYRIGHT

This manual is, however, tutorial in nature. Thus, the reader is granted permission to include the example programs, either in whole or in part, in his own programs.

DISCLAIMER

TRADEMARKS

Foreword

The CP/M-86 System Guide presents the system programming aspects of CP/M-86™, a single-user operating system for the Intel 8086 and 8088 16-bit microprocessors. The discussion assumes the reader is familiar with CP/M the Digital Research 8-bit operating system. To clarify specific differences with CP/M-86, this document refers to the 8-bit version of CP/M as CP/M-80™. Elements common to both systems are simply called CP/M features.

CP/M-80 and CP/M-86 are equivalent at the user interface level and thus the Digital Research documents:

- An Introduction to CP/M Features and Facilities
- ED: A Context Editor for the CP/M Disk System
- CP/M 2 User's Guide

are shipped with the CP/M-86 package. Also included is the CP/M-86 Programmer's Guide, which describes ASM-86™ and DDT-86™, Digital Research's 8086 assembler and interactive debugger.

This System Guide presents an overview of the CP/M-86 programming interface conventions. It also describes procedures for adapting CP/M-86 to a custom hardware enviornment. This information parallels that presented in the CP/M 2 Interface Guide and the CP/M 2 Alteration Guide.

Section 1 gives an overview of CP/M-86 and summarizes its differences with CP/M-80. Section 2 describes the general execution environment while Section 3 tells how to generate command files. Sections 4 and 5 respectively define the programming interfaces to the Basic Disk Operating System and the Basic Input/Output System. Section 6 discusses alteration of the BIOS to support custom disk configurations, and Section 7 describes the loading operation and the organization of the CP/M-86 system file.

Table of Contents

Appendixes

```
        ;read the next command line to the conbuf
        mov     dx,offset prompt
        call    print               ;command?
        mov     cl,rstring
        mov     dx,offset conbuf
        call    bdos                ;read command line
;       command line is present, scan it
        mov     ax,0                ;start with 0000
        mov     bx,offset conlin
readc:  mov     dl,[bx]             ;next command character
        inc     bx                  ;to next command positio
        mov     dh,0                ;zero high byte for add
        or      dl,dl               ;check for end of comman
        jnz     getnum
        ret
;       not zero, numeric?
getnum:
        sub     dl,'0'
        cmp     dl,10               ;carry if numeric
        jnb     endrd
        mov     cl,10
        mul     cl                  ;multipy accumulator by
        add     ax,dx               ;+digit
        jmps     readc              ;for another char
endrd:
;       end of read, restore value in a and return value
        mov     dx,ax               ;return value  in DX
        mov     al,-1[bx]
        cmp     al,'a'              ;check for lower case
        jnb     transl
        ret
transl: and     al,5fH  ;translate to upper case
        ret
;
;
; Template for Page 0 of Data Group
;   Contains default FCB and DMA buffer
;
        dseg
        org     05ch
fcb     rb      33                  ;default file control bl
ranrec  rw      1                   ;random record position
ranovf  rb      1                   ;high order (overflow) b
buff    rb      128                 ;default DMA buffer
;
;  string data area for console messages
badver          db      'sorry, you need cp/m version 2$'
nospace         db      'no directory space$'
datmsg          db      'type data: $'
errmsg          db      'error, try again.$'
prompt          db      'next command? $'
;
;
;    fixed and variable data area
;
```

```
conbuf db       conlen  ;length of console buffer
consiz rs       1       ;resulting size after read
conlin rs       32      ;length 32 buffer
conlen equu     offset $ - offset consiz
;
         rs      31       ;16 level stack
stack    rb      1
         db      0        ;end byte for GENCMD
         end
```

Appendix C
Listing of the Boot ROM

```
*******************************************************
*                                                     *
* This is the original BOOT ROM distributed with CP/M *
* for the SBC 86/12 and 204 Controller.  The listing  *
* is truncated on the right, but can be reproduced by *
* assembling ROM.A86 from the distribution disk.  Note *
* that the distributed source file should always be   *
* referenced for the latest version                   *
*                                                     *
*******************************************************
```

```
;
; ROM bootstrap for CP/M-86 on an iSBC86/12
;               with the
;   Intel SBC 204 Floppy Disk Controller
;
;       Copyright (C) 1980,1981
;       Digital Research, Inc.
;       Box 579, Pacific Grove
;       California, 93950
;
;*********************************************
;* This is the BOOT ROM which is initiated   *
;* by a system reset.  First, the ROM moves  *
;* a copy of its data area to RAM at loca-   *
;* tion 00000H, then initializes the segment*
;* registers and the stack pointer.  The     *
;* various peripheral interface chips on the*
;* SBC 86/12 are initialized.  The 8251      *
;* serial interface is configured for a 9600*
;* baud asynchronous terminal, and the in-   *
;* terrupt controller is setup for inter-    *
;* rupts 10H-17H (vectors at 00040H-0005FH)  *
;* and edge-triggered auto-EOI (end of in-   *
;* terrupt) mode with all interrupt levels   *
;* masked-off.  Next, the SBC 204 Diskette   *
;* controller is initialized, and track 1    *
;* sector 1 is read to determine the target  *
;* paragraph address for LOADER.  Finally,   *
;* the LOADER on track 0 sectors 2-26 and    *
;* track 1 sectors 1-26 is read into the     *
;* target address.  Control then transfers   *
;* to LOADER.  This program resides in two   *
;* 2716 EPROM's (2K each) at location        *
;* 0FF000H on the SBC 86/12 CPU board.  ROM  *
;* 0 contains the even memory locations, and*
;* ROM 1 contains the odd addresses.  BOOT   *
;* ROM uses RAM between 00000H and 000FFH    *
;* (absolute) for a scratch area, along with*
;* the sector 1 buffer.                      *
;*********************************************
```

```
00FF            true            equ     0ffh
FF00            false           equ     not true
                ;
00FF            debug           equ     true
                ;debug = true indicates bootstrap is in same roms
                ;with SBC 957 "Execution Vehicle" monitor
                ;at FE00:0 instead of FF00:0
                ;
000D            cr              equ     13
000A            lf              equ     10
                ;
                ;       disk ports and commands
                ;
00A0            base204         equ     0a0h
00A0            fdccom          equ     base204+0
00A0            fdcstat         equ     base204+0
00A1            fdcparm         equ     base204+1
00A1            fdcrslt         equ     base204+1
00A2            fdcrst          equ     base204+2
00A4            dmacadr         equ     base204+4
00A5            dmaccont        equ     base204+5
00A6            dmacscan        equ     base204+6
00A7            dmacsadr        equ     base204+7
00A8            dmacmode        equ     base204+8
00A8            dmacstat        equ     base204+8
00A9            fdcsel          equ     base204+9
00AA            fdcsegment      equ     base204+10
00AF            reset204        equ     base204+15
                ;
                ;actual console baud rate
2580            baud_rate       equ     9600
                ;value for 8253 baud counter
0008            baud            equ     768/(baud_rate/100)
                ;
00DA            csts            equ     0DAh    ;i8251 status port
00D8            cdata           equ     0D8h    ; "    data port
                ;
00D0            tch0            equ     0D0h    ;8253 PIC channel 0
00D2            tch1            equ     tch0+2  ;ch 1 port
00D4            tch2            equ     tch0+4  ;ch 2 port
00D6            tcmd            equ     tch0+6  ;8253 command port
                ;
00C0            icp1            equ     0C0h    ;8259a port 0
00C2            icp2            equ     0C2h    ;8259a port 1
                ;
                ;
                        IF NOT DEBUG
                ROMSEG          EQU     0FF00H  ;normal
                        ENDIF
                ;
                        IF DEBUG                ;share prom with SB
FE00            ROMSEG          EQU     0FE00H
                        ENDIF
                ;
                ;
```

```
                    ;       This long jump prom'd in by hand
                    ;       cseg    0ffffh          ;reset goes to here
                    ;       JMPF    BOTTOM          ;boot is at bottom
                    ;       EA 00 00 00 FF          ;cs = bottom of pro
                    ;                                    ip = 0
                    ;       EVEN PROM       ODD PROM
                    ;       7F8 - EA        7F8 - 00
                    ;       7F9 - 00        7F9 - 00
                    ;       7FA - FF                ;this is not done i
                    ;
  FE00                      cseg    romseg
                    ;
                    ;First, move our data area into RAM at 0000:0200
                    ;
 0000 8CC8                  mov ax,cs
 0002 8ED8                  mov ds,ax       ;point DS to CS for source
 0004 BE3F01                mov SI,drombegin        ;start of data
 0007 BF0002                mov DI,offset ram_start ;offset of destinat
 000A B80000                mov ax,0
 000D 8EC0                  mov es,ax       ;destination segment is 000
 000F B9E600                mov CX,data_length      ;how much to move i
 0012 F3A4                  rep movs al,al          ;move out of eprom
                    ;
 0014 B80000                mov ax,0
 0017 8ED8                  mov ds,ax       ;data segment now in RAM
 0019 8ED0                  mov ss,ax
 001B BC2A03                mov sp,stack_offset     ;Initialize stack s
 001E FC                    cld                     ;clear the directio
                    ;
                            IF NOT DEBUG
                    ;
                    ;Now, initialize the console USART and baud rate
                    ;
                            mov al,0Eh
                            out csts,al     ;give 8251 dummy mode
                            mov al,40h
                            out csts,al     ;reset 8251 to accept mode
                            mov al,4Eh
                            out csts,al     ;normal 8 bit asynch mode,
                            mov al,37h
                            out csts,al     ;enable Tx & Rx
                            mov al,0B6h
                            out tcmd,al     ;8253 ch.2 square wave mode
                            mov ax,baud
                            out tch2,al     ;low of the baud rate
                            mov al,ah
                            out tch2,al     ;high of the baud rate
                    ;
                            ENDIF
                    ;
                    ;Setup the 8259 Programmable Interrupt Controller
                    ;
 001F B013                  mov al,13h
 0021 E6C0                  out icp1,al     ;8259a ICW 1  8086 mode
 0023 B010                  mov al,10h
```

```
0025 E6C2                 out icp2,al      ;8259a ICW 2  vector @ 40-5
0027 B01F                 mov al,1Fh
0029 E6C2                 out icp2,al      ;8259a ICW 4  auto EOI mast
002B B0FF                 mov al,0FFh
002D E6C2                 out icp2,al      ;8259a OCW 1  mask all leve
                  ;
                  ;Reset and initialize the iSBC 204 Diskette Interfa
                  ;
                  restart:          ;also come back here on fatal error
002F E6AF                 out reset204,AL ;reset iSBC 204 logic and
0031 B001                 mov AL,1
0033 E6A2                 out fdcrst,AL    ;give 8271 FDC
0035 B000                 mov al,0
0037 E6A2                 out fdcrst,AL    ;  a reset command
0039 BB1502               mov BX,offset specs1
003C E8E100               CALL sendcom   ;program
003F BB1B02               mov BX,offset specs2
0042 E8DB00               CALL sendcom   ;  Shugart SA-800 drive
0045 BB2102               mov BX,offset specs3
0048 E8D500               call sendcom   ;    characteristics
004B BB1002       homer:  mov BX,offset home
004E E85800               CALL execute     ;home drive 0
                  ;
0051 BB2A03               mov bx,sector1  ;offset for first sector DM
0054 B80000               mov ax,0
0057 8EC0                 mov es,ax        ;segment "     "     "     "
0059 E8A700               call setup_dma
                  ;
005C BB0202               mov bx,offset read0
005F E84700               call execute     ;get T0 S1
                  ;
0062 8E062D03             mov es,ABS
0066 BB0000               mov bx,0         ;get loader load address
0069 E89700               call setup_dma  ;setup DMA to read loader
                  ;
006C BB0602               mov bx,offset read1
006F E83700               call execute     ;read track 0
0072 BB0B02               mov bx,offset read2
0075 E83100               call execute     ;read track 1
                  ;
0078 8C06E802             mov leap_segment,ES
                  ;       setup far jump vector
007C C706E6020000         mov leap_offset,0
                  ;
                  ;       enter LOADER
0082 FF2EE602             jmpf dword ptr leap_offset
                  ;
                  pmsg:
0086 8A0F                 mov cl,[BX]
0088 84C9                 test cl,cl
008A 7476                 jz return
008C E80400               call conout
008F 43                   inc BX
0090 E9F3FF               jmp pmsg
                  ;
```

```
                        conout:
0093 E4DA                       in al,csts
0095 A801                       test al,1
0097 74FA                       jz conout
0099 8AC1                       mov al,cl
009B E6D8                       out cdata,al
009D C3                         ret
                        ;
                        conin:
009E E4DA                       in al,csts
00A0 A802                       test al,2
00A2 74FA                       jz conin
00A4 E4D8                       in al,cdata
00A6 247F                       and al,7Fh
00A8 C3                         ret
                        ;
                        ;
                        ;
                        execute:        ;execute command string @ [BX]
                                        ;<BX> points to length,
                                        ;followed by Command byte
                                        ;followed by length-1 parameter byt
                        ;
00A9 891E0002                   mov     lastcom,BX      ;remember what it w
                        retry:                          ;retry if not ready
00AD E87000                     call    sendcom         ;execute the comman
                                                        ;now, let's see wha
                                                        ;of status poll was
                                                        ;for that command t
00B0 8B1E0002                   mov     BX,lastcom      ;point to command s
00B4 8A4701                     mov     AL,1[BX]        ;get command op cod
00B7 243F                       and     AL,3fh          ;drop drive code bi
00B9 B90008                     mov     CX,0800h        ;mask if it will be
00BC 3C2C                       cmp     AL,2ch          ;see if interrupt t
00BE 720B                       jb      execpoll
00C0 B98080                     mov     CX,8080h        ;else we use "not c
00C3 240F                       and     AL,0fh          ;unless . . .
00C5 3C0C                       cmp     AL,0ch          ;there isn't
00C7 B000                       mov AL,0
00C9 7737                       ja return               ;any result at all
                        ;
                        execpoll:       ;poll for bit in b, toggled with c
00CB E4A0                       in AL,FDCSTAT
00CD 22C5                       and AL,CH
00CF 32C174F8                   xor AL,CL ! JZ execpoll
                        ;
00D3 E4A1                       in      AL,fdcrslt      ;get result registe
00D5 241E                       and     AL,1eh          ;look only at resul
00D7 7429                       jz      return          ;zero means it was
                        ;
00D9 3C10                       cmp al,10h
00DB 7513                       jne fatal               ;if other than "Not
                        ;
00DD BB1302                     mov bx,offset rdstat
00E0 E83D00                     call sendcom            ;perform read statu
```

```
                    rd_poll:
00E3 E4A0                   in al,fdc_stat
00E5 A880                   test al,80h             ;wait for command n
00E7 75FA                   jnz rd_poll
00E9 8B1E0002               mov bx,last_com         ;recover last attem
00ED E9BDFF                 jmp retry               ;and try it over ag
                    ;
                    fatal:                          ; fatal error
00F0 B400                   mov ah,0
00F2 8BD8                   mov bx,ax               ;make 16 bits
00F4 8B9F2702               mov bx,errtbl[BX]
                    ;       print appropriate error message
00F8 E88BFF                 call pmsg
00FB E8A0FF                 call conin              ;wait for key strik
00FE 58                     pop ax                  ;discard unused ite
00FF E92DFF                 jmp restart             ;then start all ove
                    ;
                    return:
0102 C3                     RET                     ;return from EXECUT
                    ;
                    setupdma:
0103 B004                   mov AL,04h
0105 E6A8                   out dmacmode,AL         ;enable dmac
0107 B000                   mov al,0
0109 E6A5                   out dmaccont,AL         ;set first (dummy)
010B B040                   mov AL,40h
010D E6A5                   out dmaccont,AL         ;force read data mo
010F 8CC0                   mov AX,ES
0111 E6AA                   out fdcsegment,AL
0113 8AC4                   mov AL,AH
0115 E6AA                   out fdcsegment,AL
0117 8BC3                   mov AX,BX
0119 E6A4                   out dmacadr,AL
011B 8AC4                   mov AL,AH
011D E6A4                   out dmacadr,AL
011F C3                     RET
                    ;
                    ;
                    ;
                    sendcom:        ;routine to send a command string t
0120 E4A0                   in AL,fdcstat
0122 2480                   and AL,80h
0124 75FA                   jnz sendcom     ;insure command not busy
0126 8A0F                   mov CL,[BX]     ;get count
0128 43                     inc BX
0129 8A07                   mov al,[BX]     ;point to and fetch command
012B E6A0                   out fdccom,AL   ;send command
                    parmloop:
012D FEC9                   dec CL
012F 74D1                   jz return       ;see if any (more) paramete
0131 43                     inc BX          ;point to next parameter
                    parmpoll:
0132 E4A0                   in AL,fdcstat
0134 2420                   and AL,20h
0136 75FA                   jnz parmpoll    ;loop until parm not full
```

```
0138 8A07                   mov AL,[BX]
013A E6A1                   out fdcparm,AL  ;output next parameter
013C E9EEFF                 jmp parmloop    ;go see about another
                    ;
                    ;
                    ;       Image of data to be moved to RAM
                    ;
 013F               drombegin equ offset $
                    ;
013F 0000           clastcom        dw      0000h   ;last command
                    ;
0141 03             creadstring     db      3       ;length
0142 52                             db      52h     ;read function code
0143 00                             db      0       ;track #
0144 01                             db      1       ;sector #
                    ;
0145 04             creadtrk0       db      4
0146 53                             db      53h     ;read multiple
0147 00                             db      0       ;track 0
0148 02                             db      2       ;sectors 2
0149 19                             db      25      ;through 26
                    ;
014A 04             creadtrk1       db      4
014B 53                             db      53h
014C 01                             db      1       ;track 1
014D 01                             db      1       ;sectors 1
014E 1A                             db      26      ;through 26
                    ;
014F 026900         chome0          db      2,69h,0
0152 016C           crdstat0        db      1,6ch
0154 05350D         cspecs1         db      5,35h,0dh
0157 0808E9                         db      08h,08h,0e9h
015A 053510         cspecs2         db      5,35h,10h
015D FFFFFF                         db      255,255,255
0160 053518         cspecs3         db      5,35h,18h
0163 FFFFFF                         db      255,255,255
                    ;
0166 4702           cerrtbl dw      offset er0
0168 4702                   dw      offset er1
016A 4702                   dw      offset er2
016C 4702                   dw      offset er3
016E 5702                   dw      offset er4
0170 6502                   dw      offset er5
0172 7002                   dw      offset er6
0174 7F02                   dw      offset er7
0176 9002                   dw      offset er8
0178 A202                   dw      offset er9
017A B202                   dw      offset erA
017C C502                   dw      offset erB
017E D302                   dw      offset erC
0180 4702                   dw      offset erD
0182 4702                   dw      offset erE
0184 4702                   dw      offset erF
                    ;
0186 0D0A4E756C6C Cer0      db      cr,lf,'Null Error ??',0
```

```
     204572726F72
     203F3F00
  0186              Cer1      equ       cer0
  0186              Cer2      equ       cer0
  0186              Cer3      equ       cer0
0196 0D0A436C6F63 Cer4      db        cr,lf,'Clock Error',0
     6B204572726F
     7200
01A4 0D0A4C617465 Cer5      db        cr,lf,'Late DMA',0
     20444D4100
01AF 0D0A49442043 Cer6      db        cr,lf,'ID CRC Error',0
     524320457272
     6F7200
01BE 0D0A44617461 Cer7      db        cr,lf,'Data CRC Error',0
     204352432045
     72726F7200
01CF 0D0A44726976 Cer8      db        cr,lf,'Drive Not Ready',0
     65204E6F7420
     526561647900
01E1 0D0A57726974 Cer9      db        cr,lf,'Write Protect',0
     652050726F74
     65637400
01F1 0D0A54726B20 CerA      db        cr,lf,'Trk 00 Not Found',0
     3030204E6F74
     20466F756E64
     00
0204 0D0A57726974 CerB      db        cr,lf,'Write Fault',0
     65204661756C
     7400
0212 0D0A53656374 CerC      db        cr,lf,'Sector Not Found',0
     6F72204E6F74
     20466F756E64
     00
  0186              CerD      equ       cer0
  0186              CerE      equ       cer0
  0186              CerF      equ       cer0
                    ;
  0225              dromend equ offset $
                    ;
  00E6              data_length     equ dromend-drombegin
                    ;
                    ;         reserve space in RAM for data area
                    ;         (no hex records generated here)
                    ;
  0000                        dseg      0
                              org       0200h
                    ;
  0200              ram_start           equ       $
0200                lastcom             rw        1         ;last command
0202                read0               rb        4         ;read track 0 secto
0206                read1               rb        5         ;read T0 S2-26
020B                read2               rb        5         ;read T1 S1-26
0210                home                rb        3         ;home drive 0
0213                rdstat              rb        2         ;read status
0215                specs1              rb        6
```

```
021B        specs2          rb      6
0221        specs3          rb      6
0227        errtbl          rw      16
0247        er0             rb      length cer0     ;16
  0247      er1             equ     er0
  0247      er2             equ     er0
  0247      er3             equ     er0
0257        er4             rb      length cer4     ;14
0265        er5             rb      length cer5     ;11
0270        er6             rb      length cer6     ;15
027F        er7             rb      length cer7     ;17
0290        er8             rb      length cer8     ;18
02A2        er9             rb      length cer9     ;16
02B2        erA             rb      length cerA     ;19
02C5        erB             rb      length cerB     ;14
02D3        erC             rb      length cerC     ;19
  0247      erD             equ     er0
  0247      erE             equ     er0
  0247      erF             equ     er0
                            ;
02E6        leap_offset     rw      1
02E8        leap_segment    rw      1
                            ;
                            ;
02EA                        rw      32      ;local stack
  032A      stack_offset    equ     offset $;stack from here do
                            ;
                            ;       TO S1 read in here
  032A      sector1         equ offset $
                            ;
032A        Ty              rb      1
032B        Len             rw      1
032D        Abs             rw      1       ;ABS is all we care
032F        Min             rw      1
0331        Max             rw      1
                                    end
```

Appendix D
LDBIOS Listing

```
****************************************************
*                                                  *
* This the the LOADER BIOS, derived from the BIOS *
* program by enabling the "loader_bios" condi-     *
* tional assembly switch.  The listing has been    *
* edited to remove portions which are duplicated   *
* in the BIOS listing which appears in Appendix D *
* where elipses "..." denote the deleted portions *
* (the listing is truncated on the right, but can *
* be reproduced by assembling the BIOS.A86 file    *
* provided with CP/M-86)                           *
*                                                  *
****************************************************
```

```
                        ;*********************************************
                        ;*                                           *
                        ;* Basic Input/Output System (BIOS) for      *
                        ;* CP/M-86 Configured for iSBC 86/12 with    *
                        ;* the iSBC 204 Floppy Disk Controller       *
                        ;*                                           *
                        ;* (Note:  this file contains both embedded  *
                        ;* tabs and blanks to minimize the list file *
                        ;* width for printing purposes.  You may wish*
                        ;* to expand the blanks before performing    *
                        ;* major editing.)                           *
                        ;*********************************************

                        ;       Copyright (C) 1980,1981
                        ;       Digital Research, Inc.
                        ;       Box 579, Pacific Grove
                        ;       California, 93950
                        ;
                        ;       (Permission is hereby granted to use
                        ;       or abstract the following program in
                        ;       the implementation of CP/M, MP/M or
                        ;       CP/NET for the 8086 or 8088 Micro-
                        ;       processor)

  FFFF                  true            equ -1
  0000                  false           equ not true
```

```
                    ;*********************************************
                    ;*                                           *
                    ;* Loader_bios is true if assembling the     *
                    ;* LOADER BIOS, otherwise BIOS is for the    *
                    ;* CPM.SYS file.  Blc_list is true if we     *
                    ;* have a serial printer attached to BLC8538 *
                    ;* Bdos_int is interrupt used for earlier    *
                    ;* versions.                                 *
                    ;*                                           *
                    ;*********************************************

FFFF                loader_bios     equ true
FFFF                blc_list        equ true
00E0                bdos_int        equ 224 ;reserved BDOS Interrupt

                            IF      not loader_bios
                    ;---------------------------------------------
                    ;|                                           |
                            . . .
                    ;|                                           |
                    ;---------------------------------------------
                            ENDIF   ;not loader_bios

                            IF      loader_bios
                    ;---------------------------------------------
                    ;|                                           |
1200                bios_code       equ 1200h ;start of LDBIOS
0003                ccp_offset      equ 0003h ;base of CPMLOADER
0406                bdos_ofst       equ 0406h ;stripped BDOS entry
                    ;|                                           |
                    ;---------------------------------------------
                            ENDIF   ;loader_bios
                            . . .

                            cseg
                            org     ccpoffset
                    ccp:
                            org     bios_code

                    ;*********************************************
                    ;*                                           *
                    ;* BIOS Jump Vector for Individual Routines  *
                    ;*                                           *
                    ;*********************************************

1200 E93C00          jmp INIT        ;Enter from BOOT ROM or LOADER
1203 E96100          jmp WBOOT       ;Arrive here from BDOS call 0
                            . . .
1239 E96400          jmp GETIOBF     ;return I/O map byte (IOBYTE)
123C E96400          jmp SETIOBF     ;set I/O map byte (IOBYTE)
```

```
                         ;*********************************************
                         ;*                                           *
                         ;* INIT Entry Point, Differs for LDBIOS and  *
                         ;* BIOS, according to "Loader_Bios" value    *
                         ;*                                           *
                         ;*********************************************

                         INIT:   ;print signon message and initialize hardwa
123F 8CC8                        mov ax,cs       ;we entered with a JMPF so
1241 8ED0                        mov ss,ax       ; CS: as the initial value
1243 8ED8                        mov ds,ax       ;       DS:,
1245 8EC0                        mov es,ax       ;       and ES:
                                 ;use local stack during initialization
1247 BCA916                      mov sp,offset stkbase
124A FC                          cld             ;set forward direction

                                 IF      not loader_bios
                         ;---------------------------------------------
                         ;|                                           |
                                 ; This is a BIOS for the CPM.SYS file.
                         ;|      . . .                                |
                         ;---------------------------------------------
                                 ENDIF   ;not loader_bios

                                 IF      loader_bios
                         ;---------------------------------------------
                         ;|                                           |
                                 ;This is a BIOS for the LOADER
124B 1E                          push ds         ;save data segment
124C B80000                      mov ax,0
124F 8ED8                        mov ds,ax       ;point to segment zero
                                 ;BDOS interrupt offset
1251 C70680030604                mov bdos_offset,bdos_ofst
1257 8C0E8203                    mov bdos_segment,CS ;bdos interrupt segment
125B 1F                          pop ds          ;restore data segment
                         ;|                                           |
                         ;---------------------------------------------
                                 ENDIF   ;loader_bios

125C BB1514                      mov bx,offset signon
125F E85A00                      call pmsg       ;print signon message
1262 B100                        mov cl,0        ;default to dr A: on coldst
1264 E99CED                      jmp ccp         ;jump to cold start entry o

1267 E99FED              WBOOT:  jmp ccp+6       ;direct entry to CCP at com

                                 IF      not loader_bios
                         ;---------------------------------------------
                         ;|                                           |
                                 . . .
                         ;|                                           |
                         ;---------------------------------------------
                                 ENDIF   ;not loader_bios
```

```
                 ;*********************************************
                 ;*                                           *
                 ;*   CP/M Character I/O Interface Routines   *
                 ;*   Console is Usart (i8251a) on iSBC 86/12 *
                 ;*   at ports D8/DA                          *
                 ;*                                           *
                 ;*********************************************

                 CONST:          ;console status
126A E4DA               in al,csts
                        . . .
                 const_ret:
1272 C3                 ret                 ;Receiver Data Available

                 CONIN:                     ;console input
1273 E8F4FF             call const
                        . . .
                  CONOUT:         ;console output
127D E4DA               in al,csts
                        . . .

                 LISTOUT:                   ;list device output

                        IF      blc_list
                 ;---------------------------------------------
                 ;|                                           |
1288 E80700             call LISTST
                          . . .
                 ;|                                           |
                 ;---------------------------------------------
                        ENDIF   ;blc_list

1291 C3                 ret

                 LISTST:                    ;poll list status

                        IF      blc_list
                 ;---------------------------------------------
                 ;|                                           |
1292 E441               in al,lsts
                        . . .
                 ;|                                           |
                 ;---------------------------------------------
                        ENDIF   ;blc_list

129C C3                 ret

                 PUNCH:  ;not implemented in this configuration
                 READER:
129D B01A               mov al,lah
129F C3                 ret                 ;return EOF for now
```

```
                         GETIOBF:
12A0 B000                        mov al,0        ;TTY: for consistency
12A2 C3                          ret             ;IOBYTE not implemented

                         SETIOBF:
12A3 C3                          ret             ;iobyte not implemented

                         zero_ret:
12A4 2400                        and al,0
12A6 C3                  ret                     ;return zero in AL and flag

                         ; Routine to get and echo a console character
                         ;       and shift it to upper case

                         uconecho:
12A7 E8C9FF                      call CONIN      ;get a console character
                                 . . .
                         ;*********************************************
                         ;*                                           *
                         ;*          Disk Input/Output Routines       *
                         ;*                                           *
                         ;*********************************************

                         SELDSK:         ;select disk given by register CL
12CA BB0000                      mov bx,0000h
                                 . . .

                         HOME:   ;move selected disk to home position (Track
12EB C606311500                  mov trk,0       ;set disk i/o to track zero
                                 . . .

                         SETTRK: ;set track address given by CX
1300 880E3115                    mov trk,cl      ;we only use 8 bits of trac
1304 C3                          ret

                         SETSEC: ;set sector number given by cx
1305 880E3215                    mov sect,cl     ;we only use 8 bits of sect
1309 C3                          ret

                         SECTRAN: ;translate sector CX using table at [DX]
130A 8BD9                        mov bx,cx
                                 . . .

                         SETDMA: ;set DMA offset given by CX
1311 890E2A15                    mov dma_adr,CX
1315 C3                          ret

                         SETDMAB: ;set DMA segment given by CX
1316 890E2C15                    mov dma_seg,CX
131A C3                          ret
                         ;
                         GETSEGT:  ;return address of physical memory table
131B BB3815                      mov bx,offset seg_table
131E C3                          ret
```

```
                    ;*********************************************
                    ;*                                           *
                    ;*  All disk I/O parameters are setup:  the  *
                    ;*  Read and Write entry points transfer one *
                    ;*  sector of 128 bytes to/from the current  *
                    ;*  DMA address using the current disk drive *
                    ;*                                           *
                    ;*********************************************

                    READ:
131F B012                   mov al,12h       ;basic read sector command
1321 EB02                   jmps r_w_common

                    WRITE:
1323 B00A                   mov al,0ah       ;basic write sector command

                    r_w_common:
1325 BB2F15                 mov bx,offset io_com ;point to command stri
                            . . .

                    ;*********************************************
                    ;*                                           *
                    ;*              Data Areas                   *
                    ;*                                           *
                    ;*********************************************
  1415              data_offset     equ offset $

                            dseg
                            org     data_offset      ;contiguous with co

                            IF      loader_bios
                    ;---------------------------------------------
                    ;|                                           |
1415 0D0A0D0A       signon  db      cr,lf,cr,lf
1419 43502F4D2D38           db      'CP/M-86 Version 2.2',cr,lf,0
     362056657273
     696F6E20322E
     320D0A00
                    ;|                                           |
                    ;---------------------------------------------
                            ENDIF   ;loader_bios

                            IF      not loader_bios
                    ;---------------------------------------------
                    ;|                                           |
                            . . .
                    ;|                                           |
                    ;---------------------------------------------
                            ENDIF   ;not loader_bios

142F 0D0A486F6D65 bad_hom db        cr,lf,'Home Error',cr,lf,0
                            . . .
=                           include singles.lib ;read in disk definitio
=                   ;               DISKS 2
```

```
=  1541          dpbase  equ     $               ;Base of Disk Param
                         . . .
=1668 00                 db      0               ;Marks End of Modul


 1669            loc_stk rw  32  ;local stack for initialization
   16A9          stkbase equ offset $

                         . . .
 16A9 00                 db 0    ;fill last address for GENCMD

                 ;*********************************************
                 ;*                                           *
                 ;*          Dummy Data Section               *
                 ;*                                           *
                 ;*********************************************
   0000                  dseg    0       ;absolute low memory
                         org     0       ;(interrupt vectors)
                         . . .
                         END
```

Appendix E
BIOS Listing

```
****************************************************
*                                                  *
* This is the CP/M-86 BIOS, derived from the BIOS *
* program by disabling the "loader_bios" condi-    *
* tional assembly switch.  The listing has been    *
* truncated on the right, but can be reproduced    *
* by assembling the BIOS.A86 file provided with    *
* CP/M-86.  This BIOS allows CP/M-86 operation     *
* with the Intel SBC 86/12 with the SBC 204 con-   *
* troller.  Use this BIOS, or the skeletal CBIOS   *
* listed in Appendix E, as the basis for a cus-    *
* tomized implementation of CP/M-86.               *
* provided with CP/M-86)                           *
*                                                  *
****************************************************
```

```
                        ;*********************************************
                        ;*                                           *
                        ;* Basic Input/Output System (BIOS) for      *
                        ;* CP/M-86 Configured for iSBC 86/12 with    *
                        ;* the iSBC 204 Floppy Disk Controller       *
                        ;*                                           *
                        ;* (Note:  this file contains both embedded  *
                        ;* tabs and blanks to minimize the list file *
                        ;* width for printing purposes.  You may wish*
                        ;* to expand the blanks before performing    *
                        ;* major editing.)                           *
                        ;*********************************************

                        ;       Copyright (C) 1980,1981
                        ;       Digital Research, Inc.
                        ;       Box 579, Pacific Grove
                        ;       California, 93950
                        ;
                        ;       (Permission is hereby granted to use
                        ;       or abstract the following program in
                        ;       the implementation of CP/M, MP/M or
                        ;       CP/NET for the 8086 or 8088 Micro-
                        ;       processor)

   FFFF                 true            equ -1
   0000                 false           equ not true
```

```
                ;*********************************************
                ;*                                           *
                ;* Loader_bios is true if assembling the     *
                ;* LOADER_BIOS, otherwise BIOS is for the    *
                ;* CPM.SYS file.  Blc_list is true if we     *
                ;* have a serial printer attached to BLC8538 *
                ;* Bdos_int is interrupt used for earlier    *
                ;* versions.                                 *
                ;*                                           *
                ;*********************************************

0000            loader_bios     equ false
FFFF            blc_list        equ true
00E0            bdos_int        equ 224 ;reserved BDOS Interrupt

                        IF      not loader_bios
                ;---------------------------------------------
                ;|                                           |
2500            bios_code       equ 2500h
0000            ccp_offset      equ 0000h
0B06            bdos_ofst       equ 0B06h ;BDOS entry point
                ;|                                           |
                ;---------------------------------------------
                        ENDIF   ;not loader_bios

                        IF      loader_bios
                ;---------------------------------------------
                ;|                                           |
                bios_code       equ 1200h ;start of LDBIOS
                ccp_offset      equ 0003h ;base of CPMLOADER
                bdos_ofst       equ 0406h ;stripped BDOS entry
                ;|                                           |
                ;---------------------------------------------
                        ENDIF   ;loader_bios

00DA            csts            equ 0DAh  ;i8251 status port
00D8            cdata           equ 0D8h  ;   "  data port

                        IF      blc_list
                ;---------------------------------------------
                ;|                                           |
0041            lsts            equ 41h ;2651 No. 0 on BLC8538 stat
0040            ldata           equ 40h ; "    "   "  "    "   data
0060            blc_reset       equ 60h ;reset selected USARTS on B
                ;|                                           |
                ;---------------------------------------------
                        ENDIF   ;blc_list

                ;*********************************************
                ;*                                           *
                ;*    Intel iSBC 204 Disk Controller Ports   *
                ;*                                           *
                ;*********************************************
```

```
00A0            base204         equ 0a0h        ;SBC204 assigned ad

00A0            fdc_com         equ base204+0   ;8271 FDC out comma
00A0            fdc_stat        equ base204+0   ;8271 in status
00A1            fdc_parm        equ base204+1   ;8271 out parameter
00A1            fdc_rslt        equ base204+1   ;8271 in result
00A2            fdc_rst         equ base204+2   ;8271 out reset
00A4            dmac_adr        equ base204+4   ;8257 DMA base addr
00A5            dmac_cont       equ base204+5   ;8257 out control
00A6            dmac_scan       equ base204+6   ;8257 out scan cont
00A7            dmac_sadr       equ base204+7   ;8257 out scan addr
00A8            dmac_mode       equ base204+8   ;8257 out mode
00A8            dmac_stat       equ base204+8   ;8257 in status
00A9            fdc_sel         equ base204+9   ;FDC select port (n
00AA            fdc_segment     equ base204+10  ;segment address re
00AF            reset_204       equ base204+15  ;reset entire inter

000A            max_retries     equ 10          ;max retries on dis
                                                ;before perm error
000D            cr              equ 0dh         ;carriage return
000A            lf              equ 0ah         ;line feed

                        cseg
                        org     ccpoffset
                ccp:
                        org     bios_code

                ;*********************************************
                ;*                                           *
                ;* BIOS Jump Vector for Individual Routines  *
                ;*                                           *
                ;*********************************************

2500 E93C00       jmp INIT        ;Enter from BOOT ROM or LOADER
2503 E98400       jmp WBOOT       ;Arrive here from BDOS call 0
2506 E99000       jmp CONST       ;return console keyboard status
2509 E99600       jmp CONIN       ;return console keyboard char
250C E99D00       jmp CONOUT      ;write char to console device
250F E9A500       jmp LISTOUT     ;write character to list device
2512 E9B700       jmp PUNCH       ;write character to punch device
2515 E9B400       jmp READER      ;return char from reader device
2518 E9FF00       jmp HOME        ;move to trk 00 on cur sel drive
251B E9DB00       jmp SELDSK      ;select disk for next rd/write
251E E90E01       jmp SETTRK      ;set track for next rd/write
2521 E91001       jmp SETSEC      ;set sector for next rd/write
2524 E91901       jmp SETDMA      ;set offset for user buff (DMA)
2527 E92401       jmp READ        ;read a 128 byte sector
252A E92501       jmp WRITE       ;write a 128 byte sector
252D E99100       jmp LISTST      ;return list status
2530 E90601       jmp SECTRAN     ;xlate logical->physical sector
2533 E90F01       jmp SETDMAB     ;set seg base for buff (DMA)
2536 E91101       jmp GETSEGT     ;return offset of Mem Desc Table
2539 E99300       jmp GETIOBF     ;return I/O map byte (IOBYTE)
253C E99300       jmp SETIOBF     ;set I/O map byte (IOBYTE)
```

```
                        ;*******************************************
                        ;*                                         *
                        ;* INIT Entry Point, Differs for LDBIOS and *
                        ;* BIOS, according to "Loader_Bios" value   *
                        ;*                                         *
                        ;*******************************************

                        INIT:   ;print signon message and initialize hardwa
253F 8CC8                       mov ax,cs         ;we entered with a JMPF so
2541 8ED0                       mov ss,ax         ; CS: as the initial value
2543 8ED8                       mov ds,ax         ;         DS:,
2545 8EC0                       mov es,ax         ;         and ES:
                                ;use local stack during initialization
2547 BCE429                     mov sp,offset stkbase
254A FC                         cld               ;set forward direction

                                IF      not loader_bios
                        ;-------------------------------------------
                        ;|                                         |
                                ; This is a BIOS for the CPM.SYS file.
                                ; Setup all interrupt vectors in low
                                ; memory to address trap

254B 1E                         push ds           ;save the DS register
254C B80000                     mov ax,0
254F 8ED8                       mov ds,ax
2551 8EC0                       mov es,ax         ;set ES and DS to zero
                                ;setup interrupt 0 to address trap routine
2553 C70600008D25               mov int0_offset,offset int_trap
2559 8C0E0200                   mov int0_segment,CS
255D BF0400                     mov di,4
2560 BE0000                     mov si,0          ;then propagate
2563 B9FE01                     mov cx,510        ;trap vector to
2566 F3A5                       rep movs ax,ax    ;all 256 interrupts
                                ;BDOS offset to proper interrupt
2568 C7068003060B               mov bdos_offset,bdos_ofst
256E 1F                         pop ds            ;restore the DS register

                        ;*******************************************
                        ;*                                         *
                        ;* National "BLC 8538" Channel 0 for a serial*
                        ;* 9600 baud printer - this board uses 8 Sig-*
                        ;* netics 2651 Usarts which have on-chip baud*
                        ;* rate generators.                        *
                        ;*                                         *
                        ;*******************************************

256F B0FF                       mov al,0FFh
2571 E660                       out blc_reset,al ;reset all usarts on 8538
2573 B04E                       mov al,4Eh
2575 E642                       out ldata+2,al   ;set usart 0 in async 8 bit
2577 B03E                       mov al,3Eh
2579 E642                       out ldata+2,al   ;set usart 0 to 9600 baud
257B B037                       mov al,37h
257D E643                       out ldata+3,al   ;enable Tx/Rx, and set up
```

```
                    ;|                                        |
                    ;-----------------------------------------
                          ENDIF   ;not loader_bios

                          IF      loader_bios
                    ;-----------------------------------------
                    ;|                                        |
                          ;This is a BIOS for the LOADER
                          push ds           ;save data segment
                          mov ax,0
                          mov ds,ax         ;point to segment zero
                          ;BDOS interrupt offset
                          mov bdos_offset,bdos_ofst
                          mov bdos_segment,CS ;bdos interrupt segment
                          pop ds            ;restore data segment
                    ;|                                        |
                    ;-----------------------------------------
                          ENDIF   ;loader_bios

257F BB4427               mov bx,offset signon
2582 E86600               call pmsg         ;print signon message
2585 B100                 mov cl,0          ;default to dr A: on coldst
2587 E976DA               jmp ccp           ;jump to cold start entry o

258A E979DA         WBOOT: jmp ccp+6        ;direct entry to CCP at com

                          IF      not loader_bios
                    ;-----------------------------------------
                    ;|                                        |
                    int_trap:
258D FA                   cli               ;block interrupts
258E 8CC8                 mov ax,cs
2590 8ED8                 mov ds,ax         ;get our data segment
2592 BB7927               mov bx,offset int_trp
2595 E85300               call pmsg
2598 F4                   hlt               ;hardstop
                    ;|                                        |
                    ;-----------------------------------------
                          ENDIF   ;not loader_bios

                    ;*****************************************
                    ;*                                       *
                    ;*   CP/M Character I/O Interface Routines   *
                    ;*   Console is Usart (i8251a) on iSBC 86/12 *
                    ;*   at ports D8/DA                      *
                    ;*                                       *
                    ;*****************************************

                    CONST:            ;console status
2599 E4DA                 in al,csts
259B 2402                 and al,2
259D 7402                 jz const_ret
259F 0CFF                 or al,255         ;return non-zero if RDA
                    const_ret:
25A1 C3                   ret               ;Receiver Data Available
```

```
                        CONIN:                          ;console input
25A2 E8F4FF                     call const
25A5 74FB                       jz CONIN                ;wait for RDA
25A7 E4D8                       in al,cdata
25A9 247F                       and al,7fh              ;read data and remove parit
25AB C3                         ret

                        CONOUT:         ;console output
25AC E4DA                       in al,csts
25AE 2401                       and al,1                ;get console status
25B0 74FA                       jz CONOUT               ;wait for TBE
25B2 8AC1                       mov al,cl
25B4 E6D8                       out cdata,al            ;Transmitter Buffer Empty
25B6 C3                         ret                     ;then return data

                        LISTOUT:                        ;list device output

                                IF      blc_list
                        ;---------------------------------------------
                        ;|                                           |
25B7 E80700                     call LISTST
25BA 74FB                       jz LISTOUT              ;wait for printer not busy
25BC 8AC1                       mov al,cl
25BE E640                       out ldata,al            ;send char to TI 810
                        ;|                                           |
                        ;---------------------------------------------
                                ENDIF   ;blc_list

25C0 C3                         ret

                        LISTST:                         ;poll list status

                                IF      blc_list
                        ;---------------------------------------------
                        ;|                                           |
25C1 E441                       in al,lsts
25C3 2481                       and al,81h              ;look at both TxRDY and DTR
25C5 3C81                       cmp al,81h
25C7 750A                       jnz zero_ret            ;either false, printer is b
25C9 0CFF                       or al,255               ;both true, LPT is ready
                        ;|                                           |
                        ;---------------------------------------------
                                ENDIF   ;blc_list

25CB C3                         ret

                        PUNCH:  ;not implemented in this configuration
                        READER:
25CC B01A                       mov al,lah
25CE C3                         ret                     ;return EOF for now

                        GETIOBF:
25CF B000                       mov al,0                ;TTY: for consistency
25D1 C3                         ret                     ;IOBYTE not implemented
```

```
                          SETIOBF:
 25D2 C3                          ret                 ;iobyte not implemented

                          zero_ret:
 25D3 2400                        and al,0
 25D5 C3                  ret                         ;return zero in AL and flag

                          ; Routine to get and echo a console character
                          ;       and shift it to upper case

                          uconecho:
 25D6 E8C9FF                      call CONIN          ;get a console character
 25D9 50                          push ax
 25DA 8AC8                        mov cl,al           ;save and
 25DC E8CDFF                      call CONOUT
 25DF 58                          pop ax              ;echo to console
 25E0 3C61                        cmp al,'a'
 25E2 7206                        jb uret             ;less than 'a' is ok
 25E4 3C7A                        cmp al,'z'
 25E6 7702                        ja uret             ;greater than 'z' is ok
 25E8 2C20                        sub al,'a'-'A'      ;else shift to caps
                          uret:
 25EA C3                          ret

                          ;       utility subroutine to print messages

                          pmsg:
 25EB 8A07                        mov al,[BX]         ;get next char from message
 25ED 84C0                        test al,al
 25EF 7428                        jz return           ;if zero return
 25F1 8AC8                        mov CL,AL
 25F3 E8B6FF                      call CONOUT         ;print it
 25F6 43                          inc BX
 25F7 EBF2                        jmps pmsg           ;next character and loop

                          ;*********************************************
                          ;*                                           *
                          ;*          Disk Input/Output Routines       *
                          ;*                                           *
                          ;*********************************************

                          SELDSK:           ;select disk given by register CL
 25F9 BB0000                      mov bx,0000h
 25FC 80F902                      cmp cl,2            ;this BIOS only supports 2
 25FF 7318                        jnb return          ;return w/ 0000 in BX if ba
 2601 B080                        mov al, 80h
 2603 80F900                      cmp cl,0
 2606 7502                        jne sell            ;drive 1 if not zero
 2608 B040                        mov al, 40h         ;else drive is 0
 260A A26928               sell:  mov sel_mask,al     ;save drive select mask
                                                      ;now, we need disk paramete
 260D B500                        mov ch,0
 260F 8BD9                        mov bx,cx           ;BX = word(CL)
 2611 B104                        mov cl,4
```

```
2613 D3E3                       shl bx,cl        ;multiply drive code * 16
                                ;create offset from Disk Parameter Base
2615 81C37C28                   add bx,offset dp_base
                        return:
2619 C3                         ret

                        HOME:   ;move selected disk to home position (Track
261A C6066C2800                 mov trk,0        ;set disk i/o to track zero
261F BB6E28                     mov bx,offset hom_com
2622 E83500                     call execute
2625 74F2                       jz return        ;home drive and return if O
2627 BB6A27                     mov bx,offset bad_hom    ;else print
262A E8BEFF                     call pmsg        ;"Home Error"
262D EBEB                       jmps home        ;and retry

                        SETTRK: ;set track address given by CX
262F 880E6C28                   mov trk,cl       ;we only use 8 bits of trac
2633 C3                         ret

                        SETSEC: ;set sector number given by cx
2634 880E6D28                   mov sect,cl      ;we only use 8 bits of sect
2638 C3                         ret

                        SECTRAN: ;translate sector CX using table at [DX]
2639 8BD9                       mov bx,cx
263B 03DA                       add bx,dx        ;add sector to tran table a
263D 8A1F                       mov bl,[bx]      ;get logical sector
263F C3                         ret

                        SETDMA: ;set DMA offset given by CX
2640 890E6528                   mov dma_adr,CX
2644 C3                         ret

                        SETDMAB: ;set DMA segment given by CX
2645 890E6728                   mov dma_seg,CX
2649 C3                         ret
                        ;
                        GETSEGT:  ;return address of physical memory table
264A BB7328                     mov bx,offset seg_table
264D C3                         ret

                        ;*********************************************
                        ;*                                           *
                        ;*  All disk I/O parameters are setup:  the  *
                        ;*  Read and Write entry points transfer one *
                        ;*  sector of 128 bytes to/from the current  *
                        ;*  DMA address using the current disk drive *
                        ;*                                           *
                        ;*********************************************

                        READ:
264E B012                       mov al,12h       ;basic read sector command
2650 EB02                       jmps r_w_common

                        WRITE:
```

```
2652 B00A               mov al,0ah      ;basic write sector command

                r_w_common:
2654 BB6A28             mov bx,offset io_com ;point to command stri
2657 884701             mov byte ptr 1[BX],al ;put command into str
                ;       fall into execute and return

                execute:  ;execute command string.
                        ;[BX] points to length,
                        ;       followed by Command byte,
                        ;       followed by length-1 parameter byte

265A 891E6328           mov last_com,BX ;save command address for r
                outer_retry:
                        ;allow some retrying
265E C60662280A         mov rtry_cnt,max_retries
                retry:
2663 8B1E6328           mov BX,last_com
2667 E88900             call send_com   ;transmit command to i8271
                ;       check status poll

266A 8B1E6328           mov BX,last_com
266E 8A4701             mov al,1[bx]    ;get command op code
2671 B90008             mov cx,0800h    ;mask if it will be "int re
2674 3C2C               cmp al,2ch
2676 720B               jb exec_poll    ;ok if it is an interrupt t
2678 B98080             mov cx,8080h    ;else we use "not command b
267B 240F               and al,0fh
267D 3C0C               cmp al,0ch      ;unless there isn't
267F B000               mov al,0
2681 7736               ja exec_exit    ;       any result
                                        ;poll for bits in CH,
                exec_poll:              ;  toggled with bits in CL

2683 E4A0               in al,fdc_stat  ;read status
2685 22C5               and al,ch
2687 32C1               xor al,cl       ;  isolate what we want to
2689 74F8               jz exec_poll    ;and loop until it is done

                                        ;Operation complete,
268B E4A1               in al,fdc_rslt  ; see if result code indica
268D 241E               and al,1eh
268F 7428               jz exec_exit    ;no error, then exit
                                        ;some type of error occurre
2691 3C10               cmp al,10h
2693 7425               je dr_nrdy      ;was it a not ready drive ?
                                        ;no,
                dr_rdy: ; then we just retry read or write
2695 FE0E6228           dec rtry_cnt
2699 75C8               jnz retry       ;  up to 10 times

                ;       retries do not recover from the
                ;       hard error

269B B400               mov ah,0
```

```
269D 8BD8               mov bx,ax        ;make error code 16 bits
269F 8B9F9127           mov bx,errtbl[BX]
26A3 E845FF             call pmsg        ;print appropriate message
26A6 E4D8               in al,cdata      ;flush usart receiver buff
26A8 E82BFF             call uconecho    ;read upper case console ch
26AB 3C43               cmp al,'C'
26AD 7425               je wboot_l       ;cancel
26AF 3C52               cmp al,'R'
26B1 74AB               je outer_retry   ;retry 10 more times
26B3 3C49               cmp al,'I'
26B5 741A               je z_ret         ;ignore error
26B7 0CFF               or al,255        ;set code for permanent err
                  exec_exit:
26B9 C3                 ret

                  dr_nrdy:          ;here to wait for drive ready
26BA E81A00             call test_ready
26BD 75A4               jnz retry        ;if it's ready now we are d
26BF E81500             call test_ready
26C2 759F               jnz retry        ;if not ready twice in row,
26C4 BB0228             mov bx,offset nrdymsg
26C7 E821FF             call pmsg ;"Drive Not Ready"
                  nrdy01:
26CA E80A00             call test_ready
26CD 74FB               jz nrdy01        ;now loop until drive ready
26CF EB92               jmps retry       ;then go retry without decr
                  zret:
26D1 2400               and al,0
26D3 C3                 ret              ;return with no error code

                  wboot_l:               ;can't make it w/ a short l
26D4 E9B3FE             jmp WBOOT

                  ;*********************************************
                  ;*                                           *
                  ;*  The i8271 requires a read status command *
                  ;*  to reset a drive-not-ready after the     *
                  ;*  drive becomes ready                      *
                  ;*                                           *
                  ;*********************************************

                  test_ready:
26D7 B640               mov dh, 40h      ;proper mask if dr 1
26D9 F606692880         test sel_mask,80h
26DE 7502               jnz nrdy2
26E0 B604               mov dh, 04h      ;mask for dr 0 status bit
                  nrdy2:
26E2 BB7128             mov bx,offset rds_com
26E5 E80B00             call send_com
                  dr_poll:
26E8 E4A0               in al,fdc_stat   ;get status word
26EA A880               test al,80h
26EC 75FA               jnz dr_poll      ;wait for not command busy
26EE E4A1               in al,fdc_rslt   ;get "special result"
26F0 84C6               test al,dh       ;look at bit for this driv
```

```
26F2 C3                     ret             ;return status of ready

                    ;*******************************************
                    ;*                                         *
                    ;*  Send_com sends a command and parameters *
                    ;*  to the i8271: BX addresses parameters.  *
                    ;*  The DMA controller is also initialized  *
                    ;*  if this is a read or write              *
                    ;*                                         *
                    ;*******************************************

                    send_com:
26F3 E4A0                   in al,fdc_stat
26F5 A880                   test al,80h     ;insure command not busy
26F7 75FA                   jnz send_com    ;loop until ready

                            ;see if we have to initialize for a DMA ope

26F9 8A4701                 mov al,l[bx]    ;get command byte
26FC 3C12                   cmp al,12h
26FE 7504                   jne write_maybe ;if not a read it could be
2700 B140                   mov cl,40h
2702 EB06                   jmps init_dma   ;is a read command, go set
                    write_maybe:
2704 3C0A                   cmp al,0ah
2706 7520                   jne dma_exit    ;leave DMA alone if not rea
2708 B180                   mov cl,80h      ;we have write, not read
                    init_dma:
                    ;we have a read or write operation, setup DMA contr
                    ;       (CL contains proper direction bit)
270A B004                   mov al,04h
270C E6A8                   out dmac_mode,al    ;enable dmac
270E B000                   mov al,00
2710 E6A5                   out dmac_cont,al    ;send first byte to con
2712 8AC1                   mov al,cl
2714 E6A5                   out dmac_cont,al   ;load direction register
2716 A16528                 mov ax,dma_adr
2719 E6A4                   out dmac_adr,al    ;send low byte of DMA
271B 8AC4                   mov al,ah
271D E6A4                   out dmac_adr,al    ;send high byte
271F A16728                 mov ax,dma_seg
2722 E6AA                   out fdc_segment,al ;send low byte of segmen
2724 8AC4                   mov al,ah
2726 E6AA                   out fdc_segment,al ;then high segment addre
                    dma_exit:
2728 8A0F                   mov cl,[BX]     ;get count
272A 43                     inc BX
272B 8A07                   mov al,[BX]     ;get command
272D 0A066928               or al,sel_mask  ;merge command and drive co
2731 E6A0                   out fdc_com,al  ;send command byte
                    parm_loop:
2733 FEC9                   dec cl
2735 7482                   jz exec_exit    ;no (more) parameters, retu
2737 43                     inc BX          ;point to (next) parameter
                    parm_poll:
```

```
2738 E4A0                   in al,fdc_stat
273A A820                   test al,20h      ;test "parameter register f
273C 75FA                   jnz parm_poll    ;idle until parm reg not fu
273E 8A07                   mov al,[BX]
2740 E6A1                   out fdc_parm,al ;send next parameter
2742 EBEF                   jmps parm_loop   ;go see if there are more p

                     ;*********************************************
                     ;*                                           *
                     ;*                Data Areas                 *
                     ;*                                           *
                     ;*********************************************
 2744                data_offset     equ offset $

                            dseg
                            org     data_offset     ;contiguous with co

                            IF      loader_bios
                     ;--------------------------------------------
                     ;|                                          |
                     signon  db      cr,lf,cr,lf
                             db      'CP/M-86 Version 2.2',cr,lf,0
                     ;|                                          |
                     ;--------------------------------------------
                            ENDIF   ;loader_bios

                            IF      not loader_bios
                     ;--------------------------------------------
                     ;|                                          |
2744 0D0A0D0A        signon  db      cr,lf,cr,lf
2748 202053797374            db      '  System Generated  - 11 Jan 81',c
     656D2047656E
     657261746564
     20202D203131
     204A616E2038
     310D0A00
                     ;|                                          |
                     ;--------------------------------------------
                            ENDIF   ;not loader_bios

276A 0D0A486F6D65 bad_hom db     cr,lf,'Home Error',cr,lf,0
     204572726F72
     0D0A00
2779 0D0A496E7465 int_trp db     cr,lf,'Interrupt Trap Halt',cr,lf,0
     727275707420
     547261702048
     616C740D0A00

2791 B127B127B127 errtbl  dw er0,er1,er2,er3
     B127
2799 C127D127DE27         dw er4,er5,er6,er7
     EF27
27A1 022816282828         dw er8,er9,erA,erB
     3D28
27A9 4D28B127B127         dw erC,erD,erE,erF
```

B127

```
27B1 0D0A4E756C6C er0     db  cr,lf,'Null Error ??',0
     204572726F72
     203F3F00
 27B1              er1     equ er0
 27B1              er2     equ er0
 27B1              er3     equ er0
27C1 0D0A436C6F63 er4     db  cr,lf,'Clock Error :',0
     6B204572726F
     72203A00
27D1 0D0A4C617465 er5     db  cr,lf,'Late DMA :',0
     20444D41203A
     00
27DE 0D0A49442043 er6     db  cr,lf,'ID CRC Error :',0
     524320457272
     6F72203A00
27EF 0D0A44617461 er7     db  cr,lf,'Data CRC Error :',0
     204352432045
     72726F72203A
     00
2802 0D0A44726976 er8     db  cr,lf,'Drive Not Ready :',0
     65204E6F7420
     526561647920
     3A00
2816 0D0A57726974 er9     db  cr,lf,'Write Protect :',0
     652050726F74
     656374203A00
2828 0D0A54726B20 erA     db  cr,lf,'Trk 00 Not Found :',0
     3030204E6F74
     20466F756E64
     203A00
283D 0D0A57726974 erB     db  cr,lf,'Write Fault :',0
     65204661756C
     74203A00
284D 0D0A53656374 erC     db  cr,lf,'Sector Not Found :',0
     6F72204E6F74
     20466F756E64
     203A00
 27B1              erD     equ er0
 27B1              erE     equ er0
 27B1              erF     equ er0
 2802              nrdymsg equ er8

2862 00            rtry_cnt db 0   ;disk error retry counter
2863 0000          last_com dw 0   ;address of last command string
2865 0000          dma_adr  dw 0   ;dma offset stored here
2867 0000          dma_seg  dw 0   ;dma segment stored here
2869 40            sel_mask db 40h ;select mask, 40h or 80h

                   ;       Various command strings for i8271

286A 03            io_com  db 3    ;length
286B 00            rd_wr   db 0    ;read/write function code
286C 00            trk     db 0    ;track #
```

```
 286D 00              sect     db 0     ;sector #

 286E 022900          hom_com db 2,29h,0      ;home drive command
 2871 012C            rds_com db 1,2ch        ;read status command

                      ;        System Memory Segment Table

 2873 02              segtable db 2   ;2 segments
 2874 DF02                     dw tpa_seg      ;1st seg starts after BIOS
 2876 2105                     dw tpa_len      ;and extends to 08000
 2878 0020                     dw 2000h        ;second is 20000 -
 287A 0020                     dw 2000h        ;3FFFF (128k)

=                              include singles.lib ;read in disk definitio
=                     ;                 DISKS 2
=  287C               dpbase   equu      $               ;Base of Disk Param
=287C AB280000        dpe0     dw       xlt0,0000h      ;Translate Table
=2880 00000000                 dw       0000h,0000h     ;Scratch Area
=2884 C5289C28                 dw       dirbuf,dpb0     ;Dir Buff, Parm Blo
=2888 64294529                 dw       csv0,alv0       ;Check, Alloc Vecto
=288C AB280000        dpe1     dw       xlt1,0000h      ;Translate Table
=2890 00000000                 dw       0000h,0000h     ;Scratch Area
=2894 C5289C28                 dw       dirbuf,dpb1     ;Dir Buff, Parm Blo
=2898 93297429                 dw       csv1,alv1       ;Check, Alloc Vecto
=                     ;                 DISKDEF 0,1,26,6,1024,243,64,64,2
=  289C               dpb0     equ      offset $        ;Disk Parameter Blo
=289C 1A00                     dw       26              ;Sectors Per Track
=289E 03                       db       3               ;Block Shift
=289F 07                       db       7               ;Block Mask
=28A0 00                       db       0               ;Extnt Mask
=28A1 F200                     dw       242             ;Disk Size - 1
=28A3 3F00                     dw       63              ;Directory Max
=28A5 C0                       db       192             ;Alloc0
=28A6 00                       db       0               ;Alloc1
=28A7 1000                     dw       16              ;Check Size
=28A9 0200                     dw       2               ;Offset
=  28AB               xlt0     equ      offset $        ;Translate Table
=28AB 01070D13                 db       1,7,13,19
=28AF 19050B11                 db       25,5,11,17
=28B3 1703090F                 db       23,3,9,15
=28B7 1502080E                 db       21,2,8,14
=28BB 141A060C                 db       20,26,6,12
=28BF 1218040A                 db       18,24,4,10
=28C3 1016                     db       16,22
=  001F               als0     equ      31              ;Allocation Vector
=  0010               css0     equ      16              ;Check Vector Size
=                     ;                 DISKDEF 1,0
=  289C               dpb1     equ      dpb0            ;Equivalent Paramet
=  001F               als1     equ      als0            ;Same Allocation Ve
=  0010               css1     equ      css0            ;Same Checksum Vect
=  28AB               xlt1     equ      xlt0            ;Same Translate Tab
=                     ;                 ENDEF
=                     ;
=                     ;        Uninitialized Scratch Memory Follows:
=  28C5               begdat   equ      offset $        ;Start of Scratch A
```

```
=28C5           dirbuf  rs      128             ;Directory Buffer
=2945           alv0    rs      als0            ;Alloc Vector
 2964           csv0    rs      css0            ;Check Vector
=2974           alv1    rs      als1            ;Alloc Vector
=2993           csv1    rs      css1            ;Check Vector
=  29A3         enddat  equ     offset $        ;End of Scratch Are
=  00DE         datsiz  equ     offset $-begdat ;Size of Scratch Ar
=29A3 00                db      0               ;Marks End of Modul

 29A4           loc_stk rw  32  ;local stack for initialization
  29E4          stkbase equ offset $

  29E4          lastoff equ offset $
  02DF          tpa_seg equ (lastoff+0400h+15) / 16
  0521          tpa_len equu 0800h - tpa_seg
 29E4 00                db 0    ;fill last address for GENCMD

                ;*********************************************
                ;*                                           *
                ;*          Dummy Data Section               *
                ;*                                           *
                ;*********************************************
  0000                  dseg    0       ;absolute low memory
                        org     0       ;(interrupt vectors)
 0000           int0_offset     rw      1
 0002           int0_segment    rw      1
                ;       pad to system call vector
 0004                   rw      2*(bdos_int-1)

 0380           bdos_offset     rw      1
 0382           bdos_segment    rw      1
                        END
```

Appendix F
CBIOS Listing

```
**************************************************
*                                                *
* This is the listing of the skeletal CBIOS which *
* you can use as the basis for a customized BIOS  *
* for non-standard hardware.  The essential por-  *
* tions of the BIOS remain, with "rs" statements  *
* marking the routines to be inserted.            *
*                                                *
**************************************************
```

```
                ;*********************************************
                ;*                                           *
                ;* This Customized BIOS adapts CP/M-86 to    *
                ;* the following hardware configuration      *
                ;*     Processor:                            *
                ;*     Brand:                                *
                ;*     Controller:                           *
                ;*                                           *
                ;*                                           *
                ;*     Programmer:                           *
                ;*     Revisions :                           *
                ;*                                           *
                ;*********************************************

FFFF            true            equ -1
0000            false           equ not true
000D            cr              equ 0dh ;carriage return
000A            lf              equ 0ah ;line feed

                ;*********************************************
                ;*                                           *
                ;* Loader_bios is true if assembling the     *
                ;* LOADER BIOS, otherwise BIOS is for the    *
                ;* CPM.SYS file.                             *
                ;*                                           *
                ;*********************************************

0000            loader_bios     equ false
00E0            bdos_int        equ 224 ;reserved BDOS interrupt

                        IF      not loader_bios
                ;---------------------------------------------
                ;|                                           |
2500            bios_code       equ 2500h
0000            ccp_offset      equ 0000h
0B06            bdos_ofst       equ 0B06h ;BDOS entry point
                ;|                                           |
                ;---------------------------------------------
```

```
                         ENDIF    ;not loader_bios

                         IF       loader_bios
                 ;-----------------------------------------------
                 ;|                                             |
                 bios_code       equ 1200h ;start of LDBIOS
                 ccp_offset      equ 0003h ;base of CPMLOADER
                 bdos_ofst       equ 0406h ;stripped BDOS entry
                 ;|                                             |
                 ;-----------------------------------------------
                         ENDIF    ;loader_bios

                         cseg
                         org      ccpoffset
                 ccp:
                         org      bios_code

                 ;*********************************************
                 ;*                                           *
                 ;* BIOS Jump Vector for Individual Routines  *
                 ;*                                           *
                 ;*********************************************

2500 E93C00       jmp INIT        ;Enter from BOOT ROM or LOADER
2503 E97900       jmp WBOOT       ;Arrive here from BDOS call 0
2506 E98500       jmp CONST       ;return console keyboard status
2509 E98D00       jmp CONIN       ;return console keyboard char
250C E99A00       jmp CONOUT      ;write char to console device
250F E9A200       jmp LISTOUT     ;write character to list device
2512 E9B500       jmp PUNCH       ;write character to punch device
2515 E9BD00       jmp READER      ;return char from reader device
2518 E9F600       jmp HOME        ;move to trk 00 on cur sel drive
251B E9D900       jmp SELDSK      ;select disk for next rd/write
251E E90101       jmp SETTRK      ;set track for next rd/write
2521 E90301       jmp SETSEC      ;set sector for next rd/write
2524 E90C01       jmp SETDMA      ;set offset for user buff (DMA)
2527 E91701       jmp READ        ;read a 128 byte sector
252A E94701       jmp WRITE       ;write a 128 byte sector
252D E98F00       jmp LISTST      ;return list status
2530 E9F900       jmp SECTRAN     ;xlate logical->physical sector
2533 E90201       jmp SETDMAB     ;set seg base for buff (DMA)
2536 E90401       jmp GETSEGT     ;return offset of Mem Desc Table
2539 E9A400       jmp GETIOBF     ;return I/O map byte (IOBYTE)
253C E9A500       jmp SETIOBF     ;set I/O map byte (IOBYTE)

                 ;*********************************************
                 ;*                                           *
                 ;* INIT Entry Point, Differs for LDBIOS and  *
                 ;* BIOS, according to "Loader_Bios" value    *
                 ;*                                           *
                 ;*********************************************

                 INIT:   ;print signon message and initialize hardwa
253F 8CC8                mov ax,cs         ;we entered with a JMPF so
```

```
2541 8ED0               mov ss,ax       ;CS: as the initial value o
2543 8ED8               mov ds,ax       ;DS:,
2545 8EC0               mov es,ax       ;and ES:
                        ;use local stack during initialization
2547 BC5928             mov sp,offset stkbase
254A FC                 cld             ;set forward direction

                        IF      not loader_bios
                ;---------------------------------------------
                ;|                                           |
                        ; This is a BIOS for the CPM.SYS file.
                        ; Setup all interrupt vectors in low
                        ; memory to address trap

254B 1E                 push ds         ;save the DS register
254C C606A72600         mov IOBYTE,0    ;clear IOBYTE
2551 B80000             mov ax,0
2554 8ED8               mov ds,ax
2556 8EC0               mov es,ax       ;set ES and DS to zero
                        ;setup interrupt 0 to address trap routine
2558 C70600008225       mov int0_offset,offset int_trap
255E 8C0E0200           mov int0_segment,CS
2562 BF0400             mov di,4
2565 BE0000             mov si,0        ;then propagate
2568 B9FE01             mov cx,510      ;trap vector to
256B F3A5               rep movs ax,ax  ;all 256 interrupts
                        ;BDOS offset to proper interrupt
256D C7068003060B       mov bdos_offset,bdos_ofst
2573 1F                 pop ds          ;restore the DS register

                ;       (additional CP/M-86 initialization)
                ;|                                           |
                ;---------------------------------------------
                        ENDIF   ;not loader_bios

                        IF      loader_bios
                ;---------------------------------------------
                ;|                                           |
                        ;This is a BIOS for the LOADER
                        push ds         ;save data segment
                        mov ax,0
                        mov ds,ax       ;point to segment zero
                        ;BDOS interrupt offset
                        mov bdos_offset,bdos_ofst
                        mov bdos_segment,CS ;bdos interrupt segment
                ;       (additional LOADER initialization)
                        pop ds          ;restore data segment
                ;|                                           |
                ;---------------------------------------------
                        ENDIF   ;loader_bios

2574 BBB126             mov bx,offset signon
2577 E86F00             call pmsg       ;print signon message
257A B100               mov cl,0        ;default to dr A: on coldst
257C E981DA             jmp ccp         ;jump to cold start entry o
```

```
257F E984DA         WBOOT:  jmp ccp+6        ;direct entry to CCP at com

                            IF      not loader_bios
                    ;---------------------------------------------
                    ;|                                           |
                    int_trap:
2582 FA                     cli              ;block interrupts
2583 8CC8                   mov ax,cs
2585 8ED8                   mov ds,ax        ;get our data segment
2587 BBD126                 mov bx,offset int_trp
258A E85C00                 call pmsg
258D F4                     hlt              ;hardstop
                    ;|                                           |
                    ;---------------------------------------------
                            ENDIF   ;not loader_bios

                    ;*********************************************
                    ;*                                           *
                    ;*   CP/M Character I/O Interface Routines   *
                    ;*                                           *
                    ;*********************************************

                    CONST:          ;console status
258E                        rs      10       ;(fill-in)
2598 C3                     ret

                    CONIN:                   ;console input
2599 E8F2FF                 call CONST
259C 74FB                   jz CONIN         ;wait for RDA
259E                        rs      10       ;(fill-in)
25A8 C3                     ret

                    CONOUT:         ;console output
25A9                        rs      10       ;(fill-in)
25B3 C3                     ret              ;then return data

                    LISTOUT:                 ;list device output
25B4                        rs      10       ;(fill-in)
25BE C3                     ret

                    LISTST:                  ;poll list status
25BF                        rs      10       ;(fill-in)
25C9 C3                     ret

                    PUNCH:          ;write punch device
25CA                        rs      10       ;(fill-in)
25D4 C3                     ret

                    READER:
25D5                        rs      10       ;(fill-in)
25DF C3                     ret

                    GETIOBF:
25E0 A0A726                 mov al,IOBYTE
```

```
25E3 C3                        ret

                       SETIOBF:
25E4 880EA726                  mov IOBYTE,cl   ;set iobyte
25E8 C3                        ret             ;iobyte not implemented

                       pmsg:
25E9 8A07                      mov al,[BX]     ;get next char from message
25EB 84C0                      test al,al
25ED 7421                      jz return       ;if zero return
25EF 8AC8                      mov CL,AL
25F1 E8B5FF                    call CONOUT     ;print it
25F4 43                        inc BX
25F5 EBF2                      jmps pmsg       ;next character and loop

                       ;*********************************************
                       ;*                                           *
                       ;*         Disk Input/Output Routines        *
                       ;*                                           *
                       ;*********************************************

                       SELDSK:             ;select disk given by register CL
 0002                  ndisks  equu     2 ;number of disks (up to 16)
25F7 880EA826                  mov disk,cl     ;save disk number
25FB BB0000                    mov bx,0000h    ;ready for error return
25FE 80F902                    cmp cl,ndisks   ;n beyond max disks?
2601 730D                      jnb return      ;return if so
2603 B500                      mov ch,0        ;double(n)
2605 8BD9                      mov bx,cx       ;bx = n
2607 B104                      mov cl,4        ;ready for *16
2609 D3E3                      shl bx,cl       ;n = n * 16
260B B9F126                    mov cx,offset dpbase
260E 03D9                      add bx,cx       ;dpbase + n * 16
2610 C3               return:  ret             ;bx = .dph

                       HOME:   ;move selected disk to home position (Track
2611 C706A9260000              mov trk,0       ;set disk i/o to track zero
2617                           rs      10      ;(fill-in)
2621 C3                        ret

                       SETTRK: ;set track address given by CX
2622 890EA926                  mov trk,CX
2626 C3                        ret

                       SETSEC: ;set sector number given by cx
2627 890EAB26                  mov sect,CX
262B C3                        ret

                       SECTRAN: ;translate sector CX using table at [DX]
262C 8BD9                      mov bx,cx
262E 03DA                      add bx,dx       ;add sector to tran table a
2630 8A1F                      mov bl,[bx]     ;get logical sector
2632 C3                        ret

                       SETDMA: ;set DMA offset given by CX
```

```
2633 890EAD26              mov dma_adr,CX
2637 C3                    ret

                   SETDMAB: ;set DMA segment given by CX
2638 890EAF26              mov dma_seg,CX
263C C3                    ret
                   ;
                   GETSEGT:  ;return address of physical memory table
263D BBE826                mov bx,offset seg_table
2640 C3                    ret

                   ;*********************************************
                   ;*                                           *
                   ;*  All disk I/O parameters are setup:       *
                   ;*     DISK     is disk number      (SELDSK) *
                   ;*     TRK      is track number     (SETTRK) *
                   ;*     SECT     is sector number    (SETSEC) *
                   ;*     DMA_ADR  is the DMA offset   (SETDMA) *
                   ;*     DMA_SEG  is the DMA segment  (SETDMAB)*
                   ;*  READ reads the selected sector to the DMA*
                   ;*  address, and WRITE writes the data from  *
                   ;*  the DMA address to the selected sector   *
                   ;*  (return 00 if successful,  01 if perm err)*
                   ;*                                           *
                   ;*********************************************

                   READ:
2641                       rs      50      ;fill-in
2673 C3                    ret

                   WRITE:
2674                       rs      50      ;(fill-in)
26A6 C3                    ret


                   ;*********************************************
                   ;*                                           *
                   ;*                Data Areas                 *
                   ;*                                           *
                   ;*********************************************
  26A7             data_offset     equ offset $

                           dseg
                           org     data_offset     ;contiguous with cc
26A7 00            IOBYTE  db      0
26A8 00            disk    db      0       ;disk number
26A9 0000          trk     dw      0       ;track number
26AB 0000          sect    dw      0       ;sector number
26AD 0000          dma_adr dw      0       ;DMA offset from DS
26AF 0000          dma_seg dw      0       ;DMA Base Segment

                           IF      loader_bios
                   ;---------------------------------------------
                   ;|                                           |
                   signon  db      cr,lf,cr,lf
```

```
                                 db      'CP/M-86 Version 1.0',cr,lf,0
                         ;|                                              |
                         ;-----------------------------------------------
                                 ENDIF   ;loader_bios

                                 IF      not loader_bios
                         ;-----------------------------------------------
                         ;|                                              |
 26B1 0D0A0D0A           signon  db      cr,lf,cr,lf
 26B5 53797374656D               db      'System Generated 00/00/00'
      2047656E6572
      617465642030
      302F30302F30
      30
 26CE 0D0A00                     db      cr,lf,0
                         ;|                                              |
                         ;-----------------------------------------------
                                 ENDIF   ;not loader_bios

 26D1 0D0A               int_trp db      cr,lf
 26D3 496E74657272               db      'Interrupt Trap Halt'
      757074205472
      61702048616C
      74
 26E6 0D0A                       db      cr,lf

                         ;       System Memory Segment Table

 26E8 02                 segtable db 2   ;2 segments
 26E9 C602                       dw tpa_seg      ;1st seg starts after BIOS
 26EB 3A05                       dw tpa_len      ;and extends to 08000
 26ED 0020                       dw 2000h        ;second is 20000 -
 26EF 0020                       dw 2000h        ;3FFFF (128k)

=                                include singles.lib ;read in disk definitio
=                        ;               DISKS 2
=   26F1                 dpbase  equ     $               ;Base of Disk Param
=26F1 20270000           dpe0    dw      xlt0,0000h      ;Translate Table
=26F5 00000000                   dw      0000h,0000h     ;Scratch Area
=26F9 3A271127                   dw      dirbuf,dpb0     ;Dir Buff, Parm Blo
=26FD D927BA27                   dw      csv0,alv0       ;Check, Alloc Vecto
=2701 20270000           dpe1    dw      xlt1,0000h      ;Translate Table
=2705 00000000                   dw      0000h,0000h     ;Scratch Area
=2709 3A271127                   dw      dirbuf,dpb1     ;Dir Buff, Parm Blo
=270D 0828E927                   dw      csv1,alv1       ;Check, Alloc Vecto
=                        ;               DISKDEF 0,1,26,6,1024,243,64,64,2
=   2711                 dpb0    equ     offset $        ;Disk Parameter Blo
=2711 1A00                       dw      26              ;Sectors Per Track
=2713 03                         db      3               ;Block Shift
=2714 07                         db      7               ;Block Mask
=2715 00                         db      0               ;Extnt Mask
=2716 F200                       dw      242             ;Disk Size - 1
=2718 3F00                       dw      63              ;Directory Max
=271A C0                         db      192             ;Alloc0
=271B 00                         db      0               ;Alloc1
```

```
=271C 1000              dw      16              ;Check Size
=271E 0200              dw      2               ;Offset
=  2720         xlt0    equ     offset $        ;Translate Table
=2720 01070D13          db      1,7,13,19
=2724 19050B11          db      25,5,11,17
=2728 1703090F          db      23,3,9,15
=272C 1502080E          db      21,2,8,14
=2730 141A060C          db      20,26,6,12
=2734 1218040A          db      18,24,4,10
=2738 1016              db      16,22
=  001F         als0    equ     31              ;Allocation Vector
=  0010         css0    equ     16              ;Check Vector Size
=               ;               DISKDEF 1,0
=  2711         dpb1    equ     dpb0            ;Equivalent Paramet
=  001F         als1    equ     als0            ;Same Allocation Ve
=  0010         css1    equ     css0            ;Same Checksum Vect
=  2720         xlt1    equ     xlt0            ;Same Translate Tab
=               ;               ENDEF
=               ;
=               ;       Uninitialized Scratch Memory Follows:
=               ;
=  273A         begdat  equ     offset $        ;Start of Scratch A
=273A           dirbuf  rs      128             ;Directory Buffer
=27BA           alv0    rs      als0            ;Alloc Vector
=27D9           csv0    rs      css0            ;Check Vector
=27E9           alv1    rs      als1            ;Alloc Vector
=2808           csv1    rs      css1            ;Check Vector
=  2818         enddat  equ     offset $        ;End of Scratch Are
=  00DE         datsiz  equ     offset $-begdat ;Size of Scratch Ar
=2818 00                db      0               ;Marks End of Modul


 2819           loc_stk rw  32  ;local stack for initialization
  2859          stkbase equ offset $

  2859          lastoff equ offset $
  02C6          tpa_seg equ (lastoff+0400h+15) / 16
  053A          tpa_len equ 0800h - tpa_seg
 2859 00                db 0    ;fill last address for GENCMD

                ;*********************************************
                ;*                                           *
                ;*          Dummy Data Section               *
                ;*                                           *
                ;*********************************************
  0000                  dseg    0       ;absolute low memory
                        org     0       ;(interrupt vectors)
 0000           int0_offset     rw      1
 0002           int0_segment    rw      1
                ;       pad to system call vector
 0004                   rw      2*(bdos_int-1)

 0380           bdos_offset     rw      1
 0382           bdos_segment    rw      1
                        END
```

Chapter III

CP/M-86

Programer's Guide

Canon AS-100 Series

CP/M-86 T.M.

Operating System

Release 1.1

System Guide Release Notes

Digital Research
P.O. Box 579
160 Central Avenue
Pacific Grove, CA 93950
(408) 649-3896
TWX 910 360 5001

COPYRIGHT

DISCLAIMER

TRADEMARKS

Foreword

This manual assists the 8086 assembly language programmer working in a CP/M-86™ environment. It assumes you are familiar with the CP/M-86 implementation of CP/M and have read the following Digital Research publications:

- CP/M 2 Documentation
- CP/M-86 System Guide

The reader should also be familiar with the 8086 assembly language instruction set, which is defined in Intel's 8086 Family User's Manual.

The first section of this manual discusses ASM-86 operation and the various assembler options which may be enabled when invoking ASM-86™. One of these options controls the hexadecimal output format. ASM-86 can generate 8086 machine code in either Intel or Digital Research format. These two hexadecimal formats are described in Appendix A.

The second section discusses the elements of ASM-86 assembly language. It defines ASM-86's character set, constants, variables, identifiers, operators, expressions, and statements.

The third section discusses the ASM-86 directives, which perform housekeeping functions such as requesting conditional assembly, including multiple source files, and controlling the format of the listing printout.

The fourth section is a concise summary of the 8086 instruction mnemonics accepted by ASM-86. The mnemonics used by the Digital Research assembler are the same as those used by the Intel assembler except for four instructions: the intra-segment short jump, and inter-segment jump, return and call instructions. These differences are summarized in Appendix B.

The fifth section of this manual discusses the code-macro facilities of ASM-86. Code-macro definition, specifiers and modifiers as well as nine special code-macro directives are discussed. This information is also summarized in Appendix H.

The sixth section discusses the DDT-86 program, which allows the user to test and debug programs interactively in the CP/M-86 enviornment. Section 6 includes a DDT-86 sample debugging session.

Table of Contents

Table of Contents
(continued)

Table of Contents (continued)

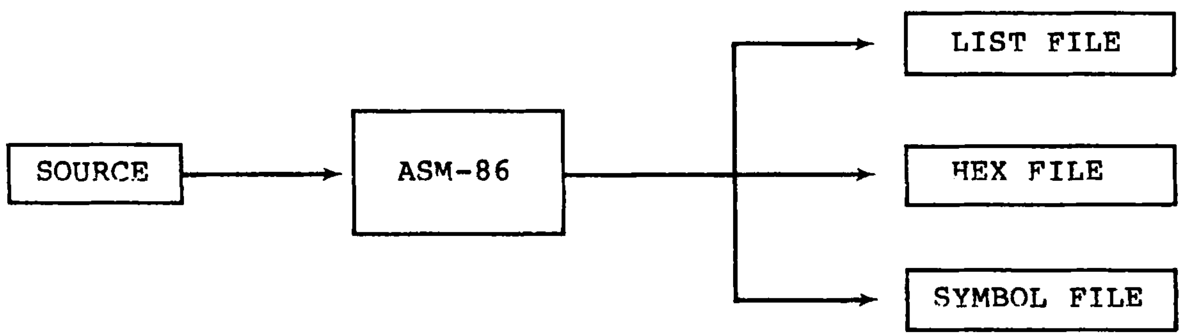
Table of Contents (continued)

Appendixes

Section 1
Introduction

1.1 Assembler Operation

ASM-86 processes an 8086 assembly language source file in three passes and produces three output files, including an 8086 machine language file in hexadecimal format. This object file may be in either Intel or Digital Research hex format, which are described in Appendix C. ASM-86 is shipped in two forms: an 8086 cross-assembler designed to run under CP/M on an Intel 8080 or Zilog Z-80 based system, and a 8086 assembler 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 Figure 1-1, below.



<file name>.A86	-	contains source
<file name>.LST	-	contains listing
<file name>.H86	-	contains assembled program in hexadecimal format
<file name>.SYM	-	contains all user-defined symbols

Figure 1-1. ASM-86 Source and Object Files

Figure 1-1 also lists ASM-86 filename extensions. ASM-86 accepts a source file with any three letter extension, but if the extension is omitted from the invoking command, it looks for the specified filename with the extension .A86 in the directory. If no filename is specified and the file has an extension other than .A86 or has no extension at all, ASM-86 returns an error message.

The other extensions listed in Figure 1-1 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.

Invoke ASM-86 by entering a command of the following form:

ASM86 <source filename> [$ <optional parameters>]

Section 1.2 explains the optional parameters. Specify the source file in the following form:

[<optional drive>:]<filename>[.<optional extension>]

where

<optional drive>	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 filename of 1 to 8 characters.
<optional extension>	is a valid file extension of 1 to 3 characters, usually .A86.

Some examples of valid ASM-86 commands are:

A>**ASM86 B:BIOS88**

A>**ASM86 BIOS88.ASM $FI AA HB PB SB**

A>**ASM86 D:TEST**

Once invoked, ASM-86 responds with the message:

CP/M 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 extension 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 current disk drive, but they may be redirected by optional parameters, or by a drive specification in the the source file name. In the latter case, ASM-86 directs the output files to the drive specified in the source file name.

During assembly, ASM-86 aborts if an error condition such as disk 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. Appendix H lists ASM-86 error messages. When the assembly is complete, ASM-86 displays the message:

```
END OF ASSEMBLY. NUMBER OF ERRORS: n
```

1.2 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 Table 1-1, below.

Table 1-1. Run-time Parameter Summary

Parameter	To Specify	Valid Arguments
A	source file device	A, B, C, ... P
H	hex output file device	A ... P, X, Y, Z
P	list file device	A ... P, X, Y, Z
S	symbol file device	A ... P, 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, ... P  or  X, Y, Z
```

Device names A through P respectively specify disk drives A through P. X specifies the user console (CON:), Y specifies the line printer (LST:), and Z suppresses output (NUL:).

If output is directed to the console, it may be temporarily stopped at any time by typing a control-S. Restart the output by typing a second control-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.

Table 1-2. Run-time Parameter Examples

Command Line	Result
ASM86 IO	Assemble file IO.A86, produce IO.HEX, IO.LST and IO.SYM, all on the default drive.
ASM86 IO.ASM $ AD SZ	Assemble file IO.ASM on device D, produce IO.LST and IO.HEX on the default device, suppress symbol file.
ASM86 IO $ PY SX	Assemble file IO.A86, produce IO.HEX, route listing directly to printer, output symbols on console.
ASM86 IO $ FD	Produce Digital Research hex format.
ASM86 IO $ FI	Produce Intel hex format.

1.3 Aborting ASM-86

You may abort ASM-86 execution at any time by hitting any key on the console keyboard. When a key is pressed, ASM-86 responds with the question:

```
USER BREAK. OK(Y/N)?
```

A Y response aborts the assembly and returns to the operating system. An N response continues the assembly.

Section 2
Elements of ASM-86 Assembly Language

2.1 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.

2.2 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.

2.3 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.

Table 2-1 describes ASM-86 separators and delimiters. Some delimiters are also operators and are explained in greater detail in Section 2.6.

Table 2-1. Separators and Delimiters

Character	Name	Use
20H	space	separator
09H	tab	separator, legal in source files, expanded in list files
CR	carriage return	terminate source lines
LF	line feed	legal after CR; if within source lines, it is interpreted as a space
;	semicolon	start 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 but ignored in identifiers
!	exclamation point	logically terminates a statement, thus allowing multiple statements on a single source line
'	apostrophe	delimits string constants

2.4 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.

2.4.1 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 Table 2-2, below.

Table 2-2. 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 either a leading 0 or 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
3377o    0FE3H    1234d    0ffffh
```

2.4.2 Character Strings

ASM-86 treats an ASCII character string delimited by apostrophes as a string constant. All instructions accept only one- or two-character string 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. ASM-86 does not translate case within character strings, so both upper- and lower-case letters can be used. 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. Table 2-3 shows valid strings and how they appear after processing:

Table 2-3. String Constant Examples

'a'	->	a
'Ab''Cd'	->	Ab'Cd
'I like CP/M'	->	I like CP/M
''''	->	'
'ONLY UPPER CASE'	->	ONLY UPPER CASE
'only lower case'	->	only lower case

2.5 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
```

2.5.1 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. For a complete list of keywords, see Appendix D.

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 Section 4. Directives are discussed in Section 3. Section 2.6 defines operators. Table 2-4 lists the ASM-86 keywords that identify 8086 registers.

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 Section 2.5.2 for a complete discussion of Type attributes.

Table 2-4. Register Keywords

Register Symbol	Size	Numeric Value	Meaning
AH	1 byte	100 B	Accumulator-High-Byte
BH	1 "	111 B	Base-Register-High-Byte
CH	1 "	101 B	Count-Register-High-Byte
DH	1 "	110 B	Data-Register-High-Byte
AL	1 "	000 B	Accumulator-Low-Byte
BL	1 "	011 B	Base-Register-Low-Byte
CL	1 "	001 B	Count-Register-Low-Byte
DL	1 "	010 B	Data-Register-Low-Byte
AX	2 bytes	000 B	Accumulator (full word)
BX	2 "	011 B	Base-Register "
CX	2 "	001 B	Count-Register "
DX	2 "	010 B	Data-Register "
BP	2 "	101 B	Base Pointer
SP	2 "	100 B	Stack Pointer
SI	2 "	110 B	Source Index
DI	2 "	111 B	Destination Index
CS	2 "	01 B	Code-Segment-Register
DS	2 "	11 B	Data-Segment-Register
SS	2 "	10 B	Stack-Segment-Register
ES	2 "	00 B	Extra-Segment-Register

2.5.2 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 Section 3.2). 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 (see Section 3). 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 label, as shown below:

```
VARIABLE  EQU  ANOTHER_VARIABLE
```

<u>Labels</u> 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
```

Section 2.6 describes operators and their effects on numbers and number symbols.

2.6 Operators

ASM-86 operators fall into the following categories: arithmetic, logical, and relational operators, segment override, variable manipulators and creators. Table 2-5 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.

Table 2-5. 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

Table 2-5. (continued)

Syntax	Result	Validity
Relational Operators		
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
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 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

Table 2-5. (continued)

<table>
<tr><th>Syntax</th><th>Result</th><th>Validity</th></tr>
<tr><td colspan="3">Segment Override</td></tr>
<tr><td><seg reg>: <addr exp></td><td>overrides assembler's choice of segment register.</td><td><seg reg> = CS, DS, SS or ES</td></tr>
<tr><td colspan="3">Variable Manipulators, Creators</td></tr>
<tr><td>SEG a</td><td>creates a number whose value is the segment value of the variable or label a.</td><td>a = label | variable</td></tr>
<tr><td>OFFSET a</td><td>creates a number whose value is the offset value of the variable or label a.</td><td>a = label | variable</td></tr>
<tr><td>TYPE a</td><td>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.</td><td>a = label | variable</td></tr>
<tr><td>LENGTH a</td><td>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.</td><td>a = label | variable</td></tr>
<tr><td>LAST a</td><td>if LENGTH a > 0, then LAST a = LENGTH a - 1; if LENGTH a = 0, then LAST a = 0.</td><td>a = label | variable</td></tr>
<tr><td>a PTR b</td><td>creates virtual variable or label with type of a and attributes of b</td><td>a = BYTE | WORD, | DWORD b = <addr exp></td></tr>
<tr><td>.a</td><td>creates variable with an offset attribute of a. Segment attribute is current segment.</td><td>a = number</td></tr>
<tr><td>$</td><td>creates label with offset equal to current value of location counter; segment attribute is current segment.</td><td>no argument</td></tr>
</table>

2.6.1 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
0014 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:

```
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
```

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 code segment. This operator takes no operand. For example:

```
0053 E9FDFF          JMP    $
0056 EBFE            JMPS   $
0058 E9FD2F          JMP    $+3000H
```

2.6.2 Operator Precedence

Expressions combine variables, labels or numbers with operators. ASM-86 allows several kinds of expressions which are discussed in Section 2.7. 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. Table 2-6 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
```

Table 2-6. 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>:
9	Variable manipulators, creators	SEG, OFFSET, PTR, TYPE, LENGTH, LAST
10	Parentheses/brackets	( ), [ ]
11	Period and Dollar	., $

2.7 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]
```

2.8 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 except in comments. 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 Section 4. 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 Section 4.

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 Section 3. 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 Section 3.

operand(s)

Analogous to the operands to 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.

Section 3
Assembler Directives

3.1 Introduction

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 Section 2.8.

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. Square brackets, [], enclose optional arguments. Angle brackets, <>, enclose descriptions of user-supplied arguments. Do not include these symbols when coding a directive.

3.2 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. Future versions of ASM-86 will support additional segments such as multiple data or code segments. All ASM-86 statements must be assigned to one of the four currently supported 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 unless a segment override is included with the instruction. 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.

3.2.1 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.

3.2.2 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.

3.2.3 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.

3.2.4 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.

3.3 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.

3.4 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, then all lines are listed but not assembled. All elements in the numeric expression must be defined before they appear in the IF directive. Nested IF directives are not legal.

3.5 The INCLUDE Directive

```
INCLUDE    <file name>
```

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 name> does not contain a file type, the file type is assumed to be .A86. If no drive name is specified with <file name>, ASM-86 assumes the drive containing the source file.

3.6 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).

3.7 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
```

3.8 The DB Directive

```
[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 Section 2.4.2. 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
```

3.9 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 except that the low-order byte is stored first, followed by the high-order byte. 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
```

3.10 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
```

3.11 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
```

3.12 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.

3.13 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          BUFF   RW    128
4161                 RW    4000H
C161                 RW    1
```

3.14 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 monitor'
```

3.15 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.

3.16 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.

3.17 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.

3.18 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.

3.19 The NOLIST and LIST Directives

NOLIST
LIST

The NOLIST directive blocks the printout of the following lines. Restart the listing with a LIST directive.

Section 4
The ASM-86 Instruction Set

4.1 Introduction

The ASM-86 instruction set includes all 8086 machine instructions. The general syntax for instruction statements is given in Section 2.7. 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. Table 4-1 shows the symbols used in the instruction-definition tables to define operand types.

Table 4-1. 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

Table 4-1. (continued)

Symbol	Operand Type
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 Table 5-2 to represent the nine Flag registers.

Table 4-2. 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

4.2 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 Table 4-3 that if acc = AL, a byte is transferred, but if acc = AX, a word is transferred.

Table 4-3. Data Transfer Instructions

Syntax		Result
IN	acc,numb8|numb16	transfer data from input port given by numb8 or numb16 (0-255) to accumulator
IN	acc,DX	transfer data from input port given by DX register (0-0FFFFH) to accumulator
LAHF		transfer SF, ZF, AF, PF, and CF flags to the AH register
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 the 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

Table 4-3. (continued)

Syntax		Result
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|numb16,acc	transfer data from accumulator to output port (0-255) given by numb8 or numb16
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
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.

4.3 Arithmetic, Logical, 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. Table 4-4 summarizes the effects of arithmetic instructions on flag bits. Table 4-5 defines arithmetic instructions and Table 4-6 logical and shift instructions.

Table 4-4. 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.

Table 4-5. Arithmetic Instructions

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
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

Table 4-5. (continued)

Syntax		Result
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
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

Table 4-6. 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 ones 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
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

Table 4-6. (continued)

Syntax		Result
SAL	mem|reg,CL	shift memory or register left, number of bits given by CL register, shift in low-order zero bits
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
SHR	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
TEST	mem|reg,numb	perform bitwise logical "and" - test of memory register and data constant - set condition flags but do not change destination

Table 4-6. (continued)

Syntax		Result
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

4.4 String Instructions

String instructions take 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).

Table 4-7. String Instructions

Syntax		Result
CMPS	mem|reg,mem|reg	subtract source from destination, affect flags, but do not return result.
LODS	mem|reg	transfer a byte or word from the source operand to the accumulator.
MOVS	mem|reg,mem|reg	move 1 byte (or word) from source to destination.
SCAS	mem|reg	subtract destination operand from accumulator (AX or AL), affect flags, but do not return result.
STOS	mem|reg	transfer a byte or word from accumulator to the destination operand.

Table 4-8 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 Section 2.8.

Table 4-8. 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"

4.5 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. Table 4-9 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.

Table 4-9. 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
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)

Table 4-9. (continued)

Syntax		Result
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
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"

Table 4-9. (continued)

Syntax		Result
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"
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"

Table 4-9. (continued)

Syntax	Result
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
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

4.6 Processor Control Instructions

Processor control instructions manipulate the flag registers. Moreover, some of these instructions can synchronize the 8086 CPU with external hardware.

Table 4-10. 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

Table 4-10. (continued)

Syntax	Results
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
HLT	cause 8086 processor to enter halt state until an interrupt is recognized
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

Section 5
Code-Macro Facilities

5.1 Introduction to Code-macros

ASM-86 does not support traditional assembly-language macros, but it does allow the user to define his own instructions by using the Code-macro directive. Like traditional macros, code-macros are assembled wherever they appear in assembly language code, but there the similarity ends. Traditional macros contain assembly language instructions, but a code-macro contains only code-macro directives. Macros are usually defined in the user's symbol table; ASM-86 code-macros are defined in the assembler's symbol table. A macro simplifies using the same block of instructions over and over again throughout a program, but a code-macro sends a bit stream to the output file and in effect adds a new instruction to the assembler.

Because ASM-86 treats a code-macro as an instruction, you can invoke code-macros by using them as instructions in your program. The example below shows how MAC, an instruction defined by a code-macro, can be invoked.

```
        .
        .
        .
XCHG    BX,WORD3
MAC     PAR1,PAR2
MUL     AX,WORD4
        .
        .
        .
```

Note that MAC accepts two operands. When MAC was defined, these two operands were also classified as to type, size, and so on by defining MAC's formal parameters. The names of formal parameters are not fixed. They are stand-ins which are replaced by the names or values supplied as operands when the code-macro is invoked. Thus formal parameters "hold the place" and indicate where and how the operands are to be used.

The definition of a code-macro starts with a line specifying its name and its formal parameters, if any:

```
CodeMacro <name> [<formal parameter list>]
```

where the optional <formal parameter list> is defined:

```
<formal name>:<specifier letter>[<modifier letter>][<range>]
```

As stated above, the formal name is not fixed, but a place holder. If formal parameter list is present, the specifier letter is required and the modifier letter is optional. Possible specifiers are A, C, D, E, M, R, S, and X. Possible modifier letters are b, d, w, and sb. The assembler ignores case except within strings, but for clarity, this section shows specifiers in upper-case and modifiers in lower-case. Following sections describe specifiers, modifiers, and the optional range in detail.

The body of the code-macro describes the bit pattern and formal parameters. Only the following directives are legal within code-macros:

```
SEGFIX
NOSEGFIX
MODRM
RELB
RELW
DB
DW
DD
DBIT
```

These directives are unique to code-macros, and those which appear to duplicate ASM-86 directives (DB, DW, and DD) have different meanings in code-macro context. These directives are discussed in detail in later sections. The definition of a code-macro ends with a line:

```
EndM
```

CodeMacro, EndM, and the code-macro directives are all reserved words. Code-macro definition syntax is defined in Backus-Naur-like form in Appendix H. The following examples are typical code-macro definitions.

```
CodeMacro AAA
  DB 37H
EndM

CodeMacro DIV divisor:Eb
  SEGFIX divisor
  DB     6FH
  MODRM  divisor
EndM

CodeMacro ESC opcode:Db(0,63),src:Eb
  SEGFIX src
  DBIT 5(1BH),3(opcode(3))
  MODRM  opcode,src
EndM
```

5.2 Specifiers

Every formal parameter must have a specifier letter that indicates what type of operand is needed to match the formal parameter. Table 5-1 defines the eight possible specifier letters.

Table 5-1. Code-macro Operand Specifiers

Letter	Operand Type
A	Accumulator register, AX or AL.
C	Code, a label expression only.
D	Data, a number to be used as an immediate value.
E	Effective address, either an M (memory address) or an R (register).
M	Memory address. This can be either a variable or a bracketed register expression.
R	A general register only.
S	Segment register only.
X	A direct memory reference.

5.3 Modifiers

The optional modifier letter is a further requirement on the operand. The meaning of the modifier letter depends on the type of the operand. For variables, the modifier requires the operand to be of type: "b" for byte, "w" for word, "d" for double-word and "sb" for signed byte. For numbers, the modifiers require the number to be of a certain size: "b" for -256 to 255 and "w" for other numbers. Table 5-2 summarizes code-macro modifiers.

Table 5-2. Code-macro Operand Modifiers

Variables		Numbers	
Modifier	Type	Modifier	Size
b	byte	b	-256 to 255
w	word	w	anything else
d	dword		
sb	signed byte		

5.4 Range Specifiers

The optional range is specified within parentheses by either one expression or two expressions separated by a comma. The following are valid formats:

```
(numberb)
(register)
(numberb,numberb)
(numberb,register)
(register,numberb)
(register,register)
```

Numberb is 8-bit number, not an address. The following example specifies that the input port must be identified by the DX register:

```
CodeMacro IN dst:Aw,port:Rw(DX)
```

The next example specifies that the CL register is to contain the "count" of rotation:

```
CodeMacro ROR dst:Ew,count:Rb(CL)
```

The last example specifies that the "opcode" is to be immediate data, and may range from 0 to 63 inclusive:

```
CodeMacro ESC opcode:Db(0,63),adds:Eb
```

5.5 Code-macro Directives

Code-macro directives define the bit pattern and make further requirements on how the operand is to be treated. Directives are reserved words, and those that appear to duplicate assembly language instructions have different meanings within a code-macro definition. Only the nine directives defined here are legal within code-macro definitions.

5.5.1 SEGFIX

If SEGFIX is present, it instructs the assembler to determine whether a segment-override prefix byte is needed to access a given memory location. If so, it is output as the first byte of the instruction. If not, no action is taken. SEGFIX takes the form:

```
SEGFIX <formal name>
```

where <formal name> is the name of a formal parameter which represents the memory address. Because it represents a memory address, the formal parameter must have one of the specifiers E, M or X.

5.5.2 NOSEGFIX

Use NOSEGFIX for operands in instructions that must use the ES register for that operand. This applies only to the destination operand of these instructions: CMPS, MOVS, SCAS, STOS. The form of NOSEGFIX is:

```
NOSEGFIX   segreg,<formname>
```

where segreg is one of the segment registers ES, CS, SS, or DS and <formname> is the name of the memory-address formal parameter, which must have a specifier E, M, or X. No code is generated from this directive, but an error check is performed. The following is an example of NOSEGFIX use:

```
CodeMacro MOVS si_ptr:Ew,di_ptr:Ew
  NOSEGFIX    ES,di_ptr
  SEGFIX      si_ptr
  DB          0A5H
EndM
```

5.5.3 MODRM

This directive intructs the assembler to generate the ModRM byte, which follows the opcode byte in many of the 8086's instructions. The ModRM byte contains either the indexing type or the register number to be used in the instruction. It also specifies which register is to be used, or gives more information to specify an instruction.

The ModRM byte carries the information in three fields: The mod field occupies the two most significant bits of the byte, and combines with the register memory field to form 32 possible values: 8 registers and 24 indexing modes.

The reg field occupies the three next bits following the mod field. It specifies either a register number or three more bits of opcode information. The meaning of the reg field is determined by the opcode byte.

The register memory field occupies the last three bits of the byte. It specifies a register as the location of an operand, or forms a part of the address-mode in combination with the mod field described above.

For further information of the 8086's instructions and their bit patterns, see Intel's 8086 Assembly Language Programing Manual and the Intel 8086 Family User's Manual. The forms of MODRM are:

```
MODRM  <form name>,<form name>
MODRM  NUMBER7,<form name>
```

where NUMBER7 is a value 0 to 7 inclusive and <form name> is the name of a formal parameter. The following examples show MODRM use:

```
CodeMacro RCR dst:Ew,count:Rb(CL)
  SEGFIX       dst
  DB           OD3H
  MODRM        3,dst
EndM

CodeMacro OR dst:Rw,src:Ew
  SEGFIX       src
  DB           OBH
  MODRM        dst,src
EndM
```

5.5.4 RELB and RELW

These directives, used in IP-relative branch instructions, instruct the assembler to generate displacement between the end of the instruction and the label which is supplied as an operand. RELB generates one byte and RELW two bytes of displacement. The directives the following forms:

```
RELB <form name>
RELW <form name>
```

where <form name> is the name of a formal parameter with a "C" (code) specifier. For example:

```
CodeMacro LOOP place:Cb
  DB          0E2H
  RELB        place
EndM
```

5.5.5 DB, DW and DD

These directives differ from those which occur outside of code-macros. The form of the directives are:

```
DB <form name> | NUMBERB
DW <form name> | NUMBERW
DD <form name>
```

where NUMBERB is a single-byte number, NUMBERW is a two-byte number, and <form name> is a name of a formal parameter. For example:

```
CodeMacro XOR dst:Ew,src:Db
  SEGFIX      dst
  DB          81H
  MODRM       6,dst
  DW          src
EndM
```

5.5.6 DBIT

This directive manipulates bits in combinations of a byte or less. The form is:

```
DBIT <field description>[,<field description>]
```

where a <field description>, has two forms:

```
<number><combination>
<number>(<form name>(<rshift>))
```

where <number> ranges from 1 to 16, and specifies the number of bits to be set. <combination> specifies the desired bit combination. The total of all the <number>s listed in the field descriptions must not exceed 16. The second form shown above contains <form name>, a formal parameter name that instructs the assembler to put a certain number in the specified position. This number normally refers to the register specified in the first line of the code-macro. The numbers used in this special case for each register are:

AL:	0
CL:	1
DL:	2
BL:	3
AH:	4
CH:	5
DH:	6
BH:	7
AX:	0
CX:	1
DX:	2
BX:	3
SP:	4
BP:	5
SI:	6
DI:	7
ES:	0
CS:	1
SS:	2
DS:	3

<rshift>, which is contained in the innermost parentheses, specifies a number of right shifts. For example, "0" specifies no shift, "1" shifts right one bit, "2" shifts right two bits, and so on. The definition below uses this form.

```
CodeMacro DEC dst:Rw
  DBIT 5(9H),3(dst(0))
EndM
```

The first five bits of the byte have the value 9H. If the remaining bits are zero, the hex value of the byte will be 48H. If the instruction:

```
DEC     DX
```

is assembled and DX has a value of 2H, then 48H + 2H = 4AH, which is the final value of the byte for execution. If this sequence had been present in the definition:

```
DBIT 5(9H),3(dst(1))
```

then the register number would have been shifted right once and the result would had been 48H + 1H = 49H, which is erroneous.

Section 6
DDT-86

6.1 DDT-86 Operation

The DDT-86™ program allows the user to test and debug programs interactively in a CP/M-86 environment. The reader should be familiar with the 8086 processor, ASM-86 and the CP/M-86 operating system as described in the CP/M-86 System Guide.

6.1.1 Invoking DDT-86

Invoke DDT-86 by entering one of the following commands:

```
DDT86
DDT86 filename
```

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 filename. If the file type is omitted from filename, .CMD is assumed. Note that DDT-86 cannot load a file of type .H86. The second form of the invoking command is equivalent to the sequence:

A>**DDT86**
DDT86 x.x
-Efilename

At this point, the program that was loaded is ready for execution.

6.1.2 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 CONTROL-C (represented in this chapter as ↑C) to end the debugging session (see Section 6.1.4). A command line may have up to 64 characters, and must be terminated with a carriage return. While entering the command, use standard CP/M line-editing functions (↑X, ↑H, ↑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. Table 6-1 summarizes DDT-86 commands. DDT-86 commands are defined individually in Section 6.2.

Table 6-1. DDT-86 Command Summary

Command	Action
A	enter assembly language statements
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 names 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.

6.1.3 Specifying a 20-Bit Address

Most DDT-86 commands require one or more addresses as operands. Because the 8086 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 Section 6.4.

6.1.4 Terminating DDT-86

Terminate DDT-86 by typing a ↑C in response to the hyphen prompt. This returns control to the CCP. Note that CP/M-86 does not have the SAVE facility found in CP/M for 8-bit machines. Thus if DDT-86 is used to patch a file, write the file to disk using the W command before exiting DDT-86.

6.1.5 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, except for a few critical regions where interrupts are disabled. While the program being tested has control of the CPU, the user's CPU state determines the state of the interrupt flag.

6.2 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.

6.2.1 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 Section 4 on 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.

6.2.2 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, the D command may be aborted by typing any character at the console.

6.2.3 The E (Load for Execution) Command

The E command loads a file into memory so that a subsequent G, T or U command can begin program execution. The E command takes the form:

```
E<filename>
```

where <filename> is the name of the file to be loaded. If no file type 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 may 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.

6.2.4 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.

6.2.5 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 respectively set one and two breakpoints 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.

6.2.6 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
```

6.2.7 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 filenames. The first filename is parsed into the default file control block at 005CH. The optional second filename (if specified) is parsed into the second part of the default file control block beginning at 006CH. The characters in <command tail> are also copied into the default command buffer at 0080H. The length of <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.

6.2.8 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.

Long displays may be aborted by typing any key during the list process. Or, enter ↑S to halt the display temporarily.

The syntax of the assembly language statements produced by the L command is described in Section 4.

6.2.9 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 will be overwritten before it is moved, because data is transferred starting from location s.

6.2.10 The R (Read) Command

The R command reads a file into a contiguous block of memory. The form is:

```
R<filename>
```

where <filename> 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 may be read into memory without overlapping. The number of files which may 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.

6.2.11 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 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, the operator may 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.

6.2.12 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 console.

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 may be aborted before n steps have been executed by typing any character at the console.

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 BDOS interrupt instruction, since DDT-86 itself makes BDOS calls and the BDOS is not reentrant. Instead, the entire sequence of instructions from the BDOS interrupt through the return from BDOS is treated as one traced instruction.

6.2.13 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 console. The U command may be aborted by striking any key at the console.

6.2.14 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, ?.

6.2.15 The W (Write) Command

The W command writes the contents of a contiguous block of memory to disk. The forms are:

```
W<filename>
W<filename>,s,f
```

where <filename> is the filename and file type 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.

6.2.16 The X (Examine CPU State) Command

The X command allows the operator 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 Table 2-1. <instruction> is the disassembled instruction at the next location to be executed, which is indicated by the CS and IP registers.

Table 6-2. Flag Name Abbreviations

Character	Name
O	Overflow
D	Direction
I	Interrupt Enable
T	Trap
S	Sign
Z	Zero
A	Auxiliary Carry
P	Parity
C	Carry

The second form allows the operator 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.

6.3 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: 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 an 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.

Table 6-3 summarizes DDT-86's default segment values.

Table 6-3. DDT-86 Default Segment Values

Command	type-1	type-2
A	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

6.4 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 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 location 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]
```

6.5 DDT-86 Sample Session

In the following sample session, the user interactively debugs a simple sort program. Comments in italic type 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
CP/M ASM86 1.1  SOURCE: SORT.A86                                PAGE   1


                        ;
                        ;       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
:0400000300000000F9
: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
*047D:0029    Breakpoint encountered.

        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.
-l
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.
-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

-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
-ald    to patch.)
047D:001D cmp si,3
047D:0020
        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 20 20 20 .............
        Set breakpoint when first 3 elements of list should be sorted.
-g,29
*047D:0029

-d100,10f See if list is sorted.
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.
-ald
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


-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,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 XCHG  AL,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.

```
-esort
```

Reload program.

```
     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.

```
-ald
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
-rsort.cmd             must read CMD file including header, so we use R
  START      END       command.
2000:0000 2000:01FF

        First 80h bytes contain header, so code starts at 80h.
-180
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
-wsort.cmd                since no length specified.)

        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
ASM-86 Invocation

Command: ASM86

Syntax: ASM86 <filename> { $ <parameters> }

where

<filename> is the 8086 assembly source file. Drive and extension are optional. The default file extension is .A86.

<parameters> are a one-letter type followed by a one-letter device from the table below.

Parameters:

form: $ Td where T = type and d = device

Table A-1. Parameter Types and Devices

Devices	Parameters				
	A	H	P	S	F
A - P	x	x	x	x	
X		x	x	x	
Y		x	x	x	
Z		x	x	x	
I					x
D					d

x = valid, d = default

<u>Valid Parameters</u>

Except for the F type, the default device is the the current default drive.

Table A-2. Parameter Types

A	controls location of ASSEMBLER source file
H	controls location of HEX file
P	controls location of PRINT file
S	controls location of SYMBOL file
F	controls type of hex output FORMAT

Table A-3. Device Types

A - P	Drives A - P
X	console device
Y	printer device
Z	byte bucket
I	Intel hex format
D	Digital Research hex format

Table A-4. Invocation Examples

ASM86 IO	Assemble file IO.A86, produce IO.HEX IO.LST and IO.SYM.
ASM86 IO.ASM $ AD SZ	Assemble file IO.ASM on device D, produce IO.LST and IO.HEX, no symbol file.
ASM86 IO $ PY SX	Assemble file IO.A86, produce IO.HEX, route listing directly to printer, output symbols on console.
ASM86 IO $ FD	Produce Digital Research hex format.
ASM86 IO $ FI	Produce Intel hex format.

Appendix B
Mnemonic Differences From the Intel Assembler

The CP/M 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:

Table B-1. Mnemonic Differences

Mnemonic Function	CP/M	INTEL
Intra segment short jump:	JMPS	JMP
Inter segment jump:	JMPF	JMP
Inter segment return:	RETF	RET
Inter segment call:	CALLF	CALL

Appendix C
ASM-86 Hexadecimal Output Format

At the user's option, ASM-86 produces machine code in either Intel or Digital Research hexadecimal format. The Intel format is identical to the format defined by Intel for the 8086. The Digital Research format is nearly identical to the Intel format, but adds segment information to hexadecimal records. Output of either format can be input to GENCMD, but the Digital Research format automatically provides segment identification. A segment is the smallest unit of a program that can be relocated.

Table C-1 defines the sequence and contents of bytes in a hexadecimal record. Each hexadecimal record has one of the four formats shown in Table C-2. An example of a hexadecimal record is shown below.

```
Byte number=>  0 1 2 3 4 5 6 7 8 9 ..............n

   Contents=>  : l l a a a a t t d d d ......... c c CR LF
```

Table C-1. Hexadecimal Record Contents

Byte	Contents	Symbol
0	record mark	:
1-2	record length	l l
3-6	load address	a a a a
7-8	record type	t t
9-(n-1)	data bytes	d d.....d
n-(n+1)	check sum	c c
n+2	carriage return	CR
n+3	line feed	LF

Table C-2. Hexadecimal Record Formats

Record type	Content	Format
00	Data record	: ll aaaa DT <data...> cc
01	End-of-file	: 00 0000 01 FF
02	Extended address mark	: 02 0000 ST ssss cc
03	Start address	: 04 0000 03 ssss iiii cc

```
ll    => record length - number of data bytes
cc    => check sum  - sum of all record bytes
aaaa  => 16 bit address
ssss  => 16 bit segment value
iiii  => offset value of start address
DT    => data record type
ST    => segment address record type
```

It is in the definition of record types 00 and 02 that Digital Research's hexadecimal format differs from Intel's. Intel defines one value each for the data record type and the segment address type. Digital Research identifies each record with the segment that contains it, as shown in Table C-3.

Table C-3. Segment Record Types

Symbol	Intel's Value	Digital's Value	Meaning
DT	00		for data belonging to all 8086 segments
		81H	for data belonging to the CODE segment
		82H	for data belonging to the DATA segment
		83H	for data belonging to the STACK segment
		84H	for data belonging to the EXTRA segment
ST	02		for all segment address records
		85H	for a CODE absolute segment address
		86H	for a DATA segment address
		87H	for a STACK segment address
		88H	for a EXTRA segment address

Appendix D
Reserved Words

Table D-1. 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	RS
RB	RW	END	ENDM	EQU
ORG	CSEG	DSEG	ESEG	SSEG
EJECT	ENDIF	TITLE	LIST	NOLIST
INCLUDE	SIMFORM	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
Instruction Mnemonics - See Appendix E.				

Appendix E
ASM-86 Instruction Summary

Table E-1. ASM-86 Instruction Summary

Mnemonic	Description	Section
AAA	ASCII adjust for Addition	4.3
AAD	ASCII adjust for Division	4.3
AAM	ASCII adjust for Multiplication	4.3
AAS	ASCII adjust for Subtraction	4.3
ADC	Add with Carry	4.3
ADD	Add	4.3
AND	And	4.3
CALL	Call (intra segment)	4.5
CALLF	Call (inter segment)	4.5
CBW	Convert Byte to Word	4.3
CLC	Clear Carry	4.6
CLD	Clear Direction	4.6
CLI	Clear Interrupt	4.6
CMC	Complement Carry	4.6
CMP	Compare	4.3
CMPS	Compare Byte or Word (of string)	4.4
CWD	Convert Word to Double Word	4.3
DAA	Decimal Adjust for Addition	4.3
DAS	Decimal Adjust for Subtraction	4.3
DEC	Decrement	4.3
DIV	Divide	4.3
ESC	Escape	4.6
HLT	Halt	4.6
IDIV	Integer Divide	4.3
IMUL	Integer Multiply	4.3
IN	Input Byte or Word	4.2
INC	Increment	4.3
INT	Interrupt	4.5
INTO	Interrupt on Overflow	4.5
IRET	Interrupt Return	4.5
JA	Jump on Above	4.5
JAE	Jump on Above or Equal	4.5
JB	Jump on Below	4.5
JBE	Jump on Below or Equal	4.5
JC	Jump on Carry	4.5
JCXZ	Jump on CX Zero	4.5
JE	Jump on Equal	4.5
JG	Jump on Greater	4.5
JGE	Jump on Greater or Equal	4.5
JL	Jump on Less	4.5
JLE	Jump on Less or Equal	4.5

Table E-1. (continued)

Mnemonic	Description	Section
JMP	Jump (intra segment)	4.5
JMPF	Jump (inter segment)	4.5
JMPS	Jump (8 bit displacement)	4.5
JNA	Jump on Not Above	4.5
JNAE	Jump on Not Above or Equal	4.5
JNB	Jump on Not Below	4.5
JNBE	Jump on Not Below or Equal	4.5
JNC	Jump on Not Carry	4.5
JNE	Jump on Not Equal	4.5
JNG	Jump on Not Greater	4.5
JNGE	Jump on Not Greater or Equal	4.5
JNL	Jump on Not Less	4.5
JNLE	Jump on Not Less or Equal	4.5
JNO	Jump on Not Overflow	4.5
JNP	Jump on Not Parity	4.5
JNS	Jump on Not Sign	4.5
JNZ	Jump on Not Zero	4.5
JO	Jump on Overflow	4.5
JP	Jump on Parity	4.5
JPE	Jump on Parity Even	4.5
JPO	Jump on Parity Odd	4.5
JS	Jump on Sign	4.5
JZ	Jump on Zero	4.5
LAHF	Load AH with Flags	4.2
LDS	Load Pointer into DS	4.2
LEA	Load Effective Address	4.2
LES	Load Pointer into ES	4.2
LOCK	Lock Bus	4.6
LODS	Load Byte or Word (of string)	4.4
LOOP	Loop	4.5
LOOPE	Loop While Equal	4.5
LOOPNE	Loop While Not Equal	4.5
LOOPNZ	Loop While Not Zero	4.5
LOOPZ	Loop While Zero	4.5
MOV	Move	4.2
MOVS	Move Byte or Word (of string)	4.4
MUL	Multiply	4.3
NEG	Negate	4.3
NOT	Not	4.3
OR	Or	4.3
OUT	Output Byte or Word	4.2

Table E-1. (continued)

Mnemonic	Description	Section
POP	Pop	4.2
POPF	Pop Flags	4.2
PUSH	Push	4.2
PUSHF	Push Flags	4.2
RCL	Rotate through Carry Left	4.3
RCR	Rotate through Carry Right	4.3
REP	Repeat	4.4
RET	Return (intra segment)	4.5
RETF	Return (inter segment)	4.5
ROL	Rotate Left	4.3
ROR	Rotate Right	4.3
SAHF	Store AH into Flags	4.2
SAL	Shift Arithmetic Left	4.3
SAR	Shift Arithmetic Right	4.3
SBB	Subtract with Borrow	4.3
SCAS	Scan Byte or Word (of string)	4.4
SHL	Shift Left	4.3
SHR	Shift Right	4.3
STC	Set Carry	4.6
STD	Set Direction	4.6
STI	Set Interrupt	4.6
STOS	Store Byte or Word (of string)	4.4
SUB	Subtract	4.3
TEST	Test	4.3
WAIT	Wait	4.6
XCHG	Exchange	4.2
XLAT	Translate	4.2
XOR	Exclusive Or	4.3

Appendix F
Sample Program

Listing F-1. Sample Program APPF.A86


```
                        title 'Terminal Input/Output'
                        pagesize 50
                        pagewidth 79
                        simform
                        ;
                        ;****** Terminal I/O subroutines ********
                        ;
                        ;       The following subroutines
                        ;       are included:
                        ;
                        ;       CONSTAT   -  console status
                        ;       CONIN     -  console input
                        ;       CONOUT    -  console output
                        ;
                        ;       Each routine requires CONSOLE NUMBER
                        ;       in the BL - register
                        ;
                        ;
                        ;       *****************
                        ;       *  Jump table:  *
                        ;       ****************
                        ;
                        CSEG            ; start of code segment
                        ;
                        jmp_tab:
0000 E90600                     jmp     constat
0003 E91900                     jmp     conin
0006 E92B00                     jmp     conout
                        ;
                        ;
                        ;       ***********************
                        ;       *  I/O port numbers   *
                        ;       ***********************
```

```
CP/M ASM86 1.1  SOURCE: APPF.A86      Terminal Input/Output          PAGE
 2

                     ;
                     ;         Terminal 1:
                     ;
  0010               instatl        equ     10h      ; input status port
  0011               indatal        equ     11h      ; input port
  0011               outdatal       equ     11h      ; output port
  0001               readyinmaskl   equ     01h      ; input ready mask
  0002               readyoutmaskl  equ     02h      ; output ready mask
                     ;
                     ;         Terminal 2:
                     ;
  0012               instat2        equ     12h      ; input status port
  0013               indata2        equ     13h      ; input port
  0013               outdata2       equ     13h      ; output port
  0004               readyinmask2   equ     04h      ; input ready mask
  0008               readyoutmask2  equ     08h      ; output ready mask
                     ;
                     ;
                     ;       ***********
                     ;       * CONSTAT *
                     ;       ***********
                     ;
                     ;       Entry: BL - reg = terminal no
                     ;       Exit:  AL - reg = 0 if not ready
                     ;                         0ffh if ready
                     ;
                     constat:
 0009 53E83F00               push bx ! call okterminal
                     constatl:
 000D 52                     push dx
 000E B600                   mov  dh,0               ; read status port
 0010 8A17                   mov  dl,instatustab [BX]
 0012 EC                     in   al,dx
 0013 224706                 and  al,readyinmasktab [bx]
 0016 7402                   jz   constatout
 0018 B0FF                   mov  al,0ffh
```

```
CP/M ASM86 1.1  SOURCE: APPF.A86     Terminal Input/Output        PAGE
 3


                    constatout:
001A 5A5B0AC0C3             pop dx ! pop bx ! or al,al ! ret
                    ;
                    ;
                    ;       *********
                    ;       * CONIN *
                    ;       *********
                    ;
                    ;       Entry: BL - reg = terminal no
                    ;       Exit:  AL - reg = read character
                    ;
001F 53E82900       conin:  push bx ! call okterminal !
0023 E8E7FF         coninl: call constatl            ; test status
0026 74FB                   jz   coninl
0028 52                     push dx                  ; read character
0029 B600                   mov  dh,0
002B 8A5702                 mov  dl,indatatab [BX]
002E EC                     in   al,dx
002F 247F                   and  al,7fh              ; strip parity bit
0031 5A5BC3                 pop dx ! pop bx ! ret
                    ;
                    ;
                    ;       **********
                    ;       * CONOUT *
                    ;       **********
                    ;
                    ;       Entry:  BL - reg = terminal no
                    ;               AL - reg = character to print
                    ;
0034 53E81400       conout: push bx ! call okterminal
0038 52                     push dx
0039 50                     push ax
003A B600                   mov  dh,0                ; test status
003C 8A17                   mov  dl,instatustab [BX]
                    conoutl:
003E EC                     in   al,dx
```

```
003F 224708             and  al,readvoutmasktab [BX]
0042 74FA               jz   conoutl
0044 58                 pop  ax                  ; write byte
0045 8A5704             mov  dl,outdatatab [BX]
0048 EE                 out  dx,al
0049 5A5BC3             pop dx ! pop bx ! ret
                ;
                ;
                ;       ++++++++++++++
                ;       + OKTERMINAL +
                ;       ++++++++++++++
                ;
                ;       Entry:  BL - reg = terminal no
                ;
                okterminal:
004C 0ADB               or   bl,bl
004E 740A               jz   error
0050 80FB03             cmp  bl,length instatustab + 1
0053 7305               jae  error
0055 FECB               dec  bl
0057 B700               mov  bh,0
0059 C3                 ret
                ;
005A 5B5BC3     error:  pop bx ! pop bx ! ret    ; do nothing
                ;
                ;************** end of code segment ***************
                ;
                ;       ****************
                ;       * Data segment *
                ;       ****************
                ;
                        dseg

                ;       **************************
                ;       * Data for each terminal *
                ;       **************************
```

```
CP/M ASM86 1.1  SOURCE: APPF.A86       Terminal Input/Output         PAGE
 5

                    ;

 0000 1012          instatustab     db      instatl,instat2
 0002 1113          indatatab       db      indatal,indata2
 0004 1113          outdatatab      db      outdatal,outdata2
 0006 0104          readyinmasktab  db      readyinmaskl,readyinmask2
 0008 0208          readyoutmasktab db      readyoutmaskl,readyoutmask2
                    ;
                    ;*************** end of file **********************
                    end

END OF ASSEMBLY. NUMBER OF ERRORS:   0
```

Appendix G
Code-Macro Definition Syntax

```
<codemacro> ::= CODEMACRO <name> [<formal$list>]
                [<list$of$macro$directives>]
                ENDM

<name> ::= IDENTIFIER

<formal$list> ::= <parameter$descr>[{,<parameter$descr>}]

<parameter$descr> ::= <form$name>:<specifier$letter>
                      <modifier$letter>[(<range>)]

<specifier$letter> ::= A | C | D | E | M | R | S | X

<modifier$letter> ::= b | w | d | sb

<range> ::= <single$range>|<double$range>

<single$range> ::= REGISTER | NUMBERB

<double$range> ::= NUMBERB,NUMBERB | NUMBERB,REGISTER |
                   REGISTER,NUMBERB | REGISTER,REGISTER

<list$of$macro$directives> ::= <macro$directive>
                               {<macro$directive>}

<macro$directive> ::= <db> | <dw> | <dd> | <segfix> |
                      <nosegfix> | <modrm> | <relb> |
                      <relw> | <dbit>

<db> ::= DB NUMBERB | DB <form$name>

<dw> ::= DW NUMBERW | DW <form$name>

<dd> ::= DD <form$name>

<segfix> ::= SEGFIX <form$name>

<nosegfix> ::= NOSEGFIX <form$name>

<modrm> ::= MODRM NUMBER7,<form$name> |
            MODRM <form$name>,<form$name>

<relb> ::= RELB <form$name>

<relw> ::= RELW <form$name>

<dbit> ::= DBIT <field$descr>{,<field$descr>}
```

```
<field$descr> ::= NUMBER15 ( NUMBERB ) |
                  NUMBER15 ( <form$name> ( NUMBERB ) )

<form$name> ::= IDENTIFIER
```

```
NUMBERB is 8-bits
NUMBERW is 16-bits
NUMBER7 are the values 0, 1,. .  , 7
NUMBER15 are the values 0, 1,. .  , 15
```

Appendix H
ASM-86 Error Messages

There are two types of error messages produced by ASM-86: fatal errors and diagnostics. Fatal errors occur when ASM-86 is unable to continue assembling. Diagnostics 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
DISK FULL
DIRECTORY FULL
DISK READ ERROR
CANNOT CLOSE
SYMBOL TABLE OVERFLOW
PARAMETER ERROR
```

ASM-86 reports semantic and syntax errors by placing a numbered ASCII message in front of the erroneous source line. If there is more than one error in the line, only the first one is reported. Table H-1 summarizes ASM-86 diagnostic error messages.

Table H-1. ASM-86 Diagnostic Error Messages

Number	Meaning
0	ILLEGAL FIRST ITEM
1	MISSING PSEUDO INSTRUCTION
2	ILLEGAL PSEUDO INSTRUCTION
3	DOUBLE DEFINED VARIABLE
4	DOUBLE DEFINED LABEL
5	UNDEFINED INSTRUCTION
6	GARBAGE AT END OF LINE - IGNORED
7	OPERAND(S) MISMATCH INSTRUCTION
8	ILLEGAL INSTRUCTION OPERANDS
9	MISSING INSTRUCTION
10	UNDEFINED ELEMENT OF EXPRESSION
11	ILLEGAL PSEUDO OPERAND
12	NESTED "IF" ILLEGAL - "IF" IGNORED
13	ILLEGAL "IF" OPERAND - "IF" IGNORED
14	NO MATCHING "IF" FOR "ENDIF"
15	SYMBOL ILLEGALLY FORWARD REFERENCED - NEGLECTED
16	DOUBLE DEFINED SYMBOL - TREATED AS UNDEFINED
17	INSTRUCTION NOT IN CODE SEGMENT
18	FILE NAME SYNTAX ERROR
19	NESTED INCLUDE NOT ALLOWED
20	ILLEGAL EXPRESSION ELEMENT
21	MISSING TYPE INFORMATION IN OPERAND(S)
22	LABEL OUT OF RANGE
23	MISSING SEGMENT INFORMATION IN OPERAND
24	ERROR IN CODEMACROBUILDING

Appendix I
DDT-86 Error Messages

Table I-1. DDT-86 Error Messages

Error Message	Meaning
AMBIGUOUS OPERAND	An attempt was made to assemble a command with an ambiguous operand. Precede the operand with the prefix "BYTE" or "WORD".
CANNOT CLOSE	The disk file written by a W command cannot be closed.
DISK READ ERROR	The disk file specified in an R command could not be read properly.
DISK WRITE ERROR	A disk write operation could not be successfully performed during a W command, probably due to a full disk.
INSUFFICIENT MEMORY	There is not enough memory to load the file specified in an R or E command.
MEMORY REQUEST DENIED	A request for memory during an R command could not be fulfilled. Up to eight blocks of memory may be allocated at a given time.
NO FILE	The file specified in an R or E command could not be found on the disk.
NO SPACE	There is no space in the directory for the file being written by a W command.
VERIFY ERROR AT s:o	The value placed in memory by a Fill, Set, Move, or Assemble command could not be read back correctly, indicating bad RAM or attempting to write to ROM or non-existent memory at the indicated location.

Chapter IV

AS-100 CP/M-86

User's Guide

Canon AS-100 Series

Preface

The CP/M-86™ operating system for the Canon AS-100 is based on the standard CP/M-86 developed by Digital Research, and includes various additional functions which make the best of the hardware of the AS-100. This manual describes primarily those functions which are peculiar to AS-100 CP/M-86; readers are expected to be familiar with standard CP/M-86. Readers who are using CP/M-86 for the first time should first read the Standard CP/M-86 User's Guide. In this manual, functions peculiar to AS-100 are identified by the notation "AS-100 CP/M-86."

* CP/M-86 is trademark of Digital

CONTENTS

APPENDIX

CHAPTER 1 SYSTEM CONFIGURATION

1-1 Features of AS-100 CP/M-86

AS-100 CP/M-86 was developed by adding a variety of functions to Digital Research's CP/M-86 to make the best use of the hardware and application system of the AS-100 computer. Since these additional functions do not reduce the capability of CP/M-86 can be used with little or no modification on AS-100. AS-100 CP/M-86 has been improved as follows.

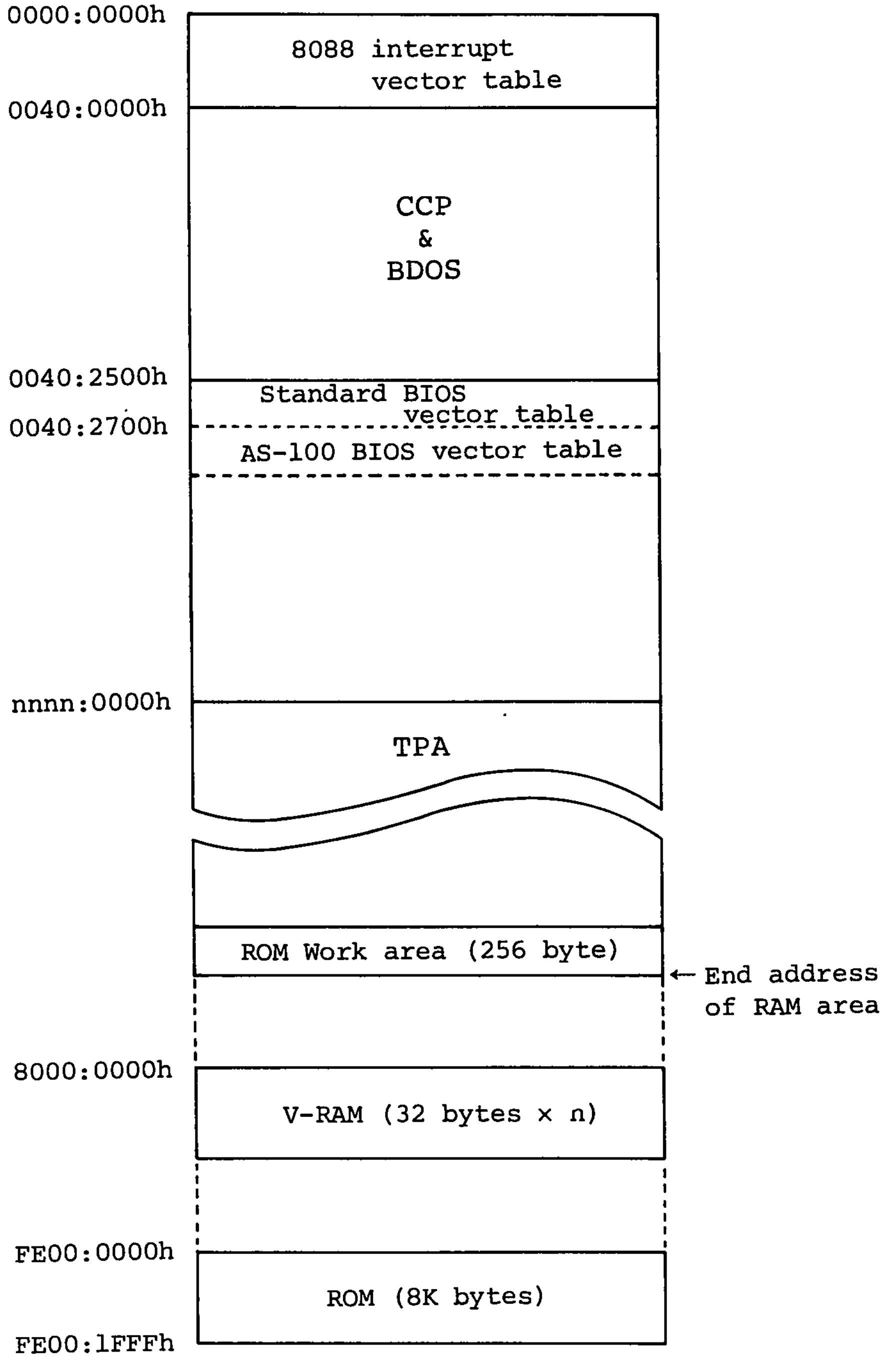
- A new BIOS with added functions has been developed and implemented.
- A new boot ROM and secondary boot have been developed.
- Extended commands have been added.

Features of AS-100 CP/M-86 are described below.

- 5-inch floppy disks are accessed in units of 512-byte sectors, while 8-inch floppy disks are accessed in units of 1024-byte sectors; this improves the efficiency of file access.
- Single sided, single density, 128-byte sector, 8-inch floppy disks for the standard CP/M-86 system can also be used by performing a simple operation.
- A function call is provided which makes it possible to access data from double sided, single density, 128-byte sector 8-inch floppy disks or from double sided, double density, 256-byte sector, 8-inch floppy disk.
- Various CRT display control functions (such as graphic display, color and attribute specification, scroll area specification and cursor control functions) are provided.
- A variety of handlers, such as a printer handler or RS232C interface support program can be linked to BIOS.
- A media backup utility is provided which makes it possible to copy the contents of floppy disks at high speed.
- Commands can be written in a SUBMIT file for automatic sequential execution when the power is turned on.

1-2 AS-100 CP/M-86 Configuration

The figure below shows the memory configuration when CP/M-86 is running.



BIOS of AS-100 CP/M-86 performs the following processing.

(1) Processing of BIOS entries from standard CP/M-86.

(2) AS-100 hardware support processing which does not use BDOS functions.

The following table shows the BIOS function numbers corresponding to the functions of standard CP/M-86 and BIOS entry points.

BIOS NO.	Function	BIOS jump address	Explanation
0	INIT	40 : 2500	Cold start
1	WBOOT	40 : 2503	Warm start
2	CONST	40 : 2506	Console status check
3	CONIN	40 : 2509	Inputs a character from the console.
4	CONOUT	40 : 250C	Outputs a character to the console.
5	LIST	40 : 250F	Outputs a character to the printer.
6	PUNCH	40 : 2512	Outputs a character to the punch device.
7	READER	40 : 2515	Inputs a character from the reader device.
8	HOME	40 : 2518	Moves the head to track 00.
9	SELDSK	40 : 251B	Selects the disk drive.
10	SETTRK	40 : 251E	Specifies the track number.
11	SETSEC	40 : 2521	Specifies the sector number.
12	SETDMA	40 : 2524	Specifies the DMA offset address.
13	READ	40 : 2527	Reads data from the specified sector.
14	WRITE	40 : 252A	Writes data to the specified sector.
15	LISTST	40 : 252D	List device status check
16	SECTRAN	40 : 2530	Converts sector number.
17	SETDMAB	40 : 2533	Specifies the DMA segment address.
18	GETSEGB	40 : 2536	Obtains the memory control table address.
19	GETIOB	40 : 2539	Obtains the contents of IOBYTE.
20	SETIOB	40 : 253C	Sets IOBYTE.

* All the above addresses are represented in hexadecimal.

After the system has been initialized, the 8088 interrupt vector table is set to 224 (E0h) which is the standard interrupt (function call) of CP/M-86. The AS-100 CP/M-86 supports the following interrupt calls in addition to the standard interrupt call.

INT 240 (F0h): Used for printer and RS-232C.
INT 241 (F1h): Used for graphic functions.
INT 242 (F2h): Used for floppy disk access.

These interrupt calls are peculiar to the AS-100 CP/M-86 and hereafter they are referred to as the AS-100 function calls. The BIOS entry table for the AS-100 function calls is in the memory area starting at 40:2700h. Refer to the related chapters for details of each AS-100 function call.

CHAPTER 2 SYSTEM ACTIVATION

The AS-100 CP/M-86 is activated by the system activation function stored in ROM. Any application system can be automatically started by the automatic SUBMIT function after the operating system has been activated. This chapter describes the system activation and automatic SUBMIT functions.

2-1 System Loading

The system activation program for the AS-100 is stored in an 8KB ROM of addresses from FE00:0000 to FE00:1FFF. This ROM has the following programs as well as the boot for system activation.

- Initialization program
- Self-diagnostic program
- Boot loader
- ROM debugger

When the power of the AS-100 computer is turned on, control is transferred to the system activation program in the ROM. Control is also transferred to the program when the reset switch (in the hole below the lower left of the screen) is pressed. (The reset switch can be pressed by using a thin object such as ball pen.)

(1) Initialization

Control is first transferred to the initialization program in the ROM to reset registeres used by the hardware and to clear work areas used by the programs in the ROM.
After initialization, control is transferred to the self-diagnostic program.

(2) Self-diagnostic program

The self-diagnostic program performs the following.

. RAM check

Data is written in and read from every byte of the RAM area installed.

. ROM check

The contents of ROM are checked by the checksum.

. Timer check

Software timers 1 to 3 are checked.

. Keyboard check

The keyboard is checked.

If an error is detected during the above checks, the corresponding error message (described later) is displayed and control is transferred to the ROM debugger. When the self-diagnostic program is completed normally, control is transferred to the boot loader.

(3) Boot loader

The boot loader loads the secondary boot program which loads the CP/M-86 system and transfers control to it. The secondary boot program is stored on the system tracks (tracks 0 and 1) of the disk. The disk drive and disk size (5 or 8 inch) are determined by the DIP switch.

For the DIP switch, refer to Appendix C. The sector size is automatically detected by the secondary boot program when it reads the contents of sector 1 of track 0.
If an error occurs while the secondary boot program is being loaded, an error message (described later) is displayed and control is transferred to the ROM debugger. When loading is normally completed, control is transferred to the secondary boot program. Control can also be transferred to the secondary boot program from the ROM debugger.

(4) Secondary boot program

This program is not included in ROM but it is loaded into memory by the boot loader. This program is written on the system track when a floppy disk is formatted by the FORMAT command.
The secondary boot program loads the system (CPM.SYS) from the system disk (from which the secondary boot program is also loaded) into memory and transfers control to CPM.SYS.
If an error occurs during loading CPM.SYS, an error message is displayed and the secondary boot program loops to stop further execution of system activation.

(5) ROM debugger

The ROM debugger is activated when an error occurs during execution of the self-diagnostic program or boot loader, or when the stop switch (right to the reset switch) is pressed.
The ROM debugger has the following functions.

. Displays and changes the contents of memory.

. Displays and changes the contents of registers.

. Inputs data to an output port.

. Executes a program (with break points set).

. Executes a program step by step.

The ROM debugger is used to check the contents of the CP/M-86 system and handler program in memory. For user programs, DDT-86 is a more effective checking tool.
For use of the ROM debugger, refer to Appendix B.

Messages displayed during system activation are as follows.

nnnK-BYTES SYSTEM

Indicates the size of RAM installed when the self-diagnostic program is executed. nnn is a decimal number.

CP/M-86 LOADER Vn.mm

Displayed when the secondary boot program is executed. Vn.mm indicates the version number of the secondary boot program.

SEGMENT ADDRESS = nnnn

Indicates the address of the segment currently being loaded in hexadecimal when CPM.SYS is loaded.

LAST OFFSET = nnnn

Indicates the last offset in hexadecimal when loading CP/M-86 is completed. TPA starts at the next address.

BIOS (A) Vn.mm by Canon Inc.

Indicates that CP/M-86 is activated. (A) indicates that the ASCII character set is being used. Vn.mm indicates the version number of BIOS.

8086/8088 DEBUGGER Vn.mm

Indicates that the ROM debugger is activated. Vn.mm indicates the version number of the ROM debugger. The debugger's prompt is "*".

RAM ERR AT nnnn:mmmm

An error was detected during execution of RAM check by the self-diagnostic program. nnnn:mmmm indicates the segment and offset addresses where the error is detected.

ERR CODE = nn

An error was detected by the self-diagnostic program.

nn = 02 : ROM checksum error

nn = 03 : Timer failure

nn = 04 : Keyboard failure

BOOT ERROR

An error occurred during loading of the secondary boot program.

ERROR IN READING CPM.SYS

An error occurred during loading of CPM.SYS by the secondary boot program.

THE FILE CPM.SYS NOT FOUND ON THIS DISK

The disk does not contain CPM.SYS.

2-2 Automatic SUBMIT Function

After the CP/M-86 system program has been loaded, control is transferred to it. CP/M-86 then searches the directory of the system load device for the START.SUB file. If the START.SUB file cannot be found, CP/M-86 displays "A>" to prompt the operator to enter a command.
When the START.SUB file is found, CP/M-86 calls the SUBMIT command with START.SUB specified as the object file, that is, it sets "SUBMIT START" in the console buffer and transfers control to CCP.

With this feature, an application program can be automatically activated by turning on the power of the AS-100 computer. This is useful when automatically executing handler commands required for the application system.

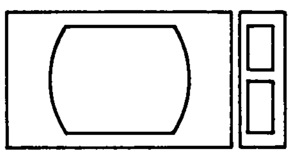
The START.SUB file must be executable by the SUBMIT command.

This chapter describes configuration of the floppy disk drives used for the AS-100 series computer, specifications of floppy disks and the function calls relating to disk I/O operation which are peculiar to the AS-100 CP/M-86.

3-1 Floppy Disk Drives and Device Names

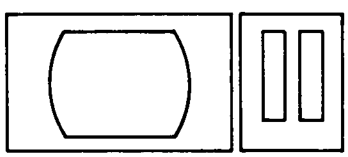
Two types of floppy disk drives are provided for the AS-100 series computer: mini floppy disk unit (A-1300) and standard floppy disk unit (A-1330). Each unit has two floppy disk drives. The former uses 5-inch (more exactly, $5^{1}/_{4}$-inch) mini floppy disks and the latter uses 8-inch floppy disks. The following four combinations of floppy disk units and the AS-100 computer are available.

1)



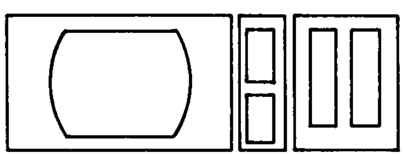
Display unit +
minifloppy unit

2)



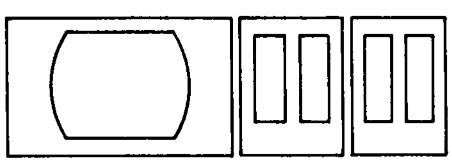
Display unit +
standard floppy unit

3)



Display unit + minifloppy
unit + standard floppy unit

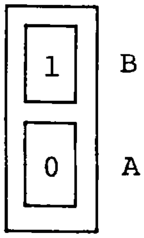
4)



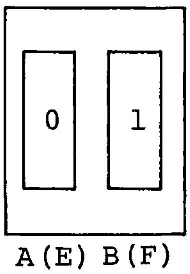
Display unit +
two standard floppy units

Physical unit addresses 0 and 1 or 2 and 3 are assigned to a disk unit according to the setting of the DIP switch in the floppy disk unit. (For setting of the DIP switch, refer to Chapter C.) Logical device names A:, B:, C: and D: are assigned to physical unit address 0, 1, 2 and 3, respectively.

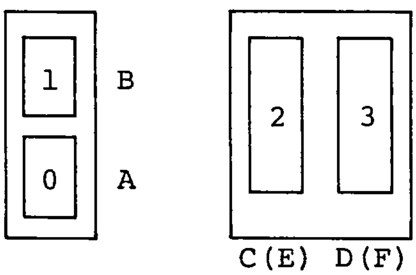
Since the CP/M-86 system is always loaded from device A:, there is no system configuration including devices C: and D: only. Any of mini-floppy unit and standard floppy unit may be assigned logical device names A: and B:. Allowable combinations of logical device names are as follows.

1)


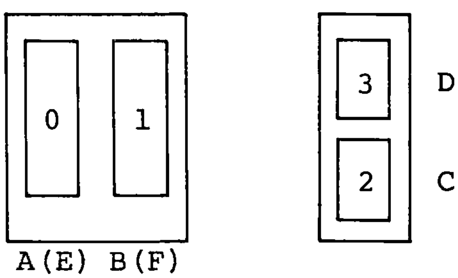
One mini floppy unit

2)


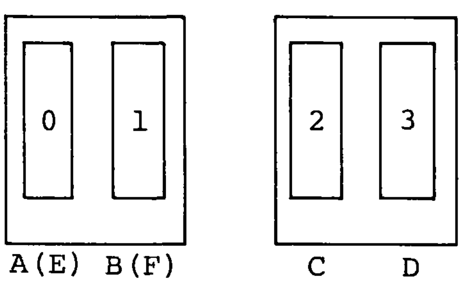
One standard floppy unit

3)


One mini floppy unit and one standard floppy unit (1)

4)


One mini floppy unit and one standard floppy unit (2)

5)


Two standard floppy units

In the above figures, numbers indicate physical unit addresses and alphabetic characters indicate logical device names. Logical device names in parentheses indicate that these names can be used instead of logical device names A: and B:, or C: and D:. These names are used when single sided, single density, 128-byte sector, 8-inch disks for the standard CP/M-86 are used. Details are explained in the next section.

3-2 Floppy Disks

Specifications of floppy disks used for floppy disk units of the AS-100 series computer are shown below.

Floppy disk unit	A-1300 mini-floppy unit	A-1330 standard floppy unit	
Type	5-inch, double sided, double density	8-inch, double sided, double density	8-inch, single sided. single density
Sector size (Byte/Sector)	512	1024	128
Track size (Sector/Track)	8 *	8	26
Number of tracks (Track)	80	77	77
Interleaved sectors	4	3	6
Block size (Byte)	2048	2048	1024
Total number of blocks (Block)	312	600	243
Total amount of data (K Bytes)	620	1196	241
Number of directory blocks (Block)	2	2	2
Total number of directory entries	128	128	64
Number of system tracks (Track)	2	2	2

* Although 9 sectors are assigned to each track, 8 sectors are used to maintain compatibility with MS-DOS.**

** MS-DOS is trade mark of Microsoft, Inc.

Device names A: to D: are assumed to be used for double sided, double density, 512-byte sector, 5-inch floppy disks or double sided, double density. 1024-byte sector, 8-inch floppy disks. These media (disks) must be formatted using the FORMAT command before use.
Refer to CHAPTER 8 for details of the FORMAT command.

3-3 Device Names E: and F:

Device names E: and F: can be assigned to standard 8-inch floppy disk drives. These names indicate that the drives are used for single sided, single density, 128-byte sector, 8-inch floppy disks which are standard for CP/M-86. That is, when either E: or F: is specified, BIOS assumes that the disk inserted in the specified drive is of the CP/M-86 standard. Note that logical device names E: and F: are assigned to disk drives which are also assigned any of logical device names A: to D:.

The procedures for copying a file on a CP/M standard disk (single sided, single density) to a file on a double sided, double density disk for the AS-100 CP/M-86 are shown below.

Assume a AS-100 system with one A-1330 standard floppy disk unit. Load the AS-100 CP/M-86 system disk in A: and a standard CP/M-86 disk which contains file TEST.TXT in B:.

File TEXT.TXT can be copied to the disk in A: by the following command.

```
PIP A:=F:TEXT.TXT
```

If device name B: is used instead of F:, an error results. The following command makes it possible to list the entries of the directory of the disk in B:.

```
DIR F:
```

Device name F: can be specified in other commands when a standard CP/M-86 disk is loaded in B:.

Which disk drive is assigned device name E: or F: is shown in Section 3-1.

3-4 AS-100 Function Call

The standard CP/M-86 allows the user to use the following BIOS functions directory by calling BDOS function 50 (direct BIOS call).

BIOS F#	Name	Function
8	HOME	Moves the head to track 00.
9	SELDSK	Selects the drive.
10	SETTRK	Sets the track number.
11	SETSEC	Sets the sector number.
12	SETDMA	Sets the DMA offset address.
13	READ	Reads data from the specified sector.
14	WRITE	Writes data to the specified sector.
16	SECTRAN	Converts the sector number.
17	SETDMAB	Sets the DMA segment address.

Refer to the Standard CP/M-86 System Guide for details of these BIOS calls.

In addition to the above, special function calls for floppy disk access are provided for the AS-100 CP/M-86. These function calls are used to change the drive information without using the normal BIOS portion. Some of these function calls support access to 8-inch floppy disks other than those of the AS-100 CP/M-86 standard format.

The function calls for floppy disk access can be executed by the following sequence.

```
MOV   AL,Pn     ; Set parameter.
MOV   CL,Fc     ; Set function code.
INT   242       ; Call function.
```

The function codes used are as follows.

Function code	Name	Function
0	MTRON	Starts the motor of the specified 5-inch disk drive.
1	MTROFF	Starts the motor off timer of the specified 5-inch disk drive.
2	SELECT	Selects the drive.
3	DISSEL	Deselects the drive.
4	CHGSPC	Switches the drive access parameter. (5/8 inch)
5	GET128	Reads data from an 8-inch, double sided, single density, 128-byte sector disk.
6	GET256	Reads data from an 8-inch, double sided, double density, 256-byte sector disk.
9	PUT128	Writes data to an 8-inch, double sided, single density, 128-byte sector disk.
10	PUT256	Writes data to an 8-inch, double sided, double density, 256-byte sector disk.

Function codes 0 to 4 are used when the user wants to control a floppy disk drive independent of the AS-100 CP/M-86 BIOS functions. To read/write data practically, knowledgement of the hardware of floppy disk controller (FDC) is necessary.

Function codes 5, 6, 9 and 10 can be used alone and knowledgement of the hardware of FDC is not required to use them.

(1) MTRON

Function code: CL ← 0

Parameter: AL ← Drive number 0 to 3

Function: This function starts the motor of the specified 5-inch disk drive. If the motor off timer has already been set, it is canceled. This func-

tion does nothing when an 8-inch drive is specified or a drive which is not installed is specified.
The specified drive must have been selected previously.

(2) MTROFF

Function code: CL ← 1

Parameter: AL ← Drive number 0 to 3

Function: This function starts the motor off timer of the specified 5-inch disk drive. When the timer is started, the motor will stop after 30 seconds.
This function does nothing if an 8-inch disk drive or a drive which is not installed is specified. The specified drive must have been selected previously.

(3) SELECT

Function code: CL ← 2

Parameter: AL ← Drive number 0 to 3

Function: This function selects the specified drive. It does nothing if a drive which is not installed is specified.

(4) DISSEL

Function code: CL ← 3

Parameter: None

Function: This function deselects all the selected drives.

(5) CHGSPC

Function code: CL ← 4

Parameter: AL ← Drive type 10h: 8 inch
02h: 5 inch

Function: This function alternates the parameters of the currently selected drives

5 inch: SRT=3, HUT=240, HLT=50 msec

8 inch: SRT=4, HUT=480, HLT=48 msec

(6) 8-inch floppy disk I/O

These functions make it possible to read/write data from/to an 8-inch disk of the double sided, single density, 128-byte sector format or the double sided, double density, 256-byte sector format.

Function codes:	Single density, 128-byte sector read	CL ← 5
	Single density, 128-byte sector write	CL ← 9
	Double density, 256-byte sector read	CL ← 6
	Double density, 256-byte sector write	CL ← 10

Parameters:

- AL ← Drive number 0 to 3
- AH ← Track number 0 to 76
- CH ← Head number 0 or 1
- DL ← Sector number 1 to 26
- DH ← Number of sectors read or written 1 to 26
- ES:BX ← I/O buffer address

Return codes: When execution is finished, one of the following return codes is set to AL.

- 0: normal completion
- 1: drive not ready
- 2: read error
- 3: write error
- 4: write protected
- 6: seek error
- 16: parameter error
- 17: number of sectors too great
- 18: buffer overrun

Note: The number of sectors to be read or written must be less than the number of sectors between the specified sector and the last sector on the specified track.

Ex) DH must be less than 8 when DL=20

The I/O buffer must be within 64K byte bank.

Ex) DH must be less than 17 when ES:BX=F00:800 and one sector consists of 128 bytes.

Read-after-write check is not performed when writing data. Access is made assuming that one track consists of 26 sectors.

CHAPTER 4 CRT DISPLAY

4-1 Outline

The CRT display unit of the AS-100 series computer uses the bit mapped system of 640 × 400 dots, which makes it possible to achieve various display functions such as graphic function.

The BIOS CONOUT function of the AS-100 CP/M-86 is improved to make the best of the above feature. In this chapter, the CONOUT function is mainly explained as well as special functions of the CRT display unit.

4-1-1 V-RAM configuration

Two models of CRT display units are available for the AS-100 computer: the color model and monochrome mode. The color model uses the RGB method and its V-RAM can store information for three screen frames.

Model	Color model	Monochrome model	
		2 frames	3 frames
Number of frames (V-RAM)	3	2	1

About 32K bytes are used for storing information of 640 x 400 dot bit map for each screen frame.

Major difference between the monochrome two frame model and one frame model is that high brightness and blinking display are possible with the two frame model. Underlined characters and reverse display are possible for all models.

4-1-2 Palette method

Selection of colors with the color model and selection of display attributes with the monochrome model are made through registers called palette. Although 8 palette registers (No. 0 to No. 7) are provided, the number of palette registers which can be used by each model differs as follows.

Color model	Palettes 0 to 7
Monochrome 2 screen frame model	0 to 3
Monochrome 1 screen frame model	0, 1

Each palette register store the initial value, but it can be changed by the control sequence described later. When a character or pattern is displayed, a foreground palette number is specified for dots which are set (foreground) and a background palette number is specified for dots which are not set (background). Each dot is displayed with the color or attribute assigned to the corresponding palette.

With the color model, one of 27 colors (combinations of three primary colors (red, green and blue) of standard brightness and three primary colors of half brightness) can be assigned to any of palette 0 to 7. With the monochrome 2-frame model, one of the non-display, standard brightness, high brightness and blinking attributes can be assigned to palettes 0 to 3. With the monochrome 1-frame model, either non-display or standard brightness attribute can be assigned to palettes 0 and 1. The figure below shows the concept of palette for the color model.

Combination of colors

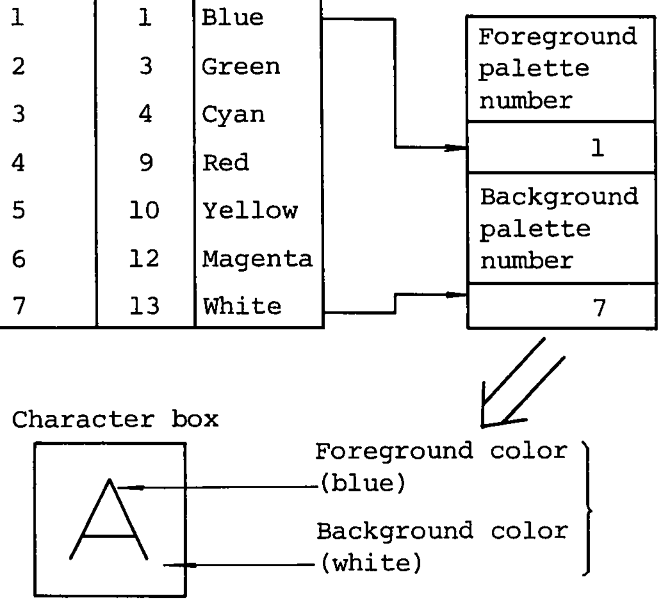
r	R	g	G	b	B	Value
0	0	0	0	0	0	0
0	0	0	0	0	1	1
0	0	0	0	1	1	2
0	0	0	1	0	0	3
0	0	0	1	0	1	4
0	0	0	1	1	1	5
1	1	0	1	1	1	23
1	1	1	1	0	0	24
1	1	1	1	0	1	25
1	1	1	1	1	1	26

RGB: Primary colors of standard brightness

rgb: Primary colors of half brightness

Palette

Number	Value	Color
0	0	Black
1	1	Blue
2	3	Green
3	4	Cyan
4	9	Red
5	10	Yellow
6	12	Magenta
7	13	White

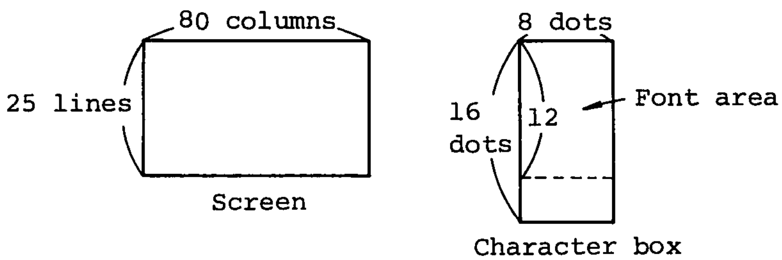


4-1-3 Display modes

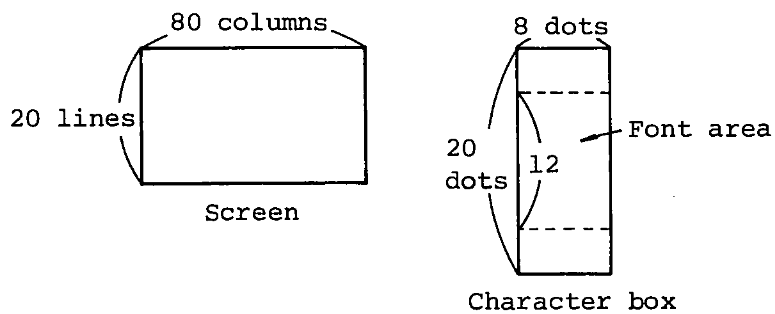
Number of lines within screen and size of character box

The 25-line mode and 20-line mode are available and the size of character box differs according to the line mode selected.

25-line mode



20-line mode



The size of each character does not vary if the number of lines within the screen is changed. The 25-line mode is automatically selected at initialization.

Scroll modes

Two scroll modes, the smooth scroll mode and jump (line) scroll mode are supported. The scroll mode is initially set to the smooth mode. These modes are effective only when the entire screen is set to the scroll area. When a scroll area is set within a partial area of the screen, the partial scroll mode is applied to that area.

Cursor types

Two types of cursor can be used: the character cursor indicates a character box location and the graphic cursor indicates a graphic dot location. For both types of cursors, their locations can be moved and read, and whether they are displayed or not can be specified by software. The two cursors can be displayed simultaneously.

Pointing device modes

A pointing device can be used in either the character cursor mode or graphic cursor mode: the pointing device is used to move the character cursor in the character cursor mode and is used to move the graphic cursor in the graphic cursor mode.

4-2 ASCII Characters

Any of alphabetic characters and semigraphic characters can be displayed at the cursor location. The cursor then moves to the right one character space. When the cursor is located at the end of a line (at column 80), it does not move even if a character is displayed at that location, and the cursor moves to the beginning of the next line when the next character to be displayed is output to the CRT display unit. Characters displayed can be reversed or underlined by the control sequence described in 4-5.
An ASCII code table is shown in Appendix A.

4-3 Control Characters

When a control character is output to the CRT display unit, the corresponding function is performed.

Control characters and their functions are as follows.

(1) BEL (07h) - bell

Generates the buzzer sound.

(2) BS (08h) - back space

Moves the cursor to the left one space, or moves it to the right end of the above line when it is at the left end of a line. When the cursor is at the home position, this character does nothing.

(3) HT (09h) - horizontal tabulation

Moves the cursor to the next tabulation position. When the cursor is at a position after column 73, this character moves it to the beginning of the next line. Tabulation positions are set every 8 columns as follows.

1.....9.....17.....25 ∿ 73.....80

(4) LF (0Ah) - line feed

Moves the cursor to the same column on the next line. When the cursor is at the lowest line of the scroll area, the area is scrolled up.

(5) VT (0Bh)

The same as LF.

(6) FF (0Ch)

The same as LF.

(7) CR (0Dh) - carriage return

Moves the cursor to the beginning (left end) of the current line.

(8) ESC (1Bh) - escape

Identifies the escape and control sequences.

(9) DEL (7Fh) - delete

Erases the character at the left of the cursor location and moves the cursor to the left one space.

4-4 Escape Sequences

An escape character (1Bh) followed by a character controls the CRT display unit as shown below.

No.	ESC sequence	Function
1	ESC D (1B44h)	Index
2	ESC E (1B45h)	New line
3	ESC M (1B4Dh)	Reverse index
4	ESC c (1B63h)	Initialization

(1) Index (ESC D)

The same as LF.

(2) New line (ESC E)

The same as CR + LF.

(3) Reverse index (ESC M)

Moves the cursor up one line. When the cursor is at the uppermost line of the scroll area, the area is scrolled down one line.

(4) Initialization (ESC c)

Initializes the screen: all modes are initialized, the entire screen is cleared and the cursor is moved to the home position. The table lists the initial conditions of various modes.

Mode	Initial condition
Character cursor display	ON
Graphic cursor display	OFF
Line mode	25-line mode
Scroll mode	Smooth scroll
Scroll area	Entire screen
Character box mode	8 x 16 (25-line mode)
Character cursor location	(1, 1)
Graphic cursor location	(320, 200)
Graphic, current point coordinates	(0, 0)
Pointing device mode	Character cursor mode

Palette registers are set as follows.

Palette number	Color CRT	Monochrome CRT	
		2-frame V-RAM	1-frame V-RAM
0	0 : Black	0: Black (non-display)	0: Black (non-display)
1	1 : Blue	2: High brightness	1: Standard brightness
2	3 : Green	27:Standard blinking	
3	4 : Cyan	1: Standard brightness	
4	9 : Red		
5	10: Magenta		
6	12: Yellow		
7	13: White		

The palette registers for displaying characters, cursors and underline are initialized as follows.

	Color CRT	Monochrome CRT	
		2-frame V-RAM	1-frame V-RAM
Character display	7: White	3: Standard brightness	1: Standard brightness
Character cursor	6: Yellow	2: Standard blinking	1: Standard brightness
Graphic cursor	6: Yellow	2: Standard blinking	1: Standard brightness
Underline	7: White	3: Standard brightness	1: Standard brightness

4-5 Control Sequences

A control sequence consists of an escape character and a square bracket "ESC [" followed by parameters. Although some of control sequences are not related to display control, all control sequences are explained in this section.

The following table summarizes the formats of control sequences and their functions. In the formats, P followed by a lowercase character (such as Pn or Ps) represents a parameter. When more than one parameters are specified, they are separated by semi-colons (;). A parameter is a string of numbers and preceding zeros are ignored. The end of control sequence is always a command character which indicates the function of the control sequence.

No.	Control sequence	Function
1	ESC [Pn A	Moves the cursor up.
2	ESC [Pn B	Moves the cursor down.
3	ESC [Pn C	Moves the cursor to the right.
4	ESC [Pn D	Moves the cursor to the left.
5	ESC [Pl ; PcH	Moves the cursor to the specified location.
6	ESC [0K	Clears a line from the cursor location to the end of the line.
7	ESC [1K	Clears a line from the beginning of the line to the cursor location.
8	ESC [2K	Clears the line on which the cursor is located.
9	ESC [0J	Clears the area from the cursor location to the end of the scroll area.
10	ESC [1J	Clears the area from the beginning of the scroll area to the cursor location.
11	ESC [2J	Clears the scroll area.
12	ESC [Pf ; Ptr	Specifies the scroll area.
13	ESC [Pn L	Insert lines.
14	ESC [Pn M	Deletes lines.
15	ESC [Pn @	Insert characters.
16	ESC [Pn P	Deletes characters.
17	ESC [6 n	Gets cursor location.
18	ESC [> Pn A	Moves the graphic cursor up.
19	ESC [> Pn B	Moves the graphic cursor down.
20	ESC [> Pn C	Moves the graphic cursor to the right.
21	ESC [> Pn D	Moves the graphic cursor to the left.
22	ESC [> Px ; PyH	Moves the graphic cursor to the specified location.
23	ESC [> 6 n	Gets the graphic cursor location.

No.	Control sequence	Function
24	ESC [> 0 h	Displays the character cursor.
25	ESC [> 1 h	Displays the graphic cursor.
26	ESC [> 2 h	Sets the 25-line mode.
27	ESC [> 3 h	Sets the smooth scroll mode.
28	ESC [> 4 h	Sets the pointing device graphic cursor mode.
29	ESC [> 5 h	Sets the character box size to 16/20.
30	ESC [> 0 ℓ	Makes the character cursor invisible.
31	ESC [> 1 ℓ	Makes the graphic cursor invisible.
32	ESC [> 2 ℓ	Sets the 20-line mode.
33	ESC [> 3 ℓ	Sets the jump scroll mode.
34	ESC [> 4 ℓ	Sets the pointing device character cursor mode.
35	ESC [> 5 ℓ	Sets the character box size to 12/16.
36	ESC [0 m	Resets the character display attribute.
37	ESC [1 m	Sets the high brightness display attribute.
38	ESC [4 m	Sets the underlined display attribute.
39	ESC [5 m	Sets the blinking display attribute.
40	ESC [7 m	Sets the reverse display attribute.
41	ESC [Pn m	Specifies the foreground and background colors.
42	ESC [>0 ; Pn c	Specifies the color of the character cursor.
43	ESC [>1 ; Pn c	Specifies the color of the graphic cursor.
44	ESC [>2 ; Pn c	Specifies the color of the underline.
45	ESC [0 ; Png	Sets the screen bank.
46	ESC [1 ; Pf ; Pb q	Specifies the foreground and background colors.
47	ESC [2 ; Pn ; Pc q	Sets the palette.
48	ESC [>Pf ; Pℓ s	Generates sound.

(1) Moving the cursor up

Format: ESC[PnA

Function: Moves the cursor up the number of lines specified with Pn. Pn is assumed as 1 when it is omitted or is specified as 0. The cursor does not move when it is on the uppermost line.

(2) Moving the cursor down

Format: ESC[PnB

Function: Moves the cursor down the number of lines specified with Pn. Pn is assumed as 1 when it is omitted or is specified as 0. The cursor does not move when it is on the lowermost line.

(3) Moving the cursor to the right

Format: ESC[PnC

Function: Moves the cursor to the right the number of columns specified with Pn. Pn is assumed as 1 when it is omitted or is specified as 0. The cursor moves to the beginning of the next line when it is at the end of the current line (column 80). However, when the cursor is at the end of the lowermost line, it is moves to the beginning of that line and scrolling is not performed.

(4) Moving the cursor to the left

Format: ESC[PnD

Function: Moves the cursor to the left the number of columns specified with Pn. Pn is assumed as 1 when it is omitted or is specified as 0. The cursor moves to the end (column 80) of the preceding line when it is at the beginning of the current line. However, when the cursor is at the beginning of the uppermost line, it is not moved and scrolling is not performed.

(5) Moving the cursor

Format: ESC[Pl,PcH or ESC[Pl;Pcf

Function: Moves the cursor to the location specified with line number Pl and column number Pc. When Pℓ or Pc is out of range, it is automatically set to the nearest location within the range.

Ex) ESC[0;90H → ESC[1;80H

ESC[H → ESC[1;1H

(6) Clear line after cursor

Format: ESC[0K or ESC[K

Function: Clears columns from the cursor location to the end of line with spaces. The cursor does not move.

(7) Clear line before cursor

Format: ESC[1K

Function: Clears columns from the beginning of the current line to the cursor location with spaces. The cursor moves to the beginning of the line.

(8) Clear current line

Format: ESC[2K

Function: Clears the current line with spaces. The cursor moves to the beginning of the line.

(9) Clear lower part of scroll area

Format: ESC[0J or ESC[J

Function: Clears the area from the cursor location to the end of the scroll area with spaces. The cursor does not move.

Note:

The scroll area varies according to the specifications of control sequence ESC[Pf;Ptr.

A	
B	Partial scroll area
C	

That is, when area B is specified as the partial scroll area and the cursor is in that area, area B is cleard to its end; when the cursor is in area A, area A is cleared to its end.

(10) Clear upper part of scroll area

Format: ESC[1J

Function: Clears a part of scroll area from its beginning to the cursor location with spaces. The cursor moves to the beginning of the scroll area. Refer to the note for ESC[0J.

(11) Clear scroll area

Format: ESC[2J

Function: Clears the scroll area with spaces. The cursor moves to the beginning of the scroll area. Refer to the note for ESC[0J.

(12) Scroll area setting

Format: ESC[Pf;Ptr

Function: Sets the scroll area to the area from line Pf to line Pt. Pf and Pt must be as follows.

25-line mode	$1 \leq Pf < Pt \leq 25$
20-line mode	$1 \leq Pf < Pt \leq 20$

When the value of Pf or Pt is out of range, it is assumed as the limit value.

Ex) ESC[0;28r → ESC[1;25r

When the scroll area is not the entire screen, a special scroll method is used. This method is performed at a lower speed than that with the smooth or jump scroll method.

(13) Line insertion

Format: ESC[PnL

Function: Inserts the number of blank lines specified with Pn between the cursor line and the preceding line. The lines after cursor lines are scrolled down. The cursor is moved to the beginning of the cursor line. This control sequence is not performed when the cursor is out of the scroll area. Pn is assumed as 1 when it is omitted or is specified as 0.

(14) Line deletion

Format: ESC[PnM

Function: Deletes the number of lines specified with Pn from the cursor line. The remaining lines are scrolled up. The cursor is moved to the beginning of the cursor line. This function is not performed when the cursor is out of the scroll area. Pn is assumed as 1 when it is omitted or is specified as 0.

(15) Character insertion

Format: ESC[Pn@

Function: Inserts the number of spaces specified with Pn in the cursor location. The following characters are shifted to the right and those exceed the end of line are discarded. Pn is assumed as 1 when it is omitted or is specified as 0.

(16) Character deletion

Format: ESC[PnP

Function: Deletes the number of character specified with Pn from the cursor location. The following characters are shifted to the left and columns at the end of line are filled with spaces. Pn is assumed as 1 when it is omitted or is specified as 0.

(17) Cursor location

Format: ESC[6n

Function: Requests the current location. The format of the return data is as follows. The user must obtain the location through CONIN.

ESC[Pl;PcR where Pl = line number, Pc = column number

(18) Moving the graphic cursor up

Format: ESC[>PnA

Function: Moves the cursor up the number of dots specified with Pn. When the Y coordinate of the cursor location is 0, the cursor does not move. Pn is assumed as 1 when it is omitted or specified as 0.

(19) Moving the graphic cursor down

Format: ESC[>PnB

Function: Moves the cursor down the number of dots specified with Pn. When the Y coordinate of the cursor location is 399, the cursor does not move. Pn is assumed as 1 when it is omitted or is specified as 0.

(20) Moving the graphic cursor to the right

Format: ESC[>PnC

Function: Moves the graphic cursor to the right the number of dots specified with Pn. When the X coordinate of the cursor location is 639, the cursor does not move. Pn is assumed as 1 when it is omitted or is specified as 0.

(21) Moving the graphic cursor to the left

Format: ESC[>PnD

Function: Moves the graphic cursor to the left the number of dots specified with Pn. When the X coordinate of the cursor location is 0, the cursor does not move. Pn is assumed as 1 when it is omitted or is specified as 0.

(22) Moving the graphic cursor

Format: ESC[>Px,PyH or ESC[>Px;Pyf

Function: Moves the graphic cursor to the location specified with coordinates Px and Py. When Px and Py are out of range, they are assumed as the nearest limit values.

Ex) ESC[>700;400H → ESC[>639;399H

(23) Getting the graphic cursor position

Format: ESC[>6n

Function: Requests the current graphic cursor location. The format of the return data is as follows. The user must obtain the location through CONIN.

ESC[>Px;Py R where Px = x coordinate, Py = y coordinate

(24) Character cursor display

Format: ESC[>0h

Function: Makes the character cursor visible (ON).

(25) Graphic cursor display

Format: ESC[>1h

Function: Makes the graphic cursor visible (ON).

(26) 25-line mode specification

Format: ESC[>2h

Function: Sets the screen to the 25-line mode, clears the entire screen and sets the scroll area to the entire screen. The cursor is moved to the home position.

(27) Smooth scroll mode specification

Format: ESC[>3h

Function: Sets the scroll mode to the smooth scroll mode. This control sequence is not effective if it is entered when a part of screen is set to the scroll area but will become effective when the scroll area is set to the entire screen.

(28) Setting the pointing device graphic cursor mode

Format: ESC[>4h

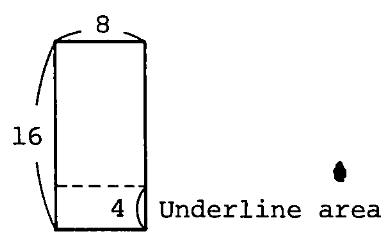
Function: Sets the pointing device graphic cursor mode in which the pointing device is used to move the graphic cursor and makes the graphic cursor visible.

(29) Setting the character box size to 16/20

Format: ESC[>5h

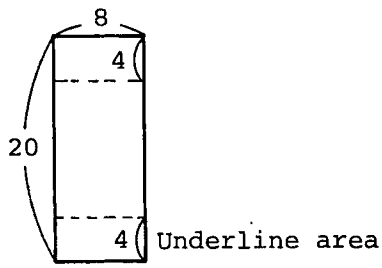
Function: Sets the size of character box as shown below.

25-line mode



Character box

20-line mode



Character box

When a character which is not underlined is displayed in this mode in a location on which an underline is already displayed, the underline is erased because the display area includes the underline area.

(Note for h-type sequences)

All the above h-type control sequences (ESC[>0h - ESC[>5h) can be executed at a time by the following format.

ESC[>h

More than one h-type control sequences can be included in a format as shown below.

ESC[>0;1;4h

(30) Erasing the character cursor

Format: ESC[>0ℓ

Function: Makes the character cursor invisible. As a result, the display processing speed is increased.

(31) Erasing the graphic cursor

Format: ESC[>1ℓ

Function: Makes the graphic cursor invisible. As a result, the display processing speed is increased.

(32) 20-line mode specification

Format: ESC[>2ℓ

Function: Sets the screen to the 20-line mode, clears the entire screen and sets the entire screen to the scroll area. The cursor is moved to the home position.

(33) Jump scroll mode specification

Format: ESC[>3l

Function: Sets the scroll method to the jump mode. This control sequence is not effective if it is entered when a part of screen is set to the scroll area but will become effective when the entire screen is set to the scroll area.

(34) Setting the pointing device character cursor mode

Format: ESC[>4l

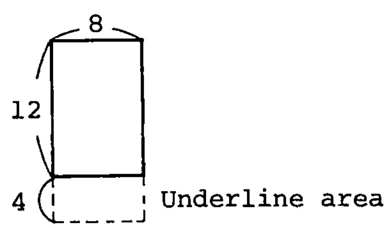
Function: Sets the pointing device character cursor mode in which the pointing device is used to move the character cursor, and makes the graphic cursor invisible.

(35) Setting the character box size to 12/16

Format: ESC[>5l

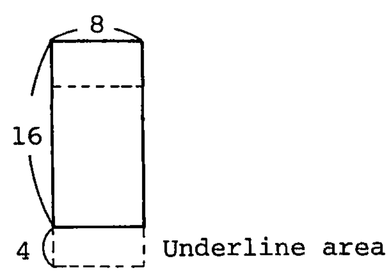
Function: Sets the size of character box as shown below.

25-line mode



Character box

20-line mode



Character box

When a character which is not underlined is displayed in this mode in a location on which an underline is displayed, the underline is not erased because the underline area is not set as the display area. If semigraphic characters are displayed in this mode, the lower part of each character which occupies the lower 4 lines of the 8 x 16 dot matrix is not displayed.

(Note for ℓ-type sequence)

All the above ℓ-type control sequences (ESC[>0ℓ - ESC[>5ℓ) can be executed at a time by the following format.

ESC[>ℓ

More than one control sequence can be specified in a format as shown below.

ESC[>1;2;3ℓ

(36) Resetting character display attributes

Format: ESC[0m

Function: Resets the underline and reverse attributes for character display. With the monochrome 2-frame model, this control sequence sets the foreground palette number to 3 (default value: standard brightness).

(37) Setting the high brightness attribute

Format: ESC[1m

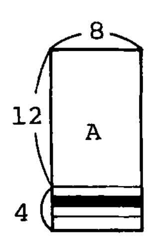
Function: Displays characters at higher brightness, that is sets the foreground palette number to 1 (default value: high brightness). This is effective only with the monochrome 2-frame model. This sequence does not operate properly if the palette setting has been changed.

(38) Setting the underline attribute

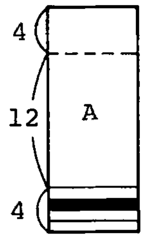
Format: ESC[4m

Function: Displays characters with underlines.

25-line mode



20-line mode



(39) Setting the blinking attribute

Format: ESC[5m

Function: Displays characters with blinking attributes. This sequence is effective with the monochrome 2-frame model. The foreground palette number is set to 2 (default value: blinking). This sequence does not operate properly if the palette setting has been changed.

(40) Setting the reverse display attribute

Format: ESC[7m

Function: Reverses the foreground and background colors or attributes to display characters.

(41) Specifying the foreground and background colors

Format: ESC[Pnm

Function: Specifies the palette numbers for the foreground and background colors when characters are displayed. This is effective for the color CRT model only.

Value of Pn		Specified palette	
Foreground	Background	Number	Default color
30	40	0	Black
31	41	4	Red
32	42	2	Green
33	43	6	Yellow
34	44	1	Blue
35	45	5	Magenta
36	46	3	Cyan
37	47	7	White

This control sequence has a similar function as the palette setting sequence (ESC[1;Pn;Pcq) excepting that this uses the default colors. Use the palette setting sequence when you what to change the palette setting. The desired result cannot be obtained with this sequence if the palette setting has been changed.

(Note for the m-type control sequences)

More than one m-type control sequence (ESC[0m - ESC[47m) can be specified in a format as shown below.

ESC[4;31;41m

Note that parameters which can be specified very according to the model as shown below.

Parameter	Function	Color CRT model	Monochrome 2-frame model	Monochrome 1-frame model
0	Resetting attributes	o	o	o
1	High brightness		o	
4	Underline	o	o	o
5	Blinking		o	
7	Reverse display	o	o	o
30 ∿ 37, 40 ∿ 47	Foreground and background colors	o		

(42) Specifying the character cursor color

Format: ESC[>0;Pnc

Function: Specifies the palette number for the attribute or color of the character cursor. Pn must be within the range shown below.

Color CRT model	$0 \le Pn \le 7$
Monochrome 2-frame model	$0 \le Pn \le 3$
Monochrome 1-frame model	$0 \le Pn \le 1$

(43) Specifying the graphic cursor color

Format: ESC[>1;Pnc

Function: Specifies the palette number for the attribute or color of the graphic cursor. Pn must be within the range shown below.

Color CRT model	$0 \le Pn \le 7$
Monochrome 2-frame model	$0 \le Pn \le 3$
Monochrome 1-frame model	$0 \le Pn \le 1$

(44) Specifying the underline color

Format: ESC[2;Pnc

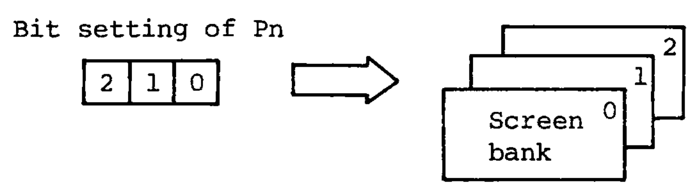
Function: Specifies the palette number for the attribute and color of the underline. Pn must be within the range shown below.

Color CRT mode	$0 \le Pn \le 7$
Monochrome 2-frame model	$0 \le Pn \le 3$
Monochrome 1-frame model	$0 \le Pn \le 1$

(45) Specifying the screen bank

Format: ESC[0;Pnq

Function: Specifies the effective screen bank when data is output to V-RAM.



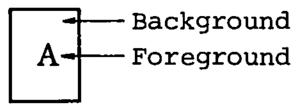
The lower 3 bits of Pn correspond to screen banks 1 to 3 as shown above. Therefore, when Pn is set to 5 (101), data is not output to screen bank 1. This control sequence is effective when each screen bank is independently controlled. Pn must be within the range shown below.

Color CRT model	$0 \le Pn \le 7$
Monochrome 2-frame model	$0 \le Pn \le 3$
Monochrome 1-frame model	$0 \le Pn \le 1$

(46) Specifying the foreground and background colors

Format: ESC[1;Pf;Pbq

Function: Specifies the palette numbers for the attributes or colors of characters or graphic patterns. Pf specifies the palette for foreground and Pb specifies the palette for background.



Character box

The background color or attribute is effective when characters are displayed or graphic patterns are painted. Pf and Pb must be within the range shown below.

Color CRT mode	$0 \le Pf$ or $Pb \le 7$
Monochrome 2-frame model	$0 \le Pf$ or $Pb \le 3$
Monochrome 1-frame model	$0 \le Pf$ or $Pb \le 1$

(47) Setting the palette

Format: ESC[2;Pn;Pcq

Function: Sets the palette specified with Pn to the color or attribute specified with Pc. More than one Pc can be specified to specify the colors or attributes to palettes Pn, Pn+1, Pn+2 ···.

Ex) ESC[2;0;1;2;3;4;5;6;7;8q

The above example sets attributes 1 to 8 to palettes 0 to 7.

PC for monochrome 1-frame mode;

Value	Attribute
0	Non-display
1	Standard brightness
2 - 28	Standard brightness

Pc for monochrome 2-frame model

Value	Attribute			Remarks
	Blinking	High brightness	Standard brightness	
0	0	0	0	Non-display (black)
1	0	0	1	Standard brightness
2	0	1	0	High brightness
3 ∿ 26	0	0	1	Standard brightness
27	1	0	1	Standard blinking
28	1	1	0	High brightness blinking

Pc for color CRT model

Value	Color						Remarks
	r	R	g	G	b	B	
0	0	0	0	0	0	0	Black (non-display)
1	0	0	0	0	0	1	Blue
2	0	0	0	0	1	1	
3	0	0	0	1	0	0	Green
4	0	0	0	1	0	1	Cyan
5	0	0	0	1	1	1	
6	0	0	1	1	0	0	
7	0	0	1	1	0	1	
8	0	0	1	1	1	1	
9	0	1	0	0	0	0	Red
10	0	1	0	0	0	1	Yellow
11	0	1	0	0	1	1	
12	0	1	0	1	0	0	Magenta
13	0	1	0	1	0	1	White
14	0	1	0	1	1	1	
15	0	1	1	1	0	0	
16	0	1	1	1	0	1	
17	0	1	1	1	1	1	
18	1	1	0	0	0	0	
19	1	1	0	0	0	1	
20	1	1	0	0	1	1	
21	1	1	0	1	0	0	
22	1	1	0	1	0	1	
23	1	1	0	1	1	1	
24	1	1	1	1	0	0	
25	1	1	1	1	0	1	
26	1	1	1	1	1	1	
27,28	1	1	1	1	1	1	

R: Red r: Half brightness red

G: Green g: Half brightness green

B: Blue b: Half brightness blue

(48) Generating sound

Format: ESC[>Pf;Pℓs

Function: Generates the sound of the frequency specified with Pf for the period of time specified with Pℓ. More than one parameter can be specified. Pf is a number from 0 to 60 which represents a note within 5 octave range. When 0 is specified no sound is generated.

Value of Pf

Tone	Octave				
	0	1	2	3	4
C	1	13	25	37	49
C#	2	14	26	38	50
D	3	15	27	39	51
D#	4	16	28	40	52
E	5	17	29	41	53
F	6	18	30	42	54
F#	7	19	31	43	55
G	8	20	32	44	56
G#	9	21	33	45	57
A	10	22	34	46	58
A#	11	23	35	47	59
B	12	24	36	48	60

A1(22)=440Hz

Pℓ must be within the range from 0 to 255. The unit period is about 0.016 seconds.

Pℓ	Time (second)
1	0.016
10	0.16
50	0.8
100	1.6
150	2.4
200	3.2
255	4.08

Multiple parameters can be specified but the total number of parameters (include;) must be 30 or less.

Ex) ESC[>Pf1;pℓ1;Pf2;Pℓ2;.....;Pfn;pℓns

4-6 Graphic Display Functions

4-6-1 Outline

There are two methods to use the graphic display functions. One uses special escape sequences entered through the BIOS CONOUT call.

The other calls the AS-100 function calls from user assembler programs. In this method, parameters are directly set in registers. The processing speed of this method is faster than that of the former method.

<u>Coordinate system</u>

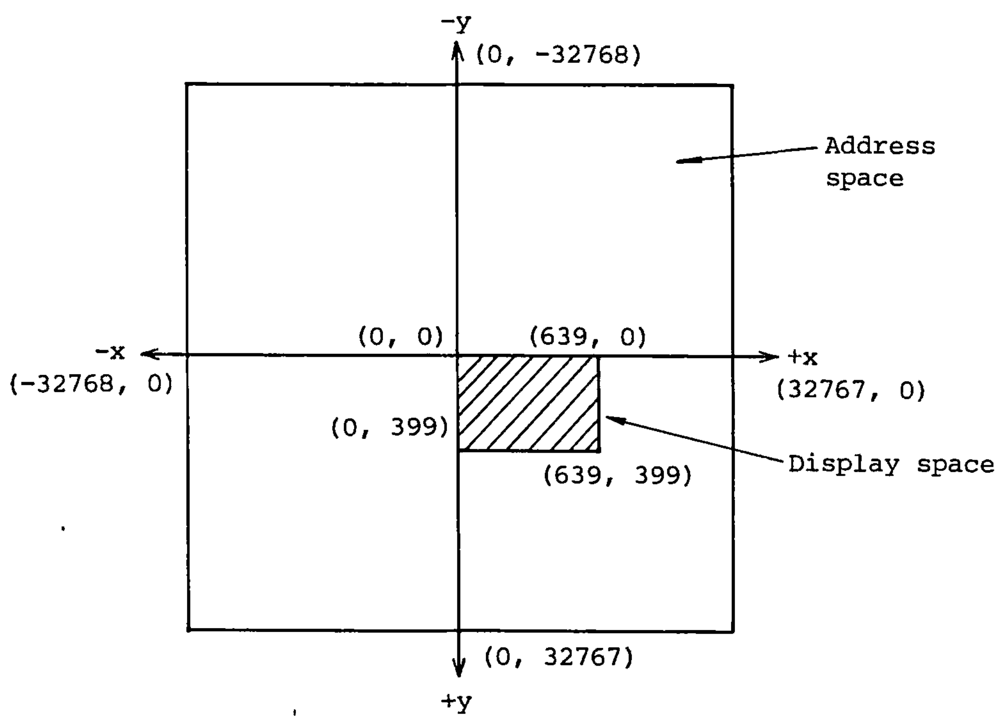
The X and Y coordinates of the address space for graphic display range from -32768 to +32767. However, the area which is actually displayed on the CRT screen is limited as follows.

$$0 \leq x \leq 639$$

$$0 \leq y \leq 399$$

Values which are out of the above ranges are ignored by the boundary check function but do not cause errors except the following cases.

1. When specified in V-RAM transfer commands G and H.
2. When specified in paint commands P and Q.



Current pointer (CP)

The coordinates which are used as the reference point when drawing dot patterns or circles are indicated by the current pointer. The initial value of the current pointer is (0, 0) and it can be varied if necessary. The current pointer can be varied only by graphic commands and initialization sequence (ESCc).

Color and attribute

The color or attribute of dot, pattern or character displayed by a graphic command is determined by the palette currently selected (foreground or background palette). Thus, the color of a mesh pattern or halftone pattern may be a composite color of the foreground and background colors.

4-6-2 Graphic display through CONOUT

Graphic sequence

Graphic commands and parameters are entered as follows to execute graphic functions through CONOUT.

```
ESC P ............ ESC /
  |        |          |
  |        |          └──── Graphic mode terminated
  |        └─────────────── Command and parameter group
  └──────────────────────── Graphic mode initiated
```

ESC P (1B50H) switches CONOUT to the graphic mode. The graphic mode continues until ESC / is entered. In the graphic mode, CONOUT assumes that all the codes entered comply the rule described below and executed them as graphic commands and parameters. The following items are initialized when the graphic mode is initiated by ESC P.

Line	Solid line
Paint pattern	Painting all over
Character enlargement	1 (not enlarged)
Character inclination	0 (not inclined)

Commands and parameters

Each graphic command consists of a command character followed by parameters. Graphic commands are executed when ESC / is entered.

Command character;

Command characters are listed below. Lowercase characters are treated as uppercase characters (e.g., M=m). Characters other than the listed are ignored.

Parameters;

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ", ', + and - are valid. Other characters are ignored. ' and " are used as delimiters for text string in which all characters can be used.

Graphic command list

No.	Command	Parameter	Function
1	M	x, y	Moving current point
2	D	None	Drawing dot
3	L	x, y	Line drawing
4	R	x, y [, p]	Rectangle drawing (paint)
5	C	r [, p]	Circle drawing (paint)
6	E	rx, ry, θ [, p]	Ellipse drawing (paint)
7	F	r, θ1, θ2 [, p]	Fan shape drawing (paint)
8	A	r, θ1, θ2	Arc drawing
9	S	p	Mark drawing
10	Q	None	Painting an area
11	P	p	Painting the area within a closed line
12	T	"string" or 'string'	Text (enlargement, inclination)
13	W	p1, p2	Specifying color (attribute)
14	X	p	Specifying line type
15	Y	p	Specifying paint pattern
16	Z	p1, p2, p3	Specifying character size and inclination
17	G	seg, off, x, y	Reading V-RAM
18	H	seg, off, x, y	Writing V-RAM

Lowercase character

x, y: Coordinates

r: Radius

θ: Angle

p: Numeric string defined for each function

seg: Segment address (decimal number)

off: Offset address (decimal number)

(1) M - Moving current point

Format: Mx, y

Parameters: x X-coordinate

y Y-coordinate

Function: Moves the current point to the specified location.

(2) D - Drawing dot

Format: D

Parameter: None

Function: Displays a dot at the current point. The color or attribute is specified by the foreground palette.

(3) L - Drawing line

Format: Lx, y

Parameter: x and y coordinates

Function:

Draws a line from the current point to the specified location. The current point is moved to the specified location after drawing the line. The line type depends on the line type mode (command X). The point of termination is always set. The color or attribute is specified by the foreground palette.

(4) R - Drawing rectangle

Format: Rx, y [,p]

Parameters: x, y coordinates

p paint attribute

0 not painted

1 painted

(Default value is 0.)

Function: Draws a rectangle with the diagonal line connecting the current point and the specified location set as the diagonal. The line type depends on the line type mode when the rectangle is not painted; the profile is set to the solid line according to the paint type when it is painted. The color or attribute is specified by the foreground palette.

(5) C - Drawing circle

Format: Cr[,p]

Parameters: r radius (positive value only)

p paint attribute

0 not painted

1 painted

(Default value is 0.)

Function: Draws a circle with the special radius with the current point set as the center. The line type depends on the line type mode when the circle is not painted; the profile is set to the solid line according to the paint pattern when it is painted. The color or attribute is specified by the foreground palette.

(6) E - Drawing ellipse

Format: Erx,ry,θ[,p]

Parameters: rx radius in the direction of x' axis

ry radius in the direction of y' axis

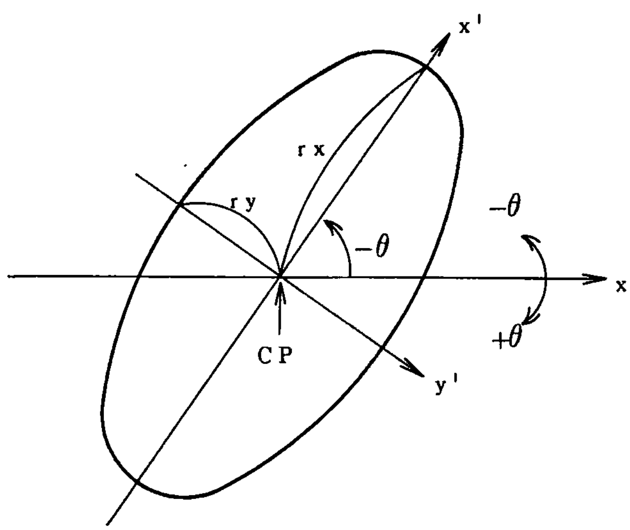
θ angle of x' axis to x axis

p paint attribute

0 not painted

1 painted

(Default value is 0.)

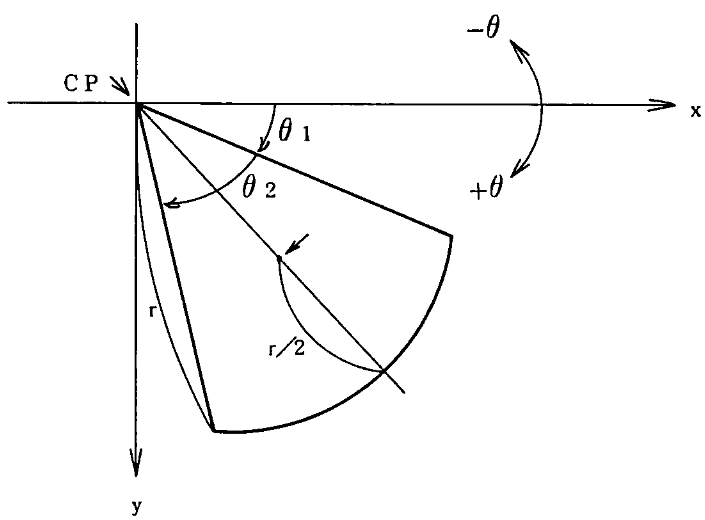


Function: Draws an ellipse with the current point as the center which has radiuses of the specified lengths and is inclined by θ degrees against the x axis. The current point is not moved. The line type depends on the line type mode when the ellipse is not painted; the profile is set to the solid line when it is painted. If another pattern or closed line exists in the ellipse, all the entire area within the ellipse may not be painted. (Painting start at CP.) The color or attribute is specified by the foreground palette.

(7) F - Drawing fan shape

Format: $Fr,\theta_1,\theta_2[,p]$

Parameters:
- r radius
- θ_1 start angle
- θ_2 interior angle
- p paint attribute
  - 0 not painted
  - 1 painted
  - (Default value is 0.)



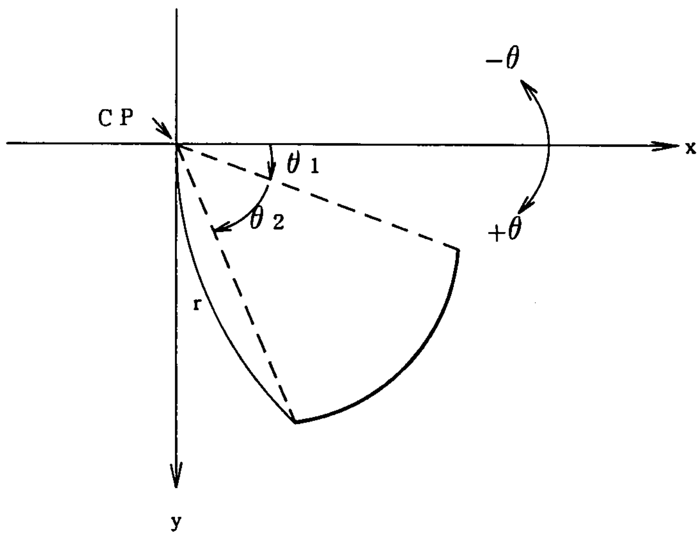
Function: Draws a fan shape with a starting angle of θ_1, an interior angle of θ_2 and a radius of r with the current point as the center. The current point is not moved. The line type depends on the line type mode when the fan shape is not painted, the profile is set to the solid line when it is painted. Painting starts at coordinates $\theta 2/2$, r/2. Therefore, if the painting start point is not in the display space, painting is not performed. If another pattern or closed line is within the shape, painting may not be performed correctly. The color or attribute is specified with the foreground palette.

(8) A - Drawing arc

Format: Ar, θ_1,θ_2

Parameters:

r		radius
	θ_1	starting angle
	θ_2	interior angle



Function: Draws an arc with the current point set as the center which has a starting angle of θ_1, an interior angle of θ_2 and a radius of r. The current point is not moved. The line type depends on the line type mode. The color or attribute is specified with the foreground palette.

(9) S - Drawing mark

Format: Sp

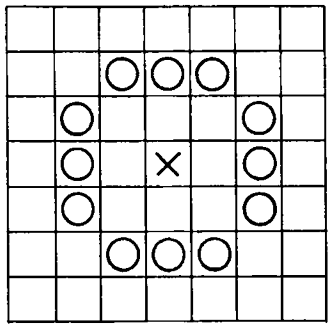
Parameters: p mark type (0 to 6)

0	○
1	●
2	□
3	■
4	×
5	△
6	▲

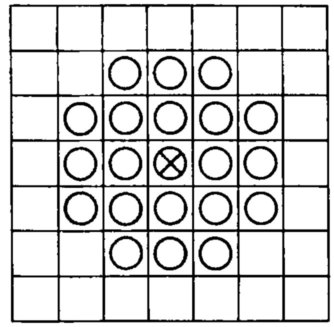
Function: Displays a mark specified with the parameter at the current point. The current point is not moved. The color or attribute is specified with the foreground palette.

Mark patterns (×: current point)

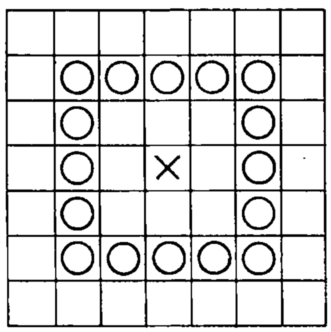
Type 0



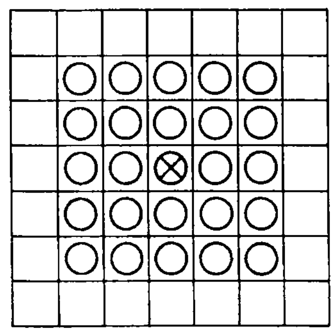
Type 1



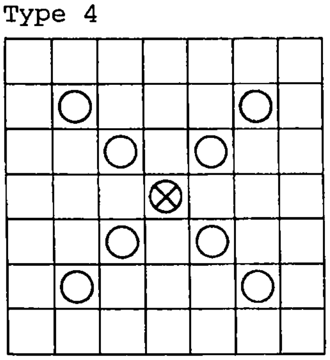
Type 2



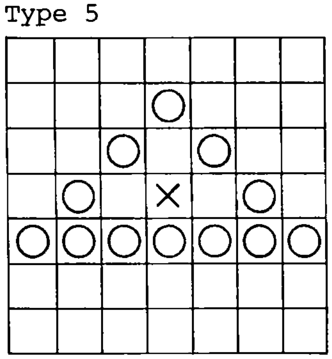
Type 3



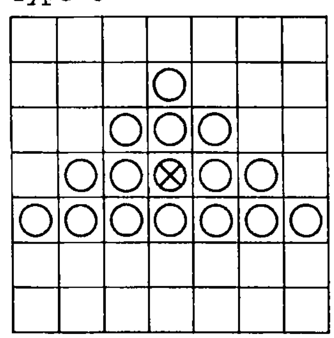
Type 4



Type 5



Type 6



(10) Q - Painting an area

Format: Q

Parameter: none

Function: Paint a continuous area, which includes the current point and has the same color or attribute as the current point, with the color or attribute specified with the foreground palette. Painting depends on the paint pattern. For patterns other than overall painting, the background c lor or attribute may be used. The current point is not moved.

(11) P - Painting the area enclosed with a closed line

Format: Pp

Parameter: p Palette number for boundary color (0 to 7)

Function: Paints the screen starting at the current point until a location which has the color or attribute specified with the parameter. Therefore, if there is no location which has the specified color or attribute in the screen, the entire screen is painted with the foreground color. Patterns which has the specified color or attribute and areas enclosed with lines of the specified color or attribute are not painted. Painting depends on the paint pattern. For patterns other than overall painting, the background color or attribute may be used. The current point is not moved.

(Note for painting)

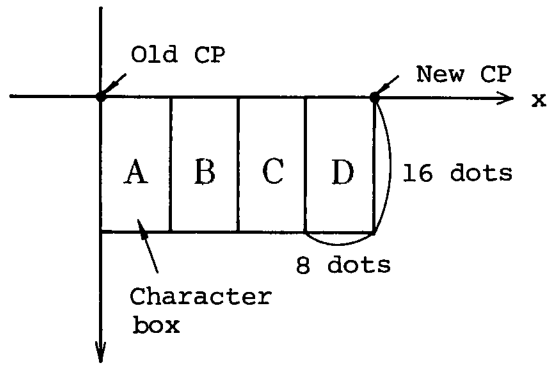
When a complex pattern is painted with a Q or P command (e.g. when a halftone pattern is painted by the mesh pattern), the work area in the graphic system may be insufficient. In such a case, painting is suspended. With the color model, if all screen banks (V-RAM) are not selected, boundary conditions become ambiguous, resulting in erroneous operation.

(12) T - Text

Format: T"text" or T'text'

Parameter: "text"

Function: Displays the specified text assuming that the currnt point is the upper left corner of the first character box. The dots which form character patterns are plotted with the color or attribute specified with the foreground palette. The color or attribute of the remaining area in each character box does not change. The size and inclination of each character are specified with the character type setting command (Z). The current point is moved to the upper right corner of the last character box.



Enlargement ratio 1, inclination 0

(13) W - Specifying color

Format: Wp1,p2

Parameter: p1 foreground palette number 0 to 7

p2 background palette number 0 to 7

Function: Assigns the foreground and background palette numbers. The palette numbers specified with this command is also effective after the graphic mode has been terminated. The background color or attribute is used only when painting is performed. The initial setting of each palette is as follows.

Color CRT model

Palette number	Color
0	Black (non-display)
1	Blue
2	Green
3	Cyan
4	Red
5	Magenta
6	Yellow
7	White

Monochrome 2-frame model

Palette number	Attribute
0	Non-display
1	High brightness
2	Blinking
3	Standard brightness

Monochrome 1-frame model

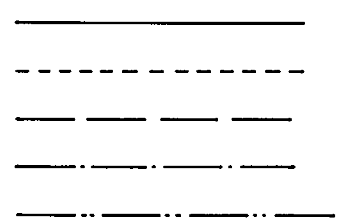
Palette number	Attribute
0	Non-display
1	Standard brightness

(14) X - Specifying line type

Format: X[p]

Parameter: p line type 0 to 4 (Default is 0.)

0: Solid line
1: Short dotted line
2: Long dotted line
3: Chain line
4: Two dots chain line



Function: Specifies the line type. The type specified is effective until the graphic mode is terminated. The initial value and default value are 0 (solid line). The line type becomes 0 when the graphic mode is terminated.

Line patterns

0 solid line

O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O

1 short dotted line

O	O			O	O			O	O			O	O		

2 long dotted line

O	O	O	O					O	O	O	O				

3 chain line

4 two dots chain line

(15) Y - Specifying paint pattern

Format: Y[p]

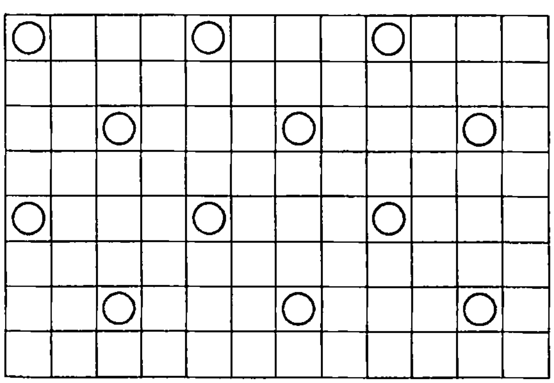
Parameter: p paint pattern 0 to 8 (Default is 0.)

0: Overall painting
1: Halftone
2: Oblique lines (right up)
3: Oblique lines (left up)
4: Vertical lines
5: Horizontal lines
6: Slanted mesh
7: Mesh
8: Deep halftone

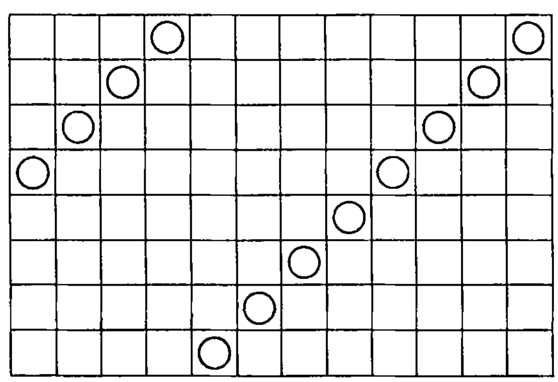
Function: Specifies the paint pattern. The pattern specified is effective until the graphic mode is terminated. The initial value and default value are 0 (overall painting). The paint pattern is initialized whenever the graphic mode is initiated (with ESC P).

Paint pattern

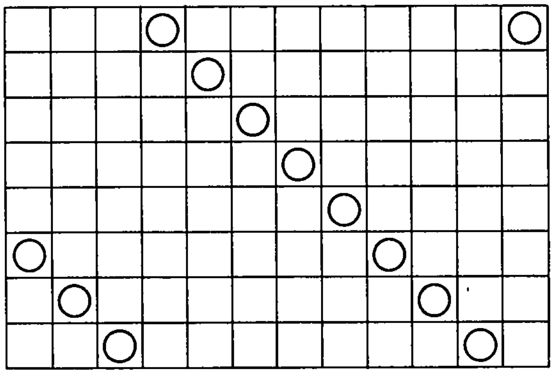
1. Halftone



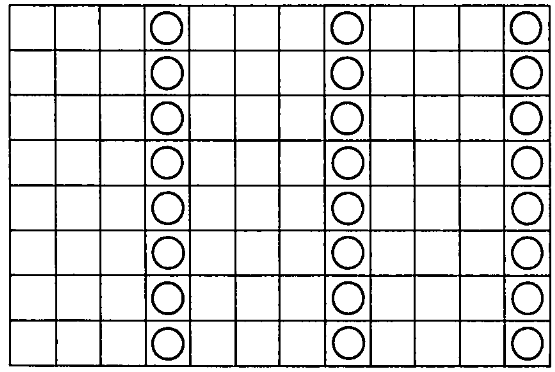
2. Oblique lines (right up)



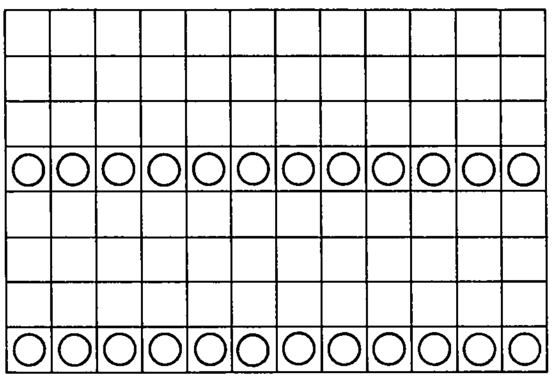
3. Oblique lines (left up)



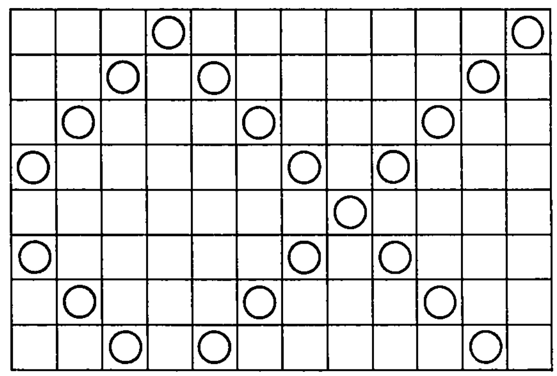
4. Vertical lines



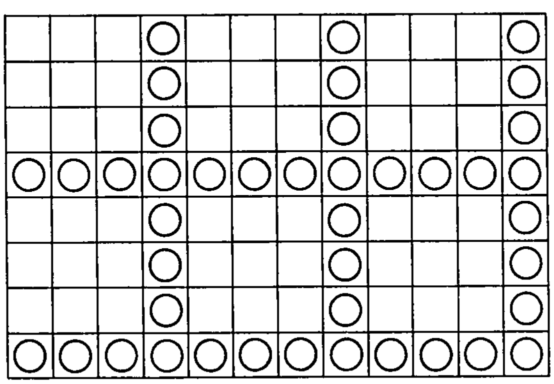
5. Horizontale lines



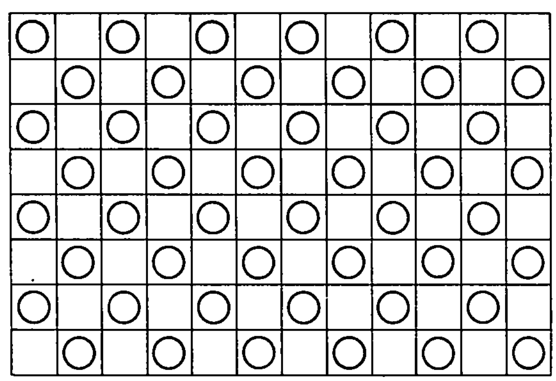
6. Slanted mesh



7. Mesh



8. Deep halftone



(16) Z - Specifying inclination of characters

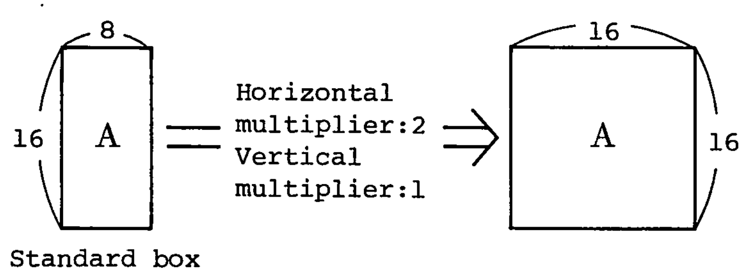
Format: Zp1,p2,p3

Parameters: p1 horizontale multiplier 1 to 16

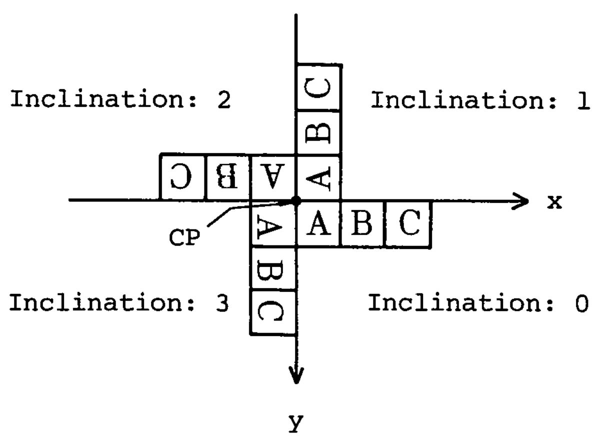
p2 vertical multiplier 1 to 16

p3 inclination 0 to 3

Function: Specifies the multipliers which determine the size of character and the inclination of character. The multipliers and inclination are effective until the graphic mode is terminated. The initial values of the multipliers are 1 and that of inclination is 0. They are initialized whenever the graphic mode is initiated with ESC P. Enlargement of character is performed for the 8 x 16 dot character box.



Inclination is performed in 90-degree units with the current point set to the center.



(17) G - Reading V-RAM

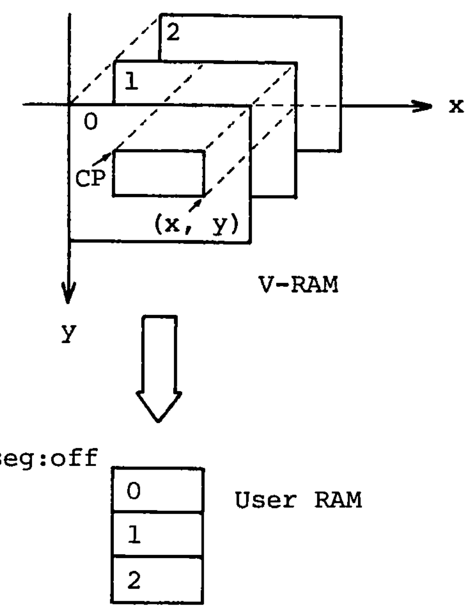
Format: Gseg,off,x,y

Parameters: seg destination segment address

off destination offset address

x, y coordinates

Function: Transfers the bit map of the rectangular with the diagonal line connecting the current point and the specified location to the specified memory area starting at the specified address. Data transfer is made in bit units. If the last byte is not filled with the bit transferred, the remaining bits become 0. When two or more screen frames are used, frame 1 is transferred after frame 0 has been transferred, and so on. If the current point and the specified location are not in the display space, data transfer is not performed.



Note: When the transfer starting address (CP) of the V-RAM bit map is at a byte boundary, data transfer is performed in word units (16 bit units), resulting in high speed processing. This also applies to the H command below.

(18) H - Writing V-RAM

Format: Hseg,off,x,y

Parameters: seg source segment address

off source offset address

x, y coordinate

Function: Transfers bit image from the memory area starting at address seg:off to the rectangular display area with the diagonal line connecting the current point and the specified location. When two or more frames are used, data is first transferred to frame 0, then frame 1 and frame 2 in succession. If the current point and the specified location are not in the display area, data transfer is not performed. This command has an opposite function of the G command.

4-6-3 AS-100 function call

The AS-100 function calls are basically the same as the functions called through CONOUT. Each function is executed by issuing INT 241 (F1h) with a function code and parameters set in registers.

A function code is set in the CL register and parameters are set in the AX, BX, DX, BP, ES and/or DI registers. The contents of all registers are not changed upon execution of the called function. The text display function is different from the corresponding function called through CONOUT: it cannot display more than one character. Other functions are the same as those called through CONOUT. The table below lists the function codes and parameters. For details, refer to the corresponding functions called through CONOUT.

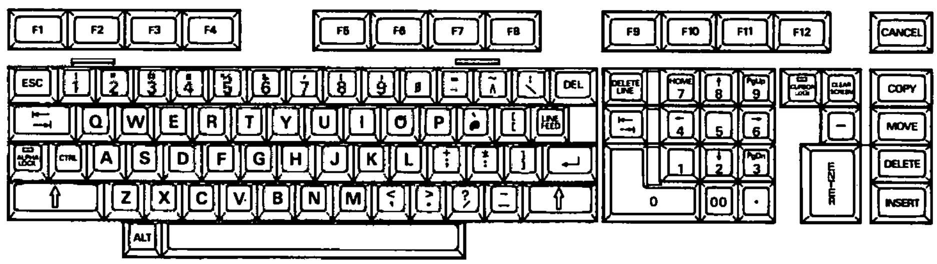
Function codes and parameters

Function code	Parameters	Function
0	AX: x coordinate BX: y coordinate	Moving current point
1	None	Displaying dot
2	AX: x coordinate BX: y coordinate	Drawing line

Function code	Parameters	Function
3	AX: x coordinate BX: y coordinate BP: paint attribute	Drawing rectangle
4	AX: radius BP: paint attribute	Drawing circle
5	AX: radius BX: starting angle DX: interior angle BP: paint attribute	Drawing fan shape
6	AX: radius BX: starting angle DX: interior angle	Drawing arc
7	AL: mark type	Drawing mark
8	AL: boundary color palette number	Painting area within closed line
9	None	Painting an area
10	AX: Character code AH=Null AL 8 bit AH≠Null AX 16 bit	Displaying one character
11	AX: x coordinate BX: y coordinate DI: offset address ES: segment address	Reading V-RAM
12	AX: x coordinate BX: y coordinate DI: offset address ES: segment address	Writing V-RAM
13	AX: major radius BX: minor radius DX: inclination BP: paint attribute	Drawing ellipse
14	AH: foreground palette number AL: background palette number	Specifying color (attribute)
15	AL: line type	Specifying paint type
16	AL: paint pattern	Specifying paint pattern
17	AH: vertical multiplier AL: horizontal multiplier BL: inclination	Specifying character size and inclination

5-1 Layout

The AS-100 keyboard keys are grouped into ASCII keys, ten numeric keys, function keys, and special keys as shown in the figure below. It allows attachment of the optional A-1100 pointing device.



Under AS-100 CP/M-86, all key code is passed to the system via the BIOS CONIN routine. This chapter describes the keyboard codes that are handled by the CONIN routine.

5-2 ASCII Keys

ASCII keys generate different codes when pressed simultaneously with the CTRL, ALPHA-LOCK, or SHIFT key. Pressing these keys while holding down the ALT key generates no code. The CURSOR LOCK key does not affect the ASCII keys. All ASCII keys have the auto repeat feature.

The table below lists the character codes produced by the ASCII keys along with various mode control keys. In the table, blank columns indicate that no code is generated when the pertinent key is pressed and numbers enclosed in parentheses denote the hexadecimal representations of the codes. The other columns contain the character representations of the codes.

KEY	CTRL ON	CTRL OFF ALPHA LOCK Shift ON	CTRL OFF ALPHA LOCK Shift OFF	CTRL OFF ALPHA LOCK OFF Shift ON	CTRL OFF ALPHA LOCK OFF Shift OFF
@	(00)	▼	@	▼	@
A	(01)	A	A	A	a
B	(02)	B	B	B	b
C	(03)	C	C	C	c
D	(04)	D	D	D	d
E	(05)	E	E	E	e
F	(06)	F	F	F	f
G	(07)	G	G	G	g
H	(08)	H	H	H	h
I	(09)	I	I	I	i
J	(0A)	J	J	J	j
K	(0B)	K	K	K	k
L	(0C)	L	L	L	l
M	(0D)	M	M	M	m
N	(0E)	N	N	N	n
O	(0F)	O	O	O	o
P	(10)	P	P	P	p
O	(11)	Q	Q	Q	q
R	(12)	R	R	R	r
S	(13)	S	S	S	s
T	(14)	T	T	T	t
U	(15)	U	U	U	u
V	(16)	V	V	V	v
W	(17)	W	W	W	w

KEY	CTRL ON	CTRL OFF ALPHA LOCK Shift ON	CTRL OFF ALPHA LOCK Shift OFF	CTRL OFF ALPHA LOCK OFF Shift ON	CTRL OFF ALPHA LOCK OFF Shift OFF
X	(18)	X	X	X	x
Y	(19)	Y	Y	Y	y
Z	(1A)	Z	Z	Z	z
[	(1B)	{	[	{	[
	(1C)	¦	\	¦	\
]	(1D)	}	]	}	]
^	(1E)	^		^	
_	(1F)	_	_	_	_
0			0		0
1		!	1	!	1
2		"	2	"	2
3		#	3	#	3
4		$	4	$	4
5		%	5	%	5
6		&	6	&	6
7		▼	7	▼	7
8		(	8	(	8
9		)	9	)	9
:		*	:	*	:
;		+	;	+	;
,		<	,	<	,
-		=	-	=	-
.		>	.	>	.
/		?	/	?	/

5-3 Ten Numeric Keys

The ten numeric key group consists of digit keys 0-9, 00 key, and decimal point (.) and minus (-) keys. You can use some of these keys to position the cursor on the screen by changing its mode with the CURSOR-LOCK key. These keys are not affected by the ALPHA-LOCK or SHIFT key.

They generate no code when pressed simultaneously with the CTRL key. All keys have the auto repeat feature. The table below presents the codes that they product with or without the CURSOR-LOCK key pressed. Blank column indicates that no code is generated.

Key	CURSOR LOCK	
	OFF	ON
0	0	ESC [N
1	1	ESC [I
2	2	ESC [B (↓)
3	3	ESC [F (PgDn)
4	4	ESC [D (←)
5	5	ESC [G
6	6	ESC [C (→)
7	7	ESC [H (HOME)
8	8	ESC [A (↑)
9	9	ESC [E (PgUp)
00	00	
.	.	
-	-	

When the CURSOR-LOCK key is ON, the 0 through 9 numeric keys generate 3-character escape code sequences which are used to control the CRT display (via CONOUT). Since these codes are not automatically echoed back to CONOUT, however, you cannot control the cursor simply pressing these keys. See CHAPTER 4 for the CONOUT functions.

When used together with the ALT key, the ten numeric keys allow you to enter character codes with decimal notation. When you enter a decimal number using the 0 through 9 keys while holding down the ALT key, the corresponding character code is generated when the ALT key is released. For example, if you enter "160" with the ALT key held down, the corresponding character code A0h will be generated when you release the ALT key. Pressing the 00 key while holding down the ALT key cancels the previous decimal number input.

5-4 Function Keys

The keyboard has twelve function keys named F1 through F12. These keys are not affected by the ALPHA-LOCK or CURSOR-LOCK keys. They generate different codes depending on whether or not they are pressed simultaneously with the SHIFT key. When pressed simultaneously with the CTRL and ALT keys, the function keys generate no code. They do not have the auto repeat function. The table below lists the codes generated by the function keys.

Key	SHIFT key state	
	Shift OFF	Shift ON
F1	ESC O	ESC o
F2	ESC P	ESC p
F3	ESC Q	ESC q
F4	ESC R	ESC r
F5	ESC S	ESC s
F6	ESC T	ESC t
F7	ESC U	ESC u
F8	ESC V	ESC v
F9	ESC W	ESC w
F10	ESC X	ESC x
F11	ESC Y	ESC y
F12	ESC Z	ESC z

5-5 Special Keys

The keys on the AS-100 keyboard other than the ASCII keys, ten numeric keys, and function keys [F1-F12] are called special keys. The special keys are not affected by whether the ALPHA-LOCK or CURSOR-LOCK key is on or not; however, some special keys generate different codes when pressed simultaneously with the SHIFT key. Their codes are suppressed when they are pressed simultaneously with either CTRL or ALT key. The space and DEL keys have the auto repeat function. The table below lists the codes generated by the special keys. Numbers enclosed in parentheses are hexadecimal representations of the codes.

Key	SHIFT key state		Repeat feature
	Shift OFF	Shift ON	
⇤ ⇥	(09)	ESC 7	No
(Space)	(20)	ESC 7	Yes
ESC	(1B)	ESC 7	No
DEL	(7F)	ESC 7	Yes
LINE FEED	(0A)	ESC 7	No
↵	(0D)	ESC 7	No
ENTER	(0D)	ESC 7	No
CLEAR SCREEN	ESC [2J	ESC 7	No
DELETE LINE	(18)	ESC 7	No
CANCEL	(03)	ESC 7	No
COPY	ESC 3	ESC 7	No
MOVE	ESC 4	ESC 7	No
DELETE	ESC 5	ESC 7	No
INSERT	ESC 6	ESC 7	No

5-6 Pointing Device

The pointing device is used to move the cursor at any given position on the screen. There are two types of cursor: the character cursor and the graphic cursor. The pointing device can control only one cursor type at a time. Which cursor the pointing device is controlling is determined by the mode in which the pointing device is currently in. This mode is controlled by issuing a control sequence to the CONOUT routine. In the initial state, the pointing device is placed in the character cursor mode. The CONOUT routine also supports the control sequence which reads the current cursor position on the screen. See Chapter 4 for details.

(1) Character cursor mode

Moving the direction control button in the character cursor mode generates escape code sequence associated with eight directions to the CONIN routine. These codes are identical to those which are generated by pressing cursor control keys in the ten numeric key group while holding down the CURSOR-LOCK key. They are also identical to the character cursor control code sequences supported by the CONOUT routine. The codes associated with the eight directions are listed in the table below.

Direction	Code
↓	ESC [A
↑	ESC [B
→	ESC [C
←	ESC [D
↖	ESC [D ESC [B
↗	ESC [C ESC [B
↙	ESC [D ESC [A
↘	ESC [C ESC [A

(2) Graphic cursor mode

In the graphic cursor mode, the pointing device passes no escape code sequences associated with the eight cursor directions to the CONIN routine. Instead, it outputs the escape code sequences directly to the CONOUT routine to speed up cursor movement.

In the graphic cursor mode, the movements of the graphic cursor are associated with the following increments of the x and y coordinates of the cursor:

Direction	Increment	
	Δx	Δy
↓	0	1
↑	0	-1
→	1	0
←	-1	0
↖	-1	-1
↗	1	-1
↙	-1	1
↘	1	1

In the high-speed mode (when the fast button is pressed), the increments of the x and y coordinates are multiplied by five and three respectively.

(3) Pointing device function keys

The pointing device has three function keys (A, B, and C) which provide the same functions as some special keys. Pressing these keys generate the corresponding escape code sequences to the CONIN routine, irrespective of whether the pointing device is in the character or graphic cursor mode. The escape code sequences generated by the function keys are listed below.

Key	Generated codes
A	ESC 0
B	ESC 1
C	ESC 2

CHAPTER 6 PRINTER INTERFACE

The AS-100 is provided as standard with a parallel printer interface which conforms to the Centronics specifications. The AS-100 allows attachment of any printer which matches this interface.

Canon supplies several printers for the AS-100. You can use any of these printers with no special handling program under AS-100 CP/M-86. However, you need a special handling program tailored to our printer when you want to produce hard copies or handle two printers at a time (another parallel interface option is required in this case).

This chapter describes the use of the printer handling program supported by AS-100 CP/M-86.

6-1 Printer Handling Commands

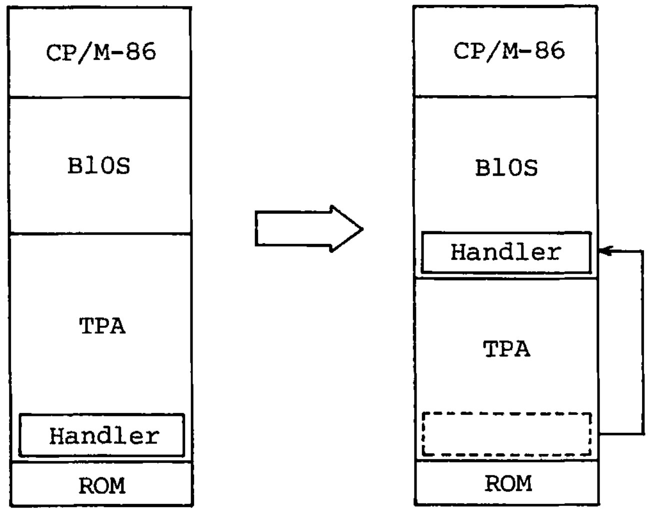
The AS-100 CP/M-86 operating system supports the following printer handling subprograms in the form of transient commands:

Handler name	Printer type	Main function	Required size*(KB)
A1200	A-1200	Produces hard copy of screen data.	3
A1210	A-1210	Produces hard copy of screen data in colors.	3
CNTHND	Centronics compatible printer	Provides no special functions (handles second printer).	0.3

* Size of the resident routines linked to BIOS.

AS-100 CP/M-86 contains a standard LPT: device handling program in its BIOS. Therefore, the user can use a printer without the above handler programs.

When a printer handling program is executed, it is loaded into the TPA area, then it relocates itself to the area following the BIOS area. The program then links to BIOS and invalidates the preceding printer handler. The printer handler programs must not be invoked indiscriminately since if the same printer handler were executed more than once, deal located areas would be created in the BIOS area.



Control is returned to CP/M-86 when the linkage between the handler and BIOS is completed.

Once the printer handler program is executed, the user can obtain hard copy of screen contents. It would therefore be convenient if the user incorporates the handler programs tailored to his printer into the automatic setup command file (START.SUB). He may not need these handlers, however, if he does not want the above listed special printer functions.

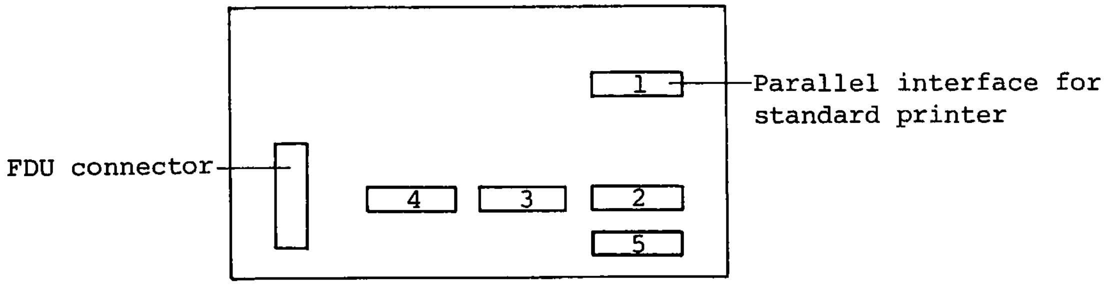
6-2 Device Assignments and Operations

A printer handler is invoked in the following command format:

<handler-name>[{␣}$_1$<device-name>]

You can specify either LPT: or UL1: as the device name. LPT: is assumed when you omit this parameter. This device name is a physical device name which is associated with the logical device name LST: by the STAT command through IOBYTE. Initially LPT: is associated with LST:.

The parallel printer interface of the AS-100 is preassigned to port number 1 as shown in the figure below. Port numbers 2 through 4 are reserved for the optional input/output devices.



AS-100 rear panel

Two parallel interfaces can be installed. The first interface have already been installed in the standard port number 1 slot and the second interface can be added to one of the slot assigned to port numbers 2 through 4.

The standard parallel interface (port number 1) is assigned to physical device LPT:. Therefore, if only one printer is connected to the AS-100, you need execute the printer handler programs only for device LPT: (LPT: is assumed when you omit the device name in the commands).

The optional parallel interface is assigned to physical device UL1:. In this case, you must execute the handler programs to device UL1: to use the second printer. Normally, you can use the standard printer without executing in advance the handler program to the standard printer interface since CP/M-86 incorporates a printer handler. Since CP/M-86 has no handling program for the second printer, you must execute the printer handler programs to device UL1:.

When a printer handler program is executed to device UL1:, the user can control two printers, LPT: and UL1:. The logical list device name LST:, however, can be associated with only LPT: or UL1: at a time. For example, you can use the printer connected to the standard interface if you associate LST: with LPT: with the STAT command and you can use the printer connected to the optional interface if you associate LST: with UL1:.

In addition to switching between physical devices LPT: and UL1: through IOBYTE, you can control the second printer using the logical device name AXO: under CP/M-86. The logical name AXO: is associated with one of physical devices TTY:, PTP:, UP1:, and UP2:. The optional parallel interface is assigned to one of port numbers 2 through 4. The table below lists the physical device names and the associated port numbers.

Port Number	2	3	4
Physical device name	TTY:	PTP:	UP1:

When a printer handler program is executed for device UL1:, it checks the port number of the optional parallel interface and makes the corresponding physical device available to the user. By associating the logical device name AXO: with the target physical device name with the STAT command, you can use the standard printer under LST: and the optional printer under AXO: simultaneously.

The above two methods of using two printers at a time are exemplified below. It is assumed that an A-1200 printer is used as the standard printer and an A-1210 printer is used as the optional printer (assigned to port number 2).

To execute the printer handlers, type in:

```
A1200
A1210    UL1:
```

To print the contents of the text file "TEST.TXT" in the floppy disk on device B: to each of the printers using the PIP command, type in:

```
PIP LST:=B:TEST.TXT
```

The above command will direct the contents of the file to the A-1200.

The commands

```
STAT LST:=UL1:
PIP LST:=B:TEST.TXT
```

will output the contents of the file to the A-1210.
In this way, you must reassign the list device using the STAT command to use the A-1200 printer again.

```
STAT LST:=LPT:
```

If you preassign the A-1210 printer to AXO: with the command

```
STAT AXO:=TTY:
```

then you can direct the file data also to the A-1210 using the command

```
PIP AXO:=B:TEXT.TXT
```

without reassigning the LST: device with the STAT command.
The second method is convenient since it dispenses with the need to between the physical devices LPT: and UL1: to reassign to the logical device LST: using the STAT command.

6-3 Executing AS-100 Function Calls

Once a printer handler program is linked to BIOS, the user program can use the handler all through BDOS. The user who used an assembler language, however, can use the handler directly using AS-100 function calls. The calling sequence is given below.

CL register ← Function number

INT 240 (F0h)

The available function numbers and applicable devices (names) are listed below.

Function	Function Number (decimal)	
	LPT:	UL1:
Initializing	35	40
Getting output status	38	43
Outputting data	39	44

(1) Initializing

Initializes the physical device (LPT: or UL1:) and the handler. The result of the function execution is returned into the AX register.

AH register: Handler identification code

A1200	01h
A1210	02h
CNTHND or built-in handler	00h
Undefined	0FFh

AL register: Status

Normal termination	00h
Abnormal termination	Nonzero

(2) Getting output status

Reads the printer status into the AL register.

Printer ready	0FFh
Printer not ready	00h

(3) Outputting data

Outputs the data stored in the AL register.

6-4 A1200 Command

This command is for the wire-dot matrix printer model A-1200.

Characters Print

The 1-byte codes output by the A1200 handler are described below.

(1) Control codes (00-1Fh)

The handler outputs control codes CR (0Dh), LF (0Ah), FF (0Ch), CAN (18h), ESC (1Bh), DC1 (11h), and DC3 (13h) as are. It ignores the other control codes.

(2) Alphanumeric characters (20h-7Fh)

The handler outputs codes 20h through 7Eh as are and ignores the DEL code (7Fh).

(3) Special codes (80h-FCh)

The handler outputs codes 80h through 0FCh as are.

(4) Ignore codes (FDh-FFh)

The handler ignores codes 0FDh through 0FFh.

Escape Sequence

The A1200 handler performs special functions (described below) specified by the character strings following an ESC code. These functions are inherent to the A1200 handler; in addition to these escape sequences, the A-1200 printer has its own escape functions. Refer to the A-1200 printer manual for details.

(1) Enlarged mode (ESC 1)

ESC 1 (1B31h) puts the A-1200 printer into the enlarged character mode in which the printer prints the characters all in an enlarged size. This does not apply when a hard copy of the screen content is to be produced.

(2) Normal mode (ESC c)

ESC c (1B63h) restores the printer from the enlarged character mode to the normal character mode.

(3) Screen hard copy (ESC #7)

ESC #7 (1B2337h) causes the data currently displayed on the screen to be printed on the printer as is. Note that the line spacing is set to 6 (LPI) after this code sequence is issued.

6-5 A1210 Command

This command is for the color ink jet printer model A-1210 which can produce a hard copy of the color CRT screen contents.

Characters Printed

The 1-byte codes output by the A1210 handler are described below.

(1) Control codes (00-1Fh)

The handler outputs control codes CR (0Dh), LF (0Ah), FF (0Ch), CAN (18H), ESC (1Bh), DC1 (11h), and DC3 (13h) as are. It ignores the other control codes.

(2) Alphanumeric characters (20h-7Fh)

The handler outputs codes 20h through 7Eh as are and ignores the DEL code (7Fh).

(3) Special codes (80h-DFh)

The handler outputs codes 80h through DFh as are.

(4) Ignore codes (FDh-FFh)

The handler ignores codes FDh through FFh.

Escape Sequence

The escape code sequence ESC #7 (1B2337h) causes the data currently didplayed on the color CRT screen to be printed on the printer. Note that the line spacing is set to 6 (LPI) after this code sequence is issued. This function is inherent to the A-1210 handler; in addition to these escape sequences, the A-1210 printer has its own escape functions. Refer to the A-1210 printer manual for details.

A hard copy of color CRT screen contents is reproduced on the printer in colors which are most similar to the colors set up for the screen data. This means that there are some cases in which data displayed on the screen in different colors is printed on the printer in the same color. This causes no problem when the data on the screen uses only basic colors.

6-6 CNTHND Handler

The CNTHND handler provides no special function and outputs print data received by the user program to the printer as is. This handler is used primarily to drive the optional parallel interface which is connected to a printer other than the A-1200 and A-1210. This is functionally equivalent to the printer handler incorporated in AS-100 CP/M-86. It provides the data buffering function.

6-7 Messages

The printer handler programs, when executed, display the following message on the screen:

XXXXXX HANDLER Vm.nn

XXXXXX indicates the name of the handler in execution and Vm.nn indicates its version number. The handlers issue error messages when they encounter error conditions during execution.

&& ILLEGAL PARAMETER

(An illegal device name was specified. The handler will not be executed properly.)

&& INSUFFICIENT MEMORY

(The TPA is too small to execute the handler. The handler will not be executed properly.)

&& NO OPTIONAL PORT

(No optional parallel interface was installed when device name UL1: was specified. The handler will not be executed properly.)

&& IRQ ENTRY FULL

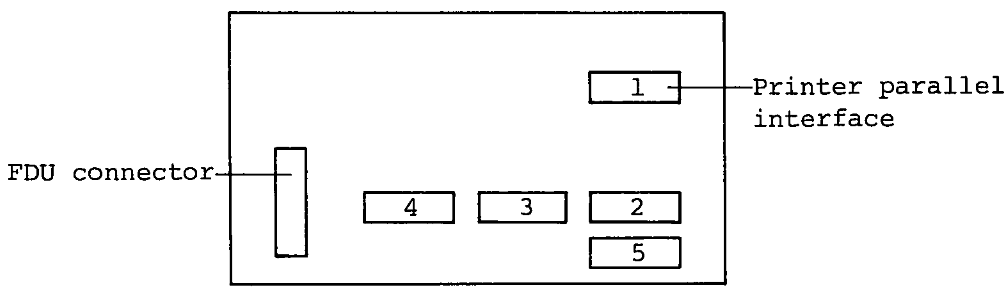
(There was no free entry in the IRQ chain when an attempt was made to link the handler to BIOS. The handler will not be executed properly.)

CHAPTER 7 RS232C INTERFACE

The AS-100 system can accommodate up to four channels of RS232C interface. This chapter describes the use of the RS232C interface under AS-100 CP/M-86.

7-1 Input/Output Port Assignments

The AS-100 display unit has five input/output port slots as shown in the figure below. Slot Nos. 2 through 5 are available for the RS232C interface. Slot No. 1 is used for the parallel interface for the standard printer.



AS-100 rear panel

The user can install RS232C interface channels in all or any of the input/output port slot Nos. 2 through 5; however, he must know the slot numbers to which RS232C interface channels are installed.

Under CP/M-86, the logical and physical devices are linked through an area called the IOBYTE. The relationship between them is summarized in the table below.

Logical device name	Physical device name			
CON:	TTY: *	CRT:	RAT:	UC1:
AXI:	TTY: *	PTR: *	UR1: *	UR2: *
AXO:	TTY: *	PTP: *	UP1: *	UP2: *
LST:	TTY: *	CRT:	LPT:	UL1:

Asterisks indicate the physical devices to which the RS232C interface can be attached.

The linkage between the logical and physical devices are established by the STAT command. Initially, they are linked as follows:

Console device	CON:=CRT:
Input device	AXI:=PTR:
Output device	ACO:=PTP:
List device	LST:=LPT:

In the above table, physical devices indicated by an asterisk can be connected to the AS-100 through an RS232C interface. They are assigned to the following input/output ports:

Input/output port No.	2	3	4	5
Physical device name	TTY:	PTR: PTP:	UR1: UP1:	UR2: UP2:

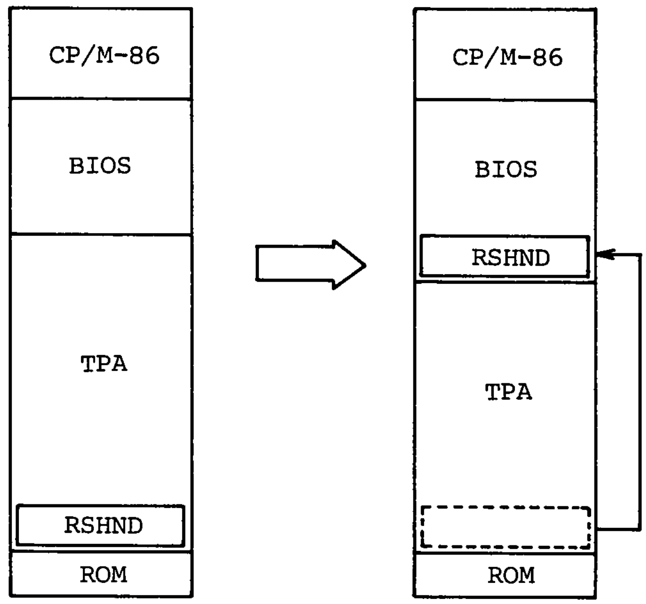
When an RS232C interface is assigned only to port No. 2, only the physical device TTY: is available for the user. The combinations of physical device names PTR: and PTP:, UR1: and UP1:, and UR2: and UP2: refer to the same port number or input/output port.

Since the RS232C interfaces in all input/output ports have the same specifications, the user can transfer data to or from port No. 3 as either PTP: (Paper Tape Punch) or PTR: (Paper Tape Reader).

However, since PTR: is associated with logical device name AXI: and PTP: with logical device name AXO according to the CP/M-86 conventions, the user program must distinguish between the input and output device names, that is AXI: for PTR: (input) and AXO: for PTP (output).

7-2 RS232C Handling Commands

CP/M-86 provides two transient commands to handle the RS232C interfaces. The first command, RSHND, links the RS232C handling routines to BIOS. The second command, RSINIT, sets the parameters which specifies the operating environment (in the RSHND handler area) of the RS232C interfaces assigned as physical devices. For this reason, the RSHND command must be executed before the RSINIT command. The RSHND command, when executed, loads the RSHND handler into the highest TPA area, then relocates it to the area following the BIOS area. The command then links the RSHND handler with BIOS and returns control to CP/M-86. The resident RSHND handler occupies approximately 1.5K bytes of memory.



Once the RSHND command is executed, the RS232C handler is made resident in the BIOS area. If another RSHND command is executed, the same handler will be placed immediately following the preceding RSHND handler, putting this handler into disuse. Since the TPA is reduced by approximately 1.5K bytes every time the RSHND command is executed. Don't execute the RSHND command more than once.

The RSINIT command sets the parameter values defining the manner in which the RSHND handler handles the physical devices. The possible parameter values and initial values (enclosed in parentheses) are listed in the table below.

Baud rate	110, 300, 600, 1200, (2400), 4800, 9600
Data format	(7 bits), 8 bits
Parity bit	None, (even), odd
Stop bits	(1), 2
XON/XOFF	(Yes), no
Wait cycle	(Yes), no
Time filler	Yes, (no)
Buffer size	(0), 16-4096 bytes

Values enclosed in parentheses are initial values set up when the RSHND command is executed.

There are two types of buffers, one for input and the other for output. At least one buffer must be specified. The buffer area of the size specified by the buffer size parameter is reserved following the BIOS area. The user must specify the appropriate buffer area size. See Section 7-4 for the parameters.

7-3 RSHND Command

Format: RSHND

Parameter: None

Description: The RSHND command links the RS232C handler to BIOS.
Although no parameter is specified, the RSHND handler is initialized when this command is executed.

7-4 RSINIT Command

Format: RSINIT{␣}$_1$<device-name>{␣}$_1$<baud-rate>{␣}$_1$<data-format>{␣}$_1$<options>

Parameter: <device-name>

Specify one of the following device names:

Device name	TTY:	PTR:	PTP:	UR1:	UP1:	UR2:	UP2:
Port No.	2	3		4		5	

<baud-rate>

Specify one of the following baud rates:

Baud rate	110	150	300	600	1200	2400	4800	9600

Note that the same baud rate must be specified to a port to which two device names are assigned.

<data-format>

Specifies the format of the data to be sent or received. Code one of the following combinations according to the character length (7 or 8 bits), parity (E or O), and the number of stop bits (S or SS).

Data format	7ESS	7OSS	7ES	7OS	8SS	8S	8ES	8OS

<options>

Specifies option(s). When coding two or more options, separate them with space(s). The options may be specified in any order. When options are omitted, the last values or initial values (which are set up when the RSHND command is executed) remain valid.

BUFSIZ(n1,n2), BUFSIZ(n1,), BUFSIZ(,n2), or BUFSIZ(n1)

Specifies in bytes the size of the input buffer (n1) and/or the output buffer (n2). You can specify a value 16 to 4096. When a zero is specified no buffer area is reserved, in which case no input/output operation is allowed. When n1 or n2 is omitted, a default value of zero is assumed. n2 is assumed to be equal to n1 when BUFSIZ(n1) is specified. The buffer area(s) are cleared each time an RSINIT command is executed.

AUTOX

Specifies that the auto XON/XOFF mode is to be used. With this specification, the handler temporarily suspends transmission when it receives the XOFF code (13h) from the remote terminal and resumes transmission when it receives the XON code (11h). This mode is used when the handler is to be connected to a terminal which cannot generate the busy signal.

NOAUTOX

Specifies that the auto XON/XOFF mode is not to be used.

WAIT

Specifies that the handler is to enter the wait state when there is no receive data.

NOWAIT

Specifies that the handler is to return control to the user program with a null code (00h) when there is no receive data.

FILL(C1, C2, n)

Specifies the time filler. When this option is specified, the handler transmits the number of C2 codes specified by n after transmitting the code C1. Code C1, C2, and n with 1- or 2-digit hexadecimal numbers. When a 1-digit number is specified, a zero is assumed in the higher order digit position (nibble).

Explanation: The RSINIT command allows you to specify the parameter values for each of the input/output devices. When options are omitted, the default values which are established when the RSHND command is executed are retained.

You must always reserve buffers. The recommended buffer size is 1 to 3 times the length of the records to be handled by the user program.

Note that the XON (11h) and XOFF (13h) cannot be handled as data bytes when AUTOX is specified.

You should specify the NOWAIT option when you do not want the program execution to be discontinued because of the handler indefinitely waiting for receive data from the terminal. When NOWAIT is specified, the handler returns a null code when there is no receive data, so that the user program can determine whether it must continue to wait for input from the terminal or it must continue its processing. The time filler should be used for a printer which does not return the Busy signal when it performs an operation (such as form feed) which takes a long processing time. When you specify FILL (0A,00, 05), then the handler will transmit five null codes successively after transmitting a line feed code (0A). At 300 bauds, it provides a wait time of approximately 33.3 x 5 milliseconds.

Examples: Transmit 15 null codes (00h) as a time filler after transmitting a CR code (0Dh) to PTP: which is assigned to a printer without the AUTO XON/XOFF beature. Assume that the baud rate is 300, that the data format is 7 data bits, even parity, and 1 stop bit, and that the output buffer size is 133 bytes (1 line length).

RSINIT PTP: 300 7ES BUFSIZ(0,133) NOAUTOX FILL (0D,00,0F)

Connect one AS-100 to another AS-100 via TTY: Assume that the baud rate is 4800, that a data byte consists of 8 data bits, 1 odd parity bit, and 1 stop bit, and that NOWAIT mode is used. Also assume that 512 bytes of buffer space are reserved for each of the input and output buffers.

RSINIT TTY: 4800 7OS BUFSIZ(512) NOWAIT

Error messages: If an error is detected during the execution of the RSINIT command, one of the following error messages is issued and the command processing is terminated:

&& Parameter not exist!!

(No parameter was specified.)

&& Port not declared!!

(No device name was specified.)

&& Invalid parameter!!

(An invalid parameter was specified.)

&& Size error!!

(An invalid buffer size was specified.)

&& Lack of memory space!!

(The specified buffer size is too large.)

&& Not I/O board!!

(The specified physical device is not installed.)

7-5 Executing AS-100 Function Calls

Once the input/output handler is linked to BIOS by the RSHND command and the necessary parameter values are set by the RSINIT command, the user program can use the handler all through BDOS calls. The user who uses an assembler language, however, can use the handler directly using AS-100 function calls. The calling sequence is given in next page.

CL register ← Function number

INT 240 (F0h)

The available function numbers and applicable devices (names) are listed below.

Function	Function number						
	TTY:	PTR:	PTP:	UR1:	UP1:	UR2:	P2:
Initializing	5	20	20	25	25	30	30
Getting input status	6	21	-	26	-	31	-
Inputting data	7	22	-	27	-	32	-
Getting output status	8	-	23	-	28	-	33
Outputting data	9	-	24	-	29	-	34

(1) Initializing

Initializes the physical device and handler. The result is returned into the AL register.

Normal termination 00h

Abnormal termination Nonzero value

(2) Getting input status

Reads the input port status into the AL register.

Input port ready 00h

Input port busy Nonzero value

(3) Getting data

Reads receive data into the AL register.

(4) Getting output status

Reads the output port status into the AL register.

Output port ready 00h

Output port busy Nonzero value

(5) Outputting data

Writes data in the AL register to the given output port.

CHAPTER 8 EXTENDED UTILITY COMMANDS

AS-100 CP/M-86 has four extended utility commands in addition to those provided by standard CP/M-86.

The four utility commands are FORMAT, VOLCOPY, MS2CPM, and MCX2CPM. The functions of these commands are as follows:

FORMAT ..... Formats a floppy disk so that it can be used by AS-100 CP/M-86.

VOLCOPY .... Copies or backs up an entire floppy disk volume to another disk.

MS2CPM ..... Converts a file created by AS-100 MS-DOS* to CP/M-86 format.

MCX2CPM .... Converts a file created by the Canon CX-1 system to CP/M-86 format.

This chapter provides detailed explanations of these extended commands and describes procedures for using them.

* MS-DOS is a trademark of Micro Soft

8-1 FORMAT Command

Function: Formats a 5- or 8-inch floppy disk and writes the secondary loader to the disk. A disk formatted by this command can be read or written by AS-100 CP/M-86. The FORMAT command can also copy system files (whose attribute is SYS) from the current disk.

Format: FORMAT$\{\sqcup\}_1$[<drive-name>]$\{\sqcup\}_1$[<loader-file name>]$\{\sqcup\}_1$[$S]

Parameters: <drive-name>

Specify the name of the floppy disk drive as A:, B:, C:, D:, E:, or F:. Drives E: and F: can be used only for an 8-inch, single-sided, with single density floppy disk. When this parameter is omitted, drive B: is assumed.

<loader-file name>

Specify the name of the secondary loader to be copied to the formatted disk. When omitted, the secondary loader on the system track of the current disk is copied. Generally this parameter is not specified, and is ignored when drive E: or F: is specified.

$S

Specify this parameter when all files (whose attribute is SYS, but whose extension is not SYS) on the current disk are to be copied after formatting. When omitted, no files are copied. This parameter is ignored when drive E: or F: is specified.

Prompt messages:

FORMAT Vn.mm

This message appears when the FORMAT command is entered.
Vn.mm indicates the version number of the command.

Disk d: will be destroyed, OK?

This message warns that the contents of the floppy disk to be formatted will be destryed. "d:" indicates the drive name specified. Enter "Y" to start processing or "N" to cancel processing.

FORMAT TRACK NUMBER=nn

This message appears only when an 8-inch floppy disk is specified. "nn" indicates the track number being formatted.

COPYING SECONDARY BOOT

This message is displayed while the secondary loader is being copied.

COPYING SYSTEM FILES

This message appears when a SYS file is being copied. The user can cancel the copy operation while this message is displayed by pressing CTRL/C.

Remarks: Operation of the FORMAT command differs depending on whether a of 5- or 8-inch disk is being formatted. For an 8-inch disk, all sectors from track 0 to the last track are physically formatted. A double sided, double density disk is formatted so that each sector contains 1024 bytes, while a single sided, single density disk is formatted so that each sector contains 128 bytes. (The latter type of disk is assumed when drive E: or F: is specified.) These are the same disk formatting specifications as are used with standard CP/M-86. The contents of the directory are cleared after all sectors of all traks have been formatted. However, for a 5-inch disk, the directory is cleared but sectors are not physically formatted. Therefore, when a 5-inch disk is required, use one which has been formatted as a double sided disk with double density, double track, and 512 bytes/sector speifications. When an 8-inch single sided disk with single density is formatted, neither the secondary loader nor SYS files are copied even if the parameter is specified.

Examples:

FORMAT

Formats the floppy disk in default drive B:.

FORMAT C: $S

Formats the floppy disk in drive C: and copies the SYS files.

Error messages:

When an error occurs during execution of the FORMAT command, one of the following messages is displayed and processing is terminated.

Illegal device name specified

An illegal drive name was specified.

Specified device is default device

The drive name specified was that of the current drive.

Too many operands

Too many parameters were specified.

Illegal operand

An illegal parameter was specified.

Illegal secondary boot file name

An illegal name was specified for the secondary loader.

File name syntax error

A syntax error was found in the secondary loader file name specification.

Secondary boot file not found

The specified secondary file was not found.

Secondary boot file specified on format media

The secondary loader file was specified for the formatted floppy disk.

TOO LONG SECONDARY BOOT FILE

The secondary loader file is too large for the system track.

READ ERROR

An I/O error occurred while the secondary loader was being read from the system track.

WRITE ERROR

An I/O error occurred while the secondary loader was being written to the system track.

FORMAT DEVICE NOT READY

The drive containing the disk to be formatted was not ready.

FORMAT ERROR

A write error occurred during formatting.

SEEK ERROR

A seek error occurred during formatting.

SYSTEM FILE OPEN ERROR

An error occurred when the source system file was opened with the $S specification.

SYSTEM FILE MAKE ERROR

No free directory space was found when the destination system file was opened with the $S specification.

EOV DETECTED

No free space was available when the system file was being copied with the $S specification.

FORMAT ABORTED

Processing was cancelled by pressing CTRL/C while the system file was being copied with the $S specification.

8-2 VOLCOPY Command

Function: Copies the entire contents of a 5- or 8-inch floppy disk to another one. The VOLCOPY command can only be used with standard format floppy disks used by AS-100 series CP/M-86. Therefore, it cannot be used for 8-inch single sided disks with single density. Moreover, the VOLCOPY command can be used to copy a 5- or 8-inch disk to another disk of the same size, but not for copying between the two disks of different size. Parameters required for use of this command are entered through an interactive sequence.

Format: VOLCOPY

Parameter: None

Prompt messages:

VOLCOPY Vn.mm

This message appears when the VOLCOPY command is entered. Vn.mm indicates the version number of the command.

Enter Source Disk Drive (A-D)?

When this message is displayed, specify the name of the source drive as A:, B:, C:, or D:.

Destination Disk Drive (A-D)?

When this message is displayed, specify the name of the destination drive as A:, B:, C:, or D:. The source and destination drive names must be different.

Copying disk s: to disk d:

Is this what you want to do (Y/N)?

This message appears before the VOLCOPY command is executed to confirm that the user really wants to copy the disk. The entire volume in drive S: is copied to the drive d:.
Enter "Y" to start processing or "N" to cancel.

COPY TRACK NUMBER=nn

This message appears while the disk is being copied. "nn" indicates the track number currently being copied.

COPY another disk (Y/N)?

This message appears when copying is completed. Enter "Y" to make a copy on another disk or "N" to terminate the command execution. When copying to another disk, be sure to insert a suitable disk in the destination drive before entering "Y".

After the VOLCOPY command is entered and the first message is displayed, the user can replace the disk in the source drive with another one. In this case, be sure to replace the system disk when this command execution is completed and the last message is displayed, or to execute a warm boot (CTRL/C) upon completion of the command.

Remarks: The VOLCOPY command copies each track from the source disk to the destination disk. Since the volume is copied from track 0, the system track containing the secondary loader is also copied. Therefore, the volume can be copied to a new 5-inch disk which has not been formatted with the FORMAT command. When copying to a new 8-inch disk whose physical sector size differs from that for a 5-inch disk, be sure to format the disk with the FORMAT command before executing VOLCOPY command.

The VOLCOPY command performs track-to-track copy operation and, at the same time, vertifies whether tracks have been properly copied. When a track is copied improperly, up to seven retries are made.

An error results if a track is copied properly after seven retries. Copying takes about 130 seconds for a 5-inch disk, and about 100 seconds for an 8-inch disk.

During execution of the command the user can cancel processing by pressing CTRL/C.

Error messages:

When one of the following errors is detected, VOLCOPY issues the corresponding message, then prompts for reentry of the proper parameter or terminates command execution.

Illegal input device specified

Illegal output device specified

An illegal source or destination drive name was specified.

Output device same to input device

The destination drive name was the same as the source drive name.

Different device type specified

The source and destination drives specified are of different types.

DISK I/O ERROR d:

An I/O error occurred in drive d:

OUTPUT DISK DESTROYED

Command execution was cancelled by pressing CTRL/C. The contents of the destination disk are not assured.

INSUFFICIENT MEMORY

Copying cannot be performed because there is not enough memory.

8-3 MS2CPM Command

Function: Converts a floppy disk file created by AS-100 MS-DOS to a file which can be used by AS-100 CP/M-86.

Format: MS2CPM$\{\sqcup\}_1$<CP/M filename>=<MS-DOS filename>[$\{\sqcup\}_1$$O]

Parameters: <CP/M filename>

Specify the CP/M-86 filename in conformance with the CP/M-86 file specifications: <drive-name>: <file-name>. <extension>. When this parameter is omitted, the MS-DOS filename is assumed.

<MS-DOS filename>

Specify the MS-DOS filename in conformance with MS-DOS file specifications: <drive-name>: <file-name>. <extension>. One or more wild card characters ("*") can be used in the filename and/or extension. In this case, place the character "*" in the first and/or last position in each parameter. (The following example is invalid: AB*EF.GHI.)

$O

Specify this parameter when the MS-DOS source file is a binary data file and blank spaces in the last record of the file are padded with NUL codes (00H). When this parameter is omitted, the MS-DOS source file is handled as a text data file and blanks in the last record is padded with CTRL/Z codes (1AH).

Prompt messages:

MS-DOS to CP/M-86 file converter Vn.mm

This message appears when the MS2CPM command is entered.
Vn.mm indicates the version number of the command.

<CP/M filename>=<MS-DOS filename>

This message indicates the file name of the file being processed.

n files copied

This message is displayed when conversion processing is completed to indicate the number of files converted.

Remarks: The MS2CPM command searches for the specified file in the directory of a 5- or 8-inch floppy disk created by AS-100 MS-DOS, then converts that file to AS-100 series CP/M-86 format. File records handled by CP/M-86 have a length of 128 bytes while those handled by MS-DOS are of variable length. Therefore, if the last record length of an MS-DOS file is less than 128 bytes, a "short record" is generated in that CP/M-86 file. In this case, blank spaces in the short record are filled with CTRL/Z codes (1Ah) or NUL codes (00H). Whether NUL or CTRL/Z is used depends on whether the optional $O parameter is specified. Conversion is done for each byte; however, specific processing such as code conversion is not performed. The user can cancel execution of this command by pressing CTRL/C.

Examples:

MS2CPM A:=B:*.DAT $O

All MS-DOS disk files on drive B: whose file names have the extension ".DAT" are converted to CP/M-86 disk files on drive A:. The file name assigned to each CP/M-86 file is the same as that of the corresponding MS-DOS source file; blank spaces in short records of CP/M-86 files are filled with NUL codes (00H).

MS2CPM A:SOURCE.LST=C:PROG.LST

The MS-DOS disk file on drive C: whose file name is "PROG. LST" is converted to a CP/M-86 disk file on drive A: with the file name "SOURCE.LST". Since the $O parameter is not specified, blank spaces in the short record of the CP/M-86 file are filled with EOF codes (1AH).

Error messages:

Error: Bad parameter

An illegal command parameter was detected.

Error: Write error

An error occurred while a record was being written to the CP/M-86 object disk.

Error: Read error

An error occurred while a record was being read from the MS-DOS source disk.

Error: File not found

The specified source file does not exist on the MS-DOS disk.

Error: Invalid Source

The source disk has not been formatted by AS-100 MS-DOS.

Error: No directory space

No free directory space is available on the CP/M-86 disk.

Abort requested

Command execution was cancelled by pressing CTRL/C.

8-4 MCX2CPM Command

Function: Converts floppy disk files created by the MCX operating system of the Canon CX-1/BX-3 to files which can be handled by AS-100 CP/M-86.

Format: MCX2CPM$\{\sqcup\}_1$<CP/M filename>=<MCX filename>

Parameters: <CP/M filename>

Specify the CP/M-86 file name in conformance with CP/M-86 file specifications: <drive-name>: <file-name>.<extension>. When this parameter is omitted, the MCX file name is assumed.

<MCX filename>

Specify the MCX file name in conformance with MCX file specifications: <drive-name>: <file-name>. <extension>. One or more wild card characters ("*") can be used in the file name and/or extension. In this case, place the wildcard character ("*") in the first and/or last position in each parameter. The following example is invalid.

AB*EF.GHI

Prompt messages:

MCX2CPM Vn.mm

This message appears when the MCX2CPM command is

entered.
Vn.mm indicates the version number of the command.

COPYING-

<filename>

This message appears during execution of the command. The file name of the file being processed is displayed when one or more wildcard characters are used in any parameter.

Remarks: The MCX2CPM command searches for the specified file from the directory of a 5-inch floppy disk in MCX format(double sided, double density with 256 bytes/sector) and converts it to a file in AS-100 series CP/M-86 format. This command cannot be used with 8-inch disks for an MCX file. CP/M-86 handles files in which each record is 128 bytes in length, while MCX handles byte-configured files which contain no records. This is because a "short record" is derived in a CP/M-86 file when the last record length of an MCX file is less than 128 bytes. In this case, blank space in the resulting short record of the CP/M-86 file is filled with one or more CTRL/Z codes of EOF codes (1Ah). Conversion is performed for one at a time; however, special processing (such as code conversion) is not performed. The user can cancel execution of this command by pressing CTRL/C.

Examples:

MCX2CPM A:=B:*.LST

All MCX disk files on drive B: whose file names include the extension ".LST" are converted to CP/M-86 disk files same as that of the corresponding MCX source file.

MCX2CPM C:PROGB=B:PROGA

The MCX disk file on drive B: whose file name is "PROGA" is converted to a CP/M-86 disk file on drive C: with the file name "PROGB".

Error messages:

Input device is same to output device or current device. The specified source drive was the same as the object or current drive.

Input file name syntax error.

An MCX syntax error was detected in the source file name specification.

Input device specified 8 inches drive.

An 8-inch floppy disk drive was specified as the source device.

Input file not found

The specified source file does not exist on the MCX disk.

Input file name not specified

No source file name was specified.

Illegal device name specified

A character other than A, B, C, and D was specified for the source or target drive name.

Illegal output file name specified

An illegal file name was specified for the object file.

Illegal wildcard specified

The wildcard character (*) was incorrectly used in the source file name.

OUTPUT FILE MAKE ERROR

No free directory space is available on the CP/M-86 disk.

EOV DETECTED

An overflow error occurred while a file was being written to the CP/M-86 object disk.

READ ERROR

An error occurred while a file was being read from the MCX source disk.

APPENDIX A CRT CODE TABLE

m n	0	1	2	3	4	5	6	7
0	NUL	DEL	Space	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL*	ETB	'	7	G	W	g	w
8	BS *	CAN	(	8	H	X	h	x
9	HT *	EM	)	9	I	Y	i	y
A	LF *	SUB	*	:	J	Z	j	z
B	VT *	ESC*	+	;	K	[	k	{
C	FF *	FS	,	<	L	\	l	¦
D	CR *	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL*

Note: 00h to 1Fh and 7F are control codes. Control codes which are valid for the CRT are those indicated by "*"; others are invalid.

n \ m	8	9	A	B	C	D	E	F
0	space		space	É			√	
1	☺		§	Å			π	
2	☻		Ä	å			×	
3	♥		Ö	Æ			÷	
4	♦		Ü	ϕ			Σ	
5	♣		ä	æ			↑	─
6	♠		ö	Ĳ			↓	┼
7	•		ü	ĳ			→	├
8	◘		ß	ò			←	┤
9	○		à	ì			μ	
A	◙		°				σ	
B	♂	↖	ç				φ	├
C	♀	↵	é				α	─
D	¿	→|	ù				β	┤
E	▲	|←	è				γ	═
F	█	█	ē				£	█

APPENDIX B ROM DEBUGGER

The ROM debugger is installed in the boot ROM and it makes it possible to execute the system program step by step and to manipulate NMI, vector tables other than break points and memory space from 400h to FFFFFh.
(The upper 256 bytes of the installed memory and 4 bytes from 3FCh to 3FFh are used by the debugger.)

The debugger can be activated by either the following two methods.

(1) Turn the power on of the system without setting any floppy disk. The debugger start message is first displayed and the debugger waits for a command input.

(2) Press the STOP key (located in the hole at the lower left of the display unit) while the system is operating. Then, "@" followed by the CS and IP contents is displayed and the debugger waits for a command input.

The debugger prompt is "*". When it is displayed, any of the following commands can be input.

Command	Meaning	Operating
B	Boot	B[D]<CR>
D	Memory display	D[W]<ADDR>[,<ADDR2>]<CR>
G	Execution	G<CS:IP><OLD>[<ADDR>[,<BRK>]]<CR>
I	Read port	I[W]<PORT><CR/,>
N	Step execution	N<CS:IP><OLD>[<ADDR>]<CR/,>
O	Write port	O[W]<PORT>,<NEW><CR/,>
R	Reading Intel HEX file	R[<BIAS>]<CR>
S	Changing memory contents	S[W]<ADDR>,<OLD>[<NEW>]<CR/,>
X	Displaying/changing register contents	X<CR> X<REG><OLD>[<NEW>]<CR/,>
T	Changing debugger console	T<CR>

Items within [ ] can be omitted.
Underlined items are output by the debugger.
Items within < > are abbreviations.

Abbreviations

W When this is specified, the commands treats data in word units; otherwise, it treats data in byte unit.

<ADDR> Address. The format is segment:offset. segment and offset are specified with 4 digit hexadecimals or names of registers (refer to REG). When segment is omitted, the value of CS is used as the default value.

Ex) SP:1234

<BIAS> Program loading bias. Specify with 4-digit hexadecimal or name of register.

<BRK> Break point. The format is the same as that of [ADDR].

<CR> Entry of carriage return code.

<CR/,> Entry of carriage return code or comma. When a comma is entered, the debugger executes the same command on the next address. The next address is the current address + 1 for byte access (except I and O commands), the current address + 2 for word access or the current address plus the number of bytes of the command executed for step execution.

<CS:IP> The message output by the debugger consisting of the code segment and instruction pointer represented in 4 digit hexadecimals.

<NEW> New data to be set in register or memory. Specify with 2- or 4- digit hexadecimal.

<OLD> Data in register or memory. Displayed with 2- or 4-digit hexadecimal.

<PORT> Port address. Specify with 2- or 4-digit hexadecimal.

<REG> Represents a register with 2 characters.

AX	Accumulator
BX	Base
CX	Count
DX	Data
SP	Stack Pointer
BP	Base Pointer

SI	Source Index
DI	Destination Index
CS	Code Segment
DS	Data Segment
SS	Stack Segment
ES	Extra Segment
IP	Instruction Pointer
FL	Flag

Command

B - Boot

B[D]<CR>

Performs initial boot from drive A:. When D is specified, control remains in the debugger after booting.

D - Memory display

D[W]<ADDR>[,<ADDR2>]<CR>

Displays the contents of the memory area from <ADDR> to <ADDR2> in word or byte units. When <ADDR2> is omitted, the contents of address <ADDR> are displayed. Pressing any key during display stops display and the debugger enters the command wait state.

G - Execution

G<CS:IP><OLD>[<ADDR>[,<BRK>]]<CR>

When G is entered, the values of CS and IP of the instruction to be executed next and the contents of that address are displayed. To start execution at that address, enter <CR> without entering <ADDR>. To change the execution starting address, enter <ADDR> and to set a break point, enter <BRK>, then enter <CR>.

When the execution reaches the break point, the following message is displayed and control is returned to the debugger.

BR @<CS:IP>

I - Read port

I[W]<port><CR/,>

Data read from the specified port is displayed. Enter <CR/,>.

When "," is entered, data read from the same port is displayed again. (This allows the user to check the changing process of specified port.)

N - Step execution

N<CS:IP><OLD>[<ADDR>]<CR/,>

Displays the values of CS and IP of the instruction to be executed next and the contents (2 digits) of that address. Specify <ADDR> to change the execution address. When "," is entered, instruction at the specified address is executed.

O - Write port

O[W]<PORT>,<NEW><CR/,>

Write data to the specified port. When "," is entered, a hyphen is displayed and the next data is written to the port.

R - Read Intel HEX file

R[<BIAS>]<CR>

Loads the Intel HEX format file through the RS232C interface at I/O port No. 5 into memory. When <BIAS> is specified, each record load address plus <BIAS> becomes the load address. The initial setting of the RS232C interface are as follows: 2400 bauds, 8S. However, if these settings have been changed by the handler, the changed settings are valid. If any key is pressed while the debugger waits for file input, it returns to the command wait state of debugger.

S - Changing memory

S[W]<ADDR>,<OLD>[<NEW>]<CR/,>

Displays the contents of memory address <ADDR> when <NEW> is not specified. When it is specified, the new data is written to the specified address and read-after-write check is performed.

X - Displaying/changing register contents

X<CR>

X<REG><OLD>[<NEW><CR/.>

X<CR>displays the name and contents of each register. When <REG> is specified, the name and contents of the specified register are displayed. To change the contents of a register, specify <NEW>. When "," is entered, the contents of the next register are displayed. (The order of registers is shown in the previous section.)

T - Changing debugger console

T<CR>

Changes the debugger console from the AS-100 itself to the RS-232C interface at I/O port No. 5, or vice versa. After T CR has been entered, the message "ARE YOU SURE?" is displayed and the debugger waits for a key entry. Change is made only when Y is entered.

Calculating function

When entering a command, addition and subtraction can be used in <ADDR>, <BRK>, <CS:IP>, <NEW> and <PORT> by using "+" or "-".

Ex) DAX+100-BX<CR>

SDS:BX-100<,>

Error processing

If a command or operand error is detected, "#" is displayed and a line feed is performed. The debugger waits for reentry of the command. When the length of numeric data is too long. the excessive part of data is discarded and no error results. There is no way to correct the command entry. Therefore, to reenter command, cause an error forcibly (for example, enter @).
Keys other than the alphanumeric keys are not accepted.
Errors occurring after execution are also identified with #.

Note

When the debugger is activated by the STOP key, the screen may be disrupted. This is not an error. This is because CRT is controlled both by the ROM debugger and OS. If the screen is hard to see, enter "ESC Z" to reset the screen. To prevent the screen from being disrupted, change the console to RS-232C with the T command.

APPENDIX C DIP SWITCH

Each of the CPU card and I/O card of the main unit and the I/F card of the floppy disk unit incorporates a DIP switch. The settings of these switches are explained below.

(1) CPU card

SW1	SW2	SW3	SW4

SW1	ON	
SW2	ON	
SW3	OFF	
SW4	Clock option	ON: no, OFF: yes

(2) I/O card

SW1	SW2	SW3	SW4	SW5	SW6	SW7	SW8

SW1	ROM debugger console	ON: RS232C OFF: AS100 itself
SW2	OFF	
SW3	ON	
SW4	CRT code	ON: ASCII, OFF: JIS
SW5	CRT model	ON: color or monochrome 1-frame OFF: monochrome 2-frame
SW6	CRT model	ON: monochrome, OFF: color
SW7	OFF	
SW8	OFF	

(3) Mini-floppy disk unit

SW1	SW2	SW3	SW4	SW5	SW6	SW7	SW8

SW1	OFF
SW2	OFF
SW3	OFF
SW4	OFF

When device names are A: and B:,

SW5: OFF, SW6: ON, SW7: OFF, SW8: ON

When device names are C: and D:,

SW5: ON, SW6: OFF, SW7: ON, SW8: OFF

(4) Standard floppy disk unit

SW1	SW2	SW3	SW4	SW5	SW6	SW7	SW8

SW1 OFF

SW2 OFF

SW3 OFF

SW4 ON

When device names are A: and B:,

SW5: ON, SW6: OFF, SW7: ON, SW8: OFF

When device names are C: and D:,

SW5: OFF, SW6: ON, SW7: OFF, SW8: ON