DIGITAL RESEARCH™

CP/M®-86

595-3082

Printed in
United States of America

ZENITH data systems

HEATH

CONTENTS

Preface iv
Essential Requirements v

Introduction
Beginning Concepts 1
Examples of User/Computer Dialog 1
Microcomputer Concepts 3
Disks 3
The Operating System 9
Application Programs 10
CP/M Concepts 12
Files 12
Disk Drives 15
Commands 18
Software Preparation Procedures 25
Startup Procedure 25
Backup Procedures 27

Utilities
ASSIGN: The Utility that Assigns Winchester Partitions to Drives . 29
BACKUP: The Utility that Facilitates File Copying from Working
Disks and/or Partitions to Backup Storage Media 34
CONFIGUR: The Utility that Customized CP/M for Your
hardware and/or Preferences 66
FORMAT: The Utility that Prepares Disk or Partition Surface . . . 91
LDCOPY: The Utility that Copies the CP/M-86
Operating System between Disks 104
LIST: The Utility that Prints Text File Contents on Paper 110
RDDOS: The Utility that Allows CP/M-86
to Read a Z-DOS Directory or File 115
RESTORE: The Utility that Facilitates File Copying to Disks
and/or Partitions for Active Use 120

Appendices A–D
Introduction 149
PART: The Utility that Rearranges Winchester Disk Partitions and/or
Changes the Partition Used For Default Bootup A.1
PREP: The Utility that Initializes a Winchester Disk B.1
SHIP: The Utility that Moves Read/Write Heads
to a Safe Position C.1
Verify: The Utility that Isolates Recent Bad
Sectors on Your Winchester Disk D.1

PREFACE

CP/M* is an acronym for the Control Program for Microcomputers, the operating system that has been the industry standard because of its convenience and wide range of practical applications.

This manual is designed to help you use CP/M-86 productively in your microcomputer environment. It is divided into four parts plus an introduction.

If microcomputers or the CP/M operating system are new to you, the Introduction will acquaint you with the basic concepts and materials you will be using. The Introduction also contains a section on Software Preparation Procedures that describes a procedure for making backup and working copies. By doing so, you will be able to protect your investment by permanently storing your CP/M-86 distribution disks in a secure place.

- Part I, "Z-100 Utilities," contains instructions for using each of the utility programs created by Zenith Data Systems to enhance the use of CP/M-86 with your H/Z-100 series computer.

- Part II, the CP/M-86 "User's Guide," was written by Digital Research Corp., the creators of CP/M-86. It contains a basic introduction to the system, instructions for using the built-in commands, and instructions for using each of the utility programs supplied by Digital Research.

- Part III, the "CP/M-86 System Guide," contains a detailed description of the system functions and internal structure of CP/M-86. It is supplied by Digital Research for persons creating software that will use system functions.

- Part IV, the "CP/M-86 Programmer's Guide," describes the use of ASM-86 to create assembly language programs for the Intel 8086 family of microprocessors. The Programmer's Guide is also supplied by Digital Research Corp.

*CP/M is a registered trademark of Digital Research Corp.

PREFACE

Essential Requirements

Use of CP/M-86 is limited to the following specifications:

1. Distribution Media—Three, soft-sectored, double-sided, 5.25-inch, 48-TPI, floppy disks at double density.

2. Disk Format—Soft-sectored, 5.25-inch, 48-TPI or 96-TPI floppy disks at double density; soft-sectored, 8-inch floppy disks; or a single soft-sectored, 5.25-inch floppy disk and Winchester system.

3. Minimum Hardware—Z-100 or H-100 microcomputer with attached or separate video screen.

INTRODUCTION

Beginning Concepts

This part of the manual explains concepts that are important when you are using the CP/M Operating System within your Heath/Zenith microcomputer environment.

The concepts explained here are grouped under the headings:

- "Microcomputer Concepts" Provided for individuals who have never used microcomputers, and for individuals with limited microcomputer experience who wish to review.
- "CP/M Concepts" Provided for individuals who are unfamiliar with the CP/M Operating System.

Individuals who are proficient at using both microcomputers and CP/M can skip this section of the manual, and proceed directly to "Software Preparation Procedures."

To control and manipulate the CP/M Operating System, you will be responding to prompts that appear on the screen. Be sure to read "Examples of User/Computer Dialog" so that you will be able to distinguish the style used in this manual to represent user input and screen displays.

Examples of User/Computer Dialog

This text contains examples of user interaction with a microcomputer. In these examples, displays presented on the screen will be represented by the following typestyle:

```
THIS TYPESTYLE represents terminal displays

0123456789#$*?:=.A>()
```

Command syntax and entries that are displayed on the screen are represented in the following typestyle:

```
THIS TYPESTYLE represents command syntax

0123456789#$*?:=.A>()
```

INTRODUCTION

Examples of User/Computer Dialog

User input (the characters that you type through the terminal) except commands or entries displayed on the screen will be represented by boldface type, as shown:

BOLDFACE TYPE represents the things you type

0123456789#$*?:=[.]()

When you should enter a carriage return by pressing the key marked RETURN, the example will show ***RETURN***.

In many instances, the exact text of a display will vary by a few characters. This manual often substitutes a few letters in place of exact characters where variations are likely to occur. For instance, this manual will illustrate a program's serial number as "Serial number sss-sssss," while your terminal might actually display it as "Serial number 357-81469."

In cases where the exact characters you type will vary, this manual presents a description of the necessary characters within curved braces, {}.

Hence, this manual might explain that an entry should be made in the form **B:=A:**{filename.ext} ***RETURN***, when you actually type the characters **B:=A:CONFIGUR.COM** ***RETURN***.

Hardware device model numbers beginning with the "H/Z" prefix are references to either a Heath device, a Zenith device, or both. For example "H/Z-100" in this manual refers to hardware devices that are labeled either "H-100" or "Z-100."

INTRODUCTION

Microcomputer Concepts

Your microcomputer is a sophisticated piece of equipment that reflects the latest technical advances in the computer field. But this machine is practically useless without the programs (instructions) that tell it what to do.

These programs are stored on disks and are used by your microcomputer when they are required to perform a task. This section explains how the programs are stored on disks, shows you how to handle your disks, and defines two important types of program: the operating system and the application program.

Disks

Stored information, or data, is arranged in concentric rings on the surface of a disk. These rings are called "tracks." Each track is divided into areas called "sectors." Each sector contains data measured in units called "bytes." A byte of data could be one letter typed at the terminal keyboard or one instruction in a program. But since a byte is such a small unit, you will more often see data measured in "kilobytes." A kilobyte (abbreviated as "K") is equal to 1024 bytes.

Data are transferred to disks in the form of magnetic impulses generated by an electromagnet called the "read-write head." As the name implies, the read-write head can read data from the disk or write data on the disk—similar to the way in which the head in a tape recorder transfers magnetic impulses to and from a cassette tape.

But unlike a tape recorder, a disk drive unit can transfer data at any location on the disk surface almost instantly, because the drive is usually spinning the disk at a high rate of speed. Whenever the read-write head is instructed to read or write data at a particular location on the disk, it positions itself along the appropriate track and skims across the surface of the disk as the appropriate sector spins by. Each disk has a directory that tells the read-write head which track and sector it should access to transfer the necessary information in the proper sequence.

There are two different kinds of tracks on every disk: system tracks and file tracks. System tracks are reserved for part of your operating system, and they are usually the two or three outermost tracks on the disk. File tracks are reserved for files, and they are the inner tracks on the disk. (Upcoming text will explain the concepts behind operating systems and files.)

NOTE: Before data can be written on a disk, the surface of the disk must be prepared. Disk preparation is performed by a program called "FORMAT," which is stored on your Distribution Disk. FORMAT prepares the disk surface by dividing it into tracks and sectors. The procedures for constructing backups and customizing the operating system will show you how to prepare disks using FORMAT.

Floppy Disks

A *floppy disk* is a circular sheet of mylar plastic with a magnetic oxide on its surface and a square plastic cover.

Floppy disks, and the data stored on them, are fragile. Therefore, you should adhere to the following precautions to ensure that disks and stored data are not damaged.

Floppy Disk Handling Precautions

- When you are holding the disk, touch only the protective square disk cover. Do not touch the brownish disk surface that shows through the read-write access slots in the disk cover.
- Keep the disk in the protective paper envelope whenever it is not within a disk drive.
- Do not allow dust, ashes, liquid, or any other foreign material to contact the disk surface.
- Keep the disk away from electric motors, appliances, telephones, etc., as these devices contain magnets that could alter the magnetic impressions on the disk.

- Never put a disk into a drive unit before turning on hardware equipment; and never leave a disk in a drive unit while the power is being turned off. Sudden fluctuations in the power supply to your hardware environment could cause the read-write head to write on the disk surface, and destroy stored information.

- Do not expose disks to temperatures above 125 degrees Fahrenheit (52 Centigrade), or temperatures below 40 degrees Fahrenheit (4 Centigrade).

- Never press a ball-point pen or a pencil directly against the cover of a disk. Instead, mark disk labels before attaching them to the disk cover, or mark them using a felt-tip pen while they are on the disk cover.

- Do not allow the disk or its cover to be bent, creased, or torn.

- Do not attach paper clips to the disk.

Write-Protecting and Write-Enabling Floppy Disks

You can mechanically prevent or allow the writing or erasing of information to or from your disks by covering or uncovering the notch in the disk cover with specially-provided tabs. The way that you use these tabs depends on the size of the disk.

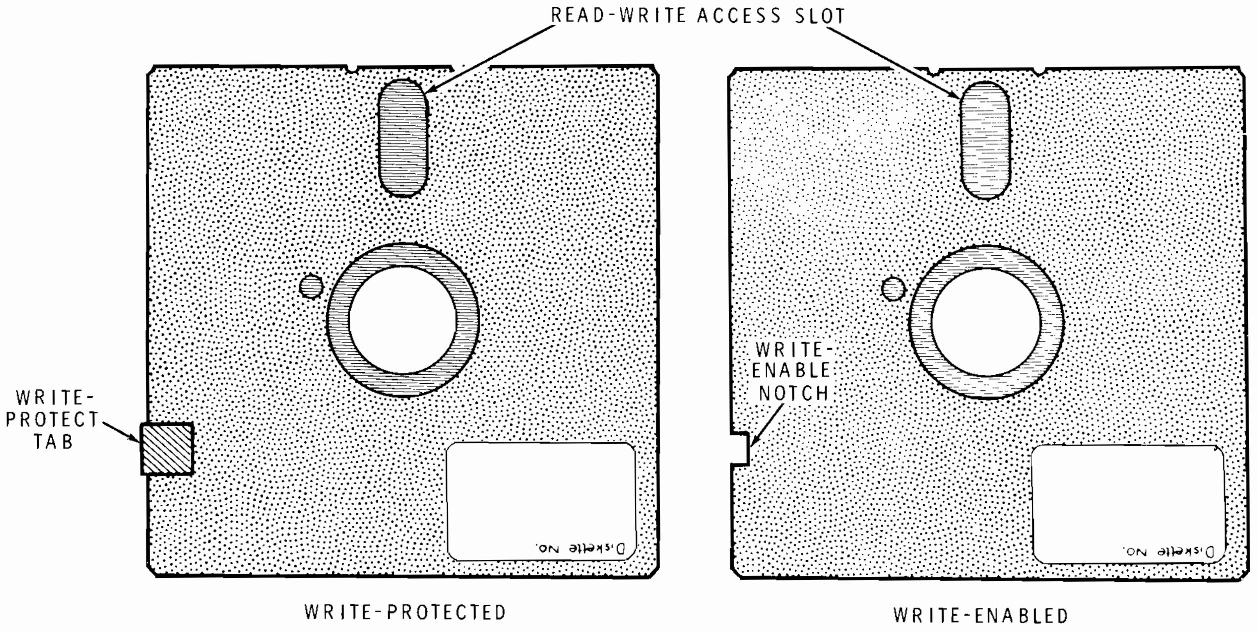


Figure 1.1 5.25-inch Floppy Disks

With 5.25-inch disks, the notch is covered to *prevent* you from writing to or erasing from the disk. Therefore, by putting the tab on a 5.25-inch disk, you are "write protecting" the disk. A 5.25-inch disk with a notch that is not covered can be written to or erased from. Therefore, a 5.25-inch disk with an uncovered notch is "write-enabled." Figure 1.1 illustrates this distinction.

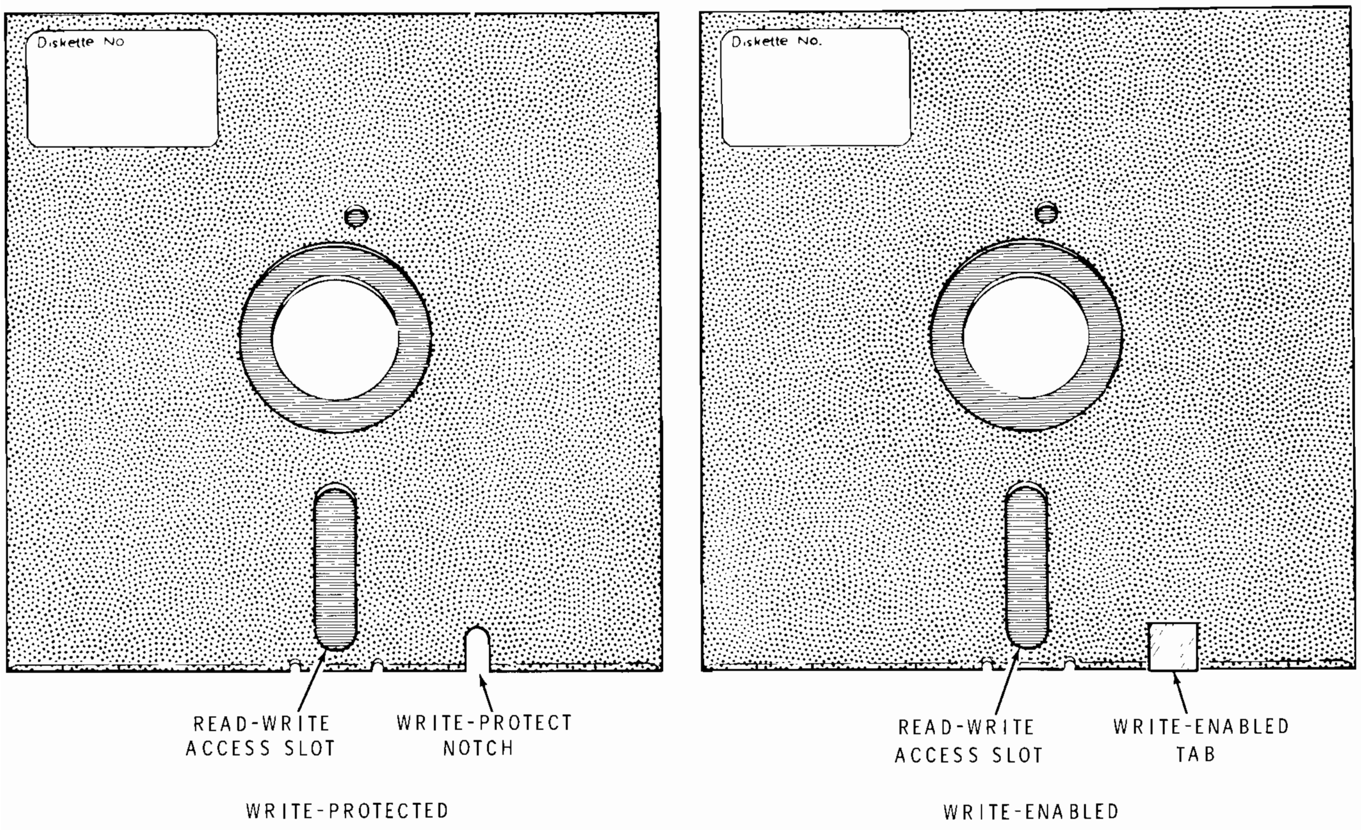


Figure 1.2 8-inch Floppy Disks

With 8-inch disks, the notch is covered to *allow* you to write to or erase from the disk. Therefore, by putting the tab on an 8-inch disk, you are "write enabling" the disk. An 8-inch disk with a notch that is not covered can not be written to or erased from. Therefore, an 8-inch disk with an uncovered notch is "write-protected." Figure 1.2 illustrates this distinction.

NOTE: Whether a disk is write-protected or write-enabled, you can usually read or copy data from it.

Inserting Floppy Disks

To ensure that data stored on your floppy disks can be safely and efficiently accessed, the disks must be inserted into drives carefully and correctly.

When you insert a floppy disk into a disk drive, point the oblong holes in the square plastic cover towards the back of the drive. A label is usually affixed to one side of the plastic disk cover, and this label should face upward on the right side as the disk is inserted into the drive. When the disk is fully inserted in the drive slot, close the drive latch.

Figure 1.3 illustrates the proper technique for inserting floppy disks into some of the drives that Heath/Zenith offers.

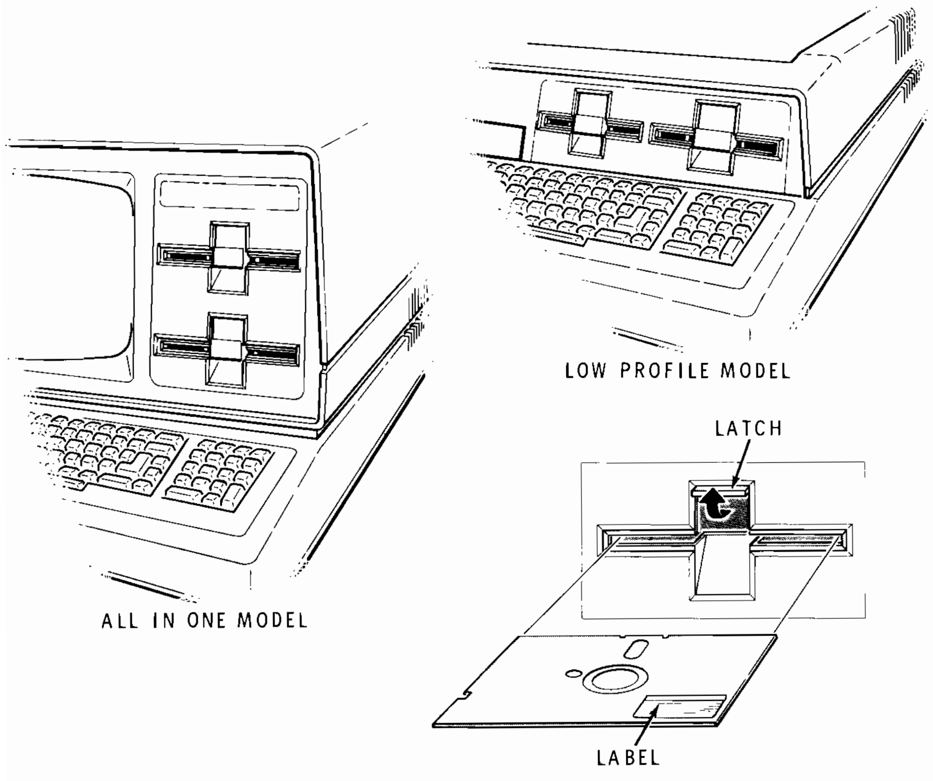


Figure 1.3 Inserting Floppy Disks

The Operating System

An *operating system* is a computer program that controls both the components of your hardware environment and subordinate programs ("application programs") that perform specific tasks.

It provides a vital link between your keyboard and your application programs, and between your application programs and your peripheral hardware. Thus, it is essential that you use an operating system whenever you use an application program in your microcomputer.

The way that you use the CP/M Operating System is to copy an image of the system from a disk and put it inside the memory of your microcomputer. This activity (called "booting up," or "cold booting") occurs automatically when you power up the computer and insert a disk containing the CP/M Operating System.

CP/M can perform several useful functions when it is alone in your computer's memory, and it also has "hollow" areas into which you can load application programs as you need them.

Operating System Components

The Heath/Zenith CP/M Operating System is divided into two software components: the "system kernel" and the "BIOS file." A disk that contains both of these components is "bootable," which means that it contains an image of the operating system that can be inserted into the computer and used.

The System Kernel

The *system kernel* is a set of programs that reside on the reserved system tracks of a disk. A special data transfer utility (called LDCOPY) is used to copy the system kernel from one disk to another. The system kernel manages files, translates the commands you enter at the keyboard, and performs other functions that do not depend upon specific hardware characteristics.

The BIOS Files

BIOS stands for Basic Input/Output System, which is the part of the operating system that enables CP/M to work in Heath/Zenith hardware. The BIOS used with this CP/M release is a file called CPM.SYS. The BIOS file can be manipulated by the same methods used to manipulate other files that are stored on the disk.

Operating System Requirements

To use the operating system, you must transfer a copy of it from a disk to the computer by performing an activity called "booting up" or "cold booting." Both the system kernel and the BIOS file *must* be together on a disk for that disk to be usable for booting up.

To protect your software investment, we strongly suggest that you make backup copies of your CP/M distribution software before using it for any practical applications. Once you have made backup copies, your CP/M software will be ready for use in Heath/Zenith hardware. "Software Preparation Procedures" shows you how to start up and back up your CP/M software.

The CP/M Operating System on your distribution software is ready to accommodate a standard configuration of hardware devices. If your hardware configuration differs from these standards, you must adjust the system before it will accommodate all of your devices. Your distribution software includes utilities to help you adjust the system for your devices.

Your CP/M-86 distribution software contains a serial number to prevent the mixing of this software with software from other CP/M distribution packages. Therefore, do not try to use this CP/M system with other CP/M systems or utilities, and do not try to use these utilities with other CP/M systems.

Application Programs

An *application program* performs specific user functions by passing data to the system and reading and processing data from the system.

Application programs are stored on the file tracks of a disk in units of data called "files." An application program might consist of several files that automatically access each other under certain circumstances. Whenever an application program file is needed to perform a specific task, an image of this file is copied from the disk and inserted into computer memory, where the CP/M Operating System has a reserved "hollow" area for it.

After the application program files have served their purpose, CP/M moves them aside and either reserves its hollow memory areas for new application programs or executes some of the programs within the operating system itself.

An important feature of CP/M-86 is that it supports both 16-bit and 8-bit software. This feature of CP/M-86 permits you to use application programs designed to run within either 16-bit or 8-bit CP/M systems. Thus, the number of application programs you may run under CP/M-86 is greater. In addition, if you are upgrading your system to CP/M-86 from one of Heath/Zenith's earlier versions of 8-bit CP/M, the 8-bit support provided by CP/M-86 will make this process more convenient. A more detailed explanation of how the support design works is included later in this chapter.

Utilities

Your distribution disk contains several application programs, that are often referred to as *transient commands* or *utilities* because they provide a number of routine functions that will help you take maximum advantage of your computer system. These programs, stored on the distribution disk, are identified by names that end with the file extension "CMD." This extension indicates that these files are valid commands that can execute under CP/M.

Application Program Requirements

It is important to remember that an application program cannot produce the results you desire without the presence of an operating system. Therefore, you must always load the CP/M operating system into your microcomputer before you can use any application program. (Start up Procedure in the Introduction will show you how to put CP/M in your computer.)

Furthermore, the CP/M Operating System must be capable of controlling all of your hardware devices before any application program can use these devices. Sometimes you must adjust the system (as described in Backup Procedures) so it can control your hardware devices.

Each application program consists of at least one file with a .CMD file name extension. Some application programs also consist of additional files. All of the files included in an application program must be present on a disk within a drive whenever you wish to use the application program.

CP/M Concepts

CP/M Concepts explains some of the basic properties of the CP/M Operating System and some of the conventions you must follow when using the system. After reading Microcomputer Concepts and CP/M Concepts you should be ready to perform the Software Preparation Procedures.

Specifically, this text shows you how CP/M stores data in units called files, how CP/M accesses these files through disk drives, and how you should issue commands to the CP/M Operating System.

Files

The CP/M Operating System enables you to create, analyze, and manipulate data by storing this data in units called "files." These files are stored on the surface of a disk and given names that conform to CP/M file naming conventions.

When you issue a command that refers to a particular file by name, CP/M goes to the appropriate disk, makes a copy of the file, and puts the copy into one of the hollow areas in computer memory.

CP/M File Naming Conventions

A file name consists of two parts: the primary name and the extension. The primary name has between one and eight characters, and is essential in all file names. The extension can have between one and three characters, or it can be omitted entirely. The primary name and the extension are separated by a period (.), in the following form:

```
{primary name}.{extension}
```

The characters used in the primary name and extension can be any character on the keyboard except the following special characters:

```
+ > . , ; : = ? * [ ]
```

The following example file names are valid because they conform to CP/M file name conventions:

```
memo.doc   CPM.SYS   PROGRAM.ASM   4/27/81.TXT

FORMAT.COM   FILE#1   33%-RATE.DAT   WSMSGS.OVR
```

Although the extension is optional, you'll probably find it useful to give your files extensions that somehow describe the type or purpose of the file you are naming. The following list shows several extensions that are applied to files used for a specific purpose:

EXTENSION	FILE PURPOSE
A86	Assembler source file
BAK	Backup file
BAS	BASIC source file
CMD	Command file (executable under CP/M-86)
COM	8080/8085 Command File
H86	Intel HEX object file
INT	Intermediate BASIC file (for BASIC-E,CBASIC)
PRN or LST	Print file of program listing
SUB	A list of commands to be executed with SUBMIT
SYM	Symbol table file
$$$	Temporary work file

Referencing Several Files at Once (Wild Card File Names)

Many of the commands that you will issue refer to files by name. But when you want to issue the same command for several files with similar names, it is often more convenient to enter a "wild card file name" with the command.

A *wild card file name* represents several file names—much like a "joker" in a deck of cards can stand for any card in the deck. (Wild card file names are sometimes called "ambiguous file names.") A wild card file name contains either asterisks (*), or question marks (?), or both.

An asterisk (*) is entered in place of the entire primary name and/or extension of a file, as shown in the following example:

```
16MAY82.*
```

In this example, the asterisk replaces the extension so that this wild card refers to files on the disk with the primary name "16MAY82" and *any* extension.

In the following example, the wild card asterisk takes the place of the primary name:

```
*.CMD
```

Therefore, all files with the extension "CMD" and *any* primary name would be referenced.

The wild card *.* stands for *any* file on the disk. (Actually, there are exceptional circumstances under which all of the files on a disk cannot be referenced at one time, but these circumstances will be explained in later text.)

Question marks (?) can be used in a wild card file name to take the place of single characters at fixed file name positions. For instance, the wild card

```
JOB?.H86
```

would refer to any file that has the extension "H86," a primary name with the characters "J," "O," "B," and one or fewer additional characters. Hence the files "JOB0.H86," "JOB1.H86," and "JOBY.H86" are just a few of the files that could be referenced by such a wild card (if these files existed on the disk).

You can use any number of question marks in a wild card (up to 11) and the actual file names referenced will be those with the same characters as those that are explicitly stated in the wild card and any characters in the place of the question mark characters.

Thus the wild card ????????.CMD will reference any file on the disk with the "CMD" extension—just like the *.CMD wild card.

Disk Drives

A disk drive is a device that transfers data to and from disk storage media.

Drive Names

To allow you to refer to disks and files within your disk drives, the CP/M Operating System recognizes each drive in your hardware environment by a distinct *drive name*. A drive name consists of a letter of the alphabet in the range A through P and a colon (:).

Example drive names are A: B: C: and D:

Default Drive

The *default drive* is the drive to which the system will refer unless you specifically tell the system to refer to a different drive. The default drive is also the drive named in the system prompt, which is displayed when the CP/M system is in control. (Drive A is always the default drive when you boot up, as shown by the A> system prompt.)

CP/M Concepts

You can execute an application program that is stored as a file in the default drive by typing the primary file name of the application program file (the part without the "CMD" extension) in response to the system prompt, as shown:

```
A>{primary name} RETURN
```

Where A> is the system prompt; and

where {primary name} is the primary name of the file that you want to execute. For this command to be valid, the file must reside on a disk in default drive A and have a "CMD" extension.

If the file represented by {primary name} does not reside on default drive A, then the system will display a message repeating your unfound entry with a question mark, as shown:

```
A>{primary name} RETURN
{primary name}?
```

This kind of error message will also occur if you use improper syntax in your entry or misspell your entry.

Changing the Default Drive

You can change the default drive by typing the name of another drive and pressing RETURN at the A> prompt, as shown:

```
A>B: RETURN
```

Such an entry will produce a new system prompt, indicating that drive B: is now the default drive, as shown:

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

NOTE: Any drive that is changed to the default drive in this fashion must be a valid drive within your hardware environment, and it must contain a floppy disk prepared by the FORMAT program. When you switch default drives in this fashion, the CP/M system will assume that any application program you wish to execute should be found on a disk in the new default drive.

Accessing a Non-Default Drive

A *non-default drive* is a disk drive whose name is not displayed in the system prompt. For instance, if the A> system prompt is displayed (meaning A is the default drive), then your non-default drives are any valid drives with names "B" through "P."

When you want to execute an application program that resides on a disk in a non-default drive, type the name of the appropriate non-default drive and a : (colon) immediately before the program's primary name and before you press RETURN:

```
A>B:{primary name} RETURN
```

Where A> is the system prompt;

where B: is the name of the desired non-default drive; and

where {primary name} is the primary name of the file that you want to execute. The file must reside on a disk in non-default drive B, and have a "CMD" extension.

The CP/M system would respond to such an entry by "logging-in" disk B to get the application program indicated by {primary name}, inserting an image of this program into computer memory, and executing the program.

Switching Floppy Disks Between Drives

When you reference a floppy disk drive (by entering a command at a default drive, by changing default drives, or by logging in a non-default drive with a command), CP/M remembers some of the characteristics of the disk in the referenced drives. Switching disks between drives can cause problems unless you tell the system to forget the old characteristics.

You can make the system forget about old disk characteristics by performing a "warm boot" (by holding down the CTRL key and pressing the C). This entry is often abbreviated as CTRL-C, and it is must be entered in response to the A> system prompt. It tells the operating system to forget what it knows about the disks that were logged in the drives and to log in (or, look at) the new disks. Then it redisplays the system prompt.

We suggest that you perform a warm boot whenever you remove a disk from a drive and replace it with another disk (unless the system or an application program prompts you to insert a different disk).

Using CP/M With One Drive

When you enter a valid command that requires more disks than you have physical drives, CP/M displays prompts that instruct you to insert the required disks into the physical drive at the appropriate time. The prompts appear in the form:

```
PUT DISK x IN DRIVE y: AND PRESS RETURN
```

Where x is the logical name that is assigned to a particular disk for the duration of the operation; and

where y: is the physical name given to the single drive, which is temporarily set to read data from or write data to a particular disk.

When such a prompt appears, you should remove the disk that is already in the drive, insert the disk that the prompt indicates, and press RETURN. Execution of the program will resume until the drive needs to write data to or read data from a different disk. Then a similar prompt will appear, requesting that you insert a different disk into the drive.

You do not have to perform a warm boot after switching disks in this manner. However, you must keep track of which disk is which. If you insert the wrong disk, the required data will not be found and you might have to restart the operation.

Commands

In general, a command is a program that can help you to create, change, analyze, or move data. Commands are entered in response to a "system prompt."

A system prompt consists of the letter for the default drive and the greater than (>) character. When you start up CP/M, the system prompt is displayed on your screen, as shown:

```
A>
```

The system prompt tells you that CP/M is ready to receive a command in the form of a "command line."

Command Lines

A command line is the form of response you make to the system prompt to bring up, or "enter," a command. A command line usually consists of three components: the "function," the "argument(s)," and the RETURN. The function is entered first, and it indicates the activity that will be performed. The first argument is entered one space after the function. The argument indicates what data (files, systems, disks, drives, etc.) the function's activity should be performed upon. Each argument that follows is separated by one space from the preceding one. After entering the function and argument, you must press RETURN to tell CP/M that the entire command line is ready for execution. A command line can contain a maximum of 127 characters.

You will enter command lines in the following form:

```
A>{function} {argument} {argument}... RETURN
```

Where A> is the system prompt;

where {function} is mandatory for all commands;

where {argument} {argument}... is optional for some commands; and

where *RETURN* is mandatory for all commands.

Always separate the command line function and the command line argument with one space. Furthermore, any command entered at a prompt that includes the ">" character must end with a RETURN. However, commands themselves often display prompts as well. And when such a prompt ends with a colon (:), a RETURN is usually not required for CP/M commands and utilities.

CP/M Concepts

NOTE: In this text, when you are to press the RETURN key the instruction will be listed as **RETURN**.

There are two kinds of commands that can be executed in a CP/M operating environment: Resident Commands and Transient Commands.

Resident Commands

Resident commands reside within the CP/M Operating System. Therefore, CP/M doesn't have to refer to a disk to know how to execute these commands—although the commands themselves might affect data that are stored on a disk.

The CP/M Operating System contains the following resident commands:

DIR	Displays the names of files without a SYS attribute that reside on a disk.
DIRS	Displays the names of files with a SYS attribute that reside on a disk.
ERA	Erases specified files from a disk.
REN	Renames a specified file on a disk.
TYPE	Displays the contents of a file on the terminal.
USER	Enables you to divide the space on a disk into separate areas for different users, to display the current user assignments, and to change those assignments.

This list shows only the command line function of the resident commands. See the "User's Guide" for a comprehensive explanation of the arguments used when these commands are entered.

Transient Commands

Transient commands are application programs that are supplied with the CP/M Operating System on your CP/M distribution software. These application programs help you to manipulate the operating system and to perform several other useful microcomputer activities.

These commands (also known as "utilities") are stored on the disk as files with the "CMD" extension. When you issue a command that makes reference to one of these files, CP/M copies an image of this file from the appropriate disk, puts this image into one of CP/M's hollow areas inside the computer memory, and begins execution of the transient command.

BIOS Source Files

One of your Distribution Disks contains the source code files used to produce CPM.SYS. These files are supplied for the benefit of assembly language system programmers, and they will not be needed by most customers. Many of the files on this disk are written in 8086 assembly language and require ASM86 for assembly.

Support of 8-Bit Software

Although a 16-bit operating system, CP/M-86 is designed to also support 8-bit software. Briefly described, this feature works by first looking for a 16-bit version (indicated by the filename extension .CMD) of the program whose name has been entered at the system prompt. If a 16-bit version of the program is not found, the system then searches for an 8-bit version (indicated by the filename extension .COM).

In more detail, the search for and execution of the correct program proceeds in the following steps:

1. A filename is entered at the system prompt (e.g., `A>8-BIT` or `A>B:8-BIT`).

2. The system searches for an internal CCP command (such as the resident commands REN or ERA); if the system does not find one that matches the filename,

3. the system searches for a file xx.CMD (e.g.8-BIT.CMD) in the current user; if the system does not find one that matches the filename,

4. the system searches for a file xx.CMD (e.g., 8-BIT.CMD) with a system attribute in User 0.

If the system still does not find the correct file, it ends it search for a 16-bit version of the program and begins to search for an 8-bit version.

5. The system searches for a file xx.COM (e.g., 8-BIT.COM) in the current user; if it does not find one that matches the filename,

6. the system searches for a file xx.COM (e.g., 8-BIT.COM) with a system attribute in User 0.

When a file xx.COM is found, the program is loaded into a free 64k bank of memory and the 8085 processor is started. An interface is automatically provided between the 8-bit program and CP/M-86. The program is then executed.

While CP/M-86 supports many 8-bit programs, there are a few restrictions you should be aware of if you plan to use the system's dual capacity on a regular basis.

1. At program termination, there is no automatic reset of the BDOS disk subsystem. This means that you cannot, for example, switch disks in a drive without pressing CTRL-C to perform a warm boot.

2. 8-bit programs that modify the operating system and stay resident (e.g., XSUB and DESPOOL) will not function.

3. BDOS and BIOS calls that return the address of tables, return a pointer to a copy of the table. Three tables are supported: 1) disk header table, 2) disk parameter table, 3) allocation vector. You should note, however, that only one copy area is provided for each type of table.

4. Only the standard Digital Research BDOS and BIOS entries are supported. This does not include any Heath/Zenith extensions.

Command Line Entry

CP/M is very precise in the form it expects command lines to have. You must spell all components of a command line correctly and include the names of non-default disk drives whenever a referenced file is not on the default disk. If you don't, CP/M will not be able to execute your command and will respond by redisplaying the invalid command line with a question mark (?). However, CP/M does allow some flexibility in the way you may respond to the system prompt, as the following special entry explanations show.

Command Editing Keys

NOTE: In this text, "CTRL" followed by a hyphen and a letter (CTRL-S, CTRL-C, CTRL-P for example) indicates that you should hold down the key marked CTRL (control key) while pressing the key marked with the letter.

The following list explains the single keys and combinations of keys that you can press to edit a command line before submitting it to CP/M for execution.

DELETE	Removes the previous character typed from the command line. Depending on how your operating system is adjusted, the removed characters might be echoed (repeated in reverse) on the screen, or erased from the display.
BACK SPACE	Removes the previous typed character. Also removes any deleted characters that were echoed in the line immediately to the left of the cursor.
CTRL-H	Same as BACK SPACE.
CTRL-X	Removes all characters typed in the command line, as if you used BACK SPACE all the way to the beginning of the line.
CTRL-U	Effectively removes all characters typed in the command line and allows you to try again on the line beneath the old line. It leaves the display of the old command line on the screen, and displays the "#" character at the end of this old line to label it as a nullified entry.
CTRL-R	Redisplays the edited version of a command line below the "scratch pad" version of the line without any of the deleted characters that might have been echoed in the line. Also displays the # character at the end of the "scratch pad" version.

Command Entry and Display Keys

The following list explains the single keys and combinations of keys that you can press to end a command line and submit it to CP/M for execution.

RETURN	Ends the command line, sends the command to the system for execution, and displays nothing on the screen. After execution of the command, CP/M redisplays the system prompt. In this manual, the RETURN will often be used in examples and instructions to remind you to make this entry at the end of a command line.
LINE FEED	Same as RETURN.
CTRL-J	Same as RETURN.
CTRL-M	Same as RETURN.
CTRL-E	Enables you to see the entire display of a command line that is longer than your screen is wide. When you type this entry, the remaining portion of your command line will be displayed at the left-hand end of the next screen line. This entry will not send your command line to the system for execution (as a RETURN entry would). It is not essential that you enter CTRL-E when typing a command line that exceeds console display range because CP/M will process your command line even if it does not fit on one screen line.

Even if you type CTRL-E entries, a command line cannot exceed 127 characters in length. If you type a 128th character in a command line, CP/M will automatically interpret it as a carriage return and try to execute your command line based upon the first 127 characters.

The following entries enable you to enter comments that CP/M will ignore:

;	(semicolon) Enables you to enter comments not intended for execution without receiving error feedback from CP/M. Comments can consist of any characters you wish, typed after the ";" entry, and followed by a RETURN or "CTRL-U" or "CTRL-X."

:	(colon) Same as ; except that you should not begin a comment line with two consecutive colons.

Cursor Movement Keys

The following entries enable you to rapidly skip several spaces in a command line or comment:

TAB	As with a regular typewriter, this key enables you to advance several spaces without pressing the space bar several times. It skips to the eighth column of the console display range or to some column numbered by a multiple of eight. Hence if you enter a TAB at the beginning of a command line, you will skip six columns (because the system prompt takes up two columns). If you immediately enter another TAB, you will skip an additional eight columns, and so forth.
CTRL-I	Same as TAB.

Command Execution Keys

The following list explains the single keys and combinations of keys that you can press to change the way in which CP/M executes your command line:

CTRL-S	Interrupts the display of data on the screen when it is pressed once. It allows CP/M to resume data display when pressed a second time. This entry is useful when data scroll by too quickly for you to read them. CP/M will resume data scroll when any other key is pressed or CTRL-S is pressed a second time.
CTRL-P	This entry after the system prompt causes CP/M to send most of the output it displays to the list device (LST:) at the same time. (The list device is usually a printer.) Pressing CTRL-P a second time will stop the display to the list device. This entry is useful when you want to record on paper the displays that appear during the execution of a command.
	NOTE: CTRL-P should not be entered unless your printer (or other LST: device) is connected, turned on, and properly configured (see CONFIGUR in Part I).

Software Preparation Procedures

Following are procedures for preparing your CP/M software so that it works efficiently with your hardware:

- **Startup Procedure** Helps you to prepare your microcomputer hardware for use and to load the CP/M Operating System into your microcomputer.

- **Backup Procedures** Helps you to copy and customize your CP/M distribution software onto backup and working disks to protect your software investment.

Startup Procedure

This procedure will explain the sequence of steps necessary for loading and running CP/M. This sequence includes the preparation of your hardware devices, the insertion of a bootable disk into the appropriate drive, and the movement of a copy of the CP/M Operating System from a disk into your microcomputer's memory.

The most significant step in this sequence is the movement of CP/M from a disk into the microcomputer's memory. This step is known as "booting up" or "cold booting." You will perform this step at least once each time you use CP/M in your microcomputer. Once loaded into memory, CP/M can control an application program or perform one of the many tasks within its own repertoire.

This procedure consists of several steps to help you start up CP/M in your Z-100 microcomputer. After you have unpacked your computer and connected any peripheral device to it according to the procedures given in the Z-100 Series User's Manual, proceed with the following steps.

NOTE: Your Z-100 computer is equipped with an automatic bootup feature. When your Z-100 is shipped, this feature is set to boot up immediately when the computer is powered up or after it is reset. Therefore, a manual bootstrap command is not necessary during the startup procedure. For more information about booting up see the Z-100 User's Manual.

1. Make certain that the power cords and transmission cables to all of your hardware devices are securely plugged into the proper receptacles. (Refer to the manual of each hardware device for assistance.)

2. Turn on the power switches to all of your hardware devices. Within a few seconds, the red light on your drive will glow.

3. Place CP/M Distribution Disk I into the left-hand drive (if you have a Low-Profile Z-100) or into the upper drive (if you have an All-in-One Z-100), and close the drive door.

Within a few more seconds, a CP/M-86 identification message and system prompt will be displayed in a form similar to the following:

```
CP/M-86 VERSION x.xx 07/07/82

A>
```

You have successfully booted the CP/M-86 Operating System. The `A>` system prompt indicates that the system is waiting for you to enter a command. Proceed to "Backup Procedures."

NOTE: If a message and prompt in this form do not appear, take one of the following steps:

- If nothing appears on the screen, hold down the **CTRL** key while pressing the **RESET** key to reset the computer. The bootup procedure should resume automatically and cause display of the message and prompt.

- If a pointing finger prompt appears on the upper left-hand corner of the screen, then the automatic bootup attempt was aborted. You must now enter a manual bootup command, as explained in the Z-100 User's Manual. (In general, manual bootup can be performed by pressing the **B** key and then **RETURN**.)

- If these steps continually fail to produce any screen display, refer to the section entitled "In Case of Difficulty" in the Z-100 User's Manual.

Backup Procedures

Before beginning regular use of your CP/M system, you should protect your software investment by making at least two exact copies of the distribution disk set. After this is done, the distribution disks should be permanently stored in a secure location.

Software Preparation Procedures

One copy of the distribution set will become your master set. It should be kept in a safe but accessible location and should be used to create all future working copies of the CP/M system.

The second copy of the distribution set will become your working copy. It should be stored near your computer system and should be used for routine daily operations. Since floppy disks can be damaged during use and do eventually wear out, you should expect to create new working disks from your master set from time to time.

These backup procedures can be used whether your computer has a single floppy disk drive or multiple floppy drives. You will need six blank disks that are not write-protected. Follow these steps:

1. Carefully read the instructions for using the FORMAT utility in Part I: "Z-100 Utilities." Following these instructions, FORMAT all six of the blank disks using the double-sided option.

2. Carefully read the instructions for using the COPYDISK utility in Part II, the "User's Guide." Following these instructions, copy distribution disk I to two of the blank disks. Then copy distribution disk II to the two disks. Finally, copy distribution disk III to the remaining two disks.

You will probably find it desirable to transfer the CP/M-86 Operating System and/or certain utility programs to the disks containing your application software. This can be accomplished using the LDCOPY utility (see Part I) and the PIP utility (see Part II).

ASSIGN

The Utility that Assigns Winchester Partitions to Drives

The ASSIGN utility enables you to access data from Winchester disk partitions by assigning the partitions to drives, or by changing partition drive assignments. ASSIGN can also identify all of the partitions on a Winchester disk, and identify which of these partitions are currently assigned to a drive name.

Winchester Disk Partitions

A partition is an area on the surface of a Winchester disk that performs as though it were a floppy disk drive during most CP/M operations. Two partitions are already established on your Winchester disk when it is shipped from the factory. Both of these partitions can be made accessible at one time, provided they have been formatted with either CP/M-85 or CP/M-86.

Partitions are identified by a partition name that consists of one to 16 characters, and optionally an operating system name that consists of 10 characters.

Although one partition is accessible as soon as you boot up to the Winchester disk, you can access no other partition until you run the ASSIGN utility to assign the second partition to a drive name. If you boot on a floppy, both partitions must be assigned.

NOTE: If you have repartitioned your Winchester with the PART utility (see Winchester User's Supplement) to contain more than two partitions, you are still limited to having a maximum of two partitions accessible at one time.

Assigning Partitions = Inserting Disk

For most CP/M operations, you can think of a Winchester partition as if it were a floppy disk. However, you must also realize that the ASSIGN activity is similar to the fundamental activity of inserting a floppy disk into a drive.

You must *insert* a floppy disk into a drive before its data can be accessed. Likewise, you must *assign* a partition to a drive before its data can be accessed (unless you booted up to the partition).

Inquiry of Available Partitions

To find out the names of *all* of the partitions on the Winchester disk, enter the following command:

```
A>ASSIGN ? RETURN
```

ASSIGN will respond with a display in the following form:

```
PARTITION NAME    OS NAME    SIZE
--------------    -------    ----
CPM-BUSINESS      :CPM       3200k
CPM-WORDPROC      :CPM       2860k
CPM-PROGRAMS      :CPM       2020k
Z-DOS1            :Z-DOS     1600k
```

Where the characters listed beneath "PARTITION NAME" are the partition names that were designated for each partition when the PART or PREP utility was run;

where the characters listed beneath "OS NAME" are the operating system names that were designated for each partition when the PART or PREP utility was run; and

where the numbers listed beneath "SIZE" show the total capacity (in kilobytes) of the partition named to the left. This capacity includes usable file space and space that might be occupied by an operating system or by bad sectors. (To determine the amount of usable file space that is free on a partition for future files, use ASSIGN to assign that partition to a drive and run the STAT utility.)

NOTE: The partitions listed in this display may or may not be assigned.

Inquiry of Current Assignments

To find out the names of the partitions on the Winchester disk that are currently accessible through a drive name, enter the following command:

```
A>ASSIGN RETURN
```

The ASSIGN utility will display equations to indicate which drive names can be used to access which partition names, as in the following example:

```
A: = CPM-WORDPROC;CPM
B: = CPM-PROGRAMS;CPM
```

When you boot up to a Winchester disk partition, that partition is automatically assigned to drive A, and a different partition can be assigned to drive B using the ASSIGN utility.

Assigning Partitions to Drive Names

To assign a partition to a drive, enter a command in the following form:

A>ASSIGN **{d}**: =**{partition name}**;**{system name}** ***RETURN***

Where **{d}** is the drive to which you wish to assign the partition;

where **{partition name}** is the partition name of the established partition you wish to assign. Partition names contain 1-16 ASCII characters excluding the semicolon, space, and tab. This entry is mandatory in a drive partition assignment command;

where ; is a semicolon that must be used as a separation character only when you also specify a system name; and

where **{system name}** is the operating system name of the established partition you wish to assign. Operating system names contain 1-10 ASCII characters excluding the semicolon, space, and tab. The operating system name is an optional entry. You only need to specify it if the specified **{partition name}** matches that of another partition on the disk.

NOTE: The **{partition name}** and **{system name}** are both components of the "bootstring," which must sometimes be specified in bootup commands.

For example, you can assign the partition named "CPM-PROGRAMS;CPM" to drive B with the following command:

A>ASSIGN **B:=CPM-PROGRAMS;CPM** ***RETURN***

Furthermore, if you have only one partition on the disk with the partition name "CPM-PROGRAMS," you can omit specification of the operating system name in the command, as shown:

A>ASSIGN **B:=CPM-PROGRAMS** ***RETURN***

Valid Assignment Commands

A maximum of two Winchester disk partitions can be assigned to drive names at one time, regardless of how many partitions have been established on the disk.

The drives to which you can assign a partition vary, depending upon the type of media you use to boot up and the types of drives you have in your hardware environment. The following table shows which forms of partition assignment commands you can enter depending upon your bootup media and upon which kind of disk drive you might have. (The command forms indicated in this table use the variable {partition name};{system name} to represent the partition you wish to assign.)

Bootup Media	Additional Drives	Valid Assignment Commands
5.25-inch floppy disk	with any 8-inch floppy drives	`A>ASSIGN E:={partition name};{system name}` *RETURN* `A>ASSIGN F:={partition name};{system name}` *RETURN*
5.25-inch floppy disk	with no 8-inch floppy drives	`A>ASSIGN C:={partition name};{system name}` *RETURN* `A>ASSIGN D:={partition name};{system name}` *RETURN*
8-inch floppy disk	with any 5.25-inch floppy drives	`A>ASSIGN E:={partition name};{system name}` *RETURN* `A>ASSIGN F:={partition name};{system name}` *RETURN*
8-inch floppy disk	with no 5.25-inch floppy drives	`A>ASSIGN C:={partition name};{system name}` *RETURN* `A>ASSIGN D:={partition name};{system name}` *RETURN*
Winchester partition	not applicable	*`A>ASSIGN A:={partition name};{system name}` *RETURN* `A>ASSIGN B:={partition name};{system name}` *RETURN*

*When you boot up with a Winchester partition that contains the CP/M-86 Operating System, this partition is automatically assigned to drive A.

Assign Error Messages

```
BAD PARTITION NAME
```

EXPLANATION: This error occurs if you try to assign a valid drive name to an invalid partition name, or if you made a syntax error during command entry. To correct this, you must reboot and re-enter any assignments made.

```
BAD DRIVE NAME
```

EXPLANATION: This error occurs if you try to assign a drive name other than A or B to a Winchester disk partition. You must re-enter assignment commands.

```
PARTITION ALREADY IN USE
```

EXPLANATION: This error occurs if you try to assign a new drive name to a partition name that already has an assigned drive name. The assignment of a currently assigned partition must be removed before a new drive name can be assigned to that partition.

```
BAD OPERATING SYSTEM NAME
```

EXPLANATION: This error occurs if you try to assign a partition name with an invalid operating system name to a valid drive name. This assignment should be attempted again with an operating system name that helps to identify an existing partition.

```
INCORRECT VERSION OF BIOS
```

EXPLANATION: The operating system being used will not accommodate Winchester disk partitions.

```
DISK READ ERROR
```

EXPLANATION: The operating system failed in an attempt to read from the Reserved Winchester Area (see text on PREP). Use VERIFY utility. If this error message occurs again, contact Zenith Data Systems Technical Consultation for assistance.

BACKUP

The Utility that Facilitates File Copying from Working Disks and/or Partitions to Backup Storage Media

The BACKUP utility enables you to conveniently copy large quantities of files from disks or partitions.

BACKUP is especially beneficial to Winchester disk users, because Winchester disk partitions hold so many files.

Because BACKUP works with both floppy disks and Winchester partitions, this text refers to recording media as "partition/disk" or "disk/partition" wherever it would be practical for you to use either a floppy disk or a Winchester partition.

BACKUP's function is complementary to the function of the RESTORE utility, which can return backed up files to their original media.

NOTE: The disks that receive backed up files in a BACKUP operation must be formatted. Therefore, before using BACKUP, you should determine how many disks will be necessary to receive the backed up files. (The STAT utility can be useful in determining the size of the files you wish to back up.) Then you must use the FORMAT utility to prepare each of the necessary disks.

Backup Operation

When the BACKUP utility copies files from a source partition/disk, it temporarily concatenates them to be stored as a single "backup file" on the destination media. Thus the backup file is a long, continuous string of data including individual files that are joined by BACKUP and separated by RESTORE. The backup file also contains a directory that lists all of the individual files in the exact sequence they were copied.

However, the backup file might be much larger than the capacity of your destination media. For instance, if the source in a BACKUP operation is a large Winchester disk partition and the destination is a 5.25-inch floppy disk drive, then the backup file will likely overflow the capacity of a single floppy disk.

Therefore BACKUP is capable of storing parts of the backup file on more than one disk. BACKUP accomplishes multidisk storage by recording up to the absolute capacity of the distribution media, prompting you to insert another disk, and then continuing to copy from where it left off. With this capability, BACKUP can even divide an individual file within the backup file between two or more disks.

The backup file's directory keeps track of the number of disks that were necessary to receive the entire backup file. In this directory, each of the disks used is numbered as a "volume."

The disk volumes used to receive the backup file are collectively called the "backup disk set."

The BACKUP and RESTORE utilities also enable you to view several statistics, such as the names of the files within a backup file, by entering special command line options.

NOTE: If you are performing a BACKUP operation from a floppy disk to a Winchester disk, the "backup disk set" can consist of only one partition. You will not be able to back up files to more than one partition during this operation—even if more than one formatted CP/M partition is already assigned to a drive name. Therefore, if you wish to back up files to a Winchester disk partition, you should make certain that this partition has enough free space to accommodate all of the backup files. Use the STAT utility to determine the size of the files to be backed up and the amount of free space on the partition.

Starting Backup

You can use the BACKUP utility through either of the following methods: the Utility Prompt Method and the System Prompt Method. Both methods enable you to use optional switches that provide more control of what is backed up.

Utility Prompt Method

With this BACKUP method, you invoke the BACKUP utility from a disk by entering the command function at the system prompt and then entering the command argument at a prompt displayed by the BACKUP utility.

The first entry under this method is in the following form:

```
A>BACKUP RETURN
```

After an entry in this form, BACKUP will display a message and prompt in the following form:

```
                              BACKUP Version 1.1.00
                       Copyright (C) 1983 Zenith Data Systems
>
```

NOTE: The version number of the BACKUP utility (shown in this example as "1.1.00") might vary.

At the > BACKUP prompt, you should enter a command line argument in the following form:

```
>{d}:{destfile}={s}:{sourcefile},{s}:{sorcfile},... [{x};{x}:...] RETURN
```

Where > is the BACKUP utility prompt;

where {d} is the optional name of the drive that is to receive the copies being transferred. This specification is necessary only if this destination drive is not also the default drive;

where {destfile} is the primary name of the backup file, which you wish to contain all of the individual files copied in this operation;

where {s} is the optional name of the drive from which files are being copied. This specification is necessary only if this source drive is not also the default drive;

where {sorcfile} is the complete name of each file, separated by commas, that you wish to back up from the source partition/disk. You can enter wild card file names (using the * and ? characters); and

where {x} is any of the optional single-letter options, separated by semicolons, that stucture the BACKUP operation.

NOTE: A space is allowed, but not necessary, between the source file specifications and the options.

When BACKUP has completed the specified operation, it will redisplay the > BACKUP prompt. You can continue entering command line arguments at BACKUP prompts indefinitely.

The BACKUP utility will display the names of each file that it copies in a vertical list, during the BACKUP operation.

When you wish to exit from the BACKUP utility to the CP/M system, press RETURN at a BACKUP prompt. Then CP/M will display the A> system prompt.

NOTE: Like all command lines entered through the CP/M system, the BACKUP command line can contain only 127 characters. If your command line is between 78 and 127 characters in length, you can keep the entire line visible on your video screen by pressing **CTRL-E** after the 79th character.

System Prompt Method

With this BACKUP method, you enter the command line function and the command line argument both at the system prompt, in the following form:

```
A>BACKUP {d}:{destfile}={s}:{sorcfile},{s}:{sorcfile},... [{x};{x};...] RETURN
```

Where BACKUP is the command line function, stored in the file BACKUP.COM on the default or logged partition/drive;

where {d} is the optional name of the drive that is to receive the copies being transferred. This specification is necessary only if this destination drive is not also the default drive;

where {destfile} is the primary name of the backup file that you want to contain all of the individual files copied in this operation;

where {s} is the optional name of the drive from which files are being copied. This specification is necessary only if this source drive is not also the default drive;

where {sorcfile} is the complete name of each file, separated by commas, that you wish to back up from the source partition/disk. You can enter wild card file names (using the * and ? characters); and

where {x} is any of the optional single letter options, separated by semicolons, that structure the BACKUP operation.

NOTE: A space must be entered between the BACKUP command line function and the drive and/or file name specifications. A space is allowed, but not necessary, between the source file specifications and the options.

During the BACKUP operation, the BACKUP utility will display the names of each file that it copies in a vertical list.

After BACKUP has completed the specified operation, CP/M will display the A> system prompt.

NOTE: Like all command lines entered through the CP/M system, the BACKUP command line can contain only 127 characters. If your command line is between 78 and 127 characters in length, you can keep the entire line visible on your video screen by entering **CTRL-E** after the 79th character.

Help Display Method

With this BACKUP method, you enter the command line function and a ? (question mark) at the system prompt, as shown:

```
A>BACKUP ? RETURN
```

This command will cause the display of messages that summarize the purpose, command line, and options of the BACKUP utility. The system prompt will appear below the BACKUP display.

This invocation method does not back up files. It is designed to provide you with a convenient quick reference to a few aspects of the BACKUP utility.

BACKUP Sources

A BACKUP command line can include one or more sources. When specifying sources, you must always specify a file, and sometimes a drive.

The source files specified in a BACKUP command line should be identified by complete file names, including the primary name (1-8 characters) and the extension (1-3 characters if used).

Whenever a specified source file does not reside on the default drive, a drive name should be specified in front of it. You can back up files from any valid drive during a BACKUP operation.

Wild card, or ambiguous, file names can be specified, using the * or ? wild card characters. (Wild card file names can be particularly useful when you have many files to copy, because a command line can contain a maximum of only 127 characters.)

Any number of source file names can be specified in a command line (as long as the limit of 127 characters per command is not exceeded). However, if two or more source file names are specified, they must be separated by commas.

NOTE: The number of source files that can be backed up in a single BACKUP operation is limited by the amount of space on the first disk/partition used to receive the backup volumes. This disk/partition must have enough free space to accommodate the entire backup file directory. The backup file directory is described in "Structure of Backup Files."

Source file specification is required in all BACKUP command lines except those entered with the [B] or [L] option. The purpose of the [B] option is to display characteristics of all of the master backup files on disk. The [L] option displays a list of the individual files within a backup file.

The source file specification

```
    *.ASC,TEST?,C:*.*
```

would cause copies to be made of all files on the default drive with an .ASC extension; all files on the default drive that are five characters long beginning with TEST and that have no extension (such as TEST0, TEST1, TESTS, TESTY, etc.); and all files on drive C.

To further demonstrate the characteristics of source file specifications:

```
    A:DEMO.* B:SYSTEM.COM E:82*.DOC ????.DAT
```

would cause copies to be made of all files on drive A with DEMO as primary name and any extension; the file SYSTEM.COM on drive B; all files on drive E that have a primary name beginning with 82 and that have a .DOC extension; and all files on the default drive that have a four-letter (or less) primary name with a .DAT extension.

NOTE: If any of the source files you specify are random files (as opposed to sequential files), these files might become larger when restored with the RESTORE utility.

BACKUP Destination

A BACKUP command line can include only one destination. When specifying this destination, you must always specify the primary name of a file and sometimes specify a drive.

The file specified as the destination will ultimately become the backup file that will contain all of the individual files that are copied.

The destination file specified in a BACKUP command line should be identified by a primary file name only (1-8 characters). The extension should not be specified, because BACKUP applies extensions to the backup file automatically. (BACKUP will apply the extension “000” to the first backup file volume, “001” to the second, “002” to the third, etc.)

Wild card file name characters (* or ?) cannot be used to specify a destination file.

Whenever a specified destination file does not reside on the default drive, a drive name should be specified in front of it. You can back up files to any valid drive during a BACKUP operation.

NOTE: However, if your destination drive is assigned a partition, you will only be able to create one backup file volume. If the files you wished to back up will not fit on this volume, then you will have to abort the BACKUP operation.

A destination file specification is required in all BACKUP command lines except those entered with the [B] option, for the sole purpose of displaying characteristics of all of the master backup files on the default disk.

Consider an example where the primary file name "C:QUITTIME" was entered as the destination in a command line. The ensuing BACKUP operation required that five disks (backup file volumes) be inserted into drive C to accommodate all of the source files.

The resulting set of destination disks (backup file volumes) would contain the following backup files:

```
QUITTIME.000
QUITTIME.001
QUITTIME.002
QUITTIME.003
QUITTIME.004
```

The file "QUITTIME.000" would be the master backup file volume, which contains the backup file directory.

BACKUP Options

The BACKUP utility enables you to structure any backup operation by specifying the following options in a command line:

B	Backup directory—displays a directory listing statistics about all master backup files.
D	Date stamping—applies today's date to the directory of individual files within a backup file.
E	Exception files—exclude exception files from backup operation.
L	List directory—list the directory of the backup files.
Q	Query each—query yes or no on each file before backing up.
S	System files—include system files in the backup.

U	User number—backup files from specified user area(s).
V	Verify—verify the file after backup operation.

Switches are always the last items specified in a BACKUP command line. They must be enclosed within square brackets and separated by semicolons when more than one is used.

If a switch requires specification of files, then the option letter should be separated from these file specifications by a colon. When more than one file specification is necessary within a single option, then the file specifications should be separated by commas.

Options are entered in the following form:

```
[{x};{x}:{details},{details},{details};{x};...]
```

Where {x} is an option letter;

where {details} are additional character strings (such as the date, file names, or user numbers) that must be specified with some options;

where the [ ] square brackets must enclose the options;

where the ; (semicolon) must separate multiple options;

where the , (comma) must separate the file names and/or user numbers used with some options; and

where the : (colon) must separate some option letters from accompanying {details}.

B—BACKUP Directory

The B option causes a directory display listing statistics about all of the master backup files (those backup files with the "000" extension) on a specified disk.

This directory lists the primary name of each master backup file, the number of volumes in the backup set that begins with each master, the number of files in each set of backup files, and the date of each backup operation.

Only the drive containing the master backup file(s) needs to be known in command lines entered for the sole purpose of producing a B option directory. Therefore, specify this drive in the command line unless it is the default drive.

You can use the B option in commands of the following form:

```
A>BACKUP {d}: [B] RETURN
```

Where BACKUP is the command line function, stored in the file BACKUP.COM on the default or logged partition/drive;

where {d} is the optional name of the drive that has received the copied files and stored them within backup file volumes. This specification is necessary only if this destination drive is not also the default drive; and

where [B] is the single-letter option that causes the display of master backup file features.

The B option may be used with both methods of invoking backup. For example, from the CP/M system prompt:

```
A>BACKUP C: [B] RETURN
```

or the same operation from the BACKUP utility prompt:

```
>C: [B] RETURN
```

The B directory listing would appear in the following form:

```
Name        Volumes    Files      Date
QUITTIME       3        117     04-27-83
SAVEME         1         48     10-24-82
STORAGE        5        231     02-30-84
```

D—Date Stamping

The date option enables you to specify the date for this BACKUP operation within the command line, without having to answer a date prompt. This date will appear in the displays caused by the B option. Specify the date option and date string in either of the following forms:

```
[D:{mm}-{dd}-{yy}]

[D:{mm}/{dd}/{yy}]
```

Where {mm} is a one- or two-digit entry for the month, within the range 1-12;

where {dd} is a one- or two-digit entry for the day of the month, within the range 1-31;

where {yy} is a one- to four-digit entry for the year, within the range 0-9999; and

where either - (hyphens) or / (slashes) can be used to separate month from day and day from year, although - (hyphen) will always appear in the dates of the display produced by the [B] option.

If you do not specify the D option, then BACKUP will prompt you to specify the date during each BACKUP session in which you specify both source and destination.

If you do specify the D option in a command in which both source and destination are specified, BACKUP should begin to copy individual files, displaying the drive name, file name, and user area number of each individual file as it is copied, in the following form:

```
E:FILENAM1.EXT from user 9
E:FILENAM2.EXT from user 9
E:FILENAM3.EXT from user 9
          .
          .
          .
E:FILENAMn.EXT from user 9
```

E—Exception Files

The BACKUP operation takes place for all of the files except that file given as an exception file. The file listed with the E option is then ignored during the operation. The E option is entered in the following form:

```
[E:{filespec},{filespec}]
```

Where E is the option;

where {filespec} represents files that are to be excluded from the operation. If a file being excluded does not reside on the default drive, then you must also specify the name of the drive that contains it. Ambiguous file names (with * or ? wild card characters) can be specified; and

where the , (comma) and : (colon) are required separation characters.

For example:

```
A>BACKUP BACK1=*.DAT [E:TEMPFILE.DAT] RETURN
```

would back up all files from the default drive that have a .DAT extension—except the file named TEMPFILE.DAT, which would be omitted.

L—List Directory

The L option causes BACKUP to give the internal directory of files that are contained within a specified backup file. The directory information would list the volume number at which each individual file starts and ends, the user area from which each individual file came, and the amount of disk space occupied by each file.

The L option is used for commands in the following form:

```
A>BACKUP {d}:{destfile} [L] RETURN
```

Where BACKUP is the command line function, stored in the file BACKUP.COM on the default or logged partition/drive;

where {d} is the optional name of the drive that has received the copies being transferred. This specification is necessary only if the destination drive is not also the default drive;

where {destfile} is the primary name of the backup file, which you wish to contain all of the individual files copied in this operation; and

where [L] is the single-letter option that causes the display of backup file directory features.

The L option may be used with both methods of invoking backup. For example, from the CP/M system prompt:

```
A>BACKUP BACK9 [L] RETURN
```

or the same operation from the BACKUP utility prompt:

```
>BACK9 [L] RETURN
```

The L directory listing will appear in the following form:

```
Filename          User    Start     End       Size in
                          Volume    Volume    Kilobytes
INDIVID1.DAT        0       1         1         3264
INDIVID2.DAT        0       1         1        19582
INDIVID1.DOC       15       2         3         7236
INDIVID2.DOC       15       3         3        22230

4 file(s) on 3 volume(s)
```

Q—Query Each

When the Q option is used, you are queried before each file is backed up, by a prompt in the following form:

```
Backup {x}:{sorcfile} from user {n} (Y/N) ?
```

Where {x} is the drive containing a source file that you specified for this operation;

where {sorcfile} is a source file that you specified for this operation; and

where {n} is the number (0-15) of the user area from which the source file could be backed up.

A prompt in this form is displayed for each file BACKUP encounters that matches the source specifications you entered.

For example, if the current user area is 0 and three .DOC files exist on source disk B in user area 0, and if the command to BACKUP was:

```
A>BACKUP C:BACK1=B:*.DOC [Q] RETURN
```

then BACKUP would ask:

```
Backup B:FILE1.DOC from user 0 (Y/N) ?
```

After you respond it would ask,

```
Backup B:FILE2.DOC from user 0 (Y/N) ?
```

and finally,

```
Backup B:FILE3.DOC from user 0 (Y/N) ?
```

In response to such a prompt, press **Y** or **y** to cause the file to be backed up. Press **N** or **n** to prevent the file from being backed up.

S—System Files

The S option allows BACKUP to back up files that have been set in the directory (by the STAT utility) as system files. System files will not be backed up merely because of this option, but only if this option is used in a command line where system files are specified as sources.

When the S option is not included in the command line, system files are ignored during the BACKUP operation—even if these files are specified in the command line.

U—User Number

The U option causes BACKUP to back up only the individual files within the specified user area(s). If you do not include a U option in the BACKUP command, then BACKUP will back up only the individual files from the current user area.

NOTE: CP/M enables you to divide each of your disks and partitions into 16 separate user access areas, numbered 0-15. When you boot up a partition or disk, you are working within user area 0 until you enter a USE command. See the "USER" text in the "User's Guide" of your CP/M-86 manual for more information on user areas.

Specify the user option in the following form:

```
[U:n1,n2,...]
```

Where `U` is the option letter;

where `:` (colon) must separate the option letter from the user number(s); and

where `n1` stands for the number (0-15) of one of the user areas from which BACKUP will copy individual files. If you specify more than one user area number, separate each with a comma.

You can specify as many as 16 user areas with a single U option. You can specify any user area(s) regardless of the user area currrently being used.

V—Verify

The V option causes BACKUP to verify all files copied. With V, BACKUP reads each source file after it is copied to make sure that the source and destination copies are identical.

During the operation, BACKUP displays messages in the following form:

```
E:FILENAM1.EXT from user 12
Verifying E:FILENAM1.EXT
E:FILENAM2.EXT from user 12
Verifying E:FILENAM2.EXT
E:FILENAM3.EXT from user 12
Verifying E:FILENAM3.EXT
          .
          .
          .
E:FILENAMn.EXT from user 12
Verifying E:FILENAMn.ext
```

Runtime Prompting

After you invoke BACKUP and enter a command to perform some operations, you will be prompted to enter the date and/or to insert various disks.

Date Prompt

Whenever you enter a BACKUP command in which both source and destination are specified and the D option is not specified, BACKUP will prompt you to enter today's date, so that the backup file directory can show the date on which the operation took place. (If you do not specify both destination and source in the command line, or if you do specify the D option, then no date prompt will appear.)

This date will appear after you have entered a command line, and it will appear in the following form:

```
Enter today's date:
```

You should respond to this prompt with an entry in the following form:

```
Enter today's date: {mm}-{dd}-{yy} RETURN
```

Where {mm} is a one- or two-digit entry for the month, within the range 1-12;

where {dd} is a one- or two-digit enty for the day of the month, within the range 1-31;

where {yy} is a one- to four-digit entry for the year, within the range 0-9999; and

where either - (hyphens) or / (slashes) can be used to separate month from day and day from year, although - (hyphen) will always appear in the dates of the display produced by the [B] option.

NOTE: If you enter BACKUP commands through the utility prompt method, you will be prompted to enter the date only once during the BACKUP session. The date that you enter at the prompt (displayed after the first command you enter with both source and destination) will be applied to any backup file copied until you exit from BACKUP.

After you have specified a date, BACKUP should begin to copy individual files, displaying the drive name and file name of each individual file as it is copied, in the following form:

```
E:FILENAM1.EXT from user 9
E:FILENAM2.EXT from user 9
E:FILENAM3.EXT from user 9
```

Disk Insertion Prompt

If a BACKUP operation requires more than one backup disk to accommodate all of the source files, then you will be prompted to insert formatted disks each time a previously inserted disk becomes full. BACKUP will keep track of the number and sequence of the disks you insert, and designate a "Volume" number for each disk in the set.

NOTE: If you are performing a BACKUP operation from a floppy disk to a Winchester disk partition, the Backup Disk Set can consist of only one partition. You will not be able to back up files to any other partition during this operation—even if another assigned formatted CP/M partition exists on the Winchester disk. Therefore, if you wish to back up files to a Winchester disk partition, you should make certain that this partition has enough free space to accommodate all of the backup files. Use the ASSIGN and STAT utilities beforehand to determine file size and free partition space.

When BACKUP has filled a backup disk to capacity, it will display a message in the following form:

```
Insert another disk in drive x for backup,
and hit RETURN when ready,
or hit any other key to abort.
```

Where `drive x` identifies the destination drive you specified in the BACKUP command.

If you are ready to continue the operation, insert a disk that has been formatted by the CP/M-86 FORMAT utility. This disk may also contain other files. Then you should close the drive latch and press the **RETURN** key.

If you do not wish to continue the operation, or if your destination medium is a Winchester disk partition, then press some key other than the RETURN key. The BACKUP operation will end, and a prompt will appear. (The BACKUP utility prompt will appear if you invoked BACKUP by the utility prompt method. The CP/M system prompt will appear if you invoked BACKUP by the system prompt method.)

Master Disk Insertion Prompt

The first disk in the backup set is known as "volume one," or the "master volume" of the backup file. This disk contains a file with the primary name that was specified in the command line as the destination, and with the extension "000."

As the specified source files for a BACKUP operation are being backed up, BACKUP accumulates a list of statistics, such as the names of the individual files it backs up and the number of volumes required.

After all of the files have been copied, BACKUP will try to record this list in the directory at the beginning of the master disk. If the operation required more than one backup volume, then BACKUP will prompt you to reinsert the master disk of the backup set.

The prompt that requests reinsertion of the master disk volume appears in the following form;

```
Insert backup master volume 1, {destfile}.000, in drive {x},
and hit RETURN when ready.
```

Where {destfile} is the primary file name you specified for the destination file; and

where {x} identifies the destination drive you specified in the BACKUP command.

In response to a prompt in this form, you should insert the master volume disk (the disk that was first inserted in the destination drive during this BACKUP operation). Then you should close the drive latch and press the **RETURN** key.

BACKUP will then try to record statistics about the BACKUP operation in the master backup file directory.

If BACKUP finds the master backup file on the disk you just inserted, it will record the statistics in the directory, and a prompt (utility prompt or system prompt) will appear to signal the end of this BACKUP operation.

If BACKUP cannot find the master backup file on the disk you just inserted, it will display a message in the following form:

```
Cannot open master backup file, {destfile}.000,
insert another disk in drive {x},
and hit RETURN when ready,
or hit any other key to abort.
```

Where {destfile} is the primary file name you specified for the destination file (the copy of this file with the "000" extension is called the master backup file); and where {x} identifies the destination drive you specified in the BACKUP command.

If you wish to complete the operation and can obtain the disk that contains the master backup file (which has the primary name displayed as {destfile} and the "000" extension), then insert this disk in the specified drive, close the drive latch, and press **RETURN**.

If you do not wish to continue the operation or cannot obtain the disk that contains the master backup file, then press any key other than RETURN. The BACKUP operation will end and a prompt (utility or system prompt) will appear.

Preparing BACKUP Routines

If you create backups on a regular basis, BACKUP can come in very handy. BACKUP was designed specially so that you would be able to store the BACKUP command lines that you enter on a regular basis, so that they can be entered automatically, with far less typing.

To store BACKUP commands for automatic execution, you will need a text editor or word processor and the SUBMIT utility.

The text editor or word processor will enable you to prefabricate and store commonly entered BACKUP command lines in a disk file. The SUBMIT utility will enable you to type a short, simple command line that causes automatic execution of all the stored BACKUP commands.

This text explains a few BACKUP routines geared toward users of some popular application software products available for your computer system. The text on each routine shows you how to prepare the routine by explaining the following essential facts:

- The type of user who would probably benefit from using the routine
- The names of the drives in which you should store particular files during the routine
- The form of the file you should create (with your text editor or word processor) to store commonly entered BACKUP command lines

- The form of the SUBMIT command you should type each time you wish to use the routine.

NOTE: To successfully prepare a BACKUP routine, you should understand the operation of the SUBMIT utility. For information on SUBMIT, refer to the "SUBMIT" text in the "User's Guide" of your CP/M-86 manual.

General Purpose BACKUP Routine

This routine can be helpful for anyone who wants to back up an entire partition regularly. Steps 1 through 3 can be considered preparation steps that you need to perform only once. Steps 4 through 6 should be performed every time you conduct the routine.

1. Using a text editor or word processor, open a text file under the name **GNBACKUP.SUB**.

2. Into this file, enter the following command line:

   ```
   BACKUP $1:GENBACK=$2:*.* [D:$3-$4-$5;V] RETURN
   ```

3. Close the text file GNBACKUP.SUB.

4. Use the PIP utility to copy the text file GNBACKUP.SUB and the utility file SUBMIT.COM to drive A, if they are not there already.

5. When you are ready to perform the BACKUP routine, type `A:` and press **RETURN** to make drive A the default drive.

6. Type a command in the following form:

   ```
   A>SUBMIT GNBACKUP {floppy} {partition} {mm} {dd} {yy} RETURN
   ```

Where `{floppy}` is the drive letter of the floppy disk drive to which you wish to back up your files;

where `{partition}` is the drive letter of the Winchester disk partition from which you wish to back up your files;

where {mm} is a one- or two-digit entry for the current month, within the range 1-12;

where {dd} is a one- or two-digit entry for today's date, within the range 1-31; and

where {yy} is a one- to four-digit entry for the current year, within the range 0-9999.

For example, if today is February 20, 1984, and you wish to back up files from the drive A partition to the drive C floppy disk, then you should type the following command:

```
A>SUBMIT GNBACKUP C A 2 20 84 RETURN
```

BACKUP will back up all files from the partition into a backup file with the primary name "GENBACK."

SuperCalc™ BACKUP Routine

This routine can be helpful to users of the SuperCalc spread sheet program who wish to regularly back up all SuperCalc data files from a Winchester disk partition. Steps 1 through 3 can be considered preparation steps that you need to perform only once. Steps 4 through 6 should be performed every time you conduct the routine.

1. Using a text editor or word processor, open a text file under the name **SCBACKUP.SUB**.

2. Into this file, enter the following command line:

```
BACKUP $1:SCBACK=$2:*.CAL [D:$3-$4-$5;V] RETURN
```

3. Close the text file SCBACKUP.SUB.

4. Use the PIP utility to copy the text file SCBACKUP.SUB and the utility file SUBMIT.COM to drive A, if they are not there already.

™SuperCalc is a registered trademark of Sorcim Corporation.

5. When you are ready to perform the BACKUP routine, type `A:` **RETURN** to make drive A the default drive.

6. Type a command in the following form:

```
A>SUBMIT SCBACKUP {floppy} {partition} {mm} {dd} {yy} RETURN
```

Where {floppy} is the drive letter of the floppy disk drive to which you wish to back up your files;

where {partition} is the drive letter of the Winchester disk partition from which you wish to back up your files;

where {mm} is a one- or two-digit entry for the current month, within the range 1-12;

where {dd} is a one- or two-digit entry for today's date, within the range 1-31; and

where {yy} is a one- to four-digit entry for the current year, within the range 0-9999.

For example, if today is February 20, 1984, and you wish to back up files from the drive A partition to the drive C floppy disk, then you should type the following command:

```
A>SUBMIT SCBACKUP C A 2 20 84 RETURN
```

BACKUP will back up all SuperCalc data files from the partition into a backup file with the primary name "SCBACK."

WordStar™ BACKUP Routine

This routine can be helpful to users of the WordStar word processor who wish to regularly back up all WordStar text files that have the same file name extension. Steps 1 through 3 can be considered preparation steps that you need to perform only once. Steps 4 through 6 should be performed every time you conduct the routine.

™WordStar is a registered trademark of MicroPro International Corporation.

1. Using WordStar's non-document editing mode, open a text file under the name **WSBACKUP.SUB**.

2. Into this file, enter the following command line:

```
BACKUP $1:WSBACK=$2:*.$3 [D:$4-$5-$6;V] RETURN
```

3. Save the text file WSBACKUP.SUB.

4. Use the PIP utility to copy the text file WSBACKUP.SUB and the utility file SUBMIT.COM to drive A, if they are not there already.

5. When you are ready to perform the BACKUP routine, type `A:` **RETURN** to make drive A the default drive.

6. Type a command in the following form:

```
A>SUBMIT WSBACKUP {floppy} {partition} {ext} {mm} {dd} {yy} RETURN
```

Where {floppy} is the drive letter of the floppy disk drive to which you wish to back up your files;

where {partition} is the drive letter of the Winchester disk partition from which you wish to back up your files;

where {ext} is the one- to three-letter file name extension of the text files you wish to back up;

where {mm} is a one- or two-digit entry for the current month, within the range 1-12;

where {dd} is a one- or two-digit entry for today's date, within the range 1-31; and

where {yy} is a one- to four-digit entry for the current year, within the range 0-9999.

For example, if today is February 20, 1984, and you wish to back up files with the "DOC" extension from the drive A partition to the drive C floppy disk, then you should type the following command:

```
A>SUBMIT WSBACKUP C A DOC 2 20 84 RETURN
```

BACKUP will back up all WordStar text files with the specified file name extension from the partition into a backup file with the primary name "WSBACK."

Structure of BACKUP Files

The BACKUP utility backs up several primary files into a single backup file. This backup file contains all of the files that were copied, in the order they were specified. At the beginning of the backup file is a directory of the copied files. The BACKUP utility maintains this directory to show the names of the files within the backup file, so that the RESTORE utility, when used, will be able to copy the files to other media.

At the beginning of the backup file directory is an entry for the backup file itself. This backup file directory entry is a sequence of bytes that are stored in the following form:

Field Function	Field Size (bytes)
ID	1
File Control Block (FCB)	32
R	4
quantity of copied files	2
date of backup	2
R	7
release number	1
version number	1
R	14
	64 bytes

Where *ID* is a byte that is set to one to distinguish this master backup directory entry from normal file directory entries;

where *File Control Block (FCB)* is a set of bytes that describes the drive name, primary file name, file name extension, and other characteristics used by CP/M to store files;

where *R* stands for reserved bytes that are not used for any particular purpose in this version of the software;

where *quantity of copied files* is the number of individual files stored within the backup file;

where *date of backup* is the date that the user entered in response to a prompt during the BACKUP operation; and

where *release number* and *version number* are included to insure that this software will be used only with compatible releases and versions of BACKUP and RESTORE.

NOTE: The file name extension in the "File Control Block (FCB)" of the backup file directory entry is the file name extension of the last volume of the backup disk set—and not necessarily the extension of the volume that contains the directory. Thus you can determine how many volumes were used to accommodate all of the backed up files by looking at the directory in the first volume. (The first volume is the volume given the extension "000" during the BACKUP operation.)

Immediately following the backup file entry in the backup file directory are several sequences of bytes, each describing attributes of one of the individual files that was copied into the backup file during the BACKUP operation. The bytes of the individual file directory entry are stored in the following form:

Field Function	Field Size (bytes)
ID	1
File Control Block (FCB)	32
R	4
start volume	1
end volume	1
start position	2
end position	2
file length	2
U	1
R	18
	64 bytes

Where *ID* indicates the location of this individual file entry among the other individual file entries in the directory. If the value of this byte is 2, then the entry describes an individual file within the backup file. If the value of this byte is FF, then the entry is a "dummy" entry that merely signals the end of the backup file and does not describe an individual file;

where *File Control Block (FCB)* is a set of bytes that describe the drive name, primary file name, file name extension, and other characteristics used by CP/M to store files;

where *R* stands for reserved bytes that are not used for any particular purpose in this version of the software;

where *start volume* indicates the number of the first backup file volume used to accommodate this individual file;

where *end volume* indicates the number of the last backup file volume used to accommodate this individual file;

where *start position* is the number of 128-byte records between the beginning of the backup file volume and the beginning of this individual file;

where *end position* is the number of 128-byte records between the beginning of the backup file volume and the record following the end of this individual file;

where *file length* is the length of this individual file, as measured in 128-byte records; and

where *U* is the number of the user area from which the individual file came.

The number of individual source files that can be backed up in a single BACKUP operation is limited by the amount of space on the first disk/partition used to receive the backup volumes. This disk/partition must have enough free space to accommodate the entire backup file directory.

NOTE: If any of the source files you specify are random files (as opposed to sequential files), these files might become larger when restored with the RESTORE utility.

BACKUP Error messages

```
Verify error, try BACKUP again (Y/N)?
```

EXPLANATION: BACKUP has detected that the copy of a backed up file on destination medium is different from the original copy of the file on the source medium. This message usually indicates a surface imperfection on the destination medium.

Press **Y** if you wish to have BACKUP try to recopy and verify the same individual file, overwriting the bad copy of this file. Then BACKUP will resume displaying the names of the individual files as they are copied and verified, beginning with the file in which the verification error occurred.

Press **N** if you wish BACKUP to skip the file that it just failed to verify, and try to copy and verify the next of the specified source files.

```
Backup filename can not be ambiguous.
```

EXPLANATION: You specified * or ? wild card characters in the destination file name for the BACKUP operation. Repeat the command specifying the destination with an explicit file name.

```
Can not open master backup file (filename).000,
not enough space on the disk.
```

EXPLANATION: The disk/partition specified as the destination medium for the backup file (filename) did not have enough free disk space to accommodate the entire backup file directory. Repeat the BACKUP operation specifying a larger partition for the destination (if you are backing up files to the Winchester disk) or insert a formatted floppy disk with more free space (if you are backing up files to a floppy disk drive).

BACKUP

```
Extension on backup file specified.
Extension 000 will be assumed.
```

EXPLANATION: This occurs whenever you try to assign a backup file an extension by specifying a full file name for a destination file. If this occurs, BACKUP ignores the extension you specified and used its standard, sequentially numbered extensions. Program operation will not be disturbed.

```
Can not find master backup file (filename).000.
```

EXPLANATION: This message occurs when the /L option is requested for a file from a disk where the master backup `{filename}.000`, is not present.

```
File {filename}.{ext} is not found.
```

EXPLANATION: This message occurs whenever a file is specified for BACKUP or RESTORE and that file is not on the disk.

```
Insert another disk in drive {x} for backup
and hit RETURN when ready,
or hit any other key to abort.
```

EXPLANATION: The disk just used to receive some of the copied files is now full. Remove this disk from the drive, label it, insert another formatted disk into the destination drive, close the drive latch, and press the **RETURN** key.

```
Invalid backup file.
```

EXPLANATION: This message occurs if the backup file specified in a command does not contain valid information. This may occur if the file specified was not a backup file, but had a "000" extension, or if the data in a backup file had degraded (possibly due to a bad sector or inadvertent exposure of the medium to an electromagnetic field).

```
Invalid date in option
```

EXPLANATION: This message occurs if the date given with the D option was not entered in the proper form. If any of the numbers in the data are out of range, if any numbers are separated by invalid characters, or if the D option letter is omitted, then this message will appear.

```
Invalid date.
```

EXPLANATION: This message occurs if the date entered in response to the date prompt does not conform to the proper form for entering dates. If any of the numbers in the date are out of range, or if any numbers are separated by invalid characters, then this message will appear.

```
Invalid drive designation.
```

EXPLANATION: This message occurs when a drive name is used that is not in the range of supported drive names (A through F).

```
Invalid exception file specifications.
```

EXPLANATION: This message will occur if the exception file specified has a syntax error in the specification.

```
Invalid filename.
```

EXPLANATION: This message appears when a file name is specified that does not conform to CP/M file naming conventions.

```
Invalid selection file specifications.
```

EXPLANATION: This message is generally caused by a typographical error in the command line. The message results when parameters in the command line appear garbled or incorrectly punctuated.

UTILITIES

BACKUP

```
Invalid option [x] specified.
```

EXPLANATION: This message occurs if BACKUP is unable to recognize the option [x] that was specified in the command.

```
Not enough parameters specified.
```

EXPLANATION: This message results when the BACKUP command is not complete enough to carry out the intended operation.

```
Can not open master backup file {filename}.000,
insert another disk in drive x
and hit RETURN when ready,
or hit any other key to abort.
```

EXPLANATION: This message occurs if the disk that has been inserted is not volume 1. Insert the correct disk.

```
Can not open backup file {filename}.{nnn},
insert another disk in drive x
and hit RETURN when ready,
or hit any other key to abort.
```

EXPLANATION: This message occurs when you are asked to insert volume nnn+1 (which would contain {filename}.{nnn} and the wrong disk is inserted. Insert the correct disk.

```
Invalid user option.
```

EXPLANATION: This message occurs if the user number specified with the U option is not in the range 0-15 or if a user number was specified with improper syntax.

```
Invalid version of BACKUP for {filename.000}.
```

EXPLANATION: This message occurs if you have used different versions of the BACKUP, for instance in a command that included the [L] option. When producing a directory of a backup file, use the same version of BACKUP as you used to create the backup file.

```
No file selected.
```

EXPLANATION: This message occurs if none of the source files you specified in the BACKUP command exist on the default or specified source media.

CONFIGUR

The Utility that Customizes CP/M for Your Hardware and/or Preferences

The CONFIGUR utility helps you to change the CP/M Operating System so that it will accommodate a particular printer or modem. CONFIGUR also enables you to set the system to automatically invoke commands upon cold boots and/or warm boots and to assign physical devices to logical devices. After you have specified these changes to the system, CONFIGUR enables you to apply them to the system in memory and/or the system on disk, or to cancel them completely.

CONFIGUR is usually run during the first session of CP/M use in a particular hardware environment. But it should also be run whenever a hardware component is added or changed, or whenever you wish to change an automatic command line.

You do *not* need to use CONFIGUR if you do not have a printer or modem, or if you have one of the following:

- A serial printer (such as the Z-125 or the H-125) that runs at 4800 baud, accepts 8 bits per character, uses no parity bit, handshakes with RTS pin number 4, is ready when handshaking signal is High, and uses no protocol.

- A modem (such as the WH-13, the Lexicon WH-23, the UDS WH-33, or the Hayes WH-43) that runs at 300 baud, accepts 8 bits per character, uses no parity bit, and uses no handshaking.

These hardware settings match the default settings of the CP/M-86 system when it is shipped.

Since CONFIGUR adjusts the system for hardware characteristics, you must sometimes answer prompts that ask about these characteristics. Always consult your hardware manuals and check the settings of your hardware devices when answering these prompts.

NOTE: Changes specified through the CONFIGUR utility can only be recorded on a disk if the disk is write-enabled. If you are performing a CONFIGUR operation with a write-protected disk, you can only apply changes to the system in memory.

Invoking CONFIGUR

You can invoke CONFIGUR by responding to the system prompt with a command line in the following form:

```
A>CONFIGUR RETURN
```

This entry causes the display of CONFIGUR's identification message, copyright notice, version number, and "MAIN MENU."

CONFIGUR Main Menu

CONFIGUR is a menu-driven utility. It first displays a main menu, which looks like this:

```
CP/M-86 System Configuration Utility rel. x.xx
  Copyright (C) 1983 by Zenith Data Systems

               *** MAIN MENU ***

           P - Printer Configuration
           M - Modem Configuration
           C - Command Configuration
           I - I/O map Configuration
           ? - Brief Help message

           X - Exit

Selection [P,M,C,I,X or ?] :
```

This menu enables you to access any of four sub-menus so that you can adjust the CP/M Operating System to accommodate your printer or modem. From this menu you may also specify an automatic command line preference and specify the I/O configuration for different physical devices. In addition, you can obtain screen displays of helpful comments about the menu and its use. This menu also enables you to exit the CONFIGUR utility, and thus gain access to the operating system.

To enter a particular sub-menu or exit selection, type the character listed to the left of that selection and press **RETURN**.

Printer Configuration

In order to adjust the system to accommodate your printer, you must select "Printer Configuration" at the main menu and specify your printer either by its name or by its characteristics.

When you enter **P** and then press **RETURN** at the main menu, CONFIGUR displays the following sub-menu:

```
                 *** Printer Configuration ***

       1- MX-80 or other PARALLEL Centronics-interface printer
       2- H/Z-25
       3- H-14 or TI-810(WH-24)
       4- Dec LA-34 or LA-36
       5- Diablo 620
       6- Diablo 630,1610,1620,1630 or 1640(WH-44)
       7- MX-80 Serial
       8- Votrax Type 'n Talk
       9- User-defined SERIAL Printer

     Choose the number that corresponds to your printer :
```

- If your printer is listed by name in this menu, and you have not changed the switch settings since it was shipped, then see "Specifying Printer Name."

- If you have a printer that is not listed by name on this menu or a listed printer on which the switch settings have been changed since shipping, then see "Specifying Printer Characteristics."

Specifying Printer Name

If your printer is listed by name on the "Printer Configuration" menu, and you have not changed the switch settings since it was shipped, then type the number listed to the left of the printer name and press **RETURN**.

This entry will adjust CP/M to accommodate the characteristics that usually apply to your type of printer when it is shipped.

CONFIGUR then displays a message describing the characteristics of the printer you selected. The printer descriptions for each menu-listed printer are shown in the following list.

NOTE: If you have a printer that is not listed by name on this menu, or a listed printer on which the switch settings have been changed since shipping, then type **9** and press **RETURN**, and see "Specifying Printer Characteristics."

Each printer you select in response to the "Printer Configuration" menu has different characteristics, which are described in the message CONFIGUR displays after your selection.

MX-80 or Centronics

```
You have selected a MX-80 Parallel, and Centronics

  Assure Centronics style parallel operation

Press RETURN to access Main menu:
```

H/Z-25

```
You have selected a H/Z-25

Standard settings:
   4800 baud
  Handshake on RTS
  Printer ready on high

Press RETURN to access Main menu:
```

H-14 or TI-810

```
You have selected a H-14, TI-810(WH-24)

Standard settings:
   4800 baud
  Handshake on RTS
  Printer ready on low

Press RETURN to access Main menu:
```

DEC LA-34 or LA-36

```
You have selected a DEC LA-34 or an LA-36

Standard settings:
   300 baud

Press RETURN to access Main menu:
```

Diablo 620

```
You have selected a Diablo 620

Standard settings:
   300   baud
 ETX/ACK protocol

Press RETURN to access Main menu:
```

Diablo 630 or 1640

```
You have selected a Diablo 630,1610,1620,1630 or 1640

Standard settings:

   1200   baud
 ETX/ACK protocol

Press RETURN to access Main menu:
```

MX-80 Serial

```
You have selected a MX-80 with serial interface

Standard settings:
   4800   baud
  Handshake on DTR
  Printer ready on low

Press RETURN to access Main menu:
```

Votrax Type 'N Talk

```
You have selected a Votrax Type 'n Talk

Standard settings:
   4800   baud
  Handshake on RTS
  Printer ready on high

Press RETURN to access Main menu:
```

After you view the message that lists your printer's characteristics, press **RETURN**. The "MAIN MENU" will be redisplayed.

- If the settings of your printer match those listed in the message for your printer, then enter the letter for another selection.

- If you specified your printer by name at the "Printer Configuration" menu, but want this printer set with characteristics that differ from those listed for your printer by CONFIGUR, type **P** and **RETURN** again at the "MAIN MENU." Then type **9** and **RETURN** at the "Printer Configuration" menu and see "Specifying Printer Characteristics."

NOTE: Printers selected by name at the "Printer Configuration" menu have their appropriate physical device assigned automatically to the LST: logical device.

Specifying Printer Characteristics

To adjust the system for a printer that is not listed by name on this menu, or a listed printer on which the switch settings have been changed since shipping, type **9** and press **RETURN** at the "Printer Configuration" menu.

CONFIGUR will then prompt you to specify characteristics of your printer such as: baud rate, bits per character, parity, handshaking pin, handshake polarity, and protocol.

NOTE: If you respond to one of these prompts by pressing RETURN without first typing one of the values listed in the prompt, the prompt will be redisplayed.

Printer Baud Rate

When you enter **9** and **RETURN** at the "Printer Configuration" menu, CONFIGUR will first display the following message:

```
                    User Defined Printer

      Baud rate selection
     (Space=next possibility,
      BACK SPACE=last selection,
      RETURN=select this one)
     Baud Rate : 45.5
```

Explanation

Baud rate is the speed at which data are transmitted to and from your printer, roughly corresponding to the number of bits per second transmitted. Since CP/M coordinates the transmission of data to and from your printer, CP/M must know how fast your printer is set to send and receive data.

Specification

To the right of `Baud Rate :` in this prompt, you will see a number that corresponds to one of the 16 baud rates possible on the Z-100 computer (45.5, 50, 75, 110, 134.5, 150, 300, 600, 1200, 1800, 2000, 2400, 4800, 9600, 19200, 38400).

Each baud rate value is stored in sequence by CONFIGUR, although only one value is visible at a time. You can view a different value by pressing the **SPACE BAR** for a greater value, or by pressing the **BACK SPACE** key for a lesser value.

When the correct baud rate value is displayed in the `Baud Rate :` prompt, press **RETURN** to specify this value. CONFIGUR will get ready to set CP/M to accommodate this baud rate value for printer data transmission.

Bits Per Character for Printer

After you have specified a baud rate, CONFIGUR will display the following prompt:

```
    Bits per character: [5,6,7 or 8] :
```

Explanation

Bits per character is the number of significant data bits your printer expects to receive in order to decipher one character (byte) from the stream of bits that are transmitted. Start bits, stop bits, and parity bits are not included in this number. (Most printers accept 7 or 8 bits per character.)

Specification

To specify the number of bits per character your printer accepts, type **5, 6, 7,** or **8** at the `Bits per character` prompt. Then press **RETURN**. CONFIGUR will prepare CP/M to transmit this many bits within every character of data it sends.

Printer Parity

After you have specified bits per character, CONFIGUR will display the following prompt:

```
    Parity [O(dd),E(ven),N(one)] :
```

Explanation

Parity is a method by which data are checked to make sure they have not changed during transmission.

When odd parity is used, a parity bit is sent along with each character that is sent to the printer. Before transmission, this bit is set to either one or zero to ensure that the sum of all of the transmitted bits is an odd number. If the printer receives a byte of data bits and a parity bit that do not all add up to an odd number, then an error must have occurred during transmission.

Even parity works the same way, except that the parity bit is set to either one or zero to ensure that the sum of the transmitted bits is an even number.

When no parity is used, no parity bit accompanies each transmitted character.

Specification

To specify the kind of parity (if any) that is used to check errors in the data sent to your printer, type **O** for odd parity, **E** for even parity, and **N** for no parity. After you type one of these specifications, press **RETURN**. CONFIGUR will prepare CP/M so that it transmits parity bits as required by your printer.

Printer Handshake Pin

After you have specified the nature of your printer's parity, CONFIGUR will display the following prompt:

```
Handshake Pin [N(one), D(tr/pin20), R(ts/pin4)] :
```

Explanation

The *handshake pin* is the circuit through which the printer and the computer signal each other to determine when data should be transmitted. This circuit is one of many that are bundled together in the RS-232C cable that connects the computer with the printer. The printer uses this cable circuit to signal the computer that the printer is ready to receive more data.

The DTR (Data Terminal Ready) handshake pin is labelled with the number 20 on the receptacle ends of the cable; and the RTS (Request To Send) handshake pin is labelled with the number 4 on the receptacle ends of the cable.

Specification

To specify the particular handshake pin (if any) that is used to signal the printer's readiness to receive data, type **N** for no pin, type **D** for the DTR (number 20) pin, or type **R** for the RTS (number 4) pin. After typing one of these specifications, press **RETURN**. CONFIGUR will prepare CP/M so that it signals the printer's readiness as required by your printer.

Printer Handshake Polarity

If you specified a handshake pin (by typing D or R at the previous prompt), then a prompt will appear in the following form:

```
Polarity [H(ready when High),L(ready when Low) ] :
```

NOTE: If you specified no handshake pin (by typing N at the previous prompt), then the "Polarity" prompt will not appear. Instead, the handshake pin prompt will be followed by a protocol prompt.

Explanation

Polarity is the voltage level at which the printer signals the computer that the printer is ready to receive more data. This signal can be sent at either a high voltage level or a low voltage level. It travels through the handshake pin that was specified at the previous prompt.

Specification

To specify a polarity, type **H** if the printer signals its readiness with a high voltage signal, and type **L** if the printer signals its readiness with a low voltage signal.

Printer Protocol

After you have answered the handshake pin prompt (and possibly the polarity prompt), CONFIGUR will display the following prompt:

```
Protocol [X(on/Xoff),E(tx/Ack),N(one)] :
```

Explanation

Protocol is a method of controlling the transmission of data between your computer and your printer by assigning one device or both devices to transmit specific characters as a signal to the other device.

With XON/XOFF protocol, the printer transmits the DC3 character Control-S (Device Control 3: turns transmitter off) when the printer must momentarily stop receiving data, either because the buffer is full or because the printer is “off line.” The printer sends the DC1 character Control-Q (Device Control 1: turns transmitter on) when the printer is ready to receive more data, as when the buffer has empty space and the printer is “on line.”

With ETX/ACK protocol, the computer transmits the ETX Control-C character at the end of the transmission of a unit of data (to signify the End of TeXt). When the printer processes the unit of data with the ETX character, the printer transmits the ACK Control-R character back to the computer (to ACKnowledge that the data unit was received).

Specification

To specify the particular protocol characters (if any) that are transmitted to signal the printer's readiness to receive more data, type **X** for XON/XOFF protocol, type **E** for ETX/ACK protocol, and type **N** for no protocol. After typing one of these letters, press **RETURN**. CONFIGUR will prepare CP/M so that it allows transmission of the proper signal characters, if any.

After you have specified a protocol, CONFIGUR will display:

```
     Press RETURN to access Main Menu:
```

Press **RETURN** and CONFIGUR will redisplay the "MAIN MENU," at which you can enter another selection.

Modem Specification

In order to adjust the system to accommodate your modem, enter **M** and press **RETURN** at the "Configur Main Menu." Then CONFIGUR will display one or more prompts that help you to specify either the model number or characteristics of your modem. The first modem prompt looks like this:

```
                    *** Modem Configuration ***

     Standard Heath/Zenith Modem? (WH-13,WH-23,WH-33,WH43) [Y/N] :
```

Your response to this prompt depends on whether your modem is listed in the prompt by its model number. If not, you must specify individual characteristics of your modem.

Specifying Modem Model Number

If the model number of your modem (as shown in the Heath/Zenith catalog) is listed in this prompt, and you have not changed the switch settings since it was shipped, they type **Y** and press **RETURN**. This entry prepares CONFIGUR to adjust CP/M for the characteristics that usually apply to these modems when they are shipped. After the entry, CONFIGUR will redisplay the "MAIN MENU." Then you can enter the letter for another selection.

Specifying Modem Characteristics

If you have a modem that is not listed in this prompt, or if you have a listed modem on which the switch settings have been changed since shipping, then type **N** and press **RETURN** at this prompt. CONFIGUR will then enable you to specify characteristics of your modem, such as baud rate, bits per character, parity, and handshaking pin. First CONFIGUR displays a prompt for baud rate selection.

NOTE: If you respond to one of these prompts by pressing **RETURN** without first typing one of the values listed in the prompt, the prompt will be redisplayed.

Modem Baud Rate

When you enter the letter **N** at the "Modem Port Selection" menu, CONFIGUR will first display the following message:

```
   Baud rate selection
 (Space=next possibility,BS=last,CR=select this one)
    Baud Rate :    45.5
```

Explanation

Baud rate is the speed with which data are transmitted to and from your modem, roughly corresponding to the number of bits per second transmitted. Since CP/M coordinates the transmission of data to and from your modem, CP/M must know how fast your modem is set to send and receive data.

Specification

This prompt enables you to specify the baud rate of your modem. To the right of the words `Baud Rate :`, you will see a number that corresponds to one of the 16 baud rates possible on the Z-100 computer (45.5, 50, 75, 110, 134.5, 150, 300, 600, 1200, 1800, 2000, 2400, 4800, 9600, 19200, 38400).

CONFIGUR displays these values, one at a time, in sequence. You can view a different value by pressing the **SPACE BAR** for a greater value, or by pressing the **BACK SPACE** key for a lesser value.

When the correct baud rate value is displayed in the `Baud Rate :` prompt, press **RETURN** to specify this value. CONFIGUR will prepare CP/M to accommodate this baud rate value for modem data transmission.

Bits Per Character for Modem

After you have specified a baud rate, CONFIGUR will display the following prompt:

```
Bits per character: [5,6,7 or 8] :
```

Explanation

Bits per character is the number of significant data bits your modem expects to receive in order to decipher one character (byte) from the stream of bits that are transmitted. Start bits, stop bits, and parity bits (if used) are not included in this number.

Specification

To specify the number of bits per character your modem accepts, type **5, 6, 7,** or **8** at the `Bits per character` prompt. Then press **RETURN**. CONFIGUR will prepare CP/M to transmit this many bits within every character of data it sends.

Modem Parity

After you have specified bits per character, CONFIGUR will display the following prompt:

```
Parity [O(dd),E(ven),N(one)] :
```

Explanation

Parity is a method by which data are checked to make sure it hasn't changed during transmission.

When odd parity is used, a parity bit is transmitted along with each character that is transmitted to the modem. Before transmission, this bit is set to either one or zero to ensure that the sum of all of the transmitted bits is an odd number. If the modem receives a byte of data bits and a parity bit that do not all add up to an odd number, then an error must have occurred during transmission.

Even parity works the same way, except that the parity bit is set to either one or zero to ensure that the sum of all of the transmitted bits is an even number.

When no parity is used, no parity bit accompanies each transmitted character.

Specification

To specify the kind of parity (if any) that is used to check errors in the data sent to your modem, type **O** for odd parity, **E** for even parity, and **N** for no parity. After typing one of these specifications, press **RETURN**. CONFIGUR will prepare CP/M so that it transmits parity bits as required by your modem.

Modem handshake Pin

After you have specified the nature of your modem's parity, CONFIGUR will display the following prompt:

```
    Handshake Pin [N(one), D(tr/pin20), R(ts/pin4)] :
```

Explanation

The *handshake pin* is the circuit through which the modem and the computer signal each other to determine when data should be transmitted. This circuit is one of many bundled together in the RS-232C cable that connects the computer with the modem. The modem uses this circuit to signal the computer that it is ready to receive more data.

The DTR (Data Terminal Ready) handshake pin is labelled with the number 20 on the receptacle ends of the cable; and the RTS (Request To Send) handshake pin is labelled with the number 4 on the receptacle ends of the cable.

Specification

To specify the particular handshake pin (if any) that is used to signal the modem's readiness to receive data, type **N** for no pin, type **D** for the DTR (number 20) pin, or type **R** for the RTS (number 4) pin. After typing one of these specifications, press **RETURN**. CONFIGUR will prepare CP/M to signal the modem's readiness, as required by your modem.

Then CONFIGUR will redisplay the "MAIN MENU," at which you can enter the letter for another selection.

Automatic Command Specification

In order to adjust the system to accommodate your preference in automatic commands, you must type **C** and **RETURN** to select "Command Configuration" at the "MAIN MENU." CONFIGUR will display the following menu:

```
              *** Command Line Configuration ***

   C - Cold Boot Command Line =
   W - Warm Boot Command Line =
   ? - Brief Help Message

   X - Exit

 Selection [C,W, ? or X]:
```

This menu enables you to specify that a command be automatically invoked upon a cold boot, or that a command be automatically invoked upon a warm boot. You can specify up to two different commands: one for execution on cold boot, and one for execution on warm boot. In addition, this menu enables you to display a screen of helpful comments about this menu and its use, if you enter a ?.

Cold Boot Command Line

To specify a command for automatic cold boot invocation, type **C** and press **RETURN** at the "Command Line Menu." CONFIGUR will display the prompt:

```
    Cold Boot Command Line :
```

At this prompt type a valid CP/M command line, just as you would type it at a CP/M system prompt. When you press **RETURN** to end the command line, CONFIGUR will display the command line on the "Command Line Menu" and prepare CP/M to invoke this command immediately after every cold boot.

If the menu already shows a cold boot command line and you wish to remove it, type **C** at the "Command Line Menu" and immediately press **RETURN**.

NOTE: Your Z-100 is capable of performing an automatic cold boot when powered up or reset, if a switch within the Z-100 is set properly. If this automatic booting feature is not switched on, you must power up or reset the computer and then enter a bootstrap command to perform a cold boot. In either case, the cold boot will be immediately followed by invocation of a command specified through CONFIGUR.

Warm Boot Command Line

To specify a command for automatic warm boot invocation, type **W** and **RETURN** at the "Command Line Menu." CONFIGUR will display the prompt:

```
    Warm Boot Command Line :
```

At this prompt, type a valid CP/M command line, just as you would type it at a CP/M system prompt. When you press **RETURN** to end the command line, CONFIGUR will display the command line on the "Command Line Menu" and prepare CP/M to invoke this command immediately after every warm boot.

If the menu already shows a warm boot command line and you wish to remove it, type **W** at the “Command Line Menu” and press **RETURN**.

NOTE: A warm boot can occur when you press **CTRL-C** or when some application programs exit to the operating system. In either case, the warm boot will be immediately followed by invocation of a command specified through CONFIGUR.

Acceptable Commands

Any valid resident command, transient command (utility), or application program is acceptable in the automatic command line. However, any file that the command line refers to must reside on the disk specified in the command line. For example, if the line reads:

```
C - Cold Boot Command Line = B:SC
```

Then the file “SC.COM” (part of the SuperCalc application program) must reside on the disk in drive B for the command to work. If any data referenced in the command line cannot be found in the specified drives, then command execution will be aborted and an error message will be displayed.

NOTE: Some transient commands (utilities) and application programs finish their execution by performing a warm boot. Therefore, we recommend that you do not enter a warm boot command line for a program that performs a warm boot after execution, because endless execution of the same program may be the result.

Returning to the Main Menu

To finish your command line menu activities and regain access to the “Configur Main Menu,” type **X** at the “Command Line Menu.” CONFIGUR will redisplay the “MAIN MENU,” which enables you to enter the letter for another selection.

I/O Device Specification

In order to adjust the system to send certain types of data through different physical devices, enter **I** and press **RETURN** at the "Configur Main Menu." Then CONFIGUR will display a menu in the following form:

```
                ***I/O Map Configuration***

       C - Console    Currently: CRT
       I - Auxin      Currently: UR1
       O - Auxout     Currently: UP1
       L - List       Currently: LPT
       ? - Brief Help Message

       X - Exit

     Selection [C,I,O,L,? or X] :
```

(The current assignments of your physical devices may differ from those in this sample menu.)

This menu shows the physical device names that are currently assigned to each logical device category.

NOTE: If you have already chosen a printer (by name or by characteristics) during this CONFIGUR run, then List device will have been automatically assigned to the proper physical device. Therefore, the letter "L" will *not* be listed in the I/O map selection prompt. Furthermore, you will not be able to select the List device during this CONFIGUR run.

Logical/Physical Device Categories

Data can be sent to or from a wide variety of peripheral hardware devices. CP/M groups these devices under the logical device categories "Console," "Auxin," "Auxout," and "List" (as shown on the left side of the menu).

However, these categories provide CP/M with only a general reference to the actual device being used. CP/M is capable of controlling data transfer through many different devices that are included in these categories.

Therefore, a more specific name must be assigned to each logical device category to inform CP/M of the kind of devices it must control. The specific name assigned to each logical device category is the "physical device name." Each physical device name helps CP/M to recognize and control a particular kind of peripheral hardware.

The physical device name currently assigned to each logical device category is listed on the right side of the menu.

Following is a table that will help you determine which physical device names can be matched with which logical device names.

SELECTION LETTER	LOGICAL DEVICE CATEGORY	PHYSICAL DEVICE NAME	DESCRIPTION AND/OR CATALOG NAME OF THE ACTUAL HARDWARE DEVICE USED
C	Console	TTY:	A printing terminal attached to serial port outlet A on the Z-100 (e.g., Decwriter).
		CRT:	A video display terminal and keyboard.
		BAT:	A batch pseudo-device using RDR: for input and LST: for output.
		UC1:	A modem attached to serial port B on the Z-100.
I	Auxin	TTY:	A printing terminal attached to serial port outlet A on the Z-100 (e.g., Decwriter).
		PTR:	Not implemented.
		UR1:	A modem attached to serial port B on the Z-100.
		UR2:	A video display terminal and keyboard.
O	Auxout	TTY:	A serial printer attached to serial port outlet A on the Z-100.
		PTP:	Not implemented.
		UP1:	A modem attached to serial port B on the Z-100.
		UP2:	A video display terminal and keyboard.
L	List	TTY:	A serial printer attached to serial port outlet A on the Z-100 (e.g., H/Z-25, H-14, TI-810, Diablo, Epson MX-80 serial, Decwriter).
		LPT:	A parallel printer attached to the parallel port on the Z-100 (e.g., Epson MX-80 parallel).
		CRT:	A video display terminal and keyboard.
		UL1:	A modem attached to serial port B on the Z-100.

NOTE: By entering a ? at this menu, you will cause a screen display of helpful comments about this menu and its use.

Assigning Physical Device Names

To select a logical device category to be changed, you must first type the letter (**C**, **R**, **P**, or **L**) listed to the left of the category in the menu and press **RETURN**. CONFIGUR will display one of the following prompts:

```
Console: [T(ty), C(rt), B(at) or 1(UC1)]:
```
or
```
Auxin: [T(ty),P(tr),1(UR1),2(UR2)] :
```
or
```
Auxout: [T(ty),P(tp),1(UP1),2(UP2)] :
```
or
```
List: [ T(ty),C(rt),L(pt),1(UL1)] :
```

Each of these prompts shows the logical device category on the left, and all of the physical device names that could be assigned to this category on the right.

To assign a physical device name to a logical device category, type the single character of the name that is displayed outside the parentheses.

CONFIGUR will respond by displaying the name of the specified physical device in the "I/O Map Configuration" menu.

You can access the CONFIGUR MAIN MENU by entering **X** and pressing **RETURN** at the "Selection" prompt beneath the "I/O Map Configuration."

NOTE: These logical/physical device assignments can be temporarily changed outside of CONFIGUR by using the STAT utility.

Assignment Examples

For instance, if you want to change the assignment of physical device name “TTY” to “LPT” in the “List” category, type **L** (for “List”) and press **RETURN** at the menu's selection prompt. Then type **L** (for “LPT”) and press **RETURN** at the “List” category prompt. CONFIGUR will redisplay the “I/O Map Configuration” menu with the “LPT” named as “Currently” the physical device assigned to “List.”

As a further example, if you wish to assign the physical device name “UR1” in place of the name “PTR” for the “Reader” category, type **R** (for “Reader”) and press **RETURN** at the menu's selection prompt. Then type **1** (for “UR1”) and press **RETURN** at the “Reader” category prompt. CONFIGUR will redisplay the “I/O Map Configuration” menu with the “UR1” named as “Currently” the physical device assigned to “Reader.”

Exiting from CONFIGUR

The “MAIN MENU” enables you to exit from the CONFIGUR utility and eventually to gain access to the CP/M Operating System.

```
CP/M-86 System Configuration Utility rel. x.xx
    Copyright (C) 1983 by Zenith Data Systems

                 *** MAIN MENU ***

           P - Printer Configuration
           M - Modem   Configuration
           C - Command Configuration
           I - I/O map Configuration
           ? - Brief Help message

           X - Exit

Selection [P,M,C,I X or ?]:
```

The Exit Selection

To begin to exit from CONFIGUR to CP/M, press **X** and press **RETURN**.

If you have *not* specified any changes to the system during this CONFIGUR activity, CONFIGUR will immediately relinquish control to CP/M, which will display the system prompt.

If you *have* specified any changes to the system during this CONFIGUR activity, CONFIGUR will display the following "Exit" menu:

```
          *** EXIT OPTIONS ***

     T - Make changes temporary (to memory only)
     P - Make changes permanent (to memory and disk)
     Q - Make no changes

     ? - Brief Help Message

     Choice [T,P,Q or ?]:
```

- If you wish to cancel all of the changes that you have specified without having them applied to the system, then press **Q** or press **RETURN**. The unchanged CP/M Operating System will then display the system prompt.

- If you wish to have CONFIGUR apply all of the changes you have specified only to the CP/M system now in active computer memory, then press **T** (for Temporary) at this prompt. These changes will remain in effect until the computer is reset. If you have specified changes pertaining to printers and/or modems, CONFIGUR will now display a diagram to assist you in connecting your hardware devices. (See "Connecting Your Hardware.") However, if you have specified no printer or modem changes, CONFIGUR will now relinquish control to CP/M, which will display the system prompt.

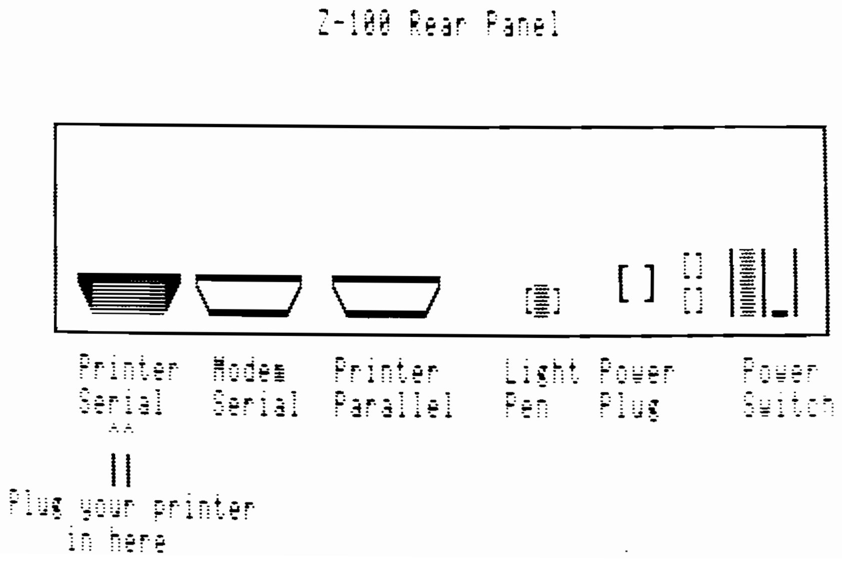
- If you wish to have CONFIGUR apply all of the changes you have specified both to the CP/M system in active computer memory and the system on the disk, then press **P** (for Permanent) at this prompt.

These changes will go in effect as soon as CP/M regains control. These changes will be recorded on the disk in drive A until the next CONFIGUR session conducted with the same disk in drive A. If you specified changes pertaining to printers and/or modems, CONFIGUR will now display a diagram to assist you in connecting your hardware devices. (See "Connecting Your Hardware.") However if you have specified no printer or modem changes, CONFIGUR will now relinquish control to CP/M, which will display a system prompt.

NOTE: If the disk in drive A was write-protected when you invoked CONFIGUR, the "P – Make changes permanent" choice will not appear in the "Exit" menu, and entering the letter P at this menu will have no effect.

Connecting Your Hardware

Before returning control to CP/M, CONFIGUR will display a diagram in the following form:



This diagram shows the location of the outlet on your Z-100 rear panel, onto which you should attach the cable that leads to your printer and/or modem. Attach the applicable cable(s) to the indicated outlet(s) before trying to use the printer and/or modem.

NOTE: This example diagram shows the message that will appear beneath the diagram if you had specified a serial printer during this CONFIGUR activity. The message appears beneath a different peripheral outlet if you have specified a parallel printer or a modem.

When your hardware devices are all properly attached, you can press **RETURN** to exit to the operating system.

CONFIGUR Error Messages

```
Incompatible Configur & CPM-86
```

EXPLANATION: You are running a CONFIGUR utility under a CP/M of a different version. The version number of your CONFIGUR and your CP/M must match. Copy CP/M (using LDCOPY) and CPM.SYS and CONFIGUR (using PIP) from CP/M-86 Distribution Disk I, boot up, and invoke CONFIGUR again.

```
Bad choice. Choice [1..9]:
```

EXPLANATION: Your entry at the “Printer selection” menu was not a number between 1 and 9. Enter the number between 1 and 9 that corresponds to your printer.

FORMAT

The Utility that Prepares Disk or Partition Surface

The FORMAT utility prepares a floppy disk or Winchester partition for the storage of data by establishing storage areas on the disk surface. At the same time, FORMAT erases any data that remain on the disk or partition from prior use, and sometimes inspects the recording surface for imperfections that could impair data storage or transmission. FORMAT also enables you to determine how much data you will be able to store on a floppy disk.

CAUTION: Because FORMAT erases all existing data on a disk or partition, make certain that you only format disks or partitions containing expendable data or no data. You can use the DIR or STAT commands (see the "User's Guide") to check for valuable data files before formatting. However, the DIR and STAT commands do not always display all of the files on a disk or partition.

You can use the FORMAT utility through either of two methods: the Utility Prompt Method or the System Prompt Method.

Utility Prompt Method

With this FORMAT method, you load the FORMAT utility into memory by entering a command at the system prompt. Then you answer a series of FORMAT prompts to define the formatting operation.

Utility Prompt Command Entry

Answer the system prompt with a command in the following form:

```
A>FORMAT RETURN
```

When invoked through the FORMAT prompt method, FORMAT first identifies itself with name, version number, and a caution about its capabilities. It also asks you if you wish to continue the operation, as shown:

```
CP/M-86 Format Version x.xx
Copyright (c) 1983 by Zenith Data Systems

This program is used to initialize a disk.
All information currently on the disk will be destroyed.
Is that what you want? (y/n):
```

Respond to this question by entering **Y** if you wish to format a disk or partition. Enter **N** if you do not.

Specifying the Disk or Partition to be Formatted

After you have confirmed your intention to format a disk or partition FORMAT asks:

```
Which drive do you wish to use for this operation?
```

Answer this prompt by entering the letter of the drive containing the disk (or assigned the partition) you wish to format. The drive you specify must be a valid disk drive in your hardware environment.

The drive you specify does not necessarily have to be a physical drive. For instance if you have only one physical 5.25-inch drive slot, then you can specify drive B at this prompt. You will later be prompted to put the appropriate 5.25-inch disk in the drive.

However if you specify a Winchester partition, this partition must have already been assigned a drive name. If the partition you wish to format is not currently assigned to the specified drive name, you must exit from the FORMAT utility and use the ASSIGN utility to assign this partition to the drive. You can exit from FORMAT at the “Which drive” prompt by entering **CTRL-C**.

Defining the FORMAT Operation

After you specify the drive containing the disk or partition you wish to format, FORMAT will display another prompt. The kind of prompt FORMAT displays depends upon whether you are trying to format a 5.25-inch floppy disk, an 8-inch floppy disk, or a Winchester partition.

Formatting a 5.25-Inch Disk

If you specified a drive that contains a 5.25-inch disk, the FORMAT utility enables you to specify the number of sides you want formatted by displaying the following prompt:

```
Number of Sides? (1=single, 2=double):
```

Entering the number **1** at this prompt will give the formatted disk a file capacity of 148 kilobytes. Entering the number **2** will give the formatted disk a file capacity of 304 kilobytes.

NOTE: All 5.25-inch disks formatted with this CP/M release are automatically formatted at double density. Therefore, the FORMAT utility will not prompt you to specify the density (level of data concentration) at which you want a 5.25-inch disk formatted.

After you respond to the `Number of Sides?` prompt, FORMAT will display a prompt in the following form to enable you to begin or cancel the FORMAT operation:

```
Put the disk you wish to be formatted in drive x.
Press RETURN to begin, anything else to abort.
```

(The character `x` stands for the letter of the disk drive you specified.) Pressing **RETURN** at this prompt will begin the actual formatting operation, while entering any other keyboard character will end the FORMAT operation.

NOTE: It takes at least one minute for FORMAT to format a disk. During this time the light on the specified disk drive will glow.

Formatting an 8-Inch Disk

If you are formatting an 8-inch disk, the FORMAT utility enables you to specify the level of density at which you wish to store data by displaying the following prompt:

```
Which density? (S=single, D=double):
```

The *density* of a disk refers to the level of concentration of the data stored on its surface. Higher density levels sometimes decrease data access reliability.

- If the disk in the specified drive is single-sided, then entering the letter **S** (for single density) at this prompt will give the formatted disk a file capacity of 241 kilobytes. Entering the letter **D** (for double density) will give the formatted disk a file capacity of 482 kilobytes.

- If the disk in the specified drive is double-sided, then entering the letter **S** (for single density) at this prompt will give the formatted disk a file capacity of 490 kilobytes. Entering the letter **D** (for double density) will give the formatted disk a file capacity of 980 kilobytes.

NOTE: Some 8-inch disks allow you to format only one side and others allow you to format both sides. These two types of disk are distinguished by the position of a small hole in the disk jacket, near the center spindle hole. The number of sides that can be formatted on a particular 8-inch disk is an unchangeable feature of that particular disk. The FORMAT utility automatically detects the position of this hole and prepares to format the disk on the appropriate number of sides.

After you respond to the `Which density?` prompt, FORMAT will display a prompt in the following form to enable you to begin or cancel the FORMAT operation:

```
Put the disk you wish to be formatted in drive x.
Press RETURN to begin, anything else to abort.
```

(The character `x` stands for the letter of the disk drive you specified.) Pressing **RETURN** at this prompt will begin the actual formatting operation, while pressing any other keyboard character will help you to end the FORMAT operation.

NOTE: It takes at least one minute for FORMAT to format a disk. During this time the light on the specified disk drive will glow.

Formatting a Winchester Disk Partition

If you specified a drive that has been assigned a Winchester partition, FORMAT will display a prompt in the following form to enable you to begin or cancel the FORMAT operation:

```
Will format partition assigned to drive x:

Press RETURN to begin, anything else to abort.
```

(The character x stands for the letter of the partition you specified.) Pressing **RETURN** at this prompt will begin the actual formatting operation, while pressing any other keyboard character will allow you to end the FORMAT operation.

The file capacity of the partition you are formatting is determined before you invoke the FORMAT utility, by the PREP utility (which automatically allocates half of the Winchester disk space to a single CP/M partition), or by the PART utility (which enables you to determine the size of your partitions). See text sections on "PREP" and "PART" in the Winchester User's Supplement for more information.

NOTE: As FORMAT formats a partition the light on the Winchester disk drive will glow.

Ending a FORMAT Operation

When FORMAT finishes preparing a disk's surface, or when you press "anything else to abort" FORMAT will display the following prompt:

```
Do you have more disks to format? (y/n):
```

If you wish to format another disk or partition without reinvoking FORMAT, press **Y** at this prompt. FORMAT will again prompt you to specify the drive containing the disk or partition you wish to format.

If you do not wish to format another disk or partition, press **N**. FORMAT will display the system prompt. Then you can enter any CP/M command.

NOTE: You can also end a FORMAT operation by entering **CTRL-C** at the `Which drive` prompt.

System Prompt Method

The System Prompt Method enables you to include all of the specifications necessary for the FORMAT operation in a single command line. Enter this command line at the CP/M system prompt.

Command Line Entry

System Prompt Method FORMAT commands are entered in the following form:

```
A>FORMAT {drive}:{[option,option]} RETURN
```

Where `FORMAT` is the command line function, stored in the file as FORMAT.COM on the logged disk;

where `{drive}` is the letter of the drive that contains the disk or partition you wish to format (this letter must represent a valid drive in your hardware environment, such as A, B, C, D, E, or F); and

where `{[option,option]}` represents letters and/or numbers enclosed in square brackets [ ] and separated by , (commas) to specify how the formatting operation should be conducted.

NOTE: You can specify logical (imaginary) drive names in FORMAT commands, as well as physical (visible) drive names. If you specify imaginary drive names, you will be prompted to insert two different disks alternately into a single physical drive slot.

FORMAT Options

FORMAT command lines entered by the System Prompt Method can include the following options:

SD	8-inch disk formatted to single density;
DD	8-inch disk formatted to double density;
1S	5.25-inch disk formatted on only one side;
2S	5.25-inch disk formatted on both sides;
F	Fast formatting, because the routine test of disk surface medium is not performed;
N	No prompt displayed between FORMAT command entry and FORMAT execution;

Options should be enclosed in square brackets and separated by commas when more than one is used. Options are the last item in a System Prompt Method FORMAT command line before you press **RETURN**.

NOTE: All 5.25-inch disks are automatically formatted double density by this version of FORMAT. 8-inch disks are automatically formatted to be single-sided or double-sided, depending on what format has been certified by the manufacturer. (FORMAT detects this certification by checking the position of the small hole in the disk cover next to the center spindle hole.)

System Prompt Method Defaults

When you enter a FORMAT command line with a drive specification, and decline to specify some or all of the possible options, FORMAT will prepare the disk according to the following default criteria:

- 5.25-inch disk formatted to double density (regardless of any options you might specify);
- 8-inch disk formatted to double density (as if you specified the DD option);

- 5.25-inch disk formatted on both sides (as if you specified the 2S option);
- 8-inch, single-sided disk formatted on one side (regardless of any options you might specify);
- 8-inch, double-sided disk formatted on both sides (regardless of any options you might specify);
- Disk surface will be tested for data retention (as if you had not specified the F option); and
- Prompts will be displayed between FORMAT command entry and FORMAT execution (as if you had not specified the N option). Therefore, whenever you enter a System Prompt Method command without the N option, the following prompt will appear:

```
CP/M-86 Format Version x.xx
Copyright (c) 1983 by Zenith Data Systems

This program is used to initialize a disk.
All information currently on the disk will be destroyed.
Is that what you want? (y/n):
```

- To confirm your intention to run a FORMAT operation, enter a **Y** at this prompt. Then FORMAT will display a prompt in the following form if you specified a floppy disk:

```
Put the disk you wish to be formatted in drive x.
Press RETURN to begin, anything else to abort.
```

  or the following prompt if you specified a Winchester partition:

```
Will format partition assigned to drive x:

Press RETURN to begin, anything else to abort.
```

To begin execution of the FORMAT operation for a floppy disk, insert the appropriate disk in drive x (where drive x is the drive you specified in the command line) and press **RETURN**. To begin execution of the FORMAT operation for a Winchester partition, simply press **RETURN**.

To abort the FORMAT utility, enter any keyboard character other than Y or y at the `Is that what you want?` prompt, or press anything other than RETURN at the `Press RETURN to begin, anything else to abort.` prompt. In either case, the FORMAT operation will end and CP/M will display the system prompt.

System Prompt Method Examples

`A>FORMAT B:[2S]` *RETURN*

FORMAT will prepare the surface of the disk in drive B (a 5.25-inch disk) to double density and on both sides, as specified by options. FORMAT will display prompts before formatting and will test the disk surface while formatting, by default.

`A>FORMAT B:` *RETURN*

FORMAT will prepare the surface of the disk in drive B (a double-sided 8-inch disk). Due to manufacturer's certification, this disk will be formatted on both sides. By default, this disk will be formatted to double density. Also by default, FORMAT will display prompts before formatting and test the disk surface while formatting.

`A>FORMAT B:[2S,1S]` *RETURN*

If your command line contains contradictory options, FORMAT will acknowledge the last one. Hence, in this case, FORMAT will format the surface of the disk in drive B (a 5.25-inch disk) on only one side, as specified by the last side quantity option. FORMAT will also display prompts before formatting and test the disk surface while formatting, by default.

```
A>C:format B:[Sd,f,N] RETURN
```

The FORMAT utility, in this case, is stored on the disk in drive C. It will prepare the surface of the disk in drive B (a single-sided 8-inch disk) to single density, as specified by the "Sd" option. Since the disk was manufactured for single-sided data storage, only one side will be formatted. The "f" option specifies that this formatting operation will be performed without a disk media test. The "N" option specifies that FORMAT will not prompt you to confirm your intentions before the formatting operation begins.

NOTE: As with any other command entered at a CP/M system prompt, you can edit a FORMAT command line with the **DELETE** key or erase the entire command line by entering **CTRL-X**.

Disk Capacities

The following tables show the amount of file space remaining on various kinds of disks after they are formatted under various kinds of conditions. (The FORMAT utility also prepares areas of the disk for the recording of the CP/M system kernel and the disk file directory, although the space reserved for such software items is not included in these tables.)

The following table shows the file capacities of 5.25-inch, soft-sectored disks formatted in 48-tpi drives:

	Single-sided	Double-sided
Double density	148 kilobytes	304 kilobytes

This table shows the file capacities of single-sided 8-inch disks:

	Single density	Double density
Single-sided	241 kilobytes	482 kilobytes

This table shows the file capacities of double-sided 8-inch disks:

	Single density	Double density
Double-sided	490 kilobytes	980 kilobytes

The file capacity of a formatted Winchester disk partition is determined before you invoke the FORMAT utility, by the PREP utility (which automatically allocates half of the Winchester disk space to a single CP/M partition), or by the PART utility (which enables you to determine the size of your partitions). See the sections on "PREP" and "PART" in the Winchester User's Supplement for more information.

NOTE: The FORMAT utility supplied with this CP/M version cannot format any disk to extended double density. However, this CP/M version can write to or read from a disk that was formatted at extended double density by the FORMAT utility of some other CP/M versions.

FORMAT Error Messages

```
Drive not available in current configuration
```

EXPLANATION: If you entered a drive name that has not been assigned a partition, or does not exist in your hardware, enter a different drive name.

```
Disk is write protected.
Unable to format this disk.
Do you have any more disks to format? (y/n):
```

EXPLANATION: Remove any adhesive tab that might be attached to the write-enable notch on a 5.25-inch disk cover, or cover the write-protect notch on an 8-inch disk. The `Do you have any more disks to format?` prompt appears with this message only when you enter a Utility Prompt Method command.

FORMAT

```
Unable to format this disk. Place a different disk in the
drive and press any key to begin...
```

EXPLANATION: The disk to be formatted is damaged or improperly inserted in the drive. Try again or replace the disk.

```
ILLEGAL FORMAT OPTION
```

EXPLANATION: System Prompt Method command line was entered with illegal values for options.

```
ILLEGAL COMMAND SYNTAX
```

EXPLANATION: System Prompt Method command line was entered with undefined characters in place of options.

```
OPTION NOT AVAILABLE
```

EXPLANATION: A Utility Prompt Method prompt was answered with inappropriate characters. Enter a pertinent option at this prompt.

```
PARTITION IS SMALLER THAN MINIMUM ALLOWABLE SIZE
```

EXPLANATION: Winchester disk partitions under CP/M can be as small as 64 kilobytes. Back up any valuable software and/or data stored on other partitions and use the PART utility to create CP/M partitions that are at least 64 kilobytes in size. Then try the FORMAT utility again.

```
PARTITION IS LARGER THAN CP/M MAXIMUM SIZE -- ONLY 8 MEG USABLE
```

EXPLANATION: The partition being formatted has been allocated 8 megabytes (8192 kilobytes) or more Winchester disk space. You will only be able to access the first 8192 kilobytes on the partition. To make the rest of the partition's space accessible, back up any valuable software and/or data stored on other partitions and use the PART utility to make the partition smaller and to allocate the remaining space to another partition. Then try the FORMAT utility again.

```
Incorrect version of BIOS
```

EXPLANATION: The CP/M-86 system and FORMAT utility being used are not of the same version. Use a CP/M-86 system and FORMAT utility that were both copied from the same CP/M-86 distribution disk. Then try the FORMAT utility again.

```
Not enough memory
```

EXPLANATION: There is not enough free memory in your computer system to run FORMAT. FORMAT is automatically aborted and control returned to the operating system.

LDCOPY

The Utility that Copies the CP/M-86 Operating System between Disks

Use the LDCOPY utility to create a bootable CP/M-86 disk. You may copy CP/M-86 from the system tracks of any bootable disk/partition if you are transferring the system between like devices (from floppy to floppy or from disk partition to disk partition). If you are transferring the system between unlike devices (from floppy to disk partition or from disk partition to floppy), you may copy CP/M-86 by using the .CMD file(s) named BOOT207 and BOOT217. Both of these copy operations are detailed below.

The CP/M-86 BIOS file, CPM.SYS, must also be present on a bootable disk. LDCOPY has no effect on CPM.SYS. To transfer it you must use the PIP utility (see PIP in the "User's Guide").

You may execute LDCOPY by either of two methods: the Utility Prompt method or the System Prompt method. With the Utility Prompt method, LDCOPY is first loaded into memory. You then enter additional parameters in response to specific prompts. This method must be used if you intend to transfer the system to more than one disk. With the System Prompt method, LDCOPY is loaded and all necessary parameters are supplied from a single command line. A detailed description of each method follows.

Utility Prompt Method

Place a disk containing LDCOPY.CMD in a disk drive (A is assumed in this example) and enter:

```
A>LDCOPY
```

When loading is complete you will see this sign-on message:

```
            LDCOPY rel x.xx
Copyright (c) 1983 Zenith Data Systems Corp.
```

Below the sign-on message the "Source:" prompt will appear:

```
Source:
```

The entry for this response depends upon whether you are transferring the system between like devices (floppy to floppy or disk partition to disk partition) or unlike devices (floppy to disk partition or disk partition to floppy). If you are transferring the system between like devices, use *Response 1* for making the entry to the `Source:` prompt. If you are transferring the system from floppy to disk partition, use *Response 2* for making the entry to the `Source:` prompt. And if you are transferring the system from disk partition to floppy, use *Response 3* for making the entry to the Source: prompt.

Response 1 – From floppy to floppy or disk partition to disk partition. If you are transferring a system between like devices, enter the letter of the drive/partition where the system you want to copy is currently located. For example:

```
Source:B
```

and press **RETURN**.

Response 2 – From floppy to disk partition. If you are transferring the system from a floppy disk to a disk partition, you must use the .CMD file named BOOT217. To use this file, respond to the `Source:` prompt by entering the letter of the drive/partition on which this file resides and the name of the file, BOOT217.CMD. For example:

```
Source:B:BOOT217.CMD
```

and press **RETURN.**

Response 3 – From disk partition to floppy. If you are transferring the system from a disk partition to a floppy (either 5.25 inch or 8 inch), you must use the .CMD file named BOOT207. To use this file, respond to the `Source:` prompt by entering the letter of the drive/partition on which BOOT207.CMD resides and the file name. For example:

```
Source:B:BOOT207.CMD
```

and press **RETURN**.

When you have made the appropriate entry to the `Source:` prompt, the destination prompt will appear:

```
Destination:
```

Enter just the letter of the drive where the destination disk will be located. No file name is necessary since the data will be transferred to the system tracks. For example:

```
Destination:A
```

and press **RETURN**.

At this point LDCOPY will verify your entries with the message:

```
Source on B
Destination on A
Press RETURN to continue;
otherwise enter CTRL-C to abort:
```

If the information is incorrect or you wish to abort the procedure, enter **CTRL-C**. Control will return to the operating system. If you press **RETURN** this prompt will appear:

```
Press RETURN when source disk is in drive B:
```

Switch disks, if necessary, to ensure the source disk is in the correct drive; then press **RETURN**. Ensure the destination disk is in the correct drive when this prompt appears:

```
Press RETURN when destination disk is in drive A:
```

Two pauses are included so LDCOPY can be used with computers that have only one drive. Press **RETURN** to continue. When the transfer process is complete, this message will appear:

```
LDCOPY successfully completed.
Press RETURN when destination disk is in drive A;
otherwise enter CTRL-C to abort:
```

If an error message is displayed instead, refer to Error Messages. The destination prompt is repeated to allow you to make any number of bootable disks. Enter **CTRL-C** when you are ready to terminate the LDCOPY session.

System Prompt Method

LDCOPY can be loaded and executed by entering a single command line of the form:

```
LDCOPY destination=source [switch]
```

To use this method, insert a disk containing LDCOPY.CMD in a disk drive (A is assumed in this example). Enter LDCOPY followed by a space. If you are transferring a system that currently resides on the system tracks of the source disk (i.e., it is a bootable disk), enter the letter of the destination drive (A-P) with a colon, an equal sign, and the letter of the source drive with a colon. For example:

```
A>LDCOPY A:=B:
```

If you are transferring a system from floppy to partition or from partition to floppy, enter the same command, but follow it with the proper file name. No extension other than .CMD is allowed. For example:

```
A>LDCOPY A:=B:BOOT207.CMD
```

You may add a switch in the form [N] to either command line (e.g., LDCOPY B:=A: [N]). This will force immediate execution of the command without further prompting. Obviously, both disks must be in the correct drives at the time the command is entered if this option is used.

If the switch is omitted, LDCOPY will verify your command with this message:

```
                    LDCOPY rel. x.xx
         Copyright (c) 1983 Zenith Data Systems Corp.

Source on B
Destination on A
Press RETURN to continue;
otherwise enter CTRL-C to abort:
```

If the information is incorrect or you wish to abort the procedure, enter **CTRL-C**. Control will return to the operating system. If you press **RETURN** this prompt will appear:

```
Press RETURN when source disk is in drive B:
```

Switch disks, if necessary, to ensure the source disk is in the correct drive, then press **RETURN**. Ensure the destination disk is in the correct drive when this prompt appears:

```
Press RETURN when destination disk is in drive A:
```

Two pauses are included so LDCOPY can be used with computers having only one drive. Press **RETURN** to continue. When the transfer process is complete a verification will appear and control will return to the operating system:

```
LDCOPY successfully completed.
A>
```

If an error message appears instead, refer to Error Messages.

Help Screen

You may display a condensed version of these instructions by entering a question mark (?) after any prompt, or after LDCOPY in a command line (e.g., A>LDCOPY ?). If you enter a question mark, the following text will appear on your screen.

```
        Use LDCOPY to create a bootable CP/M-86 disk.
    (You must also use PIP to copy the BIOS file, "CPM.SYS")

_____________________________________________________________
UTILITY PROMPT METHOD:                     EXAMPLES:
 If you are copying from system tracks,
 answer the Source: and Destination:       Source:B
 prompts with just a drive letter.         Destination:A

 If you are copying from a .CMD file,
 answer the Source: prompt in the form:    Source:B:Boot207.CMD
 {drive} : {filename.CMD}                 Destination:A
 Answer other prompts with RETURN to
 continue, or CTRL-C to abort.
_____________________________________________________________
SYSTEM PROMPT METHOD:
 Enter one command line in the form:

     LDCOPY{dest.drive}: ={source drive}: [filename.CMD]
```

```
EXAMPLES:
     A>LDCOPY B:=A:
     B>A:LDCOPY B:=A:BOOT207.CMD
```

Error Messages

```
Insufficient memory
```

EXPLANATION: There is not enough free memory available in your computer system to run LDCOPY. Control is returned to the operating system.

```
Invalid command line format
```

EXPLANATION: A syntax error was made while you entered parameters.

```
Invalid Destination
```

EXPLANATION: You entered a drive letter outside the range A-P, or you specified a drive not available on your system.

```
Invalid Source
```

EXPLANATION: You entered a file name that was misspelled or not available on the specified drive, or you entered a drive letter outside the range A-P, or you specified a drive not available on your system.

```
LDCOPY is incompatible with current BIOS
```

EXPLANATION: The CP/M-86 system and the LDCOPY utility being used are not of the same version. Use a CP/M-86 system and LDCOPY utility copied from the same distribution disk. Then try LDCOPY again.

LIST

The Utility that Prints Text File Contents on Paper

The LIST utility enables you to obtain paper copies of files by entering a command for one or more files to be printed. Special printout characteristics can be set when you enter the command with LIST parameters. You can stop a LIST printout in progress. LIST is used to print out only certain types of files.

Method of Entering LIST Commands

Two different methods can be used to enter LIST commands: the Utility Prompt Method and the System Prompt Method.

Utility Prompt Method

With the Utility Prompt Method, you enter LIST in response to the system prompt. This entry is sufficient to invoke LIST, which displays its own prompt—the asterisk (*). You can now enter the argument portion of the command line in response to the "*" prompt supplied by LIST. LIST prompt entries are made in the following form:

```
A>LIST RETURN

*{argument} RETURN
```

Where {argument} is the name of the file(s) to be listed.

After the LIST operation is finished, LIST again displays the "*" prompt. You can enter another argument or return to the operating system by pressing **RETURN**.

System Prompt Method

To invoke a LIST operation with a single command line, you must include the argument in the response to the system prompt, as in the following example:

```
A>LIST {argument} RETURN
```

Where {argument} is the name of the file(s) to be listed.

After LIST finishes the latter printing operation, it automatically returns you to the operating system, and the system prompt is displayed.

For example, if you enter the following command using the Utility Prompt Method:

```
A>LIST RETURN

*REPORT.DOC RETURN
```

or this command using the System Prompt Method:

```
A>LIST REPORT.DOC RETURN
```

the results will be the same. Both commands will produce a paper copy of the file named "REPORT.DOC."

Printing Contents of More than One File

You can specify several files in a single LIST command. To initiate a listing of several files, enter the names of the files in the argument, separated by single spaces.

If the files to be listed reside on different disks, their names should be preceded by a drive specification (drive letter and colon).

Both of the following examples demonstrate how to LIST the contents of the specified files:

```
A>LIST RETURN

*PRINTOUT.DOC B:PROGRAM.PRN B:REPORT.DOC C:SYSTEMX.PRN RETURN
```

or

```
A>LIST PRINTOUT.DOC B:PROGRAM.PRN B:REPORT.DOC C:SYSTEMX.PRN RETURN
```

LIST Parameters

You can specify parameters in a LIST argument to alter the characteristics of a standard printout. These parameters allow you to select printout characteristics such as the date, the number of copies desired, the width of tabs, etc. If no parameters are specified, LIST will print a document with default value characteristics.

The parameters used to specify printout characteristics are shown in the following table.

PARAMETER NAME	KEYBOARD ENTRY	DESCRIPTION OF PRINTOUT CHARACTERISTIC	DEFAULT VALUE
Date	[D xxx...x]	First 10 characters of specified date printed on upper right corner of each document page.	no date
Heading	[H xxx...x]	First 60 characters of specified heading printed on the left of top line of each document page.	file name (first copy only)
No Heading	[N]	No heading or date printed on any document page.	file name
Lines per Page	[L nn]	Each document page has the specified number of lines. Specification of zero lines/page causes printing without page breaks.	60 lines per page
Tab Stop Width	[T nn]	Each tab stop within text is expanded or contracted to specified number of spaces.	8 spaces
Page Number	[P nn]	Page numbering sequence begins with specified number on first file page.	page 1
Uppercase	[U]	All letters in document printed in uppercase.	upper- and lower-case
Copies Desired	[C nn]	Specified number of copies are printed.	1 copy
Erase	[E]	File is erased from disk after LIST operation is completed.	file retained on disk

Use of LIST Command Line Parameters

Parameters are entered, enclosed in square brackets, after the last file name in the LIST command. If more than one parameter is entered, each should be enclosed in a set of brackets.

A LIST command entered with parameters using the Utility Prompt Method might appear as follows:

```
A>LIST RETURN
*PRINTOUT.DOC PROGRAM.PRN [H Today's Work] [D 31-Feb-81] [P 9] RETURN
```

This command will cause the contents of the files PRINTOUT.DOC and PROGRAM.PRN to be printed, with the following heading across the top of the first page of each file:

```
Today's Work                                31-Feb-81  Page 9
```

The parameters will take effect on all of the files specified in that command line.

When you enter the command with the Utility Prompt Method, any parameters entered (except for the starting Page number parameter and Erase parameter) will remain in effect with any files specified at a successive asterisk (*) prompt, until new values are entered for the parameters or until control is returned to the operating system.

When you enter a LIST command line with the System Prompt Method, letters in the heading, date, and page number line will be automatically translated into uppercase. The following System Prompt command demonstrates this character translation:

```
A>LIST PRINTOUT.DOC PROGRAM.PRN [H Today's Work] [D 31-Feb-81] [P 9] RETURN
```

This command will cause the contents of the files PRINTOUT.DOC and PROGRAM.PRN to be printed with the following heading across the top of the first page of each file:

```
TODAY'S WORK                                31-FEB-81  PAGE 9
```

Aborting a LIST Operation

After a LIST command has been entered, the printout can be aborted by pressing any keyboard character while the printout is in progress. (If more than one copy has been requested, you must abort each copy.) LIST will then redisplay the asterisk prompt (under the Utility Prompt Method), or CP/M will display the system prompt (under the System Prompt Method).

Files that Should Be Listed

Only files containing ASCII characters should be listed. ASCII character files include files composed using a text editor (such as ED), or files with the "PRN" extension created by an assembler (such as ASM).

Files not composed of ASCII characters will produce meaningless printouts if listed.

Files composed in a word processor will LIST, but they might possess features (such as boldface characters or page breaks) that LIST will not print in the same way as the word processor's printout command.

LIST Error Messages

```
File not found
```

EXPLANATION: A file specified in the LIST command argument does not exist on the disk in the logged drive. If the file does exist on a disk, that disk should be inserted in a drive and the appropriate drive should be specified before the file name in the argument.

```
Syntax error in command line
```

EXPLANATION: The LIST command was improperly entered. This error often occurs if a space is not entered between file names, or when an invalid parameter is specified.

RDDOS

The Utility that Allows CP/M-86 to Read a Z-DOS Formatted Directory or File.

Use the RDDOS utility to read a file directory on a Z-DOS formatted disk or to copy Z-DOS formatted data files into a CP/M readable format. When reading a directory on a Z-DOS formatted disk, RDDOS simply displays the directory in Z-DOS directory form while leaving the Z-DOS files themselves unaffected. When copying data files from a Z-DOS formatted disk, RDDOS makes CP/M compatible copies of the files while leaving the original Z-DOS formatted files in tact.

You may execute either the RDDOS copy function or the RDDOS directory function using only the System Prompt method. With the System Prompt method, RDDOS is loaded into memory and all necessary parameters are supplied from a single command line.

In addition, there is both a brief and detailed version of the RDDOS help screen that you may invoke from the system prompt. A detailed description of these help screens and of the two primary RDDOS functions follows.

Reading a Z-DOS Directory

To read a file directory from a Z-DOS formatted disk, place a disk containing RDDOS.CMD into a drive (e.g., drive A) and enter a command line of the following form:

```
A>RDDOS DIR <d:>[<filespec>]
```

where `RDDOS DIR` instructs the system that you wish only to read a directory on a Z-DOS formatted disk;

where `<d:>` is the drive name where the Z-DOS disk resides. This is a required entry since the Z-DOS disk will always reside on a drive other than the default drive;

where `[<filespec>]` is any legal Z-DOS filename and extension that represent the file(s) you wish to see in the directory. You may also use wild cards. `[<Filespec>]` is an optional entry. If you do not enter a filename, RDDOS will display the entire Z-DOS directory.

When you have completed this entry by pressing **RETURN**, the RDDOS banner will display on the screen:

```
                RDDOS Rel. x.xx
    Copyright (c) 1983 Zenith Data Systems Corp.
```

Below the banner, the directory of the Z-DOS formatted disk will appear on your screen in the five columns of the Z-DOS directory form: filename, extension, file size, date, and time. When you have finished reviewing the directory, you may return to the system prompt `A>` by pressing **RETURN**.

Copying Files from Z-DOS Format into CP/M Format

To copy Z-DOS formatted files to CP/M formatted files, place a disk containing RDDOS.CMD into a drive (e.g., drive A) and enter a command line of the following form.

```
A> RDDOS <d:>[filename.ext][=]<e:><filespec>
```

where `RDDOS` followed by the parameters instructs the system that you wish to copy Z-DOS formatted files into CP/M format;

where `<d:>` is the destination drive name. The destination drive should contain the CP/M formatted disk on which the copied file(s) will reside. This entry is required since the system must be told where to find the CP/M formatted disk.

where `[filename.ext]` is any legal filename and extension for the copy that will be created on the CP/M destination disk. This entry is also optional. If you do not specify a filename and extension, RDDOS will simply adopt the original filename and extension of the Z-DOS formatted file.

where <=> is required to separate the destination parameters from the source parameters.

where <e:> specifies the source drive. The source drive should contain the Z-DOS formatted disk with the file(s) you wish to copy. This entry is required since the system must be told where to find the Z-DOS formatted disk.

where <filespec> is any legal filename and extension which represents the Z-DOS formatted file(s) you wish to copy to CP/M format. You may also use any valid combination of wild cards for the filespec.

When you have completed this entry by pressing **RETURN**, RDDOS will display its banner while it completes the copying function. When copying is complete, RDDOS will automatically return to the system prompt A>.

Help Screen

Two versions of the Help screen may be invoked from the system prompt. The first version briefly illustrates the correct syntax for the RDDOS directory and copy functions. The second version provides more detailed explanation of when and how to use RDDOS.

The Brief Help Screen

To invoke the brief version of Help, type the following:

```
A>RDDOS
```

and press **RETURN**. The following message appears on your screen:

```
                RDDOS Rel  x.xx
    Copyright (c) 1983 Zenith Data Systems Corp.

         Usage:  RDDOS [destination] source
          --or-- RDDOS DIR drive
```

When you have finished reviewing this help message, pressing **RETURN** will return you to the system prompt.

The Detailed Help Screen

To invoke the detailed version of help, type the following:

```
A>RDDOS ?
```

and press **RETURN**. The question mark entry tells the system that you want it to display the detailed version of Help. The following message appears on your screen:

```
                    RDDOS Rel. x.xx
      Copyright (c) 1983 Zenith Data Systems Corp.

   RDDOS reads a file, file group, or directory from
a disk that has been formatted using a Z-DOS compatible
system and will copy the file(s) to a disk havingbeen
formatted with CP/M.

   The RDDOS command is similar to the PIP command
except that the source file(s) are expected to be on a
Z-DOS disk.

Usage:  RDDOS [destination] source
 --or-- RDDOS DIR drive

Where "source" is a Z-DOS drive name and filespec, and
"destination" a CP/M drive name and/or filespec.
```

When you have finished reviewing the detailed help message, you may return to the system prompt by pressing **RETURN**.

Error Messages

```
   Bad destination drive specified.
```

EXPLANATION: The drive name specified for the destination is outside the possible range of drives (A-P) or the drive is not available on your system. RDDOS will automatically abort. You may then rerun the program, entering the correct drive name.

```
Bad source drive specified
```

EXPLANATION: The drive name specified for the source is outside the possible range of drives (A-P) or the drive is not available on your system. RDDOS will automatically abort. You may then rerun this program, entering the correct source drive name.

```
Bad destination filename specified
```

EXPLANATION: The filename you have specifed for the destination is invalid (e.g., it has too many characters or it contains characters that may not be used). RDDOS will automatically abort. You may then rerun the program, using a valid filename for the destination.

```
Bad source filename specified
```

EXPLANATION: The filename you have specified for the source is nonexistent. RDDOS will automatically abort. You may then rerun the program, using the correct filename.

```
Bad memory request...allocation denied
```

EXPLANATION: The computer system does not have enough free memory to run the RDDOS program. RDDOS will automatically abort.

```
Bad destination...directory full
```

EXPLANATION: There is not enough room on the destination disk to copy the file. RDDOS will automatically abort. You may then rerun RDDOS, using a disk with sufficient space to receive the copy.

```
Bad command line syntax
```

EXPLANATION: A syntax error was made while you entered parameters. RDDOS will automatically abort. You may then rerun the program, using the correct syntax.

RESTORE

The Utility that Facilitates File Copying to Disks and/or Partitions for Active Use

The RESTORE utility makes it possible for you to access working files from the backup files produced by the BACKUP utility, and to copy them back to the media you use for your daily microcomputer tasks.

RESTORE is especially beneficial to Winchester disk users, because Winchester disk partitions hold so many files.

Because RESTORE works with both floppy disks and Winchester partitions, this text refers to recording media as "partition/disk" or "disk/partition" wherever it would be practical for you to use either a floppy disk or a Winchester partition.

RESTORE's function is complementary to the function of the BACKUP utility, which makes optimal use of disk space by storing working files inside of a special backup file.

RESTORE Operation

When the BACKUP utility copies individual files from a source partition/disk, it temporarily concatenates them to be stored as a single "backup file" on the destination medium. Thus the backup file is a long, continuous string of data including individual files that are joined by BACKUP and separated by RESTORE. The backup file also contains a directory that lists all of the files in the exact sequence they were copied.

Often BACKUP stores parts of the backup file on more than one disk. BACKUP accomplishes multi-disk storage by recording up to the absolute capacity of the distribution medium, prompting you to insert another disk, and then continuing to copy from where it left off. With this capability, BACKUP can even divide a file within the backup file between two disks. Each of the disks necessary to accommodate the entire backup file is called a "volume" of the backup file and is given a volume number.

The RESTORE and BACKUP utilities also enable you to view several statistics, such as the names of the files within a backup file, by entering special command line options.

The BACKUP utility informs RESTORE of the sequence used to store backed up files by listing this sequence in the directory of the special backup file. Thus when RESTORE begins to restore files from the backup disk set, it reads the list of files from the backup file directory and restores them to the original medium in the listed sequence.

Invoking RESTORE

You can use the RESTORE utility through any of the following three methods: the Utility Prompt Method, the System Prompt Method, or the Help Display Method.

Utility Prompt Method

With the Utility Prompt RESTORE Method, you invoke the RESTORE utility from a disk by entering the command function at the system prompt and then entering the command argument at a prompt displayed by the RESTORE utility. The first entry under this method is in the following form:

```
A>RESTORE RETURN
```

After an entry in this form, RESTORE will display a message and prompt in the following form:

```
                RESTORE Version x.xx
           Copyright (C) 1983 Zenith Data Systems

>>
```

NOTE: The version number of the RESTORE utility (shown in this example as "1.1.100") might vary.

At the >> RESTORE prompt, you should enter a command line argument in the following form:

```
>>{d}:{distfile},{d}:{distfile},... ={s}:{sorcfile}[{x};...] RETURN
```

RESTORE

Where {>>} is the RESTORE utility prompt;

where {d} is the optional name of the drive that is to receive the copies being transferred. This specification is necessary only if this destination drive is not also the default drive;

where {destfile} is one of the individual file names that is being extracted from within the backup file and copied to the destination medium, where it will (once again) be accessible by its own file name;

where {s} is the optional name of the drive from which the backup file is being copied (necessary only if the source drive is not also the default drive);

where {sorcfile} is the primary name of the backup file. Wild card file name characters (* and ?) are not allowed; and

where {x} is any of the optional single letter options, separated by semicolons, that structure the RESTORE operation.

When RESTORE has completed the specified operation, it will redisplay the >> RESTORE prompt. You can continue entering command line arguments at RESTORE prompts indefinitely.

During the RESTORE operation, the RESTORE utility will display the names of each file that it copies in a vertical list.

When you wish to exit from the RESTORE utility to the operating system, press **RETURN** at a RESTORE prompt. Then CP/M will display the A> system prompt.

NOTE: Like all command lines entered through CP/M, the RESTORE command line can contain only 127 characters. If your command line is between 78 and 127 characters in length, you can keep the entire line visible on your video screen by entering **CTRL-E** after the 79th character.

System Prompt Method

With this RESTORE method, you enter the command line function and the command line argument both at the system prompt, in the following form:

```
A>RESTORE {d}:{destfile},{d}:{destfile},...={s}:{sorcfile} [{x};{x};...] RETURN
```

Where RESTORE is the command line function, stored in the file RESTORE.COM on the default or logged partition/drive;

where {d} is the optional name of the drive that is to receive the copies being transferred (necessary only if the destination drive is not also the default drive);

where {destfile} is one of the individual file names that is being extracted from within the backup file and copied to the destination medium, where it will (once again) be accessible by its own file name;

where {s} is the optional name of the drive from which the backup file is being copied. This specification is necessary only if this source drive is not also the default drive;

where {sorcfile} is the primary name of the backup file. Wild card file name characters (* and ?) are not allowed; and

where {x} is any of the optional single letter options, separated by semicolons, that structure the RESTORE operation.

During the RESTORE operation, the RESTORE utility will display the names of each file that it copies in a vertical list.

After RESTORE has completed the specified operation, CP/M will display the A> system prompt.

NOTE: Like all command lines entered through CP/M, the RESTORE command line can contain only 127 characters. If your command line is between 78 and 127 characters in length, you can keep the entire line visible on your video screen by entering **CTRL-E** after the 78th character.

Help Display Method

With this RESTORE method, you enter the command line function and a ? (question mark) at the system prompt, as shown:

```
    A>RESTORE ? RETURN
```

This command will cause the display of messages that summarize the purpose, command line, and options of the RESTORE utility. The system prompt will reappear below the RESTORE display.

This invocation method does not cause restoration of backed up files. It is designed to provide you with a convenient quick reference to a few aspects of the RESTORE utility.

RESTORE Source

A RESTORE command line can include only one source. When specifying this source, you must always specify the primary name of a file, and sometimes a drive.

The file specified as the source is the backup file into which all of the individual files were copied.

The source file specified in a RESTORE command line should be identified by a primary file name only (1-8 characters). The extension should not be specified, because the BACKUP utility has already applied extensions to each backup file volume. (BACKUP applies the extension "000" to the first backup file volume, "001" to the second, "002" to the third, etc.) During the RESTORE operation, you will be prompted to insert backup file disk volumes at appropriate times.

Wild card file name characters (* or ?) cannot be used to specify a source file.

Whenever a specified source (backup file) does not reside in the default drive, a drive name should be specified in front of it. You can restore files to any valid drive during a RESTORE operation.

A source file specification is required in all RESTORE command lines except those entered with the [B] option, for the sole purpose of displaying characteristics of all of the master backup files on the default disk.

Consider an example where the primary file name "C:QUITTIME" is entered as source in a command line. The ensuing RESTORE operation required that five disks (backup file volumes) be inserted into drive C to completely disperse all of the individual files that had been stored within QUITTIME.

The source disks contained the following backup file volumes:

```
QUITTIME.000
QUITTIME.001
QUITTIME.002
QUITTIME.003
QUITTIME.004
```

The file "QUITTIME.000" was the master backup file volume, which contained the backup file directory.

RESTORE Destinations

A RESTORE command line can include one or more destinations. When specifying destinations, you must always specify a file, and sometimes a drive.

The destination files specified in a RESTORE command line should be identified by complete file names, including the primary name (1-8 characters) and the extension (1-3 characters if used).

Whenever a specified destination file does not reside on the default drive, a drive name should be specified in front of it. You can restore files to any valid drive during a RESTORE operation.

Wild card, or ambiguous, file names can be specified, using the * or ? wild card characters. (Wild card file names can be particularly useful when you have many files to copy, because a command line can contain a maximum of only 127 characters.)

Any number of destination files can be specified in a command line (as long as the limit of 127 characters per command is not exceeded). However if two or more destination files are specified, they must be separated by commas.

A destination file specification is required in all RESTORE command lines except those entered with the [B] option, for the sole purpose of displaying characteristics of all of the master backup files on a disk; or in those entered with the [L] option, for the sole purpose of displaying a list of the individual files within a backup file.

The destination file specification

```
*.ASC,TEST?,C:*.*
```

would cause the release (from within the backup file) of all files with an .ASC extension to the default drive; all files (from the backup file) that are five characters long beginning with TEST and that have no extension (such as TEST0, TEST1, TESTS, TESTY, etc.) to the default drive; and all files (from the backup file) to drive C.

To further demonstrate the characteristics of destination file specifications:

```
A:DEMO.*,B:SYSTEM.COM,E:82*.DOC,????.DAT
```

would cause the release (from within the backup file) of all files with DEMO as the primary name and any extension to the disk/partition in drive A; the file SYSTEM.COM to drive B; all files (from the backup file) that have a primary name beginning with 82 and that have a .DOC extension to drive E; and all files (from the backup file) that have a four-letter (or shorter) primary name with a .DAT extension to the default drive.

NOTE: Many of the individual files stored within the backup file may be identified by both the name of the file and the name of the drive from which they were originally copied. However, the drive name stored with the file name in the backup file does not affect the RESTORE operation in any way. In a RESTORE command line, the drive names you specify for destination files identify the drives to which you want the individual files to go as they are released from the backup file.

RESTORE Options

The RESTORE utility enables you to structure any backup operation by specifying the following options in the command line:

B	Backup directory—displays a directory listing statistics about all master backup files.
E	Exception files—exclude exception files from RESTORE operation.
L	List directory—list the directory of the backup files.
O	Overwrite files—overwrite any existing files with the same name.
Q	Query each—query yes or no on each file before restoring.
R	Read only—overwrite read/only (R/O) files without prompt.
S	System files—restore system files if specified.
U	User number—restore files that were backed up from specified user area(s) in addition to the current user area.
V	Verify—verify the file after restoring it.

Options are always the last items specified in a RESTORE command line. They must be enclosed within square brackets and separated by semicolons when more than one is used.

If a switch requires specification of files, then the option letter should be separated from these file specifications by a colon. When more than one file specification is necessary with a single option, then the file specifications should be separated by commas.

Options are entered in the following form:

```
[{x};{x}:{details},{details},{details};{x};...]
```

Where {x} is an option letter;

where {details} are additional character strings (such as the date, file names, or user numbers) that must be specified with some options;

where the [ ] (square brackets) must enclose the options;

where the ; (semicolon) must separate multiple options;

where the , (comma) must separate the file names and/or user numbers used with some options; and

where the : (colon) must separate some option letters from accompanying {details}.

B—BACKUP Directory

The B option causes a directory display listing statistics about all of the master backup files (those backup files with the "000" extension) on a specified disk.

This directory lists the primary name of each master backup file, the number of volumes in the backup set that begins with each master, the number of files in each set of backup files, and the date of each backup operation.

Only the drive containing the master backup file(s) needs to be known in command lines entered for the sole purpose of producing a B option directory. Therefore, specify this drive in the command line unless it is the default drive.

You can use the B option in commands of the following form:

```
A>RESTORE {d}: [B] RETURN
```

Where RESTORE is the command line function, stored in the file RESTORE.COM on the default or logged partition/drive;

where {d} is the optional name of the drive that has received the copied files and stored them within backup file volumes (necessary only if the destination drive is not also the default drive); and

where [B] is the single-letter option that causes the display of master backup file features.

The B option may be used with both methods of invoking RESTORE. For example, from the CP/M system prompt:

```
A>RESTORE C: [B] RETURN
```

or the same operation from the RESTORE utility prompt:

```
>>C: [B] RETURN
```

The B directory listing would appear in the following form:

```
Name          Volumes        Files      Date
QUITTIME         3            117      4-27-83
SAVEME           1             48     10-24-82
STORAGE          5            231      2-30-84
```

E—Exception Files

The RESTORE operation takes place for all of the files except that file given as an exception file. The file listed with the E option is then ignored during the operation. The E option is entered in the following form:

```
[E:{filespec},{filespec}]
```

Where E is the option;

where {filespec} represents files that are to be excluded from the operation (if they do not reside on the default drive, then you must also specify the name of the drive that contains them); You may use wild cards * and ? to specify groups of files; and

where the , (comma) and : (colon) are required separators.

For example:

```
A>RESTORE BACK1=*.DAT [E:TEMPFILE.DAT] RETURN
```

would RESTORE all files from the default drive that have a .DAT extension except the file named TEMPFILE.DAT, which would be omitted.

L—List Directory

The L option causes RESTORE to give the internal directory of files that are contained within a specified backup file. The directory information would list the volume number where the files start and end, and the size of the file.

The L option is used for commands in the following form:

```
A>RESTORE {d}:{sorcfile} [L] RETURN
```

Where RESTORE is the command line function, stored in the file RESTORE.COM on the default or logged partition/drive;

where {d} is the optional name of the drive that has received the copies being transferred. This specification is necessary only if this drive is not also the default drive;

where {sorcfile} is the primary name of the backup file, which you wish to contain all of the individual files copied in this operation; and

where {L} is the single-letter option that causes the display of backup file directory features.

The L option may be used with both methods of invoking RESTORE. For example, from the CP/M system prompt:

```
A>RESTORE BACK9 [L] RETURN
```

or the same operation from the RESTORE utility prompt:

```
>>BACK9 [L] RETURN
```

The L directory listing will appear in the following form:

```
Filename        User     Start     End          Size in
                Area     Volume    Volume       Kilobytes
TESTFIL1.DAT     0         1         1            3264
TESTFIL2.DAT     0         1         2           19582
TESTFIL1.DOC    11         2         3            7236
TESTFIL2.DOC    11         3         3           22230

4 file(s) on 3 volume(s)
```

O—Overwrite Files

During operation, if RESTORE encounters a file with the same file name as the backup file, RESTORE asks if the file should be deleted with a message in the following form:

```
File {filename} already exists, do you wish to delete it (Y/N)?
```

When the O option is used, RESTORE overwrites the original copy of the file on the destination medium (the copy that "already exists") without the query.

Q—Query Each

When the Q option is used, you are queried before each file is restored by a prompt in the following form:

```
Restore {x}:{destfile} to user {n} (Y/N)?
```

Where {x} is the drive containing an individual file that you specified for this operation (only displayed if it is not the default drive);

where {destfile} is an individual file from within the backup file that you specified for restoration; and

where {n} represents the number (0-15) of the user area to which the {destfile} will be restored.

A prompt in this form is displayed for each file RESTORE encounters that is contained within the backup file you specified as the source.

For example, if the current user area were 0 and you wanted to restore two .DOC files from the "BACK1" backup file on drive C to their individual file names in user area 0, and if the command to RESTORE were:

```
A>RESTORE B:*.DOC=C:BACK1 [Q] RETURN
```

then RESTORE would ask:

```
Restore B:FILE1.DOC to user 0 (Y/N) ?
```

After you respond it would ask,

```
    Restore B:FILE2.DOC to user 0 (Y/N) ?
```

In response to such a prompt, press **Y** or **y** to cause the file to be restored. Press **N** or **n** to prevent the file from being restored.

R—Read/Only Files

The R option enables RESTORE to overwrite individual files that have read/only (R/O) status on the destination media without displaying a prompt.

As a default (in the absence of the R option), the following prompt is displayed if you are restoring individual files to destination medium where files by the same name exist with the read/only (R/O) status:

```
    File {file name.ext} read only (R/O), do you wish to delete it (Y/N) ?
```

Where {file name.ext} is the complete file name of the file you are trying to restore.

NOTE: The STAT utility is used to apply read/only (R/O) or read/write (R/W) status to a file, as explained in the "STAT" text in the "User's Guide."

S—System Files

The S option allows RESTORE to restore files that are displayed in the directory (by the STAT utility) as system files. System files will not be restored merely because of this option, but only if this option is used in a command line where system files are specified as destinations.

If you do not include the S option in the command line, system files are ignored during the RESTORE operation—even if you specify these files as destinations in the RESTORE command line.

U—User Number

The U option causes RESTORE to restore only the individual files within the specified user area(s). If you do not include a U option in the RESTORE command, then RESTORE will restore only the individual files to the current user area.

NOTE: CP/M enables you to divide each of your disks and partitions into 16 separate user access areas, numbered 0-15. When you boot up a partition or disk, you are working within user area 0 until you enter a USER command. See the "USER" text in the "User's Guide" for more information on user areas.

Specify the user option in the following form:

```
[U:n1,n2,...]
```

Where U is the option letter;

where : (colon) must separate the option letter from the user number(s); and

where n1 stands for the number of one of the user areas to which RESTORE will restore individual files. If you specify more than one user area number, separate each with a comma.

You can specify as many as 16 user areas with a single U option. You can specify any user area(s) regardless of the user area currently being used.

V—Verify

The V option causes RESTORE to verify all files copied. With the V option, RESTORE reads each source file after it is copied to make sure that the source and destination copies are identical.

During a verified RESTORE operation, the names of the copied and verified files will be displayed in the following form:

```
E:FILENAM1.EXT to user 12
Verifying E:FILENAM1.EXT
E:FILENAM2.EXT to user 12
Verifying E:FILENAM2.EXT
E:FILENAM3.EXT to user 12
VerifyingE:FILENAM3.EXT
E:FILENAM4.EXT to user 12
Verifying E:FILENAM4.EXT
          .
          .
          .
E:FILENAMn.EXT to user 12
Verifying E:FILENAMn.EXT
```

Runtime Prompting

After you invoke RESTORE and enter a command, RESTORE should begin to restore files, displaying the drive name and file name of each individual file as it is copied, in the following form:

```
E:FILENAM1.EXT to user 9
E:FILENAM2.EXT to user 9
E:FILENAM3.EXT to user 9
E:FILENAM4.EXT to user 9
          .
          .
          .
E:FILENAMn.EXT to user 9
```

However, during an extensive RESTORE operation, you might be prompted to insert other specifically named disks.

If the BACKUP operation that created a backup disk set required more than one backup disk to accommodate all of the individual files, then a subsequent RESTORE operation will probably require that more than one disk be inserted to allow dispersal of the individual files.

The BACKUP utility kept track of the number and sequence of the disks used to accommodate the files and designated a “Volume” number for each disk that it prompted you to insert when creating the backup disk set.

Thus the subsequent RESTORE operation must echo the prompts displayed by BACKUP to instruct you to reinsert the same disks in the proper sequence.

The RESTORE prompts that request reinsertion of the backup disk set volumes appear in the following form:

```
Insert backup volume {nnn}, {sorcfile}.{nnn-1}, in drive {x},
and hit RETURN when ready.
```

Where {nnn} is the number of the volume number of the disk RESTORE needs now in order to continue restoring individual files from the backup disk set;

where {sorcfile} is the primary file name that was applied to all of the backup files during the BACKUP operation;

where {nnn-1} is the file name extension of the particular disk that RESTORE needs now in order to continue restoring individual files from the backup disk set (this extension number is always one less than the corresponding volume number); and

where {x} identifies the source drive you specified in the RESTORE command line.

"Volume nnn" corresponds to the backup file with extension nnn−1. The volume number is always one more than the number of the volume's file name extension.

In response to a prompt in this form, you should insert the specified volume (disk) in the specified drive. Then you should close the drive latch and press RETURN.

If RESTORE finds the backup file volume that it needed on the disk you just inserted, it will continue to restore individual files in sequence, displaying the drive and file names as they are restored.

If RESTORE cannot find the master backup file on the disk you just inserted, it will display a message in the following form:

```
Can not open backup file, {sorcfile}.{nnn-1},
insert another disk in drive {x},
and hit RETURN when ready,
or hit any other key to abort.
```

Where {sorcfile} is the primary file name that was applied to all of the backup files during the BACKUP operation;

where {nnn-1} is the file name extension of the particular disk that RESTORE needs in order to continue restoring individual files from the backup disk set (this extension number is always one less than the corresponding volume number); and

where {x} identifies the source drive you specified in the RESTORE command line.

If you wish to complete the operation and can obtain the disk that contains the specified backup file, then insert this disk in the specified drive, close the drive latch, and press **RETURN**.

If you do not wish to continue the operation or cannot obtain the disk that contains the specified backup file, then press any key other than RETURN. The RESTORE operation will end and a prompt (utility or system prompt) will appear.

The BACKUP Master Volume

The first disk in the backup set is known as "volume 1," or the "master volume" of the backup file. This disk contains a file with the primary name that was specified in the command line as the destination, and with the extension "000."

Sometimes during the RESTORE operation, RESTORE will need to access the contents of the backup master volume. RESTORE displays a slightly different prompt when it needs to access this particular volume, as shown:

```
Insert backup master volume 001, {sorcfile}.000, in drive {x},
and hit RETURN when ready.
```

Where the word "master" is added;

where the volume number is always "001" for a master volume, and

where the file name extension is always "000" for a master volume.

If RESTORE is prompting the master volume in particular, and you insert a disk that does not contain the master volume, then the error message displayed is also slightly different from that displayed when other volumes are needed, as shown:

```
Can not open master backup file, {sorcfile}.000,
insert another disk in drive {x},
and hit RETURN when ready,
or hit any other key to abort.
```

Where the word "master" is added; and

where the file name extension is always "000" for a master volume.

Preparing RESTORE Routines

RESTORE can be very useful if you restore backups on a regular basis. RESTORE was designed so that you can store the RESTORE command lines that you enter regularly, so that they can be entered automatically, with far less typing.

To store RESTORE commands for automatic execution, you will need a text editor or word processor and the SUBMIT utility.

The text editor or word processor will enable you to prefabricate and store commonly entered RESTORE command lines in a disk file. The SUBMIT utility will enable you to type a short, simple command line that causes automatic execution of all the stored RESTORE commands.

This text explains a few RESTORE routines geared toward users of some popular application software products available for your computer system. The text for each routine shows you how to prepare the routine by explaining the following essential facts:

- The type of user who would probably benefit from using the routine
- The names of the drives in which you should store particular files during the routine
- The form of the file you should create (with your text editor or word processor) to store commonly entered RESTORE command lines
- The form of the SUBMIT command you should type each time you wish to use the routine.

NOTE: To successfully prepare a RESTORE routine, you should understand the operation of the SUBMIT utility. For information on SUBMIT, refer to the "SUBMIT" text in the "User's Guide."

General Purpose RESTORE Routine

This routine can be helpful for anyone who wants to restore an entire backup disk set regularly. Steps 1 through 3 can be considered preparation steps that you need to perform only once. Steps 4 through 6 should be performed every time you conduct the routine.

1. Using a text editor or word processor, open a text file under the name **GNRESTOR.SUB**.

2. Into this file, enter the following command line:

   ```
   RESTORE $1:*.*=$2:GENBACK [V] RETURN
   ```

3. Close the text file GNRESTOR.SUB.

4. Use the PIP utility to copy the text file GNRESTOR.SUB and the utility file SUBMIT.COM to drive A, if they are not there already.

5. When you are ready to perform this RESTORE routine, type `A:`*RETURN* to make drive A the default drive.

6. Type a command in the following form:

   ```
   A>SUBMIT GNRESTOR {partition} {floppy} RETURN
   ```

   Where {partition} is the drive letter of the Winchester disk partition to which you wish to restore your files; and where {floppy} is the drive letter of the floppy disk drive from which you wish to restore your files;

For example, if you wish to restore files from the backup with the primary name "GENBACK" in the drive A floppy disk to the drive C Winchester partition, then you should enter the following command:

```
A>SUBMIT GNRESTOR C A RETURN
```

RESTORE will restore all files from the backup file with the primary name "GENBACK" back to the partition under their individual file names.

SuperCalc RESTORE Routine

This routine can be helpful for anyone who wants to restore SuperCalc data files from an entire backup disk set regularly. Steps 1 through 3 can be considered preparation steps that you need to perform only once. Steps 4 through 6 should be performed every time you conduct the routine.

1. Using a text editor or word processor, open a text file under the name **SCRESTOR.SUB**.

2. Into this file, enter the following command line:

   ```
   RESTORE $1:*.*=$2:SCBACK [V] RETURN
   ```

3. Close the text file SCRESTOR.SUB.

4. Use the PIP utility to copy the text file SCRESTOR.SUB and the utility file SUBMIT.COM to drive A, if they are not there already.

5. When you are ready to perform this RESTORE routine, type `A:`*RETURN* to make drive A the default drive.

6. Type a command in the following form:

   ```
   A>SUBMIT SCRESTOR {partition} {floppy} RETURN
   ```

   Where {partition} is the drive letter of the Winchester disk partition to which you wish to restore your files; and

   where {floppy} is the drive letter of the floppy disk drive from which you wish to restore your files.

For example, to restore SuperCalc data files from the backup file "SCBACK" on the drive A floppy disk to the drive C Winchester partition, enter the following command:

```
A>SUBMIT SCRESTOR C A RETURN
```

RESTORE will restore all files from the backup file with the primary name "SCBACK" back to the partition under their individual file names.

WordStar RESTORE Routine

This routine can be helpful to users of the WordStar word processor who wish to regularly restore all WordStar text files that have the same file name extension. Steps 1 through 3 can be considered preparation steps that you need to perform only once. Steps 4 through 6 should be performed every time you conduct the routine.

1. Using WordStar's non-document editing mode, open a text file under the name **WSRESTOR.SUB**.

2. Into this file, enter the following command line:

```
RESTORE $1:*.$2=$3:WSBACK[V] RETURN
```

3. Close the text file WSRESTOR.SUB.

4. Use the PIP utility to copy the text file WSRESTOR.SUB and the utility file SUBMIT.COM to drive A, if they are not there already.

5. When you are ready to perform this RESTORE routine, type `A:`*RETURN* to make drive A the default drive.

6. Type a command in the following form:

```
A>SUBMIT WSRESTOR {partition} {ext} {floppy} RETURN
```

Where `{partition}` is the drive letter of the Winchester disk partition to which you wish to restore your files;

where `{ext}` is the one- to three-letter file name extension of the text files you wish to back up; and where {floppy} is the drive letter of the floppy disk drive from which you wish to restore your files;

For example, if you wish to restore WordStar text files that have the file name extension "DOC," and if these files are now within the backup file with the primary name "WSBACK" in the drive A floppy disk, and if you wish to restore to the drive C Winchester partition, then you should type the following command:

```
A>SUBMIT WSRESTORE C DOC A RETURN
```

RESTORE will restore all files from the backup file with the primary name "WSBACK" back to the partition under their individual file names.

Structure of BACKUP Files

The BACKUP utility backs up several primary files into a single backup file. This backup file contains all of the files that were copied, in the order they were specified. At the beginning of the backup file is a directory of the copied files. The BACKUP utility maintains this directory to show the names of the files within the backup file, so that the RESTORE utility, when used, will be able to copy the files to other media.

At the beginning of the backup file directory is an entry for the backup file itself. This backup file directory entry is a sequence of bytes that are stored in the following form:

Field Function	Field Size (bytes)
ID	1
File Control Block (FCB)	32
R	4
quantity of copied files	2
date of backup	2
R	7
release number	1
version number	1
R	14
	64 bytes

Where *ID* is a byte that is set to one to distinguish this master backup directory entry from normal file directory entries;

where *File Control Block (FCB)* is a set of bytes that describe the drive name, primary file name, file name extension, and other characteristics used by CP/M to store files;

where *R* stands for reserved bytes that are not used for any particular purpose in this version of the software;

where *quantity of copied files* is the number of individual files stored within the backup file;

where *date of backup* is the date that the user entered in response to a prompt during the BACKUP operation; and

where *release number* and *version number* are included to ensure that this software will be used only with compatible releases and versions of BACKUP and RESTORE.

NOTE: The file name extension in the "File Control Block (FCB)" of the backup file directory entry is the file name extension of the last volume of the backup disk set—and not necessarily the extension of the volume that contains the directory. Thus you can determine how many volumes were used to accommodate all of the backed up files by looking at the directory in the first volume. (The first volume is the volume given the extension "000" during the BACKUP operation.)

Immediately following the backup file entry in the backup file directory are several sequences of bytes, each describing attributes of one of the individual files that were copied into the backup file during the BACKUP operation. The bytes of the individual file directory entry are stored in the following form:

Field Function	Field Size (bytes)
ID	1
File Control block (FCB)	32
R	4
start volume	1
end volume	1
start position	2
end position	2
file length	2
U	1
R	18
	64 bytes

Where *ID* indicates the location of this individual file entry among the other individual file entries in the directory. If the value of this byte is 2, then the entry describes an individual file within the backup file. If the value of this byte is FF, then the entry is a "dummy" entry that merely signals the end of the backup file, and does not describe an individual file;

where *File Control Block (FCB)* is a set of bytes that describe the drive name, primary file name, file name extension, and other characteristics;

where *R* stands for reserved bytes that are not used for any particular purpose in this version of the software;

where *start volume* indicates the number of the first backup file volume used to accommodate this individual file;

where *end volume* indicates the number of the last backup file volume used to accommodate this individual file;

where *start position* is the number of 128-byte records between the beginning of the backup file volume and the beginning of this individual file;

where *end position* is the number of 128-byte records between the beginning of the backup file volume and the record following the end of this individual file;

where *file length* is the length of this individual file, as measured in 128-byte records; and

where *U* is the number of the user area from which the individual file came.

The number of individual source files that can be backed up in a single BACKUP operation is limited by the amount of space on the first disk/partition used to receive the backup volumes. This disk/partition must have enough free space to accommodate the entire backup file directory.

NOTE: If any of the source files you specify are random files (as opposed to sequential files), these files might become larger when restored with the RESTORE utility.

RESTORE Error Messages

```
Verify error, try RESTORE again (Y/N)?
```

EXPLANATION: RESTORE has detected that the copy of a backed up file on destination media is different from the original copy of the file on the source media.

Press **Y** if you wish to have RESTORE try to recopy and verify the file that could not be verified. RESTORE will try to overwrite the copy of the file that could not be verified. Then RESTORE will resume displaying the names of the individual files as they are copied and verified, beginning with the file in which the verification error occurred.

Press **N** if you wish to skip the file that could not be verified. Then RESTORE will try to copy and verify the next individual file that is listed in the backup file.

```
Backup file name can not be ambiguous.
```

EXPLANATION: You specified * or ? wild card characters in the source file name for the RESTORE operation. Repeat the command specifying the source with an explicit file name.

```
Extension on backup file specified.
Extension 000 will be assumed.
```

EXPLANATION: This occurs whenever you try to assign a backup file an extension by specifying a full file name for a destination file. If this occurs, RESTORE ignores the extension you specified and uses its standard, sequentially numbered extensions. Program operation will not be disturbed.

```
Can not find master backup file {file name}.000.
```

EXPLANATION: This message occurs when the /L option is requested for a file from a disk where the master backup {file name}.000, is not present.

```
File {file name}.{ext} is not found.
```

EXPLANATION: This message occurs whenever a file is specified for BACKUP or RESTORE and that file is not on the disk.

```
Invalid backup file.
```

EXPLANATION: This message occurs if the backup file specified in a command does not contain valid information. This may occur if the file specified was not a backup file, but had a “000” extension, or if the data in a backup file had degraded (possibly due to a bad sector or inadvertent exposure of the medium to an electromagnetic field).

```
Invalid drive designation.
```

EXPLANATION: This message occurs when a drive name is used that is not in the range of supported drive names (A through F).

```
Invalid exception file specifications.
```

EXPLANATION: This message will occur if the exception file specified has a syntax error in the specification.

```
Invalid filename.
```

EXPLANATION: This message appears when a file name is specified that does not conform to CP/M file naming conventions.

```
Invalid selection file specifications.
```

EXPLANATION: This message is generally caused by a typographical error in the command line. The message results when parameters in the command line appear garbled or incorrectly punctuated.

```
Invalid option [x] specified.
```

EXPLANATION: This message occurs if RESTORE is unable to recognize the option that was specified in the command.

```
Not enough parameters specified.
```

EXPLANATION: This message results when the command to RESTORE is not complete enough for RESTORE to carry out the intended operation. Enter the command again specifying additional options or file, specifications.

```
Can not open master backup file {file name}.000,
insert another disk in drive x
hit RETURN when ready,
or hit any other key to abort.
```

EXPLANATION: This message occurs if the disk that has been inserted is not volume 1. Insert the correct disk.

```
Can not open backup file {file name}.{nnn},
insert another disk in drive x
hit RETURN when ready,
or hit any other key to abort.
```

EXPLANATION: This message occurs when you are asked to insert volume nnn+1 (which would contain `{filename}.{nnn}`) and the wrong disk is inserted. Insert the correct disk.

```
Invalid user option.
```

EXPLANATION: This message occurs if the user number specified with the [U] option is not in the range 0-15 or if a user number was specified with improper syntax.

```
Invalid version of RESTORE for {file name.000}.
```

EXPLANATION: This message occurs if you have used different versions of RESTORE and BACKUP. Always use a copy of the RESTORE utility that bears the same version number as the BACKUP utility that you used to create backup file {file name.000}.

```
No files selected.
```

EXPLANATION: This message occurs if none of the source files you specified in the RESTORE COMMAND exist on the default or specified source media.

```
Out of disk space on restoration of {file name.ext}
insert another disk and hit RETURN
or hit any other key to abort
```

EXPLANATION: The disk to which you are trying to restore files does not contain enough blank space to accommodate an individual file. You should insert a disk that has enough space into the destination drive and press the **RETURN** key, or press a key other than the RETURN key and enter a RESTORE command with different destination file specifications.

APPENDIX
Introduction

The following appendices provide information about four utilities you must use if you plan to use CP/M-86 with a Winchester hard disk system. The utilities themselves — PART, PREP, SHIP, and VERIFY — are not included on the CP/M-86 distribution disks. Rather, these utilities are recorded on a disk that accompanies the Winchester hard disk hardware, and more information about them can be found in the Winchester Supplement of the Z-100 User's Manual.

The PART, PREP, SHIP, and VERIFY utilities perform important operations that prepare your Winchester hard disk to receive CP/M-86. The following list describes the operation performed by each utility. More detailed information can be found in the appendix devoted to each utility.

PART	The utility that rearranges Winchester Disk partitions and/or changes the partition used for default bootup.
PREP	The utility that initializes a Winchester Disk.
SHIP	The utility that moves read/write heads to a safe position.
VERIFY	The utility that isolates recent bad sectors on your Winchester Disk.

APPENDIX A
PART

The Utility that Rearranges Winchester Disk Partitions and/or Changes the Partition Used for Default Bootup

The PART utility enables you to change the quantity, size, and names of Winchester disk partitions. It also enables you to specify which partition should be accessed when you boot up. You do not need to run PART in order to use your Winchester disk, because a CP/M partition was prepared on the disk before it was shipped.

The PART utility is recorded on the Winchester Utility Disk, which is shipped with your Winchester disk hardware documentation. The PREP utility is also recorded on this disk. Although this disk runs under the Z-DOS Operating System, you can boot up with it — just as you would a CP/M disk.

CAUTION: Any changes you make to the quantity or size of partitions through PART can destroy all existing data on your Winchester disk. Therefore, you should back up all necessary data from all partitions before you use PART.

NOTE: After using the PART utility, you must enter **CTRL-RESET** and reboot the system with a floppy disk containing the CP/M-86 Operating System.

Partition Features

Winchester disks distributed by Zenith Data Systems or Heath have large storage capacities. To make practical use of all this storage space, it is divided into partitions. You can establish up to 16 CP/M partitions on your Winchester disk. You can also have partitions with other operating systems on the disk.

APPENDIX A

PART

A partition behaves like a floppy disk in most operations, because you can access a partition's data and/or software by entering commands that refer to the drive name that has been designated for that particular partition. However, you can only access a CP/M partition through a drive name either by booting up to that particular partition, or by first running the ASSIGN utility.

NOTE: The exact capacity of your Winchester disk drive is determined by the drive's manufacturer and by the amount of usable disk space remaining after unusable space has been made inaccessible by software such as the PREP utility. Examples in this text show disk space totals for a Winchester disk that accommodates 10,000 kilobytes. However it is possible that your disk will accommodate a different amount.

When your Winchester disk is shipped from the factory, it has already been prepared with two partitions. Each of these partitions occupies approximately one half of your total Winchester disk space. Each is also given a distinct name. One of these partitions is intended for used with the CP/M-86 Operating System and software that runs under CP/M-86. The other is intended for use of with the Z-DOS Operating System (sold separately) and software that runs under Z-DOS.

The PART utility enables you to view and change the status of the following partition features:

- Name of each partition
- Name of the operating system to be placed on each partition
- Approximate percentage of disk space allocated to each partition
- Precise capacity of each partition in kilobytes (1024-byte units)
- Total (approximate) percentage of Winchester disk space that is and is not allocated to partitions
- Total (precise) number of kilobytes of Winchester disk space that are and are not allocated to partitions
- Name of the one partition that is accessed when you boot up without specifying a partition

APPENDIX A

PART Operation

The PART utility enables you to change the status of partition features by typing different kinds of entries in response to prompts. During different phases of PART operation, features of the screen display will change and the cursor will move to the appropriate screen location after your entries. PART operation usually prompts you to make the following kinds of entries in sequence:

1) PART Invocation
2) PART Ratification
3) Choice of Operation
4) Partition Names
5) System Names
6) Allocation Percentages
7) Default Boot Partition Number
8) Choice of Operation
9) Choice of Exit Method

NOTE: If you wish to rearrange partitions, then you will probably repeat steps 4), 5), and 6) several times (once for each partition) before you perform step 7). If you merely wish to set a new default boot partition, then you will skip steps 4), 5), and 6).

PART Invocation

PART is recorded on the Winchester Utility Disk (supplied with your Winchester disk hardware and documentation) as an executable .COM file.

Entry Rules

To invoke the PART utility, enter the following command at the system prompt:

```
    A:PART RETURN
```

PART

NOTE: Enter the PART command at the Z-DOS system prompt just as you would at a CP/M system prompt.

Resulting Display

After it is invoked, the PART utility displays identification messages, a caution, and a prompt, as shown:

```
                 PART version x.xx
Copyright (C) 1982, Zenith Data Systems Corporation


The PART utility helps you to:

* change the arrangement of your Winchester disk partitions and/or
* select a partition (default boot partition) to which you can
  boot up without specifying the partition's name

PART displays a table showing the names of each partition (a
partition name and a system name) and the amount of disk space
allocated to each partition (in percentages and in kilobytes). PART
also dynamically calculates and totals the kilobyte size of all
partitions as you specify each partition's allocation percentage.

CAUTION: Using PART can destroy all files on your Winchester disk.
Do  not use PART until you  have transferred backup copies of your
Winchester disk files to floppy disks.

Proceed (Y/N)?
```

NOTE: The version number of the PART utility (shown in this example as “x.xx”) might vary.

Error Conditions

If the PART.COM file is not on the disk you are using to run PART, you will receive the following “Bad command or file name” error message. Try the command again after inserting a disk containing the PART.COM file into the drive. Use the **DIR RETURN** command at the “A:” system prompt to determine whether a disk has this file.

If the disk has been removed from drive A since bootup, you will receive a "Seek error reading drive A Abort, Retry, Ignore" error message. Insert a disk containing PART.COM in drive A, close the drive latch, and press **R**.

If you try to run PART after booting up with a disk containing the CP/M-86 Operating System, then you will receive an error message in the form: "Bdos Err On X: Select". Reset the computer, boot up to the disk containing the Z-DOS Operating System and the PART.COM file. (We recommend that you use the Winchester Utility Backup Disk.) Then repeat the PART command at the A: system prompt.

PART Ratification

At the initial PART display, you have the choice of ratifying your intentions to use the PART utility as it is described in the displayed messages (and subject to the displayed caution), or exiting from the PART utility.

Continuing to use PART

An affirmative answer to this prompt verifies your intentions to continue using PART.

Entry Rules

Type **Y** at this prompt if you fully understand the consequences of using PART and still wish to continue.

CAUTION: Rearrangement of Winchester disk partitions can destroy any software or data now stored on the disk. Therefore we recommend that you make backup copies of any valuable software and/or data now stored on the Winchester disk before verifying your intentions to use PART.

NOTE: You can use PART to select a different "default boot partition" without destroying software or data, as long as you make no entries to change the arrangement of partitions.

Resulting Display

When you have ratified your intentions to use PART, the current status of several partition features is visible on a screen display of the following form:

```
        Partition Name    Operating System Name    Percentage    Kilobytes
        --------------    ---------------------    ----------    ---------
     1. Z-DOS              Z-DOS                   50%           5000
     2. CPM                CPM                     50%           5000
     3.
     4.
     5.
     6.
     7.
     8.
     9.
    10.
    11.
    12.
    13.
    14.
    15.
    16.
    Total Utilization (Allocated/unallocated)      100/0         10000/0
    Default boot partition number: 1   <Z-DOS;Z-DOS                 >

         B - Modify default boot partition
         P - Partition maintenance
         E - Exit
    Choose desired option. <B, P or E> _
```

The PART display shown above is the display that will appear the first time you run PART after shipment of a Zenith Data Systems (or Heath) Winchester disk, or after you have run the PREP utility. If you have already used PART since shipment or since running PREP, then some features of this display might appear differently.

Declining to Use PART

Type any character other than Y or y if you do not wish to use PART at this time. The A: system prompt will be displayed.

Choice of Operation

After ratifying your intentions to use the PART utility, you have the choice of selecting a different default boot partition, changing the arrangement of your Winchester disk partitions, or exiting from PART.

A three-line menu enables you to select a PART operation by typing B, P, or E as the cursor flashes at the end of the prompt on the bottom of the screen.

Choosing to Modify Default Boot Partition

The default boot partition is usually the partition that you intend to use most often for booting up. Any established bootable partition that is selected as the default boot partition is the partition that will be automatically accessed when you type a bootup command without specifying a partition name. (In order to boot up to any other partition, you must specify the partition name, and sometimes the system name, in your bootup command.)

Entry Rules

Type **B** at this prompt to select an default boot partition. This operation will not prompt you to change the arrangement of any partitions. If you choose this activity, skip the next three steps and refer to "Default Boot Partition Number" for further instructions.

Resulting Display

After typing the B entry to "Modify default boot partition", the activities menu and prompt will be replaced by the "Enter number of new default boot partition" prompt. Additionally, the number and names of the current default boot partition will be replaced by the cursor on the display line beneath the table, as shown:

```
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
15.
16.
Total Utilization (Allocated/unallocated)         100/0            10000/0
Default boot partition number: 1    <Z-DOS;Z-DOS                   >
```

Enter number of new default boot partition.

When a display in this form appears, proceed to "Default Boot Partition Number".

Choosing Partition Maintenance

The "Partition Maintenance" operation leads you to add partitions, remove partitions, change the names of partitions, and/or change the percentage of space allocated to partitions. Then this activity also prompts you to select a default boot partition.

Entry Rules

Type **P** at this prompt to begin changing the arrangement of partitions. (At the end of this operation, you will also be prompted to change the setting of the default boot partition.)

Resulting Display

When you choose the "Partition maintenance" operation, the cursor moves to the beginning of the first partition name in the table and the operations menu is replaced by a "Minimum allocation" message at the bottom of the display, as shown:

```
   PartitionName     OperatingSystemName    Percentage    Kilobytes
   --------------     --------------------    ----------    ---------
1. Z-DOS              Z-DOS                   50%           5000
2. CPM                CPM                     50%           5000
3.

/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\

15.
16.
Total Utilization (Allocated/unallocated)  100/0          10000/0
Default boot partition number: 1   <Z-DOS;Z-DOS              >

Minimumallocation = 2%
```

When a display in this form appears, proceed to "Partition Name."

Choosing to Exit

Type **E** at this prompt to exit from the PART utility completely. The A: system prompt will be displayed immediately.

Partition Name

After you have chosen the "Partition maintenance" operation, the cursor moves to the first character of a "Partition Name".

Changing or Adding a Partition Name

When the cursor moves to a position beneath the "Partition Name" column on the table, you can change the name of an existing partition or add a new partition to the table by entering the partition name as the first feature of this partition.

APPENDIX A

PART

Entry Rules

A partition name is any string of *1-16 ASCII characters*, with the exception of the semicolon (;) and non-printing characters such as a space or tab. After entering this string, press the **RETURN** key. Then the cursor will move to the operating system name position for this partition.

PART will always display partition name letters (if any) in upper case, although you can enter them in either lower or upper case.

When the cursor arrives at the first character of a partition name, you will immediately erase the old partition name by pressing any key other than **RETURN** or the space bar.

While entering a partition name, you can press the **BACK SPACE** key to erase one character to left. If, when changing an old name, you press BACK SPACE until you have erased your entire new name, the old name will be redisplayed.

Partitions that you intend to give the same operating system name should be given different partition names.

Examples

The following examples are valid partition names:

```
CP/M_ACCOUNTING     DATABASE     WORD/PROCESSING      George

Spread_sheet     Z-DOS_ACCOUNTING        Thelma        BASIC
```

Resulting Display

After you have entered 1-16 characters and pressed the RETURN key for a partition name, the new partition name will be displayed and the cursor moves to the operating system name position for this partition, as shown in the following partial display:

```
   Partition Name    System Name       Percentage   Kilobytes
   --------------    -----------       ----------   ---------
1. DATABASE          Z-DOS                50%         5000
2. CPM               CPM                  50%         5000
3.
4.
```

/\

Error Conditions

If you press the semicolon key or the TAB key and then press the RETURN key, the partition name will appear to vanish while the other features for the partition remain. However when you have finished making an entry for this partition's allocation percentage, the old partition name for this partition will reappear.

You cannot finish PART operation after giving two partitions the same partition names unless they are given different system names. However, it will wait until you make an entry for the default boot partition before displaying the "duplicate name error" message in reverse video. If this message appears, you will have to correct the duplicate names before exiting from PART.

Bypassing a Partition Name

When the cursor moves to a position beneath the "Partition Name" column on the table, you can retain an old partition name and advance to the system name or stop rearranging the partitions completely.

APPENDIX A

PART

Entry Rules

To refrain from changing a partition name, press only the RETURN key.

Resulting Displays

If the cursor was at a partition where a partition name already existed, then the RETURN entry causes the cursor to move ahead to the system name of the same partition, as shown in the following partial display:

```
   PartitionName      SystemName             Percentage    Kilobytes
   --------------      -----------             ----------    ---------
1. Z-DOS              Z-DOS                       50%          5000
2. CPM                CPM                         50%          5000
3.
4.

\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
```

If the cursor was at a position beneath the "Partition Name" category where no partition name has yet been established, a RETURN entry will move the cursor ahead to the "Default boot partition number" feature, as shown in the following partial display:

```
   PartitionName      Operating System Name    Percentage    Kilobytes
   --------------      ---------------------     ----------    ---------
1. Z-DOS              Z-DOS                    50%           5000
2. CPM                CPM                      50%           5000
3.
4.

\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
/

15.
16.
Total Utilization (Allocated/unallocated)      100/0         10000/0
Default boot partition number: _
```

Enter the number of the default boot partition.

Removing a Partition

When the cursor moves to a position beneath the "Partition Name" column on the table, you can remove all features of this partition from the table.

Entry Rules

To remove a partition from the table without inserting a new partition in its place, simply press the **space bar**.

Resulting Display

The partition name, system name, allocation percentage, and allocation kilobytes for this partition will disappear. Then the features for all of the partitions below will move up one number. The cursor remains in the same position, although it now blinks at the partition name of a different partition.

If the partition you have removed was the default boot partition, then the "is undefined" message will be displayed.

Additionally, the "Total Utilization" (in both percentage and kilobytes) will show a different amount of "unallocated" space in reverse video (unless the removed partition was allocated zero percent of the disk).

The following partial display shows how the screen might appear after you remove a partition:

```
   Partition Name     Operating System Name    Percentage    Kilobytes
   --------------     ---------------------    ----------    ---------
1. CPM                CPM                      50%           5000
2.
3.
4.
\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
15.
16.
Total Utilization (Allocated/unallocated)      50/50         5000/5000
Default boot partition number: is undefined


  Minimum allocation = 2%
```

Error Condition

If you remove a partition that was the default boot partition, then the "is undefined" message will be displayed until you make an entry for the default boot partition. If you then press RETURN to bypass setting the default boot partition feature, PART will display the "boot partition error" message in reverse video and prevent you from exiting from PART.

Operating System Name

After you have pressed the RETURN key during a partition name entry, the cursor moves to the first character of an "Operating System Name".

System names are not mandatory for partitions, although they must be used when the same partition names are used for different partitions.

Changing a System Name

When the cursor moves beneath the "Operating System Name" column on the table, you can change the system name of a partition.

Entry Rules

A system name is a string of *1-10 ASCII characters*, with the exception of the semicolon (;) and non-printing characters such as a space or tab. After entering this string, press the **RETURN** key.

PART will display system name letters in upper case, although you can enter them in either lower or upper case.

When the cursor arrives at the first character of a system name, you can immediately erase the old system name by pressing any key other than RETURN.

While entering a system name, you can press the **BACK SPACE** key to erase one character to left. If, when changing an old name, you press BACK SPACE until you have erased your entire new name, the old name will be redisplayed.

The same system name can be used for several partitions, as long as the partition names are different.

Example Entries

The following examples are valid operating system names:

CPM   Z-DOS   UNDER-DOS   Acronym-DOS

NOTE: The Z-DOS operating system requires that you use the operating system name "Z-DOS" (as it is spelled here) for the partition(s) you plan to use for Z-DOS programs. We recommend that you use the operating system name "CPM" for the partitions you plan to use for CP/M programs, although CP/M does not require any particular system name.

Resulting Display

After you have entered 1-10 characters and pressed the RETURN key for a system name, the new system name will be displayed and the cursor moves to the allocation percentage position for this partition, as shown in the following partial display:

```
   Partition Name    System Name          Percentage    Kilobytes
   --------------    -----------           ----------    ---------
1. DATABASE          CPM                   50%           5000
2. CPM               CPM                   50%           5000
3.
4.
\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
```

Error Conditions

If you try to enter the same system name for two partitions that also have the same partition name, PART will wait until you make an entry for the default boot partition display the "duplicate name error" message in reverse video. If this message appears, you will have to correct the duplicate names before exiting from PART.

If the names you have entered duplicate the names of a partition that has a greater number on the left side of the table, and if this partition with the greater number is the default boot partition, then the number of the default boot partition will automatically change to the smallest number of the two partitions that have the same names. This change in the number of the default boot partition occurs after you have made an entry for the allocation percentage of the duplicate partition with the smallest number.

If you press the semicolon key or the TAB key or the space bar and then press the RETURN key, the operating system name will appear to vanish while the other features for the partition remain. However when you have finished making an entry for the default boot partition, the old system name for this partition will reappear.

Bypassing the System Name

When the cursor moves beneath the “Operating System Name” column on the table, you can retain the old system name (or leave this feature blank) and skip ahead to the same partition’s allocation percentage.

Entry Rules

To refrain from changing a system name, press only the RETURN key.

Resulting Display

The RETURN only entry causes the cursor to move ahead to the percentage feature of the same partition, without making any change to the system name, as shown in the following partial display:

```
    Partition Name     System Name          Percentage   Kilobytes
    --------------     -----------          ----------   ---------
 1. DATABASE           Z-DOS                   50%          5000
 2. CPM                CPM                     50%          5000
 3.
 4.

/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
```

Allocation Percentage

After you have made an entry at an “Operating System Name” position, the cursor moves to the “Percentage” position for the same partition.

Setting a Percentage

When the cursor moves to the first digit of a “Percentage” (of Winchester disk space allocation), you can set an allocation percentage.

Entry Rules

You must enter percentages by typing a whole **number in the range n-100**, where **n** is the number displayed at the bottom of the screen on the right side of the "Minimum allocation" message.

The percentage you enter for a CP/M partition must be low enough so that this partition will not be larger than 8 megabytes (8192 kilobytes). (If you allocate more than 8 megabytes to a partition, CP/M-86 will not be able to access part of this partition.)

NOTE: Different brands and models of Winchester disk require different minimum allocations.

Resulting Displays

For every allocation percentage you enter, PART automatically calculates the exact number of kilobytes that should be allocated to a partition according to the percentage you entered. PART also automatically calculates the total quantity of percentage points and kilobytes that are allocated and unallocated.

If you set an allocation percentage that brings the total allocation of disk space to 100 percent, and if you are entering it for one of the first 15 partitions on the table, then the cursor will move down to the partition name position of the next partition (whether this partition has been established yet or not) and the screen will appear in the form of the following partial display:

```
     Partition Name     Operating System Name    Percentage    Kilobytes
     --------------     ---------------------    ----------    ---------
  1. DATABASE           CPM                      30%           3000
  2. CPM                CPM                      70%           7000
  3. _
  4.
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
 15.
 16.
 Total Utilization (Allocated/unallocated)       100/0         10000/0
 Default boot partition number: is undefined

    Minimum allocation = 4%.
```

If you set an allocation percentage that brings the total allocation of disk space to 100 percent, and if you are entering it for the 16th partition on the table, then the cursor will move down to the default boot partition position and the screen will appear in the form of the following partial display:

```
    Partition Name     Operating System Name    Percentage    Kilobytes
    --------------     ---------------------    ----------    ---------
 1. DATABASE           CPM                      6%            600
 2. SPREAD-SHEET       CPM                      6%            600

/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\

15. COBOL              UNDER-DOS                6%             600
16. FUN&GAMES          ACRONYM-DOS              10%           1000
Total Utilization (Allocated/unallocated)      100/0         10000/0
Default boot partition number: _
```

Enter number for the default boot partition.

NOTE: If you establish 16 partitions on your disk, and have not allocated all of the disk space with your percentage entry for the 16th partition, PART will automatically allocate all of the remaining disk space to the 16th partition — no matter how much space you try to allocate to this partition.

If you set an allocation percentage that brings the total allocation of disk space to less than 100 percent, and if you are entering it for one of the first 15 partitions on the table, then the cursor will move down to the partition name position of the next partition (whether this partition has been established yet or not). Additionally, the "Total Allocation" figures will display the amount of "unallocated" space (in both percentages and kilobytes) in reverse video. The screen will appear in the form of the following partial display:

```
   Partition Name      Operating System Name   Percentage   Kilobytes
   --------------      ---------------------   ----------   ---------
1. DATABASE            CPM                     50%          5000
2. CPM                 CPM                     20%          2000
3. _
4.
\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
15.
16.
Total Utilization (Allocated/unallocated)      70/30    7000/3000
Default boot partition number: _


     Minimum allocation = 2%
```

NOTE: You can exit from PART with part of your disk unallocated.

Error Conditions

PART will monitor the percentages you enter for each partition, and dynamically lower any percentage you enter if it would have brought the total allocation to more than 100 percent of disk space. Thus PART will never allow you to allocate more than 100 percent of your Winchester disk space, and the displayed percentages will never total more than 100.

PART keeps the total allocation percentages at or below 100 percent by subtracting percentage points from partitions, starting with the partitions at the bottom of the table. In cases of extreme over allocation, PART might even reduce the allocation percentages of some partitions to zero percent.

The minimum possible allocation percentage, which differs depending on the kind of Winchester disk you have purchased, is displayed at the bottom of the screen whenever you are in position to enter an allocation percentage. If you enter an allocation percentage lower than the minimum limit for your Winchester disk, PART will automatically convert this percentage to zero percent.

Whenever the percentage of a partition is zero percent, PART will prevent you from exiting until you have changed this percentage to a value at or above the minimum percentage allowed for your own particular Winchester disk.

If you try to enter a non-numeric character, a fractional or decimal point number, or a number greater than 100, then the terminal will beep and the cursor will remain at the percentage position. Then you can enter a valid number.

You can exit from PART after allocating less than 100 percent of the disk space to partitions. The unallocated space on your Winchester disk will be inaccessible until you use either the PART utility or the PREP utility.

Bypassing the Percentage

When the cursor moves to the first digit of an existing “Percentage” (of Winchester disk space allocation), you can skip ahead to the partition name of the next partition, or skip ahead to the default boot partition number.

Entry Rules

To retain the percentage that is currently displayed for a partition and skip ahead to another partition feature, press the RETURN key only.

Resulting Displays

The displays that result after you have bypassed a percentage entry are similar in form to those that appear when you enter a number, although the percentage of the partition for which you just made the entry will not change.

The cursor will move to the partition name position for the next partition, unless you have just bypassed the percentage for the 16th partition in the table, in which case the cursor will move down to the default boot partition position.

Error Conditions

If you bypass the percentage feature for a newly established partition (by specifying no number and pressing the RETURN key), then PART will display "0%" for this partition. Additionally, the "allocation error" message will appear in reverse video when the operations menu reappears. Furthermore, you will be prevented from exiting from PART until you correct the partition table so that no partitions are allocated "0%" of the disk space.

Default Boot Partition Number

After you have either chosen the "Modify default boot partition" operation or finished a "Partition maintenance" operation, the cursor will move to the default boot partition position. The current default boot partition number and names will vanish from the display.

Selecting a Default Boot Partition

When the cursor moves to the default boot partition position, you can select a different partition to be accessed during bootup.

The default boot partition position is at the right side of the "Default boot partition number:" message. When the cursor moves to this position, the number and names of the current default boot partition vanish from the display, and the "Enter the number of the new default boot partition." message appears at the bottom of the screen.

Entry Rules

To establish a default boot partition, you must enter the *number of an established partition* and press RETURN.

This number must be in the range 1-16. It must be listed in the table to the left of an established partition name. The number, partition name, and system name of the partition you specified by number will be displayed between angle brackets.

Resulting Display

If you enter **3** as your default boot partition number, and if line 3 on your partition table looks like the following:

```
3. WORD/PROCESSING    CPM                    15%          1500
```

then part of the display showing your new default boot partition number will appear as follows:

```
   Partition Name     Operating System Name  Percentage   Kilobytes
   --------------     ---------------------  ----------   ---------
1. DATABASE           CPM                    30%          3000
2. SPREAD-SHEET       CPM                    30%          3000
3. WORD/PROCESSING    CPM                    15%          1500
4. BASIC              Z-DOS                  25%          2500
5.
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
15.
16.
Total Utilization (Allocated/unallocated)    100/0        10000/0
Default boot partition number: 3 <WORD/PROCESSING;CPM          >
     B - Modify default boot partition
     P - Partition maintenance
     R - Restore to original partitions
     E - Exit
Choose desired option. <B, P, R or E>
```

NOTE: The operations menu is displayed beneath the default boot partition number as soon as you make an entry for this feature. Refer to "Entering Choice of Operation" for instructions on using this menu at this stage of the program.

Error Conditions

If you try to enter a number (in the 1-16 range) for which there is no established partition on the table, the invalid number will vanish and the cursor will remain at the number position until you enter a valid number and press RETURN.

If you try to enter a number that is out of the 1-16 range, the computer beeps and the cursor remains at the number position until you enter a valid number and press RETURN.

Bypassing the Default Boot Partition

When the cursor moves to the default boot partition number, you can retain the current default boot partition, and advance to the PART operations menu.

Entry Rules

To bypass changing the default boot partition feature, press only the **RETURN** key at this position.

Resulting Display

The number and names of the current default boot partition will be redisplayed, and the cursor will move to the "Choose desired option" prompt beneath the operations menu, as shown:

```
\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
15.
16.
Total Utilization (Allocated/unallocated)       100/0        10000/0
Default boot partition number: 1 <DATABASE;CPM                 >
     B - Modify default boot partition
     P - Partition maintenance
     R - Restore to original partitions
     E - Exit
Choose desired option. <B, P, R or E> _
```

Refer to "Entering Choice of Operation" for instructions on using the operations menu at this stage of the PART utility.

Error Condition

If you have caused the default boot partition feature to become undefined by removing a partition from the table (as explained in "Removing a Partition"), and then bypass the default boot partition, you will still encounter the "boot partition error" message, as shown in the following partial display:

```
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
15.
16.
Total Utilization (Allocated/unallocated)      100/0        10000/0
Default boot partition number: is undefined
boot partition error
     B - Modify default boot partition
     P - Partition maintenance
     R - Restore to original partitions
Choose desired option. <B, P or R>_
```

Furthermore, you will not be able to exit from PART until you either enter the number of an existing partition or remove the default boot partition.

Removing the Default Boot Partition

When the cursor moves to the default boot partition number, you can remove this feature altogether and advance to a PART operations menu.

Entry Rules

To remove the default boot partition, press the **space bar** only.

Resulting Display

The "is undefined" message will be displayed at the default boot partition position. However, you will not encounter the "boot partition error" message. Additionally, the operations menu and prompt appear, as shown in the following partial display:

```
\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
15.
16.
Total Utilization (Allocated/unallocated)      100/0        10000/0
Default boot partition number: is undefined
     B - Modify default boot partition
     P - Partition maintenance
     R - Restore to original partitions
     E - Exit
Choose desired option. <B, P, R or E> _
```

Refer to "Entering Choice of Operation" for instructions on using the operations menu at this stage of the PART utility.

NOTE: If you remove the default boot partition, you will have to enter bootup commands that specify partition names (and possibly also operating system names) in order to boot up with any partition.

Entering Choice of Operation

After you completed a PART operation (such as "Modify default boot partition" or "Partition maintenance"), the operations menu will appear.

This menu is the same as the menu explained in "Choice of Operation", with the addition of the "R – Restore to original partitions" operation.

NOTE: If you have encountered an error condition, the "E – Exit" operation might not be displayed. Therefore you must choose one of the operations that enables you to change partition features, and correct the error condition before PART will allow the "E – Exit" operation.

Choosing to Modify Default Boot Partition

The partition that is selected as the default boot partition is the partition that will be automatically accessed when you type a bootup command without specifying a partition name.

Entry Rules

Type **B** at this prompt to select a default boot partition.

Resulting Display

After typing the B entry to "Modify default boot partition", the activities menu and prompt will be replaced by the "Enter number of new default boot partition" prompt. Additionally, the number and names of the current default boot partition will be replaced by the cursor on the display line beneath the table.

When a display in this form appears, refer back to "Default Boot Partition Number" for further instructions.

Choosing Partition Maintenance

The "Partition Maintenance" operation leads you to add partitions, and/or change the percentage of space allocated to partitions. Then this activity also prompts you to select a default boot partition.

Entry Rules

Type **P** at this prompt to begin changing the arrangement of partitions.

Resulting Display

When you choose the "Partition maintenance" operation, the cursor moves to the beginning of the first partition name in the table and the operations menu is replaced by a "Minimum allocation" message at the bottom of the display.

When a display in this form appears, refer back to "Partition Name".

Restoring Partition Features

When the cursor moves to the "Choose desired option." prompt, you can end the PART session, rearrange your partitions starting either with your most recent entries or with the partition features set as they appeared when you invoked PART, or reselect a default boot partition starting either with your most recent entries or with the default boot partition that was set when you invoked PART.

Entry Rules

Type **R** if you want each partition feature to revert to its status at the time you invoked the PART utility.

Resulting Display

When you choose the "R – Restore to original partitions" operation, PART will redisplay the operations menu without the "R – Restore to original partitions" option. The partition table will show the names, allocations, and default boot partitions that were already established when you invoked PART, as shown in the following display:

```
   Partition Name     Operating System Name    Percentage    Kilobytes
   --------------     ---------------------    ----------    ---------
 1. Z-DOS              Z-DOS                     50%          5000
 2. CPM                CPM                       50%          5000
 3.
 4.
 5.
 6.
 7.
 8.
 9.
10.
11.
12.
13.
14.
15.
16.
Total Utilization (Allocated/unallocated)      100/0         10000/0
Default boot partition number: 1   <Z-DOS;Z-DOS                  >

     B - Modify default boot partition
     P - Partition maintenance
     E - Exit
Choose desired option. <B, P or E> _
```

NOTE: This display is identical to the display presented in "2.3 Choice of Operation".

Error Conditions

If you enter a character other than B, P, R, or E, PART will cause the computer to beep and the cursor to remain at the end of the prompt until you enter a valid letter.

Beginning the Exit Operation

When you exit from the PART operation after changing any partition features (through the "Modify default boot partition" or "Partition maintenance" operations), you will have to choose this operation and then make an entry at another menu.

Entry Rules

Type **E** if you wish to exit from the PART utility.

Resulting Display

When you choose the "E – Exit" operation, the following exit menu and prompt will be displayed at the bottom of the screen:

```
        M - Make changes and exit
        A - Abort, make no changes and exit

    Choose desired option. <M or A> _
```

Error Conditions

If you enter a character other than B, P, R, or E, PART will cause the computer to beep and the cursor to remain at the end of the prompt until you enter a valid letter.

Choice of Exit Methods

When you choose the "E – Exit" operation after changing partition features, the final exiting menu appears.

CAUTION: Before typing an entry at this menu, review the partition table carefully to be certain that your partitions are allocated as you want them. Remember that any rearrangement of partitions can destroy data on the Winchester disk, and that no changes are actually made to the Winchester disk until you enter the M option.

NOTE: Regardless of the exit method you choose, you will have to enter **CTRL-RESET** and reboot the system after exiting from PART.

Making Changes to the Winchester Disk

When the cursor moves to the prompt beneath the final exit menu, you can change the partition features according to your entries while exiting from PART.

Entry Rules

Type **M** if you want to exit from the PART utility and change the status of Winchester disk partition features to reflect the changes that you entered during this PART session.

CAUTION: This entry has the potential to destroy any data that might exist on the Winchester disk.

Resulting Display

After you choose and enter the "Make changes" option the Z-DOS system prompt will appear, as shown:

```
    A:
```

Error Conditions

If you enter a character other than M or A, PART will cause the computer to beep and the cursor to remain at the end of the prompt until you enter a valid letter.

Aborting Changes to the Winchester Disk

When the cursor moves to the prompt beneath the final exit menu, you can nullify all of the changes you have entered while exiting from PART.

Entry Rules

Type **A** if you want to end this PART session without any changes to the Winchester disk.

Resulting Display

After you choose and enter the "Abort" option, the Z-DOS system prompt will appear, as shown:

```
    A:
```

NOTE: Each partition feature will revert to the status it maintained before you invoked the PART utility. Any and all changes you may have entered during this PART session will be nullified, as if you had not even invoked PART.

Error Conditions

If you enter a character other than M or A, PART will cause the computer to beep and the cursor to remain at the end of the prompt until you enter a valid letter.

The Superblock

NOTE: Information concerning the superblock is not essential for use of the PART utility or the Winchester disk. This information is provided for users who wish to obtain a deeper understanding of some of the activities that PART performs in order to partition a Winchester disk.

Winchester disk space is allocated according to information contained within a unit of software that is stored on a reserved area of the Winchester disk. This unit of software is called the "superblock".

The superblock is a unit of Winchester support software that enables you to access specific partitions. The superblock is initially recorded on your disk when you use the PREP utility.

The superblock is also recorded on Winchester disks obtained from Zenith Data Systems or Heath — whether or not you use the PREP utility.

To insure the integrity of the superblock information, two copies of the superblock are recorded on the disk. These copies are named "Superblock A" (the copy used in most cases) and "Superblock B" (a backup copy used only when Superblock A is unusable).

After you use PART to change partitioning features (such as partition name, system name, allocation percentage, and default boot partition), PART updates both copies of the superblock and other Winchester support software units. Other Winchester support software units are explained in the text entitled "The Reserved Winchester Area" under "PREP".

The information within each copy of the superblock resembles the information shown on the partition table that is displayed when you use the PART utility. The superblock is structured as shown in Table A-1.

APPENDIX A

PART

	16 bytes	10 bytes	1 byte	3 bytes
1.	Partition Name	System Name	flag	Start Sector
2.	Partition Name	System Name	flag	Start Sector
3.	Partition Name	System Name	flag	Start Sector
4.	Partition Name	System Name	flag	Start Sector
5.	Partition Name	System Name	flag	Start Sector
6.	Partition Name	System Name	flag	Start Sector
7.	Partition Name	System Name	flag	Start Sector
8.	Partition Name	System Name	flag	Start Sector
9.	Partition Name	System Name	flag	Start Sector
10.	Partition Name	System Name	flag	Start Sector
11.	Partition Name	System Name	flag	Start Sector
12.	Partition Name	System Name	flag	Start Sector
13.	Partition Name	System Name	flag	Start Sector
14.	Partition Name	System Name	flag	Start Sector
15.	Partition Name	System Name	flag	Start Sector
16.	Partition Name	System Name	flag	Start Sector
17.	blanks	blanks	blanks	Start Sector
	16 bytes	10 bytes	1 byte	3 bytes

Table 4-1 Superblock Structure

NOTE: The superblock does not include all of the software necessary to facilitate Winchester disk access. Other Winchester support software, used for Winchester bootup and isolation of unusable disk media, are explained in the text entitled "The Reserved Winchester Area" under "PREP".

The Start Sector

The structure of the superblock is similar to the layout of the partition table that is displayed while you are using PART. For each partition, it contains a partition name (1–16 characters) and an operating system name (1–10 characters).

However, in undating the superblock, PART converts the allocation percentages that you entered into a different kind of statistic that can be used during partition access to determine where a partition begins and ends. PART converts the percentage value into the number of the start sector number of each partition.

Although you can establish only 16 partitions on the Winchester disk, there are 17 partition entries defined in the superblock. The size of each partition is determined by the difference between the "Start Sector" values of adjacent partitions.

It is necessary to define a 17th partition so that the size of the 16th partition (if established by the user) can be calculated by subtracting the start sector value of the 16th partition from the start sector value of the 17th partition.

The partition preceding the first partition that has spaces (20H) entered for its partition name will be considered to be the last usable partition defined in the table. The start sector of this partition will be one greater than the last sector allocated. The partition name entry for the 17th partition is always spaces.

The Flag Byte

Partitions are also labeled with a flag byte.

The flag byte contains special information about the partition. For this version of the PART utility, only the high order bit is defined. This bit is set to one whenever PREP is run, or whenever PART is run with PART changes being made. The operating system that is eventually recorded on the disk can reset this byte when the system's formatting utility is run on the partition.

The FORMAT utility supplied with this version of CP/M-86 does not use or set this flag byte.

Disk Space Calculation

The PART utility allocates portions of Winchester disk space using 512-byte sectors as the primary unit of measure, and converts the quantities of sectors (512-byte units) into kilobytes (1024-byte units) for the totals displayed on the screen. PART determines how much space to allocate by performing the following internal operations in sequence:

1. observing the total size of the Winchester disk being used,
2. monitoring the percentages you enter,
3. calculating the number of sectors that is closest to this percentage,
4. converting sectors to kilobytes, and
5. displaying this kilobyte quantity on the screen.

Thus the amount of disk space that is actually allocated will not always be exactly equal to the percentage you entered, but rounded to the nearest kilobyte.

If you enter an allocation percentage that is greater than the remaining space percentage on the disk, then PART will calculate the number of kilobytes of space remaining on the disk, calculate this amount to the nearest one percent, and display this remaining percentage instead of the percentage that you tried to enter.

Because of PART's rounding of space portions to the nearest percentage, the percentages allocated to your partitions might not always add up to exactly 100 percent when the disk is full. Furthermore, this rounding can also cause partitions that were divided into equal percentages to have slightly unequal kilobyte capacities.

Verifying the Superblock

PART updates information in Superblock A and Superblock B at the locations where these superblock copies were initially recorded (by the PREP utility). By spacing these two copies of the superblock several sectors apart, PART decreases the chance that both copies could be damaged simultaneously.

A checking code called a "checksum" is calculated by PART for each of the copies of the superblock before PART records these superblock copies on the Winchester disk. The results of these checksums are recorded in a data structure known as the Software Boot Code (see text entitled "The Reserved Winchester Area" under "PREP").

Then, when execution of either PART or PREP is repeated on the same Winchester disk, the utility first performs checksums to verify that the superblocks have not changed since the original checksums were performed.

If the PART utility encounters difficulty in reading Superblock A, or if the results of the second checksum of Superblock A differ from the results of the original checksum, then PART tries to read Superblock B.

If the PART utility encounters difficulty in reading Superblock B, or if the results of the second checksum of Superblock B differ from the results of the original checksum, then PART will display an error message.

PART Error Messages

```
Allocation error
```

You have established one or more partitions that are presently allocated zero percent of the disk. (PART might have subtracted from the original amount of space you allocated to these partitions when you allocated too much space to other partitions.) You must repeat the "Partition maintenance" operation and change allocation percentages of one or more partitions to conform with percentage entry rules (see "Allocation Percentage").

```
Boot Partition error
```

You have removed the default boot partition from the table during a "Partition maintenance" operation. You can either replace this partition through the "Partition maintenance" operation, or select valid default boot partition through the "Modify default boot partition" or "Partition maintenance" operation.

```
Duplicate Names error
```

You have established more than one partition with both the same partition name or the same operating system name. You must change at least one of the names of one of these partitions through the "Partition maintenance" operation.

NOTE: The above three error messages can appear in a series if more than one error condition exists at the same time. The form of these series error messages can be:

```
Allocation and Boot Partition error
Allocation and Duplicate Names error
Boot Partition and Duplicate Names error
Allocation and Boot Partition and Duplicate Names error
```

All such error messages are displayed in reverse video, between the default boot partition position and the operations menu.

The conditions that produce any of these error messages will also prevent you from exiting from the PART utility until you have corrected the errant condition(s).

```
Unable to communicate with the Z-217 controller
```

The PART utility was unsuccessful in an attempt to access the Z-217 controller, which controls the Winchester disk. This problem could indicate that the Z-217 controller is not firmly plugged into the S-100 bus, the drive cable connectors are not securely fastened, or that the controller has a hardware malfunction. Check to see that the controller card and all cable connectors are secure. Then run PREP and PART in sequence. If this error occurs after repeated attempts to run PREP and PART, contact Zenith Data Systems Technical Consultation for assistance.

```
    Z-217 controller error on Set Drive Parameters command
```

One or more responses to the five drive characteristic questions were not valid for the particular drive connected. A malfunction of the Z-217 controller is also possible. Recheck the characteristics of your drive. Then run PREP and PART in sequence. If this error occurs after repeated attempts to run PREP and PART, contact Zenith Data Systems Technical Consultation for assistance.

```
    Error - unable to re-write tables
```

PART is unable to record changes to the superblock after you have specified changes to the partition table and exited from PART. PART might succeed in recording some of the changed superblock information over the old superblock before this error message occurs, leaving portions of new and old superblocks on your Winchester disk. Therefore, you should use the PREP utility and then repeat the PART utility.

```
    Unable to read superblock/SBC, disk unusable
```

PART either unsuccessful in reading information stored in the existing software boot code, or successfully read the software boot code and detected a checksum error. Use the PREP utility and then repeat the PART utility.

APPENDIX A

PART

```
Fatal Error -- Can not read superblock B.
```

A bad sector error has occurred in the backup copy of the superblock (Superblock B). Try to run the PREP utilities on the disk in sequence. If repeated attempts to use PREP and PART fail, then contact Zenith Data Systems Technical Consultation for assistance.

APPENDIX B PREP

The Utility that Initializes a Winchester Disk

The PREP utility prepares the magnetic surface of many different types of Winchester disk for use as mass storage devices in a Z-100 environment. The PREP utility is seldom (if ever) needed by most users.

The PREP utility is recorded on the Winchester Utility Disk, which is shipped with your Winchester disk hardware documentation. Although this disk runs under the Z-DOS Operating System, you can boot up with it — just as you would with a bootable CP/M-86 disk.

CAUTION: Using PREP will destroy all software and/or data stored on your Winchester disk. Do not use PREP until you have transferred your Winchester disk files to floppy disks. Winchester disks supplied by Zenith Data Systems or Heath have already been prepared by PREP before they are shipped. Therefore, users of these disks will need to use PREP only if they are consistently encountering an unreasonable number of disk access errors, and cannot correct this problem with the VERIFY utility.

NOTE: Before you use the PREP utility, a hardware component called a "jumper" must be installed at the "format enable" location on your Z-217 Winchester disk drive controller card. This jumper is already installed (although unused) at a different location on your Z-217 controller card when the card is shipped. Therefore, before you can use the PREP utility, you must move the jumper to the "format enable" location. This procedure is explained under the heading "PREP Hardware Adjustments" of the text on PREP.

After you use the PREP utility, you will have to reset and reboot the system with a bootable floppy disk.

Additionally, you should remove the jumper from the format enable location on the Z-217 controller card between the time you finish using PREP and the time you perform routine activities that involve data storage on the Winchester disk.

APPENDIX B

PREP

WARNING: Unplug your computer from its power source before touching any hardware component within the computer's cabinet.

Winchester Disk Features

"Winchester disks" come in a variety of sizes and configurations, but they all have common features. The central feature and core of a Winchester disk device is a rigid platter. The typical platter consists of a non-magnetic metal (generally aluminum) disk, coated with a thin plating of ferric oxide or cobalt. This platter itself is the Winchester disk, as opposed to the floppy disk, which has a plastic (usually mylar) core with a thin coating of a similar magnetic substrate.

Platters

Winchester disk platters are generally sealed to prevent particulate matter from the environment (such as dust, smoke, dirt, or hair) from contaminating a platter's surface or read/write head. Winchester disks are available in a variety of sizes and with one or more platters. Winchester disks can be either "fixed" or "removable". The fixed disk is permanently mounted inside the device, but removable Winchester disks come in disk packs or cartridges and may be removed or interchanged.

Read/Write Heads

The read/write heads are electromagnets that slide back and forth a fraction of an inch above the surface of Winchester disk platters. The movement of a Winchester disk drive's read/write heads between the hub of the platters and the edge of the platters is called "stepping". Therefore, this movement is measured by an amount known as the "step rate".

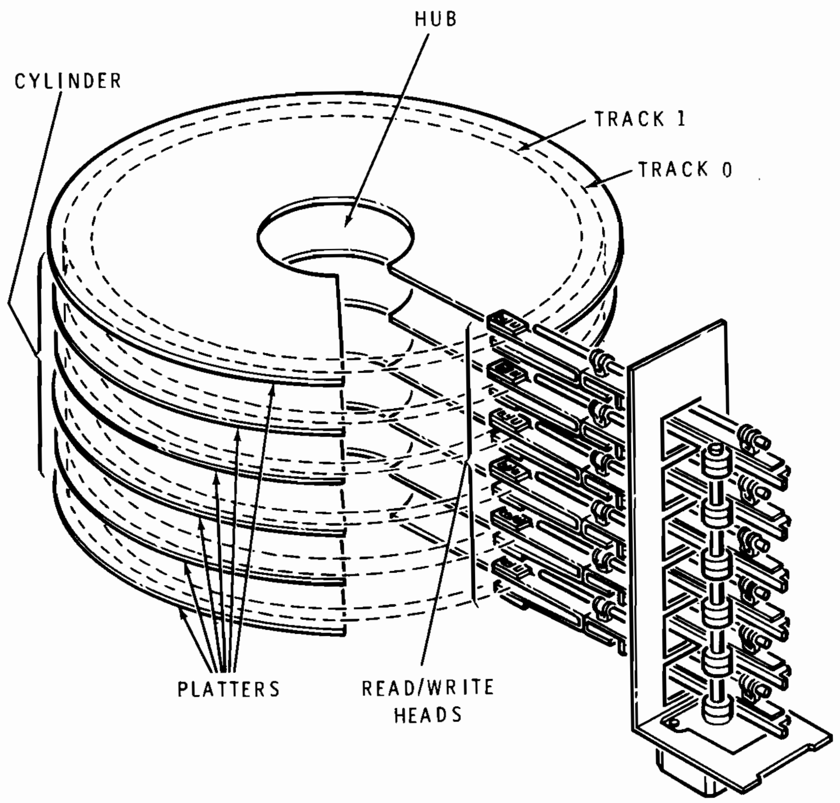


Figure B-1 Winchester Disk Components

APPENDIX B

Logical Winchester Disk Divisions

Winchester disks that are supplied by Zenith Data Systems and Heath can be divided into several logical subunits called "partitions". This is partly because large quantities of storage locations are easier to deal with if they are subdivided. The various subdivisions help speed storage and retrieval of data.

Sectors

A "sector" is the basic unit of data organization for disk drive devices. Like floppy disks, Winchester disks are divided into sectors. Winchester disk sectors under CP/M-86 are 512 bytes long.

Tracks and Cylinders

Each recording surface of a Winchester disk platter is also divided into concentric rings called "tracks", which are similar to the tracks of a floppy disk. The Winchester disks initialized by PREP are formatted with 18 sectors per track. A further division of a Winchester disks storage area is the cylinder (see Figure B-1). A "cylinder" is a collection of all tracks that are located the same distance from the outer edge of each recording surface. Winchester disk read/write heads can access all of the data stored on a particular cylinder without any stepping movement.

For example, if a Winchester disk drive has four read/write heads, the drive can access a cylinder of 72 sectors (4 tracks times 18 sectors) without stepping (moving the heads). This amounts to a total of 36864 bytes (36 kilobytes) being read or written.

Invoking PREP

Invoke PREP by entering the following command at the Z-DOS system prompt:

```
A:**PREP RETURN**
```

When you invoke PREP, the following display appears:

```
                    PREP version 1.00
         Copyright (C) 1982, Zenith Data Systems

The PREP utility helps you to:

  *  initialize surface of Winchester disk
  *  test data retention capabilities of Winchester disk media
  *  isolate questionable disk sectors
  *  divide the surface of Winchester disk into two partitions
     of equal size (one Z-DOS partition and one CP/M partition)

PREP may prompt you to specify five Winchester disk characteristics in
order to identify the type of Winchester disk you have installed.
Then PREP displays messages as it operates on the disk.

Caution:  Using PREP can destroy all files on your Winchester disk.
Winchester disks supplied by Zenith or Heath are prepared by PREP
before they are shipped. Users of these disks will use PREP only
after consistently encountering an unreasonable number of disk access
errors. Do not use PREP until you have transferred backup copies of
your Winchester disk files to floppy disks.

Do you wish to proceed with PREP (Y/N)?
```

Typing an **N** at this prompt ends the PREP utility and returns you to the system prompt.

Typing a **Y** causes PREP to display the following prompt:

```
Please type P to proceed
```

Typing any response other than P ends the PREP utility and returns you to the system prompt.

Responding to Disk Characteristic Prompts

Typing **P** at the "Please type P to proceed" prompt causes PREP to continue operation.

If the disk has been previously prepared by PREP and no errors are found in the first sector of the boot code (see the text entitled "The Reserved Winchester Area"), PREP skips these disk characteristic prompts, assumes the disk surface has been previously initialized, and proceeds immediately to disk initialization (see the text entitled "Initializing the Disk").

If the disk has not been prepared with PREP, or if there is an error found in the first sector of the boot code (see "The Reserved Winchester Area"), then the following five disk characteristic prompts will appear in sequence:

```
    Enter number of heads in hex:
    Enter number of cylinders in hex:
    Enter reduced write current cylinder in hex:
    Enter pre-comp cylinder in hex:
    Enter step rate code in hex:
```

The significance of each of these disk characteristic prompts is explained here.

```
    Enter number of heads in hex:
```

At this prompt, type a hexadecimal value for the number of read/write heads contained in the drive you are preparing. Then press **RETURN**.

```
    Enter number of cylinders in hex:
```

At this prompt, type the hexadecimal value of the number of cylinders contained in the drive you are preparing. Then press **RETURN**.

For disks with floating read/write heads, this number would be equal to the total number of tracks divided by the total number of read/write heads. For disks with fixed read/write heads, this number would be equal to the total number of tracks divided by the total number of usable platter surfaces.

```
Enter reduced write current cylinder in hex:
```

At this prompt, type the hexadecimal value for the location of the first cylinder at which read/write head current must be reduced. Then press **RETURN**.

Toward the hub of the platters, where the circumference of each cylinder is smaller, data storage sectors are recorded closer together than the sectors on cylinders near the edge of the platters. Therefore, some Winchester disk drives reduce the electrical current sent to the read/write head when they write data on cylinders that are close to the hub of the disk platters. This reduction of current helps to prevent magnetic interference between the data sectors that are recorded extremely close together.

```
Enter pre-comp cylinder in hex:
```

At this prompt, enter the hexadecimal value for the number of the first cylinder at which write precompensation must take place. Then press **RETURN**.

On the cylinders located close to the hub of a Winchester disk, where data is recorded at extremely high density, bit shift can occur. “Bit shift” is a phenomenon where the data bits written at a particular location spread apart slightly on the media after they have been written. Bit shift is most likely to occur when similar bits are written close together. The “pre-comp” (write precompensation) characteristic compensates for bit shift by writing some bits earlier or later that the normal rate of data writing. Precompensation for Winchester disks causes a slight deviation (about 12 nanoseconds for every 100 nanoseconds) from the normal rate of data writing during the writing of bits that are apt to shift.

```
Enter step rate code in hex:
```

At this prompt, you should enter the hexadecimal value code that indicates the rate at which the read/write heads step between tracks. Then press **RETURN**.

After you respond to the “step rate code” prompt, PREP will automatically begin to perform its three operations (see “PREP Operations”) in sequence.

APPENDIX B

PREP

The PREP utility is capable of preparing a wide variety of Winchester disks for data storage. Table B-2 lists several different Winchester disks available in the microcomputer market and indicates the hexadecimal values you should enter to prepare each disk when prompted by PART for a specific disk characteristic.

The labels for the columns of numbers in Table B-2 correspond to PREP characteristic prompts. If the model number of your Winchester disk is listed in the left-hand column of the table, then enter the numbers listed in the right-hand columns in sequence as the characteristic prompts are displayed.

According to Table B-2, if you have a Miniscribe Mod II 2012 Winchester disk drive, you should respond to the drive characteristic prompts as follows:

```
Enter number of heads in hex: 4 RETURN
Enter number of cylinders in hex: 132 RETURN
Enter reduced write current cylinder in hex: 200 RETURN
Enter pre-comp cylinder in hex: 80 RETURN
Enter steprate code in hex: 1 RETURN
```

Drive Model	Total Heads	Total Cylinders	Reduced Current Cylinder	Write Pre-comp Cylinder	Step Rate Code
Seagate					
ST-406	2	132	200	80	1
ST-412	4	132	200	80	1
ST-419	6	132	200	80	1
ST-506	4	99	80	40	96
ST-706	2	132	200	80	1
Miniscribe					
Mod II 2012	4	132	200	80	1
Mod III 3012	2	264	300	80	1
Mod IV 4020	4	1E0	200	80	1
IMI					
5006H	2	132	200	D6	1
5012H	4	132	200	D6	1
5018H	6	132	200	D6	1
Tandon					
TM 602S	4	99	80	40	96
TM 603S	6	99	80	40	96

Table B-2 Responses to PREP Characteristic Prompts

PREP Operations

PREP begins to prepare the surface of your Winchester disk either after displaying the initial screen messages or after you respond to the five disk characteristic prompts. PREP prepares the disk by performing three operations in sequence: Initializing the Disk, Testing the Disk Media, and Initializing the Reserved Winchester Area.

Initializing the Disk

After you have responded to the disk characteristic prompts, PREP initializes the surface of the disk for the test that will follow. While this occurs, you will see the message:

```
Initializing the disk...
```

This initialization is similar to FORMAT in that it magnetically records a map of all sectors on the disk surface. When the surface has been initialized, the message shows:

```
Initializing the disk...completed
```

and then PREP begins testing the disk media.

NOTE: If your Winchester disk hardware has not been properly adjusted, the following message will be displayed instead of the "Initializing the disk..." message:

```
Error during formatting of the drive.
```

If this message appears, you must now perform a hardware adjustment as explained in the text entitled "PREP Hardware Adjustments".

Testing the Disk Media

PREP performs seven test passes to check the integrity of the disk's storage capability. During each pass, PREP writes a predetermined code to each sector (the drive light will flicker) and then reads back that code to verify that it remained correct (the drive light will appear as constantly on). PREP keeps you informed of its progress by displaying the message:

```
Media test in progress, pass n
```

Where n is the number (in the range 1-7) of the pass that it is currently conducting.

Be patient. This PREP operation can take from 45 to 90 minutes because of the large number of sectors that PREP must test.

PREP uses a different code on each pass it makes through the test. If PREP finds sectors containing unusable media, it stores the address of these sectors, and later places these sector addresses into a bad sector table.

Initializing the Reserved Winchester Area

After completing the media tests, PREP records and verifies the Reserved Winchester Area (see "The Reserved Winchester Area") on the first several sectors of the Winchester disk. During this operation, PREP displays the following message:

```
Initializing the disk...
```

If PREP adds the word completed to the end of this display, and displays the system prompt, then all PREP operations are complete. The display should appear as follows:

```
Initializing the disk...completed

A:
```

Then you should reset the system and boot up with a bootable floppy disk. If you wish to use the PART utility immediately, then boot up with a copy of the Winchester Utility Disk. If you wish to perform any other operation, then boot up with a floppy disk.

NOTE: You will not be able to access any partition after using PREP until you reset the system and boot up with a bootable floppy disk.

CAUTION: After using PREP, you should remove a hardware component called a jumper from the "format enable" location on the Z-217 Winchester Disk Controller Board. Refer to the text entitled "PREP Hardware Adjustments" for instructions on removing this component. This procedure will help to protect the data on your Winchester disk from being destroyed accidentally.

The Reserved Winchester Area

NOTE: Information concerning the Reserved Winchester Area is not essential for use of the PREP utility or the Winchester disk. This information is provided for users who wish to obtain a deeper understanding of the operations that PREP performs in order to prepare a Winchester disk.

When the PREP utility is run, it records units of Winchester support software on the first 36 usable sectors of the Winchester disk. These software units are collectively known as the Reserved Winchester Area. They are recorded on the Winchester disk during PREP's reserved area initialization operation (see Initializing the Reserved Winchester Area). These software units are arranged as shown in Table B-3.

SECTORS USED	WINCHESTER SUPPORT SOFTWARE UNITS
5	Software Boot Code (SBC)
1	Superblock A
1	Bad Sector Table A
11	blank
1	Superblock B
1	Bad Sector Table B
16	blank
36	TOTAL RESERVED WINCHESTER AREA

Table B-3 Winchester Support Software Units within Reserved Winchester Area

APPENDIX B
PREP

The most important Winchester support software units listed in Table B-3 are the software boot code, the superblocks, and the bad sector tables.

These units are vital to you during Winchester bootup because they help you to access a particular partition after you access the Winchester disk itself. Users of CP/M-86 with the Winchester disk also use these data structures to make unusable media (bad sectors) inaccessible before FORMAT is run.

The blanks inserted between the "A" copies of the superblock and bad sector table and the "B" copies of these units help to decrease the chance that all important Winchester support software units could be damaged simultaneously.

If you must access any part of the reserved Winchester area, you can determine the location of the Winchester support software units by examining the pointers in the software boot code. The software boot code will begin at sector zero regardless of the location of the other software units.

The Software Boot Code (SBC)

The "software boot code" (SBC) is a Winchester support unit that helps locate the partition to be booted after entry of a Winchester disk bootup command. The SBC also helps you to avoid bad sectors during disk access by referring to a bad sector table.

PREP records the SBC on the first 5 sectors of the Winchester disk during initialization of the reserved Winchester area (see "Initializing the Reserved Winchester Area").

Role of SBC During Bootup

When you enter a Winchester disk bootup command (see "Bootup with a Winchester Disk"), the computer will load the SBC into Random Access Memory (RAM).

Once within RAM, the SBC begins to access a partition. The partition that is accessed is determined either by a bootstring or a default boot partition. A bootstring can be specified by the user during bootup. A default boot partition is stored in a fixed location within the SBC (see the text entitled "SBC Entries").

In order to access a partition, the SBC must match the specified bootstring or default boot partition with a partition that exists in the superblock's table of partitions (see the text entitled "The Superblock").

When the SBC finds a partition that matches the specified bootstring or default boot partition, the SBC loads the first 32 sectors of that partition into RAM. If the accessed partition contains CP/M-86, then the CP/M-86 boot loader program will begin to execute the remainder of the bootup operation.

SBC Entries

The entries included in the first 128 bytes of the SBC are described in Table B-4.

BYTES	SBC ENTRIES
3	System bytes
1	PART/SBC version number (Used to synchronize different releases of software)
1	PART/SBC revision number (Used to synchronize different releases of software)
27	Default bootstring (16 bytes define the partition name, one byte defines the semicolon, and 10 bytes define the operating system name)
3	Beginning sector address of bad sector table A
3	Beginning sector address of bad sector table B
3	Beginning sector address of superblock A
3	Beginning sector address of superblock B
2	Sector size (512 bytes per sector)
2	Sectors per track (18)
6	Reserved for future expansion
3	Number of sectors on entire Winchester disk
1	Reserved for future expansion
2	Checksum for superblock copy A
2	Checksum for superblock copy B
2	Checksum for bad sector table copy A
2	Checksum for bad sector table copy B
12	Set drive for Z-217 controller
3	Address of first user sector (first sector beyond Reserved Winchester Area)
6	Date partitioned, or default date, when PART is run (When PREP is run, the value 00 is used for each byte.)
2	Checksum of SBC (assuming initial value is zero)
39	Reserved for future expansion
128	FIRST QUARTER-SECTOR OF SBC

Table B-4 Software Boot Code Entries

The format for each three-byte sector number is low, middle, high byte.

NOTE: Table B-4 describes one quarter-sector of the five-sector SBC. The rest of the SBC consists of the assembly instructions that lead to the actual access of the specified partition.

SBC Verification

A checking code called a "checksum" is calculated by PREP for the SBC before PREP records the SBC on the disk. The results of these checksums are recorded in entries within the SBC.

Then, when execution of either PART or PREP is repeated on the same Winchester disk, verification checksums are performed to verify that the SBC has not changed since the original checksums were performed.

If the SBC has changed, or if it cannot be read, then an error message will be displayed.

The SBC also contains the checksums used to verify the other Winchester support units (the superblocks and bad sector tables).

The Superblock

The "superblock" is a Winchester support unit that contains information about each partition on the disk. It contains the following items for each of 17 defined partitions:

- partition name
- operating system name
- flag byte (to show whether PREP or PART has been run on the disk since the last time the partition was formatted)
- address of the starting sector

Superblock Entries

These items occupy 30 bytes per entry. The 17 superblock entries are structured as shown in Table B-5.

BYTES	SUPERBLOCK ENTRY
30	Entry for 1st partition (including 16-bytes partition name, 10-byte system name, 1-byte flag, and 3-byte starting sector)
30	Entry for 2nd partition (including 16-bytes partition name, 10-byte system name, 1-byte flag, and 3-byte starting sector)
30	Entry for 3rd partition (including 16-bytes partition name, 10-byte system name, 1-byte flag, and 3-byte starting sector)
.	.
.	.
.	.
30	Entry for 16th partition (including 16-bytes partition name, 10-byte system name, 1-byte flag, and 3-byte starting sector)
30	Entry for 17th partition (including 16-bytes partition name, 10-byte system name, 1-byte flag, and 3-byte starting sector)
2	Reserved for future expansion
512	TOTAL FOR EACH SUPERBLOCK

Table B-5 Superblock Entries

NOTE: Refer to the text entitled “The Superblock” in the text on PART for a detailed explanation of the components of each superblock entry and other information concerning the superblock.

Superblock Verification

During initialization of the reserved Winchester area (see the text entitled "Initializing the Reserved Winchester Area"), PREP records superblock entries twice on the Winchester disk. The primary copy of the superblock, called Superblock A, is used unless some of its contents have been damaged since it was recorded. The backup copy of the superblock, called Superblock B, is used if Superblock A is damaged. Each copy is recorded several sectors apart to decrease the chance that both could be damaged simultaneously.

A checking code called a "checksum" is calculated by PREP for each superblock copy before PREP records these superblock copies on the Winchester disk. The results of these checksums are recorded in the software boot code (see "The Software Boot Code (SBC)").

Then, when execution of either PART or PREP is repeated, verification checksums are performed to verify that the superblocks have not changed since the original checksums were performed.

If Superblock A cannot be read, or if the results of the second checksum of Superblock A differ from the results of the original checksum, then the utility tries to read Superblock B.

If Superblock B cannot be read, or if the results of the second checksum of Superblock B differ from the results of the original checksum, then all partitions will be inaccessible.

The Bad Sector Table

The "bad sector table" is an ordered list of the addresses of each sector on the disk that contains unusable media. The information in the bad sector table enables CP/M-86 to avoid bad sectors (unusable media) when it accesses a partition during your everyday activities.

The bad sector table can include the addresses of as many as 169 bad sectors. Each bad sector address is recorded in a three-byte entry. Entries that do not contain the address of a bad sector are filled with three zeroes.

Bad Sector Table Entries

The structure of the bad sector table is explained in Table B-6.

During media testing (see "Testing the Disk Media"), PREP maintains a record of the location of all the bad sectors (sectors containing unusable media) that it finds on the disk. Then, during initialization of the reserved Winchester area (see "Initializing the Reserved Winchester Area"), PREP creates a table of all bad sectors and records two copies of this table on the disk.

BYTES	BAD SECTOR TABLE ENTRY
3	Address of 1st bad sector found by PREP
3	Address of 2nd bad sector found by PREP
3	Address of 3rd bad sector found by PREP
3	Address of 4th bad sector found by PREP
.	.
.	.
.	.
3	Address of 168th bad sector found by PREP
3	Address of 169th bad sector found by PREP
3	Last entry in table (always contains 000)
2	Reserved for future expansion
512	TOTAL FOR EACH BAD SECTOR TABLE

Table B-6 Bad Sector Table Entries

Bad Sector Table Verification

The primary copy of the bad sector table, called Bad Sector Table A, is used unless some of its contents have been damaged since it was recorded. The backup copy of the bad sector table, called Bad Sector Table B, is used if Bad Sector Table A is damaged. Each copy is recorded several sectors apart, to decrease the chance that both could be damaged simultaneously.

A checking code called a "checksum" is calculated by PREP for each of the copies of the bad sector table before PREP records these bad sector table copies on the Winchester disk. The results of these checksums are recorded in the software boot code (see "The Software Boot Code (SBC)").

Then, when execution of PREP or PART is repeated on the same Winchester disk, verification checksums are performed to verify that the bad sector tables have not changed since the original checksums were performed.

If Bad Sector Table A cannot be read, or if the results of the second checksum of Bad Sector Table A differ from the results of the original checksum, then Bad Sector Table B is read.

If Bad Sector Table B cannot be read, or if the results of the second checksum of Bad Sector Table B differ from the results of the original checksum, then no bad sector table information will be available in the Reserved Winchester Area. If the FORMAT utility is then used on a partition of this disk, it will assume that the disk has no bad sectors and format without avoiding any bad sectors. If the VERIFY utility is used on this disk, it will find no bad sector table to which it can append new bad sectors. Therefore, it will search the disk for all bad sectors and create a new bad sector table.

APPENDIX B

PREP

PREP Hardware Adjustments

Before you can use the PREP utility, a hardware component within your Z-100 computer must be moved from one location to another. This text section explains the procedure you must perform in order to properly move this component.

This component must be moved because its position during use of PREP is different from its position during all other Winchester disk activities.

Jumper Description

This hardware component is called a "jumper". A jumper is an insulated metal clip or wire used to connect different locations on a circuit board.

In this case, the jumper used is a small conductive metal clip covered with a box-like plastic case. This kind of jumper is known as a Berg jumper.

This jumper is designed to fit over two metal pins protruding from the circuit board known as the Z-217 Winchester Disk Controller Board (or controller card).

When you use PREP, the jumper must cover two of the pins on the Z-217 to allow PREP to initialize the Winchester disk. When you perform any activity other than PREP, the jumper must be stored at a different location on the Z-217.

Jumper Movement Procedure

This procedure explains the sequence of steps you should perform to adjust your hardware both before and after using PREP.

WARNING: The internal components of your computer can cause severe electric shock if touched while the computer is running. Therefore, you should turn off your computer and unplug it from its power source before touching any hardware component within the computer's cabinet.

1. Remove any disk that may be in the floppy disk drive.

2. Turn off your computer's power and unplug it from the power source.

3. Remove the cabinet top from your computer. (Refer to Appendix I of the Z-100 Series User's Manual for detailed information on removing this top.)

4. Locate the Z-217 Winchester Disk Controller Board in the "Card Cage".

   NOTE: Your computer also contains a disk controller board for floppy disks (called the Z-207). The Z-217 Winchester Disk Controller Board is the board that is connected to the Winchester disk drive (rather than the floppy disk drive) by a flat, ribbon-like cable.

5. Gently slide the Z-217 controller board upward until the area shown in Inset #1 of Figure B-2 is above the top of the card cage. Do not remove the Z-217 board completely from the card cage. As you slide the board upward, be certain that some of the board is still anchored between the card cage's vertical tracks.

   NOTE: In order to slide the Z-217 controller board upward, you might first have to temporarily unplug Z-217 cables or a cable that lies above the Z-217 board.

6. Locate the jumper covering the pins shown in Inset #1 of Figure B-2. Remove this jumper by carefully sliding it away from the board. Be careful not to bend the pins.

   NOTE: The pins shown in Inset #1 are known as the Z-217 interrupt pins. These pins need not be connected by the jumper while you are using PREP.

7. Locate the two pins shown in Inset #2 of Figure B-2. Carefully slide the jumper over these pins. Be careful not to bend the pins.

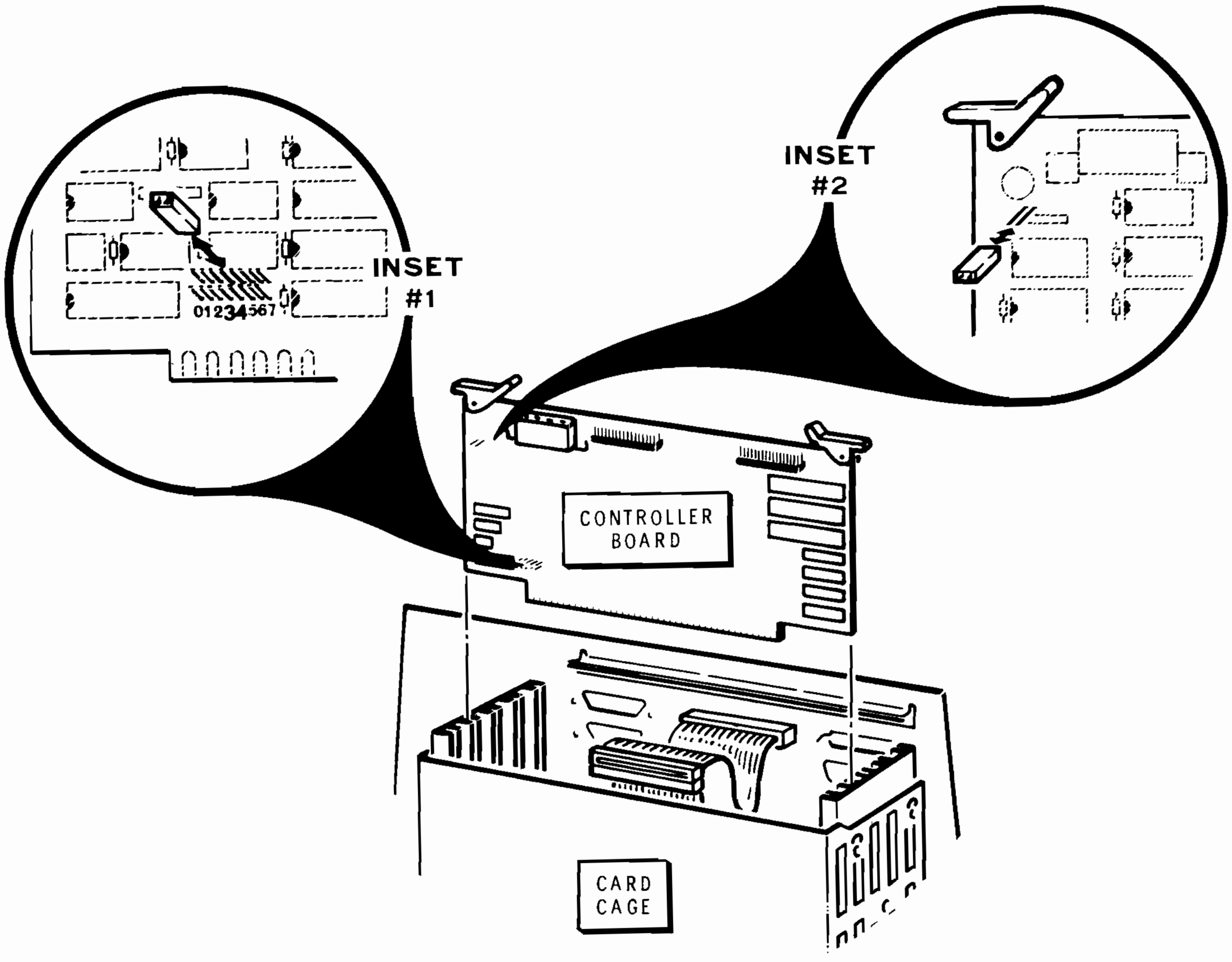


Figure B-2 Z-217 Controller Board Adjustment

NOTE: The pins shown in Inset #2 are known as the "format enable" pins. When connected by the jumper, your Winchester disk can be initialized by PREP.

8. Gently slide the Z-217 board downward until the bottom edge of the board is securely engaged to the horizontal connector at the bottom of the card cage.

   NOTE: Be careful that the Z-217 remains between the same pair of vertical card cage tracks as you slide it downward.

9. If you have unplugged any internal cables, plug them back into the apropriate sockets on the controller boards. Make sure that all cables are firmly connected.

10. Replace the top cover of your computer and make sure that it is completely latched into position.

11. Plug the computer back into the power source and turn on the computer.

Proceed to use the PREP utility as explained in the text entitled "Invoking PREP".

When you have finihsed using the PREP utility, perform the steps of this procedure in reverse sequence to remove the jumper from the Z-217 format enable pins and replace it on the Z-217 interrupt pins.

CAUTION: You must remove the jumper from the format enable pins before performing any other activity. If this jumper remains on the format enable pins, irregularities in the power supply can cause the Winchester disk to be automatically initialized during normal use of the disk. This initialization will destroy any data recorded on the disk.

PREP Error Messages

```
Bad sector count exceeded for this drive.
```

Cause: The upper bound limit for bad sectors has been exceeded. This could indicate a hardware malfunction.

Cure: Run PREP again. If this error message reappears after repeated use of PREP, then contact Zenith Data Systems Technical Consultation for assistance.

```
Error -- Can not read superblock A.
```

Cause: A bad sector error has occurred in the primary superblock (Superblock A).

Cure: PREP will automatically use the backup copy of the superblock (Superblock B) and resume the operation it was conducting when the error message was displayed. However, this message indicates that only one usable copy of the superblock remains on the disk. Although you could use the disk in this condition, all Winchester disk data will become inaccessible if Superblock B is ever damaged. Therefore, we recommend that you run PREP again if this error message appears.

```
Error -- Drive capacity > 32 megabytes!
```

Cause: PREP has calculated that the Winchester disk drive connected to the Z-217 controller is larger than the maximum allowable size of 32 megabytes.

Cure: Run PREP, being careful to respond with the correct values to the drive characteristic prompts.

```
Error during formatting of the drive.
```

Cause: This could mean that you responded incorrectly to the five prompts about the drive's characteristics. This message could also indicate either a hardware malfunction or improper positioning of the format enable jumper on the Z-217 controller board.

Cure: Refer to the text entitled "PREP Hardware Adjustment". If you have not already done so, move the jumper to the "format enable" position on the Z-217 controller board. Then invoke PREP again, and double check your responses to the five drive characteristic prompts if they appear. After using PREP, remove the jumper from the "format enable" position.

```
Invalid HEX value, Try again:
```

Cause: Value entered was not a valid hexadecimal number, or the value entered was outside of the possible range.

Cure: Double check the appropriate hexadecimal value against the disk manufacturer's documentation or Table B.2 in this supplement. Then attempt to enter the correct value at the prompt again.

```
Track 0 contains bad sector(s).
```

Cause: A bad sector error has occurred in the reserved area of the Winchester disk. This could indicate a hardware malfunction.

Cure: Run PREP again. If this error message reappears after repeated use of PREP, then contact Zenith Data Systems Technical Consultation for assistance.

```
Unable to communicate with the Z217 controller
```

Cause: PREP can not locate the Z-217 controller. This could mean that the Z-217 is not firmly plugged into the S-100 bus, that the drive cable connectors are not securely fastened, or that the controller has a hardware malfunction.

Cure: Check to see that the controller card and all cable connectors are secure, and run PREP again.

```
Unable to write default PART values
```

Cause: An error was encountered as PREP attempted to write the superblocks. This error condition can be caused by media imperfections at the sectors where PREP is trying to write a copy of the superblock.

Cure: Run PREP again. If this error message reappears after repeated use of PREP, then contact Zenith Data Systems Technical Consultation for assistance.

```
Z-217 controller error on Set Drive parameters command
```

Cause: One or more responses to the five drive characteristic prompts were not valid for the particular drive connected. A malfunction of the Z-217 controller is also possible.

Cure: Recheck the drive characteristics and run PREP again. If this error occurs after repeated attempts to run PREP, consult Zenith Technical Consultation or your service representative.

APPENDIX C
SHIP

The Utility that Moves Read/Write Heads to a Safe Position

The SHIP utility moves the read/write heads of a Winchester disk to a position where they can not contact and destroy stored data on a disk platter in case of physical shock.

The SHIP utility is recorded on the Winchester Utility Disk, which is supplied with your Winchester disk hardware documentation. Although the Winchester Utility Disk runs under the Z-DOS Operating System, you can boot up with it — just as you would with a bootable CP/M-86 disk.

Winchester Disk Safety

Winchester disks are sensitive precision instruments that can be easily affected by physical shock or impact. The data stored on a Winchester disk is also vulnerable. Because of this vulnerability, you should take special precautions by running SHIP before shipping your Winchester disk, or even before moving the disk across the room.

The SHIP command enables you to protect your Winchester disk, and the data on the disk. SHIP affords this protection by moving the disk's read/write heads towards the hub of the Winchester disk platters. When in this position, the heads and platters will not be damaged by platter vibration that can be caused by physical shock.

Although platters can be caused to vibrate at their outside edges, the platter area near the hub is rigid enough to inhibit vibration. Therefore, the heads and platters are safer when the heads are near the hub.

Run SHIP whenever you intend to physically move the unit containing your Winchester disk.

APPENDIX C

SHIP

To use SHIP, invoke the utility at the system prompt and enter a cylinder address value at the prompt, as explained below.

NOTE: The Z-217 controller card causes the read/write heads to move to cylinder zero the first time you access the Winchester disk after power up. Therefore, the head positioning caused by SHIP will remain in effect only until you turn the disk on again and access it.

Invoking SHIP

To invoke SHIP by this method, type the following command:

```
     A:SHIP RETURN
```

When invoked, SHIP displays a message in the following form:

```
           SHIP version 1.00
Copyright(C) 1983, Zenith Data Systems

The SHIP utility helps you to:

  *  Position the read/write heads of the Winchester disk
     At a safe location for subsequent transportation
     of the Winchester disk unit.

SHIP will prompt you to specify a cylinder address to
identify where the read/write heads should be moved.

Enter shipping cylinder address in hex:
```

Entering the Shipping Cylinder

At the "Enter shipping cylinder address in hex:" prompt you should enter a hexadecimal value for the address of the cylinder at which the read/write heads should be positioned when the Winchester disk is physically moved. Refer to Table C-7 or to your Winchester disk hardware documentation if you are uncertain of this number. Then press **RETURN**.

NOTE: If you do not wish to move the heads at this time, then enter **CTRL-C** at the prompt.

After you have responded to the SHIP prompt, SHIP will move the read/write heads to the specified cylinder. Then Z-DOS will display the system prompt:

```
A:
```

NOTE: After using the SHIP utility, you will have to reset and reboot the system with a bootable floppy disk.

APPENDIX C

SHIP

Table C-7 shows the values you should enter in response to the SHIP prompt if you own any of the list Winchester disks.

Drive Model	Cylinder Position for Shipping
Seagate	
ST-406	131
ST-412	131
ST-419	131
ST-506	9A
ST-706	131
Miniscribe	
Mod II 2012	14F
Mod III 3012	28D
Mod IV 4020	209
IMI	
5006H	148
5012H	148
5018H	148
Tandon	
TM 602S	9A
TM 603S	9A

Table 4-7 Responses to SHIP Prompts

NOTE: If Table C-7 and your Winchester disk hardware documentation do not explain how the read/write heads should be positioned, then respond to the SHIP prompt by entering the hexadecimal value for the last cylinder on your disk. For instance if your disk has 132 cylinders, enter **132** and **RETURN** at this prompt.

SHIP Error Messages

```
Invalid HEX value, Try again:
```

Cause: Value entered was not a valid hexadecimal number, or the value entered was outside of the possible range.

Cure: Double check the appropriate hexadecimal value against the disk manufacturer's documentation or Table C-7. Then enter the correct value at the question again, and press **RETURN**.

```
Unable to communicate with the Z-217 controller
```

Cause: SHIP can not locate the Z-217 controller. This could mean that the Z-217 is not firmly plugged into the S-100 bus, that the drive cable connectors are not securely fastened, or that the controller has a hardware malfunction.

Cure: Check to see that the controller card and all cable connectors are secure, and run SHIP again.

```
Z-217 controller error on Set Drive Parameters command
```

Cause: Your response to the Enter shipping cylinder address in hex prompt was not valid for the particular drive connected. A malfunction of the Z-217 controller is also possible.

Cure: Recheck the drive parameters and run SHIP again. If this error occurs after repeated attempts to run SHIP, consult Zenith Technical Consultation or your service representative.

APPENDIX D
VERIFY

The Utility that Isolates Recent Bad Sectors on Your Winchester Disk

The VERIFY utility examines your Winchester disk for any bad sectors (media imperfections) that have occurred since the disk was shipped or since the PREP utility was last used. Then VERIFY adds the addresses of these bad sectors to a list of bad sectors that was recorded on the Winchester disk when PREP was run. This list is called the bad sector table.

The VERIFY utility is shipped on the Winchester Utility Disk.

The PREP utility has already been run on all Winchester disks supplied by Zenith Data Systems or Heath. (Refer to the text entitled "The Bad Sector Table" under "PREP" for information on the bad sector table.)

NOTE: After using the VERIFY utility, you must reset and reboot the system. Use bootable media other than the Winchester Utility Disk to reboot after using VERIFY.

Bad Sectors

Bad sectors are media imperfections that can cause hard errors during Winchester disk access operations. Hard errors are conditions in which an operation failed after a number of repeated attempts. Recovery from a hard error usually brings an abrupt end to the operation being attempted.

However, the VERIFY utility enables you to prevent hard errors from occurring in the future if these errors were caused by bad sectors. If VERIFY finds a reasonable number of new bad sectors (between 1 and 169), it adds them to the bad sector table that was originally created by the PREP utility.

APPENDIX D

VERIFY

The PREP utility (which is used before shipping on Winchester disks supplied by Zenith Data Systems or Heath) initializes the Winchester disk. PREP also helps you to make bad sectors inaccessible by creating a table of all the bad sectors, and storing this table on the Winchester disk.

Then, the next time you format a partition, FORMAT will take into consideration the newly acknowledged bad sectors. FORMAT will set up sector boundaries that will prevent usage of the bad sectors during all operations that occur after the formatting operation.

However, you might also obtain hard errors during Winchester disk access due to the following other problems:

- Excessive physical shock
- Entry of foreign material (such as smoke) into the Winchester disk chamber
- Malfunction of the Z-217 controller card
- Temporary loss of power to the disk

If one of these problems causes a hard error, then the disk might not have any new bad sectors for VERIFY to find. In such a case, you should back up the files from your Winchester disk. Then you should use the PREP utility. If you still encounter hard errors after using PREP, contact Zenith Data Systems Technical Consultation for assistance.

APPENDIX D
VERIFY

VERIFY Entries

VERIFY does not destroy any of the data on the Winchester disk. However, if you use VERIFY to isolate bad sectors, then we recommend that after VERIFY, you use BACKUP to copy all files from the partition on which the bad sectors occurred. Then you should use FORMAT on the partition on which the bad sectors occurred. Finally you should use RESTORE to replace the backed up files on this partition.

Invoking Verify

To invoke VERIFY, type the following command at the system prompt:

```
A:VERIFY RETURN
```

VERIFY will display the following message and prompt:

```
          VERIFY version 1.00
Copyright(C) 1982, Zenith Data Systems

The VERIFY utility helps you to:

  *  Locate sectors that have failed since you last ran PREP

Do you wish to proceed with VERIFY (Y/N)?
```

Confirming Intentions to use Verify

At the "Do you wish to proceed with VERIFY (Y/N)?" prompt, you can press **Y** to continue with the utility, or press any other key to exit to the system.

If you press Y to continue, VERIFY displays the following prompt:

```
Enter bad sector address, or zero to end:
```

APPENDIX D
VERIFY

Entering Bad Sector Addresses

When error conditions are encountered during Winchester disk access operations, CP/M-86 displays a hard error message that is slightly different from a floppy disk hard error message.

This Winchester disk error message appears in the following form:

```
HARD {operation} ERROR ON DRIVE {d:} STATUS {nn} SECTOR {ssss}
```

Where {operation} identifies the operation that was being performed when the error occurred. This operation could be worded as:
READ
or
WRITE
where {d:} is the name of the drive to which the partition was assigned when an error was encountered on the partition;
where {nn} is a code for the status of the error. These status codes are explained in the Z-217 Technical Manual; and
where {ssss} is the logical hexadecimal address of the sector on which the hard error occurred. (Logical sector addresses begin with the first sector on the entire Winchester disk, which is sector 0000.)

An error message in this form does not always indicate that your Winchester disk has bad sectors. When you receive such an error message, record the status code, and refer to the Z-217 Technical Manual to determine the meaining of this code. If the error message includes a status code with the number 80 or 81 or a number from 00 through 05 or from 20 through 40, then the error message probably does **not** indicate a bad sector or a reason for using the VERIFY utility.

However, if the error message includes a status code with a number from 10 through 17, then it is likely that your Winchester disk has developed a bad sector. Therefore, you should record the sector address that is also displayed in the error message, and use this sector address to respond to the "Enter bad sector address, or zero to end:" prompt.

We recommend that you also record the partition name and system name of the partition on which the error(s) occurred.

At the "Enter bad sector address, or zero to end:" prompt, enter the address of the logical sector at which the error(s) occurred, or type the digit zero (0) to begin media verification.

If you enter the address of a logical sector, then VERIFY will continue to display the

```
Enter bad sector address, or zero to end:
```

until you enter a zero.

VERIFY Operation

When VERIFY begins to search for bad sectors (after you have typed a zero at the bad sector address prompt) VERIFY displays the message:

```
Beginning verification...
```

Verification Completion

When VERIFY is finished verifying the disk, it will display the following message:

```
Beginning verification...Completed
```

Verification Report

If VERIFY found no bad sectors during the operation, it will also display the following message:

```
No bad sectors located.
```

If VERIFY found a reasonable number (1–169) of bad sectors during its search, it will display the following message:

```
Bad sectors located. Tables modified
```

NOTE: The words Tables modified will not appear in this message if VERIFY is unsuccessful in recording the new bad sector information at the end of the bad sector table.

If VERIFY finds more than 169 bad sectors on the Winchester disk, it will display the following message:

```
Bad sector count exceeded for this drive.
```

VERIFY Followup Activities

After you use the VERIFY utility, the data stored on your Winchester disk will still be intact (except the data that was recorded over bad sectors). However, the addition to the bad sector table that VERIFY provides will not be put to use until you use the FORMAT utility on the newly verified media.

The bad sectors that VERIFY found will not become inaccessible until FORMAT is used on the partition that contained the bad sectors. FORMAT will redefine the sector boundaries of the partition so that the bad sectors cannot be accessed.

Reset	Reset the computer after using VERIFY, (No partition will be accessible until you do.)
Boot up	If you have not already done so, boot up to a floppy disk or partition (other than the partition just verified) that contains the CP/M-86 Operating System.
ASSIGN	If necessary, assign the partition on which the error occurred to a drive.
BACKUP	Use the BACKUP utility to copy all of the files from the partition to floppy disks.

FORMAT	When all of the Winchester disk files have been safely backed up to floppy disk media, use the FORMAT utility. Specify the drive that has been assigned the partition on which the bad sector(s) occurred.
LDCOPY	Use the LDCOPY utility to copy a CP/M system to the newly formatted partition If you booted up with a floppy disk, use LDCOPY with file BOOT217. If you booted up with another partition, you can use LDCOPY without BOOT217.
RESTORE	After formatting this partition, use the RESTORE utility to copy the backed up files back to the Winchester disk partition.

You should take the earliest possible opportunity after verifying to perform these activities to insure the safety of your stored software and data.

VERIFY Error Messages

```
Bad sector count exceeded for this drive.
```

Cause: The upper bound limit of 169 bad sectors has been exceeded. This could indicate a hardware malfunction.

Cure: Run VERIFY again. If this error message appears after repeating VERIFY, then run PREP. If this error message appears after running PREP, then contact Zenith Data Systems Technical Consultation for assistance.

```
Error -- Can not read superblock A.
```

Cause: A bad sector error has occurred in the primary superblock (Superblock A).

Cure: This condition is self correcting. The backup or secondary superblock (Superblock B) will now be used. However, we advise that you now back up all of your files and then run PREP and PART again. If superblock B ever becomes unreadable, all data on the Winchester disk will become inaccessible.

```
Error -- Drive capacity > 32 megabytes!
```

Cause: VERIFY has calculated that the Winchester drive connected to the Z-217 controller card is larger than the maximum allowable size of 32 megabytes.

Cure: Run VERIFY being careful to respond with the correct values to the drive parameter questions.

```
Error -- Unable to read boot code from partition
```

Cause: The boot code on the spedified partition is either not present, or it has developed a bad sector.

Cure: Boot from another drive. Then run VERIFY, reset, reboot, and run ASSIGN, BACKUP, FORMAT and RESTORE on the partition where the error occurred. If that partition is totally unavailable, you may need to run the PREP utility.

```
Fatal Error -- Can not read superblock B.
```

Cause: A bad sector error has occurred in the backup copy of the superblock.

Cure: Run VERIFY again.

```
Invalid HEX value, Try again:
```

Cause: Value entered was not a valid hexadecimal number, or the value entered was outside of the possible range.

Cure: Double check the appropriate hex value, and re-enter.

```
Track 0 contains bad sector(s).
```

Cause: A bad sector error has occurred in the reserved area of the hard disk. This could indicate a hardware malfunction.

Cure: Run VERIFY again. If this error message appears after repeating VERIFY, then run PREP. If this error message appears after running PREP, then contact Zenith Data Systems Technical Consultation for assistance.

```
Unable to communicate with the Z217 controller
```

Cause: VERIFY can not locate the Z-217 controller. This could mean that the Z-217 is not firmly plugged into the S-100 bus, all of the drive cable connectors are not securely fastened, or that the controller has a hardware malfunction.

Cure: Check to see that the controller card and all cable connectors are secure, and run VERIFY again. If this error message appears after repeating VERIFY, then run PREP. If this error message appears after running PREP, then contact Zenith Data Systems Technical Consultation for assistance.

APPENDIX D

VERIFY

```
Unable to re-write tables, disk unusable!
```

Cause: VERIFY is unable to record data in the reserved Winchester area of your Winchester disk.

Cure: Run PREP. If this error message appears after running PREP, then contact Zenith Data Systems Technical Consultation for assistance:

```
    Z-217 controller error on Set Drive Parameters command
```

Cause: One or more of the drive characteristics specified in the reserved Winchester area is not valid for your Winchester disk drive.

Cure: Run PREP, making certain that you enter the proper drive characteristics for your Winchester disk drive. If this error message appears after running PREP, then contact Zenith Data Systems Technical Consultation for assistance.

INDEX

A
Aborting a LIST operation, 114
Allocation percentage of PART, A.17–A.22
Application programs
  CMD file extension of, 11
  performance of, 10
  requirements of, 11–12
  storage of, 11
  utilities of, 11
ASCII characters
  for LIST files, 114
  with PART, A.10, A.15
ASSIGN as a utility
  for available partitions, 30
  for current assignments, 30–31
  for drive names, 31
  error messages of, 33
  valid assignment commands of, 32
  for Winchester disk partitions, 29, 92
Automatic command specifications
    in CONFIGUR, 81–83

B
Backing up disks
  methods used for, 35–38
  for software preparation, 26, 27–28
BACKUP master volume of RESTORE, 136–137
BACKUP as a utility
  destination files of, 40–41
  error messages of, 61–65
  file structure of, 58–60, 141–144
  methods used to enter, 35–38
  operation of, 34–35
  options of, 41–49
  routines for preparation of, 53–58
  runtime prompting of, 49–53
  source files of, 39–40
Bad sectors. *See* VERIFY as a utility
Bad sector table, B.19–B.21
Baud rate
  as a modem characteristic, 78
  as a printer characteristic, 72–73
BIOS files
  to enable CP/M to operate, 10
  as a source code, 21
Bit shift on cylinders in Winchester drive, B.7
Bits per character
  as a modem characteristic, 79
  as a printer characteristic, 73
  Brief help screen, 117

C
Checksum as a checking code, A.37, B.19, B.21
Cold boot command line, 82
: (colon), as an entry, 25
Command editing keys
  BACK SPACE, 23
  CTRL-H, 23
  CTRL-R, 23
  CTRL-U, 23
  CTRL-X, 23
  DELETE, 23
Command entry and display keys
  CTRL-E, 24
  CTRL-J, 24
  CTRL-M, 24
  LINE FEED, 24
  RETURN, 24
Command execution keys
  CTRL-P, 25
  CTRL-S, 25
Command lines, entry of, 19, 22
Commands
  automatic, 81–83
  as programs, 18
  resident, 20
  in response to system prompt, 19
  transient, 20–21
CONFIGUR as a utility
  for automatic command specification, 81–83
  to customize CP/M for hardware, 66
  error messages of, 90
  exiting from, 87–90
  for I/O device specification, 84–87
  main menu of, 67
  for modem specifications, 66, 77–81
  for printer specifications, 66, 68–77
  to start up, 67
CP/M
  backup procedure for, 27–28
  beginning concepts of, 1, 12–25
  file name extensions, 12–13
  with one disk drive, 18
  startup procedure for, 26–27
Cursor movement keys
  CTRL-I, 25
  TAB, 25
Cylinders of Winchester disks, B.4

INDEX

D
Date prompt in backing up disks, 49–50
Default boot partition number, A.22–A.26, A.27
Default drive, 15–16
Defaults in formatting, 97–99
Destination files
  of BACKUP, 40–41
  of RESTORE, 125–126
Detailed help screen, 117–118
Dialog, user/computer, 1–2
Disk characteristic prompts, B.6–B.9
Disk insertion prompt in backing up disks, 50–51
Disks
  capacities of, 100–101
  formatting of, 91, 92–95
  precautions in handling, 4–5
  preparation of, 4
  write-enabling, 5–7
  write-protecting, 5–7
Displays, screen
  typestyle of, 1–2
  variance of characters, 2
Drives, 15–18

E
8-bit software, 21–22
8-inch disks, formatting of, 94–95
Error messages
  of ASSIGN, 33
  of BACKUP, 61–65
  of CONFIGUR, 90
  of FORMAT, 101–103
  of LDCOPY, 109
  of LIST, 114
  of PART, A.38–A.40
  of PREP, B.26–B.28
  of RDDOS, 118–119
  of RESTORE, 144–147
  of SHIP, C.5
  of VERIFY, D.8–D.10

F
Files
  for data storage, 12
  naming of, 12–15
5.25-inch disks, formatting of, 93
Flag byte, A.36
Followup activities of VERIFY, D.6–D.7
FORMAT as a utility
  disk capacities of, 100–101
  error messages of, 101–103
  methods used to enter, 91–100
  with VERIFY, D.2, D.3, D.6–D.7
Format enable location on Winchester
    disk controller board, B.1, B.12

G
General purpose routine
  for BACKUP, 54–55
  for RESTORE, 138–139

H
Handshake pin
  for modem specification, 80–81
  for printer specification, 75
Hardware
  connection of, 89–90
  with PREP, adjustment of, B.22–B.25
Help display method
  in BACKUP, 38
  in RESTORE, 124
Help screen
  for LDCOPY, 108–109
  for RDDOS, 117–118

I
I/O device specification, 84–87

J
Jumper as a hardware component,
    B.1, B.12, B.22–B.25

K
Keys
  command editing, 23
  command entry and display, 24
  command execution, 25
  cursor movement, 25
  to enter comments, 24–25

L
LDCOPY as a utility
  error messages of, 109
  help screen of, 108
  methods used to enter, 104–108
LIST as a utility
  to abort an operation, 114
  with ASCII characters, 114
  error messages of, 114
  methods used to enter, 110–111
  parameters of, 112–113
  to print several files, 111
Logical/physical device categories, 84–86

INDEX

M
Master disk insertion prompt in backing
  up disks, 51–53
Microcomputer concepts, 1, 3–12
Modem specifications with CONFIGUR, 77–81

N
Non-default drives, 17

O
Operating system
  BIOS files, 10
  CP/M, 9, 10, 11, 12
  requirements of, 10
  system kernel, 9
Options
  of BACKUP, 41–49
  of FORMAT, 97
  of RESTORE, 127–134

P
Parameters of LIST files, 112–113
Parity
  as a modem characteristic, 79–80
  as a printer characteristic, 74
PART as a utility, 149
  error messages of, A.38–A.40
  features of, A.1–A.2
  operation of, A.3–A.32
  with superblock, A.33–A.37
Partitions
  formatting of, 91, 92, 93, 95
  as a logical division, B.4
  with the PART utility, A.1–A.40
Physical device names, assignment of, 86–87
Platters of the Winchester disk, B.2
Polarity in printer specifications, 75–76
Pre-comp as a disk characteristic
  prompt, B.7, B.9
PREP as a utility
  error messages of, B.26–B.28
  features of, B.2–B.4
  hardware adjustments of, B.22–B.25
  to initialize Winchester disks, 149, B.1
  operation of, B.10–B.12
  with reserved Winchester area, B.13–B.21
  responding to disk characteristic
    prompts of, B.6–B.9
  starting up of, B.5
Printer configuration
  to adjust the CP/M system for, 68
  to specify printer characteristics, 72–77
  to specify printer name, 68–71
Printing several files with LIST, 111
Protocol as a printer specification, 76–77

R
RDDOS as a utility
  to copy files from Z-DOS to CP/M, 116–117
  error messages of, 118–119
  help screens of, 117–118
  to read Z-DOS directory, 115–116
Read/write heads of Winchester disk platters,
  B.2, B.3, B.6–B.7, C.1
Reserved Winchester area
  with PART, A.33
  with PREP, B.12
Resident commands, 20
RESTORE as a utility
  backup files of, 136–137, 141–144
  destinations of, 125–126
  error messages of, 144–147
  methods used to enter, 121–124
  operation of, 120–121
  options of, 127–134
  routines of, 137–141
  runtime prompting of, 134–136
  to specify a source file, 124–125
Runtime prompting
  during BACKUP operation, 49–53
  during RESTORE operation, 134–136

S
SBC. *See* Software boot code as a
  Winchester support software unit
Sectors of Winchester disks, B.4
; (semicolon), as an entry, 24
SHIP as a utility
  shipping cylinder prompt of, C.3–C.4
  error messages of, C.5
  startup of, C.2
  for Winchester disk safety, 149, C.1–C.2
Software boot code as a Winchester support
  software unit, B.14–B.17
Software preparation procedures

INDEX

for backup of CP/M, 26, 27–28
for startup of CP/M, 26–27
Source files
in BACKUP, 39–40
in RESTORE, 124–125
Step rate code as a disk characteristic prompt, B.3, B.7
SUBMIT utility,
with BACKUP, 53–54
with RESTORE, 137–138
Superblock
with the PART utility, A.33–A.37
with the PREP utility, B.17–B.19
SuperCalc
BACKUP routine, 55–56
RESTORE routine, 139–140
Syntax notation, 1–2
System kernel, 9
System prompt method,
with BACKUP, 37–38
with FORMAT, 96–100
with LDCOPY, 107–108
with LIST, 110–111
with RESTORE, 123
System prompt to receive a command, 19

T
Tracks of Winchester disk platter, B.4
Transient commands, 20–21

U
Utilities. *See* Transient commands
Utility prompt method
with BACKUP, 35–37
with FORMAT, 91–96
with LDCOPY, 104–106
with LIST, 110
with RESTORE, 121–122

V
VERIFY as a utility
entries of, D.3–D.5
error messages of, D.8–D.10
followup activities of, D.6–D.7
to isolate bad sectors, 149, D.1–D.2
operation of, D.5

W
Warm boot command line, 82–83
Wild card file name, 14–15
Winchester disks
assignment of, 29–32
formatting of, 95
with PART, A.1–A.40
with PREP, B.1–B.28
with SHIP, C.1–C.2
WordStar
BACKUP routine, 56–58
RESTORE routine, 140–141

Z
Z-DOS
to read a file directory from, 115–116
to copy files from, 116–117

DIGITAL RESEARCH®

CP/M-86®

User's Guide

595-3073

Printed in the United States of America

ZENITH data systems

HEATH

DIGITAL RESEARCH™

CP/M-86
Operating System
User's Guide

595-3073

The *CP/M-86 Operating System User's Guide* was prepared using the Digital Research TEX Text Formatter and printed in the United States of America by Commercial Press/ Monterey.

Second Printing: July 1982

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

1 **Introduction**

1.1 How to Get CP/M-86 Started 1
1.2 The Command Line 2
1.3 CP/M Line Editing Control Characters 4
1.4 Why You Should Back Up Your Files 5
1.5 How to Make a Copy of Your CP/M-86 Disk 6

2 **Files, Disks, Drives and Devices**

2.1 What is a File? 9
2.2 How Are Files Created? 9
2.3 Naming Files—What's in a Name? 10
2.4 Accessing Files—Do You Have the Correct Drive? 11
2.5 Accessing More Than One File 12
2.6 How Can I Organize and Protect My Files? 13
2.7 How Are Files Stored on a Disk? 15
2.8 Changing Disks 15
2.9 Changing the Default Drive 16
2.10 More CP/M-86 Drive Features 16
2.11 Other CP/M-86 Devices 17

3 **CP/M-86 Command Concepts**

3.1 Two Types of Commands 19
3.2 Built-in Commands 19
3.3 Transient Utility Commands 20
3.4 How CP/M-86 Searches for Commands 21
3.5 Control Character Commands 23

4 **Command Summary**

4.1 Let's Get Past the Formalities 25
4.2 How Commands Are Described 26
4.3 The ASM-86 (Assembler) Command 30
4.4 The COPYDISK (Copy Disk) Command 32
4.5 The DDT-86 (Dynamic Debugging Tool) Command 34
4.6 The DIR (Directory) Built-in 36
4.7 The ED (Character File Editor) Command 39
4.8 The ERA (Erase) Built-in 45

Table of Contents (continued)

4.9 The GENCMD (Generate CMD File) Command 46
4.10 The HELP (Help) Command 48
4.11 PIP (Peripheral Interchange Program—Copy File) Command 50
 4.11.1 Single File Copy 50
 4.11.2 Multiple File Copy 53
 4.11.3 Combining Files 54
 4.11.4 Copy Files to and from Auxiliary Devices 55
 4.11.5 Multiple Command Mode 57
 4.11.6 Using Options With PIP 58
4.12 The REN (Rename) Built-in 62
4.13 The STAT (Status) Command 63
 4.13.1 Set a Drive to Read-Only Status 64
 4.13.2 Free Space on Disk 65
 4.13.3 Files—Display Space Used and Access Mode 66
 4.13.4 Set File Access Modes (Attributes) 69
 4.13.5 Display Disk Status 70
 4.13.6 Display User Numbers With Active Files 71
 4.13.7 Display STAT Commands and Device Names 72
 4.13.8 Display and Set Physical to Logical Device Assignments ... 72
4.14 The SUBMIT (Batch Processing) Command 73
4.15 The TOD (Display and Set Time of Day) Command 76
4.16 The TYPE (Display File) Built-in 78
4.17 The USER (Display and Set User Number) Built-in 79

5 ED, The CP/M-86 Editor

5.1 Introduction to ED 81
5.2 Starting ED 81
5.3 ED Operation 82
 5.3.1 Appending Text into the Buffer 84
 5.3.2 ED Exit 85
5.4 Basic Editing Commands 86
 5.4.1 Moving the Character Pointer 88
 5.4.2 Displaying Memory Buffer Contents 90
 5.4.3 Deleting Characters 91
 5.4.4 Inserting Characters into the Memory Buffer 92
 5.4.5 Replacing Characters 94

Table of Contents (continued)

5.5 Combining ED Commands 95
5.5.1 Moving the Character Pointer 96
5.5.2 Displaying Text 96
5.5.3 Editing 97
5.6 Advanced ED Commands 98
5.6.1 Moving the CP and Displaying Text 98
5.6.2 Finding and Replacing Character Strings 99
5.6.3 Moving Text Blocks 103
5.6.4 Saving or Abandoning Changes: ED Exit 105
5.7 ED Error Messages 107

Appendixes

A ASCII and Hexadecimal Conversions 111

B CP/M-86 File Types 117

C CP/M-86 Control Characters 119

D CP/M-86 Error Messages 121

E User's Glossary 133

List of Tables

1-1 Control Character Commands 4

2-1 CP/M-86 Filetypes 11
2-2 CP/M-86 Logical Devices 17

3-1 Built-In Commands 19
3-2 CP/M-86 Utilities 20
3-3 Control Character Commands 23

Table of Contents (continued)

4-1 DDT-86 Commands 34
4-2 ED Command Summary 39
4-3 PIP Options 58

5-1 Text Transfer Commands 84
5-2 Basic Editing Commands 87
5-3 CP/M-86 Line Editing Controls 93
5-4 ED Error Symbols 107
5-5 ED Disk File Error Messages 109

A-1 ASCII Symbols 111
A-2 ASCII Conversion Table 112

B-1 Filetypes 117

C-1 CP/M-86 Control Characters 119

D-1 CP/M-86 Command Messages 121

List of Figures

5-1 Overall ED Operation 83

Section 1
Introduction to CP/M-86

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.

After you have turned on the power, insert your CP/M-86 system disk into drive A, generally a built-in drive on the right side of the computer unit. Close the drive door. Press the RESET or RESTART button. 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 then press the RESET or RESTART button. This is called System Reset, 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:

```
CP/M-86         Version V.V
Copyright (c) 1981 Digital Research Inc.
```

The version number, represented above by V.V, tells you the major and minor revision level of the CP/M-86 version that you own. 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.

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 microprocessor. 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 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 Backspace key if your terminal has one, or the CTRL-H characters if it does not, to move the cursor to the left and correct the error.

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

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

A command keyword identifies one of two different types of commands: Built-in commands and Transient Utility commands. Built-in commands reside in memory as a part of CP/M-86 and can be executed immediately. Transient Utility commands are stored on 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 RETURN.

```
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 RETURN 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 Backspace 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 (sometimes labeled 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 - the same as the Backspace key.
CTRL-I	moves the cursor to the next tab stop, where tab stops are automatically placed at each eighth column - same as the TAB 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 RETURN 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 RETURN keystroke.
CTRL-R	types a # at the current cursor location, moves the cursor to the next line and retypes any partial command you have typed so far.

Table 1-1. (continued)

Command	*Meaning*
CTRL-U	discards all the characters in the command line that you've typed so far, types a # at the current cursor position and moves the cursor to the next command line.
CTRL-X	discards all the characters in the command line that you've typed so far and moves the cursor back to the beginning of the current line.

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

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 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.

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 eight-inch 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.

You might want to format new or reformat used disks with the disk formatting program that should accompany your particular computer. 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.

End of Section 1

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
BAS	CBASIC Source 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 precede 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.

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.

End of Section 2

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.

Six Built-in commands and sixteen Transient Utilities are included with CP/M-86. 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.

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.

Table 3-1. (continued)

Command	*Meaning*
ERA	erases a filename from a disk directory and releases the storage occupied by the file.
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.

Table 3-2. (continued)

Command	*Meaning*
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.

End of Section 3

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 represents 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.

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.

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.

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

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

A>
```

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 8086 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,s1	(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}{,b1{,b2}}	(Go)	Begin Execution
Hwc1,wc2	(Hex)	Hexadecimal Sum and Difference

Table 4-1 (continued)

Command		*Meaning*
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
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 Operating System 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

Table 4-2. (continued)

Command	*Action*
#A	append file until buffer is full (or end of file)
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

Table 4-2. (continued)

Command	*Action*
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
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

Table 4-2. (continued)

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

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

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

```
*a
```

reads the entire file to the Buffer.

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

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

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

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

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

nnnnn:

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

```
:
```

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

Examples:

```
A>ED MYPROG.A86
```

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

```
:*
```

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

```
:*#a
```

to fill the Buffer. Then type the command

```
:*0p
```

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

```
:*e
```

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

```
A>ED MYPROG.A86 B:NEWPROG.A86
```

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

```
A>B:ED MYPROG.A86 B:
```

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

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 MYFILE
```

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

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

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

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.

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 to 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 RETURN 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.
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.

Table 4-3. (continued)

Option	*Function*
Gn	Get source from or Go to user number n. When this parameter follows a source name, PIP searches the directory of user number n for the source file. When it follows the destination name, PIP places the destination file in the user number specified by n. The number must be in the range 0 to 15.
H	Hex data transfer. PIP checks all data for proper Intel hexadecimal file format. The console displays error messages when errors occur.
I	Ignore :00 records in the transfer of Intel hexadecimal format file. The I option automatically sets the H option.
L	Translate upper-case alphabetics in the source file to lower-case in the destination file. This parameter follows the source device or filename.
N	Add line numbers to the destination file. When this parameter follows the source filename, PIP adds a line number to each line copied, starting with 1 and incrementing by one. A colon follows the line number. If N2 is specified, PIP adds leading zeroes to the line number and inserts a tab after the number. If the T parameter is also set, PIP expands the tab.
O	Object file transfer for machine code (non-character and therefore non-printable) files. PIP ignores any CTRL-Z ends-of-file during concatenation and transfer. Use this option if you are combining object code files.
Pn	Set page length. n specifies the number of lines per page. When this parameter modifies a source file, PIP includes a page eject at the beginning of the destination file and at every n lines. If n = 1 or is not specified, PIP inserts page ejects every 60 lines. When you also specify the F option, PIP ignores form-feeds in the source data and inserts new form-feeds in the destination data at the page length specified by n.

Table 4-3. (continued)

Options	*Function*
Qs	Quit copying from the source device after the string s. When used with the S parameter, this parameter can extract a portion of a source file. The string argument must be terminated by CTRL-Z.
R	Read system (SYS) files. Normally, PIP ignores files marked with the system attribute in the disk directory. But when this parameter follows a source filename, PIP copies system files, including their attributes, to the destination.
Ss	Start copying from the source device at the string s. The string argument must be terminated by CTRL-Z. When used with the Q parameter, this parameter can extract a portion of a source file. Both start and quit strings are included in the destination file.
Tn	Expand tabs. When this parameter follows a source filename, PIP expands tab (CTRL-I) characters in the destination file. PIP replaces each CTRL-I with enough spaces to position the next character in a column divisible by n.
U	Translate lower-case alphabetic characters in the source file to upper-case in the destination file. This parameter follows the source device or filename.
V	Verify that data has been copied correctly. PIP compares the destination to the source data to ensure that the data has been written correctly. The destination must be a disk file.
W	Write over files with 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.

Table 4-3. (continued)

Options	*Function*
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 preceded 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 preceded 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 preceded 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:, RDR:, PUN: 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.

Examples:

```
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:

weekday month/day/year hour:minute:second

For example, the screen might appear as:

```
12/06/81          09:15:37
```

in response to the TOD command.

Use the command form:

TOD time-specification

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

month/day/year hour:minute:second

A command line in this form is:

```
TOD 02/09/81 10:30:00
```

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

```
Press any Key to set time
```

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

```
02/09/81          10:30:00
```

Use the command form:

```
    TOD P
```

to continuously print the date and time on the screen. 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.

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.

Use 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.

End of Section 4

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.

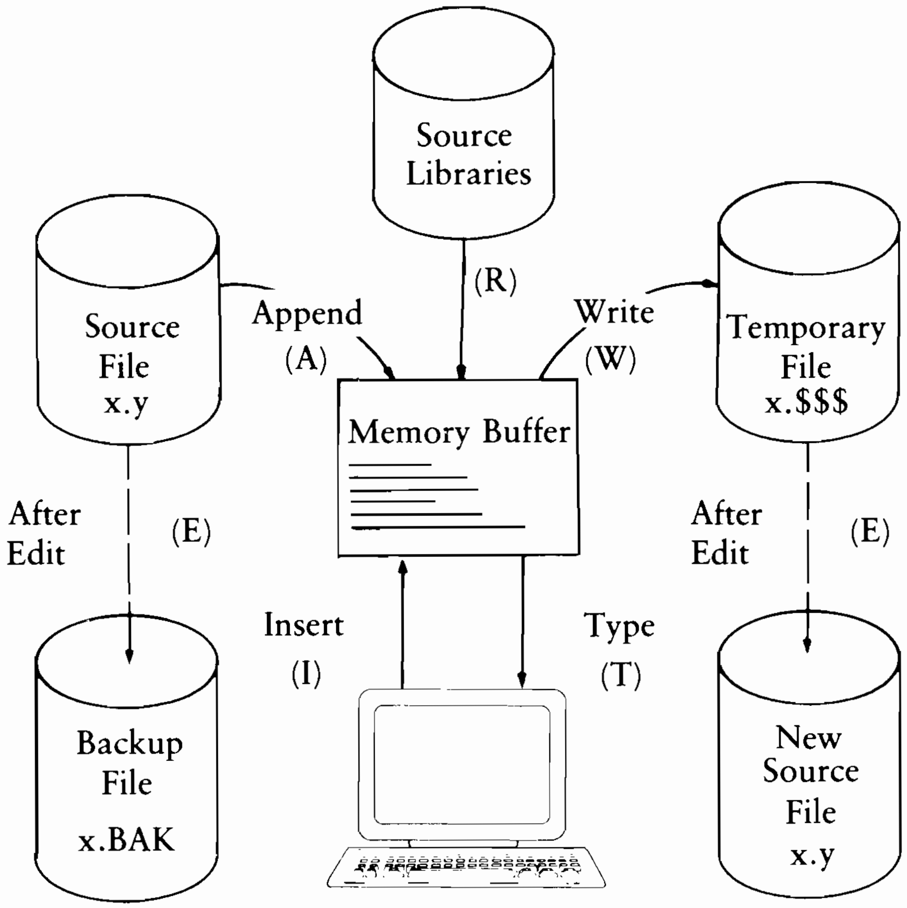


Figure 5-1. Overall ED Operation

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, then the integer 65535 is assumed. Because the memory buffer can contain most reasonably sized source files, it is often possible to issue the command #A at the beginning of the edit to read the entire source file into memory.

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 be executed backwards through the memory buffer. (The B command is an exception).
- If you enter a pound sign, #, in place of a number, ED uses the value 65535 as the argument. A pound sign argument can be preceded by a minus sign to cause the command to execute backwards through the memory buffer (−#).
- ED accepts 0 as a numeric argument only in certain commands. In some cases, 0 causes the command to be executed approximately half the possible number of times, while in other cases it prevents the movement of the CP.

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

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.
Rubout	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

The S (Substitute) Command

The S command searches the memory buffer for the specified string, but when it finds it, automatically substitutes a new string for the search string. 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: *2OCi to a friend^ZOLT
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-GdiLiving^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 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	Unrecognized command letter c. ED does not recognize the indicated command letter, or an E, H, Q, or O command is not alone on its command line.
0	No .LIB file. ED did not find the .LIB file specified in an R command.

Table 5-4. (continued)

Symbol	*Meaning*
>	Buffer full. ED cannot put any more characters in the memory buffer, or string specified in an F, N, or S command is too long.
E	Command aborted. A keystroke at the keyboard aborted command execution.
F	File error. Followed by either DISK FULL or DIRECTORY FULL.

The following examples show how to recover from common editing error conditions. For example:

```
BREAK ">" AT A
```

means that ED filled the memory buffer before completing the execution of an A command. When this occurs, the character pointer is at the end of the buffer and no editing is possible. Use the 0W command to write out half the buffer or use an O or H command and reedit the file.

```
BREAK "#" AT F
```

means that ED reached the end of the memory buffer without matching the string in an F command. At this point, the character pointer is at the end of the buffer. Move the CP with a B or n: line number command to resume editing

```
BREAK "F" AT F
DISK FULL
```

Use the 0X command to erase an unnecessary file on the disk or a B#Xd:buffer.sav command to write the contents of the memory buffer onto another disk.

```
BREAK "F" AT n
DIRECTORY FULL
```

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

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

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.

End of Section 5

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	%

Table A-2. (continued)

Binary	*Decimal*	*Hexadecimal*	*ASCII*
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
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

Table A-2. (continued)

Binary	*Decimal*	*Hexadecimal*	*ASCII*
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
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

Table A-2. (continued)

Binary	*Decimal*	*Hexadecimal*	*ASCII*
1110010	114	72	r
1110011	115	73	s
1110100	116	74	t
1110101	117	75	u
1110110	118	76	v
1110111	119	77	w
1111000	120	78	x
1111001	121	79	y
1111010	122	7A	z
1111011	123	7B	{
1111100	124	7C	|
1111101	125	7D	}
1111110	126	7E	~
1111111	127	7F	DEL

End of Appendix A

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.
BAS	CBASIC-86 program source file.
CMD	Command file that contains instructions in machine executable code.
COM	8080 executable file.
H86	Program file in hexadecimal format.
INT	CBASIC-86 program intermediate language file.
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.

Table B-1. (continued)

Filetype	*Indication*
SYM	Symbol table file.
$$$	Temporary file created by PIP.

End of Appendix B

Appendix C
CP/M-86 Control Characters

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

Keystroke	*Action*
BACKSPACE	moves cursor back one space, erases previous character.
CTRL-C	prompts to abort a program currently running at a given console.
DEL	same as RUB.
CTRL-E	forces a physical carriage return, but does not send command to CP/M-86.
CTRL-H	same as BACKSPACE.
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.
RUB	deletes character to the left of cursor; echoes character deleted - cursor moves right.

Table C-1. (continued)

Keystroke	*Action*
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.

End of Appendix C

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 contains 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.
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.

Table D-1 (continued)

Message	*Meaning*
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.
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.

Table D-1 (continued)

Message	*Meaning*
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. 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`

Table D-1 (continued)

Message	*Meaning*

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.
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.

Table D-1 (continued)

Message	*Meaning*
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.
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.
ERROR: INVALID USER NUMBER	PIP. You have specified a User Number greater than 15. User Numbers are in the range 0 to 15.

Table D-1 (continued)

Message	*Meaning*
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.
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.

Table D-1 (continued)

Message	*Meaning*
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`
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.

Table D-1 (continued)

Message	*Meaning*
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.
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:, RDR:, PUN: 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.

Table D-1 (continued)

Message	*Meaning*
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.
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`

Table D-1 (continued)

Message	*Meaning*
Nested "IF" illegal - "IF" ignored	ASM-86. The maximum nesting level for IF statements has been exceeded.
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`
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.

Table D-1 (continued)

Message	*Meaning*
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.

End of Appendix D

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 CTRL-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 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. A 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.

End of Appendix E

Index

A

A (append) command, 84
abort a copy operation, 55
access modes, 66-67, 69-70
active status, 67
active user numbers, 71
ASM-86 (assembler) command, 30
assign physical device to logical device, 73
attributes, 13, 67
AXI:, 17, 55
AXO:, 17, 55

B

B (beginning/bottom) command, 88
back-up disks, 6
basic editing commands, 86
batch command processing, 75
built-in commands, 2, 19
bytes, 67

C

C (character) command, 88, 96
character pointer, 86
character transfer, 54
CMD, 11
combined-command line, 96
combining files, 54
command description, 26-27
command files, 14
command keyword, 2, 19, 26
command line, 2, 19
command mode, 93
command tail, 2
CON:, 17, 55
concatenation, 54-55
continuous date and time display, 78
control character commands, 4-5, 23
copy, 6
copy files to and from auxiliary devices, 55
COPYDISK (copy disk) command, 6-7, 32-33, 53
CP, 86
CTRL-E, 102
CTRL-Z, 54, 56, 93, 102
current user number, 13, 79
cursor, 2

D

D (delete) command, 91
data files, 9
date and time format, 77
DDT-86 commands, 34
DDT-86 parameters, 34-35
default drive, 16
default user number, 53
delimiters, 10
dest-filespec, 29
DEV:, 72
device names, 56
DIR (directory) command, 3, 14, 37, 67, 69
DIR/SYS attribute, 14
directory, 9, 15
directory attribute, 67
directory entries, 67

directory space, 15
directory verification, 68
DIRS command, 14, 37
disk free space, 65
disk space allocation, 68
displaying disk status, 70
Dn (delete characters) option, 58
drive, 15
drive specifier, 11-12, 25

E

E (echo) option, 52, 58
E (exit) command, 86
ED command, 9, 39
ED command summary, 39-42
ED error symbols, 107-108
ED messages, 82, 100, 103, 107
ED prompt, 82, 93
ED syntax, 81
editing, 97
editors, 81
end-of-file character, 54
EOF:, 56
ERA command, 45
error message, BDOS, 15

F

F (filter form-feeds) option, 58
F (find) command, 100
FCB, 67
file, 1, 9
file attributes, 14, 51
file concatenation, 54
file families, 10, 11
file groups, 13
file specification, 10, 25, 26, 67
filename, 10, 25
filespec, 10
filetype, 10-11, 25

G

GENCMD command, 46
Gn option, 50-51, 54, 58, 59

H

H (head of file) command, 85, 101, 105
H (hex data transfer) option, 59
HELP (help) command, 48

I

I (ignore) option, 59
I (insert) command, 92
initial date and time, 77
input devices, 17
input-output port, 73
insert mode, 92
Istring ^Z (insert string) command, 94

J

J (juxtapose) command, 101

K

K (kill) command, 91
keyword error, 22
kilobytes, 67

L

L (line) command, 89, 96
L (translate case) option, 59
line editing controls, 93
line numbers, 84
loading CP/M-86, 1
logical devices, 17, 55, 72
long form of PIP, 52
LST:, 17, 56

M

M (macro) command, 103
memory buffer, 82-83
multiple command mode, 57
multiple file copy, 53

N

N (add line numbers) option, 59
n (number) command, 89
N command, 100
n: (line number) command, 98
NUL:, 56
numeric arguent, 86

O

O (object file transfer) option, 54, 59
O (original) command, 106
on-line disk, 15
on-line status, 65, 70
output devices, 17

P

P (page) command, 98
peripheral devices, 17
physical device drivers, 73
physical device names, 72
PIP, 9, 13, 29, 50
PIP messages, 53
PIP options, 51, 58
Pn (set page length) option, 59
pound sign argument, 84
PRN:, 56
program, 1
program file, 9

Q

Q (quit) command, 106
Qs (quit copying) option, 60

R

R (read system files) option, 60
R (read) command, 104
read-only, 64
read-only attribute, 66
read-only files, 51
read-write, 64
read-write attribute, 66
ready status, 15
real file size, 67
records, 67
recovering from editing error
    conditions, 108
recs, 67

reformat, 6
REN command, 62
REN messages, 63
repeated execution of ED commands, 104
reset, 15
RO, 14, 16, 64, 66, 69
RW, 14, 16, 64, 66, 69
RW/RO file attribute, 14

S

S (substitute) command, 94
set a drive to read-only status, 64
set file access modes, 69
set time of day, 76
short form of PIP, 52
single file copy, 50
SIZE, 67
source file, 83
square brackets, 51, 58
Ss (start copying) option, 60
STAT command, 14, 17, 63
STAT messages, 68
SUB file, 74
SUBMIT command, 74
SUBMIT parameters, 74
syntax notation, 28
SYS, 67, 70
SYS attribute, 14
SYS files, 14
system attribute, 67
system prompt, 2, 16
system reset, 1

T

T (type) command, 90, 96-97
tab characters, 78
temporary file, 51, 83
text editor, 9
text transfer commands, 84
time, 77
Tn (expand tabs) option, 60
TOD command, 76
TOD messages, 77
transient utilities, 3, 19, 20
TYPE command, 79
TYPE messages, 78

U

U (translate case) option, 60
U (upper-case translation) command, 94
USER, 13
USER command, 53, 79
user memory, 36
user numbers, 13, 71, 80
utility, 19

V

V (verify) option, 52, 60
VAL:, 72
version number, 2
virtual file size, 67

W

W (write over) option, 51, 60
W (write) command, 85
wildcard characters, 12, 29, 66
write-protect, 15
write-protect notch, 6

X

X (xfer) command, 104
X$$$$$$$.LIB file, 104

Z

Z (zero the parity bit) option, 61

$

$$$ file, 51

:

:n command, 99

DIGITAL RESEARCH®

CP/M-86®

System Guide

595-3119

Printed in the United States of America

ZENITH | data systems

HEATH

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

The "CP/M-86 System Guide" was prepared using the Digital Research TEX-80™ Text Formatter and printed in the United States of America

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

1 CP/M-86 System Overview

1.1 CP/M-86 General Characteristics 1
1.2 CP/M-80 and CP/M-86 Differences 3

2 Command Setup and Execution Under CP/M-86

2.1 CCP Built-in and Transient Commands 7
2.2 Transient Program Execution Models 8
2.3 The 8080 Memory Model 9
2.4 The Small Memory Model 10
2.5 The Compact Memory Model 11
2.6 Base Page Initialization 13
2.7 Transient Program Load and Exit 14

3 Command (CMD) File Generation

3.1 Intel Hex File Format 15
3.2 Operation of GENCMD 16
3.3 Operation of LMCMD 19
3.4 Command (CMD) File Format 20

4 Basic Disk Operating System (BDOS) Functions

4.1 BDOS Parameters and Function Codes 23
4.2 Simple BDOS Calls 25
4.3 BDOS File Operations 30
4.4 BDOS Memory Management and Load 48

5 Basic I/O System (BIOS) Organization

5.1 Organization of the BIOS 55
5.2 The BIOS Jump Vector 56
5.3 Simple Peripheral Devices 57
5.4 BIOS Subroutine Entry Points 60

6 BIOS Disk Definition Tables

6.1 Disk Parameter Table Format 67
6.2 Table Generation Using GENDEF 72
6.3 GENDEF Output 77

7 CP/M-86 Bootstrap and Adaptation Procedures

7.1 The Cold Start Load Operation 81
7.2 Organization of CPM.SYS 84

Appendixes

A Blocking and Deblocking Algorithms 87

B Random Access Sample Program 95

C Listing of the Boot Rom 103

D LDBIOS Listing 113

E BIOS Listing 121

F CBIOS Listing 137

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:

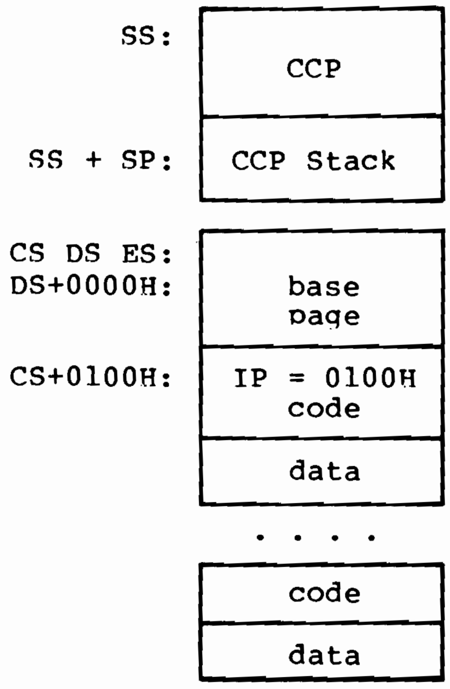


Figure 2-1. CP/M-86 8080 Memory Model

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.

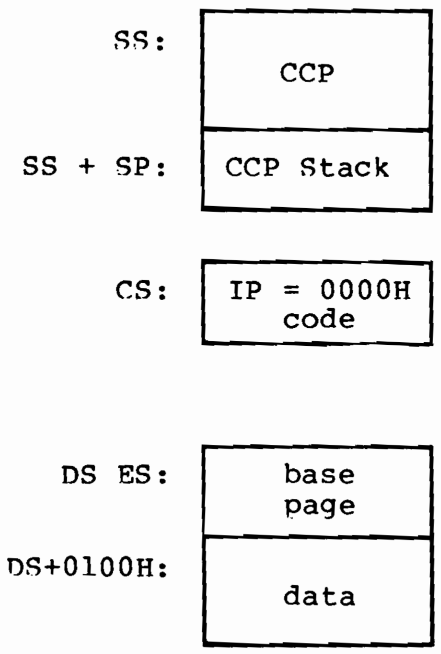


Figure 2-2. CP/M-86 Small Memory Model

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
```

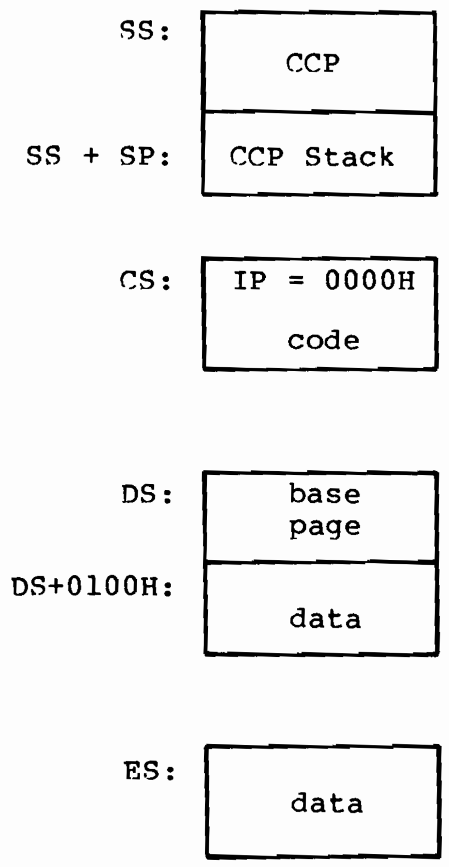


Figure 2-3. CP/M-86 Compact Memory Model

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] x1[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.

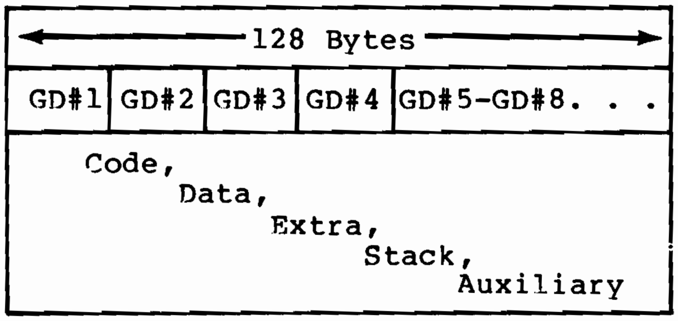


Figure 3-1. CMD File Header Format

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.

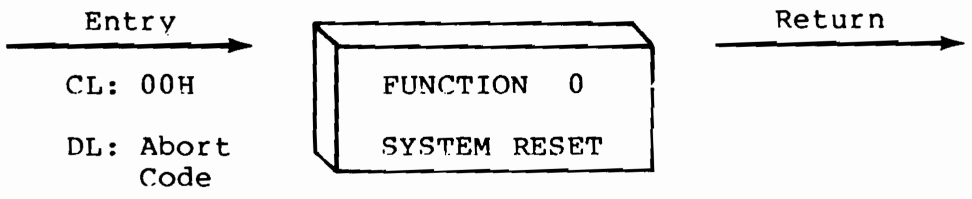
Table 4-2. CP/M-86 BDOS Functions

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

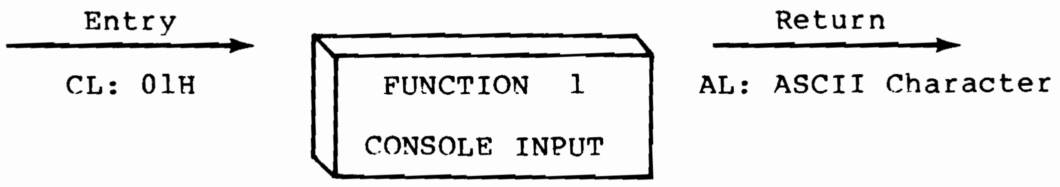
The individual BDOS functions are described below in three sections which cover the simple functions, file operations, and extended operations for memory management and program loading.

4.2 Simple BDOS Calls

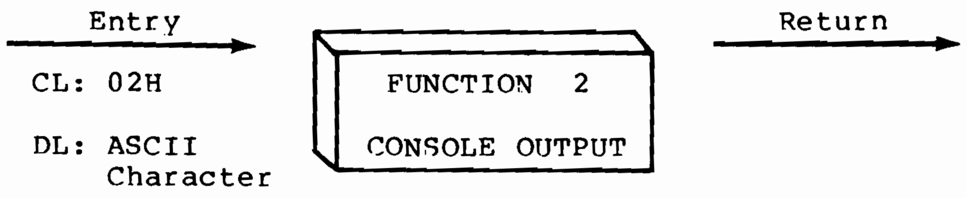
The first set of BDOS functions cover the range 0 through 12, and perform simple functions such as system reset and single character I/O.



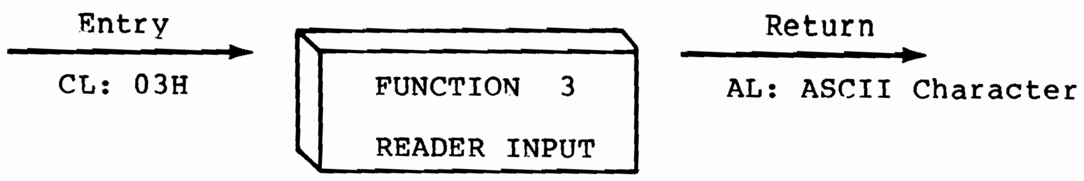
The system reset function returns control to the CP/M operating system at the CCP command level. The abort code in DL has two possible values: if DL = 00H then the currently active program is terminated and control is returned to the CCP. If DL is a 01H, the program remains in memory and the memory allocation state remains unchanged.



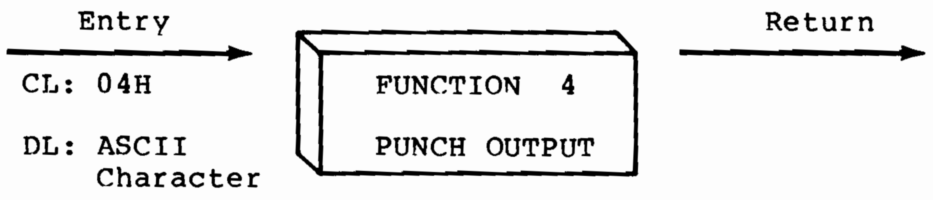
The console input function reads the next character from the logical console device (CONSOLE) to register AL. Graphic characters, along with carriage return, line feed, and backspace (CONTROL-H) are echoed to the console. Tab characters (CONTROL-I) are expanded in columns of eight characters. The BDOS does not return to the calling program until a character has been typed, thus suspending execution if a character is not ready.



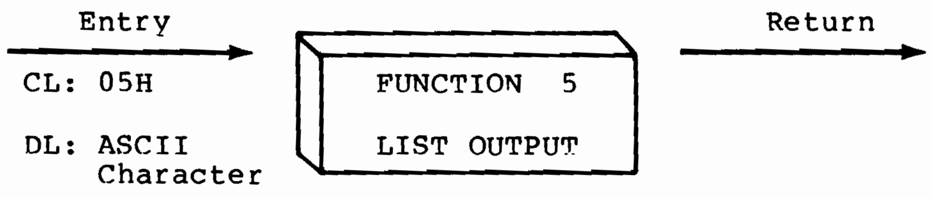
The ASCII character from DL is sent to the logical console. Tab characters expand in columns of eight characters. In addition, a check is made for start/stop scroll (CONTROL-S).



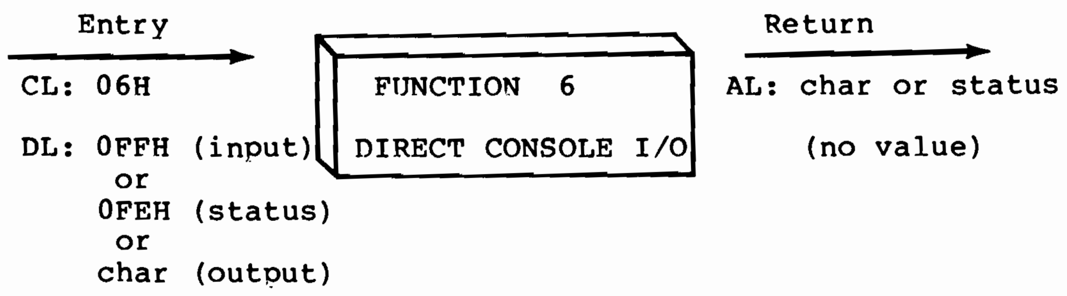
The Reader Input function reads the next character from the logical reader (READER) into register AL. Control does not return until the character has been read.



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

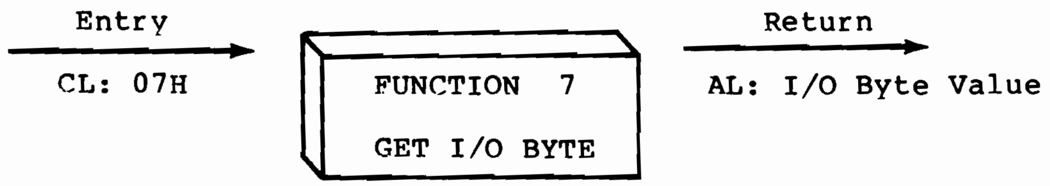


The List Output function sends the ASCII character in register DL to the logical list device (LIST).

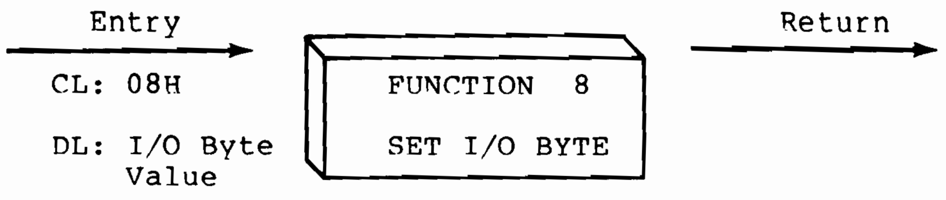


Direct console I/O is supported under CP/M-86 for those specialized applications where unadorned console input and output is required. Use of this function should, in general, be avoided since it bypasses all of CP/M-86's normal control character functions (e.g., CONTROL-S and CONTROL-P). Programs which 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.

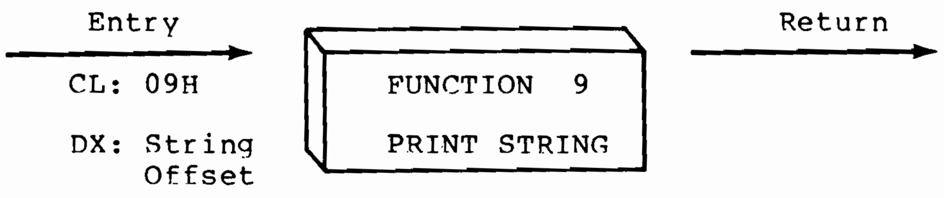
Upon entry to function 6, register DL either contains (1) a hexadecimal FF, denoting a CONSOLE input request, or (2) a hexadecimal FE, denoting a CONSOLE status request, or (3) an ASCII character to be output to CONSOLE where CONSOLE is the logical console device. If the input value is FF, then function 6 directly calls the BIOS console input primitive. The next console input character is returned in AL. If the input value is FE, then function 6 returns AL = 00 if no character is ready and AL = FF otherwise. If the input value in DL is not FE or FF, then function 6 assumes that DL contains a valid ASCII character which is sent to the console.



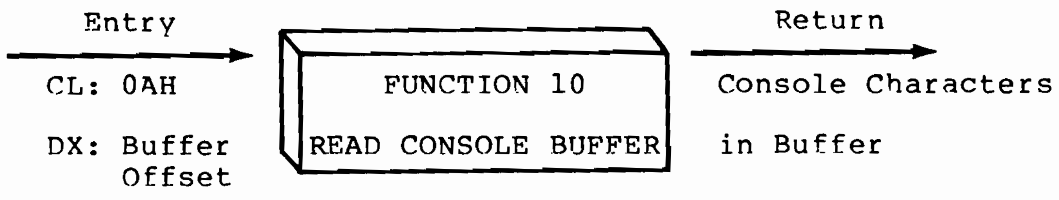
The Get I/O Byte function returns the current value of IOBYTE in register AL. The IOBYTE contains the current assignments for the logical devices CONSOLE, READER, PUNCH, and LIST provided the IOBYTE facility is implemented in the BIOS.



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.



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.



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|     . . .     |??|
   +--+--+--+--+--+--+--+--+--+---------------+--+
```

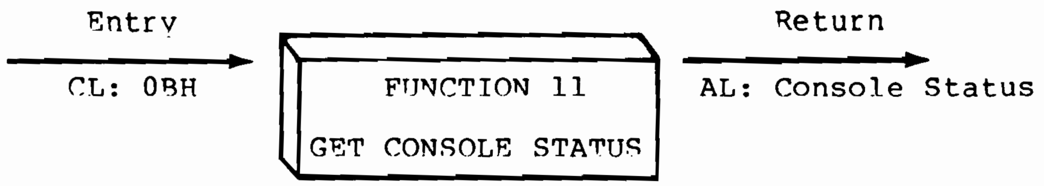
where "mx" is the maximum number of characters which the buffer will hold, and "nc" is the number of characters placed in the buffer. The characters entered by the operator follow the "nc" value. The value "mx" must be set prior to making a function 10 call and may range in value from 1 to 255. Setting 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.

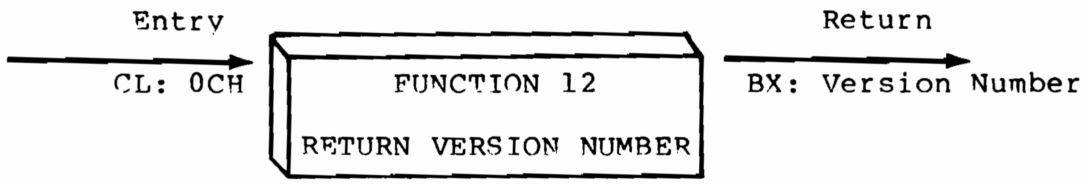
Table 4-3. Line Editing Controls

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.



The Console Status function checks to see if a character has been typed at the logical console device (CONSOLE). If a character is ready, the value 01H is returned in register AL. Otherwise a 00H value is returned.



Function 12 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:

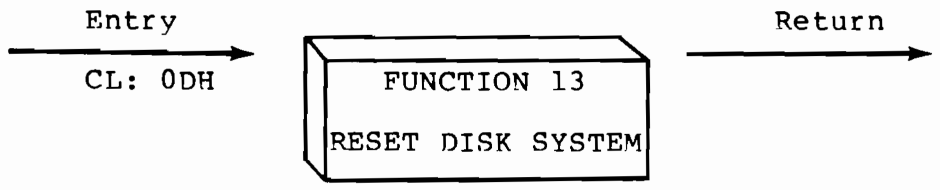
```
        BAD SECTOR      SELECT      R/O
```

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.

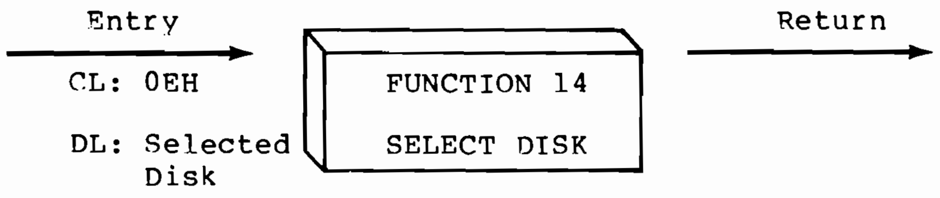
The "BAD SECTOR" error is issued as the result of an error condition returned to the BDOS from the BIOS module. The BDOS makes BIOS sector read and write commands as part of the execution of BDOS file related system calls. If the BIOS read or write routine detects a hardware error, it returns an error 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.

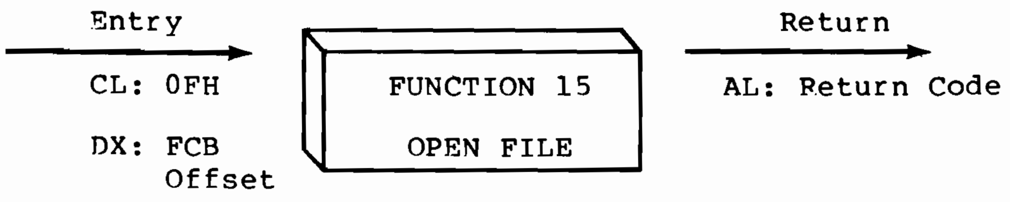
The "R/O" message occurs when the BDOS receives a command to write to a drive that is in read-only status. Drives may be placed in read-only status explicitly as the result of a STAT command or BDOS function call, or implicitly if the BDOS detects that disk media has been changed without performing a "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.



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.

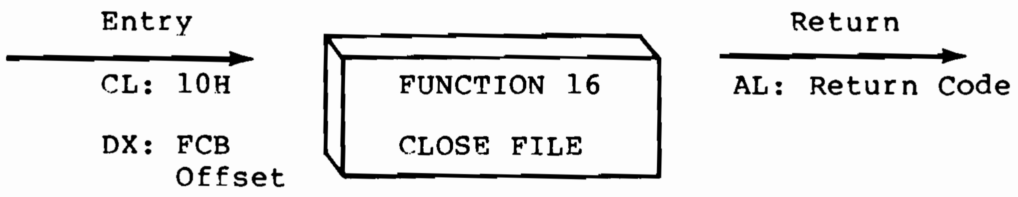


The Select Disk function designates the disk drive named in register DL as the default disk for subsequent file operations, with DL = 0 for drive A, 1 for drive B, and so-forth through 15 corresponding to drive P in a full sixteen drive system. In addition, the designated drive is logged-in if it is 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.

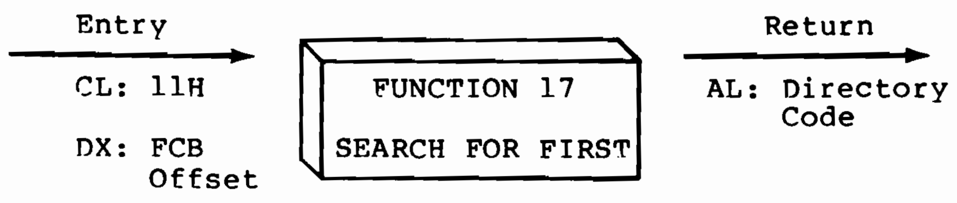


The Open File operation is used to activate a FCB specifying a file which currently exists in the disk directory for the currently active user number. The BDOS scans the disk directory of the drive specified by byte 0 of the FCB referenced by DX for a match in positions 1 through 12 of the referenced 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.

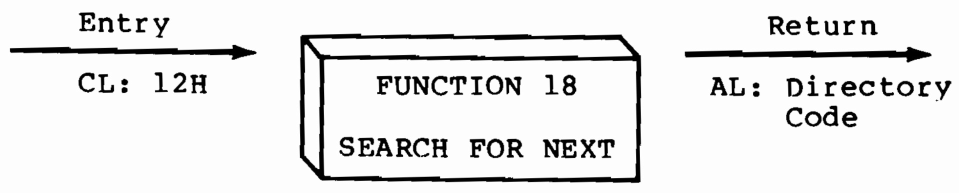


The Close File function performs the inverse of the open file function. Given that the FCB addressed by DX has been previously activated through an open or make function (see functions 15 and 22), the close function permanently records the new FCB in the referenced disk directory. The FCB matching process 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.

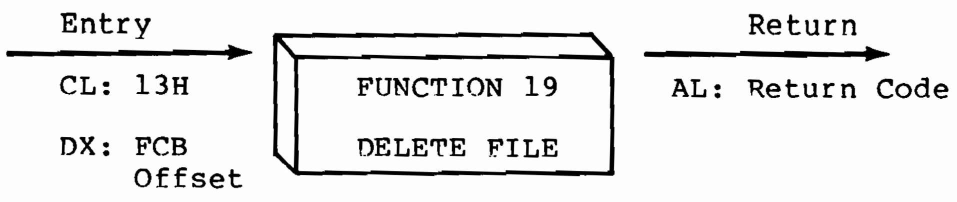


Search First scans the directory for a match with the file given by the FCB addressed by DX. The value 255 (hexadecimal FF) is returned if the file is not found, otherwise 0, 1, 2, or 3 is returned indicating the file is present. In the case that the file is found, the buffer at the current DMA address 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.

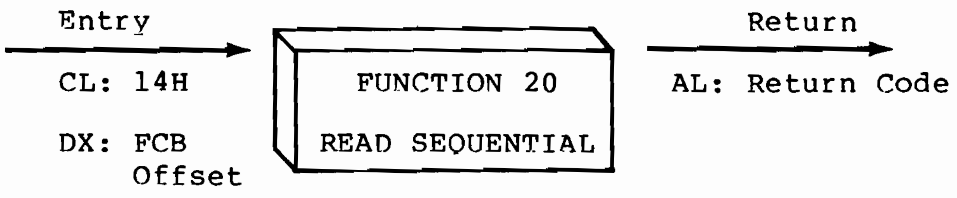
An ASCII question mark (63 decimal, 3F hexadecimal) in any position from "f1" through "ex" matches the corresponding field of any directory entry on the default or auto-selected disk drive. If the "dr" field contains an ASCII question mark, then the auto disk select function is disabled, the default 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.



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.



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.



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     | WRITE SEQUENTIAL |
            Offset  +------------------+
```

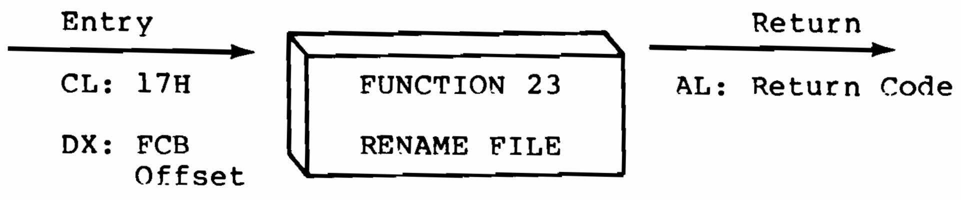
Given that the FCB addressed by DX has been activated through an open or make function (numbers 15 and 22), the Write Sequential function writes the 128 byte data record at the current DMA address to the file named by the FCB. The record is placed at position "cr" of the file, and the "cr" field is automatically incremented to the next record position. If the "cr" field overflows then the next logical extent is automatically opened and the "cr" field is reset to zero in preparation for the next write operation. Write operations can take place into an existing file, in which case newly written records overlay those which already exist in the file. The "cr" field must be set to zero following an open or make call by the user if the intent is to write sequentially from the beginning of the file. Register AL = 00H upon return from a successful write operation, while a non-zero value indicates an unsuccessful write due to one of the following conditions:

01 No available directory space - This condition occurs when the write command attempts to create a new extent that requires a new directory entry and no available directory entries exist on the selected disk drive.

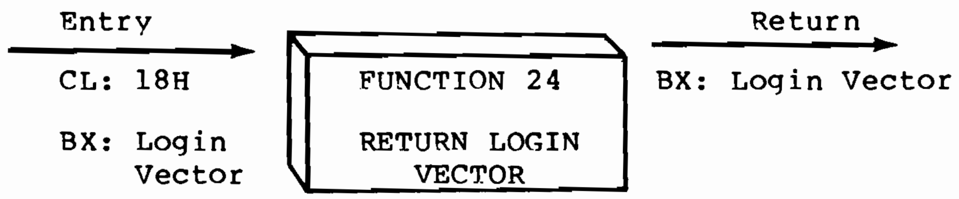
02 No available data block - This condition is encountered when the write command attempts to allocate a new data block to the file and no unallocated data blocks exist on the selected disk drive.

```
        Entry                                      Return
   ------------->   +------------------+   ------------->
        CL: 16H     |   FUNCTION 22    |    AL: Return Code
                    |                  |
        DX: FCB     |    MAKE FILE     |
            Offset  +------------------+
```

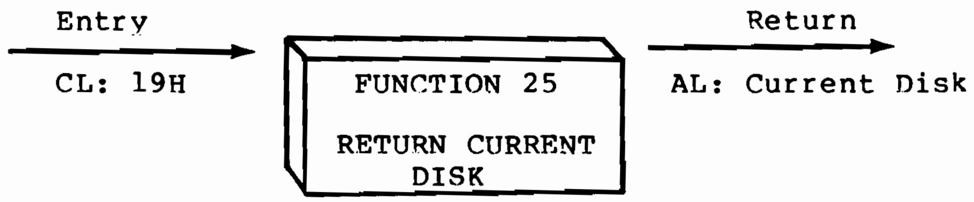
The Make File operation is similar to the open file operation except that the FCB must name a file which does not exist in the currently referenced disk directory (i.e., the one named explicitly by a non-zero "dr" code, or the default disk if "dr" is zero). The BDOS creates the file and initializes both the directory and main memory value to an empty file. The programmer must ensure that no duplicate 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.



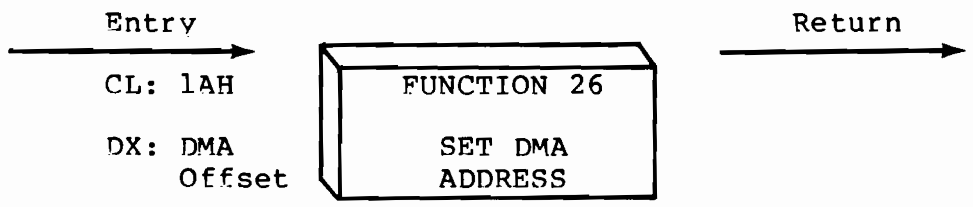
The Rename function uses the FCB addressed by DX to change all directory entries of the file specified by the file name in the first 16 bytes of the FCB to the file name in the second 16 bytes. It is the user's responsibility to insure that the file names specified are valid CP/M unambiguous file names. 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.



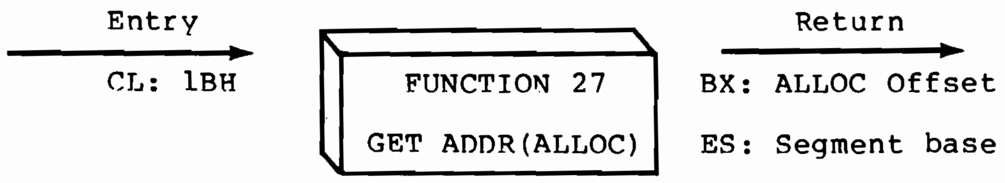
The login vector value returned by CP/M-86 is a 16-bit value in BX, where the least significant bit corresponds to the first drive A, and the high order bit corresponds to the sixteenth drive, labelled P. A "0" bit indicates that the drive is not on-line, while a "1" bit marks an drive that is actively 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.



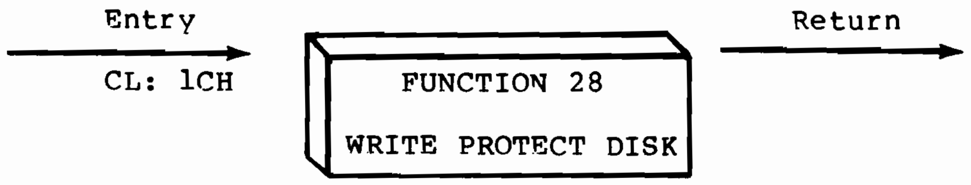
Function 25 returns the currently selected default disk number in register AL. The disk numbers range from 0 through 15 corresponding to drives A through P.



"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.

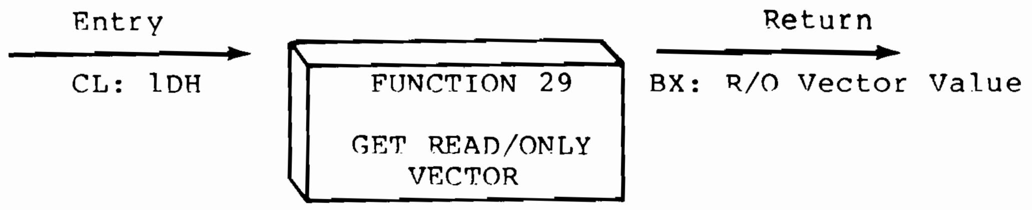


An "allocation vector" is maintained in main memory for each on-line disk drive. Various system programs use the information provided by the allocation vector to determine the amount of remaining storage (see the STAT program). Function 27 returns the segment base and the offset address of the allocation vector for the currently selected disk drive. The allocation information may, however, be invalid if the selected disk has been marked read/only.

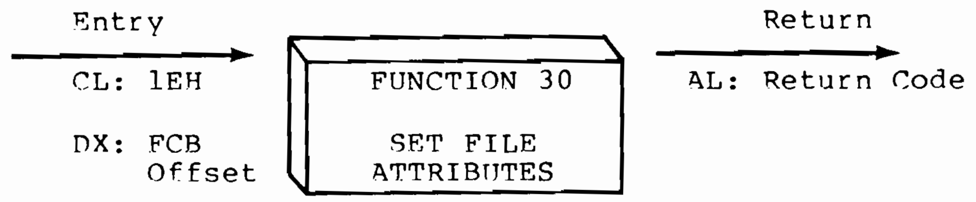


The disk write protect function provides temporary write protection for the currently selected disk. Any attempt to write to the disk, before the next cold start, warm start, disk system reset, or drive reset operation produces the message:

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



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.

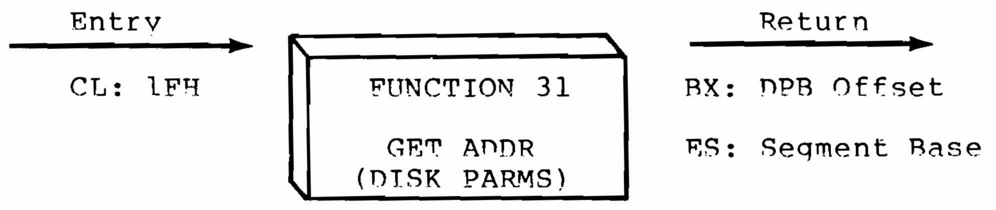


The Set File Attributes function allows programmatic manipulation of permanent indicators attached to files. In particular, the R/O, System and Archive attributes (t1', t2', and t3') can be set or reset. The DX pair addresses a FCB containing a file name with the appropriate attributes set or reset. It is the user's responsibility to insure that an ambiguous 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:

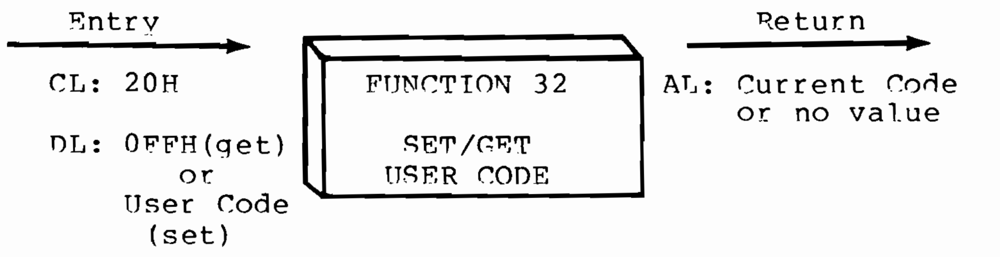
t1':	The R/O attribute indicates if set that the file is in read/only status. BDOS will not allow write commands to be issued to files in R/O status.
t2':	The System attribute is referenced by the CP/M DIR utility. If set, DIR will not display the file in a directory display.

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.

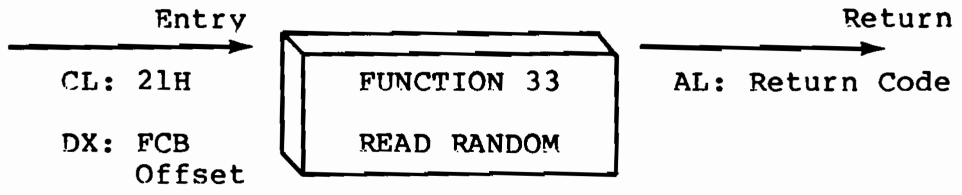
Function 30 returns with register AL set to 0FFH (255 decimal) if the referenced file cannot be found, otherwise a value of zero is returned.



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.



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).



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.

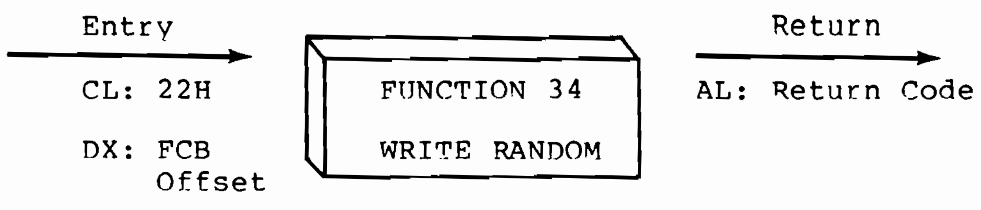
Upon each random read operation, the logical extent and current record values are automatically set. Thus, the file can be sequentially read or written, starting from the current randomly accessed position. Note, however, that in this case, the last randomly read record will be re-read as you switch 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.



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.

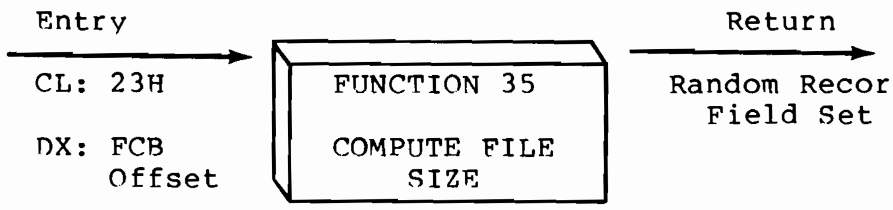
Upon return from a Write Random call, register AL either contains an error code, as listed in Table 4-5 below, or the value 00 indicating the operation was successful.

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)

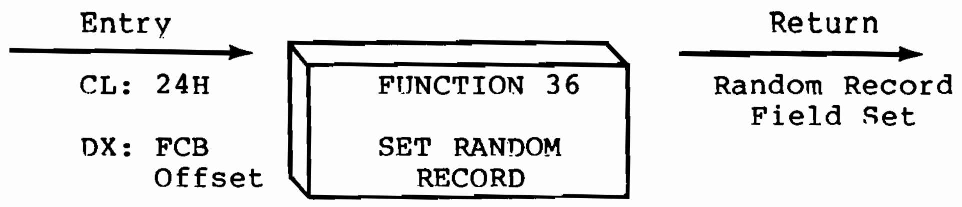
Code	Meaning
03	Cannot close current extent - This error code is returned when BDOS cannot close the current extent prior to moving to the new extent containing the record specified by bytes r0,r1 of the FCB. This error can be caused by an overwritten FCB or a write random operation on an FCB that has 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.



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.

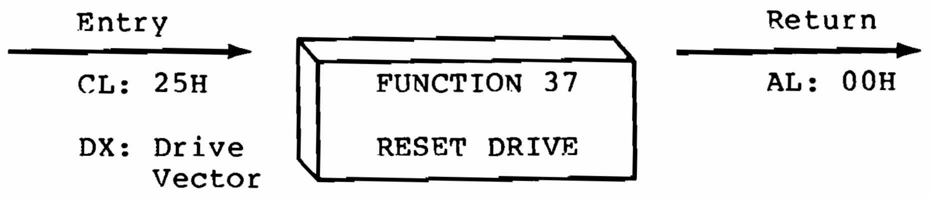
The virtual size of a file corresponds to the physical size when the file is written sequentially. If, instead, the file was created in random mode and "holes" exist in the allocation, then the file may in fact contain fewer records than the size indicates. If, for example, a single record with record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



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.

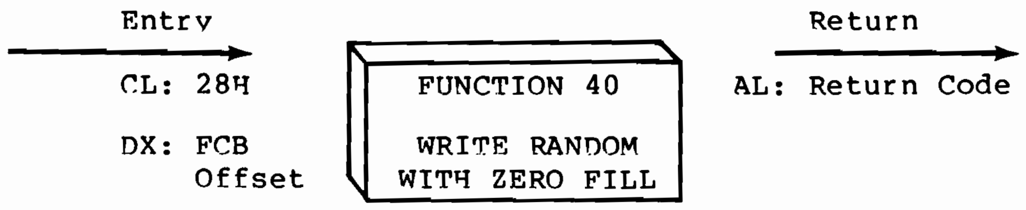
First, it is often necessary to initially read and scan a sequential file to extract the positions of various "key" fields. As each key is encountered, function 36 is called to compute the random record position for the data corresponding to this key. If the data unit size is 128 bytes, the resulting 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.

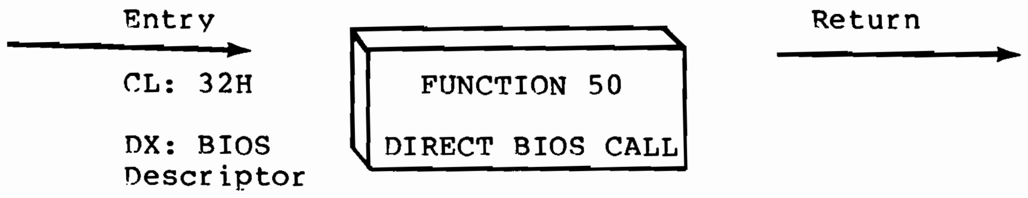


The Reset Drive function is used to programmatically restore specified drives to the reset state (a reset drive is not logged-in and is in read/write status). The passed parameter in register DX is a 16 bit vector of drives to be reset, where the least significant bit corresponds to the first drive, 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.



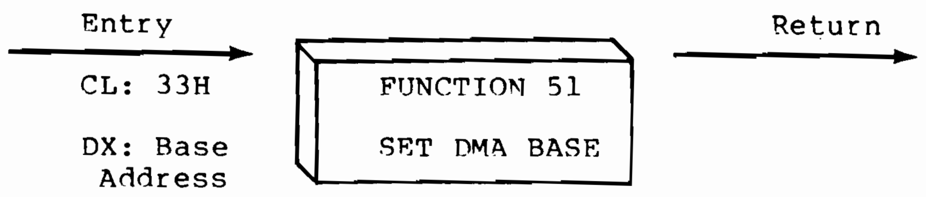
The Write Random With Zero Fill function is similar to the Write Random function (function 34) with the exception that a previously unallocated data block is initialized to records filled with zeros before the record is written. If this function has been used to create a file, records accessed by a read random operation that contain all zeros identify unwritten random record numbers. Unwritten random records in allocated data blocks of files created using the Write Random function contain uninitialized data.



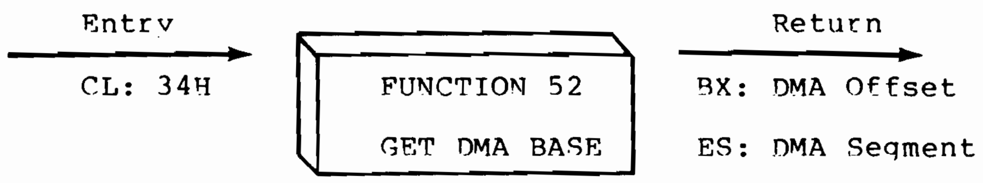
Function 50 provides a direct BIOS call and transfers control through the BDOS to the BIOS. The DX register addresses a five-byte memory area containing the BIOS call parameters:

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.



Function 51 sets the base register for subsequent DMA transfers. The word parameter in DX is a paragraph address and is used with the DMA offset to specify the address of a 128 byte buffer area to be used in the disk read and write functions. Note that upon initial program loading, the default DMA base is set to the address of the user's data segment (the initial value of DS) and the DMA offset is set to 0080H, which provides access to the default buffer in the base page.



Function 52 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:

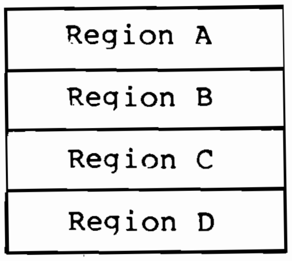


Figure 4-1. Example Memory Allocation

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.

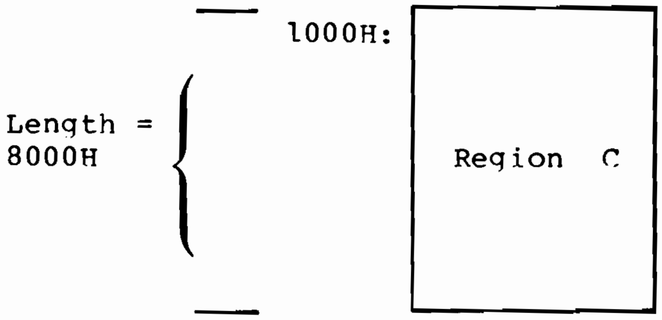


Figure 4-2. Example Memory Region

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.

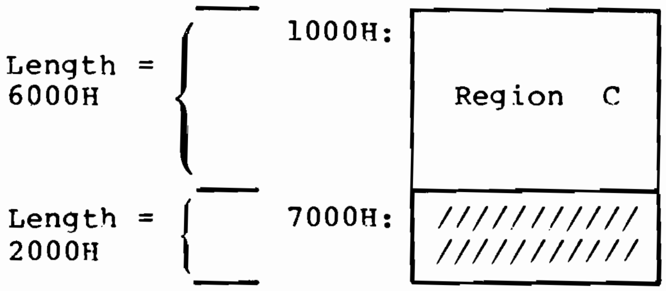


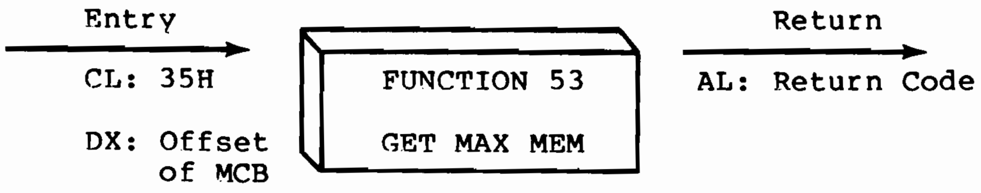
Figure 4-3. Example Memory Regions

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.

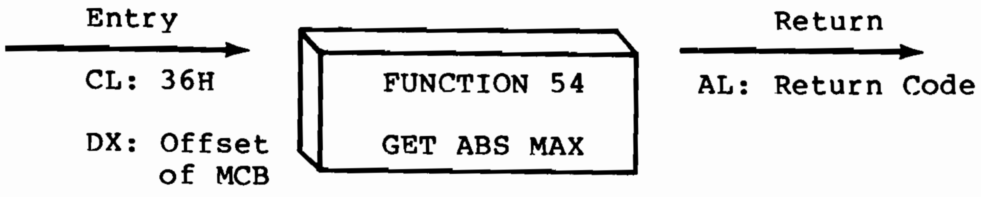
Memory management functions beginning at 53 reference a Memory Control Block (MCB), defined in the calling program, which takes the form:

	16-bit	16-bit	8-bit
MCB:	M-Base	M-Length	M-Ext

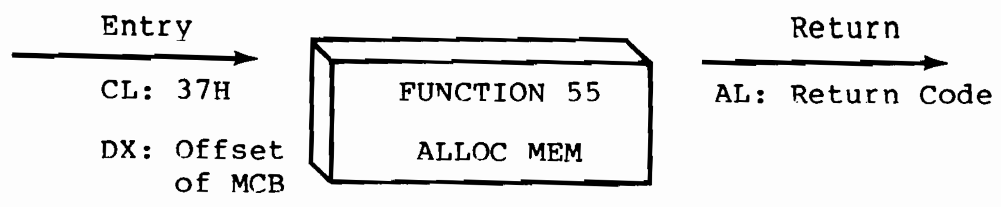
where M-Base and M-Length are either input or output values expressed in 16-byte paragraph units, and M-Ext is a returned byte value, as defined specifically with each function code. An error condition is normally flagged with a 0FFH returned value in order to match the file error conventions of CP/M.



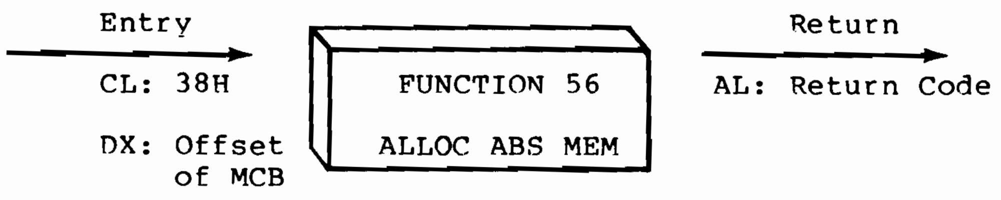
Function 53 finds the largest available memory region which is less than or equal to M-Length paragraphs. If successful, M-Base is set to the base paragraph address of the available area, and M-Length to the paragraph length. AL has the value 0FFH upon return if no memory is available, and 00H if the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



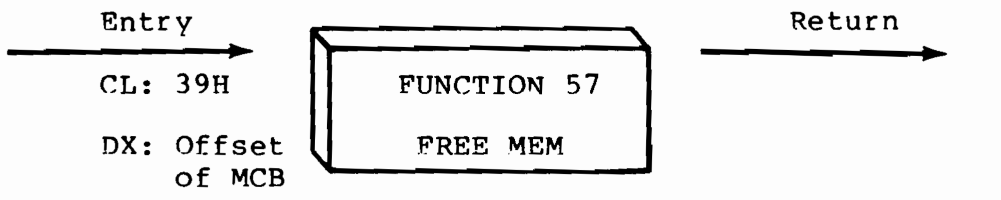
Function 54 is used to find the largest possible region at the absolute paragraph boundary given by M-Base, for a maximum of M-Length paragraphs. M-Length is set to the actual length if successful. AL has the value 0FFH upon return if no memory is available at the absolute address, and 00H if the request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



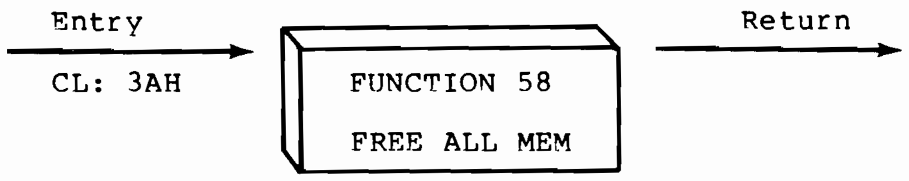
The allocate memory function allocates a memory area according to the MCB addressed by DX. The allocation request size is obtained from M-Length. Function 55 returns in the user's MCB the base paragraph address of the allocated region. Register AL contains a 00H if the request was successful and a 0FFH if the memory could not be allocated.



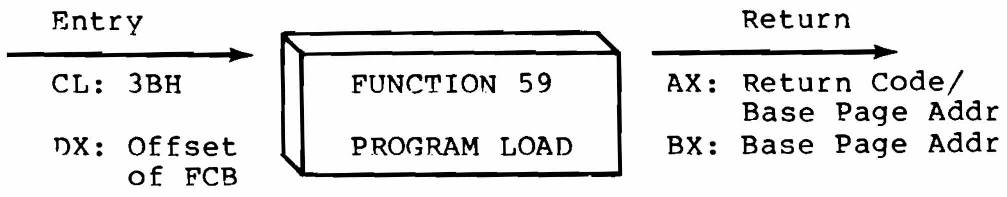
The allocate absolute memory function allocates a memory area according to the MCB addressed by DX. The allocation request size is obtained from M-Length and the absolute base address from M-Base. Register AL contains a 00H if the request was successful and a 0FFH if the memory could not be allocated.



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.



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



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:

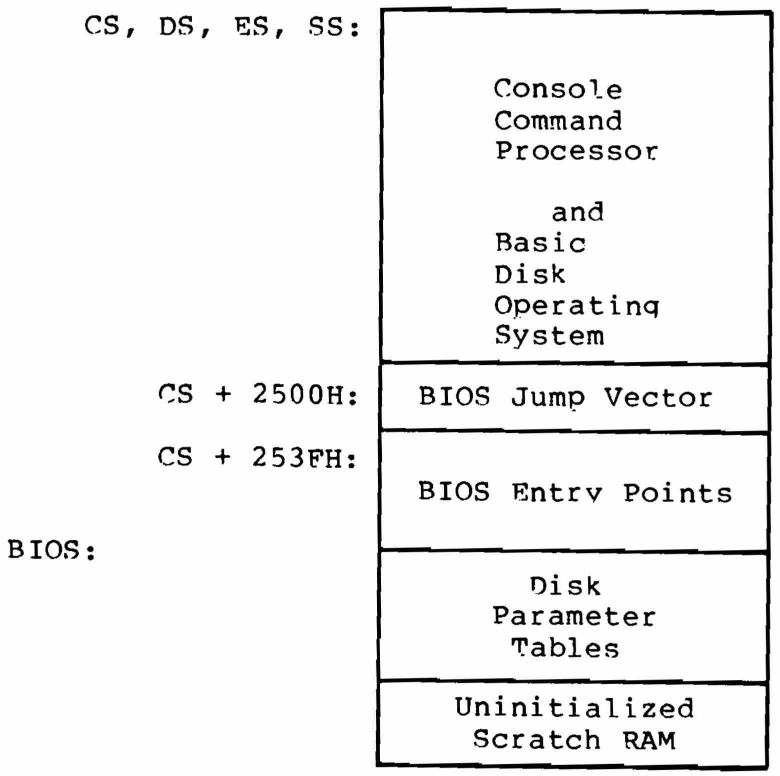


Figure 5-1. General CP/M-86 Organization

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 (0B11H) 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.

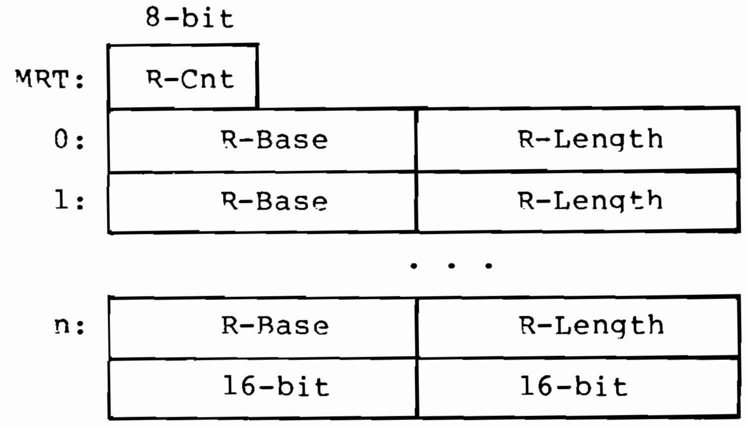
Table 5-4. (continued)

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

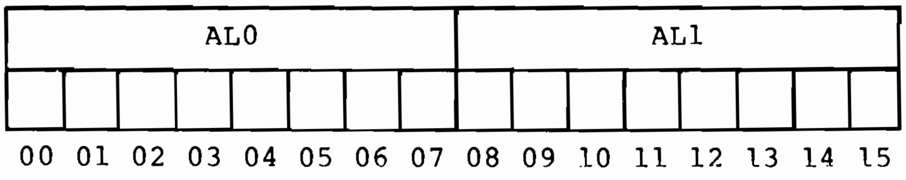
The value of EXM depends upon both the BLS and whether the DSM value is less than 256 or greater than 255, as shown in the following table.

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.



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 Adaption 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:

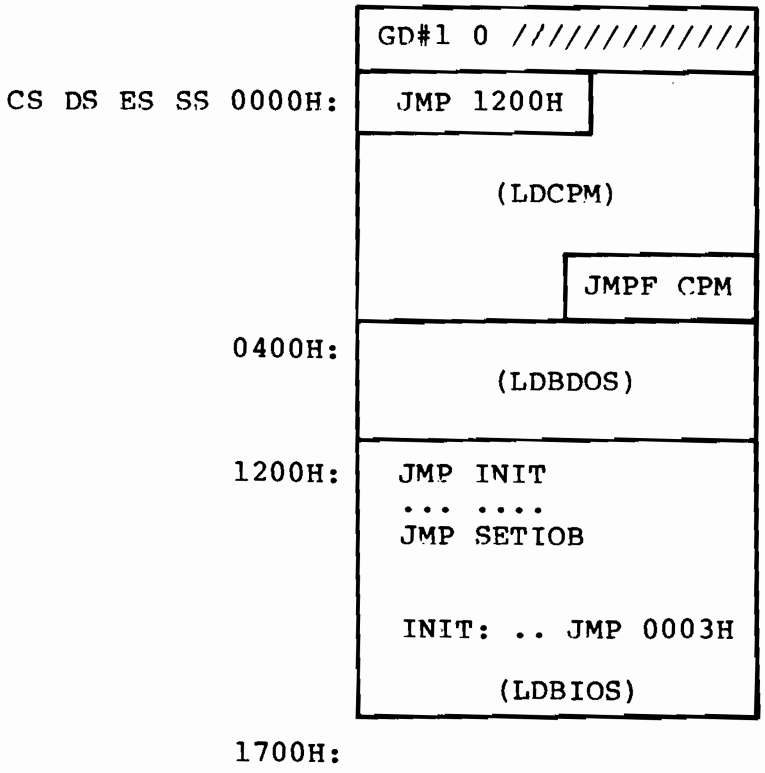


Figure 7-1. LOADER Organization

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.

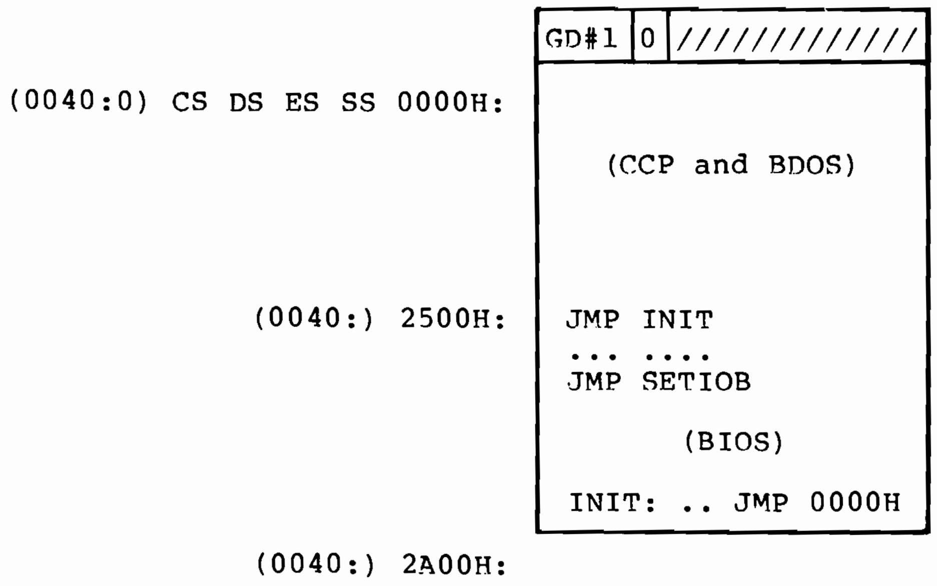


Figure 7-2. CPM.SYS File Organization

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  equ     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  equ     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 filhst1
;
        call readhst            ;yes, if 1
;       (check errors here)
;
filhst1:
        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  equu    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  equ     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:
```

```
                ;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 equ     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              Cerl     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               specsl            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
                 ;
                 ;               T0 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 R
```

```
                        ;|                                             |
                        ;-----------------------------------------------
                                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 buffe
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 drive
```

```
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,1[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 req 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 equ 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  equ     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_seq,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 co
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
```

All Information Presented Here is Proprietary to Digital Research

```
                                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
```

Index

A

allocate absolute memory, 52
allocate memory, 52

B

base page, 1
BIOS, 121
bootstrap, 4
bootstrap ROM, 81

C

CBIOS, 56, 137
close file, 34
CMD, 1, 15
cold start loader, 1, 56, 81
compact memory model, 11, 21
compute file size, 45
CONIN, 61
CONOUT, 61
console input, 25
console output, 25
console status, 30
CONST, 60
converting 8080 programs
  to CP/M-86, 3, 17, 23
cross development tools, 2

D

data block, 72, 74
delete file, 36
direct BIOS call, 47
direct console I/O, 27
directory entries, 71
disk definition tables, 4, 67
disk parameter block, 69
disk parameter header, 62,
  67, 75
DMA buffer, 14, 39, 60, 63

F

far call, 11, 14
file control block, 30
file structure, 1
free all memory, 53

G

GENCMD, 2, 3, 15, 17
GENDEF, 2
get address of disk parameter
  block, 41
get allocation vector
  address, 39
get DMA base, 48
get I/O byte, 27
get maximum memory, 51
get or set user code, 41
get read/only vector, 40
GETIOB, 65
GETSEGB, 65
group, 2

H

header record, 20
HOME, 61

I

INIT , 4, 60
Intel utilities, 17
IOBYTE, 58

L

L-module format, 19
LDCOPY, 2
LIST, 61
list output, 26
LISTST, 63
LMCMD, 19
logical to physical sector
  translation, 64

M

make file, 37
memory, 14
memory region table, 65
memory regions, 1

O

offset, 2
open file, 33

Index

P

print string, 28
program load, 53
PUNCH, 61
punch output, 26

R

random access, 95
READ, 63
read buffer, 29
read random, 42
read sequential, 36
READER, 61
reader input, 26
release all memory, 53
release memory, 52
rename, 38
reserved software interrupt, 1, 23
reset disk, 33
reset drive, 46
return current disk, 38
return login vector, 38
return version number, 30

S

search for first, 35
search for next, 35
sector blocking and deblocking, 87
SECTRAN, 64
segment, 2
segment group memory requirements, 17
segment register change, 11
segment register initialization, 8
SELDSK, 62
select disk, 33
set DMA address, 39
set DMA base, 48
set file attributes, 41
set I/O byte, 28
set random record, 46
SETDMA, 63
SETDMAB, 64
SETIOB, 65
SETSEC, 62
SETTRK, 62
small memory model, 10, 21
system reset, 4, 7, 14, 25 49, 60, 74

T

translation vectors, 69

U

utility program operation, 2

W

WBOOT, 60
WRITE, 63
write protect disk, 39
write random, 44
write random with zero fill, 47

8080 memory model, 3, 10, 14, 21

DIGITAL RESEARCH®

CP/M-86®

Programmer's Guide

595-3120

Printed in the
United States of America

ZENITH data systems

HEATH

DIGITAL RESEARCH™

CP/M-86®
Operating System

Programmer's Guide

COPYRIGHT

DISCLAIMER

TRADEMARKS

The *CP/M-86 Operating System Programmer's Guide* was prepared using the Digital Research TEX-80 text formatter and printed in the United States of America.

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 Operating System User's 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

1 **Introduction**

1.1 Assembler Operation 1
1.2 Optional Run-time Parameters 3
1.3 Aborting ASM-86 5

2 **Elements of ASM-86 Assembly Language**

2.1 ASM-86 Character Set 7
2.2 Tokens and Separators 7
2.3 Delimiters 7
2.4 Constants 9
2.4.1 Numeric Constants 9
2.4.2 Character Strings 10
2.5 Identifiers 11
2.5.1 Keywords 11
2.5.2 Symbols and Their Attributes 13
2.6 Operators 14
2.6.1 Operator Examples 18
2.6.2 Operator Precedence 20
2.7 Expressions 22
2.8 Statements 23

3 **Assembler Directives**

3.1 Introduction 25
3.2 Segment Start Directives 25
3.2.1 The CSEG Directive 26
3.2.2 The DSEG Directive 26
3.2.3 The SSEG Directive 27
3.2.4 The ESEG Directive 27
3.3 The ORG Directive 28
3.4 The IF and ENDIF Directives 28
3.5 The INCLUDE Directive 29
3.6 The END Directive 29
3.7 The EQU Directive 29
3.8 The DB Directive 30
3.9 The DW Directive 31
3.10 The DD Directive 31

Table of Contents (continued)

3.11 The RS Directive 32
3.12 The RB Directive 32
3.13 The RW Directive 32
3.14 The TITLE Directive 33
3.15 The PAGESIZE Directive 33
3.16 The PAGEWIDTH Directive 33
3.17 The EJECT Directive 33
3.18 The SIMFORM Directive 34
3.19 The NOLIST and LIST Directives 34

4 The ASM-86 Instruction Set

4.1 Introduction 35
4.2 Data Transfer Instructions 37
4.3 Arithmetic, Logical, and Shift Instructions 40
4.4 String Instructions 45
4.5 Control Transfer Instructions 47
4.6 Processor Control Instructions 51

5 Code-Macro Facilities

5.1 Introduction to Code-macros 53
5.2 Specifiers 55
5.3 Modifiers 56
5.4 Range Specifiers 56
5.5 Code-macro Directives 57
5.5.1 SEGFIX 57
5.5.2 NOSEGFIX 57
5.5.3 MODRM 58
5.5.4 RELB and RELW 59
5.5.5 DB, DW and DD 59
5.5.6 DBIT 60

6 DDT-86

6.1 DDT-86 Operation 63
6.1.1 Invoking DDT-86 63
6.1.2 DDT-86 Command Conventions 63
6.1.3 Specifying a 20-Bit Address 64
6.1.4 Terminating DDT-86 65

Table of Contents (continued)

6.1.5 DDT-86 Operation with Interrupts 65
6.2 DDT-86 Commands 66
6.2.1 The A (Assemble) Command 66
6.2.2 The D (Display) Command 66
6.2.3 The E (Load for Execution) Command 67
6.2.4 The F (Fill) Command 68
6.2.5 The G (Go) Command 68
6.2.6 The H (Hexidecimal Math) Command 69
6.2.7 The I (Input Command Tail) Command 69
6.2.8 The L (List) Command 70
6.2.9 The M (Move) Command 71
6.2.10 The R (Read) Command 71
6.2.11 The S (Set) Command 71
6.2.12 The T (Trace) Command 72
6.2.13 The U (Untrace) Command 73
6.2.14 The V (Value) Command 73
6.2.15 The W (Write) Command) 74
6.2.16 The X (Examine CPU State) Command 74
6.3 Default Segment Values 76
6.4 Assembly Language Syntax for A and L Commands 78
6.5 DDT-86 Sample Session 80

Table of Contents (continued)

Appendixes

A ASM-86 Invocation 93

B Mnemonic Differences from the Intel Assembler 95

C ASM-86 Hexadecimal Output Format 97

D Reserved Words 101

E ASM-86 Instruction Summary 103

F Sample Program 107

G Code-Macro Definition Syntax 113

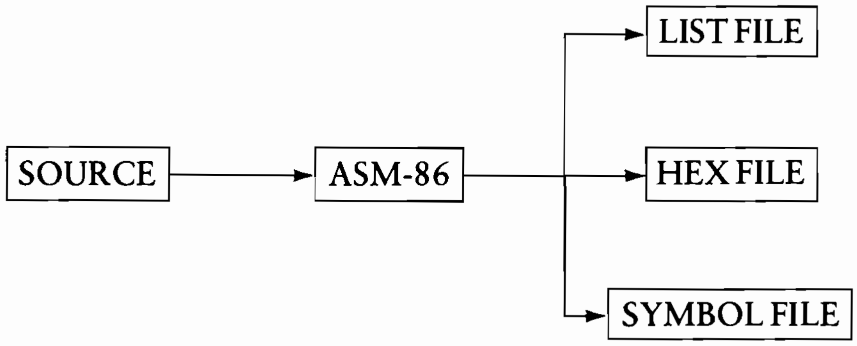
H ASM-86 Error Messages 115

I DDT-86 Error Messages 117

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 Z80® 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 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.A86  $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 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 Parameters

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, 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.

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.

End of Section 1

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	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 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 decimal digit so that ASM-86 cannot confuse a hex constant with an identifier, or leading 0 to prevent this problem. The following are valid numeric constants:

```
1234      1234D     1100B     1111000011110000B
1234H     0FFEH     33770     13772Q
33770     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 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
```

Labels identify locations in memory that contain instruction statements. They are referenced with jumps or calls. All labels have two attributes:

- Segment
- Offset

Label segment and offset attributes are essentially the same as variable segment and offset attributes. Generally, a label is defined when it precedes an instruction. A colon, :, separates the label from instruction; for example:

```
LABEL:   ADD  AX,BX
```

A label may also appear as the name for an EQU directive referencing another label; for example:

```
LABEL   EQU   ANOTHER_LABEL
```

Numbers may also be defined as symbols. A number symbol is treated as if you had explicitly coded the number it represents. For example:

```
Number_five    EQU   5
MOV    AL,Number_five
```

is equivalent to:

```
MOV    AL,5
```

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
	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

Table 2-5. (continued)

Syntax	*Result*	*Validity*
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
	Segment Override	
<seg reg>: *<addr exp>*	overrides assembler's choice of segment register.	*<seg reg>* = CS, DS, SS or ES
	Variable Manipulators, Creators	
SEG a	creates a number whose value is the segment value of the variable or label a.	a = label | variable
OFFSET a	creates a number whose value is the offset value of the variable or label a.	a = label | variable

Table 2-5. (continued)

Syntax	*Result*	*Validity*
TYPE a	creates a number. If the variable a is of type BYTE, WORD or DWORD, the value of the number will be 1, 2 or 4, respectively.	a = label | variable
LENGTH a	creates a number whose value is the LENGTH attribute of the variable a. The length attribute is the number of bytes associated with the variable.	a = label | variable
LAST a	if LENGTH a > 0, then LAST a = LENGTH a − 1; if LENGTH a = 0, then LAST a = 0.	a = label | variable
a PTR b	creates virtual variable or label with type of a and attributes of b.	a = BYTE | WORD, | DWORD b = *<addr exp>*
.a	creates variable with an offset attribute of a. Segment attribute is current segment.	a = number
$	creates label with offset equal to current value of location counter; segment attribute is current segment.	no argument

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. 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.

End of Section 2

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. 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
                                         .
                                         .
                                         .
0005D 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. 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.

End of Section 3

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
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 4-2 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 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
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

Table 4-3. (continued)

Syntax		*Result*
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

Table 4-5. (continued)

Syntax		*Result*
DAS		decimal adjust for subtraction, adjusts AL
DEC	mem|reg	subtract 1 from memory or register
INC	mem|reg	add 1 to memory or register
DIV	mem|reg	divide (unsigned) accumulator (AX or AL) by memory or register. If byte results, AL = quotient, AH = remainder. If word results, AX = quotient, DX = remainder
IDIV	mem|reg	divide (signed) accumulator (AX or AL) by memory or register—quotient and remainder stored as in DIV
IMUL	mem|reg	multiply (signed) memory or register by accumulator (AX or AL)—if byte, results in AH, AL. If word, results in DX, AX
MUL	mem|reg	multiply (unsigned) memory or register by accumulator (AX or AL)—results stored as in IMUL
NEG	mem|reg	two's complement memory or register
SBB	reg,mem|reg	subtract (with borrow) memory or register from register
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 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

Table 4-6. (continued)

Syntax		Result
ROR	mem|reg,CL	rotate memory or register right, number of bits given by CL register
SAL	mem|reg,1	shift memory or register 1 bit left, shift in low-order zero bits
SAL	mem|reg,CL	shift memory or register left, number of bits given by CL register, shift in low-order zero bits
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

Table 4-6. (continued)

Syntax		*Result*
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
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

Table 4-9. (continued)

Syntax		*Result*
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)
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)

Table 4-9. (continued)

Syntax		*Result*
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'
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'

Table 4-9. (continued)

Syntax		*Result*
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'
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

Table 4-9. (continued)

Syntax		*Result*
RETF		return to the address pushed by a previous CALLF instruction, increment stack pointer by 4
RETF	numb	return to the address pushed by a previous CALLF instruction, increment stack pointer by 4+numb

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

End of Section 4

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          0D3H
  MODRM       3,dst
EndM

CodeMacro OR dst:Rw,src:Ew
  SEGFIX      src
  DB          0BH
  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.

End of Section 5

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 or ↑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 indicated by the X command. If omitted, a default value appropriate to the command being executed, 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, which can be displayed with the X command, 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 binary, 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. 46-bit value at location 0:6. 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 an instruction is executed, DDT-86 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 before n steps have been executed 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 assumed to be 0. 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 1-character abbreviation of the flag name, indicating that the flag is set (1). The abbreviations of the flag names are shown in Table 6-2. *<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 has an internal mechanism that 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	c	c
F		x
G	c	c
H		
I		
L	x	
M		x
R	c	c
S		x
T	c	c
U	c	c
V		
W	x	
X	c	c

x — use this segment default if none specified; change default if specified explicitly
c — change 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.

use [illegible]

```
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
:1B0000B1BE0000BB0001C6060801008A003A4001760A8640018800C6060801 6C
:11001B81014683FE0875E9F606080101750 7E9FDFFEE
:0901008203080406 1F0604010035
:00000001FF
```

Generate CMD dile 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 to patch.)

```
-a1d
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
```

See if list is sorted.

```
-d100,10f
0480:0100 03 04 06 08 1F 06 04 01 00 00 00 00 00 00 00 00 ................
```

Interesting, the fourth element seems to have been sorted in.

```
-esort
     START      END
CS 047D:0000 047D:002F
DS 0480:0000 0480:010F
```

Let's try again with some tracing.

```
-a1d
047D:001D cmp si,3
047D:0020 .

-t9
           AX   BX   CX   DX   SP   BP   SI   DI   IP
-----Z-P- 0006 0100 0000 0000 119E 0000 0003 0000 0000 MOV    SI,0000
-----Z-P- 0006 0100 0000 0000 119E 0000 0000 0000 0003 MOV    BX,0100
-----Z-P- 0006 0100 0000 0000 119E 0000 0000 0000 0006 MOV    BYTE [0108],00
-----Z-P- 0006 0100 0000 0000 119E 0000 0000 0000 000B MOV    AL,[BX+SI]
-----Z-P- 0003 0100 0000 0000 119E 0000 0000 0000 000D CMP    AL,01[BX+SI]
----S-A-C 0003 0100 0000 0000 119E 0000 0000 0000 0010 JBE    001C
----S-A-C 0003 0100 0000 0000 119E 0000 0000 0000 001C INC    SI
--------C 0003 0100 0000 0000 119E 0000 0001 0000 001D CMP    SI,0003
----S-A-C 0003 0100 0000 0000 119E 0000 0001 0000 0020 JNZ    000B
*047D:000B
```

```
-l
047D:000B MOV    AL,[BX+SI]
047D:000D CMP    AL,01[BX+SI]
047D:0010 JBE    001C
047D:0012 XCHG   AL,01[BX+SI]
047D:0015 MOV    [BX+SI],AL
047D:0017 MOV    BYTE [0108],01
047D:001C INC    SI
047D:001D CMP    SI,0003
047D:0020 JNZ    000B
047D:0022 TEST   BYTE [0108],01
047D:0027 JNZ    0000
047D:0029 JMP    0029

-t3
           AX   BX   CX   DX   SP   BP   SI   DI   IP
----S-A-C 0003 0100 0000 0000 119E 0000 0001 0000 000B MOV    AL,[BX+SI]
----S-A-C 0008 0100 0000 0000 119E 0000 0001 0000 000D CMP    AL,01[BX+SI]
--------- 0008 0100 0000 0000 119E 0000 0001 0000 0010 JBE    001C
*047D:0012

-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. Reload the program.

```
-esort
     START      END
CS 047D:0000 047D:002F
DS 0480:0000 0480:010F
```

```
-l
047D:0000 MOV    SI,0000
047D:0003 MOV    BX,0100
047D:0006 MOV    BYTE [0108],00
047D:000B MOV    AL,[BX+SI]
047D:000D CMP    AL,01[BX+SI]
047D:0010 JBE    001C
047D:0012 XCHG   AL,01[BX+SI]
047D:0015 MOV    [BX+SI],AL
047D:0017 MOV    BYTE [0108],01
047D:001C INC    SI
047D:001D CMP    SI,0008
047D:0020 JNZ    000B
```

Patch length.

```
-a1d
047D:001D cmp si,7
047D:0020 .
```

Try it out.

```
-g,29
*047D:0029
```

See if list is sorted.

```
-d100,10f
0480:0100 01 03 04 04 06 06 08 1F 00 00 00 00 00 00 00 00 ................
```

Looks better; let's install patch in disk file. To do this, we must read CMB file including header, so we use R command.

```
-rsort.cmd
  START      END
2000:0000 2000:01FF
```

First 80h bytes contain header, so code starts at 80h.

```
-l80
2000:0080 MOV    SI,0000
2000:0083 MOV    BX,0100
2000:0086 MOV    BYTE [0108],00
2000:008B MOV    AL,[BX+SI]
2000:008D CMP    AL,01[BX+SI]
2000:0090 JBE    009C
2000:0092 XCHG   AL,01[BX+SI]
2000:0095 MOV    [BX+SI],AL
2000:0097 MOV    BYTE [0108],01
2000:009C INC    SI
2000:009D CMP    SI,0008
2000:00A0 JNZ    008B
```

Install patch.

```
-a9d
2000:009D cmp si,7
2000:00A0
```

Write file back to disk. (Length of file assumed to be unchanged since no length specified.)

```
-wsort.cmd
```

Reload file.

```
-esort

     START       END
CS 047D:0000 047D:002F
DS 0480:0000 0480:010F
```

Verify that patch was installed.

```
-l
047D:0000 MOV    SI,0000
047D:0003 MOV    BX,0100
047D:0006 MOV    BYTE [0108],00
047D:000B MOV    AL,[BX+SI]
047D:000D CMP    AL,01[BX+SI]
047D:0010 JBE    001C
047D:0012 XCHG   AL,01[BX+SI]
047D:0015 MOV    [BX+SI],AL
047D:0017 MOV    BYTE [0108],01
047D:001C INC    SI
047D:001D CMP    SI,0007
047D:0020 JNZ    000B
```

Run it.

```
-g,29
*047D:0029
```

Still looks good. Ship it!

```
-d100,10f
0480:0100 01 03 04 04 06 06 08 1F 00 00 00 00 00 00 00 00 ................
-^C
A>
```

End of Section 6

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

Valid Parameters

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.

End of Appendix A

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

End of Appendix B

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

End of Appendix C

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.

End of Appendix D

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

Table E-1. (continued)

Mnemonic	*Description*	*Section*
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
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

Table E-1. (continued)

Mnemonic	*Description*	*Section*
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
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

Table E-1. (continued)

Mnemonic	*Description*	*Section*
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

End of Appendix E

Appendix F
Sample Program

```
CP/M ASM86 1.1  SOURCE: APPF.A86      Terminal Input/Output           PAGE    1



                    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   /
                    ;       ***********************
```

Listing F-1. Sample Program APPF.A86

```
CP/M ASM86 1.1  SOURCE: APPF.A86      Terminal Input/Output          PAGE   2


                   ;
                   ;        Terminal 1:
                   ;
  0010             instat1        equ     10h     ; input status port
  0011             indata1        equ     11h     ; input port
  0011             outdata1       equ     11h     ; output port
  0001             readyinmask1   equ     01h     ; input ready mask
  0002             readyoutmask1  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
                   constat1:
 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
```

Listing F-1. (continued)

```
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         conin1: call constat1              ; test status
 0026 74FB                   jz   conin1
 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]
                     conout1:
 003E EC                     in   al,dx
```

Listing F-1. (continued)

```
CP/M ASM86 1.1  SOURCE: APPF.A86       Terminal Input/Output          PAGE   4


 003F 22470B                and  al,readyoutmasktab [BX]
 0042 74FA                  jz   conout1
 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 *
                    ;       **************************
```

Listing F-1. (continued)

```
CP/M ASM86 1.1  SOURCE: APPF.A86      Terminal Input/Output         PAGE    5


                  ;

 0000 1012        instatustab     db      instat1,instat2
 0002 1113        indatatab       db      indata1,indata2
 0004 1113        outdatatab      db      outdata1,outdata2
 0006 0104        readyinmasktab  db      readyinmask1,readyinmask2
 0008 0208        readyoutmasktab db      readyoutmask1,readyoutmask2
                  ;
                  ;*************** end of file **********************
                  end

END OF ASSEMBLY, NUMBER OF ERRORS:   0
```

Listing F-1. (continued)

End of Appendix F

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

End of Appendix G

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. Diagnostic 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

Table H-1. (continued)

Number	*Meaning*
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

End of Appendix H

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.

Table I-1. (continued)

Error Message	*Meaning*
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.

End of Appendix I

Index

A

AAA, 41
AAD, 41
AAM, 41
AAS, 41
ADC, 41
ADD, 41
address conventions in
  ASM-86, 25
address expression, 22
allocate storage, 32
AND, 43
arithmetic operators, 18-19

B

bracketed expression, 22

C

CALL, 47
CBW, 41
character string, 10
CLC, 51
CLD, 51
CLI, 51
CMC, 51
CMP, 41
CMPS, 46
code segment, 26
code-macro directives, 57
code-macros, 53
conditional assembly, 28
console output, 4
constants, 9
control transfer
  instructions, 47
creation of output files, 3
CSEG, 26
CWD, 41

D

DAA, 41
DAS, 42
data segment, 26
data transfer, 37
DB, 30
DD, 31
DEC, 42
defined data area, 30
delimiters, 7
directive statement, 24
DIV, 42
dollar-sign operator, 20
DSEG, 26
DW, 31

E

effective address, 25
EJECT, 33
END, 29
end-of-line, 23
ENDIF, 28
EQU, 29
ESC, 51
ESEG, 27
expressions, 22
extra segment, 27

F

filename extensions, 2
flag bits, 37, 40
flag registers, 37
formal parameters, 53

H

HLT, 52

I

identifiers, 11
IDIV, 42
IF, 28
IMUL, 42
IN, 38
INC, 42
INCLUDE, 29
initialized storage, 30
instruction statement, 23
INT, 47
INTO, 48
invoking ASM-86, 2
IRET, 48

J

JA, 48
JB, 48
JCXZ, 48
JE, 48
JG, 48
JL, 48
JLE, 49
JMP, 49
JNA, 49
JNB, 49
JNE, 49
JNG, 49
JNL, 49
JNO, 49
JNP, 49
JNS, 50
JNZ, 50
JO, 50
JP, 50
JS, 50
JZ, 50

K

keywords, 11

L

label, 23
labels, 13
LAHF, 38
LDS, 38
LEA, 38
LES, 38
LIST, 34
location counter, 28
LOCK, 52
LODS, 46
logical operators, 18
LOOP, 50

M

mnemonic, 23
modifiers, 56
MOV, 38
MOVS, 46
MUL, 42

N

name field, 24
NEG, 42
NOLIST, 34
NOT, 43
number symbols, 14
numeric constants, 9
numeric expression, 22

O

offset, 13
offset value, 25
operator precedence, 20
operators, 14
optional run-time
  parameters, 3
OR, 43
order of operations, 20
ORG, 28
OUT, 38
output files, 2, 3

P

PAGESIZE, 33
PAGEWIDTH, 33
period operator, 20
POP, 39
predefined numbers, 11
prefix, 23, 46
printer output, 4
PTR operator, 20
PUSH, 39

R

radix indicators, 9
RB, 32
RCL, 43
RCR, 43
registers, 11
relational operators, 18
REP, 46
RET, 50
ROL, 43
ROR, 43
RS, 32
run-time options, 3
RW, 32

S

SAHF, 39
SAL, 44
SAR, 44
SBB, 42
SCAS, 46
segment, 13
segment base values, 25
segment override operator, 19
segment start directives, 25
separators, 7
SHL, 44
SHR, 44
SIMFORM, 34
specifiers, 55
SSEG, 26
stack segment, 27
starting ASM-86, 2
statements, 23

STC, 52
STD, 52
STI, 52
STOS, 46
string constant, 10
string operations, 45
SUB, 42
symbols, 29

T

TEST, 44
TITLE, 33
type, 13

U

unary operators, 19

V

variable manipulator, 19
variables, 13

W

WAIT, 52

X

XCHG, 39
XLAT, 39

ZENITH | data systems

8-Inch Disk Step Rates

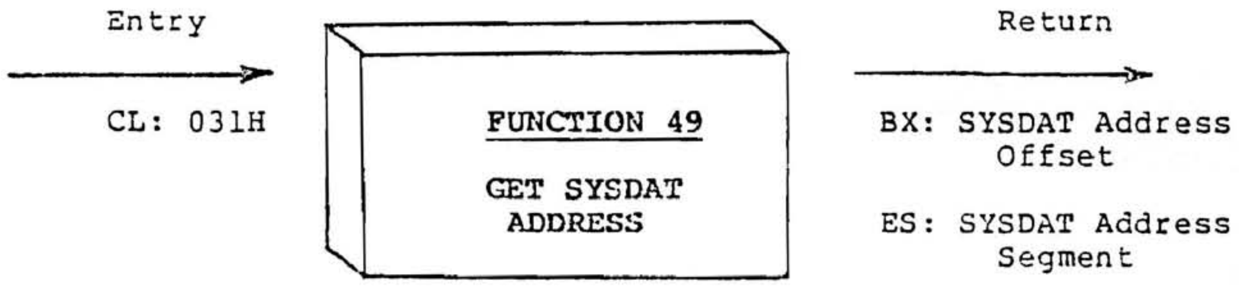
Zenith Data Systems is providing 8-inch drivers installed in the CP/M-86 disk systems for development purposes and to support future products. Zenith Data Systems does not guarantee proper operation of the 8-inch drivers with disk systems obtained from other vendors. However, for the benefit of customers who wish to experiment with non-Zenith hardware at their own risk, the track-to-track stepping rate, which is set at 3 milliseconds, may be changed by using the DDT86 command. In the example shown below, the rate is changed from 3 milliseconds to 15 milliseconds. User input is shown in bold:

```
A>STAT CPM.SYS $R/W

CPM.SYS set to R/W
A>REN CPMSYS.OLD=CPM.SYS
A>DDT86
DDT86 VER 1.1
-RCPMSYS.OLD
  START      END
nnnn:0000 nnnn:xxxx
-XCS
CS 0000 nnnn+8          ;The effect of the register
DS 0000 nnnn+8          ;change is to set all segment
SS 0040 nnnn+8          ;registers to the beginning of
ES 0000 nnnn+8          ;the program past the header
IP 0000
-S4B11
nnnn+8:4B11 00 03       (00=3mS, 01=6mS, 02=10mS, 03=mS)
nnnn+8:4B12 04 .
-S4B29
nnnn+8:4B29 00 03       (00=3mS, 01=6mS, 02=10mS, 03=mS)
nnnn+8:4b2A 04 .
-WCPM.SYS
-^C
```

PAGE 47 (continued)

Then, add:



Return the address of the System Data Area

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
listcp             equ      byte ptr 22      ;listing toggle...
                                             ;set by ctrl-p
curdrvs            equ      byte ptr 23      ;current drives to
                                             ;search
console_width      equ      byte ptr 64
printer_width      equ      byte ptr 65
console_column     equ      byte ptr 66
printer_column     equ      byte ptr 67
```

PAGE 60

In Table 5-4. BIOS Subroutine Summary, in the description of subroutine INIT,
change:

BDOS offset (0B11H)

to:

BDOS offset **(0B06H)**

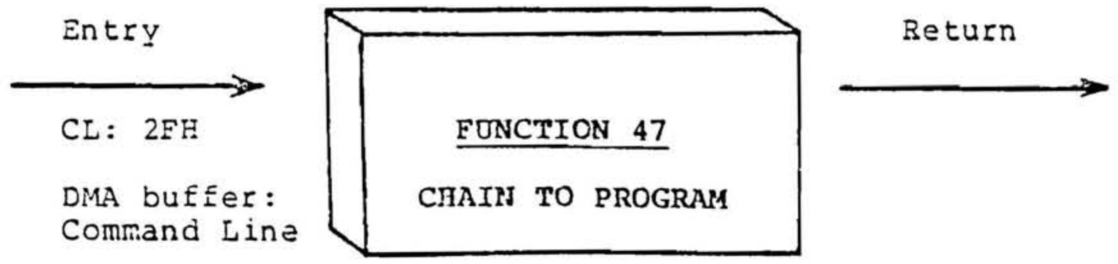
CP/M-86T.M. Operating System

SYSTEM GUIDE

Enhancements to the First Printing - 1981
Copyright © 1981 by Digital Research, Inc.
CP/M-86 is a trademark of Digital Research.
Compiled February 1, 1982

PAGE 47

In Section 4.3, BDOS File Operations,
Add two new BDOS Functions:



Load, Initialize, and Jump to specified 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-36T.M., 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.

CP/M-86$^{T.M.}$ V1.1, Application Note 01, 3/08/82

BDOS DATA PAGE "TOD/DATA" FIELDS

Applicable products and version numbers: CP/M-86 V1.1

Program: BDOS

The date field is located at the base of the data page + 32D bytes. The date field format is:

```
MM/DD/YY,

MM is the month (ASCII)
DD is the day   (ASCII)
YY is the year  (ASCII)
```

The time field is located at the base of the data page + 41D bytes. The time field format is:

```
HH:MM:SS,

HH is the hour   (ASCII)
MM is the minute (ASCII)
SS is the second (ASCII)
```

The slash, colon and comma are literal characters in both the time and date representation.

Licensed users are granted the right to include these modifications in CP/M-86 V1.1 software. CP/M-86 is a trademark of Digital Research.