SYBEX

MASTERING DISK OPERATIONS ON THE COMMODORE 128™

Alan R. Miller

CP/M Plus

SYBEX

Mastering Disk Operations on the Commodore 128™

Alan R. Miller

Berkeley • Paris • Düsseldorf • London SYBEX®

Cover design by Thomas Ingalls + Associates
Cover photography by Casey Cartwright
Book design by Lorrie Fink Graphic Design

Library of Congress Card Number: 86-61483
ISBN 0-89588-357-0
Printed by Haddon Craftsmen
Manufactured in the United States of America
10 9 8 7 6 5 4 3 2 1

Acknowledgments

I am sincerely grateful to Valerie Robbins, editor of the manuscript, for guidance and helpful suggestions, and to Olivia Shinomoto and David Clark, word processing; Dawn Amsberry, typesetting; and Cheryl Vega and Adam Cardamon, proofreading.

Alan R. Miller
Socorro, New Mexico
May 1986

Contents

Introduction

The purpose of this book to teach you to master the disk operating system for your Commodore 128 computer—CP/M Plus. You do not need any prior knowledge of computer operation. However, if you already know something about CP/M, you can use this book to learn more. You can also use the book as a reference—all the CP/M Plus commands are summarized in Chapter 7.

Digital Research, the publisher of CP/M, has steadily improved the system from its introduction as version 1.3. The latest version, 3.0, is so different from earlier ones that Digital Research has added the word Plus to the name.

This book is designed to be read from the beginning, especially by those who are beginners to CP/M. However, more-experienced readers can use it for reference. For those who are using a computer for the first time, Chapter 1 includes a brief introduction to the components and operations of microcomputers. If you feel comfortable working with a microcomputer and are ready to start learning about CP/M Plus, you can skip this chapter.

Chapter 2 introduces you to the CP/M Plus operating system and some of its basic functions. Many examples will enable you to enter simple commands and edit them; format and duplicate disks; and create, display, and print a file. By the end of the chapter you should have a feeling for one of CP/M's most useful programs, PIP, and also the various control characters you will use.

Chapter 3 explains in some detail the six built-in commands of CP/M Plus—TYPE, USER, DIR, DIRSYS, RENAME, and ERASE. It also discusses five transient programs—SETDEF, SHOW, SET, SUBMIT, and DEVICE.

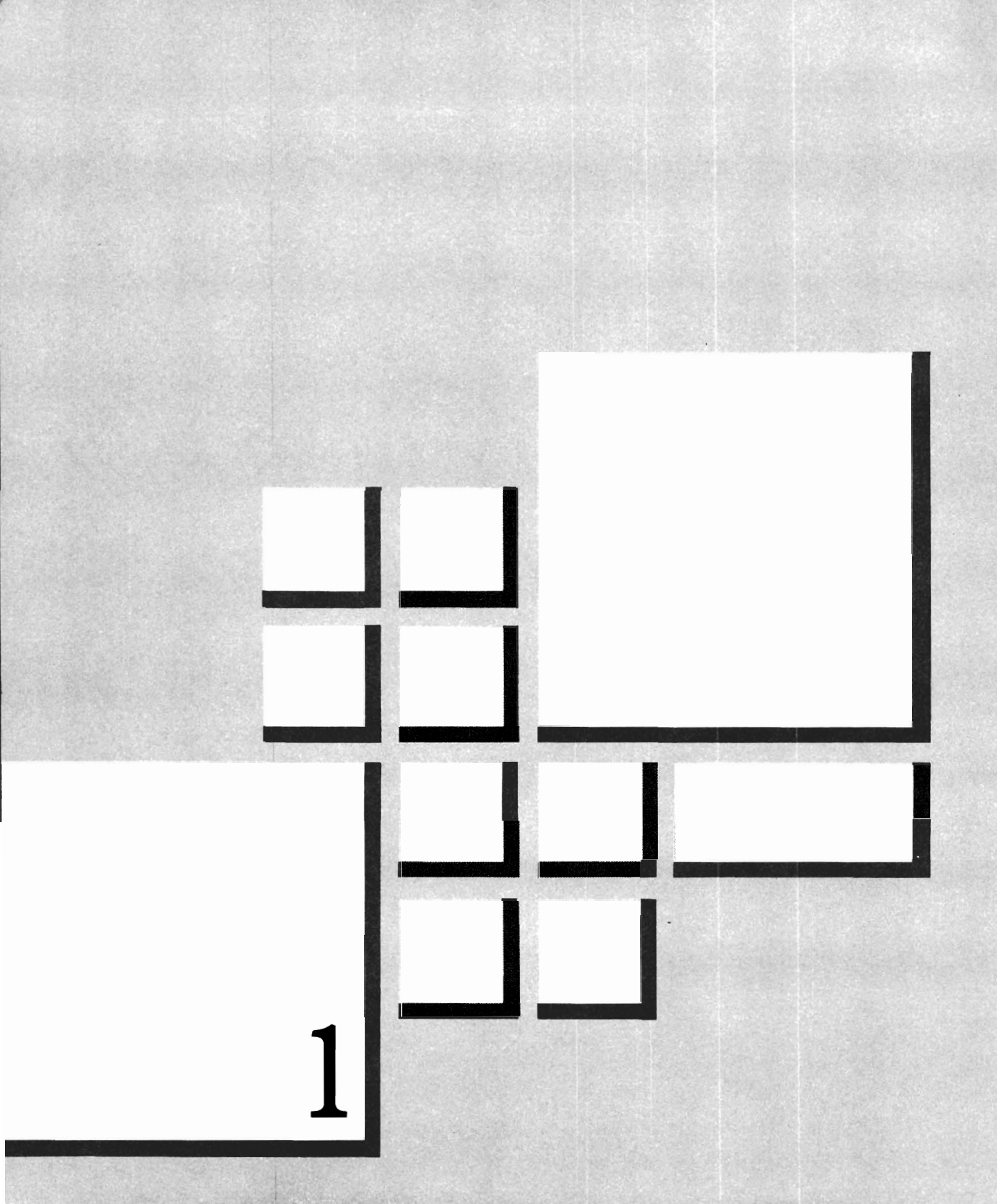
Chapter 4 is entirely devoted to PIP, a program for duplicating and altering files. Chapter 5 shows how to use the CP/M editor, ED, for creating and altering your own text files.

The first five chapters give you all the information you need to use CP/M Plus effectively. If you want to know more about how the system works, Chapter 6 provides an introduction to the inner workings of CP/M Plus. For more information on the internal workings of CP/M and how to add useful features, see my *Mastering CP/M* (SYBEX, 1983).

Chapter 7 summarizes all the CP/M commands and programs in alphabetical order. Thus, it provides a quick reference when you need details about a particular operation.

Three appendices present additional reference material. Appendix A is a collection of practical hints for using and caring for your Commodore 128, and more tips for working with files on disk. Appendix B lists and explains the CP/M control characters, and Appendix C is a table of the ASCII character set.

With CP/M Plus and the information in this book you will be able to run many useful programs on your Commodore 128. Furthermore, your Commodore 128 can run programs sold for both Osborne Executive and KAYPRO 10 computers. Thus a wide range of programs is available to you. These include Microsoft BASIC and FORTRAN, WordStar, SuperCalc (a spreadsheet), Spellguard (a spelling checker), and Grammatik (a syntax checker). Have fun!


1

Getting Started

This chapter introduces you to the operation of your Commodore 128 computer. You do not need any previous experience with computers to understand it. We will define the essential terminology and describe the basic procedures.

If you have used a microcomputer before and are familiar with the various components and operations, you should skip this chapter and begin with Chapter 2 or Chapter 3, depending on your level of experience. If you're new to computers, this chapter will give you enough information to begin working with your Commodore 128.

The purpose of a computer is to manipulate information, whether it is a huge NASA *mainframe* computer calculating the path of a rocket or a *microcomputer* organizing a mailing list. This book is concerned with the operation of the Commodore 128—a small, self-sufficient microcomputer.

A computer has two different elements that work together. The *hardware* comprises the physical parts of the computer: the floppy disks and disk drives, the video screen, the keyboard, the printer, and any additional parts. The *software* is the intangible aspect of the computer—the programs and data that make it work. A computer needs both hardware and software.

The Hardware

Computer hardware of the Commodore 128 is contained in several units. The main *system unit,* shown in Figure 1.1, houses the keyboard, the central processing unit (CPU), the main memory, and the input and output (I/O) *ports* that are used for communication with the peripherals. Each peripheral is attached to the system unit through a port.

The system unit performs all calculations and operates the other units. The separate video screen and disk drive are connected to the system unit through cables. A transformer, plugged into a wall outlet, is the source of electricity. It too is connected to the system unit. Hardware, such as a printer, a modem, and a tape recorder, may also be attached. The disk drive, video screen, printer, and any other hardware you have attached to your computer (a modem, a mouse) are called *peripherals* since they are not physically part of the system unit. The Commodore 128 is shown with four peripherals in Figure 1.2. Let us briefly look at each component.

The Central Processing Unit

The hardware element that directs all the computer's operations is called the *central processing unit (CPU).* For a microcomputer such as the Commodore 128, the CPU is a single integrated circuit, containing thousands of transistors on a silicon

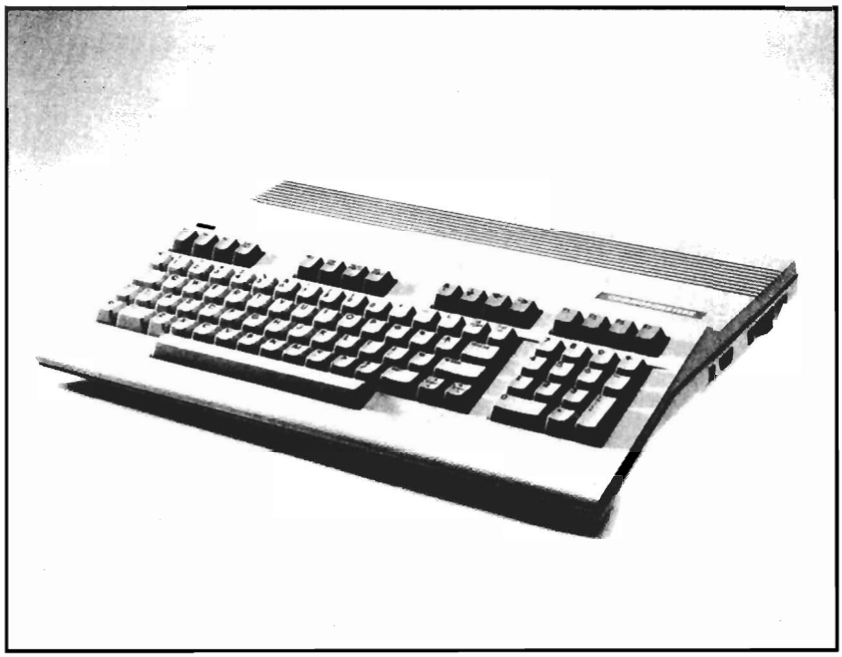


Figure 1.1: *The Commodore 128 System Unit*

chip. The operation of the CPU is controlled by programs stored in the computer's *main memory.*

The Commodore 128 is like three computers in one. If turned on without a disk drive, it begins in Commodore-128 mode. Then you can run BASIC programs. It is also possible to run in a Commodore-64 emulation mode. However, both of these modes are extremely limited. By contrast, a wide variety of programs, in such diverse fields as business, engineering, and games, is available when the Commodore 128 is operated in CP/M mode. Consequently, in this book we shall only be interested in the CP/M mode where the regular Commodore CPU operates as a peripheral processing unit and another CPU, a Z80, executes CP/M programs.

The Main Memory

The purpose of the computer's memory is to store information. This includes the program the computer is executing and

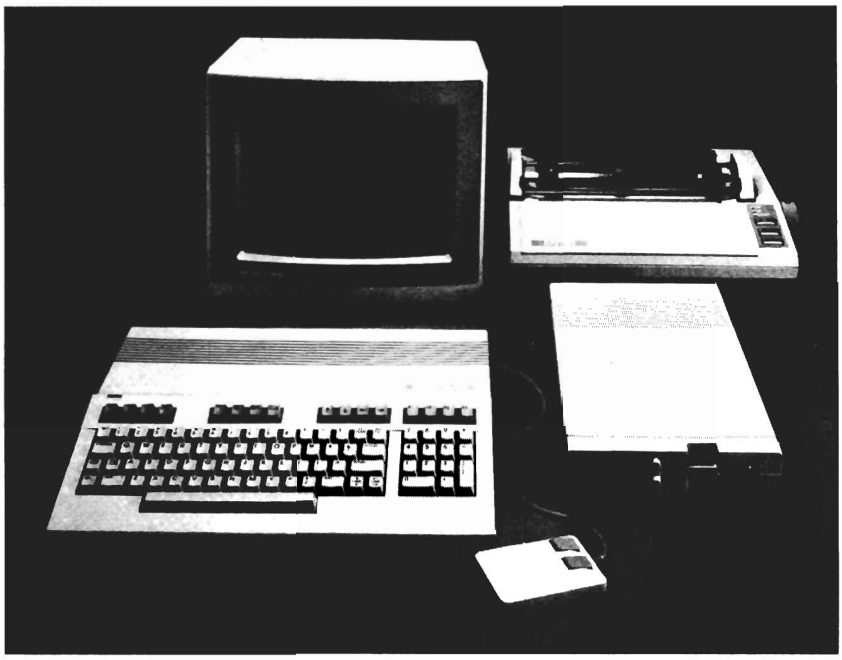


Figure 1.2: *The Commodore 128 and Peripherals*

the data it is processing. The memory size is measured in bytes or kilobytes. A *byte* represents a single character, such as the letter B. A *kilobyte,* often abbreviated to K byte or KB, is approximately a thousand bytes. The Commodore 128, as its name implies, has 128K bytes of memory, although the memory can be increased to 512K bytes. The computer's memory is often called *RAM,* which stands for random access memory.

The Keyboard

The keyboard is one of the primary means by which the user interacts with the computer. You enter information by typing on the keyboard, and the computer places your information into its memory.

The Commodore 128 keyboard has a number of special keys in addition to the regular *alphanumeric* (typewriter) keys. If you follow along with the practice techniques in this book, you will

be trying out the CONTROL key and learning some of the functions for which it can be used. Some applications programs use the CONTROL key extensively. Many applications programs also make good use of the other special keys. These are the four arrow keys (labeled ↑, ↓, ←, and →) on the top row of the keyboard; the two cursor keys (labeled CRSR) on the bottom right; and the function keys (labeled F1/F2, F3/F4, F5/F6, and F7/F8) on the top right.

Some of the arrow keys, cursor keys, and function keys can be used for giving CP/M commands. If you are interested, you will no doubt experiment with them to see what they do. However, in my experience, none of these keys offer any advantages over the alphanumeric keys if you are a reasonably competent typist, because it takes more time to locate and press a special key than to keep your hands over the typewriter keys and press them without having to look at the keyboard. If you are not such a good typist, you may not be familiar with the keyboard. Then you may also find it more difficult to locate the special keys. Therefore, we do not suggest special-key equivalents for typing the commands presented in this book.

The Video Screen

The video screen is the other primary means by which the user interacts with the computer. The computer uses the video screen to display both your entries and the results of its computations. The video screen is sometimes called a *monitor.* Three types of video screen can be attached to the Commodore 128—a regular television set, a composite monitor, or an RGB monitor. (RGB stands for red-green-blue, the three primary colors used in this type of monitor.) The composite and RGB monitors are compatible with those used on the IBM PC. That is, you can plug an IBM composite monitor into the round connector on the back edge of the system unit, or an IBM RGB monitor into the D-shaped connector also on the back edge of the system unit.

You could, of course, use a printer instead of a video screen. However, a video screen is more convenient since it can display information much more rapidly. The disadvantage of the video

screen is that there is no permanent record of the information that has been displayed. When new information appears on the video screen, the previously displayed information is lost. For this reason a computer needs a printer to produce a permanent record.

The Printer

The printer provides a permanent *printout* of information, also known as a *hard copy*. The information can be the result of computations, the listing of a program, or simply a record of the information that you entered at the keyboard.

The Commodore 128 is set up to run with a dot-matrix printer. This kind of printer uses small dots to make up the printed characters. However, if you will be needing a higher quality of printout, talk to your Commodore dealer about buying a *letter quality* printer, which prints hard copy that looks like a good-quality typewriter printout. To use such a printer, you will also need to buy an adaptor that plugs into the printer port on the Commodore system unit.

The Disk Drive

In the Commodore 128, most of the computer's memory is volatile—that is, the information it contains is lost each time the computer is turned off. In addition, the memory size is limited. Therefore, the programs and data must be stored on a more permanent medium such as a floppy disk. When the computer needs to use the stored information, it copies or *loads* the information from the disk into the main memory. This doesn't mean the information is removed from the disk. In effect, the computer just memorizes it temporarily. The original version is still present on the disk.

The Commodore 1571 disk drive, shown in Figure 1.3, contains a read/write head, a drive motor to rotate the disk, and another motor to position the head. The computer readily stores and retrieves information on the disk surface through the read/write head.

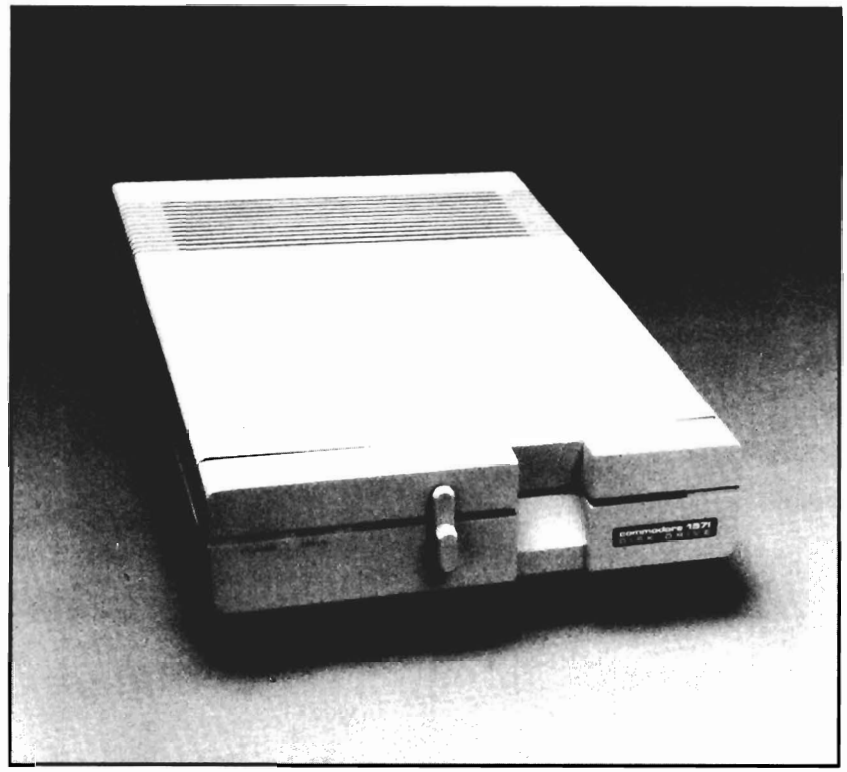


Figure 1.3: *The Commodore 1571 Disk Drive*

The chances are that you have only one disk drive attached to your Commodore. However, when you have worked with your Commodore for a while you may find that you can speed up your work significantly by attaching a second disk drive. In Chapter 2, instructions are given for copying disks for those with two drives as well as for those with only one. Later chapters assume that you have only one drive but the instructions are easy to interpret for a two-drive system.

The Surge Suppressor

There is another important piece of hardware you should have for successful computer operation—a surge suppressor. The electricity supplied to your Commodore 128 should be 120-volt,

alternating current with a frequency of 60 hertz (cycles/second). However, sometimes there are higher frequencies generated by other appliances. These higher frequencies will not harm your Commodore 128. However, they may cause errors in the recording of information on disk.

A more serious problem is a voltage spike caused by lightning or by turning a heavy appliance on or off. This can subject your Commodore 128 to a voltage much higher than the designed value and it can burn out components of your computer.

Both of these problems—high frequency or a voltage spike—can be prevented with a surge suppressor. You plug the surge suppressor into the electrical outlet and then you plug your Commodore 128 into the surge suppressor. There are many variations of this device. Some cost less than ten dollars and others are nearer $100. However, it is possible to obtain a very good one for around $50 from electronic supply stores.

The Mouse

You may have purchased a *mouse* to use with your Commodore 128. The mouse is used to quickly move a pointer around the screen and to select items presented on the screen. You can take advantage of this device only if you are running an applications program that has been especially written for use with a mouse. You will not be using a mouse for any of the CP/M functions in this book.

The Software

A computer needs a set of instructions, called a *program,* to make it operate properly. Once installed in the computer's memory, the program will direct the computer to perform specific operations. Computer programs, as well as any data needed by the programs, are called *software.* Software can be divided into three basic categories—system software, applications software, and data. Let us look at each of these.

The *system software* operates the computer and so it is known as an *operating system.* If the operating system extensively uses a

disk in its work, it is known as a *disk-operating system* or DOS for short. (DOS rhymes with boss.)

An operating system, then, is a computer program that manages the resources of the computer. The operating system reads commands from the keyboard, displays information on the video screen and printer, and executes applications programs for the user. In addition, the operating system will perform chores such as managing the disk space and the main memory space of the computer. CP/M Plus is a very useful operating system because there is such a large amount of applications software available to use with it.

The CP/M software supplied with the Commodore 128 includes auxiliary system programs such as FORMAT, which prepares new disks for use, and PIP, which makes backup copies of disks. However, several very important auxiliary programs are not included with your CP/M disk. Fortunately, for a nominal fee you can obtain from Commodore two utility disks with the missing CP/M programs. To obtain the two CP/M utility disks send $19.95 to Commodore Direct Marketing, DRI Offer, C-2651, West Chester, PA 19380.

You use *applications* software to perform specific tasks. Examples are a word processor such as WordStar, a spreadsheet program such as SuperCalc, a database manager such as dBASE, a mailing list program, an inventory program, and a general ledger program. Figure 1.4 shows an applications program displayed on the video screen.

Collections of characters and numbers—for example, the names and addresses for a mailing list—that are manipulated by programs are known as *data*. Both data and programs are stored on a floppy disk in logical groupings known as *files*. Files are assigned unique names for easy reference. In the next chapter you will learn how to create and manipulate data and files.

Floppy Disks

As mentioned earlier, microcomputers use magnetic disks to store information. The Commodore 128 uses the common 5-inch, double-sided, double-density floppy disk. The capacity of

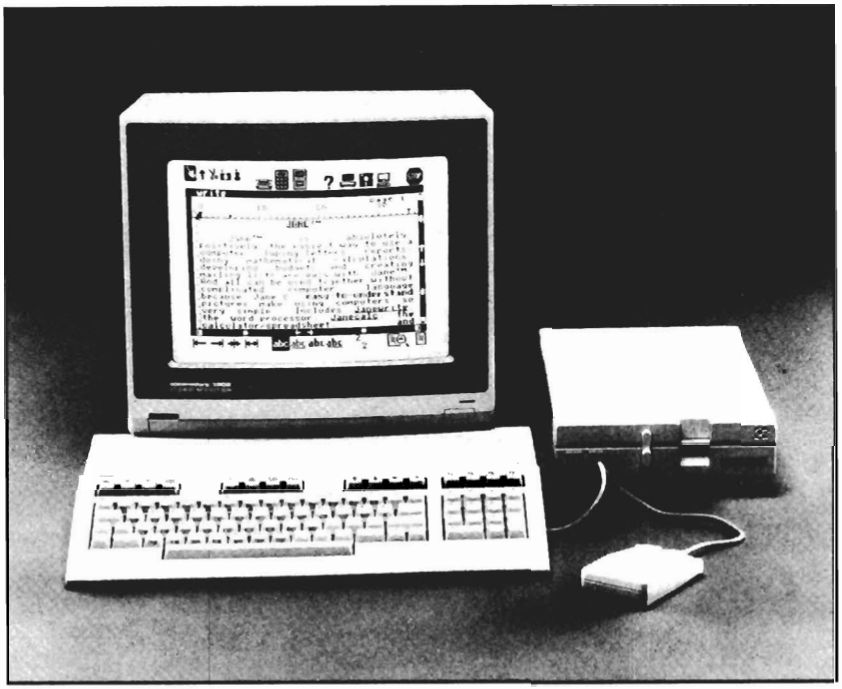


Figure 1.4: *Using Applications Software on the Commodore 128*

the Commodore disk is about 400 kilobytes (400,000 bytes). Since one letter of the alphabet can be stored in each byte, it is possible to record an entire book on one floppy disk. This section describes how floppy disks work and explains how to use them. Further tips for using disks are included in Appendix A.

A disk is made of mylar (a kind of plastic) coated with a magnetic oxide material. A square outer envelope lined with a soft material protects the flexible disk within. When the disk is in use, it rotates inside its envelope. The drive spindle hole allows the disk drive motor to rotate the disk. The head slot in the envelope (shown in Figure 1.5) allows the read/write head to contact the disk surface and to read or write information on it. The recording and playback is very similar to the system used for audio recording with magnetic tape.

An index hole in the disk surface is used to determine the orientation of the disk. The rotating disk interrupts a light beam projected through the index hole and the corresponding holes in

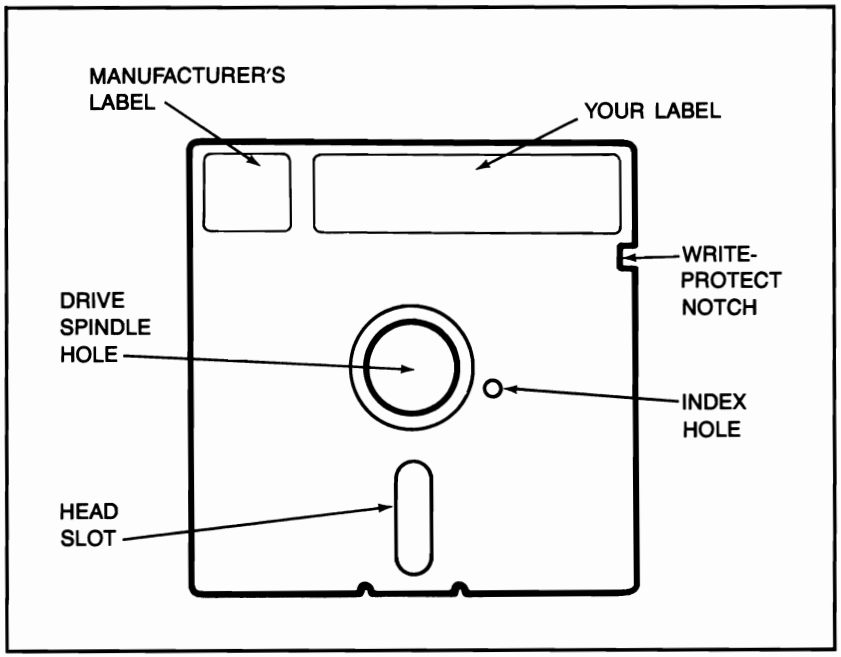


Figure 1.5: *Floppy Disk for the Commodore 128*

the disk envelope. In this way, the computer can determine the exact position of the disk.

Information is recorded on the disk surface along concentric circles called *tracks,* which are partitioned into smaller regions called *sectors* (see Figure 1.6). Disks are made in two ways, known as hard sectored and soft sectored. In addition, they will be single or double sided and single, double, or quad density. You do not need to understand the technical details of disks in order to use them. However, it is important to use the correct type. The Commodore 128 uses soft-sectored, double-sided, double-density disks. Quad density disks can also be used, but they are more expensive.

Write-Protection

Disks used with the Commodore 128 computer are equipped with a *write-protect* notch (see Figure 1.5). When this notch is covered with a piece of tape, the disk drive can no longer

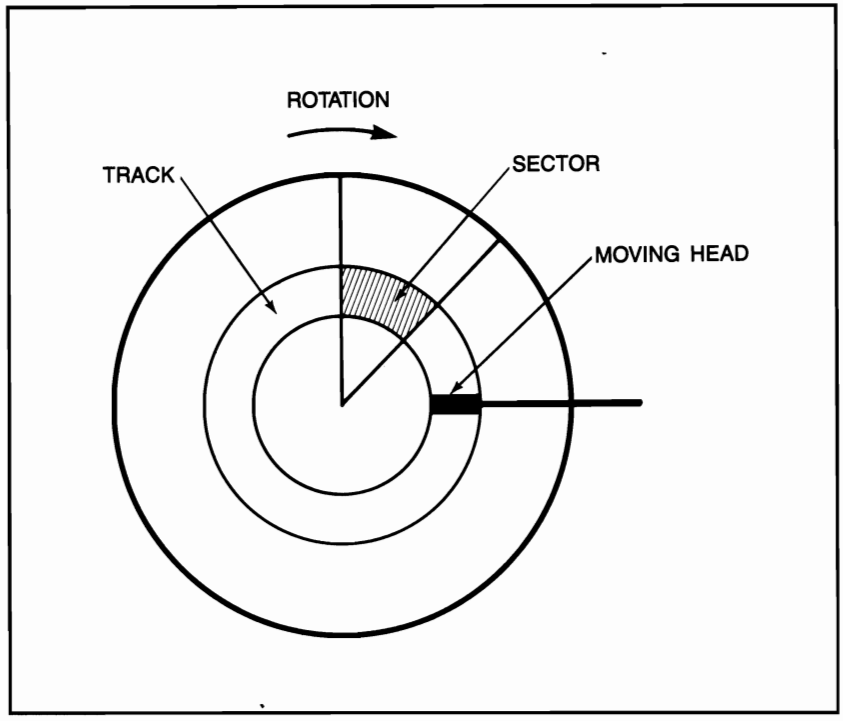


Figure 1.6: *Tracks and Sectors of a Disk Surface*

record, or *write,* on the disk. However, when the notch is exposed, that is, when the tape is not in place, the disk can be written on. Write-protection tape safeguards important information. For example, your system disks and backup disks, which are not used regularly, should be write protected. Of course you must not cover a disk's write-protection notch when you want to create or alter information on it.

The aluminum tape for write-protection is provided with boxes of new disks. Do not use ordinary tape. It is important to understand that write-protecting a disk does not prevent you from using the programs or data on the disk. Getting information from a disk is called *reading* from the disk.

Handling Disks

Floppy disks are delicate and can be easily damaged. Always treat them with care. Put each disk in its protective sleeve and

place it into a plastic box when not in use. You can grasp a disk by its outer cover, but do not touch the exposed magnetic surfaces. Be especially careful to stay away from the slotted opening for the magnetic head. The grease from your fingers can ruin both the disk and the drive head that contacts the disk surface. Do not expose a disk to dust, smoke, or liquids. It is very important to keep the disk away from magnetic fields or metal that might be magnetized. Do not write on the disk with a ball pen or a pencil because the impression of the writing instrument can damage the underlying surface. Mark the disk only with a felt-tipped pen or write on a separate label and then affix the label to the disk. Do not use tape. Read Appendix A for more suggestions on handling disks. You can also use that appendix for reminders when you are more experienced.

Inserting a Disk in the Disk Drive

A disk can be inserted into the drive eight different ways. However only one way is correct. It is important to insert disks correctly into your drive. Here, the "rule of thumb" applies—hold the disk with your thumb on the label as you insert it into the drive; that is, the label is on the upper side and goes in last. The write-protect notch is on the left side. When the disk is in position, turn the drive handle downward to close the drive and position the magnetic heads. If you are practicing for the first time, use a brand new disk rather than a disk with important information on it.

A disk should never be removed from or inserted into the disk drive when the computer is reading from or writing to the disk. Furthermore, never shut off your Commodore when it is reading from or writing to a disk. There are two indicator lights on the front of the Commodore disk drive. The red light shows that the drive is turned on while the green light means that information is being read from or written to the disk. To be safe, always check to be sure that the green light is off before removing or inserting a disk. It is good practice to remove the disk before turning off the computer even though it is not absolutely necessary.

Now that you have learned some of the basic concepts of computer operation and how to handle floppy disks, it is time to begin using your Commodore 128.

Starting Up Your Commodore 128

To begin, check the 40/80 DISPLAY key on the top row of the keyboard. It should be in its down position if you have an RGB monitor; otherwise this button should be up. (Each time you press this key, it changes from the up or down position.) Next turn on the disk drive with the switch at the rear. Insert the CP/M disk and close the handle. Turn on the video screen and then turn on the system unit with the switch on the right side. Before you can use the Commodore 128, the operating system must be loaded from disk into the main memory by a special program called a *bootstrap loader.* Each time the computer is turned on the bootstrap loader automatically copies the CP/M operating system into the memory. This step is known as a *cold start,* or *booting* the system.

Once CP/M is loaded into memory, the bootstrap loader gives control to CP/M. CP/M then writes the symbols

```
A>
```

on the video screen and waits for you to type a command. Try typing something such as

```
HELLO THERE
```

Be sure to press the space bar between the first word HELLO and the second word THERE. Notice that each time you type a character, it is displayed on the video screen. CP/M has read each character you typed, and then analyzed it. When you type an ordinary letter or number, CP/M immediately displays it on the video screen. If you type nothing more at this point, nothing else will happen. CP/M has analyzed what you typed, however, it is still waiting for more instruction. Therefore you must always remember to press the key marked RETURN when you have finished typing a command. Otherwise, CP/M will not know you have completed the command and therefore, it will not act on it.

Press the RETURN key now, telling CP/M to carry out your command. CP/M, however, writes

```
HELLO?
```

on the next line, then moves on to a new line and displays the symbols that we saw previously:

```
A>
```

CP/M is programmed to execute many commands, so it tries to interpret your first word, HELLO, as a command. However, HELLO is not one of the commands CP/M is programmed for. Therefore, it repeats your first word and adds a question mark at the end. This is an *error message* telling you that it cannot carry out your command.

CP/M then displays the A> symbol and waits for your next command. It will not go on until you type a proper command. You cannot hurt the hardware by typing at the terminal. You might, however, alter one of the programs stored on the disk if you keep typing without knowing what you are doing.

Turning Off Your Commodore 128

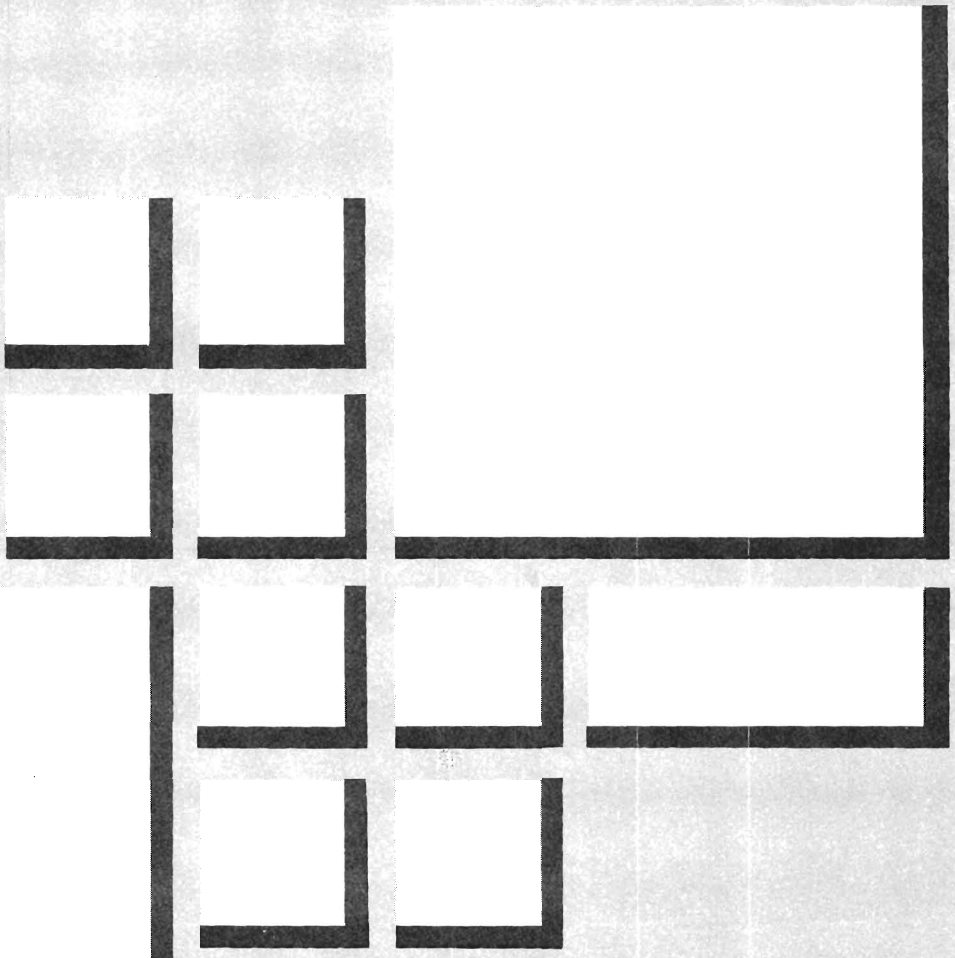
You do not turn off your automobile engine while waiting for a red light to turn green. In the same way, it is a good idea not to turn off your computer each time you have finished a task. Rather, it is usually best to turn on your computer at the beginning of the day and turn it off at the end of the day. On the other hand, if you expect electrical interruptions because of lightning, for example, then it is best to turn the computer off when it is not in use. Therefore, if you are ready to continue with the next chapter, leave your computer running. However, if you are not going to the next chapter, turn off your computer.

When you are ready to turn off your computer, make sure that CP/M is not using the disk. Otherwise, you can damage the disk and so lose the information stored on it. Look for the green activity light on the front of the disk drive; the word DRIVE is under the light. When this light is on, it means that CP/M is trying to read from or write to the disk. When the green light is off, unlock the drive by turning the handle. Remove the floppy disk, place it into its sleeve, and put it away in a box. Then turn off the computer and its peripherals.

Summary

This chapter introduced the Commodore 128 computer with its CP/M Plus operating system. We described the basic hardware elements: the system unit with its keyboard, central processing unit, I/O ports, and main memory; the video screen or monitor; the printer; and the disks and disk drive. We briefly explained the operating system and other software the computer uses to perform its tasks. The next section described floppy disks and gave some hints for using them. Then you learned how to turn the computer on, give a command, and interpret an error message. Finally, you learned how to turn the computer off. Now you have mastered the fundamentals of using your Commodore 128.

In the next chapter you will learn how to use some of the features of CP/M; how to duplicate your system disk and how to examine information stored on disk.



2

Learning to Use CP/M Plus

Now that you have been introduced to the Commodore 128 you can begin learning about the CP/M Plus operating system that runs on the Commodore 128. This chapter will introduce you to some of the basic features and commands of CP/M. We will explore some of these commands in more detail in later chapters. By the end of this chapter, you will know enough about CP/M to run applications programs such as SuperCalc and WordStar on your Commodore.

Each computer system has its own set of hardware combined in particular ways. To run on a given computer, an applications program must be able to interact with that system. With so many different types of computers and peripherals available, it would be impractical to design a separate version of each program to fit each different computer. Therefore, each computer is fitted with an operating system that provides an interface between the computer hardware and the applications programs that are to be run on the computer. For a particular type of operating system, such as CP/M, the interface to the applications program is the same, regardless of the computer type. However, the hardware interface must be especially configured for each different computer.

CP/M Plus is also known as CP/M version 3. (CP/M is an acronym for Control Program for Microprocessors.) An applications program written for the CP/M Plus operating system will not work on a different operating system (such as MS-DOS). However, a program that runs on another CP/M computer such as a Kaypro 10 or Osborne Executive can also run on the Commodore 128.

Resident and Transient Commands

CP/M consists of several parts. As you learned in Chapter 1, the *resident* (or *built-in*) part of CP/M is copied from the system disk into memory each time you turn on your computer. Therefore, you must always have the system disk in the disk drive when you first turn on the Commodore. After the computer has started up, you can remove the system disk and substitute another because the resident portion of CP/M stays in memory. This part of CP/M reads the keyboard for your input, starts the programs you request, and writes the output to the video screen and printer. Thus the resident portion of CP/M becomes an integral part of the computer.

You can view the resident portion of CP/M as a servant always ready to obey your commands as long as you are not running a program. Then, CP/M gives control to the running program and all your dialogue is with that applications program. When the applications program finishes, the memory space it uses is

released. CP/M is activated again and is ready to accept new commands. Therefore, applications programs are also known as transient programs.

Another part of CP/M consists of separate utility programs that are also transient programs. They are located on one of the CP/M disks. Each is a separate entity that can be kept on any disk you want. When one of these programs is needed, it is executed by typing its name. The program is copied from disk to memory and then excuted. You can delete a transient program from the disk if you do not need it. Since the transient programs are only present in memory while they are being used, they do not reduce the memory size available to applications programs.

CP/M resident programs are also known as *built-in commands;* transient programs are also known as *transient commands.* You will find them distinguished in this way in Chapter 7.

In the following sections you can try out CP/M and begin to learn the basic commands.

Trying Out CP/M

Place the system disk into the drive and turn on your Commodore. The automatic bootstrap loader will copy the resident portion of CP/M into memory from the disk in the drive. When CP/M has successfully loaded, the video screen displays the symbol:

```
A>
```

The A> symbol is a *system prompt,* which CP/M displays to show that it is ready to accept your next command. The letter A also identifies the current or *default* disk drive—the drive you are currently working on. If you have only one disk drive, it is known as drive A. However, it can also appear to be drive E. (This will be explained later in this chapter.) It is also possible to attach additional drives to your Commodore 128. These drives are then known as B, C, and D.

A blinking square block is positioned immediately to the right of the prompt. This is the *cursor,* a symbol that shows where you

are on the screen and where the next character you type will appear.

Typing Entries on the Keyboard

CP/M will display its own messages on the screen as well as the characters you type on the keyboard. We have just seen that CP/M displays the A> symbol when you first turn on the computer. Then, if you press the keyboard letter keys, CP/M will also display the corresponding characters on the screen.

When an example in this book shows you what characters to type, we will usually show only the characters you are to type. We will not show additional characters that CP/M might display. For example, when it is time for you to enter a command, the computer will display the system prompt. If you enter the command DIR, the display will look like this:

```
A>DIR
```

However, in our examples, we will only show the characters that you are to type:

```
DIR
```

Whenever the computer displays the system prompt, you can safely enter a command. After you type a command at the keyboard, you must tell the computer that the line is complete and ready to be processed, or *executed*. The RETURN key is used for this purpose. We will use <RETURN> to remind you to press the RETURN key at the end of the line. When you press this key, no symbol is displayed on the video screen. CP/M automatically advances to the beginning of the next line by sending both a carriage-return character and a line-feed character to your screen.

Using the RETURN and CONTROL Keys

Let us gain more familiarity with the CP/M system by giving some simple commands. Be sure that the cursor is next to the system prompt. Then, press only the RETURN key and notice

that the system prompt appears again on the next line. Do this several times and notice how the computer simply repeats the prompt on the next line. We have seen that the RETURN key is used to send a command to the CP/M. In this instance, no command was actually given, so CP/M simply repeats the prompt. We have also seen that it is necessary to press the RETURN key at the end of each *command line*. This tells CP/M to execute the line. However, as we have seen, there is an exception.

Certain commands are given by pressing two keys at once—the key marked CONTROL and a specific letter key. This combination sends a symbol known as a *control character*. In this book, control characters are represented by the symbol ^ followed by the corresponding letter because the caret (^) is the standard symbol for the CONTROL key. However, most of the time control-character symbols do not appear on the video screen at all. On the Commodore 128, when control characters do appear, CONTROL-C, for example, will appear as ↑C. Of course you do not use the white key marked ↑ to send a control character.

Usually you do not need to press the RETURN key after a control character has been typed. Rather, CP/M begins executing the command as soon as the pair of keys is pressed. There is one caution: when you give the ^C command, the cursor must be at the beginning of the line or CP/M will not interpret it as a command. (You can move the cursor to the beginning of the line with ^B. You can also move the cursor to the beginning of the line with ^X, although this will erase the line.)

Press the space bar to move the cursor one position to the right. Now hold down the CONTROL key and press the letter C. You will see the ↑C symbol on the screen. (Press CONTROL and the X key to cancel this line.) We will discuss the control characters in more detail in the next section.

Now type the line:

```
ANYTHING <RETURN>
```

Since this is not a valid command, CP/M responds with the error message

```
ANYTHING?
```

and then displays the system prompt again. The computer will not proceed until you enter a valid command.

What's Wrong?

If the CP/M prompt is displayed but your system fails to respond to keyboard entries, then something is wrong. Sometimes you can restart the computer by giving the ^C command. Alternatively, you may have to place the system disk in the drive and press the reset button (on the right edge of the keyboard). If that does not work, turn off the computer and turn it back on again.

Sometimes something unusual happens, and you do not know what caused it. There are various things you can try. If you press the RETURN key at the end of a command line and get no response from the computer, the computer may be busy doing something, such as running a program. You may have inadvertently typed the name of a program that exists, and CP/M is executing that program. Sometimes you can leave (*abort*) the program and return to the CP/M system by typing ^C, but not always.

If typing ^C has no effect, look on the front of the disk drive. Check to see if the green activity light, with the word DRIVE under it, is lighted. If the green light is on, it means that your computer is trying to read from or write to the disk in that drive. If the light is on and there is no disk in that drive, then you must start from the beginning. If there is a system disk in drive A, press the reset button on the right edge of the keyboard to perform a cold boot. If there is no disk in drive A, switch off the computer. Then turn on the computer by going through the regular startup procedure.

Control Characters

As we have already mentioned, certain symbols, known as *control characters,* have special meaning to CP/M. To type a control character, hold down the CONTROL key and press a letter

key. Remember that you do not press the RETURN key after typing a control character. It does not matter whether the shift lock is engaged when you enter a control character. In other words, a lowercase ^c is the same as an uppercase ^C.

While we represent a control character by two symbols, the ^ character and the following character, the keyboard transmits a single signal. The set of control characters includes the letters (^A through ^Z) and other characters such as ^@ and ^[. However, not all these control characters are used by CP/M. (The CP/M control characters are listed in Appendix B.)

Several keyboard keys perform the functions of frequently used control characters. For example, the RETURN key, which tells CP/M that the command line is ready to be executed, transmits the ^M character. Obviously, pressing the RETURN key is easier than pressing both the CONTROL key and the M key. Of course, the result is the same. As a second example, notice the key in the upper-left corner labeled ESC. This is known as the escape key. Pressing this key transmits a ^[character. CP/M does not use this character. However, some applications programs, such as WordStar, make extensive use of this character. A third example is the LINE FEED key. Pressing this key sends the ^J character which is interpreted as a line feed.

The backspace key found on many keyboards is not present on the Commodore 128 keyboard. You must enter the equivalent ^H character or use the Del key.

CONTROL-C (^C)

Giving the command ^C (CONTROL-C) produces what is called a *warm start,* or a *warm boot* (in contrast to the cold start, when you start up the system for the first time with a bootstrap load or press the reset button). The ^C command interrupts whatever the computer is doing, starts the operating system over again, and displays the system prompt. As we have seen, the cursor must be at the beginning of the line for ^C to be interpreted as a warm-start command. Also, many applications and system programs can be interrupted with the ^C command. Note, however, that not all applications programs can be stopped in this way.

The warm boot is used for two purposes. First of all, as we saw earlier, you may be able to use it to restart the system when it is not responding to your commands. It can also be used when you change a disk. This forces CP/M to read the directory of the new disk.

Using Control Characters to Edit a Line

You may inadvertently press a wrong key or want to change something in the line you have just typed. You can use some of the control characters to make changes as long as you haven't pressed the RETURN key yet. Table 2.1 provides a list of the control characters used for editing a CP/M command. In this section, we will examine them one by one.

Command	Action
^A	Move cursor one character left
^B	Move cursor to beginning of line (or to end of line if cursor is already at beginning)
^E	Move cursor to next line (used for long lines)
^F	Move cursor one character right
^G	Delete character at cursor position
^H	Delete character to left of cursor
^I	Move cursor to next tab position
^J	Execute command (line feed)
^K	Delete from cursor to end of line
^M	Execute command (carriage return)
^R	Redisplay line
^U	Delete all characters in line
^W	Recall previous command line
^X	Delete all characters to left of cursor

Table 2.1: *Control Characters Used for Editing*

Type the word ANYTHING but do not press the RETURN key. (You can use either uppercase or lowercase letters on the command line. Both are interpreted as uppercase.) While holding down the CONTROL key, press the H key. This is the ^H, or backspace, character. Notice that each time you give the ^H command, the cursor backs up on the video screen, erasing the previous character. Alternatively, you can use ^G to delete the character at the cursor position without disturbing the remainder of the line.

You can use ^H to delete one or more characters. However, sometimes you might want to delete more than one character in a row. You can delete all the characters to the right of the cursor with ^K (for kill) and all the characters to the left of the cursor with ^X. For example, type the word ANYTHING again but do not press the RETURN key. Now type ^X. That is, while holding down the CONTROL key, press the X key. Notice that the cursor moves all the way back to the beginning of the line, erasing the line.

You can erase the complete line no matter where the cursor is positioned in the line by giving the ^U command. Type the word ANYTHING yet another time and then give the ^U command. CP/M displays the number sign (#) and then moves the cursor to the beginning of the *next* line. The system prompt is not displayed on this line. The ^U command tells CP/M to ignore the line, although you can still read it on the video screen. The screen looks like this:

```
A>ANYTHING#       (^U given)
                  (ready for next command)
```

Typing ^W after ^U restores the canceled line, and CP/M redisplays this line as though you had just typed it again. (Do it.) You can then edit the command line by using one of the other control characters, or you can execute the line without change simply by pressing the RETURN key.

Using control characters, you can move the cursor back and forth through the command line without altering the existing characters. The ^A command moves the cursor one character to the left, and ^F moves it one character to the right. You can move the cursor directly to the beginning of the line with ^B. If the cursor is

already at the beginning of the line, ^B moves the cursor to the end of the line. That is, you can alternately move the cursor from one end of a line to the other with the ^B command.

You can insert additional characters wherever the cursor is positioned simply by typing them. The remaining characters on the line are automatically shifted to the right.

You are not likely to use ^E and ^R very often. The ^E command moves the cursor down to the beginning of the next line so you can continue typing on the same command line. It is useful for a very long command line. The ^R command redisplays the command line on the next line.

Other Control Characters

You may sometimes want to stop a moving video display at a particular point. For example, you may need to keep the information from running off the screen before you have had a chance to read it. This is especially true for programs that do not stop when the screen has been filled with text. Then, the information moves out of sight as it *scrolls* off the top of the screen.

CP/M Plus provides a means of stopping (*freezing*) the screen at any time. Press ^S to stop the display momentarily. Then press ^Q when you want the scrolling to continue. Usually, you can interrupt the scrolling prematurely by giving the ^C command. However, it may be necessary to freeze the screen with ^S first before terminating with ^C. Of course, the screen must be scrolling for the ^S command to take effect.

If you want your printer to produce a listing or hard copy of what is to appear on the video screen, you can press ^P to engage the printer and everything that appears on the screen after that will also be sent to the printer. If you want a listing of only the middle portion of the output, you can freeze the screen with the ^S command, then press ^P to engage the printer so you get a listing of the new information shown on the screen. Finally, you can resume scrolling with the ^Q command. We will discuss printouts later in this chapter.

Two other control characters—the tab key, ^I, and the line feed, ^J—are not required for the operation of CP/M. However, they are frequently used with system and applications programs.

The Commodore 128 also provides separate keys for both of these functions—TAB and LINE FEED.

Finally, ^Z marks the end of strings in certain CP/M commands. CP/M also automatically places this character at the end of each text file. Therefore, ^Z cannot occur in a text file. (Strings and text files will be discussed in later chapters.)

Now that you are familiar with some CP/M operations, we will present different ways to duplicate a disk and make a backup copy of the system disk.

The System Disk

Your system disk is your most important disk because it contains the CP/M operating system. Remember that your system disk is especially designed for your particular computer, so it will not work in a computer with different hardware. (However, applications programs, such as WordStar, that run on the Commodore 128 can also run on other computers such as the Kaypro 10 and the Osborne Executive.)

If you change any of the hardware of your system, such as the printer, the video screen, or the memory size, the operating system on your current system disk may not work. The steps needed to change CP/M are described in my *Mastering CP/M* (SYBEX, 1983). If you do make changes in CP/M, be sure to label the different system disks so they will not get mixed up.

In the next sections you will learn how to duplicate a disk. Because your system disk is so essential, you should make a backup copy right away. You can back it up on a new disk or use an old disk that you no longer need. In addition, you should carefully label the supplemental CP/M disks and make backup copies of both of them too. A new disk must be properly formatted before it is used for the first time, so in the next section you will also learn how to format a disk.

Duplicating Disks

When you store important information on a disk you must be careful that the information is not lost. A disk can be damaged in

several ways (see Appendix A). Therefore, it is prudent to make duplicate (*backup*) copies of all important disks.

A system disk is duplicated in several steps. First, the disk must be formatted by writing a special pattern on the surface. Date and time stamping can also be enabled. (This step is discussed later.) Then the CP/M system routines (CPM+.SYS and CCP-.COM) and the other programs are copied from the original disk.

Before we see how to format a disk, let us consider the magnetic pattern that is written on the surface by a disk drive. As we saw in the previous chapter, the disk surface is divided into concentric rings called tracks. Each track is further subdivided into regions known as sectors. Each sector and track is assigned a unique number. When a disk is formatted, the corresponding sector number is magnetically written at the beginning of each sector.

Information is stored on a disk using a collection of sectors known as a *block*. Different computers use different numbers of tracks and sectors; that is, the format for disks used on one computer is different from the format used on other computers. Since the number of tracks and the number of sectors changes from one computer to another, the amount of information that can be stored on a disk will also vary from one computer to another. Furthermore, one computer may use several different formats. For example, one format may use only one side of the disk while another format may use both sides. These two forms are known respectively as single sided and double sided.

Another disk variable is density. Disks may be single density, double density, quad density, or high density. The disks you use for your Commodore 128 should be marked as double sided and double density. The symbol 2S/2D might be marked on the disk label. Even though the Commodore 128 normally uses a double-sided, double-density format, it can read and write disks of other formats. For example, floppy disks created on the Osborne Executive (and other Osborne computers, if double density) can be read by the Commodore 128 even though these disks are one sided. Floppy disks formatted for the KayPro 10 computer can also be read on the Commodore 128. (However, it cannot read disks created on the KayPro II computer.)

Because the Commodore 128 can read disks formatted for other computers, it is possible to trade programs with friends

who have an Osborne or KayPro computer. You can readily read these programs or copy them to a disk formatted on the Commodore 128. However, since those with an Osborne or Kaypro may not be able to read your format, you can give them copies of your programs if they supply you with a disk formatted on their computer.

Let us see how to format a disk.

Formatting a Disk

Before a new disk can be used for the first time, the surface must be prepared to receive information in the desired format. This preparation is known as *formatting* the disk. It is performed with the CP/M system program called FORMAT.

To begin, turn on your computer and at the system prompt type the command:

```
DIR <RETURN>
```

The computer will list the programs stored on the system disk. Because CP/M Plus is provided with so many system programs, they cannot all fit on the same disk. When we speak of the system disk, we are referring to the original CP/M disk that came with your Commodore 128. We call the other two CP/M disks the supplemental CP/M disks.

Now look for a program named FORMAT.COM in the listing on the screen. (In discussing *executable programs,* we will omit the extension .COM because you do not type the extension when you enter the command. Furthermore, all executable programs must have this extension.)

We have seen that, because programs depend on the CP/M system for interaction with the hardware, the same version of a program can generally be run on all CP/M computers. However, the formatting program is an exception. It must be specifically written for a particular computer. Do not try to format a disk on one computer with a formatting program taken from a different computer, even if they both use CP/M.

Be warned that the formatting program will destroy any information previously stored on a disk. Therefore, be sure to use a

new disk or one that does not contain information you want to keep. Be careful not to format your system disk accidentally.

If you have two disk drives, you can run the formatting program from drive A and format a new disk in drive B. However, if you have only one drive you must start the format program from your system disk, then remove the system disk and substitute the new disk. Give the command:

```
FORMAT <RETURN>
```

The computer will copy the format program from the system disk into memory and then start it up. This program will present a list of disk types in the form of a menu. These are the choices for the Commodore 128:

```
C128 double sided
C128 single sided
C64 single sided
```

The first choice is the one we want to use for the Commodore 1571 disk drive. The other two choices are for the Commodore 1541 disk drive. As you can see, the second and third choices format the disk for use on only one side.

Remove the system disk and insert the disk you want to format. Choose the default format (C128 double sided), the one that is highlighted, by pressing the RETURN key.

At the end of the formatting step, the program will ask if you want to format another disk. Take the newly formatted disk out of the drive, insert it into its sleeve, and put it aside. Insert another new disk into the drive and press the RETURN key to format a second disk.

The newly formatted disks can be used to create new programs or data files. If you want to use the disks only for storing programs and data, they are now ready for you to copy files onto them or create new data files. However, these disks do not have all the parts needed to run CP/M. Therefore, you must copy two files from your system disk if you want to use these disks to start your computer.

You can use the system program called PIP to transfer the two files CPM+.SYS and CCP.COM from your system disk to one of the new disks. Then you will be able to boot from the new disk. (Do not do this yet however.) The program named COPYSYS, which must be run with other CP/M computers, does not work with the Commodore 128. However, this program is provided on the original CP/M disk anyway. Therefore it unnecessarily takes up space on your system disk.

Now let us see how to add date and time to our files.

Date and Time Stamping

CP/M can associate two dates and times with each file. However, this feature is not automatically activated. CP/M can record the date and time that a file was created or most recently altered. It can also record when it was last accessed or executed. This feature, called *date and time stamping,* enables you to distinguish the more recent version of two similar files. CP/M must be especially instructed to do this using the programs INITDIR and SET. The method is discussed in Chapters 3 and 7. In addition, the system clock must be set to the correct date and time using the DATE program. This is discussed later in this chapter. (The three programs INITDIR, SET, and DATE are not included on your original CP/M disk; they are found on one of the supplemental CP/M disks available from Commodore.)

The important thing to remember is that if you want to use date and time stamping, you must specially prepare the directory before you place any files on the disk. Let us consider this step now.

Place one of the supplemental CP/M disks in drive A. Give the DIR command to see if the program INITDIR is present. If so, give the command:

```
INITDIR E: <RETURN>
```

The screen displays this question:

```
Do you want to re-format the directory on drive E (Y/N)?
```

Remove the supplemental system disk from the disk drive and insert a newly formatted disk. Then answer Y for yes and press the RETURN key. You will then be instructed to insert disk E in the drive. After you have changed disks press the RETURN key again. The new disk will be prepared for date and time stamping. Skip this step if you do not have the INITDIR program from the supplemental disk or if you do not want to activate date and time stamping.

We are now ready to copy the programs from the original system disk to the disk in drive E (or drive B if you have two drives).

Copying Programs and Data with PIP

One Disk Drive Our copying procedure is nearly complete. We have formatted two new disks and initialized the directory for date and time stamping. In fact, if we needed a new empty disk for data, we would be finished. In the next step, we will copy all the programs from the original disk by using a powerful program called PIP. The two CP/M files CPM+.SYS and CCP.COM will be included so we can boot from this new disk.

Place the original disk into the drive. Your screen should show drive A as the current drive. Give the command:

```
PIP E:=A:*.*[V] <RETURN>
```

Be sure to insert a space after the name PIP but nowhere else. This command directs PIP to copy all the files from drive A to drive E. Since there is only one drive, the computer copies the first file from the original disk into memory and then asks you to place disk E in drive A. Replace the original disk with one of the newly formatted ones. Then press the RETURN key. The computer copies the file from memory to the new disk. Continue in this way until all the files have been copied. That is, you must exchange disks for each file that is copied.

As each file is copied, its name is displayed on the video screen. At the end of the copying step, the system prompt will appear again, and you can continue with another operation or

shut off your computer. When you have finished copying your master disk, be sure to insert it into its sleeve, label it properly, and put it away in a safe place.

Two Disk Drives In the previous section, it was assumed that you have only one disk drive that is addressable as both A and E. However, if you have two drives, addressable as A and B, it is much easier to copy a disk with PIP. Place the original disk in drive A and the new disk in drive B. Give the command

```
PIP B:=A:*.*[V] <RETURN>
```

and all the files will be automatically copied. You will not have to switch disks.

We have just used PIP to copy all files on a disk. Now let us see how to copy individual files with PIP.

Adding the CP/M Program SHOW

Later in this book we are going to need the CP/M program called SHOW.COM. Unfortunately, this program is not included on the original CP/M disk provided with the Commodore 128. Of course, it is available on one of the supplemental CP/M disks. Let us therefore place a copy of SHOW on our new system disk. If you do not have the supplemental disks, skip over this section. However, you should consider purchasing these disks from Commodore. (An order form is provided in the Commodore 128 manual.)

Place one of the supplemental disks in drive A and give the command:

```
DIR S*.*
```

Look for the name SHOW COM in the listing. If you cannot find it, change to the other supplemental disk and give this command again (by holding down the CONTROL key and pressing the W key). When you have found SHOW, give the command

```
PIP E:=A:SHOW.COM <RETURN>
```

Change to your new disk when instructed to do so and press the RETURN key. This will place a copy of SHOW on your disk.

Adding the CP/M Program DATE

Another program we will need is named DATE. This program is not provided on the original disk. Using the DIR command, look for the name DATE.COM in the directories of the supplemental CP/M disks. When you locate this program, copy it to your new system disk as you did with SHOW in the previous section.

Your Commodore 128 has a clock that can keep track of both the date and the time. Unfortunately, the clock stops each time the computer is turned off or reset. This means you must set the correct date and time each time you turn on the computer. Let us do that now.

You can set both the date and time at once by giving the command

```
DATE MO/DA/YE HO:MI:SE
```

where *MO, DA,* and *YE* are respectively the number of the month, the day, and the year; and where *HO, MI,* and *SE* are respectively the hour, minute, and second. An example of this command is

```
DATE 11/5/86 15:25:0
```

to set the date for November 5, 1986 and the time to 3:25 PM. Notice that there must be an entry for each of the six *fields,* and that 24-hour time is used. It is not necessary to include a leading zero when a number is less than 10. That is, 08 and 8 are equivalent.

Some Elementary Operations with PIP

PIP is a very powerful and versatile CP/M program. You will frequently use PIP for making backup copies of your important

files. With a two-drive system, you will always want a copy of PIP on the system disk in drive A; with a one-drive system, you might want to have a copy of PIP on each disk.

In the previous sections we used PIP to copy all the files from one disk to another. Then we used it to copy single files. PIP can do other tasks such as altering a disk file, displaying a disk file on the video screen or printer, and creating a disk file from the keyboard. Since Chapter 4 is devoted entirely to PIP, we will briefly consider only a few of the more important features now.

Creating a Data File

We have previously considered disk files such as PIP and FORMAT that are provided on the original CP/M disk. It is also possible to create a disk file of your own that contains ordinary text such as a letter or report. This type of disk file is known as a *text file* or ASCII file.

In this section we are going to create a new file to demonstrate some CP/M features. Therefore, be sure that there is sufficient space on the disk. Give the command

```
SHOW
```

and you will see something like

```
A: RW, SPACE: 144K
```

on the video screen. The number at the end of the line indicates that 144 kilobytes of space are available on the disk for new programs. We only need about 4 KB for our next step. (If you do not have a copy of SHOW on your system disk, please go back to the previous section and make the transfer.)

Generally, you use PIP for copying files but you use a system editor such as ED or a word processing program such as WordStar to create text files. You will learn to use ED in Chapter 5. However, we need to create a text file at this point. Since this text file will be very small, we can create it with PIP. Keep in mind that this is not the best way to create text files. Furthermore, it will not be possible to edit the file or correct errors

when you are using PIP in this way. Any characters you type, including the control characters, will become part of your new file. On the other hand, this is only a test file; thus typographical errors won't matter.

Put the new copy of the system disk into the disk drive. If A is not the current drive, give the command

```
A: <RETURN>
```

to make A the current drive. Then give the command

```
PIP SAMPLE.TXT=CON: <RETURN>
```

to create a new text file. Be sure to include a space after the name PIP but nowhere else on the line. This command directs PIP to read all characters that are typed at the keyboard (CON:) and place these characters into a disk file named SAMPLE.TXT.

When we previously used PIP to duplicate a disk file, the first parameter was the new file name (the *destination* file) and the second parameter was the original file name (the *source* file). A similar construction is used in this example. The difference is that the source name refers to the keyboard rather than a disk file. Notice that there is a colon at the end of the name CON. This is a CP/M convention used to distinguish a peripheral (hardware) device from a disk file (software). We could create a disk file with the name CON, but then we would not place a colon at the end of its name.

Let us now consider the difference between a command line and a line of text for a disk file.

Command Lines and Text Lines

Up to now, we have given commands to CP/M by typing information at the keyboard. CP/M displays each character on the video screen as we type it. When we have completed the *command line,* we press the RETURN key. CP/M sends a carriage-return character to the screen and then automatically sends the line-feed character too. However, when an executable program such as PIP is running, the information you type at the

keyboard goes directly to the program (PIP) rather than to CP/M. Therefore, the rules can change.

When you create a new disk file with PIP, the characters you type are displayed on the video screen as usual. PIP also places these characters into the new disk file. However, when you press the RETURN key at the end of a line, PIP sends only a carriage-return character to the screen and to the new disk file; it does not send a line feed. Therefore, you must add your own line feed each time you press the RETURN key when using PIP in this way. Notice that the LINE FEED key is on the top row of the keyboard. You can also use ^J, the character generated by the line-feed key, to send a line feed. In the following example, <LF> is used to remind you to press the line-feed key after you press the RETURN key.

After typing the PIP command line, PIP will gain control and read the characters you type at the keyboard. Enter these two lines of text into your new file:

```
This is my new text file called SAMPLE.TXT. <RETURN> <LF>
This is the second line of my new file. <RETURN> <LF>
```

Everything you type will be included in this file (and also shown on the video screen). Complete the operation by typing a ^Z. This command terminates PIP and returns control to CP/M. It also places the ^Z character in the file. You have just created a new disk file called SAMPLE.TXT that contains the lines you typed at the keyboard. Give the command

```
DIR
```

to verify that the file named SAMPLE.TXT is present on the disk.

Duplicating a Single Disk File

To be sure that you understand the operation of PIP, let us make a duplicate copy of the new disk file SAMPLE.TXT. Give the command

```
PIP COPY.TXT=SAMPLE.TXT[V] <RETURN>
```

Be sure to press the space bar after PIP but nowhere else. This command directs PIP to make a copy of your file. The first *parameter,* COPY.TXT, is the name of the new file and the second parameter, SAMPLE.TXT, is the name of the original file. The V enclosed in brackets is known as an *option.* It directs PIP to verify that the new copy is correct. Always give this *verification option* when you are copying disk files. (But do not use the V parameter when CON: is one of the parameters.)

Displaying and Printing a File

Displaying a File on the Video Screen

In the previous section, we created and then copied a text file using PIP. At the end of these steps, control returned to CP/M as indicated by the A> prompt. Now that we have two text files, we can examine them by displaying their contents on the video screen. There are several ways to do this. One method is to use the CP/M command TYPE.

At the system prompt enter the command:

```
TYPE SAMPLE.TXT <RETURN>
```

Be sure to include a space between TYPE and SAMPLE but nowhere else. (You may have noticed that a space must always follow a CP/M command.)

A word of caution: you can examine the contents of a text file with the TYPE command but you cannot inspect program files such as PIP in this way. We introduce the TYPE command at this point so we can look at our new file. We will discuss this command and its limitations in detail in Chapter 3.

You can also use PIP to display a file. This method is not as convenient as using TYPE, but it shows another use for PIP. Give the command:

```
PIP CON:=SAMPLE.TXT <RETURN>
```

Notice that the parameters are the reverse of the ones we used

to create the file in the first place. This command directs PIP to send the contents of the file SAMPLE.TXT (the second parameter) to the video screen (CON:—the first parameter). Do not forget to add a colon to the end of CON, or CP/M will create a disk file named CON containing the text in SAMPLE.TXT.

Now that we have seen how to display a text file on the video screen, let us see how to send a file to the printer instead of the screen.

Printing a File

Sometimes we need to obtain a printed listing or hard-copy version of a text file. We can use PIP to send a copy to the printer. Turn on your printer and give the command:

```
PIP LST:=SAMPLE.TXT <RETURN>
```

If you make a typing error, use the control characters we considered earlier in this chapter to correct the mistake.

Notice that this command is similar to the previous one except that the device name CON: (console) has been changed to LST: (list). Also notice that there is a colon at the end of the name LST. This distinguishes the printer from a disk file named LST. Furthermore, notice that the V option is not included.

When we used PIP to copy all the files from one disk to another, we did not have to name each file. Instead, we used the wild-card symbol *.* to refer to all files on the disk. In a similar way, we can obtain a printed listing of all text files by giving the command:

```
PIP LST:=*.TXT <RETURN>
```

There is an alternative method for printing text files. It is not as sophisticated as the first method because the commands you type also appear in the printout. However, the second method is faster. Let us try it.

Turn on the printer and then give the command ^P to engage the printer. All subsequent output will appear at both the printer

and the video screen. Giving the ^P command a second time reverses the effect, disengaging the printer. In other words, ^P is a *toggle* that alternately turns the printer on and off.

Engage the printer with the ^P command. Then give the command:

```
TYPE SAMPLE.TXT <RETURN>
```

Now the TYPE command line will appear at the beginning of the listing because the printer was engaged before the command was given. To prevent this, you can give the ^P command after typing the command line but before pressing the RETURN key.

After CP/M has completed printing the file, it will display the system prompt on the video screen. This prompt will also appear in the printout since the printer is still engaged. Give the ^P command again so that no additional characters will appear at the printer.

The TYPE command stops each time the video screen has been filled. This is convenient if you want to read the information on the screen—otherwise, the information goes by too quickly. However, if you are using the TYPE command to print a long file, it would be better to print the entire file without stopping. To do this, you place the NOPAGE *option* in brackets at the end of the TYPE command. This version of TYPE requires a system program called TYPE.COM located on one of the supplemental CP/M disks. If you do not have this program on your system disk, make a copy from the supplemental CP/M disk. Otherwise skip over the next section.

If you have already placed a copy of TYPE.COM onto your system disk, give the command:

```
TYPE SAMPLE.TXT[NO PAGE]
```

Add a space after TYPE but nowhere else. The option can be abbreviated to [NO P if you want. (Of course, this is a short file so the result is the same as before.)

Now you will learn about files and file names by examining the disk directory.

Files and File Names

A program or lines of text can be stored on the disk as a *file*. Each file is assigned a unique *file name* that you use to refer to the file. When a file is written on disk, CP/M creates an entry in the disk *directory* to keep track of where the file is located on the disk. Of course, the directory is also located on the same disk.

If you want to execute a program, you simply give CP/M the corresponding file name. CP/M will locate the program on the disk from the information contained in the disk directory. It will then copy the program into memory and execute it. If you want to examine a text file with the TYPE command, CP/M will locate the file according to the information in the disk directory.

The directory of a disk is like the directory of offices in the foyer of a building, listing names and locations corresponding to the tenants of the building. If the directory is destroyed, the programs on the disk, like the offices in the building, can no longer be located.

If a program needs data to work with, you can enter the required information directly from the keyboard. Alternatively, the information can be stored in a *data file* on the disk, and you can simply enter the name of the data file. As an example consider the PIP program we just ran. We directed CP/M to execute PIP and we included one or two file names for PIP to use. Similarly, we executed TYPE and gave the name of the file to be displayed. Let us now examine the directory on your main system disk.

Examining the Disk Directory

In this section we will look briefly at the commands DIR and DIRSYS, which are used to examine the disk directory. Both commands are discussed in more detail in the next chapter.

The disk in drive A contains the CP/M operating system and related programs. You can obtain a listing of the file names that are present on this drive by typing the command for directory:

```
DIR <RETURN>
```

This is the listing for the original CP/M disk:

```
A:CPM+    SYS : CCP      COM : HELP    COM : KEYFIG    COM
A:KEYFIG  HLP : FORMAT   COM : PIP     COM : COPYSYS   COM
A:HELP    HLP
```

If you get the message

```
SYSTEM FILE(S) EXIST
```

when you give the DIR command, it means that there are additional files, designated as system files, present on the disk. However, these files are not shown when you give the DIR command. You can see a listing of these system files by giving the command DIRSYS after the next system prompt appears.

The display in this example shows four different file names on each line. We know that these files are on the system disk in drive A because each line of the display begins with the letter A. A colon separates one file name from the next. Each file name is shown in two parts, with blank spaces separating the first part from the second part. (The output from DIR is not typical. With other CP/M programs the two parts of the name are separated by a period and no blanks are present. As an example, we usually see the form PIP.COM rather than PIP COM.) If there are many directory entries, the display automatically stops when the video screen is filled. Press the space bar to view the next screen or give the ^C command to end the listing.

CP/M File Names

A CP/M file name consists of two parts—the primary name and the extension. As we have seen, these two parts are usually separated by a period.

The *primary name,* the part to the left of the period, contains one to eight characters. This part describes the contents of the file. The *extension,* which follows the period, contains a maximum of three characters although it may contain none. Usually, the extension describes the type of file, so this part of the file name is also

called the *type*. For example, all files with the extension COM are command files or programs; files with BAS as an extension are BASIC source programs; and files with OVR and OVL as an extension are overlay files, those designed to be copied over other programs. You must not choose the extension BAK, since the system editor uses this name for backup files, or $$$, because this name marks a temporary file that is automatically erased. Text files do not need an extension; however, TXT is commonly used. Common file-name extensions are listed in Table 2.2.

When you select file names, be sure to make the primary name distinct from the primary names of other files. Do not simply give a group of files the same primary name and different extensions. For example, you could name two chapters of a book CHAP1.TXT and CHAP2.TXT, but you should not name them CHAP.1 and CHAP.2. Let us see why.

When you alter a file with an editor, the new version is automatically given the original name and the extension of the original file is changed to BAK (for backup). For example, if you edit a file named CHAP.1, the original version becomes CHAP.BAK. If you then edit CHAP.2, the original copy of this file also becomes CHAP.BAK. Two files cannot have the same name, so the first CHAP.BAK is erased automatically. You lose the backup file for Chapter 1. On the other hand, if you name the files CHAP1.TXT and CHAP2.TXT, there will be no problem because the backup files will be named CHAP1.BAK and CHAP2.BAK.

Sometimes you reference a file only by its primary name; you do not include its extension name. For example, to load the BASIC source program SORT.BAS, you only have to give the primary name SORT, not the full name SORT.BAS. As another example, all *executable programs* must have an extension of COM. However, you execute a program simply by entering the primary name. The extension must not be included. We executed PIP by typing the primary name; we did not give the name PIP.COM.

Sometimes you must give the complete file name, including a period between the primary part and the extension. For example, if you want to display a file on the video screen with the TYPE command, CP/M will not know which file you have in mind unless you give the complete name, including the

Extension	Type of File
ASM	Assembly language source file
BAK	Backup file
BAS	BASIC source program
COM	Executable program
FOR	FORTRAN source program
HEX	Hexadecimal file created by MAC
INT	Intermediate work file
LIB	Library file used by MAC and RMAC
OVL	Overlay file used by applications programs
OVR	Overlay file used by applications programs
PAS	Pascal source program
PLI	PL/I source program
PRN	Program listing created by MAC and RMAC
REL	Relocatable file created by RMAC
SPR	System page relocatable file for CP/M
SUB	Source file used by SUBMIT
SYM	Symbol table file created by MAC and RMAC
SYS	System file used by CP/M
XRF	Cross-reference file
$$$	Temporary file

Table 2.2: *Examples of File-Name Extensions*

extension. Similarly, you must give the complete name of a file when you want to alter it with the system editor.

Valid File-Name Characters

File names are formed from the 26 letters of the alphabet and

the 10 digits. You can also use some of the special characters, such as

```
+ - / % $
```

in any position of either the primary or the extension name. However, remember that you must not use $$$ as a file-name extension. This is a name assigned by PIP and other programs to designate intermediate results. Files with this extension are automatically erased at the conclusion of the task.

These characters must *not* be used in a file name because they have a special meaning to the CP/M system or to some of the system programs such as PIP:

```
< > [ ] , : ; = * ? !
```

In addition, the period cannot be used because it is reserved for separating the primary name from the extension. Spaces are also not allowed, since, as we have seen, a space separates one name from the next.

Let us consider some examples of file names. The file name

PROG-22.TXT

is valid, but the file name

PROG=22.TXT

is not because it contains the equal sign. Here are some valid file names:

SORT.BAS
CHAP1.TXT
12/15/86
PART-2-3.TXT

The following names are *not* valid:

12:15:06	(colons)
CHAP 2.TXT	(blank)

WHAT?	(question mark)
LONG-NAME.TXT	(more than eight characters in primary name)
LONG.EXTEN	(more than three characters in extension)

File Names Containing Wild Cards

Sometimes you need to refer to several files at once or you want to locate one file in a collection of similar files. If all the files in the group have a part of their names in common, you can refer to them by a single file name using wild-card characters. As we have seen, CP/M file names contain two parts, or *fields*—the primary part and the extension. There are two wild-card characters that you can use in either the primary part or the extension: ? and *. The question mark represents a single character at the given position, and the asterisk refers to all the remaining characters in the field (that is, the primary field or the extension field). Let us see how these characters are used in file names.

An example of a file name that includes a wild card is

```
SORT?
```

The computer will match this name with any file names containing the letters SORT and one more character. For example, it will match SORT1, SORT2, and SORT3 but it will not match SORT12. It will also match the file name SORT.

Primary names always occupy eight characters of a directory entry. If a name has less than eight characters, CP/M fills out the remaining positions with blanks. For example, the name SORT1 is five characters long, so three blanks are added to the end. The name SORT is four characters long and so there are four blanks at the end. Except in the directory listing, the blanks do not appear when the name is displayed by the computer.

A blank character in a file name can match a wild-card character. Therefore, the name SORT matches the name SORT? because the fifth character of the name SORT is a blank.

When an asterisk is included in a file name, CP/M fills out the field with question marks. This is an internal change that is not

shown on the screen. For example, when the name SORT* is given, CP/M converts the name internally to SORT????. Therefore, the name SORT* matches the file names SORT, SORT1, SORT12, SORT123, and SORT1234. The result is the same as if the name SORT???? had been given but it is shorter and so easier to type. Let us consider a few more examples.

CP/M will fill out the * character in either field of the file name. Thus, *.BAS is expanded internally to ????????.BAS and refers to all files with the extension BAS. The name SORT.* becomes SORT.??? and refers to all files with the primary name SORT no matter what the extension might be. As we have seen, this name also matches the file name SORT without an extension.

Notice that an asterisk applies to only one of the two fields—the primary name or the extension—but not to both. The special form *.* becomes ????????.??? and so refers to all files on the disk. There can be regular characters (letters and numbers) in front of an asterisk but not after. For example, C*.BAS is correct, but *C.BAS is not. Recall that earlier we used *.* to copy an entire disk. This was the command:

```
PIP E:=A:*.*[V]
```

Of course, regular characters can follow the ? wild-card symbol.

The symbol *.*, sometimes called star-dot-star, represents all the files on the disk. Wild-card characters can be used with DIR, TYPE, and PIP when you need to refer to several files.

Try using the ? and * characters with DIR to see how they can select groups of files. Remember the ? will match any one character for each occurrence. If drive A is not current, give the command:

```
A: <RETURN>
```

Then give the following commands, being sure to press the RETURN key at the end of each line:

```
DIR *.COM        (all COM files)
DIR C*.COM       (all COM files beginning with C)
```

DIR ???.COM	(COM files that have primary names with one, two, or three characters)
DIR C??.COM	(COM files with primary names that have one, two, or three characters and begin with C)
DIR C*.*	(all files beginning with the letter C)

In the first example, *.COM matches all COM files. The programs CCP.COM, HELP.COM, KEYFIG.COM, FORMAT.COM, and PIP.COM match this name. The second example matches all COM files that begin with the letter C. These include the files CCP.COM and COPYSYS.COM.

The third example demonstrates use of the question mark. The name ???.COM matches CCP.COM, PIP.COM, and DIR.COM because each of these names has three characters in the primary part. However, we saw that the question mark also matches a blank. Therefore, this name will also match any COM file with only one or two characters in the primary name. However, it will not match names such as KEYFIG.COM and FORMAT.COM since the primary names have more than three characters.

The fourth example is like the third except that the first character must be the letter C. The file CCP.COM matches this specification. The fifth example will match all files that start with the letter C.

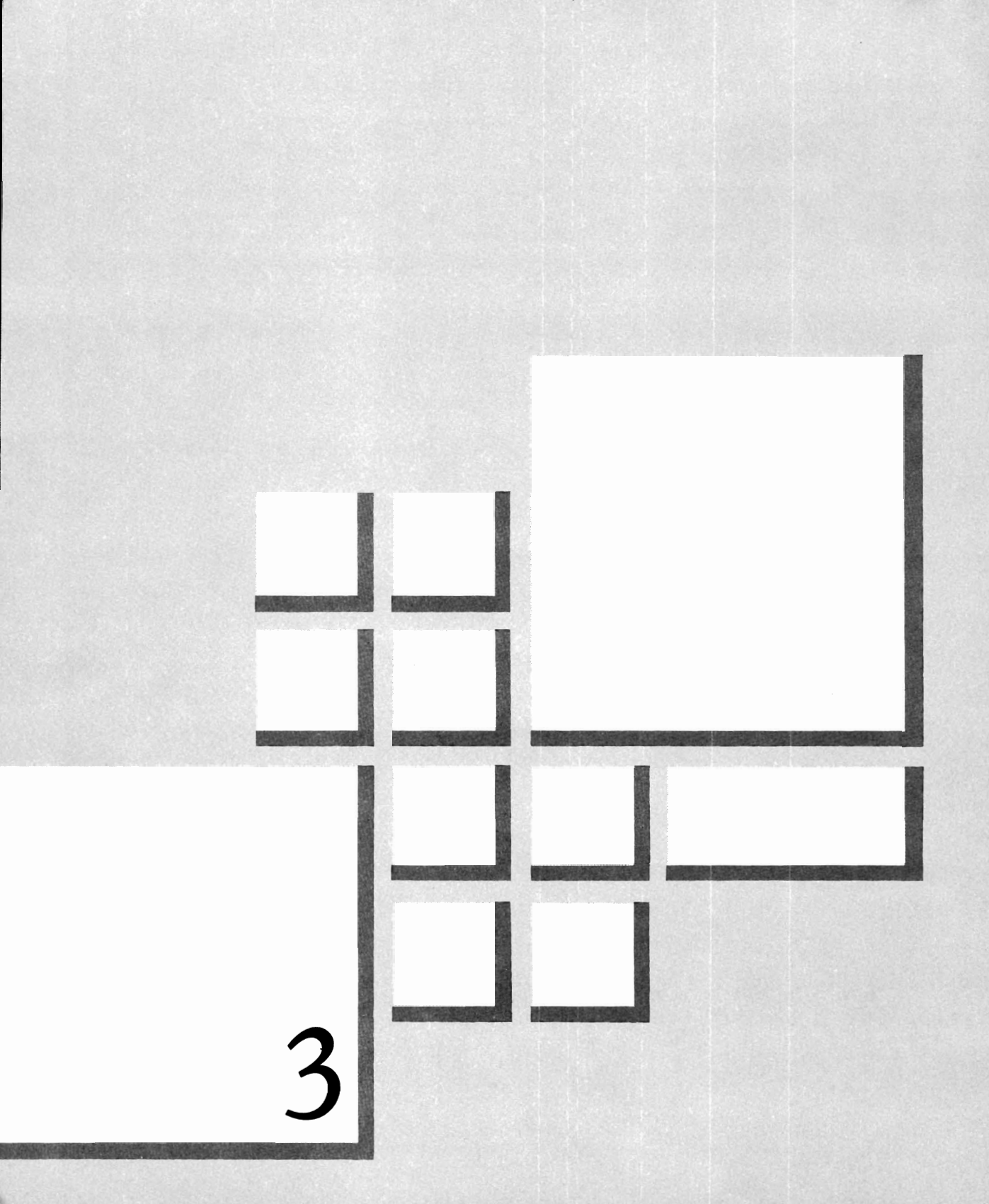
Summary

In this chapter you became acquainted with some of the features of CP/M Plus. You have seen how CP/M responds to entries from the keyboard and how it displays information on the video screen. You learned to recognize and change the current, or default, disk drive. You learned the CP/M control characters and special control keys, particularly those used for editing the command line.

You prepared a new disk using the FORMAT program and duplicated disks with the PIP program. After creating and copying a file with PIP, you learned how to display a text file on both the video screen and the printer. We finished the chapter by

viewing the disk directory with DIR and then learning the conventions for naming files and the use of wild cards in file names.

You may be surprised to learn that you now know enough about CP/M Plus to run most applications programs without problems. If you want to learn more about your operating system and its resources, read on.


3

Commonly Used Commands

In the previous chapter we learned some of the basic operations of CP/M Plus. In this chapter we begin studying the system in detail. You will learn how to use the built-in commands USER, DIRSYS, RENAME, and ERASE, and the transient commands SETDEF, SHOW, SET, SUBMIT, and DEVICE. You will also learn more about the commands DIR and TYPE, which were introduced in Chapter 2. At this point it is not necessary to memorize all these

commands. However, you should frequently review the information in Chapters 2 and 3 because it is so useful. Even infrequent Commodore 128 users should understand the following:

- How to use the control characters ^C, ^S, ^Q, ^X, ^U, ^P, ^H
- How to duplicate complete disks
- How to copy individual files with PIP
- How to inspect the disk directory with DIR
- How to erase files with ERASE
- How to change file names with RENAME
- How to determine the remaining disk space with SHOW

The material presented in this chapter will show you how to use the most important CP/M commands. You should refer to this chapter frequently until you are comfortable with the operation of CP/M. Then you will find Chapter 7, which summarizes all the CP/M Plus commands, to be a useful reference guide.

Commands and Parameters

Before we begin a discussion of the various commands, let us consider the distinction between a command and its parameters. As we have seen, a *command* is an instruction that you give to CP/M in a specific form. A *parameter* is an item of information that qualifies the command. Sometimes a parameter is the name of a disk file that is going to be affected by the command.

For example, consider the line:

```
DIR SORT.BAS
```

The first word, DIR, is a command to CP/M. The second word, SORT.BAS, is a file-name parameter; it tells CP/M to display the name SORT.BAS if the file is found in the directory. At least one space must separate the command from its parameter. Using wild-card characters in a parameter, such as DIR *.BAS, you can refer to all files with the type BAS.

Parameters may be optional or required with a particular command, depending on the situation. The discussion of each command will indicate generally when you need to add a parameter. Sometimes when you omit a required parameter, CP/M will ask for it.

A second type of parameter, known as an *option,* is enclosed in square brackets. The left bracket takes the place of the blank. Option parameters modify the operation of a command. We previously used the V option with PIP and the NO PAGE option with TYPE. For example, the command

```
TYPE SORT.BAS[NO PAGE]
```

contains both a file-name parameter and an option parameter. It commands CP/M to display on the video screen the contents of the file SORT.BAS. The listing does not stop at the end of each screen or page because the NO PAGE option is included. The file-name parameter designates the file that the command will work with. The option parameter alters the operation of the command.

As we have seen, file-name parameters must be separated from the command itself by at least one space. However, a space cannot appear in the middle of a command or a parameter.

The beginning square bracket of the option parameter is placed immediately following the file-name parameter without an intervening space. If a file-name parameter is not given, the option parameter is placed immediately after the command, without a space. The closing square bracket is optional. Longer option parameters may be abbreviated to two or three characters.

Parameters may include letters (A–Z), digits (0–9), and other symbols. You may enter the letters as either uppercase or lowercase since CP/M converts all letters on the command line to uppercase. The parameters given on the command line are known collectively as the *command line tail.*

Usually, only one or two file-name parameters follow a CP/M command. However, there is no limit to the number of file-name and option parameters that might be given to a CP/M command. If a line becomes too long for the video screen, type a ^E to start a new line. This moves the cursor to the beginning of the next line but does not otherwise affect the command line.

Built-In and Transient Commands

When CP/M displays the system prompt, you can respond with one of two types of command. One type is built into the CP/M system. We will refer to this as a *built-in command.* The other type, which we will call a *transient command* or program, is stored on disk.

The built-in commands are always present in memory during command input. Since they do not have to be copied from disk, they can be executed rapidly. However, the memory space allotted to the built-in commands is limited. Therefore, there are only six built-in commands.

Additional operations are provided by programs that are stored on disk as command, or COM, files. You execute these commands (programs) by giving the primary name on the command line without including the COM extension. In response, CP/M copies the program from disk into memory and then runs the memory version. Since the memory copy is only temporary, such a program is sometimes called a transient program or transient command. The region of memory where these programs are executed is called the *transient program area* (TPA). This is the largest portion of memory.

Transient programs can be copied from one disk to another, and they can be renamed or erased. Unlike a built-in command, which is always ready to be run, a transient program cannot be run until it is copied from disk to memory.

With earlier versions of CP/M there was a clear-cut distinction between built-in and transient commands. However, with CP/M Plus that distinction has become blurred. Four of the built-in commands have transient extensions that are COM files. CP/M automatically executes a transient extension if it needs the additional features the extension provides. Therefore, these four commands sometimes behave like built-in commands and at other times like transients.

The six built-in commands are listed in Table 3.1. They do not have to be spelled out in full; you can shorten them to the abbreviations listed.

Two of these commands—DIRSYS and USER—are wholly built-in; they do not have transient counterparts. The other four

Command	Abbreviated Name	Function
DIR	DIR	Display the disk directory
DIRSYS	DIRS	Display the directory of system files
ERASE	ERA	Erase a disk file
RENAME	REN	Rename a disk file
TYPE	TYP	Display a text file on the video screen
USER	USE	Change the user area

Table 3.1: *The Six Built-In Commands*

commands exist as both built-in commands and transient programs. If you look at a directory listing of the CP/M utility disks available from Commodore, the programs named DIR.COM, ERASE.COM, RENAME.COM, and TYPE.COM represent the transient versions of these commands. Because they are wholly built-in, DIRSYS and USER do not appear in the directory listing.

There is an additional built-in command that changes the default drive. If the current drive is A and you enter the command E:, CP/M will change the default drive to E.

You can add the drive name to the file name of a transient program to designate on which disk the program is located. For example, A:PIP tells CP/M to look for PIP on the disk in drive A. The drive name is not properly a part of the file name since it is not encoded into the directory. By contrast, you must not add a drive name to a DIRSYS and USER command because these built-in commands are not executed from a disk file. For example, the command A:USER is improper.

If RENAME, ERASE, or TYPE are given without a file-name parameter, or if DIR, ERASE, or TYPE are given with an option parameter, the corresponding transient extension is needed. Then the transient version must be located on the current drive

or the drive name must be included in the command. For example, if E is the current drive and the transient extension TYPE.COM is located on drive A, the command

```
A:TYPE SORT.BAS[NO PAGE]
```

is proper. Alternatively, make drive A current and give the command:

```
TYPE E:SORT.BAS[NO PAGE]
```

As shown in the previous example, a drive name can appear in one of the parameters to a built-in command.

Now let us look at each of the six built-in commands in detail. We begin with TYPE.

TYPE

Viewing the Contents of a Text File We have seen that the TYPE command provides an easy method for examining a text file on the video screen. To understand more about TYPE, let us look briefly at the difference between text and binary files.

Information is stored in memory or on disk as a sequence of digits. The computer also communicates with the keyboard, the video screen, and the printer in digital form. However, there are several different coding schemes used to represent digital information. Two of the most common are known as *ASCII* and *binary.* Though both of these schemes are used for information stored in memory and on disk, the console and printer are designed only for the ASCII representation.

Text files are written in ASCII; they can be viewed on the video screen or printer with the TYPE command. However, binary files, such as those with the file extension COM or REL, must first be translated to ASCII. Therefore, you cannot use the TYPE command to view these files. We will look further at the difference between text and binary files in Chapter 4.

The format for the TYPE command is

```
TYPE d:name.extension
```

where the letter d stands for the drive name. You may omit the drive name if the file is on the current drive. If you omit the file-name parameter with this command, CP/M will ask for it. If the file name contains the ? or * wild-card characters, CP/M will display all the corresponding files.

Normally, CP/M will stop the display when the screen has filled. Then you can display the next screenful by pressing the space bar. Alternatively, you can obtain a continuous listing of a file by including the option NO PAGE in square brackets. For example, the command

```
TYPE SORT.BAS[NO PAGE]
```

will display the entire file SORT.BAS without stopping when the screen is full. If the information moves too quickly, you can momentarily stop the display with ^S and then restart it with ^Q. This method is useful for quickly surveying a file.

If you want a printed listing of a file, turn on the printer and press ^P before completing the TYPE command. Be sure to include the NO PAGE option so the printout will not stop at each page. The file will be displayed on both the video screen and the printer.

Displaying a single file requires only the built-in version of TYPE. However, the transient version of TYPE is required if the ? and * wild-card symbols are included, if the file-name parameter is omitted, or if an option parameter is given. Then TYPE.COM must be on the current disk or you must include the drive name by entering, for example, A:TYPE.

USER

Changing the User Area Each CP/M disk can be partitioned into 16 regions, or areas, which are numbered from 0 to 15. Each area can be assigned to a different user or to a different subject. For example, you can store the formatting program in one user area, BASIC programs in a second area, office memos in a third area, and so forth. This is a useful technique if you are working with a hard disk because it contains so much space. However, the space on a floppy disk is limited. Therefore, it is not practical

to partition a floppy disk into more than a few regions, although unused space is not assigned to any user area.

Whenever you create a disk file, the corresponding directory entry is automatically coded with the current user number. The files in one user area are not normally accessible to those working in other user areas. (We will learn how to copy files from one user area to another in Chapter 4.)

Each time you start up CP/M, the system automatically selects user area 0. You can readily change to a different user area by giving the command USER and a parameter corresponding to the new user area. Let us see how this works. Give the command

```
USER 5 <RETURN>
```

to change to user area 5. Be sure to include a space between the command and its parameter. When the user number is not 0, CP/M displays the user number in the prompt. For example, the prompt 5A> indicates that user 5 and drive A are active. A single command can change both the user area and the default drive. For example, the command

```
USER 3E: <RETURN>
```

makes user 3 and drive E active. Try this command now (don't forget the colon).

The built-in commands are available to all user areas. However, each user normally has access to only those disk files that are in the current user area. Give the command:

```
DIR <RETURN>
```

The response will be NO FILE because there will not be any files in user area 3. Now give the command

```
USER 0 <RETURN>
```

to return to your original user area 0. Give the DIR command to see what files you now have available.

Programs needed in several user areas can be made accessible to all users if they are designated as system files in user area 0.

The SET program discussed later in this chapter is used for that purpose.

Certain programs, such as FORMAT and COPY, can destroy useful information if they are used incorrectly. If these programs are stored in a separate user area, there is less likelihood that they will be executed accidently.

DIR

Listing the Directory The DIR command was introduced in Chapter 2, where we used it to determine the names of files stored on drive A. Now let us study the command in more detail. The DIR command has both a built-in and a transient version. DIR can display a list of all file names, a list of selected file names, or the name of a single file on any disk.

The command DIR given without a parameter displays all the files of the current user area on the current disk. Sometimes the directory contains so many names that the video screen is filled. When this happens, CP/M automatically stops the display. You can continue the listing by pressing the space bar, or you can abort the listing by pressing ^C.

There are two ways to list the directory of a drive that is not current. One method is to give the drive name as a parameter to the DIR command. For example, to display the names of all files located on drive E when drive A is current, give the command:

```
DIR E: <RETURN>
```

Be sure to include the colon after the letter E and to leave a space after DIR but nowhere else. If you omit the colon, CP/M will look for a disk file named E.

An alternative method of viewing directory E is to first make drive E current and then to give the DIR command without a parameter. The commands are:

```
E: <RETURN>
DIR <RETURN>
```

The first method is quicker, and you remain on drive A at the

end of the operation. The second method requires an extra command, and you remain on drive E when the DIR command has finished.

You can obtain a permanent listing of the directory by turning on the printer and typing ^P before completing the DIR command. When the listing has finished, press ^P a second time to disengage the printer.

Now let us see how to add parameters to the DIR command. We have already learned how to use file-name parameters. Let us look at a few more examples.

Adding File-Name Parameters to DIR We have been using DIR without a file-name parameter to show all files in the directory. It is possible to limit the display to one class of files or to a single file by including a file-name parameter. For example, the command

```
DIR *.COM
```

will display the names of all files on the current disk that have the extension COM. We can use the double asterisk with DIR to display all files. However, as we have seen, this parameter is unnecessary since it has the same result as no parameter at all.

To search for a specific file on drive E, give the file name as a parameter. For example, the command

```
DIR E:SPECIFIC.BAS
```

will display the file name SPECIFIC.BAS if it exists on drive E. If the file does not exist, CP/M will give an error message.

A transient extension to DIR called DIR.COM can do many additional tasks, such as placing the file names in alphabetical order and giving the size of each file. This program must be located on the current disk or on a disk in the file-search path so that CP/M can find it. (Setting up a file-search path is discussed later in this chapter.) CP/M executes the transient version of DIR whenever options are included with the command. If CP/M cannot find DIR.COM, it will give the error message:

```
DIR COM required
```

Remember, DIR.COM is a separate program on one of the disks. If it is located on drive A but drive E is current, you can give the command:

```
A:DIR[options]
```

It is possible to include several file names as parameters to DIR. Use a space to separate each file name from the next. Then you must specifically reference the transient version by including the drive name before DIR or adding an option such as [FULL] or [DIR]. For example:

```
A:DIR *.COM *.SUB
```

Using Options with DIR Eighteen different options can be given to DIR. (See Chapter 7 for a complete list.) However, only one or two of them are usually needed at any time. Furthermore, some of these options are mutually exclusive. The options to DIR are enclosed in a single pair of square brackets. If more than one option is given, spaces or commas are used as separators. You can abbreviate an option to the first two letters if the spelling is unique. Be careful not to confuse the DIR options with the regular file-name parameters. Let us see what the difference is.

An option to DIR is a parameter. However, it is distinguished from a file-name parameter by being enclosed in square brackets. For example, the command

```
DIR SIZE
```

refers to a particular disk file named SIZE. However, the command

```
DIR[SIZE]
```

creates a listing of all file names with the corresponding file sizes.

You can give both a file-name parameter and an option parameter in the same command. Either parameter can be given first. For example, the two commands

```
DIR[SIZE]*.COM
```

and

```
DIR *.COM[SI
```

are functionally the same. They both list all COM files with their corresponding sizes in kilobytes. Notice that the second example uses the abbreviated form and that the closing square bracket is omitted. You can omit the closing bracket of an option if it is the last item on the line.

When you give any option to the DIR command, a summary at the end of the listing shows the number of files and the total number of bytes used by the files. In addition, the files are shown in alphabetical order across each line. This can be useful when you are looking for a particular file name in a long list. However, there is a disadvantage to this feature. The files are ordered by a separate sorting routine. When there are many files in the listing, it takes CP/M a noticeable amount of time to perform the sorting.

If you are interested in determining the actual order of the file names in the directory, you may want to omit the ordering. Then you can use the NOSORT option. For example, the command

```
DIR[SIZE,NOSORT]
```

lists all files in the order they occur in the disk directory and also gives the size of each file. In addition, a summary is given at the end of the listing.

Another useful option is EXCLUDE. This option displays all directory entries that do *not* match the given file-name parameter. For example, the command

```
DIR *.BAK[EXCLUDE]
```

will list all files except those with the extension BAK.

We saw that DIR will display files on the default drive if no drive parameter is given. To determine the files on a particular drive, you can give the disk drive as a parameter. There is yet another possibility. You can use the DRIVE option to determine the files on a selected drive, several selected drives, or all drives.

This option takes three forms:

```
[DRIVE = B]
[DRIVE = (B,C,D)]
[DRIVE = ALL]
```

For example, to locate the disk file named SAMPLE.TXT when you are not sure what drive it is on, give the command:

```
DIR SAMPLE.TXT[DRIVE = ALL] <RETURN>
```

Then CP/M will search all logged-in drives for the file.

Earlier we saw that a disk can be divided into several different user areas. A user in one area does not normally have access to programs belonging to other user areas. Consequently, the DIR listing shows only those programs associated with the current user number. However, it is possible to look at the directory of other user areas by including the USER option in the DIR command. There are three slightly different forms for this option:

```
[USER = 3]
[USER = (2,3,4)]
[USER = ALL]
```

The first form shows the directory for user 3 even if the current user number is not 3. The second form displays the directory for users 2, 3, and 4, and the third form gives the directory for all users on the disk.

It is important to remember that the USER option to DIR is different from the USER command, which we discussed earlier in this chapter.

Before we consider the next DIR option, let us take a moment to learn about file attributes.

File Attributes We have seen that the entry recorded in the directory for each disk file contains the file name, the user number, and the file size. The directory includes other information, known as the *file attributes.* A disk file can be marked as read only or read/write, and it can be designated as a system file or a

directory file. These file attributes are initially set to read/write and directory status, but they can be changed at any time with the SET command. We will discuss SET later in this chapter.

Important files should be protected from accidental deletion by marking them as read only. However, if you want to change a file with the system editor, ED, the file must be set to read/write status.

For large-capacity disks it is convenient to designate frequently used files as system files. They can be executed simply by typing the primary name in the usual way, but they do not show in the DIR listing. Another advantage is that system files located in user area 0 are accessible to all user areas. Thus, it is not necessary to have additional copies of such programs in each user area.

System files do not appear in the listing when the DIR command is given without options, but they can be displayed with the SYS option or with the DIRSYS command, which is discussed later in this chapter.

Determining File Attributes with DIR There are several options to the DIR command that you can use to determine file attributes.

The option FULL will display the attributes for one file or a group of files. The listing shows all files that match the file-name parameter. For example, the command

```
DIR *.COM[FULL]
```

lists all COM files in alphabetical order and displays the file attributes. Each line of the listing includes a file name and the appropriate symbols DIR or SYS and RW or RO. The symbols are defined in Table 3.2. Note that two mutually exclusive pairs are shown. That is, a file will have either the system or the directory attribute but not both. Also, a file can be either read/write or read only but not both.

Another way to determine the attributes is to use the options DIR or SYS and RW or RO in place of the FULL option. This method shows only those files that satisfy the options. For example, the command

```
DIR *.COM[SYS RO]
```

Symbol	Meaning
SYS	System file
DIR	Directory file
RO	Read-only file
RW	Read/write file

Table 3.2: *The File Attributes Displayed with the FULL Option to DIR*

displays only those system files that are marked as read-only.

We have discussed only some of the DIR options in this chapter. A complete list of the options is included with the DIR command in Chapter 7. Use the DIR command regularly to check which files are present on the disk. You should also check the directory after you create or erase a file to be sure that CP/M did what you wanted.

Now let us see how to display the directory of system files with a built-in command.

DIRSYS

Listing the Directory of System Files You have just learned that system files do not appear in a DIR listing unless you use the SYS or FULL option. However, DIR gives the message

```
SYSTEM FILE(S) EXIST
```

when appropriate to indicate the presence of system files. When this message appears, you can use the DIRSYS command to get a listing of the system files. You may include file-name parameters (but not option parameters) with the DIRSYS command just as for the DIR command. For example, the command

```
DIRSYS *.OVR
```

will list all system files that have the OVR extension.

RENAME

Renaming Files We reference files stored on disk by using their file names. Therefore, it is important to choose file names carefully. For example, it is logical to use the name FORTRAN for the FORTRAN program and the name COPY for the copy program. However, frequently file names are not descriptive. For example, Microsoft FORTRAN is known as F80 and the CP/M copy program is called PIP. In such cases we can change the name of a file by using the built-in RENAME command. This command does not alter the file in any way; it only changes the name that is listed in the disk directory.

The format is

```
RENAME NEW=OLD
```

where OLD is the original file name and NEW is the new name. Notice that two file-name parameters are given. The new name comes first and the original name second, with an equal sign between the two. Be sure to include a space after RENAME. You can also put spaces on either side of the equal sign, but this is optional. You may omit the parameters, and CP/M will ask for them. But then the transient program RENAME.COM is required.

If the file is on the current disk, you can omit the drive name from both parameters. Otherwise, you must give the drive name with either file name or with both names. If two names are given, they must agree. The following three forms are valid for renaming a disk file on drive E:

```
RENAME E:FILE.BAK=FILE.TXT
RENAME FILE.BAK=E:FILE.TXT
RENAME E:FILE.BAK=E:FILE.TXT
```

Alternatively, you can change the default drive to the one containing the file and then give the RENAME command without including a drive name.

If CP/M cannot find the original file name (the second parameter), it displays the error message

```
NO FILE
```

Two different files on the same disk cannot both have the same name. Therefore, if you try to rename a file to an existing name, CP/M will ask if you want to delete the original. Let us see how you can do this.

Suppose that there are two files, one named SAMPLE.TXT and the other named SAMPLE.BAK. You want to copy a new version of SAMPLE.TXT from another disk. But first you want to delete the file SAMPLE.BAK and rename the original file SAMPLE.TXT as SAMPLE.BAK. One way to do this is to erase SAMPLE.BAK and then rename SAMPLE.TXT as SAMPLE.BAK. However, it is not necessary to erase the backup file as a separate step. CP/M will do that for you during the renaming step. Simply give the command:

```
RENAME SAMPLE.BAK=SAMPLE.TXT <RETURN>
```

CP/M will ask for verification that you want to delete the existing file. After the command is executed, you can use PIP to copy the file from another disk.

The transient extension RENAME.COM is required when you omit the parameters or when you use the wild-card symbols ? and * to rename a group of files. These symbols must be used with care. They must appear in identical places in both file names. For example, all text files can be renamed to backup files with the command:

```
RENAME *.BAK=*.TXT
```

However, the command

```
RENAME *.BAK=SORT.TXT
```

is incorrect because the wild-card symbol appears in only one name.

ERASE

Deleting Files Since the storage space on a disk is limited, all of it may eventually be occupied by files. Therefore, it is sometimes

necessary to delete files you no longer need. The built-in command ERASE deletes a single disk file or a group of files. The format is

```
ERASE d:name
```

where d is an optional drive name. If the file name is omitted, CP/M will ask for it. But then the transient version ERASE.COM is needed.

The wild-card characters ? and * may be used (carefully) in the file name. For example, the command

```
ERASE PROG.TXT
```

will erase the single file PROG.TXT on the current drive. The command

```
ERASE PROG.*
```

will erase all files that have the primary name PROG—for example, PROG.TXT, PROG.BAK, and PROG.BK2. If you want to erase all files on a disk, give the double asterisk (*.*) as a parameter. Be sure to include a space after the ERASE command. The transient version, ERASE.COM, is required when wild cards are used.

Be careful when you use the ERASE command because a mistake can cause serious problems. For example, you can delete all backup files with the command:

```
ERASE *.BAK
```

However, if you inadvertently type *.BAS instead, CP/M will delete all BASIC files rather than all backup files.

If CP/M cannot find a file to match the parameter of ERASE, it displays the error message NO FILE.

Whenever wild-card symbols are included in the file name, there is a potential for disaster, since several files can be erased at one time. Therefore, CP/M requests verification by repeating the command. For example, if you give the command

```
ERASE *.BAS
```

CP/M responds:

```
ERASE *.BAS (Y/N)?
```

You must answer with a Y if you want CP/M to continue with the command. Otherwise, CP/M terminates the operation without further action.

Though CP/M asks for verification when wild-cards are included in the file name, it asks only once even when there are several files to be deleted. You can add an extra margin of safety by including the option parameter CONFIRM (which can be abbreviated to C). Then CP/M will request permission to erase each individual file name. The transient version, ERASE.COM, is required. For example, the command

```
ERASE *.BAK [CONFIRM]
```

will erase all backup files but ask specifically about each one in turn.

You can protect disk files from accidental erasure by setting the RO attribute, which is discussed later in this chapter in the section on SET. If you attempt to erase a protected file with the ERASE command, CP/M will refuse to perform the operation and will display an error message. If you want to erase a file that is already set to RO, you first have to change it to RW with the SET command.

Now that we have learned to use the built-in commands, let us go on to the important transient programs SETDEF, SHOW, SET, SUBMIT, and DEVICE.

Five Important Transient Programs

In this section, we will look at five of the most important transient programs. Though all the transient programs always have the COM file-name extension, we execute these programs by typing only the primary part of the name. The COM extension is

never given in the command. For example, earlier we executed PIP.COM by simply typing the primary name, PIP.

It is easy to confuse the built-in programs with the transients since both are executed similarly. You must always be aware of the difference. Remember that CP/M has only six built-in commands; all others are transient. You can execute a built-in program from any disk and any user area. However, if you want to execute a transient program that is not located on the current disk, you must include the drive name with the file name. If CP/M cannot find a requested transient program, it repeats the name and adds a question mark.

There are two ways of helping CP/M find transient programs. Earlier in this chapter we saw that transient programs can be declared as system files and stored in user area 0. These programs will then be available to all user areas on the same disk. The second method uses the command SETDEF.

SETDEF

Establishing a Search Path If you have more than one disk, it is convenient to keep all the transient programs on a single disk—say, drive A—but make another disk—say, E—the current drive. Then it is necessary to add a drive name to each transient command on drive A. For example, if drive E is current and you want to create a file on this drive with WordStar, a transient program located on drive A, this is the command:

```
A:WS REPORT.TXT
```

Notice that you must specify the drive in the transient command because E is the current drive.

However, CP/M Plus provides a mechanism for more easily locating transient programs. The program SETDEF establishes a search path so that CP/M automatically looks for a transient program on another drive or on a sequence of drives without having to be told.

Each time you turn on your computer, you will want to execute SETDEF with a drive parameter. For example, the command

```
SETDEF A:
```

directs CP/M to look on drive A for any transient program unless you have specified a drive. Of course, drive A must be current when this command is given because SETDEF.COM is on drive A. Additional drives can also be specified in this command. With SETDEF, the asterisk refers to the current drive. For example, the command

```
SETDEF A:,*
```

directs CP/M to look first on drive A and if the transient program cannot be found, to look on the current drive.

SETDEF can perform additional operations besides establishing a search path. These are summarized in Chapter 7.

SHOW

The transient program SHOW can reveal many different aspects of a disk. We will consider only three features in this chapter—finding the remaining disk space, determining the active user numbers, and determining the remaining directory space. Before we describe the SHOW program, let us consider how information is stored on a disk.

CP/M Records and Blocks CP/M stores information on a disk in 128-byte units called *records.* Each record has a unique track and sector number. However, the disk directory references a larger unit known as a *block* or *group* that is assigned a unique block number.

The block is the smallest amount of disk space allocated to a disk file. The disks used with the Commodore 128 have a block size of 1,028 bytes, or 1K bytes for short. Each 1K-byte block can hold a maximum of eight records, each containing 128 bytes. But there may be fewer. For example, a very small file might require only one sector, or 128 bytes. However, CP/M allocates a 1K block for this file nevertheless.

Now let us see how to obtain more information about disk files by using the program SHOW.

Determining the Remaining Disk Space We have seen that the DIR command can produce a listing of the disk directory. Furthermore,

if the option FULL or SIZE is included in the command, you get additional information—the size of each file, the total space occupied by the file, and the number of records. However, DIR does not give the remaining disk space. It is important to have this information. As you create more and more programs on a disk, all the available space will be used up. You must be careful not to continue working on a disk when there is no more room. Otherwise, you can lose the program you are working on. You can solve this problem with the program SHOW since it can display the remaining disk space.

To use SHOW, we must first understand the principle of resetting disks. Each time a disk is first accessed, CP/M reads the disk directory and makes a copy in memory. On subsequent accesses to this disk, CP/M assumes that the disk has not been changed and therefore it does not read the disk directory again. However, typing ^C resets the disks so that the directory will be read at the next access. You can observe this action by going to each disk in turn. For example, go to drive A, give the ^C command, then go to drive E. This drive will start up as CP/M reads the directory. Go back to drive A and then return to drive E. The drive will not start up this time. We say that the ^C command *resets* or *logs out* the disks. A drive is *logged in* when CP/M reads the directory. Now let us see how to use SHOW.

First, you must find which disk SHOW is located on. Because SHOW is a system file, it should be located on the system disk in drive A. Make sure A is the current drive and then give the ^C command to reset the disks. Drive A will turn on as CP/M reads the disk directory. Give the command:

```
SHOW <RETURN>
```

If SHOW is located on this disk, CP/M will copy it into memory and execute it. If CP/M cannot find the program named SHOW.COM, it responds with the error message

```
SHOW?
```

This message means that SHOW is not on the current disk or that you misspelled the name. Be sure that your system disk

containing SHOW.COM is located in drive A, that you are on drive A, and that you have typed the name SHOW correctly.

When SHOW is executed without a parameter, it reports the remaining free space for the currently logged-in drives. Since only drive A is currently logged in, the response will be something like

```
A: RW, SPACE: 144K
```

The letter A at the beginning of the line means that the information refers to drive A. The RW symbol means that the disk currently has a read/write status; that is, it is possible to both read from and write onto the disk. SPACE: 144K means that 144 kilobytes of space remain on disk (actually 144 times 1,024 bytes).

After SHOW has reported the condition of drive A, go to drive E by typing:

```
E: <RETURN>
```

Drive E should start up as CP/M reads the directory of this disk. Return to drive A and give the SHOW command again. In response to the command, SHOW will include information about disks A and E since both drives are logged in. For example, SHOW might display the following:

```
A: RW, SPACE: 144K
E: RW, SPACE: 166K
```

Type ^C to reset the disks and give the SHOW command again. This time, SHOW reports only on drive A since all the other disks have been logged out.

Now let us look at a second feature of SHOW.

Locating the Active User Areas We have seen that the current user does not normally have access to files in other user areas. Therefore, although the DIR command can display the disk files located in the current user area, it does not show what files are

present in other user areas of the disk unless the USERS option is given.

Suppose that you want to run a program located on a particular disk, but you do not know in what user area it is located. If you run SHOW with the USERS option, it can identify the current user number and also list other user numbers on the disk. The option can be abbreviated to the letter U. In addition, SHOW indicates how many files belong to each user. As usual, the option is enclosed in square brackets. For example, the command

```
SHOW E:[USERS]
```

might produce the listing

```
Active User:  0
Active Files:  0  1  2
E: # of files:  24  3  1
E: Number of free directory entries:      36
E: Number of time/date directory entries: 16
```

showing that the current user is 0, who has 24 files on drive E. User 1 has 3 files, and user 2 has 1 file. The fourth line of the display shows that there are 36 remaining spaces in the directory for additional entries. (We will look at directory space in the next section.) The last line will appear only if time and date tagging have been established (see the SET command in Chapter 7).

Two other SHOW options are DRIVE and LABEL. DRIVE produces a listing of the disk characteristics, including total capacity, directory space, and block size. The LABEL option is used to display the unique name that can be assigned to each disk. An error message is given if no name has been assigned.

Determining the Remaining Directory Space The space available on each disk is limited in two ways. One limit is the total storage space that can be used by programs. We just learned that the program SHOW can be used to determine the remaining disk space. A second limitation is set by the directory size. For the Commodore 128, a maximum number of 128 directory entries are allowed for each different disk.

When a program is written to a disk, for example by copying with PIP, the directory entry records the name, the user number, and the location. Large programs may require additional directory entries called *extents* to keep track of all the parts.

Directory space is limited. When all the directory entries have been used, there is no room to record additional names. Some programs, such as PIP, stop operations and display an error message when there is no directory space, while others may ignore this aspect and give unpredictable results.

When you are working on a disk, you should use SHOW frequently to determine that you are not running out of space. There are several ways to determine the remaining directory space. The next to last line in the previous SHOW listing illustrates one way. This line gives the number of remaining entries. You may have noticed a second method.

When you give the DIR command with option parameters, a summary is shown at the end of the listing. This includes the number of bytes, records, and blocks and the number of files shown in the listing. In the last line, two numbers separated by a slash give the number of used directory entries and the number of allocated entries. To recall this method, make drive A current and enter the command:

```
DIR[SIZE]
```

SET

SET is a transient program that can do many different tasks. The most important use is changing the file attributes *read only* or *read/write* and system or directory. You can also set an entire disk to read only or read/write status. In addition, you can assign a symbolic name and password to each disk and individual passwords to each file to reduce the chance of unauthorized use.

Changing the File Attributes Earlier in this chapter we learned how to determine the file attributes read only or read/write and directory or system using the DIR command with options. Now let us see how to change these attributes with the SET command.

The command-line tail we use to change the attributes is the same as the one we use with DIR to determine the attributes. For example, the command

```
DIR *.COM[SYS,RO]
```

displays a listing of all COM files that are marked both as system files and as read-only files. Therefore, the command

```
SET *.COM[SYS,RO]
```

will designate all COM files as system files and as read-only files. As with DIR, the four options are SYS, DIR, RO, and RW (see Table 3.2). Remember that only two of the four attributes can be set at a time since RO and RW are mutually exclusive, as are SYS and DIR.

Designating a Disk as Read Only We have seen that setting the RO attribute will protect individual files or groups of files from accidental erasure. A special indicator, or *flag,* is set in the directory when a file is designated RO. Sometimes it may be more convenient to set an entire disk to read-only status. You do this by omitting the file name from the command and giving just the drive name. For example, the command

```
SET E:[RO]
```

write protects all files on drive E. Note that this is a temporary designation; no change is made to the directory. The disk is automatically reset to read/write status at the next warm start. Let us see how this works.

Set drive E to read-only status with the above command. Then execute SHOW. You will see that drive A has read/write status, but drive E is read only. Give the ^C command to reset the disks and then execute SHOW again. (It is not necessary to retype the previous command. Simply type ^W <RETURN> and CP/M will redisplay the SHOW command.) The results now show drive E to be read/write again. In other words, ^C has returned drive E to read/write status.

Assigning a Disk Label Because floppy disks can be removed from the drives, it is easy to get them mixed up. One way to avoid this potential problem is to assign a name or label to each. Suppose, for example, there are five staff members in an office and each has a separate disk for correspondence. It would be logical to assign the name of each member to the corresponding disk. You can do this with the SET program.

Disk labels follow the same rules as file names. Letters of the alphabet and the ten digits can be used. The primary name can contain up to eight characters. The optional extension can have as many as three characters. For example, the command

```
SET E:[NAME=JONES]
```

uses the NAME option to assign the label JONES to the disk in drive E. Of course, if E is the default drive, the drive letter can be omitted from the command.

The assigned label does not appear in the DIR listing even when the FULL option is given. However, the label can be displayed with the LABEL option of SHOW. (The LABEL option can be abbreviated to simply L.) For example the command

```
SHOW E:[LABEL]
```

displays the label assigned to the disk in drive E.

In the next sections we consider password protection of files. If you do not need to prevent others from accessing your files, you can skip this discussion.

Assigning a Disk Password The possibility of unauthorized access to a disk file can be reduced by assigning a password to each file. But first it is necessary to assign a password to the entire disk. Let us see how to do this. (Again, remember that you can omit the drive letter if E is the current drive.)

The command

```
SET E:[PASSWORD=ORION]
```

assigns the password ORION to the disk in drive E. If a disk

name (label) has not been previously assigned, CP/M will automatically choose the unimaginative name LABEL.

If you attempt to change the disk label or disk password after a password has been assigned, CP/M will ask for the password. You must give the correct password at this point, or CP/M will not make the change. Note that when you enter the password in response to this question, the password does not appear on the screen.

It is very important to realize that if you forget the password, there is *no way* to retrieve it from the system. Be sure to keep a separate record of any passwords you assign so you don't risk losing access to your files.

Password protection is removed with the command:

```
SET E:[PASSWORD=]
```

That is, the password option is given without a value.

After assigning a password to the disk, it is necessary to activate file protection before assigning passwords to individual files. The command

```
SET E:[PROTECT=ON]
```

turns on this feature. The value OFF instead of ON deactivates this feature. Again, CP/M asks you to supply the password in order to execute the command.

Assigning File Passwords After a disk has been assigned a password and the protection option has been turned on, it is possible to protect individual files on the disk. The command is similar to the one for disk protection. For example, the command

```
SET SORT.BAS[PASSWORD=SORT]
```

assigns the password SORT to the file SORT.BAS. If you try to assign a password that is already assigned to another file, CP/M will give you an error message.

Setting the Mode of Protection Four different modes of password protection can be selected with the PROTECT option. For example

```
SET SORT.BAS[PROTECT=READ]
SET SORT.BAS[PROTECT=WRITE]
SET SORT.BAS[PROTECT=DELETE]
SET SORT.BAS[PROTECT=NONE]
```

sets the type of protection for SORT.BAS. When you choose READ, the default option, the password is required to read the file (with TYPE), alter the file (with ED), copy the file (with PIP), or rename the file (with RENAME). When you rename a file with a password, the password remains with the file. This is not true when you copy a file with PIP.

If you select the WRITE option, the password is not needed to display or copy the file, but it is needed to alter or rename it. The DELETE option requires a password only for deletion of the file. The NONE option removes password protection.

If a password is needed, CP/M may automatically ask for it. If CP/M doesn't request the password, enter it immediately after the file name, with a semicolon separating the two parts. For example, if the password for SORT.BAS is PASS, you can examine this file with the command:

```
TYPE SORT.BAS;PASS
```

SUBMIT

Executing Commands from a Disk File Up to now we have given all our commands to CP/M by entering the information from the keyboard. This is a convenient technique for entering one command at a time. However, sometimes it is necessary to give a sequence of commands or to repeat a command several times. In such cases, you can write the commands into a disk file and then direct CP/M to execute the commands directly from this file

instead of from the keyboard. The result is the same as if the commands were entered from the keyboard.

In this section you will learn how to use the transient program SUBMIT to execute a sequence of commands that are stored in a disk file. This technique is called *batch processing.* To process batch commands with SUBMIT, you write the commands into a disk file that has the file extension SUB. One precaution—the disk must not be full because SUBMIT uses working space on the disk during the batch processing.

Let us investigate the operation of SUBMIT with a simple example. We will begin by using the PIP program to create a short file, which we will call TEST.SUB. (If you know how to use the system editor, ED, which we will discuss in Chapter 5, you can use it instead of PIP to create the file.) Be sure to type your information very carefully since you cannot correct mistakes with PIP.

If you are not already on drive A, go there and give the command:

```
PIP TEST.SUB=CON: <RETURN>
```

Then type these three lines:

```
DIR <RETURN><LF>
DIR E: <RETURN><LF>
DIR[FULL] <RETURN><LF>
```

Recall that the symbol <LF> means you must press the LINE FEED key after the RETURN key.

Next, type a ^Z to terminate PIP and return to CP/M. Inspect the new file with the command:

```
TYPE TEST.SUB <RETURN>
```

If you find an error, you can correct it with ED or recreate the file with PIP. If it looks correct, execute the three commands in the SUB file by giving the command:

```
SUBMIT TEST <RETURN>
```

Remember, drive A must be the default drive, and the parameter TEST must refer to a file located on drive A that has the filename extension SUB.

As SUBMIT encounters each line in the SUB file, it will display the line on the video screen exactly as if you had typed it on the keyboard. Even the A> prompt will be shown. CP/M will then execute each command line in turn. Because the display moves so quickly, you may not have time to see your commands on the screen. Remember that you can use ^S to stop the display and then ^Q to start it again. If you want to terminate SUBMIT before it has finished, press ^C.

Our example showed only one simple application for SUBMIT. You will be able to use it for many more important applications. For example, suppose you want to obtain a printed listing of five different text files located on drive E. You can create a file called LIST.SUB on drive A that executes PIP five times. The file might look like this:

```
PIP LST:=E:FILE1.TXT
PIP LST:=E:FILE2.TXT
PIP LST:=E:FILE3.TXT
PIP LST:=E:FILE4.TXT
PIP LST:=E:FILE5.TXT
```

Turn on your printer and give the command:

```
SUBMIT LIST <RETURN>
```

Then you can go for a coffee break while the files are printing.

Automatic Execution on Startup Each time you turn on your computer, CP/M Plus looks for a disk file named PROFILE.SUB. If it is present, CP/M directs SUBMIT to execute the commands that are present. In this way, the system can automatically do various housekeeping tasks that are needed each time the computer is turned on. If the file PROFILE.SUB is not present, CP/M continues normal operation.

As with the previous example, you must create the PROFILE.SUB file with the system editor or with PIP. A good candidate

for PROFILE.SUB is the command:

```
SETDEF A:,*
```

This command directs CP/M to look first on drive A for a transient program. If the program cannot be found on A, CP/M will look on the current drive. Be sure that PROFILE.SUB is located on disk A if you want CP/M to execute it at each warm start.

Batch Processing with Dummy Parameters Another powerful feature of SUBMIT is the ability to interpret dummy parameters in the SUB file. A *dummy parameter* is a symbol that is replaced by the actual parameter at execution time. A dummy parameter in a SUB file is referenced by a dollar sign followed by a number, starting with 1. This allows slight variations in interpretation of the commands.

As an example, suppose we have a file called PRNT.SUB containing the line:

```
PIP LST:=FILE.TXT
```

The command

```
SUBMIT PRNT
```

will send the file named FILE.TXT to the printer. However, suppose we change the submit file to

```
PIP LST:=$1
```

That is, we replace the file name with the dummy name $1. Then if we give the command

```
SUBMIT PRNT FILE.TXT
```

the result will be the same as before. SUBMIT substitutes the second parameter FILE.TXT for the dummy parameter $1.

Our SUB file is now more versatile because of the dummy parameter. For example, we can print a file called SORT.BAS by giving the command:

```
SUBMIT PRNT SORT.BAS <RETURN>
```

This time, SUBMIT issues the command

```
PIP LST:=SORT.BAS
```

because SUBMIT replaced the dummy parameter $1 by the second parameter SORT.BAS.

In the previous example we used a dummy parameter for a file name. However, dummy parameters can replace any set of characters, not just a file name. We could have included the drive name in the parameter as well as the file name. For example, the command

```
SUBMIT PRNT E:SORT.BAS
```

will print the file located on drive E.

Additional dummy parameters can be added as needed. They appear in order as $1, $2, $3, and so forth.
For example, suppose we change our SUB file to

```
PIP LST:=$1
PIP LST:=$2
PIP LST:=$3
```

so there are three dummy parameters. The corresponding command

```
SUBMIT PRNT FILE1 FILE2 FILE3
```

directs CP/M to execute SUBMIT. The first parameter tells SUBMIT to use the file PRNT.SUB. The remaining parameters are to be substituted for the corresponding dummy parameters $1, $2, and $3. Thus SUBMIT will generate the following lines.

```
PIP LST:=FILE1
PIP LST:=FILE2
PIP LST:=FILE3
```

A dollar sign is used in the SUB file to designate a dummy parameter. However, you may sometimes need to use a dollar sign as a regular symbol in a command line. SUBMIT resolves this potential conflict of symbols in an interesting way. When a dollar sign is needed as a regular character in a SUB file, you must supply two dollar signs. The double dollar sign becomes a single dollar sign in the final command line presented to CP/M's command processor.

Sometimes it is necessary to include a control character in a command. However, it is not convenient to place a control character in a text file because CP/M is likely to interpret the character as a command, and execute it. In particular, CONTROL-Z is always interpreted as an end-of-file indicator and CONTROL-C may be interpreted as a warm start. To resolve this problem, designate control characters in SUB files with the ↑ symbol (press the white ↑ key on the Commodore 128 keyboard). That is, CONTROL-C is selected by entering two characters—the ↑ symbol and the letter C. Similarly, CONTROL-Z is obtained by giving both the ↑ symbol and the letter Z.

Input to an Executing Program We have seen how SUBMIT can execute regular CP/M commands from a disk file as though the commands were entered directly from the keyboard. It is also possible to include a SUBMIT command as a line in another SUBMIT file. That is, one SUBMIT file can initiate the execution of another SUBMIT file.

All the commands we have considered so far are typed on the command line in response to the CP/M prompt. Sometimes, however, we need to give information to a program after it has begun execution. Suppose, for example, we run PIP without a parameter and then give the following sequence of commands to PIP's asterisk prompt.

```
PIP
LST:=FILE1.TXT
LST:=FILE2.TXT
LST:=FILE3.TXT
LST:=FILE4.TXT
LST:=FILE5.TXT
```

The first line, containing the command PIP, is given in response to the CP/M prompt. The remaining lines, however, respond to the PIP prompt. They cannot simply be placed into a SUB file since they are input to PIP not to CP/M. Rather, they must be specially marked to tell SUBMIT that they are not commands to CP/M. The < symbol is placed at the beginning of the line for this purpose. Thus, a SUB file to perform the steps looks like this:

```
PIP
<LST:=FILE1.TXT
<LST:=FILE2.TXT
<LST:=FILE3.TXT
<LST:=FILE4.TXT
<LST:=FILE5.TXT
<
```

Notice that there is a final < symbol to terminate PIP and return to CP/M.

DEVICE

Logical and Physical Devices The peripherals such as the keyboard, video screen, printer, and phone modem are attached to the computer through *ports.* Each port, and, by extension, the corresponding peripheral attached to the port, is called a *physical device.* The software that drives a port refers to a *logical device.* CP/M incorporates software for the logical devices CON: (for console—keyboard and screen), LST: (for printer), and AUX: (for an auxiliary device such as a modem). Notice that we have already used CON: and LST: in PIP commands.

The name CON: is a shorthand way of referring to two separate sets of routines. These are the routines for console input from the keyboard (CONIN:) and the routines for console output to the screen (CONOUT:). The name AUX: also refers to auxiliary input (AUXIN:) and auxiliary output (AUXOUT:) routines. Both the console and auxiliary ports are *bidirectional;* that is, they can transfer both ways, output to the peripheral and input from the peripheral. By contrast, the printer is only an output port. Therefore, LST: refers to only one routine.

Let us see how to use the DEVICE program. Give the command

```
DEVICE NAMES
```

to determine the physical device names for the Commodore 128. The result is

```
KEYS  NONE I      80COL  NONE O      40COL  NONE O
PRT1  NONE O      PRT2   NONE O      6551   9600   IOSX
```

identifying six physical devices—keyboard (KEYS), two printers (PRT1, PRT2), two video screens (80COL, 40COL), and serial connection to the disk (6551). Notice that a colon is not used at the end of these physical device names. The first and last items (keyboard and disk) contain an I designating an input device, while all but the first have an O to designate an output device. The serial device is designated with an S and has a transfer rate of 9600 baud. The other devices are parallel rather than serial so they do not have an associated transfer rate. Any of these physical devices may be assigned to the five CP/M logical devices.

Let us next determine the current assignment. Give the command:

```
DEVICE VALUES
```

The result might be

```
CONIN:   =  KEYS
CONOUT:  =  80COL
```

```
AUXIN:    =  Null Device
AUXOUT:   =  Null Device
LST:      =  PRT1
```

showing that console input is read from KEYS, and that the keyboard and console output goes to 80COL, an 80-column video screen. Auxiliary input and output could go to a phone modem. However, in this example there is no assigned device. Printer output goes to the parallel printer named PRT1.

If you execute the DEVICE command without a parameter, both the NAMES listing and the VALUES listing are given. Then the program waits for your input. You can then change the current assignment or you can change the characteristics of the corresponding ports.

Sending Information to More Than One Device Normally, one physical device will be assigned to each logical device. However, the DEVICE command can map more than one physical device to each logical device. For example, the command

```
DEVICE CONOUT:=80COL,PRT1
```

will send the console output simultaneously to the video screen and to the printer. Return the mapping to its regular assignment with the command:

```
DEVICE CONOUT:=80COL
```

It appears that, in the previous example, mapping the console output to both the video screen and the printer is the same as pressing ^P. However, there is a subtle difference in the way the mapping is accomplished. For example, ^P cannot be used to engage the printer when certain programs such as Microsoft MBASIC are executing. By contrast, CONOUT: can be mapped to both the video screen and the printer before executing MBASIC. Then the console output will appear on both the video screen and the printer.

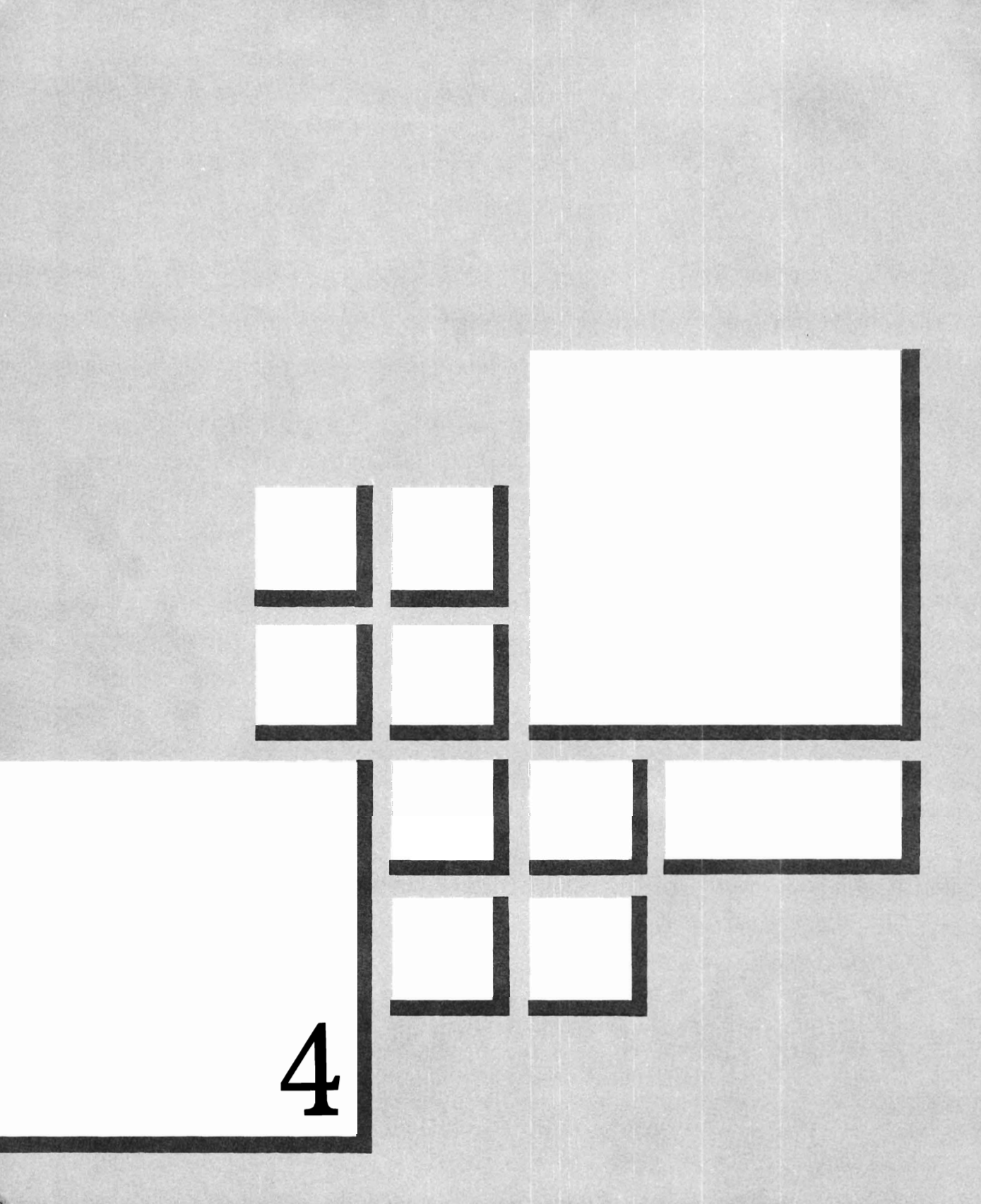
If you find that you always need to issue the same DEVICE commands each time you turn on your Commodore 128, place

these commands in your PROFILE.SUB file so they will be executed automatically each time the computer is turned on.

Summary

In this chapter, we have considered the most frequently used CP/M commands. We defined commands, parameters, and options, and discussed the difference between built-in commands and transient commands. You should now know how to use the built-in commands TYPE, USER, DIR, DIRSYS, RENAME, and ERASE; and the transient commands SETDEF, SHOW, SET, SUBMIT, and DEVICE.

A summary of these commands and all the other CP/M Plus commands, is presented in reference form in Chapter 7.


4

Handling Files with PIP

PIP.COM, or PIP for short, is a separate program that is included on the CP/M Plus system disk that came with your Commodore 128. We have already studied some of the most useful PIP commands in earlier chapters. Because PIP is one of the most important utility programs, we will devote this entire chapter to learning more about it.

PIP is primarily a file transfer program. As its name suggests, (the letters PIP stand for Peripheral Interchange Program), PIP is used to transfer files from one device, such as a disk, to another,

such as the video screen. However, it has many other features. Among other functions, PIP can do the following:

- Copy a single file from one disk to another
- Copy a group of files from one disk to another
- Copy all files from one disk to another
- Create a file that is a duplicate of another file but has a different name
- Format text that is printed
- Reduce lines that are too long
- Print a group of files with just one command
- Join several files into a single file
- Extract a portion of a text file
- Translate a file to uppercase letters
- Translate a file to lowercase letters
- Reset the parity bit
- Add line numbers to a file
- Show a file on the screen during transfer
- Copy system files
- Copy files from one user area to another
- Copy files that have not been previously copied

Though you may need only a few of PIP's capabilities, it is important to know what features are available. Therefore, you should read through this chapter completely, and then go back and study in detail the sections of specific interest to you or use the chapter for reference.

Copying Files

We will look first at PIP's most important function: copying files from disk to disk. As we have seen, PIP can copy a single file or a group of files. Let us begin with the single-file transfer.

Copying Single Files

In the previous chapters, you learned to execute CP/M programs by typing the primary name on the command line. Usually, we have included parameters on the command line with PIP. For example, in Chapter 2 we gave the command

```
PIP COPY.TXT = SAMPLE.TXT[V]
```

to make a copy of the program SAMPLE.TXT and name the new version COPY.TXT. This command contains three parameters—two file-name parameters and one option parameter. Notice that the new name comes first, and then the original name follows. Spaces on either side of the equal sign are optional. In this example both the original file and the new file are on the same disk—the current disk (also known as the default disk) since the drive name was omitted from both file names.

The third parameter, V enclosed in square brackets, is an option parameter that stands for verify. It directs PIP to read the newly created file and compare it to the original file to ensure that it is correct. (PIP gives an error message if the copy is not correct.) You should always include the V option unless one of the devices is not a disk. Then you must not include the V option, since there is no way to verify the information. Whenever you use option parameters, there must never be a space before the left square bracket; that is, the bracket must immediately follow the file-name parameter. The right bracket, however, may be omitted.

When a file name refers to the current drive, you may omit the drive name, as illustrated in the previous example. There is no harm in including the drive name when you refer to the current drive, but it is not necessary. On the other hand, the drive name must be included when the file is not on the current drive. Consider, for example, the command:

```
PIP E:COPY.TXT = A:SAMPLE.TXT[V]
```

This command copies the file SAMPLE.TXT from drive A to drive E and names the new copy COPY.TXT. Since both drives are specified, it does not matter what the current drive is.

In the previous two examples we have done two things at once—copying a file and changing its name. If we want the new file to have the same name as the original file, we can simplify the command by omitting the copy name. In this case, we must give the drive name for the new file. For example, the command

```
PIP E:=A:SAMPLE.TXT[V]
```

copies the file SAMPLE.TXT from drive A to drive E. If the original file is on the current drive, we can omit the drive name. Thus, if the current drive is A, the previous command could be shortened:

```
PIP E:=SAMPLE.TXT[V]
```

This is the most useful form for copying a single file.

Executing PIP without Parameters PIP can be executed either with or without parameters. When you execute PIP by including parameters on the command line, PIP does the requested task and then returns control to the CP/M operating system so you can give another command. However, when you execute PIP without a parameter, it displays its own prompt of * and awaits further commands. You can then give PIP a sequence of copy operations to do. After each operation, control returns to PIP rather than to CP/M. With this method the parameters you enter appear just as they would if you had given them on the command line with PIP. For example, you could execute the command in the previous section on two lines:

```
PIP <RETURN>
E:=SAMPLE.TXT[V] <RETURN>
```

This method is more complicated when you want to give only one copy command. However, it is convenient when you want to give several PIP commands, since CP/M does not have to reload PIP after each step. In either case PIP does its task in the same way.

The two methods of executing PIP respond differently if there

is an error in your parameters. If PIP cannot find the original file you want to copy, it displays the message

```
ERROR: FILE NOT FOUND
```

and prints the name of the file it was looking for. If you included parameters with PIP on the command line, PIP then completes its operation and returns to CP/M. However, if you initially executed PIP without a parameter, it does not terminate after displaying the error message. Rather, it gives the * prompt, showing that it is ready to accept another command. PIP also responds differently to the Q and S options depending on how it was executed, as we will see later in this chapter.

After PIP displays the * prompt, you give a command. PIP does the task and then gives the * prompt again. You can now enter another set of parameters for a second copy operation. At the end of this step, the * prompt appears again. You can continue in this way with an unlimited number of copying operations. For example, you can copy four files from drive A to drive E with the sequence:

```
PIP <RETURN>
E:=A:FILE1.TXT[V] <RETURN>
E:=A:FILE2.TXT[V] <RETURN>
E:=A:FILE3.TXT[V] <RETURN>
E:=A:FILE4.TXT[V] <RETURN>
```

When you are finished with PIP and want to return to CP/M, simply respond to the * prompt by pressing the RETURN key and the CP/M prompt will appear.

Rules for File Transfers We will now present the rules for transferring files with PIP. To copy a file from one disk to another, use this form for the parameter:

```
d:copy=d:original[V]
```

In this example, *d* stands for the drive, *copy* for the new name of the copy file, and *original* for the name of the original file. The two drive names may refer to the same drive or to two different ones.

If you want the copy file to have the same name as the original file, you can omit the copy name. This is the abbreviated form:

```
d: = d:original[V]
```

When only one name is given, PIP will assume that the name of the new file is the same as the original. When you use this form, the two drive names will usually be different. For example, the command

```
E: = A:TEST.BAS[V]
```

will copy TEST.BAS from drive A to drive E.

Remember, the drive name may be omitted when it refers to the current drive. For example, the command

```
E: = TEST.BAS[V]
```

copies the file TEST.BAS from the current drive (drive A) to drive E.

Remember that you cannot have two files with the same name on the same disk. If you give the command

```
A: = A:TEST.BAS[V]
```

PIP will make a copy of the file TEST.BAS. However, when PIP is done, it will erase the original file. No harm done, but no duplicate copy either. Later in this chapter, we will consider a use for this technique.

Let us look at some additional examples. Assume that the current drive is A and that the file FILE1.BAS is on drive A and the file PROGRAM.TXT is on drive E. When you execute PIP without a parameter, are the following PIP commands legal?

(1) `A: = E:PROGRAM.TXT`
(2) `E: = FILE1.BAS`
(3) `A:FILEREV.BAS = A:FILE1.BAS`
(4) `A:FILE1.TXT = FILE1.BAS`
(5) `E:FILE1.BAS = FILE1.BAS`

All the above PIP commands are proper. Notice that the command in example (2) is equivalent to

```
E:=A:FILE1.BAS
```

because A is the current drive. Remember that you can omit the drive name when referring to the current drive if you give the file name.

The drive name A: could have been omitted from both file names in example (3). In (4) FILE1.TXT is not the same as FILE1.BAS; therefore, both files will exist on A after the operation. Example (5) could have been abbreviated to look like example (2).

The following commands are improper:

```
(1)  =E:PROGRAM.TXT
(2)  A:=FILE1.BAS
(3)  E:=A:PROGRAM.TXT
(4)  FILE1.BAS=FILE1.BAS
(5)  E:FILE1.BAS=E:FILE1.BAS
```

Example (1) is incorrect because neither a drive name nor a file name is given for the first parameter. Example (2) is incorrect because both parameters refer to the same drive and file name since A is the current drive. The first two examples produce the PIP messages

```
ERROR: INVALID DESTINATION
```

and

```
ERROR: INVALID FORMAT
```

The third example is improper since it is trying to copy in the wrong direction—from A to E. The file is located on drive E rather than A. In this case PIP gives the message

```
ERROR: FILE NOT FOUND—A: PROGRAM.TXT
```

PIP will do the operations shown in examples (4) and (5). However, the result will be useless since the copy name is the same as the original and both refer to the same drive. Later we will consider adding an option parameter that will make operations (4) and (5) meaningful.

We have been copying individual files using specific file names. Now let us see how to copy groups of files.

Copying Several Files with the Wild-Card ? and * Symbols

In the previous section we copied several files one at a time by first executing PIP without a parameter and then giving each copy command separately. There is another way to copy a group of files, as we saw in Chapter 2 when we copied all the files on disk A over to disk E with the command:

```
PIP E:=A:*.*[V]
```

A file name with wild cards contains the ? or * symbols. As we have seen, the question mark matches any single character, including a blank, in the given position. The asterisk tells PIP to interpret the remainder of the field (the primary name or extension) as though it were filled with question marks.

We can use wild cards in the second PIP parameter, the name of the original file. Then the first parameter, the name of the copy file, must be simply a drive name such as A: or E:. For example, the command

```
PIP E:=SORT?.*
```

will copy all files that have SORT for the first four letters if the primary name has four or five letters—for example, SORT.BAS, SORT.BAK, SORT, and SORT2.BAS.

When the second parameter contains wild cards, PIP will copy all files that match the name. In this case, PIP displays

```
COPYING -
```

on the video screen, confirming that wild cards were entered. Then, each time PIP locates a file that matches the parameter, that file's name is displayed on the video screen and the file is copied. If PIP cannot find any file to match the parameter, it displays the message

```
ERROR: FILE NOT FOUND
```

and displays the parameter.

For the next section we will need four files to practice with. We will now create them with PIP. Since we are going to use only the file names, not the contents of these files, we will trick CP/M into creating empty files, or files that contain no information.

Execute PIP without a parameter. In response to the * prompt, give the command:

```
FILE1.TRY=CON: <RETURN>
```

Be sure to include the colon after CON. CP/M places all the characters you type next into the new file. When you enter a ^Z to mark the end of the file, CP/M places this character, too, into the file and then closes the file. Since we want an empty file, do not type any regular characters, just type a ^Z. You should then see the * prompt of PIP again. Continue in this way creating three additional files:

```
FILE2.TRY=CON: <RETURN>
FILE3.TRY=CON: <RETURN>
FILE44.TRY=CON: <RETURN>
```

After you have created the fourth file, finish PIP by pressing the RETURN key. Check the directory with the command:

```
DIR FILE*.TRY[FULL] <RETURN>
```

The listing will show these four new files. The number of bytes and number of blocks should be zero, since the files are empty and thus will not take up any disk space. (Of course, they do take directory space.)

We can copy the first three of these files from drive A to drive E with the following sequence of commands:

```
PIP <RETURN>
E:=A:FILE1.TRY <RETURN>
E:=A:FILE2.TRY <RETURN>
E:=A:FILE3.TRY <RETURN>
```

However, the command

```
E:=A:FILE?.TRY <RETURN>
```

does the same transfer with just one command because of the ? matching character. And if drive A is current, you can omit the A: drive name from the second parameter as well.

Try this command and watch it work. Notice that the file FILE44.TRY is not copied because it does not match the parameter. If there had also been a file on drive A named FILES.TRY, it, too, would have been transferred.

The other matching character, the asterisk, is even more powerful. It will match any combination of characters in the remainder of its field, regardless of the length. Remember, CP/M expands the * symbol internally to a string of question marks. For example, if we wanted to transfer all four of the previous files with PIP, we could have given the command:

```
PIP E:=A:FILE*.TRY
```

As another example, assume drive A contains these files:

```
FILE1.TRY
FILE12.TRY
LETTER.TXT
CBASIC.INT
FILE1.BAK
```

The parameter *.TRY will match these files:

```
FILE1.TRY
FILE12.TRY
```

The name FILE1.* will match these files:

```
FILE1.TRY
FILE1.BAK
```

The expression *.* will match all the files in the directory.

We have now learned how to copy both a single file and a group of files using both specific names and names containing wild cards. Next, we will learn ways to combine files.

Concatenating Files

So far, we have been using PIP parameters to ensure that the copy was a faithful reproduction of the original. However, sometimes we need to combine two or more files into a new file. This procedure is called *concatenation*. Before we go into the details of concatenation, let us consider the two types of disk file.

Text and Nontext Files

Disk files may be divided into two types—text files and nontext (binary) files. Text files are prepared by the user and include such varied items as letters, reports, and computer programs written in BASIC, Pascal, and FORTRAN. As we saw in Chapter 3, text files are written with ASCII characters—the letters; the digits; the special symbols such as $, %, &, and *; and certain control characters such as the carriage return, the line feed, and the form feed. (Appendix C presents the full set of ASCII characters.) CP/M uses the ^Z symbol to mark the end of the text in a text file although this symbol is not displayed. No text may extend beyond a ^Z. Therefore, ^Z cannot be used as a valid character within a text file.

Binary files are different from text files in that they can contain every possible combination of characters, including the ^Z character. Consequently, ^Z does not mark the end of a binary file. COM files are binary files. PIP always assumes a binary file when making a copy of any file so the copy is always correct. However, when concatenating files, PIP assumes it is working with text files.

Concatenating Text Files

The PIP symbol that indicates concatenation is the comma. This is why you should not use a comma in a file name. Here is a simple example of concatenation:

```
PIP BIG.TXT=PART1.TXT,PART2.TXT
```

This command creates the new file BIG.TXT by combining the two files PART1.TXT and PART2.TXT. In this example all three files are on the current drive, since no drive name is included with the file names. The two original files still exist unchanged. Of course, there must be room on the disk for the new file as well as the original files.

As we have seen, the ^Z symbol always marks the end of a text file. After concatenation, the new file contains the text of the first file up to but not including its ^Z character, followed by the text of the second file. Thus, the information in the new file appears in the exact order of the file names to the right of the equal sign.

You can concatenate more than two files by including additional file names, each separated from the next by a comma. For example, the command

```
PIP BIGGER.TXT=PART1.TXT,PART2.TXT,PART3.TXT
```

joins three text files. Again, all files in this example are on the current drive.

If you want PIP to verify that the concatenation is correct, you must include the V parameter after *each* file name. In the previous example, the V symbol must be inserted in three places. The command

```
PIP BIGGER.TXT=PART1.TXT[V],PART2.TXT[V],PART3.TXT[V]
```

will concatenate three files and verify that all three parts have been correctly incorporated into the new file.

You can concatenate files from different disks by including drive names with the file name. For example, the command

```
PIP A:BIG.TXT = E:PART1.TXT[V],E:PART2.TXT[V],E:PART3.TXT[V]
```

combines three files from drive E into a new file on drive A and verifies that the copy is correct.

Sometimes it is necessary to append one or two files to the end of another file with the new file having the same name as one of the original files. For example, the command

```
PIP FIRST.TXT = FIRST.TXT,SECOND.TXT,THIRD.TXT
```

combines the files FIRST.TXT, SECOND.TXT, and THIRD.TXT into a temporary file. If the concatenation is successful, the original file FIRST.TXT is deleted, and the concatenated version is given the original name FIRST.TXT. Of course, the files SECOND.TXT and THIRD.TXT continue to exist after the concatenation, but the original FIRST.TXT is erased.

A practical hint: when concatenating files, first use the SHOW command to make sure that there is enough room on the disk for the new file.

Now let us see how to concatenate nontext files.

Concatenating Nontext (Binary) Files

When PIP encounters a ^Z character during a concatenation step, it normally assumes that it has found the end of the file. However, as we have seen, ^Z does not mark the end of a binary file. So there is a potential problem when PIP concatenates binary files because ^Z is a valid character.

Since the most common type of binary program is the COM file, PIP has been programmed to ignore any ^Z it finds in a COM file and continue copying to the actual end of a file. Therefore, you can confidently concatenate COM files, although this is not very common. On the other hand, binary files with extensions other than COM will be incorrectly concatenated if they contain a ^Z character.

The way to solve this concatenation problem is to add the O option whenever you use PIP to concatenate binary files other than COM files. (The letter O stands for *object file,* another name

for a binary file.) You do not need to use the O option when concatenating COM files, though it will do no harm. However, you must *not* use the O parameter when concatenating text files, or they will be incorrectly joined. It is not necessary to give the O parameter when you simply *copy* binary files, even if the file type is not COM. PIP always copies the entire file.

Inserting Characters During Concatenation

It is possible to insert characters from the keyboard during a concatenation step. For example, suppose you give the command:

```
PIP NEW=FILE1,CON:,FILE2,CON:,FILE3
```

This command will direct PIP to create the file NEW, beginning with the existing file FILE1. Then, the CON: parameter tells PIP to read any characters you enter from the keyboard and place them in the new file. The CON: symbol stands for console, another name for the keyboard and video screen. Remember to use both <RETURN> and <LF> at the end of each line you enter from the keyboard. When you signal the end of the keyboard entry with a ^Z, PIP copies FILE2. PIP then reads characters from the keyboard until you type another ^Z. Finally, PIP copies FILE3. The characters you enter from the keyboard are displayed on the video screen. Be sure to enter a line feed after you press the RETURN key and remember that you cannot correct errors when typing characters into a file.

Now let us see how to send disk files to a peripheral device such as the video screen or printer.

Transferring a Disk File to a Peripheral Device

PIP provides general-purpose transfer capabilities that allow a file to be copied not only from disk to disk, but also between various devices such as the console (keyboard and video screen), printer, and phone modem. Let us now consider this mode of transfer. We begin with transfer from the console.

Creating a File

As we have seen, it is possible to transfer information from the keyboard to a disk file. While concatenating disk files we added characters between the files. It is also possible to create a new file. In Chapter 2, we did this operation with the command

```
PIP SAMPLE.TXT=CON:
```

where CON: refers to the (console) keyboard. (If you did not create this file in Chapter 2, please refer to that chapter and do so now. We will work with this file in the following section.)

The first parameter of this command, SAMPLE.TXT, is the name of the newly created disk file. The second parameter, CON:, refers not to a disk file but to the keyboard. The colon at the end of the word CON enables PIP to distinguish a peripheral device name from a disk file of the same name. If the colon were omitted, the command would direct PIP to copy the disk file named CON.

When you give this command, PIP reads each character typed at the keyboard and enters it into the disk file named SAMPLE.TXT. Typing ^Z completes the operation.

This PIP command is not really very useful, since it is easier to create a file with the system editor. However, it shows how PIP transfers information to a file from the keyboard.

Displaying a Disk File on the Screen

In the previous section, we saw how to create a disk file from characters entered from the keyboard. We can reverse the process simply by reversing the parameters. The command

```
PIP CON:=SAMPLE.TXT
```

will display the disk file SAMPLE.TXT on the video screen (the other part of the console). The action is diagrammed in Figure 4.1.

This command may not appear to be very useful since we can do the same thing with the built-in command:

```
TYPE SAMPLE.TXT
```

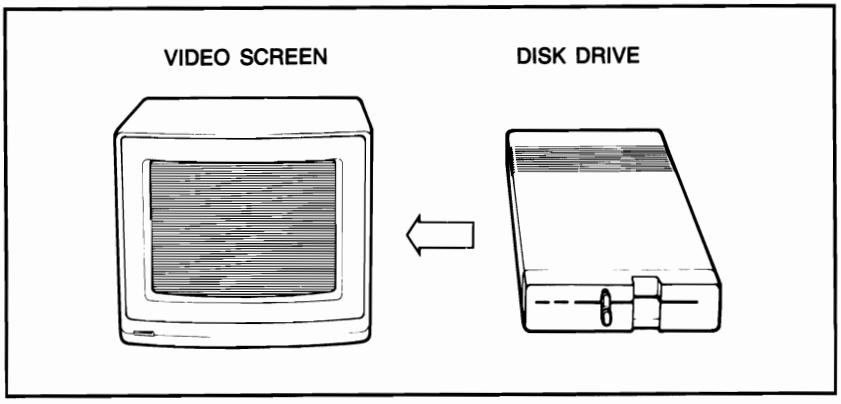


Figure 4.1: *The Command PIP CON:=SAMPLE.TXT Displays a Disk File on the Video Screen*

However, several PIP option parameters let you alter a file when it is copied to another disk or transferred to the screen or other peripheral.

Echoing Characters with the E Option When PIP makes a copy of a disk file, no indication of progress is given on the video screen. Sometimes, however, you might find it helpful to see what is happening. If you use the E option, PIP displays (echoes) each character on the video screen during the copying. (*Echo* is the name used to describe the displaying of text during an operation.) Note, however, that the text is displayed too rapidly for you to read it line by line. Rather, E gives you a general idea of what is happening. You use the E parameter only with text files; it must not be used with nontext files, such as COM files.

Try out the E parameter with the short program we made using PIP. Give the command:

```
PIP SAMPLE.BAK=SAMPLE.TXT[E] <RETURN>
```

This will create a duplicate file and echo it on the screen during the copy operation.

Adding Line Numbers Sometimes you may want to number the lines in a file. For example, this may be helpful when you want

to revise a document. Also, BASIC programs require line numbers. PIP can add a number to each line of a file during transfer. There are two variations of this parameter. The N option by itself adds line numbers that begin with 1 and are incremented by one. Blank spaces (called *leading* blanks) precede the number to fill out the field to six positions. A colon and a blank are appended at the end of the number. Thus, eight character positions are added to the beginning of each line.

The parameter N2 produces line numbers in the form required by BASIC programs. With this version the number field is filled out with leading zeros rather than blanks, the colon is omitted, and the final (or *trailing*) blank is replaced by a tab character. This form adds seven characters to each line.

Try out both the N options by displaying a test file on the video screen with the commands:

```
PIP <RETURN>
CON:=SAMPLE.TXT[N] <RETURN>
CON:=SAMPLE.TXT[N2] <RETURN>
```

As we learned previously, these commands display a file on the video screen. However, the line numbers added by PIP are also shown.

Changing Upper- and Lowercase The L and U parameters can change the case of the letters A through Z. Adding the letter L changes all uppercase letters to lowercase. The U option changes all lowercase letters to uppercase. Other characters are unaffected. To see the operation, try these commands:

```
PIP <RETURN>
CON:=SAMPLE.TXT[U] <RETURN>
CON:=SAMPLE.TXT[L] <RETURN>
```

Copying a Portion of a File

Sometimes you want to copy or display only a portion of a text file. Or perhaps you lost part of a long file listing either by

accident (a power failure) or because of a printer problem (no more paper or other mechanical problems). You will want to restart your listing where it was interrupted, rather than list the entire file all over again. PIP provides an option for listing and transferring a portion of a file.

You can copy portions of a text file by specifying strings of characters where PIP is to start and stop. (A *string* is a sequence of characters. It may be a whole word or one or more characters in a row, such as part of a word.) Use the S parameter to specify a starting string (that is, where PIP should start copying), and the Q parameter to specify a quitting string (that is, where PIP should stop copying). End each string with ^Z. PIP will automatically search for the first occurrence of each string of characters and copy the text from the starting through the stopping string.

Here is an example that copies part of the file SAMPLE.TXT into a new file, named NEWSAMP.TXT:

```
PIP
NEWSAMP.TXT = SAMPLE.TXT[Qsecond ^Z]
```

This PIP operation starts copying at the beginning of the file and stops when it encounters the string *second*. Note that the string *second* is written in lowercase letters, and that you executed PIP in two lines, not as a one-line command. If you execute PIP without a parameter, and then type a PIP expression, CP/M will search for a string that looks exactly the way you typed it. In other words, if the file contains SECOND in uppercase letters, you would receive a message that PIP had not found the string. By contrast, if you execute PIP on a single line, CP/M will always translate your strings to uppercase. If you had typed

```
PIP NEWSAMP.TXT = SAMPLE.TXT[Qsecond ^Z]
```

CP/M would automatically translate the string *second* to *SECOND* and so successfully locate the quitting string *SECOND*.

Here is another example using both the S and the Q parameters:

```
PIP
EXTRA.TXT = SAMPLE.TXT[Stext ^ZQsecond ^Z]
```

This PIP operation starts copying SAMPLE.TXT when it finds the string *text* and stops copying when it finds the string *second.* The file EXTRA.TXT will contain the portion of the file between the strings *text* and *second,* including both strings. Note that we executed PIP on two lines in order to preserve lowercase characters in the strings following the S and Q parameters.

Printing a File with PIP

We have seen that there are several possible ways to obtain a printed listing of a text file. The simplest method, which we discussed in Chapters 2 and 3, is to engage the printer with ^P and then give the TYPE command. Another way is to use PIP. The operation is similar to the one we used to display a file on the video screen, with a major difference. PIP can interact with the keyboard and screen, through the parameter CON:, as both an input and an output device. By contrast, the printer (LST:) is only an output device. That is, a file cannot be transferred *from* the LST: device. The command

```
PIP LST:=SAMPLE.TXT
```

sends the disk file SAMPLE.TXT to the printer. The action is diagrammed in Figure 4.2.

Later in this chapter, we will consider option parameters that can enhance the LST: command. Of course, all the regular PIP

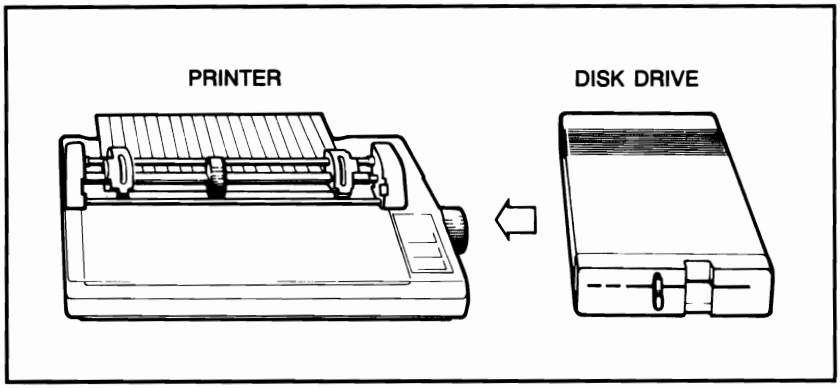


Figure 4.2: *The Command PIP LST:=SAMPLE.TXT*

features can apply when you use PIP to print a file. Several files can be concatenated and printed with a single command. For example, the command

```
PIP LST:=FILE1,FILE2,FILE3
```

will print the three files FILE1, FILE2, and FILE3.

An alternative printer device name PRN: automatically selects certain PIP parameters when printing with PIP. When you use LST: to print a text file, the file is printed literally, as you entered it. When you use the PRN: device name, however, there is additional processing of the text. Each line is numbered sequentially, the ASCII tab character is expanded to eight columns, and the page-eject (form-feed) character is inserted every 60 lines. (These features are explained later in this chapter.) To see how this works, give the command:

```
PIP PRN:=SAMPLE.TXT <RETURN>
```

LST: does the same thing if you include the corresponding option parameters (see Chapter 7).

Making Backup Copies of Altered Files

It is important that you make frequent backup copies of important disk files whenever they are changed. You learned earlier in this chapter how to make copies of disks using PIP. Of course, it is not necessary to copy the entire disk, only those files that have been changed since the last backup copy was made. PIP contains an option parameter to make this task easier.

We have seen that the attributes *read only* or *read/write* and *system* or *directory* are encoded in the directory with the file name. Another attribute encoded with the file name, the *archive flag,* indicates whether a file has been altered since the last archival copy operation. Each time a file is altered, the archive flag is automatically reset to show that the file has been changed. When a file is copied with the A (for archive) option, PIP changes the archive flag to show that a backup copy has been made. Thus, at the end of each day you can make a backup

copy of all changed files with the command

```
PIP E:=A:*.*[AV]
```

PIP will copy only those files that have been altered since the last copy operation and will then set the archive flag of these files so they will not be copied the next time. The file-name parameter must contain wild cards.

The current state of the archive attribute can be determined with the FULL or RW option of DIR. The expression *Arcv* appears in the attribute column if the option is enabled—that is, if the file has been backed up. Otherwise, no entry appears in this column.

Setting Up a New User Area

As we saw in Chapter 3, each disk can be logically partitioned into as many as 16 separate user areas. The advantage of separate user areas, the isolation of files, is also a disadvantage. Programs in one user area are not normally accessible to someone working in another area with one exception: system files located in user area 0 are accessible to all user areas.

You learned in Chapter 3 how to determine the active user areas by executing SHOW with the [USERS] option and how to determine what files are associated with each user area by executing DIR with the [USER] option. Now let us see how to copy files from one user area to another.

Before partitioning a disk into several user areas, you should set the system attribute of all transient programs in user area 0. If this has not already been done, go to drive A and give the command

```
SET *.COM[SYS RO] <RETURN>
```

to set all COM files to both system and read-only status, making them accessible to all user areas.

To copy files from your current user area to another user area and vice versa, you use PIP with the G option. Before you begin

a copy operation, determine the total size of the files on the source disk by executing DIR with the [SIZE] option. Then execute SHOW to ensure that there is sufficient space remaining on the destination disk. (Refer to Chapter 3 if you need to review these commands.)

Now let us consider a command to copy all BASIC programs from user area 0 on drive E to user area 2 on drive A. Be sure you are located in user area 0 on drive E. Give the command:

```
A:PIP A:[G2]=*.BAS[V] <RETURN>
```

We have seen that several options can be placed after the source parameter, the second file name. However, in this example we use the G option after the destination parameter, the first file name. This is the only PIP option that can appear after the first file-name parameter.

An alternative method is available when user area 2 on drive A is current and PIP has been set to a system file. This is the command:

```
PIP A:=E:*.BAS[G0 V]
```

(PIP options can be run together or separated with a space.) For either method, PIP will display each file name on the screen as it is being copied because the wild-card symbol * is included in the name.

Figure 4.3 diagrams the paths PIP follows in copying files from user areas on one disk to a user area on another disk. Suppose you are currently working in user area 1 of disk A but you need copies of files from drive E. One path shows PIP copying a file from your own user number of drive E. This is the command:

```
PIP COPY1=E:ORIG1[V]
```

The other two paths are more complicated commands since the files are in different user areas. These are the commands:

```
PIP COPY2=E:ORIG2[G2 V]
PIP COPY3=E:ORIG3[G3 V]
```

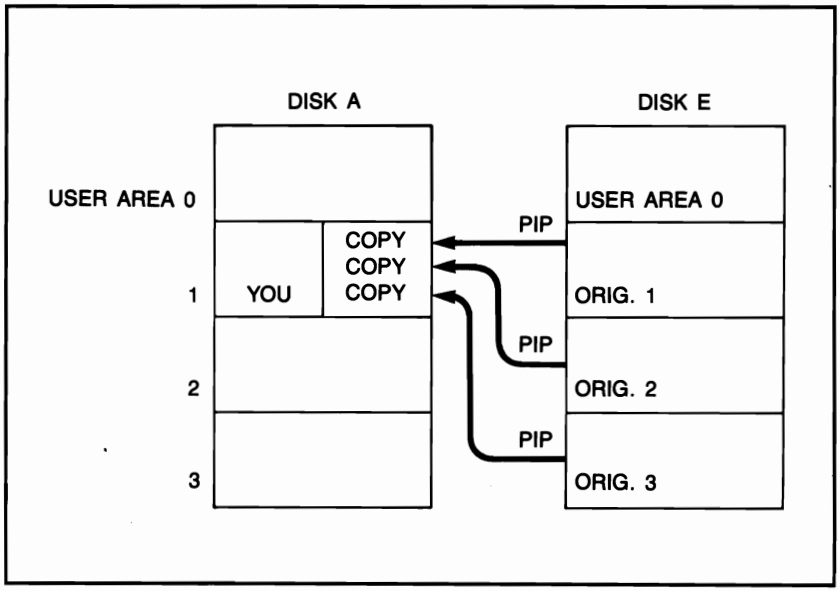


Figure 4.3: *Copying Files from Other Disks and User Areas*

Copying Read-only and System Files

PIP will not normally overwrite—in effect, delete—a file that has the read-only attribute set. For example, suppose that you give the command

```
PIP COPY=ORIG
```

but a read-only file named COPY already exists. PIP responds with the question:

```
DESTINATION FILE IS R/O, DELETE (Y/N)?
```

If the existing file named COPY did not have the RO attribute, PIP would have automatically deleted the original COPY and replaced it with the new COPY. Because of the RO attribute, PIP displays the message that the file named COPY is read only and asks if you want to delete the original version of COPY. If you reply Y (yes), PIP will delete the old COPY and replace it with

the new COPY. (Note that with a simple Y or N answer you do not press the RETURN key.) If you answer N for no, PIP terminates without performing the copy operation, and the original version of COPY is not deleted. PIP displays the message:

```
** NOT DELETED **
```

If you want to override this step of a message and response when a file is read only, you can use the W parameter. PIP will then ignore the RO attribute, delete the file, and proceed with the operation. When concatenating several files, you need to give the W parameter only once at the end of the PIP expression. For example:

```
PIP WHOLE.TXT=PART1.TXT,PART2.TXT,PART3.TXT[W]
```

If the original file has the system attribute, then PIP cannot find it in the disk directory. For example, if PIP.COM is a system file on disk A and you want to make a copy on disk E, the command

```
PIP E:=A:PIP.COM
```

will not work. You must give the R parameter to copy a system file. For example:

```
PIP E:=A:PIP.COM[R]
```

Therefore, when you want to copy a system file, you must use the R option so PIP can locate the file. Furthermore, if the COPY file exists as a system file, PIP will delete it and then make the new version a system file.

If several files are being concatenated, the R parameter must be given for each one. For example, the command

```
PIP WHOLE.TXT=PART1.TXT[R],PART2.TXT[R],PART3.TXT[RW]
```

will concatenate the three system files PART1.TXT, PART2.TXT, and PART3.TXT into a new file named WHOLE.TXT. If a file

named WHOLE.TXT already exists and is write protected, PIP will automatically erase it because the W parameter is also included.

Zeroing the Parity Bit

The ASCII character set uses only seven bits. Therefore, the remaining bit of each byte, called the *parity bit,* can be used as an error check. However, sometimes this bit is used for other purposes. For example, WordStar sets the parity bit to indicate spacing for justification. On the other hand, Microsoft BASIC requires the parity bit to be reset. Thus, if a BASIC program is edited with WordStar, the parity bit will be set for many of the characters and the BASIC program can no longer be used. When this happens, use the Z option of PIP to restore the BASIC program to its original form by zeroing the parity bit. For example, the command

```
PIP SORT.BAS = SORT.BAS[VZ]
```

will make the program SORT.BAS usable again. Notice that in this example both the original file and the new file have the same name. Do not use wild cards in this command or you will lose your file. The Z parameter is not needed for transfer to ASCII devices such as CON: and LST: and it must *not* be used for copying binary files or WordStar files.

Using Option Parameters with PIP

We have been able to modify the operation of PIP by including option parameters such as V and E enclosed in square brackets at the end of the PIP command. When you include more than one parameter in a PIP expression, the parameters can be given in any order and can be separated by spaces. The D, G, P, and T options require a number immediately following. The Q and S parameters need a string of characters and a ^Z at the end of the string.

Now let us look at some examples of PIP expressions using option parameters:

```
PIP LST:=SAMPLE.TXT[NT8P60]
```

This expression sends the file SAMPLE.TXT to the printer (LST:), adds line numbers (N), expands tabs to every eighth character column (T8), and inserts form feeds every 60 lines (P60). If you assign PRN: as the listing device, these are the default parameters. Thus, you could obtain the same result by rewriting the above example:

```
PIP PRN:=SAMPLE.TXT
```

Here is another example of PIP options:

```
PIP LST:=PROG.ASM[NT8U]
```

This expression sends PROG.ASM to the printer with line numbers (N), tabs expanded to every eighth column (T8), and lowercase characters translated to uppercase (U).

Summary

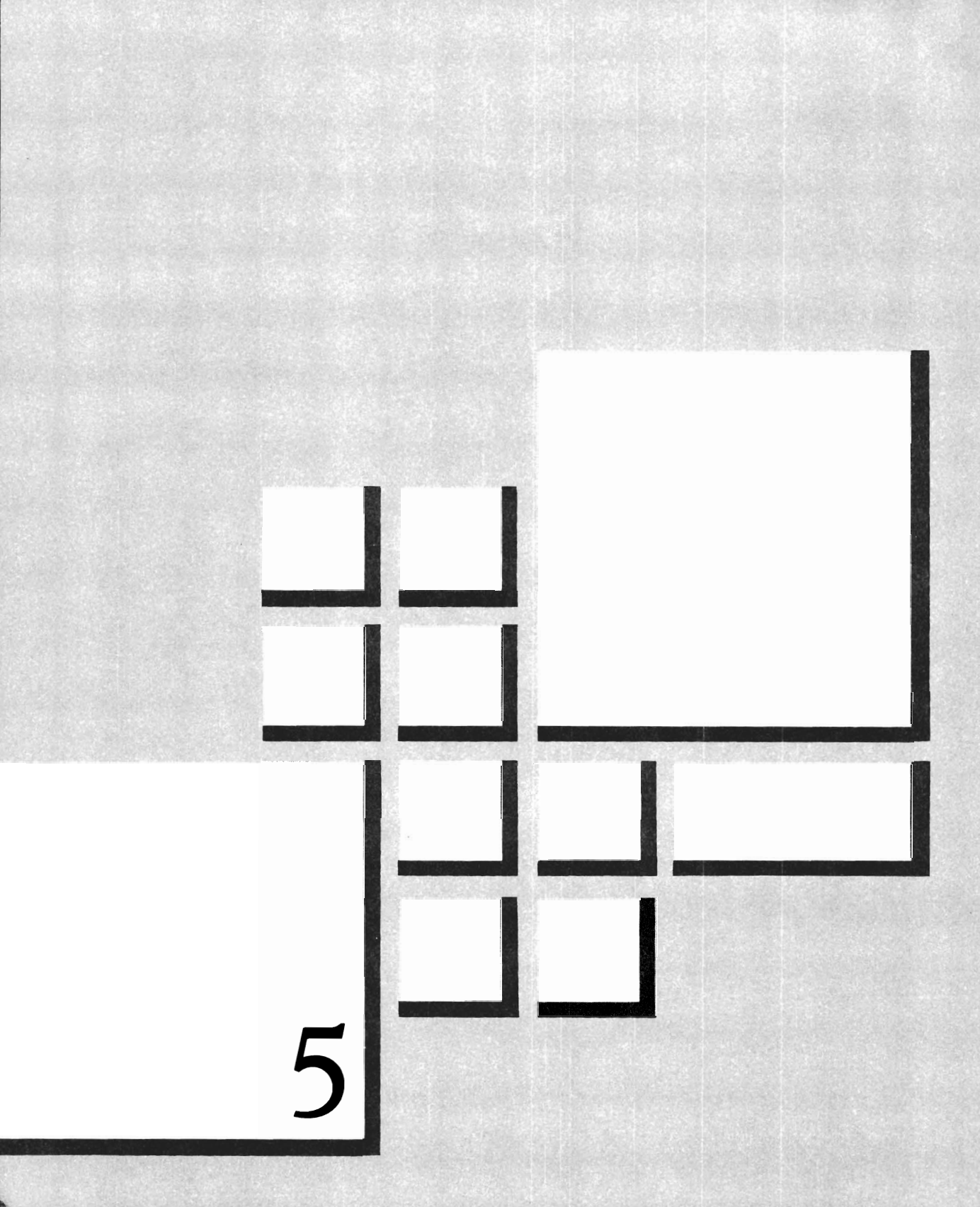
In this chapter we have explored some of the PIP options. Even though you will generally use only a few of the many options available, it is important to be aware of them in case you need them. The PIP options are summarized in Chapter 7.

PIP is a powerful general-purpose file-transfer program. Though it is most frequently used for simple disk-to-disk transfers, it can do much more. For example, PIP can do the following:

- Verify that the new copy is correct
- Transfer files between devices and disks
- Transfer several files at once
- Alter certain properties of a file

- Join several files into a new file
- Extract a portion of a file
- Copy files from one user area to another
- Copy files that have been altered

This chapter discussed the most common options used with PIP. A working knowledge of the important capabilities of this program is a definite advantage to every Commodore 128 user.


5

The System Editor, ED

In Chapter 2 we used the CP/M program PIP to create a very short text file. This is not a convenient way to make larger files since we cannot correct typing errors during entry or alter the resulting file with PIP. To help make your writing easier, CP/M includes an editor program, called ED, that can create and alter disk files. Unfortunately, ED is not present on your original CP/M disk. You must order it with other CP/M programs from Commodore.

If you use your Commodore 128 for writing computer programs or for letters and reports, you will need to use an editor. Several

editor programs are available for your computer. Some editors are easy to learn but not very sophisticated, while other editors are very powerful but may be difficult to master.

ED, the editor included with CP/M, is easy to learn, but it is quite cumbersome and inefficient to use. ED is reasonably satisfactory for writing line by line, as in computer programs, but it is less convenient for composing and altering general text that is organized into paragraphs. Therefore, if you plan to write letters and reports with your computer, you should seriously consider purchasing a more powerful program, such as WordStar. You can purchase the Kaypro version since it also runs on the Commodore 128.

In this chapter you will learn to use the CP/M system editor. If you will always be using a different editor with your computer, you can skip to the next chapter.

Before you learn the specific ED commands, let us consider what a text editor is and how it is used.

What Is a Text Editor?

If you want to write a computer program, a business letter, or a manuscript, you need to create a text file that is saved on disk. Furthermore, after you have created such a file, you may want to make alterations or additions to it. A text editor is used for both the creation and the alteration of text files.

When you create a document with an editor, you type the text as you would with a typewriter. The resulting disk file can be displayed on the video screen or sent to the printer by the methods we discussed in earlier chapters. When a text editor alters an existing file, it does not actually change the original copy. Rather, it creates a new disk file that incorporates the desired changes. The original file is then saved as a backup copy.

The real power of the editor is apparent when you want to make changes to the document. An editor should allow you to move easily from place to place in the document and to make changes by typing new text directly over the original or by deleting and inserting new characters. The editor should be able to locate a selected passage in the text so you can readily make

changes and should also let you insert text from another file. ED provides all these features.

Sophisticated editors let you easily move the cursor anywhere on the screen to make changes. Unfortunately, ED does not provide this capability.

A word processor is a special type of editor that can not only create and alter text files but can also format the resulting text for the printer. Word processor capabilities include underlining, justifying margins, subscripting, superscripting, and creating boldface type. ED is not designed for these operations.

Let us begin by learning how ED operates. Then you will see how to use the specific commands.

How ED Operates

The CP/M text editor is a file named ED.COM, which is located on one of the auxiliary disks that you must send away for. You execute the program by typing the primary name ED followed by the name of the file you want to create or alter. For example, the command

```
ED SAMPLE.TXT
```

executes ED and instructs it to look for a file named SAMPLE.TXT on the current drive. (Be sure to type a space after the word ED.) If the file can be located, ED prepares to edit it. If no such file exists, ED will create a new one. In this example, ED is the command and SAMPLE.TXT is the parameter.

You can execute ED with two parameters. In this case the first parameter refers to the original file, and the second parameter refers to the new file. If you execute ED without parameters, it will automatically ask for the first and second parameters. If only one file name is desired, as is usually the case, simply press the RETURN key when the second parameter is requested.

When you create a new disk file with ED, all the text you type is initially placed into a region of memory known as the *edit buffer.* At the completion of the editing step, ED makes a file that is a copy of the edit buffer.

The edit buffer is a block of memory in the TPA (transient program area) where the editing is done. The entire original file can be loaded into the buffer if it will fit. If the file is too large, however, only a portion of it is copied into the buffer. When this portion has been edited, it is written to a new file. Additional text is then copied into the edit buffer from the original file. In this way you can edit a file that is too long to fit into memory.

When altering an existing file, an editor does not really change the original version but creates a second file that contains the desired changes. If you enter only one parameter, ED will give the original name to the new version of the file. It will not delete the original file but will change its file-name extension to BAK. In the example of SAMPLE.TXT the original file, also called the *source file,* will be renamed to SAMPLE.BAK at the end of the editing session.

When there is enough room on the disk, both the new file and the original file will exist on the same disk at the conclusion of the editing session. However, if the file to be edited is very large or if there is a limited amount of space on the disk, there will not be room for both files. Then you must place the new file on another disk.

When you want to edit a file and place the edited file on another disk, you give two parameters to the ED command. For example, when drive E is current, the command

```
A:ED E:SAMPLE.TXT A:
```

directs ED to look for the original file on drive E, the first parameter, and to place the edited version of the file on drive A, the second parameter. If, as in this example, the second parameter consists of only a drive name, ED automatically fills out the second parameter with the file name of the first parameter. In this example the file type of the original file will not be changed since the two parameters refer to different drives. Of course, you can give a different file name for the second parameter.

Altering an existing disk file is more complicated than creating a new file. To accept any changes or additions, your file must be copied into the edit buffer. Many text editors automatically copy the original file into the edit buffer so you can make the desired

alterations. However, after you have executed the ED program, you must give an additional command to copy the original file into the edit buffer. Then you alter the version in the edit buffer by giving the appropriate commands. Finally, you create a new disk file from the edit buffer. Thus, text moves from the original file to the edit buffer and then to the new disk file. The relationship of the edit buffer to the disk files, keyboard, and video screen is shown in Figure 5.1.

The two phases of ED are *command mode* and *insert mode.* You give directions to ED in command mode and you type characters for your file in insert mode. In previous chapters you learned how to edit the command line with control characters. Three of these commands can be used with either the insert or the command mode of ED—^H, ^X, and ^U. The cursor is moved to the left with ^H, and the current line is deleted with either ^X or ^U. You can give the other CP/M Plus line-editing commands when ED is in command mode.

The regular ED commands are not control characters but simply letters of the alphabet. That is, you type the letter without holding down the CONTROL key. For most commands either uppercase or

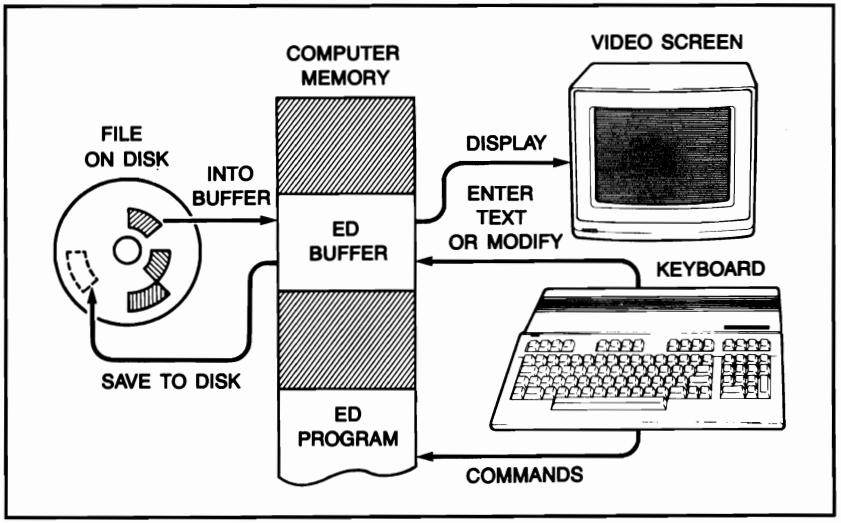


Figure 5.1: *The Edit Buffer in Relation to the Disks, Keyboard, and Video Screen*

lowercase letters can be used. (You will see the exceptions later in this chapter.) The letter usually suggests the nature of the operation—for example, A for append and W for write.

If you want to execute a command more than once, you can precede the letter by a parameter that indicates how many times the command is to be repeated. If you omit the parameter, ED generally uses a default of one. You use the number symbol, #, as a parameter when you want to repeat the command as many times as possible. (# is CP/M's shorthand for 65,535.) As you will see, a parameter of zero sometimes has special meaning. Always press the RETURN key at the end of the command line, even when there is only one character on the line.

To better understand how ED operates, we are now going to create and alter a simple file. We will discuss the various editing commands in more detail in later sections.

Using ED

Creating a File

Place the system disk containing ED.COM in the disk drive. If the current drive is not A, give the command

```
A: <RETURN>
```

(Use the DIR command to make sure this disk does not already contain files named SAMPLE.TXT and SAMPLE.BAK.) Give the command:

```
ED SAMPLE.TXT
```

ED will display the message

```
NEW FILE
```

to show that it is adding a new file name to the disk directory. If this message appears when you want to alter an existing file, it means that ED could not locate the requested file. Either you misspelled the name, or you are on the wrong drive. In such a case you should return to CP/M by giving the Q command and

then starting ED with the proper parameter. Various error messages you might receive as you work with ED are listed near the end of this chapter.

As we have seen, ED has two separate modes of operation—command mode and insert mode. As the names imply, you give commands to ED in command mode and add new text in insert mode. ED begins in command mode by displaying a colon followed by an asterisk.

Each time ED is started up, the edit buffer is empty. Therefore, the first thing you must do is copy all or part of the original file into the edit buffer. If the file is very short, it can be copied into the buffer all at once. However, if the file is large, the text must be passed through the edit buffer section by section. This operation is known as *paging* a file through the edit buffer. Let us consider the commands that transfer text through the edit buffer.

After you execute ED, a

```
:*
```

prompt should appear on the video screen. Whenever the cursor is next to the asterisk, ED is in the command mode ready for your next command. Give the command

```
i <RETURN>
```

being careful to use a lowercase i. ED will respond by moving down to the next line, which is numbered 1:. The i (or I) command puts you in insert mode and lets you insert text into the buffer. We will look at this command in more detail later in the chapter. For now we will simply use it to create a file.

With the cursor on line 1, enter the following text:

```
This is the first line of my file SAMPLE.TXT. <RETURN>
```

ED will move down to the next line, numbered 2:. Enter the following text:

```
This is the second line. <RETURN>
This is the third line. <RETURN>
This is the fourth line. <RETURN>
```

At the fifth line enter ^Z. This will mark the end of your file. It takes you out of the edit buffer and returns you to command mode. Now give the command

```
E <RETURN>
```

to both end the edit and write the text from the buffer to your disk file. The A> prompt shows that you are back in CP/M.

Next you will see how to alter an existing file.

Altering a File

To return to ED give the command:

```
ED SAMPLE.TXT
```

Notice that this time you do not see the NEW FILE message. At the ED cursor give the command

```
#A <RETURN>
```

to copy (append) as many lines as possible from the source file into the edit buffer. Since your source file contains only four lines, you could accomplish the same thing with the command 4A. This is diagrammed in Figure 5.2.

Very large files that do not entirely fit into the edit buffer are edited in steps. You first copy some lines into memory with the A command. After editing, you write these lines to the new file with the W command. Additional lines are read into the edit buffer with another A command, and so forth. In this way you can edit a large file with a series of A and W commands. The command 0A is useful for this because it copies lines from the source file until the buffer is about half full.

The cursor at line 1 means you are now in the edit buffer and can begin editing your text. You'll notice, however, that your text doesn't appear on the screen. To display all your text, give the command

```
#T <RETURN>
```

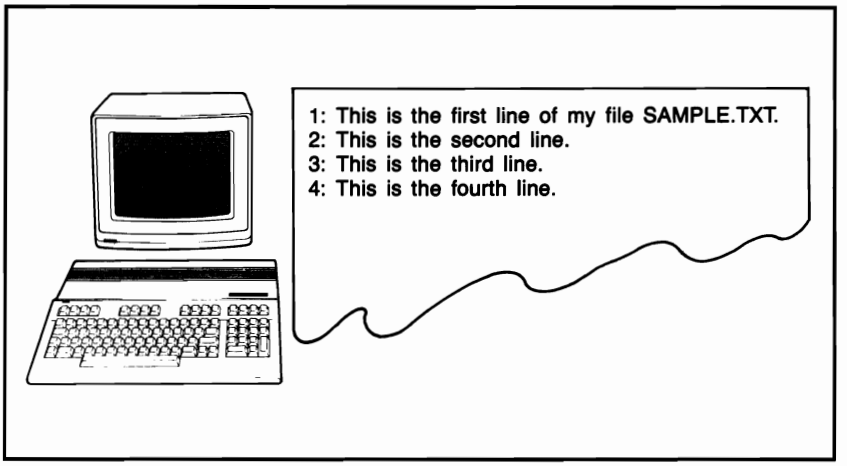


Figure 5.2: *Text Is Appended to the Buffer with the A Command*

(for type). Alternatively, you could enter T alone to display one line, or T with a number preceding it for a specific number of lines.

As a simple example of a text alteration, we will add two more lines to the file. Give the command:

```
-B <RETURN>
```

This will move the cursor to the end of the file. Next give the command:

```
i <RETURN>
```

You will see that you are now at line 5, and can insert new text. Enter the following:

```
This is the fifth line. <RETURN>
This is the sixth line. <RETURN>
^Z
```

After you finish editing the file, give the E command to end the edit. ED automatically writes any remaining lines from the edit buffer to the new file, copies unchanged any remaining lines of the

original file to the new file, and then returns to CP/M. To see how this works, give the E command followed by <RETURN>. Next give the command

```
DIR S*.*[FULL] <RETURN>
```

to see that files named SAMPLE.TXT and SAMPLE.BAK are both present. (Be sure that DIR.COM is on your current disk.)

Now that you have learned how to create a file and how to transfer text from the source file to the edit buffer and then to the new file, let us consider the character pointer and how it can be moved.

Commands for Manipulating the Character Pointer

ED can alter information in the edit buffer by deleting or inserting single characters or complete lines. To do this, you must specify exactly where you want the alterations to occur by using the *character pointer* (sometimes abbreviated to CP), an invisible marker that can be moved through the edit buffer. Note that the character pointer is not the same as the cursor. ED commands that delete or insert information refer to this pointer.

Unfortunately, most of the movements of the pointer through the text are not automatically displayed on the screen. To see exactly where the character pointer is located, you must give the T command. For example, once you have executed ED, give the following commands:

```
#A <RETURN>    (copy entire file)
#T <RETURN>    (type file on screen)
3C <RETURN>    (move CP three spaces)
```

Although the pointer has moved three characters to the right in the first line, the change doesn't show on the screen. Now type:

```
T <RETURN>
```

ED now displays the line beginning at the position of the edit pointer.

The pointer can be positioned at the beginning of the edit buffer, between any two characters of the text, or at the end of the edit buffer. ED initially sets the pointer to the beginning of the buffer. You can always return the pointer to the beginning of the buffer by issuing the B (for beginning) command (see Figure 5.3). The command may be given in either upper- or lowercase. As we have seen, the command -B moves the pointer to the end of the buffer.

The C command moves the pointer one or more characters, and the L command moves it one or more lines. Remember that the carriage return and line feed at the end of the line each count as one character rather than two. You place a number in front of the command to move more than one character or line. For example, the command 5L moves the pointer ahead (toward the end of the buffer) five lines. A negative sign moves the pointer back toward the beginning of the buffer. (Use the hyphen on your keyboard for the negative sign.) For example, a command of -7C moves the pointer backward seven characters. This is the usual interpretation of the minus sign for ED commands. Notice that in the previous example the minus sign in -B has the opposite effect.

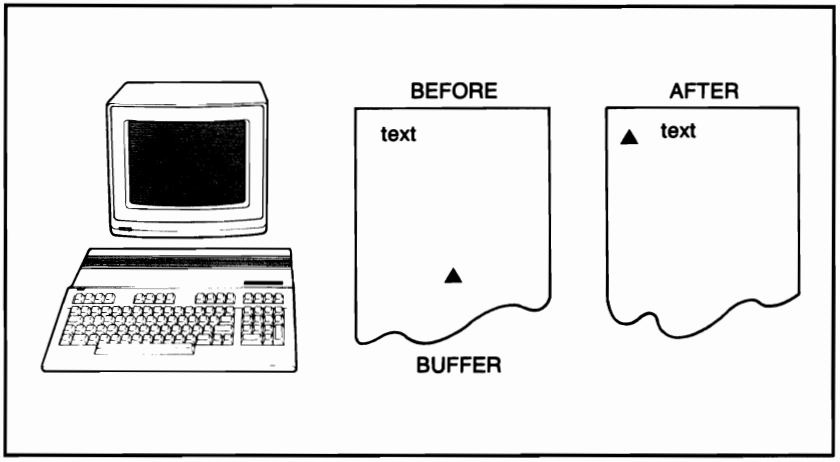


Figure 5.3: *The B Command Moves the Pointer to the Beginning of the Buffer*

Giving the L and C commands without a parameter moves the pointer one line or character. As with the A and W commands, a parameter of # moves the pointer the maximum number of characters or lines. This will be the end of the buffer. It will also be the end of the file except for a large file. The command 0L moves the pointer to the beginning of the current line.

An interesting variation on the L command occurs when you give only the parameter itself. For example, a command of 6 moves the pointer ahead six lines and then displays the sixth line on the video screen. If you give a carriage return by itself, ED moves the pointer to the next line and displays that line. A parameter of just a minus sign moves the pointer back one line and displays that line.

Searching for a String of Characters

In the previous section we moved the character pointer a fixed number of characters or lines at a time. However, you can move the pointer to a particular string of characters with the F and N commands. Let us consider F first.

The F (for find) command locates the requested string in the edit buffer and places the pointer immediately *after* the string. The simplest form of the command is f*string*, where *string* stands for the characters you want to find.

You must be aware of a few complications when you use the F command. First, if there are *any* lowercase letters in the string, you must enter the letter f in lowercase. Otherwise, ED will change all lowercase letters in your string to uppercase before starting the search and so it won't find the string you wanted. This problem is not unique to the F command. Though ED commands can generally be given in either uppercase or lowercase letters, they must be in lowercase whenever a following string of characters contains lowercase letters.

A second problem can occur because the F command can only move the pointer forward. Thus, ED searches for the next occurrence of the string starting at the character pointer. If the string you need is located between the beginning of the buffer and the pointer, the F command will not find it. The solution is

to reset the pointer to the beginning of the buffer using the B command before you give the F command.

If you are going to give additional commands after the end of the string, you must include ^Z at the end of the string.

Let us look at an example of how the F command works. Execute ED for the file SAMPLE.TXT and type

```
fline <RETURN>
```

ED will locate the next occurrence of the string *line* after the position of the character pointer. The pointer then moves automatically to the end of the given string. If you do not want to stop at the first occurrence of the string, include a parameter with the F command. For example, the command

```
3fline <RETURN>
```

locates the third occurrence of the string *line* by skipping over the first two.

CP/M text files contain both a carriage return and a line feed at the end of each line. You can refer to the combination of these two characters by including a ^L character in the search string. Give the command

```
f^LThis
```

to locate the string *This* preceded by a carriage return and a line feed—in other words, when it occurs at the beginning of a line. It is not possible to reference just a carriage return or a line feed alone.

If ED cannot find the string following the F command in the edit buffer, it displays an error message and the pointer does not move from its initial position. If this happens, check to see whether the pointer is already at the end of the buffer or if you misspelled the string.

We have seen that a very large file cannot be entirely copied into the edit buffer but must be paged through it in pieces. The F command searches only the text that is located in the edit buffer.

It does not continue the search on additional text located in the original source file. Since this may sometimes be desirable, there is a second string-search command that is used for large files.

The N command is similar to the F command except when the string cannot be found in the current buffer. At this point, ED automatically writes the current contents of the buffer to a disk file and then fills the buffer from the original file. ED continues searching through the entire disk file until the string has been found or until the entire original file has been searched.

At the conclusion of the N command, the pointer will be at the end of the file if the string cannot be found. In this case you should give the H command, which empties the edit buffer and restarts the editing procedure. You then issue a 0A command to copy the first part of the file into the edit buffer so you can continue.

Commands for Displaying Text in the Edit Buffer

Practically all the editing commands you have learned so far do not display the text on the video screen. You will now see how to view the text that is present in the edit buffer by using the T and P commands.

The T (for type) command displays lines of the edit buffer on the video screen. If the parameter is omitted or if it is a positive number, the display begins at the character pointer. This command does not move the character pointer. Thus, if the pointer is in the middle of a line, only the right half of that line will be displayed. Figure 5.4 illustrates this command.

A negative parameter displays lines in front of the pointer. For example, the command -T shows the previous line. If the pointer is in the middle of a line, the command -T displays text from the beginning of the previous line up to the pointer. The command 0T displays the text from the beginning of the current line up to the pointer. You can obtain a printed listing of the text by engaging the printer with ^P before issuing the T command. #T will display all lines from the pointer to the end of the buffer.

The P command displays whole pages of text. However, unlike the T command, the P command moves the character pointer. If you give the P command without a parameter, one screenful of 23 lines is displayed and the pointer is placed just before the first

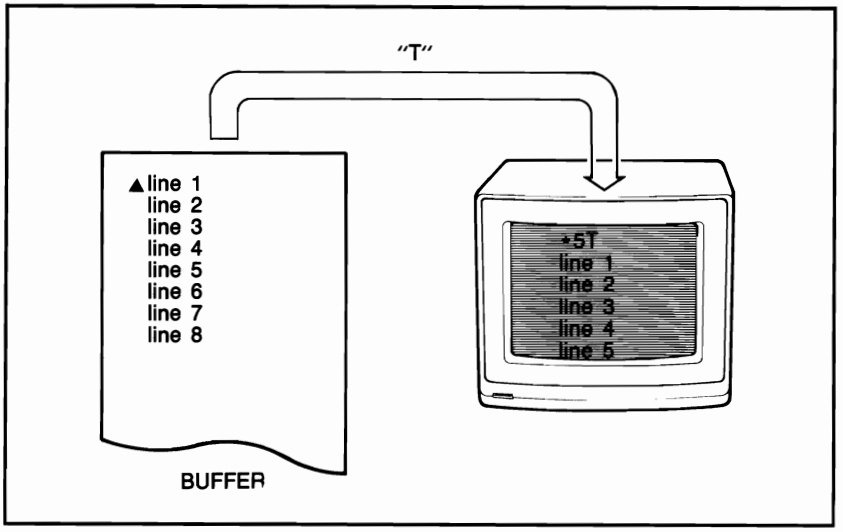


Figure 5.4: *The T Command Displays Text on the Video Screen*

character on the screen. (Since a video screen has 24 lines, this command just fills the screen.) Each time the P command is given, the next 23 lines are shown and the pointer is moved to the beginning of the text on the screen.

You may give a parameter to modify the P command. The command 0P displays the present page without moving the pointer. A number parameter displays that number of pages; for example, 6P displays the next six pages. A parameter of -1 displays the previous page and moves the pointer back one page. The command #P will display the entire buffer.

Combining Several ED Commands

Sometimes it is convenient to combine several ED commands. Most of the ED commands can be run together on the same line without spaces. However, commands that terminate the editing session must be given on a line by themselves.

We saw that when the pointer is in the middle of a line, the T command displays only the right side of the line and the 0T command displays only the left side. The combination command 0TT will display the entire line without moving the pointer. The

0T part displays text from the beginning of the line up to the pointer, and the second T displays the remainder of the line. A similar combination command, 0LT, moves the pointer to the beginning of the current line and then displays that line.

As a third example, consider the combination command B2T. This command first moves the pointer to the beginning of the buffer and then displays the first two lines of text on the video screen. Give this command and observe its effect. Notice that there are numbers at the beginning of each line of text. Let us explore this.

Line Numbers

Each line in the edit buffer is assigned a line number, which is shown on the screen, but the number is not placed in the buffer. The numbers run consecutively, starting with 1. Line numbers can be used to specify where the pointer is to be moved and also to specify ranges. You can move the character pointer to a particular line by giving the line number followed by a colon. For example, the command 2: moves the pointer to line number 2.

You may specify a range of lines by giving the inclusive line numbers separated by two colons. For example, the command

```
2::8T
```

displays lines 2 through 8. This form can be used only for the T and L commands we have considered and for the K command we will consider shortly.

ED normally adds line numbers when it displays text on the screen. You can disable this feature (that is, turn it off) with the -V command. If line numbering is off, the V command will turn it back on. The command 0V is unrelated to the other two V commands. It displays two numbers. The first is the remaining free space in the edit buffer, and the second is the total buffer size. Both values specify the number of characters.

The line numbers, like the character pointer, refer only to the text in the buffer. They do not appear in the edited disk file. Furthermore, when a line is deleted, the remaining lines are

renumbered. Therefore, you must be careful when using line numbers from a printed listing of your file because they may change after editing. To avoid this problem, you can start editing from the bottom of the file and work upward. In this way, all the numbers from the previous version will stay the same. Let us see how this works.

Suppose you have a printout of your 100-line text containing ED's line numbers. You want to delete lines 10, 30, and 50. If you start at the top of the text, deleting line 10 first, then the line originally numbered 11 becomes 10 after the deletion. Line 12 becomes 11, line 30 becomes 29, and line 50 becomes 49. If you delete the original line 30, which is now numbered 29, all remaining line numbers will be changed again. Thus, the third line to be deleted is now numbered 48 rather than the original 50.

Now suppose you reverse the process, deleting from the bottom up. If you delete line 50 first, then all lines beyond 50 are renumbered, but not those before it. The original line 30 is still 30, and the original line 10 is still 10. When line 30 is deleted next, lines numbered less than this are not changed. You can confidently delete line 10 next, since its number has not changed.

Now let us see how to alter text in the edit buffer.

Commands for Altering Text in the Edit Buffer

In the previous sections you learned how to move the character pointer and how to display portions of the edit buffer. However, we did not really alter the text in the buffer. In the following sections you will learn several different ways to insert and delete single characters, strings of characters, and entire lines.

Deleting Characters and Lines from the Buffer

Remember that alterations are made relative to the character pointer. The D command deletes the next character after the pointer. As with most of the commands, you can place a parameter in front of this command. For example, the command 5D deletes the five characters following the pointer, and -3D

deletes the three characters in front of the pointer. The pointer is not moved by this command.

The K (for kill) command is similar to the D command except that it deletes lines rather than characters. When you give K without a parameter, text from the pointer to the end of the line is deleted. The carriage return and line feed at the end of the line are also deleted. If the pointer is in the middle of the line, K deletes only the part of the line to the right of the pointer. A command of -K deletes text from the beginning of the line up to the pointer. A positive or negative number added to K will delete more than one line. A quick way to delete all the remaining text from the pointer to the end of the buffer is to give the command #K. Be sure to use this command carefully so you don't accidentally lose more text than you intended.

Inserting Characters into the Buffer

Let us now consider ways to insert characters into the buffer. The I command is one of several commands that can be used for this purpose.

We have seen that although most of the ED commands can be given in either uppercase or lowercase letters, F and N commands must be given in lowercase if the following string contains lowercase letters. In a similar way, the I command must be in lowercase if you want to insert lowercase letters. If you give the I command in uppercase, all the text you insert will automatically be converted to uppercase.

Sometimes you will want text to be entirely uppercase—for instance, if you are writing a BASIC or FORTRAN program. After you give the U command, all characters you type will be treated as uppercase. (This command might be thought of as a software shift lock.) After the U command is given, you can give the I, F, and N commands in either upper- or lowercase, and the following characters will be treated as uppercase even though they might appear as lowercase on the screen.

Give the -U command to reverse the effect. Then upper- and lowercase will be treated as distinct once again.

There are two variations of the insert command. With one

method you type only the letter I and a carriage return and then enter your text on the next line. All the following characters will be inserted into the edit buffer at the position of the character pointer. If you type another carriage return, both a carriage return and a line feed are added to the buffer. For example, the sequence

```
i <RETURN>
This is the first inserted line. <RETURN>
This is the second line. <RETURN>
```

inserts two new lines in the edit buffer. The carriage return following the I command is not inserted, but the other two carriage returns are inserted, followed by line feeds. This is the normal method of creating a new file or adding large amounts of text to an existing file. Pressing the escape key (ESC) or giving the ^Z command terminates this form of the insert command and returns you to the command level. ED responds with the familiar asterisk. (The ^Z is not placed in the buffer.)

The second variation of the insert command is useful for adding a small number of characters to an existing file. With this method you type the letter I, follow it with the characters to be inserted, and then end the string of characters with an escape or ^Z, followed by a carriage return. Neither the ^Z nor the carriage return is placed in the buffer. For example, the command

```
iSan Francisco^Z <RETURN>
```

inserts the string *San Francisco* into the edit buffer at the position of the character pointer. The pointer then moves to the end of the string. The I command is given in lowercase since the string *San Francisco* contains lowercase letters.

Notice that a ^Z character marks the end of the string. If you omit the ^Z from the end of the string, ED places both a carriage return and a line feed in the buffer and then terminates the command. In other words, the commands

```
iSan Francisco <RETURN>
```

and

```
iSan Francisco^L^Z <RETURN>
```

both insert a carriage return and line feed.

Replacing One String of Characters with Another

Sometimes we need to replace one string of characters with a different string. The S (for substitute) command is used for this purpose. For example, the command

```
sSan Francisco^ZNew York^Z
```

will locate the next occurrence of the string *San Francisco* after the character pointer and replace it with the string *New York*. Notice that the initial S command must be given in lowercase if any of the letters in *either* string is in lowercase. Also notice that one ^Z marks the end of the original string and a second ^Z marks the end of the replacement string.

You can use ^L in either string to represent the carriage return and line feed pair.

Inserting a Separate Disk File into the Buffer

You learned in Chapter 4 that you can use PIP to append one file to the end of another file. You can also do this operation with ED using the R command. However, ED can insert one text file into any position of the text file that is being edited, not just the end. The file to be inserted may have any three-character extension, but the extension must not be blank. If you want to insert a file with the extension LIB, you may omit the extension when giving the ED command.

For example, to insert the file FILENAME.EXT during editing, move the character pointer to the desired position and give the command:

```
RFILENAME.EXT <RETURN>
```

Of course, the file FILENAME.EXT continues to exist.

Moving a Block of Text

Sometimes it is necessary to move a block of text from one portion of the edit buffer to another. You can do this with a combination of the X, K, and R commands. ED makes the move in several steps. The process is complicated because you must know the exact number of lines in the block. Let us look at the operation in detail, using the file SAMPLE.TXT.

1. Execute ED with the parameter SAMPLE.TXT.
2. Move the character pointer to the beginning of the block you want to move—in this case, the beginning of the third line. (Give the command 2L.)
3. Determine the number of lines to be included in the block. We will move a block of three lines, so give the command:

   ```
   3T <RETURN>
   ```

   This command displays the three lines so you can be sure they are the ones you want to move. Remember, this command does not move the pointer or alter the text.
4. Next, give the X command with the number of lines to be moved. (The same number we used for the T command):

   ```
   3X <RETURN>
   ```

   The 3X command writes three lines to a temporary disk file named X$$$$$$$.LIB. The pointer does not move.
5. Give the K command to delete the corresponding block of text from the buffer. In this example the command is 3K <RETURN>.
6. Move the pointer to the new location. For our example, it will be following the sixth line, so the command will be 3L. Then give the command R to read the temporary file back into the edit buffer at the new position. Use the H

command to end the edit and enter the changes in the disk file.

7. Finally, give the #A and #T commands to view the file. It should look like this:

```
1. This is the first line of my file SAMPLE.TXT.
2. This is the second line.
3. This is the sixth line.
4. This is the third line.
5. This is the fourth line.
6. This is the fifth line.
```

There is a potential problem with the block-move command. The temporary file created with the X command is automatically erased when ED is terminated. However, the temporary file is not erased when it is read with the R command. Therefore, if you give a second X command, ED will append the second block of text to the end of the text already present in the temporary file. Then a second R command copies both the first and the second blocks. Therefore, before you do a second block move, give the 0X command to delete the temporary file created during the first block move. Even if no temporary file exists, the 0X command will do no harm.

Repeating Groups of Commands

We have seen that with some ED commands, you can give a parameter in front of a command to repeat an action. For example, 4D deletes four characters. It is also possible to give a sequence of ED commands on the same line to perform several commands at once. Sometimes it may be desirable to repeatedly execute an ED command line that contains a sequence of commands. The M (for multiple) command is used for this purpose.

For example, while still editing the file SAMPLE.TXT, go to the beginning with the B command. Then give the command

```
6sThis ^ZWhat ^Z <RETURN>
```

to replace the string *This* with the string *What* six times. But during the substitution, there is no indication on the video screen. Let us see how to fix that.

We saw previously that the compound command 0TT displays a complete line. Though the command

```
6sThis ^ZWhat ^Z0TT
```

will change *This* to *What* six times, it will display only the last line; it will not show the first five. By contrast, the command

```
6MsThis ^ZWhat ^Z0TT
```

will display each line as the substitution is made. With the M command, ED repeatedly performs the operations specified by the rest of the command. If you place a parameter in front of the M, ED will execute the entire line that many times. On the other hand, if there is no parameter in front of the M, ED performs the command as if the # parameter had been given; that is, the command is executed as many times as possible. Negative parameters are not valid with M. Notice that all commands following the M command up to the end of the line are included in the repeating action.

Time Delay

Sometimes the M command presents information too rapidly on the video screen. A Z (for sleep, Z-Z-Z) at the end of the command will slow down the operation. A single Z pauses for about one second; an added parameter will increase the delay. For example, the command

```
6MsThis ^ZWhat ^Z0TT2Z
```

inserts a several-second delay after each substitution. Do not confuse the Z command with ^Z.

Now let us consider several ways to end the editing session.

Completing the Editing Session

Normally, you give the E command after you have completed your editing session. However, several other ED commands will also end the edit but with different results.

The H command completes the editing session and saves the file; it then automatically restarts the editing process. Next, you can use the 0A command to add text to the buffer because it was emptied by the H command. You should give these commands every 10 to 15 minutes. Let us see why.

If a power failure occurs or if the computer is accidently turned off or unplugged during an editing session, the original file and the portion of the text that has been written to the new file will be intact because they are on a disk. However, you will lose the information that is contained in the edit buffer since it is in RAM. To avoid such a loss, you should frequently force ED to transfer the information from the edit buffer to the new file by using the H command. This procedure is called *saving* a file to disk.

Sometimes a typing error causes a drastic mistake. For example, if you accidently type the command #K, all text from the pointer to the end of the buffer is deleted. In such cases you can return to a previous version of your file with the Q or O command. The Q (for quit) command returns to CP/M and abandons the changes made during the current editing session. The original file is unchanged and can be edited again, but the backup file is deleted. Be careful with this command.

The O (for original) command abandons the most recent changes. However, control remains with ED. The effect is to start the current editing session again. You must then issue the A command to fill the buffer again. When either the Q or O command is given, ED requests verification.

The Temporary ED Work File

We have seen that it is necessary to use the W command to copy lines from the edit buffer to the new file. The new file is initially given the file-name extension $$$. For example, if you are editing the file SAMPLE.TXT, ED writes the new lines to a

temporary file named SAMPLE.$$$. Then at the conclusion of the editing session, ED renames the original file to SAMPLE.BAK and the new file to SAMPLE.TXT. Therefore, you should never see a file with the extension $$$. However, if your editing session is accidently interrupted, because of a power failure, for example, you will find the new version saved as SAMPLE.$$$. You can recover this new file by changing its name. Let us see how.

When you start up CP/M as usual, give the command TYPE SAMPLE.$$$ to inspect this file. When you are convinced that it contains the edited material you want to recover, you can change its file-name extension. (Remember, the $$$ extension marks a temporary file that might be erased at any time.) Rename the file to SAMP.TXT, a name that is slightly different from the original. You can incorporate this file into the original file by using the R command. Be sure to delete the corresponding lines of the original.

Error Messages

There are two types of error messages that might appear during an editing session. One refers to ED commands, and the other is issued by CP/M when a disk error occurs. An ED error message has the form

BREAK *X* AT *C*

where *C* is the ED command being executed and *X* is one of the indicators shown in Table 5.1. Let us consider these messages in turn.

The ?C message is given to an invalid command—G or Y. It also appears if additional commands are placed on the same line with the E, H, O, or Q commands. (These commands must stand alone.)

The > message means that the parameter to the A command was too large, and the edit buffer was filled with text. If you only want to delete text from the buffer, there is no problem. However, it will not be possible to add new text until you write some of the buffer lines to the new disk file with the W command.

Indicator	Meaning
? C	C is an invalid command, a typographical error, or the command E, H, O, or Q combined with other commands. (Do not combine an E, H, O, or Q command with other commands.)
>	The buffer is full, or a string following an F, N, or S is too long.
#	ED cannot execute F, N, or S as many times as specified.
E	The command was terminated from the keyboard.
O	ED cannot open the LIB file for the R command. (Check to see if the LIB file exists or if you used the right file name.)
F	A file error (disk or directory full) was detected.

Table 5.1: *ED Error Messages*

Alternatively, use the H or O command to empty the buffer. Then issue the A command with a smaller parameter.

The # message means that the end of the buffer was reached before ED finished the command. This can occur, for example, if ED cannot find a string of characters specified with the F command.

The O message means that ED could not find the file name specified in an R command. Perhaps you misspelled the name or referenced the wrong disk.

The F message has two forms; both indicate a serious error. When the F is followed by the word FULL, the disk data area is full. Try to save your file to another disk using the R command. Next time, check that there is sufficient space on the disk before you begin the editing session. Alternatively, the F message will be followed by the words DIRECTORY FULL. This means that there is no more directory space, and you are likely to lose your new file.

CP/M issues the message

```
FILE IS READ ONLY
```

if you attempt to edit a read-only file. No harm is done. Change the file status to RW with the SET program and then start ED again.

You should not change the disk you are working on during the editing session. If you do, CP/M issues an error message telling you that the new disk cannot be written on. This is a serious error because you will lose your new file.

Summary

In this chapter, you have learned how to use the text editor, ED. This is a general-purpose editor that allows you to modify text with just a few commands.

You will need to use ED if it is the only editor provided with your computer. Although the operation of ED is less convenient than a specialized word processor, it is a satisfactory tool for correcting and modifying existing files.

In learning about ED's operation, we studied the edit buffer and the commands for manipulating text in the buffer and transferring it to and from the disk. You learned how to manipulate the character pointer, locate strings of characters, display and alter text in the edit buffer, and repeat groups of commands. A summary of the ED commands is given in Chapter 7.

6

Inside CP/M Plus

In this chapter we present the organization of CP/M Plus on the Commodore 128 in detail. If you want to use CP/M for a specific task, such as composing a letter with an editor or solving a problem with BASIC, then you won't need the information in this chapter. On the other hand, you will want to read this chapter if you are interested in learning more about the internal operation of the system. In particular, this chapter will be useful if you want to write assembly language programs that use the resources of CP/M Plus. However, CP/M Plus is very complicated. Alterations to CP/M itself can be very difficult and therefore are not discussed here.

To fully understand this chapter you should be familiar with the operation of computers, including operating systems. In particular, you should understand the operation of a system editor, assembler, linking loader, and debugger. Therefore, terms such as memory buffer and base-page area will not be defined. If you need to review this material you should refer to my *Mastering CP/M* (SYBEX, 1983).

We first present a brief description of the logical components of CP/M, their roles, and the way they interact with each other. Then we describe the memory allocation—that is, the way in which these software modules are arranged in memory—and the organization of the file system. Finally, we will discuss the operation of CP/M Plus in detail so that programs you write can use the resources of the operating system.

The Components of CP/M Plus

CP/M has three functional modules: the console command processor (CCP), the basic input/output system (BIOS), and the basic disk operating system (BDOS). Refer to Figure 6.1 as we consider each of these modules.

The Console Command Processor (CCP)

The user communicates with the *console command processor (CCP)* by entering commands at the keyboard. The CCP responds by displaying information on the video screen. The CCP also executes the user's commands by calling on the resources of the BIOS and the BDOS.

The CCP is programmed to respond to the six built-in commands we have studied—DIR, DIRSYS, TYPE, RENAME, ERASE, and USER. It can also change the default drive. If advanced features of DIR, TYPE, RENAME, and ERASE are required, the CCP automatically loads the corresponding COM file. If a transient program is requested, the CCP loads the program and executes it. If the CCP cannot find a match for the command it issues an error message.

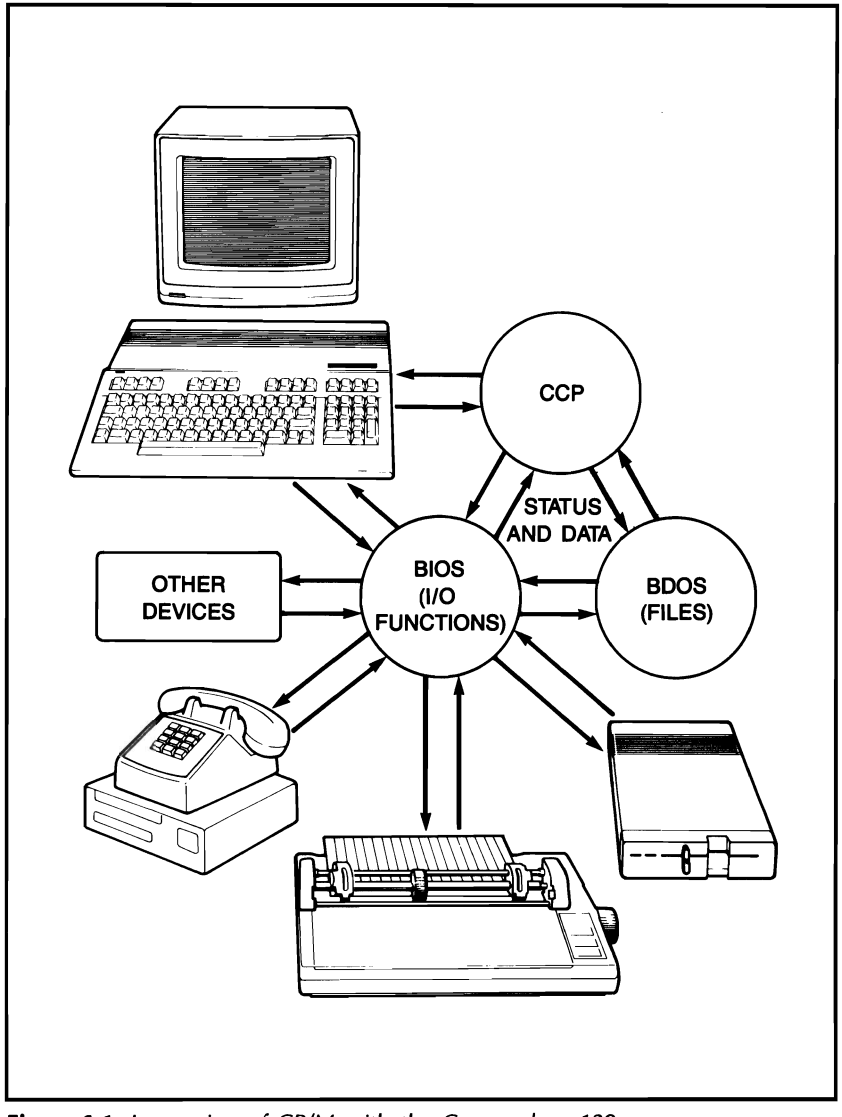


Figure 6.1: *Interaction of CP/M with the Commodore 128*

The Basic Input/Output System (BIOS)

The *basic input/output system (BIOS)* contains the routines needed to transfer information between CP/M and the peripherals connected to the computer. These include the keyboard,

the video screen, the printer, the disks, and perhaps a telephone modem. The BIOS communicates primarily with the BDOS.

The BIOS must be specifically written for each different type of computer; thus, it is device dependent. The BIOS for one computer will be different from the BIOS for another computer if the hardware is different.

The Basic Disk Operating System (BDOS)

The *basic disk operating system (BDOS)* contains the routines for operating the peripherals such as the keyboard, printer, and disks through the BIOS. The BDOS also contains routines for reading and writing disk files, displaying strings of characters on the video screen, and generally managing the resources of the computer. The CCP and transient programs call on the BDOS to perform operations with the peripherals and to manage disk files.

The BDOS is the same for all computers running the same version of CP/M; thus it is device independent. The BDOS calls the BIOS to perform the operations with the peripherals.

The Organization of CP/M Plus

Memory Allocation

CP/M Plus runs on the Z80 CPU of the Commodore 128. This CPU can directly address a maximum of 64K bytes of main memory. However, it is possible to use more than 64K bytes by alternately switching one region (bank) of memory for another. This operation, called *bank switching,* requires at least 128K bytes of memory and some form of memory management.

Though CP/M Plus is able to manage as many as 16 different memory banks, the Commodore 128 uses only three banks. Furthermore, the logic needed to accomplish bank switching must stay in memory that is accessible to all banks. Consequently, there is a fixed portion of memory common to all memory banks. In the Commodore 128, a total of 128K bytes of memory is arranged as one fixed region of 32K bytes and three switchable banks of 32K

bytes each. There are 64K bytes available for each combination—32K bytes are switched, and 32K bytes are common.

Let us look at the partitioning of memory in the Commodore 128 computer.

Partitioning of the Memory

The main memory of the Commodore 128 is divided into five regions: the BDOS, the BIOS, the transient program area (TPA), the memory buffer area, and the base-page area. The CCP is a separate module that is located in the TPA when it is active.

With the banked version of CP/M Plus, both the BDOS and BIOS are divided into two parts. A small portion of each is located in the common area of memory. The remainder is located in memory bank 0. The TPA is found in both bank 1 and the common portion of memory. The page-zero region is also assigned to the beginning of bank 1. Bank 2 and any other banks are used for disk buffering. A duplicate copy of the CCP is kept in bank 0. Figure 6.2 shows the memory for a three-bank system.

Let us now consider the organization of the disks in CP/M Plus.

Organization of the Disks

CP/M Plus is a disk operating system. We have seen that each disk is partitioned into concentric tracks, which are further divided into sectors. A few tracks, known as the system tracks, are allocated to part of the CP/M operating system. The remainder of the disk is the data area. A small portion of the data area contains the directory listing of programs stored on the disk. The remainder of the disk is available for program storage.

The system tracks are not accessible. They contain the loader programs for starting CP/M each time the computer is turned on. The BDOS and BIOS are located in a regular disk file called CPM+.SYS. This portion of CP/M must be loaded into memory only when the computer is first turned on, so the CPM+.SYS file is not needed after CP/M has begun operating. For the Commodore 128, the CCP is present as the file CCP.COM in the regular data area of the disk.

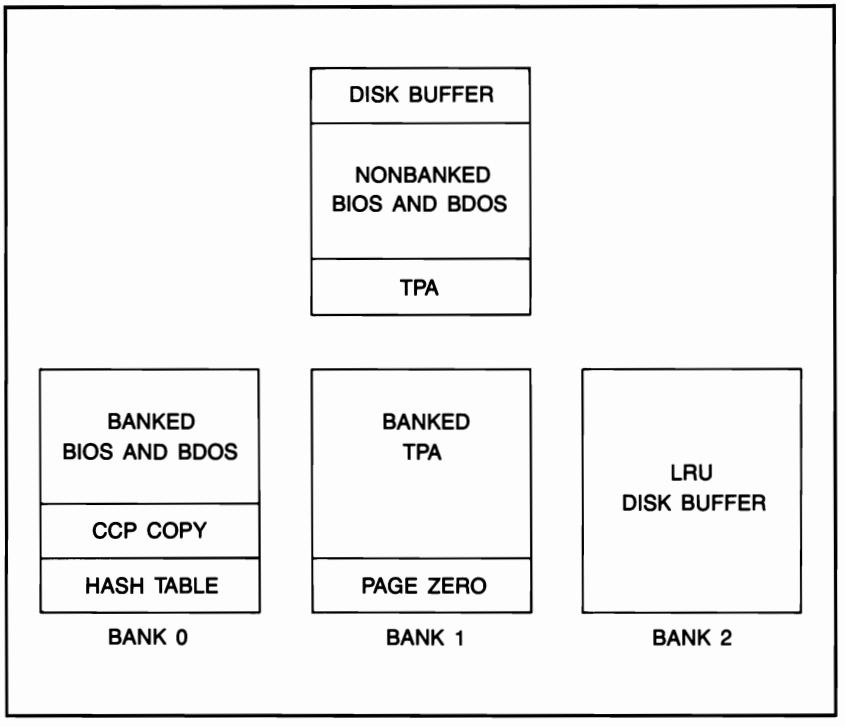


Figure 6.2: *Memory Organization for the Three-Bank Version*

The files CPM+.SYS and CCP.COM can be declared as system files, so they will not appear in the regular DIR listing. However, as supplied by Commodore, they do appear in the listing.

The Operation of CP/M Plus

The File System

One of the primary functions of CP/M, as with any disk operating system (DOS), is the effective and convenient management of disk-based files. The BDOS is the part of CP/M that contains the routines for managing disk files.

We have seen that the disks are organized into concentric tracks. CP/M further partitions the tracks into logical sectors of 128 bytes. Each sector is called a record. Every file on the disk consists of a collection of records. However, as we learned in Chapter 3, CP/M allocates disk space in multiples of 1K (actually 1,024) bytes, called a block or group. Depending on the disk format, a block size may be 1K, 2K, 4K, 8K, or 16K bytes. Thus, a 1K-byte block contains 8 sectors, and a 16K-byte block contains 128 sectors. The Commodore 128 uses 1K-byte blocks.

A basic requirement of an operating system is the ability to designate a file with a symbolic name. As we have seen, CP/M uses a name of up to eight characters and an extension of up to three characters to designate the type of file. When a file is referenced by name, CP/M searches the disk directory until it finds a matching entry and then copies the corresponding sectors into memory.

An efficient file system must include security and safety features for the period when files are being accessed. For example, files may be specified as read only or read/write. A file may require a password for access, or it may require a password to change the file but not to read it. In these cases the file is equipped with a protection device that prevents unauthorized access or execution. CP/M Plus provides all these features.

The File Control Block

Because each file may contain many records, some form of structure must be established to keep track of all the parts. For CP/M the list of blocks belonging to each disk file is contained in a special unit called a *file control block (FCB).*

Each CP/M file is described by a 32-byte FCB, which is divided into two parts. The first part contains the file name and associated user number, file length, and attributes such as write protection. The second part of the FCB gives the locations of the blocks containing the file. Large files may need additional FCBs to specify all parts of the file.

Each file control block is stored in the directory area of the disk. (Actually, the directory simply uses one or more of the

blocks at the beginning of the data area.) Each directory sector contains four FCBs. Whenever a file is accessed, the sector containing the corresponding FCB is copied into memory. The system parameter area of main memory starting at 80 hexadecimal is the default location for the memory version of the FCB. However, a program can change the location to any part of the TPA. The memory version of the FCB is illustrated in Figure 6.3; it can be accessed quickly and conveniently by the operating system.

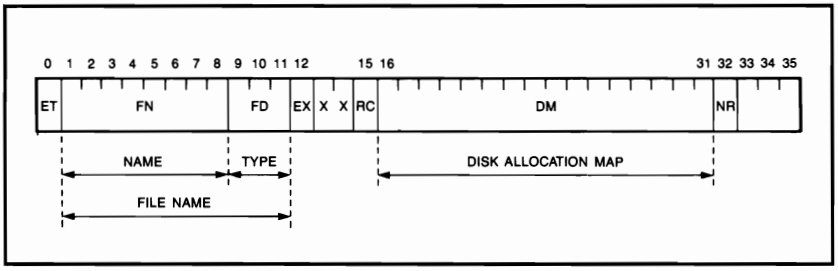


Figure 6.3: *The Memory Version of the File Control Block*

Each time you read from or write to a disk file, you must access the FCB. Therefore, let us study this device in detail.

Note that the two versions of the FCB—one located in the disk directory and the other in memory—may not always be the same. For example, when a file is created or altered, the memory FCB correctly reflects the file, but the disk FCB does not. Therefore, when it is necessary to distinguish the two versions, we will use the expressions *memory* FCB and *disk* FCB. Table 6.1 identifies each part, or *field,* of the FCB shown in Figure 6.3. Each byte of the 32-byte FCB is referenced by a position of 0–31.

When a disk file is created, the memory FCB is updated as each new block is written to the disk. However, the disk FCB has not been changed yet. Therefore, no permanent record is made until the new file is closed at the end of the write operation.

System Operation

When the computer is first turned on, the bootstrap loader, built into the computer, loads a larger loader, called CPMLDR,

Field (position)	Description
ET (0)	(Disk FCB) The user number 0–F hex, or E5 hex if the file is erased. (Memory FCB) The drive identifier, where 0 = default drive, 1 = drive A, 2 = drive B, and 5 = drive E.
FN (1–8)	The file name, one to eight ASCII characters. If there are fewer than eight characters, the remainder of the field is filled with blanks.
FD (9–11)	The file type, zero to three ASCII characters. If there are fewer than three characters, the remainder of the field is filled with blanks. Parity bits indicate attributes. The parity bit of position 9 indicates write protection. The parity bit of position 10 indicates a system file. The parity bit of position 11 indicates the archive attribute (whether a file has been altered since the last archive).
EX (12)	The file extent. 0 for small files. Large files require more than one FCB. This byte indicates the sequence for multiple FCBs of larger files.
XX (13, 14)	These fields may be used for very large disks. Normally set to 0.
RC (15)	The record count. The number of 128-byte records in the extent.
DM (16–31)	The disk allocation map designates the locations on disk for each block used by the file. There are 16 one-byte block numbers.
NR (32)	(Memory FCB only) For sequential access. The next record number to be read or written, normally 0.

Table 6.1: *The Fields of the File Control Block*

from the system tracks of drive A. The BDOS and BIOS, contained in the disk file CPM+.SYS, are copied into memory, and control is passed to the cold-start portion of BIOS. The BIOS then loads the CCP into memory. A separate copy of the CCP is copied into bank 0 at this time. Then control is passed to the CCP so it can monitor the keyboard for input.

If one of the six built-in commands is typed at the keyboard, the CCP carries out the operation and then monitors the keyboard for another command. If a transient program is to be executed, the CCP loads the requested program into the TPA and passes control to it. This destroys the working copy of the CCP, since it is located in the TPA.

When a transient program is executed, it may use the resources of CP/M by making subroutine calls to memory address 5. This address provides access to the routines of the BDOS. It is a single, fixed entry point independent of the actual memory size or types of hardware.

The call to the BDOS must be accompanied by a parameter specifying the desired operation. This parameter, called a *function number,* is contained in the C register of the Z80 CPU. Data are transmitted to BDOS in register E or DE, and BDOS returns information in the accumulator or the HL register.

If one or two parameters were entered with the command, the CCP constructs the beginning of file control blocks for each parameter before loading the program. For example, if the command line is

```
VERIFY SORT.BAS SORT.BAK
```

the CCP will construct one memory FCB for SORT.BAS and another for SORT.BAK. It will then copy VERIFY.COM into the TPA and execute it.

At the completion of the program, the BIOS reloads the CCP and returns control to it. Since a duplicate copy of the CCP is located in bank 0, the system can rapidly restore the working version of the CCP in memory bank 1. This does not require a disk access.

Next we look at program interaction with the BDOS.

Program Interaction with the BDOS

We have seen that the BDOS contains the general routines for operation of the peripherals. These routines are available to a transient program while it is operating. To use a feature of BDOS, the program places the corresponding BDOS function number in the CPU register C and the required data in register E or DE depending on the data size. Then the program calls the BDOS entry at address 5. The BDOS returns data in the accumulator or HL.

The BDOS functions are summarized in Table 6.2. Additional information can be found in other sources, including the *CP/M Plus Operating System Programmer's Guide* (Digital Research, 1982).

Let us look at a few examples of BDOS calls. A transient program can send a character to the printer by placing the character in register E, the BDOS function number 5 in register C, and calling address 5. A string of characters terminated by a dollar sign can be displayed on the video screen by placing the address of the string in the DE register and function number 9 in register C. Then BDOS is called at location 5.

Accessing an Existing Disk File

Before an existing disk file can be accessed, it must be opened with BDOS function 15. The transient program places in the DE register the address of the short FCB created by CP/M. A value of 15 is placed in register C, and address 5 is called. BDOS then searches the disk directory for the corresponding file. When the file is found, CP/M fills out the remainder of the memory FCB according to the disk version of the FCB. This information includes the size of the file, the attributes of the file, and the block numbers corresponding to the location of the file on the disk.

Let us consider a simple example of a BDOS call—the opening of an existing disk file. Suppose that a transient program with the file name READ.COM is designed to process a disk file given as a parameter on the command line. The command line might be

```
READ FNAME.EXT
```

Function Number (in C)	Operation	Value Sent	Value Returned
0	Reset disks		
1	Console input		Character in A
2	Console output	Character in E	
3	Auxiliary input		Character in A
4	Auxiliary output	Character in E	
5	Printer output	Character in E	
6	Direct console input	FF hex	A = 0 not ready
	Direct console output	Character	Character in A
7	Auxiliary input status		A = 0 not ready
8	Auxiliary output status		A = 0 not ready
9	Print console buffer	DE = address	
10	Read console buffer	DE = address	
11	Return console status		A = 0 not ready
12	Return CP/M version		Byte in A and L
13	Reset disks		
14	Select disks	E = disk	A = error code
15	Open file	DE = FCB	A = error code
16	Close file	DE = FCB	A = error code
17	Search for first	DE = FCB	A = error code
18	Search for next		A = error code
19	Delete file	DE = FCB	A = error code
20	Read sequential	DE = FCB	A = error code
21	Write sequential	DE = FCB	A = error code
22	Make new file	DE = FCB	A = error code

Table 6.2: *The BDOS Functions for CP/M Plus*

Function Number (in C)	Operation	Value Sent	Value Returned
23	Rename file	DE = FCB	A = error code
24	Return login vector		HL = vector
25	Find default drive		A = drive
26	Set DMA address	DE = address	
27	Get alloc vector		HL = vector
28	Write protect disk		
29	Get R/O vector		HL = vector
30	Set file attributes	DE = FCB	
31	Get disk parm block		HL = block
32	Get/set user number		
33	Read random	DE = FCB	A = error code
34	Write random	DE = FCB	A = error code
35	Get file size	DE = FCB	A = error code
36	Set random record	DE = FCB	
37	Reset drive	DE = drive	
40	Write random/zero fill	DE = FCB	A = error code
44	Set multisector count	E = # sectors	A = error code
45	Set BDOS error mode	E = mode	
46	Get free disk space	E = drive	A = error code
47	Chain to program	E = chain flag	
48	Flush buffers	E = purge flag	A = error code
49	Get/set system control block	DE = SCB	Word in HL or byte in A
50	Direct BIOS call	DE = BIOSPB	

Table 6.2: *The BDOS Functions for CP/M Plus (continued)*

Function Number (in C)	Operation	Value Sent	Value Returned
59	Load overlay	DE = FCB	A = error code
98	Release free blocks		
99	Truncate random file	DE = FCB	
100	Set directory label	DE = FCB	
101	Get directory label	E = drive	
102	Get date and password	DE = FCB	
103	Write file XFCB	DE = FCB	
104	Set date and time	DE = DAT address	
105	Get date and time	DE = DAT address	A = seconds
106	Set default password	DE = password addr	
107	Return serial number	DE = number field	
108	Get program return code	DE = FFFFH	HL = code
	Set program return code	DE = code	
109	Get console mode	DE = FFFFH	HL = mode
	Set console mode	DE = mode	
110	Get output delimiter	DE = FFFFH	A = value
	Set output delimiter	DE = value	
111	Print block	DE = CCB	
112	List block	DE = CCB	
152	Parse file name	DE = PFCB	

Table 6.2: *The BDOS Functions for CP/M Plus (continued)*

When this command is given, CP/M constructs the first part of the FCB for the parameter FNAME.EXT at the location 5C hex. Then it loads the program READ.COM into the TPA and executes it.

Before the transient program can access the file given as the parameter, the BDOS open function must be executed. The C register is loaded with the value of 15, and the DE register is given the location of the memory FCB. A simple 8080 assembly language program (which can be executed by a Z80 computer) to open a disk file might look like this:

```
LXI     D,5CH     ;file control block
MVI     C,15      ;open function
CALL    5
INR     A
JZ      ERROR
```

The first instruction loads the DE register with the address of the memory FCB, the one set up by CP/M. The second instruction loads the C register with function number 15. The third instruction calls BDOS to perform the open function. If the requested file cannot be found in the directory, BDOS returns the value of 255 as an error message. The INR instruction changes this to zero. Thus, the zero flag will be set if there is an error. Then the final instruction causes a branch to an error routine.

Since the BDOS does not preserve the CPU registers, a more sophisticated routine will save the registers on the stack before calling BDOS. The registers will then be restored after return from BDOS.

Creating a New Disk File

Creating a file from a memory image is a little different from reading a file. The make function, 22, is used to generate a new directory entry on the disk. Then the write function, 21, copies the file to the disk. However, at this point only the memory FCB contains the corresponding block numbers. The disk version has only the file name. Therefore, the close function, 16, must be given to update the disk FCB from the memory FCB.

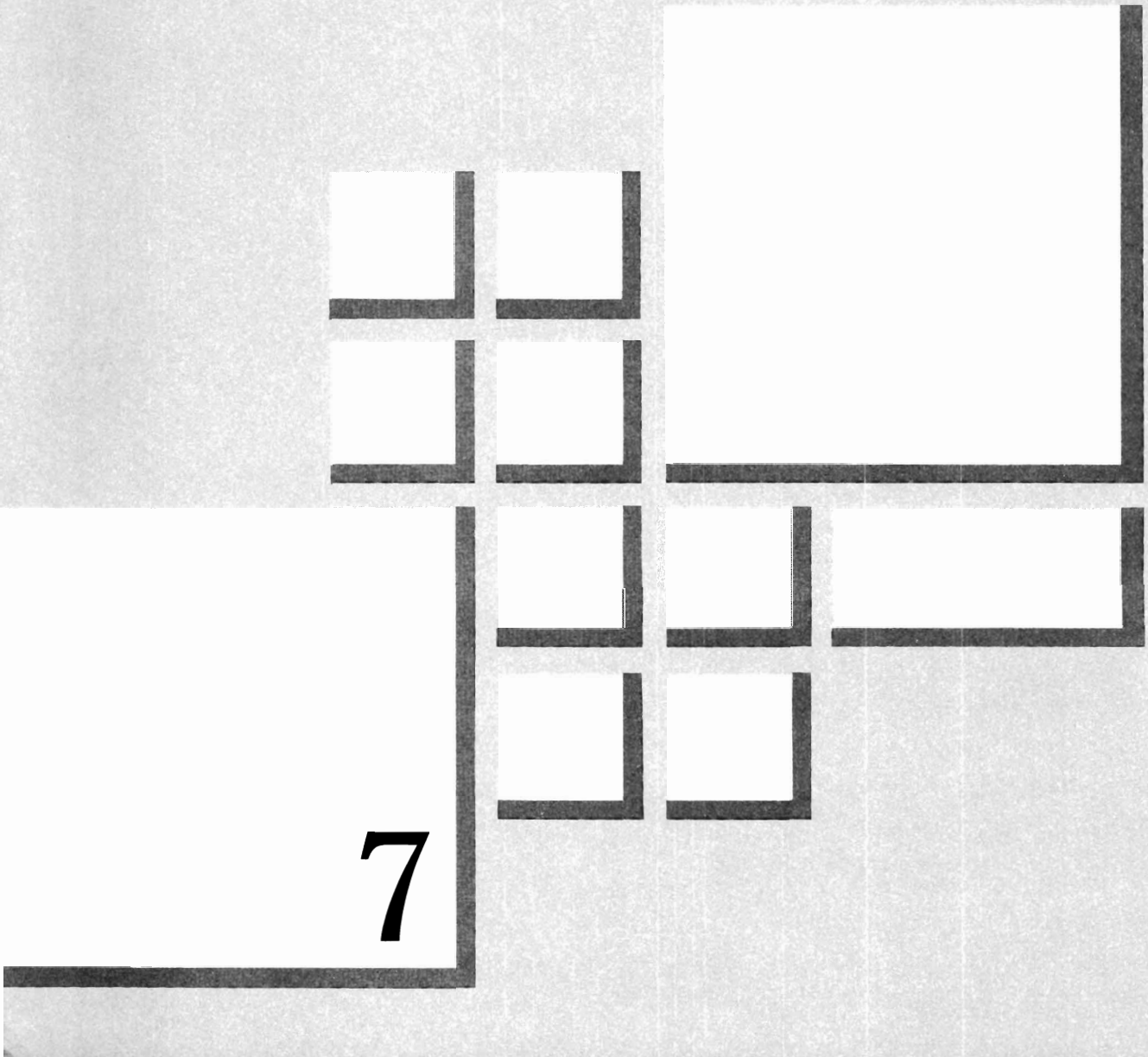
Direct BIOS Calls

When you write your own computer programs, you can indirectly access the keyboard, the printer, and the disks through the BDOS entry point at address 5. With the banked version of CP/M Plus, the BIOS is not located in the same bank as the TPA. Thus, executing programs that need to directly access the BIOS can use BDOS function 50.

Summary

The organization and operations of CP/M Plus on the Commodore 128 have been described in this chapter.

You learned that CP/M has three major parts—the CCP, the BDOS, and the BIOS—and you learned how the parts interact with each other. You also learned how the CCP interacts with the user. In addition, we saw how executing programs can use the peripherals and the disk file system by calling on the BDOS. In particular, we studied the form of the file control block that keeps track of each disk file.


7

CP/M Plus Command Summary

This chapter provides a quick reference guide to all CP/M Plus commands and programs for the Commodore 128, even those that are not described elsewhere in this book. The commands are arranged in alphabetical order, with each beginning on a new page. The keyword at the top of the page will help you locate the commands quickly. In addition, information is organized into consistent categories within each section for easy reference.

Each command is introduced by a brief statement describing its purpose. It is then identified as built-in, transient, or a combination of the two. Next, the format of the command is presented through examples.

Note that when a file-name parameter is shown in the form PNAME, you must give only the primary part of the file name; you do not include the extension. When a parameter is given in the form PNAME.EXT, then you must enter both the primary name and the extension. When a parameter refers to a file on the current drive, you need not append the drive name to the file name. However, when the file is not on the current drive, you must include the drive name with the file name.

The symbol D: by itself means that you are to enter only the drive name followed by a colon. The expression *ambig* indicates that you can include the ? and * ambiguous (wild-card) symbols in the file name to reference more than one file. The expression [option] means that you may give one or more options, enclosed in a single pair of brackets. Alternatively, a particular option may be shown, for example [FULL]. The left bracket immediately follows the previous parameter or command with no intermediate space. In this chapter we do not include the expression <RETURN> at the end of each command line. But of course, you must press the RETURN key when you want the Commodore 128 to execute the line.

Following the format is a description of the command. Some commands also include brief comments on how to use the command or give a reference for further information.

Remember that this chapter is for quick reference use. If you need more detailed explanation of a command, turn to the chapter in the text where it is discussed.

DATE

Display and set date and time

(DATE.COM transient)

Format

```
DATE
DATE CONTINUOUS
DATE SET
DATE MM/DD/YY HH:MM:SS
```

Description

This program displays and alters the date and time.

Use

When you execute the command without a parameter, CP/M presents a single display of the day of the week; the month, day, and year; and the time in hours, minutes, and seconds. For example:

```
Mon 07/14/86 11:15:33
```

In the previous example the date and time are displayed, then control returns to CP/M. However, if you want to watch the displayed time change in accordance with the actual time, give the parameter CONTINUOUS or its abbreviation, C. Then the display will emulate a digital clock. No other task can be performed while this is happening. Press any key to terminate the program.

The date and time can be altered in one of two ways. If you give the parameter SET, the program asks for the date and the time separately. You can skip over one if want to change only the other. Alternatively, you can specify the complete date and

time in the parameter with a command. For example:

```
DATE 07/14/86 11:15:83
```

Choose a time that is a few minutes ahead. The time and date will be set and then the message

```
Press any key to set time
```

will appear. Press the space bar at the correct time. Notice that you do not enter the day of the week when setting the date. The program calculates this for you. Since the clock is not battery powered, you must reset the date and time whenever the computer is turned on or reset. This task can be simplified by adding the line

```
DATE SET
```

to the end of the PROFILE.SUB file. (See the heading SUBMIT in this chapter.) Then DATE will automatically ask you for the date and time each time you perform a cold boot or reset the computer.

DEVICE

Display and alter peripheral assignments

(DEVICE.COM transient)

Format

```
DEVICE NAMES
DEVICE VALUES
DEVICE CONOUT:=CRT,PRNTR
DEVICE CON:[PAGE]
DEVICE CON:[COLUMNS=nn, LINES=mm]
DEVICE LST:=PRT2[NOXON,2400]
DEVICE
```

Description

This program is used to display and alter the mapping of the logical devices CON:, LST:, and AUX: to the actual peripheral devices such as console (video screen and keyboard), printer, and modem. It is possible to map one logical device to more than one peripheral. For example, console output can be sent simultaneously to both the video screen and the printer. DEVICE can change the number of columns and lines that are displayed on the video screen.

Use

The use of this command is dependent on the number and type of peripherals attached to your computer and the device names selected for the Commodore 128.

When DEVICE is executed with the NAMES parameter, a listing of the peripheral devices and their transfer rate (if serial) is given. These names are assigned by the computer manufacturer and so will be different from one computer to the next. DEVICE executed with the VALUES parameter shows the current mapping

of these devices to the CP/M logical devices. These are results for the Commodore 128:

```
CONIN: = KEYS
CONOUT: = 80COL
AUXIN: = Null Device
AUXOUT: = Null Device
LST: = PRT1
```

The device CONIN: refers to the keyboard and the device CONOUT: refers to the video screen. The abbreviated form CON: refers to both. CONSOLE and KEYBOARD are synonymous with CON:. (Notice that there is no colon at the end of these two names.) Similarly, PRINTER is synonymous with LST:. The DEVICE command executed with the CONOUT parameter makes output appear at both the printer and the video screen. Refer to the PUT and GET commands for similar operations.

The [PAGE] option shows the current number of columns and lines displayed on the video screen. The next option sets the video display size to the number of columns and lines you wish.

The transfer rate for a serial device can be changed by the mapping command. For example, the command:

```
DEVICE LST:=PRT2[NOXON, 2400]
```

maps the PRT2 device to the logical printer and sets the transfer rate to 2,400. You must also select either XON to enable the X-ON protocol or NOXON to disable it. Generally, you want NOXON unless an X appears in the NAMES listing.

DEVICE executed without a parameter lists both the output from the NAMES parameter and the output from the VALUES parameter. Then the program waits for your input.

For further explanation, see Chapter 3.

DIR

Display a list of file names in the disk directory.
Determine file size and file attributes

(Built-in command and DIR.COM extension)

Format

Built-in

```
DIR
DIR D:
DIR PNAME.EXT
DIR ambig
```

Transient extension

```
DIR[option]
DIR PNAME.EXT[option]
DIR ambig[option]
```

Description

DIR is a very versatile program that is used to list the disk files and their characteristics. The built-in version can be executed from any drive and user area, but its features are limited, producing a simple listing of all nonsystem file names that match the filename parameter. There are four forms of this version. The wild-card symbols ? and * may be included in the name. If no parameter is given, all nonsystem files belonging to the current drive and user area are listed.

Options given in brackets increase the power of DIR. When options are included, the file names are sorted in alphabetical order. Other specific functions of individual options are summarized in the complete listing of DIR options below.

If no files match the specification, the message NO FILE appears. If system files are present, the message SYSTEM FILE(S) EXIST is shown.

Options

Be sure to place options within brackets when giving the command.

Option	Function
ATT	Display user-defined file attributes.
DATE	List files with date and time (if activated).
DIR	List only nonsystem files (default).
DRIVE=ALL	List files for all logged-in drives.
DRIVE=A	List files for specified drive.
DRIVE=(A,E)	List files for specified drives.
EXCLUDE	List files that do not match file name.
FF	Insert form-feed character at start of listing.
FULL	List files with complete description.
LENGTH=*n*	Place heading after *n* lines.
MESSAGE	Display names of disks and user areas as they are processed.
NOPAGE	Display listing continuously.
NOSORT	Do not sort listing alphabetically.
RO	List only files that are read only.
RW	List only files that are read/write.
SIZE	Show size of each file.
SYS	List only system files.
USER=ALL	List files for all user areas.
USER=2	List files for specified user.
USER=(2,3)	List files for specified users.

Use

See Chapter 3 for a detailed explanation of how to use DIR.

DIRSYS

Directory listing of system files

(Built-in command)

Format

```
DIRSYS
DIRSYS D:
DIRSYS PNAME.EXT
DIRSYS ambig
```

Description

This built-in program is a variation of DIR that provides a simple listing of system files. The DIR command with option parameters can also list system files. However, DIRSYS is faster since it is a built-in command. This command may be abbreviated to DIRS.

Use

See Chapter 3 for a description of how to use DIRSYS.

DUMP

Display nontext files in both hexadecimal and ASCII

(DUMP.COM transient)

Format

```
DUMP
DUMP PNAME.EXT
```

Description

Text files can be examined with the TYPE command or with a system editor such as ED. However, nontext files such as COM, REL, and OVR files must be processed before they can be displayed on the video screen or printer. DUMP produces a listing in both hexadecimal and ASCII that can be displayed on the video screen or the printer.

The DUMP program is useful for assembly language programmers.

ED

Edit a text file
(ED.COM transient)

Format

```
ED
ED PNAME.EXT
ED PNAME.EXT D:
```

Description

The ED program is a simple line-oriented text editor. It can create and alter text files used for computer programs, letters, and reports. It does not include a video-screen mode, nor does it provide word processing features such as justification, subscripting, and boldfacing.

Use

The operation of ED is described in detail in Chapter 5. A brief summary of the commands is given here. For explanation of how to use the commands in combination, see Chapter 5.

In the following list, *n* specifies the number of times a command is to be repeated. If *n* is omitted, a value of 1 is usually assumed. (The M command is an exception.) The # symbol is used to indicate a large, unspecified number. The +/- symbol means that *n* may be either a positive or a negative number. The plus sign is omitted from a positive number.

Command	Explanation
*n*A	Copy (append) *n* lines from the source file to the end of the text in the edit buffer.

0A	Copy lines from the source file until edit buffer is half filled.
B	Move pointer to beginning of buffer.
-B	Move pointer to end of buffer.
+/-*n*C	Move pointer forward (+) or backward (-) *n* characters.
+/-*n*D	Delete *n* characters forward (+) or backward (-) from the pointer.
E	End the edit session normally.
F*string* ^Z	Find the next occurrence of the given *string* in the edit buffer, starting at the original pointer position. If the *string* is found, move the pointer to the end of the *string*. Otherwise, the pointer is not moved. The F command must be given in lower case if lowercase letters are included in the *string*. Only the edit buffer is searched. See the N command.
*n*F*string* ^Z	Find the *n*th occurrence of the given *string*.
H	Save all changes in a temporary disk file and then restart the edit.
I	Insert text in front of the pointer. You normally give the command in lowercase because if I is given in uppercase, all inserted text is converted to uppercase. Press escape or ^Z to terminate the command.
I*String* ^Z	Insert a short string of characters. I must be in lowercase if inserted text has lowercase letters.
n J*fstring* ^Z*istring* ^Z*dstring* ^Z	Find *fstring,* delete text up to but not including *dstring,* then insert *istring.* (J indicates juxtapose.)

+/-*n*K	Delete (kill) *n* lines starting (+) or ending (-) at the pointer.
+/-*n*L	Move pointer to the beginning of the current line and then move it *n* lines forward (+) or backward (-).
0L	Move pointer to beginning of current line.
*n*M*commandstring*	Repeat (multiple) execution of the *command string n* times. If *n* is omitted or has a value of 0 or 1, execute the *command string* repeatedly until an error occurs. See also the Z command.
N string ^Z	Find the next occurrence of the given *string* starting at the original position of the pointer. If the *string* is found, move the pointer to the end of the string. In contrast to the F command, if the *string* is not found in the buffer, ED searches the original file on disk. The N command must be given in lowercase if lowercase letters are included in the *string*. See also the F command.
*n*N*string* ^Z	Find the *n*th occurrence of the given *string*.
O	Abort (omit) the editing session and return to original source file. The new changes are discarded, and the edit buffer is emptied.
+/-*n*P	Move pointer *n* pages and display the page following the pointer.
0P	Display page following the pointer. The pointer is not moved.
Q	Abort editing session, discarding all changes. The original file is intact, but the backup is deleted.

RPNAME.EXT	Read the given disk file into the buffer at the pointer position. If the file type is LIB, it may be omitted from the command. A file with no file type cannot be read. If the entire file name is omitted, X$$$$$$$.LIB is assumed.
*n*Soldstring^Znewstring^Z	Starting at pointer position, substitute *newstring* for the next occurrence of *oldstring*. Repeat *n* times. The S command must be given in lowercase if any of the letters in either string is in lowercase.
+/-*n*T	Display (type) *n* lines of text on screen, starting (+) or ending (-) at pointer. The pointer is not moved.
0T	Display text from beginning of line to pointer. Pointer is not moved.
U	Translate lowercase input to uppercase until the -U command is given.
-U	Turn off uppercase translation. Upper- and lowercase are preserved.
V	Add line numbers to display but not to text in buffer.
-V	Turn off display of line numbers.
0V	Display the number of remaining bytes in the buffer and the total size of the buffer.
*n*W	Copy (write) *n* lines from the buffer to the new file and delete the corresponding lines from the buffer. Not needed when the session is ended with the E command.
*n*XPNAME.EXT	Copy the next *n* lines of text to the end of the file named PNAME.EXT if it exists.

	Otherwise, create a new file. The corresponding lines remain intact in the buffer. If the extension is omitted, LIB is assumed. If the file name is omitted, X$$$$$$$.LIB is assumed.
0XPNAME.EXT	Delete the file PNAME.EXT. If the file name is omitted, X$$$$$$$.LIB is assumed.
*n*Z	Suspend operation for approximately one second. An *n* parameter indicates a specific number of seconds. Used with the M command.
+/-n	Move the pointer ahead or back *n* lines and display that line.
n:	Move pointer to beginning of line number *n*.
n1::*n2*	Specify a range of line numbers beginning with *n1* and ending with *n2*. If either *n1* or *n2* is omitted, the current line number is assumed. Use only with K, L, and T commands.

ERASE

Erase disk files

(Built-in command and ERASE.COM extension)

Format

Built-in

```
ERASE
ERASE PNAME.EXT
ERASE ambig
```

Transient extension

```
ERASE PNAME.EXT[CONFIRM]
ERASE ambig[CONFIRM]
```

Description

This command erases files that match the file-name parameter. Of course, write-protected files and files on a disk that is designated as read only cannot be erased. Only files in the current user area can be erased. When the wild-card symbols ? and * are included, ERASE repeats the file name and requests permission to proceed. When the option parameter is given, ERASE requests permission to erase each file. If ERASE cannot find a file to match the given parameter, it displays the message NO FILE.

Use

ERASE executed without the option parameter is a built-in command, so it can be run from any disk drive and any user area. When the option parameter is included, the transient extension is required. ERASE.COM must exist on the current drive, a drive name must be included, or a search path must be previously established with SETDEF.

This command can be abbreviated to ERA, and the confirm option can be abbreviated to C. Below are some examples of the ERASE command. For further information, see Chapter 3.

ERASE SAMPLE.TXT	(erase file on current disk)
ERASE B:SAMPLE.TXT	(erase file on drive B)
ERA *.BAK[CONFIRM]	(erase all backup files with confirmation)

FORMAT

Format a new disk
(FORMAT.COM transient)

Format

FORMAT

Description

A new disk must be formatted before it is used for the first time. It also may be necessary to reformat a disk if the power fails during a write operation. FORMAT is not a standard CP/M program. It must be specifically programmed for your computer; do not use a version from a different computer. The Commodore 128 version also writes the CP/M system files to the disk.

This program destroys any information on the disk surface. Be careful not to accidentally format a working disk.

Use

FORMAT cycles, so you can format several new disks one after the other.

For more details, see Chapter 2.

GENCOM

Create a special version of CP/M Plus
(GENCOM.COM transient)

Format

GENCOM

Description

This program is used by programmers to create a custom version of CP/M Plus or to add new CP/M features.

GET

Get keyboard input from a disk file

(GET.COM transient)

Format

```
GET CONSOLE INPUT FROM FILE PFILE.EXT
GET CONSOLE INPUT FROM CONSOLE
GET CONSOLE INPUT FROM FILE PFILE.EXT[SYSTEM]
GET FILE PFILE.EXT[NOECHO]
```

Description

With the GET command it is possible to input information from a disk file rather than from the keyboard.

Use

The first form of the GET command instructs CP/M to execute the next command or program given at the keyboard. Then if keyboard input is needed, it is to be taken from the disk file named PFILE.EXT rather than from the keyboard. Input is again taken from the keyboard when the program terminates or when all the commands in the disk file have been executed.

The second form of the GET command, either placed into a file or given from the keyboard, returns keyboard input to its normal mode. The third form of the command immediately switches keyboard input to the given disk file so that the command line can also be read from the disk file.

As commands are read from the disk file, they are normally displayed (echoed) on the video screen as though they were typed from the keyboard. This display can be suppressed with the option NO ECHO, shown in the fourth form of the command. The words CONSOLE INPUT FROM may be omitted from any form of the GET command. Refer to SUBMIT for a similar operation.

HELP

Learn more about CP/M
(HELP.COM and HELP.HLP transients)

Format

```
HELP
HELP topic
HELP topic subtopic
HELP topic[option]
HELP.subtopic
HELP[option]
```

Description

You can execute the transient program HELP.COM, with its gigantic work file, HELP.HLP, to learn more about CP/M Plus commands and programs. Since it is not possible to use the HELP program while another program is executing, it may be more useful to refer to this book when you have a problem.

Use

When executed without a parameter, HELP produces a list of topics that are available. You can then enter the desired subtopic in response to the HELP prompt. If you choose a subtopic directly, insert a period at the beginning of the response.

The listing stops after the screen fills. To continue the listing, press the RETURN key. Alternatively, you can obtain a printed listing by toggling the printer with ^P. Then you give either the NOPAGE or the LIST option so that the listing will continue automatically. The NOPAGE option leaves gaps at the end of each page, whereas the LIST option prints continuously.

The HELP.HLP work file contains the text that is displayed on the video screen. However, the text is not in a form that can be

altered with a system editor like ED, since some of the characters are not ASCII. Therefore, you must use the EXTRACT option to create a HELP.DAT work file that can be edited. Alternatively, you can create your own HELP.DAT file containing new material. Then use the CREATE option to combine your HELP.DAT file with the original HELP.HLP file.

HEXCOM

Generate an executable COM file from an Intel HEX file

(HEXCOM.COM transient)

Format

```
HEXCOM PNAME
HEXCOM
```

Description

This is a routine for assembly language programmers. MAC produces files in the Intel hexadecimal format. HEXCOM creates a transient COM file from the corresponding HEX file. See the MAC command.

INITDIR

Prepare disk directory for date and time stamping

(INITDIR.COM transient)

Format

```
INITDIR D:
```

Description

CP/M Plus can encode two different values of time and date for each file in the disk directory. (This is sometimes called date and time tagging, or stamping.) The information is stored not with the regular directory entry, but in a separate section of the directory. Furthermore, the directory area must be specially prepared for time and date stamping. Consequently, the INITDIR command should be given immediately after a new disk is formatted and before any files have been saved. There must be sufficient directory space for the new time and date listings.

Run the program SET to choose the mode of date and time stamping or to disable this feature.

Use

Execute INITDIR with the desired disk parameter. Since INITDIR alters the directory, the program asks for confirmation of the command. Furthermore, if the power should go off while you are reformatting the directory, you can lose your files, so be sure to make a backup disk before executing INITDIR.

LIB

Create and edit a library of compiled subroutines
(LIB.COM transient)

Format

```
LIB PNAME.EXT[options]
```

Description

Many Pascal, FORTRAN, COBOL, and BASIC compilers and the RMAC and MACRO-80 assemblers produce relocatable modules with the file extension REL or IRL. A collection of useful routines can be combined into a single library using the LIB program. Refer to instructions for these languages.

LINK

Create an executable file from relocatable modules
(LINK.COM transient)

Format

```
LINK PNAME, PNAME2, PNAME, . . .[options]
```

Description

Many PASCAL, FORTRAN, COBOL, and BASIC compilers and the RMAC and MACRO-80 assemblers produce relocatable modules with the file extension REL. A collection of modules can be combined into an executable COM file with LINK. Refer to instructions for these languages.

MAC

Macroassembler for assembly language programs

(MAC.COM transient)

Format

```
MAC PNAME
MAC PNAME $options
```

Description

The macroassembler MAC creates three new files from an assembly language source program and an optional macro library. The source program has a file type ASM, and the macro library has the type LIB. The assembled code is written to a HEX file, the symbol table is found in a SYM file, and the printer listing is a PRN file. The separate program HEXCOM creates an executable file from the HEX file (see HEXCOM).

A dollar sign symbol rather than the usual bracket precedes the command line options. A space must precede the dollar sign. If no options are given, all files are referenced to the default drive. However, any other drive can be selected using the appropriate options. See a macro assembler manual and my *Mastering CP/M* (SYBEX, 1983) for more details. Also see RMAC.

PATCH

Install revision to CP/M Plus or a program
(PATCH.COM transient)

Format

PATCH PNAME
PATCH PNAME *n*

Description

When a major revision to a computer program is necessary, the program may be completely replaced with a separate version. However, minor revisions to the CP/M Plus operating system or to existing transient programs can be performed by overlaying the changes to a memory version of the original. Then the revised version is saved to disk. This process is known as *patching,* and the file containing the changes is called a *patch.*

Patches can be automatically installed by executing PATCH with the corresponding patch routine from Digital Research. If more than one revision has been issued, they are referenced by patch numbers starting with 1.

PATCH can either install a patch or indicate whether a patch has been installed.

PIP

Copy disk files and transfer files between peripherals

(PIP.COM transient)

Format

```
PIP
PIP PNAME2.EXT=PNAME1.EXT[option]
PIP ambig1=ambig2[option]
PIP D:=PNAME.EXT[option]
PIP LARGE=SMAL1,SMAL2,SMAL3
```

Description

PIP, which stands for peripheral interchange program, can copy a disk file, concatenate several files into a new file, send a disk file to the video screen or printer, or transfer a file from one peripheral to another. A file can be altered during the copy operation by making all letters lower- or uppercase, zeroing the parity bit, adding line numbers, and expanding the tab character. A portion of a file can be extracted and made into a smaller separate file. The file can be displayed on the video screen during transfer.

Chapter 4 is entirely devoted to the operation of PIP. A summary of how to use the command is included here, and the option parameters are listed. Refer to Chapter 4 if you need more detailed explanation of any PIP operation or option.

Use

PIP can be executed with or without parameters. In the latter case PIP presents an asterisk prompt so that you can give a sequence of commands. Each PIP command contains two file-name parameters separated by an equal sign. PIP creates the destination file (the first parameter) from the source file (the second parameter). There may also be option parameters surrounded by a single pair

of brackets. Then the left bracket must immediately follow the file-name parameter without a space.

The following example commands show how PIP works in various cases. Assume A is the current drive.

To copy a single file from the current disk to the other, give the command

```
PIP E:=PNAME.EXT[V]
```

The V option parameter forces PIP to verify that the copy is correct.

To copy a collection of files from drive A to drive E, give the command

```
PIP E:=PNAME?.*[V]
```

where the ? and * wild-card symbols can match several files.

To copy all files from A to E, give the command:

```
PIP E:=*.*[V]
```

To duplicate a file with a different name on the current disk, give the command:

```
PIP PNAME2.NEW=PNAME1.OLD[V]
```

To concatenate two files into a new single file, give the command:

```
PIP PNAME3.NEW=PNAME1.OLD[V],PNAME2.OLD[V]
```

To copy a file from the current drive and default user area to drive E and user area 2, give the command:

```
PIP E:[G2]=PNAME.EXT[V]
```

To extract a portion of a text file, give the commands:

```
PIP
PSHORT.EXT=PFULL.EXT[Sstart^Z Qstop^Z]
```

The new file PSHORT.EXT will contain the text from the string *start* through the string *stop*. If lowercase letters appear in either string, PIP must be executed initially without a parameter. Notice that each string is terminated by a ^Z.

Options

The following list summarizes the PIP option parameters. See Chapter 4 for more detailed explanation.

PIP options are enclosed in a single pair of brackets. The left bracket immediately follows the file-name parameters without a space. The options may be placed in any order and separated by spaces or run together. Note that in the list the letter *n* stands for an unspecified number. For example, *n* columns means the number of columns you will enter with the parameter.

Option	Function
A	Archive mode. Copies only files that have been modified since the previous archive copy operation. The name of the copied file is altered so the file will not be copied the next time.
C	Confirm. Request permission to copy each file that matches a file name containing wild cards.
D*n*	Delete after *n* columns. PIP deletes characters that extend beyond *n* characters of each line. Use this parameter to reduce the length of long lines if you are sending a file to a narrow printer or video screen. Very long lines are not handled correctly. The tab character must be expanded with the T option. Use only for text files.
E	Display (echo) text on video screen. Normally, disk-to-disk transfers are invisible. If you want to investigate the operation of several of the PIP parameters described in this section, especially those that alter the file, it

may be helpful to observe the copying operations on the video screen as it is being performed. Do not use with the N option. Use only for text files.

F — Filter (remove) form feeds. The ASCII form-feed character, ^L, appears in certain types of files to indicate when to start a new page. Some printers will properly begin a new page when this character is encountered. However, for other types of printers you should use the F option to remove the form-feed character. Also use the F parameter with the P parameter to change the page size. Use only for text files.

G*n* — Move (get) a file to or from user area *n*. PIP copies the file from the specified user area to the current user area. Place the option after the destination file name, the first parameter, to copy a file from the current user area to user area *n*. This is the only option that can appear after the first parameter. The value of *n* can range from 0 to 15. For example, the command

```
PIP E:[G2]=A:SORT.BAS[G3]
```

copies the file from user area 3 on disk A to user area 2 on disk E.

H — Hexadecimal data transfer. Always give this parameter when transferring HEX files. PIP then checks all data for proper Intel hexadecimal format. Use only for HEX files. For more information see my book *Mastering CP/M* (SYBEX, 1983).

I — Ignore HEX end-of-file record. When concatenating HEX files, include this parameter for each file but the last. This option automatically

sets the H option. Use the H option on the last file. For example, the command

```
PIP NEW.HEX = FIRST.HEX[I],SECOND.HEX[I],THIRD[H]
```

concatenates three HEX files. Use only for HEX files.

K — Omit (kill) display of file names during transfer. When the file-name parameter contains wild cards, PIP displays the word COPYING and then lists each file as it is transferred. The K option disables this feature.

L — Translate to lowercase. All uppercase letters are changed to the corresponding lowercase. Other characters are unchanged. Give the command

```
PIP CON: = FILE.TEX[L]
```

to observe the action on the video screen. Use only for text files. The Z option must be included for WordStar files.

N — Add line numbers. Numbers are inserted at the beginning of each line of a new file. The numbers begin with 1 and are incremented by one. Blanks fill out the field to six characters, and a colon and blank are placed after each number. This parameter is useful for referencing the lines of a listing. Do not use both the N and E options. WordStar text files are incorrectly numbered unless the Z option is also given. Use only for text files.

N2 — Add BASIC-style line numbers. Lines of the file are numbered as with the N parameter. However, zeros are used to fill the line instead of blanks, and an ASCII tab character follows each number instead of a colon and

blank. (The tab character can be expanded to the corresponding number of spaces by using the T option.) Use only for text files.

O — Object (nontext) file transfer. You must use this parameter when you concatenate non-text, non-COM files, but it is not needed for copying any file or for concatenating COM files. Do not use for text files.

P*n* — Set page length. PIP inserts the ^L form-feed character at the beginning of the new file and after every *n* lines. If *n* is omitted or has a value of 1, form feeds are inserted every 60 lines. (This is the default specification.) The F parameter should also be given so that any existing form feeds are removed. The printer you are using must respond to the form-feed character, or this parameter will have no effect. Use only for text files.

Q*string*^Z — Quit at this *string*. This parameter directs PIP to copy a file up to and including the given *string*. The *string* can be any regular keyboard characters. A ^Z character marks the end of the *string*. Execute the command on two lines as shown below to keep lowercase letters. A one-line command will convert lowercase letters to uppercase. For example, the commands

```
PIP
CON:=FILE.TXT[QWashington.^Z]
```

will display FILE.TXT on the video screen up to and including the expression *Washington*. Use only for text files. See the S option for starting strings.

R — Copy (read) system files. Files marked as system files are not displayed with the DIR command and are not normally copied by

PIP. Therefore, when making a backup copy of the system disk, you should give the R parameter. For example, the command

```
PIP E:=A:*.*[VR]
```

copies all files including system files from drive A to drive E.

Sstring^Z — Start at this *string*. This parameter directs PIP to start copying when it encounters the given *string*. The *string* can include any regular keyboard characters. A ^Z character marks the end of the *string*. Execute the command on two lines as shown below to keep lowercase letters. A one-line command will convert lowercase letters to uppercase. For example, the commands

```
PIP
CON:=FILE.TXT[SToday^Z]
```

will display FILE.TXT on the video screen starting with the expression *Today*. Use only for text files. See the Q option for quitting strings.

T*n* — Expand tabs. CP/M automatically expands the ASCII tab character, ^I, to eight-column fields by the addition of spaces when a file is displayed on the video screen. However, when a file is printed with the PIP device LST:, the tab character is copied unchanged. The T parameter can be included in a PIP command to expand the tab character to every *n*th column. PIP does this by adding spaces. Tabs are commonly used in assembly language and Pascal programs. For example, the command

```
PIP LST:=BIOS.ASM[T8]
```

will print the file BIOS.ASM and expand tabs to eight-column fields. Use only for text files.

U — Translate to uppercase. All lowercase letters are changed to the corresponding uppercase. Other characters are unchanged. Give the command

```
PIP CON: = FILE.TEX[U]
```

to observe the action on the video screen. Use only for text files. The Z option must be included for files prepared with WordStar.

V — Verify. This option directs PIP to verify that a new file has been correctly written. It should be used for all disk-to-disk transfers but not for transfers to peripheral devices such as CON: or LST:. An error message is displayed if the copy is incorrect.

W — Write over (delete) read-only files. Disk files can be write protected against accidental deletion by setting the RO attribute. Normally, when PIP creates a new file, it checks to see if a file with the same name already exists. If there is such a file and it is not write protected, PIP will automatically erase it. If the existing file is write protected, PIP will print the error message

```
DESTINATION IS R/O, DELETE (Y/N)?
```

and wait for your response. You answer Y to continue or N to quit. However, if you include the W option, PIP will automatically delete write-protected files on the destination drive without asking for permission.

Z — Zero the parity bit. The ASCII character set uses only seven bits. Therefore, the remaining bit of each byte, called the *parity bit,* can

be used as an error check. However, sometimes this bit is used for other purposes. For example, WordStar sets the parity bit to indicate spacing for justification. On the other hand, Microsoft BASIC requires the parity bit to be reset. Thus, if a BASIC program is edited with WordStar, the parity bit will be set for many of the characters and the BASIC program can no longer be used. When this happens, use the Z option of PIP to restore the BASIC program to its original form by zeroing the parity bit. For example, the command

```
PIP SORT.BAS = SORT.BAS[VZ]
```

will make the program SORT.BAS usable again. Notice that in this example both the original file and the new file have the same name. Do not use wild-card characters in this case or you may lose your file. The Z parameter is not needed for transfer to ASCII devices such as CON: and LST: and it must *not* be used for copying binary files or WordStar files.

PUT

Put video screen output or printer output into a disk file
(PUT.COM transient)

Format

```
PUT CONSOLE OUTPUT TO FILE PFILE.EXT[option]
PUT PRINTER OUTPUT TO FILE PFILE.EXT[option]
PUT CONSOLE OUTPUT TO CONSOLE
PUT PRINTER OUTPUT TO PRINTER
```

Description

CP/M normally sends console output to the video screen and printer output to the printer. The PUT command can send video screen and printer output to a disk file as well.

Use

The first and second forms of the PUT command instruct CP/M to create a disk file named PFILE.EXT and place into that file all video screen or printer output from the next program. When the program is completed, video screen or printer output automatically returns to the normal mode. The third and fourth forms of the command can be given to terminate the operation prematurely.

When the option NO ECHO is included, video screen output will not appear at the video screen and printer output will not appear at the printer. The FILTER option changes each control character to two printable characters—a ^ symbol and the corresponding letter.

Normally only the output from an executing program is written to the disk file. However, with the SYSTEM option all output, including the command line, appears in the file.

The expression OUTPUT TO may be omitted from the PUT command.

RENAME

Rename a disk file

(Built-in command and RENAME.COM extension)

Format

```
RENAME
RENAME NEWNAME.EXT=OLDNAME.EXT
RENAME ambig2=ambig1
```

Description

The RENAME command changes the name of a disk file. Only the name in the directory is changed; the file itself is not altered. The wild-card symbols ? and * may be used, but they must occur in identical positions in both names.

Use

If the file is not on the current drive, include the drive name in either or both file names. If the parameters are omitted, RENAME will ask for them.

See Chapter 3 for further details.

RMAC

Macroassembler for assembly language programming

(RMAC.COM transient)

Format

```
RMAC PNAME
RMAC PNAME $options
```

Description

The relocatable macroassembler RMAC creates three new files from an assembly language source program and an optional macro library. The source program has a file type of ASM, and the macro library has a type LIB. The assembled code is located in a REL file, the symbol table is found in a SYM file, and the printer listing is a PRN file. The separate program LINK creates an executable file from the REL file.

Use

A dollar sign rather than the usual bracket precedes the options. A space must precede the dollar sign. If no options are given, all files are referenced to the current drive. You can select any other drive by using the appropriate options. See a macro-assembler manual for more details. Also see MAC and LINK.

SAVE

Save contents of memory in a disk file
(SAVE.COM transient)

Format

```
SAVE
```

Description

The SAVE command creates a disk file from the contents of a selected portion of memory. This is usually the memory region starting at the address 100 hex; however, it may be anywhere. The memory image must be previously created by another program.

Use

Execute SAVE (without a parameter) before creating the memory image. SAVE relocates itself into high memory and returns control to CP/M. Run the program that creates the memory image you want to save. At the end of the program, control returns to the CP/M system, and the SAVE program automatically takes over. SAVE then asks for the name of the new disk file and the starting and ending addresses of the memory to be saved. If a disk file of the same name already exists, you are asked for permission to delete it.

This program is used primarily by programmers.

SET

Change file attributes, assign a label to a disk, set up password protection, and select the type of time and date stamping

(SET.COM transient)

Format

```
SET PNAME.EXT[option]
SET ambig[option]
SET D:[option]
SET[option]
```

Description

SET is used for several purposes. The most important application is changing the attributes **read only** or **read/write** and **system** or **directory** for a single file or a group of files. In addition, an entire disk drive can be set to read only or read/write status. The archive bit can also be changed.

A name or label and a separate password can be assigned to each disk and a password assigned to each individual file on the disk to keep unauthorized users from having access to these disk files. Both label and password follow the rules for file names. That is, the primary name can have as many as eight characters, and an optional extension can have up to three characters. The disk name can be displayed with SHOW.

A third capability of SET is selecting the type of date and time stamping. CP/M can encode two separate dates and times for each file. One of these refers to the date of last alteration, and the other is chosen to be either the creation date or the last access date. The separate program INITDIR formats a directory for time and date stamping, but SET is used to select the mode.

Use

See Chapter 3 for a description of how to use SET to change the attributes for an individual file or a group of files. In this

section we will look at the uses of SET that are not discussed in Chapter 3.

The archive attribute is used with PIP to make backup copies of altered files. PIP automatically sets the archive bit when the A option is given (see PIP). The archive bit can be changed with SET. The options are [ARCHIVE=ON] and [ARCHIVE=OFF].

A symbolic label is assigned with the NAME option. The label follows the rules for naming a disk file. For example, the command

```
SET A:[NAME=MYDISK]
```

assigns the label MYDISK to drive A. The label is encoded into the disk directory.

After a disk label has been assigned, it is possible to assign a password to the disk with the PASSWORD option. For example, the command

```
SET[PASSWORD=HIDDEN]
```

assigns the password HIDDEN to the disk. Password protection must be separately activated with the PROTECT option. The commands

```
SET[PROTECT=ON]
SET[PROTECT=OFF]
```

turn password protection on and off. Passwords can also be assigned to individual disk files by including a file-name parameter with the PASSWORD option.

There are four different password modes. For example, each of the following commands sets one of the four modes for the disk file named PNAME.EXT:

```
SET PNAME.EXT[PROTECT=READ]
SET PNAME.EXT[PROTECT=WRITE]
SET PNAME.EXT[PROTECT=DELETE]
SET PNAME.EXT[PROTECT=NONE]
```

When the READ option is selected, the password is required for

reading, altering, copying, renaming, or deleting the file. This is the default mode. If the WRITE mode is set, a password is not necessary to read or copy a file, but it is needed to alter, rename, or delete the file. When DELETE mode is set, a password is needed to delete or rename a file. The password is not needed after the NONE option is chosen.

To use date and time stamping, you must first prepare the disk directory with INITDIR and set the system clock with DATE. Then two of three modes must be activated. Give the command

```
SET[UPDATE = ON]
```

and then one of the two following commands:

```
SET[CREATE = ON]
SET[ACCESS = ON]
```

The UPDATE mode shows the date and time that a file was created with the editor or copied from another disk with PIP. The ACCESS mode shows when the file was last referenced. This includes inspecting with TYPE, making a copy to another disk with PIP, and executing a program. Since CREATE is like UPDATE, it is most useful to select the UPDATE and ACCESS options. Execute DIR with the FULL option to see the dates and times. After date and time stamping has been enabled, the DIR command with the [FULL] option will show the corresponding dates and times. See the DATE and INITDIR commands.

The archive flag is automatically turned off by the system editor when a file is altered. The flag is turned on when a file is copied by PIP using the A option. However, you may also change this flag with SET. These are the commands:

```
SET ambig[ARCHIVE = ON]
SET ambig[ARCHIVE = OFF]
```

(ARCHIVE may be abbreviated to AR.)

SETDEF

Define or display disk search path and turn paging on or off

(SETDEF.COM transient)

Format

```
SETDEF
SETDEF D:
SETDEF[option]
```

Description

You execute a transient program by typing its name. If the program is located on the current drive, you may omit the drive name. However, if it is on a different drive, you normally include the drive name with the file name so that CP/M can locate the program.

If you always work on drive E but execute programs located on drive A, you can redefine the way CP/M looks for a file. For example, CP/M can look for a file on drive A first, then on drive E, and finally on the current drive. This is called a search path.

Use

Give the command SETDEF without a parameter to display the current search path. The search path can be changed by including the drive names separated by commas. An asterisk defines the current drive. For example, the command

```
SETDEF A:,*
```

forces CP/M to first search drive A for a file to be executed. If CP/M cannot find the file on this drive, it looks on the current drive.

No drive-name parameter is included when you use the options DISPLAY, NO DISPLAY, PAGE, and NO PAGE.

The DISPLAY option of SETDEF directs CP/M to show the file name and location of programs when they are executed. The NO DISPLAY option turns this feature off.

Programs such as TYPE and DIR display a page of 24 lines on the video screen and then stop until you press the RETURN key or another key. Then the next screen is displayed. You can disable this feature with the [NO PAGE] option of SETDEF so that video screen output appears continuously. This mode is useful for quickly scanning the output or for sending the output to the printer.

The [PAGE] option reverses the effect of the [NO PAGE] option. The page length is changed with the DEVICE command. See Chapter 3 for further explanation.

SHOW

Show characteristics of a disk
(SHOW.COM transient)

Format

```
SHOW[option]
SHOW D:[option]
```

Description

The program SHOW displays the characteristics of an entire disk, in contrast to DIR, which displays the statistics of individual files. The SHOW information includes the remaining disk space; the remaining number of directory entries; the current user number; the numbers of the active user areas; and the disk drive characteristics such as capacity, block size, sector size, and sectors per track. The symbolic name assigned to the disk with SET can also be displayed.

Use

SHOW executed without a parameter gives the remaining disk space for all logged-in drives and identifies the drives as read only or read/write. With a drive parameter, SHOW gives the information for the indicated drive.

The option parameter [DIR] gives the number of remaining directory entries. (This information can also be obtained from the separate program DIR.) The [USERS] option identifies the current user number as well as all active user areas. The number of files associated with each user is also given.

The [DRIVE] option lists the physical features of the specified drive—storage capacity, number of directory entries, and block

size. The [LABEL] option identifies the assigned symbolic name for the disk. The name is encoded in the directory with the SET command. This is useful for removable media like floppy disks.

See Chapter 3 for further details.

SID

Debugger program to load, alter, and test assembly language programs
(SID.COM transient)

Format

```
SID
SID PNAME.EXT
SID PNAME.EXT PNAME.SYM
```

Description

SID (which stands for symbolic instruction debugger) is useful for assembly language programmers. It is used to develop programs assembled with MAC and RMAC. Executable COM or HEX files can be loaded with their symbol tables and run under precise control of SID. SID can disassemble executable code into its equivalent Intel 8080 mnemonics.

For additional information, refer to the CP/M SID manual from Digital Research, Inc.

SUBMIT

Execute commands from a disk file

(SUBMIT.COM transient)

Format

```
SUBMIT
SUBMIT PNAME
SUBMIT PNAME PARAM1 PARAM2 PARAM3 . . .
```

Description

A sequence of commands normally given to CP/M from the keyboard can be placed into a disk file and processed by SUBMIT instead. Input to an executing program can also be included. Then CP/M executes each line of the file as though it had been entered from the keyboard. When the list of commands has been exhausted, control returns to the keyboard. This operation is known as batch processing.

The operation can be made more versatile by adding replaceable parameters (known as dummy variables) to the file. Then one batch file can perform several slightly different tasks.

Use

You use the system editor to create a disk file containing the list of commands. The file type must be SUB. If the file name is PROFILE.SUB, CP/M Plus will automatically execute the commands in the file each time the computer is turned on.

Each line of the SUB is normally a CP/M command. However, when you include input to an executing program, the line begins with the < symbol. For example, the four lines

```
PIP
<E:=A:*.TXT[VA]
```

```
<E:=A:*.BAS[VA]
<
```

will execute PIP and then make archival backup copies of all TXT and BAS files.

Slightly different tasks can be performed with the same SUB file when dummy parameters are placed in the file. The dummy parameters are named $1, $2, and $3, as needed. For example, suppose a file SMALL.SUB contains these lines:

```
DIR $1.*
PIP $2:=$1.BAK
ERASE $1.BAK
```

The command

```
SUBMIT SMALL PROG E
```

directs SUBMIT to prepare the commands in the file SMALL.SUB for execution. Furthermore, the parameter PROG replaces the dummy parameter $1 and the parameter E replaces $2. Consequently, the following commands are issued to CP/M:

```
DIR PROG.*
PIP E:=PROG.BAK
ERASE PROG.BAK
```

For further explanation, see Chapter 3.

TYPE

Display a text file on the video screen

(Built-in command and TYPE.COM extension)

Format

```
TYPE
TYPE PNAME.EXT
TYPE ambig
TYPE PNAME.EXT[NO PAGE]
TYPE ambig[NO PAGE]
```

Description

The TYPE command displays the contents of an individual text file or a collection of files on the video screen. The display stops each time the screen fills, though this feature can be disabled with the [NO PAGE] option.

Use

The built-in version of TYPE can be executed from any disk and user area. The display freezes when the screen fills. Press the RETURN key to see the next screenful or ^C to terminate the command prematurely. Use ^S and ^Q to freeze the screen and resume scrolling, respectively.

Obtain a complete printed listing by pressing ^P before pressing the RETURN key. You can restrict the area of the listing by freezing the display with ^S at the appropriate place. Type ^P to engage the printer, then resume scrolling with ^Q. When listing a file on the printer, give the [NO PAGE] option so that the listing will not stop after each screen has been displayed.

The transient extension TYPE.COM is required in the following circumstances:

1. If the option parameter is included
2. If the ? or * wild cards appear in the file name

3. If the TYPE command is given without a parameter
4. If more than one file-name parameter is given

If you have not established a search path with SETDEF, you must include the drive location for the TYPE.COM file. You cannot use TYPE to display COM or REL files.

USER

Change current user area

(Built-in command)

Format

USER
USER *n*

Description

Each disk can be divided into as many as 16 different user areas, numbered 0 to 15. By this means, different working areas can be isolated from each other, though system files in user area 0 are available to all user areas.

User area 0 is automatically selected when CP/M is started. The built-in command USER changes the user area to the area specified.

Use

If you execute USER without a parameter, CP/M will ask for one. Alternatively, you can include the parameter with the command. When the user number is not 0, it is identified in the prompt. The active user areas can be determined with the transient program SHOW.

See Chapter 3 for more details.

XREF

Cross reference listing for assembler

(XREF.COM transient)

Format

```
XREF PNAME
XREF PNAME $P
```

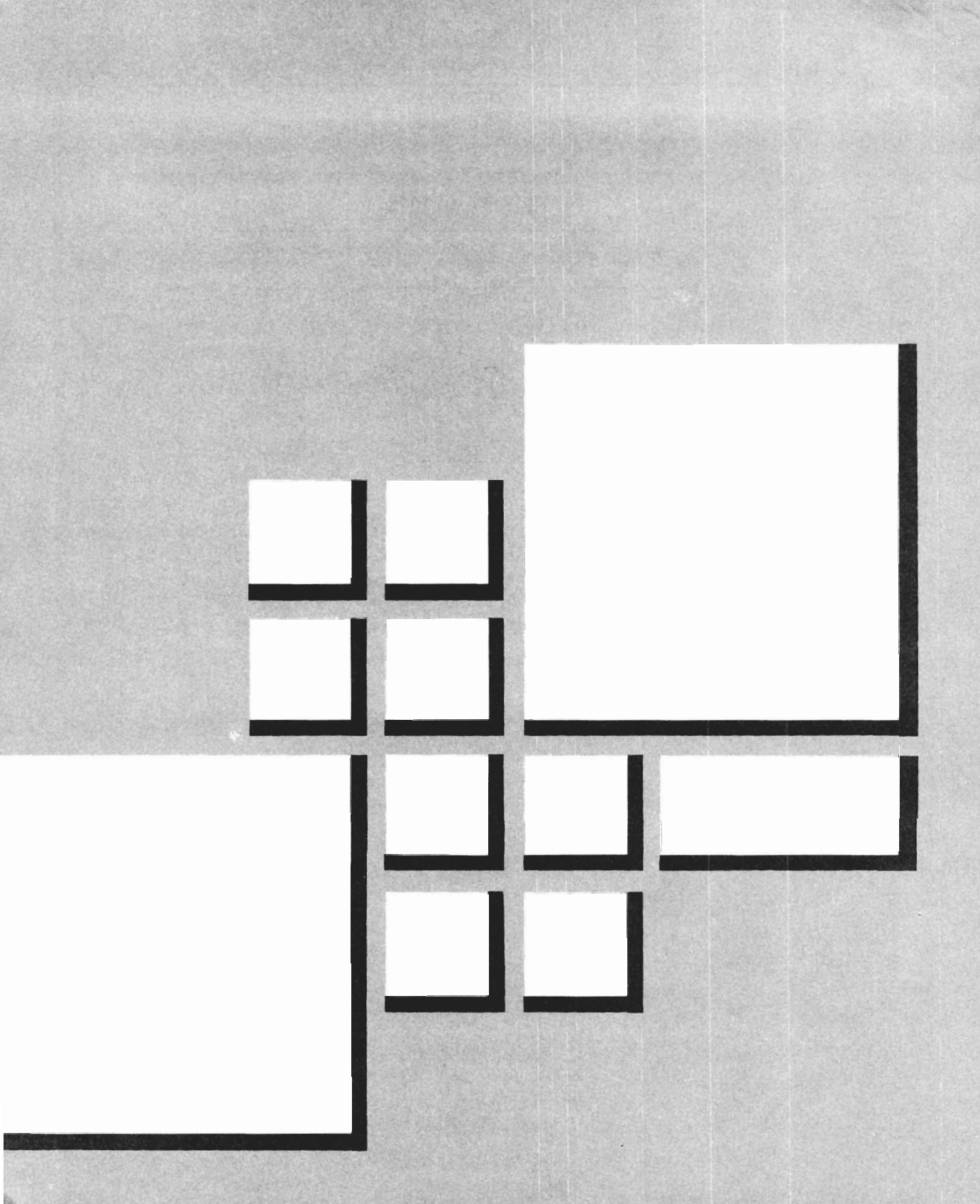
Description

The assemblers MAC and RMAC create an alphabetical listing of all symbols and their assigned values. XREF provides additional information by creating a cross-reference listing of these symbols. Each time a symbol is referenced, the corresponding line number is given. The PRN and SYM files from the assembler are used to create an XRF file.

Use

Normally, the XRF file is written to disk. However, the P option sends the listing to the printer. When only a single reference is given, it indicates that a symbol was defined but never used.

Notice that the option parameter is preceded by a dollar sign rather than the usual bracket. A space must precede the dollar sign.



Appendices

APPENDIX A
Hints for Beginners

Although your Commodore 128 is a complex machine that must be treated with respect, if you follow some simple procedures and precautions you should be able to make the best use of your computer and avoid problems. This appendix presents two kinds of information: practical techniques for using and caring for your hardware, and hints for using the system most effectively.

The Workspace

Keeping your computer workspace organized will make it easier for you to use the computer. Make sure all necessary documentation, blank disks, computer programs, and other supplies are available. Keep a listing of the steps needed to start up the Commodore 128 and the necessary precautions close at hand. Here is a typical list of supplies:

- New floppy disks
- Printer paper
- Envelopes
- Printer ribbons
- Printwheels
- System disk
- Disk(s) for applications program(s)
- Commodore manuals
- Printer manual
- Applications program manual(s)

Floppy Disks

Always treat your floppy disks carefully because they can be easily damaged. Floppy disks are magnetic media. Therefore, do not place them near a magnetic field or a steel object that might be magnetized. Magnetic fields can be produced by transformers, telephones, screwdrivers, steel desks, and any other steel objects. When traveling by air, do not pack disks in your luggage. Hand carry them and be sure to ask for a separate inspection when you board the plane. Do not let the disks be X-rayed, since the machine creates a magnetic field.

Always place disks in their sleeves when you are not using them. Do not expose them to a dusty or smoky environment. Special care must be taken in a dry climate, since disks can accumulate a static charge that makes dust and other particles stick to the surface. Do not scratch, touch, or attempt to clean the disk surface or open the outer envelope.

You should always place a label on each disk. Write something on the label that will remind you of the contents. For example, choose a descriptive name like BASIC programs, dBASE files, or GAMES.

Never write on a disk with a pen, pencil, or sharp object; the pressure can damage the surface. Instead, use a felt-tipped pen or write on the label and then affix it to the disk. Use only labels that are made especially for use on floppy disks. The wrong kind of label can damage the disk or the disk drive.

Do not bend disks or expose them to heat or direct sunlight. Remove all disks before turning off the computer.

If you live in a dry climate, or air condition your computer room, try to prevent the buildup of static electricity. A special antistatic spray can be used on carpeting.

Damaged Files

A disk file can become unusable or damaged because of operator error, a system malfunction, or power failure. A transient program may no longer execute, or it may not be possible to edit or list a text file. If you have a backup copy of the file, delete the damaged version and make a new copy from the

backup. If you do not have a backup copy, you should try to recover as much of a text file as possible. You are not likely to succeed with a transient program, but it may be possible to recover almost all of a text file.

Damage to the Disk Surface

Each sector of information stored on a disk has a cyclical redundancy code (CRC) formed from the contents of the sector. When the computer reads each sector, it recomputes the CRC and compares the information to the recorded CRC value. If the two do not agree, it means that there is a bad sector and the information is likely to be incorrect.

The surface of the disk can be damaged in one of two ways: a region can be incorrectly magnetized, or there can be a defect such as a scratch in the surface. It may be possible to correct a magnetic fault with FORMAT. However, if the surface is damaged, the disk usually must be discarded.

The Printer

Computer printers will work very reliably for months or even years without needing attention if they are treated with care. All mechanical adjustments must be correct, without exception, or the printer will not operate correctly.

An example of an adjustment that should be checked is the paper thickness lever. If you use thin paper when the lever is set for thick, the printer may malfunction in an unpredictable manner. There may appear to be a software problem, whereas only a simple adjustment is necessary. Other settings on the printer control full or half duplex, online or local, single or double line feed, paper-out sensor, baud rate, parity, and word length. Read the manufacturer's instructions for your printer so you can understand what these settings do and how to reset them if the switches are changed.

Because printers usually run quite slowly, you may be tempted to walk away from the computer while a printout is in progress.

Be sure you check the operation often to see that everything is in order. This is especially important at the beginning of a printout, because the paper might jam in the printer. The operator should always be present when printing adhesive labels, since they are more likely to cause jamming than regular paper.

Printer ribbons may be either carbon or cloth. All carbon ribbons and some cloth ribbons move in only one direction. However, some cloth ribbons occasionally reverse direction. At the moment of reversal the characters tend to be printed too lightly or not at all.

Whenever a problem occurs during printing, the printout will have to be restarted. If the first part is all right, it may be possible to restart the printout at the place where the problem occurred. Word processors such as WordStar have provisions for this. If you are printing a file with PIP, you can use the S option to start a file in the middle (see Chapter 7). On the other hand, if a file is short, it may be easier to start again from the beginning.

If the printer does not work when the system is turned on, check all printer settings. In particular, check that the online/local switch is set to online.

Disk Files

Initial Precautions

Be sure to format a new disk before you use it for the first time. Always make a backup copy of any new program or disk before it is first used. Then put the original disk away in a safe place. During a long editing session, frequently save your file in case of power failure or human error. In addition, regularly make a backup copy onto another disk.

Make the listing of the disk directory by engaging the printer with ^P and then executing DIR with the [FULL] option. Place the listing in the disk sleeve. Then you can readily determine the contents.

Keep a backup copy of any important disks in a separate location so that you do not risk losing all your information at once.

Size

There is a limit to the useful storage capacity of each disk as well as the available number of directory entries. As more and more files are saved on a disk, the free space can become exhausted. Another problem can arise if a single file is larger than the capacity of the disk. However, for fastest editing and duplication, file size should be limited to the available memory. This is about 40–50K bytes. For example, you can keep each chapter of a book as a separate file rather than making the entire book into one file. Of course, it is more difficult to divide a data file, such as a list of customer names, into several parts. One solution is to arrange the list in alphabetical order. Then you can break it into several parts—A–L and M–Z for example—on separate disks.

Before you edit a file, determine the file size with the [SIZE] or [FULL] option of DIR. Also determine the remaining disk space with SHOW. If you want to place the edited file on the same disk as the original, there must be sufficient room. In fact, the remaining space on the disk should be more than the size of the original file because the editor will need some work space. For example, if the file to be edited contains 80K bytes, the disk should contain 120K bytes of free space in addition to the original file. If there is not enough space, you can copy the edited file onto a different disk from the original file.

Sorting large files can also be a problem. Some sorting programs use the disk for working space. Then you must have room on the disk for both the original file and the sorted file. Other sorting programs let you change to a new disk when it is time to save the sorted version. A faster method of sorting works entirely in memory. However, the file size is then limited by the memory size to about 50K bytes. See my book *BASIC Programs for Scientists and Engineers* (SYBEX, 1981) for sorting routines.

System Files

Part of the CP/M Plus system is located on the system tracks of the disk. This is a region of the disk that is not normally accessible. Therefore, program storage space is not reduced by the

presence of CP/M on the system tracks. However, there are additional parts to CP/M Plus.

The disk files named CPM+.SYS and CCP.COM must be located in the data area of drive A. You will never execute them. Therefore they should have the system attribute set so that they will not show in normal DIR listings. See SET in Chapter 7 for the technique. You can give the DIRSYS command or the DIR command with the [SYS] option to see that CPM+ and CCP are present.

You must have the two CP/M files CPM+.SYS and CCP.COM on your system when the Commodore 128 is first turned on. These files must also be available when you reset the computer with the switch at the right edge of the system unit. However, these two files unnecessarily take up space if placed on every disk.

Changing a Phrase throughout a File

If you need to change a single word or expression throughout a file, the substitution command of the editor can accomplish this conveniently. (See Chapter 5 for details.) You can also use the substitution command to save yourself typing time and effort by incorporating a complicated expression that is needed at many places in a report. Let us see how to do this.

Suppose, for example, that the chemical compound naphthalene is mentioned 30 times in a report. At each point enter a symbol such as N$ instead. Then after the document is finished, use the substitution command to change each occurrence of N$ to naphthalene.

Executing a Sequence of Commands

You can easily execute a lengthy sequence of commands or a frequently used set of commands by creating a disk file of the commands and running SUBMIT. See Chapters 3 and 7 for details.

Your Commodore 128 can automatically execute a set of instructions each time your computer is turned on or reset. Place the instructions in a file named PROFILE.SUB. For example, the lines

```
DATE SET
SETDEF A:,*
```

could be placed into PROFILE.SUB. Then CP/M will ask you for the correct date and time. This will remind you to set the clock. The second command establishes a search path (see Chapter 3).

Printing Multiple Files

It is possible to print several files by using PIP with a wild-card file name or with concatenation (see Chapter 4). Alternatively, you can create a submit file, called PRINT.SUB for example, to print several files with one command (see Chapter 3). A third method is to engage the printer with ^P and give a wild-card file name to the TYPE command. Be sure to include the [NO PAGE] option so the listing does not stop at each page.

Erasing a Disk

You will need to erase all the files on a disk if you no longer need the programs or data. You can erase all files with the command

```
ERASE *.*
```

This command can be abbreviated to

```
ERA *.*
```

However, it may be easier to execute FORMAT, as described in Chapter 2. Of course, you can erase individual files by giving the ERASE command and the file name.

Terminating Execution

If you accidentally execute the wrong program or execute a program with the wrong parameters, it may be possible to terminate execution (that is, stop the program before it finishes) by

typing a ^C. This should work for all CP/M system programs such as TYPE and PIP. However, it is not possible to terminate all applications programs in this way. In this case you can interrupt execution with the reset switch. Do not turn off the computer or remove disks while a program is executing and the disks are running, or you may damage the disks.

Recovering from System Failure

Suspect the operator first:

1. Check the mechanical items.
   - Are all cables attached, with no loose connections?
   - Are all cables in the correct place?
2. Did you give the correct command?
   - Remove the disk and turn everything off.
   - Turn the system on.
   - Repeat the command.
3. Make sure that you are using the correct program and that the program you are trying to use exists on the disk in the current drive.

Suspect the disk next:

4. Use a fresh disk. The current disk may have been damaged.
   - Use a backup disk. Do not use any program on your current disk.
   - If no complete single backup exists, take the time to make one.

APPENDIX B
The CP/M Control Characters

Command	Action
^A	Move cursor one character left
^B	Move cursor to beginning of line (or to end of line if cursor is already at beginning)
^C	Terminate CP/M program; reset disks
^E	Move cursor to next line (used for long lines)
^F	Move cursor one character right
^G	Delete character at cursor position
^H	Delete character to left of cursor
^I	Move cursor to next tab position
^J	Execute command (line feed)
^K	Delete from cursor to end of line
^M	Execute command (carriage return)
^P	Engage or disengage printer
^Q	Resume scrolling after ^S command
^R	Redisplay line
^S	Freeze scrolling screen
^U	Delete all characters in line
^W	Recall previous command line
^X	Delete all characters to left of cursor
^Z	Mark end of string in PIP and ED

APPENDIX C
The ASCII Character Set

CODE	CHAR	CODE	CHAR	CODE	CHAR	CODE	CHAR
00	NUL	20[1]		40	@	60[5]	`
01	SOH	21	!	41	A	61	a
02	STX	22	"	42	B	62	b
03	ETX	23	#	43	C	63	c
04	EOT	24	$	44	D	64	d
05	ENQ	25	%	45	E	65	e
06	ACK	26	&	46	F	66	f
07	BEL	27[2]	'	47	G	67	g
08	BS	28	(	48	H	68	h
09	TAB	29	)	49	I	69	i
OA	LF	2A	*	4A	J	6A	j
OB	VT	2B	+	4B	K	6B	k
OC	FF	2C[3]	,	4C	L	6C	l
OD	CR	2D	—	4D	M	6D	m
OE	SO	2E	.	4E	N	6E	n
OF	SI	2F	/	4F	O	6F	o
10	DLE	30	0	50	P	70	p
11	DC1	31	1	51	Q	71	q
12	DC2	32	2	52	R	72	r
13	DC3	33	3	53	S	73	s
14	DC4	34	4	54	T	74	t
15	NAK	35	5	55	U	75	u
16	SYN	36	6	56	V	76	v
17	ETB	37	7	57	W	77	w
18	CAN	38	8	58	X	78	x
19	EM	39	9	59	Y	79	y
1A	SUB	3A	:	5A	Z	7A	z
1B	ESC	3B	;	5B	[	7B	{
1C	FS	3C	<	5C	\	7C	|
1D	GS	3D	=	5D	]	7D	}
1E	RS	3E	>	5E	↑	7E	~
1F	US	3F	?	5F[4]	←	7F	DEL

[1]space
[2]apostrophe
[3]comma
[4]underline
[5]accent mark

Index

SYBEX introduces one of the most sophisticated communications packages for your IBM-PC at the cost of a book!

Connect with **Elf**—

"as easy to use as it is powerful . . . easy to understand . . . simple enough for beginners . . . satisfies the needs of sophisticated users . . . becomes practically automatic . . . "

—PC Products

Elf automates virtually all calls and file transfers and turns your computer into a remote.

- Menu driven
- Easy to install
- Personality modules
- Crash-proof error handling
- Password protected post mode
- Supports XMODEM protocol
- Includes SYBEX-quality documentation

$49.95
(Diskette and Book)

ORDER NOW! CALL: 800-227-2346

SYBEX Computer Books are different.

Here is why . . .

At SYBEX, each book is designed with you in mind. Every manuscript is carefully selected and supervised by our editors, who are themselves computer experts. We publish the best authors, whose technical expertise is matched by an ability to write clearly and to communicate effectively. Programs are thoroughly tested for accuracy by our technical staff. Our computerized production department goes to great lengths to make sure that each book is well-designed.

In the pursuit of timeliness, SYBEX has achieved many publishing firsts. SYBEX was among the first to integrate personal computers used by authors and staff into the publishing process. SYBEX was the first to publish books on the CP/M operating system, microprocessor interfacing techniques, word processing, and many more topics.

Expertise in computers and dedication to the highest quality product have made SYBEX a world leader in computer book publishing. Translated into fourteen languages, SYBEX books have helped millions of people around the world to get the most from their computers. We hope we have helped you, too.

For a complete catalog of our publications:

SYBEX, Inc. 2344 Sixth Street, Berkeley, California 94710
Tel: (415) 848-8233 Telex: 336311

Mastering Disk Operations on The Commodore 128

Mastering Disk Operations on the Commodore 128 is an indispensable handbook for owners, users and programmers of the Commodore 128, with the CP/M Plus operating system. For newcomers and experienced users alike, this volume is the one place to turn for

- A step-by-step introduction to Commodore 128 hardware, software, and operating system fundamentals
- Straightforward explanations of CP/M Plus commands
- Full details on file-handling with PIP—including printing, defining user areas, manipulating system files, and other advanced topics
- An extended tutorial on ED, the CP/M Plus text editor
- A guide to the inner workings of CP/M Plus, for advanced users and programmers
- A complete CP/M Plus command summary for ongoing reference

Novice users in particular will benefit from easy-to-follow tutorials, clear examples and practical troubleshooting hints. A special appendix offers tips on efficient disk and file management, recovering from system failures, and care and maintenance for the Commodore 128.

About the Author

Alan R. Miller is a professor at the New Mexico Institute of Technology in Socorro, New Mexico, and a contributing editor to *Byte* magazine. He earned his Ph.D. in engineering from the University of California at Berkeley and has been teaching programming methods since 1967. He is the author of many popular SYBEX titles including *Mastering CP/M, Introduction to TopView* and the *Programming for Scientists and Engineers* series.

SYBEX

ISBN 0-89588-357-0 **U.S. $15.95**